The Hales–Jewett Theorem

Ujkan Sulejmani, Manuel Eberl, Katharina Kreuzer

March 24, 2023

Abstract

This article is a formalisation of a proof of the Hales–Jewett theorem presented in the textbook *Ramsey Theory* by Graham et al. [1].

The Hales–Jewett theorem is a result in Ramsey Theory which states that, for any non-negative integers r and t, there exists a minimal dimension N, such that any r-coloured N'-dimensional cube over t elements (with $N' \geq N$) contains a monochromatic line. This theorem generalises Van der Waerden's Theorem, which has already been formalised in another AFP entry [2].

Contents

1 Preliminaries **3**

1.1 The n-dimensional cube over t elements . . . 3

1.2 Lines . . . 4

1.3 Subspaces . . . 6

1.4 Equivalence classes . . . 7

2 Core proofs **19**

2.1 Theorem 4 . . . 19

2.1.1 Base case of Theorem 4 . . . 19

2.1.2 Induction step of theorem 4 . . . 27

2.2 Theorem 5 . . . 47

2.3 Corollary 6 . . . 52

2.4 Main result . . . 53

2.4.1 Edge cases and auxiliary lemmas . . . 53

2.4.2 Main theorem . . . 54

theory *Hales-Jewett*
  imports *Main HOL−Library.Disjoint-Sets HOL−Library.FuncSet*
begin

1 Preliminaries

The Hales–Jewett Theorem is at its core a statement about sets of tuples called the n-dimensional cube over t elements (denoted by C_t^n); i.e. the set $\{0, \ldots, t-1\}^n$, where $\{0, \ldots, t-1\}$ is called the base. We represent tuples by functions $f : \{0, \ldots, n-1\} \to \{0, \ldots, t-1\}$ because they're easier to deal with. The set of tuples then becomes the function space $\{0, \ldots, t-1\}^{\{0,\ldots,n-1\}}$. Furthermore, r-colourings of the cube are represented by mappings from the function space to the set $\{0, \ldots, r-1\}$.

1.1 The n-dimensional cube over t elements

Function spaces in Isabelle are supported by the library component FuncSet. In essence, $f \in A \to_E B$ means $a \in A \Longrightarrow f\ a \in B$ and $a \notin A \Longrightarrow f\ a = \mathit{undefined}$

The (canonical) n-dimensional cube over t elements is defined in the following using the variables:

n:	*nat*	dimension
t:	*nat*	number of elements

definition *cube* :: *nat* ⇒ *nat* ⇒ (*nat* ⇒ *nat*) *set*
  where *cube n t* ≡ {..<*n*} $\to_E$ {..<*t*}

For any function f whose image under a set A is a subset of another set B, there's a unique function g in the function space B^A that equals f everywhere in A. The function g is usually written as $f|_A$ in the mathematical literature.

lemma *PiE-uniqueness*: *f* ‘ *A* ⊆ *B* ⟹ ∃!*g* ∈ *A*
$\to_E$ *B*. ∀ *a*∈*A*. *g a* = *f a*
  using *exI*[*of* λ*x*. *x* ∈ *A* $\to_E$ *B* ∧ (∀ *a*∈*A*. *x a* = *f a*)
    restrict f A] *PiE-ext PiE-iff* **by** *fastforce*

Any prefix of length j of an n-tuple (i.e. element of C_t^n) is a j-tuple (i.e. element of C_t^j).

lemma *cube-restrict*:
  assumes *j* < *n*
    and *y* ∈ *cube n t*
  shows (λ*g* ∈ {..<*j*}. *y g*) ∈ *cube j t* **using** *assms* **unfolding** *cube-def* **by** *force*

Narrowing down the obvious fact $B^A \subseteq C^A$ if $B \subseteq C$ to a specific case for cubes.

lemma *cube-subset*: $cube\ n\ t \subseteq cube\ n\ (t + 1)$
unfolding *cube-def* **using** $PiE\text{-}mono[of\ \{..<n\}\ \lambda x.\ \{..<t\}\ \lambda x.\ \{..<t+1\}]$
by *simp*

A simplifying definition for the 0-dimensional cube.

lemma *cube0-alt-def*: $cube\ 0\ t = \{\lambda x.\ undefined\}$
unfolding *cube-def* **by** *simp*

The cardinality of the n-dimensional over t elements is simply a consequence of the overarching definition of the cardinality of function spaces (over finite sets).

lemma *cube-card*: $card\ (\{..<n::nat\} \rightarrow_E \{..<t::nat\}) = t\ \hat{}\ n$
by (*simp add*: *card-PiE*)

A simplifying definition for the n-dimensional cube over a single element, i.e. the single n-dimensional point $(0, \ldots, 0)$.

lemma *cube1-alt-def*: $cube\ n\ 1 = \{\lambda x{\in}\{..<n\}.\ 0\}$ **unfolding** *cube-def* **by** (*simp add*: *lessThan-Suc*)

1.2 Lines

The property of being a line in C_t^n is defined in the following using the variables:

L:	$nat \Rightarrow nat \Rightarrow nat$	line
n:	nat	dimension of cube
t:	nat	the size of the cube's base

definition *is-line* :: $(nat \Rightarrow (nat \Rightarrow nat)) \Rightarrow nat \Rightarrow nat \Rightarrow bool$
where $is\text{-}line\ L\ n\ t \equiv (L \in \{..<t\} \rightarrow_E cube\ n\ t\ \wedge$
$((\forall j<n.\ (\forall x<t.\ \forall y<t.\ L\ x\ j = L\ y\ j) \vee (\forall s<t.\ L\ s\ j = s))$
$\wedge (\exists j < n.\ (\forall s < t.\ L\ s\ j = s))))$

We introduce an elimination rule to relate lines with the more general definition of a subspace (see below).

lemma *is-line-elim-t-1*:
assumes $is\text{-}line\ L\ n\ t$ **and** $t = 1$
obtains $B_0\ B_1$
where $B_0 \cup B_1 = \{..<n\} \wedge B_0 \cap B_1 = \{\} \wedge$
$B_0 \neq \{\} \wedge (\forall j \in B_1.\ (\forall x<t.\ \forall y<t.\ L\ x\ j = L\ y\ j)) \wedge (\forall j \in B_0.\ (\forall s<t.\ L\ s\ j = s))$
proof –
define $B0$ **where** $B0 = \{..<n\}$
define $B1$ **where** $B1 = (\{\}::nat\ set)$
have $B0 \cup B1 = \{..<n\}$ **unfolding** *B0-def B1-def* **by** *simp*
moreover have $B0 \cap B1 = \{\}$ **unfolding** *B0-def B1-def* **by** *simp*
moreover have $B0 \neq \{\}$ **using** *assms* **unfolding** *B0-def is-line-def* **by** *auto*

moreover have $(\forall j \in B1.\ (\forall x<t.\ \forall y<t.\ L\ x\ j = L\ y\ j))$ **unfolding** *B1-def* **by** *simp*
moreover have $(\forall j \in B0.\ (\forall s<t.\ L\ s\ j = s))$ **using** *assms*(*1*, *2*) *cube1-alt-def* **unfolding** *B0-def is-line-def* **by** *auto*
ultimately show *?thesis* **using** *that* **by** *simp*
qed

The next two lemmas are used to simplify proofs by enabling us to use the resulting facts directly. This avoids having to unfold the definition of *is-line* each time.

lemma *line-points-in-cube*:
assumes *is-line* $L\ n\ t$
and $s < t$
shows $L\ s \in$ *cube* $n\ t$
using *assms* **unfolding** *cube-def is-line-def*
by *auto*

lemma *line-points-in-cube-unfolded*:
assumes *is-line* $L\ n\ t$
and $s < t$
and $j < n$
shows $L\ s\ j \in \{..<t\}$
using *assms line-points-in-cube* **unfolding** *cube-def* **by** *blast*

The incrementation of all elements of a set is defined in the following using the variables:

n:	*nat*	increment size
S:	*nat set*	set

definition *set-incr* :: *nat* $\Rightarrow$ *nat set* $\Rightarrow$ *nat set*
where
set-incr $n\ S \equiv (\lambda a.\ a + n)$ ‘ S

lemma *set-incr-disjnt*:
assumes *disjnt* $A\ B$
shows *disjnt* (*set-incr* $n\ A$) (*set-incr* $n\ B$)
using *assms* **unfolding** *disjnt-def set-incr-def* **by** *force*

lemma *set-incr-disjoint-family*:
assumes *disjoint-family-on* $B\ \{..k\}$
shows *disjoint-family-on* ($\lambda i.$ *set-incr* $n\ (B\ i)$) $\{..k\}$
using *assms set-incr-disjnt* **unfolding** *disjoint-family-on-def* **by** (*meson disjnt-def*)

lemma *set-incr-altdef*: *set-incr* $n\ S = (+)\ n$ ‘ S
by (*auto simp*: *set-incr-def*)

lemma *set-incr-image*:
assumes $(\bigcup i\in\{..k\}.\ B\ i) = \{..<n\}$

shows $(\bigcup i{\in}\{..k\}.\ \textit{set-incr}\ m\ (B\ i)) = \{m..{<}m{+}n\}$
using *assms* **by** (*simp add*: *set-incr-altdef add.commute flip*: *image-UN atLeast0LessThan*)

Each tuple of dimension $k+1$ can be split into a tuple of dimension 1 (the first entry) and a tuple of dimension k (the remaining entries).

lemma *split-cube*:
assumes $x \in \textit{cube}\ (k{+}1)\ t$
shows $(\lambda y \in \{..{<}1\}.\ x\ y) \in \textit{cube}\ 1\ t$
and $(\lambda y \in \{..{<}k\}.\ x\ (y + 1)) \in \textit{cube}\ k\ t$
using *assms* **unfolding** *cube-def* **by** *auto*

1.3 Subspaces

The property of being a k-dimensional subspace of C_t^n is defined in the following using the variables:

S:	$(\textit{nat} \Rightarrow \textit{nat}) \Rightarrow \textit{nat} \Rightarrow \textit{nat}$	the subspace
k:	*nat*	the dimension of the subspace
n:	*nat*	the dimension of the cube
t:	*nat*	the size of the cube's base

definition *is-subspace*
where $\textit{is-subspace}\ S\ k\ n\ t \equiv (\exists B\ f.\ \textit{disjoint-family-on}\ B\ \{..k\} \wedge \bigcup(B\ `$
$\{..k\}) = \{..{<}n\} \wedge (\{\} \notin B\ `\ \{..{<}k\}) \wedge f \in (B\ k) \rightarrow_E \{..{<}t\}$
$\wedge\ S \in (\textit{cube}\ k\ t) \rightarrow_E (\textit{cube}\ n\ t) \wedge (\forall y \in \textit{cube}\ k\ t.$
$(\forall i \in B\ k.\ S\ y\ i = f\ i) \wedge (\forall j{<}k.\ \forall i \in B\ j.\ (S\ y)\ i = y\ j)))$

A k-dimensional subspace of C_t^n can be thought of as an embedding of the C_t^k into C_t^n, akin to how a k-dimensional vector subspace of $\mathbf{R}^n$ may be thought of as an embedding of $\mathbf{R}^k$ into $\mathbf{R}^n$.

lemma *subspace-inj-on-cube*:
assumes $\textit{is-subspace}\ S\ k\ n\ t$
shows $\textit{inj-on}\ S\ (\textit{cube}\ k\ t)$
proof
fix $x\ y$
assume a: $x \in \textit{cube}\ k\ t\ \ y \in \textit{cube}\ k\ t\ \ S\ x = S\ y$
from *assms* **obtain** $B\ f$ **where** *Bf-props*: $\textit{disjoint-family-on}\ B\ \{..k\} \wedge \bigcup(B\ `$
$\{..k\}) =$
$\{..{<}n\} \wedge (\{\} \notin B\ `\ \{..{<}k\}) \wedge f \in (B\ k) \rightarrow_E \{..{<}t\} \wedge$
$S \in (\textit{cube}\ k\ t) \rightarrow_E (\textit{cube}\ n\ t) \wedge (\forall y \in \textit{cube}\ k\ t.$
$(\forall i \in B\ k.\ S\ y\ i = f\ i) \wedge (\forall j{<}k.\ \forall i \in B\ j.\ (S\ y)\ i = y\ j))$
unfolding *is-subspace-def* **by** *auto*
have $\forall i{<}k.\ x\ i = y\ i$
proof (*intro allI impI*)
fix j **assume** $j < k$
then have $B\ j \neq \{\}$ **using** *Bf-props* **by** *auto*
then obtain i **where** *i-prop*: $i \in B\ j$ **by** *blast*
then have $y\ j = S\ y\ i$ **using** *Bf-props* $a(2)$ $\langle j < k\rangle$ **by** *auto*
also have $... = S\ x\ i$ **using** a **by** *simp*

also have ... $= x\ j$ **using** *Bf-props* $a(1)$ ‹$j < k$› *i-prop* **by** *blast*
finally show $x\ j = y\ j$ **by** *simp*
qed
then show $x = y$ **using** $a(1,2)$ **unfolding** *cube-def* **by** (*meson PiE-ext lessThan-iff*)
qed

The following is required to handle base cases in the key lemmas.

lemma *dim0-subspace-ex*:
assumes $t > 0$
shows $\exists S.$ *is-subspace* $S\ 0\ n\ t$
proof–
define B **where** $B \equiv (\lambda x{::}nat.\ undefined)(0{:=}\{..<n\})$

have $\{..<t\} \neq \{\}$ **using** *assms* **by** *auto*
then have $\exists f.\ f \in (B\ 0) \rightarrow_E \{..<t\}$
by (*meson PiE-eq-empty-iff all-not-in-conv*)
then obtain f **where** *f-prop*: $f \in (B\ 0) \rightarrow_E \{..<t\}$ **by** *blast*
define S **where** $S \equiv (\lambda x{::}(nat \Rightarrow nat).\ undefined)((\lambda x.\ undefined){:=}f)$

have *disjoint-family-on* $B\ \{..0\}$ **unfolding** *disjoint-family-on-def* **by** *simp*
moreover have $\bigcup(B\ `\ \{..0\}) = \{..<n\}$ **unfolding** *B-def* **by** *simp*
moreover have $(\{\} \notin B\ `\ \{..<0\})$ **by** *simp*
moreover have $S \in (cube\ 0\ t) \rightarrow_E (cube\ n\ t)$
using *f-prop PiE-I* **unfolding** *B-def cube-def S-def* **by** *auto*
moreover have $(\forall y \in cube\ 0\ t.\ (\forall i \in B\ 0.\ S\ y\ i = f\ i) \wedge$
$(\forall j{<}0.\ \forall i \in B\ j.\ (S\ y)\ i = y\ j))$ **unfolding** *cube-def S-def* **by** *force*
ultimately have *is-subspace* $S\ 0\ n\ t$ **using** *f-prop* **unfolding** *is-subspace-def* **by** *blast*
then show $\exists S.$ *is-subspace* $S\ 0\ n\ t$ **by** *auto*
qed

1.4 Equivalence classes

Defining the equivalence classes of $cube\ n\ (t + 1)$: $\{classes\ n\ t\ 0, \ldots, classes\ n\ t\ n\}$

definition *classes*
where $classes\ n\ t \equiv (\lambda i.\ \{x\ .\ x \in (cube\ n\ (t + 1)) \wedge (\forall u \in \{(n{-}i)..{<}n\}.\ x\ u = t) \wedge t \notin x\ `\ \{..{<}(n - i)\}\})$

lemma *classes-subset-cube*: $classes\ n\ t\ i \subseteq cube\ n\ (t{+}1)$ **unfolding** *classes-def* **by** *blast*

definition *layered-subspace*
where *layered-subspace* $S\ k\ n\ t\ r\ \chi \equiv$ (*is-subspace* $S\ k\ n\ (t + 1)\ \wedge (\forall i \in \{..k\}.\ \exists c{<}r.\ \forall x \in classes\ k\ t\ i.\ \chi\ (S\ x) = c)) \wedge \chi \in cube\ n\ (t + 1) \rightarrow_E \{..{<}r\}$

lemma *layered-eq-classes*:

assumes *layered-subspace S k n t r* χ
shows $\forall\, i \in \{..k\}.\ \forall\, x \in$ *classes k t i*. $\forall\, y \in$ *classes k t i*.
χ $(S\ x) = \chi$ $(S\ y)$
proof (*safe*)
fix *i x y*
assume *a*: $i \leq k$ $x \in$ *classes k t i* $y \in$ *classes k t i*
then obtain *c* **where** $c < r \wedge \chi\ (S\ x) = c \wedge \chi\ (S\ y) = c$ **using** *assms* **unfolding** *layered-subspace-def* **by** *fast*
then show χ $(S\ x) = \chi$ $(S\ y)$ **by** *simp*
qed

lemma *dim0-layered-subspace-ex*:
assumes $\chi \in$ (*cube n* $(t + 1)$) $\rightarrow_E \{..<r::nat\}$
shows $\exists\, S$. *layered-subspace S* $(0::nat)$ *n t r* χ
proof–
obtain *S* **where** *S-prop*: *is-subspace S* $(0::nat)$ *n* $(t+1)$ **using** *dim0-subspace-ex* **by** *auto*
have *classes* $(0::nat)$ *t 0* = *cube 0* $(t+1)$ **unfolding** *classes-def* **by** *simp*
moreover have $(\forall\, i \in \{..0::nat\}.\ \exists\, c<r.\ \forall\, x \in$ *classes* $(0::nat)$ *t i*. χ $(S\ x) = c)$
proof(*safe*)
fix *i*
have $\forall\, x \in$ *classes 0 t 0*. χ $(S\ x) = \chi$ $(S\ (\lambda x.\ undefined))$ **using** *cube0-alt-def* **using** ‹*classes 0 t 0* = *cube 0* $(t + 1)$› **by** *auto*
moreover have S $(\lambda x.\ undefined) \in$ *cube n* $(t+1)$ **using** *S-prop cube0-alt-def* **unfolding** *is-subspace-def* **by** *auto*
moreover have χ $(S\ (\lambda x.\ undefined)) < r$ **using** *assms calculation* **by** *auto*
ultimately show $\exists\, c<r.\ \forall\, x \in$*classes 0 t 0*. χ $(S\ x) = c$ **by** *auto*
qed
ultimately have *layered-subspace S 0 n t r* χ **using** *S-prop assms* **unfolding** *layered-subspace-def* **by** *blast*
then show $\exists\, S$. *layered-subspace S* $(0::nat)$ *n t r* χ **by** *auto*
qed

lemma *disjoint-family-onI* [*intro*]:
assumes $\bigwedge m\ n.\ m \in S \Longrightarrow n \in S \Longrightarrow m \neq n \Longrightarrow A\ m \cap A\ n = \{\}$
shows *disjoint-family-on A S*
using *assms* **by** (*auto simp*: *disjoint-family-on-def*)

lemma *fun-ex*: $a \in A \Longrightarrow b \in B \Longrightarrow \exists f \in A \rightarrow_E B.\ f\ a = b$
proof–
assume *assms*: $a \in A$ $b \in B$
then obtain *g* **where** *g-def*: $g \in A \rightarrow B \wedge g\ a = b$ **by** *fast*
then have *restrict g A* $\in A \rightarrow_E B \wedge$ (*restrict g A*) $a = b$ **using** *assms*(*1*) **by** *auto*
then show *?thesis* **by** *blast*
qed

lemma *ex-bij-betw-nat-finite-2*:
  assumes *card A* = *n*
    and *n* > *0*
  shows ∃*f*. *bij-betw f A* {..<*n*}
 using *assms ex-bij-betw-finite-nat*[*of A*] *atLeast0LessThan card-ge-0-finite* **by** *auto*

lemma *one-dim-cube-eq-nat-set*: *bij-betw* (λ*f*. *f 0*) (*cube 1 k*) {..<*k*}
proof (*unfold bij-betw-def*)
  have ∗: (λ*f*. *f 0*) ' *cube 1 k* = {..<*k*}
  proof(*safe*)
    fix *x f*
    assume *f* ∈ *cube 1 k*
    then show *f 0* < *k* **unfolding** *cube-def* **by** *blast*
  next
    fix *x*
    assume *x* < *k*
    then have *x* ∈ {..<*k*} **by** *simp*
    moreover have *0* ∈ {..<*1*::*nat*} **by** *simp*
    ultimately have ∃ *y* ∈ {..<*1*::*nat*} $\rightarrow_E$ {..<*k*}. *y 0* = *x* **using**
        fun-ex[*of 0* {..<*1*::*nat*} *x* {..<*k*}] **by** *auto*
    then show *x* ∈ (λ*f*. *f 0*) ' *cube 1 k* **unfolding** *cube-def* **by** *blast*
  qed
  moreover
  {
    have *card* (*cube 1 k*) = *k* **using** *cube-card* **by** (*simp add*: *cube-def*)
    moreover have *card* {..<*k*} = *k* **by** *simp*
    ultimately have *inj-on* (λ*f*. *f 0*) (*cube 1 k*) **using** ∗ *eq-card-imp-inj-on*[*of cube 1 k* λ*f*. *f 0*]
      by *force*
  }
  ultimately show *inj-on* (λ*f*. *f 0*) (*cube 1 k*) ∧ (λ*f*. *f 0*) ' *cube 1 k* = {..<*k*} **by** *simp*
qed

An alternative introduction rule for the ∃!*x* quantifier, which means "there exists exactly one *x*".

lemma *ex1I-alt*: (∃ *x*. *P x* ∧ (∀ *y*. *P y* ⟶ *x* = *y*)) ⟹ (∃!*x*. *P x*)
  by *auto*
lemma *nat-set-eq-one-dim-cube*: *bij-betw* (λ*x*. λ*y*∈{..<*1*::*nat*}. *x*) {..<*k*::*nat*} (*cube 1 k*)
proof (*unfold bij-betw-def*)
  have ∗: (λ*x*. λ*y*∈{..<*1*::*nat*}. *x*) ' {..<*k*} = *cube 1 k*
  proof (*safe*)
    fix *x y*
    assume *y* < *k*
    then show (λ*z*∈{..<*1*}. *y*) ∈ *cube 1 k* **unfolding** *cube-def* **by** *simp*
  next
    fix *x*
    assume *x* ∈ *cube 1 k*

have $x = (\lambda z.\ \lambda y{\in}\{..<\mathit{1}::nat\}.\ z)\ (x\ \mathit{0}::nat)$
proof
fix j
consider $j \in \{..<\mathit{1}\} \mid j \notin \{..<\mathit{1}::nat\}$ **by** *linarith*
then show $x\ j = (\lambda z.\ \lambda y{\in}\{..<\mathit{1}::nat\}.\ z)\ (x\ \mathit{0}::nat)\ j$ **using** ‹x
$\in$ *cube 1 k*› **unfolding** *cube-def* **by** *auto*
qed
moreover have $x\ \mathit{0} \in \{..<k\}$ **using** ‹$x \in$ *cube 1 k*› **by** (*auto simp add*: *cube-def*)
ultimately show $x \in (\lambda z.\ \lambda y{\in}\{..<\mathit{1}\}.\ z)$ ‘ $\{..<k\}$ **by** *blast*
qed
moreover
{
have *card* (*cube 1 k*) $= k$ **using** *cube-card* **by** (*simp add*: *cube-def*)
moreover have *card* $\{..<k\} = k$ **by** *simp*
ultimately have *inj-on* $(\lambda x.\ \lambda y{\in}\{..<\mathit{1}::nat\}.\ x)\ \{..<k\}$ **using** $*$
eq-card-imp-inj-on[*of* $\{..<k\}\ \lambda x.\ \lambda y{\in}\{..<\mathit{1}::nat\}.\ x$] **by** *force*
}
ultimately show *inj-on* $(\lambda x.\ \lambda y{\in}\{..<\mathit{1}::nat\}.\ x)\ \{..<k\} \wedge (\lambda x.$
$\lambda y{\in}\{..<\mathit{1}::nat\}.\ x)$ ‘ $\{..<k\} =$ *cube 1 k* **by** *blast*
qed

A bijection f between domains A_1 and A_2 creates a correspondence between functions in $A_1 \to B$ and $A_2 \to B$.

lemma *bij-domain-PiE*:
assumes *bij-betw f A1 A2*
and $g \in A2 \to_E B$
shows (*restrict* $(g \circ f)$ *A1*) $\in A1 \to_E B$
using *bij-betwE assms* **by** *fastforce*

The following three lemmas relate lines to 1-dimensional subspaces (in the natural way). This is a direct consequence of the elimination rule *is-line-elim* introduced above.

lemma *line-is-dim1-subspace-t-1*:
assumes $n > 0$
and *is-line L n 1*
shows *is-subspace* (*restrict* $(\lambda y.\ L\ (y\ \mathit{0}))$ (*cube 1 1*)) *1 n 1*
proof –
obtain $B_0\ B_1$ **where** *B-props*: $B_0 \cup B_1 = \{..<n\} \wedge B_0$
$\cap B_1 = \{\} \wedge B_0 \neq \{\} \wedge (\forall j \in B_1.$
$(\forall x<\mathit{1}.\ \forall y<\mathit{1}.\ L\ x\ j = L\ y\ j)) \wedge (\forall j \in B_0.\ (\forall s<\mathit{1}.\ L$
$s\ j = s))$ **using** *is-line-elim-t-1*[*of L n 1*] *assms* **by** *auto*
define B **where** $B \equiv (\lambda i::nat.\ \{\}::nat\ set)(\mathit{0}:=B_0,\ \mathit{1}:=B_1)$
define f **where** $f \equiv (\lambda i \in B\ \mathit{1}.\ L\ \mathit{0}\ i)$
have $*$: $L\ \mathit{0} \in \{..<n\} \to_E \{..<\mathit{1}\}$ **using** *assms*(*2*) **unfolding** *cube-def is-line-def*
by *auto*
have *disjoint-family-on* $B\ \{..\mathit{1}\}$ **unfolding** *B-def* **using** *B-props*
by (*simp add*: *Int-commute disjoint-family-onI*)
moreover have $\bigcup\ (B$ ‘ $\{..\mathit{1}\}) = \{..<n\}$ **unfolding** *B-def* **using** *B-props* **by**
auto

```
  moreover have {} ∉ B ` {..<1} unfolding B-def using B-props by auto
  moreover have f ∈ B 1 →E {..<1} using * calculation(2) unfolding f-def by auto
  moreover have (restrict (λy. L (y 0)) (cube 1 1)) ∈ cube 1 1 →E cube n 1
    using assms(2) cube1-alt-def unfolding is-line-def by auto
  moreover have (∀y∈cube 1 1. (∀i∈B 1. (restrict (λy. L (y 0)) (cube 1 1)) y i = f i)
  ∧ (∀j<1. ∀i∈B j. (restrict (λy. L (y 0)) (cube 1 1)) y i = y j))
    using cube1-alt-def B-props * unfolding B-def f-def by auto
  ultimately show ?thesis unfolding is-subspace-def by blast
qed

lemma line-is-dim1-subspace-t-ge-1:
  assumes n > 0
    and t > 1
    and is-line L n t
  shows is-subspace (restrict (λy. L (y 0)) (cube 1 t)) 1 n t
proof –
  let ?B1 = {i::nat . i < n ∧ (∀x<t. ∀y<t. L x i =  L y i)}
  let ?B0 = {i::nat . i < n ∧ (∀s < t. L s i = s)}
  define B where B ≡ (λi::nat. {}::nat set)(0:=?B0, 1:=?B1)
  let ?L = (λy ∈ cube 1 t. L (y 0))
  have ?B0 ≠ {} using assms(3) unfolding is-line-def by simp

  have L1: ?B0 ∪ ?B1 = {..<n} using assms(3) unfolding is-line-def by auto
  {
    have (∀s < t. L s i = s) ⟶ ¬(∀x<t. ∀y<t. L x i =
    L y i) if i < n for i using assms(2) less-trans by auto
    then have *:i ∉ ?B0 if i ∈ ?B1 for i using that by blast
  }
  moreover
  {
    have (∀x<t. ∀y<t. L x i =  L y i) ⟶ ¬(∀s < t. L s i = s)
      if i < n for i using that calculation by blast
    then have **: ∀i ∈ ?B0. i ∉ ?B1
      by blast
  }
  ultimately have L2: ?B0 ∩ ?B1 = {} by blast

  let ?f = (λi. if i ∈ B 1 then L 0 i else undefined)
  {
    have {..1::nat} = {0, 1} by auto
    then have ⋃(B ` {..1::nat}) = B 0 ∪ B 1 by simp
    then have ⋃(B ` {..1::nat}) = ?B0 ∪ ?B1 unfolding B-def by simp
    then have A1: disjoint-family-on B {..1::nat} using L2
      by (simp add: B-def Int-commute disjoint-family-onI)
  }
  moreover
  {
```

have $\bigcup(B$ ‘ $\{..1::nat\}) = B\ 0 \cup B\ 1$ **unfolding** *B-def* **by** *auto*
then have $\bigcup(B$ ‘ $\{..1::nat\}) = \{..<n\}$ **using** *L1* **unfolding** *B-def* **by** *simp*
}
moreover
{
have $\forall\, i \in \{..<1::nat\}.\ B\ i \neq \{\}$
using ‹$\{i.\ i < n \wedge (\forall\, s<t.\ L\ s\ i = s)\} \neq \{\}$› *fun-upd-same lessThan-iff less-one*

unfolding *B-def* **by** *auto*
then have $\{\} \notin B$ ‘ $\{..<1::nat\}$ **by** *blast*
}
moreover
{
have $?f \in (B\ 1) \rightarrow_E \{..<t\}$
proof
fix i
assume *asm*: $i \in (B\ 1)$
have $L\ a\ b \in \{..<t\}$ **if** $a < t$ **and** $b < n$ **for** $a\ b$ **using** *assms*(*3*) *that* **unfolding**
is-line-def cube-def **by** *auto*
then have $L\ 0\ i \in \{..<t\}$ **using** *assms*(*2*) *asm calculation*(*2*) **by** *blast*
then show $?f\ i \in \{..<t\}$ **using** *asm* **by** *presburger*
qed (*auto*)
}

moreover
{
have $L \in \{..<t\} \rightarrow_E (cube\ n\ t)$ **using** *assms*(*3*) **by** (*simp add*: *is-line-def*)
then have $?L \in (cube\ 1\ t) \rightarrow_E (cube\ n\ t)$
using *bij-domain-PiE*[*of* ($\lambda f.\ f\ 0$) (*cube 1 t*) $\{..<t\}$ *L cube n t*] *one-dim-cube-eq-nat-set*[*of*
t]
by *auto*
}
moreover
{
have $\forall\, y \in cube\ 1\ t.\ (\forall\, i \in B\ 1.\ ?L\ y\ i = ?f\ i) \wedge (\forall\, j < 1.$
$\forall\, i \in B\ j.\ (?L\ y)\ i = y\ j)$
proof
fix y
assume $y \in cube\ 1\ t$
then have $y\ 0 \in \{..<t\}$ **unfolding** *cube-def* **by** *blast*

have $(\forall\, i \in B\ 1.\ ?L\ y\ i = ?f\ i)$
proof
fix i
assume $i \in B\ 1$
then have $?f\ i = L\ 0\ i$
by *meson*
moreover have $?L\ y\ i = L\ (y\ 0)\ i$ **using** ‹$y \in cube\ 1\ t$› **by** *simp*
moreover have $L\ (y\ 0)\ i = L\ 0\ i$

proof –
have $i \in$ *?B1* **using** ‹$i \in B\ 1$› **unfolding** *B-def fun-upd-def* **by** *presburger*
then have $(\forall x<t.\ \forall y<t.\ L\ x\ i = L\ y\ i)$ **by** *blast*
then show $L\ (y\ 0)\ i = L\ 0\ i$ **using** ‹$y\ 0 \in \{..<t\}$› **by** *blast*
qed
ultimately show *?L* $y\ i =$ *?f* i **by** *simp*
qed

moreover have (*?L* y) $i = y\ j$ **if** $j < 1$ **and** $i \in B\ j$ **for** $i\ j$
proof–
have $i \in B\ 0$ **using** *that* **by** *blast*
then have $i \in$ *?B0* **unfolding** *B-def* **by** *auto*
then have $(\forall s < t.\ L\ s\ i = s)$ **by** *blast*
moreover have $y\ 0 < t$ **using** ‹$y \in$ *cube* $1\ t$› **unfolding** *cube-def* **by** *auto*
ultimately have $L\ (y\ 0)\ i = y\ 0$ **by** *simp*
then show *?L* $y\ i = y\ j$ **using** *that* **using** ‹$y \in$ *cube* $1\ t$› **by** *force*
qed

ultimately show $(\forall i \in B\ 1.$ *?L* $y\ i =$ *?f* $i) \wedge (\forall j < 1.\ \forall i$
$\in B\ j.$ (*?L* y) $i = y\ j)$
by *blast*
qed
}
ultimately show *is-subspace ?L* $1\ n\ t$ **unfolding** *is-subspace-def* **by** *blast*
qed

lemma *line-is-dim1-subspace*:
assumes $n > 0$
and $t > 0$
and *is-line* $L\ n\ t$
shows *is-subspace* (*restrict* ($\lambda y.\ L\ (y\ 0)$) (*cube* $1\ t$)) $1\ n\ t$
using *line-is-dim1-subspace-t-1*[*of n L*] *line-is-dim1-subspace-t-ge-1*[*of n t L*] *assms not-less-iff-gr-or-eq* **by** *blast*

The key property of the existence of a minimal dimension N, such that for any r-colouring in $C_t^{N'}$ (for $N' \geq N$) there exists a monochromatic line is defined in the following using the variables:

r:	*nat*	the number of colours
t:	*nat*	the size of of the base

definition *hj*
where *hj* $r\ t \equiv (\exists N>0.\ \forall N' \geq N.\ \forall \chi.\ \chi \in$ (*cube* N'
$t) \rightarrow_E \{..<r::nat\} \longrightarrow (\exists L.\ \exists c<r.$ *is-line* $L\ N'\ t$
$\wedge (\forall y \in L\ `\ \{..<t\}.\ \chi\ y = c)))$

The key property of the existence of a minimal dimension N, such that for any r-colouring in $C_t^{N'}$ (for $N' \geq N$) there exists a layered subspace of dimension k is defined in the following using the variables:

r:	*nat*	the number of colours
t:	*nat*	the size of of the base
k:	*nat*	the dimension of the subspace

definition *lhj*
where $lhj\ r\ t\ k \equiv (\exists N > 0.\ \forall N' \geq N.\ \forall \chi.\ \chi \in (cube\ N'\ (t + 1)) \rightarrow_E \{..<r::nat\} \longrightarrow (\exists S.\ layered\text{-}subspace\ S\ k\ N'\ t\ r\ \chi))$

We state some useful facts about 1-dimensional subspaces.

lemma *dim1-subspace-elims*:
assumes $disjoint\text{-}family\text{-}on\ B\ \{..1::nat\}$ **and** $\bigcup(B\ `\ \{..1::nat\}) = \{..<n\}$ **and** $(\{\} \notin B\ `\ \{..<1::nat\})$ **and** $f \in (B\ 1) \rightarrow_E \{..<t\}$ **and** $S \in (cube\ 1\ t) \rightarrow_E (cube\ n\ t)$ **and** $(\forall y \in cube\ 1\ t.\ (\forall i \in B\ 1.\ S\ y\ i = f\ i) \wedge (\forall j{<}1.\ \forall i \in B\ j.\ (S\ y)\ i = y\ j))$
shows $B\ 0 \cup B\ 1 = \{..<n\}$
and $B\ 0 \cap B\ 1 = \{\}$
and $(\forall y \in cube\ 1\ t.\ (\forall i \in B\ 1.\ S\ y\ i = f\ i) \wedge (\forall i \in B\ 0.\ (S\ y)\ i = y\ 0))$
and $B\ 0 \neq \{\}$
proof −
have $\{..1\} = \{0::nat,\ 1\}$ **by** *auto*
then show $B\ 0 \cup B\ 1 = \{..<n\}$ **using** *assms*(*2*) **by** *simp*
next
show $B\ 0 \cap B\ 1 = \{\}$ **using** *assms*(*1*) **unfolding** *disjoint-family-on-def* **by** *simp*
next
show $(\forall y \in cube\ 1\ t.\ (\forall i \in B\ 1.\ S\ y\ i = f\ i) \wedge (\forall i \in B\ 0.\ (S\ y)\ i = y\ 0))$
using *assms*(*6*) **by** *simp*
next
show $B\ 0 \neq \{\}$ **using** *assms*(*3*) **by** *auto*
qed

We state some properties of cubes.

lemma *cube-props*:
assumes $s < t$
shows $\exists p \in cube\ 1\ t.\ p\ 0 = s$
and $(SOME\ p.\ p \in cube\ 1\ t \wedge p\ 0 = s)\ 0 = s$
and $(\lambda s{\in}\{..<t\}.\ S\ (SOME\ p.\ p{\in}cube\ 1\ t \wedge p\ 0 = s))\ s = (\lambda s{\in}\{..<t\}.\ S\ (SOME\ p.\ p{\in}cube\ 1\ t \wedge p\ 0 = s))\ ((SOME\ p.\ p \in cube\ 1\ t \wedge p\ 0 = s)\ 0)$
and $(SOME\ p.\ p \in cube\ 1\ t \wedge p\ 0 = s) \in cube\ 1\ t$
proof −
show *1*: $\exists p \in cube\ 1\ t.\ p\ 0 = s$ **using** *assms* **unfolding** *cube-def* **by** (*simp add*: *fun-ex*)
show *2*: $(SOME\ p.\ p \in cube\ 1\ t \wedge p\ 0 = s)\ 0 = s$ **using** *assms 1 someI-ex*[*of* $\lambda x.\ x \in cube\ 1\ t \wedge x\ 0 = s$] **by** *blast*
show *3*: $(\lambda s{\in}\{..<t\}.\ S\ (SOME\ p.\ p{\in}cube\ 1\ t \wedge p\ 0 = s))\ s = (\lambda s{\in}\{..<t\}.\ S\ (SOME\ p.\ p{\in}cube\ 1\ t \wedge p\ 0 = s))\ ((SOME\ p.\ p \in cube\ 1\ t$

$\wedge$ *p 0* = *s*) *0*) **using** *2* **by** *simp*
show *4*: (*SOME p. p* $\in$ *cube 1 t* $\wedge$ *p 0* = *s*) $\in$ *cube 1 t* **using** *1 someI-ex*[*of* λ*p. p* $\in$ *cube 1 t* $\wedge$ *p 0* = *s*] *assms* **by** *blast*
qed

The following lemma relates 1-dimensional subspaces to lines, thus establishing a bidirectional correspondence between the two together with *line-is-dim1-subspace*.

lemma *dim1-subspace-is-line*:
assumes *t* > *0*
and *is-subspace S 1 n t*
shows *is-line* (λ*s*$\in${..<*t*}. *S* (*SOME p. p*$\in$*cube 1 t* $\wedge$ *p 0* = *s*)) *n t*
proof−
define *L* **where** *L* $\equiv$ (λ*s*$\in${..<*t*}. *S* (*SOME p. p*$\in$*cube 1 t* $\wedge$ *p 0* = *s*))
have {..*1*} = {*0*::*nat*, *1*} **by** *auto*
obtain *B f* **where** *Bf-props*: *disjoint-family-on B* {..*1*::*nat*} $\wedge$ $\bigcup$(*B* ' {..*1*::*nat*}) =
{..<*n*} $\wedge$ ({} $\notin$ *B* ' {..<*1*::*nat*}) $\wedge$ *f* $\in$ (*B 1*) $\to_E$ {..<*t*}
$\wedge$ *S* $\in$ (*cube 1 t*) $\to_E$ (*cube n t*) $\wedge$ ($\forall$ *y* $\in$ *cube 1 t*.
($\forall$ *i* $\in$ *B 1*. *S y i* = *f i*) $\wedge$ ($\forall$ *j*<*1*. $\forall$ *i* $\in$ *B j*. (*S y*) *i* = *y j*))
using *assms*(*2*) **unfolding** *is-subspace-def* **by** *auto*
then have *1*: *B 0* $\cup$ *B 1* = {..<*n*} $\wedge$ *B 0* $\cap$ *B 1* = {} **using** *dim1-subspace-elims*(*1*, *2*)[*of B n f t S*] **by** *simp*

have *L* $\in$ {..<*t*} $\to_E$ *cube n t*
proof
fix *s* **assume** *a*: *s* $\in$ {..<*t*}
then have *L s* = *S* (*SOME p. p*$\in$*cube 1 t* $\wedge$ *p 0* = *s*) **unfolding** *L-def* **by** *simp*
moreover have (*SOME p. p*$\in$*cube 1 t* $\wedge$ *p 0* = *s*) $\in$ *cube 1 t* **using** *cube-props*(*1*) *a*
someI-ex[*of* λ*p. p* $\in$ *cube 1 t* $\wedge$ *p 0* = *s*] **by** *blast*
moreover have *S* (*SOME p. p*$\in$*cube 1 t* $\wedge$ *p 0* = *s*) $\in$ *cube n t*
using *assms*(*2*) *calculation*(*2*) *is-subspace-def* **by** *auto*
ultimately show *L s* $\in$ *cube n t* **by** *simp*
next
fix *s* **assume** *a*: *s* $\notin$ {..<*t*}
then show *L s* = *undefined* **unfolding** *L-def* **by** *simp*
qed
moreover have ($\forall$ *x*<*t*. $\forall$ *y*<*t*. *L x j* = *L y j*) $\vee$ ($\forall$ *s*<*t*. *L s j* = *s*) **if** *j* < *n* **for** *j*
proof−
consider *j* $\in$ *B 0* | *j* $\in$ *B 1* **using** ‹*j* < *n*› *1* **by** *blast*
then show ($\forall$ *x*<*t*. $\forall$ *y*<*t*. *L x j* = *L y j*) $\vee$ ($\forall$ *s*<*t*. *L s j* = *s*)
proof (*cases*)
case *1*
have *L s j* = *s* **if** *s* < *t* **for** *s*
proof−
have $\forall$ *y* $\in$ *cube 1 t*. (*S y*) *j* = *y 0* **using** *Bf-props 1* **by** *simp*
then show *L s j* = *s* **using** *that cube-props*(*2*,*4*) **unfolding** *L-def* **by** *auto*
qed
then show *?thesis* **by** *blast*

```
    next
      case 2
      have L x j = L y j if x < t and y < t for x y
      proof−
        have ∗: S y j = f j if y ∈ cube 1 t for y using 2 that Bf-props by simp
      then have L y j = f j using that(2) cube-props(2,4) lessThan-iff restrict-apply
unfolding L-def by fastforce
       moreover from ∗ have L x j = f j using that(1) cube-props(2,4) lessThan-iff
restrict-apply unfolding L-def
          by fastforce
        ultimately show L x j = L y j by simp
      qed
      then show ?thesis by blast
    qed
  qed
  moreover have (∃j<n. ∀s<t. (L s j = s))
  proof −
    obtain j where j-prop: j ∈ B 0 ∧ j < n using Bf-props by blast
    then have (S y) j = y 0 if y ∈ cube 1 t for y using that Bf-props by auto
   then have L s j = s if s < t for s using that cube-props(2,4) unfolding L-def
by auto
    then show ∃j<n. ∀s<t. (L s j = s) using j-prop by blast
  qed
  ultimately show is-line (λs∈{..<t}. S (SOME p. p∈cube 1 t ∧ p 0 = s)) n t
    unfolding L-def is-line-def by auto
qed

lemma bij-unique-inv:
  assumes bij-betw f A B
    and x ∈ B
  shows ∃!y ∈ A. (the-inv-into A f) x = y
  using assms unfolding bij-betw-def inj-on-def the-inv-into-def
  by blast

lemma inv-into-cube-props:
  assumes s < t
  shows the-inv-into (cube 1 t) (λf. f 0) s ∈ cube 1 t
    and the-inv-into (cube 1 t) (λf. f 0) s 0 = s
  using assms bij-unique-inv one-dim-cube-eq-nat-set f-the-inv-into-f-bij-betw
  by fastforce+

lemma some-inv-into:
  assumes s < t
  shows (SOME p. p∈cube 1 t ∧ p 0 = s) = (the-inv-into (cube 1 t) (λf. f 0) s)
   using inv-into-cube-props[of s t] one-dim-cube-eq-nat-set[of t] assms unfolding
bij-betw-def inj-on-def by auto

lemma some-inv-into-2:
  assumes s < t
```

shows (*SOME p. p∈cube 1 (t+1) ∧ p 0 = s*) = (*the-inv-into* (*cube 1 t*) (*λf. f 0*) *s*)
proof–
have ∗: (*SOME p. p∈cube 1 (t+1) ∧ p 0 = s*) ∈ *cube 1 (t+1)* **using** *cube-props assms* **by** *simp*
then have (*SOME p. p∈cube 1 (t+1) ∧ p 0 = s*) *0 = s* **using** *cube-props assms* **by** *simp*
moreover
{
have (*SOME p. p∈cube 1 (t+1) ∧ p 0 = s*) ' {..<*1*} ⊆ {..<*t*} **using** *calculation assms* **by** *force*
then have (*SOME p. p∈cube 1 (t+1) ∧ p 0 = s*) ∈ *cube 1 t* **using** ∗ **unfolding** *cube-def* **by** *auto*
}
moreover have *inj-on* (*λf. f 0*) (*cube 1 t*) **using** *one-dim-cube-eq-nat-set*[*of t*]
unfolding *bij-betw-def inj-on-def* **by** *auto*
ultimately show (*SOME p. p∈cube 1 (t+1) ∧ p 0 = s*) = (*the-inv-into* (*cube 1 t*) (*λf. f 0*) *s*)
using *the-inv-into-f-eq* [*of λf. f 0 cube 1 t* (*SOME p. p∈cube 1 (t+1) ∧ p 0 = s*) *s*] **by** *auto*
qed

lemma *dim1-layered-subspace-as-line*:
assumes *t > 0*
and *layered-subspace S 1 n t r χ*
shows ∃ *c1 c2. c1<r ∧ c2<r ∧* (∀ *s<t. χ* (*S* (*SOME p. p∈cube 1*
(*t+1*) *∧ p 0 = s*)) *= c1*) *∧ χ* (*S* (*SOME p. p∈cube 1 (t+1) ∧ p 0 = t*)) *= c2*
proof –
have *x u < t* **if** *x ∈ classes 1 t 0* **and** *u < 1* **for** *x u*
proof –
have *x ∈ cube 1 (t+1)* **using** *that* **unfolding** *classes-def* **by** *blast*
then have *x u ∈* {..<*t+1*} **using** *that* **unfolding** *cube-def* **by** *blast*
then have *x u ∈* {..<*t*} **using** *that*
using *that less-Suc-eq* **unfolding** *classes-def* **by** *auto*
then show *x u < t* **by** *simp*
qed
then have *classes 1 t 0 ⊆ cube 1 t* **unfolding** *cube-def classes-def* **by** *auto*
moreover have *cube 1 t ⊆ classes 1 t 0* **using** *cube-subset*[*of 1 t*] **unfolding** *cube-def classes-def* **by** *auto*
ultimately have *X*: *classes 1 t 0 = cube 1 t* **by** *blast*

obtain *c1* **where** *c1-prop*: *c1 < r ∧* (∀ *x∈classes 1 t 0. χ* (*S x*) *= c1*) **using** *assms*(*2*)
unfolding *layered-subspace-def* **by** *blast*
then have (*χ* (*S x*) *= c1*) **if** *x ∈ cube 1 t* **for** *x* **using** *X that* **by** *blast*
then have *χ* (*S* (*the-inv-into* (*cube 1 t*) (*λf. f 0*) *s*)) *= c1* **if** *s < t* **for** *s*
using *one-dim-cube-eq-nat-set*[*of t*] **by** (*meson that bij-betwE bij-betw-the-inv-into lessThan-iff*)
then have *K1*: *χ* (*S* (*SOME p. p∈cube 1 (t+1) ∧ p 0 = s*)) *= c1* **if** *s < t* **for** *s*

using *that some-inv-into-2* **by** *simp*

have ∗: $\exists\, c{<}r.\ \forall\, x \in$ *classes 1 t 1*. χ $(S\ x) = c$
using *assms*(*2*) **unfolding** *layered-subspace-def* **by** *blast*

have $x\ 0 = t$ **if** $x \in$ *classes 1 t 1* **for** x **using** *that* **unfolding** *classes-def* **by** *simp*
moreover have $\exists !x \in$ *cube 1* $(t{+}1)$. $x\ 0 = t$ **using** *one-dim-cube-eq-nat-set*[*of* $t{+}1$]
unfolding *bij-betw-def inj-on-def* **using** *inv-into-cube-props*(*1*) *inv-into-cube-props*(*2*) **by** *force*
moreover have ∗∗: $\exists !x.\ x \in$ *classes 1 t 1* **unfolding** *classes-def* **using** *calculation*(*2*) **by** *simp*
ultimately have *the-inv-into* (*cube 1* $(t{+}1)$) $(\lambda f.\ f\ 0)$ $t \in$ *classes 1 t 1*
using *inv-into-cube-props*[*of t t+1*] **unfolding** *classes-def* **by** *simp*

then have $\exists\, c2.\ c2 < r \wedge \chi$ (S (*the-inv-into* (*cube 1* $(t{+}1)$) $(\lambda f.\ f\ 0)$ t)) $= c2$
using ∗ ∗∗ **by** *blast*
then have *K2*: $\exists\, c2.\ c2 < r \wedge \chi$ (S (*SOME p*. $p{\in}$*cube 1* $(t{+}1) \wedge p\ 0 = t$)) $= c2$
using *some-inv-into* **by** *simp*

from *K1 K2* **show** *?thesis*
using *c1-prop* **by** *blast*
qed

lemma *dim1-layered-subspace-mono-line*:
assumes $t > 0$
and *layered-subspace S 1 n t r* χ
shows $\forall\, s{<}t.\ \forall\, l{<}t.\ \ \chi$ (S (*SOME p*. $p{\in}$*cube 1* $(t{+}1) \wedge p\ 0 = s$)) $=$
χ (S (*SOME p*. $p{\in}$*cube 1* $(t{+}1) \wedge p\ 0 = l$)) $\wedge\ \ \chi$ (S (*SOME p*. $p{\in}$*cube 1* $(t{+}1) \wedge p\ 0 = s$)) $< r$
using *dim1-layered-subspace-as-line*[*of t S n r* χ] *assms* **by** *auto*

definition *join* :: $(nat \Rightarrow {'a}) \Rightarrow (nat \Rightarrow {'a}) \Rightarrow nat \Rightarrow nat \Rightarrow (nat \Rightarrow {'a})$
where
join f g n m $\equiv$ (λx. *if* $x \in \{..{<}n\}$ *then f x else* (*if* $x \in \{n..{<}n{+}m\}$ *then g* $(x - n)$ *else undefined*))

lemma *join-cubes*:
assumes $f \in$ *cube n* $(t{+}1)$
and $g \in$ *cube m* $(t{+}1)$
shows *join f g n m* $\in$ *cube* $(n{+}m)$ $(t{+}1)$
proof (*unfold cube-def*; *intro PiE-I*)
fix i
assume $i \in \{..{<}n{+}m\}$
then consider $i < n \mid i \geq n \wedge i < n{+}m$ **by** *fastforce*
then show *join f g n m i* $\in \{..{<}t + 1\}$
proof (*cases*)

case *1*
then have $join\ f\ g\ n\ m\ i = f\ i$ **unfolding** *join-def* **by** *simp*
moreover have $f\ i \in \{..<t+1\}$ **using** *assms*(*1*) *1* **unfolding** *cube-def* **by** *blast*
ultimately show *?thesis* **by** *simp*
next
case *2*
then have $join\ f\ g\ n\ m\ i = g\ (i - n)$ **unfolding** *join-def* **by** *simp*
moreover have $i - n \in \{..<m\}$ **using** *2* **by** *auto*
moreover have $g\ (i - n) \in \{..<t+1\}$ **using** *calculation*(*2*) *assms*(*2*) **unfolding** *cube-def* **by** *blast*
ultimately show *?thesis* **by** *simp*
qed
next
fix *i*
assume $i \notin \{..<n+m\}$
then show $join\ f\ g\ n\ m\ i = undefined$ **unfolding** *join-def* **by** *simp*
qed

lemma *subspace-elems-embed*:
assumes *is-subspace S k n t*
shows $S\ `\ (cube\ k\ t) \subseteq cube\ n\ t$
using *assms* **unfolding** *cube-def is-subspace-def* **by** *blast*

2 Core proofs

The numbering of the theorems has been borrowed from the textbook [1].

2.1 Theorem 4

2.1.1 Base case of Theorem 4

lemma *hj-imp-lhj-base*:
fixes *r t*
assumes $t > 0$
and $\bigwedge r'.\ hj\ r'\ t$
shows *lhj r t 1*
proof–
from *assms*(*2*) **obtain** *N* **where** *N-def*: $N > 0 \wedge (\forall N' \geq N.\ \forall \chi.\ \chi \in (cube\ N'\ t) \rightarrow_E \{..<r::nat\} \longrightarrow (\exists L.\ \exists c<r.\ is\text{-}line\ L\ N'\ t \wedge (\forall y \in L\ `\ \{..<t\}.\ \chi\ y = c)))$ **unfolding** *hj-def* **by** *blast*

have $(\exists S.\ is\text{-}subspace\ S\ 1\ N'\ (t + 1) \wedge (\forall i \in \{..1\}.\ \exists c < r.\ (\forall x \in classes\ 1\ t\ i.\ \chi\ (S\ x) = c)))$ **if** *asm*: $N' \geq N\ \ \chi \in (cube\ N'\ (t + 1)) \rightarrow_E \{..<r::nat\}$ **for** $N'\ \chi$
proof–
have *N′-props*: $N' > 0 \wedge (\forall \chi.\ \chi \in (cube\ N'\ t) \rightarrow_E \{..<r::nat\} \longrightarrow (\exists L.\ \exists c<r.\ is\text{-}line\ L\ N'\ t \wedge (\forall y \in L\ `\ \{..<t\}.\ \chi\ y = c)))$ **using** *asm N-def* **by** *simp*

let *?chi-t* = $\lambda x \in$ *cube* N' *t*. χ *x*
have *?chi-t* $\in$ *cube* N' *t* $\rightarrow_E$ {..<*r*::*nat*} **using** *cube-subset asm* **by** *auto*
then obtain *L* **where** *L-def*: *is-line* *L* N' *t* $\wedge$ ($\exists$ *c*<*r*. ($\forall$ *y* $\in$ *L* ' {..<*t*}. *?chi-t* *y* = *c*))
using N'*-props* **by** *blast*

have *is-subspace* (*restrict* (λy. *L* (*y* *0*)) (*cube* *1* *t*)) *1* N' *t* **using** *line-is-dim1-subspace* N'*-props L-def*
using *assms*(*1*) **by** *auto*
then obtain *B f* **where** *Bf-defs*: *disjoint-family-on B* {..*1*} $\wedge$ $\bigcup$(*B* ' {..*1*}) = {..<N'}
$\wedge$ ({} $\notin$ *B* ' {..<*1*}) $\wedge$ *f* $\in$ (*B* *1*) $\rightarrow_E$ {..<*t*} $\wedge$
(*restrict* (λy. *L* (*y* *0*)) (*cube* *1* *t*)) $\in$ (*cube* *1* *t*) $\rightarrow_E$ (*cube* N' *t*)
$\wedge$ ($\forall$ *y* $\in$ *cube* *1* *t*. ($\forall$ *i* $\in$ *B* *1*. (*restrict* (λy. *L* (*y* *0*)) (*cube* *1* *t*)) *y* *i* = *f* *i*) $\wedge$ ($\forall$ *j*<*1*. $\forall$ *i* $\in$ *B* *j*. ((*restrict* (λy. *L* (*y* *0*)) (*cube* *1* *t*)) *y*) *i* = *y* *j*)) **unfolding** *is-subspace-def* **by** *auto*

have {..*1*::*nat*} = {*0*, *1*} **by** *auto*
then have *B-props*: *B* *0* $\cup$ *B* *1* = {..<N'} $\wedge$ (*B* *0* $\cap$ *B* *1* = {})
using *Bf-defs* **unfolding** *disjoint-family-on-def* **by** *auto*
define L' **where** $L' \equiv$ *L*(*t*:=(λj. *if* *j* $\in$ *B* *1* *then* *L* (*t* − *1*) *j* *else* (*if* *j* $\in$ *B* *0* *then* *t* *else* *undefined*)))

S1 is the corresponding 1-dimensional subspace of L'.

define *S1* **where** *S1* $\equiv$ *restrict* (λy. L' (*y* (*0*::*nat*))) (*cube* *1* (*t*+*1*))
have *line-prop*: *is-line* L' N' (*t* + *1*)
proof−
have *A1*: $L' \in$ {..<*t*+*1*} $\rightarrow_E$ *cube* N' (*t* + *1*)
proof
fix *x*
assume *asm*: *x* $\in$ {..<*t* + *1*}
then show L' *x* $\in$ *cube* N' (*t* + *1*)
proof (*cases* *x* < *t*)
case *True*
then have L' *x* = *L* *x* **by** (*simp add*: L'*-def*)
then have L' *x* $\in$ *cube* N' *t* **using** *L-def True* **unfolding** *is-line-def* **by** *auto*
then show L' *x* $\in$ *cube* N' (*t* + *1*) **using** *cube-subset* **by** *blast*
next
case *False*
then have *x* = *t* **using** *asm* **by** *simp*
show L' *x* $\in$ *cube* N' (*t* + *1*)
proof(*unfold cube-def*, *intro PiE-I*)
fix *j*
assume *j* $\in$ {..<N'}
have *j* $\in$ *B* *1* $\vee$ *j* $\in$ *B* *0* $\vee$ *j* $\notin$ (*B* *0* $\cup$ *B* *1*) **by** *blast*
then show L' *x* *j* $\in$ {..<*t* + *1*}
proof (*elim disjE*)
assume *j* $\in$ *B* *1*

```
          then have L′ x j = L (t − 1) j
            by (simp add: ‹x = t› L′-def)
          have L (t − 1) ∈ cube N′ t using line-points-in-cube L-def
            by (meson assms(1) diff-less less-numeral-extra(1))
           then have L (t − 1) j < t using ‹j ∈ {..<N′}› unfolding cube-def
by auto
          then show L′ x j ∈ {..<t + 1} using ‹L′ x j = L (t − 1) j› by simp
        next
          assume j ∈ B 0
         then have j ∉ B 1 using Bf-defs unfolding disjoint-family-on-def by
auto
          then have L′ x j = t  by (simp add: ‹j ∈ B 0› ‹x = t› L′-def)
          then show L′ x j ∈ {..<t + 1} by simp
        next
          assume a: j ∉ (B 0 ∪ B 1)
          have {..1::nat} = {0, 1} by auto
          then have B 0 ∪ B 1 = (⋃(B ‘ {..1::nat})) by simp
       then have B 0 ∪ B 1 = {..<N′} using Bf-defs unfolding partition-on-def
by simp
          then have ¬(j ∈ {..<N′}) using a by simp
          then have False using ‹j ∈ {..<N′}› by simp
          then show ?thesis by simp
        qed
      next
        fix j
        assume j ∉ {..<N′}
       then have j ∉ (B 0) ∧ j ∉ B 1 using Bf-defs unfolding partition-on-def
by auto
        then show L′ x j = undefined using ‹x = t› by (simp add: L′-def)
      qed
    qed
  next
    fix x
    assume asm: x ∉ {..<t+1}
    then have x ∉ {..<t} ∧ x ≠ t by simp
    then show L′ x = undefined using L-def unfolding L′-def is-line-def by
auto
  qed
  have A2: (∃j<N′. (∀s < (t + 1). L′ s j = s))
  proof (cases t = 1)
    case True
    obtain j where j-prop: j ∈ B 0 ∧ j < N′ using Bf-defs by blast
    then have L′ s j = L s j if s < t for s using that by (auto simp: L′-def)
     moreover have L s j = 0 if s < t for s  using that True L-def j-prop
line-points-in-cube-unfolded[of L N′ t]
      by simp
     moreover have L′ s j = s if s < t for s using True calculation that by
simp
    moreover have L′ t j = t using j-prop B-props by (auto simp: L′-def)
```

ultimately show *?thesis* **unfolding** *L′-def* **using** *j-prop* **by** *auto*
next
case *False*
then show *?thesis*
proof–
have $(\exists j<N'.\ (\forall s < t.\ L'\ s\ j = s))$ **using** *L-def* **unfolding** *is-line-def* **by** (*auto simp*: *L′-def*)
then obtain *j* **where** *j-def*: $j < N' \wedge (\forall s < t.\ L'\ s\ j = s)$ **by** *blast*
have $j \notin B\ 1$
proof
assume *a*:$j \in B\ 1$
then have $(restrict\ (\lambda y.\ L\ (y\ 0))\ (cube\ 1\ t))\ y\ j = f\ j$ **if** $y \in cube\ 1\ t$ **for** *y*
using *Bf-defs that* **by** *simp*
then have $L\ (y\ 0)\ j = f\ j$ **if** $y \in cube\ 1\ t$ **for** *y* **using** *that* **by** *simp*
moreover have $\exists!i.\ i < t \wedge y\ 0 = i$ **if** $y \in cube\ 1\ t$ **for** *y*
using *that one-dim-cube-eq-nat-set*[*of t*] **unfolding** *bij-betw-def* **by** *blast*
moreover have $\exists!y.\ y \in cube\ 1\ t \wedge y\ 0 = i$ **if** $i < t$ **for** *i*
proof (*intro ex1I-alt*)
define *y* **where** $y \equiv (\lambda x{::}nat.\ \lambda y{\in}\{..<1{::}nat\}.\ x)$
have $y\ i \in (cube\ 1\ t)$ **using** *that* **unfolding** *cube-def y-def* **by** *simp*
moreover have $y\ i\ 0 = i$ **unfolding** *y-def* **by** *simp*
moreover have $z = y\ i$ **if** $z \in cube\ 1\ t$ **and** $z\ 0 = i$ **for** *z*
proof (*rule ccontr*)
assume $z \neq y\ i$
then obtain *l* **where** *l-prop*: $z\ l \neq y\ i\ l$ **by** *blast*
consider $l \in \{..<1{::}nat\} \mid l \notin \{..<1{::}nat\}$ **by** *blast*
then show *False*
proof *cases*
case *1*
then show *?thesis* **using** *l-prop that*(*2*) **unfolding** *y-def* **by** *auto*
next
case *2*
then have $z\ l = undefined$ **using** *that* **unfolding** *cube-def* **by** *blast*
moreover have $y\ i\ l = undefined$ **unfolding** *y-def* **using** *2* **by** *auto*
ultimately show *?thesis* **using** *l-prop* **by** *presburger*
qed
qed
ultimately show $\exists y.\ (y \in cube\ 1\ t \wedge y\ 0 = i) \wedge (\forall ya.\ ya \in cube\ 1\ t \wedge ya\ 0 = i \longrightarrow y = ya)$ **by** *blast*
qed

moreover have $L\ i\ j = f\ j$ **if** $i < t$ **for** *i* **using** *that calculation* **by** *blast*
moreover have $(\exists j<N'.\ (\forall s < t.\ L\ s\ j = s))$ **using**
‹$(\exists j<N'.\ (\forall s < t.\ L'\ s\ j = s))$› **by** (*auto simp*: *L′-def*)
ultimately show *False* **using** *False*
by (*metis* (*no-types*, *lifting*) *L′-def assms*(*1*) *fun-upd-apply j-def less-one nat-neq-iff*)
qed

```
        then have j ∈ B 0 using ‹j ∉ B 1› j-def B-props by auto

        then have L' t j = t using ‹j ∉ B 1› by (auto simp: L'-def)
        then have L' s j = s if s < t + 1 for s using j-def that by (auto simp:
L'-def)
        then show ?thesis using j-def by blast
      qed
    qed
    have A3: (∀x<t+1. ∀y<t+1. L' x j =  L' y j) ∨ (∀s<t+1. L' s j = s) if j
< N' for j
    proof−
      consider j ∈ B 1 | j ∈ B 0 using ‹j < N'› B-props by auto
      then show (∀x<t+1. ∀y<t+1. L' x j =  L' y j) ∨ (∀s<t+1. L' s j = s)
      proof (cases)
        case 1
       then have (restrict (λy. L (y 0)) (cube 1 t)) y j = f j if y ∈ cube 1 t for y
          using that Bf-defs by simp
        moreover have ∃!i. i < t ∧ y 0 = i if y ∈ cube 1 t for y
          using that one-dim-cube-eq-nat-set[of t] unfolding bij-betw-def by blast
        moreover have ∃!y. y ∈ cube 1 t ∧ y 0 = i if i < t for i
        proof (intro ex1I-alt)
          define y where y ≡ (λx::nat. λy∈{..<1::nat}. x)
          have y i ∈ (cube 1 t) using that unfolding cube-def y-def by simp
          moreover have y i 0 = i unfolding y-def by auto
          moreover have z = y i if z ∈ cube 1 t and z 0 = i for z
          proof (rule ccontr)
            assume z ≠ y i
            then obtain l where l-prop: z l ≠ y i l by blast
            consider l ∈ {..<1::nat} | l ∉ {..<1::nat} by blast
            then show False
            proof cases
              case 1
              then show ?thesis using l-prop that(2) unfolding y-def by auto
            next
              case 2
              then have z l = undefined using that unfolding cube-def by blast
              moreover have y i l = undefined unfolding y-def using 2 by auto
              ultimately show ?thesis using l-prop by presburger
            qed
          qed
          ultimately show ∃y. (y ∈ cube 1 t ∧ y 0 = i) ∧ (∀ya. ya
          ∈ cube 1 t ∧ ya 0 = i ⟶ y = ya) by blast

        qed
        moreover have L i j = f j if i < t for i using calculation that by force
       moreover have  L i j = L x j if x < t i < t for x i using that calculation
by simp
        moreover have L' x j = L x j if x < t for x using that fun-upd-other[of
x t L
```

```
        λj. if j ∈ B 1 then L (t − 1) j else if j ∈ B 0 then t else undefined]
      unfolding L′-def by simp
    ultimately have ∗: L′ x j = L′ y j if x < t y < t for x y using that by
presburger

    have L′ t j = L′ (t − 1) j using ‹j ∈ B 1› by (auto simp: L′-def)
   also have ... = L′ x j if x < t for x using ∗ by (simp add: assms(1) that)
    finally have ∗∗: L′ t j = L′ x j if x < t for x using that by auto
    have L′ x j = L′ y j if x < t + 1 y < t + 1 for x y
    proof−
     consider x < t ∧ y = t | y < t ∧ x = t | x = t ∧ y = t | x < t ∧ y < t
        using ‹x < t + 1› ‹y < t + 1› by linarith
      then show L′ x j = L′ y j
      proof cases
        case 1
        then show ?thesis using ∗∗ by auto
      next
        case 2
        then show ?thesis using ∗∗ by auto
      next
        case 3
        then show ?thesis by simp
      next
        case 4
        then show ?thesis using ∗ by auto
      qed
    qed
    then show ?thesis by blast
  next
    case 2
    then have ∀ y ∈ cube 1 t. ((restrict (λy. L (y 0)) (cube 1 t)) y) j = y 0
      using ‹j ∈ B 0› Bf-defs by auto
    then have ∀ y ∈ cube 1 t. L (y 0) j = y 0  by auto
    moreover have ∃!y. y ∈ cube 1 t ∧ y 0 = i if i < t for i
    proof (intro ex1I-alt)
      define y where y ≡ (λx::nat. λy∈{..<1::nat}. x)
      have y i ∈ (cube 1 t) using that unfolding cube-def y-def by simp
      moreover have y i 0 = i unfolding y-def by auto
      moreover have z = y i if z ∈ cube 1 t and z 0 = i for z
      proof (rule ccontr)
        assume z ≠ y i
        then obtain l where l-prop: z l ≠ y i l by blast
        consider l ∈ {..<1::nat} | l ∉ {..<1::nat} by blast
        then show False
        proof cases
          case 1
          then show ?thesis using l-prop that(2) unfolding y-def by auto
        next
          case 2
```

```
            then have z l = undefined using that unfolding cube-def by blast
            moreover have y i l = undefined unfolding y-def using 2 by auto
            ultimately show ?thesis using l-prop by presburger
          qed
        qed
        ultimately show ∃ y. (y ∈ cube 1 t ∧ y 0 = i) ∧ (∀ ya. ya
        ∈ cube 1 t ∧ ya 0 = i ⟶ y = ya) by blast

      qed
      ultimately have L s j = s if s < t for s using that by blast
      then have L′ s j = s if s < t for s using that by (auto simp: L′-def)
      moreover have L′ t j = t using 2 B-props by (auto simp: L′-def)
      ultimately have L′ s j = s if s < t+1 for s using that by (auto simp:
L′-def)
      then show ?thesis by blast
    qed
  qed
  from A1 A2 A3 show ?thesis unfolding is-line-def by simp
qed
then have F1: is-subspace S1 1 N′ (t + 1) unfolding S1-def
  using line-is-dim1-subspace[of N′ t+1] N′-props assms(1) by force
moreover have F2: ∃ c < r. (∀ x ∈ classes 1 t i. χ (S1 x) = c) if i ≤ 1 for i
proof−
  have ∃ c < r. (∀ y ∈ L′ ‘ {..<t}. ?chi-t y = c) unfolding L′-def using L-def
by fastforce
  have ∀ x ∈ (L ‘ {..<t}). x ∈ cube N′ t using L-def
    using line-points-in-cube by blast
  then have ∀ x ∈ (L′ ‘ {..<t}). x ∈ cube N′ t by (auto simp: L′-def)
  then have ∗:∀ x ∈ (L′ ‘ {..<t}). χ x = ?chi-t x by simp
  then have ?chi-t ‘ (L′ ‘ {..<t}) = χ ‘ (L′ ‘ {..<t}) by force
  then have ∃ c < r. (∀ y ∈ L′ ‘ {..<t}. χ y = c) using
      ‹∃ c < r. (∀ y ∈ L′ ‘ {..<t}. ?chi-t y = c)› by fastforce
  then obtain linecol where lc-def: linecol < r ∧ (∀ y ∈ L′ ‘ {..<t}. χ y =
linecol) by blast
  consider i = 0 | i = 1 using ‹i ≤ 1› by linarith
  then show ∃ c < r. (∀ x ∈ classes 1 t i. χ (S1 x) = c)
  proof (cases)
    case 1
    assume i = 0
    have ∗: ∀ a t. a ∈ {..<t+1} ∧ a ≠ t ⟷ a ∈ {..<(t::nat)} by auto
    from ‹i = 0› have classes 1 t 0 = {x . x ∈ (cube 1 (t + 1)) ∧
    (∀ u ∈ {((1::nat) − 0)..<1}. x u = t) ∧ t ∉ x ‘ {..<(1 − (0::nat))}}
      using classes-def by simp
    also have ... = {x . x ∈ cube 1 (t+1) ∧ t ∉ x ‘ {..<(1::nat)}} by simp
    also have ... = {x . x ∈ cube 1 (t+1) ∧ (x 0 ≠ t)} by blast
    also have ... = {x . x ∈ cube 1 (t+1) ∧ (x 0 ∈ {..<t+1} ∧ x 0 ≠ t)}
      unfolding cube-def by blast
    also have ... = {x . x ∈ cube 1 (t+1) ∧ (x 0 ∈ {..<t})} using ∗ by simp
    finally have redef: classes 1 t 0 = {x . x ∈ cube 1 (t+1) ∧ (x 0 ∈ {..<t})}
```

by *simp*
have $\{x\ 0 \mid x\ .\ x \in classes\ 1\ t\ 0\} \subseteq \{..<t\}$ **using** *redef* **by** *auto*
moreover have $\{..<t\} \subseteq \{x\ 0 \mid x\ .\ x \in classes\ 1\ t\ 0\}$
proof
fix x **assume** x: $x \in \{..<t\}$
hence $\exists\, a{\in}cube\ 1\ t.\ a\ 0 = x$
unfolding *cube-def* **by** (*intro fun-ex*) *auto*
then show $x \in \{x\ 0 \mid x.\ x \in classes\ 1\ t\ 0\}$
using *x cube-subset* **unfolding** *redef* **by** *auto*
qed
ultimately have $**$: $\{x\ 0 \mid x\ .\ x \in classes\ 1\ t\ 0\} = \{..<t\}$ **by** *blast*

have $\chi\ (S1\ x) = linecol$ **if** $x \in classes\ 1\ t\ 0$ **for** x
proof–
have $x \in cube\ 1\ (t{+}1)$ **unfolding** *classes-def* **using** *that redef* **by** *blast*
then have $S1\ x = L'\ (x\ 0)$ **unfolding** *S1-def* **by** *simp*
moreover have $x\ 0 \in \{..<t\}$ **using** $**$ **using** ‹$x \in classes\ 1\ t\ 0$› **by** *blast*
ultimately show $\chi\ (S1\ x) = linecol$ **using** *lc-def* **using** *fun-upd-triv image-eqI* **by** *blast*
qed
then show *?thesis* **using** *lc-def* ‹$i = 0$› **by** *auto*
next
case *2*
assume $i = 1$
have $classes\ 1\ t\ 1 = \{x\ .\ x \in (cube\ 1\ (t + 1)) \wedge (\forall\, u \in \{0{::}nat..<1\}.\ x\ u = t) \wedge t \notin x\ `\ \{..<0\}\}$ **unfolding** *classes-def* **by** *simp*
also have ... $= \{x\ .\ x \in cube\ 1\ (t{+}1) \wedge (\forall\, u \in \{0\}.\ x\ u = t)\}$ **by** *simp*
finally have *redef*: $classes\ 1\ t\ 1 = \{x\ .\ x \in cube\ 1\ (t{+}1) \wedge (x\ 0 = t)\}$ **by** *auto*
have $\forall\, s \in \{..<t{+}1\}.\ \exists!x \in cube\ 1\ (t{+}1).\ (\lambda p.\ \lambda y{\in}\{..<1{::}nat\}.\ p)\ s = x$ **using** *nat-set-eq-one-dim-cube*[*of t+1*]
unfolding *bij-betw-def* **by** *blast*
then have $\exists!x \in cube\ 1\ (t{+}1).\ (\lambda p.\ \lambda y{\in}\{..<1{::}nat\}.\ p)\ t = x$ **by** *auto*
then obtain x **where** *x-prop*: $x \in cube\ 1\ (t{+}1)$ **and** $(\lambda p.\ \lambda y{\in}\{..<1{::}nat\}.\ p)\ t = x$ **and** $\forall\, z \in cube\ 1\ (t{+}1).\ (\lambda p.\ \lambda y{\in}\{..<1{::}nat\}.\ p)\ t = z \longrightarrow z = x$ **by** *blast*
then have $(\lambda p.\ \lambda y{\in}\{0\}.\ p)\ \ t\ = x \wedge (\forall\, z \in cube\ 1\ (t{+}1).\ (\lambda p.\ \lambda y{\in}\{0\}.\ p)\ t = z \longrightarrow z = x)$ **by** *force*
then have $*$:$((\lambda p.\ \lambda y{\in}\{0\}.\ p)\ t)\ 0\ = x\ 0 \wedge (\forall\, z \in cube\ 1\ (t{+}1).\ (\lambda p.\ \lambda y{\in}\{0\}.\ p)\ t\ = z\ \longrightarrow z = x)$
using *x-prop* **by** *force*

then have $\exists!y \in cube\ 1\ (t + 1).\ y\ 0 = t$
proof (*intro ex1I-alt*)
define y **where** $y \equiv (\lambda x{::}nat.\ \lambda y{\in}\{..<1{::}nat\}.\ x)$
have $y\ t \in (cube\ 1\ (t + 1))$ **unfolding** *cube-def y-def* **by** *simp*
moreover have $y\ t\ 0 = t$ **unfolding** *y-def* **by** *auto*
moreover have $z = y\ t$ **if** $z \in cube\ 1\ (t + 1)$ **and** $z\ 0 = t$ **for** z
proof (*rule ccontr*)

```
        assume z ≠ y t
        then obtain l where l-prop: z l ≠ y t l by blast
        consider l ∈ {..<1::nat} | l ∉ {..<1::nat} by blast
        then show False
        proof cases
          case 1
          then show ?thesis using l-prop that(2) unfolding y-def by auto
        next
          case 2
          then have z l = undefined using that unfolding cube-def by blast
          moreover have y t l = undefined unfolding y-def using 2 by auto
          ultimately show ?thesis using l-prop by presburger
        qed
      qed
      ultimately show ∃y. (y ∈ cube 1 (t + 1) ∧ y 0 = t) ∧ (∀ya.
      ya ∈ cube 1 (t + 1) ∧ ya 0 = t ⟶ y = ya) by blast
    qed
    then have ∃!x ∈ classes 1 t 1. True using redef by simp
     then obtain x where x-def: x ∈ classes 1 t 1 ∧ (∀y ∈ classes 1 t 1. x =
y) by auto

    have χ (S1 y) < r if y ∈ classes 1 t 1 for y
    proof–
      have y = x using x-def that by auto
      then have χ (S1 y) = χ (S1 x) by auto
      moreover have S1 x ∈ cube N′ (t+1) unfolding S1-def is-line-def
        using line-prop line-points-in-cube redef x-def by fastforce
      ultimately show χ (S1 y) < r using asm unfolding cube-def by auto
    qed
    then show ?thesis using lc-def ‹i = 1› using x-def by fast
  qed
 qed
 ultimately show (∃S. is-subspace S 1 N′ (t + 1) ∧ (∀i ∈ {..1}.
 ∃c < r. (∀x ∈ classes 1 t i. χ (S x) = c))) by blast
qed
then show ?thesis using N-def unfolding layered-subspace-def lhj-def by auto
qed
```

2.1.2 Induction step of theorem 4

The proof has four parts:

1. We obtain two layered subspaces of dimension 1 and k (respectively), whose existence is guaranteed by the assumption *lhj* (i.e. the induction hypothesis). Additionally, we prove some useful facts about these.

2. We construct a *k+1*-dimensional subspace with the goal of showing that it is layered.

3. We prove that our construction is a subspace in the first place.

4. We prove that it is a layered subspace.

lemma *hj-imp-lhj-step*:
fixes $r\ k$
assumes $t > 0$
and $k \geq 1$
and *True*
and $(\bigwedge r\ k'.\ k' \leq k \Longrightarrow lhj\ r\ t\ k')$
and $r > 0$
shows $lhj\ r\ t\ (k+1)$
proof–
obtain m **where** *m-props*: $(m > 0 \wedge (\forall M' \geq m.\ \forall \chi.\ \chi \in (cube\ M'\ (t + 1)) \rightarrow_E \{..<r::nat\} \longrightarrow (\exists S.\ layered\text{-}subspace\ S\ k\ M'\ t\ r\ \chi)))$ **using** *assms*(4)[*of k r*] **unfolding** *lhj-def* **by** *blast*
define s **where** $s \equiv r\hat{}((t + 1)\hat{}m)$
obtain n' **where** *n'-props*: $(n' > 0 \wedge (\forall N \geq n'.\ \forall \chi.\ \chi \in (cube\ N\ (t + 1)) \rightarrow_E \{..<s::nat\} \longrightarrow (\exists S.\ layered\text{-}subspace\ S\ 1\ N\ t\ s\ \chi)))$ **using** *assms*(2) *assms*(4)[*of 1 s*] **unfolding** *lhj-def* **by** *auto*

have $(\exists T.\ layered\text{-}subspace\ T\ (k + 1)\ (M')\ t\ r\ \chi)$ **if** *χ-prop*: $\chi \in cube\ M'\ (t + 1) \rightarrow_E \{..<r\}$ **and** *M'-prop*: $M' \geq n' + m$ **for** $\chi\ M'$
proof –
define d **where** $d \equiv M' - (n' + m)$
define n **where** $n \equiv n' + d$
have $n \geq n'$ **unfolding** *n-def d-def* **by** *simp*
have $n + m = M'$ **unfolding** *n-def d-def* **using** *M'-prop* **by** *simp*
have *line-subspace-s*: $\exists S.\ layered\text{-}subspace\ S\ 1\ n\ t\ s\ \chi \wedge is\text{-}line\ (\lambda s{\in}\{..<t{+}1\}.\ S\ (SOME\ p.\ p{\in}cube\ 1\ (t{+}1) \wedge p\ 0 = s))\ n\ (t{+}1)$ **if** $\chi \in (cube\ n\ (t + 1)) \rightarrow_E \{..<s::nat\}$ **for** χ
proof–
have $\exists S.\ layered\text{-}subspace\ S\ 1\ n\ t\ s\ \chi$ **using** *that n'-props* ‹$n \geq n'$› **by** *blast*
then obtain L **where** $layered\text{-}subspace\ L\ 1\ n\ t\ s\ \chi$ **by** *blast*
then have $is\text{-}subspace\ L\ 1\ n\ (t{+}1)$ **unfolding** *layered-subspace-def* **by** *simp*
then have $is\text{-}line\ (\lambda s{\in}\{..<t{+}1\}.\ L\ (SOME\ p.\ p{\in}cube\ 1\ (t{+}1) \wedge p\ 0 = s))\ n\ (t + 1)$
using *dim1-subspace-is-line*[*of t+1 L n*] *assms*(1) **by** *simp*
then show $\exists S.\ layered\text{-}subspace\ S\ 1\ n\ t\ s\ \chi \wedge is\text{-}line\ (\lambda s{\in}\{..<t + 1\}.\ S\ (SOME\ p.\ p \in cube\ 1\ (t{+}1) \wedge p\ 0 = s))\ n\ (t + 1)$ **using** ‹$layered\text{-}subspace\ L\ 1\ n\ t\ s\ \chi$› **by** *auto*
qed

Part 1: Obtaining the subspaces L and S

Recall that *lhj* claims the existence of a layered subspace for any colouring (of a fixed size, where the size of a colouring refers to the number of colours). Therefore, the colourings have to be defined first, before the layered subspaces can be obtained. The colouring χL here is χ^* in the book [1], an s-colouring; see the fact *s-coloured* a couple of lines below.

define χL **where** $\chi L \equiv (\lambda x \in cube\ n\ (t{+}1).\ (\lambda y \in cube\ m$

$(t + 1)$. χ (*join x y n m*)))
have *A*: $\forall x \in$ *cube n* $(t+1)$. $\forall y \in$ *cube m* $(t+1)$. χ (*join x y n m*) $\in \{..<r\}$
proof(*safe*)
fix *x y*
assume $x \in$ *cube n* $(t+1)$ $y \in$ *cube m* $(t+1)$
then have *join x y n m* $\in$ *cube* $(n+m)$ $(t+1)$ **using** *join-cubes*[*of x n t y m*] **by** *simp*
then show χ (*join x y n m*) $< r$ **using** χ*-prop* ‹$n + m = M'$› **by** *blast*
qed
have χ*L-prop*: $\chi L \in$ *cube n* $(t+1) \rightarrow_E$ *cube m* $(t+1) \rightarrow_E \{..<r\}$
using *A* **by** (*auto simp*: χ*L-def*)

have *card* (*cube m* $(t+1) \rightarrow_E \{..<r\}$) $=$ (*card* $\{..<r\}$) ^(*card* (*cube m* $(t+1)$))

using *card-PiE*[*of cube m* $(t + 1)$ λ-. $\{..<r\}$] **by** (*simp add*: *cube-def finite-PiE*)
also have ... $= r$ ^(*card* (*cube m* $(t+1)$)) **by** *simp*
also have ... $= r$ ^($(t+1)$^m) **using** *cube-card* **unfolding** *cube-def* **by** *simp*
finally have *card* (*cube m* $(t+1) \rightarrow_E \{..<r\}$) $= r$ ^($(t+1)$^m) **.**
then have *s-coloured*: *card* (*cube m* $(t+1) \rightarrow_E \{..<r\}$) $= s$ **unfolding** *s-def* **by** *simp*
have $s > 0$ **using** *assms*(*5*) **unfolding** *s-def* **by** *simp*
then obtain φ **where** φ*-prop*: *bij-betw* φ (*cube m* $(t+1) \rightarrow_E \{..<r\}$) $\{..<s\}$
using *assms*(*5*) *ex-bij-betw-nat-finite-2*[*of cube m* $(t+1) \rightarrow_E \{..<r\}$ *s*] *s-coloured* **by** *blast*
define χ*L-s* **where** χ*L-s* $\equiv$ ($\lambda x{\in}$*cube n* $(t+1)$. φ (χL *x*))
have χ*L-s* $\in$ *cube n* $(t+1) \rightarrow_E \{..<s\}$
proof
fix *x* **assume** *a*: $x \in$ *cube n* $(t+1)$
then have χ*L-s x* $= \varphi$ (χL *x*) **unfolding** χ*L-s-def* **by** *simp*
moreover have χL *x* $\in$ (*cube m* $(t+1) \rightarrow_E \{..<r\}$)
using *a* χ*L-def* χ*L-prop* **unfolding** χ*L-def* **by** *blast*
moreover have φ (χL *x*) $\in \{..<s\}$ **using** φ*-prop calculation*(*2*) **unfolding** *bij-betw-def* **by** *blast*
ultimately show χ*L-s x* $\in \{..<s\}$ **by** *auto*
qed (*auto simp*: χ*L-s-def*)

L is the layered line which we obtain from the monochromatic line guaranteed to exist by the assumption *hj s t*.

then obtain *L* **where** *L-prop*: *layered-subspace L 1 n t s* χ*L-s* **using** *line-subspace-s* **by** *blast*
define *L-line* **where** *L-line* $\equiv$ ($\lambda s{\in}\{..<t+1\}$. *L* (*SOME p. p*$\in$*cube 1* $(t+1) \wedge p$ $0 = s$))
have *L-line-base-prop*: $\forall s \in \{..<t+1\}$. *L-line s* $\in$ *cube n* $(t+1)$
using *assms*(*1*) *dim1-subspace-is-line*[*of t+1 L n*] *L-prop line-points-in-cube*[*of L-line n t+1*]
unfolding *layered-subspace-def L-line-def* **by** *auto*

Here, χS is χ^{**} in the book [1], an r-colouring.

define χS **where** $\chi S \equiv$ ($\lambda y{\in}$*cube m* $(t+1)$. χ (*join* (*L-line 0*) *y n m*))

have $\chi S \in$ (*cube m* (*t* + *1*)) $\rightarrow_E$ {..<*r*::*nat*}
proof
fix *x* **assume** *a*: $x \in$ *cube m* (*t*+*1*)
then have χS *x* = χ (*join* (*L-line 0*) *x n m*) **unfolding** χ*S-def* **by** *simp*
moreover have *L-line 0* = *L* (*SOME p. p*$\in$*cube 1* (*t*+*1*) $\wedge$ *p 0* = *0*)
using *L-prop assms*(*1*) **unfolding** *L-line-def* **by** *simp*
moreover have (*SOME p. p*$\in$*cube 1* (*t*+*1*) $\wedge$ *p 0* = *0*) $\in$ *cube 1* (*t*+*1*) **using** *cube-props*(*4*)[*of 0 t*+*1*]
using *assms*(*1*) **by** *auto*
moreover have *L* $\in$ *cube 1* (*t*+*1*) $\rightarrow_E$ *cube n* (*t*+*1*)
using *L-prop* **unfolding** *layered-subspace-def is-subspace-def* **by** *blast*
moreover have *L* (*SOME p. p*$\in$*cube 1* (*t*+*1*) $\wedge$ *p 0* = *0*) $\in$ *cube n* (*t*+*1*)
using *calculation* (*3*,*4*) **unfolding** *cube-def* **by** *auto*
moreover have *join* (*L-line 0*) *x n m* $\in$ *cube* (*n* + *m*) (*t*+*1*) **using** *join-cubes a calculation*(*2*, *5*) **by** *auto*
ultimately show χS *x* $\in$ {..<*r*} **using** *A a* **by** *fastforce*
qed (*auto simp*: χ*S-def*)

S is the k-dimensional layered subspace that arises as a consequence of the induction hypothesis. Note that the colouring is χS, an r-colouring.

then obtain *S* **where** *S-prop*: *layered-subspace S k m t r* χS **using** *assms*(*4*) *m-props* **by** *blast*

Remark: *L-Line i* returns the i-th point of the line.

Part 2: Constructing the $(k + 1)$-dimensional subspace T

Below, *Tset* is the set as defined in the book [1]. It represents the $(k+1)$-dimensional subspace. In this construction, subspaces (e.g. T) are functions whose image is a set. See the fact *im-T-eq-Tset* below.

Having obtained our subspaces S and L, we define the $(k + 1)$-dimensional subspace very straightforwardly Namely, T = L ×S. Since we represent tuples by function sets, we need an appropriate operator that mirrors the Cartesian product × for these. We call this *join* and define it for elements of a function set.

define *Tset* **where** *Tset* $\equiv$ {*join* (*L-line i*) *s n m* | *i s* . *i* $\in$ {..<*t*+*1*} $\wedge$ *s* $\in$ *S* ` (*cube k* (*t*+*1*))}
define T' **where** $T' \equiv$ ($\lambda x \in$ *cube 1* (*t*+*1*). $\lambda y \in$ *cube k* (*t*+*1*). *join* (*L-line* (*x 0*)) (*S y*) *n m*)
have T'*-prop*: $T' \in$ *cube 1* (*t*+*1*) $\rightarrow_E$ *cube k* (*t*+*1*) $\rightarrow_E$ *cube* (*n* + *m*) (*t*+*1*)
proof
fix *x* **assume** *a*: $x \in$ *cube 1* (*t*+*1*)
show T' *x* $\in$ *cube k* (*t* + *1*) $\rightarrow_E$ *cube* (*n* + *m*) (*t* + *1*)
proof
fix *y* **assume** *b*: $y \in$ *cube k* (*t*+*1*)
then have T' *x y* = *join* (*L-line* (*x 0*)) (*S y*) *n m* **using** *a* **unfolding** T'*-def* **by** *simp*

moreover have *L-line (x 0)* ∈ *cube n (t+1)* **using** *a L-line-base-prop* **unfolding** *cube-def* **by** *blast*
moreover have *S y* ∈ *cube m (t+1)*
using *subspace-elems-embed[of S k m t+1] S-prop b* **unfolding** *layered-subspace-def* **by** *blast*
ultimately show *T′ x y* ∈ *cube (n + m) (t + 1)* **using** *join-cubes* **by** *presburger*
next
qed (*unfold T′-def*; *use a* **in** *simp*)
qed (*auto simp*: *T′-def*)

define *T* **where** *T* ≡ (λ*x* ∈ *cube (k + 1) (t+1)*. *T′* (λ*y* ∈ {..<*1*}. *x y*) (λ*y* ∈ {..<*k*}. *x (y + 1)*)))
have *T-prop*: *T* ∈ *cube (k+1) (t+1)* $\rightarrow_E$ *cube (n+m) (t+1)*
proof
fix *x* **assume** *a*: *x* ∈ *cube (k+1) (t+1)*
then have *T x* = *T′* (λ*y* ∈ {..<*1*}. *x y*) (λ*y* ∈ {..<*k*}. *x (y + 1)*) **unfolding** *T-def* **by** *auto*
moreover have (λ*y* ∈ {..<*1*}. *x y*) ∈ *cube 1 (t+1)* **using** *a* **unfolding** *cube-def* **by** *auto*
moreover have (λ*y* ∈ {..<*k*}. *x (y + 1)*) ∈ *cube k (t+1)* **using** *a* **unfolding** *cube-def* **by** *auto*
moreover have *T′* (λ*y* ∈ {..<*1*}. *x y*) (λ*y* ∈ {..<*k*}. *x (y + 1)*) ∈ *cube (n + m) (t+1)*
using *T′-prop calculation* **unfolding** *T′-def* **by** *blast*
ultimately show *T x* ∈ *cube (n + m) (t+1)* **by** *argo*
qed (*auto simp*: *T-def*)

have *im-T-eq-Tset*: *T* ‘ *cube (k+1) (t+1)* = *Tset*
proof
show *T* ‘ *cube (k + 1) (t + 1)* ⊆ *Tset*
proof
fix *x* **assume** *x* ∈ *T* ‘ *cube (k+1) (t+1)*
then obtain *y* **where** *y-prop*: *y* ∈ *cube (k+1) (t+1)* ∧ *x* = *T y* **by** *blast*
then have *T y* = *T′* (λ*i* ∈ {..<*1*}. *y i*) (λ*i* ∈ {..<*k*}. *y (i + 1)*) **unfolding** *T-def* **by** *simp*
moreover have (λ*i* ∈ {..<*1*}. *y i*) ∈ *cube 1 (t+1)* **using** *y-prop* **unfolding** *cube-def* **by** *auto*
moreover have (λ*i* ∈ {..<*k*}. *y (i + 1)*) ∈ *cube k (t+1)* **using** *y-prop* **unfolding** *cube-def* **by** *auto*
moreover have *T′* (λ*i* ∈ {..<*1*}. *y i*) (λ*i* ∈ {..<*k*}. *y (i + 1)*) =
join (L-line ((λ*i* ∈ {..<*1*}. *y i*) *0*)) (*S* (λ*i* ∈ {..<*k*}. *y (i + 1)*))) *n m*
using *calculation* **unfolding** *T′-def* **by** *auto*
ultimately have ∗: *T y* = *join (L-line* ((λ*i* ∈ {..<*1*}. *y i*) *0*))
(*S* (λ*i* ∈ {..<*k*}. *y (i + 1)*))) *n m* **by** *simp*

have (λ*i* ∈ {..<*1*}. *y i*) *0* ∈ {..<*t+1*} **using** *y-prop* **unfolding** *cube-def* **by** *auto*
moreover have *S* (λ*i* ∈ {..<*k*}. *y (i + 1)*) ∈ *S* ‘ (*cube k (t+1)*)

```
        using ‹(λi∈{..<k}. y (i + 1)) ∈ cube k (t + 1)› by blast
      ultimately have T y ∈ Tset using ∗ unfolding Tset-def by blast
      then show x ∈ Tset using y-prop by simp
    qed

    show Tset ⊆ T ‘ cube (k + 1) (t + 1)
    proof
      fix x assume x ∈ Tset
      then obtain i sx sxinv where isx-prop: x = join (L-line i) sx n m ∧ i ∈
{..<t+1}
      ∧ sx ∈ S ‘ (cube k (t+1)) ∧ sxinv ∈ cube k (t+1) ∧ S sxinv = sx
        unfolding Tset-def by blast
      let ?f1 = (λj ∈ {..<1::nat}. i)
      let ?f2 = sxinv
      have ?f1 ∈ cube 1 (t+1) using isx-prop unfolding cube-def by simp
      moreover have ?f2 ∈ cube k (t+1) using isx-prop by blast
        moreover have x = join (L-line (?f1 0)) (S ?f2) n m by (simp add:
isx-prop)
      ultimately have ∗: x = T′ ?f1 ?f2 unfolding T′-def by simp

      define f where f ≡ (λj ∈ {1..<k+1}. ?f2 (j − 1))(0:=i)
      have f ∈ cube (k+1) (t+1)
      proof (unfold cube-def; intro PiE-I)
        fix j assume j ∈ {..<k+1}
        then consider j = 0 | j ∈ {1..<k+1} by fastforce
        then show f j ∈ {..<t+1}
        proof (cases)
          case 1
          then have f j = i unfolding f-def by simp
          then show ?thesis using isx-prop by simp
        next
          case 2
          then have j − 1 ∈ {..<k} by auto
          moreover have f j = ?f2 (j − 1) using 2 unfolding f-def by simp
           moreover have ?f2 (j − 1) ∈ {..<t+1} using calculation(1) isx-prop
unfolding cube-def by blast
          ultimately show ?thesis by simp
        qed
      qed (auto simp: f-def)
      have ?f1 = (λj ∈ {..<1}. f j) unfolding f-def using isx-prop by auto
      moreover have ?f2 = (λj∈{..<k}. f (j+1))
        using calculation isx-prop unfolding cube-def f-def by fastforce
     ultimately have T′ ?f1 ?f2 = T f using ‹f ∈ cube (k+1) (t+1)› unfolding
T-def by simp
      then show x ∈ T ‘ cube (k + 1) (t + 1) using ∗
        using ‹f ∈ cube (k + 1) (t + 1)› by blast
    qed
```

qed
have *Tset* ⊆ *cube* (*n* + *m*) (*t+1*)
proof
fix *x* **assume** *a*: *x*∈*Tset*
then obtain *i sx* **where** *isx-props*: *x* = *join* (*L-line i*) *sx n m* ∧ *i* ∈ {..<*t+1*} ∧
sx ∈ *S* ‘ (*cube k* (*t+1*)) **unfolding** *Tset-def* **by** *blast*
then have *L-line i* ∈ *cube n* (*t+1*) **using** *L-line-base-prop* **by** *blast*
moreover have *sx* ∈ *cube m* (*t+1*)
using *subspace-elems-embed*[*of S k m t+1*] *S-prop isx-props* **unfolding** *layered-subspace-def* **by** *blast*
ultimately show *x* ∈ *cube* (*n* + *m*) (*t+1*) **using** *join-cubes*[*of L-line i n t sx m*] *isx-props* **by** *simp*
qed

Part 3: Proving that T is a subspace

To prove something is a subspace, we have to provide the B and f satisfying the subspace properties. We construct BT and fT from BS, fS and BL, fL, which correspond to the k-dimensional subspace S and the 1-dimensional subspace (i.e. line) L, respectively.

obtain *BS fS* **where** *BfS-props*: *disjoint-family-on BS* {..*k*} ⋃(*BS* ‘ {..*k*}) = {..<*m*} ({}
∉ *BS* ‘ {..<*k*}) *fS* ∈ (*BS k*) →*E* {..<*t+1*} *S* ∈ (*cube k* (*t+1*))
→*E* (*cube m* (*t+1*)) (∀ *y* ∈ *cube k* (*t+1*). (∀ *i* ∈ *BS k*.
S y i = *fS i*) ∧ (∀ *j*<*k*. ∀ *i* ∈ *BS j*. (*S y*) *i* = *y j*)) **using** *S-prop*
unfolding *layered-subspace-def is-subspace-def* **by** *auto*

obtain *BL fL* **where** *BfL-props*: *disjoint-family-on BL* {..*1*} ⋃(*BL* ‘ {..*1*}) = {..<*n*}
({} ∉ *BL* ‘ {..<*1*}) *fL* ∈ (*BL 1*) →*E* {..<*t+1*} *L* ∈ (*cube 1*
(*t+1*)) →*E* (*cube n* (*t+1*)) (∀ *y* ∈ *cube 1* (*t+1*). (∀ *i* ∈
BL 1. *L y i* = *fL i*) ∧ (∀ *j*<*1*. ∀ *i* ∈ *BL j*. (*L y*) *i* = *y j*)) **using** *L-prop*
unfolding *layered-subspace-def is-subspace-def* **by** *auto*

define *Bstat* **where** *Bstat* ≡ *set-incr n* (*BS k*) ∪ *BL 1*
define *Bvar* **where** *Bvar* ≡ (λ*i*::*nat*. (*if i* = *0 then BL 0 else set-incr n* (*BS* (*i* − *1*))))
define *BT* **where** *BT* ≡ (λ*i* ∈ {..<*k+1*}. *Bvar i*)((*k+1*):=*Bstat*)
define *fT* **where** *fT* ≡ (λ*x*. (*if x* ∈ *BL 1 then fL x else* (*if x* ∈ *set-incr n*
(*BS k*) *then fS* (*x* − *n*) *else undefined*)))

have *fact1*: *set-incr n* (*BS k*) ∩ *BL 1* = {} **using** *BfL-props BfS-props* **unfolding** *set-incr-def* **by** *auto*
have *fact2*: *BL 0* ∩ (⋃ *i*∈{..<*k*}. *set-incr n* (*BS i*)) = {}
using *BfL-props BfS-props* **unfolding** *set-incr-def* **by** *auto*
have *fact3*: ∀ *i* ∈ {..<*k*}. *BL 0* ∩ *set-incr n* (*BS i*) = {}
using *BfL-props BfS-props* **unfolding** *set-incr-def* **by** *auto*
have *fact4*: ∀ *i* ∈ {..<*k+1*}. ∀ *j* ∈ {..<*k+1*}. *i* ≠ *j*

$\longrightarrow$ *set-incr n* (*BS i*) $\cap$ *set-incr n* (*BS j*) = {}
using *set-incr-disjoint-family*[*of BS k*] *BfS-props* **unfolding** *disjoint-family-on-def* **by** *simp*
have *fact5*: $\forall$ *i* $\in$ {..<*k*+*1*}. *Bvar i* $\cap$ *Bstat* = {}
proof
fix *i* **assume** *a*: *i* $\in$ {..<*k*+*1*}
show *Bvar i* $\cap$ *Bstat* = {}
proof (*cases i*)
case *0*
then have *Bvar i* = *BL 0* **unfolding** *Bvar-def* **by** *simp*
moreover have *BL 0* $\cap$ *BL 1* = {} **using** *BfL-props* **unfolding** *disjoint-family-on-def* **by** *simp*
moreover have *set-incr n* (*BS k*) $\cap$ *BL 0* = {} **using** *BfL-props BfS-props* **unfolding** *set-incr-def* **by** *auto*
ultimately show *?thesis* **unfolding** *Bstat-def* **by** *blast*
next
case (*Suc nat*)
then have *Bvar i* = *set-incr n* (*BS nat*) **unfolding** *Bvar-def* **by** *simp*
moreover have *set-incr n* (*BS nat*) $\cap$ *BL 1* = {} **using** *BfS-props BfL-props a Suc* **unfolding** *set-incr-def*
by *auto*
moreover have *set-incr n* (*BS nat*) $\cap$ *set-incr n* (*BS k*) = {} **using** *a Suc fact4* **by** *simp*
ultimately show *?thesis* **unfolding** *Bstat-def* **by** *blast*
qed
qed

The facts *F1*, ..., *F5* are the disjuncts in the subspace definition.

have *Bvar* ‘ {..<*k*+*1*} = *BL* ‘ {..<*1*} $\cup$ *Bvar* ‘ {*1*..<*k*+*1*} **unfolding** *Bvar-def* **by** *force*
also have ... = *BL* ‘ {..<*1*} $\cup$ {*set-incr n* (*BS i*) | *i* . *i* $\in$ {..<*k*}} **unfolding** *Bvar-def* **by** *fastforce*
moreover have {} $\notin$ *BL* ‘ {..<*1*} **using** *BfL-props* **by** *auto*
moreover have {} $\notin$ {*set-incr n* (*BS i*) | *i* . *i* $\in$ {..<*k*}} **using** *BfS-props*(*2*, *3*) *set-incr-def* **by** *fastforce*
ultimately have {} $\notin$ *Bvar* ‘ {..<*k*+*1*} **by** *simp*
then have *F1*: {} $\notin$ *BT* ‘ {..<*k*+*1*} **unfolding** *BT-def* **by** *simp*
moreover
{
have *F2-aux*: *disjoint-family-on Bvar* {..<*k*+*1*}
proof (*unfold disjoint-family-on-def*; *safe*)
fix *m n x* **assume** *a*: *m* < *k* + *1* *n* < *k* + *1* *m* $\neq$ *n* *x* $\in$ *Bvar m* *x* $\in$ *Bvar n*
show *x* $\in$ {}
proof (*cases n*)
case *0*
then show *?thesis* **using** *a fact3* **unfolding** *Bvar-def* **by** *auto*
next
case (*Suc nnat*)
then have $*$: *n* = *Suc nnat* **by** *simp*

then show *?thesis*
proof (*cases m*)
case *0*
then show *?thesis* **using** *a fact3* **unfolding** *Bvar-def* **by** *auto*
next
case (*Suc mnat*)
then show *?thesis* **using** *a fact4* $*$ **unfolding** *Bvar-def* **by** *fastforce*
qed
qed
qed

have *F2*: *disjoint-family-on BT* $\{..k+1\}$
proof
fix *m n* **assume** *a*: $m\in\{..k+1\}$ $n\in\{..k+1\}$ $m \neq n$
have $\forall x.\ x \in BT\ m \cap BT\ n \longrightarrow x \in \{\}$
proof (*intro allI impI*)
fix *x* **assume** *b*: $x \in BT\ m \cap BT\ n$
have $m < k + 1 \wedge n < k + 1 \vee m = k + 1 \wedge n = k + 1 \vee m < k + 1 \wedge n = k + 1 \vee m = k + 1 \wedge n < k + 1$ **using** *a le-eq-less-or-eq* **by** *auto*
then show $x \in \{\}$
proof (*elim disjE*)
assume *c*: $m < k + 1 \wedge n < k + 1$
then have $BT\ m = Bvar\ m \wedge BT\ n = Bvar\ n$ **unfolding** *BT-def* **by** *simp*
then show $x \in \{\}$ **using** *a b c fact4 F2-aux* **unfolding** *Bvar-def disjoint-family-on-def* **by** *auto*
qed (*use a b fact5* **in** ‹*auto simp*: *BT-def*›)
qed
then show $BT\ m \cap BT\ n = \{\}$ **by** *auto*
qed
}
moreover have *F3*: $\bigcup(BT\ `\ \{..k+1\}) = \{..<n + m\}$
proof
show $\bigcup\ (BT\ `\ \{..k + 1\}) \subseteq \{..<n + m\}$
proof
fix *x* **assume** $x \in \bigcup\ (BT\ `\ \{..k + 1\})$
then obtain *i* **where** *i-prop*: $i \in \{..k+1\} \wedge x \in BT\ i$ **by** *blast*
then consider $i = k +1 \mid i \in \{..<k+1\}$ **by** *fastforce*
then show $x \in \{..<n + m\}$
proof (*cases*)
case *1*
then have $x \in Bstat$ **using** *i-prop* **unfolding** *BT-def* **by** *simp*
then have $x \in BL\ 1 \vee x \in set\text{-}incr\ n\ (BS\ k)$ **unfolding** *Bstat-def* **by** *blast*
then have $x \in \{..<n\} \vee x \in \{n..<n+m\}$ **using** *BfL-props BfS-props*(*2*) *set-incr-image*[*of BS k m n*]
by *blast*
then show *?thesis* **by** *auto*
next

```
          case 2
          then have x ∈ Bvar i using i-prop unfolding BT-def by simp
          then have x ∈ BL 0 ∨ x ∈ set-incr n (BS (i − 1)) unfolding Bvar-def
by presburger
          then show ?thesis
          proof (elim disjE)
            assume x ∈ BL 0
            then have x ∈ {..<n} using BfL-props by auto
            then show x ∈ {..<n + m} by simp
          next
            assume a: x ∈ set-incr n (BS (i − 1))
            then have i − 1 ≤ k
              by (meson atMost-iff i-prop le-diff-conv)
          then have set-incr n (BS (i − 1)) ⊆ {n..<n+m} using set-incr-image[of
BS k m n] BfS-props
              by auto
            then show x ∈ {..<n+m} using a by auto
          qed
        qed
      qed
    next
      show {..<n + m} ⊆ ⋃ (BT ‘ {..k + 1})
      proof
        fix x assume x ∈ {..<n + m}
        then consider x ∈ {..<n} | x ∈ {n..<n+m} by fastforce
        then show x ∈ ⋃ (BT ‘ {..k + 1})
        proof (cases)
          case 1
          have ∗: {..1::nat} = {0, 1::nat} by auto
          from 1 have x ∈ ⋃ (BL ‘ {..1::nat}) using BfL-props by simp
          then have x ∈ BL 0 ∨ x ∈ BL 1 using ∗ by simp
          then show ?thesis
          proof (elim disjE)
            assume x ∈ BL 0
            then have x ∈ Bvar 0 unfolding Bvar-def by simp
            then have x ∈ BT 0 unfolding BT-def by simp
            then show x ∈ ⋃ (BT ‘ {..k + 1}) by auto
          next
            assume x ∈ BL 1
            then have x ∈ Bstat unfolding Bstat-def by simp
            then have x ∈ BT (k+1) unfolding BT-def by simp
            then show x ∈ ⋃ (BT ‘ {..k + 1}) by auto
          qed
        next
          case 2
          then have x ∈ (⋃ i≤k. set-incr n (BS i)) using set-incr-image[of BS k
m n] BfS-props by simp
          then obtain i where i-prop: i ≤ k ∧ x ∈ set-incr n (BS i) by blast
          then consider i = k | i < k by fastforce
```

```
        then show ?thesis
        proof (cases)
          case 1
          then have x ∈ Bstat unfolding Bstat-def using i-prop by auto
          then have x ∈ BT (k+1) unfolding BT-def by simp
          then show ?thesis by auto
        next
          case 2
          then have x ∈ Bvar (i + 1) unfolding Bvar-def using i-prop by simp
          then have x ∈ BT (i + 1) unfolding BT-def using 2 by force
          then show ?thesis using 2 by auto
        qed
      qed
    qed
  qed

  moreover have F4: fT ∈ (BT (k+1)) →E {..<t+1}
  proof
    fix x assume x ∈ BT (k+1)
    then have x ∈ Bstat unfolding BT-def by simp
    then have x ∈ BL 1 ∨ x ∈ set-incr n (BS k) unfolding Bstat-def by auto
    then show fT x ∈ {..<t + 1}
    proof (elim disjE)
      assume x ∈ BL 1
      then have fT x = fL x unfolding fT-def by simp
      then show fT x ∈ {..<t+1} using BfL-props ‹x ∈ BL 1› by auto
    next
      assume a: x ∈ set-incr n (BS k)
      then have fT x = fS (x − n) using fact1 unfolding fT-def by auto
      moreover have x − n ∈ BS k using a unfolding set-incr-def by auto
      ultimately show fT x ∈ {..<t+1} using BfS-props by auto
    qed
  qed(auto simp: BT-def Bstat-def fT-def)
  moreover have F5: ((∀ i ∈ BT (k + 1). T y i = fT i) ∧ (∀ j<k+1.
  ∀ i ∈ BT j. (T y) i = y j)) if y ∈ cube (k + 1) (t + 1) for y
  proof(intro conjI allI impI ballI)
    fix i assume i ∈ BT (k + 1)
    then have i ∈ Bstat unfolding BT-def by simp
     then consider i ∈ set-incr n (BS k) |  i ∈ BL 1 unfolding Bstat-def by
blast
    then show T y i = fT i
    proof (cases)
      case 1
       then have ∃ s<m. i = n + s unfolding set-incr-def using BfS-props(2)
by auto
      then obtain s where s-prop: s < m ∧ i = n + s by blast
      then have ∗:  i ∈ {n..<n+m} by simp
      have i ∉ BL 1 using 1 fact1 by auto
      then have fT i = fS (i − n) using 1 unfolding fT-def by simp
```

then have $**$: *fT i* = *fS s* **using** *s-prop* **by** *simp*

have *XX*: ($\lambda z \in \{..<k\}$. *y* (*z* + *1*)) $\in$ *cube k* (*t+1*) **using** *split-cube that* **by** *simp*
have *XY*: *s* $\in$ *BS k* **using** *s-prop 1* **unfolding** *set-incr-def* **by** *auto*

from *that* **have** *T y i* = (T' ($\lambda z \in \{..<1\}$. *y z*) ($\lambda z \in \{..<k\}$. *y* (*z* + *1*))) *i*
unfolding *T-def* **by** *auto*
also have ... = (*join* (*L-line* (($\lambda z \in \{..<1\}$. *y z*) *0*)) (*S* ($\lambda z \in \{..<k\}$. *y* (*z* + *1*))) *n m*) *i* **using** *split-cube that* **unfolding** T'*-def* **by** *simp*
also have ... = (*join* (*L-line* (*y 0*)) (*S* ($\lambda z \in \{..<k\}$. *y* (*z* + *1*))) *n m*) *i* **by** *simp*
also have ... = (*S* ($\lambda z \in \{..<k\}$. *y* (*z* + *1*))) *s* **using** $*$ *s-prop* **unfolding** *join-def* **by** *simp*
also have ... = *fS s* **using** *XX XY BfS-props*(*6*) **by** *blast*
finally show *?thesis* **using** $**$ **by** *simp*
next
case *2*
have *XZ*: *y 0* $\in \{..<t+1\}$ **using** *that* **unfolding** *cube-def* **by** *auto*
have *XY*: *i* $\in \{..<n\}$ **using** *2 BfL-props*(*2*) **by** *blast*
have *XX*: ($\lambda z \in \{..<1\}$. *y z*) $\in$ *cube 1* (*t+1*) **using** *that split-cube* **by** *simp*

have *some-eq-restrict*: (*SOME p. p*$\in$*cube 1* (*t+1*) $\wedge$ *p 0* = (($\lambda z \in \{..<1\}$. *y z*) *0*)) = ($\lambda z \in \{..<1\}$. *y z*)
proof
show *restrict y* $\{..<1\}$ $\in$ *cube 1* (*t* + *1*) $\wedge$ *restrict y* $\{..<1\}$ *0* = *restrict y* $\{..<1\}$ *0*
using *XX* **by** *simp*
next
fix *p*
assume *p* $\in$ *cube 1* (*t+1*) $\wedge$ *p 0* = *restrict y* $\{..<1\}$ *0*
moreover have *p u* = *restrict y* $\{..<1\}$ *u* **if** *u* $\notin \{..<1\}$ **for** *u*
using *that calculation XX* **unfolding** *cube-def*
using *PiE-arb*[*of restrict y* $\{..<1\}$ $\{..<1\}$ λx. $\{..<t + 1\}$ *u*]
PiE-arb[*of p* $\{..<1\}$ λx. $\{..<t + 1\}$ *u*] **by** *simp*
ultimately show *p* = *restrict y* $\{..<1\}$ **by** *auto*
qed

from *that* **have** *T y i* = (T' ($\lambda z \in \{..<1\}$. *y z*) ($\lambda z \in \{..<k\}$. *y* (*z* + *1*))) *i*
unfolding *T-def* **by** *auto*
also have ... = (*join* (*L-line* (($\lambda z \in \{..<1\}$. *y z*) *0*)) (*S* ($\lambda z \in \{..<k\}$. *y* (*z* + *1*))) *n m*) *i*
using *split-cube that* **unfolding** T'*-def* **by** *simp*
also have ... = (*L-line* (($\lambda z \in \{..<1\}$. *y z*) *0*)) *i* **using** *XY* **unfolding** *join-def* **by** *simp*
also have ... = *L* (*SOME p. p*$\in$*cube 1* (*t+1*) $\wedge$ *p 0* = (($\lambda z \in \{..<1\}$. *y z*) *0*)) *i*
using *XZ* **unfolding** *L-line-def* **by** *auto*
also have ... = *L* ($\lambda z \in \{..<1\}$. *y z*) *i* **using** *some-eq-restrict* **by** *simp*

```
      also have ... = fL i using BfL-props(6) XX 2 by blast
      also have ... = fT i using 2 unfolding fT-def by simp
      finally show ?thesis .
    qed
  next
    fix j i assume j < k + 1 i ∈ BT j
    then have i-prop: i ∈ Bvar j unfolding BT-def by auto
    consider j = 0 | j > 0 by auto
    then show T y i = y j
    proof cases
      case 1
      then have i ∈ BL 0 using i-prop unfolding Bvar-def by auto
      then have XY: i ∈ {..<n} using 1 BfL-props(2) by blast
     have XX: (λz ∈ {..<1}. y z)  ∈ cube 1 (t+1) using that split-cube by simp
      have XZ: y 0 ∈ {..<t+1} using that unfolding cube-def by auto

      have some-eq-restrict: (SOME p. p∈cube 1 (t+1) ∧ p 0 = ((λz ∈ {..<1}.
      y z) 0)) = (λz ∈ {..<1}. y z)
      proof
        show restrict y {..<1} ∈ cube 1 (t + 1) ∧ restrict y {..<1} 0 = restrict y
{..<1} 0 using XX by simp
      next
        fix p
        assume p ∈ cube 1 (t+1) ∧ p 0 = restrict y {..<1} 0
        moreover have p u = restrict y {..<1} u if u ∉ {..<1} for u
          using that calculation XX unfolding cube-def
          using PiE-arb[of restrict y {..<1} {..<1} λx. {..<t + 1} u]
            PiE-arb[of p {..<1} λx. {..<t + 1} u] by simp
        ultimately show p = restrict y {..<1} by auto
      qed

     from that have T y i = (T' (λz ∈ {..<1}. y z) (λz ∈ {..<k}. y (z + 1))) i
        unfolding T-def by auto
      also have ... = (join (L-line ((λz ∈ {..<1}. y z) 0)) (S (λz ∈ {..<k}. y (z
+ 1))) n m) i
        using split-cube that unfolding T'-def by simp
        also have ... = (L-line ((λz ∈ {..<1}. y z) 0)) i using XY unfolding
join-def by simp
       also have ... = L (SOME p. p∈cube 1 (t+1) ∧ p 0 = ((λz ∈ {..<1}. y z)
0)) i
        using XZ unfolding L-line-def by auto
      also have ... = L (λz ∈ {..<1}. y z) i using some-eq-restrict by simp
     also have ... =  (λz ∈ {..<1}. y z) j using BfL-props(6) XX 1 ‹i ∈ BL 0›
by blast
      also have ... = (λz ∈ {..<1}. y z) 0 using 1 by blast
      also have ... = y 0 by simp
      also have ... = y j using 1 by simp
      finally show ?thesis .
    next
```

case *2*
then have $i \in$ *set-incr n* (*BS* (*j* − *1*)) **using** *i-prop* **unfolding** *Bvar-def* **by** *simp*
then have $\exists s{<}m.\ n + s = i$ **using** *BfS-props*(*2*) ‹*j* < *k* + *1*› **unfolding** *set-incr-def* **by** *force*
then obtain *s* **where** *s-prop*: $s < m$ $i = s + n$ **by** *auto*
then have ∗: $i \in \{n..<n+m\}$ **by** *simp*

have *XX*: ($\lambda z \in \{..<k\}.\ y\ (z + 1)$) $\in$ *cube k* (*t+1*) **using** *split-cube that* **by** *simp*
have *XY*: $s \in$ *BS* (*j* − *1*) **using** *s-prop 2* ‹$i \in$ *set-incr n* (*BS* (*j* − *1*))›
unfolding *set-incr-def* **by** *force*

from *that* **have** *T y i* = (*T′* ($\lambda z \in \{..<1\}.\ y\ z$) ($\lambda z \in \{..<k\}.\ y\ (z + 1)$)) *i*
unfolding *T-def* **by** *auto*
also have ... = (*join* (*L-line* (($\lambda z \in \{..<1\}.\ y\ z$) *0*)) (*S* ($\lambda z \in \{..<k\}.\ y\ (z + 1)$)) *n m*) *i*
using *split-cube that* **unfolding** *T′-def* **by** *simp*
also have ... = (*join* (*L-line* (*y 0*)) (*S* ($\lambda z \in \{..<k\}.\ y\ (z + 1)$)) *n m*) *i* **by** *simp*
also have ... = (*S* ($\lambda z \in \{..<k\}.\ y\ (z + 1)$)) *s* **using** ∗ *s-prop* **unfolding** *join-def* **by** *simp*
also have ... = ($\lambda z \in \{..<k\}.\ y\ (z + 1)$) (*j*−*1*)
using *XX XY BfS-props*(*6*) *2* ‹*j* < *k* + *1*› **by** *auto*
also have ... = *y j* **using** *2* ‹*j* < *k* + *1*› **by** *force*
finally show *?thesis* **.**
qed
qed

ultimately have *subspace-T*: *is-subspace T* (*k+1*) (*n+m*) (*t+1*) **unfolding** *is-subspace-def* **using** *T-prop* **by** *metis*

Part 4: Proving T is layered

The following redefinition of the classes makes proving the layered property easier.

define *T-class* **where** *T-class* ≡ ($\lambda j{\in}\{..k\}$. {*join* (*L-line i*) *s n m* | *i s* . $i \in \{..<t\} \wedge s \in S$ ‘ (*classes k t j*)})(*k+1*:= {*join* (*L-line t*) (*SOME s.* $s \in S$ ‘ (*cube m* (*t+1*))) *n m*})
have *classprop*: *T-class j* = *T* ‘ *classes* (*k* + *1*) *t j* **if** *j-prop*: $j \le k$ **for** *j*
proof
show *T-class j* ⊆ *T* ‘ *classes* (*k* + *1*) *t j*
proof
fix *x* **assume** $x \in$ *T-class j*
from *that* **have** *T-class j* = {*join* (*L-line i*) *s n m* | *i s* . $i \in \{..<t\} \wedge s \in S$ ‘ (*classes k t j*)}
unfolding *T-class-def* **by** *simp*
then obtain *i s* **where** *is-defs*: *x* = *join* (*L-line i*) *s n m* ∧ *i* < *t* ∧ $s \in S$ ‘ (*classes k t j*)

using ‹x ∈ *T-class j*› **unfolding** *T-class-def* **by** *auto*
moreover have ∗:*classes k t j* ⊆ *cube k* (*t+1*) **unfolding** *classes-def* **by** *simp*
moreover have ∃!*y*. *y* ∈ *classes k t j* ∧ *s* = *S y*
using *subspace-inj-on-cube*[*of S k m t+1*] *S-prop inj-onD*[*of S cube k* (*t+1*)] *calculation*
unfolding *layered-subspace-def inj-on-def* **by** *blast*
ultimately obtain *y* **where** *y-prop*: *y* ∈ *classes k t j* ∧ *s* = *S y* ∧
(∀ *z*∈*classes k t j*. *s* = *S z* ⟶ *y* = *z*) **by** *auto*

define *p* **where** *p* ≡ *join* (λ*g*∈{..<1}. *i*) *y 1 k*
have (λ*g*∈{..<1}. *i*) ∈ *cube 1* (*t+1*) **using** *is-defs* **unfolding** *cube-def* **by** *simp*
then have *p-in-cube*: *p* ∈ *cube* (*k* + *1*) (*t+1*)
using *join-cubes*[*of* (λ*g*∈{..<1}. *i*) *1 t y k*] *y-prop* ∗ **unfolding** *p-def* **by** *auto*
then have ∗∗: *p 0* = *i* ∧ (∀ *l* < *k*. *p* (*l* + *1*) = *y l*) **unfolding** *p-def join-def* **by** *simp*

have *t* ∉ *y* ‘ {..<(*k* − *j*)} **using** *y-prop* **unfolding** *classes-def* **by** *simp*
then have ∀ *u* < *k* − *j*. *y u* ≠ *t* **by** *auto*
then have ∀ *u* < *k* − *j*. *p* (*u* + *1*) ≠ *t* **using** ∗∗ **by** *simp*
moreover have *p 0* ≠ *t* **using** *is-defs* ∗∗ **by** *simp*
moreover have ∀ *u* < *k* − *j* + *1*. *p u* ≠ *t*
using *calculation* **by** (*auto simp*: *algebra-simps less-Suc-eq-0-disj*)
ultimately have ∀ *u* < (*k* + *1*) − *j*. *p u* ≠ *t* **using** *that* **by** *auto*
then have *A1*: *t* ∉ *p* ‘ {..<((*k+1*) − *j*)} **by** *blast*

have *p u* = *t* **if** *u* ∈ {*k* − *j* + *1*..<*k+1*} **for** *u*
proof −
from *that* **have** *u* − *1* ∈ {*k* − *j*..<*k*} **by** *auto*
then have *y* (*u* − *1*) = *t* **using** *y-prop* **unfolding** *classes-def* **by** *blast*
then show *p u* = *t* **using** ∗∗ *that* ‹*u* − *1* ∈ {*k* − *j*..<*k*}› **by** *auto*
qed
then have *A2*: ∀ *u*∈{(*k+1*) − *j*..<*k+1*}. *p u* = *t* **using** *that* **by** *auto*

from *A1 A2 p-in-cube* **have** *p* ∈ *classes* (*k+1*) *t j* **unfolding** *classes-def* **by** *blast*

moreover have *x* = *T p*
proof−
have *loc-useful*:(λ*y* ∈ {..<*k*}. *p* (*y* + *1*)) = (λ*z* ∈ {..<*k*}. *y z*) **using** ∗∗ **by** *auto*
have *T p* = *T′* (λ*y* ∈ {..<1}. *p y*) (λ*y* ∈ {..<*k*}. *p* (*y* + *1*))
using *p-in-cube* **unfolding** *T-def* **by** *auto*

have *T′* (λ*y* ∈ {..<1}. *p y*) (λ*y* ∈ {..<*k*}. *p* (*y* + *1*))
= *join* (*L-line* ((λ*y* ∈ {..<1}. *p y*) *0*)) (*S* (λ*y* ∈ {..<*k*}. *p* (*y* + *1*))) *n*

m
using *split-cube p-in-cube* **unfolding** *T′-def* **by** *simp*
also have ... = *join* (*L-line* (*p 0*)) (*S* ($\lambda y \in \{..<k\}$. *p* (*y* + *1*))) *n m* **by** *simp*
also have ... = *join* (*L-line i*) (*S* ($\lambda y \in \{..<k\}$. *p* (*y* + *1*))) *n m* **by** (*simp add*: ∗∗)
also have ... = *join* (*L-line i*) (*S* ($\lambda z \in \{..<k\}$. *y z*)) *n m* **using** *loc-useful* **by** *simp*
also have ... = *join* (*L-line i*) (*S y*) *n m* **using** *y-prop* ∗ **unfolding** *cube-def* **by** *auto*
also have ... = *x* **using** *is-defs y-prop* **by** *simp*
finally show *x* = *T p*
using ‹*T p* = *T′* (*restrict p* {..<*1*}) ($\lambda y \in \{..<k\}$. *p* (*y* + *1*))› **by** *presburger*
qed
ultimately show *x* ∈ *T* ‘ *classes* (*k* + *1*) *t j* **by** *blast*
qed
next
show *T* ‘ *classes* (*k* + *1*) *t j* ⊆ *T-class j*
proof
fix *x* **assume** *x* ∈ *T* ‘ *classes* (*k*+*1*) *t j*
then obtain *y* **where** *y-prop*: *y* ∈ *classes* (*k*+*1*) *t j* ∧ *T y* = *x* **by** *blast*
then have *y-props*: (∀ *u* ∈ {((*k*+*1*)−*j*)..<*k*+*1*}. *y u* = *t*) ∧ *t* ∉ *y* ‘ {..<(*k*+*1*) − *j* }
unfolding *classes-def* **by** *blast*

define *z* **where** *z* ≡ ($\lambda v \in \{..<k\}$. *y* (*v*+*1*))
have *z* ∈ *cube k* (*t*+*1*) **using** *y-prop classes-subset-cube*[*of k*+*1 t j*] **unfolding** *z-def cube-def* **by** *auto*
moreover
{
have *z* ‘ {..<*k* − *j*} = *y* ‘ ((+) *1* ‘ {..<*k*−*j*}) **unfolding** *z-def* **by** *fastforce*
also have ... = *y* ‘ {*1*..<*k*−*j*+*1*} **by** (*simp add*: *atLeastLessThanSuc-atLeastAtMost image-Suc-lessThan*)
also have ... = *y* ‘ {*1*..<(*k*+*1*)−*j*} **using** *j-prop* **by** *auto*
finally have *z* ‘ {..<*k* − *j*} ⊆ *y* ‘ {..<(*k*+*1*)−*j*} **by** *auto*
then have *t* ∉ *z* ‘ {..<*k* − *j*} **using** *y-props* **by** *blast*

}
moreover have ∀ *u* ∈ {*k*−*j*..<*k*}. *z u* = *t* **unfolding** *z-def* **using** *y-props* **by** *auto*
ultimately have *z-in-classes*: *z* ∈ *classes k t j* **unfolding** *classes-def* **by** *blast*

have *y 0* ≠ *t*
proof−
from *that* **have** *0* ∈ {..<*k* + *1* − *j*} **by** *simp*
then show *y 0* ≠ *t* **using** *y-props* **by** *blast*
qed
then have *tr*: *y 0* < *t* **using** *y-prop classes-subset-cube*[*of k*+*1 t j*] **unfolding**

cube-def **by** *fastforce*

have $(\lambda g \in \{..<1\}.\ y\ g) \in cube\ 1\ (t+1)$
using *y-prop classes-subset-cube*[*of* $k+1\ t\ j$] *cube-restrict*[*of* $1\ (k+1)\ y\ t+1$] *assms*(*2*) **by** *auto*
then have $T\ y = T'\ (\lambda g \in \{..<1\}.\ y\ g)\ z$ **using** *y-prop classes-subset-cube*[*of* $k+1\ t\ j$]
unfolding *T-def z-def* **by** *auto*
also have $... = join\ (L\text{-}line\ ((\lambda g \in \{..<1\}.\ y\ g)\ 0))\ (S\ z)\ n\ m$
unfolding *T′-def*
using ‹$(\lambda g \in \{..<1\}.\ y\ g) \in cube\ 1\ (t+1)$› ‹$z \in cube\ k\ (t+1)$›
by *auto*
also have $... = join\ (L\text{-}line\ (y\ 0))\ (S\ z)\ n\ m$ **by** *simp*
also have $... \in T\text{-}class\ j$ **using** *tr z-in-classes that* **unfolding** *T-class-def* **by** *force*
finally show $x \in T\text{-}class\ j$ **using** *y-prop* **by** *simp*
qed
qed

The core case $i \leq k$. The case $i = k + 1$ is trivial since $k + 1$ has only one point.

have $\chi\ x = \chi\ y \wedge \chi\ x < r$ **if** *a*: $i \leq k\ x \in T\ `\ classes\ (k+1)\ t\ i$
$y \in T\ `\ classes\ (k+1)\ t\ i$ **for** $i\ x\ y$
proof–
from *a* **have** $*$: $T\ `\ classes\ (k+1)\ t\ i = T\text{-}class\ i$ **by** (*simp add*: *classprop*)
then have $x \in T\text{-}class\ i$ **using** *that* **by** *simp*
moreover have $**$: $T\text{-}class\ i = \{join\ (L\text{-}line\ l)\ s\ n\ m \mid l\ s\ .\ l \in \{..<t\} \wedge s \in S\ `\ (classes\ k\ t\ i)\}$
using *a* **unfolding** *T-class-def* **by** *simp*
ultimately obtain *xs xi* **where** *xdefs*: $x = join\ (L\text{-}line\ xi)\ xs\ n\ m \wedge xi < t \wedge xs \in S\ `\ (classes\ k\ t\ i)$
by *blast*

from $*\ **$ **obtain** *ys yi* **where** *ydefs*: $y = join\ (L\text{-}line\ yi)\ ys\ n\ m \wedge yi < t \wedge ys \in S\ `\ (classes\ k\ t\ i)$
using *a* **by** *auto*

have $(L\text{-}line\ xi) \in cube\ n\ (t+1)$ **using** *L-line-base-prop xdefs* **by** *simp*
moreover have $xs \in cube\ m\ (t+1)$
using *xdefs S-prop subspace-elems-embed imageE image-subset-iff mem-Collect-eq*

unfolding *layered-subspace-def classes-def* **by** *blast*
ultimately have *AA1*: $\chi\ x = \chi L\ (L\text{-}line\ xi)\ xs$ **using** *xdefs* **unfolding** *χL-def* **by** *simp*

have $(L\text{-}line\ yi) \in cube\ n\ (t+1)$ **using** *L-line-base-prop ydefs* **by** *simp*
moreover have $ys \in cube\ m\ (t+1)$
using *ydefs S-prop subspace-elems-embed imageE image-subset-iff mem-Collect-eq*

unfolding *layered-subspace-def classes-def* **by** *blast*
ultimately have *AA2*: χ *y* = χ*L* (*L-line yi*) *ys* **using** *ydefs* **unfolding** χ*L-def* **by** *simp*

have $\forall$ *s*<*t*. $\forall$ *l* < *t*. χ*L-s* (*L* (*SOME p*. *p*$\in$*cube 1* (*t*+*1*) $\wedge$ *p 0* = *s*)) = χ*L-s* (*L* (*SOME p*. *p*$\in$*cube 1* (*t*+*1*) $\wedge$ *p 0* = *l*)) **using** *dim1-layered-subspace-mono-line*[*of t L n s* χ*L-s*] *L-prop assms*(*1*) **by** *blast*
then have *key-aux*: χ*L-s* (*L-line s*) = χ*L-s* (*L-line l*) **if** *s* $\in$ {..<*t*} *l* $\in$ {..<*t*} **for** *s l*
using *that* **unfolding** *L-line-def*
by (*metis* (*no-types*, *lifting*) *add.commute lessThan-iff less-Suc-eq plus-1-eq-Suc restrict-apply*)
have *key*: χ*L* (*L-line s*) = χ*L* (*L-line l*) **if** *s* < *t l* < *t* **for** *s l*
proof–
have *L1*: χ*L* (*L-line s*) $\in$ *cube m* (*t* + *1*) $\rightarrow_E$ {..<*r*} **unfolding** χ*L-def* **using** *A L-line-base-prop* ‹*s* < *t*› **by** *simp*
have *L2*: χ*L* (*L-line l*) $\in$ *cube m* (*t* + *1*) $\rightarrow_E$ {..<*r*} **unfolding** χ*L-def* **using** *A L-line-base-prop* ‹*l* < *t*› **by** *simp*
have φ (χ*L* (*L-line s*)) = χ*L-s* (*L-line s*) **unfolding** χ*L-s-def* **using** ‹*s* < *t*› *L-line-base-prop* **by** *simp*
also have ... = χ*L-s* (*L-line l*) **using** *key-aux* ‹*s* <*t*› ‹*l* < *t*› **by** *blast*
also have ... = φ (χ*L* (*L-line l*)) **unfolding** χ*L-s-def* **using** *L-line-base-prop* ‹*l*<*t*› **by** *simp*
finally have φ (χ*L* (*L-line s*)) = φ (χ*L* (*L-line l*)) **by** *simp*
then show χ*L* (*L-line s*) = χ*L* (*L-line l*) **using** φ*-prop L-line-base-prop L1 L2* **unfolding** *bij-betw-def inj-on-def* **by** *blast*
qed
then have χ*L* (*L-line xi*) *xs* = χ*L* (*L-line 0*) *xs* **using** *xdefs assms*(*1*) **by** *metis*
also have ... = χ*S xs* **unfolding** χ*S-def* χ*L-def* **using** *xdefs L-line-base-prop* **by** *auto*
also have ... = χ*S ys* **using** *xdefs ydefs layered-eq-classes*[*of S k m t r* χ*S*] *S-prop a* **by** *blast*
also have ... = χ*L* (*L-line 0*) *ys* **unfolding** χ*S-def* χ*L-def* **using** *xdefs L-line-base-prop* **by** *auto*
also have ... = χ*L* (*L-line yi*) *ys* **using** *ydefs key assms*(*1*) **by** *metis*
finally have *core-prop*: χ*L* (*L-line xi*) *xs* = χ*L* (*L-line yi*) *ys* **by** *simp*
then have χ *x* = χ *y* **using** *AA1 AA2* **by** *simp*
then show χ *x* = χ *y* $\wedge$ χ *x* < *r*
using *xdefs AA1 key assms*(*1*) *A* ‹*L-line xi* $\in$ *cube n* (*t* + *1*)› ‹*xs* $\in$ *cube m* (*t* + *1*)› **by** *blast*
qed
then have $\exists$ *c*<*r*. $\forall$ *x* $\in$ *T* ‘ *classes* (*k*+*1*) *t i*. χ *x* = *c* **if** *i* $\leq$ *k* **for** *i*
using *that assms*(*5*) **by** *blast*

moreover have $\exists$ *c*<*r*. $\forall$ *x* $\in$ *T* ‘ *classes* (*k*+*1*) *t* (*k*+*1*). χ *x* = *c*

```
    proof -
      have ∀x ∈ classes (k+1) t (k+1). ∀u < k + 1. x u = t unfolding classes-def
by auto
      have (λu. t) ` {..<k + 1} ⊆ {..<t + 1} by auto
      then have ∃!y ∈ cube (k+1) (t+1). (∀u < k + 1. y u = t)
        using PiE-uniqueness[of (λu. t) {..<k+1} {..<t+1}] unfolding cube-def
by auto
      then have ∃!y ∈ classes (k+1) t (k+1). (∀u < k + 1. y u = t)
        unfolding classes-def using classes-subset-cube[of k+1 t k+1] by auto
      then have ∃!y. y ∈ classes (k+1) t (k+1)
        using ‹∀x ∈ classes (k+1) t (k+1). ∀u < k + 1. x u = t› by auto
      have ∃c<r. ∀y ∈ classes (k+1) t (k+1). χ (T y) = c
      proof -
        have ∀y ∈ classes (k+1) t (k+1). T y ∈ cube (n+m) (t+1) using T-prop
classes-subset-cube
          by blast
        then have ∀y ∈ classes (k+1) t (k+1). χ (T y) < r using χ-prop
          unfolding n-def d-def using M'-prop by auto
        then show ∃c<r. ∀y ∈ classes (k+1) t (k+1). χ (T y) = c
          using ‹∃!y. y ∈ classes (k+1) t (k+1)› by blast
      qed
      then show ∃c<r. ∀x ∈ T ` classes (k+1) t (k+1). χ x = c by blast
    qed
    ultimately have ∃c<r. ∀x ∈ T ` classes (k+1) t i. χ x = c if i ≤ k + 1 for i
      using that by (metis Suc-eq-plus1 le-Suc-eq)
    then have ∃c<r. ∀x ∈ classes (k+1) t i. χ (T x) = c if i ≤ k + 1 for i
      using that by simp
    then have layered-subspace T (k+1) (n + m) t r χ using subspace-T that(1)
‹n + m = M'›
      unfolding layered-subspace-def by blast
   then show ?thesis using ‹n + m = M'› by blast
  qed
  then show ?thesis unfolding lhj-def
    using m-props
      exI[of λM. ∀M'≥M. ∀χ. χ ∈ cube M' (t + 1)
      →E {..<r} ⟶ (∃S. layered-subspace S (k + 1) M' t r
      χ) m]
    by blast
qed

theorem hj-imp-lhj:
  fixes k
  assumes ⋀r'. hj r' t
  shows lhj r t k
proof (induction k arbitrary: r rule: less-induct)
  case (less k)
  consider k = 0 | k = 1 | k ≥ 2 by linarith
  then show ?case
  proof (cases)
```

case *1*
then show *?thesis* **using** *dim0-layered-subspace-ex* **unfolding** *lhj-def* **by** *auto*
next
case *2*
then show *?thesis*
proof (*cases* $t > 0$)
case *True*
then show *?thesis* **using** *hj-imp-lhj-base*[*of t*] *assms 2* **by** *blast*
next
case *False*
then show *?thesis* **using** *assms* **unfolding** *hj-def lhj-def cube-def* **by** *fastforce*
qed
next
case *3*
note *less*
then show *?thesis*
proof (*cases* $t > 0 \wedge r > 0$)
case *True*
then show *?thesis* **using** *hj-imp-lhj-step*[*of t k−1 r*]
using *assms less.IH 3 One-nat-def Suc-pred* **by** *fastforce*
next
case *False*
then consider $t = 0 \mid t > 0 \wedge r = 0 \mid t = 0 \wedge r = 0$ **by** *fastforce*
then show *?thesis*
proof *cases*
case *1*
then show *?thesis* **using** *assms* **unfolding** *hj-def lhj-def cube-def* **by** *fastforce*
next
case *2*
then obtain N **where** *N-props*: $N > 0$ $\forall N' \geq N.\ \forall \chi \in cube\ N'\ t$
$\rightarrow_E \{..<r\}.\ (\exists L\ c.\ c < r \wedge is\text{-}line\ L\ N'\ t \wedge (\forall y$
$\in L\ `\ \{..<t\}.\ \chi\ y = c))$ **using** *assms*[*of r*] **unfolding** *hj-def* **by** *force*
have $cube\ N'\ (t + 1) \rightarrow_E \{..<r\} = \{\}$ **if** $N' \geq N$ **for** N'
proof−
have $cube\ N'\ t \neq \{\}$ **using** *N-props*(*2*) *that 2* **by** *fastforce*
then have $cube\ N'\ (t + 1) \neq \{\}$ **using** *cube-subset*[*of N′ t*] **by** *blast*
then show *?thesis* **using** *2* **by** *blast*
qed
then show *?thesis* **unfolding** *lhj-def* **using** *N-props*(*1*) **by** *blast*
next
case *3*
then have $(\exists L\ c.\ c < r \wedge is\text{-}line\ L\ N'\ t \wedge (\forall y \in L\ `\ \{..<t\}.\ \chi\ y = c))$
$\Longrightarrow False$ **for** $N'\ \chi$ **by** *blast*
then have *False* **using** *assms 3* **unfolding** *hj-def cube-def* **by** *fastforce*
then show *?thesis* **by** *blast*
qed

qed

```
  qed
qed
```

2.2 Theorem 5

We provide a way to construct a monochromatic line in C^n_{t+1} from a k-dimensional k-coloured layered subspace S in C^n_{t+1}. The idea is to rely on the fact that there are $k+1$ classes in S, but only k colours. It thus follows from the Pigeonhole Principle that two classes must share the same colour. The way classes are defined allows for a straightforward construction of a line with points only from those two classes. Thus we have our monochromatic line.

```
theorem layered-subspace-to-mono-line:
  assumes layered-subspace S k n t k χ
    and t > 0
  shows (∃ L. ∃ c<k. is-line L n (t+1) ∧ (∀ y ∈ L ‘ {..<t+1}. χ y = c))
proof–
  define x where x ≡ (λi∈{..k}. λj∈{..<k}. (if j < k − i then 0 else t))

  have A: x i ∈ cube k (t + 1) if i ≤ k for i using that unfolding cube-def x-def
by simp
  then have S (x i) ∈ cube n (t+1) if i ≤ k for i using that assms(1)
    unfolding layered-subspace-def is-subspace-def by fast

  have χ ∈ cube n (t + 1) →E {..<k} using assms unfolding layered-subspace-def
by linarith
  then have χ ‘ (cube n (t+1)) ⊆ {..<k} by blast
  then have card (χ ‘ (cube n (t+1))) ≤ card {..<k}
    by (meson card-mono finite-lessThan)
  then have ∗: card (χ ‘ (cube n (t+1))) ≤ k by auto
  have k > 0 using assms(1) unfolding layered-subspace-def by auto
  have inj-on x {..k}
  proof −
    have ∗:x i1 (k − i2) ≠ x i2 (k − i2) if i1 ≤ k i2 ≤ k i1 ≠ i2 i1 < i2 for i1 i2
      using that assms(2) unfolding x-def by auto
    have ∃j<k. x i1 j ≠ x i2 j if i1 ≤ k i2 ≤ k i1 ≠ i2 for i1 i2
    proof (cases i1 ≤ i2)
      case True
      then have k − i2 < k
        using ‹0 < k› that(3) by linarith
      then show ?thesis using that ∗
        by (meson True nat-less-le)
    next
      case False
      then have i2 < i1 by simp
      then show ?thesis using that ∗[of i2 i1] ‹k > 0›
        by (metis diff-less gr-implies-not0 le0 nat-less-le)
    qed
```

then have $x\ i1 \neq x\ i2$ **if** $i1 \leq k\ i2 \leq k\ i1 \neq i2\ i1 < i2$ **for** *i1 i2* **using** *that*
by *fastforce*
then show *?thesis* **unfolding** *inj-on-def* **by** (*metis atMost-iff linorder-cases*)
qed
then have *card* $(x\ `\ \{..k\})$ = *card* $\{..k\}$ **using** *card-image* **by** *blast*
then have B: *card* $(x\ `\ \{..k\}) = k{+}1$ **by** *simp*
have $x\ `\ \{..k\} \subseteq$ *cube* $k\ (t{+}1)$ **using** A **by** *blast*
then have $S\ `\ x\ `\ \{..k\} \subseteq S\ `$ *cube* $k\ (t{+}1)$ **by** *fast*
also have ... $\subseteq$ *cube* $n\ (t{+}1)$
by (*meson assms*(*1*) *layered-subspace-def subspace-elems-embed*)
finally have $S\ `\ x\ `\ \{..k\} \subseteq$ *cube* $n\ (t{+}1)$ **by** *blast*
then have $\chi\ `\ S\ `\ x\ `\ \{..k\} \subseteq \chi\ `$ *cube* $n\ (t{+}1)$ **by** *auto*
then have *card* $(\chi\ `\ S\ `\ x\ `\ \{..k\}) \leq$ *card* $(\chi\ `$ *cube* $n\ (t{+}1))$
by (*simp add*: *card-mono cube-def finite-PiE*)
also have ... $\leq k$ **using** $*$ **by** *blast*
also have ... $< k + 1$ **by** *auto*
also have ... = *card* $\{..k\}$ **by** *simp*
also have ... = *card* $(x\ `\ \{..k\})$ **using** B **by** *auto*
also have ... = *card* $(S\ `\ x\ `\ \{..k\})$
using *subspace-inj-on-cube*[*of* $S\ k\ n\ t{+}1$] *card-image*[*of* $S\ x\ `\ \{..k\}$]
inj-on-subset[*of* S *cube* $k\ (t{+}1)\ x\ `\ \{..k\}$] *assms*(*1*) ‹$x\ `\ \{..k\} \subseteq$ *cube* $k\ (t +$
$1)$›
unfolding *layered-subspace-def* **by** *simp*
finally have *card* $(\chi\ `\ S\ `\ x\ `\ \{..k\}) <$ *card* $(S\ `\ x\ `\ \{..k\})$ **by** *blast*
then have $\neg$*inj-on* $\chi\ (S\ `\ x\ `\ \{..k\})$ **using** *pigeonhole*[*of* $\chi\ S\ `\ x\ `\ \{..k\}$] **by** *blast*
then have $\exists\, a\ b.\ a \in S\ `\ x\ `\ \{..k\} \wedge b \in S\ `\ x\ `\ \{..k\} \wedge a \neq b \wedge \chi\ a =$
$\chi\ b$ **unfolding** *inj-on-def* **by** *auto*
then obtain *ax bx* **where** *ab-props*: $ax \in S\ `\ x\ `\ \{..k\} \wedge bx \in S\ `\ x\ `\ \{..k\} \wedge ax$
$\neq bx \wedge$
$\chi\ ax = \chi\ bx$ **by** *blast*
then have $\exists\, u\ v.\ u \in \{..k\} \wedge v \in \{..k\} \wedge u \neq v \wedge \chi\ (S\ (x\ u)) = \chi\ (S\ (x$
$v))$ **by** *blast*
then obtain $u\ v$ **where** *uv-props*: $u \in \{..k\} \wedge v \in \{..k\} \wedge u < v \wedge \chi\ (S\ (x\ u))$
$= \chi\ (S\ (x\ v))$ **by** (*metis linorder-cases*)

let $?f = \lambda s.\ (\lambda i \in \{..{<}k\}.$ *if* $i < k - v$ *then* 0 *else* (*if* $i < k - u$ *then* s *else* t))
define y **where** $y \equiv (\lambda s \in \{..t\}.\ S\ (?f\ s))$

have *line1*: $?f\ s \in$ *cube* $k\ (t{+}1)$ **if** $s \leq t$ **for** s **unfolding** *cube-def* **using** *that* **by**
auto

have *f-cube*: $?f\ j \in$ *cube* $k\ (t{+}1)$ **if** $j < t{+}1$ **for** j **using** *line1 that* **by** *simp*
have *f-classes-u*: $?f\ j \in$ *classes* $k\ t\ u$ **if** *j-prop*: $j < t$ **for** j
using *that j-prop uv-props f-cube* **unfolding** *classes-def* **by** *auto*
have *f-classes-v*: $?f\ j \in$ *classes* $k\ t\ v$ **if** *j-prop*: $j = t$ **for** j
using *that j-prop uv-props assms*(*2*) *f-cube* **unfolding** *classes-def* **by** *auto*

obtain $B\ f$ **where** *Bf-props*: *disjoint-family-on* $B\ \{..k\}\ \bigcup(B\ `\ \{..k\}) = \{..{<}n\}$
$(\{\} \notin B\ `\ \{..{<}k\})$

$f \in (B\ k) \rightarrow_E \{..<t+1\}$ $S \in (cube\ k\ (t+1)) \rightarrow_E (cube\ n\ (t+1))$
$(\forall y \in cube\ k\ (t+1).\ (\forall i \in B\ k.\ S\ y\ i = f\ i) \wedge (\forall j<k.\ \forall i \in B\ j.$
$(S\ y)\ i = y\ j))$
using *assms*(*1*) **unfolding** *layered-subspace-def is-subspace-def* **by** *auto*

have $y \in \{..<t+1\} \rightarrow_E cube\ n\ (t+1)$ **unfolding** *y-def* **using** *line1* ‹*S* ‘ *cube k*
(*t* + *1*)
⊆ *cube n* (*t* + *1*)› **by** *auto*
moreover have $(\forall u<t+1.\ \forall v<t+1.\ y\ u\ j = y\ v\ j) \vee (\forall s<t+1.\ y\ s\ j = s)$
if *j-prop*: $j<n$ **for** *j*
proof−
show $(\forall u<t+1.\ \forall v<t+1.\ y\ u\ j = y\ v\ j) \vee (\forall s<t+1.\ y\ s\ j = s)$
proof −
consider $j \in B\ k \mid \exists ii<k.\ j \in B\ ii$ **using** *Bf-props*(*2*) *j-prop*
by (*metis UN-E atMost-iff le-neq-implies-less lessThan-iff*)
then have $y\ a\ j = y\ b\ j \vee y\ s\ j = s$ **if** $a < t + 1$ $b < t +1$ $s < t +1$ **for** *a b s*
proof *cases*
case *1*
then have $y\ a\ j = S\ (?f\ a)\ j$ **using** *that*(*1*) **unfolding** *y-def* **by** *auto*
also have ... $= f\ j$ **using** *Bf-props*(*6*) *f-cube 1 that*(*1*) **by** *auto*
also have ... $= S\ (?f\ b)\ j$ **using** *Bf-props*(*6*) *f-cube 1 that*(*2*) **by** *auto*
also have ... $= y\ b\ j$ **using** *that*(*2*) **unfolding** *y-def* **by** *simp*
finally show *?thesis* **by** *simp*
next
case *2*
then obtain *ii* **where** *ii-prop*: $ii < k \wedge j \in B\ ii$ **by** *blast*
then consider $ii < k - v \mid ii \geq k - v \wedge ii < k - u \mid ii \geq k - u \wedge ii < k$
using *not-less*
by *blast*
then show *?thesis*
proof *cases*
case *1*
then have $y\ a\ j = S\ (?f\ a)\ j$ **using** *that*(*1*) **unfolding** *y-def* **by** *auto*
also have ... $= (?f\ a)\ ii$ **using** *Bf-props*(*6*) *f-cube that*(*1*) *ii-prop* **by** *auto*
also have ... $= 0$ **using** *1* **by** (*simp add*: *ii-prop*)
also have ... $= (?f\ b)\ ii$ **using** *1* **by** (*simp add*: *ii-prop*)
also have ... $= S\ (?f\ b)\ j$ **using** *Bf-props*(*6*) *f-cube that*(*2*) *ii-prop* **by**
auto
also have ... $= y\ b\ j$ **using** *that*(*2*) **unfolding** *y-def* **by** *auto*
finally show *?thesis* **by** *simp*
next
case *2*
then have $y\ s\ j = S\ (?f\ s)\ j$ **using** *that*(*3*) **unfolding** *y-def* **by** *auto*
also have ... $= (?f\ s)\ ii$ **using** *Bf-props*(*6*) *f-cube that*(*3*) *ii-prop* **by** *auto*
also have ... $= s$ **using** *2* **by** (*simp add*: *ii-prop*)
finally show *?thesis* **by** *simp*
next
case *3*
then have $y\ a\ j = S\ (?f\ a)\ j$ **using** *that*(*1*) **unfolding** *y-def* **by** *auto*

also have ... = (*?f a*) *ii* **using** *Bf-props*(*6*) *f-cube that*(*1*) *ii-prop* **by** *auto*
also have ... = *t* **using** *3 uv-props* **by** *auto*
also have ... = (*?f b*) *ii* **using** *3 uv-props* **by** *auto*
also have ... = *S* (*?f b*) *j* **using** *Bf-props*(*6*) *f-cube that*(*2*) *ii-prop* **by** *auto*
also have ... = *y b j* **using** *that*(*2*) **unfolding** *y-def* **by** *auto*
finally show *?thesis* **by** *simp*
qed
qed
then show *?thesis* **by** *blast*
qed
qed
moreover have $\exists j < n. \forall s<t+1.\ y\ s\ j = s$
proof −
have $k > 0$ **using** *uv-props* **by** *simp*
have $k - v < k$ **using** *uv-props* **by** *auto*
have $k - v < k - u$ **using** *uv-props* **by** *auto*
then have $B\ (k - v) \neq \{\}$ **using** *Bf-props*(*3*) *uv-props* **by** *auto*
then obtain *j* **where** *j-prop*: $j \in B\ (k - v) \wedge j < n$ **using** *Bf-props*(*2*) *uv-props* **by** *force*
then have $y\ s\ j = s$ **if** $s<t+1$ **for** *s*
proof
have $y\ s\ j = S$ (*?f s*) *j* **using** *that* **unfolding** *y-def* **by** *auto*
also have ... = (*?f s*) $(k - v)$ **using** *Bf-props*(*6*) *f-cube that j-prop* ‹$k - v < k$› **by** *fast*
also have ... = *s* **using** *that j-prop* ‹$k - v < k - u$› **by** *simp*
finally show *?thesis* **.**
qed
then show $\exists j < n. \forall s<t+1.\ y\ s\ j = s$ **using** *j-prop* **by** *blast*
qed
ultimately have *Z1*: *is-line y n* $(t+1)$ **unfolding** *is-line-def* **by** *blast*
moreover
{
have *k-colour*: $\chi\ e < k$ **if** $e \in y$ ‘ $\{..<t+1\}$ **for** *e*
using ‹$y \in \{..<t+1\} \rightarrow_E$ *cube n* $(t + 1)$› ‹$\chi \in$ *cube n* $(t + 1) \rightarrow_E \{..<k\}$› *that* **by** *auto*
have $\chi\ e1 = \chi\ e2 \wedge \chi\ e1 < k$ **if** $e1 \in y$ ‘ $\{..<t+1\}$ $e2 \in y$ ‘ $\{..<t+1\}$ **for** *e1 e2*
proof
from *that* **obtain** *i1 i2* **where** *i-props*: $i1 < t + 1$ $i2 < t + 1$ $e1 = y\ i1$ $e2 = y\ i2$ **by** *blast*
from *i-props*(*1*,*2*) **have** $\chi\ (y\ i1) = \chi\ (y\ i2)$
proof (*induction i1 i2 rule*: *linorder-wlog*)
case (*le a b*)
then show *?case*
proof (*cases* $a = b$)
case *True*
then show *?thesis* **by** *blast*
next
case *False*

```
        then have a < b using le by linarith
        then consider b = t | b < t using le.prems(2) by linarith
        then show ?thesis
        proof cases
          case 1
          then have y b ∈ S ‘ classes k t v
          proof –
            have y b = S (?f b) unfolding y-def using ‹b = t› by auto
            moreover have ?f b ∈ classes k t v using ‹b = t› f-classes-v by blast
            ultimately show y b ∈ S ‘ classes k t v by blast
          qed
          moreover have x u ∈ classes k t u
          proof –
            have x u cord = t if cord ∈ {k − u..<k} for cord using uv-props that
unfolding x-def by simp
            moreover
            {
              have x u cord ≠ t if cord ∈ {..<k − u} for cord
                using uv-props that assms(2) unfolding x-def by auto
              then have t ∉ x u ‘ {..<k − u} by blast
            }
            ultimately show x u ∈ classes k t u unfolding classes-def
              using ‹x ‘ {..k} ⊆ cube k (t + 1)› uv-props by blast
          qed
          moreover have x v ∈ classes k t v
          proof –
            have x v cord = t if cord ∈ {k − v..<k} for cord using uv-props that
unfolding x-def by simp
            moreover
            {
              have x v cord ≠ t if cord ∈ {..<k − v} for cord
                using uv-props that assms(2) unfolding x-def by auto
              then have t ∉ x v ‘ {..<k − v} by blast
            }
            ultimately show x v ∈ classes k t v unfolding classes-def
              using ‹x ‘ {..k} ⊆ cube k (t + 1)› uv-props by blast
          qed
          moreover have χ (y b) = χ (S (x v))
             using assms(1) calculation(1, 3) unfolding layered-subspace-def by
(metis imageE uv-props)
          moreover have y a ∈ S ‘ classes k t u
          proof –
            have y a = S (?f a) unfolding y-def using ‹a < b› 1 by simp
             moreover have ?f a ∈ classes k t u using ‹a < b› 1 f-classes-u by
blast
            ultimately show y a ∈ S ‘ classes k t u by blast
          qed
          moreover have χ (y a) = χ (S (x u)) using assms(1) calculation(2, 5)
            unfolding layered-subspace-def by (metis imageE uv-props)
```

ultimately have χ $(y\ a) = \chi$ $(y\ b)$ **using** *uv-props* **by** *simp*
then show *?thesis* **by** *blast*
next
case *2*
then have $a < t$ **using** ‹$a < b$› *less-trans* **by** *blast*
then have $y\ a \in S$ ‘ *classes k t u*
proof −
have $y\ a = S$ (*?f a*) **unfolding** *y-def* **using** ‹$a < t$› **by** *auto*
moreover have *?f a* $\in$ *classes k t u* **using** ‹$a < t$› *f-classes-u* **by** *blast*
ultimately show $y\ a \in S$ ‘ *classes k t u* **by** *blast*
qed
moreover have $y\ b \in S$ ‘ *classes k t u*
proof −
have $y\ b = S$ (*?f b*) **unfolding** *y-def* **using** ‹$b < t$› **by** *auto*
moreover have *?f b* $\in$ *classes k t u* **using** ‹$b < t$› *f-classes-u* **by** *blast*
ultimately show $y\ b \in S$ ‘ *classes k t u* **by** *blast*
qed
ultimately have χ $(y\ a) = \chi$ $(y\ b)$ **using** *assms*(*1*) *uv-props* **unfolding** *layered-subspace-def*
by (*metis imageE*)
then show *?thesis* **by** *blast*
qed
qed
next
case (*sym a b*)
then show *?case* **by** *presburger*
qed
then show χ *e1* $= \chi$ *e2* **using** *i-props*(*3*,*4*) **by** *blast*
qed (*use that*(*1*) *k-colour* **in** *blast*)
then have *Z2*: $\exists\, c < k.\ \forall\, e \in y$ ‘ $\{..<t+1\}.\ \chi\ e = c$
by (*meson image-eqI lessThan-iff less-add-one*)
}
ultimately show $\exists\, L\ c.\ c < k \wedge$ *is-line* $L\ n\ (t + 1) \wedge (\forall\, y {\in} L$ ‘ $\{..<t + 1\}.\ \chi\ y = c)$
by *blast*

qed

2.3 Corollary 6

corollary *lhj-imp-hj*:
assumes $(\bigwedge r\ k.\ lhj\ r\ t\ k)$
and $t{>}0$
shows (*hj r* $(t{+}1)$)
using *assms*(*1*)[*of r r*] *assms*(*2*) **unfolding** *lhj-def hj-def* **using** *layered-subspace-to-mono-line*[*of - r - t*] **by** *metis*

2.4 Main result

2.4.1 Edge cases and auxiliary lemmas

lemma *single-point-line*:
  assumes $N > 0$
  shows *is-line* $(\lambda s\in\{..<1\}.\ \lambda a\in\{..<N\}.\ 0)\ N\ 1$
  using *assms* **unfolding** *is-line-def cube-def* **by** *auto*

lemma *single-point-line-is-monochromatic*:
  assumes $\chi \in$ *cube* $N\ 1 \rightarrow_E \{..<r\}$ $N > 0$
  shows $(\exists\, c < r.$ *is-line* $(\lambda s\in\{..<1\}.\ \lambda a\in\{..<N\}.\ 0)\ N\ 1 \wedge (\forall\, i \in (\lambda s\in\{..<1\}.\ \lambda a\in\{..<N\}.\ 0)$ ‘ $\{..<1\}.\ \chi\ i = c))$
proof –
  have *is-line* $(\lambda s\in\{..<1\}.\ \lambda a\in\{..<N\}.\ 0)\ N\ 1$ **using** *assms*(2) *single-point-line* **by** *blast*
  moreover have $\exists\, c < r.\ \chi\ ((\lambda s\in\{..<1\}.\ \lambda a\in\{..<N\}.\ 0)\ j) = c$
    if $(j::nat) < 1$ **for** j **using** *assms line-points-in-cube calculation that* **unfolding** *cube-def* **by** *blast*
  ultimately show *?thesis* **by** *auto*
qed

lemma *hj-r-nonzero-t-0*:
  assumes $r > 0$
  shows *hj* $r\ 0$
proof–
  have $(\exists\, L\ c.\ c < r \wedge$ *is-line* $L\ N'\ 0 \wedge (\forall\, y \in L$ ‘ $\{..<0::nat\}.\ \chi\ y = c))$
    if $N' \geq 1$ $\chi \in$ *cube* $N'\ 0 \rightarrow_E \{..<r\}$ **for** $N'\ \chi$ **using** *assms is-line-def that*(1) **by** *fastforce*
  then show *?thesis* **unfolding** *hj-def* **by** *auto*
qed

Any cube over 1 element always has a single point, which also forms the only line in the cube. Since it's a single point line, it's trivially monochromatic. We show the result for dimension 1.

lemma *hj-t-1*: *hj* $r\ 1$
  unfolding *hj-def*
proof–
  let $?N = 1$
  have $\exists\, L\ c.\ c < r \wedge$ *is-line* $L\ N'\ 1 \wedge (\forall\, y\in L$ ‘ $\{..<1\}.\ \chi\ y = c)$ **if** $N' \geq ?N$ $\chi \in$ *cube* $N'\ 1 \rightarrow_E \{..<r\}$ **for** $N'\ \chi$
    using *single-point-line-is-monochromatic*[*of* $\chi\ N'\ r$] *that* **by** *force*
  then show $\exists\, N{>}0.\ \forall\, N'{\geq}N.\ \forall\,\chi.\ \chi \in$ *cube* $N'\ 1 \rightarrow_E \{..<r\} \longrightarrow (\exists\, L\ c.\ c < r \wedge$ *is-line* $L\ N'\ 1 \wedge (\forall\, y\in L$ ‘ $\{..<1\}.\ \chi\ y = c))$
    by *blast*
qed

2.4.2 Main theorem

We state the main result *hj r t*. The explanation for the choice of assumption is offered subsequently.

theorem *hales-jewett*:
  assumes $\neg(r = 0 \wedge t = 0)$
  shows *hj r t*
  using *assms*
proof (*induction t arbitrary*: *r*)
  case *0*
  then show *?case* **using** *hj-r-nonzero-t-0*[*of r*] **by** *blast*
next
  case (*Suc t*)
  then show *?case* **using** *hj-t-1*[*of r*] *hj-imp-lhj*[*of t*] *lhj-imp-hj*[*of t r*] **by** *auto*
qed

We offer a justification for having excluded the special case $r = t = 0$ from the statement of the main theorem *hales-jewett*. The exclusion is a consequence of the fact that colourings are defined as members of the function set $\mathit{cube}\ n\ t \rightarrow_E \{..<r\}$, which for $r = t = 0$ means there's a dummy colouring $\lambda x.\ \mathit{undefined}$, even though $\mathit{cube}\ n\ 0 = \{\}$ for $n > 0$. Hence, in this case, no line exists at all (let alone one monochromatic under the aforementioned colouring). This means $\mathit{hj}\ 0\ 0 \Longrightarrow \mathit{False}$—but only because of the quirky behaviour of the FuncSet $\mathit{cube}\ n\ t \rightarrow_E \{..<r\}$. This could have been circumvented by letting colourings χ be arbitrary functions constraint only by χ ‘ $\mathit{cube}\ n\ t \subseteq \{..<r\}$. We avoided this in order to have consistency with the cube's definition, for which FuncSets were crucial because the proof heavily relies on arguments about the cardinality of the cube. he constraint x ‘ $\{..<n\} \subseteq \{..<t\}$ for elements x of C_t^n would not have sufficed there, as there are infinitely many functions over the naturals satisfying it.

end

References

[1] R. L. Graham, B. L. Rothschild, and J. H. Spencer. *Ramsey Theory, 2nd Edition*. Wiley-Interscience, March 1990.

[2] K. Kreuzer and M. Eberl. Van der Waerden's Theorem. *Archive of Formal Proofs*, June 2021. https://isa-afp.org/entries/Van_der_Waerden.html, Formal proof development.