Geometric Axioms for Minkowski Spacetime

Richard Schmoetten, Jake Palmer, Jacques Fleuriot

July 17, 2021

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 *Main 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 *ordering2*).

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 *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*: $(\forall n.\ (\mathit{finite}\ \{a,b\} \longrightarrow n < \mathit{card}\ \{a,b\}) \longrightarrow ?f\ n \in \{a,b\})$ **by** *simp*
  have *ordering2*: $(\forall x\in\{a,b\}.\ \exists n.\ (\mathit{finite}\ \{a,b\} \longrightarrow n < \mathit{card}\ \{a,b\}) \wedge ?f\ n = x)$
    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*: $(\forall n\ n'\ n''.\ (\mathit{finite}\ \{a,b\} \longrightarrow n'' < \mathit{card}\ \{a,b\}) \wedge n < n' \wedge n' < n''$
    $\longrightarrow \mathit{ord}\ (?f\ n)\ (?f\ n')\ (?f\ n''))$ **using** *a-neq-b* **by** *auto*
  have *ordering ?f ord* $\{a,\ b\}$ **using** *ordering-def ordering1 ordering2 ordering3* **by** *blast*
  thus *?thesis* **by** *auto*
qed

lemma *card-le2-ordering*:
  assumes *finiteX*: *finite X*
    and *card-le2*: $\mathit{card}\ X \leq 2$
  shows $\exists f.$ *ordering f ord X*
proof −
  have *card012*: $\mathit{card}\ X = 0 \vee \mathit{card}\ X = 1 \vee \mathit{card}\ X = 2$ **using** *card-le2* **by** *auto*
  have *card0*: $\mathit{card}\ X = 0 \longrightarrow$ *?thesis* **using** *finiteX* **by** *simp*
  have *card1*: $\mathit{card}\ X = 1 \longrightarrow$ *?thesis* **using** *card-eq-SucD* **by** *fastforce*
  have *card2*: $\mathit{card}\ X = 2 \longrightarrow$ *?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* = λ*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*: ∀ *n*. (*finite ?X* ⟶ *n* < *card ?X*) ⟶ *?f n* ∈ *?X* **by** *simp*
  have *ordering2*: ∀ *x*∈*?X*. ∃ *n*. (*finite ?X* ⟶ *n* < *card ?X*) ∧ *?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*: (∀ *n n′ n″*. (*finite ?X* ⟶ *n″* < *card ?X*) ∧ *n* < *n′* ∧ *n′* < *n″*
        ⟶ *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 ordering2 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*: ∀ *a b c d*. *ord a b c* ∧ *ord b c d* ⟶ *ord a b d*
    and *abc-bcd-acd*: ∀ *a b c d*. *ord a b c* ∧ *ord b c d* ⟶ *ord a c d*
    and *ord-distinct*: ∀ *a b c*. (*ord a b c* ⟶ *a* ≠ *b* ∧ *a* ≠ *c* ∧ *b* ≠ *c*)
  shows ∃ *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*: ∀ *a b c*. (*ord a b c* ⟶ *a* ≠ *b* ∧ *a* ≠ *c* ∧ *b* ≠ *c*)
  shows ∃ *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*: ∀ *a b c d*. *ord a b c* ∧ *ord b c d* ⟶ *ord a b d*
    and *abc-bcd-acd*: ∀ *a b c d*. *ord a b c* ∧ *ord b c d* ⟶ *ord a c d*
    and *abc-neq*: *a* ≠ *b* ∧ *a* ≠ *c* ∧ *a* ≠ *d* ∧ *b* ≠ *c* ∧ *b* ≠ *d* ∧ *c* ≠ *d*
  shows ∃ *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*: $[\![X \neq \{\}; \textit{finite } X; \textit{ordering } f \textit{ ord } X]\!] \Longrightarrow f \text{ ` } \{0..\textit{card } X - 1\} = X$
by (*force simp add*: *ordering-def image-def*)

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

lemma *finite-ordering-inv-img*: $[\![X \neq \{\}; \textit{finite } X; \textit{ordering } f \textit{ ord } X]\!] \Longrightarrow f -\text{` } X = \{0..\textit{card } X - 1\}$
apply (*auto simp add*: *ordering-def*)
oops

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

lemma *inf-ordering-img-inv-img*: $[\![\textit{infinite } X; \textit{ordering } f \textit{ ord } X]\!] \Longrightarrow f \text{ ` } f -\text{` } X = X$
using *inf-ordering-img* **by** *auto*

lemma *finite-ordering-inj-on*: $[\![\textit{finite } X; \textit{ordering } f \textit{ ord } X]\!] \Longrightarrow \textit{inj-on } f \{0..\textit{card } X - 1\}$
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* $\{0..\textit{card } X - 1\}$ *X*
proof −
  have *f-image*: $f \text{ ` } \{0..\textit{card } X - 1\} = X$ **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*: $\forall a\ b\ c.\ (\textit{ord } a\ b\ c \longrightarrow a \neq b \wedge a \neq c \wedge b \neq c)$
    and *f-eq*: $f\ m = f\ n$
  shows $m = n$

proof (*rule ccontr*)
  assume *m-not-n*: $m \neq n$
  have *betw-3n*: $\forall n\ n'\ n''.\ n < n' \wedge n' < n'' \longrightarrow \textit{ord } (f\ n)\ (f\ n')\ (f\ n'')$
    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* ≤ *2*
have *card0*: *card X* = *0* ⟶ *False* **using** *m-less-card* **by** *simp*
have *card1*: *card X* = *1* ⟶ *False* **using** *m-less-card n-less-card m-not-n* **by** *simp*
have *card2*: *card X* = *2* ⟶ *False*
proof (*rule impI*)
assume *card-is-2*: *card X* = *2*
then have *mn01*: *m* = *0* ∧ *n* = *1* ∨ *n* = *0* ∧ *m* = *1* **using** *m-less-card n-less-card m-not-n* **by** *auto*
then have *f m* ≠ *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 ¬ *card X* ≤ *2*
then have *card-ge3*: *card X* ≥ *3* **by** *simp*
thus *False*
proof *cases*
assume *m-less-n*: *m* < *n*
then obtain *k* **where** *k-pos*: *k* < *m* ∨ (*m* < *k* ∧ *k* < *n*) ∨ (*n* < *k* ∧ *k* < *card X*)
using *is-free-nat m-less-n n-less-card card-ge3* **by** *blast*
have *k1*: *k* < *m* ⟶*ord* (*f k*) (*f m*) (*f n*) **using** *m-less-n n-less-card betw-3n* **by** *simp*
have *k2*: *m* < *k* ∧ *k* < *n* ⟶ *ord* (*f m*) (*f k*) (*f n*) **using** *m-less-n n-less-card betw-3n* **by** *simp*
have *k3*: *n* < *k* ∧ *k* < *card X* ⟶ *ord* (*f m*) (*f n*) (*f k*) **using** *m-less-n betw-3n* **by** *simp*
have *f m* ≠ *f n* **using** *k1 k2 k3 k-pos ord-distinct* **by** *auto*
thus *False* **using** *f-eq* **by** *simp*
next
assume ¬ *m* < *n*
then have *n-less-m*: *n* < *m* **using** *m-not-n* **by** *simp*
then obtain *k* **where** *k-pos*: *k* < *n* ∨ (*n* < *k* ∧ *k* < *m*) ∨ (*m* < *k* ∧ *k* < *card X*)
using *is-free-nat n-less-m m-less-card card-ge3* **by** *blast*
have *k1*: *k* < *n* ⟶*ord* (*f k*) (*f n*) (*f m*) **using** *n-less-m m-less-card betw-3n* **by** *simp*
have *k2*: *n* < *k* ∧ *k* < *m* ⟶ *ord* (*f n*) (*f k*) (*f m*) **using** *n-less-m m-less-card betw-3n* **by** *simp*
have *k3*: *m* < *k* ∧ *k* < *card X* ⟶ *ord* (*f n*) (*f m*) (*f k*) **using** *n-less-m betw-3n* **by** *simp*
have *f n* ≠ *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 *ordering2* :: $(nat \Rightarrow 'a) \Rightarrow ('a \Rightarrow 'a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a\ set \Rightarrow bool$
where
$ordering2\ f\ ord\ X \equiv (\forall\, n.\ (finite\ X \longrightarrow n < card\ X) \longrightarrow f\ n \in X)$
$\wedge\ (\forall\, x{\in}X.\ (\exists\, n.\ (finite\ X \longrightarrow n < card\ X) \wedge f\ n = x))$
$\wedge\ (\forall\, n\ n'\ n''.\ (finite\ X \longrightarrow n'' < card\ X) \wedge Suc\ n = n' \wedge Suc\ n'$
$= n''$
$\longrightarrow ord\ (f\ n)\ (f\ n')\ (f\ n''))$

Analogue to *ordering-ord-ijk*, which is quicker to use in sledgehammer than the definition.

lemma *ordering2-ord-ijk*:
assumes $ordering2\ f\ ord\ X$
and $Suc\ i = j \wedge Suc\ j = k \wedge (finite\ X \longrightarrow k < card\ X)$
shows $ord\ (f\ i)\ (f\ j)\ (f\ k)$
by (*metis ordering2-def assms*)

end

theory *Minkowski*

imports *Main 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*)

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-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-of $a\ b \equiv THE\ ab.\ path\ ab\ a\ b$

lemma *path-of-ex*: *path* (*path-of* $a\ b$) $a\ b \longleftrightarrow$ *path-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*

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* :: $'a\ set \Rightarrow\ 'a \Rightarrow\ 'a\ set$ ($\emptyset$ - - [*100*, *100*] *100*) **where**
$\quad$ *unreachable-subset* $Q\ b \equiv \{x{\in}Q.\ Q \in \mathcal{P} \wedge b \in \mathcal{E} \wedge b \notin Q \wedge \neg(\textit{path-ex}\ b\ x)\}$

5 Primitives: Kinematic Triangle

definition *kinematic-triangle* :: $'a \Rightarrow\ 'a \Rightarrow\ 'a \Rightarrow \textit{bool}$ ($\triangle$ - - - [*100*, *100*, *100*] *100*) **where**

$$
\begin{aligned}
&\textit{kinematic-triangle}\ a\ b\ c \equiv \\
&\quad a \in \mathcal{E} \wedge b \in \mathcal{E} \wedge c \in \mathcal{E} \wedge a \neq b \wedge a \neq c \wedge b \neq c \\
&\quad \wedge (\exists\, Q{\in}\mathcal{P}.\ \exists\, R{\in}\mathcal{P}.\ Q \neq R \wedge (\exists\, S{\in}\mathcal{P}.\ Q \neq S \wedge R \neq S \\
&\qquad\qquad \wedge a \in Q \wedge b \in Q \\
&\qquad\qquad \wedge a \in R \wedge c \in R \\
&\qquad\qquad \wedge b \in S \wedge c \in S))
\end{aligned}
$$

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

lemma *tri-full*:

$$
\begin{aligned}
\triangle\ a\ b\ c = (&a \in \mathcal{E} \wedge b \in \mathcal{E} \wedge c \in \mathcal{E} \wedge a \neq b \wedge a \neq c \wedge b \neq c \\
&\wedge (\exists\, Q{\in}\mathcal{P}.\ \exists\, R{\in}\mathcal{P}.\ Q \neq R \wedge (\exists\, S{\in}\mathcal{P}.\ Q \neq S \wedge R \neq S \\
&\qquad \wedge a \in Q \wedge b \in Q \wedge c \notin Q \\
&\qquad \wedge a \in R \wedge c \in R \wedge b \notin R \\
&\qquad \wedge b \in S \wedge c \in S \wedge a \notin S)))
\end{aligned}
$$

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**
$\quad$ *SPRAY* $x \equiv \{R{\in}\mathcal{P}.\ x \in R\}$

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

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

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

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* :: $'a\ set \Rightarrow\ 'a\ set \Rightarrow\ 'a\ set \Rightarrow\ 'a \Rightarrow bool$ **where**
  $dep3\text{-}event\ Q\ R\ S\ x \equiv Q \neq R \wedge Q \neq S \wedge R \neq S \wedge Q \in SPRAY\ x \wedge R \in SPRAY\ x \wedge S \in SPRAY\ x$
  $\wedge (\exists\, T{\in}\mathcal{P}.\ T \notin SPRAY\ x \wedge (\exists\, y{\in}Q.\ y \in T) \wedge (\exists\, y{\in}R.\ y \in T) \wedge (\exists\, y{\in}S.\ y \in T))$

definition *dep3-spray* :: $'a\ set \Rightarrow\ 'a\ set \Rightarrow\ 'a\ set \Rightarrow (\,'a\ set)\ set \Rightarrow bool$ **where**
  $dep3\text{-}spray\ Q\ R\ S\ SPR \equiv \exists\, x.\ SPRAY\ x = SPR \wedge dep3\text{-}event\ Q\ R\ S\ x$

definition *dep3* :: $'a\ set \Rightarrow\ 'a\ set \Rightarrow\ 'a\ set \Rightarrow bool$ **where**
  $dep3\ Q\ R\ S \equiv \exists\, x.\ dep3\text{-}event\ Q\ R\ S\ x$

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

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

lemma *dep3-path*:
  assumes *dep3-QRSx*: $dep3\text{-}event\ Q\ R\ S\ x$
  shows $Q \in \mathcal{P}\ \ R \in \mathcal{P}\ \ S \in \mathcal{P}$
proof −
  have $\{Q,R,S\} \subseteq SPRAY\ x$ **using** *dep3-event-def* **using** *dep3-QRSx* **by** *simp*
  thus $Q \in \mathcal{P}\ \ R \in \mathcal{P}\ \ S \in \mathcal{P}$ **using** *SPRAY-path* **by** *auto*
qed

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

lemma *dep3-event-permute* [*no-atp*]:
  assumes $dep3\text{-}event\ Q\ R\ S\ x$

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-def assms* **by** *auto*

"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 $n \geq 3$)

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

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* :: $'a\ set \Rightarrow ('a\ set)\ set \Rightarrow 'a \Rightarrow bool$ **where**
dep-two: *dep3-event T A B x* $\Longrightarrow$ *dep-path T* $\{A, B\}$ *x*
| *dep-n*: $[\![S \subseteq SPRAY\ x;\ card\ S \geq 3;\ dep\text{-}path\ T\ \{S1,\ S2\}\ x;$
$S' \subseteq S;\ S'' \subseteq S;\ card\ S' = card\ S - 1;\ card\ S'' = card\ S - 1;$
$dep\text{-}path\ S1\ S'\ x;\ dep\text{-}path\ S2\ S''\ x]\!] \Longrightarrow dep\text{-}path\ T\ S\ x$

"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* :: $('a\ set)\ set \Rightarrow bool$ **where**
dep-set S $\equiv \exists x.\ \exists S' \subseteq S.\ \exists P \in (S - S').$ *dep-path P S' x*

lemma *dependent-superset*:
assumes *dep-set A* **and** $A \subseteq B$
shows *dep-set B*
using *assms(1) assms(2) dep-set-def*
by (*meson Diff-mono dual-order.trans in-mono order-refl*)

lemma *path-in-dep-set*:
assumes *dep3-event P Q R x*
shows *dep-set* $\{P,Q,R\}$
using *dep-two assms dep3-event-def dep-set-def*
by (*metis DiffI insertE insertI1 singletonD subset-insertI*)

lemma *path-in-dep-set2*:
assumes *dep3-event P Q R x*
shows *dep-path P* $\{P,Q,R\}$ *x*
proof
let *?S1* = *Q*
let *?S2* = *R*
let *?S'* = $\{P,R\}$
let *?S''* = $\{P,Q\}$
show $\{P,\ Q,\ R\} \subseteq SPRAY\ x$ **using** *assms dep3-event-def* **by** *blast*
show $3 \leq card\ \{P,\ Q,\ R\}$ **using** *assms dep3-event-def* **by** *auto*
show *dep-path P* $\{?S1,\ ?S2\}$ *x* **using** *assms dep3-event-def* **by** (*simp add*: *dep-two*)
show $?S' \subseteq \{P,\ Q,\ R\}$ **by** *simp*

show $?S'' \subseteq \{P,\ Q,\ R\}$ **by** *simp*
show $card\ ?S' = card\ \{P,\ Q,\ R\} - 1$ **using** *assms dep3-event-def* **by** *auto*
show $card\ ?S'' = card\ \{P,\ Q,\ R\} - 1$ **using** *assms dep3-event-def* **by** *auto*
show *dep-path ?S1 ?S′ x* **by** (*simp add*: *assms dep3-event-permute*(*2*) *dep-two*)
show *dep-path ?S2 ?S″ x* **using** *assms dep3-event-permute*(*2*,*4*) *dep-two* **by** *blast*
qed

definition *indep-set* :: (*′a set*) *set* ⇒ *bool* **where**
$indep\text{-}set\ S \equiv \neg(\exists\ T \subseteq S.\ dep\text{-}set\ T)$

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 *three-SPRAY* :: *′a* ⇒ *bool* **where**
$three\text{-}SPRAY\ x \equiv \exists\ S1{\in}\mathcal{P}.\ \exists\ S2{\in}\mathcal{P}.\ \exists\ S3{\in}\mathcal{P}.\ \exists\ S4{\in}\mathcal{P}.$
$S1 \neq S2 \wedge S1 \neq S3 \wedge S1 \neq S4 \wedge S2 \neq S3 \wedge S2 \neq S4 \wedge S3 \neq S4$
$\wedge\ S1 \in SPRAY\ x \wedge S2 \in SPRAY\ x \wedge S3 \in SPRAY\ x \wedge S4 \in SPRAY\ x$
$\wedge\ (indep\text{-}set\ \{S1,\ S2,\ S3,\ S4\})$
$\wedge\ (\forall\ S{\in}SPRAY\ x.\ dep\text{-}path\ S\ \{S1,S2,S3,S4\}\ x)$

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**
$is\text{-}three\text{-}SPRAY\ SPR \equiv \exists\ x.\ SPR = SPRAY\ x \wedge three\text{-}SPRAY\ x$

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-def* **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* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$ ([[- - -]])

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 ¬ [[*b a c*]] ¬ [[*a c b*]] ¬ [[*b c a*]] ¬ [[*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*
  (∅ *- from - via - at -* [*100*, *100*, *100*, *100*] *100*) **where**
  unreachable-subset-via Q Qa R x ≡ {*Qy*. [[*x Qy Qa*]] ∧ (∃ *Rw*∈*R*. *Qa* ∈ ∅ *Q Rw* ∧ *Qy* ∈ ∅ *Q Rw*)}

11 Betweenness: Chains

11.1 Totally ordered chains with indexing

definition *short-ch* :: *′a set* ⇒ *bool* **where**
  short-ch X ≡
    — EITHER two distinct events connected by a path
    ∃ *x*∈*X*. ∃ *y*∈*X*. *path-ex x y* ∧ ¬(∃ *z*∈*X*. *z*≠*x* ∧ *z*≠*y*)

Infinite sets have card 0, because card gives a natural number always.

definition *long-ch-by-ord* :: (*nat* ⇒ *′a*) ⇒ *′a set* ⇒ *bool* **where**
  long-ch-by-ord f X ≡
    — OR at least three events such that any three events are ordered
    ∃ *x*∈*X*. ∃ *y*∈*X*. ∃ *z*∈*X*. *x*≠*y* ∧ *y*≠*z* ∧ *x*≠*z* ∧ *ordering f betw X*

Does this restrict chains to lie on paths? Proven in Ch3's Interlude!

definition *ch-by-ord* :: (*nat* ⇒ *′a*) ⇒ *′a set* ⇒ *bool* **where**
  ch-by-ord f X ≡ *short-ch X* ∨ *long-ch-by-ord f X*

definition *ch* :: *′a set* ⇒ *bool* **where**
  ch X ≡ ∃ *f*. *ch-by-ord f X*

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 *semifin-chain*:: $(nat \Rightarrow 'a) \Rightarrow 'a \Rightarrow 'a\ set \Rightarrow bool$ ([-[- ..]-]) **where**
  semifin-chain f x Q ≡
    infinite Q ∧ *long-ch-by-ord f Q*
    ∧ *f 0* = *x*

definition *fin-long-chain*:: $(nat \Rightarrow 'a) \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a\ set \Rightarrow bool$
  ([-[- .. - .. -]-]) **where**
  fin-long-chain f x y z Q ≡
    x≠*y* ∧ *x*≠*z* ∧ *y*≠*z*
    ∧ *finite Q* ∧ *long-ch-by-ord f Q*
    ∧ *f 0* = *x* ∧ *y*∈*Q* ∧ *f* (*card Q* − *1*) = *z*

lemma *index-middle-element*:
  assumes [*f*[*a*..*b*..*c*]*X*]
  shows ∃ *n*. *0*<*n* ∧ *n*<(*card X* − *1*) ∧ *f n* = *b*
proof −
  obtain *n* **where** *n-def*: *n* < *card X f n* = *b*
   by (*metis TernaryOrdering.ordering-def assms fin-long-chain-def long-ch-by-ord-def*)
  have *0*<*n* ∧ *n*<(*card X* − *1*) ∧ *f n* = *b*
    using *assms fin-long-chain-def n-def*
    by (*metis Suc-pred′ gr-implies-not0 less-SucE not-gr-zero*)
  thus *?thesis* **by** *blast*
qed

lemma *fin-ch-betw*:
  assumes [*f*[*a*..*b*..*c*]*X*]
  shows [[*a b c*]]
proof −
  obtain *nb* **where** *n-def*: *nb*≠*0 nb*<*card X* − *1 f nb* = *b*
    using *assms index-middle-element* **by** *blast*
  have [[(*f 0*) (*f nb*) (*f* (*card X* − *1*))]]
   using *fin-long-chain-def long-ch-by-ord-def assms n-def ordering-ord-ijk zero-less-iff-neq-zero*
    by *fastforce*
  thus *?thesis* **using** *assms fin-long-chain-def n-def*(*3*) **by** *auto*
qed

lemma *chain-sym-obtain*:
  assumes [*f*[*a*..*b*..*c*]*X*]
  obtains *g* **where** [*g*[*c*..*b*..*a*]*X*] **and** *g*=(λ*n*. *f* (*card X* − *1* − *n*))
using *ordering-sym assms*(*1*) **unfolding** *fin-long-chain-def long-ch-by-ord-def*
by (*metis* (*mono-tags*, *lifting*) *abc-sym diff-self-eq-0 diff-zero*)

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

definition *fin-long-chain-2*:: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ *set* $\Rightarrow$ *bool* **where**
  fin-long-chain-2 x y z Q $\equiv \exists f.\ [f[x..y..z]Q]$

definition *fin-chain*:: $(nat \Rightarrow 'a) \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ *set* $\Rightarrow$ *bool* ([-[- .. -]-]) **where**
  fin-chain f x y Q $\equiv$
    (*short-ch Q* $\wedge$ $x \in Q \wedge y \in Q \wedge x \neq y$)
    $\vee$ ($\exists z \in Q.\ [f[x..z..y]Q]$)

lemma *points-in-chain*:
  assumes $[f[x..y..z]Q]$
  shows $x \in Q \wedge y \in Q \wedge z \in Q$
proof $-$
  have $x \in Q$
   using *ordering-def assms card-gt-0-iff emptyE fin-long-chain-def long-ch-by-ord-def*
    by *metis*
  moreover have $y \in Q$
    using *assms fin-long-chain-def*
    by *auto*
  moreover have $z \in Q$
   using *ordering-def assms card-gt-0-iff emptyE fin-long-chain-def long-ch-by-ord-def*
    by (*metis* (*no-types*, *hide-lams*) *Suc-diff-1 lessI*)
  ultimately show *?thesis*
    by *blast*
qed

lemma *ch-long-if-card-ge3*:
  assumes *ch X*
      and *card X* $\geq$ *3*
    shows $\exists f.$ *long-ch-by-ord f X*
proof (*rule ccontr*)
  assume $\nexists f.$ *long-ch-by-ord f X*
  hence *short-ch X*
    using *assms*(*1*) *ch-by-ord-def ch-def*
    by *auto*
  obtain *x y z* **where** $x \in X \wedge y \in X \wedge z \in X$ **and** $x \neq y \wedge y \neq z \wedge x \neq z$
    using *assms*(*2*)
    by (*auto simp add*: *card-le-Suc-iff numeral-3-eq-3*)
  thus *False*
    using ‹*short-ch X*› *short-ch-def*
    by *metis*
qed

11.2 Locally ordered chains with indexing

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

definition *long-ch-by-ord2* :: $(nat \Rightarrow 'a) \Rightarrow 'a$ *set* $\Rightarrow$ *bool* **where**
  long-ch-by-ord2 f X $\equiv$
    $\exists x \in X.\ \exists y \in X.\ \exists z \in X.\ x \neq y \wedge y \neq z \wedge x \neq z \wedge$ *ordering2 f betw X*

11.3 Chains using betweenness

Old definitions of chains. Shown equivalent to *fin-long-chain-2* in TemporalOrderOnPath.thy.

definition *chain-with* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a\ set \Rightarrow bool$ ([[.. - .. - .. - ..]-]) **where**
chain-with $x\ y\ z\ X \equiv [[x\ y\ z]] \wedge x \in X \wedge y \in X \wedge z \in X \wedge (\exists f.\ ordering\ f\ betw\ X)$
definition *finite-chain-with3* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a\ set \Rightarrow bool$ ([[- .. - .. -]-]) **where**
finite-chain-with3 $x\ y\ z\ X \equiv [[..x..y..z..]X] \wedge \neg(\exists w{\in}X.\ [[w\ x\ y]] \vee [[y\ z\ w]])$

lemma *long-chain-betw*: $[[..a..b..c..]X] \Longrightarrow [[a\ b\ c]]$
by (*simp add*: *chain-with-def*)

lemma *finite-chain3-betw*: $[[a..b..c]X] \Longrightarrow [[a\ b\ c]]$
by (*simp add*: *chain-with-def finite-chain-with3-def*)

definition *finite-chain-with2* :: $'a \Rightarrow 'a \Rightarrow 'a\ set \Rightarrow bool$ ([[- .. -]-]) **where**
finite-chain-with2 $x\ z\ X \equiv \exists y{\in}X.\ [[x..y..z]X]$

lemma *finite-chain2-betw*: $[[a..c]X] \Longrightarrow \exists b.\ [[a\ b\ c]]$
using *finite-chain-with2-def finite-chain3-betw* **by** *meson*

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-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-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-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* $\in$ *segment a b*› *seg-betw* **by** *auto*
  have *x*$\in$*P* $\wedge$ *y*$\in$*P*
    using ‹*x* $\in$ *segment a b*› ‹*y* $\in$ *segment a b*› *assms*(*2*) **by** *blast*
  have *x*$\in$*ab* $\wedge$ *y*$\in$*ab*
    using ‹*segment a b* $\subseteq$ *ab*› ‹*x* $\in$ *segment a b*› ‹*y* $\in$ *segment a b*› **by** *blast*
  have *ab*=*P*
     using ‹*path ab a b*› ‹*x* $\in$ *P* $\wedge$ *y* $\in$ *P*› ‹*x* $\in$ *ab* $\wedge$ *y* $\in$ *ab*› ‹*x* $\neq$ *y*› *assms*(*3*) *path-unique* **by** *auto*
  thus *?thesis*
    using ‹*path ab a b*› **by** *auto*
qed

lemma *pro-path*:
  assumes *x* $\in$ *prolongation a b*
  obtains *ab* **where** *path ab a b prolongation a b* $\subseteq$ *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* $\subseteq$ *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*: $[\![Q \in \mathcal{P};\ R \in \mathcal{P};\ S \in \mathcal{P};\ T \in \mathcal{P};\ Q \neq R;\ Q \neq S;\ R \neq S;\ a \in Q \cap R \wedge b \in Q \cap S \wedge c \in R \cap S;$
 $\exists\, d \in S.\ [[b\ c\ d]] \wedge (\exists\, e \in R.\ d \in T \wedge e \in T \wedge [[c\ e\ a]])]\!]$
 $\Longrightarrow \exists\, f \in T \cap Q.\ \exists\, X.\ [[a..f..b]X]$
begin

14 Chains: (Closest) Bounds

definition *is-bound-f* :: $'a \Rightarrow 'a\ set \Rightarrow (nat \Rightarrow 'a) \Rightarrow bool$ **where**
 $is\text{-}bound\text{-}f\ Q_b\ Q\ f \equiv$
 $\forall\, i\ j :: nat.\ [f[(f\ 0)..]Q] \wedge (i<j \longrightarrow [[(f\ i)\ (f\ j)\ Q_b]])$

definition *is-bound* :: $'a \Rightarrow 'a\ set \Rightarrow bool$ **where**
 $is\text{-}bound\ Q_b\ Q \equiv$
 $\exists\, f :: (nat \Rightarrow 'a).\ is\text{-}bound\text{-}f\ Q_b\ Q\ f$

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* :: $'a\ set \Rightarrow 'a\ set$ **where**
 $all\text{-}bounds\ Q = \{Q_b.\ is\text{-}bound\ Q_b\ Q\}$

definition *bounded* :: $'a\ set \Rightarrow bool$ **where**
 $bounded\ Q \equiv \exists\ Q_b.\ is\text{-}bound\ Q_b\ Q$

Just to make sure Continuity is not too strong.

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

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

definition *closest-bound* :: $'a \Rightarrow 'a\ set \Rightarrow bool$ **where**

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

end

15 MinkowskiUnreachable: I5-I7

locale *MinkowskiUnreachable* = *MinkowskiChain* +

assumes *two-in-unreach*: $[\![Q \in \mathcal{P};\ b \in \mathcal{E};\ b \notin Q]\!] \Longrightarrow \exists x{\in}\emptyset\ Q\ b.\ \exists y{\in}\emptyset\ Q\ b.\ x \neq y$

and *I6*: $[\![Q \in \mathcal{P};\ b \notin Q;\ b \in \mathcal{E};\ Qx \in (\emptyset\ Q\ b);\ Qz \in (\emptyset\ Q\ b);\ Qx{\neq}Qz]\!]$
$\Longrightarrow \exists X.\ \exists f.\ \textit{ch-by-ord}\ f\ X \wedge f\ 0 = Qx \wedge f\ (\textit{card}\ X - 1) = Qz$
$\wedge\ (\forall i{\in}\{1\ ..\ \textit{card}\ X - 1\}.\ (f\ i) \in \emptyset\ Q\ b$
$\wedge\ (\forall Qy{\in}\mathcal{E}.\ [[(f(i{-}1))\ Qy\ (f\ i)]] \longrightarrow Qy \in \emptyset\ Q\ b))$
$\wedge\ (\textit{short-ch}\ X \longrightarrow Qx{\in}X \wedge Qz{\in}X \wedge (\forall Qy{\in}\mathcal{E}.\ [[Qx\ Qy\ Qz]] \longrightarrow Qy \in \emptyset\ Q\ b))$
and *I7*: $[\![Q \in \mathcal{P};\ b \notin Q;\ b \in \mathcal{E};\ Qx \in Q - \emptyset\ Q\ b;\ Qy \in \emptyset\ Q\ b]\!]$
$\Longrightarrow \exists g\ X\ Qn.\ [g[Qx..Qy..Qn]X] \wedge Qn \in Q - \emptyset\ Q\ b$
begin

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

end

16 MinkowskiSymmetry: Symmetry

locale *MinkowskiSymmetry* = *MinkowskiUnreachable* +
assumes *Symmetry*: $[\![Q \in \mathcal{P};\ R \in \mathcal{P};\ S \in \mathcal{P};\ Q \neq R;\ Q \neq S;\ R \neq S;$
$x \in Q{\cap}R{\cap}S;\ Q_a \in Q;\ Q_a \neq x;$
$\emptyset\ Q\ \textit{from}\ Q_a\ \textit{via}\ R\ \textit{at}\ x = \emptyset\ Q\ \textit{from}\ Q_a\ \textit{via}\ S\ \textit{at}\ x]\!]$
$\Longrightarrow \exists \vartheta{::}'a{\Rightarrow}'a.$ ~~*i) there is a map* $\vartheta{:}\mathcal{E}{\Rightarrow}\mathcal{E}$~~
$\textit{bij-betw}\ (\lambda P.\ \{\vartheta\ y \mid y.\ y{\in}P\})\ \mathcal{P}\ \mathcal{P}$ ~~*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*~~

17 MinkowskiContinuity: Continuity

locale *MinkowskiContinuity* = *MinkowskiSymmetry* +
assumes *Continuity*: $\textit{bounded}\ Q \Longrightarrow (\exists Q_b.\ \textit{closest-bound}\ Q_b\ Q)$

18 MinkowskiSpacetime: Dimension (I4)

locale *MinkowskiSpacetime* = *MinkowskiContinuity* +

assumes *ex-3SPRAY* [*simp*]: $[\![\mathcal{E} \neq \{\}]\!] \Longrightarrow \exists\, x{\in}\mathcal{E}.\ \textit{three-SPRAY}\ x$
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*:
$\exists\, Q1{\in}\mathcal{P}.\ \exists\, Q2{\in}\mathcal{P}.\ \exists\, Q3{\in}\mathcal{P}.\ \exists\, Q4{\in}\mathcal{P}.\ Q1 \neq Q2 \wedge Q1 \neq Q3 \wedge Q1 \neq Q4 \wedge Q2 \neq Q3 \wedge Q2 \neq Q4 \wedge Q3 \neq Q4$
using *nonempty-events ex-3SPRAY three-SPRAY-ge4* **by** *blast*

end

end

theory *TemporalOrderOnPath*

imports *Main Minkowski TernaryOrdering*
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.

Disable list syntax.

no-translations
  $[x, xs] == x\#[xs]$
  $[x] == x\#[]$
no-syntax
  — list Enumeration
  -list :: *args* => *'a list* ([(-)])
no-notation *Cons* (**infixr** # *65*)
no-notation *Nil* ([])

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 *three-in-set3*:
  assumes $card\ X \geq 3$
  obtains $x\ y\ z$ **where** $x{\in}X$ **and** $y{\in}X$ **and** $z{\in}X$ **and** $x{\neq}y$ **and** $x{\neq}z$ **and** $y{\neq}z$
  using *assms* **by** (*auto simp add*: *card-le-Suc-iff numeral-3-eq-3*)

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

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** $\exists f$. *long-ch-by-ord f* {a,b,c}
  using *abc-abc-neq ord-ordered long-ch-by-ord-def assms*
  by (*smt insertI1 insert-commute*)

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

lemma *overlap-chain*: ⟦[[a b c]]; [[b c d]]⟧ $\Longrightarrow$ *ch* {a,b,c,d}
proof −
  assume [[a b c]] **and** [[b c d]]
  have $\exists f$. *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*)
    qed
    hence $\exists f$. *long-ch-by-ord f* {a,b,c,d}
      using ‹[[a b c]]› *abc-abc-neq long-ch-by-ord-def* **by** *auto*
    thus *?thesis*
      by (*simp add*: *ch-by-ord-def ch-def*)
  qed

end

20 3.1 Order on a finite chain

context *MinkowskiBetweenness* **begin**

20.1 Theorem 1, p18

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, p19

The lemma *abc-bcd-acd*, equal to the start of Schutz's proof, is given in *Minkowski* in order to prove some equivalences. Splitting it up into the proof of: "there is a betweenness relation for each ordered triple", and "all events of a chain are distinct" The first part is obvious with total chains (using *ordering*), and will be proved using the local definition as well (*ordering2*), following Schutz' proof. The second part is proved as injectivity of the indexing function (see *index-injective*).

For the case of two-element chains: the elements are distinct by definition, and the statement on ordering is void (respectively, *False* $\Longrightarrow$ P for any P).

theorem *order-finite-chain*:
  assumes *chX*: *long-ch-by-ord f X*
    and *finiteX*: *finite X*
    and *ordered-nats*: $0 \le (i{::}nat) \wedge i < j \wedge j < l \wedge l < card\ X$
  shows [[(*f i*) (*f j*) (*f l*)]]
  by (*metis chX long-ch-by-ord-def ordered-nats ordering-ord-ijk*)

lemma *thm2-ind1*:
  assumes *chX*: *long-ch-by-ord2 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* = λ *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 long-ch-by-ord2-def ordering2-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: long-ch-by-ord2 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 long-ch-by-ord2-def ordering2-ord-ijk)
        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 long-ch-by-ord2-def ordering2-def
            by (meson asm(3) less-imp-diff-less)
          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: long-ch-by-ord2 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-chain2*:
  assumes *chX*: *long-ch-by-ord2 f X*
    and *finiteX*: *finite X*
    and *ordered-nats*: $0 \leq (i{::}nat) \wedge i < j \wedge j < l \wedge l < card\ X$
  shows $[[(f\ i)\ (f\ j)\ (f\ l)]]$
proof −
  let $?n = card\ X - 1$
  have *ord1*: $0{\leq}i \wedge i{<}j \wedge j{<}?n$
    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*: $\forall k.\ 0{<}k \wedge k{<}l \longrightarrow [[(f\ (k{-}1))\ (f\ k)\ (f\ l)]]$
    using *thm2-ind2b chX finiteX ordered-nats*
    by *blast*
  have $j{<}l{-}1 \vee j{=}l{-}1$
    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

lemma *three-in-long-chain2*:
  assumes *long-ch-by-ord2 f X*
  obtains *x y z* **where** $x{\in}X$ **and** $y{\in}X$ **and** $z{\in}X$ **and** $x{\neq}y$ **and** $x{\neq}z$ **and** $y{\neq}z$
  using *assms*(*1*) *long-ch-by-ord2-def* **by** *auto*

lemma *short-ch-card-2*:
  assumes *ch-by-ord f X*
  shows *short-ch X* $\longleftrightarrow$ *card X* $= 2$
  by (*metis assms card-2-iff′ ch-by-ord-def long-ch-by-ord-def short-ch-def*)

lemma *long-chain2-card-geq*:
  assumes *long-ch-by-ord2 f X* **and** *fin*: *finite X*
  shows *card X* $\geq$ *3*
proof $-$
  obtain *x y z* **where** *xyz*: *x*$\in$*X y*$\in$*X z*$\in$*X* **and** *neq*: *x*$\neq$*y x*$\neq$*z y*$\neq$*z*
    using *three-in-long-chain2 assms*(*1*) **by** *blast*
  let *?S* = {*x*,*y*,*z*}
  have *?S* $\subseteq$ *X*
    by (*simp add*: *xyz*)
  moreover have *card ?S* $\geq$ *3*
    using *antisym* ‹*x* $\neq$ *y*› ‹*x* $\neq$ *z*› ‹*y* $\neq$ *z*› **by** *auto*
  ultimately show *?thesis*
    by (*meson neq fin three-subset*)
qed

lemma *fin-chain-card-geq-2*:
  assumes [*f*[*a*..*b*]*X*]
  shows *card X* $\geq$ *2*
  using *fin-chain-def* **apply** (*cases short-ch X*)
  using *short-ch-card-2*
  apply (*metis card-2-iff' dual-order.eq-iff short-ch-def*)
  using *assms fin-long-chain-def not-less* **by** *fastforce*

theorem *index-injective*:
  fixes *i*::*nat* **and** *j*::*nat*
  assumes *chX*: *long-ch-by-ord2 f X*
      and *finiteX*: *finite X*
      and *indices*: *i*<*j j*<*card X*
    shows *f i* $\neq$ *f j*
proof (*cases*)
  assume *Suc i* < *j*
  then have [[(*f i*) (*f* (*Suc*(*i*))) (*f j*)]]
    using *order-finite-chain2 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 *Suc*(*Suc i*) = *card X*
        by (*simp add*: ‹*Suc i* = *j*› ‹*Suc j* = *card X*›)

have *card X* $\geq$ *3*
using *assms(1) finiteX long-chain2-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-chain2* **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-chain2* **by** *blast*
thus *?thesis*
using *abc-abc-neq* **by** *blast*
qed
qed

end

21 Finite chain equivalence: local ¡-¿ global

context *MinkowskiBetweenness* **begin**

lemma *ch-equiv1*:
assumes *long-ch-by-ord f X finite X*
shows *long-ch-by-ord2 f X*
using *assms*
unfolding *long-ch-by-ord-def long-ch-by-ord2-def ordering-def ordering2-def*
by (*metis lessI*)

lemma *ch-equiv2*:
assumes *long-ch-by-ord2 f X finite X*
shows *long-ch-by-ord f X*
using *order-finite-chain2 assms*
unfolding *long-ch-by-ord-def long-ch-by-ord2-def ordering-def ordering2-def*
apply *safe* **by** *blast*

lemma *ch-equiv*:
assumes *finite X*
shows *long-ch-by-ord f X* ⟷ *long-ch-by-ord2 f X*
using *ch-equiv1 ch-equiv2 assms* **by** *blast*

end

22 Preliminary Results for Kinematic Triangles and Paths/Betweenness

Theorem 3-3.2 (collinearity), p20 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*: $\triangle\ a\ b\ c$
shows $\neg\ [[a\ b\ c]]$
using *tri-abc abc-ex-path triangle-diff-paths* **by** *blast*

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

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

lemma *triangle-not-betw-any*:
assumes *tri-abc*: $\triangle\ a\ b\ c$
shows $\neg\ (\exists\, d{\in}\{a,b,c\}.\ \exists\, e{\in}\{a,b,c\}.\ \exists f{\in}\{a,b,c\}.\ [[d\ e\ f]])$
by (*metis abc-ex-path abc-abc-neq empty-iff insertE tri-abc triangle-diff-paths*)

end

23 3.2 First collinearity theorem

theorem (**in** *MinkowskiChain*) *collinearity-alt2*:
assumes *tri-abc*: $\triangle\ a\ b\ c$
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 $\exists f{\in}de{\cap}ab.\ [[a\ f\ b]]$
proof −
have $\exists f{\in}ab \cap de.\ \exists X.\ [[a..f..b]X]$
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* [**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** *finite-chain3-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 $\exists f{\in}ab \cap de.\ \exists X.\ [[a..f..b]X]$
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* [**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** *finite-chain3-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 ($\exists f \in de \cap (path\text{-}of\ a\ b).\ [[a\ f\ b]]$)
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 $\exists f \in ?ab \cap de.\ \exists X.\ [[a..f..b]X]$
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* [**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** *finite-chain3-betw* **by** *blast*
qed

24 Additional results for Paths and Unreachables

context *MinkowskiPrimitive* **begin**

The degenerate case.

lemma *big-bang*:
assumes *no-paths*: $\mathcal{P} = \{\}$
shows $\exists\, a.\ \mathcal{E} = \{a\}$
proof −

have $\exists\, a.\ a \in \mathcal{E}$ **using** *nonempty-events* **by** *blast*
then obtain *a* **where** *a-event*: $a \in \mathcal{E}$ **by** *auto*
have $\neg\ (\exists\, b{\in}\mathcal{E}.\ b \neq a)$
proof (*rule notI*)
assume $\exists\, b{\in}\mathcal{E}.\ b \neq a$
then have $\exists\, Q.\ Q \in \mathcal{P}$ **using** *events-paths a-event* **by** *auto*
thus *False* **using** *no-paths* **by** *simp*
qed
then have $\forall\, b{\in}\mathcal{E}.\ b = a$ **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*:
$[\![Q \in \mathcal{P};\ a \in Q]\!] \Longrightarrow \emptyset\ Q\ a = \{\}$
apply (*unfold unreachable-subset-def*)
by *simp*

lemma *same-path-reachable*:
$[\![Q \in \mathcal{P};\ a \in Q;\ b \in Q]\!] \Longrightarrow a \in Q - \emptyset\ Q\ b$
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*:
$[\![Q \in \mathcal{P};\ R \in \mathcal{P};\ a \in Q;\ a \in R;\ b \in Q]\!] \Longrightarrow a \in R - \emptyset\ R\ b$
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 $b \in R - \emptyset\ R\ a$
unfolding *unreachable-subset-def* **using** *a-inQ b-inQ b-inR path-Q* **by** *auto*

lemma *reachable-path*:

```
  assumes path-Q: Q ∈ 𝒫
      and b-event: b ∈ ℰ
      and a-reachable: a ∈ Q − ∅ Q b
  shows ∃ R∈𝒫. a ∈ R ∧ b ∈ R
proof −
  have a-inQ: a ∈ Q using a-reachable by simp
  have Q ∉ 𝒫 ∨ b ∉ ℰ ∨ b ∈ Q ∨ (∃ R∈𝒫. b ∈ R ∧ a ∈ R)
      using a-reachable unreachable-subset-def by auto
  then have b ∈ Q ∨ (∃ R∈𝒫. b ∈ R ∧ a ∈ R) using path-Q b-event by simp
  thus ?thesis
  proof (rule disjE)
    assume b ∈ Q
    thus ?thesis using a-inQ path-Q by auto
  next
    assume ∃ R∈𝒫. b ∈ R ∧ a ∈ R
    thus ?thesis using conj-commute by simp
  qed
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∈ℰ using nonempty-events by blast
  have a∈Q ∨ a∉Q by auto
  thus ?thesis
  proof
    assume a∈Q
    thus ?thesis by blast
  next
    assume a∉Q
    then obtain b where b∈∅ Q a
      using two-in-unreach ‹a ∈ ℰ› assms
      by blast
    thus ?thesis
      using unreachable-subset-def by auto
  qed
qed

lemma events-ex-path:
  assumes ge1-path: 𝒫 ≠ {}
  shows ∀ x∈ℰ. ∃ Q∈𝒫. x ∈ 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 $\exists\, x{\in}\emptyset\ Q\ b.\ \exists\, y{\in}\emptyset\ Q\ b.\ x \neq y$
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*: $Q_x \in \emptyset\ Q\ b\ Q_z \in \emptyset\ Q\ b\ Q_x \neq Q_z$
and *X-def*: *ch-by-ord f X f 0* $= Q_x\ f\ (card\ X - 1) = Q_z$
$(\forall\, i{\in}\{1\ ..\ card\ X - 1\}.\ (f\ i) \in \emptyset\ Q\ b \wedge (\forall\, Q_y{\in}\mathcal{E}.\ [[(f(i-1))\ Q_y\ (f\ i)]] \longrightarrow Q_y \in \emptyset\ Q\ b))$
$(short\text{-}ch\ X \longrightarrow Q_x{\in}X \wedge Q_z{\in}X \wedge (\forall\, Q_y{\in}\mathcal{E}.\ [[Q_x\ Q_y\ Q_z]] \longrightarrow Q_y \in \emptyset\ Q\ b))$
shows $X{\subseteq}Q$

proof –
have *in-Q*: $Q_x{\in}Q \wedge Q_z{\in}Q$
using *unreachable-subset-def unreach*(*1*,*2*) **by** *blast*
have *fin-X*: *finite X*
using *unreach*(*3*) *not-less X-def* **by** *fastforce*
{
assume *short-ch X*
hence *?thesis*
by (*metis X-def*(*5*) *in-Q short-ch-def subsetI unreach*(*3*))
} moreover {
assume *asm*: *long-ch-by-ord f X*
have *?thesis*

proof
  fix x **assume** $x \in X$
  then obtain i **where** $f\ i = x\ i < card\ X$
    using *asm* **unfolding** *ch-by-ord-def long-ch-by-ord-def ordering-def*
    using *fin-X* **by** *auto*
  show $x \in Q$
  proof (*cases*)
    assume $x = Q_x \vee x = Q_z$
    thus *?thesis*
      using *in-Q* **by** *blast*
  next
    assume $\neg(x = Q_x \vee x = Q_z)$
    hence $x \neq Q_x\ x \neq Q_z$ **by** *linarith+*
    have $i > 0$
      using *X-def*(*2*) ‹$x \neq Q_x$› ‹$f\ i = x$› *gr-zeroI* **by** *force*
    have $i < card\ X - 1$
    using *X-def*(*3*) ‹$f\ i = x$› ‹$i < card\ X$› ‹$x \neq Q_z$› *less-imp-diff-less less-SucE*
      by (*metis Suc-pred′ cancel-comm-monoid-add-class.diff-cancel*)
    have $[[Q_x\ (f\ i)\ Q_z]]$
      using *X-def*(*2*,*3*) ‹$0 < i$› ‹$i < card\ X - 1$› *asm fin-X order-finite-chain*
      by *auto*
    thus *?thesis*
      by (*simp add*: ‹$f\ i = x$› *betw-b-in-path in-Q path-Q unreach*(*3*))
  qed
qed
}
ultimately show *?thesis*
  using *X-def*(*1*) *ch-by-ord-def* **by** *blast*
qed

end

25 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 $\exists\, d.\ d \notin Q \land d \in R$
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 $\exists\, R{\in}\mathcal{P}.\ R \neq Q \land d \in R$
using *assms external-path* **by** *fastforce*

end

26 3.3 Boundedness of the unreachable set

26.1 Theorem 4 (boundedness of the unreachable set), p20

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: $\exists\, X\ Q0\ Qm\ Qn.\ [[Q0\ ..\ Qm\ ..\ Qn]X] \wedge Q0 = ?Qx \wedge Qm = ?Qy \wedge Qn \in ?Q - \emptyset\ ?Q\ ?b$

theorem (**in** *MinkowskiUnreachable*) *unreachable-set-bounded*:
  assumes *path-Q*: $Q \in \mathcal{P}$
    and *b-nin-Q*: $b \notin Q$
    and *b-event*: $b \in \mathcal{E}$
    and *Qx-reachable*: $Qx \in Q - \emptyset\ Q\ b$
    and *Qy-unreachable*: $Qy \in \emptyset\ Q\ b$
  shows $\exists\, Qz \in Q - \emptyset\ Q\ b.\ [[Qx\ Qy\ Qz]] \wedge Qx \neq Qz$
  using *assms I7 order-finite-chain fin-long-chain-def*
  by (*metis fin-ch-betw*)

26.2 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*: $Q \in \mathcal{P}$
    and *all-inQ*: $\forall\, a \in \mathcal{E}.\ a \in Q$
    and *path-R*: $R \in \mathcal{P}$
  shows $R = Q$
proof (*rule ccontr*)
  assume $\neg\ R = Q$
  then have *R-neq-Q*: $R \neq Q$ **by** *simp*
  have $\mathcal{E} = Q$
    by (*simp add*: *all-inQ antisym path-Q path-sub-events subsetI*)
  hence $R \subset Q$
    using *R-neq-Q path-R path-sub-events* **by** *auto*
  obtain c **where** $c \notin R$ $c \in Q$
    using ‹$R \subset Q$› **by** *blast*
  then obtain a b **where** *path* $R\ a\ b$
    using ‹$\mathcal{E} = Q$› *path-R two-in-unreach unreach-ge2-then-ge2* **by** *blast*
  have $a \in Q$ $b \in Q$
    using ‹$\mathcal{E} = Q$› ‹*path* $R\ a\ b$› *in-path-event* **apply** *blast+* **done**
  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*: $Q \in \mathcal{P}$
  shows $\exists\, d{\in}\mathcal{E}.\ d \notin Q$
proof (*rule ccontr*)
  assume $\neg\ (\exists\, d{\in}\mathcal{E}.\ d \notin Q)$
  then have *all-inQ*: $\forall\, d{\in}\mathcal{E}.\ d \in Q$ **by** *simp*
  then have *only-one-path*: $\forall\, P{\in}\mathcal{P}.\ P = Q$ **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*: $b \in R - \emptyset\ R\ a$ **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** $u \in \emptyset\ R\ a$
using *two-in-unreach a-inQ in-path-event path-Q path-R* **by** *blast*
then obtain c **where** *c-reachable*: $c \in R - \emptyset\ R\ a$
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*: $a \neq b$ **using** *a-inQ ge2-events path-Q* **by** *blast*
have $\exists\, R \in \mathcal{P}.\ R \neq Q \wedge (\exists\, e.\ e \in R \wedge e \in Q)$ **by** (*simp add*: *ex-crossing-path path-Q*)
then obtain *R e* **where** *path-R*: $R \in \mathcal{P}$
and *R-neq-Q*: $R \neq Q$
and *e-inR*: $e \in R$
and *e-inQ*: $e \in Q$ **by** *auto*
thus *?thesis*
proof *cases*
assume *e-eq-a*: $e = a$
then have $\exists\, c.\ c \in \emptyset\ R\ b$ **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*: $e \neq a$

then have $\exists\, S \in \mathcal{P}.\ S \neq Q \wedge a \in S \wedge (\exists\, c.\ c \in S \wedge c \in R)$
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*: $Q \in \mathcal{P}$
and *a-inQ*: $a \in Q$
shows $\exists\, ac.\ \exists\, c.\ path\ ac\ a\ c \wedge ac \neq Q \wedge c \notin Q$
using *ex-crossing-at assms* **by** *fastforce*

end

27 3.4 Prolongation

context *MinkowskiSpacetime* **begin**

lemma (**in** *MinkowskiPrimitive*) *unreach-on-path*:
$a \in \emptyset\ Q\ b \Longrightarrow a \in Q$
using *unreachable-subset-def* **by** *simp*

lemma (**in** *MinkowskiUnreachable*) *unreach-equiv*:
$[\![Q \in \mathcal{P};\ R \in \mathcal{P};\ a \in Q;\ b \in R;\ a \in \emptyset\ Q\ b]\!] \Longrightarrow b \in \emptyset\ R\ a$
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*: $f \in \emptyset\ ae\ b$
using *two-in-unreach b-inQ in-path-event path-Q path-ae* **by** *blast*
then have *b-unreachable*: $b \in \emptyset\ Q\ f$ **using** *unreach-equiv*
by (*metis* (*mono-tags*, *lifting*) *CollectD b-inQ path-Q unreachable-subset-def*)
have *a-reachable*: $a \in Q - \emptyset\ Q\ f$
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 pro-*

long-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*: $a \neq b\ a \neq c\ b \neq c$
    by (*metis* ‹*card* $X = 3$› *empty-iff insert-iff order-refl three-in-set3*)+
  then consider $[[a\ b\ c]] \mid [[b\ c\ a]] \mid [[c\ a\ b]]$
    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** *auto*
  qed
next
  case *IH*: (*Suc n*)
  obtain $Y\ x$ **where** *X-eq*: $X = insert\ x\ Y$ **and** $x \notin Y$
    by (*meson IH.prems*(*4*) *Set.set-insert three-in-set3*)
  then have *card* $Y - 3 = n$ *card* $Y \geq 3$
    using *IH.hyps*(*2*) *IH.prems*(*3*) *X-eq* ‹$x \notin Y$› **by** *auto*
  then obtain $a\ b$ **where** *ab-Y*: $a \in Y\ b \in Y\ a \neq b$
          and *Y-ends*: $\forall\, c{\in}Y.\ (a \neq c \wedge b \neq c) \longrightarrow [[a\ c\ b]]$
    using *IH*(*1*) [*of* Y] *IH.prems*(*1*−*3*) *X-eq* **by** *auto*
  consider $[[a\ x\ b]] \mid [[x\ b\ a]] \mid [[b\ a\ x]]$
    using *some-betw* [*of* $Q\ a\ x\ b$] *ab-Y IH.prems*(*1*,*2*) *X-eq* ‹$x \notin Y$› **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* ∈ *X b* ≠ *c x* ≠ *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* ∈ 𝒫› *betw-a-in-path insert-iff some-betw subsetD*)
}
thus *?thesis*
using *X-eq* ‹*x* ∉ *Y*› *ab-Y(2)* **by** *fastforce*
qed
qed

lemma *obtain-fin-path-ends*:
assumes *path-X*: *X*∈𝒫
and *fin-Q*: *finite Q*
and *card-Q*: *card Q* ≥ *3*
and *events-Q*: *Q*⊆*X*
obtains *a b* **where** *a*≠*b* **and** *a*∈*Q* **and** *b*∈*Q* **and** ∀ *c*∈*Q*. (*a*≠*c* ∧ *b*≠*c*) ⟶ [[*a c b*]]
proof −
obtain *n* **where** *n*≥*0* **and** *card Q* = *n+3*
using *card-Q nat-le-iff-add*
by *auto*
then obtain *a b* **where** *a*≠*b* **and** *a*∈*Q* **and** *b*∈*Q* **and** ∀ *c*∈*Q*. (*a*≠*c* ∧ *b*≠*c*) ⟶ [[*a c b*]]
using *finite-path-has-ends assms* ‹*n*≥*0*›
by *metis*
thus *?thesis*
using *that* **by** *auto*
qed

lemma *path-card-nil*:
assumes *Q*∈𝒫
shows *card Q* = *0*
proof (*rule ccontr*)

```
  assume card Q ≠ 0
  obtain n where n = card Q
    by auto
  hence n≥1
    using ‹card Q ≠ 0› by linarith
  then consider (n1) n=1 | (n2) n=2 | (n3) n≥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 $P \neq \{\}$
by (*simp add*: *assms*)
hence $card\ P \neq 0$
by (*simp add*: *fin-P*)
moreover have $\neg(card\ P \geq 1)$
using *path-card-nil*
by (*simp add*: *assms*)
ultimately show *False*
by *simp*
qed

end

28 3.5 Second collinearity theorem

We start with a useful betweenness lemma.

lemma (**in** *MinkowskiBetweenness*) *some-betw2*:
assumes *path-Q*: $Q \in \mathcal{P}$
and *a-inQ*: $a \in Q$
and *b-inQ*: $b \in Q$
and *c-inQ*: $c \in Q$
shows $a = b \lor a = c \lor b = c \lor [[a\ b\ c]] \lor [[b\ c\ a]] \lor [[c\ a\ b]]$
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*: $a \notin bc$
shows $\triangle\ a\ b\ c$
proof −
have *abc-events*: $a \in \mathcal{E} \land b \in \mathcal{E} \land c \in \mathcal{E}$
using *path-ab path-bc path-ca in-path-event* **by** *auto*
have *abc-neq*: $a \neq b \land a \neq c \land b \neq c$
using *path-ab path-bc path-ca* **by** *auto*
have *paths-neq*: $ab \neq bc \land ab \neq ca \land bc \neq ca$
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*: $ab \neq bc$
shows $\triangle$ *a b c*
by (*meson ab-neq-bc cross-once-notin path-ab path-bc path-ca paths-tri*)

Schutz states it more like $[\![tri\text{-}abc;\ bcd;\ cea]\!] \Longrightarrow (path\ de\ d\ e \longrightarrow \exists f{\in}de.\ [[a\ f\ b]]{\wedge}[[d\ e\ f]])$ Equivalent up to usage of *impI*.

theorem (**in** *MinkowskiChain*) *collinearity2*:
assumes *tri-abc*: $\triangle$ *a b c*
and *bcd*: [[*b c d*]]
and *cea*: [[*c e a*]]
and *path-de*: *path de d e*
shows $\exists f{\in}de.$ [[*a f b*]] $\wedge$ [[*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*: $f \in de$ **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*: $d \in de \wedge e \in de \wedge f \in de$ **using** *path-de f-in-de* **by** *simp*
then have *five-poss*: $f = d \vee f = e \vee$ [[*e f d*]] $\vee$ [[*f d e*]] $\vee$ [[*d e f*]]
using *path-de some-betw2* **by** *blast*
have $f = d \vee f = e \longrightarrow (\exists\, Q{\in}\mathcal{P}.\ a \in Q \wedge b \in Q \wedge c \in Q)$
by (*metis abc-abc-neq afb bcd betw-a-in-path betw-b-in-path cea path-ab*)
then have *f-neq-d-e*: $f \neq d \wedge f \neq e$ **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 $dc{\neq}ce$
using *bcd betw-a-in-path betw-c-in-path cea path-ce path-dc tri-abc triangle-diff-paths*
by *blast*
hence $\triangle$ *d c e*
using *paths-tri2 path-ce path-dc path-de* **by** *blast*
then obtain *x* **where** *x-in-af*: $x \in af$
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*: $x \in dc$ **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 $fe \neq ea$
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 $\triangle$ 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

29 3.6 Order on a path - Theorems 8 and 9

context *MinkowskiSpacetime* **begin**

29.1 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*: $\triangle\ a\ b\ c$
    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*))
  show *?thesis*
  proof (*rule notI*)
    assume *path-a′b′c′*: $\exists\, Q{\in}\mathcal{P}.\ a' \in Q \wedge b' \in Q \wedge c' \in Q$
    consider $[[a'\ b'\ c']]\ |\ [[b'\ c'\ a']]\ |\ [[c'\ a'\ b']]$ **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*
    proof (*cases*)
      assume *a′b′c′*: $[[a'\ b'\ c']]$
      then have *c′b′a′*: $[[c'\ b'\ a']]$ **using** *abc-sym* **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 path-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 *False* **using** *ca′b abc-only-cba* **by** *blast*
next
assume *b′c′a′*: $[[b'\ c'\ a']]$
then have *a′c′b′*: $[[a'\ c'\ b']]$ **using** *abc-sym* **by** *simp*
have *nopath-a′cb′*: $\neg\ (\exists\, Q{\in}\mathcal{P}.\ a' \in Q \wedge c \in Q \wedge b' \in Q)$
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 *ab-inQ*: $a \in Q \wedge b \in Q$
using *eq-paths*
by (*metis ab′c abc-a′b′c′-neq betw-a-in-path betw-c-in-path ca′b*)
thus *False* **using** *c-inQ path-Q tri-abc triangle-diff-paths* **by** *blast*
qed
then have *tri-a′cb′*: $\triangle\ a'\ c\ b'$
by (*smt ab′c abc-ex-path-unique b′c′a′ ca′b paths-tri*)
obtain bc' **where** *path-bc′*: *path* $bc'\ b\ c'$
using *abc-a′b′c′-neq abc-ex-path-unique bc′a*
by *blast*
obtain $b'c$ **where** *path-b′c*: *path* $b'c\ b'\ c$ **using** *tri-a′cb′ triangle-paths*(*3*) **by** *blast*
then have $\exists\, x{\in}b'c.\ [[b'\ x\ c]] \wedge [[b\ c'\ x]]$
using *collinearity2* [**where** $a = b'$ **and** $b = c$ **and** $c = a'$
and $e = c'$ **and** $d = b$ **and** $de = bc'$]
bc′a betw-b-in-path a′c′b′ path-bc′ tri-a′cb′
by (*meson ca′b triangle-permutes*(*5*))
then obtain x **where** *x-in-b′c*: $x \in b'c$
and *b′xc*: $[[b'\ x\ c]]$
and *bc′x*: $[[b\ c'\ x]]$ **by** *blast*
have *a-in-bc′*: $a \in bc'$ **using** *bc′a betw-c-in-path path-bc′* **by** *blast*
have *a-in-b′c*: $a \in b'c$ **using** *ab′c betw-a-in-path path-b′c* **by** *blast*
have *bc′-b′c-distinct*: $bc' \neq b'c$
using *a-in-bc′ path-b′c path-bc′ tri-abc triangle-diff-paths* **by** *blast*
have $bc' \cap b'c = \{a\}$
using *paths-cross-at bc′-b′c-distinct a-in-b′c a-in-bc′ path-b′c path-bc′* **by**

auto
    then have $x = a$ **using** *bc′x betw-c-in-path path-bc′ x-in-b′c* **by** *auto*
    then have $[[b' \ a \ c]]$ **using** *b′xc* **by** *auto*
    thus *False* **using** *ab′c abc-only-cba* **by** *blast*
  next
    assume *c′a′b′*: $[[c' \ a' \ b']]$
    then have *b′a′c′*: $[[b' \ a' \ c']]$ **using** *abc-sym* **by** *simp*
    have *nopath-c′ab′*: $\neg (\exists Q\in\mathcal{P}. \ c' \in Q \wedge a \in Q \wedge b' \in Q)$
    proof (*rule notI*)
      assume $\exists Q\in\mathcal{P}. \ c' \in Q \wedge a \in Q \wedge b' \in Q$
      then obtain Q **where** *path-Q*: $Q \in \mathcal{P}$
                and *c′-inQ*: $c' \in Q$
                and *a-inQ*: $a \in Q$
                and *b′-inQ*: $b' \in Q$ **by** *blast*
      then have *bc-inQ*: $b \in Q \wedge c \in Q$
          using *eq-paths ab′c abc-a′b′c′-neq bc′a betw-a-in-path betw-c-in-path* **by**
blast
      thus *False* **using** *a-inQ path-Q tri-abc triangle-diff-paths* **by** *blast*
    qed
    then have *tri-a′cb′*: $\triangle \ b' \ a \ c'$
        by (*smt bc′a abc-ex-path-unique c′a′b′ ab′c paths-tri*)
    obtain *ca′* **where** *path-ca′*: *path ca′ c a′*
        using *abc-a′b′c′-neq abc-ex-path-unique ca′b*
        by *blast*
    obtain *c′a* **where** *path-c′a*: *path c′a c′ a* **using** *tri-a′cb′ triangle-paths*(*3*) **by**
blast
    then have $\exists x\in c'a. \ [[c' \ x \ a]] \wedge [[c \ a' \ x]]$
        using *collinearity2* [**where** $a = c'$ **and** $b = a$ **and** $c = b'$
                and $e = a'$ **and** $d = c$ **and** $de = ca'$]
            ab′c b′a′c′ betw-b-in-path path-ca′ tri-a′cb′ triangle-permutes(*5*) **by**
blast
    then obtain x **where** *x-in-c′a*: $x \in c'a$
                and *c′xa*: $[[c' \ x \ a]]$
                and *ca′x*: $[[c \ a' \ x]]$ **by** *blast*
    have *b-in-ca′*: $b \in ca'$ **using** *betw-c-in-path ca′b path-ca′* **by** *blast*
    have *b-in-c′a*: $b \in c'a$ **using** *bc′a betw-a-in-path path-c′a* **by** *auto*
    have *ca′-c′a-distinct*: $ca' \neq c'a$
        using *b-in-c′a path-c′a path-ca′ tri-abc triangle-diff-paths* **by** *blast*
    have $ca' \cap c'a = \{b\}$
        using *b-in-c′a b-in-ca′ ca′-c′a-distinct path-c′a path-ca′ paths-cross-at* **by**
auto
    then have $x = b$ **using** *betw-c-in-path ca′x path-ca′ x-in-c′a* **by** *auto*
    then have $[[c' \ b \ a]]$ **using** *c′xa* **by** *auto*
    thus *False* **using** *abc-only-cba bc′a* **by** *blast*
  qed
 qed
qed

29.2 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*: $d' \notin \emptyset\ ab\ e\ d' \in ab\ d' \neq e$
    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 $\neg(\exists\, R \in \mathcal{P}.\ d' \in R \wedge e \in R \wedge d' \neq e)$
    by *blast*
  hence $\neg(\exists\, R \in \mathcal{P}.\ \ e \in R \wedge d' \in R)$
    using *d′-def(3)* **by** *blast*
  moreover have $ab \in \mathcal{P} \wedge e \in \mathcal{E} \wedge e \notin ab$
    by (*simp add*: *e-event e-notin-S path-ab*)
  ultimately have $d' \in \emptyset\ ab\ e$
    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 $b \in ab - \emptyset\ ab\ e$
    using *cross-in-reachable path-ab path-be*
    by *blast*
  have $d \in \emptyset\ ab\ e$

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 $[[b\ d\ d']]$ $d' \in ab$ $d' \notin \emptyset\ ab\ e$ $b \neq d'$ $e \neq d'$
using *unreachable-set-bounded* ‹$b \in ab - \emptyset\ ab\ e$› ‹$d \in \emptyset\ ab\ e$› *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*: $c \neq d$
and *path-ab*: *path ab a b*
and *path-S*: $S \in \mathcal{P}$
and *a-inS*: $a \in S$
and *e-inS*: $e \in S$
and *e-neq-a*: $e \neq a$
and *S-neq-ab*: $S \neq ab$
and *path-be*: *path be b e*
shows $\exists\, c'\ d'.\ \exists\, d'e\ c'e.\ c' \in ab \wedge d' \in ab$
$\wedge [[a\ b\ d']] \wedge [[c'\ b\ a]] \wedge [[c'\ b\ d']]$
$\wedge\ path\ d'e\ d'\ e \wedge path\ c'e\ c'\ e$
proof (*cases*)
assume $\exists\, de.\ path\ de\ d\ e$
then obtain *de* where *path de d e*
by *blast*
hence $[[a\ b\ d]] \wedge d \in ab$
using *abd betw-c-in-path path-ab* by *blast*
thus *?thesis*
proof (*cases*)
assume $\exists\, ce.\ path\ ce\ c\ e$
then obtain *ce* where *path ce c e* by *blast*

have $c \in ab$
using *abc betw-c-in-path path-ab* **by** *blast*
thus *?thesis*
using ‹$[[a\ b\ d]] \wedge d \in ab$› ‹$\exists\, ce.\ path\ ce\ c\ e$› ‹$c \in ab$› ‹$path\ de\ d\ e$› *abc abc-sym dbc*
by *blast*
next
assume $\neg(\exists\, ce.\ path\ ce\ c\ e)$
obtain $c'\ c'e$ **where** $c' \in ab \wedge path\ c'e\ c'\ e \wedge [[b\ c\ c']]$
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 ‹$\neg(\exists\, ce.\ path\ ce\ c\ e)$› *a-inS abc e-inS e-neq-a path-S path-ab path-be*
by (*metis* (*mono-tags*, *lifting*))
hence $[[a\ b\ c']] \wedge [[d\ b\ c']]$
using *abc dbc* **by** *blast*
hence $[[c'\ b\ a]] \wedge [[c'\ b\ d]]$
using *theorem1* **by** *blast*
thus *?thesis*
using ‹$[[a\ b\ d]] \wedge d \in ab$› ‹$c' \in ab \wedge path\ c'e\ c'\ e \wedge [[b\ c\ c']]$› ‹$path\ de\ d\ e$›
by *blast*
qed
next
assume $\neg\ (\exists\, de.\ path\ de\ d\ e)$
obtain $d'\ d'e$ **where** d'*-in-ab*: $d' \in ab$
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 ‹$\nexists\, de.\ path\ de\ d\ e$› *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 $\exists\, ce.\ path\ ce\ c\ e$
then obtain ce **where** $path\ ce\ c\ e$ **by** *blast*
have $c \in ab$
using *abc betw-c-in-path path-ab* **by** *blast*
thus *?thesis*
using ‹$[[a\ b\ d']]$› ‹$d' \in ab$› ‹$path\ ce\ c\ e$› ‹$c \in ab$› ‹$path\ d'e\ d'\ e$› *abc abc-sym dbc*
by (*meson abc-bcd-acd bdd'*)
next
assume $\neg(\exists\, ce.\ path\ ce\ c\ e)$
obtain $c'\ c'e$ **where** $c' \in ab \wedge path\ c'e\ c'\ e \wedge [[b\ c\ c']]$
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 ‹$\neg(\exists\, ce.\ path\ ce\ c\ e)$› *a-inS abc e-inS e-neq-a path-S path-ab path-be*
by (*metis* (*mono-tags*, *lifting*))
hence $[[a\ b\ c']] \wedge [[d\ b\ c']]$
using *abc dbc* **by** *blast*

hence $[[c'\ b\ a]] \wedge [[c'\ b\ d]]$
using *theorem1* **by** *blast*
thus *?thesis*
using ‹$[[a\ b\ d']]$› ‹$c' \in ab \wedge path\ c'e\ c'\ e \wedge [[b\ c\ c']]$› ‹$path\ d'e\ d'\ e$› 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*: $S \neq$ *path-of a b*
shows $\exists\, c'\ d'.\ [[a\ b\ d']] \wedge [[c'\ b\ a]] \wedge [[c'\ b\ d']] \wedge$ *path-ex* $d'\ e \wedge$ *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 $c{\neq}d$ **using** *abc-ac-neq dbc* **by** *blast*
{
case *True*
then obtain *de* **where** *path de d e*
by *blast*
hence $[[a\ b\ d]] \wedge d{\in}?ab$
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 $c \in ?ab$
using *abc betw-c-in-path path-ab* **by** *blast*
thus *?thesis*
using ‹$[[a\ b\ d]] \wedge d \in ?ab$› ‹$\exists\, ce.\ path\ ce\ c\ e$› ‹$c \in ?ab$› ‹$path\ de\ d\ e$› *abc abc-sym dbc*
by *blast*
next
case *False*
obtain $c'\ c'e$ **where** $c'{\in}?ab \wedge path\ c'e\ c'\ e \wedge [[b\ c\ c']]$
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 ‹$\neg(\exists\, ce.\ path\ ce\ c\ e)$› *abc path-S path-ab path-be*
by (*metis* (*mono-tags*, *lifting*))
hence $[[a\ b\ c']] \wedge [[d\ b\ c']]$
using *abc dbc* **by** *blast*

hence $[[c'\ b\ a]] \wedge [[c'\ b\ d]]$
using *theorem1* **by** *blast*
thus *?thesis*
using $\langle[[a\ b\ d]] \wedge d \in ?ab\rangle$ $\langle c' \in ?ab \wedge path\ c'e\ c'\ e \wedge [[b\ c\ c']]\rangle$ $\langle path\ de\ d\ e\rangle$
by *blast*
qed
} {
case *False*
obtain $d'\ d'e$ **where** d'*-in-ab*: $d' \in ?ab$
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 $\langle\neg$*path-ex d e*$\rangle$ *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 $c \in ?ab$
using *abc betw-c-in-path path-ab* **by** *blast*
thus *?thesis*
using $\langle[[a\ b\ d']]\rangle$ $\langle d' \in ?ab\rangle$ $\langle path\ ce\ c\ e\rangle$ $\langle c \in ?ab\rangle$ $\langle path\ d'e\ d'\ e\rangle$ *abc abc-sym dbc*
by (*meson abc-bcd-acd* bdd')
next
case *False*
obtain $c'\ c'e$ **where** $c'{\in}?ab \wedge path\ c'e\ c'\ e \wedge [[b\ c\ c']]$
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 $\langle\neg$(*path-ex c e*)$\rangle$ *abc path-S path-ab path-be*
by (*metis* (*mono-tags*, *lifting*))
hence $[[a\ b\ c']] \wedge [[d\ b\ c']]$
using *abc dbc* **by** *blast*
hence $[[c'\ b\ a]] \wedge [[c'\ b\ d]]$
using *theorem1* **by** *blast*
thus *?thesis*
using $\langle[[a\ b\ d']]\rangle$ $\langle c' \in ?ab \wedge path\ c'e\ c'\ e \wedge [[b\ c\ c']]\rangle$ $\langle path\ d'e\ d'\ e\rangle$ bdd' d'*-in-ab*
by *blast*
qed
}
qed

lemma *exist-*f'*-alt*:
assumes *path-ab*: *path ab a b*
and *path-S*: $S \in \mathcal{P}$

and *a-inS*: $a \in S$
and *e-inS*: $e \in S$
and *e-neq-a*: $e \neq a$
and *f-def*: $[[e\ c'\ f]]\ f{\in}c'e$
and *S-neq-ab*: $S \neq ab$
and *c'd'-def*: $c'{\in}ab \wedge d'{\in}ab$
$\wedge [[a\ b\ d']] \wedge [[c'\ b\ a]] \wedge [[c'\ b\ d']]$
$\wedge$ *path* $d'e\ d'\ e \wedge$ *path* $c'e\ c'\ e$
shows $\exists f'.\ \exists f'b.\ [[e\ c'\ f']] \wedge$ *path* $f'b\ f'\ b$
proof (*cases*)
assume $\exists\, bf.$ *path* $bf\ b\ f$
thus *?thesis*
using ‹$[[e\ c'\ f]]$› **by** *blast*
next
assume $\neg(\exists\, bf.$ *path* $bf\ b\ f)$
hence $f \in \emptyset\ c'e\ b$
using *assms*$(1{-}5,7{-}9)$ *abc-abc-neq betw-events eq-paths unreachable-bounded-path-only*
by *metis*
moreover have $c' \in c'e - \emptyset\ c'e\ b$
using *c'd'-def cross-in-reachable path-ab* **by** *blast*
moreover have $b{\in}\mathcal{E} \wedge b{\notin}c'e$
using ‹$f \in \emptyset\ c'e\ b$› *betw-events c'd'-def same-empty-unreach* **by** *auto*
ultimately obtain f' **where** *f'-def*: $[[c'\ f\ f']]\ f'{\in}c'e\ f'{\notin}\ \emptyset\ c'e\ b\ c'{\neq}f'\ b{\neq}f'$
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 *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* **apply** *blast*+ **done**
have $f \in \emptyset\ c'e\ b$
  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 $c' \in c'e - \emptyset\ c'e\ b$
  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 ‹$f \in \emptyset\ c'e\ b$› *betw-events* $c'd'$*-def same-empty-unreach* **by** *auto*
ultimately obtain f' **where** f'*-def*: $[[c'\ f\ f']]$ $f'{\in}c'e$ $f'{\notin}\ \emptyset\ c'e\ b$ $c'{\neq}f'$ $b{\neq}f'$
  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 $[[b\ c\ d]] \vee [[b\ d\ c]]$
proof −
  have $\neg\ [[d\ b\ c]]$
  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*: $x \in \emptyset\ S\ b$
              and *y-in-unreach*: $y \in \emptyset\ S\ b$
              and *x-neq-y*: $x \neq y$
              and *z-in-reach*: $z \in S - \emptyset\ S\ b$

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*: $w \in S - \emptyset\ S\ b$
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 $\exists\, c'\ d'.$
$c' \in ab \wedge d' \in ab$
$\wedge$ [[$a\ b\ d'$]] $\wedge$ [[$c'\ b\ a$]] $\wedge$ [[$c'\ b\ d'$]]
$\wedge$ *path-ex* $d'\ e \wedge$ *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*: $c' \in ab \wedge d' \in ab$
$\wedge$ [[$a\ b\ d'$]] $\wedge$ [[$c'\ b\ a$]] $\wedge$ [[$c'\ b\ d'$]]
$\wedge$ *path* $d'e\ d'\ e \wedge$ *path* $c'e\ c'\ e$
by *blast*

obtain f **where** *f-def*: $f \in c'e$ [[$e\ c'\ f$]]
using *c'd'-def prolong-betw2* **by** *blast*
then obtain $f'\ f'b$ **where** *f'-def*: [[$e\ c'\ f'$]] $\wedge$ *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*: $\triangle\ a\ e\ c'$

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*: $h \in f'b$
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 **by** *blast*
have *tri-dec*: $\triangle\ d'\ e\ c'$
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*: $g \in f'b$
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 **by** *blast*
have $\triangle\ e\ a\ d'$ **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*: $a \neq d$ **using** *abc-ac-neq acd* **by** *auto*
      then have $[[c\ a\ d]] \vee [[c\ d\ a]]$ **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*: $b{\neq}c$
   shows $[[a\ b\ c]] \lor [[a\ c\ b]]$
proof −
  obtain P **where** *P-def*: $P{\in}\mathcal{P}\ a{\in}P\ b{\in}P\ d{\in}P$
   using *abd abc-ex-path* **by** *blast*
  hence $c{\in}P$
   using *acd abc-abc-neq betw-b-in-path* **by** *blast*
  have $\neg[[b\ a\ c]]$
   using *abc-only-cba abd acd* **by** *blast*
  thus *?thesis*
   by (*metis P-def(1−3)* ‹$c \in P$› *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*: $[[a\ b\ d]] \lor [[a\ d\ b]]$
   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*: $[[a\ b\ c]] \lor [[a\ c\ b]] \lor [[b\ a\ c]]$
    using *assms* **by** (*meson in-path-event abc-sym some-betw insert-subset*)
  have *ch1*: $[[a\ b\ c]] \longrightarrow ch\ \{a,b,c\}$
    using *abc-abc-neq ch-by-ord-def ch-def ord-ordered between-chain* **by** *auto*
  have *ch2*: $[[a\ c\ b]] \longrightarrow ch\ \{a,c,b\}$
    using *abc-abc-neq ch-by-ord-def ch-def ord-ordered between-chain* **by** *auto*
  have *ch3*: $[[b\ a\ c]] \longrightarrow ch\ \{b,a,c\}$
    using *abc-abc-neq ch-by-ord-def ch-def ord-ordered between-chain* **by** *auto*
  show *?thesis*
    using *abc-betw ch1 ch2 ch3* **by** (*metis insert-commute*)
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 ordering (abcd) explicitly (for arbitrarily named events), but is equivalent.

theorem *chain4*:
  assumes *path-Q*: $Q \in \mathcal{P}$
    and *inQ*: $a \in Q\ b \in Q\ c \in Q\ d \in Q$
    and *abcd-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 $ch\ \{a,b,c,d\}$
proof –
  obtain $a'\ b'\ c'$ **where** *a′-pick*: $a' \in \{a,b,c,d\}$
    and *b′-pick*: $b' \in \{a,b,c,d\}$
    and *c′-pick*: $c' \in \{a,b,c,d\}$
    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*: $d' \neq a' \land d' \neq b' \land d' \neq c'$
    and *d′-pick*: $d' \in \{a,b,c,d\}$
    using *insert-iff abcd-neq* **by** *metis*
  have *all-picked-on-path*: $a' \in Q\ b' \in Q\ c' \in Q\ d' \in Q$
    using *a′-pick b′-pick c′-pick d′-pick inQ* **by** *blast+*
  consider $[[d'\ a'\ b']] \mid [[a'\ d'\ b']] \mid [[a'\ b'\ d']]$
    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*: $[[b'\ c'\ d']] \lor [[b'\ d'\ c']]$ **using** *abc-abd-bcdbdc a′b′c′ d′-neq*
by *blast*

have *case1*: $[\![b'\ c'\ d']\!] \Longrightarrow$ *?thesis* **using** *a′b′c′ overlap-chain* **by** *blast*
have *case2*: $[\![b'\ d'\ c']\!] \Longrightarrow$ *?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

end

30 Interlude - Chains and Equivalences

This section is meant for our alternative definitions of chains, and proofs of equivalence. If we want to regain full independence of our axioms, we probably need to shuffle a few things around. Some of this may be redundant, but is kept for compatibility with legacy proofs.

Three definitions are given (cf 'Betweenness: Chains' in Minkowski.thy): - one relying on explicit betweenness conditions - one relying on a total ordering and explicit indexing - one equivalent to the above except for use of the weaker, local-only ordering2

context *MinkowskiChain* **begin**

30.1 Proofs for totally ordered index-chains

30.1.1 General results

lemma *inf-chain-is-long*:
assumes *semifin-chain f x X*
shows *long-ch-by-ord f X* $\wedge$ *f 0* $= x \wedge$ *infinite X*
proof −
have *infinite X* $\longrightarrow$ *card X* $\neq$ *2* **using** *card.infinite* **by** *simp*
hence *semifin-chain f x X* $\longrightarrow$ *long-ch-by-ord f X*

using *long-ch-by-ord-def semifin-chain-def short-ch-def*
by *simp*
thus *?thesis* **using** *assms semifin-chain-def* **by** *blast*
qed

A reassurance that the starting point x is implied.

lemma *long-inf-chain-is-semifin*:
assumes *long-ch-by-ord f X* $\wedge$ *infinite X*
shows $\exists$ *x*. $[f[x..]X]$
by (*simp add*: *assms semifin-chain-def*)

lemma *endpoint-in-semifin*:
assumes *semifin-chain f x X*
shows $x \in X$
using *assms semifin-chain-def zero-into-ordering inf-chain-is-long long-ch-by-ord-def*
by (*metis finite.emptyI*)

lemma *three-in-long-chain*:
assumes *long-ch-by-ord f X* **and** *fin*: *finite X*
obtains *x y z* **where** $x \in X$ **and** $y \in X$ **and** $z \in X$ **and** $x \neq y$ **and** $x \neq z$ **and** $y \neq z$
using *assms*(*1*) *long-ch-by-ord-def* **by** *auto*

30.1.2 Index-chains lie on paths

lemma *all-aligned-on-semifin-chain*:
assumes $[f[x..]X]$
and *a*: $y \in X$ **and** *b*:$z \in X$ **and** *xy*: $x \neq y$ **and** *xz*: $x \neq z$ **and** *yz*: $y \neq z$
shows $[[x\ y\ z]] \vee [[x\ z\ y]]$
proof −
obtain n_y n_z **where** $f\ n_y = y$ **and** $f\ n_z = z$
by (*metis TernaryOrdering.ordering-def a assms*(*1*) *b inf-chain-is-long long-ch-by-ord-def*)
have $(0 < n_y \wedge n_y < n_z) \vee (0 < n_z \wedge n_z < n_y)$
using $\langle f\ n_y = y \rangle$ $\langle f\ n_z = z \rangle$ *assms less-linear semifin-chain-def xy xz yz* **by**
auto
hence $[[(f\ 0)\ (f\ n_y)\ (f\ n_z)]] \vee [[(f\ 0)\ (f\ n_z)\ (f\ n_y)]]$
using *ordering-def assms*(*1*) *long-ch-by-ord-def semifin-chain-def*
by (*metis long-ch-by-ord-def*)
thus $[[x\ y\ z]] \vee [[x\ z\ y]]$
using $\langle f\ n_y = y \rangle$ $\langle f\ n_z = z \rangle$ *assms semifin-chain-def* **by** *auto*
qed

lemma *semifin-chain-on-path*:
assumes $[f[x..]X]$
shows $\exists P \in \mathcal{P}.\ X \subseteq P$
proof −
obtain *y* **where** $y \in X$ **and** $y \neq x$
using *assms inf-chain-is-long*
by (*metis Diff-iff all-not-in-conv finite-Diff2 finite-insert infinite-imp-nonempty*

insert-iff)
  have *path-exists*: $\exists\, P{\in}\mathcal{P}.$ *path P x y*
  proof −
    obtain *e* **where** $e{\in}X$ **and** $e{\neq}x$ **and** $e{\neq}y$ **and** $[[x\ y\ e]] \vee [[x\ e\ y]]$
      using *all-aligned-on-semifin-chain inf-chain-is-long long-ch-by-ord-def assms ordering-def lessI* ‹$y \in X$› ‹$y \neq x$› *finite.emptyI finite-insert finite-subset insert-iff subsetI*
      by *smt*
    obtain *P* **where** *path P x y*
      using ‹$[[x\ y\ e]] \vee [[x\ e\ y]]$› *abc-abc-neq abc-ex-path*
      by *blast*
    show *?thesis*
      using ‹*path P x y*›
      by *blast*
  qed
  obtain *P* **where** *path P x y*
    using *path-exists*
    by *blast*
  have $X{\subseteq}P$
  proof
    fix *e*
    assume $e{\in}X$
    show $e{\in}P$
    proof −
      have $e{=}x \vee e{=}y \vee (e{\neq}x \wedge e{\neq}y)$ **by** *auto*
      moreover **{** **assume** $e{\neq}x \wedge e{\neq}y$
        have $[[x\ y\ e]] \vee [[x\ e\ y]]$
          using *all-aligned-on-semifin-chain assms* ‹$e \in X$› ‹$e \neq x \wedge e \neq y$› ‹$y \in X$› ‹$y \neq x$›
          by *blast*
        hence *?thesis*
          using ‹*path P x y*› *abc-ex-path path-unique*
          by *blast*
      } **moreover** **{** **assume** $e{=}x$
        have *?thesis*
          by (*simp add*: ‹$e = x$› ‹*path P x y*›)
      } **moreover** **{** **assume** $e{=}y$
        have $e{\in}P$
          by (*simp add*: ‹$e = y$› ‹*path P x y*›)
      }
      ultimately show *?thesis* **by** *blast*
    qed
  qed
  thus *?thesis*
    using ‹*path P x y*›
    by *blast*
qed

lemma *card2-either-elt1-or-elt2*:
  assumes *card X* = *2* **and** $x \in X$ **and** $y \in X$ **and** $x \neq y$
    and $z \in X$ **and** $z \neq x$
  shows $z = y$
by (*metis assms card-2-iff'*)

lemma *short-chain-on-path*:
  assumes *short-ch X*
  shows $\exists P \in \mathcal{P}.\ X \subseteq P$
proof −
  obtain *x y* **where** $x \neq y$ **and** $x \in X$ **and** $y \in X$
    using *assms short-ch-def* **by** *auto*
  obtain *P* **where** *path P x y*
    using ‹$x \in X$› ‹$x \neq y$› ‹$y \in X$› *assms short-ch-def*
    by *metis*
  have $X \subseteq P$
  proof
    fix *z*
    assume $z \in X$
    show $z \in P$
    proof *cases*
      assume $z = x$
      show $z \in P$ **using** ‹*path P x y*› **by** (*simp add*: ‹$z = x$›)
    next
      assume $z \neq x$
      have $z = y$
        using ‹$x \in X$› ‹$y \in X$› ‹$z \neq x$› ‹$z \in X$› ‹$x \neq y$› *assms short-ch-def*
        by *metis*
      thus $z \in P$ **using** ‹*path P x y*› **by** (*simp add*: ‹$z = y$›)
    qed
  qed
  thus *?thesis*
    using ‹*path P x y*› **by** *blast*
qed

lemma *all-aligned-on-long-chain*:
  assumes *long-ch-by-ord f X* **and** *finite X*
  and *a*: $x \in X$ **and** *b*: $y \in X$ **and** *c*:$z \in X$ **and** *xy*: $x \neq y$ **and** *xz*: $x \neq z$ **and** *yz*: $y \neq z$
shows $[[x\ y\ z]] \vee [[x\ z\ y]] \vee [[z\ x\ y]]$
proof −
  obtain n_x n_y n_z **where** *fx*: $f\ n_x = x$ **and** *fy*: $f\ n_y = y$ **and** *fz*: $f\ n_z = z$
        and *xx*: $n_x < card\ X$ **and** *yy*: $n_y < card\ X$ **and** *zz*: $n_z < card\ X$
  proof −
    assume *a1*: $\bigwedge n_x\ n_y\ n_z.\ [\![f\ n_x = x;\ f\ n_y = y;\ f\ n_z = z;\ n_x < card\ X;\ n_y < card\ X;\ n_z < card\ X]\!] \Longrightarrow thesis$
    obtain *nn* :: *'a set* ⇒ (*nat* ⇒ *'a*) ⇒ *'a* ⇒ *nat* **where**
      $\bigwedge a\ A\ f\ p\ pa.\ (a \notin A \vee \neg\ ordering\ f\ p\ A \vee f\ (nn\ A\ f\ a) = a)$
          $\wedge\ (infinite\ A \vee a \notin A \vee \neg\ ordering\ f\ pa\ A \vee nn\ A\ f\ a < card\ A)$

by (*metis* (*no-types*) *ordering-def*)
then show *?thesis*
using *a1* **by** (*metis a assms*(*1*) *assms*(*2*) *b c long-ch-by-ord-def*)
qed
have *less-or*: $(n_x<n_y \wedge n_y<n_z) \vee (n_x<n_z \wedge n_z<n_y) \vee (n_z<n_x \wedge n_x<n_y) \vee$
$(n_z<n_y \wedge n_y<n_x) \vee (n_y<n_z \wedge n_z<n_x) \vee (n_y<n_x \wedge n_x<n_z)$
using *fx fy fz assms less-linear*
by *metis*
have *int-imp-1*: $(n_x<n_y \wedge n_y<n_z) \wedge$ *long-ch-by-ord f X* $\wedge\ n_z <$ *card X* $\longrightarrow$ [[(*f* n_x) (*f* n_y) (*f* n_z)]]
using *assms long-ch-by-ord-def ordering-def*
by *metis*
hence [[(*f* n_x) (*f* n_y) (*f* n_z)]] $\vee$ [[(*f* n_x) (*f* n_z) (*f* n_y)]] $\vee$ [[(*f* n_z) (*f* n_x) (*f* n_y)]] $\vee$
[[(*f* n_z) (*f* n_y) (*f* n_x)]] $\vee$ [[(*f* n_y) (*f* n_z) (*f* n_x)]] $\vee$ [[(*f* n_y) (*f* n_x) (*f* n_z)]]
proof −
have *f1*: $\bigwedge n\ na\ nb.\ \neg\ n < na \vee \neg\ nb < n \vee \neg\ na <$ *card X* $\vee$ [[(*f nb*) (*f n*) (*f na*)]]
by (*metis* (*no-types*) *ordering-def* ‹*long-ch-by-ord f X*› *long-ch-by-ord-def*)
then have *f2*: $\neg\ n_z < n_y \vee \neg\ n_x < n_z \vee$ [[*x z y*]]
using *fx fy fz yy*
by *blast*
have $\neg\ n_x < n_y \vee \neg\ n_z < n_x \vee$ [[*z x y*]]
using *f1 fx fy fz yy* **by** *blast*
then show *?thesis*
using *f2 f1 fx fy fz less-or xx zz* **by** *auto*
qed
hence [[*x y z*]] $\vee$ [[*x z y*]] $\vee$ [[*z x y*]] $\vee$
[[*z y x*]] $\vee$ [[*y z x*]] $\vee$ [[*y x z*]]
using *fx fy fz assms semifin-chain-def long-ch-by-ord-def*
by *metis*
thus *?thesis*
using *abc-sym*
by *blast*
qed

lemma *long-chain-on-path*:
assumes *long-ch-by-ord f X* **and** *finite X*
shows $\exists\, P\in\mathcal{P}.\ X\subseteq P$
proof −
obtain *x y* **where** $x\in X$ **and** $y\in X$ **and** $y\neq x$
using *long-ch-by-ord-def assms*
by (*metis* (*mono-tags*, *hide-lams*))
obtain *z* **where** $z\in X$ **and** $x\neq z$ **and** $y\neq z$
using *long-ch-by-ord-def assms*
by *metis*
have [[*x y z*]] $\vee$ [[*x z y*]] $\vee$ [[*z x y*]]
using *all-aligned-on-long-chain assms*
using ‹$x \in X$› ‹$x \neq z$› ‹$y \in X$› ‹$y \neq x$› ‹$y \neq z$› ‹$z \in X$›

by *auto*
then have *path-exists*: $\exists P \in \mathcal{P}.\ path\ P\ x\ y$
using *all-aligned-on-long-chain abc-ex-path*
by (*metis* ‹$y \neq x$›)
obtain P **where** *path* $P\ x\ y$
using *path-exists*
by *blast*
have $X \subseteq P$
proof
fix e
assume $e \in X$
show $e \in P$
proof −
have $e=x \vee e=y \vee (e \neq x \wedge e \neq y)$ **by** *auto*
moreover {
assume $e \neq x \wedge e \neq y$
have $[[x\ y\ e]] \vee [[x\ e\ y]] \vee [[e\ x\ y]]$
using *all-aligned-on-long-chain all-aligned-on-long-chain assms* ‹$e \in X$› ‹$e \neq x \wedge e \neq y$› ‹$y \in X$› ‹$y \neq x$› ‹$x \in X$›
by *metis*
hence *?thesis*
using ‹*path* $P\ x\ y$› *abc-ex-path path-unique*
by *blast*
}
moreover {
assume $e=x$
have *?thesis*
by (*simp add*: ‹$e = x$› ‹*path* $P\ x\ y$›)
}
moreover {
assume $e=y$
have $e \in P$
by (*simp add*: ‹$e = y$› ‹*path* $P\ x\ y$›)
}
ultimately show *?thesis* **by** *blast*
qed
qed
thus *?thesis*
using ‹*path* $P\ x\ y$›
by *blast*
qed

Notice that this whole proof would be unnecessary if includig 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 *chain-on-path*:
assumes *ch-by-ord* $f\ X$

shows $\exists P \in \mathcal{P}.\ X \subseteq P$
using *assms ch-by-ord-def*
using *semifin-chain-on-path long-chain-on-path short-chain-on-path long-inf-chain-is-semifin*
by *meson*

30.1.3 More general results

lemma *ch-some-betw*: $[\![x \in X;\ y \in X;\ z \in X;\ x \neq y;\ x \neq z;\ y \neq z;\ ch\ X]\!]$
$\Longrightarrow [[x\ y\ z]] \vee [[y\ x\ z]] \vee [[y\ z\ x]]$
proof −
assume *asm*: $x \in X\ y \in X\ z \in X\ x \neq y\ x \neq z\ y \neq z\ ch\ X$
{
fix *f* **assume** *f-def*: *long-ch-by-ord f X*
assume *evts*: $x \in X\ y \in X\ z \in X\ x \neq y\ x \neq z\ y \neq z$
assume *ords*: $\neg\ [[x\ y\ z]]\ \neg\ [[y\ z\ x]]$
obtain *P* **where** $X \subseteq P\ P \in \mathcal{P}$
using *chain-on-path f-def ch-by-ord-def*
by *meson*
have $[[y\ x\ z]]$
proof −
have *f1*: $\forall A\ Aa\ a.\ \neg\ A \subseteq Aa \vee (a::'a) \notin A \vee a \in Aa$
by *blast*
have *f2*: $y \in P$
using ‹$X \subseteq P$› *evts*(*2*) **by** *blast*
have *f3*: $x \in P$
using *f1* **by** (*metis* ‹$X \subseteq P$› *evts*(*1*))
have $z \in P$
using ‹$X \subseteq P$› *evts*(*3*) **by** *blast*
then show *?thesis*
using *f3 f2* **by** (*metis some-betw-xor* ‹$P \in \mathcal{P}$› *abc-sym evts*(*4*,*5*,*6*) *ords*)
qed
}
thus *?thesis*
unfolding *ch-def long-ch-by-ord-def ch-by-ord-def ordering-def short-ch-def*
using *asm ch-by-ord-def ch-def short-ch-def*
by (*metis* ‹$\bigwedge f.\ [\![$*long-ch-by-ord f X*; $x \in X;\ y \in X;\ z \in X;\ x \neq y;\ x \neq z;\ y \neq z;$
$\neg\ [[x\ y\ z]];\ \neg\ [[y\ z\ x]]]\!] \Longrightarrow [[y\ x\ z]]$›)
qed

lemma *ch-all-betw-f*:
assumes $[f[x..yy..z]X]$ **and** $y \in X$ **and** $y \neq x$ **and** $y \neq z$
shows $[[x\ y\ z]]$
proof (*rule ccontr*)
assume *asm*: $\neg\ [[x\ y\ z]]$
obtain *Q* **where** $Q \in \mathcal{P}$ **and** $x \in Q \wedge y \in Q \wedge z \in Q$
using *chain-on-path assms ch-by-ord-def asm fin-ch-betw fin-long-chain-def*
by *auto*
hence $[[x\ y\ z]] \vee [[y\ x\ z]] \vee [[y\ z\ x]]$

using *some-betw assms*
by (*metis abc-sym fin-long-chain-def*)
hence $[[y\ x\ z]] \vee [[x\ z\ y]]$
using *asm abc-sym*
by *blast*
thus *False*
using *fin-long-chain-def long-ch-by-ord-def asm assms fin-ch-betw*
by (*metis* (*no-types*, *hide-lams*))
qed

lemma *get-fin-long-ch-bounds*:
assumes *long-ch-by-ord f X*
and *finite X*
shows $\exists x{\in}X.\ \exists y{\in}X.\ \exists z{\in}X.\ [f[x..y..z]X]$
proof −
obtain x **where** $x = f\ 0$ **by** *simp*
obtain z **where** $z = f\ (card\ X - 1)$ **by** *simp*
obtain y **where** *y-def*: $y{\neq}x \wedge y{\neq}z \wedge y{\in}X$
by (*metis assms*(*1*) *long-ch-by-ord-def*)
have $x{\in}X$
using *ordering-def* ‹$x = f\ 0$› *assms*(*1*) *long-ch-by-ord-def*
by (*metis card-gt-0-iff equals0D*)
have $z{\in}X$
using *ordering-def* ‹$z = f\ (card\ X - 1)$› *assms*(*1*) *long-ch-by-ord-def*
by (*metis card-gt-0-iff equals0D Suc-diff-1 lessI*)
obtain n **where** $n{<}card\ X$ **and** $f\ n = y$
using *ordering-def y-def long-ch-by-ord-def assms*
by *metis*
have $n{>}0$
using *y-def* ‹$f\ n = y$› ‹$x = f\ 0$›
using *neq0-conv* **by** *blast*
moreover have $n{<}card\ X - 1$
using *y-def* ‹$f\ n = y$› ‹$n < card\ X$› ‹$z = f\ (card\ X - 1)$› *assms*(*2*)
by (*metis card.remove card-Diff-singleton less-SucE*)
ultimately have $[f[x..y..z]X]$
using *long-ch-by-ord-def y-def* ‹$x = f\ 0$› ‹$z = f\ (card\ X - 1)$› *abc-abc-neq assms ordering-ord-ijk*
unfolding *fin-long-chain-def*
by (*metis* (*no-types*, *lifting*) *card-gt-0-iff diff-less equals0D zero-less-one*)
thus *?thesis*
using *points-in-chain*
by *blast*
qed

lemma *get-fin-long-ch-bounds2*:
assumes *long-ch-by-ord f X*
and *finite X*
obtains $x\ y\ z\ n_x\ n_y\ n_z$

where $x{\in}X \wedge y{\in}X \wedge z{\in}X \wedge [f[x..y..z]X] \wedge f\ n_x = x \wedge f\ n_y = y \wedge f\ n_z = z$
by (*meson assms*(*1*) *assms*(*2*) *fin-long-chain-def get-fin-long-ch-bounds index-middle-element*)

lemma *long-ch-card-ge3*:
assumes *ch-by-ord f X finite X*
shows *long-ch-by-ord f X* $\longleftrightarrow$ *card X* $\geq$ *3*
proof
assume *long-ch-by-ord f X*
then obtain *a b c* **where** $[f[a..b..c]X]$
using *get-fin-long-ch-bounds assms*(*2*) **by** *blast*
thus *3* $\leq$ *card X*
by (*metis* (*no-types*, *hide-lams*) *One-nat-def card-eq-0-iff diff-Suc-1 empty-iff fin-long-chain-def index-middle-element leI less-3-cases less-one*)
next
assume *3* $\leq$ *card X*
hence $\neg$*short-ch X*
using *assms*(*1*) *short-ch-card-2* **by** *auto*
thus *long-ch-by-ord f X*
using *assms*(*1*) *ch-by-ord-def* **by** *auto*
qed

lemma *chain-bounds-unique*:
assumes $[f[a..b..c]X]$ $[g[x..y..z]X]$
shows $(a{=}x \wedge c{=}z) \vee (a{=}z \wedge c{=}x)$
proof −
have $\forall p{\in}X.\ (a = p \vee p = c) \vee [[a\ p\ c]]$
using *assms*(*1*) *ch-all-betw-f* **by** *force*
then show *?thesis*
by (*metis* (*full-types*) *abc-abc-neq abc-bcd-abd abc-sym assms*(*1*,*2*) *ch-all-betw-f points-in-chain*)
qed

lemma *chain-bounds-unique2*:
assumes $[f[a..c]X]$ $[g[x..z]X]$ *card X* $\geq$ *3*
shows $(a{=}x \wedge c{=}z) \vee (a{=}z \wedge c{=}x)$
using *chain-bounds-unique*
by (*metis abc-ac-neq assms*(*1*,*2*) *ch-all-betw-f fin-chain-def points-in-chain short-ch-def*)

30.2 Chain Equivalences

30.2.1 Betweenness-chains and strong index-chains

lemma *equiv-chain-1a*:
assumes $[[..a..b..c..]X]$
shows $\exists f.$ *ch-by-ord f X* $\wedge a{\in}X \wedge b{\in}X \wedge c{\in}X \wedge a{\neq}b \wedge a{\neq}c \wedge b{\neq}c$
proof −
have *in-X*: $a{\in}X \wedge b{\in}X \wedge c{\in}X$
using *assms chain-with-def* **by** *auto*
have *all-neq*: $a{\neq}c \wedge a{\neq}b \wedge b{\neq}c$
using *abc-abc-neq assms chain-with-def* **by** *auto*

obtain *f* **where** *ordering f betw X*
using *assms chain-with-def* **by** *auto*
hence *long-ch-by-ord f X*
using *in-X all-neq long-ch-by-ord-def* **by** *blast*
hence *ch-by-ord f X*
by (*simp add*: *ch-by-ord-def*)
thus *?thesis*
using *all-neq in-X* **by** *blast*
qed

lemma *equiv-chain-1b*:
assumes $ch\text{-}by\text{-}ord\ f\ X \land a{\in}X \land b{\in}X \land c{\in}X \land a{\neq}b \land a{\neq}c \land b{\neq}c \land [[a\ b\ c]]$
shows $[[..a..b..c..]X]$
using *assms chain-with-def ch-by-ord-def*
by (*metis long-ch-by-ord-def short-ch-def*)

lemma *equiv-chain-1*:
$[[..a..b..c..]X] \longleftrightarrow (\exists f.\ ch\text{-}by\text{-}ord\ f\ X \land a{\in}X \land b{\in}X \land c{\in}X \land a{\neq}b \land a{\neq}c \land b{\neq}c \land [[a\ b\ c]])$
using *equiv-chain-1a equiv-chain-1b long-chain-betw*
by *meson*

lemma *index-order*:
assumes *chain-with x y z X*
and *ch-by-ord f X* **and** $f\ a = x$ **and** $f\ b = y$ **and** $f\ c = z$
and $finite\ X \longrightarrow a{<}card\ X$ **and** $finite\ X \longrightarrow b{<}card\ X$ **and** $finite\ X \longrightarrow c{<}card\ X$
shows $(a{<}b \land b{<}c) \lor (c{<}b \land b{<}a)$
proof (*rule ccontr*)
assume *a1*: $\neg (a < b \land b < c \lor c < b \land b < a)$
hence $(a{\geq}b \lor b{\geq}c) \land (c{\geq}b \lor b{\geq}a)$
by *auto*
have *all-neq*: $x{\neq}y \land x{\neq}z \land y{\neq}z$
using *assms*(*1*) *equiv-chain-1* **by** *blast*
hence *is-long*: *long-ch-by-ord f X*
by (*metis assms*(*1*) *assms*(*2*) *ch-by-ord-def equiv-chain-1 short-ch-def*)
have $a{\neq}b \land a{\neq}c \land b{\neq}c$
using *assms*(*3*) *assms*(*4*) *assms*(*5*) *all-neq* **by** *blast*
hence $(a{>}b \lor b{>}c) \land (c{>}b \lor b{>}a)$
using *a1 linorder-neqE-nat* **by** *blast*
hence $(a{>}b \land c{>}b) \lor (b{>}c \land b{>}a)$
using *not-less-iff-gr-or-eq* **by** *blast*
have $a{>}c \lor c{>}a$
using $\langle a \neq b \land a \neq c \land b \neq c\rangle$ **by** *auto*
hence $(a{>}c \land c{>}b) \lor (a{>}c \land b{>}a) \lor (a{>}b \land c{>}a) \lor (b{>}c \land c{>}a)$
using $\langle(b < a \lor c < b) \land (b < c \lor a < b)\rangle$ **by** *blast*

hence *o1*: $(b<c \wedge c<a) \vee (c<a \wedge a<b) \vee (b<a \wedge a<c) \vee (a<c \wedge c<b)$
  by *blast*
  have $(b<c \wedge c<a) \longrightarrow [[y\ z\ x]]$
    using *assms ordering-ord-ijk long-ch-by-ord-def is-long*
    by *metis*
  moreover have $(c<a \wedge a<b) \longrightarrow [[z\ x\ y]]$
    using *assms ordering-ord-ijk long-ch-by-ord-def is-long*
    by *metis*
  moreover have $(b<a \wedge a<c) \longrightarrow [[y\ x\ z]]$
    using *assms ordering-ord-ijk long-ch-by-ord-def is-long*
    by *metis*
  moreover have $(a<c \wedge c<b) \longrightarrow [[x\ z\ y]]$
    using *assms ordering-ord-ijk long-ch-by-ord-def is-long*
    by *metis*
  ultimately have $[[y\ z\ x]] \vee [[z\ x\ y]] \vee [[y\ x\ z]] \vee [[x\ z\ y]]$
    using *assms long-ch-by-ord-def is-long o1*
    by *metis*
  thus *False*
    by (*meson abc-only-cba assms*(*1*) *chain-with-def*)
qed

lemma *old-fin-chain-finite*:
  assumes *finite-chain-with3 x y z X*
  shows *finite X*
proof (*rule ccontr*)
  assume *infinite X*
  have $x \in X$
    using *assms finite-chain-with3-def chain-with-def* **by** *simp*
  have $y \in X$
    using *assms finite-chain-with3-def chain-with-def* **by** *simp*
  have $z \in X$
    using *assms finite-chain-with3-def chain-with-def* **by** *simp*
  obtain *f* **where** *ch-by-ord f X*
    using *assms equiv-chain-1 finite-chain-with3-def*
    by *auto*
  obtain *a* **where** $f\ a = x$
    using *equiv-chain-1 ordering-def* ‹*ch-by-ord f X*› *assms*
    by (*metis ch-by-ord-def finite-chain-with3-def long-ch-by-ord-def short-ch-def*)
  obtain *c* **where** $f\ c = z$ **and** $a \neq c$
    using *equiv-chain-1 ordering-def* ‹*ch-by-ord f X*› ‹$f\ a = x$› *assms*
    using *ch-by-ord-def finite-chain-with3-def long-ch-by-ord-def short-ch-def*
    by *metis*
  obtain *b* **where** $f\ b = y$ **and** $a \neq b$ **and** $b \neq c$
    using *equiv-chain-1 ordering-def* ‹*ch-by-ord f X*› ‹$f\ a = x$› ‹$f\ c = z$› *assms*
    using *ch-by-ord-def finite-chain-with3-def long-ch-by-ord-def short-ch-def*
    by *metis*
  obtain *n* **where** $a<n$ **and** $c<n$
    using ‹*ch-by-ord f X*› ‹$f\ a = x$› ‹$f\ c = z$› *assms equiv-chain-1* ‹*infinite X*›

using *ch-by-ord-def finite-chain-with3-def long-ch-by-ord-def short-ch-def*
by (*metis less-Suc-eq-le not-le not-less-iff-gr-or-eq*)
have [[*x y z*]]
using *assms chain-with-def finite-chain-with3-def* **by** *auto*
hence $(a<b \wedge b<c) \vee (c<b \wedge b<a)$
using ‹$f\ a = x$› ‹$f\ b = y$› ‹$f\ c = z$› ‹*ch-by-ord f X*› ‹$x \in X$› ‹$y \in X$› ‹$z \in X$› *index-order*
using ‹*infinite X*› *assms finite-chain-with3-def*
by *blast*
hence $(a<b \wedge b<c \wedge c<n) \vee (c<b \wedge b<a \wedge a<n)$
using ‹$a \neq c$› ‹$a \neq b$› ‹$b \neq c$› ‹$a<n$› ‹$c<n$› *less-linear*
by *blast*
hence *acn-can*: $(b<c \wedge c<n) \vee (b<a \wedge a<n)$
by *blast*
have $f\ n \in X$
by (*metis ordering-def* ‹*ch-by-ord f X*› ‹*infinite X*› *assms ch-by-ord-def equiv-chain-1 finite-chain-with3-def long-ch-by-ord-def short-ch-def*)
hence *outside*: [[*y z* (*f n*)]] ∨ [[(*f n*) *x y*]]
using *acn-can* ‹*ch-by-ord f X*› ‹$f\ a = x$› ‹$f\ c = z$› ‹*infinite X*› *assms equiv-chain-1 abc-sym*
using *ch-by-ord-def finite-chain-with3-def long-ch-by-ord-def ordering-ord-ijk short-ch-def*
by (*metis* ‹$f\ b = y$›)
thus *False*
using ‹$f\ n \in X$› *assms finite-chain-with3-def*
by *blast*
qed

lemma *index-from-with3*:
assumes *finite-chain-with3 a b c X*
shows $\exists f.\ (f\ 0 = a \vee f\ 0 = c) \wedge$ *ch-by-ord f X*
proof −
obtain *f* **where** *ch-by-ord f X*
using *assms equiv-chain-1 finite-chain-with3-def*
by *auto*
have *no-elt*: ¬(∃ *w*∈*X*. [[*w a b*]] ∨ [[*b c w*]])
using *assms finite-chain-with3-def*
by *blast*
obtain n_a n_b **where** $f\ n_a = a$ **and** $n_a <$ *card X*
and $f\ n_b = b$ **and** $n_b <$ *card X*
using *assms old-fin-chain-finite ch-by-ord-def ordering-def*
using ‹*ch-by-ord f X*› *equiv-chain-1 finite-chain-with3-def long-ch-by-ord-def short-ch-def*
by *metis*
obtain n_c **where** $f\ n_c = c$ **and** $n_c <$ *card X*
using *assms old-fin-chain-finite ch-by-ord-def ordering-def*
using ‹*ch-by-ord f X*› *equiv-chain-1 finite-chain-with3-def long-ch-by-ord-def short-ch-def*
by *metis*

have $a{\neq}b \wedge b{\neq}c \wedge a{\neq}c$
  using *assms equiv-chain-1 finite-chain-with3-def* **by** *auto*
have $a{\neq}b \longrightarrow n_a{\neq}n_b \wedge b{\neq}c \longrightarrow n_a{\neq}n_c \wedge a{\neq}c \longrightarrow n_b{\neq}n_c$
  using ‹$f\ n_a = a$› ‹$f\ n_b = b$› ‹$f\ n_c = c$› **by** *blast*
hence $n_a{\neq}n_b \wedge n_a{\neq}n_c \wedge n_b{\neq}n_c$
  using ‹$a \neq b \wedge b \neq c \wedge a \neq c$› ‹$f\ n_a = a$› ‹$f\ n_b = b$› ‹$f\ n_c = c$›
  by *auto*
have $n_a = 0 \vee n_c = 0$
  proof (*rule ccontr*)
    assume $\neg\ (n_a = 0 \vee n_c = 0)$
    hence *not-0*: $n_a \neq 0 \wedge n_c \neq 0$
      by *linarith*
    then obtain p **where** $f\ 0 = p$
      by *simp*
    hence $p{\in}X$
    using ‹*ch-by-ord f X*› ‹$n_a < card\ X$› *assms card-0-eq ch-by-ord-def zero-into-ordering*
    using *equiv-chain-1 finite-chain-with3-def inf.strict-coboundedI2 inf.strict-order-iff less-one long-ch-by-ord-def old-fin-chain-finite short-ch-def*
      by *metis*
    have $n_a{<}n_c \vee n_c{<}n_a$
      using ‹$n_a \neq n_b \wedge n_a \neq n_c \wedge n_b \neq n_c$› *less-linear* **by** *blast*
    {
      assume $n_a < n_c$
      hence $n_a < n_b$
        using *index-order* ‹*ch-by-ord f X*› ‹$f\ n_a = a$› ‹$f\ n_b = b$› ‹$f\ n_c = c$› ‹$n_c < card\ X$›
        using *finite-chain-with3-def assms*
        by *fastforce*
      have $0{<}n_a \wedge n_a{<}n_b$
        using *index-order* ‹$n_a < n_b$› *not-0*
        by *blast*
      hence $[[p\ a\ b]]$
        using ‹*ch-by-ord f X*› ‹$f\ 0{=}p$› ‹$f\ n_a{=}a$› ‹$f\ n_b{=}b$› ‹$n_b{<}card\ X$› *assms equiv-chain-1 short-ch-def*
        by (*metis ch-by-ord-def finite-chain-with3-def long-ch-by-ord-def ordering-ord-ijk*)
      hence *False*
        using *finite-chain-with3-def* ‹$p{\in}X$›
        by (*metis no-elt*)
    }
    moreover {
      assume $n_c < n_a$
      hence $n_c < n_b$
        using *index-order* ‹*ch-by-ord f X*› ‹$f\ n_a = a$› ‹$f\ n_b = b$› ‹$f\ n_c = c$› ‹$n_a < card\ X$›
        using *finite-chain-with3-def assms*
        by *fastforce*
      have $0{<}n_c \wedge n_c{<}n_b$
        using *index-order* ‹$n_c < n_b$› *not-0*

by *blast*
hence $[[p\ c\ b]]$
using ‹*ch-by-ord f X*› ‹$f\ 0{=}p$› ‹$f\ n_c{=}c$› ‹$f\ n_b{=}b$› ‹$n_b{<}card\ X$› *assms equiv-chain-1 short-ch-def*
using *ch-by-ord-def finite-chain-with3-def long-ch-by-ord-def ordering-ord-ijk*
by *metis*
hence $[[b\ c\ p]]$
by (*simp add*: *abc-sym*)
hence *False*
using *finite-chain-with3-def* ‹$p{\in}X$›
by (*metis no-elt*)
}
ultimately show *False*
using ‹$n_a < n_c \lor n_c < n_a$› **by** *blast*
qed
thus *?thesis*
using ‹*ch-by-ord f X*› ‹$f\ n_a = a$› ‹$f\ n_c = c$›
by *blast*
qed

lemma (**in** *MinkowskiSpacetime*) *with3-and-index-is-fin-chain*:
assumes $f\ 0 = a$ **and** *ch-by-ord f X* **and** *finite-chain-with3 a b c X*
shows $[f[a..b..c]X]$
proof −
have *finite X*
using *ordering-def assms old-fin-chain-finite*
by *auto*
moreover have *long-ch-by-ord f X*
using *assms*(*2*) *assms*(*3*) *ch-by-ord-def equiv-chain-1 finite-chain-with3-def short-ch-def*
by *metis*
moreover have $a{\neq}b \land a{\neq}c \land b{\neq}c \land f\ 0 = a \land b{\in}X$
using *assms*(*1*) *assms*(*3*) *equiv-chain-1 finite-chain-with3-def*
by *auto*
moreover have $f\ (card\ X - 1) = c$
proof −
obtain n **where** $f\ n = c$ **and** $n < card\ X$
using *ordering-def equiv-chain-1 finite-chain-with3-def long-ch-by-ord-def*
by (*metis assms*(*3*) *calculation*(*1*,*2*))
{
assume $n < card\ X - 1$
then obtain m **where** $n{<}m$ **and** $m{<}card\ X$ **by** *simp*
hence $[[a\ c\ (f\ m)]] \land (f\ m){\in}X$
proof −
have *f1*: *TernaryOrdering.ordering f betw X*
using ‹*long-ch-by-ord f X*› *long-ch-by-ord-def* **by** *blast*
have *f2*: $\forall f\ A\ p\ na.\ ((p\ (f\ na{::}'a)\ (f\ n)\ (f\ m) \lor \neg\ m < card\ A) \lor \neg$ *ordering f p A*)

$\vee \neg$ *na* $<$ *n*
by (*metis ordering-def* ‹*n* $<$ *m*›)
have *f m* $\in$ *X*
using *f1* **by** (*simp add*: *ordering-def* ‹*m* $<$ *card X*›)
then show *?thesis*
using *f2 f1* ‹*a*$\neq$*b* $\wedge$ *a*$\neq$*c* $\wedge$ *b*$\neq$*c* $\wedge$ *f 0* $=$ *a* $\wedge$ *b*$\in$*X*› ‹*f n* $=$ *c*› ‹*m* $<$ *card X*›
using *gr-implies-not0 linorder-neqE-nat*
by (*metis* (*no-types*))
qed
hence [[*b c* (*f m*)]] **using** *abc-acd-bcd*
by (*meson assms*(*3*) *chain-with-def finite-chain-with3-def*)
hence *False*
using *assms*(*3*) ‹[[*a c* (*f m*)]] $\wedge$ *f m* $\in$ *X*›
by (*metis finite-chain-with3-def*)
}
hence *n* $=$ *card X* $-$ *1*
using ‹*n* $<$ *card X*› **by** *fastforce*
thus *?thesis*
using ‹*f n* $=$ *c*› **by** *blast*
qed
ultimately show *?thesis*
by (*simp add*: *fin-long-chain-def*)
qed

lemma (**in** *MinkowskiSpacetime*) *g-from-with3*:
assumes *finite-chain-with3 a b c X*
obtains *g* **where** [*g*[*a*..*b*..*c*]*X*] $\vee$ [*g*[*c*..*b*..*a*]*X*]
proof $-$
have *old-chain-sym*: *finite-chain-with3 c b a X*
by (*metis abc-sym assms chain-with-def finite-chain-with3-def*)
obtain *f* **where** *f-def*: (*f 0* $=$ *a* $\vee$ *f 0* $=$ *c*) $\wedge$ *ch-by-ord f X*
using *index-from-with3 assms*
by *blast*
hence *f 0* $=$ *a* $\longrightarrow$ [*f*[*a*..*b*..*c*]*X*]
using *with3-and-index-is-fin-chain f-def assms*
by *simp*
moreover have *f 0* $=$ *c* $\longrightarrow$ [*f*[*c*..*b*..*a*]*X*]
using *with3-and-index-is-fin-chain f-def assms old-chain-sym*
by *simp*
ultimately show *?thesis*
using *f-def that*
by *auto*
qed

lemma (**in** *MinkowskiSpacetime*) *equiv-chain-2a*:
assumes *finite-chain-with3 a b c X*

obtains *f* **where** $[f[a..b..c]X]$
proof –
obtain *g* **where** $[g[a..b..c]X] \vee [g[c..b..a]X]$
using *assms g-from-with3* **by** *blast*
thus *?thesis*
proof
assume $[g[a..b..c]X]$
show *?thesis*
using ‹$[g[a\ ..\ b\ ..\ c]X]$› *that*
by *blast*
next
assume $[g[c..b..a]X]$
show *?thesis*
using ‹$[g[c\ ..\ b\ ..\ a]X]$› *chain-sym that*
by *blast*
qed
qed

lemma *equiv-chain-2b*:
assumes $[f[a..b..c]X]$
shows *finite-chain-with3 a b c X*
proof –
have *aligned*: $[[a\ b\ c]]$
using *assms fin-ch-betw*
by *auto*
hence *some-chain*: $[[..a..b..c..]X]$
using *assms ch-by-ord-def equiv-chain-1b fin-long-chain-def points-in-chain*
by *metis*
have $\neg(\exists\, w{\in}X.\ [[w\ a\ b]] \vee [[b\ c\ w]])$
proof (*safe*)
fix *w* **assume** $w{\in}X$
{
assume *case1*: $[[w\ a\ b]]$
then obtain *n* **where** $f\ n = w$ **and** $n{<}card\ X$
using ‹$w{\in}X$› *abc-bcd-abd abc-only-cba aligned assms fin-ch-betw fin-long-chain-def*
by (*metis* (*no-types*, *hide-lams*))
have $f\ 0 = a$
using *assms fin-long-chain-def*
by *blast*
hence $n < 0$
proof –
have *f1*: $f\ (card\ X - 1) = c$
by (*meson MinkowskiBetweenness.fin-long-chain-def MinkowskiBetweenness-axioms assms*)
have $\neg\ [[a\ w\ c]]$
by (*meson abc-bcd-abd abc-only-cba assms case1 fin-ch-betw*)
thus *?thesis*
using *f1 fin-long-chain-def* ‹$w \in X$› *abc-only-cba assms case1 fin-ch-betw*

by (*metis* (*no-types*))
qed
thus *False*
by *simp*
}
moreover {
assume *case2*: $[[b\ c\ w]]$
then obtain n **where** $f\ n = w$ **and** $n<card\ X$
using ‹$w{\in}X$› *ordering-def abc-bcd-abd abc-only-cba aligned assms fin-ch-betw*
using *fin-long-chain-def long-ch-by-ord-def*
by *metis*
have $f\ (card\ X - 1) = c$
using *assms fin-long-chain-def*
by *blast*
have $\neg\ [[a\ w\ c]]$
using *abc-bcd-abd abc-only-cba assms case2 fin-ch-betw abc-bcd-acd*
by *meson*
hence $n > card\ X - 1$
using ‹$\neg\ [[a\ w\ c]]$› ‹$w \in X$› *abc-only-cba assms case2 fin-ch-betw*
unfolding *fin-long-chain-def*
by (*metis* (*no-types*))
thus *False*
using ‹$n < card\ X$›
by *linarith*
}
qed
thus *?thesis*
by (*simp add*: *finite-chain-with3-def some-chain*)
qed

lemma (**in** *MinkowskiSpacetime*) *equiv-chain-2*:
$\exists f.\ [f[a..b..c]X] \longleftrightarrow [[a..b..c]X]$
using *equiv-chain-2a equiv-chain-2b*
by *meson*

end

31 Results for segments, rays and chains

context *MinkowskiChain* **begin**

lemma *inside-not-bound*:
assumes $[f[a..b..c]X]$
and $j<card\ X$
shows $j>0 \Longrightarrow f\ j \neq a$ $j<card\ X - 1 \Longrightarrow f\ j \neq c$
proof –
have *bound-indices*: $f\ 0 = a \wedge f\ (card\ X - 1) = c$
using *assms*(*1*) *fin-long-chain-def* **by** *auto*

show *f j* ≠ *a* **if** *j>0*
proof (*cases*)
assume *f j* = *c*
then have [[(*f 0*) (*f j*) *b*]] ∨ [[(*f 0*) *b* (*f j*)]]
using *assms*(*1*) *fin-ch-betw fin-long-chain-def*
by *metis*
thus *?thesis* **using** *abc-abc-neq bound-indices* **by** *blast*
next
assume *f j* ≠ *c*
then have [[(*f 0*) (*f j*) *c*]] ∨ [[(*f 0*) *c* (*f j*)]]
using *assms fin-ch-betw*
unfolding *fin-long-chain-def long-ch-by-ord-def ordering-def*
by (*metis abc-abc-neq assms that ch-all-betw-f nat-neq-iff*)
thus *?thesis*
using *abc-abc-neq bound-indices* **by** *blast*
qed
show *f j* ≠ *c* **if** *j<card X* − *1*
proof (*cases*)
assume *f j* = *a*
show *?thesis*
using ‹*f j* = *a*› *assms*(*1*) *fin-long-chain-def*
by *blast*
next
assume *f j* ≠ *a*
have *0* < *card X*
using *assms*(*2*) **by** *linarith*
hence [[*a* (*f j*) (*f* (*card X* − *1*))]] ∨ [[(*f j*) *a* (*f* (*card X* − *1*))]]
using *assms fin-ch-betw fin-long-chain-def order-finite-chain*
by (*metis* ‹*f j* ≠ *a*› *diff-less le-numeral-extra*(*1*−*3*) *neq0-conv that*)
thus *f j* ≠ *c*
using *abc-abc-neq bound-indices* **by** *auto*
qed
qed

lemma *some-betw2*:
assumes [*f*[*a*..*b*..*c*]*X*]
and *j<card X j>0 f j* ≠ *b*
shows [[*a b* (*f j*)]] ∨ [[*a* (*f j*) *b*]]
proof −
obtain *ab* **where** *ab-def*: *path ab a b X*⊆*ab*
by (*metis fin-long-chain-def long-chain-on-path assms*(*1*) *points-in-chain subsetD*)
have *bound-indices*: *f 0* = *a* ∧ *f* (*card X* − *1*) = *c*
using *assms*(*1*) *fin-long-chain-def* **by** *auto*
have *f j* ≠ *a*
using *inside-not-bound*(*1*) *assms*(*1*) *assms*(*2*) *assms*(*3*)
by *blast*
have ¬[[(*f j*) *a b*]]

using *abc-bcd-abd abc-only-cba assms*(*1*,*2*) *fin-ch-betw fin-long-chain-def*
by (*metis ordering-def ch-all-betw-f long-ch-by-ord-def*)
thus $[[a\ b\ (f\ j)]] \vee [[a\ (f\ j)\ b]]$
using *some-betw* [**where** *Q*=*ab* **and** *a*=*a* **and** *b*=*b* **and** *c*=*f j*]
using *ab-def assms*(*4*) ‹*f j* ≠ *a*›
by (*metis ordering-def abc-sym assms*(*1*,*2*) *fin-long-chain-def long-ch-by-ord-def subsetD*)
qed

lemma *i-le-j-events-neq1*:
assumes [*f*[*a*..*b*..*c*]*X*]
and *i*<*j j*<*card X f j* ≠ *b*
shows *f i* ≠ *f j*
proof −
have *in-X*: *f i* ∈ *X* ∧ *f j* ∈ *X*
by (*metis ordering-def assms*(*1*,*2*,*3*) *fin-long-chain-def less-trans long-ch-by-ord-def*)
have *bound-indices*: *f 0* = *a* ∧ *f* (*card X* − *1*) = *c*
using *assms*(*1*) *fin-long-chain-def* **by** *auto*
obtain *ab* **where** *ab-def*: *path ab a b X*⊆*ab*
by (*metis fin-long-chain-def long-chain-on-path assms*(*1*) *points-in-chain subsetD*)
show *?thesis*
proof (*cases*)
assume *f i* = *a*
hence $[[a\ (f\ j)\ b]] \vee [[a\ b\ (f\ j)]]$
using *some-betw2 assms* **by** *blast*
thus *?thesis*
using ‹*f i* = *a*› *abc-abc-neq* **by** *blast*
next assume *f i* ≠ *a*
hence $[[a\ (f\ i)\ (f\ j)]]$
using *assms*(*1*,*2*,*3*) *ch-equiv fin-long-chain-def order-finite-chain2*
by (*metis gr-implies-not-zero le-numeral-extra*(*3*) *less-linear*)
thus *?thesis*
using *abc-abc-neq* **by** *blast*
qed
qed

lemma *i-le-j-events-neq*:
assumes [*f*[*a*..*b*..*c*]*X*]
and *i*<*j j*<*card X*
shows *f i* ≠ *f j*
proof −
have *in-X*: *f i* ∈ *X* ∧ *f j* ∈ *X*
by (*metis ordering-def assms*(*1*,*2*,*3*) *fin-long-chain-def less-trans long-ch-by-ord-def*)
have *bound-indices*: *f 0* = *a* ∧ *f* (*card X* − *1*) = *c*
using *assms*(*1*) *fin-long-chain-def* **by** *auto*
obtain *ab* **where** *ab-def*: *path ab a b X*⊆*ab*
by (*metis fin-long-chain-def long-chain-on-path assms*(*1*) *points-in-chain subsetD*)

show *?thesis*
proof (*cases*)
assume $f\ i = a$
show *?thesis*
proof (*cases*)
assume $(f\ j) = b$
thus *?thesis*
by (*simp add*: ‹$(f\ i) = a$› *ab-def*(*1*))
next assume $(f\ j) \neq b$
have $[[a\ (f\ j)\ b]] \vee [[a\ b\ (f\ j)]]$
using *some-betw2 assms* ‹$(f\ j) \neq b$› **by** *blast*
thus *?thesis*
using ‹$(f\ i) = a$› *abc-abc-neq* **by** *blast*
qed
next assume $(f\ i) \neq a$
hence $[[a\ (f\ i)\ (f\ j)]]$
using *assms*(*1*,*2*,*3*) *ch-equiv fin-long-chain-def order-finite-chain2*
by (*metis gr-implies-not-zero le-numeral-extra*(*3*) *less-linear*)
thus *?thesis*
using *abc-abc-neq* **by** *blast*
qed
qed

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

lemma *index-order2*:
assumes $[f[x..y..z]X]$ **and** $f\ a = x$ **and** $f\ b = y$ **and** $f\ c = z$
and *finite* $X \longrightarrow a < card\ X$ **and** *finite* $X \longrightarrow b < card\ X$ **and** *finite* $X \longrightarrow c < card\ X$
shows $(a<b \wedge b<c) \vee (c<b \wedge b<a)$
using *index-order* [**where** $x=x$ **and** $y=y$ **and** $z=z$ **and** $a=a$ **and** $b=b$ **and** $c=c$ **and** $f=f$ **and** $X=X$]
by (*metis assms ch-by-ord-def equiv-chain-2b fin-long-chain-def finite-chain-with3-def*)

lemma *index-order3*:
assumes $[[x\ y\ z]]$ **and** $f\ a = x$ **and** $f\ b = y$ **and** $f\ c = z$ **and** *long-ch-by-ord* $f\ X$
and *finite* $X \longrightarrow a < card\ X$ **and** *finite* $X \longrightarrow b < card\ X$ **and** *finite* $X \longrightarrow c < card\ X$
shows $(a<b \wedge b<c) \vee (c<b \wedge b<a)$
using *index-order2* [**where** $x=x$ **and** $y=y$ **and** $z=z$ **and** $a=a$ **and** $b=b$ **and** $c=c$ **and** $f=f$ **and** $X=X$]
using *assms long-ch-by-ord-def ordering-ord-ijk*
by (*smt abc-abc-neq abc-only-cba*(*1−3*) *linorder-neqE-nat*)

```
end

context MinkowskiSpacetime begin

lemma bound-on-path:
  assumes Q∈𝒫 [f[(f 0)..]X] 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 ordering-def inf-chain-is-long is-bound-f-def 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** $x \in segment\ a\ b$
  hence $[[a\ x\ b]]$ **by** (*simp add*: *seg-betw*)
  have $\neg[[a\ b\ x]]$ **by** (*meson* ‹$[[a\ x\ b]]$› *abc-only-cba*)
  have $\neg[[b\ x\ c]]$
    using ‹$\neg\ [[a\ b\ x]]$› *abd-bcd-abc assms* **by** *blast*
  thus $x \notin segment\ b\ c$
    by (*simp add*: *seg-betw*)
qed
thus *?thesis* **by** *blast*
qed

end

32 3.6 Order on a path - Theorems 10 and 11

context *MinkowskiSpacetime* **begin**

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

lemma (**in** *MinkowskiBetweenness*) *two-event-chain*:
  assumes *finiteX*: *finite X*
    and *path-Q*: $Q \in \mathcal{P}$
    and *events-X*: $X \subseteq Q$
    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*: $\neg(\exists\, c\in\{a,b\}.\ c\neq a \wedge c\neq b)$
    by *blast*
  have $a\neq b \wedge a\in Q\ \&\ b\in Q$
    using *X-is card-X events-X* **by** *force*
  hence *short-ch* $\{a,b\}$
    using *path-Q short-ch-def no-c* **by** *blast*
  thus *?thesis*
    by (*simp add*: *X-is ch-by-ord-def ch-def*)
qed

lemma (**in** *MinkowskiBetweenness*) *three-event-chain*:
  assumes *finiteX*: *finite X*
    and *path-Q*: $Q \in \mathcal{P}$
    and *events-X*: $X \subseteq Q$
    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*: $a \neq b \wedge a \neq c \wedge b \neq c$
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*: $a \in Q \wedge b \in Q \wedge c \in Q$
using *X-is events-X* **by** *blast*
hence $[[a\ b\ c]] \vee [[b\ c\ a]] \vee [[c\ a\ b]]$
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

lemma *chain-append-at-left-edge*:
assumes *long-ch-Y*: $[f[a_1..a..a_n]Y]$
and *bY*: $[[b\ a_1\ a_n]]$
fixes *g* **defines** *g-def*: $g \equiv (\lambda j{::}nat.\ if\ j \geq 1\ then\ f\ (j-1)\ else\ b)$
shows $[g[b\ ..\ a_1\ ..\ a_n](insert\ b\ Y)]$
proof −
let *?X = insert b Y*
have $b \notin Y$
by (*metis abc-ac-neq abc-only-cba*(*1*) *bY ch-all-betw-f long-ch-Y*)
have *bound-indices*: $f\ 0 = a_1 \wedge f\ (card\ Y - 1) = a_n$
using *long-ch-Y* **by** (*simp add*: *fin-long-chain-def*)
have *fin-Y*: $card\ Y \geq 3$
using *fin-long-chain-def long-ch-Y numeral-2-eq-2*
by (*metis ch-by-ord-def long-ch-card-ge3*)
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 fin-long-chain-def long-ch-Y*
by *auto*

hence $[[b\ a_1\ (f\ 1)]]$
using *bY abd-bcd-abc* **by** *blast*
have *ordering2 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 fin-long-chain-def long-ch-Y points-in-chain*
by (*metis* ‹$g\ n = f\ (n - 1)$›)
next
case *False*
hence $n < card\ Y$
using *points-in-chain* ‹$finite\ ?X \longrightarrow n < card\ ?X$› ‹$g\ n = f\ (n - 1)$› ‹$g\ n \notin Y$› ‹$b \notin Y$›
by (*metis card.insert fin-long-chain-def 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* **unfolding** *fin-long-chain-def long-ch-by-ord-def ordering-def*
by (*meson less-trans num-ord*)
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 $n\ n'\ n''$ **assume** (*finite* $?X \longrightarrow n'' < card\ ?X$) $Suc\ n = n' \wedge Suc\ n' = n''$
hence $[[(g\ n)\ (g\ n')\ (g\ n'')]]$
using ‹$b \notin Y$› ‹$[[b\ a_1\ (f\ 1)]]$› *g-def long-ch-Y ordering-ord-ijk*
by (*smt* (*verit*, *ccfv-threshold*) *fin-long-chain-def long-ch-by-ord-def*
One-nat-def card.insert diff-Suc-Suc diff-diff-cancel diff-is-0-eq
finite-insert nat-less-le not-less not-less-eq-eq)
} moreover {
fix x **assume** $x \in ?X$ $x = b$
have (*finite* $?X \longrightarrow 0 < card\ ?X$) $\wedge g\ 0 = x$
by (*simp add*: ‹$b \notin Y$› ‹$x = b$› *g-def*)
} moreover {
fix x **assume** $x \in ?X$ $x \neq b$
hence $\exists n.$ (*finite* $?X \longrightarrow n < card\ ?X$) $\wedge g\ n = x$
proof −
obtain n **where** $f\ n = x$ $n < card\ Y$
using ‹$x \in ?X$› ‹$x \neq b$›
by (*metis ordering-def fin-long-chain-def insert-iff long-ch-Y long-ch-by-ord-def*)
have (*finite* $?X \longrightarrow n{+}1 < card\ ?X$) $g(n{+}1) = x$
apply (*simp add*: ‹$b \notin Y$› ‹$n < card\ Y$›)
by (*simp add*: ‹$f\ n = x$› *g-def*)
thus *?thesis* **by** *auto*
qed
}
ultimately show *?thesis*
unfolding *ordering2-def*
by *smt*
qed

hence *long-ch-by-ord2 g ?X*
unfolding *long-ch-by-ord2-def*
using *points-in-chain fin-long-chain-def* ‹$b \notin Y$›
by (*metis abc-abc-neq bY insert-iff long-ch-Y points-in-chain*)
hence *long-ch-by-ord g ?X*
using *ch-equiv fin-Y*
by (*meson fin-long-chain-def finite-insert long-ch-Y*)
thus *?thesis*
unfolding *fin-long-chain-def*
using *bound-indices* ‹$b \notin Y$› *g-def num-ord points-in-chain long-ch-Y fin-long-chain-def*
by (*metis card.insert diff-Suc-1 finite-insert insert-iff less-trans nat-less-le*)
qed

lemma *chain-append-at-right-edge*:
assumes *long-ch-Y*: $[f[a_1..a..a_n]Y]$
and *Yb*: $[[a_1\ a_n\ b]]$
fixes g **defines** *g-def*: $g \equiv (\lambda j{::}nat.\ if\ j \leq (card\ Y - 1)\ then\ f\ j\ else\ b)$
shows $[g[a_1\ ..\ a_n\ ..\ b](insert\ b\ Y)]$
proof −
let $?X = insert\ b\ Y$
have $b \notin Y$
by (*metis Yb abc-abc-neq abc-only-cba*(*2*) *ch-all-betw-f long-ch-Y*)
have *fin-X*: *finite ?X*
using *fin-long-chain-def long-ch-Y* **by** *blast*
have *fin-Y*: $card\ Y \geq 3$
by (*meson ch-by-ord-def fin-long-chain-def long-ch-Y long-ch-card-ge3*)
have $a_1 \in Y \wedge a_n \in Y \wedge a \in Y$
using *long-ch-Y points-in-chain* **by** *blast*
have $a_1 \neq a \wedge a \neq a_n \wedge a_1 \neq a_n$
using *fin-long-chain-def long-ch-Y* **by** *auto*
have $Suc\ (card\ Y) = card\ ?X$
using ‹$b \notin Y$› *fin-X fin-long-chain-def long-ch-Y* **by** *auto*
obtain *f2* **where** *f2-def*: $[f2[a_n..a..a_1]Y]\ f2=(\lambda n.\ f\ (card\ Y - 1 - n))$
using *chain-sym long-ch-Y* **by** *blast*
obtain *g2* **where** *g2-def*: $g2 = (\lambda j{::}nat.\ if\ j \geq 1\ then\ f2\ (j-1)\ else\ b)$
by *simp*
have $[[b\ a_n\ a_1]]$
using *abc-sym Yb* **by** *blast*
hence *g2-ord-X*: $[g2[b\ ..\ a_n\ ..\ a_1]?X]$
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*: $[g1[a_1..a_n..b]?X]\ g1=(\lambda n.\ g2\ (card\ ?X - 1 -$
$n))$
using *chain-sym* **by** *blast*
have *sYX*: $(card\ Y) = (card\ ?X) - 1$
using *assms*(*2*,*3*) *fin-long-chain-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 ≤ 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› fin-long-chain-def long-ch-Y sYX
           by (metis Suc-1 Suc-diff-1 Suc-diff-le card-gt-0-iff diff-Suc-eq-diff-pred
diff-commute
            diff-diff-cancel equals0D less-one nat.simps(3) not-less)
    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[a₁..a..aₙ]Y]
      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 −
  have k∈S using k-def Max-in S-def
    by (metis finite-Collect-conjI finite-Collect-less-nat)
  show k' ∈ S
  proof (rule ccontr)
    assume ¬k'∈S
    hence [[a₁ b (f k')]]
      using order-finite-chain S-def abc-acd-bcd abc-bcd-acd abc-sym long-ch-Y
      by (smt fin-long-chain-def ‹0 < k'› ‹k ∈ S› ‹k' < k› le-numeral-extra(3))
```

less-trans 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*
have $k' < card\ Y$
using *S-def* ‹$k \in S$› ‹$k' < k$› *less-trans* **by** *blast*
thus *False*
using *abc-bcd-abd order-finite-chain S-def abc-only-cba*(*2*) *long-ch-Y*
‹$0 < k'$› ‹$[[(f\ k)\ b\ (f\ k')]]$› ‹$k \in S$› ‹$k' < k$›
unfolding *fin-long-chain-def*
by (*metis* (*mono-tags*, *lifting*) *le-numeral-extra*(*3*) *mem-Collect-eq*)
qed
qed

lemma *smallest-k-ex*:
assumes *long-ch-Y*: $[f[a_1..a..a_n]Y]$
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 *fin-long-chain-def long-ch-Y* **by** *auto*
have *fin-Y*: *finite Y*
using *fin-long-chain-def long-ch-Y* **by** *blast*
have *card-Y*: $card\ Y \geq 3$
using *fin-long-chain-def long-ch-Y points-in-chain*
by (*metis* (*no-types*, *lifting*) *One-nat-def antisym card2-either-elt1-or-elt2 diff-is-0-eq*′
not-less-eq-eq numeral-2-eq-2 numeral-3-eq-3)

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 (*rule conjI4*)

```
        show (0::nat)<1 by simp
        show 1 < card Y
          using Yb abc-ac-neq bound-indices not-le by fastforce

        show ¬ (∃k'::nat. k' < 1 ∧ [[a₁ b (f k')]])
          using abc-abc-neq bound-indices
          by blast
        show [[a₁ b (f 1)]]
        proof -
          have f 1 ∈ Y
             by (metis ordering-def diff-0-eq-0 fin-long-chain-def less-one long-ch-Y
long-ch-by-ord-def nat-neq-iff)

          hence [[a₁ (f 1) aₙ]]
            using bound-indices long-ch-Y
            unfolding fin-long-chain-def long-ch-by-ord-def ordering-def
               by (smt One-nat-def card.remove card-Diff1-less card-Diff-singleton
diff-is-0-eq'
               le-eq-less-or-eq less-SucE neq0-conv zero-less-diff zero-less-one)
          hence [[a₁ b (f 1)]] ∨ [[a₁ (f 1) b]] ∨ [[b a₁ (f 1)]]
            using abc-ex-path-unique some-betw abc-sym
            by (smt Y-def Yb ‹f 1 ∈ Y› abc-abc-neq cross-once-notin)
          thus [[a₁ b (f 1)]]

          proof -
            have ∀n. ¬ ([[a₁ (f n) b]] ∧ n < card Y)
              using S-def ‹S = {}›
              by blast
            then have [[a₁ b (f 1)]] ∨ ¬ [[aₙ (f 1) b]] ∧ ¬ [[a₁ (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 ‹[[a₁ b (f 1)]] ∨ [[a₁ (f 1) b]] ∨ [[b a₁ (f 1)]]› ‹[[a₁ (f 1) aₙ]]›)
          qed
        qed
      qed
    qed
  next assume ¬S={}
    obtain k where k = Max S
      by simp
    hence  k ∈ S using Max-in
      by (simp add: ‹S ≠ {}› ‹finite S›)
    have k≥1
    proof (rule ccontr)
      assume ¬ 1 ≤ k
      hence k=0 by simp
      have [[a₁ (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 $0<?k \wedge [[a_1\ b\ (f\ ?k)]] \wedge ?k < card\ Y \wedge \neg\ (\exists k'::nat.\ k' < ?k \wedge [[a_1\ b\ (f\ k')]])$
proof (*rule conjI4*)
show $(0::nat)<?k$ **by** *simp*
show $?k < card\ Y$
by (*metis* (*no-types*, *lifting*) *S-def Yb* ‹$k \in S$› *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 $[[a_1\ b\ (f\ ?k)]]$
proof −
have $f\ ?k \in Y$
using ‹$k + 1 < card\ Y$›
by (*metis ordering-def fin-long-chain-def long-ch-Y long-ch-by-ord-def*)
have $[[a_1\ (f\ ?k)\ a_n]] \vee f\ ?k = a_n$
using *bound-indices long-ch-Y* ‹$k + 1 < card\ Y$›
unfolding *fin-long-chain-def long-ch-by-ord-def ordering-def*
by (*metis* (*no-types*, *lifting*) *Suc-lessI add.commute add-gr-0 card-Diff1-less card-Diff-singleton less-diff-conv plus-1-eq-Suc zero-less-one*)
thus $[[a_1\ b\ (f\ ?k)]]$
proof (*rule disjE*)
assume $[[a_1\ (f\ ?k)\ a_n]]$
hence $f\ ?k \neq a_n$
by (*simp add*: *abc-abc-neq*)
hence $[[a_1\ b\ (f\ ?k)]] \vee [[a_1\ (f\ ?k)\ b]] \vee [[b\ a_1\ (f\ ?k)]]$
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 $\neg\ [[a_1\ (f\ ?k)\ b]]$
proof
assume $[[a_1\ (f\ ?k)\ b]]$
hence $?k \in S$
using *S-def* ‹$[[a_1\ (f\ ?k)\ b]]$› ‹$k + 1 < card\ Y$› **by** *blast*
hence $?k \leq k$
by (*simp add*: ‹*finite S*› ‹$k = Max\ S$›)
thus *False*
by *linarith*
qed
moreover have $\neg\ [[b\ a_1\ (f\ ?k)]]$
using *Yb* ‹$[[a_1\ (f\ ?k)\ a_n]]$› *abc-only-cba*
by *blast*

ultimately show $[[a_1\ b\ (f\ ?k)]]$
by *blast*
next assume $f\ ?k = a_n$
show *?thesis*
using *Yb* ‹$f\ (k + 1) = a_n$› **by** *blast*
qed
qed
show $\neg(\exists k'::nat.\ k' < k + 1 \wedge [[a_1\ b\ (f\ k')]])$
proof
assume $\exists k'::nat.\ k' < k + 1 \wedge [[a_1\ b\ (f\ k')]]$
then obtain k' **where** *k′-def*: $k'>0$ $k' < k + 1$ $[[a_1\ b\ (f\ k')]]$
using *abc-ac-neq bound-indices neq0-conv*
by *blast*
hence $k'<k$
using *S-def* ‹$k \in S$› *abc-only-cba*(*2*) *less-SucE* **by** *fastforce*
hence $k'\in S$
using *S-is-dense long-ch-Y S-def* ‹$\neg S=\{\}$› ‹$k = Max\ S$› ‹$k'>0$›
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*: $[f[a_1..a..a_n]Y]$
and *Y-def*: $b\notin Y$
and *Yb*: $[[a_1\ b\ a_n]]$
shows $\exists k.\ [[(f\ k)\ b\ a_n]] \wedge k < card\ Y - 1 \wedge \neg(\exists k'<card\ Y.\ k'>k \wedge [[(f\ k')\ b\ a_n]])$
proof −

have *bound-indices*: $f\ 0 = a_1 \wedge f\ (card\ Y - 1) = a_n$
using *fin-long-chain-def long-ch-Y* **by** *auto*
have *fin-Y*: *finite Y*
using *fin-long-chain-def long-ch-Y* **by** *blast*
have *card-Y*: $card\ Y \geq 3$
using *fin-long-chain-def long-ch-Y points-in-chain*
by (*metis* (*no-types*, *lifting*) *One-nat-def antisym card2-either-elt1-or-elt2 diff-is-0-eq′ not-less-eq-eq numeral-2-eq-2 numeral-3-eq-3*)

let $?S = \{k::nat.\ [[a_n\ (f\ k)\ b]] \wedge k < card\ Y\}$
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 (rule conjI3)
        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 ?n ∈ Y
          by (metis ordering-def diff-less fin-long-chain-def gr-implies-not0 long-ch-Y
              long-ch-by-ord-def neq0-conv not-less-eq numeral-2-eq-2)
          hence [[a_1 (f ?n) a_n]]
            using bound-indices long-ch-Y
            unfolding fin-long-chain-def long-ch-by-ord-def ordering-def
            using card-Y by force
          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
            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 $k \in S$ **using** *Max-in*
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 (*rule conjI3*)
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 long-ch-by-ord-def ordering-def*
by (*metis diff-less fin-long-chain-def less-trans neq0-conv zero-less-one*)
have $[[a_1\ (f\ ?k)\ a_n]] \vee f\ ?k = a_1$
using *bound-indices long-ch-Y* ‹$k - 1 < card\ Y - 1$›
unfolding *fin-long-chain-def long-ch-by-ord-def ordering-def*
by (*smt S-def* ‹$k \in S$› *add-diff-inverse-nat card-Diff1-less card-Diff-singleton*
less-numeral-extra(*4*) *less-trans mem-Collect-eq nat-add-left-cancel-less*
neq0-conv zero-less-diff)
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 ‹$f\ (k - 1) \neq a_1$› *fin-long-chain-def long-ch-Y*
by *auto*

qed
moreover have $\neg$ $[[b\ a_n\ (f\ ?k)]]$
using *Yb* $\langle[[a_1\ (f\ ?k)\ a_n]]\rangle$ *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* $\langle f\ (k - 1) = a_1\rangle$ **by** *blast*
qed
qed
show $\neg(\exists\, k'{<}card\ Y.\ k{-}1 < k' \wedge [[(f\, k')\ b\ a_n]])$
proof
assume $\exists\, k'{<}card\ Y.\ k{-}1 < k' \wedge [[(f\, k')\ b\ a_n]]$
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* $\langle k \in S\rangle$ *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*)
hence $k'{\in}S$
using *S-is-dense long-ch-Y S-def* $\langle\neg S{=}\{\}\rangle$ $\langle k = Min\ S\rangle$ $\langle k'{<}card\ Y - 1\rangle$
by (*smt Yb* $\langle k \in S\rangle$ *abc-acd-bcd abc-only-cba*(*3*) *card-Diff1-less card-Diff-singleton fin-long-chain-def k′-def*(*3*) *less-le mem-Collect-eq neq0-conv order-finite-chain*)
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*: $[f[a_0..a..a_n]\,Y]$
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 *fin-long-chain-def long-ch-Y* **by** *auto*

have *finite Y*
using *fin-long-chain-def long-ch-Y* **by** *blast*
obtain P **where** *P-def*: $P{\in}\mathcal{P}$ $Y{\subseteq}P$
using *chain-on-path long-ch-Y*
unfolding *fin-long-chain-def ch-by-ord-def*
by *blast*
hence $x{\in}P$
using *betw-b-in-path x-def*(*2*) *long-ch-Y points-in-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 *long-ch-Y long-ch-by-ord-def fin-long-chain-def*
by (*metis ordering-def*)
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 long-ch-Y long-ch-by-ord-def fin-long-chain-def ordering-def*
by *metis*
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*
‹$\bigwedge thesis.\ (\bigwedge n_b.\ [\![\neg\ (\exists k < card\ Y.\ [[(f\ k)\ x\ a_n]] \wedge n_b < k); [[(f\ n_b)\ x\ a_n]]; n_b < card\ Y - 1]\!]$
$\Longrightarrow thesis) \Longrightarrow thesis$› *abc-abd-acdadc abc-ac-neq abc-only-cba diff-less fin-long-chain-def le-antisym le-trans less-imp-le-nat less-numeral-extra*(*1*)
linorder-neqE-nat long-ch-Y nb-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 ‹$n_b \neq n_c - 1$› *fin-long-chain-def long-ch-Y nc-def*(*3*) *order-finite-chain*
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*
by (*metis* ‹$[[(f\ n_b)\ (f\ (n_c - 1))\ (f\ n_c)]]$› *abc-abc-neq abd-bcd-abc b-def*(*1*,*2*) *ch-all-betw-f*
long-ch-Y nb-def(*2*) *nc-def*(*2*))
have $x \neq f(n_c-1)$
using *x-def*(*1*) *nc-def*(*3*) *long-ch-Y*
unfolding *fin-long-chain-def long-ch-by-ord-def ordering-def*
by (*metis less-imp-diff-less*)
hence $[[a_0\ (f\ (n_c-1))\ x]]$
using *some-betw P-def*(*1*,*2*) *abc-abc-neq abc-acd-bcd abc-bcd-acd abc-sym b-def*(*1*,*2*)
c-def(*1*,*2*) *ch-all-betw-f in-mono long-ch-Y nc-def*(*2*) *betw-b-in-path*
by (*smt* ‹$[[(f\ n_b)\ (f\ (n_c-1))\ (f\ n_c)]]$› ‹$\neg\ [[a_0\ x\ (f\ (n_c-1))]]$› ‹$x \in P$› ‹$f(n_c-1) \neq a_0 \wedge a_0 \neq x$›)
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

lemma *chain-append-inside*:
assumes *long-ch-Y*: $[f[a_1..a..a_n]Y]$
and *Y-def*: $b \notin Y$
and *Yb*: $[[a_1\ b\ a_n]]$
and *k-def*: $[[a_1\ b\ (f\ k)]]$ $k < card\ Y$ $\neg(\exists k'.\ (0::nat)<k' \wedge k'<k \wedge [[a_1\ b\ (f\ k')]])$
fixes *g*
defines *g-def*: $g \equiv (\lambda j::nat.\ if\ (j \le k-1)\ then\ f\ j\ else\ (if\ (j=k)\ then\ b\ else\ f\ (j-1)))$
shows $[g[a_1\ ..\ b\ ..\ a_n]insert\ b\ Y]$
proof −
let $?X = insert\ b\ Y$
have *fin-X*: *finite ?X*
by (*meson fin-long-chain-def finite.insertI long-ch-Y*)
have *bound-indices*: $f\ 0 = a_1 \wedge f\ (card\ Y - 1) = a_n$
using *fin-long-chain-def long-ch-Y*
by *auto*
have *fin-Y*: *finite Y*
using *fin-long-chain-def long-ch-Y* **by** *blast*
have *f-def*: *long-ch-by-ord f Y*
using *fin-long-chain-def long-ch-Y* **by** *blast*
have $\langle a_1 \neq a_n \wedge a_1 \neq b \wedge b \neq a_n \rangle$
using *Yb abc-abc-neq* **by** *blast*
have $k \neq 0$
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 $\langle [[a_1\ b\ (f\ k)]] \rangle$ $\langle k = 1 \rangle$ *bound-indices* **by** *auto*
next assume $k \neq 1$ **show** $[[(f\ (k-1))\ b\ (f\ k)]]$
proof −
have $[[a_1\ (f\ (k-1))\ (f\ k)]]$ **using** *bound-indices*
using $\langle k < card\ Y \rangle$ $\langle k \neq 0 \rangle$ $\langle k \neq 1 \rangle$ *long-ch-Y fin-Y order-finite-chain*
unfolding *fin-long-chain-def*
by *auto*

have *ch-with-b*: $ch\ \{a_1, (f\ (k-1)), b, (f\ k)\}$ **using** *chain4*
using *k-def*(*1*) *abc-ex-path-unique between-chain cross-once-notin*
by (*smt* $\langle [[a_1\ (f\ (k - 1))\ (f\ k)]] \rangle$ *abc-abc-neq insert-absorb2*)

have $f\ (k-1) \neq b \wedge (f\ k) \neq (f\ (k-1)) \wedge b \neq (f\ k)$
using *abc-abc-neq f-def k-def(2) Y-def*
by (*metis ordering-def* ‹[[a_1 $(f\ (k - 1))$ $(f\ k)$]]› *less-imp-diff-less long-ch-by-ord-def*)
hence *some-ord-bk*: [[$(f\ (k-1))$ b $(f\ k)$]] $\vee$ [[b $(f\ (k-1))$ $(f\ k)$]] $\vee$ [[$(f\ (k-1))$ $(f\ k)$ b]]
using *chain-on-path ch-with-b some-betw Y-def* **unfolding** *ch-def*
by (*metis abc-sym insert-subset*)
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 $(f\ k)$]]› ‹[[a_1 $(f\ (k - 1))$ $(f\ k)$]]›)
qed
qed
qed

let *?case1* $\vee$ *?case2* $= k-2 \geq 0 \vee k+1 \leq$ *card* $Y -1$

have *b-right*: [[$(f\ (k-2))$ $(f\ (k-1))$ b]] **if** $k \geq 2$
proof −
have $k-1 < (k{::}nat)$
using ‹$k \neq 0$› *diff-less zero-less-one* **by** *blast*
hence $k-2 < k-1$
using ‹$2 \leq k$› **by** *linarith*
have [[$(f\ (k-2))$ $(f\ (k-1))$ $(f\ k)$]]
using *f-def k-def(2)* ‹$k-2 < k-1$› ‹$k-1 < k$› **unfolding** *long-ch-by-ord-def ordering-def*
by *blast*
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** $k+1 \leq$ *card* $Y -1$
proof −
have [[$(f\ (k-1))$ $(f\ k)$ $(f\ (k+1))$]]
using ‹$k \neq 0$› *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 *ordering2 g betw ?X*
proof −
have $\forall\, n.$ (*finite ?X* $\longrightarrow n <$ *card ?X*) $\longrightarrow g\ n \in$ *?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) long-ch-by-ord-def 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
            by (metis ordering-def f-def long-ch-by-ord-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 long-ch-by-ord-def 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* $\longrightarrow$ *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* $\longrightarrow$ *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* $\longrightarrow$ *n″* < *card ?X*) ∧ *Suc n* = *n′* ∧ *Suc*
n′ = *n″*
$\longrightarrow$ [[(*g n*) (*g* (*Suc n*)) (*g* (*Suc* (*Suc n*)))]]
proof (*clarify*)
fix *n n′ n″* **assume** *a*: (*finite ?X* $\longrightarrow$ (*Suc* (*Suc n*)) < *card ?X*)

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 f-def g-def*
k-def(*2*)

by (*metis* (*no-types*, *lifting*) *add-diff-inverse-nat lessI less-Suc-eq-le less-imp-le-nat less-le-trans less-one 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* $\langle Suc\ n = k\rangle$ $\langle n \le k - 1\rangle$ *g-def*
by *auto*
next show $n \le k{-}1$ **using** $\langle n \le k - 1\rangle$ **by** *blast*
qed
next assume $n{\ge}k{+}1$ **show** *?thesis*
proof −
have $g\ n = f\ (n{-}1)$
using $\langle k + 1 \le n\rangle$ *less-imp-diff-less g-def*
by *auto*
moreover have $g\ (Suc\ n) = f\ (n)$
using $\langle k + 1 \le n\rangle$ *g-def* **by** *auto*
moreover have $g\ (Suc\ (Suc\ n)) = f\ (Suc\ n)$
using $\langle k + 1 \le n\rangle$ *g-def* **by** *auto*
moreover have $n{-}1{<}n \land n{<}Suc\ n$
using $\langle k + 1 \le n\rangle$ **by** *auto*
moreover have $finite\ Y \longrightarrow Suc\ n < card\ Y$
using *Y-def a* **by** *auto*
ultimately show *?thesis*
using *f-def* **unfolding** *long-ch-by-ord-def ordering-def*
by *auto*
qed
next assume $n{=}k$
show *?thesis*
using $\langle k \ne 0\rangle$ $\langle n = k\rangle$ *b-left g-def Y-def*(*1*) *a assms*(*3*) *fin-Y*
by *auto*
qed
qed
ultimately show *ordering2 g betw ?X*
unfolding *ordering2-def*
by *presburger*
qed
hence *long-ch-by-ord2 g ?X*
using *Y-def f-def long-ch-by-ord2-def long-ch-by-ord-def*
by *auto*
thus $[g[a_1..b..a_n]?X]$
unfolding *fin-long-chain-def*
using *ch-equiv fin-X* $\langle a_1 \ne a_n \land a_1 \ne b \land b \ne a_n\rangle$ *bound-indices k-def*(*2*) *Y-def g-def*
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 $card\ X = 2 \mid card\ X = 3 \mid card\ X \geq 4$
    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*: *long-ch-by-ord f Y*
using *ch-long-if-card-ge3* ‹$4 \leq card\ Y$› **by** *fastforce*
then obtain $a_1\ a\ a_n$ **where** *long-ch-Y*: $[f[a_1..a..a_n]Y]$
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*: *fin-long-chain-def*)
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-chain* **by** *blast*
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-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 $[?g[b..a_1..a_n]X]$
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 *ch-by-ord-def ch-def fin-long-chain-def* **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 $[?g[a_1..a_n..b]X]$
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*
    unfolding *ch-def ch-by-ord-def* **using** *fin-long-chain-def* **by** *auto*
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 $\langle[[a_1\ b\ a_n]]\rangle$ $\langle b \notin Y\rangle$ *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 $[g[a_1..b..a_n]X]$
    using *chain-append-inside* [*of f* a_1 *a* a_n *Y b k*] *IH.prems*(*4*) *X-eq*
    $\langle[[a_1\ b\ a_n]]\rangle$ $\langle b \notin Y\rangle$ *k-def long-ch-Y*
    by *auto*
  thus *ch X*
    using *ch-by-ord-def ch-def fin-long-chain-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** $[f[a..b]X]$
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*: $[f[a..b]X]$ $a{\in}X \wedge b{\in}X$
    using *assms finX ch-X ch-some-betw get-fin-long-ch-bounds ch-long-if-card-ge3*
    by (*metis ch-by-ord-def ch-def fin-chain-def short-ch-def*)
  thus *?thesis*
    using *that* **by** *auto*
qed

32.2 Theorem 11 page 27

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*: $[f[a..b]Q]$
and *S-def*: $S = \{segment\ a\ b\}$
and *P1-def*: $P1 = prolongation\ b\ a$
and *P2-def*: $P2 = prolongation\ a\ b$
shows $P = ((\bigcup S) \cup P1 \cup P2 \cup Q) \wedge$
$card\ S = (N-1) \wedge (\forall x \in S.\ is\text{-}segment\ x) \wedge$
$P1 \cap P2 = \{\} \wedge (\forall x \in S.\ (x \cap P1 = \{\} \wedge x \cap P2 = \{\} \wedge (\forall y \in S.\ x \neq y \longrightarrow$
$x \cap y = \{\})))$
proof −
have $a \in Q \wedge b \in Q \wedge a \neq b$
by (*metis f-def fin-chain-def fin-long-chain-def points-in-chain*)
hence $Q=\{a,b\}$
using *assms*(*3*,*5*)
by (*smt card-2-iff insert-absorb insert-commute insert-iff singleton-insert-inj-eq*)
have $a \in P \wedge b \in P$
using ‹$Q=\{a,b\}$› *assms*(*4*) **by** *auto*
have $a \neq b$ **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 $P = ((\bigcup \{s\}) \cup P1 \cup P2 \cup Q) \wedge$
$card\ \{s\} = (N-1) \wedge (\forall x \in \{s\}.\ is\text{-}segment\ x) \wedge$
$P1 \cap P2 = \{\} \wedge (\forall x \in \{s\}.\ (x \cap P1 = \{\} \wedge x \cap P2 = \{\} \wedge (\forall y \in \{s\}.\ x \neq y \longrightarrow$
$x \cap y = \{\})))$
proof (*rule conjI*)
{ fix *x* **assume** $x \in P$
have $[[a\ x\ b]] \vee [[b\ a\ x]] \vee [[a\ b\ x]] \vee x=a \vee x=b$
using ‹$a \in P \wedge b \in P$› *some-betw path-P* ‹$a \neq b$›
by (*meson* ‹$x \in P$› *abc-sym*)
then have $x \in s \vee x \in P1 \vee x \in P2 \vee x=a \vee x=b$
using *pro-betw seg-betw P1-def P2-def s-def* ‹$Q = \{a,\ b\}$›
by *auto*
hence $x \in (\bigcup \{s\}) \cup P1 \cup P2 \cup Q$
using ‹$Q = \{a,\ b\}$› **by** *auto*
} moreover {
fix *x* **assume** $x \in (\bigcup \{s\}) \cup P1 \cup P2 \cup Q$
hence $x \in s \vee x \in P1 \vee x \in P2 \vee x=a \vee x=b$
using ‹$Q = \{a,\ b\}$› **by** *blast*
hence $[[a\ x\ b]] \vee [[b\ a\ x]] \vee [[a\ b\ x]] \vee x=a \vee x=b$
using *s-def P1-def P2-def*
unfolding *segment-def prolongation-def*
by *auto*
hence $x \in P$
using ‹$a \in P \wedge b \in P$› ‹$a \neq b$› *betw-b-in-path betw-c-in-path path-P*
by *blast*
}
ultimately show *union-P*: $P = ((\bigcup \{s\}) \cup P1 \cup P2 \cup Q)$

```
    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[a..b..c]Q]
  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-chain by blast
  hence ?N ≥ 3
   by (meson ch-by-ord-def f-def fin-long-chain-def long-ch-card-ge3)
  have bound-indices: f 0 = a ∧ f (card Q − 1) = c
   using f-def fin-long-chain-def 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 ch-all-betw-f f-def by blast
      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** $p \in s$ $s \in S$
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*: $(f\ y) \in Q \wedge f\ (y+1) \in Q$
using *f-def* **unfolding** *fin-long-chain-def long-ch-by-ord-def ordering-def*
by (*meson add-lessD1*)
have $[[a\ (f\ y)\ c]] \vee y=0$
using ‹$y < ?N - 1$› *assms*(*2*) *f-def fin-long-chain-def order-finite-chain* **by** *auto*
moreover have $[[a\ (f\ (y+1))\ c]] \vee y = ?N-2$
using ‹$y + 1 < card\ Q$› *assms*(*2*) *f-def fin-long-chain-def order-finite-chain*
by (*smt One-nat-def Suc-diff-1 Suc-eq-plus1 diff-Suc-eq-diff-pred gr-implies-not0 lessI less-Suc-eq-le linorder-neqE-nat not-le numeral-2-eq-2*)
ultimately consider $y=0 | y=?N-2 | ([[a\ (f\ y)\ c]] \wedge [[a\ (f\ (y+1))\ c]])$
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 ‹$p \in s$› ‹$s = segment\ (f\ y)\ (f\ (y + 1))$› *seg-betw*
by *auto*
moreover have $[[a\ (f(y+1))\ c]]$
using ‹$[[a\ (f(y+1))\ c]] \vee y = ?N - 2$› ‹$y = 0$› ‹$?N \geq 3$›
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* ‹$?N \geq 3$› *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 ‹$p \in s$› ‹$s = segment\ (f\ y)\ (f\ (y + 1))$› *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[a..b..c]Q]
    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 fin-long-chain-def by auto
 have points-neq: a≠b ∧ b≠c ∧ a≠c
  using f-def fin-long-chain-def by auto


 { 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 ch-all-betw-f f-def)
  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 ∈ interval a c
      using ‹[[a x c]]› interval-def seg-betw by auto
```

```
    hence x∈Q ∨ x∈⋃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 ∈ (⋃S) ∪ P1 ∪ P2 ∪ Q by blast
} moreover {
  fix x assume x ∈ (⋃S) ∪ P1 ∪ P2 ∪ Q
  hence (∃ s∈S. x∈s) ∨ x∈P1 ∨ x∈P2 ∨ x∈Q
    by blast
  hence x∈⋃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∈⋃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∈⋃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∉⋃S
    hence [[b a x]] ∨ [[b c x]] ∨ x∈Q
      using ‹x ∈ ⋃ 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 $P = ((\bigcup S) \cup P1 \cup P2 \cup Q)$
by *blast*
qed

lemma *inseg-axc*:
assumes *path-P*: $P \in \mathcal{P}$
and *Q-def*: *finite* $(Q::'a\ set)$ *card* $Q = N$ $Q \subseteq P$ $N \geq 3$
and *f-def*: $a \in Q \wedge b \in Q \wedge c \in Q$ $[f[a..b..c]Q]$
and *S-def*: $S = \{s.\ \exists i<(N-1).\ s = segment\ (f\ i)\ (f\ (i+1))\}$
and *x-def*: $x \in s$ $s \in S$
shows $[[a\ x\ c]]$
proof −
have *inseg-neq-ac*: $x \neq a \wedge x \neq c$ **if** $x \in s$ $s \in S$ **for** *x s*
proof
show $x \neq a$
proof (*rule notI*)
assume $x=a$
obtain *n* **where** *s-def*: $s = segment\ (f\ n)\ (f\ (n+1))$ $n<N-1$
using *S-def* ‹$s \in S$› **by** *blast*
have $f\ n \in Q$
using *f-def* ‹$n < N - 1$› *fin-long-chain-def long-ch-by-ord-def ordering-def*
by (*metis assms*(*3*) *diff-diff-cancel less-imp-diff-less less-irrefl-nat not-less*)
hence $[[a\ (f\ n)\ c]]$
using *f-def fin-long-chain-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 ch-all-betw-f inside-not-bound*(*2*) *less-diff-conv*)
moreover have $[[(f(n))\ x\ (f(n+1))]]$
using ‹$x \in s$› *seg-betw s-def*(*1*) **by** *simp*
ultimately show *False*
using ‹$x=a$› *abc-only-cba*(*1*) *assms*(*3*) *f-def fin-long-chain-def s-def*(*2*)
order-finite-chain
by (*metis le-numeral-extra*(*3*) *less-add-one less-diff-conv neq0-conv*)
qed

show $x \neq c$
proof (*rule notI*)
assume $x=c$
obtain *n* **where** *s-def*: $s = segment\ (f\ n)\ (f\ (n+1))$ $n<N-1$
using *S-def* ‹$s \in S$› **by** *blast*
hence $n+1<N$ **by** *simp*
have $[[(f(n))\ x\ (f(n+1))]]$
using ‹$x \in s$› *seg-betw s-def*(*1*) **by** *simp*
have $f\ (n) \in Q$
using *f-def* ‹$n+1 < N$› *fin-long-chain-def long-ch-by-ord-def ordering-def*
by (*metis add-lessD1 assms*(*3*))
have $f\ (n+1) \in Q$
using *f-def* ‹$n+1 < N$› *fin-long-chain-def long-ch-by-ord-def 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 fin-long-chain-def assms(3) order-finite-chain seg-betw that(1)
          abc-abc-neq abc-only-cba ch-all-betw-f
        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) fin-long-chain-def order-finite-chain
        by (metis ‹f n ∈ Q› abc-bcd-acd abc-only-cba(1,2) ch-all-betw-f)
    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[a..b..c]Q]
      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)
  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 $s \cap P1 = \{\} \wedge s \cap P2 = \{\} \wedge (\forall y{\in}S.\ s \neq y \longrightarrow s \cap y = \{\})$
proof (*rule conjI3*, *rule-tac*[*3*] *ballI*, *rule-tac*[*3*] *impI*)
show $s{\cap}P1{=}\{\}$
proof (*safe*)
fix *x* **assume** $x{\in}s$ $x{\in}P1$
hence [[*a x c*]]
using *inseg-axc* ‹$s \in S$› *assms* **by** *blast*
thus $x{\in}\{\}$
by (*metis P1-def* ‹$x \in P1$› *abc-bcd-abd abc-only-cba*(*1*) *f-def*(*2*) *fin-ch-betw pro-betw*)
qed
show $s{\cap}P2{=}\{\}$
proof (*safe*)
fix *x* **assume** $x{\in}s$ $x{\in}P2$
hence [[*a x c*]]
using *inseg-axc* ‹$s \in S$› *assms* **by** *blast*
thus $x{\in}\{\}$
by (*metis P2-def* ‹$x \in P2$› *abc-bcd-acd abc-only-cba*(*2*) *f-def*(*2*) *fin-ch-betw pro-betw*)
qed
fix *r* **assume** $r{\in}S$ $s{\neq}r$
show $s{\cap}r{=}\{\}$
proof (*safe*)
fix *y* **assume** $y \in r$ $y \in s$
obtain *n m* **where** *rs-def*: $r = segment\ (f\ n)\ (f(n{+}1))$ $s = segment\ (f\ m)\ (f(m{+}1))$
$n{\neq}m$ $n{<}N{-}1$ $m{<}N{-}1$
using *S-def* ‹$r \in S$› ‹$s \neq r$› ‹$s \in S$› **by** *blast*
have *y-betw*: [[$(f\ n)\ y\ (f(n{+}1))$]] $\wedge$ [[$(f\ m)\ y\ (f(m{+}1))$]]
using *seg-betw* ‹$y{\in}r$› ‹$y{\in}s$› *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*) *f-def fin-long-chain-def order-finite-chain rs-def*(*5*) **by** *auto*
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 *f-def assms*(*3*) *rs-def*(*5*)
unfolding *fin-long-chain-def long-ch-by-ord-def ordering-def*
by (*metis add-lessD1 less-add-one less-diff-conv*)
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) f-def fin-long-chain-def order-finite-chain
rs-def(4) by auto
          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 f-def assms(3) rs-def(4)
            unfolding fin-long-chain-def long-ch-by-ord-def ordering-def
            by (metis add-lessD1 less-add-one less-diff-conv)
          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[a..b..c]Q]
      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 −
  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={})))
  proof (rule conjI3)
    show P = ((⋃S) ∪ P1 ∪ P2 ∪ Q)
      using path-is-union assms
      by blast
    show ∀x∈S. is-segment x
    proof
      fix s assume s∈S
      thus is-segment s using S-def by auto
```

qed
show $P1 \cap P2=\{\} \wedge (\forall x{\in}S.\ x \cap P1 = \{\} \wedge x \cap P2 = \{\} \wedge (\forall y{\in}S.\ x \neq y \longrightarrow x \cap y = \{\}))$
using *assms disjoint-segmentation*
[**where** $P=P$ **and** $Q=Q$ **and** $N=N$ **and** $a=a$ **and** $b=b$ **and** $c=c$ **and** $f=f$ **and** $S=S$]
by *presburger*
qed
then show *?thesis* **by** *auto*
qed

We define 'disjoint' to be the same as in HOL-Library.DisjointSets. This saves importing a lot of baggage we don't need. The two lemmas below are just for safety.

abbreviation *disjoint*
where *disjoint* $A \equiv (\forall a{\in}A.\ \forall b{\in}A.\ a \neq b \longrightarrow a \cap b = \{\})$

lemma
fixes S:: $('a\ set)\ set$ **and** $P1$:: $'a\ set$ **and** $P2$:: $'a\ set$
assumes $\forall x{\in}S.\ (x{\cap}P1=\{\} \wedge x{\cap}P2=\{\} \wedge (\forall y{\in}S.\ x{\neq}y \longrightarrow x{\cap}y=\{\}))$ $P1{\cap}P2=\{\}$
shows *disjoint* $(S \cup \{P1,P2\})$
proof (*rule ballI*)
let $?U = S \cup \{P1,P2\}$
fix a **assume** $a{\in}?U$
then consider $(aS)\ a{\in}S \mid (a1)\ a{=}P1 \mid (a2)\ a{=}P2$
by *fastforce*
thus $\forall b{\in}?U.\ a \neq b \longrightarrow a \cap b = \{\}$
proof *cases*
case *aS*
{ fix b **assume** $b{\in}?U$ $a{\neq}b$
then consider $b{\in}S \mid b{=}P1 \mid b{=}P2$
by *fastforce*
hence $a{\cap}b=\{\}$
apply *cases*
apply (*simp add*: ‹$a \in S$› ‹$a \neq b$› *assms*)
apply (*meson* ‹$a \in S$› *assms*)
by (*simp add*: ‹$a \in S$› *assms*)
}
thus *?thesis*
by *meson*
next
case *a1*
{ fix b **assume** $b{\in}?U$ $a{\neq}b$
then consider $b{\in}S \mid b{=}P2$
using *a1* **by** *fastforce*
hence $a{\cap}b=\{\}$
apply *cases*
apply (*metis a1 assms(1) inf-commute*)
by (*simp add*: *a1 assms(2)*)

```
    }
    thus ?thesis
      by meson
  next
    case a2
    { fix b assume b∈?U a≠b
      then consider b∈S|b=P1
        using a2 by fastforce
      hence a∩b={}
        apply cases
        apply (metis a2 assms(1) inf-commute)
        by (simp add: a2 assms(2) inf-commute)
    }
    thus ?thesis
      by meson
  qed
qed
lemma
  fixes S:: ('a set) set and P1:: 'a set and P2:: 'a set
  assumes disjoint (S∪{P1,P2}) P1∉S P2∉S P1≠P2
  shows ∀x∈S. (x∩P1={} ∧ x∩P2={} ∧ (∀y∈S. x≠y ⟶ x∩y={})) P1∩P2={}
proof (rule ballI)
  show P1∩P2={}
    using assms(1,4) by simp
  fix x assume x∈S
  show x∩P1={} ∧ x∩P2={} ∧ (∀y∈S. x≠y ⟶ x∩y={})
  proof (rule conjI, rule-tac[2] conjI, rule-tac[3] ballI, rule-tac[3] impI)
    show x∩P1={}
      using ‹x ∈ S› assms(1,2) by fastforce
    show x∩P2={}
      using ‹x ∈ S› assms(1,3) by fastforce
    fix y assume y∈S x≠y
    thus x∩y={}
      by (simp add: ‹x ∈ S› assms(1))
  qed
qed
```

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∈𝒫
    and Q-def: Q⊆P
    and f-def: [f[a..b]Q]
  fixes P1 defines P1-def: P1 ≡ prolongation b a
  fixes P2 defines P2-def: P2 ≡ prolongation a b
  fixes S defines S-def: S ≡ if card Q=2 then {segment a b}
```

else {*segment* (*f i*) (*f* (*i*+*1*)) | *i*. *i*<*card Q*−*1*}
  shows *P* = ((⋃*S*) ∪ *P1* ∪ *P2* ∪ *Q*) (∀ *x*∈*S*. *is-segment x*)
    disjoint (*S*∪{*P1*,*P2*}) *P1*≠*P2* *P1*∉*S* *P2*∉*S*
proof −
  have *card-Q*: *card Q* ≥ *2*
    using *fin-chain-card-geq-2 f-def* **by** *blast*
  have *finite Q*
    by (*metis card.infinite card-Q rel-simps*(*28*))

  have *ch-Q*: *ch Q*
    using *Q-def card-Q path-P path-finsubset-chain* [**where** *X*=*Q* **and** *Q*=*P*]
    by *blast*
  have *f-def-2*: *a*∈*Q* ∧ *b*∈*Q*
    using *f-def points-in-chain fin-chain-def* **by** *auto*
  have *a*≠*b*
    using *f-def fin-chain-def fin-long-chain-def* **by** *auto*
  {
    assume *card Q* = *2*
    hence *S*={*segment a b*}
      by (*simp add*: *S-def*)
    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 *assms ch-Q* ‹*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*[*a*..*c*..*b*]*Q*]
      using *assms*(*3*,*5*) ‹*a*≠*b*›
      by (*metis f-def fin-chain-def short-ch-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 ch-Q* ‹*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-chain* ‹*finite Q*› *S-def2* **by** *presburger*+
  }
  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*

$P = ((\bigcup S) \cup P1 \cup P2 \cup Q)$ $(\forall x \in S.\ is\text{-}segment\ x)$
apply (*simp add*: *Int-commute*)
apply (*metis P2-def Un-iff old-thesis*(*1*,*3*) ‹$a \neq b$› *disjoint-iff f-def-2 path-P pro-betw prolong-betw2*)
apply (*metis P1-def Un-iff old-thesis*(*1*,*4*) ‹$a \neq b$› *disjoint-iff f-def-2 path-P pro-betw prolong-betw3*)
apply (*metis P2-def Un-iff old-thesis*(*1*,*4*) ‹$a \neq b$› *disjoint-iff f-def-2 path-P pro-betw prolong-betw*)
using *old-thesis*(*1*,*2*) **by** *linarith+*
qed

theorem *segmentation*:
assumes *path-P*: $P \in \mathcal{P}$
and *Q-def*: *card* $Q \geq 2$ $Q \subseteq P$
shows $\exists S\ P1\ P2.\ P = ((\bigcup S) \cup P1 \cup P2 \cup Q) \wedge$
disjoint $(S \cup \{P1, P2\}) \wedge P1 \neq P2 \wedge P1 \notin S \wedge P2 \notin S \wedge$
$(\forall x \in S.\ is\text{-}segment\ x) \wedge is\text{-}prolongation\ P1 \wedge is\text{-}prolongation\ P2$
proof −
let $?N = card\ Q$
obtain *f a b* **where** *f-def*: $[f[a..b]Q]$
using *path-finsubset-chain2*[*OF path-P Q-def*(*2*,*1*)]
by *metis*
let $?S = if\ ?N=2\ then\ \{segment\ a\ b\}\ else\ \{segment\ (f\ i)\ (f\ (i+1))\ |\ i.\ i<card\ Q-1\}$
let $?P1 = prolongation\ b\ a$
let $?P2 = prolongation\ a\ b$
have *from-seg*: $P = ((\bigcup ?S) \cup ?P1 \cup ?P2 \cup Q)$ $(\forall x \in ?S.\ is\text{-}segment\ x)$
disjoint $(?S \cup \{?P1, ?P2\})$ $?P1 \neq ?P2$ $?P1 \notin ?S$ $?P2 \notin ?S$
using *show-segmentation*[*OF path-P Q-def*(*2*) ‹$[f[a..b]Q]$›]
by *force+*
thus *?thesis*
by *blast*
qed

end

33 Chains are unique up to reversal

lemma (**in** *MinkowskiSpacetime*) *chain-remove-at-right-edge*:
assumes $[f[a..c]X]$ $f\ (card\ X - 2) = p$ $3 \leq card\ X$ $X = insert\ c\ Y$ $c \notin Y$
shows $[f[a..p]Y]$
proof −

have *lch-X*: *long-ch-by-ord f X*
using *assms*(*1*,*3*) *fin-chain-def fin-long-chain-def ch-by-ord-def short-ch-card-2*
by *fastforce*

have *p∈X*
  by (*metis ordering-def assms*(*2*,*3*) *card.empty card-gt-0-iff diff-less lch-X 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** *fin-chain-def fin-long-chain-def*
  by (*metis* (*no-types*, *hide-lams*) *One-nat-def Suc-1 ch-by-ord-def diff-Suc-Suc less-Suc-eq-le neq0-conv numeral-3-eq-3 short-ch-card-2 zero-less-diff*)

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 *assms*(*2*) *lch-X bound-ind* ‹*3* ≤ *card X*› **unfolding** *long-ch-by-ord-def ordering-def*
    by (*metis One-nat-def diff-Suc-less less-le-trans zero-less-numeral*)
qed
hence *p*≠*c*
  using *abc-abc-neq* **by** *blast*
hence *p*∈*Y*
  using ‹*p* ∈ *X*› *assms*(*4*) **by** *blast*

show *?thesis*
proof (*cases*)
  assume *3* = *card X*
  hence *2* = *card Y*
  by (*metis assms*(*4*,*5*) *card.insert card.infinite diff-Suc-1 finite-insert nat.simps*(*3*) *numeral-2-eq-2 numeral-3-eq-3*)
  have *a*≠*p*
    using ‹[[*a p c*]]› *abc-abc-neq* **by** *auto*
  moreover have *a*∈*Y* ∧ *p*∈*Y*
    using ‹[[*a p c*]]› ‹*p* ∈ *Y*› *abc-abc-neq assms*(*1*,*4*) *fin-chain-def points-in-chain*
    by *fastforce*
  moreover have *short-ch Y*
  proof −
    obtain *ap* **where** *path ap a p*
      using ‹[[*a p c*]]› *abc-ex-path-unique calculation*(*1*) **by** *blast*
    hence ∃ *Q*. *path Q a p*
      by *blast*
    moreover have ¬ (∃ *z*∈*Y*. *z* ≠ *a* ∧ *z* ≠ *p*)
      using ‹*2* = *card Y*› ‹*a* ∈ *Y* ∧ *p* ∈ *Y*› ‹*a* ≠ *p*›
      by (*metis card-2-iff'*)
    ultimately show *?thesis*
      unfolding *short-ch-def* **using** ‹*a* ∈ *Y* ∧ *p* ∈ *Y*›
      by *blast*
  qed
  ultimately show *?thesis* **unfolding** *fin-chain-def* **by** *blast*

```
next
  assume 3 ≠ card X
  hence 4 ≤ card X
    using assms(3) by auto

  obtain b where b = f 1 by simp
  have ∃ b. [f[a..b..p]Y]
  proof
    have [[a b p]]
     using bound-ind ‹b = f 1› ‹3 ≠ card X› assms(2,3) lch-X order-finite-chain
      by fastforce
    hence all-neq: b≠a ∧ b≠p ∧ a≠p
      using abc-abc-neq by blast
    have b∈X
      using ‹b = f 1› lch-X assms(3) unfolding long-ch-by-ord-def ordering-def
      by force
    hence b∈Y
      using ‹[[a b p]]› ‹[[a p c]]› abc-only-cba(2) assms(4) by blast

    have ordering f betw Y
      unfolding ordering-def
    proof (safe)
      show ⋀n. infinite Y ⟹ f n ∈ Y
        using assms(3) assms(4) by auto
      show ⋀n. n < card Y ⟹ f n ∈ Y
        using assms(3,4,5) bound-ind lch-X
        unfolding long-ch-by-ord-def ordering-def
        using get-fin-long-ch-bounds indices-neq-imp-events-neq
          by (smt Suc-less-eq add-leD1 cancel-comm-monoid-add-class.diff-cancel
card-Diff1-less
              card-Diff-singleton card-eq-0-iff card-insert-disjoint gr-implies-not0
insert-iff lch-X
          le-add-diff-inverse less-SucI numeral-3-eq-3 plus-1-eq-Suc zero-less-diff)
      {
        fix x assume x∈Y
        hence x∈X
          using assms(4) by blast
        then obtain n where n < card X f n = x
          using lch-X unfolding long-ch-by-ord-def ordering-def
          using assms(3) by auto
        show ∃ n. (finite Y ⟶ n < card Y) ∧ f n = x
        proof
          show (finite Y ⟶ n < card Y) ∧ f n = x
            using ‹f n = x› ‹n < card X› ‹x ∈ Y› assms(4,5) bound-ind
            by (metis Diff-insert-absorb card.remove card-Diff-singleton
                finite.insertI insertI1 less-SucE)
        qed
      }
      fix n n′ n″
```

assume $(n::nat)<n'$ $n'<n''$
{
assume *infinite Y*
show $[[(f\ n)\ (f\ n')\ (f\ n'')]]$
using $\langle\bigwedge n.$ *infinite* $Y \Longrightarrow f\ n \in Y\rangle$ $\langle$*infinite* $Y\rangle$ *assms*(*5*) *bound-ind* **by** *blast*
} {
assume $n'' <$ *card* Y
show $[[(f\ n)\ (f\ n')\ (f\ n'')]]$
using $\langle n < n'\rangle$ $\langle n' < n''\rangle$ $\langle n'' <$ *card* $Y\rangle$ *assms*(*4*,*5*) *lch-X order-finite-chain*
using $\langle$*infinite* $Y \Longrightarrow [[(f\ n)\ (f\ n')\ (f\ n'')]]\rangle$ **by** *fastforce*
}
qed
hence *lch-Y*: *long-ch-by-ord f Y*
using $\langle[[a\ p\ c]]\rangle$ $\langle b \in Y\rangle$ $\langle p \in X\rangle$ *abc-abc-neq all-neq assms*(*4*) *bound-ind*
long-ch-by-ord-def zero-into-ordering
by *fastforce*

show $[f[a..b..p]Y]$
using *all-neq lch-Y bound-ind* $\langle b{\in}Y\rangle$ *assms*(*2*,*3*,*4*,*5*) **unfolding** *fin-long-chain-def*
by (*metis Diff-insert-absorb One-nat-def add-leD1 card.infinite finite-insert plus-1-eq-Suc*
diff-diff-left card-Diff-singleton not-one-le-zero insertI1 numeral-2-eq-2 numeral-3-eq-3)
qed

thus *?thesis* **unfolding** *fin-chain-def*
using *points-in-chain* **by** *blast*
qed
qed

lemma (**in** *MinkowskiChain*) *fin-long-ch-imp-fin-ch*:
assumes $[f[a..b..c]X]$
shows $[f[a..c]X]$
using *assms fin-chain-def points-in-chain* **by** *auto*

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 (**in** *MinkowskiSpacetime*) *chain-unique-induction-ax*:
assumes *card* $X \geq 3$
and $i <$ *card* X
and $[f[a..c]X]$
and $[g[x..z]X]$
and $a = x \lor c = z$
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*: $[f[a..b..c]X]$
  by (*metis Nil.prems(1,3) fin-chain-def short-ch-def three-in-set3*)
obtain *y* **where** *g-ch*: $[g[x..y..z]X]$
  using *Nil.prems fin-chain-def short-ch-card-2*
  by (*metis Suc-n-not-le-n ch-by-ord-def numeral-2-eq-2 numeral-3-eq-3*)

have $i=1 \lor i=0 \lor i=2$
  using ‹*card X = 3*› *Nil.prems(2)* **by** *linarith*
thus *?case*
proof (*rule disjE*)
  assume $i=1$
  hence $f\ i = b \land g\ i = y$
    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 $f\ i \neq g\ i$
    hence $g\ i \neq b$
      by (*simp add*: ‹$f\ i = b \land g\ i = y$›)
    have $g\ i \in X$
      using ‹$f\ i = b \land g\ i = y$› *g-ch points-in-chain* **by** *blast*
    hence $(g\ i = a \lor g\ i = c)$
      using ‹$g\ i \neq b$› ‹*card X = 3*› *points-in-chain*
      by (*smt f-ch card2-either-elt1-or-elt2 card-Diff-singleton diff-Suc-1*
        fin-long-chain-def insert-Diff insert-iff numeral-2-eq-2 numeral-3-eq-3)
    hence $\neg\ [[a\ (g\ i)\ c]]$
      using *abc-abc-neq* **by** *blast*
    hence $g\ i \notin X$
      using ‹$f\ i=b \land g\ i=y$› ‹$g\ i=a \lor g\ i=c$› *f-ch g-ch chain-bounds-unique*
fin-long-chain-def
      by *blast*
    thus *False*
      by (*simp add*: ‹$g\ i \in X$›)
  qed
  thus *?thesis*
    by (*simp add*: ‹*card X = 3*› ‹$i = 1$›)
next
  assume $i = 0 \lor i = 2$
  show *?thesis*
    using *Nil.prems(5)* ‹*card X = 3*› ‹$i = 0 \lor i = 2$› *chain-bounds-unique f-ch*
g-ch
    by (*metis diff-Suc-1 fin-long-chain-def numeral-2-eq-2 numeral-3-eq-3*)
qed
next
  case *IH*: (*Suc n*)
  have *lch-fX*: *long-ch-by-ord f X*

using *ch-by-ord-def fin-chain-def fin-long-chain-def long-ch-card-ge3 IH*(*3*,*5*)
by *fastforce*
have *lch-gX*: *long-ch-by-ord g X*
using *IH*(*3*,*6*) *ch-by-ord-def fin-chain-def fin-long-chain-def 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* $\geq$ *4*
using *IH.hyps*(*2*) **by** *linarith*

obtain *b* **where** *f-ch*: [*f*[*a*..*b*..*c*]*X*]
using ‹*ch-by-ord f X*› *IH*(*3*,*5*) *fin-chain-def short-ch-card-2*
by *auto*
obtain *y* **where** *g-ch*: [*g*[*x*..*y*..*z*]*X*]
using ‹*ch-by-ord f X*› *IH.prems*(*1*,*4*) *fin-chain-def short-ch-card-2*
by *auto*

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* **unfolding** *fin-long-chain-def long-ch-by-ord-def ordering-def*
by (*metis card-Diff1-less card-Diff-singleton*)
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-chain*
by (*meson mk-disjoint-insert*)
hence *fin-Y*: *finite Y*
using *f-ch fin-long-chain-def* **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 fin-chain-def points-in-chain)
  have [f[a..p]Y]
     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: long-ch-by-ord f Y
    unfolding fin-chain-def
    using card-Y ch-by-ord-def fin-Y fin-long-chain-def 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›
      by (metis ordering-def long-ch-by-ord-def)
    have j > card X − 2 ∧ j < card X − 1
    proof −
      have j > card X − 2 ∧ j < card X − 1 ∨ j < card X − 2 ∧ j > card X − 1
        using index-order3 [where b=j and a=card X − 2 and c=card X − 1]
        using ‹[[p q c]]› ‹f j = q› ‹j < card X› f-ch p-def
        by (metis (no-types, lifting) One-nat-def card-gt-0-iff diff-less empty-iff
          fin-long-chain-def lessI zero-less-numeral)
      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, hide-lams) chain-bounds-unique f-ch
        fin-long-chain-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* ‹$j < card\ Y - 1$› ‹$card\ Y = card\ X - 1$› ‹$d = g\ (card\ X - 2)$› ‹$g\ j = p$›
unfolding *long-ch-by-ord-def ordering-def*
by (*metis* (*mono-tags*, *lifting*) *One-nat-def card-Diff1-less card-Diff-singleton diff-diff-left fin-long-chain-def g-ch numeral-2-eq-2 plus-1-eq-Suc*)
qed
moreover have $d \in X$
using *lch-gX* ‹$d = g\ (card\ X - 2)$› **unfolding** *long-ch-by-ord-def ordering-def*
by *auto*
ultimately show *False*
using *p-closest abc-sym IH.prems*(*5*) *chain-bounds-unique f-ch g-ch*
by *blast*
qed

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

have $f\ i \in Y \vee f\ i = c$
by (*metis ordering-def* ‹$X = insert\ c\ Y$› ‹$i < card\ X$› *lch-fX insert-iff long-ch-by-ord-def*)
thus $f\ i = g\ i$
proof (*rule disjE*)
assume $f\ i \in Y$
hence $f\ i \neq c$
using ‹$c \notin Y$› **by** *blast*
hence $i < card\ Y$
using ‹$X = insert\ c\ Y$› ‹$c \notin Y$› *IH*(*3*,*4*) *f-ch fin-Y fin-long-chain-def not-less-less-Suc-eq*
by *fastforce*
hence $3 \leq card\ Y$
using *card-Y le-add2* **by** *presburger*
show $f\ i = g\ i$
using *IH*(*1*) [*of Y*]
using ‹$n = card\ Y - 3$› ‹$3 \leq card\ Y$› ‹$i < card\ Y$›
using *Y-def card-Y chain-remove-at-right-edge le-add2*
by (*metis IH.prems*(*1*,*3*,*4*,*5*) *chain-bounds-unique2*)
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*
by (*metis* ‹$card\ Y = card\ X - 1$› *Y-def card-insert-disjoint fin-Y fin-long-chain-def lessI*)
qed
qed

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

lemma (**in** *MinkowskiSpacetime*) *chain-unique-induction-cx*:
assumes $card\ X \geq 3$

and $i < card\ X$
and $[f[a..c]X]$
and $[g[x..z]X]$
and $c = x \vee a = z$
shows $f\ i = g\ (card\ X - i - 1)$
using *chain-sym chain-unique-induction-ax*
by (*smt* (*verit*, *best*) *assms diff-right-commute fin-chain-def fin-long-ch-imp-fin-ch*)

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) ”ordering” of the chain. This could be made generic over the ordering similar to *chain-sym* relying on *ordering-sym*.

lemma (**in** *MinkowskiSpacetime*) *chain-unique-upto-rev-cases*:
assumes *ch-f*: $[f[a..c]X]$
and *ch-g*: $[g[x..z]X]$
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 (**in** *MinkowskiSpacetime*) *chain-unique-upto-rev*:
assumes $[f[a..c]X]$ $[g[x..z]X]$ $card\ X \geq 3$ $i < card\ X$
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*) *fin-chain-def points-in-chain short-ch-def*)
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-unique2*)
qed

34 Subchains

context *MinkowskiSpacetime* **begin**

```
lemma f-img-is-subset:
  assumes [f[(f 0) ..]X] 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 ordering-def assms(1) inf-chain-is-long long-ch-by-ord-def)
  thus x∈X
    using ‹f n = x› by blast
qed
```

```
lemma f-inj-on-index-subset:
  assumes [f[(f 0) ..]X] i≥0 j>i Y=f‘{i..j}
  shows inj-on f {i..j}
  unfolding inj-on-def
proof (safe)
  fix x y assume x∈{i..j} y∈{i..j} f x = f y
  show x=y
  proof (rule ccontr)
    assume x≠y
    let ?P = λr s. f r ≠ f s
    {
      assume x≤y
      hence x<y
        using ‹x ≠ y› le-imp-less-or-eq by blast
      obtain n where n>y by blast
      hence [[(f x)(f y)(f n)]]
         using assms(1) ‹x<y› inf-chain-is-long long-ch-by-ord-def ordering-ord-ijk
by fastforce
      hence ?P x y
        using abc-abc-neq by blast
    } moreover {
      assume x>y
      obtain n where n>x by blast
      hence [[(f y)(f x)(f n)]]
         using assms(1) ‹x>y› inf-chain-is-long long-ch-by-ord-def ordering-ord-ijk
by fastforce
      hence ?P y x
        using abc-abc-neq by blast
    }
    ultimately show False
```

using *not-le-imp-less* ‹$f\ x = f\ y$› **by** *auto*
qed
qed

lemma *f-bij-on-index-subset*:
assumes $[f[(f\ 0)\ ..]X]$ $i \geq 0$ $j > i$ $Y = f`\{i..j\}$
shows *bij-betw* f $\{i..j\}$ Y
using *f-inj-on-index-subset*
by (*metis assms inj-on-imp-bij-betw*)

lemma *only-one-index*:
assumes $[f[(f\ 0)\ ..]X]$ $i \geq 0$ $j > i$ $Y = f`\{i..j\}$ $f\ n \in Y$
shows $n \in \{i..j\}$
proof −
obtain m **where** $m \in \{i..j\}$ $f\ m = f\ n$
using *assms*(*4*) *assms*(*5*) **by** *auto*
have *inj-on* f $\{i..j\}$
using *assms*(*1*,*3*) *f-inj-on-index-subset* **by** *blast*
have $m = n$
proof (*rule ccontr*)
assume $m \neq n$
obtain l **where** $f\ l \in X$ $l \neq m$ $l \neq n$
using *assms*(*1*) *inf-chain-is-long*
by (*metis ordering-def le-eq-less-or-eq lessI long-ch-by-ord-def not-less-eq-eq*)
hence $[[(f\ l)(f\ m)(f\ n)]] \lor [[(f\ m)(f\ l)(f\ n)]] \lor [[(f\ l)(f\ n)(f\ m)]]$
using ‹$f\ m = f\ n$› ‹$m \neq n$›
using *abc-abc-neq assms*(*1*) *inf-chain-is-long inf-ordering-inj′ long-ch-by-ord-def*
by *blast*
thus *False*
using ‹$f\ m = f\ n$› *abc-abc-neq* **by** *auto*
qed
thus *?thesis*
using ‹$m \in \{i..j\}$› **by** *blast*
qed

lemma *f-one-to-one-on-index-subset*:
assumes $[f[(f\ 0)\ ..]X]$ $i \geq 0$ $j > i$ $Y = f`\{i..j\}$ $y \in Y$
shows $\exists!k \in \{i..j\}.\ f\ k = y$ $f\ k = y \longrightarrow k \in \{i..j\}$
using *f-inj-on-index-subset only-one-index assms image-iff inj-on-eq-iff* **apply** *metis*
using *assms*(*1*,*3*,*4*,*5*) *only-one-index* **by** *blast*

lemma *card-of-subchain*:
assumes $[f[(f\ 0)\ ..]X]$ $i \geq 0$ $j > i$ $Y = f`\{i..j\}$
shows *card* Y = *card* $\{i..j\}$ *card* $Y = j - i + 1$

```
proof -
  show card Y = card {i..j}
    by (metis assms bij-betw-same-card f-bij-on-index-subset)
  thus card Y = j−i+1
    using card-Collect-nat
    by (simp add: assms(3))
qed


lemma fin-long-subchain-of-semifin:
  assumes [f[(f 0) ..]X] i≥0 j>i+1 Y=f‘{i..j}
    g = (λn. f(n+i))
  shows [g[(f i)..(f j)]Y]
proof -
  obtain k where k=i+1 by simp
  hence ind-ord: i<k ∧ k<j using assms(3) by simp
  have [g[(f i) .. (f k) .. (f j)]Y]
  proof -
    have f i ≠ f k ∧ f i ≠ f j ∧ f k ≠ f j
    proof -
      have [[(f i) (f k) (f j)]]
       using assms(1) ind-ord long-ch-by-ord-def ordering-ord-ijk semifin-chain-def
        by fastforce
      thus ?thesis
        using abc-abc-neq by blast
    qed
    moreover have finite Y
    proof -
      have inj f
        using inf-ordering-inj [where ord=betw] abc-abc-neq
        using assms(1) long-ch-by-ord-def semifin-chain-def by auto
      hence card Y ≤ card {i..j}
        using assms(4) inf-ordering-inj
        using card-image-le by blast
      have finite {i..j}
        by simp
      thus finite Y
        by (simp add: assms(4))
    qed
    moreover have long-ch-by-ord g Y
    proof -
      obtain x y z where x=f i y=f k z=f j
        by auto
      have x∈Y ∧ y∈Y ∧ z∈Y ∧ x ≠ y ∧ y ≠ z ∧ x ≠ z
        using ‹x = f i› ‹y = f k› ‹z = f j› assms(4) calculation(1) ind-ord by auto
      moreover have ordering g betw Y
        unfolding ordering-def
      proof (rule conjI3)
        show ∀n. (finite Y ⟶ n < card Y) ⟶ g n ∈ Y
```

```
      apply (safe) apply (simp add: ‹finite Y›)
    proof -
      fix n assume n<card Y
      then obtain n' where n+i = n' n'∈{i..j}
      proof -
        assume asm: ⋀n'. ⟦n + i = n'; n' ∈ {i..j}⟧ ⟹ thesis
        have n < card {i..j}
          by (metis ‹n < card Y› assms(4) card-image-le finite-atLeastAtMost
less-le-trans)
        thus ?thesis
          using asm by simp
      qed
      show g n ∈ Y
        using ‹n + i = n'› ‹n' ∈ {i..j}› assms(4,5) by blast
    qed
    show ∀x∈Y. ∃n. (finite Y ⟶ n < card Y) ∧ g n = x
    proof (rule ballI)
      fix x assume x∈Y
      hence x∈X
        using f-img-is-subset assms(1,4)
        by (metis ordering-def imageE inf-chain-is-long long-ch-by-ord-def)
      then obtain n where f n = x
        using ‹x ∈ Y› assms(4) by blast
      have n∈{i..j} using only-one-index
        by (metis ‹f n = x› ‹x ∈ Y› assms(1,2,4) ind-ord less-trans)
      show ∃n. (finite Y ⟶ n < card Y) ∧ g n = x
      proof (rule exI, rule conjI)
        have n−i≥0
          by blast
        have g (n−i) = f (n−i+i)
          using assms(5) by blast
        show g (n−i) = x
        proof (cases)
          assume n−i>0
          thus ?thesis
            by (simp add: ‹f n = x› ‹g (n − i) = f (n − i + i)›)
        next assume ¬n−i>0
          hence n−i=0 by blast
          thus ?thesis
            using ‹n∈{i..j}› ‹f n = x› ‹g (n − i) = f (n − i + i)› by auto
        qed
        show finite Y ⟶ (n−i) < card Y
        proof
          assume finite Y
          show n−i<card Y
            using card-of-subchain
            using ‹n ∈ {i..j}› assms(1,4) ind-ord by auto
        qed
      qed
```

qed
show $\forall n\ n'\ n''$. (*finite Y* $\longrightarrow n'' <$ *card Y*) $\wedge$ $n{<}n'$ $\wedge$ $n'{<}n''$ $\longrightarrow$ [[(*g n*)(*g n'*)(*g n''*)]]
apply (*safe*) **using** ‹*finite Y*› **apply** *blast*
proof −
fix *l m n*
assume *l<m m<n n<card Y*
hence *l+i<m+i m+i<n+i*
apply *simp* **by** (*simp add*: ‹*m < n*›)
hence [[(*f*(*l+i*))(*f*(*m+i*))(*f*(*n+i*))]]
using *assms*(*1*) *inf-chain-is-long long-ch-by-ord-def ordering-ord-ijk* **by** *fastforce*
thus [[(*g l*)(*g m*)(*g n*)]]
using *assms*(*5*) **by** *blast*
qed
qed
ultimately show *?thesis*
using *long-ch-by-ord-def* **by** *auto*
qed
moreover have *g 0 = f i* $\wedge$ *f k* $\in$ *Y* $\wedge$ *g* (*card Y* − *1*) = *f j*
using *card-of-subchain assms*(*1*,*4*,*5*) *ind-ord less-imp-le-nat*
by *force*
ultimately show *?thesis*
using *fin-long-chain-def* **by** *blast*
qed
thus *?thesis*
using *fin-long-ch-imp-fin-ch* **by** *blast*
qed

end

35 Extensions of results to infinite chains

context *MinkowskiSpacetime* **begin**

lemma *i-neq-j-imp-events-neq-inf*:
assumes [*f*[(*f 0*)..]*X*] *i≠j*
shows *f i ≠ f j*
proof −
let *?P* = λ *i j*. *i≠j* ⟶ *f i ≠ f j*
{
fix *i j* **assume** (*i*::*nat*)≤*j*
have *?P i j*
proof (*cases*)
assume *i<j*
then obtain *k* **where** *k>j* **by** *blast*
hence [[(*f i*)(*f j*)(*f k*)]]
using ‹*i < j*› *assms*(*1*) *inf-chain-is-long long-ch-by-ord-def ordering-ord-ijk* **by** *fastforce*

thus *?P i j*
using *abc-abc-neq* **by** *blast*
next
assume $\neg i<j$ **hence** $i=j$ **using** ‹$i \leq j$› **by** *auto*
show *?P i j* **by** (*simp add*: ‹$i = j$›)
qed
} **moreover** **{**
fix *i j* **assume** *?P j i*
hence *?P i j* **by** *auto*
}
ultimately show *?thesis*
by (*metis assms*(*2*) *leI less-imp-le-nat*)
qed

lemma *i-neq-j-imp-events-neq*:
assumes *long-ch-by-ord f X* $i \neq j$ *finite X* $\longrightarrow$ ($i<$*card X* $\wedge$ $j<$*card X*)
shows $f\ i \neq f\ j$
using *i-neq-j-imp-events-neq-inf indices-neq-imp-events-neq*
by (*meson assms get-fin-long-ch-bounds semifin-chain-def*)

lemma *inf-chain-origin-unique*:
assumes [*f*[*f 0*..]*X*] [*g*[*g 0*..]*X*]
shows $f\ 0 = g\ 0$
proof (*rule ccontr*)
assume $f\ 0 \neq g\ 0$
obtain *P* **where** $P \in \mathcal{P}$ $X \subseteq P$
using *assms*(*1*) *semifin-chain-on-path* **by** *blast*
obtain *x* **where** $x = g\ 1$ **by** *simp*
hence $x \neq g\ 0$
using *assms*(*2*) *i-neq-j-imp-events-neq-inf zero-neq-one* **by** *blast*
have $x \in X$
by (*metis ordering-def* ‹$x = g\ 1$› *assms*(*2*) *inf-chain-is-long long-ch-by-ord-def*)
have $x = f\ 0 \vee x \neq f\ 0$ **by** *auto*
thus *False*
proof (*rule disjE*)
assume $x = f\ 0$
hence [[(*g 0*)(*f 0*)(*g 2*)]]
using ‹$x=g\ 1$› ‹$x=f\ 0$› *assms*(*2*) *inf-chain-is-long long-ch-by-ord-def ordering-ord-ijk*
by *fastforce*
then obtain *m n* **where** $f\ m = g\ 0$ $f\ n = g\ 2$
by (*metis ordering-def assms*(*1*) *assms*(*2*) *inf-chain-is-long long-ch-by-ord-def*)
hence [[(*f m*)(*f 0*)(*f n*)]]
by (*simp add*: ‹[[(*g 0*)(*f 0*)(*g 2*)]]›)
hence $m \neq n$
using *abc-abc-neq* **by** *blast*
have $m>0 \wedge n>0$

using ‹[[(f m)(f 0)(f n)]]› *abc-abc-neq neq0-conv* **by** *blast*
hence $(0<m \wedge m<n) \vee (0<n \wedge n<m)$
using ‹$m \neq n$› **by** *auto*
thus *False*
using ‹[[(f m)(f 0)(f n)]]› *assms(1) index-order3 inf-chain-is-long* **by** *blast*
next
assume $x \neq f\ 0$

have *fn*: $\forall n.\ f\ n \in X$
by (*metis* (*no-types*) *ordering-def assms(1) inf-chain-is-long long-ch-by-ord-def*)
have *gn*: $\forall n.\ g\ n \in X$
by (*metis ordering-def assms(2) inf-chain-is-long long-ch-by-ord-def*)

have [[(g 0)x(f 0)]]
proof −
have [[(f 0)(g 0)x]] ∨ [[(g 0)(f 0)x]] ∨ [[(g 0)x(f 0)]]
using ‹$f\ 0 \neq g\ 0$› ‹$x \neq f\ 0$› ‹$x \neq g\ 0$› *all-aligned-on-semifin-chain*
by (*metis ordering-def* ‹$x \in X$› *assms inf-chain-is-long long-ch-by-ord-def*)
moreover have ¬[[(f 0)(g 0)x]]
using *abc-only-cba(1,3) all-aligned-on-semifin-chain assms(2) fn*
by (*metis* ‹$x \in X$› ‹$x \neq f\ 0$› ‹$x \neq g\ 0$›)
moreover have ¬[[(g 0)(f 0)x]]
using *fn gn* ‹$x \in X$› ‹$x \neq g\ 0$›
by (*metis* (*no-types*) *abc-only-cba(1,2,4) all-aligned-on-semifin-chain assms(1)*)
ultimately show *?thesis* **by** *blast*
qed

obtain $m\ m'$ **where** $g\ m' = f\ 0$ $m = Suc\ m'$
using *ordering-def assms inf-chain-is-long long-ch-by-ord-def* **by** *metis*
hence [[(g 0)(f 0)(g m)]]
by (*metis Suc-le-eq* ‹$f\ 0 \neq g\ 0$› *assms(2) inf-chain-is-long lessI linorder-neqE-nat long-ch-by-ord-def not-le ordering-ord-ijk zero-less-Suc*)
then obtain $n\ p$ **where** $f\ n = g\ 0$ $f\ p = g\ m$
by (*metis abc-abc-neq abc-only-cba(1,4) all-aligned-on-semifin-chain assms(1) gn*)
hence $m<0 \vee n<0$
using *all-aligned-on-semifin-chain assms(1)* ‹[[(g 0)(f 0)(g m)]]›
by (*metis abc-abc-neq abc-only-cba(1,4) fn*)
thus *False* **by** *simp*
qed
qed

lemma *inf-chain-unique*:
assumes [f[f 0..]X] [g[g 0..]X]
shows $\forall i::nat.\ f\ i = g\ i$
proof −
{

assume *asm*: $[f[f\ 0..]X]$ $[g[f\ 0..]X]$
have $\forall$ *i::nat*. $f\ i = g\ i$
proof
  fix *i::nat*
  show $f\ i = g\ i$
  proof (*induct i*)
    show $f\ 0 = g\ 0$
      using *asm*(*2*) *inf-chain-is-long* **by** *fastforce*
    fix *i* **assume** $f\ i = g\ i$
    show $f\ (Suc\ i) = g\ (Suc\ i)$
    proof (*rule ccontr*)
      assume $f\ (Suc\ i) \neq g\ (Suc\ i)$
      let $?i = Suc\ i$
      have $f\ 0{\in}X \wedge g?i{\in}X \wedge f?i{\in}X$
     by (*metis ordering-def assms*(*1*) *assms*(*2*) *inf-chain-is-long long-ch-by-ord-def*)
      hence $[[(f\ 0)(f\ ?i)(g\ ?i)]] \vee [[(f\ 0)(g\ ?i)(f\ ?i)]] \vee [[(f\ ?i)(f\ 0)(g\ ?i)]]$
        using *all-aligned-on-semifin-chain assms*(*1*,*2*) *i-neq-j-imp-events-neq-inf*
        by (*metis* ‹$f?i \neq g?i$› ‹$f\ 0 = g\ 0$›)
      hence $[[(f\ 0)(f\ ?i)(g\ ?i)]] \vee [[(f\ 0)(g\ ?i)(f\ ?i)]]$
        using *all-aligned-on-semifin-chain asm*(*2*)
        by (*metis* ‹$f\ 0 \in X \wedge g\ (Suc\ i) \in X \wedge f\ (Suc\ i) \in X$› *abc-abc-neq*)
      have $([[(f\ 0)(f\ i)(f\ ?i)]] \wedge [[(f\ 0)(g\ i)(g\ ?i)]]) \vee i{=}0$
        using *long-ch-by-ord-def ordering-ord-ijk asm*(*1*,*2*)
     by (*metis Suc-inject Suc-lessI Suc-less-eq inf-chain-is-long lessI zero-less-Suc*)
      thus *False*
      proof (*rule disjE*)
        assume $i{=}0$
        have $[[(g\ 0)(f\ 1)(g\ 1)]]$
        proof −
          obtain *x* **where** $x = g\ 1$ **by** *simp*
          hence $x{\in}X$
            using ‹$f\ 0 \in X \wedge g\ (Suc\ i) \in X \wedge f\ (Suc\ i) \in X$› ‹$i = 0$› **by** *force*
          then obtain *m* **where** $f\ m = x$
           by (*metis ordering-def assms*(*1*) *inf-chain-is-long long-ch-by-ord-def*)
          hence $f\ m = g\ 1$
            using ‹$x = g\ 1$› **by** *blast*
          have $m{>}1$
            using *assms*(*2*) *i-neq-j-imp-events-neq-inf* ‹$f?i \neq g?i$›
           by (*metis One-nat-def Suc-lessI* ‹$f\ 0 = g\ 0$› ‹$f\ m = x$› ‹$i = 0$› ‹$x = g$
1› *neq0-conv*)
          thus $[[(g\ 0)(f\ 1)(g\ 1)]]$
             using ‹$[[(f\ 0)(f?i)(g?i)]] \vee [[(f\ 0)(g?i)(f?i)]]$› ‹$f\ 0 = g\ 0$› ‹$f\ m = x$›
‹$i{=}0$› ‹$x = g\ 1$›
               by (*metis One-nat-def assms*(*1*) *gr-implies-not-zero index-order3*
inf-chain-is-long order.asym)
        qed
        have $f\ 1 \in X$
          using ‹$f\ 0 \in X \wedge g\ (Suc\ i) \in X \wedge f\ (Suc\ i) \in X$› ‹$i = 0$› **by** *auto*
        then obtain m' **where** $g\ m' = f\ 1$

by (*metis ordering-def assms*(*2*) *inf-chain-is-long long-ch-by-ord-def*)
hence $[[(g\ 0)(g\ m')(g\ 1)]]$
using ‹$[[(g\ 0)(f\ 1)(g\ 1)]]$› **by** *auto*
have $[[(g\ 0)(g\ 1)(g\ m')]]$
proof −
have $m' \neq 1 \wedge m' \neq 0$
using ‹$[[(g\ 0)(g\ m')(g\ 1)]]$› **by** (*meson abc-abc-neq*)
hence $m'>1$ **by** *auto*
thus $[[(g\ 0)(g\ 1)(g\ m')]]$
using ‹$[[(g\ 0)(g\ m')(g\ 1)]]$› *assms*(*2*) *index-order3 inf-chain-is-long* **by** *blast*
qed
thus *False*
using ‹$[[(g\ 0)(g\ m')(g\ 1)]]$› *abc-only-cba*(*2*) **by** *blast*
next
assume $[[(f\ 0)(f\ i)(f\ ?i)]] \wedge [[(f\ 0)(g\ i)(g\ ?i)]]$
have $[[(g\ 0)(f\ ?i)(g\ ?i)]]$
proof −
obtain x **where** $x = g\ ?i$ **by** *simp*
hence $x \in X$
by (*simp add*: ‹$f\ 0 \in X \wedge g\ (Suc\ i) \in X \wedge f\ (Suc\ i) \in X$›)
then obtain m **where** $f\ m = x$
by (*metis ordering-def assms*(*1*) *inf-chain-is-long long-ch-by-ord-def*)
hence $f\ m = g\ ?i$
using ‹$x = g\ ?i$› **by** *blast*
have $m>?i$
using *assms*(*2*) *i-neq-j-imp-events-neq-inf* ‹$f?i \neq g?i$›
by (*metis Suc-lessI* ‹$[[(f\ 0)(f\ i)(f\ ?i)]] \wedge [[(f\ 0)(g\ i)(g\ ?i)]]$› ‹$f\ i = g\ i$› ‹$f\ m = x$›
‹$x = g\ (Suc\ i)$› *assms*(*1*) *index-order3 less-nat-zero-code semifin-chain-def*)
thus $[[(g\ 0)(f\ ?i)(g\ ?i)]]$
using ‹$[[(f\ 0)(f?i)(g?i)]] \vee [[(f\ 0)(g?i)(f?i)]]$› ‹$f\ 0 = g\ 0$› ‹$f\ m = x$› ‹$x = g\ ?i$›
by (*metis assms*(*1*) *gr-implies-not-zero index-order3 inf-chain-is-long order.asym*)
qed
obtain m **where** $g\ m = f\ ?i$
using ‹$(f\ 0) \in X \wedge g?i \in X \wedge f?i \in X$› *assms*(*2*)
by (*metis ordering-def inf-chain-is-long long-ch-by-ord-def*)
hence $[[(g\ i)(g\ m)(g\ ?i)]]$
using *abc-acd-bcd* ‹$[[(f\ 0)(f\ i)(f?i)]] \wedge [[(f\ 0)(g\ i)(g\ ?i)]]$› ‹$[[(g\ 0)(f\ ?i)(g\ ?i)]]$›
by (*metis* ‹$f\ 0 = g\ 0$› ‹$f\ i = g\ i$›)
have $[[(g\ i)(g\ ?i)(g\ m)]]$
proof −
have $m>?i$
using ‹$[[(g\ i)(g\ m)(g\ ?i)]]$› *assms*(*2*) *index-order3 inf-chain-is-long* **by** *fastforce*

thus *?thesis*
using *assms(2) inf-chain-is-long long-ch-by-ord-def ordering-ord-ijk*
by *fastforce*
qed
thus *False*
using ‹[[(*g i*)(*g m*)(*g ?i*)]]› *abc-only-cba* **by** *blast*
qed
qed
qed
qed
}
moreover have *f 0 = g 0* **using** *inf-chain-origin-unique assms* **by** *blast*
ultimately show *?thesis* **using** *assms* **by** *auto*
qed

end

36 Interlude: betw4 and WLOG

36.1 betw4 - strict and non-strict, basic lemmas

context *MinkowskiBetweenness* **begin**

Define additional notation for non-strict ordering - cf Schutz p.27.

abbreviation *nonstrict-betw-right* :: *'a ⇒ 'a ⇒ 'a ⇒ bool* ([[- - -⟧) **where**
nonstrict-betw-right a b c ≡ [[*a b c*]] ∨ *b = c*

abbreviation *nonstrict-betw-left* :: *'a ⇒ 'a ⇒ 'a ⇒ bool* (⟦- - -]]) **where**
nonstrict-betw-left a b c ≡ [[*a b c*]] ∨ *b = a*

abbreviation *nonstrict-betw-both* :: *'a ⇒ 'a ⇒ 'a ⇒ bool* **where**
nonstrict-betw-both a b c ≡ *nonstrict-betw-left a b c* ∨ *nonstrict-betw-right a b c*

abbreviation *betw4* :: *'a ⇒ 'a ⇒ 'a ⇒ 'a ⇒ bool* ([[- - - -]]) **where**
betw4 a b c d ≡ [[*a b c*]] ∧ [[*b c d*]]

abbreviation *nonstrict-betw-right4* :: *'a ⇒ 'a ⇒ 'a ⇒ 'a ⇒ bool* ([[- - - -⟧) **where**
nonstrict-betw-right4 a b c d ≡ *betw4 a b c d* ∨ *c = d*

abbreviation *nonstrict-betw-left4* :: *'a ⇒ 'a ⇒ 'a ⇒ 'a ⇒ bool* (⟦- - - -]]) **where**
nonstrict-betw-left4 a b c d ≡ *betw4 a b c d* ∨ *a = b*

abbreviation *nonstrict-betw-both4* :: *'a ⇒ 'a ⇒ 'a ⇒ 'a ⇒ bool* **where**
nonstrict-betw-both4 a b c d ≡ *nonstrict-betw-left4 a b c d* ∨ *nonstrict-betw-right4 a b c d*

lemma *betw4-strong*:
assumes *betw4 a b c d*
shows [[*a b d*]] ∧ [[*a c d*]]

using *abc-bcd-acd assms* **by** *blast*

lemma *betw4-imp-neq*:
assumes *betw4 a b c d*
shows $a \neq b \wedge a \neq c \wedge a \neq d \wedge b \neq c \wedge b \neq d \wedge c \neq d$
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 *assms* **apply** (*rule disjE*) **using** *abc-acd-bcd* **apply** *blast*
using *assms* **apply** (*rule disjE*) **using** *abc-bcd-acd* **apply** *blast*
by (*meson abc-bcd-acd abd-bcd-abc*)

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 *betw4 a b c d*
shows ¬*betw4 a b d c* ¬*betw4 a c b d* ¬*betw4 a c d b* ¬*betw4 a d b c* ¬*betw4 a d c b*
¬*betw4 b a c d* ¬*betw4 b a d c* ¬*betw4 b c a d* ¬*betw4 b c d a* ¬*betw4 b d c a* ¬*betw4 b d a c*
¬*betw4 c a b d* ¬*betw4 c a d b* ¬*betw4 c b a d* ¬*betw4 c b d a* ¬*betw4 c d a b* ¬*betw4 c d b a*
¬*betw4 d a b c* ¬*betw4 d a c b* ¬*betw4 d b a c* ¬*betw4 d b c a* ¬*betw4 d c a b*
using *abc-only-cba assms* **apply** *blast+* **done**

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 ¬(*betw4 a b c d* ∨ *betw4 a b d c* ∨ *betw4 a c b d* ∨ *betw4 a c d b* ∨ *betw4 a d b c* ∨ *betw4 a d c b*)
shows *betw4 b a c d* ∨ *betw4 b a d c* ∨ *betw4 b c a d* ∨ *betw4 b d a c* ∨ *betw4 c a b d* ∨ *betw4 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 $\neg(betw4\ b\ a\ c\ d \vee betw4\ b\ a\ d\ c \vee betw4\ b\ c\ a\ d \vee betw4\ b\ d\ a\ c \vee betw4\ c\ a\ b\ d \vee betw4\ c\ b\ a\ d)$
shows $betw4\ a\ b\ c\ d \vee betw4\ a\ b\ d\ c \vee betw4\ a\ c\ b\ d \vee betw4\ a\ c\ d\ b \vee betw4\ a\ d\ b\ c \vee betw4\ a\ d\ c\ b$
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 $betw4\ a\ b\ c\ d \vee betw4\ a\ c\ b\ d$
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 ‹$P \in \mathcal{P}$› ‹$a \in P$› ‹$d \in P$› *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

36.2 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 $\bigwedge I\ J\ a\ b\ c\ d.\ [\![Q\ I\ a\ b;\ Q\ J\ c\ d;\ betw4\ a\ b\ c\ d]\!] \Longrightarrow P\ I\ J$
shows $\bigwedge I\ J\ a\ b\ c\ d.\ [\![Q\ I\ a\ b;\ Q\ J\ c\ d;$
$betw4\ b\ a\ c\ d \vee betw4\ a\ b\ d\ c \vee betw4\ b\ a\ d\ c \vee betw4\ d\ c\ b\ a]\!] \Longrightarrow P\ I\ J$
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 $\bigwedge I\ J\ a\ b\ c\ d.\ [\![Q\ I\ a\ b;\ Q\ J\ c\ d;\ betw4\ a\ c\ b\ d]\!] \Longrightarrow P\ I\ J$
shows $\bigwedge I\ J\ a\ b\ c\ d.\ [\![Q\ I\ a\ b;\ Q\ J\ c\ d;$
$betw4\ b\ c\ a\ d \vee betw4\ a\ d\ b\ c \vee betw4\ b\ d\ a\ c \vee betw4\ d\ b\ c\ a]\!] \Longrightarrow P\ I\ J$
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 $\bigwedge I\ J\ a\ b\ c\ d.\ [\![Q\ I\ a\ b;\ Q\ J\ c\ d;\ betw4\ a\ c\ d\ b]\!] \Longrightarrow P\ I\ J$
shows $\bigwedge I\ J\ a\ b\ c\ d.\ [\![Q\ I\ a\ b;\ Q\ J\ c\ d;$
$betw4\ a\ d\ c\ b \vee betw4\ b\ c\ d\ a \vee betw4\ b\ d\ c\ a \vee betw4\ c\ a\ b\ d]\!] \Longrightarrow P\ I\ J$
by (*meson assms*)

lemma (**in** *MinkowskiSpacetime*) *wlog-endpoints-distinct4*:
fixes Q:: $('a\ set) \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$
and P:: $('a\ set) \Rightarrow ('a\ set) \Rightarrow bool$
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 $\bigwedge I\ J\ a\ b\ c\ d.$

⟦$Q\ I\ a\ b$; $Q\ J\ c\ d$; $I \subseteq A$; $J \subseteq A$; $betw4\ a\ b\ c\ d \vee betw4\ a\ c\ b\ d \vee betw4\ a\ c\ d\ b$⟧ $\Longrightarrow P\ I\ J$
  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* **apply** *blast+* **done**
 show $P\ I\ J$
 proof (*cases*)
  assume $betw4\ b\ a\ c\ d \vee betw4\ b\ a\ d\ c \vee betw4\ b\ c\ a\ d \vee$
    $betw4\ b\ d\ a\ c \vee betw4\ c\ a\ b\ d \vee betw4\ c\ b\ a\ d$
  then consider $betw4\ b\ a\ c\ d | betw4\ b\ a\ d\ c | betw4\ b\ c\ a\ d |$
      $betw4\ b\ d\ a\ c | betw4\ c\ a\ b\ d | betw4\ c\ b\ a\ d$
   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 ¬($betw4\ b\ a\ c\ d \vee betw4\ b\ a\ d\ c \vee betw4\ b\ c\ a\ d \vee$
    $betw4\ b\ d\ a\ c \vee betw4\ c\ a\ b\ d \vee betw4\ c\ b\ a\ d$)
  hence $betw4\ a\ b\ c\ d \vee betw4\ a\ b\ d\ c \vee betw4\ a\ c\ b\ d \vee$
    $betw4\ a\ c\ d\ b \vee betw4\ a\ d\ b\ c \vee betw4\ a\ d\ c\ b$
   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 $betw4\ a\ b\ c\ d | betw4\ a\ b\ d\ c | betw4\ a\ c\ b\ d |$
      $betw4\ a\ c\ d\ b | betw4\ a\ d\ b\ c | betw4\ a\ d\ c\ b$
   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 $\bigwedge I\ J\ a\ b\ c\ d.$
$[\![Q\ I\ a\ b;\ Q\ J\ c\ d;\ I{\subseteq}A;\ J{\subseteq}A;\ betw4\ a\ b\ c\ d \vee betw4\ a\ c\ b\ d \vee betw4\ a\ c\ d\ b]\!] \Longrightarrow P\ I\ J$
shows $\bigwedge I\ J\ a\ b\ c\ d.\ [\![Q\ I\ a\ b;\ Q\ J\ c\ d;\ I{\subseteq}A;\ J{\subseteq}A;$
$a{\neq}b \wedge a{\neq}c \wedge a{\neq}d \wedge b{\neq}c \wedge b{\neq}d \wedge c{\neq}d]\!] \Longrightarrow P\ I\ J$
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*: $\bigwedge I\ J\ a\ b\ c\ d.\ [\![Q\ I\ a\ b;\ Q\ J\ c\ d;$
$(a{=}b \wedge b{=}c \wedge c{=}d) \vee (a{=}b \wedge b{\neq}c \wedge c{=}d)]\!] \Longrightarrow P\ I\ J$

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

and *no*: $\bigwedge I\ J\ a\ b\ c\ d.\ [\![Q\ I\ a\ b;\ Q\ J\ c\ d;$
$(a{\neq}b \wedge b{\neq}c \wedge c{\neq}d \wedge a{=}d) \vee (a{\neq}b \wedge b{=}c \wedge c{\neq}d \wedge a{=}d)]\!] \Longrightarrow P\ I\ J$
shows $\bigwedge I\ J\ a\ b\ c\ d.\ [\![Q\ I\ a\ b;\ Q\ J\ c\ d;\ \neg(a{\neq}b \wedge b{\neq}c \wedge c{\neq}d \wedge a{\neq}d \wedge a{\neq}c \wedge b{\neq}d)]\!] \Longrightarrow P\ I\ J$
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*: $\bigwedge a\ b\ I\ A.\ [\![I\subseteq A;\ A\in\mathcal{P};\ Q\ I\ a\ b]\!] \Longrightarrow b\in A \wedge a\in A$
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 $\bigwedge I\ J\ a\ b\ c\ d\ A.\ [\![Q\ I\ a\ b;\ Q\ J\ c\ d;\ I\subseteq A;\ J\subseteq A;\ A\in\mathcal{P};$
$[[a\ b\ c]] \wedge a=d]\!] \Longrightarrow P\ I\ J$
and $\bigwedge I\ J\ a\ b\ c\ d\ A.\ [\![Q\ I\ a\ b;\ Q\ J\ c\ d;\ I\subseteq A;\ J\subseteq A;\ A\in\mathcal{P};$
$[[b\ a\ c]] \wedge a=d]\!] \Longrightarrow P\ I\ J$
shows $\bigwedge I\ J\ a\ b\ c\ d\ A.\ [\![Q\ I\ a\ b;\ Q\ J\ c\ d;\ I\subseteq A;\ J\subseteq A;\ A\in\mathcal{P};$
$a\neq b \wedge b\neq c \wedge c\neq d \wedge a=d]\!] \Longrightarrow P\ I\ J$
proof −
have *last-case*: $\bigwedge I\ J\ a\ b\ c\ d\ A.\ [\![Q\ I\ a\ b;\ Q\ J\ c\ d;\ I\subseteq A;\ J\subseteq A;\ A\in\mathcal{P};$
$[[b\ c\ a]] \wedge a=d]\!] \Longrightarrow P\ I\ J$
using *assms*(*1*,*3*−*5*) **by** (*metis abc-sym*)
thus $\bigwedge I\ J\ a\ b\ c\ d\ A.\ [\![Q\ I\ a\ b;\ Q\ J\ c\ d;\ I\subseteq A;\ J\subseteq A;\ A\in\mathcal{P};$
$a\neq b \wedge b\neq c \wedge c\neq d \wedge a=d]\!] \Longrightarrow P\ I\ J$
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*: $\bigwedge a\ b\ I.\ [\![I\subseteq A;\ Q\ I\ a\ b]\!] \Longrightarrow b\in A \wedge a\in A$
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 $\bigwedge I\ J\ a\ b\ c\ d.\ [\![Q\ I\ a\ b;\ Q\ J\ c\ d;\ I\subseteq A;\ J\subseteq A]\!]$
$\Longrightarrow ((a=b \wedge b=c \wedge c=d) \longrightarrow P\ I\ J) \wedge ((a=b \wedge b\neq c \wedge c=d) \longrightarrow P\ I\ J)$
$\wedge\ ((a=b \wedge b=c \wedge c\neq d) \longrightarrow P\ I\ J) \wedge ((a=b \wedge b\neq c \wedge c\neq d \wedge a\neq d) \longrightarrow$
$P\ I\ J)$
$\wedge\ ((a\neq b \wedge b=c \wedge c\neq d \wedge a=d) \longrightarrow P\ I\ J)$
$\wedge\ (([[a\ b\ c]] \wedge a=d) \longrightarrow P\ I\ J) \wedge (([[b\ a\ c]] \wedge a=d) \longrightarrow P\ I\ J)$
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 −

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 $I\ J$
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) **apply** *blast*+ **done**
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) **apply** *blast*+ **done**
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*

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

end

36.3 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 $\bigwedge I\ J.\ [\![\textit{is-interval}\ I;\ \textit{is-interval}\ J;\ P\ I\ J]\!] \Longrightarrow P\ J\ I$

$\bigwedge I\ J\ a\ b\ c\ d.\ [\![I = interval\ a\ b;\ J = interval\ c\ d]\!]$
$\Longrightarrow (betw4\ a\ b\ c\ d \longrightarrow P\ I\ J) \land (betw4\ a\ c\ b\ d \longrightarrow P\ I\ J) \land (betw4\ a\ c\ d\ b \longrightarrow P\ I\ J)$
shows $\bigwedge I\ J\ Q\ a\ b\ c\ d.\ [\![I = interval\ a\ b;\ J = interval\ c\ d;\ I \subseteq Q;\ J \subseteq Q;\ Q \in \mathcal{P};$
$a \neq b \land a \neq c \land a \neq d \land b \neq c \land b \neq d \land c \neq d]\!] \Longrightarrow P\ I\ J$
proof −
let $?Q = \lambda\ I\ a\ b.\ I = interval\ a\ b$

fix $I\ J\ A\ a\ b\ c\ d$
assume *asm*: $?Q\ I\ a\ b\ ?Q\ J\ c\ d\ I \subseteq A\ J \subseteq A\ A \in \mathcal{P}\ a \neq b \land a \neq c \land a \neq d \land b \neq c \land b \neq d \land c \neq d$
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 $\bigwedge a\ b\ I.\ I \subseteq A \Longrightarrow ?Q\ I\ a\ b \Longrightarrow b \in A \land a \in A$
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 $\bigwedge I\ J\ a\ b\ c\ d.\ [\![?Q\ I\ a\ b;\ ?Q\ J\ c\ d;\ betw4\ a\ b\ c\ d \lor betw4\ a\ c\ b\ d \lor betw4\ a\ c\ d\ b]\!]$
$\Longrightarrow P\ I\ J$
by (*meson assms*(*2*))
show $I = interval\ a\ b\ J = interval\ c\ d\ I \subseteq A\ J \subseteq A\ A \in \mathcal{P}$
$a \neq b \land a \neq c \land a \neq d \land b \neq c \land b \neq d \land c \neq d$
using *asm* **apply** *simp+* **done**
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 $\bigwedge I\ J\ a\ b\ c\ d\ Q.\ [\![I = interval\ a\ b;\ J = interval\ c\ d;\ I \subseteq Q;\ J \subseteq Q;\ Q \in \mathcal{P}]\!]$
$\Longrightarrow ((a=b \land b=c \land c=d) \longrightarrow P\ I\ J) \land ((a=b \land b \neq c \land c=d) \longrightarrow P\ I\ J)$
$\land ((a=b \land b=c \land c \neq d) \longrightarrow P\ I\ J) \land ((a=b \land b \neq c \land c \neq d \land a \neq d) \longrightarrow P\ I\ J)$
$\land ((a \neq b \land b=c \land c \neq d \land a=d) \longrightarrow P\ I\ J)$
$\land (([[a\ b\ c]] \land a=d) \longrightarrow P\ I\ J) \land (([[b\ a\ c]] \land a=d) \longrightarrow P\ I\ J)$
shows $\bigwedge I\ J\ a\ b\ c\ d\ Q.\ [\![I = interval\ a\ b;\ J = interval\ c\ d;\ I \subseteq Q;\ J \subseteq Q;\ Q \in \mathcal{P};$
$\neg(a \neq b \land b \neq c \land c \neq d \land a \neq d \land a \neq c \land b \neq d)]\!] \Longrightarrow P\ I\ J$
proof −
let $?Q = \lambda\ I\ a\ b.\ I = interval\ a\ b$

fix $I\ J\ a\ b\ c\ d\ A$
assume *asm*: $?Q\ I\ a\ b\ ?Q\ J\ c\ d\ I \subseteq A\ J \subseteq A\ A \in \mathcal{P}\ \neg(a \neq b \land b \neq c \land c \neq d \land a \neq d \land a \neq c \land b \neq d)$
show $P\ I\ J$
apply (*rule wlog-endpoints-degenerate*)
proof −

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* **apply** *auto*+ **done**
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

37 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 (*betw4 a b c d* ⟶ *?prop I J*)
(*betw4 a c b d* ⟶ *?prop I J*)
(*betw4 a c d b* ⟶ *?prop I J*)
apply (*rule impI*) **apply** (*rule-tac*[*2*] *impI*) **apply** (*rule-tac*[*3*] *impI*)
proof −
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*›
apply (*simp add*: *interval-def seg-betw*)+ **done**
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 *betw4 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*
apply (*rule disjE*)
apply (*simp add*: ‹*I* = *interval a b*› *ends-in-int*)
using ‹*betw4 a c b d*› ‹*I* = *interval a b*› *interval-def seg-betw* **by** *auto*
have *x*∈*J*
using ‹*x*=*b* ∨ *x*=*c*› **apply** (*rule disjE*)
using ‹*betw4 a c b d*› ‹*J* = *interval c d*› *interval-def seg-betw* **apply** *auto*
done
hence *x*∈*I* ∧ *x*∈*J* **using** ‹*x* ∈ *I*› **by** *blast*

} **moreover** **{**
assume $\neg(x=b \vee x=c)$
hence $[[c\ x\ b]]$
using ‹$x\in$*interval c b*› **unfolding** *interval-def segment-def* **by** *simp*
hence $[[a\ x\ b]]$
by (*meson* ‹*betw4 a c b d*› *abc-acd-abd abc-sym*)
have $[[c\ x\ d]]$
using ‹*betw4 a c b d*› ‹$[[c\ x\ b]]$› *abc-acd-abd* **by** *blast*
have $x\in I$ $x\in J$
using ‹I = *interval a b*› ‹$[[a\ x\ b]]$› *interval-def seg-betw* **apply** *auto*[*1*]
using ‹J = *interval c d*› ‹$[[c\ x\ d]]$› *interval-def seg-betw* **by** *auto*
}
ultimately show $x\in I$ $x\in J$ **by** *blast+*
next
fix x
assume $x\in I$ $x\in J$
show $x \in$ *interval c b*
proof (*cases*)
assume *not-eq*: $x\neq a \wedge x\neq b \wedge x\neq c \wedge x\neq d$
have $[[a\ x\ b]]$ $[[c\ x\ d]]$
using ‹$x\in I$› ‹I = *interval a b*› *not-eq* **unfolding** *interval-def segment-def*
apply *blast*
using ‹$x\in J$› ‹J = *interval c d*› *not-eq* **unfolding** *interval-def segment-def*
by *blast*
hence $[[c\ x\ b]]$
by (*meson* ‹*betw4 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*: $\neg(x\neq a \wedge x\neq b \wedge x\neq c \wedge x\neq d)$
{
assume $x=a$
have $\neg[[d\ a\ c]]$
using ‹*betw4 a c b d*› *abcd-dcba-only*(*9*) **by** *blast*
hence $a \notin$ *interval c d* **unfolding** *interval-def segment-def*
using *abc-sym* ‹$a \neq b \wedge a \neq c \wedge a \neq d \wedge b \neq c \wedge b \neq d \wedge c \neq d$› **by**
blast
hence *False* **using** ‹$x\in J$› ‹J = *interval c d*› ‹$x=a$› **by** *blast*
} **moreover** **{**
assume $x=d$
have $\neg[[a\ d\ b]]$ **using** ‹*betw4 a c b d*› *abc-sym abcd-dcba-only*(*9*) **by** *blast*
hence $d\notin$*interval a b* **unfolding** *interval-def segment-def*
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** *blast*
hence *False* **using** ‹$x\in I$› ‹$x=d$› ‹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 betw4 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› ‹betw4 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 $?p = \lambda\ I\ J.\ (is\text{-}interval\ (I \cap J) \lor I{\cap}J = \{\})$

have *symmetry*: $\bigwedge I\ J.\ [\![is\text{-}interval\ I;\ is\text{-}interval\ J;\ ?p\ I\ J]\!] \Longrightarrow ?p\ J\ I$
by (*simp add*: *inf-commute*)

have *degen-cases*: $\bigwedge I\ J\ a\ b\ c\ d\ Q.\ [\![I = interval\ a\ b;\ J = interval\ c\ d;\ I{\subseteq}Q;\ J{\subseteq}Q;\ Q{\in}\mathcal{P}]\!]$
$\Longrightarrow ((a{=}b \land b{=}c \land c{=}d) \longrightarrow ?p\ I\ J) \land ((a{=}b \land b{\neq}c \land c{=}d) \longrightarrow ?p\ I\ J)$
$\land ((a{=}b \land b{=}c \land c{\neq}d) \longrightarrow ?p\ I\ J) \land ((a{=}b \land b{\neq}c \land c{\neq}d \land a{\neq}d) \longrightarrow ?p\ I\ J)$
$\land ((a{\neq}b \land b{=}c \land c{\neq}d \land a{=}d) \longrightarrow ?p\ I\ J)$
$\land (([[a\ b\ c]] \land a{=}d) \longrightarrow ?p\ I\ J) \land (([[b\ a\ c]] \land a{=}d) \longrightarrow ?p\ I\ J)$
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 $((a{=}b \land b{=}c \land c{=}d) \longrightarrow ?p\ I\ J) \land ((a{=}b \land b{\neq}c \land c{=}d) \longrightarrow ?p\ I\ J)$
$\land ((a{=}b \land b{=}c \land c{\neq}d) \longrightarrow ?p\ I\ J) \land ((a{=}b \land b{\neq}c \land c{\neq}d \land a{\neq}d) \longrightarrow ?p\ I\ J)$
$\land ((a{\neq}b \land b{=}c \land c{\neq}d \land a{=}d) \longrightarrow ?p\ I\ J)$
$\land (([[a\ b\ c]] \land a{=}d) \longrightarrow ?p\ I\ J) \land (([[b\ a\ c]] \land a{=}d) \longrightarrow ?p\ I\ J)$
apply (*rule conjI7*) **apply** (*rule-tac*[*1*−*7*] *impI*)
proof −
assume $a = b \land b = c \land c = d$ **thus** $?p\ I\ J$
using ‹$I = interval\ a\ b$› ‹$J = interval\ c\ d$› **by** *auto*
next
assume $a = b \land b \neq c \land c = d$ **thus** $?p\ I\ J$
using ‹$J = interval\ c\ d$› *empty-segment interval-def* **by** *auto*
next
assume $a = b \land b = c \land c \neq d$ **thus** $?p\ I\ J$
using ‹$I = interval\ a\ b$› *empty-segment interval-def* **by** *auto*
next
assume $a = b \land b \neq c \land c \neq d \land a \neq d$ **thus** $?p\ I\ J$
using ‹$I = interval\ a\ b$› *empty-segment interval-def* **by** *auto*
next
assume $a \neq b \land b = c \land c \neq d \land a = d$ **thus** $?p\ I\ J$
using ‹$I = interval\ a\ b$› ‹$J = interval\ c\ d$› *int-sym* **by** *auto*
next
assume $[[a\ b\ c]] \land a = d$ **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∈interval a b
      show x∈I
        by (simp add: ‹I = interval a b› ‹x ∈ interval a b›)
      have [[d b c]]
        using ‹[[a b c]] ∧ 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 apply (cases)
          apply (simp add: ‹J = interval c d› abc-abc-neq abc-sym interval-def
seg-betw)
        apply (simp add: ‹J = interval c d› ‹[[a b c]] ∧ a = d› ends-in-int)
        using ‹J = interval c d› ‹[[d b c]]› int-sym interval-def seg-betw by auto
    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 betw4 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*: $\bigwedge S.\ [\![\forall x \in S.\ (\textit{is-interval}\ x \wedge x \subseteq P);\ P \in \mathcal{P};\ \bigcap S \neq \{\};\ \textit{finite}\ S;\ S \neq \{\};\ m+2=\textit{card}\ S]\!]$
$\Longrightarrow$ *is-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 $I \subseteq P$ $J \subseteq P$
apply (*simp add*: *0.prems*(*1*))+ **done**
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** $I \in S$ $S = \textit{insert}\ I\ S'$ $I \notin S'$
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*) ‹$S = \textit{insert}\ I\ S'$› ‹$I \notin S'$› **by** *auto*
moreover have $\forall x \in S'.\ \textit{is-interval}\ x \wedge x \subseteq P$
by (*simp add*: *Suc.prems*(*1*) ‹$S = \textit{insert}\ I\ S'$›)
moreover have $\bigcap\ S' \neq \{\}$
using *Suc.prems*(*3*) ‹$S = \textit{insert}\ I\ S'$› **by** *auto*
moreover have *finite* S'
using *Suc.prems*(*4*) ‹$S = \textit{insert}\ I\ S'$› **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

38 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 "a of one ray and any r a 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 –
obtain b **where** $b\in\mathcal{E}$ **and** $b\in Q$ **and** $b\neq a$
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*
apply *safe*
using *inQ* **apply** *blast*
apply (*simp add*: ‹*b* ∈ *Q*› *abc-abc-neq betw-a-in-path event-a path-Q*)
apply (*simp add*: *event-a*)
apply (*simp add*: ‹*b* ∈ *Q*› *abc-abc-neq betw-b-in-path event-a path-Q*)
apply (*simp add*: ‹*b* ∈ *Q*› *abc-abc-neq betw-c-in-path event-a path-Q*)
by (*simp add*: ‹*b* ∈ *Q*›)
qed
have *disjointLR*: *?L* ∩ *?R* = {}
using *abc-abc-neq abc-only-cba* **by** *blast*

have *wxyz-ord*: *nonstrict-betw-right4 x a y b* ∨ *nonstrict-betw-right4 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-chain2*
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 (*rule conjI6*)
show *is-ray-on ?L Q*
unfolding *is-ray-on-def* **apply** *safe*
apply (*simp add*: *path-Q*)
using ‹*b* ∈ *Q*› ‹*b* ≠ *a*› *betw-a-in-path event-a path-Q* **apply** *blast*
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
  show is-ray-on ?R Q
    unfolding is-ray-on-def
    apply safe
    apply (simp add: path-Q)
    apply (simp add: ‹b ∈ Q› abc-abc-neq betw-b-in-path event-a path-Q)
    apply (simp add: ‹b ∈ Q› abc-abc-neq betw-c-in-path event-a path-Q)
    using ‹b ∈ Q› ‹b ≠ a› betw-a-in-path event-a path-Q apply blast
  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
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


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[(f 0)..]X] 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 $Q \in \mathcal{P}$ $[f[(f\ 0)..]X]$ $X \subseteq Q$ *closest-bound-f c X f is-bound-f b X f* $b \neq c$
assumes $x=b \vee x=c \vee [[c\ x\ b]] \vee [[c\ b\ x]]$
shows *is-bound-f x X f*
proof −
have $x \in Q$
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 $[f[f\ 0..]X]$
by (*simp add*: *assms*(*2*))
assume $i<j$
hence $[[(f\ i)(f\ j)b]]$
using *assms*(*5*) *is-bound-f-def* **by** *blast*
hence $[[(f\ j)\ b\ c]] \vee [[(f\ j)\ c\ b]]$
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 $[f[f\ 0..]X]$
by (*simp add*: *assms*(*2*))
assume $i<j$
hence $[[(f\ i)(f\ j)b]]$
using *assms*(*5*) *is-bound-f-def* **by** *blast*
hence $[[(f\ j)\ b\ c]] \vee [[(f\ j)\ c\ b]]$
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[(f 0)..]X] 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 ray-of-bounds:
  assumes [f[(f 0)..]X] closest-bound-f c X f is-bound-f b X f b≠c
```

shows *all-bounds X* = *insert c* (*ray c b*)
using *ray-of-bounds3 assms semifin-chain-on-path* **by** *blast*

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*∈*ray a b*
using *abc-abc-neq pro-betw ray-def* **by** *auto*
have *c*∉*interval a b*
using ‹[[*a b c*]]› *abc-abc-neq abc-only-cba*(*2*) *interval-def seg-betw* **by** *auto*
thus *?thesis*
using ‹*c* ∈ *ray a b*› **by** *blast*
qed
show *interval a b* ⊆ *insert a* (*ray a b*)
using *interval-def ray-def* **by** *auto*
qed

lemma *bound-any-f*:
assumes *Q*∈$\mathcal{P}$ [*f*[(*f 0*)..]*X*] *X*⊆*Q is-bound c X*
shows *is-bound-f c X f*
proof −
obtain *g* **where** *is-bound-f c X g* [*g*[*g 0*..]*X*]
using *assms*(*4*) *is-bound-def is-bound-f-def* **by** *blast*
show *?thesis*
unfolding *is-bound-f-def*
proof (*safe*)
fix *i j*::*nat*
show [*f*[*f 0* ..]*X*] **by** (*simp add*: *assms*(*2*))
assume *i*<*j*
have [[(*g i*)(*g j*)*c*]]
using ‹*i* < *j*› ‹*is-bound-f c X g*› *is-bound-f-def* **by** *blast*
thus [[(*f i*)(*f j*)*c*]]
using *inf-chain-unique* ‹[*g*[*g 0* ..]*X*]› *assms*(*2*) **by** *force*
qed
qed

lemma *closest-bound-any-f*:
assumes *Q*∈$\mathcal{P}$ [*f*[(*f 0*)..]*X*] *X*⊆*Q closest-bound c X*
shows *closest-bound-f c X f*
unfolding *closest-bound-f-def* **apply** *safe*

using *bound-any-f assms closest-bound-def is-bound-def* **apply** *blast*
by (*metis* (*full-types*) *assms*(*2*,*4*) *closest-bound-def inf-chain-unique is-bound-f-def*)

end

39 3.8 Connectedness of the unreachable set

context *MinkowskiSpacetime* **begin**

39.1 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*: $Q_x \in \emptyset\ Q\ b$ $Q_z \in \emptyset\ Q\ b$ $Q_x \neq Q_z$
and *xyz*: $[[Q_x\ \ Q_y\ \ Q_z]]$
shows $Q_y \in \emptyset\ Q\ b$
proof −

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*,*3*) *unreach-on-path xyz* **by** *blast*
hence *event-y*: $Q_y\in\mathcal{E}$
using *in-path-event path-Q* **by** *blast*
obtain *X f* **where** *X-def*: *ch-by-ord f X* $f\ 0 = Q_x$ $f\ (card\ X - 1) = Q_z$
$(\forall i\in\{1\ ..\ card\ X - 1\}.\ (f\ i) \in \emptyset\ Q\ b \wedge (\forall\ Qy\in\mathcal{E}.\ [[(f\ (i - 1))\ Qy\ (f\ i)]] \longrightarrow Qy \in \emptyset\ Q\ b))$
short-ch X $\longrightarrow Q_x \in X \wedge Q_z \in X \wedge (\forall\ Q_y\in\mathcal{E}.\ [[Q_x\ \ Q_y\ \ Q_z]] \longrightarrow Q_y \in \emptyset\ Q\ b)$
using *I6* [*OF assms*(*1*−*6*)] **by** *blast*
hence *fin-X*: *finite X*
using *unreach*(*3*) *not-less* **by** *fastforce*
obtain *N* **where** $N{=}card\ X$ $N{\geq}2$
using *X-def*(*2*,*3*) *unreach*(*3*) **by** *fastforce*

let $?a = f\ 0$
let $?d = f\ (card\ X - 1)$
{
assume $card\ X = 2$
hence *short-ch X* $?a \in X \wedge ?d \in X$ $?a \neq ?d$
using *X-def*(*1*) *short-ch-card-2* **apply** *blast*
using *X-def*(*1*,*2*,*3*,*5*) ‹$card\ X = 2$› *short-ch-card-2* **apply** *blast*
using *X-def*(*2*,*3*) *unreach*(*3*) **by** *blast*
}
hence $[f[Q_x..Q_z]X]$
unfolding *fin-chain-def*
by (*metis X-def*(*1*−*3*,*5*) *ch-by-ord-def fin-X fin-long-chain-def get-fin-long-ch-bounds unreach*(*3*))

have *y-cases*: $Q_y \in X \vee Q_y \notin X$ **by** *blast*
have *y-int*: $Q_y \in interval\ Q_x\ Q_z$
using *interval-def seg-betw xyz* **by** *auto*
have *X-in-Q*: $X \subseteq Q$
using *chain-on-path-I6* [**where** $Q=Q$ **and** $X=X$] *X-def event-b path-Q unreach*
by *blast*

show *?thesis*
proof (*cases*)

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

assume $N \neq 2$
hence $N \geq 3$ **using** ‹$2 \leq N$› **by** *auto*
have $2 \leq card\ X$ **using** ‹$2 \leq N$› ‹$N = card\ X$› **by** *blast*
show *?thesis* **using** *y-cases*
proof (*rule disjE*)
assume $Q_y \in X$
then obtain *i* **where** *i-def*: $i < card\ X\ Q_y = f\ i$
using *X-def*(*1*)
unfolding *ch-by-ord-def long-ch-by-ord-def ordering-def*
by (*metis X-def*(*5*) *abc-abc-neq fin-X short-ch-def xyz*)
have $i \neq 0 \wedge i \neq card\ X - 1$
using *X-def*(*2*,*3*)
by (*metis abc-abc-neq i-def*(*2*) *xyz*)
hence $i \in \{1..card\ X - 1\}$
using *i-def*(*1*) **by** *fastforce*
thus *?thesis* **using** *X-def*(*4*) *i-def*(*2*) **by** *metis*
next
assume $Q_y \notin X$

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

have $Q_y \in \bigcup ?S$
proof –
obtain *c* **where** $[f[Q_x..c..Q_z]X]$
using *X-def*(*1*) ‹$N = card\ X$› ‹$N \neq 2$› ‹$[f[Q_x..Q_z]X]$› *fin-chain-def short-ch-card-2*
by *auto*
have $interval\ Q_x\ Q_z = \bigcup ?S \cup X$
using *int-split-to-segs* [*OF* ‹$[f[Q_x..c..Q_z]X]$›] **by** *auto*
thus *?thesis*
using ‹$Q_y \notin X$› *y-int* **by** *blast*
qed
then obtain *s* **where** $s \in ?S\ Q_y \in s$ **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
```

39.2 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
      obtain x y where x∈A y∈A x≠y
        using asmA apply (rule disjE)
```

using *asmA numeral-2-eq-2* **apply** (*metis One-nat-def Suc-le-eq card-le-Suc-iff insert-iff*)
using *infinite-imp-nonempty* **by** (*metis finite-insert finite-subset singletonI subsetI*)
have $x{\in}Q$ $y{\in}Q$
using ‹$A \subseteq Q$› ‹$x \in A$› ‹$y \in A$› **by** *auto*
have $[[a\ x\ b]]$ $[[a\ y\ b]]$
apply (*simp add*: ‹($1 <$ *card* $A \vee$ *infinite* A) $\wedge$ ($\forall x{\in}A.\ [[a\ x\ b]]$)› ‹$x \in A$› ‹$y \in A$›)+ **done**
hence *betw4* $a\ x\ y\ b \vee$ *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*)

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

show *?I A a b ?I B c d* $A{\subseteq}Q$ $B{\subseteq}Q$ $Q{\in}\mathcal{P}$
using *assms* **apply** *simp*+ **done**
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*

show *?P I J*
if *?I I a b ?I J c d* $I{\subseteq}Q$ $J{\subseteq}Q$
and *betw4* $a\ b\ c\ d \vee$ *betw4* $a\ c\ b\ d \vee$ *betw4* $a\ c\ d\ b$
for $I\ J\ a\ b\ c\ d$
proof −
consider *betw4* $a\ b\ c\ d$|*betw4* $a\ c\ b\ d$|*betw4* $a\ c\ d\ b$
using ‹*betw4* $a\ b\ c\ d \vee$ *betw4* $a\ c\ b\ d \vee$ *betw4* $a\ c\ d\ b$› **by** *fastforce*
thus *?thesis*
proof (*cases*)
assume *asm*: *betw4* $a\ b\ c\ d$ **show** *?P I J*
proof −
have $\forall x{\in}\ I{\cup}J.\ [[a\ x\ d]]$

by (*metis Un-iff asm betw4-strong betw4-weak that*(*1*) *that*(*2*))
moreover have $a \in Q$ $d \in Q$
using *assms*(*5*) *on-path that*(*1*−*4*) **apply** *blast*+ **done**
ultimately show *?thesis* **by** *blast*
qed
next
assume *betw4 a c b d* **show** *?P I J*
proof −
have $\forall x \in I \cup J$. [[*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 $a \in Q$ $d \in Q$
using *assms*(*5*) *on-path that*(*1*−*4*) **apply** *blast*+ **done**
ultimately show *?thesis* **by** *blast*
qed
next
assume *betw4 a c d b* **show** *?P I J*
proof −
have $\forall x \in I \cup J$. [[*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 $a \in Q$ $b \in Q$
using *assms*(*5*) *on-path that*(*1*−*4*) **apply** *blast*+ **done**
ultimately show *?thesis* **by** *blast*
qed
qed
qed
qed
next
assume $\neg(a \neq b \wedge a \neq c \wedge a \neq d \wedge b \neq c \wedge b \neq d \wedge c \neq d)$

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

show $\bigwedge a\ b\ I.\ ?I\ I\ a\ b \Longrightarrow\ ?I\ I\ b\ a$ **using** *abc-sym* **by** *blast*
show $\bigwedge a\ b\ A.\ A \subseteq Q \Longrightarrow\ ?I\ A\ a\ b \Longrightarrow b \in Q \wedge a \in Q$ **using** *on-path* ‹$Q \in \mathcal{P}$› **by** *blast*
show $\bigwedge I\ J.\ ?R\ I \Longrightarrow\ ?R\ J \Longrightarrow\ ?P\ I\ J \Longrightarrow\ ?P\ J\ I$ **by** (*simp add*: *Un-commute*)

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

show $(a = b \wedge b = c \wedge c = d \longrightarrow\ ?P\ I\ J) \wedge (a = b \wedge b \neq c \wedge c = d \longrightarrow\ ?P\ I\ J) \wedge$

$(a = b \wedge b = c \wedge c \neq d \longrightarrow ?P\ I\ J) \wedge (a = b \wedge b \neq c \wedge c \neq d \wedge a \neq d \longrightarrow ?P\ I\ J) \wedge$
$(a \neq b \wedge b = c \wedge c \neq d \wedge a = d \longrightarrow ?P\ I\ J) \wedge$
$([[a\ b\ c]] \wedge a = d \longrightarrow ?P\ I\ J) \wedge ([[b\ a\ c]] \wedge a = d \longrightarrow ?P\ I\ J)$
if $?I\ I\ a\ b\ ?I\ J\ c\ d\ I \subseteq Q\ J \subseteq Q$
for $I\ J\ a\ b\ c\ d$
proof (*rule conjI7*, *rule-tac*[*1*−*7*] *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* **apply** *metis*+ **done**
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* **apply** *metis*+ **done**
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* **apply** *metis*+ **done**
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*) **apply** *blast*+ **done**
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* **apply** *blast+* **done**
  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∈$\mathcal{P}$
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∉∅ Q 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∈$\mathcal{P}$
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∈∅ Q a. [[y x z]]) ∧ (∀ x∈∅ Q b. [[y x z]])
proof −

have ∃ q∈Q. q∉(∅ Q a) ∃ q∈Q. q∉(∅ Q b)
using *cross-in-reachable reachable* **by** *blast+*

have *get-bds*: ∃ *la*∈Q. ∃ *ua*∈Q. *la*∉∅ Q a ∧ *ua*∉∅ Q a ∧ (∀ x∈∅ Q a. [[*la* x *ua*]])
if *asm*: a∉Q *path-ex* a q q∈Q
for a q
proof −
obtain *Qy* **where** *Qy*∈∅ Q a

using *asm(2)* ‹$a \notin Q$› *in-path-event path-Q two-in-unreach* **by** *blast*
then obtain *la* **where** $la \in Q - \emptyset\ Q\ a$
using *asm(2,3) cross-in-reachable* **by** *blast*
then obtain *ua* **where** $ua \in Q - \emptyset\ Q\ a$ $[[la\ Qy\ ua]]$ $la \neq ua$
using *unreachable-set-bounded* [**where** $Q=Q$ **and** $b=a$ **and** $Qx=la$ **and** $Qy=Qy$]
using ‹$Qy \in \emptyset\ Q\ a$› *asm in-path-event path-Q* **by** *blast*
have $la \notin \emptyset\ Q\ a \wedge ua \notin \emptyset\ Q\ a \wedge (\forall x{\in}\emptyset\ Q\ a.\ (x{\neq}la \wedge x{\neq}ua) \longrightarrow [[la\ x\ ua]])$
apply (*rule conjI*) **apply** (*rule-tac*[*2*] *conjI*)
using ‹$la \in Q - \emptyset\ Q\ a$› **apply** *blast*
using ‹$ua \in Q - \emptyset\ Q\ a$› **apply** *blast*
proof (*safe*)
fix *x* **assume** $x{\in}\emptyset\ Q\ a$ $x{\neq}la$ $x{\neq}ua$
{
assume $x=Qy$ **hence** $[[la\ x\ ua]]$ **by** (*simp add*: ‹$[[la\ Qy\ ua]]$›)
} moreover {
assume $x{\neq}Qy$
have $[[Qy\ x\ la]] \vee [[la\ Qy\ x]]$
proof −
{ assume $[[x\ la\ Qy]]$
hence $la{\in}\emptyset\ Q\ a$
using *unreach-connected* ‹$Qy{\in}\emptyset\ Q\ a$›‹$x{\in}\emptyset\ Q\ a$›‹$x{\neq}Qy$› *in-path-event path-Q that* **by** *blast*
hence *False*
using ‹$la \in Q - \emptyset\ Q\ a$› **by** *blast* **}**
thus $[[Qy\ x\ la]] \vee [[la\ Qy\ x]]$
using *some-betw* [**where** $Q=Q$ **and** $a=x$ **and** $b=la$ **and** $c=Qy$] *path-Q unreach-on-path*
using ‹$Qy \in \emptyset\ Q\ a$› ‹$la \in Q - \emptyset\ Q\ a$› ‹$x \in \emptyset\ Q\ a$› ‹$x \neq Qy$› ‹$x \neq la$› **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 $[[la\ x\ ua]] \vee [[la\ ua\ x]]$
using ‹$[[la\ Qy\ ua]]$› ‹$x \neq ua$› *abc-abd-acdadc* **by** *auto*
have $\neg[[la\ ua\ x]]$
using *unreach-connected that abc-abc-neq abc-acd-bcd in-path-event path-Q*
by (*metis DiffD2* ‹$Qy \in \emptyset\ Q\ a$› ‹$[[la\ Qy\ ua]]$› ‹$ua \in Q - \emptyset\ Q\ a$› ‹$x \in \emptyset\ Q\ a$›)
show *?thesis*
using ‹$[[la\ x\ ua]] \vee [[la\ ua\ x]]$› ‹$\neg\ [[la\ ua\ x]]$› **by** *linarith*
qed
}
ultimately show $[[la\ x\ ua]]$ **by** *blast*

qed
thus *?thesis* **using** ‹$la \in Q - \emptyset\ Q\ a$› ‹$ua \in Q - \emptyset\ Q\ a$› **by** *force*
qed

have $\exists\, y{\in}Q.\ \exists\, z{\in}Q.\ (\forall\, x{\in}(\emptyset\ Q\ a)\cup(\emptyset\ Q\ b).\ [[y\ x\ z]])$
proof −
obtain *la ua* **where** $\forall\, x{\in}\emptyset\ Q\ a.\ [[la\ x\ ua]]$
using *events*(*1*) *get-bds reachable*(*1*,*3*) **by** *blast*
obtain *lb ub* **where** $\forall\, x{\in}\emptyset\ Q\ b.\ [[lb\ x\ ub]]$
using *events*(*2*) *get-bds reachable*(*2*,*4*) **by** *blast*
have $\emptyset\ Q\ a \subseteq Q$ $\emptyset\ Q\ b \subseteq Q$
by (*simp add*: *subsetI unreach-on-path*)+
moreover have $1 < card\ (\emptyset\ Q\ a) \lor infinite\ (\emptyset\ Q\ a)$
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 $1 < card\ (\emptyset\ Q\ b) \lor infinite\ (\emptyset\ Q\ b)$
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** $A{=}\emptyset\ Q\ a$ **and** $B{=}\emptyset\ Q\ b$]
using *get-bds assms* ‹$\forall\, x{\in}\emptyset\ Q\ a.\ [[la\ x\ ua]]$› ‹$\forall\, x{\in}\emptyset\ Q\ b.\ [[lb\ x\ ub]]$›
by *blast*
qed

then obtain *y z* **where** $y{\in}Q$ $z{\in}Q$ $(\forall\, x{\in}(\emptyset\ Q\ a)\cup(\emptyset\ Q\ b).\ [[y\ x\ z]])$
by *blast*
show *?thesis*
proof (*rule bexI*)+
show $y{\in}Q$ **by** (*simp add*: ‹$y \in Q$›)
show $z{\in}Q$ **by** (*simp add*: ‹$z \in Q$›)
show $(\forall\, x{\in}\emptyset\ Q\ a.\ [[z\ x\ y]]) \land (\forall\, x{\in}\emptyset\ Q\ b.\ [[z\ x\ y]])$
by (*simp add*: ‹$\forall\, x{\in}\emptyset\ Q\ a \cup \emptyset\ Q\ b.\ [[y\ x\ z]]$› *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 $\exists\, e{\in}Q.\ \exists\, ae{\in}\mathcal{P}.\ \exists\, be{\in}\mathcal{P}.\ path\ ae\ a\ e \land path\ be\ b\ e \land [[c\ d\ e]]$
proof −
obtain *y z* **where** *bounds-yz*: $(\forall\, x{\in}\emptyset\ Q\ a.\ [[z\ x\ y]]) \land (\forall\, x{\in}\emptyset\ Q\ b.\ [[z\ x\ y]])$
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 $y{\notin}(\emptyset\ Q\ a)\cup(\emptyset\ Q\ b)$ $z{\notin}(\emptyset\ Q\ a)\cup(\emptyset\ Q\ b)$
apply (*meson Un-iff* ‹$(\forall\, x{\in}\emptyset\ Q\ a.\ [[z\ x\ y]]) \land (\forall\, x{\in}\emptyset\ Q\ b.\ [[z\ x\ y]])$› *abc-abc-neq*)+

done
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 $a{\notin}Q\ \exists\, P{\in}\mathcal{P}.\ \exists\, q{\in}Q.\ path\ P\ a\ q\ y{\notin}(\emptyset\ Q\ a)\ y{\in}Q$
for $a\ y$
apply (*rule ccontr*) **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 $y{\notin}(\emptyset\ Q\ a)\ y{\notin}(\emptyset\ Q\ b)$
using ‹$y \notin \emptyset\ Q\ a \cup \emptyset\ Q\ b$› **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 $z{\notin}(\emptyset\ Q\ a)\ z{\notin}(\emptyset\ Q\ b)$
using ‹$z \notin \emptyset\ Q\ a \cup \emptyset\ Q\ b$› **apply** *blast+* **done**
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*› ‹$z \in Q$› **by** *blast*
thus *?thesis* **by** *blast*
next
assume $([\![y\ c\ d]] \vee [[c\ y\ d]\!]) \wedge ([\![z\ c\ d]] \vee [[c\ z\ d]\!])$
have $\exists\, e.\ [[c\ d\ e]]$
using *prolong-betw*
using *events*(*3−5*) *path-Q* **by** *blast*
then obtain *e* **where** $[[c\ d\ e]]$ **by** *auto*
have $\neg[[y\ e\ z]]$
proof (*rule notI*)

assume $[[y\ e\ z]]$
consider $([\![y\ c\ d]] \wedge [\![z\ c\ d]])\ |\ ([\![y\ c\ d]] \wedge [[c\ z\ d]\!])\ |$
$([[c\ y\ d]\!] \wedge [\![z\ c\ d]])\ |\ ([[c\ y\ d]\!] \wedge [[c\ z\ d]\!])$
using ‹$([\![y\ c\ d]] \vee [[c\ y\ d]\!]) \wedge ([\![z\ c\ d]] \vee [[c\ z\ d]\!])$› **by** *linarith*
thus *False*

apply (*cases*)
using ‹$[[y\ e\ z]]$› ‹$[[c\ d\ e]]$› *abc-only-cba betw4-weak betw4-strong* **apply** *metis*
using ‹$[[c\ d\ e]]$› ‹$[[y\ e\ z]]$› *abc-acd-abd abc-only-cba(2) abc-sym* **apply** *blast*
using ‹$[[c\ d\ e]]$› ‹$[[y\ e\ z]]$› *abc-acd-abd abc-only-cba(2,3) abc-sym* **apply** *blast*
using ‹$[[c\ d\ e]]$› ‹$[[y\ e\ z]]$› *abc-acd-bcd abc-only-cba(2)* **by** *blast*
qed
have $e \in Q$
using ‹$[[c\ d\ e]]$› *betw-c-in-path events(3−5) path-Q* **by** *blast*
have $e \notin \emptyset\ Q\ a$ $e \notin \emptyset\ Q\ b$
using *bounds-yz* ‹$\neg\ [[y\ e\ z]]$› *abc-sym* **apply** *blast+* **done**
hence *ex-aebe*: $\exists\, ae\ be.\ path\ ae\ a\ e \wedge path\ be\ b\ e$
using ‹$e \in Q$› *events(1,2) in-path-event path-Q reachable(1,2) unreachable-bounded-path-only*
by *metis*
thus *?thesis*
using ‹$[[c\ d\ e]]$› ‹$e \in Q$› **by** *blast*
qed
qed

theorem *second-existence-thm-3*:

assumes *paths*: $Q \in \mathcal{P}$ $R \in \mathcal{P}$ $Q \neq R$
and *events*: $x \in Q$ $x \in R$ $a \in R$ $a \neq x$ $b \notin Q$
and *reachable*: $\exists\, P \in \mathcal{P}.\ \exists\, q \in Q.\ path\ P\ b\ q$
shows $\exists\, e \in \mathcal{E}.\ \exists\, ae \in \mathcal{P}.\ \exists\, be \in \mathcal{P}.\ path\ ae\ a\ e \wedge path\ be\ b\ e \wedge (\forall\, y \in \emptyset\ Q\ a.\ [[x\ y\ e]])$
proof −
have $a \notin Q$
using *events(1−4) paths eq-paths* **by** *blast*
hence $\emptyset\ Q\ a \neq \{\}$
by (*metis events(3) ex-in-conv in-path-event paths(1,2) two-in-unreach*)

then obtain d **where** $d \in \emptyset\ Q\ a$
by *blast*
have $x \neq d$
using ‹$d \in \emptyset\ Q\ a$› *cross-in-reachable events(1) events(2) events(3) paths(2)* **by** *auto*
have $d \in Q$
using ‹$d \in \emptyset\ Q\ a$› *unreach-on-path* **by** *blast*

have $\exists\, e \in Q.\ \exists\, ae\ be.\ [[x\ d\ e]] \wedge path\ ae\ a\ e \wedge path\ be\ b\ e$
using *second-existence-thm-2* [**where** $c=x$ **and** $Q=Q$ **and** $a=a$ **and** $b=b$ **and** $d=d$]
using ‹$a \notin Q$› ‹$d \in Q$› ‹$x \neq d$› *events(1−3,5) paths(1,2) reachable* **by** *blast*
then obtain $e\ ae\ be$ **where** *conds*: $[[x\ d\ e]] \wedge path\ ae\ a\ e \wedge path\ be\ b\ e$ **by** *blast*
have $\forall\, y \in (\emptyset\ Q\ a).\ [[x\ y\ e]]$
proof
fix y **assume** $y \in (\emptyset\ Q\ a)$
hence $y \in Q$

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* ∈ ∅ *Q 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* ∈ ∅ *Q a*›)
next
assume [[*y x e*]]
hence [[*y x d*]]
using *abd-bcd-abc conds* **by** *blast*
hence *x*∈(∅ *Q 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*∈∅ *Q a*› ‹*y*∈∅ *Q 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*∈(∅ *Q a*)
using *unreach-connected* [**where** *Q*=*Q* **and** Q_x=*y* **and** Q_y=*e* **and** Q_z=*d* **and** *b*=*a*]
using ‹*a* ∉ *Q*› ‹*d* ∈ ∅ *Q a*› ‹*y* ∈ ∅ *Q 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

40 Theorem 11 - with path density assumed

locale *MinkowskiDense* = *MinkowskiSpacetime* +
  assumes *path-dense*: *path ab a b* $\Longrightarrow \exists x.$ [[*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 ordering case).

lemma *segment-nonempty*:
  assumes *path ab a b*
  obtains *x* **where** $x \in$ *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*[*a..b..c*]*Q*]
   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*
   by (*meson ch-by-ord-def f-def fin-long-chain-def long-ch-card-ge3*)
  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 [[(f i) (f j) (f (j+1))]]
          using asm(2) assms fin-long-chain-def order-finite-chain ‹?N≥3›
       by (metis (no-types, lifting) One-nat-def Suc-diff-Suc Suc-less-eq add.commute
             add-leD2 atLeastAtMost-iff card.remove card-Diff-singleton less-Suc-eq-le
               less-add-one numeral-2-eq-2 numeral-3-eq-3 plus-1-eq-Suc)
        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 fin-long-chain-def long-ch-by-ord-def ordering-def
          by blast
        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[a..b]Q]
    fixes P1 defines P1-def: P1 ≡ prolongation b a
    fixes P2 defines  P2-def: P2 ≡ prolongation a b
    fixes S defines S-def: S ≡ (if card Q=2 then {segment a b} else {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 apply fastforce+ done
  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[a..c..b]Q]
      using assms ch-by-ord-def fin-chain-def short-ch-card-2 ‹2 ≤ card Q› ‹card Q
≠ 2›
      by force
    show ?thesis
      using number-of-segments [OF assms(1,2) ‹[f[a..c..b]Q]›]
      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.