Geometric Axioms for Minkowski Spacetime

Richard Schmoetten, Jake Palmer, Jacques Fleuriot

March 19, 2025

Abstract

This is a formalisation of Schutz' system of axioms for Minkowski spacetime [1], as well as the results in his third chapter ("Temporal Order on a Path"), with the exception of the second part of Theorem 12. Many results are proven here that cannot be found in Schutz, either preceding the theorem they are needed for, or in their own thematic section.

Contents

theory *TernaryOrdering*
imports *Util*

begin

Definition of chains using an ordering on sets of events based on natural numbers, plus some proofs.

1 Totally ordered chains

Based on page 110 of Phil Scott's thesis and the following HOL Light definition:

```
let ORDERING = new_definition
  `ORDERING f X <=> (!n. (FINITE X ==> n < CARD X) ==> f n IN X)
                    /\ (!x. x IN X ==> ?n. (FINITE X ==> n < CARD X)
                        /\ f n = x)
                    /\ !n n' n''. (FINITE X ==> n'' < CARD X)
                          /\ n < n' /\ n' < n''
                          ==> between (f n) (f n') (f n'')`;;
```

I've made it strict for simplicity, and because that's how Schutz's ordering is. It could be made more generic by taking in the function corresponding to $<$ as a paramater. Main difference to Schutz: he has local order, not total (cf Theorem 2 and *local-ordering*).

definition *ordering* :: $(nat \Rightarrow 'a) \Rightarrow ('a \Rightarrow 'a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a\ set \Rightarrow bool$
where
$ordering\ f\ ord\ X \equiv (\forall n.\ (finite\ X \longrightarrow n < card\ X) \longrightarrow f\ n \in X)$
$\wedge (\forall x{\in}X.\ (\exists n.\ (finite\ X \longrightarrow n < card\ X) \wedge f\ n = x))$
$\wedge (\forall n\ n'\ n''.\ (finite\ X \longrightarrow n'' < card\ X) \wedge n < n' \wedge n' < n''$
$\longrightarrow ord\ (f\ n)\ (f\ n')\ (f\ n''))$

lemma *finite-ordering-intro*:
assumes *finite X*
and $\forall n < card\ X.\ f\ n \in X$
and $\forall x \in X.\ \exists n < card\ X.\ f\ n = x$
and $\forall n\ n'\ n''.\ n < n' \wedge n' < n'' \wedge n'' < card\ X \longrightarrow ord\ (f\ n)\ (f\ n')\ (f\ n'')$
shows *ordering f ord X*
unfolding *ordering-def* **by** (*simp add*: *assms*)

lemma *infinite-ordering-intro*:
assumes *infinite X*
and $\forall n{::}nat.\ f\ n \in X$
and $\forall x \in X.\ \exists n{::}nat.\ f\ n = x$

and $\forall\, n\ n'\ n''.\ n < n' \wedge n' < n'' \longrightarrow ord\ (f\ n)\ (f\ n')\ (f\ n'')$
shows *ordering f ord X*
unfolding *ordering-def* **by** (*simp add*: *assms*)

lemma *ordering-ord-ijk*:
assumes *ordering f ord X*
and $i < j \wedge j < k \wedge (finite\ X \longrightarrow k < card\ X)$
shows *ord* (*f i*) (*f j*) (*f k*)
by (*metis ordering-def assms*)

lemma *empty-ordering* [*simp*]: $\exists f.$ *ordering f ord* {}
by (*simp add*: *ordering-def*)

lemma *singleton-ordering* [*simp*]: $\exists f.$ *ordering f ord* {*a*}
apply (*rule-tac* $x = \lambda n.\ a$ **in** *exI*)
by (*simp add*: *ordering-def*)

lemma *two-ordering* [*simp*]: $\exists f.$ *ordering f ord* {*a*, *b*}
proof *cases*
assume $a = b$
thus *?thesis* **using** *singleton-ordering* **by** *simp*
next
assume *a-neq-b*: $a \neq b$
let $?f = \lambda n.\ if\ n = 0\ then\ a\ else\ b$
have *ordering1*: $(\forall\, n.\ (finite\ \{a,b\} \longrightarrow n < card\ \{a,b\}) \longrightarrow ?f\ n \in \{a,b\})$ **by** *simp*
have *local-ordering*: $(\forall\, x{\in}\{a,b\}.\ \exists\, n.\ (finite\ \{a,b\} \longrightarrow n < card\ \{a,b\}) \wedge ?f\ n = x)$
using *a-neq-b all-not-in-conv card-Suc-eq card-0-eq card-gt-0-iff insert-iff lessI* **by** *auto*
have *ordering3*: $(\forall\, n\ n'\ n''.\ (finite\ \{a,b\} \longrightarrow n'' < card\ \{a,b\}) \wedge n < n' \wedge n' < n''$
$\longrightarrow ord\ (?f\ n)\ (?f\ n')\ (?f\ n''))$ **using** *a-neq-b* **by** *auto*
have *ordering ?f ord* {*a*, *b*} **using** *ordering-def ordering1 local-ordering ordering3* **by** *blast*
thus *?thesis* **by** *auto*
qed

lemma *card-le2-ordering*:
assumes *finiteX*: *finite X*
and *card-le2*: $card\ X \leq 2$
shows $\exists f.$ *ordering f ord X*
proof −
have *card012*: $card\ X = 0 \vee card\ X = 1 \vee card\ X = 2$ **using** *card-le2* **by** *auto*
have *card0*: $card\ X = 0 \longrightarrow ?thesis$ **using** *finiteX* **by** *simp*
have *card1*: $card\ X = 1 \longrightarrow ?thesis$ **using** *card-eq-SucD* **by** *fastforce*
have *card2*: $card\ X = 2 \longrightarrow ?thesis$ **by** (*metis two-ordering card-eq-SucD numeral-2-eq-2*)
thus *?thesis* **using** *card012 card0 card1 card2* **by** *auto*

qed

lemma *ord-ordered*:
  assumes *abc*: *ord a b c*
    and *abc-neq*: $a \neq b \wedge a \neq c \wedge b \neq c$
  shows $\exists f.$ *ordering f ord* $\{a,b,c\}$
  apply (*rule-tac* $x = \lambda n.$ *if* $n = 0$ *then a else if* $n = 1$ *then b else c* **in** *exI*)
  apply (*unfold ordering-def*)
  using *abc abc-neq* **by** *auto*

lemma *overlap-ordering*:
  assumes *abc*: *ord a b c*
    and *bcd*: *ord b c d*
    and *abd*: *ord a b d*
    and *acd*: *ord a c d*
    and *abc-neq*: $a \neq b \wedge a \neq c \wedge a \neq d \wedge b \neq c \wedge b \neq d \wedge c \neq d$
  shows $\exists f.$ *ordering f ord* $\{a,b,c,d\}$
proof −
  let *?X* = $\{a,b,c,d\}$
  let *?f* = $\lambda n.$ *if* $n = 0$ *then a else if* $n = 1$ *then b else if* $n = 2$ *then c else d*
  have *card4*: *card ?X* = *4* **using** *abc bcd abd abc-neq* **by** *simp*
  have *ordering1*: $\forall n.$ (*finite ?X* $\longrightarrow n <$ *card ?X*) $\longrightarrow$ *?f n* $\in$ *?X* **by** *simp*
  have *local-ordering*: $\forall x \in$ *?X*. $\exists n.$ (*finite ?X* $\longrightarrow n <$ *card ?X*) $\wedge$ *?f n* = *x*
    by (*metis card4 One-nat-def Suc-1 Suc-lessI empty-iff insertE numeral-3-eq-3 numeral-eq-iff*
      numeral-eq-one-iff rel-simps(*51*) *semiring-norm*(*85*) *semiring-norm*(*86*) *semiring-norm*(*87*)
      semiring-norm(*89*) *zero-neq-numeral*)
  have *ordering3*: ($\forall n\ n'\ n''.$ (*finite ?X* $\longrightarrow n'' <$ *card ?X*) $\wedge n < n' \wedge n' < n''$
      $\longrightarrow$ *ord* (*?f n*) (*?f n'*) (*?f n''*))
    using *card4 abc bcd abd acd card-0-eq card-insert-if finite.emptyI finite-insert less-antisym*
      less-one less-trans-Suc not-less-eq not-one-less-zero numeral-2-eq-2 **by** *auto*
  have *ordering ?f ord ?X* **using** *ordering1 local-ordering ordering3 ordering-def* **by** *blast*
  thus *?thesis* **by** *auto*
qed

lemma *overlap-ordering-alt1*:
  assumes *abc*: *ord a b c*
    and *bcd*: *ord b c d*
    and *abc-bcd-abd*: $\forall$ *a b c d*. *ord a b c* $\wedge$ *ord b c d* $\longrightarrow$ *ord a b d*
    and *abc-bcd-acd*: $\forall$ *a b c d*. *ord a b c* $\wedge$ *ord b c d* $\longrightarrow$ *ord a c d*
    and *ord-distinct*: $\forall$ *a b c*. (*ord a b c* $\longrightarrow a \neq b \wedge a \neq c \wedge b \neq c$)
  shows $\exists f.$ *ordering f ord* $\{a,b,c,d\}$
  by (*metis* (*full-types*) *assms overlap-ordering*)

lemma *overlap-ordering-alt2*:
  assumes *abc*: *ord a b c*

and *bcd*: *ord b c d*
and *abd*: *ord a b d*
and *acd*: *ord a c d*
and *ord-distinct*: $\forall$ *a b c*. (*ord a b c* $\longrightarrow$ $a \neq b \land a \neq c \land b \neq c$)
shows $\exists f$. *ordering f ord* $\{a,b,c,d\}$
by (*metis assms overlap-ordering*)

lemma *overlap-ordering-alt*:
assumes *abc*: *ord a b c*
and *bcd*: *ord b c d*
and *abc-bcd-abd*: $\forall$ *a b c d*. *ord a b c* $\land$ *ord b c d* $\longrightarrow$ *ord a b d*
and *abc-bcd-acd*: $\forall$ *a b c d*. *ord a b c* $\land$ *ord b c d* $\longrightarrow$ *ord a c d*
and *abc-neq*: $a \neq b \land a \neq c \land a \neq d \land b \neq c \land b \neq d \land c \neq d$
shows $\exists f$. *ordering f ord* $\{a,b,c,d\}$
by (*meson assms overlap-ordering*)

The lemmas below are easy to prove for $X = \{\}$, and if I included that case then I would have to write a conditional definition in place of $\{0..|X| - 1\}$.

lemma *finite-ordering-img*: $[\![X \neq \{\}$; *finite X*; *ordering f ord X*$]\!] \Longrightarrow f$ ‘ $\{0..card\ X - 1\} = X$
by (*force simp add*: *ordering-def image-def*)

lemma *inf-ordering-img*: $[\![$*infinite X*; *ordering f ord X*$]\!] \Longrightarrow f$ ‘ $\{0..\} = X$
by (*auto simp add*: *ordering-def image-def*)

lemma *inf-ordering-inv-img*: $[\![$*infinite X*; *ordering f ord X*$]\!] \Longrightarrow f$ −‘ $X = \{0..\}$
by (*auto simp add*: *ordering-def image-def*)

lemma *inf-ordering-img-inv-img*: $[\![$*infinite X*; *ordering f ord X*$]\!] \Longrightarrow f$ ‘ f −‘ $X = X$
using *inf-ordering-img* **by** *auto*

lemma *finite-ordering-inj-on*: $[\![$*finite X*; *ordering f ord X*$]\!] \Longrightarrow$ *inj-on f* $\{0..card\ X - 1\}$
by (*metis finite-ordering-img Suc-diff-1 atLeastAtMost-iff card-atLeastAtMost card-eq-0-iff*
diff-0-eq-0 diff-zero eq-card-imp-inj-on gr0I inj-onI le-0-eq)

lemma *finite-ordering-bij*:
assumes *orderingX*: *ordering f ord X*
and *finiteX*: *finite X*
and *non-empty*: $X \neq \{\}$
shows *bij-betw f* $\{0..card\ X - 1\}$ *X*
proof −
have *f-image*: f ‘ $\{0..card\ X - 1\} = X$ **by** (*metis orderingX finiteX finite-ordering-img non-empty*)
thus *?thesis* **by** (*metis inj-on-imp-bij-betw orderingX finiteX finite-ordering-inj-on*)

qed

lemma *inf-ordering-inj′*:
  assumes *infX*: *infinite X*
    and *f-ord*: *ordering f ord X*
    and *ord-distinct*: $\forall a\ b\ c.\ (ord\ a\ b\ c \longrightarrow a \neq b \wedge a \neq c \wedge b \neq c)$
    and *f-eq*: $f\ m = f\ n$
  shows $m = n$
proof (*rule ccontr*)
  assume *m-not-n*: $m \neq n$
  have *betw-3n*: $\forall n\ n'\ n''.\ n < n' \wedge n' < n'' \longrightarrow ord\ (f\ n)\ (f\ n')\ (f\ n'')$
    using *f-ord* **by** (*simp add*: *ordering-def infX*)
  thus *False*
  proof *cases*
    assume *m-less-n*: $m < n$
    then obtain k **where** $n < k$ **by** *auto*
    then have $ord\ (f\ m)\ (f\ n)\ (f\ k)$ **using** *m-less-n betw-3n* **by** *simp*
    then have $f\ m \neq f\ n$ **using** *ord-distinct* **by** *simp*
    thus *?thesis* **using** *f-eq* **by** *simp*
  next
    assume $\neg\ m < n$
    then have *n-less-m*: $n < m$ **using** *m-not-n* **by** *simp*
    then obtain k **where** $m < k$ **by** *auto*
    then have $ord\ (f\ n)\ (f\ m)\ (f\ k)$ **using** *n-less-m betw-3n* **by** *simp*
    then have $f\ n \neq f\ m$ **using** *ord-distinct* **by** *simp*
    thus *?thesis* **using** *f-eq* **by** *simp*
  qed
qed

lemma *inf-ordering-inj*:
  assumes *infinite X*
    and *ordering f ord X*
    and $\forall a\ b\ c.\ (ord\ a\ b\ c \longrightarrow a \neq b \wedge a \neq c \wedge b \neq c)$
  shows *inj f*
  using *inf-ordering-inj′ assms* **by** (*metis injI*)

The finite case is a little more difficult as I can't just choose some other natural number to form the third part of the betweenness relation and the initial simplification isn't as nice. Note that I cannot prove *inj f* (over the whole type that *f* is defined on, i.e. natural numbers), because I need to capture the *m* and *n* that obey specific requirements for the finite case. In order to prove *inj f*, I would have to extend the definition for ordering to include *m* and *n* beyond *card X*, such that it is still injective. That would probably not be very useful.

lemma *finite-ordering-inj*:
  assumes *finiteX*: *finite X*
    and *f-ord*: *ordering f ord X*
    and *ord-distinct*: $\forall a\ b\ c.\ (ord\ a\ b\ c \longrightarrow a \neq b \wedge a \neq c \wedge b \neq c)$
    and *m-less-card*: $m < card\ X$

and *n-less-card*: $n < card\ X$
and *f-eq*: $f\ m = f\ n$
shows $m = n$
proof (*rule ccontr*)
assume *m-not-n*: $m \neq n$
have *surj-f*: $\forall x \in X.\ \exists n < card\ X.\ f\ n = x$
using *f-ord* **by** (*simp add*: *ordering-def finiteX*)
have *betw-3n*: $\forall n\ n'\ n''.\ n'' < card\ X \wedge n < n' \wedge n' < n'' \longrightarrow ord\ (f\ n)\ (f\ n')\ (f\ n'')$
using *f-ord* **by** (*simp add*: *ordering-def*)
show *False*
proof *cases*
assume *card-le2*: $card\ X \leq 2$
have *card0*: $card\ X = 0 \longrightarrow False$ **using** *m-less-card* **by** *simp*
have *card1*: $card\ X = 1 \longrightarrow False$ **using** *m-less-card n-less-card m-not-n* **by** *simp*
have *card2*: $card\ X = 2 \longrightarrow False$
proof (*rule impI*)
assume *card-is-2*: $card\ X = 2$
then have *mn01*: $m = 0 \wedge n = 1 \vee n = 0 \wedge m = 1$ **using** *m-less-card n-less-card m-not-n* **by** *auto*
then have $f\ m \neq f\ n$ **using** *card-is-2 surj-f One-nat-def card-eq-SucD insertCI less-2-cases numeral-2-eq-2* **by** (*metis* (*no-types*, *lifting*))
thus *False* **using** *f-eq* **by** *simp*
qed
show *False* **using** *card0 card1 card2 card-le2* **by** *simp*
next
assume $\neg\ card\ X \leq 2$
then have *card-ge3*: $card\ X \geq 3$ **by** *simp*
thus *False*
proof *cases*
assume *m-less-n*: $m < n$
then obtain k **where** *k-pos*: $k < m \vee (m < k \wedge k < n) \vee (n < k \wedge k < card\ X)$
using *is-free-nat m-less-n n-less-card card-ge3* **by** *blast*
have *k1*: $k < m \longrightarrow ord\ (f\ k)\ (f\ m)\ (f\ n)$ **using** *m-less-n n-less-card betw-3n* **by** *simp*
have *k2*: $m < k \wedge k < n \longrightarrow ord\ (f\ m)\ (f\ k)\ (f\ n)$ **using** *m-less-n n-less-card betw-3n* **by** *simp*
have *k3*: $n < k \wedge k < card\ X \longrightarrow ord\ (f\ m)\ (f\ n)\ (f\ k)$ **using** *m-less-n betw-3n* **by** *simp*
have $f\ m \neq f\ n$ **using** *k1 k2 k3 k-pos ord-distinct* **by** *auto*
thus *False* **using** *f-eq* **by** *simp*
next
assume $\neg\ m < n$
then have *n-less-m*: $n < m$ **using** *m-not-n* **by** *simp*
then obtain k **where** *k-pos*: $k < n \vee (n < k \wedge k < m) \vee (m < k \wedge k < card\ X)$
using *is-free-nat n-less-m m-less-card card-ge3* **by** *blast*

have *k1*: $k < n \longrightarrow ord\ (f\ k)\ (f\ n)\ (f\ m)$ **using** *n-less-m m-less-card betw-3n* **by** *simp*
have *k2*: $n < k \wedge k < m \longrightarrow ord\ (f\ n)\ (f\ k)\ (f\ m)$ **using** *n-less-m m-less-card betw-3n* **by** *simp*
have *k3*: $m < k \wedge k < card\ X \longrightarrow ord\ (f\ n)\ (f\ m)\ (f\ k)$ **using** *n-less-m betw-3n* **by** *simp*
have $f\ n \neq f\ m$ **using** *k1 k2 k3 k-pos ord-distinct* **by** *auto*
thus *False* **using** *f-eq* **by** *simp*
qed
qed
qed

lemma *ordering-inj*:
assumes *ordering f ord X*
and $\forall a\ b\ c.\ (ord\ a\ b\ c \longrightarrow a \neq b \wedge a \neq c \wedge b \neq c)$
and $finite\ X \longrightarrow m < card\ X$
and $finite\ X \longrightarrow n < card\ X$
and $f\ m = f\ n$
shows $m = n$
using *inf-ordering-inj′ finite-ordering-inj assms* **by** *blast*

lemma *ordering-sym*:
assumes *ord-sym*: $\bigwedge a\ b\ c.\ ord\ a\ b\ c \Longrightarrow ord\ c\ b\ a$
and *finite X*
and *ordering f ord X*
shows $ordering\ (\lambda n.\ f\ (card\ X - 1 - n))\ ord\ X$
unfolding *ordering-def* **using** *assms(2)*
apply *auto*
apply (*metis ordering-def assms(3) card-0-eq card-gt-0-iff diff-Suc-less gr-implies-not0*)
proof −
fix *x*
assume *finite X*
assume $x \in X$
obtain *n* **where** $finite\ X \longrightarrow n < card\ X$ **and** $f\ n = x$
by (*metis ordering-def* ‹$x \in X$› *assms(3)*)
have $f\ (card\ X - ((card\ X - 1 - n) + 1)) = x$
by (*simp add*: *Suc-leI* ‹$f\ n = x$› ‹$finite\ X \longrightarrow n < card\ X$› *assms(2)*)
thus $\exists n{<}card\ X.\ f\ (card\ X - Suc\ n) = x$
by (*metis* ‹$x \in X$› *add.commute assms(2) card-Diff-singleton card-Suc-Diff1 diff-less-Suc plus-1-eq-Suc*)
next
fix $n\ n'\ n''$
assume *finite X*
assume $n'' < card\ X\ n < n'\ n' < n''$
have $ord\ (f\ (card\ X - Suc\ n''))\ (f\ (card\ X - Suc\ n'))\ (f\ (card\ X - Suc\ n))$
using *assms(3)* **unfolding** *ordering-def*
using ‹$n < n'$› ‹$n' < n''$› ‹$n'' < card\ X$› *diff-less-mono2* **by** *auto*
thus $ord\ (f\ (card\ X - Suc\ n))\ (f\ (card\ X - Suc\ n'))\ (f\ (card\ X - Suc\ n''))$
using *ord-sym* **by** *blast*

qed

lemma *zero-into-ordering*:
assumes *ordering f betw X*
and $X \neq \{\}$
shows $(f\ 0) \in X$
using *ordering-def*
by (*metis assms card-eq-0-iff gr-implies-not0 linorder-neqE-nat*)

2 Locally ordered chains

Definitions for Schutz-like chains, with local order only.

definition *local-ordering* :: $(nat \Rightarrow 'a) \Rightarrow ('a \Rightarrow 'a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a\ set \Rightarrow bool$
where *local-ordering f ord X*
$\equiv (\forall n.\ (finite\ X \longrightarrow n < card\ X) \longrightarrow f\ n \in X) \wedge$
$(\forall x{\in}X.\ \exists n.\ (finite\ X \longrightarrow n < card\ X) \wedge f\ n = x) \wedge$
$(\forall n.\ (finite\ X \longrightarrow Suc\ (Suc\ n) < card\ X) \longrightarrow ord\ (f\ n)\ (f\ (Suc\ n))\ (f\ (Suc\ (Suc\ n))))$

lemma *finite-local-ordering-intro*:
assumes *finite X*
and $\forall n < card\ X.\ f\ n \in X$
and $\forall x \in X.\ \exists n < card\ X.\ f\ n = x$
and $\forall n\ n'\ n''.\ Suc\ n = n' \wedge Suc\ n' = n'' \wedge n'' < card\ X \longrightarrow ord\ (f\ n)\ (f\ n')\ (f\ n'')$
shows *local-ordering f ord X*
unfolding *local-ordering-def* **by** (*simp add*: *assms*)

lemma *infinite-local-ordering-intro*:
assumes *infinite X*
and $\forall n{::}nat.\ f\ n \in X$
and $\forall x \in X.\ \exists n{::}nat.\ f\ n = x$
and $\forall n\ n'\ n''.\ Suc\ n = n' \wedge Suc\ n' = n'' \longrightarrow ord\ (f\ n)\ (f\ n')\ (f\ n'')$
shows *local-ordering f ord X*
using *assms* **unfolding** *local-ordering-def* **by** *metis*

lemma *total-implies-local*:
$ordering\ f\ ord\ X \Longrightarrow local\text{-}ordering\ f\ ord\ X$
unfolding *ordering-def local-ordering-def*
using *lessI* **by** *presburger*

lemma *ordering-ord-ijk-loc*:
assumes *local-ordering f ord X*
and $finite\ X \longrightarrow Suc\ (Suc\ i) < card\ X$
shows $ord\ (f\ i)\ (f\ (Suc\ i))\ (f\ (Suc\ (Suc\ i)))$
by (*metis local-ordering-def assms*)

lemma *empty-ordering-loc* [*simp*]:

$\exists f.$ *local-ordering f ord* $\{\}$
by (*simp add*: *local-ordering-def*)

lemma *singleton-ordered-loc* [*simp*]:
local-ordering f ord $\{f\ 0\}$
unfolding *local-ordering-def* **by** *simp*

lemma *singleton-ordering-loc* [*simp*]:
$\exists f.$ *local-ordering f ord* $\{a\}$
using *singleton-ordered-loc* **by** *fast*

lemma *two-ordered-loc*:
assumes $a = f\ 0$ **and** $b = f\ 1$
shows *local-ordering f ord* $\{a,\ b\}$
proof *cases*
assume $a = b$
thus *?thesis* **using** *assms singleton-ordered-loc* **by** (*metis insert-absorb2*)
next
assume *a-neq-b*: $a \neq b$
hence $(\forall n.\ (\mathit{finite}\ \{a,b\} \longrightarrow n < \mathit{card}\ \{a,b\}) \longrightarrow f\ n \in \{a,b\})$
using *assms* **by** (*metis One-nat-def card.infinite card-2-iff fact-0 fact-2 insert-iff less-2-cases-iff*)
moreover have $(\forall x{\in}\{a,b\}.\ \exists n.\ (\mathit{finite}\ \{a,b\} \longrightarrow n < \mathit{card}\ \{a,b\}) \land f\ n = x)$
using *assms a-neq-b all-not-in-conv card-Suc-eq card-0-eq card-gt-0-iff insert-iff lessI* **by** *auto*
moreover have $(\forall n.\ (\mathit{finite}\ \{a,b\} \longrightarrow \mathit{Suc}\ (\mathit{Suc}\ n) < \mathit{card}\ \{a,b\})$
$\longrightarrow \mathit{ord}\ (f\ n)\ (f\ (\mathit{Suc}\ n))\ (f\ (\mathit{Suc}\ (\mathit{Suc}\ n))))$
using *a-neq-b* **by** *auto*
ultimately have *local-ordering f ord* $\{a,\ b\}$
using *local-ordering-def* **by** *blast*
thus *?thesis* **by** *auto*
qed

lemma *two-ordering-loc* [*simp*]:
$\exists f.$ *local-ordering f ord* $\{a,\ b\}$
using *total-implies-local two-ordering* **by** *fastforce*

lemma *card-le2-ordering-loc*:
assumes *finiteX*: *finite X*
and *card-le2*: $\mathit{card}\ X \leq 2$
shows $\exists f.$ *local-ordering f ord X*
using *assms total-implies-local card-le2-ordering* **by** *metis*

lemma *ord-ordered-loc*:
assumes *abc*: *ord a b c*
and *abc-neq*: $a \neq b \land a \neq c \land b \neq c$
shows $\exists f.$ *local-ordering f ord* $\{a,b,c\}$
using *assms total-implies-local ord-ordered* **by** *metis*

lemma *overlap-ordering-loc*:
assumes *abc*: *ord a b c*
and *bcd*: *ord b c d*
and *abd*: *ord a b d*
and *acd*: *ord a c d*
and *abc-neq*: $a \neq b \wedge a \neq c \wedge a \neq d \wedge b \neq c \wedge b \neq d \wedge c \neq d$
shows ∃*f*. *local-ordering f ord* {*a*,*b*,*c*,*d*}
using *overlap-ordering*[*OF assms*] *total-implies-local* **by** *blast*

lemma *ordering-sym-loc*:
assumes *ord-sym*: ⋀*a b c*. *ord a b c* ⟹ *ord c b a*
and *finite X*
and *local-ordering f ord X*
shows *local-ordering* (λ*n*. *f* (*card X* − *1* − *n*)) *ord X*
unfolding *local-ordering-def* **using** *assms*(*2*) **apply** *auto*
apply (*metis local-ordering-def assms*(*3*) *card-0-eq card-gt-0-iff diff-Suc-less gr-implies-not0*)
proof −
fix *x*
assume *finite X*
assume *x* ∈ *X*
obtain *n* **where** *finite X* ⟶ *n* < *card X* **and** *f n* = *x*
by (*metis local-ordering-def* ‹*x* ∈ *X*› *assms*(*3*))
have *f* (*card X* − ((*card X* − *1* − *n*) + *1*)) = *x*
by (*simp add*: *Suc-leI* ‹*f n* = *x*› ‹*finite X* ⟶ *n* < *card X*› *assms*(*2*))
thus ∃ *n*<*card X*. *f* (*card X* − *Suc n*) = *x*
by (*metis* ‹*x* ∈ *X*› *add.commute assms*(*2*) *card-Diff-singleton card-Suc-Diff1 diff-less-Suc plus-1-eq-Suc*)
next
fix *n*
let *?n1* = *Suc n*
let *?n2* = *Suc ?n1*
assume *finite X*
assume *Suc* (*Suc n*) < *card X*
have *ord* (*f* (*card X* − *Suc ?n2*)) (*f* (*card X* − *Suc ?n1*)) (*f* (*card X* − *Suc n*))
using *assms*(*3*) **unfolding** *local-ordering-def*
using ‹*Suc* (*Suc n*) < *card X*› **by** (*metis*
Suc-diff-Suc Suc-lessD card-eq-0-iff card-gt-0-iff diff-less gr-implies-not0 zero-less-Suc)
thus *ord* (*f* (*card X* − *Suc n*)) (*f* (*card X* − *Suc ?n1*)) (*f* (*card X* − *Suc ?n2*))
using *ord-sym* **by** *blast*
qed

lemma *zero-into-ordering-loc*:
assumes *local-ordering f betw X*
and *X* ≠ {}
shows (*f 0*) ∈ *X*
using *local-ordering-def* **by** (*metis assms card-eq-0-iff gr-implies-not0 linorder-neqE-nat*)

end

theory *Minkowski*
imports *TernaryOrdering*
begin

Primitives and axioms as given in [1, pp. 9-17].

I've tried to do little to no proofs in this file, and keep that in other files. So, this is mostly locale and other definitions, except where it is nice to prove something about definitional equivalence and the like (plus the intermediate lemmas that are necessary for doing so).

Minkowski spacetime = $(\mathcal{E}, \mathcal{P}, [\ldots])$ except in the notation here I've used $[[\ldots]]$ for $[\ldots]$ as Isabelle uses $[\ldots]$ for lists.

Except where stated otherwise all axioms are exactly as they appear in Schutz97. It is the independent axiomatic system provided in the main body of the book. The axioms O1-O6 are the axioms of order, and largely concern properties of the betweenness relation. I1-I7 are the axioms of incidence. I1-I3 are similar to axioms found in systems for Euclidean geometry. As compared to Hilbert's Foundations (HIn), our incidence axioms (In) are loosely identifiable as I1 $\rightarrow$ HI3, HI8; I2 $\rightarrow$ HI1; I3 $\rightarrow$ HI2. I4 fixes the dimension of the space. I5-I7 are what makes our system non-Galilean, and lead (I think) to Lorentz transforms (together with S?) and the ultimate speed limit. Axioms S and C and the axioms of symmetry and continuity, where the latter is what makes the system second order. Symmetry replaces all of Hilbert's axioms of congruence, when considered in the context of I5-I7.

3 MinkowskiPrimitive: I1-I3

Events $\mathcal{E}$, paths $\mathcal{P}$, and sprays. Sprays only need to refer to $\mathcal{E}$ and $\mathcal{P}$. Axiom *in-path-event* is covered in English by saying "a path is a set of events", but is necessary to have explicitly as an axiom as the types do not force it to be the case.

I think part of why Schutz has I1, together with the trickery $[\![\ \mathcal{E} \neq \{\}\ ]\!] \Longrightarrow \ldots$ in I4, is that then I4 talks *only* about dimension, and results such as *no-empty-paths* can be proved using only existence of elements and unreachable sets. In our case, it's also a question of ordering the sequence of axiom introductions: dimension should really go at the end, since it is not needed for quite a while; but many earlier proofs rely on the set of events being non-empty. It may be nice to have the existence of paths as a separate axiom too, which currently still relies on the axiom of dimension (Schutz has no such axiom either).

locale *MinkowskiPrimitive* =

fixes $\mathcal{E}$ :: $'a$ *set*
and $\mathcal{P}$ :: $('a\ set)$ *set*
assumes *in-path-event* [*simp*]: $[\![Q \in \mathcal{P};\ a \in Q]\!] \Longrightarrow a \in \mathcal{E}$

and *nonempty-events* [*simp*]: $\mathcal{E} \neq \{\}$

and *events-paths*: $[\![a \in \mathcal{E};\ b \in \mathcal{E};\ a \neq b]\!] \Longrightarrow \exists R{\in}\mathcal{P}.\ \exists S{\in}\mathcal{P}.\ a \in R \wedge b \in S \wedge R \cap S \neq \{\}$

and *eq-paths* [*intro*]: $[\![P \in \mathcal{P};\ Q \in \mathcal{P};\ a \in P;\ b \in P;\ a \in Q;\ b \in Q;\ a \neq b]\!] \Longrightarrow P = Q$
begin

This should be ensured by the additional axiom.

lemma *path-sub-events*:
$Q \in \mathcal{P} \Longrightarrow Q \subseteq \mathcal{E}$
by (*simp add*: *subsetI*)

lemma *paths-sub-power*:
$\mathcal{P} \subseteq Pow\ \mathcal{E}$
by (*simp add*: *path-sub-events subsetI*)

Define *path* for more terse statements. $a \neq b$ because a and b are being used to identify the path, and $a = b$ would not do that.

abbreviation *path* :: $'a\ set \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$ **where**
$path\ ab\ a\ b \equiv ab \in \mathcal{P} \wedge a \in ab \wedge b \in ab \wedge a \neq b$

abbreviation *path-ex* :: $'a \Rightarrow 'a \Rightarrow bool$ **where**
$path\text{-}ex\ a\ b \equiv \exists\, Q.\ path\ Q\ a\ b$

lemma *path-permute*:
$path\ ab\ a\ b = path\ ab\ b\ a$
by *auto*

abbreviation *path-of* :: $'a \Rightarrow 'a \Rightarrow 'a\ set$ **where**
$path\text{-}of\ a\ b \equiv THE\ ab.\ path\ ab\ a\ b$

lemma *path-of-ex*: $path\ (path\text{-}of\ a\ b)\ a\ b \longleftrightarrow path\text{-}ex\ a\ b$
using *theI'* [**where** $P{=}\lambda x.\ path\ x\ a\ b$] *eq-paths* **by** *blast*

lemma *path-unique*:
assumes $path\ ab\ a\ b$ **and** $path\ ab'\ a\ b$
shows $ab = ab'$
using *eq-paths assms* **by** *blast*

lemma *paths-cross-once*:
assumes *path-Q*: $Q \in \mathcal{P}$
and *path-R*: $R \in \mathcal{P}$

and *Q-neq-R*: $Q \neq R$
and *QR-nonempty*: $Q \cap R \neq \{\}$
shows $\exists! a \in \mathcal{E}.\ Q \cap R = \{a\}$
proof −
have *ab-inQR*: $\exists\, a \in \mathcal{E}.\ a \in Q \cap R$ **using** *QR-nonempty in-path-event path-Q* **by** *auto*
then obtain a **where** *a-event*: $a \in \mathcal{E}$ **and** *a-inQR*: $a \in Q \cap R$ **by** *auto*
have $Q \cap R = \{a\}$
proof (*rule ccontr*)
assume $Q \cap R \neq \{a\}$
then have $\exists\, b \in Q \cap R.\ b \neq a$ **using** *a-inQR* **by** *blast*
then have $Q = R$ **using** *eq-paths a-inQR path-Q path-R* **by** *auto*
thus *False* **using** *Q-neq-R* **by** *simp*
qed
thus *?thesis* **using** *a-event* **by** *blast*
qed

4 Primitives: Unreachable Subset (from an Event)

The $Q \in \mathcal{P} \wedge b \in \mathcal{E}$ constraints are necessary as the types as not expressive enough to do it on their own. Schutz's notation is: $Q(b, \emptyset)$.

definition *unreachable-subset* :: $'a\ set \Rightarrow 'a \Rightarrow 'a\ set$ (‹*unreach−on - from -*› [*100*, *100*]) **where**
unreach−on Q from b $\equiv \{x \in Q.\ Q \in \mathcal{P} \wedge b \in \mathcal{E} \wedge b \notin Q \wedge \neg(\textit{path-ex}\ b\ x)\}$

5 Primitives: Kinematic Triangle

definition *kinematic-triangle* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$ (‹△ - - -› [*100*, *100*, *100*] *100*) **where**
kinematic-triangle a b c $\equiv$
$a \in \mathcal{E} \wedge b \in \mathcal{E} \wedge c \in \mathcal{E} \wedge a \neq b \wedge a \neq c \wedge b \neq c$
$\wedge (\exists\, Q \in \mathcal{P}.\ \exists\, R \in \mathcal{P}.\ Q \neq R \wedge (\exists\, S \in \mathcal{P}.\ Q \neq S \wedge R \neq S$
$\wedge a \in Q \wedge b \in Q$
$\wedge a \in R \wedge c \in R$
$\wedge b \in S \wedge c \in S))$

A fuller, more explicit equivalent of △, to show that the above definition is sufficient.

lemma *tri-full*:
$\triangle\ a\ b\ c = (a \in \mathcal{E} \wedge b \in \mathcal{E} \wedge c \in \mathcal{E} \wedge a \neq b \wedge a \neq c \wedge b \neq c$
$\wedge (\exists\, Q \in \mathcal{P}.\ \exists\, R \in \mathcal{P}.\ Q \neq R \wedge (\exists\, S \in \mathcal{P}.\ Q \neq S \wedge R \neq S$
$\wedge a \in Q \wedge b \in Q \wedge c \notin Q$
$\wedge a \in R \wedge c \in R \wedge b \notin R$
$\wedge b \in S \wedge c \in S \wedge a \notin S)))$
unfolding *kinematic-triangle-def* **by** (*meson path-unique*)

6 Primitives: SPRAY

It's okay to not require $x \in \mathcal{E}$ because if $x \notin \mathcal{E}$ the *SPRAY* will be empty anyway, and if it's nonempty then $x \in \mathcal{E}$ is derivable.

definition *SPRAY* :: $'a \Rightarrow ('a\ set)\ set$ **where**
$SPRAY\ x \equiv \{R{\in}\mathcal{P}.\ x \in R\}$

definition *spray* :: $'a \Rightarrow 'a\ set$ **where**
$spray\ x \equiv \{y.\ \exists R{\in}SPRAY\ x.\ y \in R\}$

definition *is-SPRAY* :: $('a\ set)\ set \Rightarrow bool$ **where**
$is\text{-}SPRAY\ S \equiv \exists x{\in}\mathcal{E}.\ S = SPRAY\ x$

definition *is-spray* :: $'a\ set \Rightarrow bool$ **where**
$is\text{-}spray\ S \equiv \exists x{\in}\mathcal{E}.\ S = spray\ x$

Some very simple SPRAY and spray lemmas below.

lemma *SPRAY-event*:
$SPRAY\ x \neq \{\} \Longrightarrow x \in \mathcal{E}$
proof (*unfold SPRAY-def*)
assume *nonempty-SPRAY*: $\{R \in \mathcal{P}.\ x \in R\} \neq \{\}$
then have *x-in-path-R*: $\exists R \in \mathcal{P}.\ x \in R$ **by** *blast*
thus $x \in \mathcal{E}$ **using** *in-path-event* **by** *blast*
qed

lemma *SPRAY-nonevent*:
$x \notin \mathcal{E} \Longrightarrow SPRAY\ x = \{\}$
using *SPRAY-event* **by** *auto*

lemma *SPRAY-path*:
$P \in SPRAY\ x \Longrightarrow P \in \mathcal{P}$
by (*simp add*: *SPRAY-def*)

lemma *in-SPRAY-path*:
$P \in SPRAY\ x \Longrightarrow x \in P$
by (*simp add*: *SPRAY-def*)

lemma *source-in-SPRAY*:
$SPRAY\ x \neq \{\} \Longrightarrow \exists P \in SPRAY\ x.\ x \in P$
using *in-SPRAY-path* **by** *auto*

lemma *spray-event*:
$spray\ x \neq \{\} \Longrightarrow x \in \mathcal{E}$
proof (*unfold spray-def*)
assume $\{y.\ \exists R \in SPRAY\ x.\ y \in R\} \neq \{\}$
then have $\exists y.\ \exists R \in SPRAY\ x.\ y \in R$ **by** *simp*
then have $SPRAY\ x \neq \{\}$ **by** *blast*
thus $x \in \mathcal{E}$ **using** *SPRAY-event* **by** *simp*

qed

lemma *spray-nonevent*:
  $x \notin \mathcal{E} \Longrightarrow$ *spray* $x = \{\}$
using *spray-event* **by** *auto*

lemma *in-spray-event*:
  $y \in$ *spray* $x \Longrightarrow y \in \mathcal{E}$
proof (*unfold spray-def*)
  assume $y \in \{y.\ \exists R{\in}SPRAY\ x.\ y \in R\}$
  then have $\exists R{\in}SPRAY\ x.\ y \in R$ **by** (*rule CollectD*)
  then obtain R **where** *path-R*: $R \in \mathcal{P}$
    and *y-inR*: $y \in R$ **using** *SPRAY-path* **by** *auto*
  thus $y \in \mathcal{E}$ **using** *in-path-event* **by** *simp*
qed

lemma *source-in-spray*:
  spray $x \neq \{\} \Longrightarrow x \in$ *spray* x
proof −
  assume *nonempty-spray*: *spray* $x \neq \{\}$
  have *spray-eq*: *spray* $x = \{y.\ \exists R{\in}SPRAY\ x.\ y \in R\}$ **using** *spray-def* **by** *simp*
  then have *ex-in-SPRAY-path*: $\exists y.\ \exists R{\in}SPRAY\ x.\ y \in R$ **using** *nonempty-spray*
by *simp*
  show $x \in$ *spray* x **using** *ex-in-SPRAY-path spray-eq source-in-SPRAY* **by** *auto*
qed

7 Primitives: Path (In)dependence

"A subset of three paths of a SPRAY is dependent if there is a path which does not belong to the SPRAY and which contains one event from each of the three paths: we also say any one of the three paths is dependent on the other two. Otherwise the subset is independent." [Schutz97]

The definition of *SPRAY* constrains x, Q, R, S to be in $\mathcal{E}$ and $\mathcal{P}$.

definition *dep3-event* $Q\ R\ S\ x$
  $\equiv$ *card* $\{Q,R,S\} = 3 \wedge \{Q,R,S\} \subseteq SPRAY\ x$
  $\wedge\ (\exists T{\in}\mathcal{P}.\ T \notin SPRAY\ x \wedge Q{\cap}T{\neq}\{\} \wedge R{\cap}T{\neq}\{\} \wedge S{\cap}T{\neq}\{\})$

definition *dep3-spray* $Q\ R\ S\ SPR \equiv \exists x.\ SPRAY\ x = SPR\ \wedge$ *dep3-event* $Q\ R\ S\ x$

definition *dep3* $Q\ R\ S \equiv \exists x.$ *dep3-event* $Q\ R\ S\ x$

Some very simple lemmas related to *dep3-event*.

lemma *dep3-nonspray*:
  assumes *dep3-event* $Q\ R\ S\ x$
    shows $\exists P{\in}\mathcal{P}.\ P \notin SPRAY\ x$
  by (*metis assms dep3-event-def*)

lemma *dep3-path*:
 assumes *dep3-QRSx*: *dep3 Q R S*
 shows $Q \in \mathcal{P}$ $R \in \mathcal{P}$ $S \in \mathcal{P}$
 using *assms dep3-event-def dep3-def SPRAY-path insert-subset* **by** *auto*

lemma *dep3-distinct*:
 assumes *dep3-QRSx*: *dep3 Q R S*
 shows $Q \neq R$ $Q \neq S$ $R \neq S$
 using *assms dep3-def dep3-event-def* **by** (*simp-all add*: *card-3-dist*)

lemma *dep3-is-event*:
 dep3-event Q R S x $\Longrightarrow x \in \mathcal{E}$
using *SPRAY-event dep3-event-def* **by** *auto*

lemma *dep3-event-old*:
 dep3-event Q R S x $\longleftrightarrow Q \neq R \wedge Q \neq S \wedge R \neq S \wedge Q \in$ *SPRAY x* $\wedge R \in$ *SPRAY x* $\wedge S \in$ *SPRAY x*
 $\wedge$ ($\exists T \in \mathcal{P}$. $T \notin$ *SPRAY x* $\wedge$ ($\exists y \in Q$. $y \in T$) $\wedge$ ($\exists y \in R$. $y \in T$) $\wedge$ ($\exists y \in S$. $y \in T$))
 by (*rule iffI*; *unfold dep3-event-def*, (*simp add*: *card-3-dist*), *blast*)

lemma *dep3-event-permute* [*no-atp*]:
 assumes *dep3-event Q R S x*
 shows *dep3-event Q S R x dep3-event R Q S x dep3-event R S Q x*
 dep3-event S Q R x dep3-event S R Q x
using *dep3-event-old assms* **by** *auto*

lemma *dep3-permute* [*no-atp*]:
 assumes *dep3 Q R S*
 shows *dep3 Q S R dep3 R Q S dep3 R S Q*
 and *dep3 S Q R dep3 S R Q*
 using *dep3-event-permute dep3-def assms* **by** *meson+*

"We next give recursive definitions of dependence and independence which will be used to characterize the concept of dimension. A path T is dependent on the set of n paths (where $n \geq 3$)

$$S = \{Q_i : i = 1, 2, ..., n; Q_i \in \mathrm{SPRAY} x\}$$

if it is dependent on two paths S_1 and S_2, where each of these two paths is dependent on some subset of $n - 1$ paths from the set S." [Schutz97]

inductive *dep-path* :: *'a set* $\Rightarrow$ (*'a set*) *set* $\Rightarrow$ *bool* **where**
 dep-3: *dep3 T A B* $\Longrightarrow$ *dep-path T* {*A*, *B*}
| *dep-n*: ⟦*dep3 T S1 S2*; *dep-path S1 S'*; *dep-path S2 S''*; $S \subseteq$ *SPRAY x*;
 $S' \subseteq S$; $S'' \subseteq S$; *Suc* (*card S'*) = *card S*; *Suc* (*card S''*) = *card S*⟧ $\Longrightarrow$ *dep-path T S*

lemma *card-Suc-ex*:
 assumes *card A* = *Suc* (*card B*) $B \subseteq A$

shows ∃ *b*. *A* = *insert b B* ∧ *b*∉*B*
proof −
have *finite A* **using** *assms*(*1*) *card-ge-0-finite card.infinite* **by** *fastforce*
obtain *b* **where** *b*∈*A*−*B*
by (*metis Diff-eq-empty-iff all-not-in-conv assms n-not-Suc-n subset-antisym*)
show ∃ *b*. *A* = *insert b B* ∧ *b*∉*B*
proof
show *A* = *insert b B* ∧ *b*∉*B*
using ‹*b*∈*A*−*B*› ‹*finite A*› *assms*
by (*metis DiffD1 DiffD2 Diff-insert-absorb Diff-single-insert card-insert-disjoint card-subset-eq insert-absorb rev-finite-subset*)
qed
qed

lemma *union-of-subsets-by-singleton*:
assumes *Suc* (*card S′*) = *card S Suc* (*card S″*) = *card S*
and *S′* ≠ *S″ S′* ⊆ *S S″* ⊆ *S*
shows *S′* ∪ *S″* = *S*
proof −
obtain *x y* **where** *x*: *insert x S′* = *S x*∉*S′* **and** *y*: *insert y S″* = *S y*∉*S″*
using *assms*(*1*,*2*,*4*,*5*) **by** (*metis card-Suc-ex*)
have *x*≠*y* **using** *x y assms*(*3*) **by** (*metis insert-eq-iff*)
thus *?thesis* **using** *x*(*1*) *y*(*1*) **by** *blast*
qed

lemma *dep-path-card-2*: *dep-path T S* ⟹ *card S* ≥ *2*
by (*induct rule*: *dep-path.induct*, *simp add*: *dep3-def dep3-event-old*, *linarith*)

"We also say that the set of $n+1$ paths $S \cup \{T\}$ is a dependent set." [Schutz97] Starting from this constructive definition, the below gives an analytical one.

definition *dep-set* :: (*′a set*) *set* ⇒ *bool* **where**
dep-set S ≡ ∃ *S′*⊆*S*. ∃ *P*∈(*S*−*S′*). *dep-path P S′*

Notice that the relation between *dep-set* and *dep-path* becomes somewhat meaningless in the case where we apply *dep-path* to an element of the set. This is because sets have no duplicate members, and we do not mirror the idea that scalar multiples of vectors linearly depend on those vectors: paths in a SPRAY are (in the $\mathbb{R}^4$ model) already equivalence classes of vectors that are scalar multiples of each other.

lemma *dep-path-imp-dep-set*:
assumes *dep-path P S P*∉*S*
shows *dep-set* (*insert P S*)
using *assms dep-set-def* **by** *auto*

lemma *dep-path-for-set-members*:
assumes *P*∈*S*
shows *dep-set S* = *dep-set* (*insert P S*)
by (*simp add*: *assms*(*1*) *insert-absorb*)

lemma *dependent-superset*:
  assumes *dep-set A* **and** $A \subseteq B$
  shows *dep-set B*
  using *assms dep-set-def*
  by (*meson Diff-mono dual-order.trans in-mono order-refl*)

lemma *path-in-dep-set*:
  assumes *dep3 P Q R*
  shows *dep-set* $\{P,Q,R\}$
  using *dep-3 assms dep3-def dep-set-def dep3-event-old*
  by (*metis DiffI insert-iff singletonD subset-insertI*)

lemma *path-in-dep-set2a*:
  assumes *dep3 P Q R*
  shows *dep-path P* $\{P,Q,R\}$
proof
  let $?S' = \{P,R\}$
  let $?S'' = \{P,Q\}$
  have *all-neq*: $P \neq Q$ $P \neq R$ $R \neq Q$ **using** *assms dep3-def dep3-event-old* **by** *auto*
  show *dep3 P Q R* **using** *assms dep3-event-def* **by** (*simp add*: *dep-3*)
  show *dep-path Q* $?S'$ **using** *assms dep3-event-permute*(*2*) *dep-3 dep3-def* **by** *meson*
  show *dep-path R* $?S''$ **using** *assms dep3-event-permute*(*4*) *dep-3 dep3-def* **by** *meson*
  show $?S' \subseteq \{P, Q, R\}$ **by** *simp*
  show $?S'' \subseteq \{P, Q, R\}$ **by** *simp*
  show *Suc* (*card* $?S'$) = *card* $\{P, Q, R\}$ *Suc* (*card* $?S''$) = *card* $\{P, Q, R\}$
    using *all-neq card-insert-disjoint* **by** *auto*
  show $\{P, Q, R\} \subseteq$ *SPRAY* (*SOME x. dep3-event P Q R x*)
    using *assms dep3-def dep3-event-def* **by** (*metis some-eq-ex*)
qed

definition *indep-set* :: ('*a set*) *set* $\Rightarrow$ *bool* **where**
  indep-set S $\equiv \neg$ *dep-set S*

lemma *no-dep-in-indep*: *indep-set S* $\Longrightarrow \neg(\exists\, T \subseteq S.$ *dep-set T*)
  using *indep-set-def dependent-superset* **by** *blast*

lemma *indep-set-alt-intro*: $\neg(\exists\, T \subseteq S.$ *dep-set T*) $\Longrightarrow$ *indep-set S*
  using *indep-set-def* **by** *blast*

lemma *indep-set-alt*: *indep-set S* $\longleftrightarrow \neg(\exists\, S' \subseteq S.$ *dep-set* S')
  using *no-dep-in-indep indep-set-alt-intro* **by** *blast*

lemma *dep-set S* $\vee$ *indep-set S*
  by (*simp add*: *indep-set-def*)

8 Primitives: 3-SPRAY

"We now make the following definition which enables us to specify the dimensions of Minkowski space-time. A SPRAY is a 3-SPRAY if: i) it contains four independent paths, and ii) all paths of the SPRAY are dependent on these four paths." [Schutz97]

definition *n-SPRAY-basis* :: *nat* ⇒ *'a set set* ⇒ *'a* ⇒ *bool* **where**
n-SPRAY-basis n S x ≡ *S*⊆*SPRAY x* ∧ *card S* = (*Suc n*) ∧ *indep-set S* ∧ (∀ *P*∈*SPRAY x*. *dep-path P S*)

definition *n-SPRAY* (‹-−*SPRAY* -› [*100*,*100*]) **where**
n−*SPRAY x* ≡ ∃ *S*⊆*SPRAY x*. *card S* = (*Suc n*) ∧ *indep-set S* ∧ (∀ *P*∈*SPRAY x*. *dep-path P S*)

abbreviation *three-SPRAY x* ≡ *3*−*SPRAY x*

lemma *n-SPRAY-intro*:
assumes *S*⊆*SPRAY x card S* = (*Suc n*) *indep-set S* ∀ *P*∈*SPRAY x*. *dep-path P S*
shows *n*−*SPRAY x*
using *assms n-SPRAY-def* **by** *blast*

lemma *three-SPRAY-alt*:
three-SPRAY x = (∃ *S1 S2 S3 S4*.
S1 ≠ *S2* ∧ *S1* ≠ *S3* ∧ *S1* ≠ *S4* ∧ *S2* ≠ *S3* ∧ *S2* ≠ *S4* ∧ *S3* ≠ *S4*
∧ *S1* ∈ *SPRAY x* ∧ *S2* ∈ *SPRAY x* ∧ *S3* ∈ *SPRAY x* ∧ *S4* ∈ *SPRAY x*
∧ (*indep-set* {*S1*, *S2*, *S3*, *S4*})
∧ (∀ *S*∈*SPRAY x*. *dep-path S* {*S1*,*S2*,*S3*,*S4*}))
(**is** *three-SPRAY x* ⟷ *?three-SPRAY' x*)
proof
assume *three-SPRAY x*
then obtain *S* **where** *ns*: *S*⊆*SPRAY x card S* = *4 indep-set S* ∀ *P*∈*SPRAY x*. *dep-path P S*
using *n-SPRAY-def* **by** *auto*
then obtain S_1 S_2 S_3 S_4 **where**
$S = \{S_1, S_2, S_3, S_4\}$ **and**
$S_1 \neq S_2 \wedge S_1 \neq S_3 \wedge S_1 \neq S_4 \wedge S_2 \neq S_3 \wedge S_2 \neq S_4 \wedge S_3 \neq S_4$ **and**
$S_1 \in SPRAY\ x \wedge S_2 \in SPRAY\ x \wedge S_3 \in SPRAY\ x \wedge S_4 \in SPRAY\ x$
using *card-4-eq* **by** (*smt* (*verit*) *insert-subset ns*)
thus *?three-SPRAY' x*
by (*metis ns*(*3*,*4*))
next
assume *?three-SPRAY' x*
then obtain S_1 S_2 S_3 S_4 **where** *ns*:
$S_1 \neq S_2 \wedge S_1 \neq S_3 \wedge S_1 \neq S_4 \wedge S_2 \neq S_3 \wedge S_2 \neq S_4 \wedge S_3 \neq S_4$
$S_1 \in SPRAY\ x \wedge S_2 \in SPRAY\ x \wedge S_3 \in SPRAY\ x \wedge S_4 \in SPRAY\ x$
indep-set $\{S_1, S_2, S_3, S_4\}$
∀ *S*∈*SPRAY x*. *dep-path S* $\{S_1,S_2,S_3,S_4\}$

by *metis*
show *three-SPRAY x*
apply (*intro n-SPRAY-intro*[*of* $\{S_1, S_2, S_3, S_4\}$])
by (*simp add*: *ns*)+
qed

lemma *three-SPRAY-intro*:
assumes $S1 \neq S2 \wedge S1 \neq S3 \wedge S1 \neq S4 \wedge S2 \neq S3 \wedge S2 \neq S4 \wedge S3 \neq S4$
and $S1 \in SPRAY\ x \wedge S2 \in SPRAY\ x \wedge S3 \in SPRAY\ x \wedge S4 \in SPRAY\ x$
and *indep-set* $\{S1, S2, S3, S4\}$
and $\forall S{\in}SPRAY\ x.$ *dep-path* $S\ \{S1,S2,S3,S4\}$
shows *three-SPRAY x*
unfolding *three-SPRAY-alt* **by** (*metis assms*)

Lemma *is-three-SPRAY* says "this set of sets of elements is a set of paths which is a 3-SPRAY". Lemma *three-SPRAY-ge4* just extracts a bit of the definition.

definition *is-three-SPRAY* :: ('*a set*) *set* ⇒ *bool* **where**
is-three-SPRAY $S \equiv \exists\ x.\ S = SPRAY\ x \wedge 3{-}SPRAY\ x$

lemma *three-SPRAY-ge4*:
assumes *three-SPRAY x*
shows $\exists\, Q1{\in}\mathcal{P}.\ \exists\, Q2{\in}\mathcal{P}.\ \exists\, Q3{\in}\mathcal{P}.\ \exists\, Q4{\in}\mathcal{P}.\ Q1 \neq Q2 \wedge Q1 \neq Q3 \wedge Q1 \neq Q4 \wedge Q2 \neq Q3 \wedge Q2 \neq Q4 \wedge Q3 \neq Q4$
using *assms three-SPRAY-alt SPRAY-path* **by** *meson*

end

9 MinkowskiBetweenness: O1-O5

In O4, I have removed the requirement that $a \neq d$ in order to prove negative betweenness statements as Schutz does. For example, if we have $[abc]$ and $[bca]$ we want to conclude $[aba]$ and claim "contradiction!", but we can't as long as we mandate that $a \neq d$.

locale *MinkowskiBetweenness* = *MinkowskiPrimitive* +
fixes *betw* :: '*a* ⇒ '*a* ⇒ '*a* ⇒ *bool* (‹[-;-;-]›)

assumes *abc-ex-path*: $[a;b;c] \Longrightarrow \exists\, Q{\in}\mathcal{P}.\ a \in Q \wedge b \in Q \wedge c \in Q$

and *abc-sym*: $[a;b;c] \Longrightarrow [c;b;a]$

and *abc-ac-neq*: $[a;b;c] \Longrightarrow a \neq c$

and *abc-bcd-abd* [*intro*]: $[\![[a;b;c];\ [b;c;d]]\!] \Longrightarrow [a;b;d]$

and *some-betw*: $[\![Q \in \mathcal{P};\ a \in Q;\ b \in Q;\ c \in Q;\ a \neq b;\ a \neq c;\ b \neq c]\!]$
$\Longrightarrow [a;b;c] \vee [b;c;a] \vee [c;a;b]$

begin

The next few lemmas either provide the full axiom from the text derived from a new simpler statement, or provide some very simple fundamental additions which make sense to prove immediately before starting, usually related to set-level things that should be true which fix the type-level ambiguity of 'a.

lemma *betw-events*:
  assumes *abc*: $[a;b;c]$
  shows $a \in \mathcal{E} \wedge b \in \mathcal{E} \wedge c \in \mathcal{E}$
proof −
  have $\exists\, Q{\in}\mathcal{P}.\ a \in Q \wedge b \in Q \wedge c \in Q$ **using** *abc-ex-path abc* **by** *simp*
  thus *?thesis* **using** *in-path-event* **by** *auto*
qed

This shows the shorter version of O5 is equivalent.

lemma *O5-still-O5* [*no-atp*]:
  $((Q \in \mathcal{P} \wedge \{a,b,c\} \subseteq Q \wedge a \in \mathcal{E} \wedge b \in \mathcal{E} \wedge c \in \mathcal{E} \wedge a \neq b \wedge a \neq c \wedge b \neq c)$
    $\longrightarrow [a;b;c] \vee [b;c;a] \vee [c;a;b])$
   $=$
  $((Q \in \mathcal{P} \wedge \{a,b,c\} \subseteq Q \wedge a \in \mathcal{E} \wedge b \in \mathcal{E} \wedge c \in \mathcal{E} \wedge a \neq b \wedge a \neq c \wedge b \neq c)$
    $\longrightarrow [a;b;c] \vee [b;c;a] \vee [c;a;b] \vee [c;b;a] \vee [a;c;b] \vee [b;a;c])$
by (*auto simp add*: *abc-sym*)

lemma *some-betw-xor*:
  $[\![Q \in \mathcal{P};\ a \in Q;\ b \in Q;\ c \in Q;\ a \neq b;\ a \neq c;\ b \neq c]\!]$
        $\Longrightarrow ([a;b;c] \wedge \neg\, [b;c;a] \wedge \neg\, [c;a;b])$
         $\vee\ ([b;c;a] \wedge \neg\, [a;b;c] \wedge \neg\, [c;a;b])$
         $\vee\ ([c;a;b] \wedge \neg\, [a;b;c] \wedge \neg\, [b;c;a])$
by (*meson abc-ac-neq abc-bcd-abd some-betw*)

The lemma *abc-abc-neq* is the full O3 as stated by Schutz.

lemma *abc-abc-neq*:
  assumes *abc*: $[a;b;c]$
  shows $a \neq b \wedge a \neq c \wedge b \neq c$
using *abc-sym abc-ac-neq assms abc-bcd-abd* **by** *blast*

lemma *abc-bcd-acd*:
  assumes *abc*: $[a;b;c]$
    and *bcd*: $[b;c;d]$
  shows $[a;c;d]$
proof −
  have *cba*: $[c;b;a]$ **using** *abc-sym abc* **by** *simp*
  have *dcb*: $[d;c;b]$ **using** *abc-sym bcd* **by** *simp*
  have $[d;c;a]$ **using** *abc-bcd-abd dcb cba* **by** *blast*
  thus *?thesis* **using** *abc-sym* **by** *simp*
qed

lemma *abc-only-cba*:
assumes $[a;b;c]$
shows $\neg\ [b;a;c]$ $\neg\ [a;c;b]$ $\neg\ [b;c;a]$ $\neg\ [c;a;b]$
using *abc-sym abc-abc-neq abc-bcd-abd assms* **by** *blast+*

10 Betweenness: Unreachable Subset Via a Path

definition *unreachable-subset-via* :: *'a set ⇒ 'a ⇒ 'a set ⇒ 'a ⇒ 'a set* **where**
unreachable-subset-via Q Qa R x ≡ {*Qy*. [*x*;*Qy*;*Qa*] ∧ (∃ *Rw*∈*R*. *Qa* ∈ *unreach−on Q from Rw* ∧ *Qy* ∈ *unreach−on Q from Rw*)}

definition *unreachable-subset-via-notation* (‹*unreach−via - on - from - to -*› [*100*, *100*, *100*, *100*] *100*)
where *unreach−via P on Q from a to x* ≡ *unreachable-subset-via Q a P x*

11 Betweenness: Chains

named-theorems *chain-defs*
named-theorems *chain-alts*

11.1 Locally ordered chains with indexing

Definitions for Schutz's chains, with local order only.

A chain can be: (i) a set of two distinct events connected by a path, or ...

definition *short-ch* :: *'a set ⇒ bool* **where**
short-ch X ≡ *card X* = *2* ∧ (∃ *P*∈𝒫. *X* ⊆ *P*)

lemma *short-ch-alt*[*chain-alts*]:
short-ch X = (∃ *x*∈*X*. ∃ *y*∈*X*. *path-ex x y* ∧ ¬(∃ *z*∈*X*. *z*≠*x* ∧ *z*≠*y*))
short-ch X = (∃ *x y*. *X* = {*x*,*y*} ∧ *path-ex x y*)
unfolding *short-ch-def*
apply (*simp add*: *card-2-iff'*, *smt* (*verit*, *ccfv-SIG*) *in-mono subsetI*)
by (*metis card-2-iff empty-subsetI insert-subset*)

lemma *short-ch-intros*:
⟦*x*∈*X*; *y*∈*X*; *path-ex x y*; ¬(∃ *z*∈*X*. *z*≠*x* ∧ *z*≠*y*)⟧ ⟹ *short-ch X*
⟦*X* = {*x*,*y*}; *path-ex x y*⟧ ⟹ *short-ch X*
by (*auto simp*: *short-ch-alt*)

lemma *short-ch-path*: *short-ch* {*x*,*y*} ⟷ *path-ex x y*
unfolding *short-ch-def* **by** *force*

... a set of at least three events such that any three adjacent events are ordered. Notice infinite sets have card 0, because card gives a natural number always.

definition *local-long-ch-by-ord* :: (*nat ⇒ 'a*) ⇒ *'a set ⇒ bool* **where**

local-long-ch-by-ord f X ≡ (*infinite X* ∨ *card X* ≥ *3*) ∧ *local-ordering f betw X*

lemma *local-long-ch-by-ord-alt* [*chain-alts*]:
  local-long-ch-by-ord f X =
    (∃ *x*∈*X*. ∃ *y*∈*X*. ∃ *z*∈*X*. *x*≠*y* ∧ *y*≠*z* ∧ *x*≠*z* ∧ *local-ordering f betw X*)
  (**is** - = *?ch f X*)
proof
  assume *asm*: *local-long-ch-by-ord f X*
  {
    assume *card X* ≥ *3*
    then have ∃ *x y z*. *x*≠*y* ∧ *y*≠*z* ∧ *x*≠*z* ∧ {*x*,*y*,*z*}⊆*X*
      apply (*simp add*: *eval-nat-numeral*)
      by (*auto simp add*: *card-le-Suc-iff*)
  } moreover {
    assume *infinite X*
    then have ∃ *x y z*. *x*≠*y* ∧ *y*≠*z* ∧ *x*≠*z* ∧ {*x*,*y*,*z*}⊆*X*
      using *inf-3-elms bot.extremum* **by** *fastforce*
  }
  ultimately show *?ch f X* **using** *asm* **unfolding** *local-long-ch-by-ord-def* **by** *auto*
next
  assume *asm*: *?ch f X*
  then obtain *x y z* **where** *xyz*: {*x*,*y*,*z*}⊆*X* ∧ *x* ≠ *y* ∧ *y* ≠ *z* ∧ *x* ≠ *z*
    apply (*simp add*: *eval-nat-numeral*) **by** *auto*
  hence *card X* ≥ *3* ∨ *infinite X*
    apply (*simp add*: *eval-nat-numeral*)
    by (*smt* (*z3*) *xyz card.empty card-insert-if card-subset finite.emptyI finite-insert insertE*
      insert-absorb insert-not-empty)
  thus *local-long-ch-by-ord f X* **unfolding** *local-long-ch-by-ord-def* **using** *asm* **by** *auto*
qed

lemma *short-xor-long*:
  shows *short-ch Q* ⟹ ∄*f*. *local-long-ch-by-ord f Q*
    and *local-long-ch-by-ord f Q* ⟹ ¬ *short-ch Q*
  unfolding *chain-alts* **by** (*metis*)+

Any short chain can have an "ordering" defined on it: this isn't the ternary ordering *betw* that is used for triplets of elements, but merely an indexing function that fixes the "direction" of the chain, i.e. maps *0* to one element and *1* to the other. We define this in order to be able to unify chain definitions with those for long chains. Thus the indexing function *f* of *short-ch-by-ord f Q* has a similar status to the ordering on a long chain in many regards: e.g. it implies that $f(0 \ldots |Q| - 1) \subseteq Q$.

definition *short-ch-by-ord* :: (*nat*⇒*'a*) ⇒ *'a set* ⇒ *bool*
  where *short-ch-by-ord f Q* ≡ *Q* = {*f 0*, *f 1*} ∧ *path-ex* (*f 0*) (*f 1*)

lemma *short-ch-equiv* [*chain-alts*]: ∃*f*. *short-ch-by-ord f Q* ⟷ *short-ch Q*
proof –
  { assume *asm*: *short-ch Q*
    obtain *x y* **where** *xy*: {*x*,*y*}⊆*Q path-ex x y*
      using *asm short-ch-alt*(*2*) **by** (*auto simp*: *short-ch-def*)
    let *?f* = λ*n*::*nat*. *if n*=*0 then x else y*
    have ∃*f*. (∃*x y*. *Q* = {*x*, *y*} ∧ *f* (*0*::*nat*) = *x* ∧ *f 1* = *y* ∧ (∃*Q*. *path Q x y*))
      apply (*rule exI*[*of - ?f*]) **using** *asm xy short-ch-alt*(*2*) **by** *auto*
  } moreover {
    fix *f* **assume** *asm*: *short-ch-by-ord f Q*
    have *card Q* = *2* (∃*P*∈𝒫. *Q* ⊆ *P*)
      using *asm short-ch-by-ord-def* **by** *auto*
  } ultimately show *?thesis* **by** (*metis short-ch-by-ord-def short-ch-def*)
qed

lemma *short-ch-card*:
  short-ch-by-ord f Q ⟹ *card Q* = *2*
  short-ch Q ⟹ *card Q* = *2*
  using *short-ch-by-ord-def short-ch-def short-ch-equiv* **by** *auto*

lemma *short-ch-sym*:
  assumes *short-ch-by-ord f Q*
  shows *short-ch-by-ord* (λ*n*. *if n*=*0 then f 1 else f 0*) *Q*
  using *assms* **unfolding** *short-ch-by-ord-def* **by** *auto*

lemma *short-ch-ord-in*:
  assumes *short-ch-by-ord f Q*
  shows *f 0* ∈ *Q f 1* ∈ *Q*
  using *assms* **unfolding** *short-ch-by-ord-def* **by** *auto*

Does this restrict chains to lie on paths? Proven in *TemporalOrderingOnPath*'s Interlude!

definition *ch-by-ord* (‹[-⇝-]›) **where**
  [*f*⇝*X*] ≡ *short-ch-by-ord f X* ∨ *local-long-ch-by-ord f X*

definition *ch* :: *'a set* ⇒ *bool* **where** *ch X* ≡ ∃*f*. [*f*⇝*X*]

declare *short-ch-def* [*chain-defs*]
  and *local-long-ch-by-ord-def* [*chain-defs*]
  and *ch-by-ord-def* [*chain-defs*]
  and *short-ch-by-ord-def* [*chain-defs*]

We include alternative definitions in the *chain-defs* set, because we do not want arbitrary orderings to appear on short chains. Unless an ordering for a short chain is explicitly written down by the user, we shouldn't introduce a *short-ch-by-ord* when e.g. unfolding.

lemma *ch-alt*[*chain-defs*]: *ch X* ≡ *short-ch X* ∨ (∃*f*. *local-long-ch-by-ord f X*)
  unfolding *ch-def ch-by-ord-def* **using** *chain-defs short-ch-intros*(*2*)
  by (*smt* (*verit*) *short-ch-equiv*)

Since $f(0)$ is always in the chain, and plays a special role particularly for infinite chains (as the 'endpoint', the non-finite edge) let us fix it straight in the definition. Notice we require both *infinite X* and *long-ch-by-ord*, thus circumventing infinite Isabelle sets having cardinality 0.

definition *infinite-chain* :: (*nat* ⇒ *'a*) ⇒ *'a set* ⇒ *bool* **where**
infinite-chain f Q ≡ *infinite Q* ∧ [*f*⇝*Q*]

declare *infinite-chain-def* [*chain-defs*]

lemma *infinite-chain-alt*[*chain-alts*]:
infinite-chain f Q ⟷ *infinite Q* ∧ *local-ordering f betw Q*
unfolding *chain-defs* **by** *fastforce*

definition *infinite-chain-with* :: (*nat* ⇒ *'a*) ⇒ *'a set* ⇒ *'a* ⇒ *bool* (‹[-⇝-|- ..]›)
where
infinite-chain-with f Q x ≡ *infinite-chain f Q* ∧ *f 0* = *x*

declare *infinite-chain-with-def* [*chain-defs*]

lemma *infinite-chain f Q* ⟷ [*f*⇝*Q*|*f 0*..]
by (*simp add*: *infinite-chain-with-def*)

definition *finite-chain* :: (*nat* ⇒ *'a*) ⇒ *'a set* ⇒ *bool* **where**
finite-chain f Q ≡ *finite Q* ∧ [*f*⇝*Q*]

declare *finite-chain-def* [*chain-defs*]

lemma *finite-chain-alt*[*chain-alts*]: *finite-chain f Q* ⟷ *short-ch-by-ord f Q* ∨ (*finite Q* ∧ *local-long-ch-by-ord f Q*)
unfolding *chain-defs* **by** *auto*

definition *finite-chain-with* :: (*nat* ⇒ *'a*) ⇒ *'a set* ⇒ *'a* ⇒ *'a* ⇒ *bool* (‹[-⇝-|- .. -]›) **where**
[*f*⇝*Q*|*x*..*y*] ≡ *finite-chain f Q* ∧ *f 0* = *x* ∧ *f* (*card Q* − *1*) = *y*

declare *finite-chain-with-def* [*chain-defs*]

lemma *finite-chain f Q* ⟷ [*f*⇝*Q*|*f 0* .. *f* (*card Q* − *1*)]
by (*simp add*: *finite-chain-with-def*)

lemma *finite-chain-with-alt* [*chain-alts*]:
[*f*⇝*Q*|*x*..*z*] ⟷ (*short-ch-by-ord f Q* ∨ (*card Q* ≥ *3* ∧ *local-ordering f betw Q*)) ∧
x = *f 0* ∧ *z* = *f* (*card Q* − *1*)
unfolding *chain-defs*
by (*metis card.infinite finite.emptyI finite.insertI not-numeral-le-zero*)

lemma *finite-chain-with-cases*:
assumes [*f*⇝*Q*|*x*..*z*]

obtains
  (*short*) *x* = *f 0 z* = *f* (*card Q* − *1*) *short-ch-by-ord f Q*
  | (*long*) *x* = *f 0 z* = *f* (*card Q* − *1*) *card Q* ≥ *3 local-long-ch-by-ord f Q*
  using *assms finite-chain-with-alt* **by** (*meson local-long-ch-by-ord-def*)

definition *finite-long-chain-with*:: (*nat*⇒*'a*) ⇒ *'a set* ⇒ *'a* ⇒ *'a* ⇒ *'a* ⇒ *bool*
(‹[-⇝-|-..-..-]›)
  where [*f*⇝*Q*|*x*..*y*..*z*] ≡ [*f*⇝*Q*|*x*..*z*] ∧ *x*≠*y* ∧ *y*≠*z* ∧ *y*∈*Q*

declare *finite-long-chain-with-def* [*chain-defs*]

lemma *points-in-chain*:
  assumes [*f*⇝*Q*|*x*..*z*]
  shows *x*∈*Q* ∧ *z*∈*Q*
  apply (*cases rule*: *finite-chain-with-cases*[*OF assms*])
  using *short-ch-card*(*1*) *short-ch-ord-in* **by** (*simp add*: *chain-defs local-ordering-def*[*of f betw Q*])+

lemma *points-in-long-chain*:
  assumes [*f*⇝*Q*|*x*..*y*..*z*]
  shows *x*∈*Q* **and** *y*∈*Q* **and** *z*∈*Q*
  using *points-in-chain finite-long-chain-with-def assms* **by** *meson*+

lemma *finite-chain-with-card-less3*:
  assumes [*f*⇝*Q*|*x*..*z*]
    and *card Q* < *3*
  shows *short-ch-by-ord f Q z* = *f 1*
proof −
  show *1*: *short-ch-by-ord f Q*
    using *finite-chain-with-alt assms* **by** *simp*
  thus *z* = *f 1*
    using *assms*(*1*) **by** (*auto simp*: *eval-nat-numeral chain-defs*)
qed

lemma *ch-long-if-card-geq3*:
  assumes *ch X*
      and *card X* ≥ *3*
    shows ∃*f*. *local-long-ch-by-ord f X*
proof −
  show ∃*f*. *local-long-ch-by-ord f X*
  proof (*rule ccontr*)
    assume ∄*f*. *local-long-ch-by-ord f X*
    hence *short-ch X*
      using *assms*(*1*) **unfolding** *chain-defs* **by** *auto*
    obtain *x y z* **where** *x*∈*X* ∧ *y*∈*X* ∧ *z*∈*X* **and** *x*≠*y* ∧ *y*≠*z* ∧ *x*≠*z*
      using *assms*(*2*) **by** (*auto simp add*: *card-le-Suc-iff numeral-3-eq-3*)
    thus *False*
      using ‹*short-ch X*› **by** (*metis short-ch-alt*(*1*))

qed
qed

lemma *ch-short-if-card-less3*:
assumes *ch Q*
and *card Q < 3*
and *finite Q*
shows ∃*f. short-ch-by-ord f Q*
using *short-ch-equiv finite-chain-with-card-less3*
by (*metis assms ch-alt diff-is-0-eq′ less-irrefl-nat local-long-ch-by-ord-def zero-less-diff*)

lemma *three-in-long-chain*:
assumes *local-long-ch-by-ord f X*
obtains *x y z* **where** *x*∈*X* **and** *y*∈*X* **and** *z*∈*X* **and** *x*≠*y* **and** *x*≠*z* **and** *y*≠*z*
using *assms*(*1*) *local-long-ch-by-ord-alt* **by** *auto*

lemma *short-ch-card-2*:
assumes *ch-by-ord f X*
shows *short-ch X* ⟷ *card X = 2*
using *assms* **unfolding** *chain-defs* **using** *card-2-iff′ card-gt-0-iff* **by** *fastforce*

lemma *long-chain-card-geq*:
assumes *local-long-ch-by-ord f X* **and** *fin*: *finite X*
shows *card X* ≥ *3*
proof −
obtain *x y z* **where** *xyz*: *x*∈*X y*∈*X z*∈*X* **and** *neq*: *x*≠*y x*≠*z y*≠*z*
using *three-in-long-chain assms* **by** *blast*
let *?S* = {*x*,*y*,*z*}
have *?S* ⊆ *X*
by (*simp add*: *xyz*)
moreover have *card ?S* ≥ *3*
using *antisym* ‹*x* ≠ *y*› ‹*x* ≠ *z*› ‹*y* ≠ *z*› **by** *auto*
ultimately show *?thesis*
by (*meson neq fin three-subset*)
qed

lemma *fin-chain-card-geq-2*:
assumes [*f*⇝*X*|*a*..*b*]
shows *card X* ≥ *2*
using *finite-chain-with-def* **apply** (*cases short-ch X*)
using *short-ch-card-2*
apply (*metis dual-order.eq-iff short-ch-def*)
using *assms chain-defs not-less* **by** *fastforce*

12 Betweenness: Rays and Intervals

"Given any two distinct events a, b of a path we define the segment $(ab) = \{x : [a\ x\ b],\ x \in ab\}$" [Schutz97] Our version is a little different, because it is defined for any a, b of type $'a$. Thus we can have empty set segments, while Schutz can prove (once he proves path density) that segments are never empty.

definition *segment* :: $'a \Rightarrow 'a \Rightarrow 'a\ set$
where $segment\ a\ b \equiv \{x{::}'a.\ \exists\, ab.\ [a;x;b] \wedge x{\in}ab \wedge path\ ab\ a\ b\}$

abbreviation *is-segment* :: $'a\ set \Rightarrow bool$
where $is\text{-}segment\ ab \equiv (\exists\, a\ b.\ ab = segment\ a\ b)$

definition *interval* :: $'a \Rightarrow 'a \Rightarrow 'a\ set$
where $interval\ a\ b \equiv insert\ b\ (insert\ a\ (segment\ a\ b))$

abbreviation *is-interval* :: $'a\ set \Rightarrow bool$
where $is\text{-}interval\ ab \equiv (\exists\, a\ b.\ ab = interval\ a\ b)$

definition *prolongation* :: $'a \Rightarrow 'a \Rightarrow 'a\ set$
where $prolongation\ a\ b \equiv \{x{::}'a.\ \exists\, ab.\ [a;b;x] \wedge x{\in}ab \wedge path\ ab\ a\ b\}$

abbreviation *is-prolongation* :: $'a\ set \Rightarrow bool$
where $is\text{-}prolongation\ ab \equiv \exists\, a\ b.\ ab = prolongation\ a\ b$

I think this is what Schutz actually meant, maybe there is a typo in the text? Notice that $b \in ray\ a\ b$ for any a, always. Cf the comment on *segment-def*. Thus $\exists\, ray\ a\ b \neq \{\}$ is no guarantee that a path ab exists.

definition *ray* :: $'a \Rightarrow 'a \Rightarrow 'a\ set$
where $ray\ a\ b \equiv insert\ b\ (segment\ a\ b \cup prolongation\ a\ b)$

abbreviation *is-ray* :: $'a\ set \Rightarrow bool$
where $is\text{-}ray\ R \equiv \exists\, a\ b.\ R = ray\ a\ b$

definition *is-ray-on* :: $'a\ set \Rightarrow 'a\ set \Rightarrow bool$
where $is\text{-}ray\text{-}on\ R\ P \equiv P{\in}\mathcal{P} \wedge R{\subseteq}P \wedge is\text{-}ray\ R$

This is as in Schutz. Notice b is not in the ray through b?

definition *ray-Schutz* :: $'a \Rightarrow 'a \Rightarrow 'a\ set$
where $ray\text{-}Schutz\ a\ b \equiv insert\ a\ (segment\ a\ b \cup prolongation\ a\ b)$

lemma *ends-notin-segment*: $a \notin segment\ a\ b \wedge b \notin segment\ a\ b$
using *abc-abc-neq segment-def* **by** *fastforce*

lemma *ends-in-int*: $a \in interval\ a\ b \wedge b \in interval\ a\ b$
using *interval-def* **by** *auto*

lemma *seg-betw*: $x \in segment\ a\ b \longleftrightarrow [a;x;b]$

using *segment-def abc-abc-neq abc-ex-path* **by** *fastforce*

lemma *pro-betw*: $x \in$ *prolongation a b* $\longleftrightarrow$ $[a;b;x]$
using *prolongation-def abc-abc-neq abc-ex-path* **by** *fastforce*

lemma *seg-sym*: *segment a b* = *segment b a*
using *abc-sym segment-def* **by** *auto*

lemma *empty-segment*: *segment a a* = {}
by (*simp add*: *segment-def*)

lemma *int-sym*: *interval a b* = *interval b a*
by (*simp add*: *insert-commute interval-def seg-sym*)

lemma *seg-path*:
assumes $x \in$ *segment a b*
obtains *ab* **where** *path ab a b segment a b* $\subseteq$ *ab*
proof −
obtain *ab* **where** *path ab a b*
using *abc-abc-neq abc-ex-path assms seg-betw*
by *meson*
have *segment a b* $\subseteq$ *ab*
using ‹*path ab a b*› *abc-ex-path path-unique seg-betw*
by *fastforce*
thus *?thesis*
using ‹*path ab a b*› *that* **by** *blast*
qed

lemma *seg-path2*:
assumes *segment a b* $\neq$ {}
obtains *ab* **where** *path ab a b segment a b* $\subseteq$ *ab*
using *assms seg-path* **by** *force*

Path density (theorem 17) will extend this by weakening the assumptions to *segment a b* $\neq$ {}.

lemma *seg-endpoints-on-path*:
assumes *card* (*segment a b*) $\geq$ *2 segment a b* $\subseteq$ *P P*$\in\mathcal{P}$
shows *path P a b*
proof −
have *non-empty*: *segment a b* $\neq$ {} **using** *assms*(*1*) *numeral-2-eq-2* **by** *auto*
then obtain *ab* **where** *path ab a b segment a b* $\subseteq$ *ab*
using *seg-path2* **by** *force*
have $a \neq b$ **by** (*simp add*: ‹*path ab a b*›)
obtain *x y* **where** *x*$\in$*segment a b y*$\in$*segment a b x*$\neq$*y*
using *assms*(*1*) *numeral-2-eq-2*
by (*metis card.infinite card-le-Suc0-iff-eq not-less-eq-eq not-numeral-le-zero*)
have $[a;x;b]$
using ‹$x \in$ *segment a b*› *seg-betw* **by** *auto*
have $[a;y;b]$

using ‹*y* ∈ *segment a b*› *seg-betw* **by** *auto*
have *x*∈*P* ∧ *y*∈*P*
using ‹*x* ∈ *segment a b*› ‹*y* ∈ *segment a b*› *assms*(*2*) **by** *blast*
have *x*∈*ab* ∧ *y*∈*ab*
using ‹*segment a b* ⊆ *ab*› ‹*x* ∈ *segment a b*› ‹*y* ∈ *segment a b*› **by** *blast*
have *ab*=*P*
using ‹*path ab a b*› ‹*x* ∈ *P* ∧ *y* ∈ *P*› ‹*x* ∈ *ab* ∧ *y* ∈ *ab*› ‹*x* ≠ *y*› *assms*(*3*) *path-unique* **by** *auto*
thus *?thesis*
using ‹*path ab a b*› **by** *auto*
qed

lemma *pro-path*:
assumes *x* ∈ *prolongation a b*
obtains *ab* **where** *path ab a b prolongation a b* ⊆ *ab*
proof −
obtain *ab* **where** *path ab a b*
using *abc-abc-neq abc-ex-path assms pro-betw*
by *meson*
have *prolongation a b* ⊆ *ab*
using ‹*path ab a b*› *abc-ex-path path-unique pro-betw*
by *fastforce*
thus *?thesis*
using ‹*path ab a b*› *that* **by** *blast*
qed

lemma *ray-cases*:
assumes *x* ∈ *ray a b*
shows [*a*;*x*;*b*] ∨ [*a*;*b*;*x*] ∨ *x* = *b*
proof −
have *x*∈*segment a b* ∨ *x*∈ *prolongation a b* ∨ *x*=*b*
using *assms ray-def* **by** *auto*
thus [*a*;*x*;*b*] ∨ [*a*;*b*;*x*] ∨ *x* = *b*
using *pro-betw seg-betw* **by** *auto*
qed

lemma *ray-path*:
assumes *x* ∈ *ray a b x*≠*b*
obtains *ab* **where** *path ab a b* ∧ *ray a b* ⊆ *ab*
proof −
let *?r* = *ray a b*
have *?r* ≠ {*b*}
using *assms* **by** *blast*
have ∃ *ab*. *path ab a b* ∧ *ray a b* ⊆ *ab*
proof −
have *betw-cases*: [*a*;*x*;*b*] ∨ [*a*;*b*;*x*] **using** *ray-cases assms*
by *blast*
then obtain *ab* **where** *path ab a b*
using *abc-abc-neq abc-ex-path* **by** *blast*

have *?r* ⊆ *ab* **using** *betw-cases*
proof (*rule disjE*)
assume [*a*;*x*;*b*]
show *?r* ⊆ *ab*
proof
fix *x* **assume** *x*∈*?r*
show *x*∈*ab*
by (*metis* ‹*path ab a b*› ‹*x* ∈ *ray a b*› *abc-ex-path eq-paths ray-cases*)
qed
next assume [*a*;*b*;*x*]
show *?r* ⊆ *ab*
proof
fix *x* **assume** *x*∈*?r*
show *x*∈*ab*
by (*metis* ‹*path ab a b*› ‹*x* ∈ *ray a b*› *abc-ex-path eq-paths ray-cases*)
qed
qed
thus *?thesis*
using ‹*path ab a b*› **by** *blast*
qed
thus *?thesis*
using *that* **by** *blast*
qed

end

13 MinkowskiChain: O6

O6 supposedly serves the same purpose as Pasch's axiom.

locale *MinkowskiChain* = *MinkowskiBetweenness* +
assumes *O6*: ⟦{*Q*,*R*,*S*,*T*} ⊆ 𝒫; *card*{*Q*,*R*,*S*} = *3*; *a* ∈ *Q*∩*R*; *b* ∈ *Q*∩*S*; *c* ∈ *R*∩*S*; *d*∈*S*∩*T*; *e*∈*R*∩*T*; [*b*;*c*;*d*]; [*c*;*e*;*a*]⟧
⟹ ∃ *f*∈*T*∩*Q*. ∃ *g X*. [*g*⇝*X*|*a*..*f*..*b*]
begin

lemma *O6-old*: ⟦*Q* ∈ 𝒫; *R* ∈ 𝒫; *S* ∈ 𝒫; *T* ∈ 𝒫; *Q* ≠ *R*; *Q* ≠ *S*; *R* ≠ *S*; *a* ∈ *Q*∩*R* ∧ *b* ∈ *Q*∩*S* ∧ *c* ∈ *R*∩*S*;
∃ *d*∈*S*. [*b*;*c*;*d*] ∧ (∃ *e*∈*R*. *d* ∈ *T* ∧ *e* ∈ *T* ∧ [*c*;*e*;*a*])⟧
⟹ ∃ *f*∈*T*∩*Q*. ∃ *g X*. [*g*⇝*X*|*a*..*f*..*b*]
using *O6*[*of Q R S T a b c*] **by** (*metis IntI card-3-dist empty-subsetI insert-subset*)

14 Chains: (Closest) Bounds

definition *is-bound-f* :: '*a* ⇒ '*a set* ⇒ (*nat*⇒'*a*) ⇒ *bool* **where**
is-bound-f Q_b *Q f* ≡
∀ *i j* ::*nat*. [*f*⇝*Q*|(*f 0*)..] ∧ (*i*<*j* ⟶ [*f i*; *f j*; Q_b])

definition *is-bound* **where**
is-bound Q_b $Q \equiv$
$\exists f$::(*nat*$\Rightarrow$'*a*). *is-bound-f* Q_b Q f

Q_b has to be on the same path as the chain Q. This is left implicit in the betweenness condition (as is $Q_b \in \mathcal{E}$). So this is equivalent to Schutz only if we also assume his axioms, i.e. the statement of the continuity axiom is no longer independent of other axioms.

definition *all-bounds* **where**
all-bounds $Q = \{Q_b.$ *is-bound* Q_b $Q\}$

definition *bounded* **where**
bounded $Q \equiv \exists\ Q_b.$ *is-bound* Q_b Q

lemma *bounded-imp-inf*:
assumes *bounded* Q
shows *infinite* Q
using *assms bounded-def is-bound-def is-bound-f-def chain-defs* **by** *meson*

definition *closest-bound-f* **where**
closest-bound-f Q_b Q $f \equiv$
~~Q is an infinite chain indexed by f bound by Qb~~
is-bound-f Q_b Q f $\wedge$
~~Any other bound must be further from the start of the chain than the closest bound~~
$(\forall\ Q_b'.\ ($*is-bound* Q_b' $Q \wedge Q_b' \neq Q_b) \longrightarrow [f\ 0;\ Q_b;\ Q_b'])$

definition *closest-bound* **where**
closest-bound Q_b $Q \equiv$
$\exists f.$ *is-bound-f* Q_b Q f
$\wedge$ $(\forall\ Q_b'.\ ($*is-bound* Q_b' $Q \wedge Q_b' \neq Q_b) \longrightarrow [f\ 0;\ Q_b;\ Q_b'])$

lemma *closest-bound* Q_b $Q = (\exists f.$ *closest-bound-f* Q_b Q $f)$
unfolding *closest-bound-f-def closest-bound-def* **by** *simp*

end

15 MinkowskiUnreachable: I5-I7

locale *MinkowskiUnreachable* = *MinkowskiChain* +
assumes *I5*: $[\![Q \in \mathcal{P};\ b \in \mathcal{E}-Q]\!] \Longrightarrow \exists x\ y.\ \{x,y\} \subseteq$ *unreach−on* Q *from* $b \wedge x \neq y$
and *I6*: $[\![Q \in \mathcal{P};\ b \in \mathcal{E}-Q;\ \{Qx,Qz\} \subseteq$ *unreach−on* Q *from* $b;\ Qx \neq Qz]\!]$
$\Longrightarrow \exists X\ f.\ [f \leadsto X|Qx..Qz]$
$\wedge$ $(\forall i \in \{1\ ..\ card\ X\ -\ 1\}.\ (f\ i) \in$ *unreach−on* Q *from* b
$\wedge$ $(\forall\ Qy \in \mathcal{E}.\ [f(i-1);\ Qy;\ f\ i] \longrightarrow Qy \in$ *unreach−on* Q *from* $b))$

and *I7*: $[\![Q \in \mathcal{P};\ b \in \mathcal{E}-Q;\ Qx \in Q - unreach{-}on\ Q\ from\ b;\ Qy \in unreach{-}on\ Q\ from\ b]\!]$
$\Longrightarrow \exists\, g\ X\ Qn.\ [g \leadsto X|Qx..Qy..Qn] \wedge Qn \in Q - unreach{-}on\ Q\ from\ b$
begin

lemma *two-in-unreach*:
$[\![Q \in \mathcal{P};\ b \in \mathcal{E};\ b \notin Q]\!] \Longrightarrow \exists\, x{\in}unreach{-}on\ Q\ from\ b.\ \exists\, y{\in}unreach{-}on\ Q\ from\ b.\ x \neq y$
using *I5* **by** *fastforce*

lemma *I6-old*:
assumes $Q \in \mathcal{P}\ b \notin Q\ b \in \mathcal{E}\ Qx \in (unreach{-}on\ Q\ from\ b)\ Qz \in (unreach{-}on\ Q\ from\ b)\ Qx{\neq}Qz$
shows $\exists X.\ \exists f.\ ch\text{-}by\text{-}ord\ f\ X \wedge f\ 0 = Qx \wedge f\ (card\ X - 1) = Qz \wedge$
$(\forall\, i{\in}\{1..card\ X - 1\}.\ (f\ i) \in unreach{-}on\ Q\ from\ b \wedge (\forall\, Qy{\in}\mathcal{E}.\ [f(i{-}1);\ Qy;\ f\ i] \longrightarrow Qy \in unreach{-}on\ Q\ from\ b)) \wedge$
$(short\text{-}ch\ X \longrightarrow Qx{\in}X \wedge Qz{\in}X \wedge (\forall\, Qy{\in}\mathcal{E}.\ [Qx;Qy;Qz] \longrightarrow Qy \in unreach{-}on\ Q\ from\ b))$
proof −
from *assms I6*[*of Q b Qx Qz*] **obtain** *f X*
where *fX*: $[f \leadsto X|Qx..Qz]$
$(\forall\, i{\in}\{1\ ..\ card\ X - 1\}.\ (f\ i) \in unreach{-}on\ Q\ from\ b \wedge (\forall\, Qy{\in}\mathcal{E}.\ [f(i{-}1);\ Qy;\ f\ i] \longrightarrow Qy \in unreach{-}on\ Q\ from\ b))$
using *DiffI Un-Diff-cancel* **by** *blast*
show *?thesis*
proof ((*rule exI*)+, *intro conjI*, *rule-tac*[*4*] *ballI*, *rule-tac*[*5*] *impI*; (*intro conjI*)*?*)
show *1*: $[f \leadsto X]\ f\ 0 = Qx\ f\ (card\ X - 1) = Qz$
using *fX*(*1*) *chain-defs* **by** *meson*+
{
fix *i* **assume** *i-asm*: $i{\in}\{1..card\ X - 1\}$
show *2*: $f\ i \in unreach{-}on\ Q\ from\ b$
using *fX*(*2*) *i-asm* **by** *fastforce*
show *3*: $\forall\, Qy{\in}\mathcal{E}.\ [f\ (i - 1);Qy;f\ i] \longrightarrow Qy \in unreach{-}on\ Q\ from\ b$
using *fX*(*2*) *i-asm* **by** *blast*
} {
assume *X-asm*: *short-ch X*
show *4*: $Qx \in X\ Qz \in X$
using *fX*(*1*) *points-in-chain* **by** *auto*
have $\{1..card\ X{-}1\} = \{1\}$
using *X-asm short-ch-alt*(*2*) **by** *force*
thus *5*: $\forall\, Qy{\in}\mathcal{E}.\ [Qx;Qy;Qz] \longrightarrow Qy \in unreach{-}on\ Q\ from\ b$
using *fX*(*2*) *1*(*2*,*3*) **by** *auto*
}
qed
qed

lemma *I7-old*:
assumes $Q \in \mathcal{P}\ b \notin Q\ b \in \mathcal{E}\ Qx \in Q - unreach{-}on\ Q\ from\ b\ Qy \in unreach{-}on\ Q\ from\ b$

shows $\exists\, g\ X\ Qn.\ [g \rightsquigarrow X | Qx..Qy..Qn] \wedge Qn \in Q - $ *unreach−on Q from b*
using *I7 assms* **by** *auto*

lemma *card-unreach-geq-2*:
assumes $Q \in \mathcal{P}\ b \in \mathcal{E} - Q$
shows $2 \leq$ *card* (*unreach−on Q from b*) $\vee$ (*infinite* (*unreach−on Q from b*))
using *DiffD1 assms*(*1*) *assms*(*2*) *card-le-Suc0-iff-eq two-in-unreach* **by** *fastforce*

In order to more faithfully capture Schutz' definition of unreachable subsets via a path, we show that intersections of distinct paths are unique, and then define a new notation that doesn't carry the intersection of two paths around.

lemma *unreach-empty-on-same-path*:
assumes $P \in \mathcal{P}\ Q \in \mathcal{P}\ P = Q$
shows $\forall x.$ *unreach−via P on Q from a to x* $= \{\}$
unfolding *unreachable-subset-via-notation-def unreachable-subset-via-def unreachable-subset-def*
by (*simp add*: *assms*(*3*))

definition *unreachable-subset-via-notation-2* (‹*unreach−via - on - from -*› [*100*, *100*, *100*] *100*)
where *unreach−via P on Q from a* $\equiv$ *unreachable-subset-via Q a P* (*THE x.* $x \in Q \cap P$)

lemma *unreach-via-for-crossing-paths*:
assumes $P \in \mathcal{P}\ Q \in \mathcal{P}\ P \cap Q = \{x\}$
shows *unreach−via P on Q from a to x* $=$ *unreach−via P on Q from a*
unfolding *unreachable-subset-via-notation-2-def is-singleton-def unreachable-subset-via-notation-def*
using *the-equality assms* **by** (*metis Int-commute empty-iff insert-iff*)

end

16 MinkowskiSymmetry: Symmetry

locale *MinkowskiSymmetry* = *MinkowskiUnreachable* +
assumes *Symmetry*: $[\![\{Q,R,S\} \subseteq \mathcal{P};\ card\ \{Q,R,S\} = 3;$
$x \in Q \cap R \cap S;\ Q_a \in Q;\ Q_a \neq x;$
unreach−via R on Q from Q_a = *unreach−via S on Q from* $Q_a]\!]$
$\Longrightarrow \exists \vartheta{::}'a \Rightarrow 'a.$ ~~i) there is a map ϑ:E⇒E~~
bij-betw $(\lambda P.\ \{\vartheta\ y \mid y.\ y \in P\})\ \mathcal{P}\ \mathcal{P}$ ~~ii) which induces a bijection Θ~~
$\wedge\ (y \in Q \longrightarrow \vartheta\ y = y)$ ~~iii) ϑ leaves Q invariant~~
$\wedge\ (\lambda P.\ \{\vartheta\ y \mid y.\ y \in P\})\ R = S$ ~~iv) Θ maps R to S~~
begin

lemma *Symmetry-old*:
assumes $Q \in \mathcal{P}\ R \in \mathcal{P}\ S \in \mathcal{P}\ Q \neq R\ Q \neq S\ R \neq S$
and $x \in Q \cap R \cap S\ Q_a \in Q\ Q_a \neq x$

and *unreach−via R on Q from* Q_a *to x* = *unreach−via S on Q from* Q_a *to x*
shows $\exists\vartheta$::'a⇒'a. *bij-betw* (λP. {ϑ y | y. y∈P}) 𝒫 𝒫
∧ (y∈Q ⟶ ϑ y = y)
∧ (λP. {ϑ y | y. y∈P}) R = S
proof −
have *QS*: Q∩S = {x} **and** *QR*: Q∩R = {x}
using *assms*(1−7) *paths-cross-once* **by** (*metis Int-iff empty-iff insertE*)+
have *unreach−via R on Q from* Q_a = *unreach−via R on Q from* Q_a *to x*
using *unreach-via-for-crossing-paths QR* **by** (*simp add*: *Int-commute assms*(1,2))
moreover have *unreach−via S on Q from* Q_a = *unreach−via S on Q from* Q_a *to x*
using *unreach-via-for-crossing-paths QS* **by** (*simp add*: *Int-commute assms*(1,3))
ultimately show *?thesis*
using *Symmetry assms* **by** *simp*
qed

end

17 MinkowskiContinuity: Continuity

locale *MinkowskiContinuity* = *MinkowskiSymmetry* +
assumes *Continuity*: *bounded Q* ⟹ ∃ Q_b. *closest-bound* Q_b *Q*

18 MinkowskiSpacetime: Dimension (I4)

locale *MinkowskiSpacetime* = *MinkowskiContinuity* +

assumes *ex-3SPRAY* [*simp*]: ⟦ℰ ≠ {}⟧ ⟹ ∃x∈ℰ. 3−*SPRAY x*
begin

There exists an event by *nonempty-events*, and by *ex-3SPRAY* there is a three-SPRAY, which by *three-SPRAY-ge4* means that there are at least four paths.

lemma *four-paths*:
∃ Q1∈𝒫. ∃ Q2∈𝒫. ∃ Q3∈𝒫. ∃ Q4∈𝒫. Q1 ≠ Q2 ∧ Q1 ≠ Q3 ∧ Q1 ≠ Q4 ∧ Q2 ≠ Q3 ∧ Q2 ≠ Q4 ∧ Q3 ≠ Q4
using *nonempty-events ex-3SPRAY three-SPRAY-ge4* **by** *blast*

end

end

theory *TemporalOrderOnPath*
imports *Minkowski HOL−Library.Disjoint-Sets*
begin

In Schutz [1, pp. 18-30], this is "Chapter 3: Temporal order on a path". All theorems are from Schutz, all lemmas are either parts of the Schutz proofs extracted, or additional lemmas which needed to be added, with the exception of the three transitivity lemmas leading to Theorem 9, which are given by Schutz as well. Much of what we'd like to prove about chains with respect to injectivity, surjectivity, bijectivity, is proved in *TernaryOrdering.thy*. Some more things are proved in interlude sections.

19 Preliminary Results for Primitives

First some proofs that belong in this section but aren't proved in the book or are covered but in a different form or off-handed remark.

context *MinkowskiPrimitive* **begin**

lemma *cross-once-notin*:
  assumes $Q \in \mathcal{P}$
    and $R \in \mathcal{P}$
    and $a \in Q$
    and $b \in Q$
    and $b \in R$
    and $a \neq b$
    and $Q \neq R$
  shows $a \notin R$
using *assms paths-cross-once eq-paths* **by** *meson*

lemma *paths-cross-at*:
  assumes *path-Q*: $Q \in \mathcal{P}$ **and** *path-R*: $R \in \mathcal{P}$
    and *Q-neq-R*: $Q \neq R$
    and *QR-nonempty*: $Q \cap R \neq \{\}$
    and *x-inQ*: $x \in Q$ **and** *x-inR*: $x \in R$
  shows $Q \cap R = \{x\}$
proof (*rule equalityI*)
  show $Q \cap R \subseteq \{x\}$
  proof (*rule subsetI*, *rule ccontr*)
    fix *y*
    assume *y-in-QR*: $y \in Q \cap R$
      and *y-not-in-just-x*: $y \notin \{x\}$
    then have *y-neq-x*: $y \neq x$ **by** *simp*
    then have $\neg (\exists z.\ Q \cap R = \{z\})$
      by (*meson Q-neq-R path-Q path-R x-inQ x-inR y-in-QR cross-once-notin IntD1 IntD2*)

thus *False* **using** *paths-cross-once* **by** (*meson QR-nonempty Q-neq-R path-Q path-R*)
qed
show $\{x\} \subseteq Q \cap R$ **using** *x-inQ x-inR* **by** *simp*
qed

lemma *events-distinct-paths*:
assumes *a-event*: $a \in \mathcal{E}$
and *b-event*: $b \in \mathcal{E}$
and *a-neq-b*: $a \neq b$
shows $\exists R{\in}\mathcal{P}.\ \exists S{\in}\mathcal{P}.\ a \in R \wedge b \in S \wedge (R \neq S \longrightarrow (\exists!c{\in}\mathcal{E}.\ R \cap S = \{c\}))$
by (*metis events-paths assms paths-cross-once*)

end
context *MinkowskiBetweenness* **begin**

lemma assumes $[a;b;c]$ **shows** $\exists f.$ *local-long-ch-by-ord f* $\{a,b,c\}$
using *abc-abc-neq*[*OF assms*] **unfolding** *chain-defs*
by (*simp add*: *assms ord-ordered-loc*)

lemma *between-chain*: $[a;b;c] \Longrightarrow$ *ch* $\{a,b,c\}$
proof –
assume $[a;b;c]$
hence $\exists f.$ *local-ordering f betw* $\{a,b,c\}$
by (*simp add*: *abc-abc-neq ord-ordered-loc*)
hence $\exists f.$ *local-long-ch-by-ord f* $\{a,b,c\}$
using ‹$[a;b;c]$› *abc-abc-neq local-long-ch-by-ord-def* **by** *auto*
thus *?thesis*
by (*simp add*: *chain-defs*)
qed

end

20 3.1 Order on a finite chain

context *MinkowskiBetweenness* **begin**

20.1 Theorem 1

See *Minkowski.abc-only-cba*. Proving it again here to show it can be done following the prose in Schutz.

theorem *theorem1* [*no-atp*]:
assumes *abc*: $[a;b;c]$
shows $[c;b;a] \wedge \neg\ [b;c;a] \wedge \neg\ [c;a;b]$
proof –

have *part-i*: $[c;b;a]$ **using** *abc abc-sym* **by** *simp*

have *part-ii*: ¬ [*b*;*c*;*a*]
proof (*rule notI*)
assume [*b*;*c*;*a*]
then have [*a*;*b*;*a*] **using** *abc abc-bcd-abd* **by** *blast*
thus *False* **using** *abc-ac-neq* **by** *blast*
qed

have *part-iii*: ¬ [*c*;*a*;*b*]
proof (*rule notI*)
assume [*c*;*a*;*b*]
then have [*c*;*a*;*c*] **using** *abc abc-bcd-abd* **by** *blast*
thus *False* **using** *abc-ac-neq* **by** *blast*
qed
thus *?thesis* **using** *part-i part-ii part-iii* **by** *auto*
qed

20.2 Theorem 2

The lemma *abc-bcd-acd*, equal to the start of Schutz's proof, is given in *Minkowski* in order to prove some equivalences. We're splitting up Theorem 2 into two named results:

order-finite-chain there is a betweenness relation for each triple of adjacent events, and

index-injective all events of a chain are distinct.

We will be following Schutz' proof for both. Distinctness of chain events is interpreted as injectivity of the indexing function (see *index-injective*): we assume that this corresponds to what Schutz means by distinctness of elements in a sequence.

For the case of two-element chains: the elements are distinct by definition, and the statement on *local-ordering* is void (respectively, *False* $\Longrightarrow$ *P* for any *P*). We exclude this case from our proof of *order-finite-chain*. Two helper lemmas are provided, each capturing one of the proofs by induction in Schutz' writing.

lemma *thm2-ind1*:
assumes *chX*: *local-long-ch-by-ord f X*
and *finiteX*: *finite X*
shows $\forall j\ i.\ ((i{::}nat) < j \wedge j < card\ X - 1) \longrightarrow [f\ i;\ f\ j;\ f\ (j + 1)]$
proof (*rule allI*)+
let $?P = \lambda\ i\ j.\ [f\ i;\ f\ j;\ f\ (j{+}1)]$
fix *i j*
show $(i{<}j \wedge j{<}card\ X\ {-}1) \longrightarrow\ ?P\ i\ j$
proof (*induct j*)
case *0*
show *?case* **by** *blast*
next

```
    case (Suc j)
    show ?case
    proof (clarify)
      assume asm: i<Suc j Suc j<card X −1
      have pj: ?P j (Suc j)
        using asm(2) chX less-diff-conv local-long-ch-by-ord-def local-ordering-def
        by (metis Suc-eq-plus1)
      have i<j ∨ i=j using asm(1)
        by linarith
      thus ?P i (Suc j)
      proof
        assume i=j
        hence Suc i = Suc j ∧ Suc (Suc j) = Suc (Suc j)
          by simp
        thus ?P i (Suc j)
          using pj by auto
      next
        assume i<j
        have j < card X − 1
          using asm(2) by linarith
        thus ?P i (Suc j)
         using ‹i<j› Suc.hyps asm(1) asm(2) chX finiteX Suc-eq-plus1 abc-bcd-acd
pj
          by presburger
      qed
    qed
  qed
qed

lemma thm2-ind2:
  assumes chX: local-long-ch-by-ord f X
      and finiteX: finite X
    shows ∀ m l. (0<(l−m) ∧ (l−m) < l ∧ l < card X) ⟶ [f (l−m−1); f (l−m);
(f l)]
proof (rule allI)+
  fix l m
  let ?P = λ k l. [f (k−1); f k; f l]
  let ?n = card X
  let ?k = (l::nat)−m
  show 0 < ?k ∧ ?k < l ∧ l < ?n ⟶ ?P ?k l
  proof (induct m)
    case 0
    show ?case by simp
  next
    case (Suc m)
    show ?case
    proof (clarify)
      assume asm: 0 < l − Suc m l − Suc m < l l < ?n
      have Suc m = 1 ∨ Suc m > 1 by linarith
```

```
      thus [f (l − Suc m − 1); f (l − Suc m); f l] (is ?goal)
      proof
        assume Suc m = 1
        show ?goal
        proof −
          have l − Suc m < card X
            using asm(2) asm(3) less-trans by blast
          then show ?thesis
            using ‹Suc m = 1› asm finiteX thm2-ind1 chX
            using Suc-eq-plus1 add-diff-inverse-nat diff-Suc-less
                gr-implies-not-zero less-one plus-1-eq-Suc
            by (smt local-long-ch-by-ord-def ordering-ord-ijk-loc)
        qed
      next
        assume Suc m > 1
        show ?goal
          apply (rule-tac a=f l and c=f(l − Suc m − 1) in abc-sym)
           apply (rule-tac a=f l and c=f(l−Suc m) and d=f(l−Suc m−1) and
b=f(l−m) in abc-bcd-acd)
        proof −
          have [f(l−m−1); f(l−m); f l]
            using Suc.hyps ‹1 < Suc m› asm(1,3) by force
          thus [f l; f(l − m); f(l − Suc m)]
            using abc-sym One-nat-def diff-zero minus-nat.simps(2)
            by metis
          have Suc(l − Suc m − 1) = l − Suc m Suc(l − Suc m) = l−m
            using Suc-pred asm(1) by presburger+
          hence [f(l − Suc m − 1); f(l − Suc m); f(l − m)]
            using chX unfolding local-long-ch-by-ord-def local-ordering-def
            by (metis asm(2,3) less-trans-Suc)
          thus [f(l − m); f(l − Suc m); f(l − Suc m − 1)]
            using abc-sym by blast
        qed
      qed
    qed
  qed
qed

lemma thm2-ind2b:
  assumes chX: local-long-ch-by-ord f X
      and finiteX: finite X
      and ordered-nats: 0<k ∧ k<l ∧ l < card X
    shows [f (k−1); f k; f l]
  using thm2-ind2 finiteX chX ordered-nats
  by (metis diff-diff-cancel less-imp-le)
```

This is Theorem 2 properly speaking, except for the "chain elements are distinct" part (which is proved as injectivity of the index later). Follows Schutz fairly well! The statement Schutz proves under (i) is given in *Minkowski-*

Betweenness.abc-bcd-acd instead.

```
theorem order-finite-chain:
  assumes chX: local-long-ch-by-ord f X
      and finiteX: finite X
      and ordered-nats: 0 ≤ (i::nat) ∧ i < j ∧ j < l ∧ l < card X
    shows [f i; f j; f l]
proof −
  let ?n = card X − 1
  have ord1: 0≤i ∧ i<j ∧ j<?n
    using ordered-nats by linarith
  have e2: [f i; f j; f (j+1)] using thm2-ind1
    using Suc-eq-plus1 chX finiteX ord1
    by presburger
  have e3: ∀ k. 0<k ∧ k<l ⟶ [f (k−1); f k; f l]
    using thm2-ind2b chX finiteX ordered-nats
    by blast
  have j<l−1 ∨ j=l−1
    using ordered-nats by linarith
  thus ?thesis
  proof
    assume j<l−1
    have [f j; f (j+1); f l]
      using e3 abc-abc-neq ordered-nats
      using ‹j < l − 1› less-diff-conv by auto
    thus ?thesis
      using e2 abc-bcd-abd
      by blast
  next
    assume j=l−1
    thus ?thesis using e2
      using ordered-nats by auto
  qed
qed
```

```
corollary order-finite-chain2:
  assumes chX: [f↝X]
      and finiteX: finite X
      and ordered-nats: 0 ≤ (i::nat) ∧ i < j ∧ j < l ∧ l < card X
    shows [f i; f j; f l]
proof −
  have card X > 2 using ordered-nats by (simp add: eval-nat-numeral)
  thus ?thesis using order-finite-chain chain-defs short-ch-card(1) by (metis assms
nat-neq-iff)
qed
```

```
theorem index-injective:
  fixes i::nat and j::nat
```

```
  assumes chX: local-long-ch-by-ord f X
      and finiteX: finite X
      and indices: i<j j<card X
    shows f i ≠ f j
proof (cases)
  assume Suc i < j
  then have [f i; f (Suc(i)); f j]
    using order-finite-chain chX finiteX indices(2) by blast
  then show ?thesis
    using abc-abc-neq by blast
next
  assume ¬Suc i < j
  hence Suc i = j
    using Suc-lessI indices(1) by blast
  show ?thesis
  proof (cases)
    assume Suc j = card X
    then have 0<i
    proof −
      have card X ≥ 3
        using assms(1) finiteX long-chain-card-geq by blast
      thus ?thesis
        using ‹Suc i = j› ‹Suc j = card X› by linarith
    qed
    then have [f 0; f i; f j]
      using assms order-finite-chain by blast
    thus ?thesis
      using abc-abc-neq by blast
  next
    assume ¬Suc j = card X
    then have Suc j < card X
      using Suc-lessI indices(2) by blast
    then have [f i; f j; f(Suc j)]
      using chX finiteX indices(1) order-finite-chain by blast
    thus ?thesis
      using abc-abc-neq by blast
  qed
qed

theorem index-injective2:
  fixes i::nat and j::nat
  assumes chX: [f⇝X]
      and finiteX: finite X
      and indices: i<j j<card X
    shows f i ≠ f j
  using assms(1) unfolding ch-by-ord-def
proof (rule disjE)
  assume asm: short-ch-by-ord f X
  hence card X = 2 using short-ch-card(1) by simp
```

hence *j=1 i=0* **using** *indices plus-1-eq-Suc* **by** *auto*
thus *?thesis* **using** *asm* **unfolding** *chain-defs* **by** *force*
next
assume *local-long-ch-by-ord f X* **thus** *?thesis* **using** *index-injective assms* **by** *presburger*
qed

Surjectivity of the index function is easily derived from the definition of *local-ordering*, so we obtain bijectivity as an easy corollary to the second part of Theorem 2.

corollary *index-bij-betw*:
assumes *chX*: *local-long-ch-by-ord f X*
and *finiteX*: *finite X*
shows *bij-betw f* {*0..<card X*} *X*
proof (*unfold bij-betw-def*, (*rule conjI*))
show *inj-on f* {*0..<card X*}
using *index-injective*[*OF assms*] **by** (*metis* (*mono-tags*) *atLeastLessThan-iff inj-onI nat-neq-iff*)
{
fix *n* **assume** *n* ∈ {*0..<card X*}
then have *f n* ∈ *X*
using *assms* **unfolding** *chain-defs local-ordering-def* **by** *auto*
} moreover {
fix *x* **assume** *x* ∈ *X*
then have ∃ *n* ∈ {*0..<card X*}. *f n* = *x*
using *assms* **unfolding** *chain-defs local-ordering-def*
using *atLeastLessThan-iff bot-nat-0.extremum* **by** *blast*
} ultimately show *f* ‘ {*0..<card X*} = *X* **by** *blast*
qed

corollary *index-bij-betw2*:
assumes *chX*: [*f*⇝*X*]
and *finiteX*: *finite X*
shows *bij-betw f* {*0..<card X*} *X*
using *assms*(*1*) **unfolding** *ch-by-ord-def*
proof (*rule disjE*)
assume *local-long-ch-by-ord f X*
thus *bij-betw f* {*0..<card X*} *X* **using** *index-bij-betw assms* **by** *presburger*
next
assume *asm*: *short-ch-by-ord f X*
show *bij-betw f* {*0..<card X*} *X*
proof (*unfold bij-betw-def*, (*rule conjI*))
show *inj-on f* {*0..<card X*}
using *index-injective2*[*OF assms*] **by** (*metis* (*mono-tags*) *atLeastLessThan-iff inj-onI nat-neq-iff*)
{
fix *n* **assume** *asm2*: *n* ∈ {*0..<card X*}
have *f n* ∈ *X*
using *asm asm2 short-ch-card*(*1*) **apply** (*simp add*: *eval-nat-numeral*)

by (*metis One-nat-def less-Suc0 less-antisym short-ch-ord-in*)
} **moreover** **{**
fix *x* **assume** *asm2*: $x \in X$
have $\exists\, n \in \{0..<card\ X\}.\ f\ n = x$
using *short-ch-card*(*1*) *short-ch-by-ord-def asm asm2 atLeast0-lessThan-Suc*
by (*auto simp*: *eval-nat-numeral*)[*1*]
} **ultimately show** $f\ `\ \{0..<card\ X\} = X$ **by** *blast*
qed
qed

20.3 Additional lemmas about chains

lemma *first-neq-last*:
assumes [*f*⇝*Q*|*x*..*z*]
shows $x \neq z$
apply (*cases rule*: *finite-chain-with-cases*[*OF assms*])
using *chain-defs* **apply** (*metis Suc-1 card-2-iff diff-Suc-1*)
using *index-injective*[*of f Q 0 card Q − 1*]
by (*metis card.infinite diff-is-0-eq diff-less gr0I le-trans less-imp-le-nat less-numeral-extra*(*1*) *numeral-le-one-iff semiring-norm*(*70*))

lemma *index-middle-element*:
assumes [*f*⇝*X*|*a*..*b*..*c*]
shows $\exists n.\ 0<n \land n<(card\ X - 1) \land f\ n = b$
proof −
obtain *n* **where** *n-def*: $n < card\ X$ $f\ n = b$
using *local-ordering-def assms chain-defs* **by** (*metis two-ordered-loc*)
have $0<n \land n<(card\ X - 1) \land f\ n = b$
using *assms chain-defs n-def*
by (*metis* (*no-types*, *lifting*) *Suc-pred′ gr-implies-not0 less-SucE not-gr-zero*)
thus *?thesis* **by** *blast*
qed

Another corollary to Theorem 2, without mentioning indices.

corollary *fin-ch-betw*: [*f*⇝*X*|*a*..*b*..*c*] ⟹ [*a*;*b*;*c*]
using *order-finite-chain2 index-middle-element*
using *finite-chain-def finite-chain-with-def finite-long-chain-with-def*
by (*metis* (*no-types*, *lifting*) *card-gt-0-iff diff-less empty-iff le-eq-less-or-eq less-one*)

lemma *long-chain-2-imp-3*: ⟦[*f*⇝*X*|*a*..*c*]; $card\ X > 2$⟧ ⟹ ∃ *b*. [*f*⇝*X*|*a*..*b*..*c*]
using *points-in-chain first-neq-last finite-long-chain-with-def*
by (*metis card-2-iff′ numeral-less-iff semiring-norm*(*75*,*78*))

lemma *finite-chain2-betw*: ⟦[*f*⇝*X*|*a*..*c*]; $card\ X > 2$⟧ ⟹ ∃ *b*. [*a*;*b*;*c*]
using *fin-ch-betw long-chain-2-imp-3* **by** *metis*

lemma *finite-long-chain-with-alt* [*chain-alts*]: $[f \leadsto Q|x..y..z] \longleftrightarrow [f \leadsto Q|x..z] \wedge [x;y;z] \wedge y \in Q$
proof
{
assume $[f \leadsto Q|x\ ..\ z] \wedge [x;y;z] \wedge y \in Q$
thus $[f \leadsto Q|x..y..z]$
using *abc-abc-neq finite-long-chain-with-def* **by** *blast*
} {
assume *asm*: $[f \leadsto Q|x..y..z]$
show $[f \leadsto Q|x\ ..\ z] \wedge [x;y;z] \wedge y \in Q$
using *asm fin-ch-betw finite-long-chain-with-def* **by** *blast*
}
qed

lemma *finite-long-chain-with-card*: $[f \leadsto Q|x..y..z] \Longrightarrow$ *card* $Q \geq 3$
unfolding *chain-defs numeral-3-eq-3* **by** *fastforce*

lemma *finite-long-chain-with-alt2*:
assumes *finite Q local-long-ch-by-ord f Q* $f\ 0 = x$ f (*card* $Q - 1) = z$ $[x;y;z] \wedge y \in Q$
shows $[f \leadsto Q|x..y..z]$
using *assms finite-chain-alt finite-chain-with-def finite-long-chain-with-alt* **by** *blast*

lemma *finite-long-chain-with-alt3*:
assumes *finite Q local-long-ch-by-ord f Q* $f\ 0 = x$ f (*card* $Q - 1) = z$ $y \neq x \wedge y \neq z \wedge y \in Q$
shows $[f \leadsto Q|x..y..z]$
using *assms finite-chain-alt finite-chain-with-def finite-long-chain-with-def* **by** *auto*

lemma *chain-sym-obtain*:
assumes $[f \leadsto X|a..b..c]$
obtains g **where** $[g \leadsto X|c..b..a]$ **and** $g = (\lambda n.\ f$ (*card* $X - 1 - n))$
using *ordering-sym-loc*[*of betw X f*] *abc-sym assms* **unfolding** *chain-defs*
using *first-neq-last points-in-long-chain*(*3*)
by (*metis assms diff-self-eq-0 empty-iff finite-long-chain-with-def insert-iff minus-nat.diff-0*)

lemma *chain-sym*:
assumes $[f \leadsto X|a..b..c]$
shows $[\lambda n.\ f$ (*card* $X - 1 - n) \leadsto X|c..b..a]$
using *chain-sym-obtain* [**where** $f=f$ **and** $a=a$ **and** $b=b$ **and** $c=c$ **and** $X=X$]
using *assms*(*1*) **by** *blast*

lemma *chain-sym2*:
  assumes $[f \leadsto X|a..c]$
    shows $[\lambda n.\ f\ (card\ X\ -\ 1\ -\ n) \leadsto X|c..a]$
proof −
  {
    assume *asm*: $a = f\ 0$ $c = f\ (card\ X\ -\ 1)$
      and *asm-short*: *short-ch-by-ord f X*
    hence *cardX*: $card\ X = 2$
      using *short-ch-card*(*1*) **by** *auto*
    hence *ac*: $f\ 0 = a$ $f\ 1 = c$
      by (*simp add*: *asm*)+
    have $n=1 \lor n=0$ **if** $n<card\ X$ **for** n
      using *cardX that* **by** *linarith*
    hence *fn-eq*: $(\lambda n.\ if\ n = 0\ then\ f\ 1\ else\ f\ 0) = (\lambda n.\ f\ (card\ X\ -\ Suc\ n))$ **if** $n<card\ X$ **for** n
      by (*metis One-nat-def Zero-not-Suc ac*(*2*) *asm*(*2*) *not-gr-zero old.nat.inject zero-less-diff*)
    have $c = f\ (card\ X\ -\ 1\ -\ 0)$ **and** $a = f\ (card\ X\ -\ 1\ -\ (card\ X\ -\ 1))$
    and *short-ch-by-ord* $(\lambda n.\ f\ (card\ X\ -\ 1\ -\ n))\ X$
      apply (*simp add*: *asm*)+
      using *short-ch-sym*[*OF asm-short*] *fn-eq* ‹$f\ 1 = c$› *asm*(*2*) *short-ch-by-ord-def* **by** *fastforce*
  }
  consider *short-ch-by-ord f X*|$\exists\, b.\ [f \leadsto X|a..b..c]$
    using *assms long-chain-2-imp-3 finite-chain-with-alt* **by** *fastforce*
  thus *?thesis*
    apply *cases*
    using ‹$[\![a=f\ 0;\ c=f\ (card\ X-1);$ *short-ch-by-ord f X*$]\!] \Longrightarrow$ *short-ch-by-ord* $(\lambda n.\ f\ (card\ X\ -1-n))\ X$›
      assms finite-chain-alt finite-chain-with-def **apply** *auto*[*1*]
    using *chain-sym finite-long-chain-with-alt* **by** *blast*
qed

lemma *chain-sym-obtain2*:
  assumes $[f \leadsto X|a..c]$
  obtains g **where** $[g \leadsto X|c..a]$ **and** $g=(\lambda n.\ f\ (card\ X\ -\ 1\ -\ n))$
  using *assms chain-sym2* **by** *auto*

end

21 Preliminary Results for Kinematic Triangles and Paths/Betweenness

Theorem 3 (collinearity) First we prove some lemmas that will be very helpful.

context *MinkowskiPrimitive* **begin**

lemma *triangle-permutes* [*no-atp*]:
  assumes $\triangle$ *a b c*
  shows $\triangle$ *a c b* $\triangle$ *b a c* $\triangle$ *b c a* $\triangle$ *c a b* $\triangle$ *c b a*
  using *assms* **by** (*auto simp add*: *kinematic-triangle-def*)+

lemma *triangle-paths* [*no-atp*]:
  assumes *tri-abc*: $\triangle$ *a b c*
  shows *path-ex a b path-ex a c path-ex b c*
using *tri-abc* **by** (*auto simp add*: *kinematic-triangle-def*)+

lemma *triangle-paths-unique*:
  assumes *tri-abc*: $\triangle$ *a b c*
  shows $\exists!ab$. *path ab a b*
  using *path-unique tri-abc triangle-paths*(*1*) **by** *auto*

The definition of the kinematic triangle says that there exist paths that a and b pass through, and a and c pass through etc that are not equal. But we can show there is a *unique ab* that a and b pass through, and assuming there is a path *abc* that a, b, c pass through, it must be unique. Therefore $ab = abc$ and $ac = abc$, but $ab \neq ac$, therefore *False*. Lemma *tri-three-paths* is not in the books but might simplify some path obtaining.

lemma *triangle-diff-paths*:
  assumes *tri-abc*: $\triangle$ *a b c*
  shows $\neg$ ($\exists\, Q\in\mathcal{P}.\ a \in Q \wedge b \in Q \wedge c \in Q$)
proof (*rule notI*)
  assume *not-thesis*: $\exists\, Q\in\mathcal{P}.\ a \in Q \wedge b \in Q \wedge c \in Q$

  then obtain *abc* **where** *path-abc*: $abc \in \mathcal{P} \wedge a \in abc \wedge b \in abc \wedge c \in abc$ **by** *auto*
  have *abc-neq*: $a \neq b \wedge a \neq c \wedge b \neq c$ **using** *tri-abc kinematic-triangle-def* **by** *simp*

  have $\exists\, ab\in\mathcal{P}.\ \exists\, ac\in\mathcal{P}.\ ab \neq ac \wedge a \in ab \wedge b \in ab \wedge a \in ac \wedge c \in ac$
    using *tri-abc kinematic-triangle-def* **by** *metis*
  then obtain *ab ac* **where** *ab-ac-relate*: $ab \in \mathcal{P} \wedge ac \in \mathcal{P} \wedge ab \neq ac \wedge \{a,b\} \subseteq ab \wedge \{a,c\} \subseteq ac$
    by *blast*
  have $\exists!ab\in\mathcal{P}.\ a \in ab \wedge b \in ab$ **using** *tri-abc triangle-paths-unique* **by** *blast*
  then have *ab-eq-abc*: $ab = abc$ **using** *path-abc ab-ac-relate* **by** *auto*
  have $\exists!ac\in\mathcal{P}.\ a \in ac \wedge b \in ac$ **using** *tri-abc triangle-paths-unique* **by** *blast*
  then have *ac-eq-abc*: $ac = abc$ **using** *path-abc ab-ac-relate eq-paths abc-neq* **by** *auto*
  have $ab = ac$ **using** *ab-eq-abc ac-eq-abc* **by** *simp*
  thus *False* **using** *ab-ac-relate* **by** *simp*
qed

lemma *tri-three-paths* [*elim*]:

assumes *tri-abc*: $\triangle$ *a b c*
shows $\exists$ *ab bc ca. path ab a b* $\wedge$ *path bc b c* $\wedge$ *path ca c a* $\wedge$ *ab* $\neq$ *bc* $\wedge$ *ab* $\neq$ *ca* $\wedge$ *bc* $\neq$ *ca*
using *tri-abc triangle-diff-paths triangle-paths(2,3) triangle-paths-unique*
by *fastforce*

lemma *triangle-paths-neq*:
assumes *tri-abc*: $\triangle$ *a b c*
and *path-ab*: *path ab a b*
and *path-ac*: *path ac a c*
shows *ab* $\neq$ *ac*
using *assms triangle-diff-paths* **by** *blast*

end
context *MinkowskiBetweenness* **begin**

lemma *abc-ex-path-unique*:
assumes *abc*: [*a*;*b*;*c*]
shows $\exists!Q\in\mathcal{P}.\ a \in Q \wedge b \in Q \wedge c \in Q$
proof −
have *a-neq-c*: $a \neq c$ **using** *abc-ac-neq abc* **by** *simp*
have $\exists\, Q\in\mathcal{P}.\ a \in Q \wedge b \in Q \wedge c \in Q$ **using** *abc-ex-path abc* **by** *simp*
then obtain *P Q* **where** *path-P*: $P \in \mathcal{P}$ **and** *abc-inP*: $a \in P \wedge b \in P \wedge c \in P$
and *path-Q*: $Q \in \mathcal{P}$ **and** *abc-in-Q*: $a \in Q \wedge b \in Q \wedge c \in Q$ **by** *auto*
then have $P = Q$ **using** *a-neq-c eq-paths* **by** *blast*
thus *?thesis* **using** *eq-paths a-neq-c* **using** *abc-inP path-P* **by** *auto*
qed

lemma *betw-c-in-path*:
assumes *abc*: [*a*;*b*;*c*]
and *path-ab*: *path ab a b*
shows $c \in ab$

using *eq-paths abc-ex-path assms* **by** *blast*

lemma *betw-b-in-path*:
assumes *abc*: [*a*;*b*;*c*]
and *path-ab*: *path ac a c*
shows $b \in ac$
using *assms abc-ex-path-unique path-unique* **by** *blast*

lemma *betw-a-in-path*:
assumes *abc*: [*a*;*b*;*c*]
and *path-ab*: *path bc b c*
shows $a \in bc$
using *assms abc-ex-path-unique path-unique* **by** *blast*

lemma *triangle-not-betw-abc*:

assumes *tri-abc*: △ *a b c*
shows ¬ [*a*;*b*;*c*]
using *tri-abc abc-ex-path triangle-diff-paths* **by** *blast*

lemma *triangle-not-betw-acb*:
assumes *tri-abc*: △ *a b c*
shows ¬ [*a*;*c*;*b*]
by (*simp add*: *tri-abc triangle-not-betw-abc triangle-permutes*(*1*))

lemma *triangle-not-betw-bac*:
assumes *tri-abc*: △ *a b c*
shows ¬ [*b*;*a*;*c*]
by (*simp add*: *tri-abc triangle-not-betw-abc triangle-permutes*(*2*))

lemma *triangle-not-betw-any*:
assumes *tri-abc*: △ *a b c*
shows ¬ (∃ *d*∈{*a*,*b*,*c*}. ∃ *e*∈{*a*,*b*,*c*}. ∃*f*∈{*a*,*b*,*c*}. [*d*;*e*;*f*])
by (*metis abc-ex-path abc-abc-neq empty-iff insertE tri-abc triangle-diff-paths*)

end

22 3.2 First collinearity theorem

theorem (**in** *MinkowskiChain*) *collinearity-alt2*:
assumes *tri-abc*: △ *a b c*
and *path-de*: *path de d e*

and *path-ab*: *path ab a b*
and *bcd*: [*b*;*c*;*d*]
and *cea*: [*c*;*e*;*a*]
shows ∃*f*∈*de*∩*ab*. [*a*;*f*;*b*]
proof −
have ∃*f*∈*ab* ∩ *de*. ∃*X ord*. [*ord*⇝*X*|*a*..*f*..*b*]
proof −
have *path-ex a c* **using** *tri-abc triangle-paths*(*2*) **by** *auto*
then obtain *ac* **where** *path-ac*: *path ac a c* **by** *auto*
have *path-ex b c* **using** *tri-abc triangle-paths*(*3*) **by** *auto*
then obtain *bc* **where** *path-bc*: *path bc b c* **by** *auto*
have *ab-neq-ac*: *ab* ≠ *ac* **using** *triangle-paths-neq path-ab path-ac tri-abc* **by** *fastforce*
have *ab-neq-bc*: *ab* ≠ *bc* **using** *eq-paths ab-neq-ac path-ab path-ac path-bc* **by** *blast*
have *ac-neq-bc*: *ac* ≠ *bc* **using** *eq-paths ab-neq-bc path-ab path-ac path-bc* **by** *blast*
have *d-in-bc*: *d* ∈ *bc* **using** *bcd betw-c-in-path path-bc* **by** *blast*
have *e-in-ac*: *e* ∈ *ac* **using** *betw-b-in-path cea path-ac* **by** *blast*
show *?thesis*
using *O6-old* [**where** *Q* = *ab* **and** *R* = *ac* **and** *S* = *bc* **and** *T* = *de* **and** *a* = *a* **and** *b* = *b* **and** *c* = *c*]

ab-neq-ac ab-neq-bc ac-neq-bc path-ab path-bc path-ac path-de bcd cea d-in-bc e-in-ac
by *auto*
qed
thus *?thesis* **using** *fin-ch-betw* **by** *blast*
qed

theorem (**in** *MinkowskiChain*) *collinearity-alt*:
assumes *tri-abc*: $\triangle$ *a b c*
and *path-de*: *path de d e*
and *bcd*: [*b*;*c*;*d*]
and *cea*: [*c*;*e*;*a*]
shows $\exists$ *ab*. *path ab a b* $\wedge$ ($\exists$ *f*$\in$*de*$\cap$*ab*. [*a*;*f*;*b*])
proof −
have *ex-path-ab*: *path-ex a b*
using *tri-abc triangle-paths-unique* **by** *blast*
then obtain *ab* **where** *path-ab*: *path ab a b*
by *blast*
have $\exists$ *f*$\in$*ab* $\cap$ *de*. $\exists$ *X g*. [*g*⇝*X*|*a*..*f*..*b*]
proof −
have *path-ex a c* **using** *tri-abc triangle-paths*(*2*) **by** *auto*
then obtain *ac* **where** *path-ac*: *path ac a c* **by** *auto*
have *path-ex b c* **using** *tri-abc triangle-paths*(*3*) **by** *auto*
then obtain *bc* **where** *path-bc*: *path bc b c* **by** *auto*
have *ab-neq-ac*: *ab* $\neq$ *ac* **using** *triangle-paths-neq path-ab path-ac tri-abc* **by** *fastforce*
have *ab-neq-bc*: *ab* $\neq$ *bc* **using** *eq-paths ab-neq-ac path-ab path-ac path-bc* **by** *blast*
have *ac-neq-bc*: *ac* $\neq$ *bc* **using** *eq-paths ab-neq-bc path-ab path-ac path-bc* **by** *blast*
have *d-in-bc*: *d* $\in$ *bc* **using** *bcd betw-c-in-path path-bc* **by** *blast*
have *e-in-ac*: *e* $\in$ *ac* **using** *betw-b-in-path cea path-ac* **by** *blast*
show *?thesis*
using *O6-old* [**where** *Q* = *ab* **and** *R* = *ac* **and** *S* = *bc* **and** *T* = *de* **and** *a* = *a* **and** *b* = *b* **and** *c* = *c*]
ab-neq-ac ab-neq-bc ac-neq-bc path-ab path-bc path-ac path-de bcd cea d-in-bc e-in-ac
by *auto*
qed
thus *?thesis* **using** *fin-ch-betw path-ab* **by** *fastforce*
qed

theorem (**in** *MinkowskiChain*) *collinearity*:
assumes *tri-abc*: $\triangle$ *a b c*
and *path-de*: *path de d e*
and *bcd*: [*b*;*c*;*d*]
and *cea*: [*c*;*e*;*a*]

shows (∃ *f*∈*de*∩(*path-of a b*). [*a*;*f*;*b*])
proof −
let *?ab* = *path-of a b*
have *path-ab*: *path ?ab a b*
using *tri-abc theI′* [*OF triangle-paths-unique*] **by** *blast*
have ∃ *f*∈*?ab* ∩ *de*. ∃ *X ord*. [*ord*⇝*X*|*a*..*f*..*b*]
proof −
have *path-ex a c* **using** *tri-abc triangle-paths*(*2*) **by** *auto*
then obtain *ac* **where** *path-ac*: *path ac a c* **by** *auto*
have *path-ex b c* **using** *tri-abc triangle-paths*(*3*) **by** *auto*
then obtain *bc* **where** *path-bc*: *path bc b c* **by** *auto*
have *ab-neq-ac*: *?ab* ≠ *ac* **using** *triangle-paths-neq path-ab path-ac tri-abc* **by** *fastforce*
have *ab-neq-bc*: *?ab* ≠ *bc* **using** *eq-paths ab-neq-ac path-ab path-ac path-bc* **by** *blast*
have *ac-neq-bc*: *ac* ≠ *bc* **using** *eq-paths ab-neq-bc path-ab path-ac path-bc* **by** *blast*
have *d-in-bc*: *d* ∈ *bc* **using** *bcd betw-c-in-path path-bc* **by** *blast*
have *e-in-ac*: *e* ∈ *ac* **using** *betw-b-in-path cea path-ac* **by** *blast*
show *?thesis*
using *O6-old* [**where** *Q* = *?ab* **and** *R* = *ac* **and** *S* = *bc* **and** *T* = *de* **and** *a* = *a* **and** *b* = *b* **and** *c* = *c*]
ab-neq-ac ab-neq-bc ac-neq-bc path-ab path-bc path-ac path-de bcd cea d-in-bc e-in-ac
IntI Int-commute
by (*metis* (*no-types*, *lifting*))
qed
thus *?thesis* **using** *fin-ch-betw* **by** *blast*
qed

23 Additional results for Paths and Unreachables

context *MinkowskiPrimitive* **begin**

The degenerate case.

lemma *big-bang*:
assumes *no-paths*: $\mathcal{P}$ = {}
shows ∃ *a*. $\mathcal{E}$ = {*a*}
proof −
have ∃ *a*. *a* ∈ $\mathcal{E}$ **using** *nonempty-events* **by** *blast*
then obtain *a* **where** *a-event*: *a* ∈ $\mathcal{E}$ **by** *auto*
have ¬ (∃ *b*∈$\mathcal{E}$. *b* ≠ *a*)
proof (*rule notI*)
assume ∃ *b*∈$\mathcal{E}$. *b* ≠ *a*
then have ∃ *Q*. *Q* ∈ $\mathcal{P}$ **using** *events-paths a-event* **by** *auto*
thus *False* **using** *no-paths* **by** *simp*
qed
then have ∀ *b*∈$\mathcal{E}$. *b* = *a* **by** *simp*
thus *?thesis* **using** *a-event* **by** *auto*

qed

lemma *two-events-then-path*:
  assumes *two-events*: $\exists\, a{\in}\mathcal{E}.\ \exists\, b{\in}\mathcal{E}.\ a \neq b$
  shows $\exists\, Q.\ Q \in \mathcal{P}$
proof −
  have $(\forall\, a.\ \mathcal{E} \neq \{a\}) \longrightarrow \mathcal{P} \neq \{\}$ **using** *big-bang* **by** *blast*
  then have $\mathcal{P} \neq \{\}$ **using** *two-events* **by** *blast*
  thus *?thesis* **by** *blast*
qed

lemma *paths-are-events*: $\forall\, Q{\in}\mathcal{P}.\ \forall\, a{\in}Q.\ a \in \mathcal{E}$
  by *simp*

lemma *same-empty-unreach*:
  $[\![Q \in \mathcal{P};\ a \in Q]\!] \Longrightarrow$ *unreach−on Q from a* $= \{\}$
apply (*unfold unreachable-subset-def*)
by *simp*

lemma *same-path-reachable*:
  $[\![Q \in \mathcal{P};\ a \in Q;\ b \in Q]\!] \Longrightarrow a \in Q\ -$ *unreach−on Q from b*
by (*simp add*: *same-empty-unreach*)

If we have two paths crossing and a is on the crossing point, and b is on one of the paths, then a is in the reachable part of the path b is on.

lemma *same-path-reachable2*:
  $[\![Q \in \mathcal{P};\ R \in \mathcal{P};\ a \in Q;\ a \in R;\ b \in Q]\!] \Longrightarrow a \in R\ -$ *unreach−on R from b*
  unfolding *unreachable-subset-def* **by** *blast*

lemma *cross-in-reachable*:
  assumes *path-Q*: $Q \in \mathcal{P}$
    and *a-inQ*: $a \in Q$
    and *b-inQ*: $b \in Q$
    and *b-inR*: $b \in R$
  shows $b \in R\ -$ *unreach−on R from a*
unfolding *unreachable-subset-def* **using** *a-inQ b-inQ b-inR path-Q* **by** *auto*

lemma *reachable-path*:
  assumes *path-Q*: $Q \in \mathcal{P}$
    and *b-event*: $b \in \mathcal{E}$
    and *a-reachable*: $a \in Q\ -$ *unreach−on Q from b*
  shows $\exists\, R{\in}\mathcal{P}.\ a \in R \wedge b \in R$
proof −
  have *a-inQ*: $a \in Q$ **using** *a-reachable* **by** *simp*
  have $Q \notin \mathcal{P} \vee b \notin \mathcal{E} \vee b \in Q \vee (\exists\, R{\in}\mathcal{P}.\ b \in R \wedge a \in R)$
    using *a-reachable unreachable-subset-def* **by** *auto*
  then have $b \in Q \vee (\exists\, R{\in}\mathcal{P}.\ b \in R \wedge a \in R)$ **using** *path-Q b-event* **by** *simp*
  thus *?thesis*

proof (*rule disjE*)
assume $b \in Q$
thus *?thesis* **using** *a-inQ path-Q* **by** *auto*
next
assume $\exists R\in\mathcal{P}.\ b \in R \wedge a \in R$
thus *?thesis* **using** *conj-commute* **by** *simp*
qed
qed

end
context *MinkowskiBetweenness* **begin**

lemma *ord-path-of*:
assumes $[a;b;c]$
shows $a \in$ *path-of* b c $b \in$ *path-of* a c $c \in$ *path-of* a b
and *path-of* a b = *path-of* a c *path-of* a b = *path-of* b c
proof −
show $a \in$ *path-of* b c
using *betw-a-in-path*[*of a b c path-of b c*] *path-of-ex abc-ex-path-unique abc-abc-neq assms*
by (*smt* (*z3*) *betw-a-in-path the1-equality*)
show $b \in$ *path-of* a c
using *betw-b-in-path*[*of a b c path-of a c*] *path-of-ex abc-ex-path-unique abc-abc-neq assms*
by (*smt* (*z3*) *betw-b-in-path the1-equality*)
show $c \in$ *path-of* a b
using *betw-c-in-path*[*of a b c path-of a b*] *path-of-ex abc-ex-path-unique abc-abc-neq assms*
by (*smt* (*z3*) *betw-c-in-path the1-equality*)
show *path-of* a b = *path-of* a c
by (*metis* (*mono-tags*) *abc-ac-neq assms betw-b-in-path betw-c-in-path ends-notin-segment seg-betw*)
show *path-of* a b = *path-of* b c
by (*metis* (*mono-tags*) *assms betw-a-in-path betw-c-in-path ends-notin-segment seg-betw*)
qed

Schutz defines chains as subsets of paths. The result below proves that even though we do not include this fact in our definition, it still holds, at least for finite chains.

Notice that this whole proof would be unnecessary if including path-belongingness in the definition, as Schutz does. This would also keep path-belongingness independent of axiom O1 and O4, thus enabling an independent statement of axiom O6, which perhaps we now lose. In exchange, our definition is slightly weaker (for *card* $X \geq 3$ and *infinite* X).

lemma *obtain-index-fin-chain*:
assumes $[f \leadsto X]$ $x\in X$ *finite* X

obtains *i* **where** *f i* = *x i*<*card X*
proof –
  have ∃ *i*<*card X*. *f i* = *x*
    using *assms*(*1*) **unfolding** *ch-by-ord-def*
  proof (*rule disjE*)
    assume *asm*: *short-ch-by-ord f X*
    hence *card X* = *2*
      using *short-ch-card*(*1*) **by** *auto*
    thus ∃ *i*<*card X*. *f i* = *x*
      using *asm assms*(*2*) **unfolding** *chain-defs* **by** *force*
  next
    assume *asm*: *local-long-ch-by-ord f X*
    thus ∃ *i*<*card X*. *f i* = *x*
      using *asm assms*(*2*,*3*) **unfolding** *chain-defs local-ordering-def* **by** *blast*
  qed
  thus *?thesis* **using** *that* **by** *blast*
qed

lemma *obtain-index-inf-chain*:
  assumes [*f*⇝*X*] *x*∈*X infinite X*
  obtains *i* **where** *f i* = *x*
  using *assms* **unfolding** *chain-defs local-ordering-def* **by** *blast*

lemma *fin-chain-on-path2*:
  assumes [*f*⇝*X*] *finite X*
  shows ∃ *P*∈𝒫. *X*⊆*P*
  using *assms*(*1*) **unfolding** *ch-by-ord-def*
proof (*rule disjE*)
  assume *short-ch-by-ord f X*
  thus *?thesis*
    using *short-ch-by-ord-def* **by** *auto*
next
  assume *asm*: *local-long-ch-by-ord f X*
  have [*f 0*;*f 1*;*f 2*]
    using *order-finite-chain asm assms*(*2*) *local-long-ch-by-ord-def* **by** *auto*
  then obtain *P* **where** *P*∈𝒫 {*f 0*,*f 1*,*f 2*} ⊆ *P*
    by (*meson abc-ex-path empty-subsetI insert-subset*)
  then have *path P* (*f 0*) (*f 1*)
    using ‹[*f 0*;*f 1*;*f 2*]› **by** (*simp add*: *abc-abc-neq*)
  {
    fix *x* **assume** *x*∈*X*
    then obtain *i* **where** *i*: *f i* = *x i*<*card X*
      using *obtain-index-fin-chain assms* **by** *blast*
    consider *i*=*0*∨*i*=*1*|*i*>*1* **by** *linarith*
    hence *x*∈*P*
    proof (*cases*)
      case *1* **thus** *?thesis*
      using *i*(*1*) ‹{*f 0*, *f 1*, *f 2*} ⊆ *P*› **by** *auto*

```
    next
      case 2
      hence [f 0;f 1;f i]
        using assms i(2) order-finite-chain2 by auto
      hence {f 0,f 1,f i}⊆P
        using ‹path P (f 0) (f 1)› betw-c-in-path by blast
      thus ?thesis by (simp add: i(1))
    qed
  }
  thus ?thesis
    using ‹P∈𝒫› by auto
qed


lemma fin-chain-on-path:
  assumes [f↝X] finite X
  shows ∃!P∈𝒫. X⊆P
proof −
  obtain P where P: P∈𝒫 X⊆P
    using fin-chain-on-path2[OF assms] by auto
  obtain a b where ab: a∈X b∈X a≠b
   using assms(1) unfolding chain-defs by (metis assms(2) insertCI three-in-set3)
  thus ?thesis using P ab by (meson eq-paths in-mono)
qed


lemma fin-chain-on-path3:
  assumes [f↝X] finite X a∈X b∈X a≠b
  shows X ⊆ path-of a b
proof −
  let ?ab = path-of a b
   obtain P where P: P∈𝒫 X⊆P using fin-chain-on-path2[OF assms(1,2)] by
auto
  have path P a b using P assms(3−5) by auto
  then have path ?ab a b using path-of-ex by blast
  hence ?ab = P using eq-paths ‹path P a b› by auto
  thus X ⊆ path-of a b using P by simp
qed


end
context MinkowskiUnreachable begin
```

First some basic facts about the primitive notions, which seem to belong here. I don't think any/all of these are explicitly proved in Schutz.

```
lemma no-empty-paths [simp]:
  assumes Q∈𝒫
  shows Q≠{}
```

proof −
  obtain *a* **where** $a \in \mathcal{E}$ **using** *nonempty-events* **by** *blast*
  have $a \in Q \vee a \notin Q$ **by** *auto*
  thus *?thesis*
  proof
    assume $a \in Q$
    thus *?thesis* **by** *blast*
  next
    assume $a \notin Q$
    then obtain *b* **where** $b \in unreach\text{−}on\ Q\ from\ a$
      using *two-in-unreach* ‹$a \in \mathcal{E}$› *assms*
      by *blast*
    thus *?thesis*
      using *unreachable-subset-def* **by** *auto*
  qed
qed

lemma *events-ex-path*:
  assumes *ge1-path*: $\mathcal{P} \neq \{\}$
  shows $\forall x \in \mathcal{E}.\ \exists Q \in \mathcal{P}.\ x \in Q$

proof
  fix *x*
  assume *x-event*: $x \in \mathcal{E}$
  have $\exists Q.\ Q \in \mathcal{P}$ **using** *ge1-path* **using** *ex-in-conv* **by** *blast*
  then obtain *Q* **where** *path-Q*: $Q \in \mathcal{P}$ **by** *auto*
  then have $\exists y.\ y \in Q$ **using** *no-empty-paths* **by** *blast*
  then obtain *y* **where** *y-inQ*: $y \in Q$ **by** *auto*
  then have *y-event*: $y \in \mathcal{E}$ **using** *in-path-event path-Q* **by** *simp*
  have $\exists P \in \mathcal{P}.\ x \in P$
  proof *cases*
    assume $x = y$
    thus *?thesis* **using** *y-inQ path-Q* **by** *auto*
  next
    assume $x \neq y$
    thus *?thesis* **using** *events-paths x-event y-event* **by** *auto*
  qed
  thus $\exists Q \in \mathcal{P}.\ x \in Q$ **by** *simp*
qed

lemma *unreach-ge2-then-ge2*:
  assumes $\exists x \in unreach\text{−}on\ Q\ from\ b.\ \exists y \in unreach\text{−}on\ Q\ from\ b.\ x \neq y$
  shows $\exists x \in Q.\ \exists y \in Q.\ x \neq y$
using *assms unreachable-subset-def* **by** *auto*

This lemma just proves that the chain obtained to bound the unreachable set of a path is indeed on that path. Extends I6; requires Theorem 2; used in Theorem 13. Seems to be assumed in Schutz' chain notation in I6.

lemma *chain-on-path-I6*:

assumes *path-Q*: $Q{\in}\mathcal{P}$
and *event-b*: $b{\notin}Q\ b{\in}\mathcal{E}$
and *unreach*: $Q_x \in$ *unreach−on Q from b* $Q_z \in$ *unreach−on Q from b* $Q_x \neq Q_z$
and *X-def*: $[f{\leadsto}X|Q_x..Q_z]$
$(\forall\, i{\in}\{1\ ..\ card\ X\ -\ 1\}.\ (f\ i) \in$ *unreach−on Q from b* $\wedge$ $(\forall\, Q_y{\in}\mathcal{E}.\ [(f(i{-}1));\ Q_y;\ f\ i] \longrightarrow Q_y \in$ *unreach−on Q from b*$))$
shows $X{\subseteq}Q$
proof −
have *1*: *path Q* $Q_x\ Q_z$ **using** *unreachable-subset-def unreach path-Q* **by** *simp*
then have *2*: $Q =$ *path-of* $Q_x\ Q_z$ **using** *path-of-ex*[*of* $Q_x\ Q_z$] **by** (*meson eq-paths*)
have $X{\subseteq}$*path-of* $Q_x\ Q_z$
proof (*rule fin-chain-on-path3*[*of f*])
from *unreach*(*3*) **show** $Q_x \neq Q_z$ **by** *simp*
from *X-def chain-defs* **show** $[f{\leadsto}X]$ *finite X* **by** *metis*+
from *assms*(*7*) *points-in-chain* **show** $Q_x \in X\ Q_z \in X$ **by** *auto*
qed
thus *?thesis* **using** *2* **by** *simp*
qed

end

24 Results about Paths as Sets

Note several of the following don't need MinkowskiPrimitive, they are just Set lemmas; nevertheless I'm naming them and writing them this way for clarity.

context *MinkowskiPrimitive* **begin**

lemma *distinct-paths*:
assumes $Q \in \mathcal{P}$
and $R \in \mathcal{P}$
and $d \notin Q$
and $d \in R$
shows $R \neq Q$
using *assms* **by** *auto*

lemma *distinct-paths2*:
assumes $Q \in \mathcal{P}$
and $R \in \mathcal{P}$
and $\exists\, d.\ d \notin Q \wedge d \in R$
shows $R \neq Q$
using *assms* **by** *auto*

lemma *external-events-neq*:
$[\![Q \in \mathcal{P};\ a \in Q;\ b \in \mathcal{E};\ b \notin Q]\!] \Longrightarrow a \neq b$
by *auto*

lemma *notin-cross-events-neq*:
$[\![Q \in \mathcal{P};\ R \in \mathcal{P};\ Q \neq R;\ a \in Q;\ b \in R;\ a \notin R \cap Q]\!] \Longrightarrow a \neq b$
by *blast*

lemma *nocross-events-neq*:
$[\![Q \in \mathcal{P};\ R \in \mathcal{P};\ a \in Q;\ b \in R;\ R \cap Q = \{\}]\!] \Longrightarrow a \neq b$
by *auto*

Given a nonempty path Q, and an external point d, we can find another path R passing through d (by I2 aka *events-paths*). This path is distinct from Q, as it passes through a point external to it.

lemma *external-path*:
assumes *path-Q*: $Q \in \mathcal{P}$
and *a-inQ*: $a \in Q$
and *d-notinQ*: $d \notin Q$
and *d-event*: $d \in \mathcal{E}$
shows $\exists R \in \mathcal{P}.\ d \in R$
proof −
have *a-neq-d*: $a \neq d$ **using** *a-inQ d-notinQ* **by** *auto*
thus $\exists R \in \mathcal{P}.\ d \in R$ **using** *events-paths* **by** (*meson a-inQ d-event in-path-event path-Q*)
qed

lemma *distinct-path*:
assumes $Q \in \mathcal{P}$
and $a \in Q$
and $d \notin Q$
and $d \in \mathcal{E}$
shows $\exists R \in \mathcal{P}.\ R \neq Q$
using *assms external-path* **by** *metis*

lemma *external-distinct-path*:
assumes $Q \in \mathcal{P}$
and $a \in Q$
and $d \notin Q$
and $d \in \mathcal{E}$
shows $\exists R \in \mathcal{P}.\ R \neq Q \wedge d \in R$
using *assms external-path* **by** *fastforce*

end

25 3.3 Boundedness of the unreachable set

25.1 Theorem 4 (boundedness of the unreachable set)

The same assumptions as I7, different conclusion. This doesn't just give us boundedness, it gives us another event outside of the unreachable set, as long as we have one already. I7 conclusion: $\exists\, g\ X\ Qn.\ [g{\leadsto}X|Qx..Qy..Qn] \wedge$

Qn ∈ *Q* − *unreach*−*on Q from b*

theorem (**in** *MinkowskiUnreachable*) *unreachable-set-bounded*:
assumes *path-Q*: *Q* ∈ 𝒫
and *b-nin-Q*: *b* ∉ *Q*
and *b-event*: *b* ∈ ℰ
and *Qx-reachable*: *Qx* ∈ *Q* − *unreach*−*on Q from b*
and *Qy-unreachable*: *Qy* ∈ *unreach*−*on Q from b*
shows ∃ *Qz*∈*Q* − *unreach*−*on Q from b*. [*Qx*;*Qy*;*Qz*] ∧ *Qx* ≠ *Qz*
using *assms I7-old finite-long-chain-with-def fin-ch-betw*
by (*metis first-neq-last*)

25.2 Theorem 5 (first existence theorem)

The lemma below is used in the contradiction in *external-event*, which is the essential part to Theorem 5(i).

lemma (**in** *MinkowskiUnreachable*) *only-one-path*:
assumes *path-Q*: *Q* ∈ 𝒫
and *all-inQ*: ∀ *a*∈ℰ. *a* ∈ *Q*
and *path-R*: *R* ∈ 𝒫
shows *R* = *Q*
proof (*rule ccontr*)
assume ¬ *R* = *Q*
then have *R-neq-Q*: *R* ≠ *Q* **by** *simp*
have ℰ = *Q*
by (*simp add*: *all-inQ antisym path-Q path-sub-events subsetI*)
hence *R*⊂*Q*
using *R-neq-Q path-R path-sub-events* **by** *auto*
obtain *c* **where** *c*∉*R* *c*∈*Q*
using ‹*R* ⊂ *Q*› **by** *blast*
then obtain *a b* **where** *path R a b*
using ‹ℰ = *Q*› *path-R two-in-unreach unreach-ge2-then-ge2* **by** *blast*
have *a*∈*Q* *b*∈*Q*
using ‹ℰ = *Q*› ‹*path R a b*› *in-path-event* **by** *blast*+
thus *False* **using** *eq-paths*
using *R-neq-Q* ‹*path R a b*› *path-Q* **by** *blast*
qed

context *MinkowskiSpacetime* **begin**

Unfortunately, we cannot assume that a path exists without the axiom of dimension.

lemma *external-event*:
assumes *path-Q*: *Q* ∈ 𝒫
shows ∃ *d*∈ℰ. *d* ∉ *Q*
proof (*rule ccontr*)
assume ¬ (∃ *d*∈ℰ. *d* ∉ *Q*)
then have *all-inQ*: ∀ *d*∈ℰ. *d* ∈ *Q* **by** *simp*
then have *only-one-path*: ∀ *P*∈𝒫. *P* = *Q* **by** (*simp add*: *only-one-path path-Q*)

thus *False* **using** *ex-3SPRAY three-SPRAY-ge4 four-paths* **by** *auto*
qed

Now we can prove the first part of the theorem's conjunction. This follows pretty much exactly the same pattern as the book, except it relies on more intermediate lemmas.

theorem *ge2-events*:
assumes *path-Q*: $Q \in \mathcal{P}$
and *a-inQ*: $a \in Q$
shows $\exists\, b{\in}Q.\ b \neq a$
proof −
have *d-notinQ*: $\exists\, d{\in}\mathcal{E}.\ d \notin Q$ **using** *path-Q external-event* **by** *blast*
then obtain d **where** $d \in \mathcal{E}$ **and** $d \notin Q$ **by** *auto*
thus *?thesis* **using** *two-in-unreach* [**where** $Q = Q$ **and** $b = d$] *path-Q unreach-ge2-then-ge2* **by** *metis*
qed

Simple corollary which is easier to use when we don't have one event on a path yet. Anything which uses this implicitly used *no-empty-paths* on top of *ge2-events*.

lemma *ge2-events-lax*:
assumes *path-Q*: $Q \in \mathcal{P}$
shows $\exists\, a{\in}Q.\ \exists\, b{\in}Q.\ a \neq b$
proof −
have $\exists\, a{\in}\mathcal{E}.\ a \in Q$ **using** *path-Q no-empty-paths* **by** (*meson ex-in-conv in-path-event*)
thus *?thesis* **using** *path-Q ge2-events* **by** *blast*
qed

lemma *ex-crossing-path*:
assumes *path-Q*: $Q \in \mathcal{P}$
shows $\exists\, R{\in}\mathcal{P}.\ R \neq Q \wedge (\exists\, c.\ c \in R \wedge c \in Q)$
proof −
obtain a **where** *a-inQ*: $a \in Q$ **using** *ge2-events-lax path-Q* **by** *blast*
obtain d **where** *d-event*: $d \in \mathcal{E}$
and *d-notinQ*: $d \notin Q$ **using** *external-event path-Q* **by** *auto*
then have $a \neq d$ **using** *a-inQ* **by** *auto*
then have *ex-through-d*: $\exists\, R{\in}\mathcal{P}.\ \exists\, S{\in}\mathcal{P}.\ a \in R \wedge d \in S \wedge R \cap S \neq \{\}$
using *events-paths* [**where** $a = a$ **and** $b = d$]
path-Q a-inQ in-path-event d-event **by** *simp*
then obtain R S **where** *path-R*: $R \in \mathcal{P}$
and *path-S*: $S \in \mathcal{P}$
and *a-inR*: $a \in R$
and *d-inS*: $d \in S$
and *R-crosses-S*: $R \cap S \neq \{\}$ **by** *auto*
have *S-neq-Q*: $S \neq Q$ **using** *d-notinQ d-inS* **by** *auto*
show *?thesis*
proof *cases*
assume $R = Q$
then have $Q \cap S \neq \{\}$ **using** *R-crosses-S* **by** *simp*

thus *?thesis* **using** *S-neq-Q path-S* **by** *blast*
next
assume $R \neq Q$
thus *?thesis* **using** *a-inQ a-inR path-R* **by** *blast*
qed
qed

If we have two paths Q and R with a on Q and b at the intersection of Q and R, then by *two-in-unreach* (I5) and Theorem 4 (boundedness of the unreachable set), there is an unreachable set from a on one side of b on R, and on the other side of that there is an event which is reachable from a by some path, which is the path we want.

lemma *path-past-unreach*:
assumes *path-Q*: $Q \in \mathcal{P}$
and *path-R*: $R \in \mathcal{P}$
and *a-inQ*: $a \in Q$
and *b-inQ*: $b \in Q$
and *b-inR*: $b \in R$
and *Q-neq-R*: $Q \neq R$
and *a-neq-b*: $a \neq b$
shows $\exists S{\in}\mathcal{P}.\ S \neq Q \wedge a \in S \wedge (\exists c.\ c \in S \wedge c \in R)$
proof −
obtain d **where** *d-event*: $d \in \mathcal{E}$
and *d-notinR*: $d \notin R$ **using** *external-event path-R* **by** *blast*
have *b-reachable*: $b \in R - unreach{-}on\ R\ from\ a$ **using** *cross-in-reachable path-R a-inQ b-inQ b-inR path-Q* **by** *simp*
have *a-notinR*: $a \notin R$ **using** *cross-once-notin*
Q-neq-R a-inQ a-neq-b b-inQ b-inR path-Q path-R **by** *blast*
then obtain u **where** $u \in unreach{-}on\ R\ from\ a$
using *two-in-unreach a-inQ in-path-event path-Q path-R* **by** *blast*
then obtain c **where** *c-reachable*: $c \in R - unreach{-}on\ R\ from\ a$
and *c-neq-b*: $b \neq c$ **using** *unreachable-set-bounded*
[**where** $Q = R$ **and** $Qx = b$ **and** $b = a$ **and** $Qy = u$]
path-R d-event d-notinR
using *a-inQ a-notinR b-reachable in-path-event path-Q* **by** *blast*
then obtain S **where** *S-facts*: $S \in \mathcal{P} \wedge a \in S \wedge (c \in S \wedge c \in R)$ **using** *reachable-path*
by (*metis Diff-iff a-inQ in-path-event path-Q path-R*)
then have $S \neq Q$ **using** *Q-neq-R b-inQ b-inR c-neq-b eq-paths path-R* **by** *blast*
thus *?thesis* **using** *S-facts* **by** *auto*
qed

theorem *ex-crossing-at*:
assumes *path-Q*: $Q \in \mathcal{P}$
and *a-inQ*: $a \in Q$
shows $\exists ac{\in}\mathcal{P}.\ ac \neq Q \wedge (\exists c.\ c \notin Q \wedge a \in ac \wedge c \in ac)$
proof −
obtain b **where** *b-inQ*: $b \in Q$

```
      and a-neq-b: a ≠ b using a-inQ ge2-events path-Q by blast
  have ∃R∈𝒫. R ≠ Q ∧ (∃e. e ∈ R ∧ e ∈ Q) by (simp add: ex-crossing-path
path-Q)
  then obtain R e where path-R: R ∈ 𝒫
                  and R-neq-Q: R ≠ Q
                  and e-inR: e ∈ R
                  and e-inQ: e ∈ Q by auto
  thus ?thesis
  proof cases
    assume e-eq-a: e = a
    then have ∃c. c ∈ unreach−on R from b using R-neq-Q a-inQ a-neq-b b-inQ
e-inR path-Q path-R
                        two-in-unreach path-unique in-path-event by metis
    thus ?thesis using R-neq-Q e-eq-a e-inR path-Q path-R
                 eq-paths ge2-events-lax by metis
  next
    assume e-neq-a: e ≠ a


    then have ∃S∈𝒫. S ≠ Q ∧ a ∈ S ∧ (∃c. c ∈ S ∧ c ∈ R)
        using path-past-unreach
              R-neq-Q a-inQ e-inQ e-inR path-Q path-R by auto
    thus ?thesis by (metis R-neq-Q e-inR e-neq-a eq-paths path-Q path-R)
  qed
qed


lemma ex-crossing-at-alt:
  assumes path-Q: Q ∈ 𝒫
      and a-inQ: a ∈ Q
    shows ∃ac. ∃c. path ac a c ∧ ac ≠ Q ∧ c ∉ Q
  using ex-crossing-at assms by fastforce

end
```

26 3.4 Prolongation

```
context MinkowskiSpacetime begin

lemma (in MinkowskiPrimitive) unreach-on-path:
  a ∈ unreach−on Q from b ⟹ a ∈ Q
using unreachable-subset-def by simp

lemma (in MinkowskiUnreachable) unreach-equiv:
  ⟦Q ∈ 𝒫; R ∈ 𝒫; a ∈ Q; b ∈ R; a ∈ unreach−on Q from b⟧ ⟹ b ∈ unreach−on
R from a
  unfolding unreachable-subset-def by auto

theorem prolong-betw:
```

assumes *path-Q*: $Q \in \mathcal{P}$
and *a-inQ*: $a \in Q$
and *b-inQ*: $b \in Q$
and *ab-neq*: $a \neq b$
shows $\exists\, c{\in}\mathcal{E}.\ [a;b;c]$
proof –
obtain *e ae* **where** *e-event*: $e \in \mathcal{E}$
and *e-notinQ*: $e \notin Q$
and *path-ae*: *path ae a e*
using *ex-crossing-at a-inQ path-Q in-path-event* **by** *blast*
have $b \notin ae$ **using** *a-inQ ab-neq b-inQ e-notinQ eq-paths path-Q path-ae* **by** *blast*
then obtain *f* **where** *f-unreachable*: $f \in$ *unreach−on ae from b*
using *two-in-unreach b-inQ in-path-event path-Q path-ae* **by** *blast*
then have *b-unreachable*: $b \in$ *unreach−on Q from f* **using** *unreach-equiv*
by (*metis* (*mono-tags*, *lifting*) *CollectD b-inQ path-Q unreachable-subset-def*)
have *a-reachable*: $a \in Q -$ *unreach−on Q from f*
using *same-path-reachable2* [**where** $Q = ae$ **and** $R = Q$ **and** $a = a$ **and** $b = f$]
path-ae a-inQ path-Q f-unreachable unreach-on-path **by** *blast*
thus *?thesis*
using *unreachable-set-bounded* [**where** $Qy = b$ **and** $Q = Q$ **and** $b = f$ **and** $Qx = a$]
b-unreachable unreachable-subset-def **by** *auto*
qed

lemma (**in** *MinkowskiSpacetime*) *prolong-betw2*:
assumes *path-Q*: $Q \in \mathcal{P}$
and *a-inQ*: $a \in Q$
and *b-inQ*: $b \in Q$
and *ab-neq*: $a \neq b$
shows $\exists\, c{\in}Q.\ [a;b;c]$
by (*metis assms betw-c-in-path prolong-betw*)

lemma (**in** *MinkowskiSpacetime*) *prolong-betw3*:
assumes *path-Q*: $Q \in \mathcal{P}$
and *a-inQ*: $a \in Q$
and *b-inQ*: $b \in Q$
and *ab-neq*: $a \neq b$
shows $\exists\, c{\in}Q.\ \exists\, d{\in}Q.\ [a;b;c] \wedge [a;b;d] \wedge c{\neq}d$
by (*metis* (*full-types*) *abc-abc-neq abc-bcd-abd a-inQ ab-neq b-inQ path-Q prolong-betw2*)

lemma *finite-path-has-ends*:
assumes $Q \in \mathcal{P}$
and $X \subseteq Q$
and *finite X*
and *card* $X \geq 3$
shows $\exists\, a{\in}X.\ \exists\, b{\in}X.\ a \neq b \wedge (\forall\, c{\in}X.\ a \neq c \wedge b \neq c \longrightarrow [a;c;b])$
using *assms*

```
proof (induct card X − 3 arbitrary: X)
  case 0
  then have card X = 3
    by linarith
  then obtain a b c where X-eq: X = {a, b, c}
    by (metis card-Suc-eq numeral-3-eq-3)
  then have abc-neq: a ≠ b a ≠ c b ≠ c
    by (metis ‹card X = 3› empty-iff insert-iff order-refl three-in-set3)+
  then consider [a;b;c] | [b;c;a] | [c;a;b]
    using some-betw [of Q a b c] 0.prems(1) 0.prems(2) X-eq by auto
  thus ?case
  proof (cases)
    assume [a;b;c]
    thus ?thesis — All d not equal to a or c is just d = b, so it immediately follows.
      using X-eq abc-neq(2) by blast
  next
    assume [b;c;a]
    thus ?thesis
      by (simp add: X-eq abc-neq(1))
  next
    assume [c;a;b]
    thus ?thesis
      using X-eq abc-neq(3) by blast
  qed
next
  case IH: (Suc n)
  obtain Y x where X-eq: X = insert x Y and x ∉ Y
    by (meson IH.prems(4) Set.set-insert three-in-set3)
  then have card Y − 3 = n card Y ≥ 3
    using IH.hyps(2) IH.prems(3) X-eq ‹x ∉ Y› by auto
  then obtain a b where ab-Y: a ∈ Y b ∈ Y a ≠ b
                  and Y-ends: ∀ c∈Y. (a ≠ c ∧ b ≠ c) ⟶ [a;c;b]
    using IH(1) [of Y] IH.prems(1−3) X-eq by auto
  consider [a;x;b] | [x;b;a] | [b;a;x]
    using some-betw [of Q a x b] ab-Y IH.prems(1,2) X-eq ‹x ∉ Y› by auto
  thus ?case
  proof (cases)
    assume [a;x;b]
    thus ?thesis
      using Y-ends X-eq ab-Y by auto
  next
    assume [x;b;a]
    { fix c
      assume c ∈ X x ≠ c a ≠ c
      then have [x;c;a]
        by (smt IH.prems(2) X-eq Y-ends ‹[x;b;a]› ab-Y(1) abc-abc-neq abc-bcd-abd
abc-only-cba(3) abc-sym ‹Q ∈ 𝒫› betw-b-in-path insert-iff some-betw subsetD)
    }
    thus ?thesis
```

using *X-eq* ‹[*x*;*b*;*a*]› *ab-Y*(*1*) *abc-abc-neq insert-iff* **by** *force*
next
assume [*b*;*a*;*x*]
{ fix *c*
assume $c \in X$ $b \neq c$ $x \neq c$
then have [*b*;*c*;*x*]
by (*smt IH.prems*(*2*) *X-eq Y-ends* ‹[*b*;*a*;*x*]› *ab-Y*(*1*) *abc-abc-neq abc-bcd-acd abc-only-cba*(*1*)
abc-sym ‹$Q \in \mathcal{P}$› *betw-a-in-path insert-iff some-betw subsetD*)
}
thus *?thesis*
using *X-eq* ‹$x \notin Y$› *ab-Y*(*2*) **by** *fastforce*
qed
qed

lemma *obtain-fin-path-ends*:
assumes *path-X*: $X \in \mathcal{P}$
and *fin-Q*: *finite Q*
and *card-Q*: *card* $Q \geq 3$
and *events-Q*: $Q \subseteq X$
obtains *a b* **where** $a \neq b$ **and** $a \in Q$ **and** $b \in Q$ **and** $\forall c \in Q.\ (a \neq c \wedge b \neq c) \longrightarrow$ [*a*;*c*;*b*]
proof −
obtain *n* **where** $n \geq 0$ **and** *card* $Q = n+3$
using *card-Q nat-le-iff-add*
by *auto*
then obtain *a b* **where** $a \neq b$ **and** $a \in Q$ **and** $b \in Q$ **and** $\forall c \in Q.\ (a \neq c \wedge b \neq c) \longrightarrow$ [*a*;*c*;*b*]
using *finite-path-has-ends assms* ‹$n \geq 0$›
by *metis*
thus *?thesis*
using *that* **by** *auto*
qed

lemma *path-card-nil*:
assumes $Q \in \mathcal{P}$
shows *card* $Q = 0$
proof (*rule ccontr*)
assume *card* $Q \neq 0$
obtain *n* **where** $n =$ *card Q*
by *auto*
hence $n \geq 1$
using ‹*card* $Q \neq 0$› **by** *linarith*
then consider (*n1*) $n=1$ | (*n2*) $n=2$ | (*n3*) $n \geq 3$
by *linarith*
thus *False*
proof (*cases*)

```
    case n1
    thus ?thesis
      using One-nat-def card-Suc-eq ge2-events-lax singletonD assms(1)
      by (metis ‹n = card Q›)
  next
    case n2
    then obtain a b where a≠b and a∈Q and b∈Q
      using ge2-events-lax assms(1) by blast
    then obtain c where c∈Q and c≠a and c≠b
      using prolong-betw2 by (metis abc-abc-neq assms(1))
    hence card Q ≠ 2
      by (metis ‹a ∈ Q› ‹a ≠ b› ‹b ∈ Q› card-2-iff')
    thus False
      using ‹n = card Q› ‹n = 2› by blast
  next
    case n3
    have fin-Q: finite Q
    proof −
      have (0::nat) ≠ 1
        by simp
      then show ?thesis
        by (meson ‹card Q ≠ 0› card.infinite)
    qed
    have card-Q: card Q ≥ 3
      using ‹n = card Q› n3 by blast
    have Q⊆Q by simp
    then obtain a b where a∈Q and b∈Q and a≠b
        and acb: ∀ c∈Q. (c≠a ∧ c≠b) ⟶ [a;c;b]
      using obtain-fin-path-ends card-Q fin-Q assms(1)
      by metis
    then obtain x where [a;b;x] and x∈Q
      using prolong-betw2 assms(1) by blast
    thus False
      by (metis acb abc-abc-neq abc-only-cba(2))
  qed
qed


theorem infinite-paths:
  assumes P∈𝒫
  shows infinite P
proof
  assume fin-P: finite P
  have P≠{}
    by (simp add: assms)
  hence card P ≠ 0
    by (simp add: fin-P)
  moreover have ¬(card P ≥ 1)
    using path-card-nil
```

```
    by (simp add: assms)
  ultimately show False
    by simp
qed


end
```

27 3.5 Second collinearity theorem

We start with a useful betweenness lemma.

```
lemma (in MinkowskiBetweenness) some-betw2:
  assumes path-Q: Q ∈ 𝒫
      and a-inQ: a ∈ Q
      and b-inQ: b ∈ Q
      and c-inQ: c ∈ Q
  shows a = b ∨ a = c ∨ b = c ∨ [a;b;c] ∨ [b;c;a] ∨ [c;a;b]
  using a-inQ b-inQ c-inQ path-Q some-betw by blast

lemma (in MinkowskiPrimitive) paths-tri:
  assumes path-ab: path ab a b
      and path-bc: path bc b c
      and path-ca: path ca c a
      and a-notin-bc: a ∉ bc
  shows △ a b c
proof −
  have abc-events: a ∈ ℰ ∧ b ∈ ℰ ∧ c ∈ ℰ
    using path-ab path-bc path-ca in-path-event by auto
  have abc-neq: a ≠ b ∧ a ≠ c ∧ b ≠ c
    using path-ab path-bc path-ca by auto
  have paths-neq: ab ≠ bc ∧ ab ≠ ca ∧ bc ≠ ca
    using a-notin-bc cross-once-notin path-ab path-bc path-ca by blast
  show ?thesis
    unfolding kinematic-triangle-def
    using abc-events abc-neq paths-neq path-ab path-bc path-ca
    by auto
qed

lemma (in MinkowskiPrimitive) paths-tri2:
  assumes path-ab: path ab a b
      and path-bc: path bc b c
      and path-ca: path ca c a
      and ab-neq-bc: ab ≠ bc
  shows △ a b c
by (meson ab-neq-bc cross-once-notin path-ab path-bc path-ca paths-tri)
```

Schutz states it more like ⟦*tri-abc*; *bcd*; *cea*⟧ ⟹ (*path de d e* ⟶ ∃*f*∈*de*. [*a*;*f*;*b*]∧[*d*;*e*;*f*]). Equivalent up to usage of *impI*.

theorem (**in** *MinkowskiChain*) *collinearity2*:
assumes *tri-abc*: △ *a b c*
and *bcd*: [*b*;*c*;*d*]
and *cea*: [*c*;*e*;*a*]
and *path-de*: *path de d e*
shows ∃ *f*. [*a*;*f*;*b*] ∧ [*d*;*e*;*f*]
proof −
obtain *ab* **where** *path-ab*: *path ab a b* **using** *tri-abc triangle-paths-unique* **by** *blast*
then obtain *f* **where** *afb*: [*a*;*f*;*b*]
and *f-in-de*: *f* ∈ *de* **using** *collinearity tri-abc path-de path-ab bcd cea* **by** *blast*

obtain *af* **where** *path-af*: *path af a f* **using** *abc-abc-neq afb betw-b-in-path path-ab* **by** *blast*
have [*d*;*e*;*f*]
proof −
have *def-in-de*: *d* ∈ *de* ∧ *e* ∈ *de* ∧ *f* ∈ *de* **using** *path-de f-in-de* **by** *simp*
then have *five-poss*:*f* = *d* ∨ *f* = *e* ∨ [*e*;*f*;*d*] ∨ [*f*;*d*;*e*] ∨ [*d*;*e*;*f*]
using *path-de some-betw2* **by** *blast*
have *f* = *d* ∨ *f* = *e* ⟶ (∃ *Q*∈𝒫. *a* ∈ *Q* ∧ *b* ∈ *Q* ∧ *c* ∈ *Q*)
by (*metis abc-abc-neq afb bcd betw-a-in-path betw-b-in-path cea path-ab*)
then have *f-neq-d-e*: *f* ≠ *d* ∧ *f* ≠ *e* **using** *tri-abc*
using *triangle-diff-paths* **by** *simp*
then consider [*e*;*f*;*d*] | [*f*;*d*;*e*] | [*d*;*e*;*f*] **using** *five-poss* **by** *linarith*
thus *?thesis*
proof (*cases*)
assume *efd*: [*e*;*f*;*d*]
obtain *dc* **where** *path-dc*: *path dc d c* **using** *abc-abc-neq abc-ex-path bcd* **by** *blast*
obtain *ce* **where** *path-ce*: *path ce c e* **using** *abc-abc-neq abc-ex-path cea* **by** *blast*
have *dc*≠*ce*
using *bcd betw-a-in-path betw-c-in-path cea path-ce path-dc tri-abc triangle-diff-paths*
by *blast*
hence △ *d c e*
using *paths-tri2 path-ce path-dc path-de* **by** *blast*
then obtain *x* **where** *x-in-af*: *x* ∈ *af*
and *dxc*: [*d*;*x*;*c*]
using *collinearity*
[**where** *a* = *d* **and** *b* = *c* **and** *c* = *e* **and** *d* = *a* **and** *e* = *f* **and** *de* = *af*]
cea efd path-dc path-af **by** *blast*
then have *x-in-dc*: *x* ∈ *dc* **using** *betw-b-in-path path-dc* **by** *blast*
then have *x* = *b* **using** *eq-paths* **by** (*metis path-af path-dc afb bcd tri-abc x-in-af betw-a-in-path betw-c-in-path triangle-diff-paths*)
then have [*d*;*b*;*c*] **using** *dxc* **by** *simp*

then have *False* **using** *bcd abc-only-cba* [**where** $a = b$ **and** $b = c$ **and** $c = d$] **by** *simp*
thus *?thesis* **by** *simp*
next
assume *fde*: [f;d;e]
obtain *bd* **where** *path-bd*: *path bd b d* **using** *abc-abc-neq abc-ex-path bcd* **by** *blast*
obtain *ea* **where** *path-ea*: *path ea e a* **using** *abc-abc-neq abc-ex-path-unique cea* **by** *blast*
obtain *fe* **where** *path-fe*: *path fe f e* **using** *f-in-de f-neq-d-e path-de* **by** *blast*
have *fe*≠*ea*
using *tri-abc afb cea path-ea path-fe*
by (*metis abc-abc-neq betw-a-in-path betw-c-in-path triangle-paths-neq*)
hence △ *e a f*
by (*metis path-unique path-af path-ea path-fe paths-tri2*)
then obtain *y* **where** *y-in-bd*: $y \in bd$
and *eya*: [e;y;a] **thm** *collinearity*
using *collinearity*
[**where** $a = e$ **and** $b = a$ **and** $c = f$ **and** $d = b$ **and** $e = d$ **and** $de = bd$]
afb fde path-bd path-ea **by** *blast*
then have $y = c$ **by** (*metis* (*mono-tags*, *lifting*)
afb bcd cea path-bd tri-abc
abc-ac-neq betw-b-in-path path-unique triangle-paths(*2*)
triangle-paths-neq)
then have [e;c;a] **using** *eya* **by** *simp*
then have *False* **using** *cea abc-only-cba* [**where** $a = c$ **and** $b = e$ **and** $c = a$] **by** *simp*
thus *?thesis* **by** *simp*
next
assume [d;e;f]
thus *?thesis* **by** *assumption*
qed
qed
thus *?thesis* **using** *afb f-in-de* **by** *blast*
qed

28 3.6 Order on a path - Theorems 8 and 9

context *MinkowskiSpacetime* **begin**

28.1 Theorem 8 (as in Veblen (1911) Theorem 6)

Note $a'b'c'$ don't necessarily form a triangle, as there still needs to be paths between them.

theorem (**in** *MinkowskiChain*) *tri-betw-no-path*:
assumes *tri-abc*: △ *a b c*
and *ab′c*: [a; b'; c]

and $bc'a$: $[b;\ c';\ a]$
and $ca'b$: $[c;\ a';\ b]$
shows $\neg\ (\exists\, Q{\in}\mathcal{P}.\ a' \in Q \wedge b' \in Q \wedge c' \in Q)$
proof −
have *abc-a′b′c′-neq*: $a \neq a' \wedge a \neq b' \wedge a \neq c'$
$\wedge\ b \neq a' \wedge b \neq b' \wedge b \neq c'$
$\wedge\ c \neq a' \wedge c \neq b' \wedge c \neq c'$
using *abc-ac-neq*
by (*metis ab′c abc-abc-neq bc′a ca′b tri-abc triangle-not-betw-abc triangle-permutes*(*4*))

have *tri-betw-no-path-single-case*: *False*
if $a'b'c'$: $[a';\ b';\ c']$ **and** *tri-abc*: $\triangle\ a\ b\ c$
and $ab'c$: $[a;\ b';\ c]$ **and** $bc'a$: $[b;\ c';\ a]$ **and** $ca'b$: $[c;\ a';\ b]$
for $a\ b\ c\ a'\ b'\ c'$
proof −
have *abc-a′b′c′-neq*: $a \neq a' \wedge a \neq b' \wedge a \neq c'$
$\wedge\ b \neq a' \wedge b \neq b' \wedge b \neq c'$
$\wedge\ c \neq a' \wedge c \neq b' \wedge c \neq c'$
using *abc-abc-neq that* **by** (*metis triangle-not-betw-abc triangle-permutes*(*4*))
have $c'b'a'$: $[c';\ b';\ a']$ **using** *abc-sym* $a'b'c'$ **by** *simp*
have *nopath-a′c′b*: $\neg\ (\exists\, Q{\in}\mathcal{P}.\ a' \in Q \wedge c' \in Q \wedge b \in Q)$
proof (*rule notI*)
assume $\exists\, Q{\in}\mathcal{P}.\ a' \in Q \wedge c' \in Q \wedge b \in Q$
then obtain Q **where** *path-Q*: $Q \in \mathcal{P}$
and *a′-inQ*: $a' \in Q$
and *c′-inQ*: $c' \in Q$
and *b-inQ*: $b \in Q$ **by** *blast*
then have *ac-inQ*: $a \in Q \wedge c \in Q$ **using** *eq-paths*
by (*metis abc-a′b′c′-neq ca′b bc′a betw-a-in-path betw-c-in-path*)
thus *False* **using** *b-inQ path-Q tri-abc triangle-diff-paths* **by** *blast*
qed
then have *tri-a′bc′*: $\triangle\ a'\ b\ c'$
by (*smt bc′a ca′b a′b′c′ paths-tri abc-ex-path-unique*)
obtain ab' **where** *path-ab′*: *path* $ab'\ a\ b'$ **using** *ab′c abc-a′b′c′-neq abc-ex-path* **by** *blast*
obtain $a'b$ **where** *path-a′b*: *path* $a'b\ a'\ b$ **using** *tri-a′bc′ triangle-paths*(*1*) **by** *blast*
then have $\exists\, x{\in}a'b.\ [a';\ x;\ b] \wedge [a;\ b';\ x]$
using *collinearity2* [**where** $a = a'$ **and** $b = b$ **and** $c = c'$ **and** $e = b'$ **and** $d = a$ **and** $de = ab'$]
bc′a betw-b-in-path c′b′a′ path-ab′ tri-a′bc′ **by** *blast*
then obtain x **where** *x-in-a′b*: $x \in a'b$
and $a'xb$: $[a';\ x;\ b]$
and $ab'x$: $[a;\ b';\ x]$ **by** *blast*

have *c-in-ab′*: $c \in ab'$ **using** *ab′c betw-c-in-path path-ab′* **by** *auto*
have *c-in-a′b*: $c \in a'b$ **using** *ca′b betw-a-in-path path-a′b* **by** *auto*
have *ab′-a′b-distinct*: $ab' \neq a'b$

using *c-in-a′b path-a′b path-ab′ tri-abc triangle-diff-paths* **by** *blast*
have $ab' \cap a'b = \{c\}$
using *paths-cross-at ab′-a′b-distinct c-in-a′b c-in-ab′ path-a′b path-ab′* **by** *auto*
then have $x = c$ **using** *ab′x path-ab′ x-in-a′b betw-c-in-path* **by** *auto*
then have $[a'; c; b]$ **using** *a′xb* **by** *auto*
thus *?thesis* **using** *ca′b abc-only-cba* **by** *blast*
qed

show *?thesis*
proof (*rule notI*)
assume *path-a′b′c′*: $\exists\, Q \in \mathcal{P}.\ a' \in Q \land b' \in Q \land c' \in Q$
consider $[a'; b'; c'] \mid [b'; c'; a'] \mid [c'; a'; b']$ **using** *some-betw*
by (*smt abc-a′b′c′-neq path-a′b′c′ bc′a ca′b ab′c tri-abc abc-ex-path cross-once-notin triangle-diff-paths*)
thus *False*
apply (*cases*)
using *tri-betw-no-path-single-case*[*of a′ b′ c′*] *ab′c bc′a ca′b tri-abc* **apply** *blast*
using *tri-betw-no-path-single-case ab′c bc′a ca′b tri-abc triangle-permutes abc-sym* **by** *blast+*
qed
qed

28.2 Theorem 9

We now begin working on the transitivity lemmas needed to prove Theorem 9. Multiple lemmas below obtain primed variables (e.g. d'). These are starred in Schutz (e.g. $d*$), but that notation is already reserved in Isabelle.

lemma *unreachable-bounded-path-only*:
assumes *d′-def*: $d' \notin$ *unreach−on ab from e* $d' \in ab$ $d' \neq e$
and *e-event*: $e \in \mathcal{E}$
and *path-ab*: $ab \in \mathcal{P}$
and *e-notin-S*: $e \notin ab$
shows $\exists\, d'e.$ *path d′e d′ e*
proof (*rule ccontr*)
assume $\neg(\exists\, d'e.$ *path d′e d′ e*$)$
hence $\neg(\exists\, R \in \mathcal{P}.\ d' \in R \land e \in R \land d' \neq e)$
by *blast*
hence $\neg(\exists\, R \in \mathcal{P}.\ e \in R \land d' \in R)$
using *d′-def*(*3*) **by** *blast*
moreover have $ab \in \mathcal{P} \land e \in \mathcal{E} \land e \notin ab$
by (*simp add*: *e-event e-notin-S path-ab*)
ultimately have $d' \in$ *unreach−on ab from e*
unfolding *unreachable-subset-def* **using** *d′-def*(*2*)
by *blast*
thus *False*
using *d′-def*(*1*) **by** *auto*
qed

lemma *unreachable-bounded-path*:
assumes *S-neq-ab*: $S \neq ab$
and *a-inS*: $a \in S$
and *e-inS*: $e \in S$
and *e-neq-a*: $e \neq a$
and *path-S*: $S \in \mathcal{P}$
and *path-ab*: *path ab a b*
and *path-be*: *path be b e*
and *no-de*: $\neg(\exists\, de.\ path\ de\ d\ e)$
and *abd*:$[a;b;d]$
obtains d' $d'e$ **where** $d'{\in}ab \wedge path\ d'e\ d'\ e \wedge [b;\ d;\ d']$
proof −
have *e-event*: $e{\in}\mathcal{E}$
using *e-inS path-S* **by** *auto*
have $e{\notin}ab$
using *S-neq-ab a-inS e-inS e-neq-a eq-paths path-S path-ab* **by** *auto*
have $ab{\in}\mathcal{P} \wedge e{\notin}ab$
using *S-neq-ab a-inS e-inS e-neq-a eq-paths path-S path-ab*
by *auto*
have $b \in ab - unreach{-}on\ ab\ from\ e$
using *cross-in-reachable path-ab path-be*
by *blast*
have $d \in unreach{-}on\ ab\ from\ e$
using *no-de abd path-ab e-event* ‹$e{\notin}ab$›
betw-c-in-path unreachable-bounded-path-only
by *blast*
have $\exists\, d'\ d'e.\ d'{\in}ab \wedge path\ d'e\ d'\ e \wedge [b;\ d;\ d']$
proof −
obtain d' **where** $[b;\ d;\ d']$ $d'{\in}ab$ $d'{\notin}\ unreach{-}on\ ab\ from\ e$ $b{\neq}d'$ $e{\neq}d'$
using *unreachable-set-bounded* ‹$b \in ab - unreach{-}on\ ab\ from\ e$› ‹$d \in un$-
$reach{-}on\ ab\ from\ e$› *e-event* ‹$e{\notin}ab$› *path-ab*
by (*metis DiffE*)
then obtain $d'e$ **where** *path* $d'e\ d'\ e$
using *unreachable-bounded-path-only e-event* ‹$e{\notin}ab$› *path-ab*
by *blast*
thus *?thesis*
using ‹$[b;\ d;\ d']$› ‹$d' \in ab$›
by *blast*
qed
thus *?thesis*
using *that* **by** *blast*
qed

This lemma collects the first three paragraphs of Schutz' proof of Theorem 9 - Lemma 1. Several case splits need to be considered, but have no further importance outside of this lemma: thus we parcel them away from the main proof.

lemma *exist-*$c'd'$*-alt*:
assumes *abc*: $[a;b;c]$

```
    and abd: [a;b;d]
    and dbc: [d;b;c]
    and c-neq-d: c ≠ d
    and path-ab: path ab a b
    and path-S: S ∈ 𝒫
    and a-inS: a ∈ S
    and e-inS: e ∈ S
    and e-neq-a: e ≠ a
    and S-neq-ab: S ≠ ab
    and path-be: path be b e
  shows ∃ c' d'. ∃ d'e c'e. c'∈ab ∧ d'∈ab
                    ∧ [a; b; d'] ∧ [c'; b; a] ∧ [c'; b; d']
                    ∧ path d'e d' e ∧ path c'e c' e
proof (cases)
  assume ∃ de. path de d e
  then obtain de where path de d e
    by blast
  hence [a;b;d] ∧ d∈ab
    using abd betw-c-in-path path-ab by blast
  thus ?thesis
  proof (cases)
    assume ∃ ce. path ce c e
    then obtain ce where path ce c e by blast
    have c ∈ ab
      using abc betw-c-in-path path-ab by blast
    thus ?thesis
     using ‹[a;b;d] ∧ d ∈ ab› ‹∃ ce. path ce c e› ‹c ∈ ab› ‹path de d e› abc abc-sym
dbc
      by blast
  next
    assume ¬(∃ ce. path ce c e)
    obtain c' c'e where c'∈ab ∧ path c'e c' e ∧ [b; c; c']
      using unreachable-bounded-path [where ab=ab and e=e and b=b and d=c
and a=a and S=S and be=be]
       S-neq-ab ‹¬(∃ ce. path ce c e)› a-inS abc e-inS e-neq-a path-S path-ab path-be
    by (metis (mono-tags, lifting))
    hence [a; b; c'] ∧ [d; b; c']
      using abc dbc by blast
    hence [c'; b; a] ∧ [c'; b; d]
      using theorem1 by blast
    thus ?thesis
      using ‹[a;b;d] ∧ d ∈ ab› ‹c' ∈ ab ∧ path c'e c' e ∧ [b; c; c']› ‹path de d e›
      by blast
  qed
next
  assume ¬ (∃ de. path de d e)
  obtain d' d'e where d'-in-ab: d' ∈ ab
                and bdd': [b; d; d']
                and path d'e d' e
```

```
    using unreachable-bounded-path [where ab=ab and e=e and b=b and d=d
and a=a and S=S and be=be]
      S-neq-ab ‹∄ de. path de d e› a-inS abd e-inS e-neq-a path-S path-ab path-be
    by (metis (mono-tags, lifting))
  hence [a; b; d′] using abd by blast
  thus ?thesis
  proof (cases)
    assume ∃ ce. path ce c e
    then obtain ce where path ce c e by blast
    have c ∈ ab
      using abc betw-c-in-path path-ab by blast
    thus ?thesis
      using ‹[a; b; d′]› ‹d′ ∈ ab› ‹path ce c e› ‹c ∈ ab› ‹path d′e d′ e› abc abc-sym
dbc
      by (meson abc-bcd-acd bdd′)
  next
    assume ¬(∃ ce. path ce c e)
    obtain c′ c′e where c′∈ab ∧ path c′e c′ e ∧ [b; c; c′]
      using unreachable-bounded-path [where ab=ab and e=e and b=b and d=c
and a=a and S=S and be=be]
        S-neq-ab ‹¬(∃ ce. path ce c e)› a-inS abc e-inS e-neq-a path-S path-ab path-be
    by (metis (mono-tags, lifting))
    hence [a; b; c′] ∧ [d; b; c′]
      using abc dbc by blast
    hence [c′; b; a] ∧ [c′; b; d]
      using theorem1 by blast
    thus ?thesis
      using ‹[a; b; d′]› ‹c′ ∈ ab ∧ path c′e c′ e ∧ [b; c; c′]› ‹path d′e d′ e› bdd′
d′-in-ab
      by blast
  qed
qed

lemma exist-c′d′:
  assumes abc: [a;b;c]
      and abd: [a;b;d]
      and dbc: [d;b;c]
      and path-S: path S a e
      and path-be: path be b e
      and S-neq-ab: S ≠ path-of a b
    shows ∃ c′ d′. [a; b; d′] ∧ [c′; b; a] ∧ [c′; b; d′] ∧
               path-ex d′ e ∧ path-ex c′ e
proof (cases path-ex d e)
  let ?ab = path-of a b
  have path-ex a b
    using abc abc-abc-neq abc-ex-path by blast
  hence path-ab: path ?ab a b using path-of-ex by simp
  have c≠d using abc-ac-neq dbc by blast
  {
```

case *True*
then obtain *de* **where** *path de d e*
by *blast*
hence [*a*;*b*;*d*] ∧ *d*∈*?ab*
using *abd betw-c-in-path path-ab* **by** *blast*
thus *?thesis*
proof (*cases path-ex c e*)
case *True*
then obtain *ce* **where** *path ce c e* **by** *blast*
have *c* ∈ *?ab*
using *abc betw-c-in-path path-ab* **by** *blast*
thus *?thesis*
using ‹[*a*;*b*;*d*] ∧ *d* ∈ *?ab*› ‹∃ *ce*. *path ce c e*› ‹*c* ∈ *?ab*› ‹*path de d e*› *abc abc-sym dbc*
by *blast*
next
case *False*
obtain *c′ c′e* **where** *c′*∈*?ab* ∧ *path c′e c′ e* ∧ [*b*; *c*; *c′*]
using *unreachable-bounded-path* [**where** *ab*=*?ab* **and** *e*=*e* **and** *b*=*b* **and** *d*=*c* **and** *a*=*a* **and** *S*=*S* **and** *be*=*be*]
S-neq-ab ‹¬(∃ *ce*. *path ce c e*)› *abc path-S path-ab path-be*
by (*metis* (*mono-tags*, *lifting*))
hence [*a*; *b*; *c′*] ∧ [*d*; *b*; *c′*]
using *abc dbc* **by** *blast*
hence [*c′*; *b*; *a*] ∧ [*c′*; *b*; *d*]
using *theorem1* **by** *blast*
thus *?thesis*
using ‹[*a*;*b*;*d*] ∧ *d* ∈ *?ab*› ‹*c′* ∈ *?ab* ∧ *path c′e c′ e* ∧ [*b*; *c*; *c′*]› ‹*path de d e*›
by *blast*
qed
} {
case *False*
obtain *d′ d′e* **where** *d′-in-ab*: *d′* ∈ *?ab*
and *bdd′*: [*b*; *d*; *d′*]
and *path d′e d′ e*
using *unreachable-bounded-path* [**where** *ab*=*?ab* **and** *e*=*e* **and** *b*=*b* **and** *d*=*d* **and** *a*=*a* **and** *S*=*S* **and** *be*=*be*]
S-neq-ab ‹¬*path-ex d e*› *abd path-S path-ab path-be*
by (*metis* (*mono-tags*, *lifting*))
hence [*a*; *b*; *d′*] **using** *abd* **by** *blast*
thus *?thesis*
proof (*cases path-ex c e*)
case *True*
then obtain *ce* **where** *path ce c e* **by** *blast*
have *c* ∈ *?ab*
using *abc betw-c-in-path path-ab* **by** *blast*
thus *?thesis*
using ‹[*a*; *b*; *d′*]› ‹*d′* ∈ *?ab*› ‹*path ce c e*› ‹*c* ∈ *?ab*› ‹*path d′e d′ e*› *abc abc-sym dbc*

```
        by (meson abc-bcd-acd bdd')
    next
      case False
      obtain c' c'e where c'∈?ab ∧ path c'e c' e ∧ [b; c; c']
         using unreachable-bounded-path [where ab=?ab and e=e and b=b and
d=c and a=a and S=S and be=be]
          S-neq-ab ‹¬(path-ex c e)› abc path-S path-ab path-be
      by (metis (mono-tags, lifting))
      hence [a; b; c'] ∧ [d; b; c']
        using abc dbc by blast
      hence [c'; b; a] ∧ [c'; b; d]
        using theorem1 by blast
      thus ?thesis
        using ‹[a; b; d']› ‹c' ∈ ?ab ∧ path c'e c' e ∧ [b; c; c']› ‹path d'e d' e› bdd'
d'-in-ab
        by blast
    qed
  }
qed


lemma exist-f'-alt:
  assumes path-ab: path ab a b
      and path-S: S ∈ 𝒫
      and a-inS: a ∈ S
      and e-inS: e ∈ S
      and e-neq-a: e ≠ a
      and f-def: [e; c'; f] f∈c'e
      and S-neq-ab: S ≠ ab
      and c'd'-def: c'∈ab ∧ d'∈ab
            ∧ [a; b; d'] ∧ [c'; b; a] ∧ [c'; b; d']
            ∧ path d'e d' e ∧ path c'e c' e
    shows ∃f'. ∃f'b. [e; c'; f'] ∧ path f'b f' b
proof (cases)
  assume ∃bf. path bf b f
  thus ?thesis
    using ‹[e; c'; f]› by blast
next
  assume ¬(∃bf. path bf b f)
  hence f ∈ unreach−on c'e from b
   using assms(1−5,7−9) abc-abc-neq betw-events eq-paths unreachable-bounded-path-only
    by metis
  moreover have c' ∈ c'e − unreach−on c'e from b
    using c'd'-def cross-in-reachable path-ab by blast
  moreover have b∈ℰ ∧ b∉c'e
     using ‹f ∈ unreach−on c'e from b› betw-events c'd'-def same-empty-unreach
by auto
  ultimately obtain f' where f'-def: [c'; f; f'] f'∈c'e f'∉ unreach−on c'e from
b c'≠f' b≠f'
```

using *unreachable-set-bounded* $c'd'$*-def*
by (*metis DiffE*)
hence $[e;\ c';\ f']$
using ‹$[e;\ c';\ f]$› **by** *blast*
moreover obtain $f'b$ **where** *path* $f'b\ f'\ b$
using ‹$b \in \mathcal{E} \wedge b \notin c'e$› $c'd'$*-def* f'*-def*(2,3) *unreachable-bounded-path-only*
by *blast*
ultimately show *?thesis* **by** *blast*
qed

lemma *exist-*f':
assumes *path-ab*: *path ab a b*
and *path-S*: *path S a e*
and *f-def*: $[e;\ c';\ f]$
and *S-neq-ab*: $S \neq ab$
and $c'd'$*-def*: $[a;\ b;\ d']$ $[c';\ b;\ a]$ $[c';\ b;\ d']$
path $d'e\ d'\ e$ *path* $c'e\ c'\ e$
shows $\exists f'.\ [e;\ c';\ f'] \wedge$ *path-ex* $f'\ b$
proof (*cases*)
assume *path-ex b f*
thus *?thesis*
using *f-def* **by** *blast*
next
assume *no-path*: $\neg$(*path-ex b f*)
have *path-S-2*: $S \in \mathcal{P}\ a \in S\ e \in S\ e \neq a$
using *path-S* **by** *auto*
have $f{\in}c'e$
using *betw-c-in-path f-def* $c'd'$*-def*(5) **by** *blast*
have $c'{\in}\ ab\ d'{\in}\ ab$
using *betw-a-in-path betw-c-in-path* $c'd'$*-def*(1,2) *path-ab* **by** *blast+*
have $f \in$ *unreach*−*on* $c'e$ *from* b
using *no-path assms*(1,4−9) *path-S-2* ‹$f{\in}c'e$› ‹$c'{\in}ab$› ‹$d'{\in}ab$›
abc-abc-neq betw-events eq-paths unreachable-bounded-path-only
by *metis*
moreover have $c' \in c'e$ − *unreach*−*on* $c'e$ *from* b
using $c'd'$*-def cross-in-reachable path-ab* ‹$c' \in ab$› **by** *blast*
moreover have $b{\in}\mathcal{E} \wedge b{\notin}c'e$
using ‹$f \in$ *unreach*−*on* $c'e$ *from* b› *betw-events* $c'd'$*-def same-empty-unreach*
by *auto*
ultimately obtain f' **where** f'*-def*: $[c';\ f;\ f']$ $f'{\in}c'e$ $f'{\notin}$ *unreach*−*on* $c'e$ *from* b $c'{\neq}f'$ $b{\neq}f'$
using *unreachable-set-bounded* $c'd'$*-def*
by (*metis DiffE*)
hence $[e;\ c';\ f']$
using ‹$[e;\ c';\ f]$› **by** *blast*
moreover obtain $f'b$ **where** *path* $f'b\ f'\ b$
using ‹$b \in \mathcal{E} \wedge b \notin c'e$› $c'd'$*-def* f'*-def*(2,3) *unreachable-bounded-path-only*
by *blast*
ultimately show *?thesis* **by** *blast*

qed

lemma *abc-abd-bcdbdc*:
  assumes *abc*: [*a*;*b*;*c*]
    and *abd*: [*a*;*b*;*d*]
    and *c-neq-d*: $c \neq d$
  shows [*b*;*c*;*d*] ∨ [*b*;*d*;*c*]
proof −
  have ¬ [*d*;*b*;*c*]
  proof (*rule notI*)
    assume *dbc*: [*d*;*b*;*c*]
    obtain *ab* **where** *path-ab*: *path ab a b*
      using *abc-abc-neq abc-ex-path-unique abc* **by** *blast*
    obtain *S* **where** *path-S*: $S \in \mathcal{P}$
        and *S-neq-ab*: $S \neq ab$
        and *a-inS*: $a \in S$
      using *ex-crossing-at path-ab*
      by *auto*

    have $\exists\, e{\in}S.\ e \neq a \wedge (\exists\, be{\in}\mathcal{P}.\ path\ be\ b\ e)$
    proof −
      have *b-notinS*: $b \notin S$ **using** *S-neq-ab a-inS path-S path-ab path-unique* **by** *blast*
      then obtain *x y z* **where** *x-in-unreach*: $x \in$ *unreach−on S from b*
          and *y-in-unreach*: $y \in$ *unreach−on S from b*
          and *x-neq-y*: $x \neq y$
          and *z-in-reach*: $z \in S -$ *unreach−on S from b*
        using *two-in-unreach* [**where** $Q = S$ **and** $b = b$]
          in-path-event path-S path-ab a-inS cross-in-reachable
        by *blast*
      then obtain *w* **where** *w-in-reach*: $w \in S -$ *unreach−on S from b*
          and *w-neq-z*: $w \neq z$
        using *unreachable-set-bounded* [**where** $Q = S$ **and** $b = b$ **and** $Qx = z$ **and** $Qy = x$]
          b-notinS in-path-event path-S path-ab **by** *blast*
      thus *?thesis* **by** (*metis DiffD1 b-notinS in-path-event path-S path-ab reachable-path z-in-reach*)
    qed
    then obtain *e be* **where** *e-inS*: $e \in S$
        and *e-neq-a*: $e \neq a$
        and *path-be*: *path be b e*
      by *blast*
    have *path-ae*: *path S a e*
      using *a-inS e-inS e-neq-a path-S* **by** *auto*
    have *S-neq-ab-2*: $S \neq$ *path-of a b*
      using *S-neq-ab cross-once-notin path-ab path-of-ex* **by** *blast*

have $\exists$ *c' d'*.
c'∈*ab* ∧ *d'*∈*ab*
∧ [*a*; *b*; *d'*] ∧ [*c'*; *b*; *a*] ∧ [*c'*; *b*; *d'*]
∧ *path-ex d' e* ∧ *path-ex c' e*
using *exist-c'd'* [**where** *a*=*a* **and** *b*=*b* **and** *c*=*c* **and** *d*=*d* **and** *e*=*e* **and** *be*=*be* **and** *S*=*S*]
using *assms*(*1*−*2*) *dbc e-neq-a path-ae path-be S-neq-ab-2*
using *abc-sym betw-a-in-path path-ab* **by** *blast*
then obtain *c' d' d'e c'e*
where *c'd'-def*: *c'*∈*ab* ∧ *d'*∈*ab*
∧ [*a*; *b*; *d'*] ∧ [*c'*; *b*; *a*] ∧ [*c'*; *b*; *d'*]
∧ *path d'e d' e* ∧ *path c'e c' e*
by *blast*

obtain *f* **where** *f-def*: *f*∈*c'e* [*e*; *c'*; *f*]
using *c'd'-def prolong-betw2* **by** *blast*
then obtain *f' f'b* **where** *f'-def*: [*e*; *c'*; *f'*] ∧ *path f'b f' b*
using *exist-f'*
[**where** *e*=*e* **and** *c'*=*c'* **and** *b*=*b* **and** *f*=*f* **and** *S*=*S* **and** *ab*=*ab* **and** *d'*=*d'* **and** *a*=*a* **and** *c'e*=*c'e*]
using *path-ab path-S a-inS e-inS e-neq-a f-def S-neq-ab c'd'-def*
by *blast*

obtain *ae* **where** *path-ae*: *path ae a e* **using** *a-inS e-inS e-neq-a path-S* **by** *blast*
have *tri-aec*: △ *a e c'*
by (*smt cross-once-notin S-neq-ab a-inS abc abc-abc-neq abc-ex-path e-inS e-neq-a path-S path-ab c'd'-def paths-tri*)

then obtain *h* **where** *h-in-f'b*: *h* ∈ *f'b*
and *ahe*: [*a*;*h*;*e*]
and *f'bh*: [*f'*; *b*; *h*]
using *collinearity2* [**where** *a* = *a* **and** *b* = *e* **and** *c* = *c'* **and** *d* = *f'* **and** *e* = *b* **and** *de* = *f'b*]
f'-def c'd'-def f'-def betw-c-in-path **by** *blast*
have *tri-dec*: △ *d' e c'*
using *cross-once-notin S-neq-ab a-inS abc abc-abc-neq abc-ex-path e-inS e-neq-a path-S path-ab c'd'-def paths-tri* **by** *smt*
then obtain *g* **where** *g-in-f'b*: *g* ∈ *f'b*
and *d'ge*: [*d'*; *g*; *e*]
and *f'bg*: [*f'*; *b*; *g*]
using *collinearity2* [**where** *a* = *d'* **and** *b* = *e* **and** *c* = *c'* **and** *d* = *f'* **and** *e* = *b* **and** *de* = *f'b*]
f'-def c'd'-def betw-c-in-path **by** *blast*
have △ *e a d'* **by** (*smt betw-c-in-path paths-tri2 S-neq-ab a-inS abc-ac-neq abd e-inS e-neq-a c'd'-def path-S path-ab*)
thus *False*

using *tri-betw-no-path* [**where** $a = e$ **and** $b = a$ **and** $c = d'$ **and** $b' = g$ **and** $a' = b$ **and** $c' = h$]
f′-def c′d′-def h-in-f′b g-in-f′b abd d′ge ahe abc-sym
by *blast*
qed
thus *?thesis*
by (*smt abc abc-abc-neq abc-ex-path abc-sym abd c-neq-d cross-once-notin some-betw*)
qed

lemma *abc-abd-acdadc*:
assumes *abc*: $[a;b;c]$
and *abd*: $[a;b;d]$
and *c-neq-d*: $c \neq d$
shows $[a;c;d] \vee [a;d;c]$
proof −
have *cba*: $[c;b;a]$ **using** *abc-sym abc* **by** *simp*
have *dba*: $[d;b;a]$ **using** *abc-sym abd* **by** *simp*
have *dcb-over-cba*: $[d;c;b] \wedge [c;b;a] \Longrightarrow [d;c;a]$ **by** *auto*
have *cdb-over-dba*: $[c;d;b] \wedge [d;b;a] \Longrightarrow [c;d;a]$ **by** *auto*

have *bcdbdc*: $[b;c;d] \vee [b;d;c]$ **using** *abc abc-abd-bcdbdc abd c-neq-d* **by** *auto*
then have *dcb-or-cdb*: $[d;c;b] \vee [c;d;b]$ **using** *abc-sym* **by** *blast*
then have $[d;c;a] \vee [c;d;a]$ **using** *abc-only-cba dcb-over-cba cdb-over-dba cba dba* **by** *blast*
thus *?thesis* **using** *abc-sym* **by** *auto*
qed

lemma *abc-acd-bcd*:
assumes *abc*: $[a;b;c]$
and *acd*: $[a;c;d]$
shows $[b;c;d]$
proof −
have *path-abc*: $\exists\, Q{\in}\mathcal{P}.\ a \in Q \wedge b \in Q \wedge c \in Q$ **using** *abc* **by** (*simp add*: *abc-ex-path*)
have *path-acd*: $\exists\, Q{\in}\mathcal{P}.\ a \in Q \wedge c \in Q \wedge d \in Q$ **using** *acd* **by** (*simp add*: *abc-ex-path*)
then have $\exists\, Q{\in}\mathcal{P}.\ b \in Q \wedge c \in Q \wedge d \in Q$ **using** *path-abc abc-abc-neq acd cross-once-notin* **by** *metis*
then have *bcd3*: $[b;c;d] \vee [b;d;c] \vee [c;b;d]$ **by** (*metis abc abc-only-cba*(*1*,*2*) *acd some-betw2*)
show *?thesis*
proof (*rule ccontr*)
assume $\neg\ [b;c;d]$
then have $[b;d;c] \vee [c;b;d]$ **using** *bcd3* **by** *simp*
thus *False*
proof (*rule disjE*)

```
      assume [b;d;c]
      then have [c;d;b] using abc-sym by simp
      then have [a;c;b] using acd abc-bcd-abd by blast
      thus False using abc abc-only-cba by blast
    next
      assume cbd: [c;b;d]
      have cba: [c;b;a] using abc abc-sym by blast
      have a-neq-d: a ≠ d using abc-ac-neq acd by auto
      then have [c;a;d] ∨ [c;d;a] using abc-abd-acdadc cbd cba by simp
      thus False using abc-only-cba acd by blast
    qed
  qed
qed
```

A few lemmas that don't seem to be proved by Schutz, but can be proven now, after Lemma 3. These sometimes avoid us having to construct a chain explicitly.

```
lemma abd-bcd-abc:
  assumes abd: [a;b;d]
      and bcd: [b;c;d]
  shows [a;b;c]
proof −
  have dcb: [d;c;b] using abc-sym bcd by simp
  have dba: [d;b;a] using abc-sym abd by simp
  have [c;b;a] using abc-acd-bcd dcb dba by blast
  thus ?thesis using abc-sym by simp
qed

lemma abc-acd-abd:
  assumes abc: [a;b;c]
      and acd: [a;c;d]
    shows [a;b;d]
  using abc abc-acd-bcd acd by blast

lemma abd-acd-abcacb:
  assumes abd: [a;b;d]
      and acd: [a;c;d]
      and bc: b≠c
    shows [a;b;c] ∨ [a;c;b]
proof −
  obtain P where P-def: P∈𝒫 a∈P b∈P d∈P
    using abd abc-ex-path by blast
  hence c∈P
    using acd abc-abc-neq betw-b-in-path by blast
  have ¬[b;a;c]
    using abc-only-cba abd acd by blast
  thus ?thesis
    by (metis P-def(1−3) ‹c ∈ P› abc-abc-neq abc-sym abd acd bc some-betw)
qed
```

lemma *abe-ade-bcd-ace*:
  assumes *abe*: [*a*;*b*;*e*]
      and *ade*: [*a*;*d*;*e*]
      and *bcd*: [*b*;*c*;*d*]
    shows [*a*;*c*;*e*]
proof −
  have *abdadb*: [*a*;*b*;*d*] ∨ [*a*;*d*;*b*]
    using *abc-ac-neq abd-acd-abcacb abe ade bcd* **by** *auto*
  thus *?thesis*
  proof
    assume [*a*;*b*;*d*] **thus** *?thesis*
      by (*meson abc-acd-abd abc-sym ade bcd*)
  next assume [*a*;*d*;*b*] **thus** *?thesis*
      by (*meson abc-acd-abd abc-sym abe bcd*)
  qed
qed

Now we start on Theorem 9. Based on Veblen (1904) Lemma 2 p357.

lemma (**in** *MinkowskiBetweenness*) *chain3*:
  assumes *path-Q*: $Q \in \mathcal{P}$
      and *a-inQ*: $a \in Q$
      and *b-inQ*: $b \in Q$
      and *c-inQ*: $c \in Q$
      and *abc-neq*: $a \neq b \land a \neq c \land b \neq c$
  shows *ch* {*a*,*b*,*c*}
proof −
  have *abc-betw*: [*a*;*b*;*c*] ∨ [*a*;*c*;*b*] ∨ [*b*;*a*;*c*]
    using *assms* **by** (*meson in-path-event abc-sym some-betw insert-subset*)
  have *ch1*: [*a*;*b*;*c*] ⟶ *ch* {*a*,*b*,*c*}
    using *abc-abc-neq ch-by-ord-def ch-def ord-ordered-loc between-chain* **by** *auto*
  have *ch2*: [*a*;*c*;*b*] ⟶ *ch* {*a*,*c*,*b*}
    using *abc-abc-neq ch-by-ord-def ch-def ord-ordered-loc between-chain* **by** *auto*
  have *ch3*: [*b*;*a*;*c*] ⟶ *ch* {*b*,*a*,*c*}
    using *abc-abc-neq ch-by-ord-def ch-def ord-ordered-loc between-chain* **by** *auto*
  show *?thesis*
    using *abc-betw ch1 ch2 ch3* **by** (*metis insert-commute*)
qed

lemma *overlap-chain*: ⟦[*a*;*b*;*c*]; [*b*;*c*;*d*]⟧ ⟹ *ch* {*a*,*b*,*c*,*d*}
proof −
  assume [*a*;*b*;*c*] **and** [*b*;*c*;*d*]
  have ∃*f*. *local-ordering f betw* {*a*,*b*,*c*,*d*}
  proof −
    have *f1*: [*a*;*b*;*d*]
      using ‹[*a*;*b*;*c*]› ‹[*b*;*c*;*d*]› **by** *blast*
    have [*a*;*c*;*d*]
      using ‹[*a*;*b*;*c*]› ‹[*b*;*c*;*d*]› *abc-bcd-acd* **by** *blast*
    then show *?thesis*

```
    using f1 by (metis (no-types) ‹[a;b;c]› ‹[b;c;d]› abc-abc-neq overlap-ordering-loc)
  qed
  hence ∃f. local-long-ch-by-ord f {a,b,c,d}
    apply (simp add: chain-defs eval-nat-numeral)
    using ‹[a;b;c]› abc-abc-neq
   by (smt (z3) ‹[b;c;d]› card.empty card-insert-disjoint card-insert-le finite.emptyI
      finite.insertI insertE insert-absorb insert-not-empty)
  thus ?thesis
    by (simp add: chain-defs)
qed
```

The book introduces Theorem 9 before the above three lemmas but can only complete the proof once they are proven. This doesn't exactly say it the same way as the book, as the book gives the *local-ordering* (abcd) explicitly (for arbitrarly named events), but is equivalent.

```
theorem chain4:
  assumes path-Q: Q ∈ 𝒫
      and inQ: a ∈ Q b ∈ Q c ∈ Q d ∈ Q
      and abcd-neq: a ≠ b ∧ a ≠ c ∧ a ≠ d ∧ b ≠ c ∧ b ≠ d ∧ c ≠ d
    shows ch {a,b,c,d}
proof −
  obtain a′ b′ c′ where a′-pick: a′ ∈ {a,b,c,d}
                  and b′-pick: b′ ∈ {a,b,c,d}
                  and c′-pick: c′ ∈ {a,b,c,d}
                  and a′b′c′: [a′; b′; c′]
      using some-betw by (metis inQ(1,2,4) abcd-neq insert-iff path-Q)
  then obtain d′ where d′-neq: d′ ≠ a′ ∧ d′ ≠ b′ ∧ d′ ≠ c′
                 and d′-pick: d′ ∈ {a,b,c,d}
    using insert-iff abcd-neq by metis
  have all-picked-on-path: a′∈Q b′∈Q c′∈Q d′∈Q
    using a′-pick b′-pick c′-pick d′-pick inQ by blast+
  consider [d′; a′; b′] | [a′; d′; b′] | [a′; b′; d′]
    using some-betw abc-only-cba all-picked-on-path(1,2,4)
    by (metis a′b′c′ d′-neq path-Q)
  then have picked-chain: ch {a′,b′,c′,d′}
  proof (cases)
    assume [d′; a′; b′]
    thus ?thesis using a′b′c′ overlap-chain by (metis (full-types) insert-commute)
  next
    assume a′d′b′: [a′; d′; b′]
    then have [d′; b′; c′] using abc-acd-bcd a′b′c′ by blast
    thus ?thesis using a′d′b′ overlap-chain by (metis (full-types) insert-commute)
  next
    assume a′b′d′: [a′; b′; d′]
    then have two-cases: [b′; c′; d′] ∨ [b′; d′; c′] using abc-abd-bcdbdc a′b′c′ d′-neq
by blast

    have case1: [b′; c′; d′] ⟹ ?thesis using a′b′c′ overlap-chain by blast
    have case2: [b′; d′; c′] ⟹ ?thesis
```

using *abc-only-cba abc-acd-bcd a′b′d′ overlap-chain*
by (*metis* (*full-types*) *insert-commute*)
show *?thesis* **using** *two-cases case1 case2* **by** *blast*
qed
have $\{a',b',c',d'\} = \{a,b,c,d\}$
proof (*rule Set.set-eqI*, *rule iffI*)
fix x
assume $x \in \{a',b',c',d'\}$
thus $x \in \{a,b,c,d\}$ **using** *a′-pick b′-pick c′-pick d′-pick* **by** *auto*
next
fix x
assume *x-pick*: $x \in \{a,b,c,d\}$
have $a' \neq b' \wedge a' \neq c' \wedge a' \neq d' \wedge b' \neq c' \wedge c' \neq d'$
using *a′b′c′ abc-abc-neq d′-neq* **by** *blast*
thus $x \in \{a',b',c',d'\}$
using *a′-pick b′-pick c′-pick d′-pick x-pick d′-neq* **by** *auto*
qed
thus *?thesis* **using** *picked-chain* **by** *simp*
qed

theorem *chain4-alt*:
assumes *path-Q*: $Q \in \mathcal{P}$
and *abcd-inQ*: $\{a,b,c,d\} \subseteq Q$
and *abcd-distinct*: *card* $\{a,b,c,d\} = 4$
shows *ch* $\{a,b,c,d\}$
proof −
have *abcd-neq*: $a \neq b \wedge a \neq c \wedge a \neq d \wedge b \neq c \wedge b \neq d \wedge c \neq d$
using *abcd-distinct numeral-3-eq-3*
by (*smt* (*z3*) *card-1-singleton-iff card-2-iff card-3-dist insert-absorb2 insert-commute numeral-1-eq-Suc-0 numeral-eq-iff semiring-norm*(*85*) *semiring-norm*(*88*) *verit-eq-simplify*(*8*))
have *inQ*: $a \in Q$ $b \in Q$ $c \in Q$ $d \in Q$
using *abcd-inQ* **by** *auto*
show *?thesis* **using** *chain4*[*OF assms*(*1*) *inQ*] *abcd-neq* **by** *simp*
qed

end

29 Interlude - Chains, segments, rays

context *MinkowskiBetweenness* **begin**

29.1 General results for chains

lemma *inf-chain-is-long*:
assumes $[f \leadsto X|x..]$
shows *local-long-ch-by-ord* $f\ X \wedge f\ 0 = x \wedge$ *infinite* X
using *chain-defs* **by** (*metis assms infinite-chain-alt*)

A reassurance that the starting point x is implied.

```
lemma long-inf-chain-is-semifin:
  assumes local-long-ch-by-ord f X ∧ infinite X
  shows ∃ x. [f⇝X|x..]
  using assms infinite-chain-with-def chain-alts by auto
```

```
lemma endpoint-in-semifin:
  assumes [f⇝X|x..]
  shows x∈X
 using zero-into-ordering-loc by (metis assms empty-iff inf-chain-is-long local-long-ch-by-ord-alt)
```

Yet another corollary to Theorem 2, without indices, for arbitrary events on the chain.

```
corollary all-aligned-on-fin-chain:
  assumes [f⇝X] finite X
  and x: x∈X and y: y∈X and z:z∈X and xy: x≠y and xz: x≠z and yz: y≠z
  shows [x;y;z] ∨ [x;z;y] ∨ [y;x;z]
proof −
  have card X ≥ 3 using assms(2−5) three-subset[OF xy xz yz] by blast
  hence 1: local-long-ch-by-ord f X
   using assms(1,3−) chain-defs by (metis short-ch-alt(1) short-ch-card(1) short-ch-card-2)
  obtain i j k where ijk: x=f i i<card X y=f j j<card X z=f k k<card X
    using obtain-index-fin-chain assms(1−5) by metis
  have 2: [f i;f j;f k] if i<j ∧ j<k k<card X for i j k
    using assms order-finite-chain2 that(1,2) by auto
  consider i<j ∧ j<k|i<k ∧ k<j|j<i ∧ i<k|i>j ∧ j>k|i>k ∧ k>j|j>i ∧ i>k
    using xy xz yz ijk(1,3,5) by (metis linorder-neqE-nat)
  thus ?thesis
    apply cases using 2 abc-sym ijk by presburger+
qed
```

```
lemma (in MinkowskiPrimitive) card2-either-elt1-or-elt2:
  assumes card X = 2 and x∈X and y∈X and x≠y
    and z∈X and z≠x
  shows z=y
by (metis assms card-2-iff')
```

```
lemma get-fin-long-ch-bounds:
  assumes local-long-ch-by-ord f X
    and finite X
  shows ∃x∈X. ∃y∈X. ∃z∈X. [f⇝X|x..y..z]
proof (rule bexI)+
  show 1:[f⇝X|f 0..f 1..f (card X − 1)]
    using assms unfolding finite-long-chain-with-def using index-injective
    by (auto simp: finite-chain-with-alt local-long-ch-by-ord-def local-ordering-def)
  show f (card X − 1) ∈ X
    using 1 points-in-long-chain(3) by auto
  show f 0 ∈ X f 1 ∈ X
```

using *1 points-in-long-chain* **by** *auto*
qed

lemma *get-fin-long-ch-bounds2*:
assumes *local-long-ch-by-ord f X*
and *finite X*
obtains $x\ y\ z\ n_x\ n_y\ n_z$
where $x{\in}X\ y{\in}X\ z{\in}X$ $[f{\leadsto}X|x..y..z]$ $f\ n_x = x\ f\ n_y = y\ f\ n_z = z$
using *get-fin-long-ch-bounds assms*
by (*meson finite-chain-with-def finite-long-chain-with-alt index-middle-element*)

lemma *long-ch-card-ge3*:
assumes *ch-by-ord f X finite X*
shows *local-long-ch-by-ord f X* $\longleftrightarrow$ *card X* $\geq$ *3*
using *assms ch-by-ord-def local-long-ch-by-ord-def short-ch-card*(*1*) **by** *auto*

lemma *fin-ch-betw2*:
assumes $[f{\leadsto}X|a..c]$ **and** $b{\in}X$
obtains $b{=}a|b{=}c|[a;b;c]$
by (*metis assms finite-long-chain-with-alt finite-long-chain-with-def*)

lemma *chain-bounds-unique*:
assumes $[f{\leadsto}X|a..c]$ $[g{\leadsto}X|x..z]$
shows $(a{=}x \wedge c{=}z) \vee (a{=}z \wedge c{=}x)$
using *assms points-in-chain abc-abc-neq abc-bcd-acd abc-sym*
by (*metis* (*full-types*) *fin-ch-betw2*)

end

29.2 Results for segments, rays and (sub)chains

context *MinkowskiBetweenness* **begin**

lemma *inside-not-bound*:
assumes $[f{\leadsto}X|a..c]$
and $j{<}card\ X$
shows $j{>}0 \Longrightarrow f\ j \neq a$ $j{<}card\ X - 1 \Longrightarrow f\ j \neq c$
using *index-injective2 assms finite-chain-def finite-chain-with-def* **apply** *metis*
using *index-injective2 assms finite-chain-def finite-chain-with-def* **by** *auto*

Converse to Theorem 2(i).

lemma (**in** *MinkowskiBetweenness*) *order-finite-chain-indices*:
assumes *chX*: *local-long-ch-by-ord f X finite X*
and *abc*: $[a;b;c]$
and *ijk*: $f\ i = a\ f\ j = b\ f\ k = c\ i{<}card\ X\ j{<}card\ X\ k{<}card\ X$
shows $i{<}j \wedge j{<}k \vee k{<}j \wedge j{<}i$
by (*metis abc-abc-neq abc-only-cba*(*1*,*2*,*3*) *assms bot-nat-0.extremum linorder-neqE-nat order-finite-chain*)

lemma *order-finite-chain-indices2*:
assumes [*f*⇝*X*|*a*..*c*]
and *f j* = *b j*<*card X*
obtains *0*<*j* ∧ *j*<(*card X* − *1*)|*j*=(*card X* − *1*) ∧ *b*=*c*|*j*=*0* ∧ *b*=*a*
proof −
have *finX*: *finite X*
using *assms*(*3*) *card.infinite gr-implies-not0* **by** *blast*
have *b*∈*X*
using *assms* **unfolding** *chain-defs local-ordering-def*
by (*metis One-nat-def card-2-iff insertI1 insert-commute less-2-cases*)
have *a*: *f 0* = *a* **and** *c*: *f* (*card X* − *1*) = *c*
using *assms*(*1*) *finite-chain-with-def* **by** *auto*

have *0*<*j* ∧ *j*<(*card X* − *1*) ∨ *j*=(*card X* − *1*) ∧ *b*=*c* ∨ *j*=*0* ∧ *b*=*a*
proof (*cases short-ch-by-ord f X*)
case *True*
hence *X*={*a*,*c*}
using *a assms*(*1*) *first-neq-last points-in-chain short-ch-by-ord-def* **by** *fastforce*
then consider *b*=*a*|*b*=*c*
using ‹*b*∈*X*› **by** *fastforce*
thus *?thesis*
apply *cases* **using** *assms*(*2*,*3*) *a c le-less* **by** *fastforce+*
next
case *False*
hence *chX*: *local-long-ch-by-ord f X*
using *assms*(*1*) **unfolding** *finite-chain-with-alt* **using** *chain-defs* **by** *meson*
consider [*a*;*b*;*c*]|*b*=*a*|*b*=*c*
using ‹*b*∈*X*› *assms*(*1*) *fin-ch-betw2* **by** *blast*
thus *?thesis* **apply** *cases*
using ‹*f 0* = *a*› *chX finX assms*(*2*,*3*) *a c order-finite-chain-indices* **apply** *fastforce*
using ‹*f 0* = *a*› *chX finX assms*(*2*,*3*) *index-injective* **apply** *blast*
using *a c assms chX finX index-injective linorder-neqE-nat inside-not-bound*(*2*) **by** *metis*
qed
thus *?thesis* **using** *that* **by** *blast*
qed

lemma *index-bij-betw-subset*:
assumes *chX*: [*f*⇝*X*|*a*..*b*..*c*] *f i* = *b card X* > *i*
shows *bij-betw f* {*0*<..<*i*} {*e*∈*X*. [*a*;*e*;*b*]}
proof (*unfold bij-betw-def*, *intro conjI*)
have *chX2*: *local-long-ch-by-ord f X finite X*
using *assms* **unfolding** *chain-defs* **apply** (*metis chX*(*1*)
abc-ac-neq fin-ch-betw points-in-long-chain(*1*,*3*) *short-ch-alt*(*1*) *short-ch-path*)
using *assms* **unfolding** *chain-defs* **by** *simp*

```
  from index-bij-betw[OF this] have 1: bij-betw f {0..<card X} X .
  have {0<..<i} ⊂ {0..<card X}
    using assms(1,3) unfolding chain-defs by fastforce
  show inj-on f {0<..<i}
    using 1 assms(3) unfolding bij-betw-def
   by (smt (z3) atLeastLessThan-empty-iff2 atLeastLessThan-iff empty-iff greaterThanLessThan-iff
      inj-on-def less-or-eq-imp-le)
  show f ‘ {0<..<i} = {e ∈ X. [a;e;b]}
  proof
    show f ‘ {0<..<i} ⊆ {e ∈ X. [a;e;b]}
    proof (auto simp add: image-subset-iff conjI)
      fix j assume asm: j>0 j<i
      hence j < card X using chX(3) less-trans by blast
      thus f j ∈ X [a;f j;b]
        using chX(1) asm(1) unfolding chain-defs local-ordering-def
      apply (metis chX2(1) chX(1) fin-chain-card-geq-2 short-ch-card-2 short-xor-long(2)
       le-antisym set-le-two finite-chain-def finite-chain-with-def finite-long-chain-with-alt)
      using chX asm chX2(1) order-finite-chain unfolding chain-defs local-ordering-def
by force
    qed
    show {e ∈ X. [a;e;b]} ⊆ f ‘ {0<..<i}
    proof (auto)
      fix e assume e: e ∈ X [a;e;b]
      obtain j where f j = e j<card X
        using e chX2 unfolding chain-defs local-ordering-def by blast
      show e ∈ f ‘ {0<..<i}
      proof
        have 0<j∧j<i ∨ i<j∧j<0
          using order-finite-chain-indices chX chain-defs
           by (smt (z3) ‹f j = e› ‹j < card X› chX2(1) e(2) gr-implies-not-zero
linorder-neqE-nat)
        hence j<i by simp
        thus j∈{0<..<i} e = f j
          using ‹0 < j ∧ j < i ∨ i < j ∧ j < 0› greaterThanLessThan-iff
          by (blast,(simp add: ‹f j = e›))
      qed
    qed
  qed
qed


lemma bij-betw-extend:
  assumes bij-betw f A B
    and f a = b a∉A b∉B
  shows bij-betw f (insert a A) (insert b B)
 by (smt (verit, ccfv-SIG) assms(1) assms(2) assms(4) bij-betwI' bij-betw-iff-bijections
insert-iff)
```

lemma *insert-iff2*:
  assumes *a*∈*X* **shows** *insert a* {*x*∈*X*. *P x*} = {*x*∈*X*. *P x* ∨ *x*=*a*}
  using *insert-iff assms* **by** *blast*

lemma *index-bij-betw-subset2*:
  assumes *chX*: [*f*⇝*X*|*a*..*b*..*c*] *f i* = *b card X* > *i*
  shows *bij-betw f* {*0*..*i*} {*e*∈*X*. [*a*;*e*;*b*]∨*a*=*e*∨*b*=*e*}
proof −
  have *bij-betw f* {*0*<..<*i*} {*e*∈*X*. [*a*;*e*;*b*]} **using** *index-bij-betw-subset*[*OF assms*]
.
  moreover have *0*∉{*0*<..<*i*} *i*∉{*0*<..<*i*} **by** *simp*+
  moreover have *a*∉{*e*∈*X*. [*a*;*e*;*b*]} *b*∉{*e*∈*X*. [*a*;*e*;*b*]} **using** *abc-abc-neq* **by** *auto*+
  moreover have *f 0* = *a f i* = *b* **using** *assms* **unfolding** *chain-defs* **by** *simp*+
  moreover have (*insert b* (*insert a* {*e*∈*X*. [*a*;*e*;*b*]})) = {*e*∈*X*. [*a*;*e*;*b*]∨*a*=*e*∨*b*=*e*}
  proof −
    have *1*: (*insert a* {*e*∈*X*. [*a*;*e*;*b*]}) = {*e*∈*X*. [*a*;*e*;*b*]∨*a*=*e*}
      using *insert-iff2*[*OF points-in-long-chain*(*1*)[*OF chX*(*1*)]] **by** *auto*
    have *b*∉{*e*∈*X*. [*a*;*e*;*b*]∨*a*=*e*}
      using *abc-abc-neq chX*(*1*) *fin-ch-betw* **by** *fastforce*
    thus (*insert b* (*insert a* {*e*∈*X*. [*a*;*e*;*b*]})) = {*e*∈*X*. [*a*;*e*;*b*]∨*a*=*e*∨*b*=*e*}
      using *1 insert-iff2 points-in-long-chain*(*2*)[*OF chX*(*1*)] **by** *auto*
  qed
  moreover have (*insert i* (*insert 0* {*0*<..<*i*})) = {*0*..*i*} **using** *image-Suc-lessThan*
by *auto*
  ultimately show *?thesis* **using** *bij-betw-extend*[*of f*]
    by (*metis* (*no-types*, *lifting*) *chX*(*1*) *finite-long-chain-with-def insert-iff*)
qed

lemma *chain-shortening*:
  assumes [*f*⇝*X*|*a*..*b*..*c*]
  shows [*f* ⇝ {*e*∈*X*. [*a*;*e*;*b*] ∨ *e*=*a* ∨ *e*=*b*} |*a*..*b*]
proof (*unfold finite-chain-with-def finite-chain-def*, (*intro conjI*))

Different forms of assumptions for compatibility with needed antecedents later.

  show *f 0* = *a* **using** *assms* **unfolding** *chain-defs* **by** *simp*
  have *chX*: *local-long-ch-by-ord f X*
    using *assms first-neq-last points-in-long-chain*(*1*,*3*) *short-ch-card*(*1*) *chain-defs*
    by (*metis card2-either-elt1-or-elt2*)
  have *finX*: *finite X*
    by (*meson assms chain-defs*)

General facts about the shortened set, which we will call Y.

  let *?Y* = {*e*∈*X*. [*a*;*e*;*b*] ∨ *e*=*a* ∨ *e*=*b*}
  show *finY*: *finite ?Y*
    using *assms finite-chain-def finite-chain-with-def finite-long-chain-with-alt* **by**
auto

have $a\neq b$ $a\in ?Y$ $b\in ?Y$ $c\notin ?Y$
using *assms finite-long-chain-with-def* **apply** *simp*
using *assms points-in-long-chain(1,2)* **apply** *auto[1]*
using *assms points-in-long-chain(2)* **apply** *auto[1]*
using *abc-ac-neq abc-only-cba(2) assms fin-ch-betw* **by** *fastforce*
from *this(1−3) finY* **have** *cardY*: $card\ ?Y \geq 2$
by (*metis* (*no-types*, *lifting*) *card-le-Suc0-iff-eq not-less-eq-eq numeral-2-eq-2*)

Obtain index for b (a is at index 0): this index i is $card\ ?Y - 1$.

obtain i **where** i: $i<card\ X$ $f\ i=b$
using *assms* **unfolding** *chain-defs local-ordering-def* **using** *Suc-leI diff-le-self*
by *force*
hence $i<card\ X - 1$
using *assms* **unfolding** *chain-defs*
by (*metis Suc-lessI diff-Suc-Suc diff-Suc-eq-diff-pred minus-nat.diff-0 zero-less-diff*)
have *card01*: $i+1 = card\ \{0..i\}$ **by** *simp*
have *bb*: *bij-betw* f $\{0..i\}$ $?Y$ **using** *index-bij-betw-subset2*[*OF assms i*(2,1)]
Collect-cong **by** *smt*
hence *i-eq*: $i = card\ ?Y - 1$ **using** *bij-betw-same-card* **by** *force*
thus $f\ (card\ ?Y - 1) = b$ **using** $i(2)$ **by** *simp*

The path P on which X lies. If $?Y$ has two arguments, P makes it a short chain.

obtain P **where** *P-def*: $P\in\mathcal{P}$ $X\subseteq P$ $\bigwedge Q.\ Q\in\mathcal{P} \wedge X\subseteq Q \Longrightarrow Q=P$
using *fin-chain-on-path*[*of f X*] *assms* **unfolding** *chain-defs* **by** *force*
have $a\in P$ $b\in P$ **using** *P-def* **by** (*meson assms in-mono points-in-long-chain*)+

consider (*eq-1*)$i=1$|(*gt-1*)$i>1$ **using** ‹$a \neq b$› ‹$f\ 0 = a$› $i(2)$ *less-linear* **by** *blast*
thus $[f\leadsto ?Y]$
proof (*cases*)
case *eq-1*
hence $\{0..i\}=\{0,1\}$ **by** *auto*
hence *bij-betw* f $\{0,1\}$ $?Y$ **using** *bb* **by** *auto*
from *bij-betw-imp-surj-on*[*OF this*] **show** *?thesis*
unfolding *chain-defs* **using** *P-def eq-1* ‹$a \neq b$› ‹$f\ 0 = a$› $i(2)$ **by** *blast*
next
case *gt-1*
have 1: $3\leq card\ ?Y$ **using** *gt-1 cardY i-eq* **by** *linarith*
{
fix n **assume** $n < card\ ?Y$
hence $n<card\ X$
using ‹$i<card\ X - 1$› *add-diff-inverse-nat i-eq nat-diff-split-asm* **by** *linarith*
have $f\ n \in ?Y$
proof (*simp*, *intro conjI*)
show $f\ n \in X$
using ‹$n<card\ X$› *assms chX chain-defs local-ordering-def* **by** *metis*
consider $0<n \wedge n<card\ ?Y - 1$|$n=card\ ?Y - 1$|$n=0$
using ‹$n<card\ ?Y$› *nat-less-le zero-less-diff* **by** *linarith*
thus $[a;f\ n;b] \vee f\ n = a \vee f\ n = b$

    using *i i-eq* ‹*f 0 = a*› *chX finX le-numeral-extra*(*3*) *order-finite-chain* **by** *fastforce*
   qed
  } moreover {
   fix *x* **assume** *x*∈*?Y* **hence** *x*∈*X* **by** *simp*
   obtain i_x **where** i_x: i_x < *card X f* i_x = *x*
    using *assms obtain-index-fin-chain chain-defs* ‹*x*∈*X*› **by** *metis*
   have i_x < *card ?Y*
   proof −
    consider [*a*;*x*;*b*]|*x*=*a*|*x*=*b* **using** ‹*x*∈*?Y*› **by** *auto*
    hence (i_x<*i* ∨ i_x<*0*) ∨ i_x=*0* ∨ i_x=*i*
     apply *cases*
      apply (*metis* ‹*f 0*=*a*› *chX finX i* i_x *less-nat-zero-code neq0-conv order-finite-chain-indices*)
     using ‹*f 0 = a*› *chX finX* i_x *index-injective* **apply** *blast*
     by (*metis chX finX i*(*2*) i_x *index-injective linorder-neqE-nat*)
    thus *?thesis* **using** *gt-1 i-eq* **by** *linarith*
   qed
   hence ∃ *n*. *n* < *card ?Y* ∧ *f n* = *x* **using** i_x(*2*) **by** *blast*
  } moreover {
   fix *n* **assume** *Suc* (*Suc n*) < *card ?Y*
   hence *Suc* (*Suc n*) < *card X*
    using *i*(*1*) *i-eq* **by** *linarith*
   hence [*f n*; *f* (*Suc n*); *f* (*Suc* (*Suc n*))]
    using *assms* **unfolding** *chain-defs local-ordering-def* **by** *auto*
  }
  ultimately have *2*: *local-ordering f betw ?Y*
   by (*simp add*: *local-ordering-def finY*)
  show *?thesis* **using** *1 2 chain-defs* **by** *blast*
 qed
qed

corollary *ord-fin-ch-right*:
 assumes [*f*⇝*X*|*a*..*f i*..*c*] *j*≥*i j*<*card X*
 shows [*f i*;*f j*;*c*] ∨ *j* = *card X* − *1* ∨ *j* = *i*
proof −
 consider (*inter*)*j*>*i* ∧ *j*<*card X* − *1*|(*left*)*j*=*i*|(*right*)*j*=*card X* − *1*
  using *assms*(*3*,*2*) **by** *linarith*
 thus *?thesis*
  apply *cases*
  using *assms*(*1*) *chain-defs order-finite-chain2* **apply** *force*
  by *simp*+
qed

lemma *f-img-is-subset*:
 assumes [*f*⇝*X*|(*f 0*) ..] *i*≥*0 j*>*i Y*=*f*‘{*i*..*j*}
 shows *Y*⊆*X*
proof

fix *x* **assume** *x*∈*Y*
then obtain *n* **where** *n*∈{*i*..*j*} *f n* = *x*
using *assms*(*4*) **by** *blast*
hence *f n* ∈ *X*
by (*metis local-ordering-def assms*(*1*) *inf-chain-is-long local-long-ch-by-ord-def*)
thus *x*∈*X*
using ‹*f n* = *x*› **by** *blast*
qed

lemma *i-le-j-events-neq*:
assumes [*f*↝*X*|*a*..*b*..*c*]
and *i*<*j* *j*<*card X*
shows *f i* ≠ *f j*
using *chain-defs* **by** (*meson assms index-injective2*)

lemma *indices-neq-imp-events-neq*:
assumes [*f*↝*X*|*a*..*b*..*c*]
and *i*≠*j* *j*<*card X* *i*<*card X*
shows *f i* ≠ *f j*
by (*metis assms i-le-j-events-neq less-linear*)

end

context *MinkowskiSpacetime* **begin**

lemma *bound-on-path*:
assumes *Q*∈𝒫 [*f*↝*X*|(*f 0*)..] *X*⊆*Q* *is-bound-f b X f*
shows *b*∈*Q*
proof −
obtain *a c* **where** *a*∈*X* *c*∈*X* [*a*;*c*;*b*]
using *assms*(*4*)
by (*metis local-ordering-def inf-chain-is-long is-bound-f-def local-long-ch-by-ord-def zero-less-one*)
thus *?thesis*
using *abc-abc-neq assms*(*1*) *assms*(*3*) *betw-c-in-path* **by** *blast*
qed

lemma *pro-basis-change*:
assumes [*a*;*b*;*c*]
shows *prolongation a c* = *prolongation b c* (**is** *?ac*=*?bc*)
proof
show *?ac* ⊆ *?bc*
proof
fix *x* **assume** *x*∈*?ac*
hence [*a*;*c*;*x*]
by (*simp add*: *pro-betw*)
hence [*b*;*c*;*x*]
using *assms abc-acd-bcd* **by** *blast*

```
    thus x∈?bc
      using abc-abc-neq pro-betw by blast
  qed
  show ?bc ⊆ ?ac
  proof
    fix x assume x∈?bc
    hence [b;c;x]
      by (simp add: pro-betw)
    hence [a;c;x]
      using assms abc-bcd-acd by blast
    thus x∈?ac
      using abc-abc-neq pro-betw by blast
  qed
qed

lemma adjoining-segs-exclusive:
  assumes [a;b;c]
  shows segment a b ∩ segment b c = {}
proof (cases)
  assume segment a b = {} thus ?thesis by blast
next
  assume segment a b ≠ {}
  have x∈segment a b ⟶ x∉segment b c for x
  proof
    fix x assume x∈segment a b
    hence [a;x;b] by (simp add: seg-betw)
    have ¬[a;b;x] by (meson ‹[a;x;b]› abc-only-cba)
    have ¬[b;x;c]
      using ‹¬ [a;b;x]› abd-bcd-abc assms by blast
    thus x∉segment b c
      by (simp add: seg-betw)
  qed
  thus ?thesis by blast
qed

end
```

30 3.6 Order on a path - Theorems 10 and 11

```
context MinkowskiSpacetime begin
```

30.1 Theorem 10 (based on Veblen (1904) theorem 10).

```
lemma (in MinkowskiBetweenness) two-event-chain:
  assumes finiteX: finite X
      and path-Q: Q ∈ 𝒫
      and events-X: X ⊆ Q
      and card-X: card X = 2
    shows ch X
```

```
proof −
  obtain a b where X-is: X={a,b}
    using card-le-Suc-iff numeral-2-eq-2
    by (meson card-2-iff card-X)
  have no-c: ¬(∃ c∈{a,b}. c≠a ∧ c≠b)
    by blast
  have a≠b ∧ a∈Q & b∈Q
    using X-is card-X events-X by force
  hence short-ch {a,b}
    using path-Q no-c by (meson short-ch-intros(2))
  thus ?thesis
    by (simp add: X-is chain-defs)
qed

lemma (in MinkowskiBetweenness) three-event-chain:
  assumes finiteX: finite X
      and path-Q: Q ∈ 𝒫
      and events-X: X ⊆ Q
      and card-X: card X = 3
    shows ch X
proof −
  obtain a b c where X-is: X={a,b,c}
    using numeral-3-eq-3 card-X by (metis card-Suc-eq)
  then have all-neq: a≠b ∧ a≠c ∧ b≠c
    using card-X numeral-2-eq-2 numeral-3-eq-3
    by (metis Suc-n-not-le-n insert-absorb2 insert-commute set-le-two)
  have in-path: a∈Q ∧ b∈Q ∧ c∈Q
    using X-is events-X by blast
  hence [a;b;c] ∨ [b;c;a] ∨ [c;a;b]
    using some-betw all-neq path-Q by auto
  thus ch X
    using between-chain X-is all-neq chain3 in-path path-Q by auto
qed
```

This is case (i) of the induction in Theorem 10.

```
lemma chain-append-at-left-edge:
  assumes long-ch-Y: [f↝Y|a₁..a..aₙ]
      and bY: [b; a₁; aₙ]
    fixes g defines g-def: g ≡ (λj::nat. if j≥1 then f (j−1) else b)
    shows [g↝(insert b Y)|b .. a₁ .. aₙ]
proof −
  let ?X = insert b Y
  have ord-fY: local-ordering f betw Y using long-ch-Y finite-long-chain-with-card
chain-defs
    by (meson long-ch-card-ge3)
  have b∉Y
   using abc-ac-neq abc-only-cba(1) assms by (metis fin-ch-betw2 finite-long-chain-with-alt)
  have bound-indices: f 0 = a₁ ∧ f (card Y − 1) = aₙ
    using long-ch-Y by (simp add: chain-defs)
```

have *fin-Y*: *card Y* $\geq$ *3*
using *finite-long-chain-with-def long-ch-Y numeral-2-eq-2 points-in-long-chain*
by (*metis abc-abc-neq bY card2-either-elt1-or-elt2 fin-chain-card-geq-2 leI le-less-Suc-eq numeral-3-eq-3*)
hence *num-ord*: $0 \leq (0::nat) \wedge 0<(1::nat) \wedge 1 < card\ Y - 1 \wedge card\ Y - 1 < card\ Y$
by *linarith*
hence $[a_1; f\ 1; a_n]$
using *order-finite-chain chain-defs long-ch-Y*
by *auto*

Schutz has a step here that says $[ba_1a_2a_n]$ is a chain (using Theorem 9). We have no easy way (yet) of denoting an ordered 4-element chain, so we skip this step using a *local-ordering* lemma from our script for 3.6, which Schutz doesn't list.

hence $[b; a_1; f\ 1]$
using *bY abd-bcd-abc* **by** *blast*
have *local-ordering g betw ?X*
proof −
{
fix *n* **assume** *finite ?X* $\longrightarrow$ *n<card ?X*
have *g n* $\in$ *?X*
apply (*cases n*$\geq$*1*)
prefer *2* **apply** (*simp add*: *g-def*)
proof
assume *1*$\leq$*n g n* $\notin$ *Y*
hence *g n* = *f*(*n*−*1*) **unfolding** *g-def* **by** *auto*
hence *g n* $\in$ *Y*
proof (*cases n* = *card ?X* − *1*)
case *True*
thus *?thesis*
using ‹*b*$\notin$*Y*› *card.insert diff-Suc-1 long-ch-Y points-in-long-chain chain-defs*
by (*metis* ‹*g n* = *f* (*n* − *1*)›)
next
case *False*
hence *n* < *card Y*
using *points-in-long-chain* ‹*finite ?X* $\longrightarrow$ *n* < *card ?X*› ‹*g n* = *f* (*n* − *1*)› ‹*g n* $\notin$ *Y*› ‹*b*$\notin$*Y*› *chain-defs*
by (*metis card.insert finite-insert long-ch-Y not-less-simps*(*1*))
hence *n*−*1* < *card Y* − *1*
using ‹*1* $\leq$ *n*› *diff-less-mono* **by** *blast*
hence *f*(*n*−*1*)$\in$*Y*
using *long-ch-Y fin-Y* **unfolding** *chain-defs local-ordering-def*
by (*metis Suc-le-D card-3-dist diff-Suc-1 insert-absorb2 le-antisym less-SucI numeral-3-eq-3 set-le-three*)
thus *?thesis*
using ‹*g n* = *f* (*n* − *1*)› **by** *presburger*
qed

hence *False* **using** ‹$g\ n \notin Y$› **by** *auto*
thus $g\ n = b$ **by** *simp*
qed
} moreover {
fix *n* **assume** (*finite ?X* ⟶ *Suc*(*Suc n*) < *card ?X*)
hence [*g n*; *g* (*Suc n*); *g* (*Suc*(*Suc n*))]
apply (*cases n≥1*)
using ‹*b∉Y*› ‹[*b*; a_1; *f 1*]› *g-def ordering-ord-ijk-loc*[*OF ord-fY*] *fin-Y*
apply (*metis Suc-diff-le card-insert-disjoint diff-Suc-1 finite-insert le-Suc-eq not-less-eq*)
by (*metis One-nat-def Suc-leI* ‹[*b*;a_1;*f 1*]› *bound-indices diff-Suc-1 g-def not-less-less-Suc-eq zero-less-Suc*)
} moreover {
fix *x* **assume** *x∈?X x=b*
have (*finite ?X* ⟶ *0* < *card ?X*) ∧ *g 0* = *x*
by (*simp add*: ‹*b∉Y*› ‹*x* = *b*› *g-def*)
} moreover {
fix *x* **assume** *x∈?X x≠b*
hence ∃ *n*. (*finite ?X* ⟶ *n* < *card ?X*) ∧ *g n* = *x*
proof −
obtain *n* **where** *f n* = *x n* < *card Y*
using ‹*x∈?X*› ‹*x≠b*› *local-ordering-def insert-iff long-ch-Y chain-defs* **by** (*metis ord-fY*)
have (*finite ?X* ⟶ *n+1* < *card ?X*) *g*(*n+1*) = *x*
apply (*simp add*: ‹*b∉Y*› ‹*n* < *card Y*›)
by (*simp add*: ‹*f n* = *x*› *g-def*)
thus *?thesis* **by** *auto*
qed
}
ultimately show *?thesis*
unfolding *local-ordering-def*
by *smt*
qed
hence *local-long-ch-by-ord g ?X*
unfolding *local-long-ch-by-ord-def*
using *fin-Y* ‹*b∉Y*›
by (*meson card-insert-le finite-insert le-trans*)
show *?thesis*
proof (*intro finite-long-chain-with-alt2*)
show *local-long-ch-by-ord g ?X* **using** ‹*local-long-ch-by-ord g ?X*› **by** *simp*
show [*b*;a_1;a_n] ∧ a_1 ∈ *?X* **using** *bY long-ch-Y points-in-long-chain*(*1*) **by** *auto*
show *g 0* = *b* **using** *g-def* **by** *simp*
show *finite ?X*
using *fin-Y* ‹*b∉Y*› *eval-nat-numeral* **by** (*metis card.infinite finite.insertI not-numeral-le-zero*)
show *g* (*card ?X* − *1*) = a_n
using *g-def* ‹*b∉Y*› *bound-indices eval-nat-numeral*
by (*metis One-nat-def card.infinite card-insert-disjoint diff-Suc-Suc diff-is-0-eq′ less-nat-zero-code minus-nat.diff-0 nat-le-linear num-ord*)

qed
qed

This is case (iii) of the induction in Theorem 10. Schutz says merely "The proof for this case is similar to that for Case (i)." Thus I feel free to use a result on symmetry, rather than going through the pain of Case (i) (*chain-append-at-left-edge*) again.

lemma *chain-append-at-right-edge*:
  assumes *long-ch-Y*: $[f{\leadsto}Y|a_1..a..a_n]$
    and *Yb*: $[a_1;\ a_n;\ b]$
  fixes *g* **defines** *g-def*: $g \equiv (\lambda j{::}nat.\ if\ j \le (card\ Y - 1)\ then\ f\ j\ else\ b)$
  shows $[g{\leadsto}(insert\ b\ Y)|a_1\ ..\ a_n\ ..\ b]$
proof −
  let $?X = insert\ b\ Y$
  have $b{\notin}Y$
    using *Yb abc-abc-neq abc-only-cba*(*2*) *long-ch-Y*
    by (*metis fin-ch-betw2 finite-long-chain-with-def*)
  have *fin-Y*: $card\ Y \ge 3$
    using *finite-long-chain-with-card long-ch-Y* **by** *auto*
  hence *fin-X*: *finite ?X*
    by (*metis card.infinite finite.insertI not-numeral-le-zero*)
  have $a_1{\in}Y \wedge a_n{\in}Y \wedge a{\in}Y$
    using *long-ch-Y points-in-long-chain* **by** *meson*
  have $a_1{\neq}a \wedge a{\neq}\ a_n \wedge a_1{\neq}a_n$
    using *Yb abc-abc-neq finite-long-chain-with-def long-ch-Y* **by** *auto*
  have $Suc\ (card\ Y) = card\ ?X$
    using ‹$b{\notin}Y$› *fin-X finite-long-chain-with-def long-ch-Y* **by** *auto*
  obtain *f2* **where** *f2-def*: $[f2{\leadsto}Y|a_n..a..a_1]\ f2{=}(\lambda n.\ f\ (card\ Y - 1 - n))$
    using *chain-sym long-ch-Y* **by** *blast*
  obtain *g2* **where** *g2-def*: $g2 = (\lambda j{::}nat.\ if\ j{\ge}1\ then\ f2\ (j{-}1)\ else\ b)$
    by *simp*
  have $[b;\ a_n;\ a_1]$
    using *abc-sym Yb* **by** *blast*
  hence *g2-ord-X*: $[g2{\leadsto}?X|b\ ..\ a_n\ ..\ a_1]$
    using *chain-append-at-left-edge* [**where** $a_1{=}a_n$ **and** $a_n{=}a_1$ **and** $f{=}f2$]
      fin-X ‹$b{\notin}Y$› *f2-def g2-def*
    by *blast*
  then obtain *g1* **where** *g1-def*: $[g1{\leadsto}?X|a_1..a_n..b]\ g1{=}(\lambda n.\ g2\ (card\ ?X - 1 -$
$n))$
    using *chain-sym* **by** *blast*
  have *sYX*: $(card\ Y) = (card\ ?X) - 1$
    using *assms*(*2*,*3*) *finite-long-chain-with-def long-ch-Y* ‹$Suc\ (card\ Y) = card$
$?X$› **by** *linarith*
  have $g1{=}g$
    unfolding *g1-def g2-def f2-def g-def*
  proof
    fix *n*
    show (
        $if\ 1 \le card\ ?X - 1 - n\ then$

```
        f (card Y − 1 − (card ?X − 1 − n − 1))
       else b
      ) = (
       if n ≤ card Y − 1 then
        f n
       else b
      ) (is ?lhs=?rhs)
  proof (cases)
   assume n ≤ card ?X − 2
   show ?lhs=?rhs
     using ‹n ≤ card ?X − 2› finite-long-chain-with-def long-ch-Y sYX ‹Suc
(card Y) = card ?X›
   by (metis (mono-tags, opaque-lifting) Suc-1 Suc-leD diff-Suc-Suc diff-commute
diff-diff-cancel
     diff-le-mono2 fin-chain-card-geq-2)
  next
   assume ¬ n ≤ card ?X − 2
   thus ?lhs=?rhs
     by (metis ‹Suc (card Y) = card ?X› Suc-1 diff-Suc-1 diff-Suc-eq-diff-pred
diff-diff-cancel
       diff-is-0-eq′ nat-le-linear not-less-eq-eq)
  qed
 qed
 thus ?thesis
  using g1-def(1) by blast
qed


lemma S-is-dense:
 assumes long-ch-Y: [f⇝Y|a₁..a..aₙ]
   and S-def: S = {k::nat. [a₁; f k; b] ∧ k < card Y}
   and k-def: S≠{} k = Max S
   and k′-def: k′>0 k′<k
 shows k′ ∈ S
proof −
```

We will prove this by contradiction. We can obtain the path that Y lies on, and show b is on it too. Then since f‘S must be on this path, there must be an ordering involving b, $f\ k$ and $f\ k'$ that leads to contradiction with the definition of S and $k \notin S$. Notice we need no knowledge about b except how it relates to S.

```
 have [f⇝Y] using long-ch-Y chain-defs by meson
 have card Y ≥ 3 using finite-long-chain-with-card long-ch-Y by blast
 hence finite Y by (metis card.infinite not-numeral-le-zero)
 have k∈S using k-def Max-in S-def by (metis finite-Collect-conjI finite-Collect-less-nat)
 hence k<card Y using S-def by auto
 have k′<card Y using S-def k′-def ‹k∈S› by auto
 show k′ ∈ S
 proof (rule ccontr)
```

assume *asm*: $\neg k' \in S$
have *1*: $[f\ 0;f\ k;f\ k']$
proof −
have $[a_1;\ b;\ f\ k']$
using *order-finite-chain2 long-ch-Y* ‹$k \in S$› ‹$k' <$ *card Y*› *chain-defs*
by (*smt* (*z3*) *abc-acd-abd asm le-numeral-extra*(*3*) *assms mem-Collect-eq*)
have $[a_1;\ f\ k;\ b]$
using *S-def* ‹$k \in S$› **by** *blast*
have $[f\ k;\ b;\ f\ k']$
using *abc-acd-bcd* ‹$[a_1;\ b;\ f\ k']$› ‹$[a_1;\ f\ k;\ b]$› **by** *blast*
thus *?thesis*
using ‹$[a_1;f\ k;b]$› *long-ch-Y* **unfolding** *finite-long-chain-with-def finite-chain-with-def*
by *blast*
qed
have *2*: $[f\ 0;f\ k';f\ k]$
apply (*intro order-finite-chain2*[*OF* ‹$[f \leadsto Y]$› ‹*finite Y*›]) **by** (*simp add*: ‹$k <$ *card Y*› *k'-def*)
show *False* **using** *1 2 abc-only-cba*(*2*) **by** *blast*
qed
qed

lemma *smallest-k-ex*:
assumes *long-ch-Y*: $[f \leadsto Y|a_1..a..a_n]$
and *Y-def*: $b \notin Y$
and *Yb*: $[a_1;\ b;\ a_n]$
shows $\exists k>0.\ [a_1;\ b;\ f\ k] \wedge k < card\ Y \wedge \neg(\exists k'<k.\ [a_1;\ b;\ f\ k'])$
proof −

have *bound-indices*: $f\ 0 = a_1 \wedge f\ (card\ Y - 1) = a_n$
using *chain-defs long-ch-Y* **by** *auto*
have *fin-Y*: *finite Y*
using *chain-defs long-ch-Y* **by** *presburger*
have *card-Y*: $card\ Y \geq 3$
using *long-ch-Y points-in-long-chain finite-long-chain-with-card* **by** *blast*

We consider all indices of chain elements between a_1 and b, and find the maximal one.

let $?S = \{k::nat.\ [a_1;\ f\ k;\ b] \wedge k < card\ Y\}$
obtain *S* **where** *S-def*: $S=?S$
by *simp*
have $S \subseteq \{0..card\ Y\}$
using *S-def* **by** *auto*
hence *finite S*
using *finite-subset* **by** *blast*

show *?thesis*
proof (*cases*)
assume $S=\{\}$

show *?thesis*
proof
show $(0::nat)<1 \wedge [a_1; b; f\ 1] \wedge 1 < card\ Y \wedge \neg (\exists k'::nat.\ k' < 1 \wedge [a_1; b; f\ k'])$
proof (*intro conjI*)
show $(0::nat)<1$ **by** *simp*
show $1 < card\ Y$
using *Yb abc-ac-neq bound-indices not-le* **by** *fastforce*
show $\neg (\exists k'::nat.\ k' < 1 \wedge [a_1; b; f\ k'])$
using *abc-abc-neq bound-indices*
by *blast*
show $[a_1; b; f\ 1]$
proof −
have $f\ 1 \in Y$
using *long-ch-Y chain-defs local-ordering-def* **by** (*metis* ‹$1 < card\ Y$› *short-ch-ord-in*(*2*))
hence $[a_1; f\ 1; a_n]$
using *bound-indices long-ch-Y chain-defs local-ordering-def card-Y*
by (*smt* (*z3*) *Nat.lessE One-nat-def Suc-le-lessD Suc-lessD diff-Suc-1 diff-Suc-less*
fin-ch-betw2 i-le-j-events-neq less-numeral-extra(*1*) *numeral-3-eq-3*)
hence $[a_1; b; f\ 1] \vee [a_1; f\ 1; b] \vee [b; a_1; f\ 1]$
using *abc-ex-path-unique some-betw abc-sym*
by (*smt Y-def Yb* ‹$f\ 1 \in Y$› *abc-abc-neq cross-once-notin*)
thus $[a_1; b; f\ 1]$
proof −
have $\forall n.\ \neg ([a_1; f\ n; b] \wedge n < card\ Y)$
using *S-def* ‹$S = \{\}$›
by *blast*
then have $[a_1; b; f\ 1] \vee \neg [a_n; f\ 1; b] \wedge \neg [a_1; f\ 1; b]$
using *bound-indices abc-sym abd-bcd-abc Yb*
by (*metis* (*no-types*) *diff-is-0-eq′ nat-le-linear nat-less-le*)
then show *?thesis*
using *abc-bcd-abd abc-sym*
by (*meson* ‹$[a_1; b; f\ 1] \vee [a_1; f\ 1; b] \vee [b; a_1; f\ 1]$› ‹$[a_1; f\ 1; a_n]$›)
qed
qed
qed
qed
next assume $\neg S=\{\}$
obtain k **where** $k = Max\ S$
by *simp*
hence $k \in S$ **using** *Max-in*
by (*simp add*: ‹$S \neq \{\}$› ‹*finite S*›)
have $k \geq 1$
proof (*rule ccontr*)
assume $\neg\ 1 \leq k$
hence $k=0$ **by** *simp*
have $[a_1; f\ k; b]$

```
      using ‹k∈S› S-def
      by blast
    thus False
      using bound-indices ‹k = 0› abc-abc-neq
      by blast
  qed

  show ?thesis
  proof
    let ?k = k+1
    show 0<?k ∧ [a₁; b; f ?k] ∧ ?k < card Y ∧ ¬ (∃k'::nat. k' < ?k ∧ [a₁; b; f
k'])
    proof (intro conjI)
      show (0::nat)<?k by simp
      show ?k < card Y
       by (metis (no-types, lifting) S-def Yb ‹k ∈ S› abc-only-cba(2) add.commute
         add-diff-cancel-right' bound-indices less-SucE mem-Collect-eq nat-add-left-cancel-less
            plus-1-eq-Suc)
      show [a₁; b; f ?k]
      proof −
        have f ?k ∈ Y
          using ‹k + 1 < card Y› long-ch-Y card-Y unfolding local-ordering-def
chain-defs
          by (metis One-nat-def Suc-numeral not-less-eq-eq numeral-3-eq-3 numer-
als(1) semiring-norm(2) set-le-two)
        have [a₁; f ?k; aₙ] ∨ f ?k = aₙ
          using fin-ch-betw2 inside-not-bound(1) long-ch-Y chain-defs
          by (metis ‹0 < k + 1› ‹k + 1 < card Y› ‹f (k + 1) ∈ Y›)
        thus  [a₁; b; f ?k]
        proof (rule disjE)
          assume [a₁; f ?k; aₙ]
          hence f ?k ≠ aₙ
            by (simp add: abc-abc-neq)
          hence [a₁; b; f ?k] ∨ [a₁; f ?k; b] ∨ [b; a₁; f ?k]
            using abc-ex-path-unique some-betw abc-sym ‹[a₁; f ?k; aₙ]›
              ‹f ?k ∈ Y› Yb abc-abc-neq assms(3) cross-once-notin
            by (smt Y-def)
          moreover have ¬ [a₁; f ?k; b]
          proof
            assume [a₁; f ?k; b]
            hence ?k ∈ S
              using S-def ‹[a₁; f ?k; b]› ‹k + 1 < card Y› by blast
            hence ?k ≤ k
              by (simp add: ‹finite S› ‹k = Max S›)
            thus False
              by linarith
          qed
          moreover have ¬ [b; a₁; f ?k]
            using Yb ‹[a₁; f ?k; aₙ]› abc-only-cba
```

```
            by blast
          ultimately show [a₁; b; f ?k]
            by blast
        next assume f ?k = aₙ
          show ?thesis
            using Yb ‹f (k + 1) = aₙ› by blast
        qed
      qed
      show ¬(∃k'::nat. k' < k + 1 ∧ [a₁; b; f k'])
      proof
        assume ∃k'::nat. k' < k + 1 ∧ [a₁; b; f k']
        then obtain k' where k'-def: k'>0 k' < k + 1 [a₁; b; f k']
          using abc-ac-neq bound-indices neq0-conv
          by blast
        hence k'<k
          using S-def ‹k ∈ S› abc-only-cba(2) less-SucE by fastforce
        hence k'∈S
          using S-is-dense long-ch-Y S-def ‹¬S={}› ‹k = Max S› ‹k'>0›
          by blast
        thus False
          using S-def abc-only-cba(2) k'-def(3) by blast
      qed
    qed
  qed
 qed
qed
```

```
lemma greatest-k-ex:
  assumes long-ch-Y: [f↝Y|a₁..a..aₙ]
      and Y-def: b∉Y
      and Yb: [a₁; b; aₙ]
    shows ∃k. [f k; b; aₙ] ∧ k < card Y − 1 ∧ ¬(∃k'<card Y. k'>k ∧ [f k'; b; aₙ])
proof −
  have bound-indices: f 0 = a₁ ∧ f (card Y − 1) = aₙ
    using chain-defs long-ch-Y by simp
  have fin-Y: finite Y
    using chain-defs long-ch-Y by presburger
  have card-Y: card Y ≥ 3
    using long-ch-Y points-in-long-chain finite-long-chain-with-card by blast
  have chY2: local-long-ch-by-ord f Y
    using long-ch-Y chain-defs by (meson card-Y long-ch-card-ge3)
```

Again we consider all indices of chain elements between a_1 and b.

```
  let ?S = {k::nat. [aₙ; f k; b] ∧ k < card Y}
  obtain S where S-def: S=?S
    by simp
  have S⊆{0..card Y}
```

using *S-def* **by** *auto*
hence *finite S*
using *finite-subset* **by** *blast*

show *?thesis*
proof (*cases*)
assume *S*={}
show *?thesis*
proof
let *?n* = *card Y* − *2*
show [*f ?n*; *b*; a_n] ∧ *?n* < *card Y* − *1* ∧ ¬(∃ *k′*<*card Y*. *k′*>*?n* ∧ [*f k′*; *b*; a_n])
proof (*intro conjI*)
show *?n* < *card Y* − *1*
using *Yb abc-ac-neq bound-indices not-le* **by** *fastforce*
next show ¬(∃ *k′*<*card Y*. *k′*>*?n* ∧ [*f k′*; *b*; a_n])
using *abc-abc-neq bound-indices*
by (*metis One-nat-def Suc-diff-le Suc-leD Suc-lessI card-Y diff-Suc-1 diff-Suc-Suc*
not-less-eq numeral-2-eq-2 numeral-3-eq-3)
next show [*f ?n*; *b*; a_n]
proof −
have [*f 0*;*f ?n*; *f* (*card Y* − *1*)]
apply (*intro order-finite-chain*[*of f Y*], (*simp-all add*: *chY2 fin-Y*))
using *card-Y* **by** *linarith*
hence [a_1; *f ?n*; a_n]
using *long-ch-Y* **unfolding** *chain-defs* **by** *simp*
have *f ?n* ∈ *Y*
using *long-ch-Y eval-nat-numeral* **unfolding** *local-ordering-def chain-defs*
by (*metis card-1-singleton-iff card-Suc-eq card-gt-0-iff diff-Suc-less diff-self-eq-0 insert-iff numeral-2-eq-2*)
hence [a_n; *b*; *f ?n*] ∨ [a_n; *f ?n*; *b*] ∨ [*b*; a_n; *f ?n*]
using *abc-ex-path-unique some-betw abc-sym* ‹[a_1; *f ?n*; a_n]›
by (*smt Y-def Yb* ‹*f ?n* ∈ *Y*› *abc-abc-neq cross-once-notin*)
thus [*f ?n*; *b*; a_n]
proof −
have ∀ *n*. ¬ ([a_n; *f n*; *b*] ∧ *n* < *card Y*)
using *S-def* ‹*S* = {}›
by *blast*
then have [a_n; *b*; *f ?n*] ∨ ¬ [a_1; *f ?n*; *b*] ∧ ¬ [a_n; *f ?n*; *b*]
using *bound-indices abc-sym abd-bcd-abc Yb*
by (*metis* (*no-types*, *lifting*) ‹*f* (*card Y* − *2*) ∈ *Y*› *card-gt-0-iff diff-less empty-iff fin-Y zero-less-numeral*)
then show *?thesis*
using *abc-bcd-abd abc-sym*
by (*meson* ‹[a_n; *b*; *f ?n*] ∨ [a_n; *f ?n*; *b*] ∨ [*b*; a_n; *f ?n*]› ‹[a_1; *f ?n*; a_n]›)
qed
qed
qed

qed
next assume $\neg S=\{\}$
obtain k **where** $k = Min\ S$
by *simp*
hence $k \in S$
by (*simp add*: ‹$S \neq \{\}$› ‹*finite S*›)

show *?thesis*
proof
let $?k = k-1$
show $[f\ ?k;\ b;\ a_n] \wedge ?k < card\ Y - 1 \wedge \neg (\exists k'<card\ Y.\ ?k < k' \wedge [f\ k';\ b;\ a_n])$
proof (*intro conjI*)
show $?k < card\ Y - 1$
using *S-def* ‹$k \in S$› *less-imp-diff-less card-Y*
by (*metis* (*no-types*, *lifting*) *One-nat-def diff-is-0-eq′ diff-less-mono lessI less-le-trans mem-Collect-eq nat-le-linear numeral-3-eq-3 zero-less-diff*)
show $[f\ ?k;\ b;\ a_n]$
proof −
have $f\ ?k \in Y$
using ‹$k - 1 < card\ Y - 1$› *long-ch-Y card-Y eval-nat-numeral* **unfolding** *local-ordering-def chain-defs*
by (*metis Suc-pred′ less-Suc-eq less-nat-zero-code not-less-eq not-less-eq-eq set-le-two*)
have $[a_1;\ f\ ?k;\ a_n] \vee f\ ?k = a_1$
using *bound-indices long-ch-Y* ‹$k - 1 < card\ Y - 1$› *chain-defs*
unfolding *finite-long-chain-with-alt*
by (*metis* ‹$f\ (k - 1) \in Y$› *card-Diff1-less card-Diff-singleton-if chY2 index-injective*)
thus $[f\ ?k;\ b;\ a_n]$
proof (*rule disjE*)
assume $[a_1;\ f\ ?k;\ a_n]$
hence $f\ ?k \neq a_1$
using *abc-abc-neq* **by** *blast*
hence $[a_n;\ b;\ f\ ?k] \vee [a_n;\ f\ ?k;\ b] \vee [b;\ a_n;\ f\ ?k]$
using *abc-ex-path-unique some-betw abc-sym* ‹$[a_1;\ f\ ?k;\ a_n]$›
‹$f\ ?k \in Y$› *Yb abc-abc-neq assms*(*3*) *cross-once-notin*
by (*smt Y-def*)
moreover have $\neg [a_n;\ f\ ?k;\ b]$
proof
assume $[a_n;\ f\ ?k;\ b]$
hence $?k \in S$
using *S-def* ‹$[a_n;\ f\ ?k;\ b]$› ‹$k - 1 < card\ Y - 1$›
by *simp*
hence $?k \geq k$
by (*simp add*: ‹*finite S*› ‹$k = Min\ S$›)
thus *False*
using ‹$f\ (k - 1) \neq a_1$› *chain-defs long-ch-Y*

```
          by auto
        qed
        moreover have ¬ [b; a_n; f ?k]
          using Yb ‹[a_1; f ?k; a_n]› abc-only-cba(2) abc-bcd-acd
          by blast
        ultimately show [f ?k; b; a_n]
          using abc-sym by auto
      next assume f ?k = a_1
        show ?thesis
          using Yb ‹f (k − 1) = a_1› by blast
      qed
    qed
    show ¬(∃k'<card Y. k−1 < k' ∧ [f k'; b; a_n])
    proof
      assume ∃k'<card Y. k−1 < k' ∧ [f k'; b; a_n]
      then obtain k' where k'-def: k'<card Y −1 k' > k − 1 [a_n; b; f k']
        using abc-ac-neq bound-indices neq0-conv
        by (metis Suc-diff-1 abc-sym gr-implies-not0 less-SucE)
      hence k'>k
        using S-def ‹k ∈ S› abc-only-cba(2) less-SucE
       by (metis (no-types, lifting) add-diff-inverse-nat less-one mem-Collect-eq
            not-less-eq plus-1-eq-Suc)thm S-is-dense
      hence k'∈S
        apply (intro S-is-dense[of f Y a_1 a a_n - b Max S])
        apply (simp add: long-ch-Y)
        apply (smt (verit, ccfv-SIG) S-def ‹k ∈ S› abc-acd-abd abc-only-cba(4)
           add-diff-inverse-nat bound-indices chY2 diff-add-zero diff-is-0-eq fin-Y
k'-def(1,3)
         less-add-one less-diff-conv2 less-nat-zero-code mem-Collect-eq nat-diff-split
order-finite-chain)
        apply (simp add: ‹S ≠ {}›, simp, simp)
        using k'-def S-def
     by (smt (verit, ccfv-SIG) ‹k ∈ S› abc-acd-abd abc-only-cba(4) add-diff-cancel-right'
       add-diff-inverse-nat bound-indices chY2 fin-Y le-eq-less-or-eq less-nat-zero-code
          mem-Collect-eq nat-diff-split nat-neq-iff order-finite-chain zero-less-diff
zero-less-one)
      thus False
        using S-def abc-only-cba(2) k'-def(3)
        by blast
    qed
  qed
 qed
qed
qed
```

lemma *get-closest-chain-events*:
  assumes *long-ch-Y*: $[f \leadsto Y|a_0..a..a_n]$
    and *x-def*: $x \notin Y$ $[a_0;\ x;\ a_n]$

obtains n_b n_c b c
where $b=f\ n_b$ $c=f\ n_c$ $[b;x;c]$ $b\in Y$ $c\in Y$ $n_b = n_c - 1$ $n_c<card\ Y$ $n_c>0$
$\neg(\exists k < card\ Y.\ [f\ k;\ x;\ a_n] \wedge k>n_b)$ $\neg(\exists k<n_c.\ [a_0;\ x;\ f\ k])$
proof −
have $\exists\ n_b\ n_c\ b\ c.\ b=f\ n_b \wedge c=f\ n_c \wedge [b;x;c] \wedge b\in Y \wedge c\in Y \wedge n_b = n_c - 1 \wedge n_c<card\ Y \wedge n_c>0$
$\wedge \neg(\exists k < card\ Y.\ [f\ k;\ x;\ a_n] \wedge k>n_b) \wedge \neg(\exists k < n_c.\ [a_0;\ x;\ f\ k])$
proof −
have *bound-indices*: $f\ 0 = a_0 \wedge f\ (card\ Y - 1) = a_n$
using *chain-defs long-ch-Y* **by** *simp*
have *fin-Y*: *finite Y*
using *chain-defs long-ch-Y* **by** *presburger*
have *card-Y*: $card\ Y \geq 3$
using *long-ch-Y points-in-long-chain finite-long-chain-with-card* **by** *blast*
have *chY2*: *local-long-ch-by-ord f Y*
using *long-ch-Y chain-defs* **by** (*meson card-Y long-ch-card-ge3*)
obtain P **where** *P-def*: $P\in\mathcal{P}$ $Y\subseteq P$
using *fin-chain-on-path long-ch-Y fin-Y chain-defs* **by** *meson*
hence $x\in P$
using *betw-b-in-path x-def*(*2*) *long-ch-Y points-in-long-chain*
by (*metis abc-abc-neq in-mono*)
obtain n_c **where** *nc-def*: $\neg(\exists k.\ [a_0;\ x;\ f\ k] \wedge k<n_c)$ $[a_0;\ x;\ f\ n_c]$ $n_c<card\ Y$ $n_c>0$
using *smallest-k-ex* [**where** $a_1=a_0$ **and** $a=a$ **and** $a_n=a_n$ **and** $b=x$ **and** $f=f$ **and** $Y=Y$]
long-ch-Y x-def
by *blast*
then obtain c **where** *c-def*: $c=f\ n_c$ $c\in Y$
using *chain-defs local-ordering-def* **by** (*metis chY2*)
have *c-goal*: $c=f\ n_c \wedge c\in Y \wedge n_c<card\ Y \wedge n_c>0 \wedge \neg(\exists k < card\ Y.\ [a_0;\ x;\ f\ k] \wedge k<n_c)$
using *c-def nc-def*(*1*,*3*,*4*) **by** *blast*
obtain n_b **where** *nb-def*: $\neg(\exists k < card\ Y.\ [f\ k;\ x;\ a_n] \wedge k>n_b)$ $[f\ n_b;\ x;\ a_n]$ $n_b<card\ Y-1$
using *greatest-k-ex* [**where** $a_1=a_0$ **and** $a=a$ **and** $a_n=a_n$ **and** $b=x$ **and** $f=f$ **and** $Y=Y$]
long-ch-Y x-def
by *blast*
hence $n_b<card\ Y$
by *linarith*
then obtain b **where** *b-def*: $b=f\ n_b$ $b\in Y$
using *nb-def chY2 local-ordering-def* **by** (*metis local-long-ch-by-ord-alt*)
have $[b;x;c]$
proof −
have $[b;\ x;\ a_n]$
using *b-def*(*1*) *nb-def*(*2*) **by** *blast*
have $[a_0;\ x;\ c]$
using *c-def*(*1*) *nc-def*(*2*) **by** *blast*
moreover have $\forall a.\ [a;x;b] \vee \neg\ [a;\ a_n;\ x]$

using ‹[b; x; a_n]› *abc-bcd-acd*
by (*metis* (*full-types*) *abc-sym*)
moreover have $\forall a.$ [a;x;b] $\vee \neg$ [a_n; a; x]
using ‹[b; x; a_n]› **by** (*meson abc-acd-bcd abc-sym*)
moreover have $a_n = c \longrightarrow$ [b;x;c]
using ‹[b; x; a_n]› **by** *meson*
ultimately show *?thesis*
using *abc-abd-bcdbdc abc-sym x-def*(*2*)
by *meson*
qed
have $n_b < n_c$
using ‹[b;x;c]› ‹$n_c <$ *card Y*› ‹$n_b <$ *card Y*› ‹$c = f\ n_c$› ‹$b = f\ n_b$›
by (*smt* (*z3*) *abc-abd-bcdbdc abc-ac-neq abc-acd-abd abc-only-cba*(*4*) *abc-sym* *bot-nat-0.extremum*
bound-indices chY2 fin-Y nat-neq-iff nc-def(*2*) *nc-def*(*4*) *order-finite-chain*)
have $n_b = n_c - 1$
proof (*rule ccontr*)
assume $n_b \neq n_c - 1$
have $n_b < n_c - 1$
using ‹$n_b \neq n_c - 1$› ‹$n_b < n_c$› **by** *linarith*
hence [$f\ n_b$; ($f(n_c - 1)$); $f\ n_c$]
using ‹$n_b \neq n_c - 1$› *long-ch-Y nc-def*(*3*) *order-finite-chain chain-defs*
by *auto*
have $\neg$[a_0; x; ($f(n_c - 1)$)]
using *nc-def*(*1*,*4*) *diff-less less-numeral-extra*(*1*)
by *blast*
have $n_c - 1 \neq 0$
using ‹$n_b < n_c$› ‹$n_b \neq n_c - 1$› **by** *linarith*
hence $f(n_c - 1) \neq a_0 \wedge a_0 \neq x$
using *bound-indices* ‹$n_b < n_c - 1$› *abc-abc-neq less-imp-diff-less nb-def*(*1*) *nc-def*(*3*) *x-def*(*2*)
by *blast*
have $x \neq f(n_c - 1)$
using *x-def*(*1*) *nc-def*(*3*) *chY2* **unfolding** *chain-defs local-ordering-def*
by (*metis One-nat-def Suc-pred less-Suc-eq nc-def*(*4*) *not-less-eq*)
hence [a_0; $f\ (n_c - 1)$; x]
using *long-ch-Y nc-def c-def chain-defs*
by (*metis* ‹[$f\ n_b$;$f\ (n_c - 1)$;$f\ n_c$]› ‹$\neg$ [a_0;x;$f\ (n_c - 1)$]› *abc-ac-neq abc-acd-abd* *abc-bcd-acd*
abd-acd-abcacb abd-bcd-abc b-def(*1*) *b-def*(*2*) *fin-ch-betw2 nb-def*(*2*))
hence [($f(n_c - 1)$); x; a_n]
using *abc-acd-bcd x-def*(*2*) **by** *blast*
thus *False* **using** *nb-def*(*1*)
using ‹$n_b < n_c - 1$› *less-imp-diff-less nc-def*(*3*)
by *blast*
qed
have *b-goal*: $b = f\ n_b \wedge b \in Y \wedge n_b = n_c - 1 \wedge \neg(\exists\, k < card\ Y.$ [$f\ k$; x; a_n] $\wedge\ k > n_b)$
using *b-def nb-def*(*1*) *nb-def*(*3*) ‹$n_b = n_c - 1$› **by** *blast*
thus *?thesis*

```
      using ‹[b;x;c]› c-goal
      using ‹n_b < card Y› nc-def(1) by auto
  qed
  thus ?thesis
    using that by auto
qed
```

This is case (ii) of the induction in Theorem 10.

```
lemma chain-append-inside:
  assumes long-ch-Y: [f⇝Y|a₁..a..aₙ]
      and Y-def: b∉Y
      and Yb: [a₁; b; aₙ]
      and k-def: [a₁; b; f k] k < card Y ¬(∃k'. (0::nat)<k' ∧ k'<k ∧ [a₁; b; f k'])
    fixes g
   defines g-def: g ≡ (λj::nat. if (j≤k−1) then f j else (if (j=k) then b else f
(j−1)))
    shows [g⇝insert b Y|a₁ .. b .. aₙ]
proof −
  let ?X = insert b Y
  have fin-X: finite ?X
    by (meson chain-defs finite.insertI long-ch-Y)
  have bound-indices: f 0 = a₁ ∧ f (card Y − 1) = aₙ
    using chain-defs long-ch-Y
    by auto
  have fin-Y: finite Y
    using chain-defs long-ch-Y by presburger
  have f-def: local-long-ch-by-ord f Y
   using chain-defs long-ch-Y by (meson finite-long-chain-with-card long-ch-card-ge3)
  have ‹a₁ ≠ aₙ ∧ a₁ ≠ b ∧ b ≠ aₙ›
    using Yb abc-abc-neq by blast
  have k ≠ 0
    using abc-abc-neq bound-indices k-def
    by metis

  have b-middle: [f(k−1); b; f k]
  proof (cases)
    assume k=1 show [f(k−1); b; f k]
      using ‹[a₁; b; f k]› ‹k = 1› bound-indices by auto
  next assume k≠1 show [f(k−1); b; f k]
    proof −
      have a1k: [a₁; f (k−1); f k] using bound-indices
        using ‹k < card Y› ‹k ≠ 0› ‹k ≠ 1› long-ch-Y fin-Y order-finite-chain
        unfolding chain-defs by auto
```

In fact, the comprehension below gives the order of elements too. Our notation and Theorem 9 are too weak to say that just now.

```
      have ch-with-b: ch {a₁, (f (k−1)), b, (f k)} using chain4
        using k-def(1) abc-ex-path-unique between-chain cross-once-notin
        by (smt ‹[a₁; f (k−1); f k]› abc-abc-neq insert-absorb2)
```

have *f* (*k*−*1*) ≠ *b* ∧ (*f k*) ≠ (*f* (*k*−*1*)) ∧ *b* ≠ (*f k*)
using *abc-abc-neq f-def k-def*(*2*) *Y-def*
by (*metis local-ordering-def* ‹[a_1; *f* (*k*−*1*); *f k*]› *less-imp-diff-less local-long-ch-by-ord-def*)
hence *some-ord-bk*: [*f*(*k*−*1*); *b*; *f k*] ∨ [*b*; *f* (*k*−*1*); *f k*] ∨ [*f* (*k*−*1*); *f k*; *b*]
using *fin-chain-on-path ch-with-b some-betw Y-def chain-defs*
by (*metis a1k abc-acd-bcd abd-acd-abcacb k-def*(*1*))
thus [*f*(*k*−*1*); *b*; *f k*]
proof −
have ¬ [a_1; *f k*; *b*]
by (*simp add*: ‹[a_1; *b*; *f k*]› *abc-only-cba*(*2*))
thus *?thesis*
using *some-ord-bk k-def abc-bcd-acd abd-bcd-abc bound-indices*
by (*metis diff-is-0-eq′ diff-less less-imp-diff-less less-irrefl-nat not-less zero-less-diff zero-less-one* ‹[a_1; *b*; *f k*]› *a1k*)
qed
qed
qed

let *?case1* ∨ *?case2* = *k*−*2* ≥ *0* ∨ *k*+*1* ≤ *card Y* −*1*

have *b-right*: [*f* (*k*−*2*); *f* (*k*−*1*); *b*] **if** *k* ≥ *2*
proof −
have *k*−*1* < (*k*::*nat*)
using ‹*k* ≠ *0*› *diff-less zero-less-one* **by** *blast*
hence *k*−*2* < *k*−*1*
using ‹*2* ≤ *k*› **by** *linarith*
have [*f* (*k*−*2*); *f* (*k*−*1*); *b*]
using *abd-bcd-abc b-middle f-def k-def*(*2*) *fin-Y* ‹*k*−*2* < *k*−*1*› ‹*k*−*1* < *k*› *thm2-ind2 chain-defs*
by (*metis Suc-1 Suc-le-lessD diff-Suc-eq-diff-pred that zero-less-diff*)
thus [*f* (*k*−*2*); *f* (*k*−*1*); *b*]
using ‹[*f*(*k* − *1*); *b*; *f k*]› *abd-bcd-abc*
by *blast*
qed

have *b-left*: [*b*; *f k*; *f* (*k*+*1*)] **if** *k*+*1* ≤ *card Y* −*1*
proof −
have [*f* (*k*−*1*); *f k*; *f* (*k*+*1*)]
using ‹*k* ≠ *0*› *f-def fin-Y order-finite-chain that*
by *auto*
thus [*b*; *f k*; *f* (*k*+*1*)]
using ‹[*f* (*k* − *1*); *b*; *f k*]› *abc-acd-bcd*
by *blast*
qed

have *local-ordering g betw ?X*
proof −
have ∀ *n*. (*finite ?X* ⟶ *n* < *card ?X*) ⟶ *g n* ∈ *?X*

proof (*clarify*)
  fix *n* **assume** *finite ?X* ⟶ *n* < *card ?X g n* ∉ *Y*
  consider *n*≤*k*−*1* | *n*≥*k*+*1* | *n*=*k*
    by *linarith*
  thus *g n* = *b*
  proof (*cases*)
    assume *n* ≤ *k* − *1*
    thus *g n* = *b*
      using *f-def k-def*(*2*) *Y-def*(*1*) *chain-defs local-ordering-def g-def*
      by (*metis* ‹*g n* ∉ *Y*› ‹*k* ≠ *0*› *diff-less le-less less-one less-trans not-le*)
  next
    assume *k* + *1* ≤ *n*
    show *g n* = *b*
    proof −
      have *f n* ∈ *Y* ∨ ¬(*n* < *card Y*) **for** *n*
        using *chain-defs* **by** (*metis local-ordering-def f-def*)
      then show *g n* = *b*
        using ‹*finite ?X* ⟶ *n* < *card ?X*› *fin-Y g-def Y-def* ‹*g n* ∉ *Y*› ‹*k* + *1*
≤ *n*›
          not-less not-less-simps(*1*) *not-one-le-zero*
        by *fastforce*
    qed
  next
    assume *n*=*k*
    thus *g n* = *b*
      using *Y-def* ‹*k* ≠ *0*› *g-def*
      by *auto*
  qed
qed
moreover have ∀ *x*∈*?X*. ∃ *n*. (*finite ?X* ⟶ *n* < *card ?X*) ∧ *g n* = *x*
proof
  fix *x* **assume** *x*∈*?X*
  show ∃ *n*. (*finite ?X* ⟶ *n* < *card ?X*) ∧ *g n* = *x*
  proof (*cases*)
    assume *x*∈*Y*
    show *?thesis*
    proof −
      obtain *ix* **where** *f ix* = *x ix* < *card Y*
        using ‹*x* ∈ *Y*› *f-def fin-Y*
        unfolding *chain-defs local-ordering-def*
        by *auto*
      have *ix*≤*k*−*1* ∨ *ix*≥*k*
        by *linarith*
      thus *?thesis*
      proof
        assume *ix*≤*k*−*1*
        hence *g ix* = *x*
          using ‹*f ix* = *x*› *g-def* **by** *auto*
        moreover have *finite ?X* ⟶ *ix* < *card ?X*

using *Y-def* ‹*ix* < *card Y*› **by** *auto*
ultimately show *?thesis* **by** *metis*
next assume *ix*≥*k*
hence *g* (*ix*+*1*) = *x*
using ‹*f ix* = *x*› *g-def* **by** *auto*
moreover have *finite ?X* ⟶ *ix*+*1* < *card ?X*
using *Y-def* ‹*ix* < *card Y*› **by** *auto*
ultimately show *?thesis* **by** *metis*
qed
qed
next assume *x*∉*Y*
hence *x*=*b*
using *Y-def* ‹*x* ∈ *?X*› **by** *blast*
thus *?thesis*
using *Y-def* ‹*k* ≠ *0*› *k-def*(*2*) *ordered-cancel-comm-monoid-diff-class.le-diff-conv2*
g-def
by *auto*
qed
qed
moreover have ∀ *n n′ n″*. (*finite ?X* ⟶ *n″* < *card ?X*) ∧ *Suc n* = *n′* ∧ *Suc*
n′ = *n″*
⟶ [*g n*; *g* (*Suc n*); *g* (*Suc* (*Suc n*))]
proof (*clarify*)
fix *n n′ n″* **assume** *a*: (*finite ?X* ⟶ (*Suc* (*Suc n*)) < *card ?X*)

Introduce the two-case splits used later.

have *cases-sn*: *Suc n*≤*k*−*1* ∨ *Suc n*=*k* **if** *n*≤*k*−*1*
using ‹*k* ≠ *0*› *that* **by** *linarith*
have *cases-ssn*: *Suc*(*Suc n*)≤*k*−*1* ∨ *Suc*(*Suc n*)=*k* **if** *n*≤*k*−*1* *Suc n*≤*k*−*1*
using *that*(*2*) **by** *linarith*

consider *n*≤*k*−*1* | *n*≥*k*+*1* | *n*=*k*
by *linarith*
then show [*g n*; *g* (*Suc n*); *g* (*Suc* (*Suc n*))]
proof (*cases*)
assume *n*≤*k*−*1* **show** *?thesis*
using *cases-sn*
proof (*rule disjE*)
assume *Suc n* ≤ *k* − *1*
show *?thesis* **using** *cases-ssn*
proof (*rule disjE*)
show *n* ≤ *k* − *1* **using** ‹*n* ≤ *k* − *1*› **by** *blast*
show ‹*Suc n* ≤ *k* − *1*› **using** ‹*Suc n* ≤ *k* − *1*› **by** *blast*
next
assume *Suc* (*Suc n*) ≤ *k* − *1*
thus *?thesis*
using ‹*Suc n* ≤ *k* − *1*› ‹*k* ≠ *0*› ‹*n* ≤ *k* − *1*› *ordering-ord-ijk-loc f-def*
g-def k-def(*2*)
by (*metis* (*no-types*, *lifting*) *add-diff-inverse-nat less-Suc-eq-le*

```
          less-imp-le-nat less-le-trans less-one local-long-ch-by-ord-def plus-1-eq-Suc)
        next
          assume Suc (Suc n) = k
          thus ?thesis
            using b-right g-def by force
        qed
      next
        assume Suc n = k
        show ?thesis
          using b-middle ‹Suc n = k› ‹n ≤ k − 1› g-def
          by auto
      next show n ≤ k−1 using ‹n ≤ k − 1› by blast
      qed
    next assume n≥k+1 show ?thesis
      proof −
        have g n = f (n−1)
          using ‹k + 1 ≤ n› less-imp-diff-less g-def
          by auto
        moreover have g (Suc n) = f (n)
          using ‹k + 1 ≤ n› g-def by auto
        moreover have g (Suc (Suc n)) = f (Suc n)
          using ‹k + 1 ≤ n› g-def by auto
        moreover have n−1<n ∧ n<Suc n
          using ‹k + 1 ≤ n› by auto
        moreover have finite Y ⟶ Suc n < card Y
          using Y-def a by auto
        ultimately show ?thesis
          using f-def unfolding chain-defs local-ordering-def
          by (metis ‹k + 1 ≤ n› add-leD2 le-add-diff-inverse plus-1-eq-Suc)
      qed
    next assume n=k
      show ?thesis
        using ‹k ≠ 0› ‹n = k› b-left g-def Y-def(1) a assms(3) fin-Y
        by auto
    qed
  qed
  ultimately show local-ordering g betw ?X
    unfolding local-ordering-def
    by presburger
 qed
 hence local-long-ch-by-ord g ?X
   using Y-def f-def local-long-ch-by-ord-def local-long-ch-by-ord-def
   by auto
 thus [g⇝?X|a1..b..an]
     using fin-X ‹a1 ≠ an ∧ a1 ≠ b ∧ b ≠ an› bound-indices k-def(2) Y-def g-def
chain-defs
     by simp
qed
```

lemma *card4-eq*:
  assumes *card X = 4*
  shows $\exists$ *a b c d. a* $\neq$ *b* $\wedge$ *a* $\neq$ *c* $\wedge$ *a* $\neq$ *d* $\wedge$ *b* $\neq$ *c* $\wedge$ *b* $\neq$ *d* $\wedge$ *c* $\neq$ *d* $\wedge$ *X* = {*a*, *b*, *c*, *d*}
proof −
  obtain *a X′* **where** *X = insert a X′* **and** *a* $\notin$ *X′*
    by (*metis Suc-eq-numeral assms card-Suc-eq*)
  then have *card X′ = 3*
    by (*metis add-2-eq-Suc′ assms card-eq-0-iff card-insert-if diff-Suc-1 finite-insert numeral-3-eq-3 numeral-Bit0 plus-nat.add-0 zero-neq-numeral*)
  then obtain *b X″* **where** *X′ = insert b X″* **and** *b* $\notin$ *X″*
    by (*metis card-Suc-eq numeral-3-eq-3*)
  then have *card X″ = 2*
    by (*metis Suc-eq-numeral* ‹*card X′ = 3*› *card.infinite card-insert-if finite-insert pred-numeral-simps*(*3*) *zero-neq-numeral*)
  then have $\exists$ *c d. c* $\neq$ *d* $\wedge$ *X″* = {*c*, *d*}
    by (*meson card-2-iff*)
  thus *?thesis*
    using ‹*X = insert a X′*› ‹*X′ = insert b X″*› ‹*a* $\notin$ *X′*› ‹*b* $\notin$ *X″*› **by** *blast*
qed

theorem *path-finsubset-chain*:
  assumes *Q* $\in$ $\mathcal{P}$
    and *X* $\subseteq$ *Q*
    and *card X* $\geq$ *2*
  shows *ch X*
proof −
  have *finite X*
    using *assms*(*3*) *not-numeral-le-zero* **by** *fastforce*
  consider *card X = 2* | *card X = 3* | *card X* $\geq$ *4*
    using ‹*card X* $\geq$ *2*› **by** *linarith*
  thus *?thesis*
  proof (*cases*)
    assume *card X = 2*
    thus *?thesis*
      using ‹*finite X*› *assms two-event-chain* **by** *blast*
  next
    assume *card X = 3*
    thus *?thesis*
      using ‹*finite X*› *assms three-event-chain* **by** *blast*
  next
    assume *card X* $\geq$ *4*
    thus *?thesis*
      using *assms*(*1*,*2*) ‹*finite X*›
    proof (*induct card X − 4 arbitrary*: *X*)
      case *0*
      then have *card X = 4*

by *auto*
then have $\exists\, a\ b\ c\ d.\ a \neq b \wedge a \neq c \wedge a \neq d \wedge b \neq c \wedge b \neq d \wedge c \neq d \wedge X = \{a,\ b,\ c,\ d\}$
using *card4-eq* **by** *fastforce*
thus *?case*
using *0.prems(3) assms(1) chain4* **by** *auto*
next
case *IH*: (*Suc n*)

then obtain *Y b* **where** *X-eq*: $X = insert\ b\ Y$ **and** $b \notin Y$
by (*metis Diff-iff card-eq-0-iff finite.cases insertI1 insert-Diff-single not-numeral-le-zero*)
have $card\ Y \geq 4\ n = card\ Y - 4$
using *IH.hyps(2) IH.prems(4) X-eq* ‹$b \notin Y$› **by** *auto*
then have *ch Y*
using *IH(1) [of Y] IH.prems(3,4) X-eq assms(1)* **by** *auto*

then obtain *f* **where** *f-ords*: *local-long-ch-by-ord f Y*
using ‹$4 \leq card\ Y$› *ch-alt short-ch-card(2)* **by** *auto*
then obtain $a_1\ a\ a_n$ **where** *long-ch-Y*: $[f \leadsto Y|a_1..a..a_n]$
using ‹$4 \leq card\ Y$› *get-fin-long-ch-bounds* **by** *fastforce*
hence *bound-indices*: $f\ 0 = a_1 \wedge f\ (card\ Y - 1) = a_n$
by (*simp add*: *chain-defs*)
have $a_1 \neq a_n \wedge a_1 \neq b \wedge b \neq a_n$
using ‹$b \notin Y$› *abc-abc-neq fin-ch-betw long-ch-Y points-in-long-chain* **by** *metis*
moreover have $a_1 \in Q \wedge a_n \in Q \wedge b \in Q$
using *IH.prems(3) X-eq long-ch-Y points-in-long-chain* **by** *auto*
ultimately consider $[b;\ a_1;\ a_n] \mid [a_1;\ a_n;\ b] \mid [a_n;\ b;\ a_1]$
using *some-betw* [*of Q b* $a_1\ a_n$] ‹$Q \in \mathcal{P}$› **by** *blast*
thus *ch X*
proof (*cases*)

assume $[b;\ a_1;\ a_n]$
have *X-eq'*: $X = Y \cup \{b\}$
using *X-eq* **by** *auto*
let $?g = \lambda j.\ if\ j \geq 1\ then\ f\ (j - 1)\ else\ b$
have $[?g \leadsto X|b..a_1..a_n]$
using *chain-append-at-left-edge IH.prems(4) X-eq'* ‹$[b;\ a_1;\ a_n]$› ‹$b \notin Y$› *long-ch-Y X-eq*
by *presburger*
thus *ch X*
using *chain-defs* **by** *auto*
next

assume $[a_1;\ a_n;\ b]$
let $?g = \lambda j.\ if\ j \leq (card\ X - 2)\ then\ f\ j\ else\ b$
have $[?g \leadsto X|a_1..a_n..b]$
using *chain-append-at-right-edge IH.prems(4) X-eq* ‹$[a_1;\ a_n;\ b]$› ‹$b \notin Y$› *long-ch-Y*

by *auto*
thus *ch X* **using** *chain-defs* **by** (*meson ch-def*)
next

assume $[a_n;\ b;\ a_1]$
then have $[a_1;\ b;\ a_n]$
by (*simp add*: *abc-sym*)
obtain *k* **where**
k-def: $[a_1;\ b;\ f\ k]$ $k < card\ Y$ $\neg\ (\exists\, k'.\ 0 < k' \wedge k' < k \wedge [a_1;\ b;\ f\ k'])$
using ‹$[a_1;\ b;\ a_n]$› ‹$b \notin Y$› *long-ch-Y smallest-k-ex* **by** *blast*
obtain *g* **where** $g = (\lambda j{::}nat.\ if\ j \leq k - 1$
$then\ f\ j$
$else\ if\ j = k$
$then\ b\ else\ f\ (j - 1))$
by *simp*
hence $[g{\leadsto}X|a_1..b..a_n]$
using *chain-append-inside* [*of f Y* a_1 *a* a_n *b k*] *IH.prems*(*4*) *X-eq*
‹$[a_1;\ b;\ a_n]$› ‹$b \notin Y$› *k-def long-ch-Y*
by *auto*
thus *ch X*
using *chain-defs ch-def* **by** *auto*
qed
qed
qed
qed

lemma *path-finsubset-chain2*:
assumes $Q \in \mathcal{P}$ **and** $X \subseteq Q$ **and** $card\ X \geq 2$
obtains *f a b* **where** $[f{\leadsto}X|a..b]$
proof −
have *finX*: *finite X*
by (*metis assms*(*3*) *card.infinite rel-simps*(*28*))
have *ch-X*: *ch X*
using *path-finsubset-chain*[*OF assms*] **by** *blast*
obtain *f a b* **where** *f-def*: $[f{\leadsto}X|a..b]$ $a{\in}X \wedge b{\in}X$
using *assms finX ch-X get-fin-long-ch-bounds chain-defs*
by (*metis ch-def points-in-chain*)
thus *?thesis*
using *that* **by** *auto*
qed

30.2 Theorem 11

Notice this case is so simple, it doesn't even require the path density larger sets of segments rely on for fixing their cardinality.

lemma *segmentation-ex-N2*:
assumes *path-P*: $P{\in}\mathcal{P}$
and *Q-def*: *finite* $(Q{::}'a\ set)$ $card\ Q = N$ $Q{\subseteq}P$ $N{=}2$

```
      and f-def: [f↝Q|a..b]
      and S-def: S = {segment a b}
      and P1-def: P1 = prolongation b a
      and P2-def: P2 = prolongation a b
    shows P = ((⋃S) ∪ P1 ∪ P2 ∪ Q) ∧
           card S = (N−1) ∧ (∀x∈S. is-segment x) ∧
               P1∩P2={} ∧ (∀x∈S. (x∩P1={} ∧ x∩P2={} ∧ (∀y∈S. x≠y ⟶
x∩y={})))
proof −
  have a∈Q ∧ b∈Q ∧ a≠b
    using chain-defs f-def points-in-chain first-neq-last
    by (metis)
  hence Q={a,b}
    using assms(3,5)
   by (smt card-2-iff insert-absorb insert-commute insert-iff singleton-insert-inj-eq)
  have a∈P ∧ b∈P
    using ‹Q={a,b}› assms(4) by auto
  have a≠b using ‹Q={a,b}›
    using ‹N = 2› assms(3) by force
  obtain s where s-def: s = segment a b by simp
  let ?S = {s}
  have P = ((⋃{s}) ∪ P1 ∪ P2 ∪ Q) ∧
          card {s} = (N−1) ∧ (∀x∈{s}. is-segment x) ∧
            P1∩P2={} ∧ (∀x∈{s}. (x∩P1={} ∧ x∩P2={} ∧ (∀y∈{s}. x≠y ⟶
x∩y={})))
  proof (rule conjI)
    { fix x assume x∈P
      have [a;x;b] ∨ [b;a;x] ∨ [a;b;x] ∨ x=a ∨ x=b
        using ‹a∈P ∧ b∈P› some-betw path-P ‹a≠b›
        by (meson ‹x ∈ P› abc-sym)
      then have x∈s ∨ x∈P1 ∨ x∈P2 ∨ x=a ∨ x=b
        using pro-betw seg-betw P1-def P2-def s-def ‹Q = {a, b}›
        by auto
      hence x ∈ (⋃{s}) ∪ P1 ∪ P2 ∪ Q
        using ‹Q = {a, b}› by auto
    } moreover {
      fix x assume x ∈ (⋃{s}) ∪ P1 ∪ P2 ∪ Q
      hence x∈s ∨ x∈P1 ∨ x∈P2 ∨ x=a ∨ x=b
        using ‹Q = {a, b}› by blast
      hence [a;x;b] ∨ [b;a;x] ∨ [a;b;x] ∨ x=a ∨ x=b
        using s-def P1-def P2-def
        unfolding segment-def prolongation-def
        by auto
      hence x∈P
        using ‹a ∈ P ∧ b ∈ P› ‹a ≠ b› betw-b-in-path betw-c-in-path path-P
        by blast
    }
    ultimately show union-P: P = ((⋃{s}) ∪ P1 ∪ P2 ∪ Q)
      by blast
```

```
    show card {s} = (N−1) ∧ (∀x∈{s}. is-segment x) ∧ P1∩P2={} ∧
          (∀x∈{s}. (x∩P1={} ∧ x∩P2={} ∧ (∀y∈{s}. x≠y ⟶ x∩y={})))
    proof (safe)
      show card {s} = N − 1
        using ‹Q = {a, b}› ‹a ≠ b› assms(3) by auto
      show is-segment s
        using s-def by blast
      show ⋀x. x ∈ P1 ⟹ x ∈ P2 ⟹ x ∈ {}
      proof −
        fix x assume x∈P1 x∈P2
        show x∈{}
          using P1-def P2-def ‹x ∈ P1› ‹x ∈ P2› abc-only-cba pro-betw
          by metis
      qed
      show ⋀x xa. xa ∈ s ⟹ xa ∈ P1 ⟹ xa ∈ {}
      proof −
        fix x xa assume xa∈s xa∈P1
        show xa∈{}
          using abc-only-cba seg-betw pro-betw  P1-def ‹xa ∈ P1› ‹xa ∈ s› s-def
          by (metis)
      qed
      show ⋀x xa. xa ∈ s ⟹ xa ∈ P2 ⟹ xa ∈ {}
      proof −
        fix x xa assume xa∈s xa∈P2
        show xa∈{}
          using abc-only-cba seg-betw pro-betw
          by (metis P2-def ‹xa ∈ P2› ‹xa ∈ s› s-def)
      qed
    qed
  qed
  thus ?thesis
    by (simp add: S-def s-def)
qed
```

```
lemma int-split-to-segs:
  assumes f-def: [f↝Q|a..b..c]
  fixes S defines S-def: S ≡ {segment (f i) (f(i+1)) | i. i<card Q−1}
  shows interval a c = (⋃S) ∪ Q
proof
  let ?N = card Q
  have f-def-2: a∈Q ∧ b∈Q ∧ c∈Q
    using f-def points-in-long-chain by blast
  hence ?N ≥ 3
    using f-def long-ch-card-ge3 chain-defs
    by (meson finite-long-chain-with-card)
  have bound-indices: f 0 = a ∧ f (card Q − 1) = c
    using f-def chain-defs by auto
```

```
  let ?i = ?u = interval a c = (⋃ S) ∪ Q
  show ?i⊆?u
  proof
    fix p assume p ∈ ?i
    show p∈?u
    proof (cases)
      assume p∈Q thus ?thesis by blast
    next assume p∉Q
      hence p≠a ∧ p≠c
        using f-def f-def-2 by blast
      hence [a;p;c]
        using seg-betw ‹p ∈ interval a c› interval-def
        by auto
      then obtain n_y n_z y z
        where yz-def: y=f n_y z=f n_z [y;p;z] y∈Q z∈Q n_y=n_z−1 n_z<card Q
          ¬(∃ k < card Q. [f k; p; c] ∧ k>n_y) ¬(∃ k<n_z. [a; p; f k])
        using get-closest-chain-events [where f=f and x=p and Y=Q and a_n=c
and a_0=a and a=b]
          f-def ‹p∉Q›
        by metis
      have n_y<card Q−1
        using yz-def(6,7) f-def index-middle-element
        by fastforce
      let ?s = segment (f n_y) (f n_z)
      have p∈?s
        using ‹[y;p;z]› abc-abc-neq seg-betw yz-def(1,2)
        by blast
      have n_z = n_y + 1
        using yz-def(6)
      by (metis abc-abc-neq add.commute add-diff-inverse-nat less-one yz-def(1,2,3)
zero-diff)
      hence ?s∈S
        using S-def ‹n_y<card Q−1› assms(2)
        by blast
      hence p∈⋃ S
        using ‹p ∈ ?s› by blast
      thus ?thesis by blast
    qed
  qed
  show ?u⊆?i
  proof
    fix p assume p ∈ ?u
    hence p∈⋃ S ∨ p∈Q by blast
    thus p∈?i
    proof
      assume p∈Q
      then consider p=a|p=c|[a;p;c]
        using f-def by (meson fin-ch-betw2 finite-long-chain-with-alt)
      thus ?thesis
```

proof (*cases*)
assume *p=a*
thus *?thesis* **by** (*simp add*: *interval-def*)
next assume *p=c*
thus *?thesis* **by** (*simp add*: *interval-def*)
next assume [*a*;*p*;*c*]
thus *?thesis* **using** *interval-def seg-betw* **by** *auto*
qed
next assume $p\in\bigcup S$
then obtain *s* **where** *p∈s s∈S*
by *blast*
then obtain *y* **where** *s = segment* (*f y*) (*f* (*y+1*)) *y<?N−1*
using *S-def* **by** *blast*
hence *y+1<?N* **by** (*simp add*: *assms*(*2*))
hence *fy-in-Q*: (*f y*)∈*Q* ∧ *f* (*y+1*) ∈ *Q*
using *f-def add-lessD1* **unfolding** *chain-defs local-ordering-def*
by (*metis One-nat-def Suc-eq-plus1 Zero-not-Suc* ‹*3≤card Q*› *card-1-singleton-iff card-gt-0-iff*
card-insert-if diff-add-inverse2 diff-is-0-eq′ less-numeral-extra(*1*) *numeral-3-eq-3 plus-1-eq-Suc*)
have [*a*; *f y*; *c*] ∨ *y=0*
using ‹*y <?N − 1*› *assms*(*2*) *f-def chain-defs order-finite-chain* **by** *auto*
moreover have [*a*; *f* (*y+1*); *c*] ∨ *y = ?N−2*
using ‹*y+1 < card Q*› *assms*(*2*) *f-def chain-defs order-finite-chain i-le-j-events-neq*
using *indices-neq-imp-events-neq fin-ch-betw2 fy-in-Q*
by (*smt* (*z3*) *Nat.add-0-right Nat.add-diff-assoc add-gr-0 card-Diff1-less card-Diff-singleton-if*
diff-diff-left diff-is-0-eq′ le-numeral-extra(*4*) *less-numeral-extra*(*1*) *nat-1-add-1*)
ultimately consider *y=0*|*y=?N−2*|([*a*; *f y*; *c*] ∧ [*a*; *f* (*y+1*); *c*])
by *linarith*
hence [*a*;*p*;*c*]
proof (*cases*)
assume *y=0*
hence *f y = a*
by (*simp add*: *bound-indices*)
hence [*a*; *p*; (*f*(*y+1*))]
using ‹*p ∈ s*› ‹*s = segment* (*f y*) (*f* (*y + 1*))› *seg-betw*
by *auto*
moreover have [*a*; (*f*(*y+1*)); *c*]
using ‹[*a*; (*f*(*y+1*)); *c*] ∨ *y = ?N − 2*› ‹*y = 0*› ‹*?N≥3*›
by *linarith*
ultimately show [*a*;*p*;*c*]
using *abc-acd-abd* **by** *blast*
next
assume *y=?N−2*
hence *f* (*y+1*) = *c*
using *bound-indices* ‹*?N≥3*› *numeral-2-eq-2 numeral-3-eq-3*
by (*metis One-nat-def Suc-diff-le add.commute add-leD2 diff-Suc-Suc plus-1-eq-Suc*)

hence [*f y*; *p*; *c*]
  using ‹*p* ∈ *s*› ‹*s* = *segment* (*f y*) (*f* (*y* + *1*))› *seg-betw*
  by *auto*
moreover have [*a*; *f y*; *c*]
  using ‹[*a*; *f y*; *c*] ∨ *y* = *0*› ‹*y* = *?N* − *2*› ‹*?N*≥*3*›
  by *linarith*
ultimately show [*a*;*p*;*c*]
  by (*meson abc-acd-abd abc-sym*)
next
assume [*a*; *f y*; *c*] ∧ [*a*; (*f*(*y*+*1*)); *c*]
thus [*a*;*p*;*c*]
  using *abe-ade-bcd-ace* [**where** *a*=*a* **and** *b*=*f y* **and** *d*=*f* (*y*+*1*) **and** *e*=*c* **and** *c*=*p*]
  using ‹*p* ∈ *s*› ‹*s* = *segment* (*f y*) (*f*(*y*+*1*))› *seg-betw*
  by *auto*
qed
thus *?thesis*
  using *interval-def seg-betw* **by** *auto*
qed
qed
qed

lemma *path-is-union*:
assumes *path-P*: *P*∈𝒫
  and *Q-def*: *finite* (*Q*::′*a set*) *card Q* = *N Q*⊆*P N*≥*3*
  and *f-def*: *a*∈*Q* ∧ *b*∈*Q* ∧ *c*∈*Q* [*f*⇝*Q*|*a*..*b*..*c*]
  and *S-def*: *S* = {*s*. ∃ *i*<(*N*−*1*). *s* = *segment* (*f i*) (*f* (*i*+*1*))}
  and *P1-def*: *P1* = *prolongation b a*
  and *P2-def*: *P2* = *prolongation b c*
shows *P* = ((⋃ *S*) ∪ *P1* ∪ *P2* ∪ *Q*)
proof −

have *in-P*: *a*∈*P* ∧ *b*∈*P* ∧ *c*∈*P*
  using *assms*(*4*) *f-def* **by** *blast*
have *bound-indices*: *f 0* = *a* ∧ *f* (*card Q* − *1*) = *c*
  using *f-def chain-defs* **by** *auto*
have *points-neq*: *a*≠*b* ∧ *b*≠*c* ∧ *a*≠*c*
  using *f-def chain-defs* **by** (*metis first-neq-last*)

The proof in two parts: subset inclusion one way, then the other.

{ fix *x* **assume** *x*∈*P*
have [*a*;*x*;*c*] ∨ [*b*;*a*;*x*] ∨ [*b*;*c*;*x*] ∨ *x*=*a* ∨ *x*=*c*
  using *in-P some-betw path-P points-neq* ‹*x* ∈ *P*› *abc-sym*
  by (*metis* (*full-types*) *abc-acd-bcd fin-ch-betw f-def*(*2*))
then have (∃ *s*∈*S*. *x*∈*s*) ∨ *x*∈*P1* ∨ *x*∈*P2* ∨ *x*∈*Q*
proof (*cases*)
  assume [*a*;*x*;*c*]
  hence *only-axc*: ¬([*b*;*a*;*x*] ∨ [*b*;*c*;*x*] ∨ *x*=*a* ∨ *x*=*c*)

using *abc-only-cba*
by (*meson abc-bcd-abd abc-sym f-def fin-ch-betw*)
have $x \in$ *interval a c*
using ‹[*a*;*x*;*c*]› *interval-def seg-betw* **by** *auto*
hence $x \in Q \vee x \in \bigcup S$
using *int-split-to-segs S-def assms*(*2*,*3*,*5*) *f-def*
by *blast*
thus *?thesis* **by** *blast*
next assume ¬[*a*;*x*;*c*]
hence [*b*;*a*;*x*] ∨ [*b*;*c*;*x*] ∨ *x*=*a* ∨ *x*=*c*
using ‹[*a*;*x*;*c*] ∨ [*b*;*a*;*x*] ∨ [*b*;*c*;*x*] ∨ *x* = *a* ∨ *x* = *c*› **by** *blast*
hence *x*∈*P1* ∨ *x*∈*P2* ∨ *x*∈*Q*
using *P1-def P2-def f-def pro-betw* **by** *auto*
thus *?thesis* **by** *blast*
qed
hence $x \in (\bigcup S) \cup P1 \cup P2 \cup Q$ **by** *blast*
} moreover {
fix *x* **assume** $x \in (\bigcup S) \cup P1 \cup P2 \cup Q$
hence (∃ *s*∈*S*. *x*∈*s*) ∨ *x*∈*P1* ∨ *x*∈*P2* ∨ *x*∈*Q*
by *blast*
hence $x \in \bigcup S$ ∨ [*b*;*a*;*x*] ∨ [*b*;*c*;*x*] ∨ *x*∈*Q*
using *S-def P1-def P2-def*
unfolding *segment-def prolongation-def*
by *auto*
hence *x*∈*P*
proof (*cases*)
assume $x \in \bigcup S$
have *S* = {*segment* (*f i*) (*f*(*i*+*1*)) | *i*. *i*<*N*−*1*}
using *S-def* **by** *blast*
hence *x*∈*interval a c*
using *int-split-to-segs* [*OF f-def*(*2*)] *assms* ‹$x \in \bigcup S$›
by (*simp add*: *UnCI*)
hence [*a*;*x*;*c*] ∨ *x*=*a* ∨ *x*=*c*
using *interval-def seg-betw* **by** *auto*
thus *?thesis*
proof (*rule disjE*)
assume *x*=*a* ∨ *x*=*c*
thus *?thesis*
using *in-P* **by** *blast*
next
assume [*a*;*x*;*c*]
thus *?thesis*
using *betw-b-in-path in-P path-P points-neq* **by** *blast*
qed
next assume $x \notin \bigcup S$
hence [*b*;*a*;*x*] ∨ [*b*;*c*;*x*] ∨ *x*∈*Q*
using ‹$x \in \bigcup S$ ∨ [*b*;*a*;*x*] ∨ [*b*;*c*;*x*] ∨ *x* ∈ *Q*›
by *blast*
thus *?thesis*

```
      using assms(4) betw-c-in-path in-P path-P points-neq
      by blast
    qed
  }
  ultimately show P = ((⋃S) ∪ P1 ∪ P2 ∪ Q)
    by blast
qed


lemma inseg-axc:
  assumes path-P: P∈𝒫
      and Q-def: finite (Q::'a set) card Q = N Q⊆P N≥3
      and f-def: a∈Q ∧ b∈Q ∧ c∈Q [f↝Q|a..b..c]
      and S-def: S = {s. ∃ i<(N−1). s = segment (f i) (f (i+1))}
      and x-def: x∈s s∈S
    shows [a;x;c]
proof −
  have fQ: local-long-ch-by-ord f Q
   using f-def Q-def chain-defs by (metis ch-long-if-card-geq3 path-P short-ch-card(1)
short-xor-long(2))
  have inseg-neq-ac: x≠a ∧ x≠c if x∈s s∈S for x s
  proof
    show x≠a
    proof (rule notI)
      assume x=a
      obtain n where s-def: s = segment (f n) (f (n+1)) n<N−1
        using S-def ‹s ∈ S› by blast
      hence n<card Q using assms(3) by linarith
      hence f n ∈ Q
        using fQ unfolding chain-defs local-ordering-def by blast
      hence [a; f n; c]
          using f-def finite-long-chain-with-def assms(3) order-finite-chain seg-betw
that(1)
        using ‹n < N − 1› ‹s = segment (f n) (f (n + 1))› ‹x = a›
      by (metis abc-abc-neq add-lessD1 fin-ch-betw inside-not-bound(2) less-diff-conv)
      moreover have [(f(n)); x; (f(n+1))]
        using ‹x∈s› seg-betw s-def(1) by simp
      ultimately show False
        using ‹x=a› abc-only-cba(1) assms(3) fQ chain-defs s-def(2)
        by (smt (z3) ‹n < card Q› f-def(2) order-finite-chain-indices2 thm2-ind1)
    qed

    show x≠c
    proof (rule notI)
      assume x=c
      obtain n where s-def: s = segment (f n) (f (n+1)) n<N−1
        using S-def ‹s ∈ S› by blast
      hence n+1<N by simp
      have [(f(n)); x; (f(n+1))]
```

```
        using ‹x∈s› seg-betw s-def(1) by simp
      have f (n) ∈ Q
        using fQ ‹n+1 < N› chain-defs local-ordering-def
        by (metis add-lessD1 assms(3))
      have f (n+1) ∈ Q
        using ‹n+1 < N› fQ chain-defs local-ordering-def
        by (metis assms(3))
      have f(n+1) ≠ c
        using ‹x=c› ‹[(f(n)); x; (f(n+1))]› abc-abc-neq
        by blast
      hence [a; (f(n+1)); c]
         using f-def finite-long-chain-with-def assms(3) order-finite-chain seg-betw
that(1)
          abc-abc-neq abc-only-cba fin-ch-betw
        by (metis ‹[f n; x; f (n + 1)]› ‹f (n + 1) ∈ Q› ‹f n ∈ Q› ‹x = c›)
      thus False
        using ‹x=c› ‹[(f(n)); x; (f(n+1))]› assms(3) f-def s-def(2)
          abc-only-cba(1) finite-long-chain-with-def order-finite-chain
        by (metis ‹f n ∈ Q› abc-bcd-acd abc-only-cba(1,2) fin-ch-betw)
    qed
  qed

  show [a;x;c]
  proof −
    have x∈interval a c
      using int-split-to-segs [OF f-def(2)] S-def assms(2,3,5) x-def
      by blast
    have x≠a ∧ x≠c using inseg-neq-ac
      using x-def by auto
    thus ?thesis
      using seg-betw ‹x ∈ interval a c› interval-def
      by auto
  qed
qed


lemma disjoint-segmentation:
  assumes path-P: P∈𝒫
      and Q-def: finite (Q::'a set) card Q = N Q⊆P N≥3
      and f-def: a∈Q ∧ b∈Q ∧ c∈Q [f⇝Q|a..b..c]
      and S-def: S = {s. ∃ i<(N−1). s = segment (f i) (f (i+1))}
      and P1-def: P1 = prolongation b a
      and P2-def: P2 = prolongation b c
     shows P1∩P2={} ∧ (∀ x∈S. (x∩P1={} ∧ x∩P2={} ∧ (∀ y∈S. x≠y ⟶
x∩y={})))
proof (rule conjI)
  have fQ: local-long-ch-by-ord f Q
   using f-def Q-def chain-defs by (metis ch-long-if-card-geq3 path-P short-ch-card(1)
short-xor-long(2))
```

```
  show P1 ∩ P2 = {}
  proof (safe)
    fix x assume x∈P1 x∈P2
    show x∈{}
      using abc-only-cba pro-betw P1-def P2-def
      by (metis ‹x ∈ P1› ‹x ∈ P2› abc-bcd-abd f-def(2) fin-ch-betw)
  qed

  show ∀x∈S. (x∩P1={} ∧ x∩P2={} ∧ (∀y∈S. x≠y ⟶ x∩y={}))
  proof (rule ballI)
    fix s assume s∈S
    show s ∩ P1 = {} ∧ s ∩ P2 = {} ∧ (∀y∈S. s ≠ y ⟶ s ∩ y = {})
    proof (intro conjI ballI impI)
      show s∩P1={}
      proof (safe)
        fix x assume x∈s x∈P1
        hence [a;x;c]
          using inseg-axc ‹s ∈ S› assms by blast
        thus x∈{}
         by (metis P1-def ‹x ∈ P1› abc-bcd-abd abc-only-cba(1) f-def(2) fin-ch-betw
pro-betw)
      qed
      show s∩P2={}
      proof (safe)
        fix x assume x∈s x∈P2
        hence [a;x;c]
          using inseg-axc ‹s ∈ S› assms by blast
        thus x∈{}
         by (metis P2-def ‹x ∈ P2› abc-bcd-acd abc-only-cba(2) f-def(2) fin-ch-betw
pro-betw)
      qed
      fix r assume r∈S s≠r
      show s∩r={}
      proof (safe)
        fix y assume y ∈ r y ∈ s
        obtain n m where rs-def: r = segment (f n) (f(n+1)) s = segment (f m)
(f(m+1))
                                n≠m n<N−1 m<N−1
          using S-def ‹r ∈ S› ‹s ≠ r› ‹s ∈ S› by blast
        have y-betw: [f n; y; (f(n+1))] ∧ [f m; y; (f(m+1))]
          using seg-betw ‹y∈r› ‹y∈s› rs-def(1,2) by simp
        have False
        proof (cases)
          assume n<m
          have [f n; f m; (f(m+1))]
             using ‹n < m› assms(3) fQ chain-defs order-finite-chain rs-def(5) by
(metis assms(2) thm2-ind1)
          have n+1<m
           using ‹[f n; f m; f(m + 1)]› ‹n < m› abc-only-cba(2) abd-bcd-abc y-betw
```

```
          by (metis Suc-eq-plus1 Suc-leI le-eq-less-or-eq)
        hence [f n; (f(n+1)); f m]
          using fQ assms(3) rs-def(5) unfolding chain-defs local-ordering-def
            by (metis (full-types) ‹[f n;f m;f (m + 1)]› abc-only-cba(1) abc-sym
abd-bcd-abc assms(2) fQ thm2-ind1 y-betw)
        hence [f n; (f(n+1)); y]
          using ‹[f n; f m; f(m + 1)]› abc-acd-abd abd-bcd-abc y-betw
          by blast
        thus ?thesis
          using abc-only-cba y-betw by blast
      next
        assume ¬n<m
        hence n>m using nat-neq-iff rs-def(3) by blast
        have [f m; f n; (f(n+1))]
           using ‹n > m› assms(3) fQ chain-defs rs-def(4) by (metis assms(2)
thm2-ind1)
        hence m+1<n
          using ‹n > m› abc-only-cba(2) abd-bcd-abc y-betw
          by (metis Suc-eq-plus1 Suc-leI le-eq-less-or-eq)
        hence [f m; (f(m+1)); f n]
          using fQ assms(2,3) rs-def(4) unfolding chain-defs local-ordering-def
         by (metis (no-types, lifting) ‹[f m;f n;f (n + 1)]› abc-only-cba(1) abc-sym
abd-bcd-abc fQ thm2-ind1 y-betw)
        hence [f m; (f(m+1)); y]
          using ‹[f m; f n; f(n + 1)]› abc-acd-abd abd-bcd-abc y-betw
          by blast
        thus ?thesis
          using abc-only-cba y-betw by blast
      qed
      thus y∈{} by blast
    qed
  qed
 qed
qed


lemma segmentation-ex-Nge3:
 assumes path-P: P∈𝒫
     and Q-def: finite (Q::'a set) card Q = N Q⊆P N≥3
     and f-def: a∈Q ∧ b∈Q ∧ c∈Q [f↝Q|a..b..c]
     and S-def: S = {s. ∃ i<(N−1). s = segment (f i) (f (i+1))}
     and P1-def: P1 = prolongation b a
     and P2-def: P2 = prolongation b c
   shows P = ((⋃S) ∪ P1 ∪ P2 ∪ Q) ∧
          (∀ x∈S. is-segment x) ∧
             P1∩P2={} ∧ (∀ x∈S. (x∩P1={} ∧ x∩P2={} ∧ (∀ y∈S. x≠y ⟶
x∩y={})))
proof (intro disjoint-segmentation conjI)
 show P = ((⋃S) ∪ P1 ∪ P2 ∪ Q)
```

using *path-is-union assms*
by *blast*
show $\forall x \in S$. *is-segment x*
proof
fix *s* **assume** $s \in S$
thus *is-segment s* **using** *S-def* **by** *auto*
qed
qed (*use assms disjoint-segmentation* **in** *auto*)

Some unfolding of the definition for a finite chain that happens to be short.

lemma *finite-chain-with-card-2*:
assumes *f-def*: $[f \leadsto Q|a..b]$
and *card-Q*: *card* $Q = 2$
shows *finite Q* $f\ 0 = a$ $f\ (card\ Q - 1) = b$ $Q = \{f\ 0, f\ 1\}$ $\exists Q.$ *path Q* $(f\ 0)$ $(f\ 1)$
using *assms* **unfolding** *chain-defs* **by** *auto*

Schutz says "As in the proof of the previous theorem [...]" - does he mean to imply that this should really be proved as induction? I can see that quite easily, induct on N, and add a segment by either splitting up a segment or taking a piece out of a prolongation. But I think that might be too much trouble.

theorem *show-segmentation*:
assumes *path-P*: $P \in \mathcal{P}$
and *Q-def*: $Q \subseteq P$
and *f-def*: $[f \leadsto Q|a..b]$
fixes *P1* **defines** *P1-def*: $P1 \equiv$ *prolongation b a*
fixes *P2* **defines** *P2-def*: $P2 \equiv$ *prolongation a b*
fixes *S* **defines** *S-def*: $S \equiv \{segment\ (f\ i)\ (f\ (i+1)) \mid i.\ i < card\ Q - 1\}$
shows $P = ((\bigcup S) \cup P1 \cup P2 \cup Q)$ $(\forall x \in S.$ *is-segment x*$)$
disjoint $(S \cup \{P1, P2\})$ $P1 \neq P2$ $P1 \notin S$ $P2 \notin S$
proof –
have *card-Q*: *card* $Q \geq 2$
using *fin-chain-card-geq-2 f-def* **by** *blast*
have *finite Q*
by (*metis card.infinite card-Q rel-simps*(*28*))
have *f-def-2*: $a \in Q \land b \in Q$
using *f-def points-in-chain finite-chain-with-def* **by** *auto*
have $a \neq b$
using *f-def chain-defs* **by** (*metis first-neq-last*)
{
assume *card* $Q = 2$
hence *card* $Q - 1 = Suc\ 0$ **by** *simp*
have $Q = \{f\ 0, f\ 1\}$ $\exists Q.$ *path Q* $(f\ 0)$ $(f\ 1)$ $f\ 0 = a$ $f\ (card\ Q - 1) = b$
using ‹*card* $Q = 2$› *finite-chain-with-card-2 f-def* **by** *auto*
hence $S = \{segment\ a\ b\}$
unfolding *S-def* **using** ‹*card* $Q - 1 = Suc\ 0$› **by** (*simp add*: *eval-nat-numeral*)
hence $P = ((\bigcup S) \cup P1 \cup P2 \cup Q)$ $(\forall x \in S.$ *is-segment x*$)$ $P1 \cap P2 = \{\}$
$(\forall x \in S. (x \cap P1 = \{\} \land x \cap P2 = \{\} \land (\forall y \in S. x \neq y \longrightarrow x \cap y = \{\})))$

using *assms f-def ‹finite Q› segmentation-ex-N2*
[**where** *P=P* **and** *Q=Q* **and** *N=card Q*]
by (*metis* (*no-types, lifting*) *‹card Q = 2›*)+
} moreover {
assume *card Q ≠ 2*
hence *card Q ≥ 3*
using *card-Q* **by** *auto*
then obtain *c* **where** *c-def*: [*f⇝Q|a..c..b*]
using *assms(3,5) ‹a≠b› chain-defs*
by (*metis f-def three-in-set3*)
have *pro-equiv*: *P1 = prolongation c a ∧ P2 = prolongation c b*
using *pro-basis-change*
using *P1-def P2-def abc-sym c-def fin-ch-betw* **by** *auto*
have *S-def2*: *S = {s. ∃ i<(card Q−1). s = segment (f i) (f (i+1))}*
using *S-def ‹card Q ≥ 3›* **by** *auto*
have *P = ((⋃ S) ∪ P1 ∪ P2 ∪ Q) (∀ x∈S. is-segment x) P1∩P2={}*
(*∀ x∈S. (x∩P1={} ∧ x∩P2={} ∧ (∀ y∈S. x≠y ⟶ x∩y={})))*
using *f-def-2 assms f-def ‹card Q ≥ 3› c-def pro-equiv*
segmentation-ex-Nge3 [**where** *P=P* **and** *Q=Q* **and** *N=card Q* **and** *S=S*
and *a=a* **and** *b=c* **and** *c=b* **and** *f=f*]
using *points-in-long-chain ‹finite Q› S-def2* **by** *metis*+
}
ultimately have *old-thesis*: *P = ((⋃ S) ∪ P1 ∪ P2 ∪ Q) (∀ x∈S. is-segment x)*
P1∩P2={}
(*∀ x∈S. (x∩P1={} ∧ x∩P2={} ∧ (∀ y∈S. x≠y ⟶ x∩y={})))* **by**
meson+
thus *disjoint (S∪{P1,P2}) P1≠P2 P1∉S P2∉S*
P = ((⋃ S) ∪ P1 ∪ P2 ∪ Q) (∀ x∈S. is-segment x)
unfolding *disjoint-def* **apply** (*simp add*: *Int-commute*)
apply (*metis P2-def Un-iff old-thesis(1,3) ‹a ≠ b› disjoint-iff f-def-2 path-P*
pro-betw prolong-betw2)
apply (*metis P1-def Un-iff old-thesis(1,4) ‹a ≠ b› disjoint-iff f-def-2 path-P*
pro-betw prolong-betw3)
apply (*metis P2-def Un-iff old-thesis(1,4) ‹a ≠ b› disjoint-iff f-def-2 path-P*
pro-betw prolong-betw)
using *old-thesis(1,2)* **by** *linarith*+
qed

theorem *segmentation*:
assumes *path-P*: *P∈𝒫*
and *Q-def*: *card Q≥2 Q⊆P*
shows *∃ S P1 P2. P = ((⋃ S) ∪ P1 ∪ P2 ∪ Q) ∧*
disjoint (S∪{P1,P2}) ∧ P1≠P2 ∧ P1∉S ∧ P2∉S ∧
(∀ x∈S. is-segment x) ∧ is-prolongation P1 ∧ is-prolongation P2
proof −
let *?N = card Q*

obtain *f a b* **where** *f-def*: [*f⇝Q|a..b*]

```
    using path-finsubset-chain2[OF path-P Q-def(2,1)]
    by metis
  let ?S = {segment (f i) (f (i+1)) | i. i<card Q−1}
  let ?P1 = prolongation b a
  let ?P2 = prolongation a b
  have from-seg: P = ((⋃ ?S) ∪ ?P1 ∪ ?P2 ∪ Q) (∀x∈?S. is-segment x)
        disjoint (?S∪{?P1,?P2}) ?P1≠?P2 ?P1∉?S ?P2∉?S
    using show-segmentation[OF path-P Q-def(2) ‹[f↝Q|a..b]›]
    by force+
  thus ?thesis
    by blast
qed

end
```

31 Chains are unique up to reversal

```
context MinkowskiSpacetime begin

lemma chain-remove-at-right-edge:
  assumes [f↝X|a..c] f (card X − 2) = p 3 ≤ card X X = insert c Y c∉Y
  shows [f↝Y|a..p]
proof −
  have lch-X: local-long-ch-by-ord f X
    using assms(1,3) chain-defs short-ch-card-2
    by fastforce
  have p∈X
    by (metis local-ordering-def assms(2) card.empty card-gt-0-iff diff-less lch-X
        local-long-ch-by-ord-def not-numeral-le-zero zero-less-numeral)
  have bound-ind: f 0 = a ∧ f (card X − 1) = c
   using lch-X assms(1,3) unfolding finite-chain-with-def finite-long-chain-with-def
    by metis

  have [a;p;c]
  proof −
    have card X − 2 < card X − 1
      using ‹3 ≤ card X› by auto
    moreover have card X − 2 > 0
      using ‹3 ≤ card X› by linarith
    ultimately show ?thesis
      using order-finite-chain[OF lch-X] ‹3 ≤ card X› assms(2) bound-ind
        by (metis card.infinite diff-less le-numeral-extra(3) less-numeral-extra(1)
not-gr-zero not-numeral-le-zero)
  qed

  have [f↝X|a..p..c]
    unfolding finite-long-chain-with-alt by (simp add: assms(1) ‹[a;p;c]› ‹p∈X›)
```

have *1*: *x* = *a* **if** *x* ∈ *Y* ¬ [*a*;*x*;*p*] *x* ≠ *p* **for** *x*
proof −
have *x*∈*X*
using *that*(*1*) *assms*(*4*) **by** *simp*
hence *01*: *x*=*a* ∨ [*a*;*p*;*x*]
by (*metis that*(*2*,*3*) ‹[*a*;*p*;*c*]› *abd-acd-abcacb assms*(*1*) *fin-ch-betw2*)
have *02*: *x*=*c* **if** [*a*;*p*;*x*]
proof −
obtain *i* **where** *i-def*: *f i* = *x i*<*card X*
using ‹*x*∈*X*› *chain-defs* **by** (*meson assms*(*1*) *obtain-index-fin-chain*)
have *f 0* = *a*
by (*simp add*: *bound-ind*)
have *card X* − *2* < *i*
using *order-finite-chain-indices*[*OF lch-X - that* ‹*f 0* = *a*› *assms*(*2*) *i-def*(*1*) - - *i-def*(*2*)]
by (*metis card-eq-0-iff card-gt-0-iff diff-less i-def*(*2*) *less-nat-zero-code zero-less-numeral*)
hence *i* = *card X* − *1* **using** *i-def*(*2*) **by** *linarith*
thus *?thesis* **using** *bound-ind i-def*(*1*) **by** *blast*
qed
show *?thesis* **using** *01 02 assms*(*5*) *that*(*1*) **by** *auto*
qed

have *Y* = {*e* ∈ *X*. [*a*;*e*;*p*] ∨ *e* = *a* ∨ *e* = *p*}
apply (*safe*, *simp-all add*: *assms*(*4*) *1*)
using ‹[*a*;*p*;*c*]› *abc-only-cba*(*2*) *abc-abc-neq assms*(*4*) **by** *blast*+

thus *?thesis* **using** *chain-shortening*[*OF* ‹[*f*↝*X*|*a*..*p*..*c*]›] **by** *simp*
qed

lemma (**in** *MinkowskiChain*) *fin-long-ch-imp-fin-ch*:
assumes [*f*↝*X*|*a*..*b*..*c*]
shows [*f*↝*X*|*a*..*c*]
using *assms* **by** (*simp add*: *finite-long-chain-with-alt*)

If we ever want to have chains less strongly identified by endpoints, this result should generalise - a, c, x, z are only used to identify reversal/no-reversal cases.

lemma *chain-unique-induction-ax*:
assumes *card X* ≥ *3*
and *i* < *card X*
and [*f*↝*X*|*a*..*c*]
and [*g*↝*X*|*x*..*z*]
and *a* = *x* ∨ *c* = *z*
shows *f i* = *g i*
using *assms*
proof (*induct card X* − *3 arbitrary*: *X a c x z*)

case *Nil*: *0*
have *card X = 3*
using *Nil.hyps Nil.prems(1)* **by** *auto*

obtain *b* **where** *f-ch*: [*f*⇝*X*|*a..b..c*]
using *chain-defs* **by** (*metis Nil.prems(1,3) three-in-set3*)
obtain *y* **where** *g-ch*: [*g*⇝*X*|*x..y..z*]
using *Nil.prems chain-defs* **by** (*metis three-in-set3*)

have *i=1* ∨ *i=0* ∨ *i=2*
using ‹*card X = 3*› *Nil.prems(2)* **by** *linarith*
thus *?case*
proof (*rule disjE*)
assume *i=1*
hence *f i = b* ∧ *g i = y*
using *index-middle-element f-ch g-ch* ‹*card X = 3*› *numeral-3-eq-3*
by (*metis One-nat-def add-diff-cancel-left′ less-SucE not-less-eq plus-1-eq-Suc*)
have *f i = g i*
proof (*rule ccontr*)
assume *f i* ≠ *g i*
hence *g i* ≠ *b*
by (*simp add*: ‹*f i = b* ∧ *g i = y*›)
have *g i* ∈ *X*
using ‹*f i = b* ∧ *g i = y*› *g-ch points-in-long-chain* **by** *blast*
have *X* = {*a*,*b*,*c*}
using *f-ch* **unfolding** *finite-long-chain-with-alt*
using ‹*card X = 3*› *points-in-long-chain*[*OF f-ch*] *abc-abc-neq*[*of a b c*]
by (*simp add*: *card-3-eq′(2)*)
hence (*g i = a* ∨ *g i = c*)
using ‹*g i* ≠ *b*› ‹*g i* ∈ *X*› **by** *blast*
hence ¬ [*a*; *g i*; *c*]
using *abc-abc-neq* **by** *blast*
hence *g i* ∉ *X*
using ‹*f i=b* ∧ *g i=y*› ‹*g i=a* ∨ *g i=c*› *f-ch g-ch chain-bounds-unique finite-long-chain-with-def*
by *blast*
thus *False*
by (*simp add*: ‹*g i* ∈ *X*›)
qed
thus *?thesis*
by (*simp add*: ‹*card X = 3*› ‹*i = 1*›)
next
assume *i = 0* ∨ *i = 2*
show *?thesis*
using *Nil.prems(5)* ‹*card X = 3*› ‹*i = 0* ∨ *i = 2*› *chain-bounds-unique f-ch g-ch chain-defs*
by (*metis diff-Suc-1 numeral-2-eq-2 numeral-3-eq-3*)
qed
next

```
case IH: (Suc n)
have lch-fX: local-long-ch-by-ord f X
  using chain-defs long-ch-card-ge3 IH(3,5)
  by fastforce
have lch-gX: local-long-ch-by-ord g X
  using IH(3,6) chain-defs long-ch-card-ge3
  by fastforce
have fin-X: finite X
  using IH(4) le-0-eq by fastforce

have ch-by-ord f X
  using lch-fX unfolding ch-by-ord-def by blast
have card X ≥ 4
  using IH.hyps(2) by linarith

obtain b where f-ch: [f⇝X|a..b..c]
  using IH(3,5) chain-defs by (metis three-in-set3)
obtain y where g-ch: [g⇝X|x..y..z]
  using IH.prems(1,4) chain-defs by (metis three-in-set3)

obtain p where p-def: p = f (card X − 2) by simp
have [a;p;c]
proof −
  have card X − 2 < card X − 1
    using ‹4 ≤ card X› by auto
  moreover have card X − 2 > 0
    using ‹3 ≤ card X› by linarith
  ultimately show ?thesis
    using f-ch p-def chain-defs ‹[f⇝X]› order-finite-chain2 by force
qed
hence p≠c ∧ p≠a
  using abc-abc-neq by blast

obtain Y where Y-def: X = insert c Y c∉Y
  using f-ch points-in-long-chain
  by (meson mk-disjoint-insert)
hence fin-Y: finite Y
  using f-ch chain-defs by auto
hence n = card Y − 3
  using ‹Suc n = card X − 3› ‹X = insert c Y› ‹c∉Y› card-insert-if
  by auto
hence card-Y: card Y = n + 3
  using Y-def(1) Y-def(2) fin-Y IH.hyps(2) by fastforce
have card Y = card X − 1
  using Y-def(1,2) fin-X by auto
have p∈Y
    using ‹X = insert c Y› ‹[a;p;c]› abc-abc-neq lch-fX p-def IH.prems(1,3)
Y-def(2)
  by (metis chain-remove-at-right-edge points-in-chain)
```

have [*f*⇝*Y*|*a..p*]
using *chain-remove-at-right-edge* [**where** *f=f* **and** *a=a* **and** *c=c* **and** *X=X* **and** *p=p* **and** *Y=Y*]
using *fin-long-ch-imp-fin-ch* [**where** *f=f* **and** *a=a* **and** *c=c* **and** *b=b* **and** *X=X*]
using *f-ch p-def* ‹*card X* ≥ *3*› *Y-def*
by *blast*
hence *ch-fY*: *local-long-ch-by-ord f Y*
using *card-Y fin-Y chain-defs long-ch-card-ge3*
by *force*

have *p-closest*: ¬ (∃ *q*∈*X*. [*p*;*q*;*c*])
proof
assume (∃ *q*∈*X*. [*p*;*q*;*c*])
then obtain *q* **where** *q*∈*X* [*p*;*q*;*c*] **by** *blast*
then obtain *j* **where** *j* < *card X f j* = *q*
using *lch-fX lch-gX fin-X points-in-chain* ‹*p*≠*c* ∧ *p*≠*a*› *chain-defs*
by (*metis local-ordering-def*)
have *j* > *card X* − *2* ∧ *j* < *card X* − *1*
proof −
have *j* > *card X* − *2* ∧ *j* < *card X* − *1* ∨ *j* > *card X* − *1* ∧ *j* < *card X* − *2*
apply (*intro order-finite-chain-indices*[*OF lch-fX* ‹*finite X*› ‹[*p*;*q*;*c*]›])
using *p-def* ‹*f j* = *q*› *IH.prems*(*3*) *finite-chain-with-def* ‹*j* < *card X*› **by** *auto*
thus *?thesis* **by** *linarith*
qed
thus *False* **by** *linarith*
qed

have *g* (*card X* − *2*) = *p*
proof (*rule ccontr*)
assume *asm-false*: *g* (*card X* − *2*) ≠ *p*
obtain *j* **where** *g j* = *p j* < *card X* − *1 j*>*0*
using ‹*X* = *insert c Y*› ‹*p*∈*Y*› *points-in-chain* ‹*p*≠*c* ∧ *p*≠*a*›
by (*metis* (*no-types*) *chain-bounds-unique f-ch*
finite-long-chain-with-def g-ch index-middle-element insert-iff)
hence *j* < *card X* − *2*
using *asm-false le-eq-less-or-eq* **by** *fastforce*
hence *j* < *card Y* − *1*
by (*simp add*: *Y-def*(*1*,*2*) *fin-Y*)
obtain *d* **where** *d* = *g* (*card X* − *2*) **by** *simp*
have [*p*;*d*;*z*]
proof −
have *card X* − *1* > *card X* − *2*
using ‹*j* < *card X* − *1*› **by** *linarith*
thus *?thesis*
using *lch-gX* ‹*j* < *card Y* − *1*› ‹*card Y* = *card X* − *1*› ‹*d* = *g* (*card X* − *2*)› ‹*g j* = *p*›
order-finite-chain[*OF lch-gX*] *chain-defs local-ordering-def*

```
      by (smt (z3) IH.prems(3−5) asm-false chain-bounds-unique chain-remove-at-right-edge
p-def
           ‹⋀thesis. (⋀Y. ⟦X = insert c Y; c ∉ Y⟧ ⟹ thesis) ⟹ thesis›)
    qed
    moreover have d∈X
      using lch-gX ‹d = g (card X − 2)› unfolding local-long-ch-by-ord-def lo-
cal-ordering-def
      by auto
    ultimately show False
      using p-closest abc-sym IH.prems(3−5) chain-bounds-unique f-ch g-ch
      by blast
  qed

  hence ch-gY: local-long-ch-by-ord g Y
   using IH.prems(1,4,5) g-ch f-ch ch-fY card-Y chain-remove-at-right-edge fin-Y
chain-defs
    by (metis Y-def chain-bounds-unique long-ch-card-ge3)

  have f i ∈ Y ∨ f i = c
    by (metis local-ordering-def ‹X = insert c Y› ‹i < card X› lch-fX insert-iff
local-long-ch-by-ord-def)
  thus f i = g i
  proof (rule disjE)
    assume f i ∈ Y
    hence f i ≠ c
      using ‹c ∉ Y› by blast
    hence i < card Y
     using ‹X = insert c Y› ‹c∉Y› IH(3,4) f-ch fin-Y chain-defs not-less-less-Suc-eq
      by (metis ‹card Y = card X − 1› card-insert-disjoint)
    hence 3 ≤ card Y
      using card-Y le-add2 by presburger
    show f i = g i
      using IH(1) [of Y]
      using ‹n = card Y − 3› ‹3 ≤ card Y› ‹i < card Y›
      using Y-def card-Y chain-remove-at-right-edge le-add2
      by (metis IH.prems(1,3,4,5) chain-bounds-unique)
  next
    assume f i = c
    show ?thesis
     using IH.prems(2,5) ‹f i = c› chain-bounds-unique f-ch g-ch indices-neq-imp-events-neq
chain-defs
      by (metis ‹card Y = card X − 1› Y-def card-insert-disjoint fin-Y lessI)
  qed
qed
```

I'm really impressed *sledgehammer/smt* can solve this if I just tell them "Use symmetry!".

```
lemma chain-unique-induction-cx:
  assumes card X ≥ 3
```

and $i < card\ X$
and $[f \leadsto X|a..c]$
and $[g \leadsto X|x..z]$
and $c = x \vee a = z$
shows $f\ i = g\ (card\ X - i - 1)$
using *chain-sym-obtain2 chain-unique-induction-ax assms diff-right-commute* **by** *smt*

This lemma has to exclude two-element chains again, because no order exists within them. Alternatively, the result is trivial: any function that assigns one element to index 0 and the other to 1 can be replaced with the (unique) other assignment, without destroying any (trivial, since ternary) *local-ordering* of the chain. This could be made generic over the *local-ordering* similar to $[?f \leadsto ?X|?a..?b..?c] \Longrightarrow [\lambda n.\ ?f\ (card\ ?X - 1 - n) \leadsto ?X|?c..?b..?a]$ relying on $[\![\bigwedge a\ b\ c.\ ?ord\ a\ b\ c \Longrightarrow ?ord\ c\ b\ a;\ finite\ ?X;\ local\text{-}ordering\ ?f\ ?ord\ ?X]\!] \Longrightarrow local\text{-}ordering\ (\lambda n.\ ?f\ (card\ ?X - 1 - n))\ ?ord\ ?X$.

lemma *chain-unique-upto-rev-cases*:
assumes *ch-f*: $[f \leadsto X|a..c]$
and *ch-g*: $[g \leadsto X|x..z]$
and *card-X*: $card\ X \geq 3$
and *valid-index*: $i < card\ X$
shows $((a=x \vee c=z) \longrightarrow (f\ i = g\ i))\ ((a=z \vee c=x) \longrightarrow (f\ i = g\ (card\ X - i - 1)))$
proof −
obtain *n* **where** *n-def*: $n = card\ X - 3$
by *blast*
hence *valid-index-2*: $i < n + 3$
by (*simp add*: *card-X valid-index*)

show $((a=x \vee c=z) \longrightarrow (f\ i = g\ i))$
using *card-X ch-f ch-g chain-unique-induction-ax valid-index* **by** *blast*
show $((a=z \vee c=x) \longrightarrow (f\ i = g\ (card\ X - i - 1)))$
using *assms*(*3*) *ch-f ch-g chain-unique-induction-cx valid-index* **by** *blast*
qed

lemma *chain-unique-upto-rev*:
assumes $[f \leadsto X|a..c]\ [g \leadsto X|x..z]\ card\ X \geq 3\ i < card\ X$
shows $f\ i = g\ i \vee f\ i = g\ (card\ X - i - 1)\ a=x \wedge c=z \vee c=x \wedge a=z$
proof −
have $(a=x \vee c=z) \vee (a=z \vee c=x)$
using *chain-bounds-unique* **by** (*metis assms*(*1*,*2*))
thus $f\ i = g\ i \vee f\ i = g\ (card\ X - i - 1)$
using *assms*(*3*) ‹$i < card\ X$› *assms chain-unique-upto-rev-cases* **by** *blast*
thus $(a=x \wedge c=z) \vee (c=x \wedge a=z)$
by (*meson assms*(*1*−*3*) *chain-bounds-unique*)
qed

end

32 Interlude: betw4 and WLOG

32.1 betw4 - strict and non-strict, basic lemmas

context *MinkowskiBetweenness* **begin**

Define additional notation for non-strict *local-ordering* - cf Schutz' monograph [1, p. 27].

abbreviation *nonstrict-betw-right* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$ (‹[-;-;-⟧›) **where**
nonstrict-betw-right $a\ b\ c \equiv [a;b;c] \lor b = c$

abbreviation *nonstrict-betw-left* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$ (‹⟦-;-;-]›) **where**
nonstrict-betw-left $a\ b\ c \equiv [a;b;c] \lor b = a$

abbreviation *nonstrict-betw-both* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$ **where**
nonstrict-betw-both $a\ b\ c \equiv$ *nonstrict-betw-left* $a\ b\ c \lor$ *nonstrict-betw-right* $a\ b\ c$

abbreviation *betw4* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$ (‹[-;-;-;-]›) **where**
betw4 $a\ b\ c\ d \equiv [a;b;c] \land [b;c;d]$

abbreviation *nonstrict-betw-right4* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$ (‹[-;-;-;-⟧›) **where**
nonstrict-betw-right4 $a\ b\ c\ d \equiv$ *betw4* $a\ b\ c\ d \lor c = d$

abbreviation *nonstrict-betw-left4* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$ (‹⟦-;-;-;-]›) **where**
nonstrict-betw-left4 $a\ b\ c\ d \equiv$ *betw4* $a\ b\ c\ d \lor a = b$

abbreviation *nonstrict-betw-both4* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$ **where**
nonstrict-betw-both4 $a\ b\ c\ d \equiv$ *nonstrict-betw-left4* $a\ b\ c\ d \lor$ *nonstrict-betw-right4* $a\ b\ c\ d$

lemma *betw4-strong*:
assumes *betw4* $a\ b\ c\ d$
shows $[a;b;d] \land [a;c;d]$
using *abc-bcd-acd assms* **by** *blast*

lemma *betw4-imp-neq*:
assumes *betw4* $a\ b\ c\ d$
shows $a\neq b \land a\neq c \land a\neq d \land b\neq c \land b\neq d \land c\neq d$
using *abc-only-cba assms* **by** *blast*

end
context *MinkowskiSpacetime* **begin**

lemma *betw4-weak*:
fixes $a\ b\ c\ d :: 'a$

assumes $[a;b;c] \wedge [a;c;d]$
$\vee [a;b;c] \wedge [b;c;d]$
$\vee [a;b;d] \wedge [b;c;d]$
$\vee [a;b;d] \wedge [b;c;d]$
shows *betw4 a b c d*
using *abc-acd-bcd abd-bcd-abc assms* **by** *blast*

lemma *betw4-sym*:
fixes $a::'a$ **and** $b::'a$ **and** $c::'a$ **and** $d::'a$
shows *betw4 a b c d* $\longleftrightarrow$ *betw4 d c b a*
using *abc-sym* **by** *blast*

lemma *abcd-dcba-only*:
fixes $a::'a$ **and** $b::'a$ **and** $c::'a$ **and** $d::'a$
assumes $[a;b;c;d]$
shows $\neg[a;b;d;c]$ $\neg[a;c;b;d]$ $\neg[a;c;d;b]$ $\neg[a;d;b;c]$ $\neg[a;d;c;b]$
$\neg[b;a;c;d]$ $\neg[b;a;d;c]$ $\neg[b;c;a;d]$ $\neg[b;c;d;a]$ $\neg[b;d;c;a]$ $\neg[b;d;a;c]$
$\neg[c;a;b;d]$ $\neg[c;a;d;b]$ $\neg[c;b;a;d]$ $\neg[c;b;d;a]$ $\neg[c;d;a;b]$ $\neg[c;d;b;a]$
$\neg[d;a;b;c]$ $\neg[d;a;c;b]$ $\neg[d;b;a;c]$ $\neg[d;b;c;a]$ $\neg[d;c;a;b]$
using *abc-only-cba assms* **by** *blast+*

lemma *some-betw4a*:
fixes $a::'a$ **and** $b::'a$ **and** $c::'a$ **and** $d::'a$ **and** P
assumes $P\in\mathcal{P}$ $a\in P$ $b\in P$ $c\in P$ $d\in P$ $a\neq b \wedge a\neq c \wedge a\neq d \wedge b\neq c \wedge b\neq d \wedge c\neq d$
and $\neg([a;b;c;d] \vee [a;b;d;c] \vee [a;c;b;d] \vee [a;c;d;b] \vee [a;d;b;c] \vee [a;d;c;b])$
shows $[b;a;c;d] \vee [b;a;d;c] \vee [b;c;a;d] \vee [b;d;a;c] \vee [c;a;b;d] \vee [c;b;a;d]$
by (*smt abc-bcd-acd abc-sym abd-bcd-abc assms some-betw-xor*)

lemma *some-betw4b*:
fixes $a::'a$ **and** $b::'a$ **and** $c::'a$ **and** $d::'a$ **and** P
assumes $P\in\mathcal{P}$ $a\in P$ $b\in P$ $c\in P$ $d\in P$ $a\neq b \wedge a\neq c \wedge a\neq d \wedge b\neq c \wedge b\neq d \wedge c\neq d$
and $\neg([b;a;c;d] \vee [b;a;d;c] \vee [b;c;a;d] \vee [b;d;a;c] \vee [c;a;b;d] \vee [c;b;a;d])$
shows $[a;b;c;d] \vee [a;b;d;c] \vee [a;c;b;d] \vee [a;c;d;b] \vee [a;d;b;c] \vee [a;d;c;b]$
by (*smt abc-bcd-acd abc-sym abd-bcd-abc assms some-betw-xor*)

lemma *abd-acd-abcdacbd*:
fixes $a::'a$ **and** $b::'a$ **and** $c::'a$ **and** $d::'a$
assumes *abd*: $[a;b;d]$ **and** *acd*: $[a;c;d]$ **and** $b\neq c$
shows $[a;b;c;d] \vee [a;c;b;d]$
proof −
obtain P **where** $P\in\mathcal{P}$ $a\in P$ $b\in P$ $d\in P$
using *abc-ex-path abd* **by** *blast*
have $c\in P$
using $\langle P \in \mathcal{P}\rangle$ $\langle a \in P\rangle$ $\langle d \in P\rangle$ *abc-abc-neq acd betw-b-in-path* **by** *blast*
have $\neg[b;d;c]$
using *abc-sym abcd-dcba-only(5) abd acd* **by** *blast*
hence $[b;c;d] \vee [c;b;d]$
using *abc-abc-neq abc-sym abd acd assms(3) some-betw*

by (*metis* ‹$P \in \mathcal{P}$› ‹$b \in P$› ‹$c \in P$› ‹$d \in P$›)
thus *?thesis*
using *abd acd betw4-weak* **by** *blast*
qed

end

32.2 WLOG for two general symmetric relations of two elements on a single path

context *MinkowskiBetweenness* **begin**

This first one is really just trying to get a hang of how to write these things. If you have a relation that does not care which way round the "endpoints" (if Q is the interval-relation) go, then anything you want to prove about both undistinguished endpoints, follows from a proof involving a single endpoint.

lemma *wlog-sym-element*:
assumes *symmetric-rel*: $\bigwedge a\ b\ I.\ Q\ I\ a\ b \Longrightarrow Q\ I\ b\ a$
and *one-endpoint*: $\bigwedge a\ b\ x\ I.\ [\![Q\ I\ a\ b;\ x{=}a]\!] \Longrightarrow P\ x\ I$
shows *other-endpoint*: $\bigwedge a\ b\ x\ I.\ [\![Q\ I\ a\ b;\ x{=}b]\!] \Longrightarrow P\ x\ I$
using *assms* **by** *fastforce*

This one gives the most pertinent case split: a proof involving e.g. an element of an interval must consider the edge case and the inside case.

lemma *wlog-element*:
assumes *symmetric-rel*: $\bigwedge a\ b\ I.\ Q\ I\ a\ b \Longrightarrow Q\ I\ b\ a$
and *one-endpoint*: $\bigwedge a\ b\ x\ I.\ [\![Q\ I\ a\ b;\ x{=}a]\!] \Longrightarrow P\ x\ I$
and *neither-endpoint*: $\bigwedge a\ b\ x\ I.\ [\![Q\ I\ a\ b;\ x{\in}I;\ (x{\neq}a \wedge x{\neq}b)]\!] \Longrightarrow P\ x\ I$
shows *any-element*: $\bigwedge x\ I.\ [\![x{\in}I;\ (\exists\ a\ b.\ Q\ I\ a\ b)]\!] \Longrightarrow P\ x\ I$
by (*metis assms*)

Summary of the two above. Use for early case splitting in proofs. Doesn't need P to be symmetric - the context in the conclusion is explicitly symmetric.

lemma *wlog-two-sets-element*:
assumes *symmetric-Q*: $\bigwedge a\ b\ I.\ Q\ I\ a\ b \Longrightarrow Q\ I\ b\ a$
and *case-split*: $\bigwedge a\ b\ c\ d\ x\ I\ J.\ [\![Q\ I\ a\ b;\ Q\ J\ c\ d]\!] \Longrightarrow$
$(x{=}a \vee x{=}c \longrightarrow P\ x\ I\ J) \wedge (\neg(x{=}a \vee x{=}b \vee x{=}c \vee x{=}d) \longrightarrow P\ x\ I\ J)$
shows $\bigwedge x\ I\ J.\ [\![\exists\ a\ b.\ Q\ I\ a\ b;\ \exists\ a\ b.\ Q\ J\ a\ b]\!] \Longrightarrow P\ x\ I\ J$
by (*smt case-split symmetric-Q*)

Now we start on the actual result of interest. First we assume the events are all distinct, and we deal with the degenerate possibilities after.

lemma *wlog-endpoints-distinct1*:
assumes *symmetric-Q*: $\bigwedge a\ b\ I.\ Q\ I\ a\ b \Longrightarrow Q\ I\ b\ a$
and $\bigwedge I\ J\ a\ b\ c\ d.\ [\![Q\ I\ a\ b;\ Q\ J\ c\ d;\ [a;b;c;d]]\!] \Longrightarrow P\ I\ J$
shows $\bigwedge I\ J\ a\ b\ c\ d.\ [\![Q\ I\ a\ b;\ Q\ J\ c\ d;$

$[b;a;c;d] \vee [a;b;d;c] \vee [b;a;d;c] \vee [d;c;b;a]]\!] \Longrightarrow P\ I\ J$
by (*meson abc-sym assms*(*2*) *symmetric-Q*)

lemma *wlog-endpoints-distinct2*:
assumes *symmetric-Q*: $\bigwedge a\ b\ I.\ Q\ I\ a\ b \Longrightarrow Q\ I\ b\ a$
and $\bigwedge I\ J\ a\ b\ c\ d.\ [\![Q\ I\ a\ b;\ Q\ J\ c\ d;\ [a;c;b;d]]\!] \Longrightarrow P\ I\ J$
shows $\bigwedge I\ J\ a\ b\ c\ d.\ [\![Q\ I\ a\ b;\ Q\ J\ c\ d;$
$[b;c;a;d] \vee [a;d;b;c] \vee [b;d;a;c] \vee [d;b;c;a]]\!] \Longrightarrow P\ I\ J$
by (*meson abc-sym assms*(*2*) *symmetric-Q*)

lemma *wlog-endpoints-distinct3*:
assumes *symmetric-Q*: $\bigwedge a\ b\ I.\ Q\ I\ a\ b \Longrightarrow Q\ I\ b\ a$
and *symmetric-P*: $\bigwedge I\ J.\ [\![\exists\ a\ b.\ Q\ I\ a\ b;\ \exists\ a\ b.\ Q\ J\ a\ b;\ P\ I\ J]\!] \Longrightarrow P\ J\ I$
and $\bigwedge I\ J\ a\ b\ c\ d.\ [\![Q\ I\ a\ b;\ Q\ J\ c\ d;\ [a;c;d;b]]\!] \Longrightarrow P\ I\ J$
shows $\bigwedge I\ J\ a\ b\ c\ d.\ [\![Q\ I\ a\ b;\ Q\ J\ c\ d;$
$[a;d;c;b] \vee [b;c;d;a] \vee [b;d;c;a] \vee [c;a;b;d]]\!] \Longrightarrow P\ I\ J$
by (*meson assms*)

lemma (**in** *MinkowskiSpacetime*) *wlog-endpoints-distinct4*:
fixes *Q*:: (*'a set*) ⇒ *'a* ⇒ *'a* ⇒ *bool*
and *P*:: (*'a set*) ⇒ (*'a set*) ⇒ *bool*
and *A*:: (*'a set*)
assumes *path-A*: $A \in \mathcal{P}$
and *symmetric-Q*: $\bigwedge a\ b\ I.\ Q\ I\ a\ b \Longrightarrow Q\ I\ b\ a$
and *Q-implies-path*: $\bigwedge a\ b\ I.\ [\![I \subseteq A;\ Q\ I\ a\ b]\!] \Longrightarrow b \in A \wedge a \in A$
and *symmetric-P*: $\bigwedge I\ J.\ [\![\exists\ a\ b.\ Q\ I\ a\ b;\ \exists\ a\ b.\ Q\ J\ a\ b;\ P\ I\ J]\!] \Longrightarrow P\ J\ I$
and $\bigwedge I\ J\ a\ b\ c\ d.$
$[\![Q\ I\ a\ b;\ Q\ J\ c\ d;\ I \subseteq A;\ J \subseteq A;\ [a;b;c;d] \vee [a;c;b;d] \vee [a;c;d;b]]\!] \Longrightarrow P\ I\ J$
shows $\bigwedge I\ J\ a\ b\ c\ d.\ [\![Q\ I\ a\ b;\ Q\ J\ c\ d;\ I \subseteq A;\ J \subseteq A;$
$a \neq b \wedge a \neq c \wedge a \neq d \wedge b \neq c \wedge b \neq d \wedge c \neq d]\!] \Longrightarrow P\ I\ J$
proof −
fix *I J a b c d*
assume *asm*: $Q\ I\ a\ b$ $Q\ J\ c\ d$ $I \subseteq A$ $J \subseteq A$
$a \neq b \wedge a \neq c \wedge a \neq d \wedge b \neq c \wedge b \neq d \wedge c \neq d$
have *endpoints-on-path*: $a \in A$ $b \in A$ $c \in A$ $d \in A$
using *Q-implies-path asm* **by** *blast*+
show *P I J*
proof (*cases*)
assume $[b;a;c;d] \vee [b;a;d;c] \vee [b;c;a;d] \vee$
$[b;d;a;c] \vee [c;a;b;d] \vee [c;b;a;d]$
then consider $[b;a;c;d]|[b;a;d;c]|[b;c;a;d]|$
$[b;d;a;c]|[c;a;b;d]|[c;b;a;d]$
by *linarith*
thus *P I J*
apply (*cases*)
apply (*metis*(*mono-tags*) *asm*(*1*−*4*) *assms*(*5*) *symmetric-Q*)+
apply (*metis asm*(*1*−*4*) *assms*(*4*,*5*))
by (*metis asm*(*1*−*4*) *assms*(*2*,*4*,*5*) *symmetric-Q*)
next

assume $\neg([b;a;c;d] \vee [b;a;d;c] \vee [b;c;a;d] \vee$
$[b;d;a;c] \vee [c;a;b;d] \vee [c;b;a;d])$
hence $[a;b;c;d] \vee [a;b;d;c] \vee [a;c;b;d] \vee$
$[a;c;d;b] \vee [a;d;b;c] \vee [a;d;c;b]$
using *some-betw4b* [**where** $P{=}A$ **and** $a{=}a$ **and** $b{=}b$ **and** $c{=}c$ **and** $d{=}d$]
using *endpoints-on-path asm path-A* **by** *simp*
then consider $[a;b;c;d] | [a;b;d;c] | [a;c;b;d] |$
$[a;c;d;b] | [a;d;b;c] | [a;d;c;b]$
by *linarith*
thus $P\ I\ J$
apply (*cases*)
by (*metis asm*(*1*−*4*) *assms*(*5*) *symmetric-Q*)+
qed
qed

lemma (**in** *MinkowskiSpacetime*) *wlog-endpoints-distinct′*:
assumes $A \in \mathcal{P}$
and $\bigwedge a\ b\ I.\ Q\ I\ a\ b \Longrightarrow Q\ I\ b\ a$
and $\bigwedge a\ b\ I.\ [\![I \subseteq A;\ Q\ I\ a\ b]\!] \Longrightarrow a \in A$
and $\bigwedge I\ J.\ [\![\exists\, a\ b.\ Q\ I\ a\ b;\ \exists\, a\ b.\ Q\ J\ a\ b;\ P\ I\ J]\!] \Longrightarrow P\ J\ I$
and $\bigwedge I\ J\ a\ b\ c\ d.$
$[\![Q\ I\ a\ b;\ Q\ J\ c\ d;\ I{\subseteq}A;\ J{\subseteq}A;\ betw4\ a\ b\ c\ d \vee betw4\ a\ c\ b\ d \vee betw4\ a\ c$
$d\ b]\!] \Longrightarrow P\ I\ J$
and $Q\ I\ a\ b$
and $Q\ J\ c\ d$
and $I \subseteq A$
and $J \subseteq A$
and $a \neq b\ a \neq c\ a \neq d\ b \neq c\ b \neq d\ c \neq d$
shows $P\ I\ J$
proof −
{
let $?R = (\lambda I.\ (\exists\, a\ b.\ Q\ I\ a\ b))$
have $\bigwedge I\ J.\ [\![?R\ I;\ ?R\ J;\ P\ I\ J]\!] \Longrightarrow P\ J\ I$
using *assms*(*4*) **by** *blast*
}
thus *?thesis*
using *wlog-endpoints-distinct4*
[**where** $P{=}P$ **and** $Q{=}Q$ **and** $A{=}A$ **and** $I{=}I$ **and** $J{=}J$ **and** $a{=}a$ **and** $b{=}b$
and $c{=}c$ **and** $d{=}d$]
by (*smt assms*(*1*−*3*,*5*−))
qed

lemma (**in** *MinkowskiSpacetime*) *wlog-endpoints-distinct*:
assumes *path-A*: $A{\in}\mathcal{P}$
and *symmetric-Q*: $\bigwedge a\ b\ I.\ Q\ I\ a\ b \Longrightarrow Q\ I\ b\ a$
and *Q-implies-path*: $\bigwedge a\ b\ I.\ [\![I{\subseteq}A;\ Q\ I\ a\ b]\!] \Longrightarrow b{\in}A \wedge a{\in}A$
and *symmetric-P*: $\bigwedge I\ J.\ [\![\exists\, a\ b.\ Q\ I\ a\ b;\ \exists\, a\ b.\ Q\ J\ a\ b;\ P\ I\ J]\!] \Longrightarrow P\ J\ I$
and $\bigwedge I\ J\ a\ b\ c\ d.$

$[\![Q\ I\ a\ b;\ Q\ J\ c\ d;\ I{\subseteq}A;\ J{\subseteq}A;\ [a;b;c;d] \lor [a;c;b;d] \lor [a;c;d;b]]\!] \Longrightarrow P\ I\ J$
shows $\bigwedge I\ J\ a\ b\ c\ d.\ [\![Q\ I\ a\ b;\ Q\ J\ c\ d;\ I{\subseteq}A;\ J{\subseteq}A;$
$a{\neq}b \land a{\neq}c \land a{\neq}d \land b{\neq}c \land b{\neq}d \land c{\neq}d]\!] \Longrightarrow P\ I\ J$
by (*smt* (*verit*, *ccfv-SIG*) *assms some-betw4b*)

lemma *wlog-endpoints-degenerate1*:
assumes *symmetric-Q*: $\bigwedge a\ b\ I.\ Q\ I\ a\ b \Longrightarrow Q\ I\ b\ a$
and *symmetric-P*: $\bigwedge I\ J.\ [\![\exists\, a\ b.\ Q\ I\ a\ b;\ \exists\, a\ b.\ Q\ I\ a\ b;\ P\ I\ J]\!] \Longrightarrow P\ J\ I$

and *two*: $\bigwedge I\ J\ a\ b\ c\ d.\ [\![Q\ I\ a\ b;\ Q\ J\ c\ d;$
$(a{=}b \land b{=}c \land c{=}d) \lor (a{=}b \land b{\neq}c \land c{=}d)]\!] \Longrightarrow P\ I\ J$

and *one*: $\bigwedge I\ J\ a\ b\ c\ d.\ [\![Q\ I\ a\ b;\ Q\ J\ c\ d;$
$(a{=}b \land b{=}c \land c{\neq}d) \lor (a{=}b \land b{\neq}c \land c{\neq}d \land a{\neq}d)]\!] \Longrightarrow P\ I\ J$

and *no*: $\bigwedge I\ J\ a\ b\ c\ d.\ [\![Q\ I\ a\ b;\ Q\ J\ c\ d;$
$(a{\neq}b \land b{\neq}c \land c{\neq}d \land a{=}d) \lor (a{\neq}b \land b{=}c \land c{\neq}d \land a{=}d)]\!] \Longrightarrow P\ I$
J
shows $\bigwedge I\ J\ a\ b\ c\ d.\ [\![Q\ I\ a\ b;\ Q\ J\ c\ d;\ \neg(a{\neq}b \land b{\neq}c \land c{\neq}d \land a{\neq}d \land a{\neq}c \land$
$b{\neq}d)]\!] \Longrightarrow P\ I\ J$
by (*metis assms*)

lemma *wlog-endpoints-degenerate2*:
assumes *symmetric-Q*: $\bigwedge a\ b\ I.\ Q\ I\ a\ b \Longrightarrow Q\ I\ b\ a$
and *Q-implies-path*: $\bigwedge a\ b\ I\ A.\ [\![I{\subseteq}A;\ A{\in}\mathcal{P};\ Q\ I\ a\ b]\!] \Longrightarrow b{\in}A \land a{\in}A$
and *symmetric-P*: $\bigwedge I\ J.\ [\![\exists\, a\ b.\ Q\ I\ a\ b;\ \exists\, a\ b.\ Q\ J\ a\ b;\ P\ I\ J]\!] \Longrightarrow P\ J\ I$
and $\bigwedge I\ J\ a\ b\ c\ d\ A.\ [\![Q\ I\ a\ b;\ Q\ J\ c\ d;\ I{\subseteq}A;\ J{\subseteq}A;\ A{\in}\mathcal{P};$
$[a;b;c] \land a{=}d]\!] \Longrightarrow P\ I\ J$
and $\bigwedge I\ J\ a\ b\ c\ d\ A.\ [\![Q\ I\ a\ b;\ Q\ J\ c\ d;\ I{\subseteq}A;\ J{\subseteq}A;\ A{\in}\mathcal{P};$
$[b;a;c] \land a{=}d]\!] \Longrightarrow P\ I\ J$
shows $\bigwedge I\ J\ a\ b\ c\ d\ A.\ [\![Q\ I\ a\ b;\ Q\ J\ c\ d;\ I{\subseteq}A;\ J{\subseteq}A;\ A{\in}\mathcal{P};$
$a{\neq}b \land b{\neq}c \land c{\neq}d \land a{=}d]\!] \Longrightarrow P\ I\ J$
proof −
have *last-case*: $\bigwedge I\ J\ a\ b\ c\ d\ A.\ [\![Q\ I\ a\ b;\ Q\ J\ c\ d;\ I{\subseteq}A;\ J{\subseteq}A;\ A{\in}\mathcal{P};$
$[b;c;a] \land a{=}d]\!] \Longrightarrow P\ I\ J$
using *assms*(*1*,*3*−*5*) **by** (*metis abc-sym*)
thus $\bigwedge I\ J\ a\ b\ c\ d\ A.\ [\![Q\ I\ a\ b;\ Q\ J\ c\ d;\ I{\subseteq}A;\ J{\subseteq}A;\ A{\in}\mathcal{P};$
$a{\neq}b \land b{\neq}c \land c{\neq}d \land a{=}d]\!] \Longrightarrow P\ I\ J$
by (*smt* (*z3*) *abc-sym assms*(*2*,*4*,*5*) *some-betw*)
qed

lemma *wlog-endpoints-degenerate*:
assumes *path-A*: $A{\in}\mathcal{P}$
and *symmetric-Q*: $\bigwedge a\ b\ I.\ Q\ I\ a\ b \Longrightarrow Q\ I\ b\ a$
and *Q-implies-path*: $\bigwedge a\ b\ I.\ [\![I{\subseteq}A;\ Q\ I\ a\ b]\!] \Longrightarrow b{\in}A \land a{\in}A$
and *symmetric-P*: $\bigwedge I\ J.\ [\![\exists\, a\ b.\ Q\ I\ a\ b;\ \exists\, a\ b.\ Q\ J\ a\ b;\ P\ I\ J]\!] \Longrightarrow P\ J\ I$
and $\bigwedge I\ J\ a\ b\ c\ d.\ [\![Q\ I\ a\ b;\ Q\ J\ c\ d;\ I{\subseteq}A;\ J{\subseteq}A]\!]$

$\Longrightarrow ((a{=}b \wedge b{=}c \wedge c{=}d) \longrightarrow P\ I\ J) \wedge ((a{=}b \wedge b{\neq}c \wedge c{=}d) \longrightarrow P\ I\ J)$
$\wedge ((a{=}b \wedge b{=}c \wedge c{\neq}d) \longrightarrow P\ I\ J) \wedge ((a{=}b \wedge b{\neq}c \wedge c{\neq}d \wedge a{\neq}d) \longrightarrow$
$P\ I\ J)$
$\wedge ((a{\neq}b \wedge b{=}c \wedge c{\neq}d \wedge a{=}d) \longrightarrow P\ I\ J)$
$\wedge (([a;b;c] \wedge a{=}d) \longrightarrow P\ I\ J) \wedge (([b;a;c] \wedge a{=}d) \longrightarrow P\ I\ J)$
shows $\bigwedge I\ J\ a\ b\ c\ d.\ [\![Q\ I\ a\ b;\ Q\ J\ c\ d;\ I{\subseteq}A;\ J{\subseteq}A;$
$\neg(a{\neq}b \wedge b{\neq}c \wedge c{\neq}d \wedge a{\neq}d \wedge a{\neq}c \wedge b{\neq}d)]\!] \Longrightarrow P\ I\ J$
proof −

We first extract some of the assumptions of this lemma into the form of other WLOG lemmas' assumptions.

have *ord1*: $\bigwedge I\ J\ a\ b\ c\ d.\ [\![Q\ I\ a\ b;\ Q\ J\ c\ d;\ I{\subseteq}A;\ J{\subseteq}A;$
$[a;b;c] \wedge a{=}d]\!] \Longrightarrow P\ I\ J$
using *assms*(*5*) **by** *auto*
have *ord2*: $\bigwedge I\ J\ a\ b\ c\ d.\ [\![Q\ I\ a\ b;\ Q\ J\ c\ d;\ I{\subseteq}A;\ J{\subseteq}A;$
$[b;a;c] \wedge a{=}d]\!] \Longrightarrow P\ I\ J$
using *assms*(*5*) **by** *auto*
have *last-case*: $\bigwedge I\ J\ a\ b\ c\ d.\ [\![Q\ I\ a\ b;\ Q\ J\ c\ d;\ I{\subseteq}A;\ J{\subseteq}A;$
$a{\neq}b \wedge b{\neq}c \wedge c{\neq}d \wedge a{=}d]\!] \Longrightarrow P\ I\ J$
using *ord1 ord2 wlog-endpoints-degenerate2 symmetric-P symmetric-Q Q-implies-path path-A*
by (*metis abc-sym some-betw*)
show $\bigwedge I\ J\ a\ b\ c\ d.\ [\![Q\ I\ a\ b;\ Q\ J\ c\ d;\ I{\subseteq}A;\ J{\subseteq}A;$
$\neg(a{\neq}b \wedge b{\neq}c \wedge c{\neq}d \wedge a{\neq}d \wedge a{\neq}c \wedge b{\neq}d)]\!] \Longrightarrow P\ I\ J$
proof −

Fix the sets on the path, and obtain the assumptions of *wlog-endpoints-degenerate1*.

fix *I J*
assume *asm1*: $I{\subseteq}A\ \ J{\subseteq}A$
have *two*: $\bigwedge a\ b\ c\ d.\ [\![Q\ I\ a\ b;\ Q\ J\ c\ d;\ a{=}b \wedge b{=}c \wedge c{=}d]\!] \Longrightarrow P\ I\ J$
$\bigwedge a\ b\ c\ d.\ [\![Q\ I\ a\ b;\ Q\ J\ c\ d;\ a{=}b \wedge b{\neq}c \wedge c{=}d]\!] \Longrightarrow P\ I\ J$
using ‹$J \subseteq A$› ‹$I \subseteq A$› *path-A assms*(*5*) **by** *blast+*
have *one*: $\bigwedge\ a\ b\ c\ d.\ [\![Q\ I\ a\ b;\ Q\ J\ c\ d;\ a{=}b \wedge b{=}c \wedge c{\neq}d]\!] \Longrightarrow P\ I\ J$
$\bigwedge\ a\ b\ c\ d.\ [\![Q\ I\ a\ b;\ Q\ J\ c\ d;\ a{=}b \wedge b{\neq}c \wedge c{\neq}d \wedge a{\neq}d]\!] \Longrightarrow P\ I\ J$
using ‹$I \subseteq A$› ‹$J \subseteq A$› *path-A assms*(*5*) **by** *blast+*
have *no*: $\bigwedge\ a\ b\ c\ d.\ [\![Q\ I\ a\ b;\ Q\ J\ c\ d;\ a{\neq}b \wedge b{\neq}c \wedge c{\neq}d \wedge a{=}d]\!] \Longrightarrow P\ I\ J$
$\bigwedge\ a\ b\ c\ d.\ [\![Q\ I\ a\ b;\ Q\ J\ c\ d;\ a{\neq}b \wedge b{=}c \wedge c{\neq}d \wedge a{=}d]\!] \Longrightarrow P\ I\ J$
using ‹$I \subseteq A$› ‹$J \subseteq A$› *path-A last-case* **apply** *blast*
using ‹$I \subseteq A$› ‹$J \subseteq A$› *path-A assms*(*5*) **by** *auto*

Now unwrap the remaining object logic and finish the proof.

fix *a b c d*
assume *asm2*: $Q\ I\ a\ b\ \ Q\ J\ c\ d\ \ \neg(a{\neq}b \wedge b{\neq}c \wedge c{\neq}d \wedge a{\neq}d \wedge a{\neq}c \wedge b{\neq}d)$
show *P I J*
using *two* [**where** $a{=}a$ **and** $b{=}b$ **and** $c{=}c$ **and** $d{=}d$]
using *one* [**where** $a{=}a$ **and** $b{=}b$ **and** $c{=}c$ **and** $d{=}d$]
using *no* [**where** $a{=}a$ **and** $b{=}b$ **and** $c{=}c$ **and** $d{=}d$]
using *wlog-endpoints-degenerate1*

[**where** *I*=*I* **and** *J*=*J* **and** *a*=*a* **and** *b*=*b* **and** *c*=*c* **and** *d*=*d* **and** *P*=*P* **and** *Q*=*Q*]
using *asm1 asm2 symmetric-P last-case assms*(*5*) *symmetric-Q*
by *smt*
qed
qed

lemma (**in** *MinkowskiSpacetime*) *wlog-intro*:
assumes *path-A*: $A \in \mathcal{P}$
and *symmetric-Q*: $\bigwedge a\ b\ I.\ Q\ I\ a\ b \Longrightarrow Q\ I\ b\ a$
and *Q-implies-path*: $\bigwedge a\ b\ I.\ [\![I \subseteq A;\ Q\ I\ a\ b]\!] \Longrightarrow b \in A \wedge a \in A$
and *symmetric-P*: $\bigwedge I\ J.\ [\![\exists\ a\ b.\ Q\ I\ a\ b;\ \exists\ c\ d.\ Q\ J\ c\ d;\ P\ I\ J]\!] \Longrightarrow P\ J\ I$
and *essential-cases*: $\bigwedge I\ J\ a\ b\ c\ d.\ [\![Q\ I\ a\ b;\ Q\ J\ c\ d;\ I \subseteq A;\ J \subseteq A]\!]$
$\Longrightarrow ((a=b \wedge b=c \wedge c=d) \longrightarrow P\ I\ J)$
$\wedge ((a=b \wedge b \neq c \wedge c=d) \longrightarrow P\ I\ J)$
$\wedge ((a=b \wedge b=c \wedge c \neq d) \longrightarrow P\ I\ J)$
$\wedge ((a=b \wedge b \neq c \wedge c \neq d \wedge a \neq d) \longrightarrow P\ I\ J)$
$\wedge ((a \neq b \wedge b=c \wedge c \neq d \wedge a=d) \longrightarrow P\ I\ J)$
$\wedge (([a;b;c] \wedge a=d) \longrightarrow P\ I\ J)$
$\wedge (([b;a;c] \wedge a=d) \longrightarrow P\ I\ J)$
$\wedge ([a;b;c;d] \longrightarrow P\ I\ J)$
$\wedge ([a;c;b;d] \longrightarrow P\ I\ J)$
$\wedge ([a;c;d;b] \longrightarrow P\ I\ J)$
and *antecedants*: $Q\ I\ a\ b\ Q\ J\ c\ d\ I \subseteq A\ J \subseteq A$
shows *P I J*
using *essential-cases antecedants*
and *wlog-endpoints-degenerate*[*OF path-A symmetric-Q Q-implies-path symmetric-P*]
and *wlog-endpoints-distinct*[*OF path-A symmetric-Q Q-implies-path symmetric-P*]
by (*smt* (*z3*) *Q-implies-path path-A symmetric-P symmetric-Q some-betw2 some-betw4b abc-only-cba*(*1*))

end

32.3 WLOG for two intervals

context *MinkowskiBetweenness* **begin**

This section just specifies the results for a generic relation Q in the previous section to the interval relation.

lemma *wlog-two-interval-element*:
assumes $\bigwedge x\ I\ J.\ [\![\textit{is-interval}\ I;\ \textit{is-interval}\ J;\ P\ x\ J\ I]\!] \Longrightarrow P\ x\ I\ J$
and $\bigwedge a\ b\ c\ d\ x\ I\ J.\ [\![I = \textit{interval}\ a\ b;\ J = \textit{interval}\ c\ d]\!] \Longrightarrow$
$(x=a \vee x=c \longrightarrow P\ x\ I\ J) \wedge (\neg(x=a \vee x=b \vee x=c \vee x=d) \longrightarrow P\ x\ I\ J)$
shows $\bigwedge x\ I\ J.\ [\![\textit{is-interval}\ I;\ \textit{is-interval}\ J]\!] \Longrightarrow P\ x\ I\ J$
by (*metis assms*(*2*) *int-sym*)

lemma (**in** *MinkowskiSpacetime*) *wlog-interval-endpoints-distinct*:
assumes $\bigwedge I\ J.\ [\![is\text{-}interval\ I;\ is\text{-}interval\ J;\ P\ I\ J]\!] \Longrightarrow P\ J\ I$
$\bigwedge I\ J\ a\ b\ c\ d.\ [\![I = interval\ a\ b;\ J = interval\ c\ d]\!]$
$\Longrightarrow ([a;b;c;d] \longrightarrow P\ I\ J) \wedge ([a;c;b;d] \longrightarrow P\ I\ J) \wedge ([a;c;d;b] \longrightarrow P\ I\ J)$
shows $\bigwedge I\ J\ Q\ a\ b\ c\ d.\ [\![I = interval\ a\ b;\ J = interval\ c\ d;\ I \subseteq Q;\ J \subseteq Q;\ Q \in \mathcal{P};$
$a \neq b \wedge a \neq c \wedge a \neq d \wedge b \neq c \wedge b \neq d \wedge c \neq d]\!] \Longrightarrow P\ I\ J$
proof −
let $?Q = \lambda\ I\ a\ b.\ I = interval\ a\ b$

fix $I\ J\ A\ a\ b\ c\ d$
assume *asm*: $?Q\ I\ a\ b\ ?Q\ J\ c\ d\ I \subseteq A\ J \subseteq A\ A \in \mathcal{P}\ a \neq b \wedge a \neq c \wedge a \neq d \wedge b \neq c \wedge b \neq d \wedge c \neq d$
show $P\ I\ J$
proof (*rule wlog-endpoints-distinct*)
show $\bigwedge a\ b\ I.\ ?Q\ I\ a\ b \Longrightarrow ?Q\ I\ b\ a$
by (*simp add*: *int-sym*)
show $\bigwedge a\ b\ I.\ I \subseteq A \Longrightarrow ?Q\ I\ a\ b \Longrightarrow b \in A \wedge a \in A$
by (*simp add*: *ends-in-int subset-iff*)
show $\bigwedge I\ J.\ is\text{-}interval\ I \Longrightarrow is\text{-}interval\ J \Longrightarrow P\ I\ J \Longrightarrow P\ J\ I$
using *assms*(*1*) **by** *blast*
show $\bigwedge I\ J\ a\ b\ c\ d.\ [\![?Q\ I\ a\ b;\ ?Q\ J\ c\ d;\ [a;b;c;d] \vee [a;c;b;d] \vee [a;c;d;b]]\!]$
$\Longrightarrow P\ I\ J$
by (*meson assms*(*2*))
show $I = interval\ a\ b\ J = interval\ c\ d\ I \subseteq A\ J \subseteq A\ A \in \mathcal{P}$
$a \neq b \wedge a \neq c \wedge a \neq d \wedge b \neq c \wedge b \neq d \wedge c \neq d$
using *asm* **by** *simp+*
qed
qed

lemma *wlog-interval-endpoints-degenerate*:
assumes *symmetry*: $\bigwedge I\ J.\ [\![is\text{-}interval\ I;\ is\text{-}interval\ J;\ P\ I\ J]\!] \Longrightarrow P\ J\ I$
and $\bigwedge I\ J\ a\ b\ c\ d\ Q.\ [\![I = interval\ a\ b;\ J = interval\ c\ d;\ I \subseteq Q;\ J \subseteq Q;\ Q \in \mathcal{P}]\!]$
$\Longrightarrow ((a=b \wedge b=c \wedge c=d) \longrightarrow P\ I\ J) \wedge ((a=b \wedge b \neq c \wedge c=d) \longrightarrow P\ I\ J)$
$\wedge ((a=b \wedge b=c \wedge c \neq d) \longrightarrow P\ I\ J) \wedge ((a=b \wedge b \neq c \wedge c \neq d \wedge a \neq d) \longrightarrow P\ I\ J)$
$\wedge ((a \neq b \wedge b=c \wedge c \neq d \wedge a=d) \longrightarrow P\ I\ J)$
$\wedge (([a;b;c] \wedge a=d) \longrightarrow P\ I\ J) \wedge (([b;a;c] \wedge a=d) \longrightarrow P\ I\ J)$
shows $\bigwedge I\ J\ a\ b\ c\ d\ Q.\ [\![I = interval\ a\ b;\ J = interval\ c\ d;\ I \subseteq Q;\ J \subseteq Q;\ Q \in \mathcal{P};$
$\neg(a \neq b \wedge b \neq c \wedge c \neq d \wedge a \neq d \wedge a \neq c \wedge b \neq d)]\!] \Longrightarrow P\ I\ J$
proof −
let $?Q = \lambda\ I\ a\ b.\ I = interval\ a\ b$

fix $I\ J\ a\ b\ c\ d\ A$
assume *asm*: $?Q\ I\ a\ b\ ?Q\ J\ c\ d\ I \subseteq A\ J \subseteq A\ A \in \mathcal{P}\ \neg(a \neq b \wedge b \neq c \wedge c \neq d \wedge a \neq d \wedge a \neq c \wedge b \neq d)$
show $P\ I\ J$

proof (*rule wlog-endpoints-degenerate*)
  show $\bigwedge a\ b\ I.\ ?Q\ I\ a\ b \Longrightarrow\ ?Q\ I\ b\ a$
    by (*simp add*: *int-sym*)
  show $\bigwedge a\ b\ I.\ I \subseteq A \Longrightarrow\ ?Q\ I\ a\ b \Longrightarrow b \in A \wedge a \in A$
    by (*simp add*: *ends-in-int subset-iff*)
  show $\bigwedge I\ J.$ *is-interval* $I \Longrightarrow$ *is-interval* $J \Longrightarrow P\ I\ J \Longrightarrow P\ J\ I$
    using *symmetry* **by** *blast*
  show $I =$ *interval* $a\ b\ J =$ *interval* $c\ d\ I{\subseteq}A\ J{\subseteq}A\ A{\in}\mathcal{P}$
    $\neg\ (a{\neq}b \wedge b{\neq}c \wedge c{\neq}d \wedge a{\neq}d \wedge a{\neq}c \wedge b{\neq}d)$
    using *asm* **by** *auto+*
  show $\bigwedge I\ J\ a\ b\ c\ d.\ [\![?Q\ I\ a\ b;\ ?Q\ J\ c\ d;\ I \subseteq A;\ J \subseteq A]\!] \Longrightarrow$
      $(a = b \wedge b = c \wedge c = d \longrightarrow P\ I\ J) \wedge$
      $(a = b \wedge b \neq c \wedge c = d \longrightarrow P\ I\ J) \wedge$
      $(a = b \wedge b = c \wedge c \neq d \longrightarrow P\ I\ J) \wedge$
      $(a = b \wedge b \neq c \wedge c \neq d \wedge a \neq d \longrightarrow P\ I\ J) \wedge$
      $(a \neq b \wedge b = c \wedge c \neq d \wedge a = d \longrightarrow P\ I\ J) \wedge$
      $([a;b;c] \wedge a = d \longrightarrow P\ I\ J) \wedge ([b;a;c] \wedge a = d \longrightarrow P\ I\ J)$
    using *assms*(*2*) ‹$A{\in}\mathcal{P}$› **by** *auto*
qed
qed

end

33 Interlude: Intervals, Segments, Connectedness

context *MinkowskiSpacetime* **begin**

In this secion, we apply the WLOG lemmas from the previous section in order to reduce the number of cases we need to consider when thinking about two arbitrary intervals on a path. This is used to prove that the (countable) intersection of intervals is an interval. These results cannot be found in Schutz, but he does use them (without justification) in his proof of Theorem 12 (even for uncountable intersections).

lemma *int-of-ints-is-interval-neq*:
  assumes $I1 =$ *interval* $a\ b\ I2 =$ *interval* $c\ d\ I1{\subseteq}P\ I2{\subseteq}P\ P{\in}\mathcal{P}\ I1{\cap}I2 \neq \{\}$
    and *events-neq*: $a{\neq}b\ a{\neq}c\ a{\neq}d\ b{\neq}c\ b{\neq}d\ c{\neq}d$
  shows *is-interval* $(I1 \cap I2)$
proof −
  have *on-path*: $a{\in}P \wedge b{\in}P \wedge c{\in}P \wedge d{\in}P$
    using *assms*(*1*−*4*) *interval-def* **by** *auto*

  let $?prop = \lambda\ I\ J.$ *is-interval* $(I{\cap}J) \vee (I{\cap}J) = \{\}$

  have *symmetry*: $(\bigwedge I\ J.$ *is-interval* $I \Longrightarrow$ *is-interval* $J \Longrightarrow\ ?prop\ I\ J \Longrightarrow\ ?prop$ $J\ I)$
    by (*simp add*: *Int-commute*)

  {

```
fix I J a b c d
assume I = interval a b J = interval c d
have ([a;b;c;d] ⟶ ?prop I J)
     ([a;c;b;d] ⟶ ?prop I J)
     ([a;c;d;b] ⟶ ?prop I J)
proof (rule-tac [!] impI)
  assume betw4 a b c d
  have I∩J = {}
  proof (rule ccontr)
    assume I∩J≠{}
    then obtain x where x∈I∩J
      by blast
    show False
    proof (cases)
      assume x≠a ∧ x≠b ∧ x≠c ∧ x≠d
      hence [a;x;b] [c;x;d]
        using ‹I=interval a b› ‹x∈I∩J› ‹J=interval c d› ‹x∈I∩J›
        by (simp add: interval-def seg-betw)+
      thus False
        by (meson ‹betw4 a b c d› abc-only-cba(3) abc-sym abd-bcd-abc)
    next
      assume ¬(x≠a ∧ x≠b ∧ x≠c ∧ x≠d)
      thus False
           using interval-def seg-betw ‹I = interval a b› ‹J = interval c d›
abcd-dcba-only(21)
           ‹x ∈ I ∩ J› ‹betw4 a b c d› abc-bcd-abd abc-bcd-acd abc-only-cba(1,2)
        by (metis (full-types) insert-iff Int-iff)
    qed
  qed
  thus ?prop I J by simp
next
  assume [a;c;b;d]
  then have a ≠ b ∧ a ≠ c ∧ a ≠ d ∧ b ≠ c ∧ b ≠ d ∧ c ≠ d
    using betw4-imp-neq by blast
  have I∩J = interval c b
  proof (safe)
    fix x
    assume x ∈ interval c b
    {
      assume x=b ∨ x=c
      hence x∈I
        using ‹[a;c;b;d]› ‹I = interval a b› interval-def seg-betw by auto
      have x∈J
        using ‹x=b ∨ x=c›
        using ‹[a;c;b;d]› ‹J = interval c d› interval-def seg-betw by auto
      hence x∈I ∧ x∈J using ‹x ∈ I› by blast
    } moreover {
      assume ¬(x=b ∨ x=c)
      hence [c;x;b]
```

using ‹*x*∈*interval c b*› **unfolding** *interval-def segment-def* **by** *simp*
hence [*a*;*x*;*b*]
by (*meson* ‹[*a*;*c*;*b*;*d*]› *abc-acd-abd abc-sym*)
have [*c*;*x*;*d*]
using ‹[*a*;*c*;*b*;*d*]› ‹[*c*;*x*;*b*]› *abc-acd-abd* **by** *blast*
have *x*∈*I* *x*∈*J*
using ‹*I* = *interval a b*› ‹[*a*;*x*;*b*]› ‹*J* = *interval c d*› ‹[*c*;*x*;*d*]›
interval-def seg-betw **by** *auto*
}
ultimately show *x*∈*I* *x*∈*J* **by** *blast*+
next
fix *x*
assume *x*∈*I* *x*∈*J*
show *x* ∈ *interval c b*
proof (*cases*)
assume *not-eq*: *x*≠*a* ∧ *x*≠*b* ∧ *x*≠*c* ∧ *x*≠*d*
have [*a*;*x*;*b*] [*c*;*x*;*d*]
using ‹*x*∈*I*› ‹*I* = *interval a b*› ‹*x*∈*J*› ‹*J* = *interval c d*›
not-eq **unfolding** *interval-def segment-def* **by** *blast*+
hence [*c*;*x*;*b*]
by (*meson* ‹[*a*;*c*;*b*;*d*]› *abc-bcd-acd betw4-weak*)
thus *?thesis*
unfolding *interval-def segment-def* **using** *seg-betw segment-def* **by** *auto*
next
assume *not-not-eq*: ¬(*x*≠*a* ∧ *x*≠*b* ∧ *x*≠*c* ∧ *x*≠*d*)
{
assume *x*=*a*
have ¬[*d*;*a*;*c*]
using ‹[*a*;*c*;*b*;*d*]› *abcd-dcba-only*(*9*) **by** *blast*
hence *a* ∉ *interval c d* **unfolding** *interval-def segment-def*
using *abc-sym* ‹*a* ≠ *b* ∧ *a* ≠ *c* ∧ *a* ≠ *d* ∧ *b* ≠ *c* ∧ *b* ≠ *d* ∧ *c* ≠ *d*› **by**
blast
hence *False* **using** ‹*x*∈*J*› ‹*J* = *interval c d*› ‹*x*=*a*› **by** *blast*
} moreover {
assume *x*=*d*
have ¬[*a*;*d*;*b*] **using** ‹*betw4 a c b d*› *abc-sym abcd-dcba-only*(*9*) **by** *blast*
hence *d*∉*interval a b* **unfolding** *interval-def segment-def*
using ‹*a* ≠ *b* ∧ *a* ≠ *c* ∧ *a* ≠ *d* ∧ *b* ≠ *c* ∧ *b* ≠ *d* ∧ *c* ≠ *d*› **by** *blast*
hence *False* **using** ‹*x*∈*I*› ‹*x*=*d*› ‹*I* = *interval a b*› **by** *blast*
}
ultimately show *?thesis*
using *interval-def not-not-eq* **by** *auto*
qed
qed
thus *?prop I J* **by** *auto*
next
assume [*a*;*c*;*d*;*b*]
have *I*∩*J* = *interval c d*
proof (*safe*)

```
        fix x
        assume x ∈ interval c d
        {
          assume x≠c ∧ x≠d
          have x ∈ J
            by (simp add: ‹J = interval c d› ‹x ∈ interval c d›)
          have [c;x;d]
            using ‹x ∈ interval c d› ‹x ≠ c ∧ x ≠ d› interval-def seg-betw by auto
          have [a;x;b]
           by (meson ‹betw4 a c d b› ‹[c;x;d]› abc-bcd-abd abc-sym abe-ade-bcd-ace)
          have x ∈ I
            using ‹I = interval a b› ‹[a;x;b]› interval-def seg-betw by auto
          hence x∈I ∧ x∈J by (simp add: ‹x ∈ J›)
        } moreover {
          assume ¬ (x≠c ∧ x≠d)
          hence x∈I ∧ x∈J
            by (metis ‹I = interval a b› ‹J = interval c d› ‹[a;c;d;b]› ‹x ∈ interval
c d›
                abc-bcd-abd abc-bcd-acd insertI2 interval-def seg-betw)
        }
        ultimately show x∈I x∈J by blast+
      next
        fix x
        assume x∈I x∈J
        show x ∈ interval c d
          using ‹J = interval c d› ‹x ∈ J› by auto
      qed
      thus ?prop I J by auto
    qed
  }

  then show is-interval (I1∩I2)
    using wlog-interval-endpoints-distinct
      [where P=?prop and I=I1 and J=I2 and Q=P and a=a and b=b and
c=c and d=d]
    using symmetry assms by simp
qed


lemma int-of-ints-is-interval-deg:
  assumes I = interval a b J = interval c d I∩J ≠ {} I⊆P J⊆P P∈𝒫
      and events-deg: ¬(a≠b ∧ b≠c ∧ c≠d ∧ a≠d ∧ a≠c ∧ b≠d)
    shows is-interval (I ∩ J)
proof −

  let ?p = λ I J. (is-interval (I ∩ J) ∨ I∩J = {})

  have symmetry: ⋀I J. ⟦is-interval I; is-interval J; ?p I J⟧ ⟹ ?p J I
    by (simp add: inf-commute)
```

have *degen-cases*: $\bigwedge I\ J\ a\ b\ c\ d\ Q.\ [\![I = interval\ a\ b;\ J = interval\ c\ d;\ I \subseteq Q;$ $J \subseteq Q;\ Q \in \mathcal{P}]\!]$
$\Longrightarrow ((a=b \wedge b=c \wedge c=d) \longrightarrow ?p\ I\ J) \wedge ((a=b \wedge b \neq c \wedge c=d) \longrightarrow ?p\ I$ $J)$
$\wedge ((a=b \wedge b=c \wedge c \neq d) \longrightarrow ?p\ I\ J) \wedge ((a=b \wedge b \neq c \wedge c \neq d \wedge a \neq d)$ $\longrightarrow ?p\ I\ J)$
$\wedge ((a \neq b \wedge b=c \wedge c \neq d \wedge a=d) \longrightarrow ?p\ I\ J)$
$\wedge (([a;b;c] \wedge a=d) \longrightarrow ?p\ I\ J) \wedge (([b;a;c] \wedge a=d) \longrightarrow ?p\ I\ J)$
proof −
fix $I\ J\ a\ b\ c\ d\ Q$
assume $I = interval\ a\ b\ J = interval\ c\ d\ I \subseteq Q\ J \subseteq Q\ Q \in \mathcal{P}$
show $((a=b \wedge b=c \wedge c=d) \longrightarrow ?p\ I\ J) \wedge ((a=b \wedge b \neq c \wedge c=d) \longrightarrow ?p\ I\ J)$
$\wedge ((a=b \wedge b=c \wedge c \neq d) \longrightarrow ?p\ I\ J) \wedge ((a=b \wedge b \neq c \wedge c \neq d \wedge a \neq d)$ $\longrightarrow ?p\ I\ J)$
$\wedge ((a \neq b \wedge b=c \wedge c \neq d \wedge a=d) \longrightarrow ?p\ I\ J)$
$\wedge (([a;b;c] \wedge a=d) \longrightarrow ?p\ I\ J) \wedge (([b;a;c] \wedge a=d) \longrightarrow ?p\ I\ J)$
proof (*intro conjI impI*)
assume $a = b \wedge b = c \wedge c = d$ **thus** $?p\ I\ J$
using ‹$I = interval\ a\ b$› ‹$J = interval\ c\ d$› **by** *auto*
next
assume $a = b \wedge b \neq c \wedge c = d$ **thus** $?p\ I\ J$
using ‹$J = interval\ c\ d$› *empty-segment interval-def* **by** *auto*
next
assume $a = b \wedge b = c \wedge c \neq d$ **thus** $?p\ I\ J$
using ‹$I = interval\ a\ b$› *empty-segment interval-def* **by** *auto*
next
assume $a = b \wedge b \neq c \wedge c \neq d \wedge a \neq d$ **thus** $?p\ I\ J$
using ‹$I = interval\ a\ b$› *empty-segment interval-def* **by** *auto*
next
assume $a \neq b \wedge b = c \wedge c \neq d \wedge a = d$ **thus** $?p\ I\ J$
using ‹$I = interval\ a\ b$› ‹$J = interval\ c\ d$› *int-sym* **by** *auto*
next
assume $[a;b;c] \wedge a = d$ **show** $?p\ I\ J$
proof (*cases*)
assume $I \cap J = \{\}$ **thus** *?thesis* **by** *simp*
next
assume $I \cap J \neq \{\}$
have $I \cap J = interval\ a\ b$
proof (*safe*)
fix x **assume** $x \in I\ x \in J$
thus $x \in interval\ a\ b$
using ‹$I = interval\ a\ b$› **by** *blast*
next
fix x **assume** $x \in interval\ a\ b$
show $x \in I$
by (*simp add*: ‹$I = interval\ a\ b$› ‹$x \in interval\ a\ b$›)
have $[d;b;c]$
using ‹$[a;b;c] \wedge a = d$› **by** *blast*

```
          have [a;x;b] ∨ x=a ∨ x=b
            using ‹I = interval a b› ‹x ∈ I› interval-def seg-betw by auto
          consider [d;x;c]|x=a ∨ x=b
            using ‹[a;b;c] ∧ a = d› ‹[a;x;b] ∨ x = a ∨ x = b› abc-acd-abd by blast
          thus x∈J
          proof (cases)
            case 1
            then show ?thesis
                 by (simp add: ‹J = interval c d› abc-abc-neq abc-sym interval-def
seg-betw)
          next
            case 2
            then have x ∈ interval c d
              using ‹[a;b;c] ∧ a = d› int-sym interval-def seg-betw
              by force
            then show ?thesis
              using ‹J = interval c d› by blast
          qed
        qed
        thus ?p I J by blast
      qed
    next
      assume [b;a;c] ∧ a = d show ?p I J
      proof (cases)
        assume I∩J = {} thus ?thesis by simp
      next
        assume I∩J ≠ {}
        have I∩J = {a}
        proof (safe)
          fix x assume x∈I x∈J x∉{}
          have cxd: [c;x;d] ∨ x=c ∨ x=d
            using ‹J = interval c d› ‹x ∈ J› interval-def seg-betw by auto
          consider [a;x;b]|x=a|x=b
            using ‹I = interval a b› ‹x ∈ I› interval-def seg-betw by auto
          then show x=a
          proof (cases)
            assume [a;x;b]
            hence [b;x;d;c]
              using ‹[b;a;c] ∧ a = d› abc-acd-bcd abc-sym by meson
            hence False
              using cxd abc-abc-neq by blast
            thus ?thesis by simp
          next
            assume x=b
            hence [b;d;c]
              using ‹[b;a;c] ∧ a = d› by blast
            hence False
              using cxd ‹x = b› abc-abc-neq by blast
            thus ?thesis
```

```
            by simp
          next
            assume x=a thus x=a by simp
          qed
        next
          show a∈I
            by (simp add: ‹I = interval a b› ends-in-int)
          show a∈J
            by (simp add: ‹J = interval c d› ‹[b;a;c] ∧ a = d› ends-in-int)
        qed
        thus ?p I J
          by (simp add: empty-segment interval-def)
      qed
    qed
  qed

  have ?p I J
    using wlog-interval-endpoints-degenerate
      [where P=?p and I=I and J=J and a=a and b=b and c=c and d=d
and Q=P]
    using degen-cases
    using symmetry assms
    by smt

  thus ?thesis
    using assms(3) by blast
qed


lemma int-of-ints-is-interval:
  assumes is-interval I is-interval J I⊆P J⊆P P∈𝒫 I∩J ≠ {}
  shows is-interval (I ∩ J)
  using int-of-ints-is-interval-neq int-of-ints-is-interval-deg
  by (meson assms)


lemma int-of-ints-is-interval2:
  assumes ∀x∈S. (is-interval x ∧ x⊆P) P∈𝒫 ⋂S ≠ {} finite S S≠{}
  shows is-interval (⋂S)
proof −
  obtain n where n = card S
    by simp
  consider n=0|n=1|n≥2
    by linarith
  thus ?thesis
  proof (cases)
    assume n=0
    then have False
      using ‹n = card S› assms(4,5) by simp
```

thus *?thesis*
by *simp*
next
assume $n=1$
then obtain I **where** $S = \{I\}$
using ‹$n = card\ S$› *card-1-singletonE* **by** *auto*
then have $\bigcap S = I$
by *simp*
moreover have *is-interval I*
by (*simp add*: ‹$S = \{I\}$› *assms*(*1*))
ultimately show *?thesis*
by *blast*
next
assume $2 \le n$
obtain m **where** $m+2=n$
using ‹$2 \le n$› *le-add-diff-inverse2* **by** *blast*
have *ind*: $\bigwedge S.\ [\![\forall x \in S.\ (is\text{-}interval\ x \wedge x \subseteq P);\ P \in \mathcal{P};\ \bigcap S \neq \{\};\ finite\ S;\ S \neq \{\};$
$m+2=card\ S]\!]$
$\Longrightarrow\ is\text{-}interval\ (\bigcap S)$
proof (*induct m*)
case *0*
then have $card\ S = 2$
by *auto*
then obtain $I\ J$ **where** $S=\{I,J\}\ I \neq J$
by (*meson card-2-iff*)
then have $I \in S\ J \in S$
by *blast+*
then have *is-interval I is-interval J* $I \subseteq P\ J \subseteq P$
by (*simp add*: *0.prems*(*1*))+
also have $I \cap J \neq \{\}$
using ‹$S=\{I,J\}$› *0.prems*(*3*) **by** *force*
then have *is-interval*($I \cap J$)
using *assms*(*2*) *calculation int-of-ints-is-interval*[**where** $I=I$ **and** $J=J$ **and**
$P=P$]
by *fastforce*
then show *?case*
by (*simp add*: ‹$S = \{I, J\}$›)
next
case (*Suc m*)
obtain $S'\ I$ **where** $I \in S\ S = insert\ I\ S'\ I \notin S'$
using *Suc.prems*(*4*,*5*) **by** (*metis Set.set-insert finite.simps insertI1*)
then have *is-interval* ($\bigcap S'$)
proof −
have $m+2 = card\ S'$
using *Suc.prems*(*4*,*6*) ‹$S = insert\ I\ S'$› ‹$I \notin S'$› **by** *auto*
moreover have $\forall x \in S'.\ is\text{-}interval\ x \wedge x \subseteq P$
by (*simp add*: *Suc.prems*(*1*) ‹$S = insert\ I\ S'$›)
moreover have $\bigcap\ S' \neq \{\}$
using *Suc.prems*(*3*) ‹$S = insert\ I\ S'$› **by** *auto*

moreover have *finite* S'
using *Suc.prems(4)* ‹$S = insert\ I\ S'$› **by** *auto*
ultimately show *?thesis*
using *assms(2) Suc(1)* [**where** $S=S'$] **by** *fastforce*
qed
then have *is-interval* $((\bigcap S') \cap I)$
proof (*rule int-of-ints-is-interval*)
show *is-interval* I
by (*simp add*: *Suc.prems(1)* ‹$I \in S$›)
show $\bigcap S' \subseteq P$
using ‹$I \notin S'$› ‹$S = insert\ I\ S'$› *Suc.prems(1,4,6) Inter-subset*
by (*metis Suc-n-not-le-n card.empty card-insert-disjoint finite-insert le-add2 numeral-2-eq-2 subset-eq subset-insertI*)
show $I \subseteq P$
by (*simp add*: *Suc.prems(1)* ‹$I \in S$›)
show $P \in \mathcal{P}$
using *assms(2)* **by** *auto*
show $\bigcap S' \cap I \neq \{\}$
using *Suc.prems(3)* ‹$S = insert\ I\ S'$› **by** *auto*
qed
thus *?case*
using ‹$S = insert\ I\ S'$› **by** (*simp add*: *inf.commute*)
qed
then show *?thesis*
using ‹$m + 2 = n$› ‹$n = card\ S$› *assms* **by** *blast*
qed
qed

end

34 3.7 Continuity and the monotonic sequence property

context *MinkowskiSpacetime* **begin**

This section only includes a proof of the first part of Theorem 12, as well as some results that would be useful in proving part (ii).

theorem *two-rays*:
assumes *path-Q*: $Q \in \mathcal{P}$
and *event-a*: $a \in Q$
shows $\exists R\ L.$ (*is-ray-on* $R\ Q$ $\wedge$ *is-ray-on* $L\ Q$
$\wedge\ Q - \{a\} \subseteq (R \cup L)$ ~~events of Q excl. a belong to two rays~~
$\wedge\ (\forall r \in R.\ \forall l \in L.\ [l;a;r])$ ~~a is betw. any l of one ray and any r of the other~~
$\wedge\ (\forall x \in R.\ \forall y \in R.\ \neg\ [x;a;y])$ ~~but a is not betw. any two events~~
$\wedge\ (\forall x \in L.\ \forall y \in L.\ \neg\ [x;a;y]))$ ~~of the same ray~~
proof –

Schutz here uses Theorem 6, but we don't need it.

```
obtain b where b∈ℰ and b∈Q and b≠a
  using event-a ge2-events in-path-event path-Q by blast
let ?L = {x. [x;a;b]}
let ?R = {y. [a;y;b] ∨ [a;b;y]]}
have Q = ?L ∪ {a} ∪ ?R
proof −
  have inQ: ∀x∈Q. [x;a;b] ∨ x=a ∨ [a;x;b] ∨ [a;b;x]]
    by (meson ‹b ∈ Q› ‹b ≠ a› abc-sym event-a path-Q some-betw)
  show ?thesis
  proof (safe)
    fix x
    assume x ∈ Q x ≠ a ¬ [x;a;b] ¬ [a;x;b] b ≠ x
    then show [a;b;x]
      using inQ by blast
  next
    fix x
    assume [x;a;b]
    then show x ∈ Q
      by (simp add: ‹b ∈ Q› abc-abc-neq betw-a-in-path event-a path-Q)
  next
    show a ∈ Q
      by (simp add: event-a)
  next
    fix x
    assume [a;x;b]
    then show  x ∈ Q
      by (simp add: ‹b ∈ Q› abc-abc-neq betw-b-in-path event-a path-Q)
  next
    fix x
    assume [a;b;x]
    then show  x ∈ Q
      by (simp add: ‹b ∈ Q› abc-abc-neq betw-c-in-path event-a path-Q)
  next
    show b ∈ Q using ‹b ∈ Q› .
  qed
qed
have disjointLR: ?L ∩ ?R = {}
  using abc-abc-neq abc-only-cba by blast

have wxyz-ord: [x;a;y;b]] ∨ [x;a;b;y]]
    ∧ (([w;x;a] ∧ [x;a;y]) ∨ ([x;w;a] ∧ [w;a;y]))
    ∧ (([x;a;y] ∧ [a;y;z]) ∨ ([x;a;z] ∧ [a;z;y]))
  if x∈?L w∈?L y∈?R z∈?R w≠x y≠z for x w y z
  using path-finsubset-chain order-finite-chain
  by (smt abc-abd-bcdbdc abc-bcd-abd abc-sym abd-bcd-abc mem-Collect-eq that)

obtain x y where x∈?L y∈?R
 by (metis (mono-tags) ‹b ∈ Q› ‹b ≠ a› abc-sym event-a mem-Collect-eq path-Q
```

prolong-betw2)
obtain *w* **where** *w*∈*?L w*≠*x*
by (*metis* ‹*b* ∈ *Q*› ‹*b* ≠ *a*› *abc-sym event-a mem-Collect-eq path-Q prolong-betw3*)
obtain *z* **where** *z*∈*?R y*≠*z*
by (*metis* (*mono-tags*) ‹*b* ∈ *Q*› ‹*b* ≠ *a*› *event-a mem-Collect-eq path-Q prolong-betw3*)

have *is-ray-on ?R Q* ∧
is-ray-on ?L Q ∧
Q − {*a*} ⊆ *?R* ∪ *?L* ∧
(∀ *r*∈*?R*. ∀ *l*∈*?L*. [*l*;*a*;*r*]) ∧
(∀ *x*∈*?R*. ∀ *y*∈*?R*. ¬ [*x*;*a*;*y*]) ∧
(∀ *x*∈*?L*. ∀ *y*∈*?L*. ¬ [*x*;*a*;*y*])
proof (*intro conjI*)
show *is-ray-on ?L Q*
proof (*unfold is-ray-on-def*, *safe*)
show *Q* ∈ 𝒫
by (*simp add*: *path-Q*)
next
fix *x*
assume [*x*;*a*;*b*]
then show *x* ∈ *Q*
using ‹*b* ∈ *Q*› ‹*b* ≠ *a*› *betw-a-in-path event-a path-Q* **by** *blast*
next
show *is-ray* {*x*. [*x*;*a*;*b*]}
proof −
have [*x*;*a*;*b*]
using ‹*x*∈*?L*› **by** *simp*
have *?L* = *ray a x*
proof
show *ray a x* ⊆ *?L*
proof
fix *e* **assume** *e*∈*ray a x*
show *e*∈*?L*
using *wxyz-ord ray-cases abc-bcd-abd abd-bcd-abc abc-sym*
by (*metis* ‹[*x*;*a*;*b*]› ‹*e* ∈ *ray a x*› *mem-Collect-eq*)
qed
show *?L* ⊆ *ray a x*
proof
fix *e* **assume** *e*∈*?L*
hence [*e*;*a*;*b*]
by *simp*
show *e*∈*ray a x*
proof (*cases*)
assume *e*=*x*
thus *?thesis*
by (*simp add*: *ray-def*)
next

```
          assume e≠x
          hence [e;x;a] ∨ [x;e;a] using wxyz-ord
            by (meson ‹[e;a;b]› ‹[x;a;b]› abc-abd-bcdbdc abc-sym)
          thus e∈ray a x
            by (metis Un-iff abc-sym insertCI pro-betw ray-def seg-betw)
        qed
      qed
    qed
    thus is-ray ?L by auto
  qed
qed
show is-ray-on ?R Q
proof (unfold is-ray-on-def, safe)
  show Q ∈ 𝒫
    by (simp add: path-Q)
next
  fix x
  assume [a;x;b]
  then show x ∈ Q
    by (simp add: ‹b ∈ Q› abc-abc-neq betw-b-in-path event-a path-Q)
next
  fix x
  assume [a;b;x]
  then show x ∈ Q
    by (simp add: ‹b ∈ Q› abc-abc-neq betw-c-in-path event-a path-Q)
next
  show b ∈ Q using ‹b ∈ Q› .
next
  show is-ray {y. [a;y;b] ∨ [a;b;y⟧}
  proof −
    have [a;y;b] ∨ [a;b;y] ∨ y=b
      using ‹y∈?R› by blast
    have ?R = ray a y
    proof
      show ray a y ⊆ ?R
      proof
        fix e assume e∈ray a y
        hence [a;e;y] ∨ [a;y;e] ∨ y=e
          using ray-cases by auto
        show e∈?R
        proof −
          { assume e ≠ b
            have (e ≠ y ∧ e ≠ b) ∧ [w;a;y] ∨ [a;e;b] ∨ [a;b;e⟧
                using ‹[a;y;b] ∨ [a;b;y] ∨ y = b› ‹w ∈ {x. [x;a;b]}› abd-bcd-abc by
blast
            hence [a;e;b] ∨ [a;b;e⟧
              using abc-abd-bcdbdc abc-bcd-abd abd-bcd-abc
              by (metis ‹[a;e;y] ∨ [a;y;e⟧› ‹w ∈ ?L› mem-Collect-eq)
          }
```

```
            thus ?thesis
              by blast
          qed
        qed
        show ?R ⊆ ray a y
        proof
          fix e assume e∈?R
          hence aeb-cases: [a;e;b] ∨ [a;b;e] ∨ e=b
            by blast
          hence aey-cases: [a;e;y] ∨ [a;y;e] ∨ e=y
            using abc-abd-bcdbdc abc-bcd-abd abd-bcd-abc
            by (metis ‹[a;y;b] ∨ [a;b;y] ∨ y = b› ‹x ∈ {x. [x;a;b]}› mem-Collect-eq)
          show e∈ray a y
          proof −
            {
              assume e=b
              hence ?thesis
               using ‹[a;y;b] ∨ [a;b;y] ∨ y = b› ‹b ≠ a› pro-betw ray-def seg-betw by
auto
            } moreover {
              assume [a;e;b] ∨ [a;b;e]
              assume y≠e
              hence [a;e;y] ∨ [a;y;e]
                using aey-cases by auto
              hence e∈ray a y
                unfolding ray-def using abc-abc-neq pro-betw seg-betw by auto
            } moreover {
              assume [a;e;b] ∨ [a;b;e]
              assume y=e
              have e∈ray a y
                unfolding ray-def by (simp add: ‹y = e›)
            }
            ultimately show ?thesis
              using aeb-cases by blast
          qed
        qed
      qed
      thus is-ray ?R by auto
    qed
  qed
    show (∀ r∈?R. ∀ l∈?L. [l;a;r])
      using abd-bcd-abc by blast
    show ∀ x∈?R. ∀ y∈?R. ¬ [x;a;y]
      by (smt abc-ac-neq abc-bcd-abd abd-bcd-abc mem-Collect-eq)
    show ∀ x∈?L. ∀ y∈?L. ¬ [x;a;y]
      using abc-abc-neq abc-abd-bcdbdc abc-only-cba by blast
    show Q−{a} ⊆ ?R ∪ ?L
      using ‹Q = {x. [x;a;b]} ∪ {a} ∪ {y. [a;y;b] ∨ [a;b;y]}› by blast
  qed
```

thus *?thesis*
  by (*metis* (*mono-tags*, *lifting*))
qed

The definition *closest-to* in prose: Pick any $r \in R$. The closest event c is such that there is no closer event in L, i.e. all other events of L are further away from r. Thus in L, c is the element closest to R.

definition *closest-to* :: (*'a set*) ⇒ *'a* ⇒ (*'a set*) ⇒ *bool*
  where *closest-to L c R* ≡ *c*∈*L* ∧ (∀ *r*∈*R*. ∀ *l*∈*L*−{*c*}. [*l*;*c*;*r*])

lemma *int-on-path*:
  assumes *l*∈*L* *r*∈*R* *Q*∈𝒫
    and *partition*: *L*⊆*Q* *L*≠{} *R*⊆*Q* *R*≠{} *L*∪*R*=*Q*
  shows *interval l r* ⊆ *Q*
proof
  fix *x* **assume** *x*∈*interval l r*
  thus *x*∈*Q*
    unfolding *interval-def segment-def*
    using *betw-b-in-path partition*(*5*) ‹*Q*∈𝒫› *seg-betw* ‹*l* ∈ *L*› ‹*r* ∈ *R*›
    by *blast*
qed

lemma *ray-of-bounds1*:
  assumes *Q*∈𝒫 [*f*⇝*X*|(*f 0*)..] *X*⊆*Q* *closest-bound c X* *is-bound-f b X f* *b*≠*c*
  assumes *is-bound-f x X f*
  shows *x*=*b* ∨ *x*=*c* ∨ [*c*;*x*;*b*] ∨ [*c*;*b*;*x*]
proof −
  have *x*∈*Q*
   using *bound-on-path assms*(*1*,*3*,*7*) **unfolding** *all-bounds-def is-bound-def is-bound-f-def*
    by *auto*
  {
    assume *x*=*b*
    hence *?thesis* **by** *blast*
  } moreover {
    assume *x*=*c*
    hence *?thesis* **by** *blast*
  } moreover {
    assume *x*≠*b* *x*≠*c*
    hence *?thesis*
      by (*meson abc-abd-bcdbdc assms*(*4*,*5*,*6*,*7*) *closest-bound-def is-bound-def*)
  }
  ultimately show *?thesis* **by** *blast*
qed

lemma *ray-of-bounds2*:
  assumes *Q*∈𝒫 [*f*⇝*X*|(*f 0*)..] *X*⊆*Q* *closest-bound-f c X f* *is-bound-f b X f* *b*≠*c*

assumes *x=b* ∨ *x=c* ∨ [*c*;*x*;*b*] ∨ [*c*;*b*;*x*]
shows *is-bound-f x X f*
proof −
have *x*∈*Q*
using *assms*(*1*,*3*,*4*,*5*,*6*,*7*) *betw-b-in-path betw-c-in-path bound-on-path*
using *closest-bound-f-def is-bound-f-def* **by** *metis*
{
assume *x=b*
hence *?thesis*
by (*simp add*: *assms*(*5*))
} moreover {
assume *x=c*
hence *?thesis* **using** *assms*(*4*)
by (*simp add*: *closest-bound-f-def*)
} moreover {
assume [*c*;*x*;*b*]
hence *?thesis* **unfolding** *is-bound-f-def*
proof (*safe*)
fix *i j*::*nat*
show [*f*⇝*X*|*f 0*..]
by (*simp add*: *assms*(*2*))
assume *i*<*j*
hence [*f i*; *f j*; *b*]
using *assms*(*5*) *is-bound-f-def* **by** *blast*
hence [*f j*; *b*; *c*] ∨ [*f j*; *c*; *b*]
using ‹*i* < *j*› *abc-abd-bcdbdc assms*(*4*,*6*) *closest-bound-f-def is-bound-f-def*
by *auto*
thus [*f i*; *f j*; *x*]
by (*meson* ‹[*c*;*x*;*b*]› ‹[*f i*; *f j*; *b*]› *abc-bcd-acd abc-sym abd-bcd-abc*)
qed
} moreover {
assume [*c*;*b*;*x*]
hence *?thesis* **unfolding** *is-bound-f-def*
proof (*safe*)
fix *i j*::*nat*
show [*f*⇝*X*|*f 0*..]
by (*simp add*: *assms*(*2*))
assume *i*<*j*
hence [*f i*; *f j*; *b*]
using *assms*(*5*) *is-bound-f-def* **by** *blast*
hence [*f j*; *b*; *c*] ∨ [*f j*; *c*; *b*]
using ‹*i* < *j*› *abc-abd-bcdbdc assms*(*4*,*6*) *closest-bound-f-def is-bound-f-def*
by *auto*
thus [*f i*; *f j*; *x*]
proof −
have (*c* = *b*) ∨ [*f 0*; *c*; *b*]
using *assms*(*4*,*5*) *closest-bound-f-def is-bound-def* **by** *auto*
hence [*f j*; *b*; *c*] ⟶ [*x*; *f j*; *f i*]
by (*metis abc-bcd-acd abc-only-cba*(*2*) *assms*(*5*) *is-bound-f-def neq0-conv*)

```
      thus ?thesis
        using ‹[c;b;x]› ‹[f i; f j; b]› ‹[f j; b; c] ∨ [f j; c; b]› abc-bcd-acd abc-sym
        by blast
    qed
  qed
 }
 ultimately show ?thesis using assms(7) by blast
qed


lemma ray-of-bounds3:
 assumes Q∈𝒫 [f↝X|(f 0)..] X⊆Q closest-bound-f c X f is-bound-f b X f b≠c
 shows all-bounds X = insert c (ray c b)
proof
 let ?B = all-bounds X
 let ?C = insert c (ray c b)
 show ?B ⊆ ?C
 proof
  fix x assume x∈?B
  hence is-bound x X
   by (simp add: all-bounds-def)
  hence x=b ∨ x=c ∨ [c;x;b] ∨ [c;b;x]
   using ray-of-bounds1 abc-abd-bcdbdc assms(4,5,6)
   by (meson closest-bound-f-def is-bound-def)
  thus x∈?C
   using pro-betw ray-def seg-betw by auto
 qed
 show ?C ⊆ ?B
 proof
  fix x assume x∈?C
  hence x=b ∨ x=c ∨ [c;x;b] ∨ [c;b;x]
   using pro-betw ray-def seg-betw by auto
  hence is-bound x X
   unfolding is-bound-def using ray-of-bounds2 assms
   by blast
  thus x∈?B
   by (simp add: all-bounds-def)
 qed
qed


lemma int-in-closed-ray:
 assumes path ab a b
 shows interval a b ⊂ insert a (ray a b)
proof
 let ?i = interval a b
 show interval a b ≠ insert a (ray a b)
 proof −
  obtain c where [a;b;c] using prolong-betw2
```

using *assms* **by** *blast*
hence $c \in ray\ a\ b$
using *abc-abc-neq pro-betw ray-def* **by** *auto*
have $c \notin interval\ a\ b$
using ‹$[a;b;c]$› *abc-abc-neq abc-only-cba(2) interval-def seg-betw* **by** *auto*
thus *?thesis*
using ‹$c \in ray\ a\ b$› **by** *blast*
qed
show $interval\ a\ b \subseteq insert\ a\ (ray\ a\ b)$
using *interval-def ray-def* **by** *auto*
qed

end

35 3.8 Connectedness of the unreachable set

context *MinkowskiSpacetime* **begin**

35.1 Theorem 13 (Connectedness of the Unreachable Set)

theorem *unreach-connected*:
assumes *path-Q*: $Q \in \mathcal{P}$
and *event-b*: $b \notin Q\ b \in \mathcal{E}$
and *unreach*: $Q_x \in unreach{-}on\ Q\ from\ b\ \ Q_z \in unreach{-}on\ Q\ from\ b$
and *xyz*: $[Q_x;\ Q_y;\ Q_z]$
shows $Q_y \in unreach{-}on\ Q\ from\ b$
proof −
have *xz*: $Q_x \neq Q_z$ **using** *abc-ac-neq xyz* **by** *blast*

First we obtain the chain from $[\![?Q \in \mathcal{P};\ ?b \in \mathcal{E} - ?Q;\ \{?Qx,\ ?Qz\} \subseteq unreach{-}on\ ?Q\ from\ ?b;\ ?Qx \neq ?Qz]\!] \Longrightarrow \exists X\ f.\ [f \leadsto X|?Qx\ ..\ ?Qz] \wedge (\forall i \in \{1..card\ X - 1\}.\ f\ i \in unreach{-}on\ ?Q\ from\ ?b \wedge (\forall Qy \in \mathcal{E}.\ [f\ (i - 1);Qy;f\ i] \longrightarrow Qy \in unreach{-}on\ ?Q\ from\ ?b))$.

have *in-Q*: $Q_x \in Q \wedge Q_y \in Q \wedge Q_z \in Q$
using *betw-b-in-path path-Q unreach(1,2) xz unreach-on-path xyz* **by** *blast*
hence *event-y*: $Q_y \in \mathcal{E}$
using *in-path-event path-Q* **by** *blast*

legacy: $[\![?Q \in \mathcal{P};\ ?b \notin ?Q;\ ?b \in \mathcal{E};\ ?Qx \in unreach{-}on\ ?Q\ from\ ?b;\ ?Qz \in unreach{-}on\ ?Q\ from\ ?b;\ ?Qx \neq ?Qz]\!] \Longrightarrow \exists X\ f.\ [f \leadsto X] \wedge f\ 0 = ?Qx \wedge f\ (card\ X - 1) = ?Qz \wedge (\forall i \in \{1..card\ X - 1\}.\ f\ i \in unreach{-}on\ ?Q\ from\ ?b \wedge (\forall Qy \in \mathcal{E}.\ [f\ (i - 1);Qy;f\ i] \longrightarrow Qy \in unreach{-}on\ ?Q\ from\ ?b)) \wedge (short{-}ch\ X \longrightarrow ?Qx \in X \wedge ?Qz \in X \wedge (\forall Qy \in \mathcal{E}.\ [?Qx;Qy;?Qz] \longrightarrow Qy \in unreach{-}on\ ?Q\ from\ ?b))$ instead of $[\![?Q \in \mathcal{P};\ ?b \in \mathcal{E} - ?Q;\ \{?Qx,\ ?Qz\} \subseteq unreach{-}on\ ?Q\ from\ ?b;\ ?Qx \neq ?Qz]\!] \Longrightarrow \exists X\ f.\ [f \leadsto X|?Qx\ ..\ ?Qz] \wedge (\forall i \in \{1..card\ X - 1\}.\ f\ i \in unreach{-}on\ ?Q\ from\ ?b \wedge (\forall Qy \in \mathcal{E}.\ [f\ (i - 1);Qy;f\ i] \longrightarrow Qy \in unreach{-}on\ ?Q\ from\ ?b))$

obtain *X f* **where** *X-def*: *ch-by-ord f X f 0* = Q_x *f* (*card X* − *1*) = Q_z
(∀ *i*∈{*1* .. *card X* − *1*}. (*f i*) ∈ *unreach−on Q from b* ∧ (∀ *Qy*∈ℰ. [*f* (*i* − *1*); *Qy*; *f i*] ⟶ *Qy* ∈ *unreach−on Q from b*))
short-ch X ⟶ Q_x ∈ *X* ∧ Q_z ∈ *X* ∧ (∀ Q_y∈ℰ. [Q_x; Q_y; Q_z] ⟶ Q_y ∈ *unreach−on Q from b*)
using *I6-old* [*OF assms*(*1*−*5*) *xz*] **by** *blast*
hence *fin-X*: *finite X*
using *xz not-less* **by** *fastforce*
obtain *N* **where** *N*=*card X N*≥*2*
using *X-def*(*2*,*3*) *xz* **by** *fastforce*

Then we have to manually show the bounds, defined via indices only, are in the obtained chain.

let *?a* = *f 0*
let *?d* = *f* (*card X* − *1*)
{
assume *card X* = *2*
hence *short-ch X ?a* ∈ *X* ∧ *?d* ∈ *X ?a* ≠ *?d*
using *X-def* ‹*card X* = *2*› *short-ch-card-2 xz* **by** *blast+*
}
hence [*f*⇝*X*|Q_x..Q_z]
using *chain-defs* **by** (*metis X-def*(*1*−*3*) *fin-X*)

Further on, we split the proof into two cases, namely the split Schutz absorbs into his non-strict *local-ordering*. Just below is the statement we use ⟦*?P* ∨ *?Q*; *?P* ⟹ *?R*; *?Q* ⟹ *?R*⟧ ⟹ *?R* with.

have *y-cases*: Q_y∈*X* ∨ Q_y∉*X* **by** *blast*
have *y-int*: Q_y∈*interval* Q_x Q_z
using *interval-def seg-betw xyz* **by** *auto*
have *X-in-Q*: *X*⊆*Q*
using *chain-on-path-I6* [**where** *Q*=*Q* **and** *X*=*X*] *X-def event-b path-Q unreach xz* ‹[*f*⇝*X*|Q_x .. Q_z]› **by** *blast*

show *?thesis*
proof (*cases*)

We treat short chains separately. (Legacy: they used to have a separate clause in ⟦*?Q* ∈ 𝒫; *?b* ∈ ℰ − *?Q*; {*?Qx*, *?Qz*} ⊆ *unreach−on ?Q from ?b*; *?Qx* ≠ *?Qz*⟧ ⟹ ∃ *X f*. [*f*⇝*X*|*?Qx* .. *?Qz*] ∧ (∀ *i*∈{*1*..*card X* − *1*}. *f i* ∈ *unreach−on ?Q from ?b* ∧ (∀ *Qy*∈ℰ. [*f* (*i* − *1*);*Qy*;*f i*] ⟶ *Qy* ∈ *unreach−on ?Q from ?b*)), now ⟦*?Q* ∈ 𝒫; *?b* ∉ *?Q*; *?b* ∈ ℰ; *?Qx* ∈ *unreach−on ?Q from ?b*; *?Qz* ∈ *unreach−on ?Q from ?b*; *?Qx* ≠ *?Qz*⟧ ⟹ ∃ *X f*. [*f*⇝*X*] ∧ *f 0* = *?Qx* ∧ *f* (*card X* − *1*) = *?Qz* ∧ (∀ *i*∈{*1*..*card X* − *1*}. *f i* ∈ *unreach−on ?Q from ?b* ∧ (∀ *Qy*∈ℰ. [*f* (*i* − *1*);*Qy*;*f i*] ⟶ *Qy* ∈ *unreach−on ?Q from ?b*)) ∧ (*short-ch X* ⟶ *?Qx* ∈ *X* ∧ *?Qz* ∈ *X* ∧ (∀ *Qy*∈ℰ. [*?Qx*;*Qy*;*?Qz*] ⟶ *Qy* ∈ *unreach−on ?Q from ?b*)))

assume *N*=*2*

thus *?thesis*
using *X-def(1,5) xyz* ‹N = *card X*› *event-y short-ch-card-2* **by** *auto*
next

This is where Schutz obtains the chain from Theorem 11. We instead use the chain we already have with only a part of Theorem 11, namely $[?f \leadsto ?Q|?a..?b..?c] \Longrightarrow$ *interval ?a ?c* $= \bigcup$ {*segment* (*?f i*) (*?f* (i + *1*)) |i. i < *card ?Q* − *1*} ∪ *?Q*. *?S* is defined like in ⟦*?P* ∈ $\mathcal{P}$; *2* ≤ *card ?Q*; *?Q* ⊆ *?P*⟧ ⟹ ∃ *S P1 P2*. *?P* = ⋃ *S* ∪ *P1* ∪ *P2* ∪ *?Q* ∧ *disjoint* (*S* ∪ {*P1*, *P2*}) ∧ *P1* ≠ *P2* ∧ *P1* ∉ *S* ∧ *P2* ∉ *S* ∧ (∀ *x*∈*S*. *is-segment x*) ∧ *is-prolongation P1* ∧ *is-prolongation P2*.

assume *N*≠*2*
hence *N*≥*3* **using** ‹*2* ≤ *N*› **by** *auto*
have *2*≤*card X* **using** ‹*2* ≤ *N*› ‹*N* = *card X*› **by** *blast*
show *?thesis* **using** *y-cases*
proof (*rule disjE*)
assume Q_y∈*X*
then obtain *i* **where** *i-def*: *i*<*card X* Q_y = *f i*
using *X-def(1)* **by** (*metis fin-X obtain-index-fin-chain*)
have *i*≠*0* ∧ *i*≠*card X* − *1*
using *X-def(2,3)*
by (*metis abc-abc-neq i-def(2) xyz*)
hence *i*∈{*1*..*card X* −*1*}
using *i-def(1)* **by** *fastforce*
thus *?thesis* **using** *X-def(4) i-def(2)* **by** *metis*
next
assume Q_y∉*X*

let *?S* = *if card X* = *2 then* {*segment ?a ?d*} *else* {*segment* (*f i*) (*f*(*i*+*1*)) | *i*. *i*<*card X* − *1*}

have Q_y∈⋃ *?S*
proof −
obtain *c* **where** [*f*⇝*X*|Q_x..*c*..Q_z]
using *X-def(1)* ‹*N* = *card X*› ‹*N*≠*2*› ‹[*f*⇝*X*|Q_x..Q_z]› *short-ch-card-2*
by (*metis* ‹*2* ≤ *N*› *le-neq-implies-less long-chain-2-imp-3*)
have *interval* Q_x Q_z = ⋃ *?S* ∪ *X*
using *int-split-to-segs* [*OF* ‹[*f*⇝*X*|Q_x..*c*..Q_z]›] **by** *auto*
thus *?thesis*
using ‹Q_y∉*X*› *y-int* **by** *blast*
qed
then obtain *s* **where** *s*∈*?S* Q_y∈*s* **by** *blast*

have ∃ *i*. *i*∈{*1*..(*card X*)−*1*} ∧ [(*f*(*i*−*1*)); Q_y; *f i*]
proof −
obtain *i'* **where** *i'-def*: *i'* < *N*−*1* *s* = *segment* (*f i'*) (*f* (*i'* + *1*))
using ‹Q_y∈*s*› ‹*s*∈*?S*› ‹*N*=*card X*›
by (*smt* ‹*2* ≤ *N*› ‹*N* ≠ *2*› *le-antisym mem-Collect-eq not-less*)

```
      show ?thesis
      proof (rule exI, rule conjI)
        show (i'+1) ∈ {1..card X − 1}
          using i'-def(1)
          by (simp add: ‹N = card X›)
        show [f((i'+1) − 1); Q_y; f(i'+1)]
          using i'-def(2) ‹Q_y∈s› seg-betw by simp
      qed
    qed
    then obtain i where i-def: i∈{1..(card X)−1} [(f(i−1)); Q_y; f i]
      by blast

    show ?thesis
      by (meson X-def(4) i-def event-y)
  qed
 qed
qed
```

35.2 Theorem 14 (Second Existence Theorem)

```
lemma union-of-bounded-sets-is-bounded:
  assumes ∀x∈A. [a;x;b] ∀x∈B. [c;x;d] A⊆Q B⊆Q Q∈𝒫
    card A > 1 ∨ infinite A card B > 1 ∨ infinite B
  shows ∃l∈Q. ∃u∈Q. ∀x∈A∪B. [l;x;u]
proof −
  let ?P = λ A B. ∃l∈Q. ∃u∈Q. ∀x∈A∪B. [l;x;u]
  let ?I = λ A a b. (card A > 1 ∨ infinite A) ∧ (∀x∈A. [a;x;b])
  let ?R = λA. ∃a b. ?I A a b

  have on-path: ⋀a b A. A ⊆ Q ⟹ ?I A a b ⟹ b ∈ Q ∧ a ∈ Q
  proof −
    fix a b A assume A⊆Q ?I A a b
    show b∈Q∧a∈Q
    proof (cases)
      assume card A ≤ 1 ∧ finite A
      thus ?thesis
        using ‹?I A a b› by auto
    next
      assume ¬ (card A ≤ 1 ∧ finite A)
      hence asmA: card A > 1 ∨ infinite A
        by linarith
      then obtain x y where x∈A y∈A x≠y
      proof
        assume 1 < card A ⋀x y. ⟦x ∈ A; y ∈ A; x ≠ y⟧ ⟹ thesis
        then show ?thesis
          by (metis One-nat-def Suc-le-eq card-le-Suc-iff insert-iff)
      next
        assume infinite A ⋀x y. ⟦x ∈ A; y ∈ A; x ≠ y⟧ ⟹ thesis
        then show ?thesis
```

using *infinite-imp-nonempty* **by** (*metis finite-insert finite-subset singletonI subsetI*)
qed
have $x \in Q$ $y \in Q$
using ‹$A \subseteq Q$› ‹$x \in A$› ‹$y \in A$› **by** *auto*
have [*a*;*x*;*b*] [*a*;*y*;*b*]
by (*simp add*: ‹($1 < card\ A \vee infinite\ A$) ∧ (∀ *x*∈*A*. [*a*;*x*;*b*])› ‹$x \in A$› ‹$y \in A$›)+
hence *betw4 a x y b* ∨ *betw4 a y x b*
using ‹$x \neq y$› *abd-acd-abcdacbd* **by** *blast*
hence $a \in Q \wedge b \in Q$
using ‹$Q \in \mathcal{P}$› ‹$x \in Q$› ‹$x \neq y$› ‹$x \in Q$› ‹$y \in Q$› *betw-a-in-path betw-c-in-path* **by** *blast*
thus *?thesis* **by** *simp*
qed
qed

show *?thesis*
proof (*cases*)
assume $a \neq b \wedge a \neq c \wedge a \neq d \wedge b \neq c \wedge b \neq d \wedge c \neq d$
show *?P A B*
proof (*rule-tac P=?P* **and** *A=Q* **in** *wlog-endpoints-distinct*)

First, some technicalities: the relations P, I, R have the symmetry required.

show ⋀*a b I*. *?I I a b* ⟹ *?I I b a* **using** *abc-sym* **by** *blast*
show ⋀*a b A*. $A \subseteq Q$ ⟹ *?I A a b* ⟹ $b \in Q \wedge a \in Q$ **using** *on-path assms*(*5*) **by** *blast*
show ⋀*I J*. *?R I* ⟹ *?R J* ⟹ *?P I J* ⟹ *?P J I* **by** (*simp add*: *Un-commute*)

Next, the lemma/case assumptions have to be repeated for Isabelle.

show *?I A a b ?I B c d* $A \subseteq Q$ $B \subseteq Q$ $Q \in \mathcal{P}$
using *assms* **by** *simp*+
show $a \neq b \wedge a \neq c \wedge a \neq d \wedge b \neq c \wedge b \neq d \wedge c \neq d$
using ‹$a \neq b \wedge a \neq c \wedge a \neq d \wedge b \neq c \wedge b \neq d \wedge c \neq d$› **by** *simp*

Finally, the important bit: proofs for the necessary cases of betweenness.

show *?P I J*
if *?I I a b ?I J c d* $I \subseteq Q$ $J \subseteq Q$
and [*a*;*b*;*c*;*d*] ∨ [*a*;*c*;*b*;*d*] ∨ [*a*;*c*;*d*;*b*]
for *I J a b c d*
proof −
consider [*a*;*b*;*c*;*d*]|[*a*;*c*;*b*;*d*]|[*a*;*c*;*d*;*b*]
using ‹[*a*;*b*;*c*;*d*] ∨ [*a*;*c*;*b*;*d*] ∨ [*a*;*c*;*d*;*b*]› **by** *fastforce*
thus *?thesis*
proof (*cases*)
assume *asm*: [*a*;*b*;*c*;*d*] **show** *?P I J*
proof −
have ∀ *x*∈ *I*∪*J*. [*a*;*x*;*d*]
by (*metis Un-iff asm betw4-strong betw4-weak that*(*1*) *that*(*2*))

moreover have *a*∈*Q* *d*∈*Q*
using *assms*(*5*) *on-path that*(*1*−*4*) **by** *blast*+
ultimately show *?thesis* **by** *blast*
qed
next
assume [*a*;*c*;*b*;*d*] **show** *?P I J*
proof −
have ∀ *x*∈ *I*∪*J*. [*a*;*x*;*d*]
by (*metis Un-iff* ‹*betw4 a c b d*› *abc-bcd-abd abc-bcd-acd betw4-weak that*(*1*,*2*))
moreover have *a*∈*Q* *d*∈*Q*
using *assms*(*5*) *on-path that*(*1*−*4*) **by** *blast*+
ultimately show *?thesis* **by** *blast*
qed
next
assume [*a*;*c*;*d*;*b*] **show** *?P I J*
proof −
have ∀ *x*∈ *I*∪*J*. [*a*;*x*;*b*]
using ‹*betw4 a c d b*› *abc-bcd-abd abc-bcd-acd abe-ade-bcd-ace*
by (*meson UnE that*(*1*,*2*))
moreover have *a*∈*Q* *b*∈*Q*
using *assms*(*5*) *on-path that*(*1*−*4*) **by** *blast*+
ultimately show *?thesis* **by** *blast*
qed
qed
qed
qed
next
assume ¬(*a*≠*b* ∧ *a*≠*c* ∧ *a*≠*d* ∧ *b*≠*c* ∧ *b*≠*d* ∧ *c*≠*d*)

show *?P A B*
proof (*rule-tac P*=*?P* **and** *A*=*Q* **in** *wlog-endpoints-degenerate*)

This case follows the same pattern as above: the next five *show* statements are effectively bookkeeping.

show ⋀*a b I*. *?I I a b* ⟹ *?I I b a* **using** *abc-sym* **by** *blast*
show ⋀*a b A*. *A* ⊆ *Q* ⟹ *?I A a b* ⟹ *b* ∈ *Q* ∧ *a* ∈ *Q* **using** *on-path* ‹*Q*∈𝒫› **by** *blast*
show ⋀*I J*. *?R I* ⟹ *?R J* ⟹ *?P I J* ⟹ *?P J I* **by** (*simp add*: *Un-commute*)

show *?I A a b* *?I B c d* *A*⊆*Q* *B*⊆*Q* *Q*∈𝒫
using *assms* **by** *simp*+
show ¬ (*a* ≠ *b* ∧ *b* ≠ *c* ∧ *c* ≠ *d* ∧ *a* ≠ *d* ∧ *a* ≠ *c* ∧ *b* ≠ *d*)
using ‹¬ (*a* ≠ *b* ∧ *a* ≠ *c* ∧ *a* ≠ *d* ∧ *b* ≠ *c* ∧ *b* ≠ *d* ∧ *c* ≠ *d*)› **by** *blast*

Again, this is the important bit: proofs for the necessary cases of degeneracy.

show (*a* = *b* ∧ *b* = *c* ∧ *c* = *d* ⟶ *?P I J*) ∧ (*a* = *b* ∧ *b* ≠ *c* ∧ *c* = *d* ⟶ *?P I J*) ∧

($a = b \wedge b = c \wedge c \neq d \longrightarrow$ *?P I J*) $\wedge$ ($a = b \wedge b \neq c \wedge c \neq d \wedge a \neq d$ $\longrightarrow$ *?P I J*) $\wedge$
($a \neq b \wedge b = c \wedge c \neq d \wedge a = d \longrightarrow$ *?P I J*) $\wedge$
([*a*;*b*;*c*] $\wedge$ $a = d \longrightarrow$ *?P I J*) $\wedge$ ([*b*;*a*;*c*] $\wedge$ $a = d \longrightarrow$ *?P I J*)
if *?I I a b ?I J c d* $I \subseteq Q$ $J \subseteq Q$
for *I J a b c d*
proof (*intro conjI impI*)
assume $a = b \wedge b = c \wedge c = d$
show $\exists l \in Q.\ \exists u \in Q.\ \forall x \in I \cup J.$ [*l*;*x*;*u*]
using ‹$a = b \wedge b = c \wedge c = d$› *abc-ac-neq assms*(*5*) *ex-crossing-path that*(*1*,*2*)
by *fastforce*
next
assume $a = b \wedge b \neq c \wedge c = d$
show $\exists l \in Q.\ \exists u \in Q.\ \forall x \in I \cup J.$ [*l*;*x*;*u*]
using ‹$a = b \wedge b \neq c \wedge c = d$› *abc-ac-neq assms*(*5*) *ex-crossing-path that*(*1*,*2*)
by (*metis Un-iff*)
next
assume $a = b \wedge b = c \wedge c \neq d$
hence $\forall x \in I \cup J.$ [*c*;*x*;*d*]
using *abc-abc-neq that*(*1*,*2*) **by** *fastforce*
moreover have $c \in Q$ $d \in Q$
using *on-path* ‹$a = b \wedge b = c \wedge c \neq d$› *that*(*1*,*3*) *abc-abc-neq* **by** *metis+*
ultimately show $\exists l \in Q.\ \exists u \in Q.\ \forall x \in I \cup J.$ [*l*;*x*;*u*] **by** *blast*
next
assume $a = b \wedge b \neq c \wedge c \neq d \wedge a \neq d$
hence $\forall x \in I \cup J.$ [*c*;*x*;*d*]
using *abc-abc-neq that*(*1*,*2*) **by** *fastforce*
moreover have $c \in Q$ $d \in Q$
using *on-path* ‹$a = b \wedge b \neq c \wedge c \neq d \wedge a \neq d$› *that*(*1*,*3*) *abc-abc-neq* **by** *metis+*
ultimately show $\exists l \in Q.\ \exists u \in Q.\ \forall x \in I \cup J.$ [*l*;*x*;*u*] **by** *blast*
next
assume $a \neq b \wedge b = c \wedge c \neq d \wedge a = d$
hence $\forall x \in I \cup J.$ [*c*;*x*;*d*]
using *abc-sym that*(*1*,*2*) **by** *auto*
moreover have $c \in Q$ $d \in Q$
using *on-path* ‹$a \neq b \wedge b = c \wedge c \neq d \wedge a = d$› *that*(*1*,*3*) *abc-abc-neq* **by** *metis+*
ultimately show $\exists l \in Q.\ \exists u \in Q.\ \forall x \in I \cup J.$ [*l*;*x*;*u*] **by** *blast*
next
assume [*a*;*b*;*c*] $\wedge$ $a = d$
hence $\forall x \in I \cup J.$ [*c*;*x*;*d*]
by (*metis UnE abc-acd-abd abc-sym that*(*1*,*2*))
moreover have $c \in Q$ $d \in Q$
using *on-path that*(*2*,*4*) **by** *blast+*
ultimately show $\exists l \in Q.\ \exists u \in Q.\ \forall x \in I \cup J.$ [*l*;*x*;*u*] **by** *blast*
next

```
      assume [b;a;c] ∧ a = d
      hence ∀x∈ I∪J. [c;x;b]
        using abc-sym abd-bcd-abc betw4-strong that(1,2) by (metis Un-iff)
      moreover have c∈Q b∈Q
        using on-path that by blast+
      ultimately show ∃l∈Q. ∃u∈Q. ∀x∈I ∪ J. [l;x;u] by blast
    qed
  qed
 qed
qed
```

```
lemma union-of-bounded-sets-is-bounded2:
 assumes ∀x∈A. [a;x;b] ∀x∈B. [c;x;d] A⊆Q B⊆Q Q∈𝒫
     1<card A ∨ infinite A 1<card B ∨ infinite B
  shows ∃l∈Q−(A∪B). ∃u∈Q−(A∪B). ∀x∈A∪B. [l;x;u]
 using assms union-of-bounded-sets-is-bounded
  [where A=A and a=a and b=b and B=B and c=c and d=d and Q=Q]
 by (metis Diff-iff abc-abc-neq)
```

Schutz proves a mildly stronger version of this theorem than he states. Namely, he gives an additional condition that has to be fulfilled by the bounds y, z in the proof (y,z∉*unreach−on Q from ab*). This condition is trivial given *abc-abc-neq*. His stating it in the proof makes me wonder whether his (strictly speaking) undefined notion of bounded set is somehow weaker than the version using strict betweenness in his theorem statement and used here in Isabelle. This would make sense, given the obvious analogy with sets on the real line.

```
theorem second-existence-thm-1:
 assumes path-Q: Q∈𝒫
     and events: a∉Q b∉Q
     and reachable: path-ex a q1 path-ex b q2 q1∈Q q2∈Q
  shows ∃y∈Q. ∃z∈Q. (∀x∈unreach−on Q from a. [y;x;z]) ∧ (∀x∈unreach−on
Q from b. [y;x;z])
proof −
```

Slightly annoying: Schutz implicitly extends *bounded* to sets, so his statements are neater.

```
 have ∃q∈Q. q∉(unreach−on Q from a) ∃q∈Q. q∉(unreach−on Q from b)
  using cross-in-reachable reachable by blast+
```

This is a helper statement for obtaining bounds in both directions of both unreachable sets. Notice this needs Theorem 13 right now, Schutz claims only Theorem 4. I think this is necessary?

```
 have get-bds: ∃la∈Q. ∃ua∈Q. la∉unreach−on Q from a ∧ ua∉unreach−on Q
from a ∧ (∀x∈unreach−on Q from a. [la;x;ua])
  if asm: a∉Q path-ex a q q∈Q
```

for *a q*
proof −
obtain *Qy* **where** *Qy*∈*unreach*−*on Q from a*
using *asm(2)* ‹*a* ∉ *Q*› *in-path-event path-Q two-in-unreach* **by** *blast*
then obtain *la* **where** *la* ∈ *Q* − *unreach*−*on Q from a*
using *asm(2,3) cross-in-reachable* **by** *blast*
then obtain *ua* **where** *ua* ∈ *Q* − *unreach*−*on Q from a* [*la*;*Qy*;*ua*] *la* ≠ *ua*
using *unreachable-set-bounded* [**where** *Q*=*Q* **and** *b*=*a* **and** *Qx*=*la* **and** *Qy*=*Qy*]
using ‹*Qy* ∈ *unreach*−*on Q from a*› *asm in-path-event path-Q* **by** *blast*
have *la* ∉ *unreach*−*on Q from a* ∧ *ua* ∉ *unreach*−*on Q from a* ∧ (∀ *x*∈*unreach*−*on Q from a*. (*x*≠*la* ∧ *x*≠*ua*) ⟶ [*la*;*x*;*ua*])
proof (*intro conjI*)
show *la* ∉ *unreach*−*on Q from a*
using ‹*la* ∈ *Q* − *unreach*−*on Q from a*› **by** *force*
next
show *ua* ∉ *unreach*−*on Q from a*
using ‹*ua* ∈ *Q* − *unreach*−*on Q from a*› **by** *force*
next show ∀ *x*∈*unreach*−*on Q from a*. *x* ≠ *la* ∧ *x* ≠ *ua* ⟶ [*la*;*x*;*ua*]
proof (*safe*)
fix *x* **assume** *x*∈*unreach*−*on Q from a x*≠*la x*≠*ua*
{
assume *x*=*Qy* **hence** [*la*;*x*;*ua*] **by** (*simp add*: ‹[*la*;*Qy*;*ua*]›)
} moreover {
assume *x*≠*Qy*
have [*Qy*;*x*;*la*] ∨ [*la*;*Qy*;*x*]
proof −
{ assume [*x*;*la*;*Qy*]
hence *la*∈*unreach*−*on Q from a*
using *unreach-connected* ‹*Qy*∈*unreach*−*on Q from a*›‹*x*∈*unreach*−*on Q from a*›‹*x*≠*Qy*› *in-path-event path-Q that* **by** *blast*
hence *False*
using ‹*la* ∈ *Q* − *unreach*−*on Q from a*› **by** *blast* **}**
thus [*Qy*;*x*;*la*] ∨ [*la*;*Qy*;*x*]
using *some-betw* [**where** *Q*=*Q* **and** *a*=*x* **and** *b*=*la* **and** *c*=*Qy*] *path-Q unreach-on-path*
using ‹*Qy* ∈ *unreach*−*on Q from a*› ‹*la* ∈ *Q* − *unreach*−*on Q from a*› ‹*x* ∈ *unreach*−*on Q from a*› ‹*x* ≠ *Qy*› ‹*x* ≠ *la*› **by** *force*
qed
hence [*la*;*x*;*ua*]
proof
assume [*Qy*;*x*;*la*]
thus *?thesis* **using** ‹[*la*;*Qy*;*ua*]› *abc-acd-abd abc-sym* **by** *blast*
next
assume [*la*;*Qy*;*x*]
hence [*la*;*x*;*ua*] ∨ [*la*;*ua*;*x*]
using ‹[*la*;*Qy*;*ua*]› ‹*x* ≠ *ua*› *abc-abd-acdadc* **by** *auto*
have ¬[*la*;*ua*;*x*]
using *unreach-connected that abc-abc-neq abc-acd-bcd in-path-event path-Q*

```
          by (metis DiffD2 ‹Qy ∈ unreach−on Q from a› ‹[la;Qy;ua]› ‹ua ∈ Q −
unreach−on Q from a› ‹x ∈ unreach−on Q from a›)
         show ?thesis
          using ‹[la;x;ua] ∨ [la;ua;x]› ‹¬ [la;ua;x]› by linarith
        qed
       }
       ultimately show [la;x;ua] by blast
     qed
   qed
     thus ?thesis using ‹la ∈ Q − unreach−on Q from a› ‹ua ∈ Q − unreach−on
Q from a› by force
   qed

    have ∃y∈Q. ∃z∈Q. (∀x∈(unreach−on Q from a)∪(unreach−on Q from b).
[y;x;z])
   proof −
     obtain la ua where ∀x∈unreach−on Q from a. [la;x;ua]
       using events(1) get-bds reachable(1,3) by blast
     obtain lb ub where ∀x∈unreach−on Q from b. [lb;x;ub]
       using events(2) get-bds reachable(2,4) by blast
     have unreach−on Q from a ⊆ Q unreach−on Q from b ⊆ Q
       by (simp add: subsetI unreach-on-path)+
     moreover have 1 < card (unreach−on Q from a) ∨ infinite (unreach−on Q
from a)
       using two-in-unreach events(1) in-path-event path-Q reachable(1)
       by (metis One-nat-def card-le-Suc0-iff-eq not-less)
     moreover have 1 < card (unreach−on Q from b) ∨ infinite (unreach−on Q
from b)
       using two-in-unreach events(2) in-path-event path-Q reachable(2)
       by (metis One-nat-def card-le-Suc0-iff-eq not-less)
     ultimately show ?thesis
      using union-of-bounded-sets-is-bounded [where Q=Q and A=unreach−on Q
from a and B=unreach−on Q from b]
       using get-bds assms ‹∀x∈unreach−on Q from a. [la;x;ua]› ‹∀x∈unreach−on
Q from b. [lb;x;ub]›
       by blast
   qed

   then obtain y z where y∈Q z∈Q (∀x∈(unreach−on Q from a)∪(unreach−on
Q from b). [y;x;z])
     by blast
   show ?thesis
   proof (rule bexI)+
     show y∈Q by (simp add: ‹y ∈ Q›)
     show z∈Q by (simp add: ‹z ∈ Q›)
    show (∀x∈unreach−on Q from a. [z;x;y]) ∧ (∀x∈unreach−on Q from b. [z;x;y])
       by (simp add: ‹∀x∈unreach−on Q from a ∪ unreach−on Q from b. [y;x;z]›
abc-sym)
   qed
```

qed

theorem *second-existence-thm-2*:
  assumes *path-Q*: $Q{\in}\mathcal{P}$
    and *events*: $a{\notin}Q$ $b{\notin}Q$ $c{\in}Q$ $d{\in}Q$ $c{\neq}d$
    and *reachable*: $\exists P{\in}\mathcal{P}.\ \exists q{\in}Q.\ path\ P\ a\ q$ $\exists P{\in}\mathcal{P}.\ \exists q{\in}Q.\ path\ P\ b\ q$
  shows $\exists e{\in}Q.\ \exists ae{\in}\mathcal{P}.\ \exists be{\in}\mathcal{P}.\ path\ ae\ a\ e \wedge path\ be\ b\ e \wedge [c;d;e]$
proof –
 obtain *y z* **where** *bounds-yz*: $(\forall x{\in}unreach{-}on\ Q\ from\ a.\ [z;x;y]) \wedge (\forall x{\in}unreach{-}on\ Q\ from\ b.\ [z;x;y])$
      and *yz-inQ*: $y{\in}Q$ $z{\in}Q$
  using *second-existence-thm-1* [**where** $Q{=}Q$ **and** $a{=}a$ **and** $b{=}b$]
  using *path-Q events(1,2) reachable* **by** *blast*
 have $y{\notin}(unreach{-}on\ Q\ from\ a){\cup}(unreach{-}on\ Q\ from\ b)$ $z{\notin}(unreach{-}on\ Q\ from\ a){\cup}(unreach{-}on\ Q\ from\ b)$
  by (*meson Un-iff* ‹$(\forall x{\in}unreach{-}on\ Q\ from\ a.\ [z;x;y]) \wedge (\forall x{\in}unreach{-}on\ Q\ from\ b.\ [z;x;y])$› *abc-abc-neq*)+
 let $?P = \lambda e\ ae\ be.\ (e{\in}Q \wedge path\ ae\ a\ e \wedge path\ be\ b\ e \wedge [c;d;e])$

 have *exist-ay*: $\exists ay.\ path\ ay\ a\ y$
  if $a{\notin}Q$ $\exists P{\in}\mathcal{P}.\ \exists q{\in}Q.\ path\ P\ a\ q$ $y{\notin}(unreach{-}on\ Q\ from\ a)$ $y{\in}Q$
  for *a y*
  using *in-path-event path-Q that unreachable-bounded-path-only*
  by *blast*

 have $[c;d;y] \vee [\![y;c;d] \vee [c;y;d]\!]$
  by (*meson* ‹$y \in Q$› *abc-sym events(3–5) path-Q some-betw*)
 moreover have $[c;d;z] \vee [\![z;c;d] \vee [c;z;d]\!]$
  by (*meson* ‹$z \in Q$› *abc-sym events(3–5) path-Q some-betw*)
 ultimately consider $[c;d;y]$ | $[c;d;z]$ |
       $(([\![y;c;d] \vee [c;y;d]\!]) \wedge ([\![z;c;d] \vee [c;z;d]\!]))$
  by *auto*
 thus *?thesis*
 proof (*cases*)
  assume $[c;d;y]$
  have $y{\notin}(unreach{-}on\ Q\ from\ a)$ $y{\notin}(unreach{-}on\ Q\ from\ b)$
   using ‹$y \notin unreach{-}on\ Q\ from\ a \cup unreach{-}on\ Q\ from\ b$› **by** *blast*+
  then obtain *ay yb* **where** *path ay a y path yb b y*
   using ‹$y{\in}Q$› *exist-ay events(1,2) reachable(1,2)* **by** *blast*
  have *?P y ay yb*
   using ‹$[c;d;y]$› ‹*path ay a y*› ‹*path yb b y*› ‹$y \in Q$› **by** *blast*
  thus *?thesis* **by** *blast*
 next
  assume $[c;d;z]$
  have $z{\notin}(unreach{-}on\ Q\ from\ a)$ $z{\notin}(unreach{-}on\ Q\ from\ b)$
   using ‹$z \notin unreach{-}on\ Q\ from\ a \cup unreach{-}on\ Q\ from\ b$› **by** *blast*+
  then obtain *az bz* **where** *path az a z path bz b z*
   using ‹$z{\in}Q$› *exist-ay events(1,2) reachable(1,2)* **by** *blast*

have *?P z az bz*
using ‹[*c*;*d*;*z*]› ‹*path az a z*› ‹*path bz b z*› ‹*z* ∈ *Q*› **by** *blast*
thus *?thesis* **by** *blast*
next
assume (⟦*y*;*c*;*d*] ∨ [*c*;*y*;*d*⟧) ∧ (⟦*z*;*c*;*d*] ∨ [*c*;*z*;*d*⟧)
have ∃ *e*. [*c*;*d*;*e*]
using *prolong-betw*
using *events*(*3*−*5*) *path-Q* **by** *blast*
then obtain *e* **where** [*c*;*d*;*e*] **by** *auto*
have ¬[*y*;*e*;*z*]
proof (*rule notI*)

Notice Theorem 10 is not needed for this proof, and does not seem to help *sledgehammer*. I think this is because it cannot be easily/automatically reconciled with non-strict notation.

assume [*y*;*e*;*z*]
moreover consider (⟦*y*;*c*;*d*] ∧ ⟦*z*;*c*;*d*]) | (⟦*y*;*c*;*d*] ∧ [*c*;*z*;*d*⟧) |
([*c*;*y*;*d*⟧ ∧ ⟦*z*;*c*;*d*]) | ([*c*;*y*;*d*⟧ ∧ [*c*;*z*;*d*⟧)
using ‹(⟦*y*;*c*;*d*] ∨ [*c*;*y*;*d*⟧) ∧ (⟦*z*;*c*;*d*] ∨ [*c*;*z*;*d*⟧)› **by** *linarith*
ultimately show *False*
by (*smt* ‹[*c*;*d*;*e*]› *abc-ac-neq betw4-strong betw4-weak*)
qed
have *e*∈*Q*
using ‹[*c*;*d*;*e*]› *betw-c-in-path events*(*3*−*5*) *path-Q* **by** *blast*
have *e*∉ *unreach*−*on Q from a e*∉ *unreach*−*on Q from b*
using *bounds-yz* ‹¬ [*y*;*e*;*z*]› *abc-sym* **by** *blast+*
hence *ex-aebe*: ∃ *ae be*. *path ae a e* ∧ *path be b e*
using ‹*e* ∈ *Q*› *events*(*1*,*2*) *in-path-event path-Q reachable*(*1*,*2*) *unreachable-bounded-path-only*
by *metis*
thus *?thesis*
using ‹[*c*;*d*;*e*]› ‹*e* ∈ *Q*› **by** *blast*
qed
qed

The assumption *Q*≠*R* in Theorem 14(iii) is somewhat implicit in Schutz. If *Q*=*R*, *unreach*−*on Q from a* is empty, so the third conjunct of the conclusion is meaningless.

theorem *second-existence-thm-3*:
assumes *paths*: *Q*∈𝒫 *R*∈𝒫 *Q*≠*R*
and *events*: *x*∈*Q x*∈*R a*∈*R a*≠*x b*∉*Q*
and *reachable*: ∃ *P*∈𝒫. ∃ *q*∈*Q*. *path P b q*
shows ∃ *e*∈ℰ. ∃ *ae*∈𝒫. ∃ *be*∈𝒫. *path ae a e* ∧ *path be b e* ∧ (∀ *y*∈*unreach*−*on Q from a*. [*x*;*y*;*e*])
proof −
have *a*∉*Q*
using *events*(*1*−*4*) *paths eq-paths* **by** *blast*
hence *unreach*−*on Q from a* ≠ {}
by (*metis events*(*3*) *ex-in-conv in-path-event paths*(*1*,*2*) *two-in-unreach*)

then obtain *d* **where** *d*∈ *unreach−on Q from a*
by *blast*
have *x*≠*d*
using ‹*d* ∈ *unreach−on Q from a*› *cross-in-reachable events*(*1*) *events*(*2*) *events*(*3*) *paths*(*2*) **by** *auto*
have *d*∈*Q*
using ‹*d* ∈ *unreach−on Q from a*› *unreach-on-path* **by** *blast*

have ∃ *e*∈*Q*. ∃ *ae be*. [*x*;*d*;*e*] ∧ *path ae a e* ∧ *path be b e*
using *second-existence-thm-2* [**where** *c*=*x* **and** *Q*=*Q* **and** *a*=*a* **and** *b*=*b* **and** *d*=*d*]
using ‹*a* ∉ *Q*› ‹*d* ∈ *Q*› ‹*x* ≠ *d*› *events*(*1*−*3*,*5*) *paths*(*1*,*2*) *reachable* **by** *blast*
then obtain *e ae be* **where** *conds*: [*x*;*d*;*e*] ∧ *path ae a e* ∧ *path be b e* **by** *blast*
have ∀ *y*∈(*unreach−on Q from a*). [*x*;*y*;*e*]
proof
fix *y* **assume** *y*∈(*unreach−on Q from a*)
hence *y*∈*Q*
using *unreach-on-path* **by** *blast*
show [*x*;*y*;*e*]
proof (*rule ccontr*)
assume ¬[*x*;*y*;*e*]
then consider *y*=*x* | *y*=*e* | [*y*;*x*;*e*] | [*x*;*e*;*y*]
by (*metis* ‹*d*∈*Q*› ‹*y*∈*Q*› *abc-abc-neq abc-sym betw-c-in-path conds events*(*1*) *paths*(*1*) *some-betw*)
thus *False*
proof (*cases*)
assume *y*=*x* **thus** *False*
using ‹*y* ∈ *unreach−on Q from a*› *events*(*2*,*3*) *paths*(*1*,*2*) *same-empty-unreach unreach-equiv unreach-on-path*
by *blast*
next
assume *y*=*e* **thus** *False*
by (*metis* ‹*y*∈*Q*› *assms*(*1*) *conds empty-iff same-empty-unreach unreach-equiv* ‹*y* ∈ *unreach−on Q from a*›)
next
assume [*y*;*x*;*e*]
hence [*y*;*x*;*d*]
using *abd-bcd-abc conds* **by** *blast*
hence *x*∈(*unreach−on Q from a*)
using *unreach-connected* [**where** *Q*=*Q* **and** Q_x=*y* **and** Q_y=*x* **and** Q_z=*d* **and** *b*=*a*]
using ‹¬[*x*;*y*;*e*]› ‹*a*∉*Q*› ‹*d*∈*unreach−on Q from a*› ‹*y*∈*unreach−on Q from a*› *conds in-path-event paths*(*1*) **by** *blast*
thus *False*
using *empty-iff events*(*2*,*3*) *paths*(*1*,*2*) *same-empty-unreach unreach-equiv unreach-on-path*
by *metis*
next

assume $[x;e;y]$
hence $[d;e;y]$
using *abc-acd-bcd conds* **by** *blast*
hence *e*∈(*unreach−on Q from a*)
using *unreach-connected* [**where** $Q=Q$ **and** $Q_x=y$ **and** $Q_y=e$ **and** $Q_z=d$ **and** $b=a$]
using ‹$a \notin Q$› ‹*d* ∈ *unreach−on Q from a*› ‹*y* ∈ *unreach−on Q from a*›
abc-abc-neq abc-sym events(*3*) *in-path-event paths*(*1*,*2*)
by *blast*
thus *False*
by (*metis conds empty-iff paths*(*1*) *same-empty-unreach unreach-equiv unreach-on-path*)
qed
qed
qed
thus *?thesis*
using *conds in-path-event* **by** *blast*
qed

end

36 Theorem 11 - with path density assumed

locale *MinkowskiDense* = *MinkowskiSpacetime* +
assumes *path-dense*: *path ab a b* ⟹ ∃ *x*. $[a;x;b]$
begin

Path density: if *a* and *b* are connected by a path, then the segment between them is nonempty. Since Schutz insists on the number of segments in his segmentation (Theorem 11), we prove it here, showcasing where his missing assumption of path density fits in (it is used three times in *number-of-segments*, once in each separate meaningful *local-ordering* case).

lemma *segment-nonempty*:
assumes *path ab a b*
obtains *x* **where** *x* ∈ *segment a b*
using *path-dense* **by** (*metis seg-betw assms*)

lemma *number-of-segments*:
assumes *path-P*: $P \in \mathcal{P}$
and *Q-def*: $Q \subseteq P$
and *f-def*: [*f*⇝*Q*|*a*..*b*..*c*]
shows *card* {*segment* (*f i*) (*f* (*i*+*1*)) | *i*. *i*<(*card Q*−*1*)} = *card Q* − *1*
proof −
let *?S* = {*segment* (*f i*) (*f* (*i*+*1*)) | *i*. *i*<(*card Q*−*1*)}
let *?N* = *card Q*
let *?g* = λ *i*. *segment* (*f i*) (*f* (*i*+*1*))

have *?N* ≥ *3* **using** *chain-defs f-def* **by** (*meson finite-long-chain-with-card*)
have *?g* ' {*0..?N−2*} = *?S*
proof (*safe*)
fix *i* **assume** *i*∈{(*0::nat*)..*?N−2*}
show ∃ *ia. segment* (*f i*) (*f* (*i+1*)) = *segment* (*f ia*) (*f* (*ia+1*)) ∧ *ia*<*card Q − 1*
proof
have *i*<*?N−1*
using *assms* ‹*i*∈{(*0::nat*)..*?N−2*}› ‹*?N*≥*3*›
by (*metis One-nat-def Suc-diff-Suc atLeastAtMost-iff le-less-trans lessI less-le-trans less-trans numeral-2-eq-2 numeral-3-eq-3*)
then show *segment* (*f i*) (*f* (*i* + *1*)) = *segment* (*f i*) (*f* (*i* + *1*)) ∧ *i*<*?N−1*
by *blast*
qed
next
fix *x i* **assume** *i* < *card Q* − *1*
let *?s* = *segment* (*f i*) (*f* (*i* + *1*))
show *?s* ∈ *?g* ' {*0..?N* − *2*}
proof −
have *i*∈{*0..?N−2*}
using ‹*i* < *card Q* − *1*› **by** *force*
thus *?thesis* **by** *blast*
qed
qed
moreover have *inj-on ?g* {*0..?N−2*}
proof
fix *i j* **assume** *asm*: *i*∈{*0..?N−2*} *j*∈{*0..?N−2*} *?g i* = *?g j*
show *i*=*j*
proof (*rule ccontr*)
assume *i*≠*j*
hence *f i* ≠ *f j*
using *asm*(*1*,*2*) *f-def assms*(*3*) *indices-neq-imp-events-neq*
[**where** *X*=*Q* **and** *f*=*f* **and** *a*=*a* **and** *b*=*b* **and** *c*=*c* **and** *i*=*i* **and** *j*=*j*]
by *auto*
show *False*
proof (*cases*)
assume *j*=*i+1* **hence** *j*=*Suc i* **by** *linarith*
have *Suc*(*Suc i*) < *?N* **using** *asm*(*1*,*2*) *eval-nat-numeral* ‹*j* = *Suc i*› **by** *auto*
hence [*f i*; *f* (*Suc i*); *f* (*Suc* (*Suc i*))]
using *assms short-ch-card* ‹*?N*≥*3*› *chain-defs local-ordering-def*
by (*metis short-ch-alt*(*1*) *three-in-set3*)
hence [*f i*; *f j*; *f* (*j+1*)] **by** (*simp add*: ‹*j* = *i* + *1*›)
obtain *e* **where** *e*∈*?g j* **using** *segment-nonempty abc-ex-path asm*(*3*)
by (*metis* ‹[*f i*; *f j*; *f* (*j+1*)]› ‹*f i* ≠ *f j*› ‹*j* = *i* + *1*›)
hence *e*∈*?g i*
using *asm*(*3*) **by** *blast*
have [*f i*; *f j*; *e*]

```
        using abd-bcd-abc ‹[f i; f j; f (j+1)]›
        by (meson ‹e ∈ segment (f j) (f (j + 1))› seg-betw)
      thus False
       using ‹e ∈ segment (f i) (f (i + 1))› ‹j = i + 1› abc-only-cba(2) seg-betw
        by auto
    next assume j≠i+1
      have i < card Q ∧ j < card Q ∧ (i+1) < card Q
       using add-mono-thms-linordered-field(3) asm(1,2) assms ‹?N≥3› by auto
      hence f i ∈ Q ∧ f j ∈ Q ∧ f (i+1) ∈ Q
        using f-def unfolding chain-defs local-ordering-def
          by (metis One-nat-def Suc-diff-le Suc-eq-plus1 ‹3 ≤ card Q› add-Suc
card-1-singleton-iff
       card-gt-0-iff card-insert-if diff-Suc-1 diff-Suc-Suc less-natE less-numeral-extra(1)
          nat.discI numeral-3-eq-3)
      hence f i ∈ P ∧ f j ∈ P ∧ f (i+1) ∈ P
        using path-is-union assms
        by (simp add: subset-iff)
      then consider [f i; (f(i+1)); f j] | [f i; f j; (f(i+1))] |
                [(f(i+1)); f i; f j]
        using some-betw path-P f-def indices-neq-imp-events-neq
          ‹f i ≠ f j› ‹i < card Q ∧ j < card Q ∧ i + 1 < card Q› ‹j ≠ i + 1›
        by (metis abc-sym less-add-one less-irrefl-nat)
      thus False
      proof (cases)
        assume [(f(i+1)); f i; f j]
        then obtain e where e∈?g i using segment-nonempty
          by (metis ‹f i ∈ P ∧ f j ∈ P ∧ f (i + 1) ∈ P› abc-abc-neq path-P)
        hence [e; f j; (f(j+1))]
          using ‹[(f(i+1)); f i; f j]›
          by (smt abc-acd-abd abc-acd-bcd abc-only-cba abc-sym asm(3) seg-betw)
        moreover have e∈?g j
          using ‹e ∈ ?g i› asm(3) by blast
        ultimately show False
          by (simp add: abc-only-cba(1) seg-betw)
      next
        assume [f i; f j; (f(i+1))]
        thus False
            using abc-abc-neq [where b=f j and a=f i and c=f(i+1)] asm(3)
seg-betw [where x=f j]
          using ends-notin-segment by blast
      next
        assume [f i; (f(i+1)); f j]
        then obtain e where e∈?g i using segment-nonempty
          by (metis ‹f i ∈ P ∧ f j ∈ P ∧ f (i + 1) ∈ P› abc-abc-neq path-P)
        hence [e; f j; (f(j+1))]
        proof −
          have f (i+1) ≠ f j
            using ‹[f i; (f(i+1)); f j]› abc-abc-neq by presburger
          then show ?thesis
```

using ‹*e* ∈ *segment* (*f i*) (*f* (*i*+*1*))› ‹[*f i*; (*f*(*i*+*1*)); *f j*]› *asm*(*3*) *seg-betw*
by (*metis* (*no-types*) *abc-abc-neq abc-acd-abd abc-acd-bcd abc-sym*)
qed
moreover have *e*∈*?g j*
using ‹*e* ∈ *?g i*› *asm*(*3*) **by** *blast*
ultimately show *False*
by (*simp add*: *abc-only-cba*(*1*) *seg-betw*)
qed
qed
qed
qed
ultimately have *bij-betw ?g* {*0*..*?N*−*2*} *?S*
using *inj-on-imp-bij-betw* **by** *fastforce*
thus *?thesis*
using *assms*(*2*) *bij-betw-same-card numeral-2-eq-2 numeral-3-eq-3* ‹*?N*≥*3*›
by (*metis* (*no-types*, *lifting*) *One-nat-def Suc-diff-Suc card-atLeastAtMost le-less-trans less-Suc-eq-le minus-nat.diff-0 not-less not-numeral-le-zero*)
qed

theorem *segmentation-card*:
assumes *path-P*: *P*∈𝒫
and *Q-def*: *Q*⊆*P*
and *f-def*: [*f*⇝*Q*|*a*..*b*]
fixes *P1* **defines** *P1-def*: *P1* ≡ *prolongation b a*
fixes *P2* **defines** *P2-def*: *P2* ≡ *prolongation a b*
fixes *S* **defines** *S-def*: *S* ≡ {*segment* (*f i*) (*f* (*i*+*1*)) | *i*. *i*<*card Q*−*1*}
shows *P* = ((⋃*S*) ∪ *P1* ∪ *P2* ∪ *Q*)

card S = (*card Q*−*1*) ∧ (∀ *x*∈*S*. *is-segment x*)

disjoint (*S*∪{*P1*,*P2*}) *P1*≠*P2 P1*∉*S P2*∉*S*

proof −
let *?N* = *card Q*
have *2* ≤ *card Q*
using *f-def fin-chain-card-geq-2* **by** *blast*
have *seg-facts*: *P* = (⋃*S* ∪ *P1* ∪ *P2* ∪ *Q*) (∀ *x*∈*S*. *is-segment x*)
disjoint (*S*∪{*P1*,*P2*}) *P1*≠*P2 P1*∉*S P2*∉*S*
using *show-segmentation* [*OF path-P Q-def f-def*]
using *P1-def P2-def S-def* **by** *fastforce*+
show *P* = ⋃*S* ∪ *P1* ∪ *P2* ∪ *Q* **by** (*simp add*: *seg-facts*(*1*))
show *disjoint* (*S*∪{*P1*,*P2*}) *P1*≠*P2 P1*∉*S P2*∉*S*
using *seg-facts*(*3*−*6*) **by** *blast*+
have *card S* = (*?N*−*1*)
proof (*cases*)
assume *?N*=*2*
hence *card S* = *1*
by (*simp add*: *S-def*)

```
    thus ?thesis
      by (simp add: ‹?N = 2›)
  next
    assume ?N≠2
    hence ?N≥3
      using ‹2 ≤ card Q› by linarith
    then obtain c where [f↝Q|a..c..b]
      using assms chain-defs short-ch-card-2 ‹2 ≤ card Q› ‹card Q ≠ 2›
      by (metis three-in-set3)
    show ?thesis
      using number-of-segments [OF assms(1,2) ‹[f↝Q|a..c..b]›]
      using S-def ‹card Q ≠ 2› by presburger
  qed
  thus card S = card Q − 1 ∧ Ball S is-segment
    using seg-facts(2) by blast
qed


end



end
```

References

[1] J. W. Schutz. *Independent Axioms for Minkowski Space-Time.* CRC Press, Oct. 1997.