The Poincaré-Bendixson Theorem

Fabian Immler and Yong Kiam Tan

March 17, 2025

Contents

1 Additions to HOL-Analysis

```
theory Analysis-Misc
  imports
    Ordinary-Differential-Equations.ODE-Analysis
begin
```

1.1 Unsorted Lemmas (TODO: sort!)

```
lemma uminus-uminus-image: uminus ‘ uminus ‘ S = S
  for S::'r::ab-group-add set
  by (auto simp: image-image)

lemma in-uminus-image-iff[simp]: x ∈ uminus ‘ S ⟷ − x ∈ S
  for S::'r::ab-group-add set
  by force

lemma closed-subsegmentI:
  w + t *R (z − w) ∈ {x−−y}
  if w ∈ {x −− y} z ∈ {x −− y} and t: 0 ≤ t t≤ 1
proof −
  from that obtain u v where
    w-def: w = (1 − u) *R x + u *R y and u: 0 ≤ u u ≤ 1
    and z-def: z = (1 − v) *R x + v *R y and v: 0 ≤ v v ≤ 1
    by (auto simp: in-segment)
  have w + t *R (z − w) =
    (1 − (u − t * (u − v))) *R x + (u − t * (u − v)) *R y
    by (simp add: algebra-simps w-def z-def)
  also have ... ∈ {x −− y}
    unfolding closed-segment-image-interval
    apply (rule imageI)
    using t u v
    apply auto
      apply (metis (full-types) diff-0-right diff-left-mono linear mult-left-le-one-le
mult-nonneg-nonpos order.trans)
   by (smt mult-left-le-one-le mult-nonneg-nonneg vector-space-over-itself.scale-right-diff-distrib)
  finally show ?thesis .
qed
```

lemma *tendsto-minus-cancel-right*: $((\lambda x.\ -g\ x) \longrightarrow l)\ F \longleftrightarrow (g \longrightarrow -l)\ F$
— cf $(?f \longrightarrow -\ ?y)\ ?F = ((\lambda x.\ -\ ?f\ x) \longrightarrow ?y)\ ?F$
for *g*::- ⇒ ′*b*::*topological-group-add*
by (*simp add*: *tendsto-minus-cancel-left*)

lemma *tendsto-nhds-continuousI*: $(f \longrightarrow l)\ (nhds\ x)$ **if** $(f \longrightarrow l)\ (at\ x)\ f\ x = l$
— TODO: the assumption is continuity of f at x
proof (*rule topological-tendstoI*)
fix *S*::′*b set* **assume** *open S l* ∈ *S*
from *topological-tendstoD*[*OF that*(*1*) *this*]
have $\forall_F$ *x in at x*. *f x* ∈ *S* **.**
then show $\forall_F$ *x in nhds x*. *f x* ∈ *S*
unfolding *eventually-at-filter*
by *eventually-elim* (*auto simp*: *that* ‹*l* ∈ *S*›)
qed

lemma *inj-composeD*:
assumes *inj* (λ*x*. *g* (*t x*))
shows *inj t*
using *assms*
by (*auto simp*: *inj-def*)

lemma *compact-sequentialE*:
fixes *S T*::′*a*::*first-countable-topology set*
assumes *compact S*
assumes *infinite T*
assumes *T* ⊆ *S*
obtains *t*::*nat* ⇒ ′*a* **and** *l*::′*a*
where $\bigwedge n.\ t\ n \in T\ \bigwedge n.\ t\ n \neq l\ t \longrightarrow l\ l \in S$
proof −
from *Heine-Borel-imp-Bolzano-Weierstrass*[*OF assms*]
obtain *l* **where** *l* ∈ *S l islimpt T* **by** *metis*
then obtain *t* **where** $t\ n \in T\ t\ n \neq l\ t \longrightarrow l\ l \in S$ **for** *n* **unfolding** *islimpt-sequential*
by *auto*
then show *?thesis* **..**
qed

lemma *infinite-countable-subsetE*:
fixes *S*::′*a set*
assumes *infinite S*
obtains *g*::*nat*⇒′*a* **where** *inj g range g* ⊆ *S*
using *assms*
by *atomize-elim* (*simp add*: *infinite-countable-subset*)

lemma *real-quad-ge*: $2 * (an * bn) \leq an * an + bn * bn$ **for** *an bn*::*real*
by (*sos* (((*A*<*0* ∗ *R*<*1*) + (*R*<*1* ∗ (*R*<*1* ∗ [*an* + ~*1*∗*bn*]^*2*)))))

lemma *inner-quad-ge*: $2 * (a \cdot b) \le a \cdot a + b \cdot b$
  for a b::$'a$::*euclidean-space*— generalize?
proof −
  show *?thesis*
    by (*subst* (*1 2 3*) *euclidean-inner*)
      (*auto simp add*: *sum.distrib*[*symmetric*] *sum-distrib-left intro*!: *sum-mono real-quad-ge*)
qed

lemma *inner-quad-gt*: $2 * (a \cdot b) < a \cdot a + b \cdot b$
  if $a \neq b$
  for a b::$'a$::*euclidean-space*— generalize?
proof −
  from *that* **obtain** i **where** $i \in Basis$ $a \cdot i \neq b \cdot i$
    by (*auto simp*: *euclidean-eq-iff*[**where** $'a='a$])
  then have $2 * (a \cdot i * (b \cdot i)) < a \cdot i * (a \cdot i) + b \cdot i * (b \cdot i)$
    using *sum-sqs-eq*[*of* $a{\cdot}i$ $b{\cdot}i$]
    by (*auto intro*!: *le-neq-trans real-quad-ge*)
  then show *?thesis*
    by (*subst* (*1 2 3*) *euclidean-inner*)
      (*auto simp add*: ‹$i \in Basis$› *sum.distrib*[*symmetric*] *sum-distrib-left*
        intro!: *sum-strict-mono-ex1 real-quad-ge*)
qed

lemma *closed-segment-line-hyperplanes*:
  $\{a -- b\} = range\ (\lambda u.\ a + u *_R (b - a)) \cap \{x.\ a \cdot (b - a) \le x \cdot (b - a) \wedge x \cdot (b - a) \le b \cdot (b - a)\}$
  if $a \neq b$
  for a b::$'a$::*euclidean-space*
proof *safe*
  fix x **assume** x: $x \in \{a{-}{-}b\}$
  then obtain u **where** u: $0 \le u$ $u \le 1$ **and** *x-eq*: $x = a + u *_R (b - a)$
    by (*auto simp add*: *in-segment algebra-simps*)
  show $x \in range\ (\lambda u.\ a + u *_R (b - a))$ **using** *x-eq* **by** *auto*
  have $2 * (a \cdot b) \le a \cdot a + b \cdot b$
    by (*rule inner-quad-ge*)
  then have $u * (2 * (a \cdot b) - a \cdot a - b \cdot b) \le 0$
    $0 \le (1 - u) * (a \cdot a + b \cdot b - a \cdot b * 2)$
    by (*simp-all add*: *mult-le-0-iff u*)
  then show $a \cdot (b - a) \le x \cdot (b - a)$ $x \cdot (b - a) \le b \cdot (b - a)$
    by (*auto simp*: *x-eq algebra-simps power2-eq-square inner-commute*)
next
  fix u **assume**
    $a \cdot (b - a) \le (a + u *_R (b - a)) \cdot (b - a)$
    $(a + u *_R (b - a)) \cdot (b - a) \le b \cdot (b - a)$
  then have $0 \le u * ((b - a) \cdot (b - a))$ $0 \le (1 - u) * ((b - a) \cdot (b - a))$
    by (*auto simp*: *algebra-simps*)
  then have $0 \le u$ $u \le 1$
    using *inner-ge-zero*[*of* $(b - a)$] *that*

by (*auto simp add*: *zero-le-mult-iff*)
then show $a + u *_R (b - a) \in \{a{-}{-}b\}$
by (*auto simp*: *in-segment algebra-simps*)
qed

lemma *open-segment-line-hyperplanes*:
$\{a <{-}{-}< b\} = range\ (\lambda u.\ a + u *_R (b - a)) \cap \{x.\ a \cdot (b - a) < x \cdot (b - a) \wedge x \cdot (b - a) < b \cdot (b - a)\}$
if $a \neq b$
for $a\ b{::}'a{::}euclidean\text{-}space$
proof *safe*
fix x **assume** x: $x \in \{a{<}{-}{-}{<}b\}$
then obtain u **where** u: $0 < u\ u < 1$ **and** *x-eq*: $x = a + u *_R (b - a)$
by (*auto simp add*: *in-segment algebra-simps*)
show $x \in range\ (\lambda u.\ a + u *_R (b - a))$ **using** *x-eq* **by** *auto*
have $2 * (a \cdot b) < a \cdot a + b \cdot b$ **using** *that*
by (*rule inner-quad-gt*)
then have $u * (2 * (a \cdot b) - a \cdot a - b \cdot b) < 0$
$0 < (1 - u) * (a \cdot a + b \cdot b - a \cdot b * 2)$
by (*simp-all add*: *mult-less-0-iff u*)
then show $a \cdot (b - a) < x \cdot (b - a)\ x \cdot (b - a) < b \cdot (b - a)$
by (*auto simp*: *x-eq algebra-simps power2-eq-square inner-commute*)
next
fix u **assume**
$a \cdot (b - a) < (a + u *_R (b - a)) \cdot (b - a)$
$(a + u *_R (b - a)) \cdot (b - a) < b \cdot (b - a)$
then have $0 < u * ((b - a) \cdot (b - a))\ 0 < (1 - u) * ((b - a) \cdot (b - a))$
by (*auto simp*: *algebra-simps*)
then have $0 < u\ u < 1$
using *inner-ge-zero*[*of* $(b - a)$] *that*
by (*auto simp add*: *zero-less-mult-iff*)
then show $a + u *_R (b - a) \in \{a{<}{-}{-}{<}b\}$
by (*auto simp*: *in-segment algebra-simps that*)
qed

lemma *at-within-interior*: *NO-MATCH UNIV S* $\Longrightarrow x \in$ *interior S* $\Longrightarrow$ *at x within S* $=$ *at x*
by (*auto intro*: *at-within-interior*)

lemma *tendsto-at-topI*:
$(f \longrightarrow l)$ *at-top* **if** $\bigwedge e.\ 0 < e \Longrightarrow \exists x0.\ \forall x \geq x0.\ dist\ (f\ x)\ l < e$
for $f{::}'a{::}linorder\text{-}topology \Rightarrow 'b{::}metric\text{-}space$
using *that*
apply (*intro tendstoI*)
unfolding *eventually-at-top-linorder*
by *auto*

lemma *tendsto-at-topE*:
fixes $f{::}'a{::}linorder\text{-}topology \Rightarrow 'b{::}metric\text{-}space$

assumes (f ⟶ l) at-top
assumes e > 0
obtains x0 **where** ⋀x. x ≥ x0 ⟹ dist (f x) l < e
proof −
from assms(1)[THEN tendstoD, OF assms(2)]
have ∀$_F$ x in at-top. dist (f x) l < e **.**
then show ?thesis
unfolding eventually-at-top-linorder
by (auto intro: that)
qed
lemma tendsto-at-top-iff: (f ⟶ l) at-top ⟷ (∀ e>0. ∃ x0. ∀ x≥x0. dist (f x) l < e)
for f::'a::linorder-topology ⇒ 'b::metric-space
by (auto intro!: tendsto-at-topI elim!: tendsto-at-topE)

lemma tendsto-at-top-eq-left:
fixes f g::'a::linorder-topology ⇒ 'b::metric-space
assumes (f ⟶ l) at-top
assumes ⋀x. x ≥ x0 ⟹ f x = g x
shows (g ⟶ l) at-top
unfolding tendsto-at-top-iff
by (metis (no-types, opaque-lifting) assms(1) assms(2) linear order-trans tendsto-at-topE)

lemma lim-divide-n: (λx. e / real x) ⟶ 0
proof −
have (λx. e ∗ inverse (real x)) ⟶ 0
by (auto intro: tendsto-eq-intros lim-inverse-n)
then show ?thesis **by** (simp add: inverse-eq-divide)
qed

definition at-top-within :: ('a::order) set ⇒ 'a filter
where at-top-within s = (INF k ∈ s. principal ({k ..} ∩ s))

lemma at-top-within-at-top[simp]:
shows at-top-within UNIV = at-top
unfolding at-top-within-def at-top-def
by (auto)

lemma at-top-within-empty[simp]:
shows at-top-within {} = top
unfolding at-top-within-def
by (auto)

definition nhds-set X = (INF S∈{S. open S ∧ X ⊆ S}. principal S)

lemma eventually-nhds-set:
(∀$_F$ x in nhds-set X. P x) ⟷ (∃ S. open S ∧ X ⊆ S ∧ (∀ x∈S. P x))
unfolding nhds-set-def **by** (subst eventually-INF-base) (auto simp: eventually-principal)

term *filterlim f* (*nhds-set* (*frontier X*)) *F* — f tends to the boundary of X?

somewhat inspired by *?l islimpt range ?f* $\Longrightarrow \exists\, r.$ *strict-mono r* $\wedge$ (*?f* $\circ$ *r*) $\longrightarrow$ *?l* and its dependencies. The class constraints seem somewhat arbitrary, perhaps this can be generalized in some way.

lemma *limpt-closed-imp-exploding-subsequence*:— TODO: improve name?!
fixes *f*::*'a*::{*heine-borel*,*real-normed-vector*} $\Rightarrow$ *'b*::{*first-countable-topology*, *t2-space*}
assumes *cont*[*THEN continuous-on-compose2*, *continuous-intros*]: *continuous-on T f*
assumes *closed*: *closed T*
assumes *bound*: $\bigwedge t.\ t \in T \Longrightarrow f\ t \neq l$
assumes *limpt*: *l islimpt* (*f* ' *T*)
obtains *s* **where**
($f \circ s$) $\longrightarrow l$
$\bigwedge i.\ s\ i \in T$
$\bigwedge C.$ *compact C* $\Longrightarrow C \subseteq T \Longrightarrow \forall_F$ *i in sequentially.* $s\ i \notin C$
proof −
from *countable-basis-at-decseq*[*of l*]
obtain *A* **where** *A*: $\bigwedge i.$ *open* (*A i*) $\bigwedge i.\ l \in A\ i$
and *evA*: $\bigwedge S.$ *open S* $\Longrightarrow l \in S \Longrightarrow$ *eventually* ($\lambda i.\ A\ i \subseteq S$) *sequentially*
by *blast*

from *closed-Union-compact-subsets*[*OF closed*]
obtain *C*
where *C*: ($\bigwedge n.$ *compact* (*C n*)) ($\bigwedge n.\ C\ n \subseteq T$) ($\bigwedge n.\ C\ n \subseteq C$ (*Suc n*)) $\bigcup$ (*range C*) $= T$
and *evC*: ($\bigwedge K.$ *compact K* $\Longrightarrow K \subseteq T \Longrightarrow \forall_F$ *i in sequentially.* $K \subseteq C\ i$)
by (*metis eventually-sequentially*)

have *AC*: $l \in A\ i - f$ ' $C\ i$ *open* ($A\ i - f$ ' $C\ i$) **for** *i*
using *C bound*
by (*fastforce intro*!: *open-Diff A compact-imp-closed compact-continuous-image continuous-intros*)+

from *islimptE*[*OF limpt AC*] **have** $\exists t{\in}T.\ f\ t \in A\ i - f$ ' $C\ i \wedge f\ t \neq l$ **for** *i* **by** *blast*
then obtain *t* **where** *t*: $\bigwedge i.\ t\ i \in T$ $\bigwedge i.\ f\ (t\ i) \in A\ i - f$ ' $C\ i$ $\bigwedge i.\ f\ (t\ i) \neq l$
by *metis*

have (*f o t*) $\longrightarrow l$
using *t*
by (*auto intro*!: *topological-tendstoI dest*!: *evA elim*!: *eventually-mono*)
moreover
have $\bigwedge i.\ t\ i \in T$ **by** *fact*
moreover
have $\forall_F$ *i in sequentially.* $t\ i \notin K$ **if** *compact K* $K \subseteq T$ **for** *K*
using *evC*[*OF that*]
by *eventually-elim* (*use t* **in** *auto*)

ultimately show *?thesis* **..**
qed

lemma *Inf-islimpt*: *bdd-below* $S \Longrightarrow$ *Inf* $S \notin S \Longrightarrow S \neq \{\} \Longrightarrow$ *Inf* S *islimpt* S **for** S::*real set*
by (*auto simp*: *islimpt-in-closure intro*!: *closure-contains-Inf*)

context *linorder*
begin

HOL-analysis doesn't seem to have these, maybe they were never needed. Some variants are around $\{?a..?b\} \cap \{?c..?d\} = \{max\ ?a\ ?c..min\ ?b\ ?d\}$, but with old-style naming conventions. Change to the "modern" I.. convention there?

lemma *Int-Ico*[*simp*]:
shows $\{a..\} \cap \{b..\} = \{max\ a\ b\ ..\}$
by (*auto*)

lemma *Int-Ici-Ico*[*simp*]:
shows $\{a..\} \cap \{b..<c\} = \{max\ a\ b\ ..<c\}$
by *auto*

lemma *Int-Ico-Ici*[*simp*]:
shows $\{a..<c\} \cap \{b..\} = \{max\ a\ b\ ..<c\}$
by *auto*

lemma *subset-Ico-iff*[*simp*]:
$\{a..<b\} \subseteq \{c..<b\} \longleftrightarrow b \leq a \vee c \leq a$
unfolding *atLeastLessThan-def*
by *auto*

lemma *Ico-subset-Ioo-iff*[*simp*]:
$\{a..<b\} \subseteq \{c<..<b\} \longleftrightarrow b \leq a \vee c < a$
unfolding *greaterThanLessThan-def atLeastLessThan-def*
by *auto*

lemma *Icc-Un-Ici*[*simp*]:
shows $\{a..b\} \cup \{b..\} = \{min\ a\ b..\}$
unfolding *atLeastAtMost-def atLeast-def atMost-def min-def*
by *auto*

end

lemma *at-top-within-at-top-unbounded-right*:
fixes a::$'a$::*linorder*
shows *at-top-within* $\{a..\}$ = *at-top*
unfolding *at-top-within-def at-top-def*
apply (*auto intro*!: *INF-eq*)
by (*metis linorder-class.linear linorder-class.max.cobounded1 linorder-class.max.idem*

ord-class.atLeast-iff)

lemma *at-top-within-at-top-unbounded-rightI*:
fixes a::'a::*linorder*
assumes $\{a..\} \subseteq s$
shows *at-top-within* s = *at-top*
unfolding *at-top-within-def at-top-def*
apply (*auto intro*!: *INF-eq*)
apply (*meson Ici-subset-Ioi-iff Ioi-le-Ico assms dual-order.refl dual-order.trans leI*)
by (*metis assms atLeast-iff atLeast-subset-iff inf.cobounded1 linear subsetD*)

lemma *at-top-within-at-top-bounded-right*:
fixes a b::'a::{*dense-order*,*linorder-topology*}
assumes $a < b$
shows *at-top-within* $\{a..<b\}$ = *at-left* b
unfolding *at-top-within-def at-left-eq*[*OF assms*(*1*)]
apply (*auto intro*!: *INF-eq*)
apply (*smt atLeastLessThan-iff greaterThanLessThan-iff le-less lessThan-iff max.absorb1 subset-eq*)
by (*metis assms atLeastLessThan-iff dense linear max.absorb1 not-less order-trans*)

lemma *at-top-within-at-top-bounded-right'*:
fixes a b::'a::{*dense-order*,*linorder-topology*}
assumes $a < b$
shows *at-top-within* $\{..<b\}$ = *at-left* b
unfolding *at-top-within-def at-left-eq*[*OF assms*(*1*)]
apply (*auto intro*!: *INF-eq*)
apply (*meson atLeast-iff greaterThanLessThan-iff le-less lessThan-iff subset-eq*)
by (*metis Ico-subset-Ioo-iff atLeastLessThan-def dense lessThan-iff*)

lemma *eventually-at-top-within-linorder*:
assumes *sn*:$s \neq \{\}$
shows *eventually* P (*at-top-within* s) $\longleftrightarrow$ ($\exists x0$::'a::{*linorder-topology*} $\in s$. $\forall x \geq x0$. $x \in s \longrightarrow P\ x$)
unfolding *at-top-within-def*
apply (*subst eventually-INF-base*)
apply (*auto simp*:*eventually-principal sn*)
by (*metis atLeast-subset-iff inf.coboundedI2 inf-commute linear*)

lemma *tendsto-at-top-withinI*:
fixes f::'a::*linorder-topology* $\Rightarrow$ 'b::*metric-space*
assumes $s \neq \{\}$
assumes $\bigwedge e.\ 0 < e \Longrightarrow \exists x0 \in s.\ \forall x \in \{x0..\} \cap s.\ dist\ (f\ x)\ l < e$
shows ($f \longrightarrow l$) (*at-top-within* s)
apply(*intro tendstoI*)
unfolding *at-top-within-def* **apply** (*subst eventually-INF-base*)
apply (*auto simp*:*eventually-principal assms*)
by (*metis atLeast-subset-iff inf.coboundedI2 inf-commute linear*)

lemma *tendsto-at-top-withinE*:
  fixes $f::'a::linorder\text{-}topology \Rightarrow 'b::metric\text{-}space$
  assumes $s \neq \{\}$
  assumes $(f \longrightarrow l)$ $(at\text{-}top\text{-}within\ s)$
  assumes $e > 0$
  obtains $x0$ **where** $x0 \in s \bigwedge x.\ x \in \{x0..\} \cap s \Longrightarrow dist\ (f\ x)\ l < e$
proof −
  from $assms(2)[THEN\ tendstoD,\ OF\ assms(3)]$
  have $\forall_F\ x\ in\ at\text{-}top\text{-}within\ s.\ dist\ (f\ x)\ l < e$ **.**
  then show *?thesis* **unfolding** $eventually\text{-}at\text{-}top\text{-}within\text{-}linorder[OF\ \langle s \neq \{\}\rangle]$
    by $(auto\ intro:\ that)$
qed

lemma *tendsto-at-top-within-iff*:
  fixes $f::'a::linorder\text{-}topology \Rightarrow 'b::metric\text{-}space$
  assumes $s \neq \{\}$
  shows $(f \longrightarrow l)$ $(at\text{-}top\text{-}within\ s) \longleftrightarrow (\forall\ e>0.\ \exists\ x0 \in s.\ \forall\ x \in \{x0..\} \cap s.\ dist\ (f\ x)\ l < e)$
 by $(auto\ intro!:\ tendsto\text{-}at\text{-}top\text{-}withinI[OF\ \langle s \neq \{\}\rangle]\ elim!:\ tendsto\text{-}at\text{-}top\text{-}withinE[OF\ \langle s \neq \{\}\rangle])$

lemma *filterlim-at-top-at-top-within-bounded-right*:
  fixes $a\ b::'a::\{dense\text{-}order,linorder\text{-}topology\}$
  fixes $f::'a \Rightarrow real$
  assumes $a < b$
  shows $filterlim\ f\ at\text{-}top\ (at\text{-}top\text{-}within\ \{..<b\}) = (f \longrightarrow \infty)\ (at\text{-}left\ b)$
  unfolding *filterlim-at-top-dense*
    $at\text{-}top\text{-}within\text{-}at\text{-}top\text{-}bounded\text{-}right'[OF\ assms(1)]$
    $eventually\text{-}at\text{-}left[OF\ assms(1)]$
    tendsto-PInfty
  by *auto*

Extract a sequence (going to infinity) bounded away from l

lemma *not-tendsto-frequentlyE*:
  assumes $\neg((f \longrightarrow l)\ F)$
  obtains S **where** $open\ S\ l \in S\ \exists_F\ x\ in\ F.\ f\ x \notin S$
  using *assms*
  by $(auto\ simp:\ tendsto\text{-}def\ not\text{-}eventually)$

lemma *not-tendsto-frequently-metricE*:
  assumes $\neg((f \longrightarrow l)\ F)$
  obtains e **where** $e > 0\ \exists_F\ x\ in\ F.\ e \leq dist\ (f\ x)\ l$
  using *assms*
  by $(auto\ simp:\ tendsto\text{-}iff\ not\text{-}eventually\ not\text{-}less)$

lemma *eventually-frequently-conj*: $frequently\ P\ F \Longrightarrow eventually\ Q\ F \Longrightarrow frequently\ (\lambda x.\ P\ x \wedge Q\ x)\ F$
  unfolding *frequently-def*

```
  apply (erule contrapos-nn)
  subgoal premises prems
    using prems by eventually-elim auto
  done

lemma frequently-at-top:
  (∃F t in at-top. P t) ⟷ (∀t0. ∃t>t0. P t)
  for P::'a::{linorder,no-top}⇒bool
  by (auto simp: frequently-def eventually-at-top-dense)

lemma frequently-at-topE:
  fixes P::nat ⇒ 'a::{linorder,no-top}⇒-
  assumes freq[rule-format]: ∀n. ∃F a in at-top. P n a
  obtains s::nat⇒'a
  where ⋀i. P i (s i) strict-mono s
proof –
  have ∃f. ∀n. P n (f n) ∧ f n < f (Suc n)
  proof (rule dependent-nat-choice)
    from frequently-ex[OF freq[of 0]] show ∃x. P 0 x .
    fix x n assume P n x
    from freq[unfolded frequently-at-top, rule-format, of x Suc n]
    obtain y where P (Suc n) y y > x by auto
    then show ∃y. P (Suc n) y ∧ x < y
      by auto
  qed
  then obtain s where ⋀i. P i (s i) strict-mono s
    unfolding strict-mono-Suc-iff by auto
  then show ?thesis ..
qed

lemma frequently-at-topE':
  fixes P::nat ⇒ 'a::{linorder,no-top}⇒-
  assumes freq[rule-format]: ∀n. ∃F a in at-top. P n a
    and g: filterlim g at-top sequentially
  obtains s::nat⇒'a
  where ⋀i. P i (s i) strict-mono s ⋀n. g n ≤ s n
proof –
  have ∀n. ∃F a in at-top. P n a ∧ g n ≤ a
    using freq
    by (auto intro!: eventually-frequently-conj)
  from frequently-at-topE[OF this] obtain s where ⋀i. P i (s i) strict-mono s
⋀n. g n ≤ s n
    by metis
  then show ?thesis ..
qed

lemma frequently-at-top-at-topE:
  fixes P::nat ⇒ 'a::{linorder,no-top}⇒- and g::nat⇒'a
  assumes ∀n. ∃F a in at-top. P n a filterlim g at-top sequentially
```

obtains s::*nat*⇒′*a*
where $\bigwedge i.\ P\ i\ (s\ i)$ *filterlim s at-top sequentially*
proof −
from *frequently-at-topE*′[*OF assms*]
obtain s **where** s: $(\bigwedge i.\ P\ i\ (s\ i))$ *strict-mono s* $(\bigwedge n.\ g\ n \leq s\ n)$ **by** *blast*
have *s-at-top*: *filterlim s at-top sequentially*
by (*rule filterlim-at-top-mono*) (*use assms s* **in** *auto*)
with *s*(*1*) **show** *?thesis* **..**
qed

lemma *not-tendsto-convergent-seq*:
fixes f::*real* ⇒ ′*a*::*metric-space*
assumes X: *compact* (X::′*a set*)
assumes *im*: $\bigwedge x.\ x \geq 0 \Longrightarrow f\ x \in X$
assumes *nl*: $\neg\ ((f \longrightarrow (l{::}'a))\ at\text{-}top)$
obtains $s\ k$ **where**
$k \in X$ $k \neq l$ $(f \circ s) \longrightarrow k$ *strict-mono s* $\forall n.\ s\ n \geq n$
proof −
from *not-tendsto-frequentlyE*[*OF nl*]
obtain S **where** *open S* $l \in S$ $\exists_F\ x$ *in at-top*. $f\ x \notin S$ **.**
have $\forall n.\ \exists_F\ x$ *in at-top*. $f\ x \notin S \wedge$ *real* $n \leq x$
apply (*rule allI*)
apply (*rule eventually-frequently-conj*)
apply *fact*
by (*rule eventually-ge-at-top*)
from *frequently-at-topE*[*OF this*]
obtain s **where** $\bigwedge i.\ f\ (s\ i) \notin S$ **and** *s*: *strict-mono s* **and** *s-ge*: $(\bigwedge i.$ *real* $i \leq s\ i)$ **by** *metis*
then have $0 \leq s\ i$ **for** i **using** *dual-order.trans of-nat-0-le-iff* **by** *blast*
then have $\forall n.\ (f \circ s)\ n \in X$ **using** *im* **by** *auto*
from X[*unfolded compact-def*, *THEN spec*, *THEN mp*, *OF this*]
obtain $k\ r$ **where** k: $k \in X$ **and** r: *strict-mono r* **and** *kLim*: $(f \circ s \circ r) \longrightarrow k$ **by** *metis*
have $k \in X - S$
by (*rule Lim-in-closed-set*[*of X − S*, *OF - - - kLim*])
(*auto simp*: *im* ‹$0 \leq s$ -› ‹$\bigwedge i.\ f\ (s\ i) \notin S$› *intro*!: ‹*open S*› *X intro*: *compact-imp-closed*)

note k
moreover have $k \neq l$ **using** ‹$k \in X - S$› ‹$l \in S$› **by** *auto*
moreover have $(f \circ (s \circ r)) \longrightarrow k$ **using** *kLim* **by** (*simp add*: *o-assoc*)
moreover have *strict-mono* $(s \circ r)$ **using** *s r* **by** (*rule strict-mono-o*)
moreover have $\forall n.\ (s \circ r)\ n \geq n$ **using** *s-ge r*
by (*metis comp-apply dual-order.trans of-nat-le-iff seq-suble*)
ultimately show *?thesis* **..**
qed

lemma *harmonic-bound*:

```
  shows 1 / 2 ^(Suc n) < 1 / real (Suc n)
proof (induction n)
  case 0
  then show ?case by auto
next
  case (Suc n)
  then show ?case
    by (smt frac-less2 of-nat-0-less-iff of-nat-less-two-power zero-less-Suc)
qed

lemma INF-bounded-imp-convergent-seq:
  fixes f::real ⇒ real
  assumes cont: continuous-on {a..} f
  assumes bound: ⋀t. t ≥ a ⟹ f t > l
  assumes inf: (INF t∈{a..}. f t) = l
  obtains s where
    (f ∘ s) ⟶ l
    ⋀i. s i ∈ {a..}
    filterlim s at-top sequentially
proof −
  have bound′: t ∈ {a..} ⟹ f t ≠ l for t using bound[of t] by auto
  have limpt: l islimpt f ‘ {a..}
  proof −
    have Inf (f ‘ {a..}) islimpt f ‘ {a..}
      by (rule Inf-islimpt) (auto simp: inf intro!: bdd-belowI2[where m=l] dest:
bound)
    then show ?thesis by (simp add: inf)
  qed
  from limpt-closed-imp-exploding-subsequence[OF cont closed-atLeast bound′ limpt]
  obtain s where s: (f ∘ s) ⟶ l
    ⋀i. s i ∈ {a..}
    compact C ⟹ C ⊆ {a..} ⟹ ∀F i in sequentially. s i ∉ C for C
    by metis
  have ∀F i in sequentially. s i ≥ n for n
    using s(3)[of {a..n}] s(2)
    by (auto elim!: eventually-mono)
  then have filterlim s at-top sequentially
    unfolding filterlim-at-top
    by auto
  from s(1) s(2) this
  show ?thesis ..
qed


lemma filterlim-at-top-strict-mono:
  fixes s :: - ⇒ ′a::linorder
  fixes r :: nat ⇒ -
  assumes strict-mono s
  assumes strict-mono r
```

assumes *filterlim s at-top F*
shows *filterlim* $(s \circ r)$ *at-top F*
apply (*rule filterlim-at-top-mono*[*OF assms*(*3*)])
by (*simp add*: *assms*(*1*) *assms*(*2*) *seq-suble strict-mono-leD*)

lemma *LIMSEQ-lb*:
assumes *fl*: $s \longrightarrow (l::real)$
assumes *u*: $l < u$
shows $\exists n0. \forall n \geq n0. s\ n < u$
proof −
from *fl* **have** $\exists no>0. \forall n \geq no.$ *dist* $(s\ n)\ l < u-l$ **unfolding** *LIMSEQ-iff-nz*
using *u*
by *simp*
thus *?thesis* **using** *dist-real-def* **by** *fastforce*
qed

lemma *filterlim-at-top-choose-lower*:
assumes *filterlim s at-top sequentially*
assumes $(f \circ s) \longrightarrow l$
obtains *t* **where**
filterlim t at-top sequentially
$(f \circ t) \longrightarrow l$
$\forall n.\ t\ n \geq (b::real)$
proof −
obtain *k* **where** *k*: $\forall n \geq k.\ s\ n \geq b$ **using** *assms*(*1*)
unfolding *filterlim-at-top eventually-sequentially* **by** *blast*
define *t* **where** $t = (\lambda n.\ s\ (n+k))$
then have $\forall n.\ t\ n \geq b$ **using** *k* **by** *simp*
have *filterlim t at-top sequentially* **using** *assms*(*1*)
unfolding *filterlim-at-top eventually-sequentially t-def*
by (*metis* (*full-types*) *add.commute trans-le-add2*)
from *LIMSEQ-ignore-initial-segment*[*OF assms*(*2*), *of k*]
have $(\lambda n.\ (f \circ s)\ (n + k)) \longrightarrow l$ **.**
then have $(f \circ t) \longrightarrow l$ **unfolding** *t-def o-def* **by** *simp*
show *?thesis*
using ‹$(f \circ t) \longrightarrow l$› ‹$\forall n.\ b \leq t\ n$› ‹*filterlim t at-top sequentially*› *that* **by**
blast
qed

lemma *frequently-at-top-realE*:
fixes P::*nat* ⇒ *real* ⇒ *bool*
assumes $\forall n.\ \exists_F$ *t in at-top. P n t*
obtains *s*::*nat*⇒*real*
where $\bigwedge i.\ P\ i\ (s\ i)$ *filterlim s at-top at-top*
by (*metis assms frequently-at-top-at-topE*[*OF - filterlim-real-sequentially*])

lemma *approachable-sequenceE*:
fixes *f*::*real* ⇒ *'a*::*metric-space*

assumes $\bigwedge t\ e.\ 0 \leq t \Longrightarrow 0 < e \Longrightarrow \exists\, tt \geq t.\ dist\ (f\ tt)\ p < e$
obtains *s* **where** *filterlim s at-top sequentially* $(f \circ s) \longrightarrow p$
proof −
have $\forall\, n.\ \exists_F\ i$ *in at-top. dist* $(f\ i)\ p < 1/real\ (Suc\ n)$
unfolding *frequently-at-top*
apply (*auto*)
subgoal for *n m*
using *assms*[*of max 0* (*m*+*1*) *1*/(*Suc n*)]
by *force*
done
from *frequently-at-top-realE*[*OF this*]
obtain *s* **where** *s*: $\bigwedge i.\ dist\ (f\ (s\ i))\ p < 1\ /\ real\ (Suc\ i)$ *filterlim s at-top sequentially*
by *metis*
note *this*(*2*)
moreover
have (*f o s*) $\longrightarrow$ *p*
proof (*rule tendstoI*)
fix *e*::*real* **assume** $e > 0$
have $\forall_F\ i$ *in sequentially.* $1\ /\ real\ (Suc\ i) < e$
apply (*rule order-tendstoD*[*OF* - ‹$0 < e$›])
apply (*rule real-tendsto-divide-at-top*)
apply (*rule tendsto-intros*)
by (*rule filterlim-compose*[*OF filterlim-real-sequentially filterlim-Suc*])
then show $\forall_F\ x$ *in sequentially. dist* $((f \circ s)\ x)\ p < e$
by *eventually-elim* (*use dual-order.strict-trans s* ‹$e > 0$› **in** *auto*)
qed
ultimately show *?thesis* **..**
qed

lemma *mono-inc-bdd-above-has-limit-at-topI*:
fixes *f*::*real* ⇒ *real*
assumes *mono f*
assumes $\bigwedge x.\ f\ x \leq u$
shows $\exists\, l.\ (f \longrightarrow l)$ *at-top*
proof −
define *l* **where** $l = Sup\ (range\ (\lambda n.\ f\ (real\ n)))$
have *t*:$(\lambda n.\ f\ (real\ n)) \longrightarrow l$ **unfolding** *l-def*
apply (*rule LIMSEQ-incseq-SUP*)
apply (*meson assms*(*2*) *bdd-aboveI2*)
by (*meson assms*(*1*) *mono-def of-nat-mono*)
from *tendsto-at-topI-sequentially-real*[*OF assms*(*1*) *t*]
have $(f \longrightarrow l)$ *at-top* **.**
thus *?thesis* **by** *blast*
qed

lemma *gen-mono-inc-bdd-above-has-limit-at-topI*:
fixes *f*::*real* ⇒ *real*
assumes $\bigwedge x\ y.\ x \geq b \Longrightarrow x \leq y \Longrightarrow f\ x \leq f\ y$

assumes $\bigwedge x.\ x \geq b \Longrightarrow f\ x \leq u$
shows $\exists\, l.\ (f \longrightarrow l)$ *at-top*
proof −
define *ff* **where** *ff* = ($\lambda x.$ *if* $x \geq b$ *then* $f\ x$ *else* $f\ b$)
have *m1*:*mono ff* **unfolding** *ff-def mono-def* **using** *assms*(*1*) **by** *simp*
have *m2*:$\bigwedge x.$ *ff* $x \leq u$ **unfolding** *ff-def* **using** *assms*(*2*) **by** *simp*
from *mono-inc-bdd-above-has-limit-at-topI*[*OF m1 m2*]
obtain *l* **where** (*ff* $\longrightarrow$ *l*) *at-top* **by** *blast*
thus *?thesis*
by (*meson* ‹(*ff* $\longrightarrow$ *l*) *at-top*› *ff-def tendsto-at-top-eq-left*)
qed

lemma *gen-mono-dec-bdd-below-has-limit-at-topI*:
fixes *f*::*real* ⇒ *real*
assumes $\bigwedge x\ y.\ x \geq b \Longrightarrow x \leq y \Longrightarrow f\ x \geq f\ y$
assumes $\bigwedge x.\ x \geq b \Longrightarrow f\ x \geq u$
shows $\exists\, l.\ (f \longrightarrow l)$ *at-top*
proof −
define *ff* **where** *ff* = ($\lambda x.$ *if* $x \geq b$ *then* $f\ x$ *else* $f\ b$)
have *m1*:*mono* (−*ff*) **unfolding** *ff-def mono-def* **using** *assms*(*1*) **by** *simp*
have *m2*:$\bigwedge x.$ (−*ff*) $x \leq -u$ **unfolding** *ff-def* **using** *assms*(*2*) **by** *simp*
from *mono-inc-bdd-above-has-limit-at-topI*[*OF m1 m2*]
obtain *l* **where** (−*ff* $\longrightarrow$ *l*) *at-top* **by** *blast*
then have (*ff* $\longrightarrow$ −*l*) *at-top*
using *tendsto-at-top-eq-left tendsto-minus-cancel-left* **by** *fastforce*
thus *?thesis*
by (*meson* ‹(*ff* $\longrightarrow$ −*l*) *at-top*› *ff-def tendsto-at-top-eq-left*)
qed

lemma *infdist-closed*:
shows *closed* ({*z. infdist z S* $\geq$ *e*})
by (*auto intro*!:*closed-Collect-le simp add*:*continuous-on-infdist*)

lemma *LIMSEQ-norm-0-pow*:
assumes $k > 0$ $b > 1$
assumes $\bigwedge n$::*nat*. *norm* (*s n*) $\leq k\ /\ b\hat{\ }n$
shows $s \longrightarrow 0$
proof (*rule metric-LIMSEQ-I*)
fix *e*
assume $e > (0$::*real*)
then have $k\ /\ e > 0$ **using** *assms*(*1*) **by** *auto*
obtain *N* **where** *N*: $b\hat{\ }(N$::*nat*) $> k\ /\ e$ **using** *assms*(*2*)
using *real-arch-pow* **by** *blast*
then have *norm* (*s n*) $< e$ **if** $n \geq N$ **for** *n*
proof −
have $k\ /\ b\hat{\ }n \leq k\ /\ b\hat{\ }N$
by (*smt assms*(*1*) *assms*(*2*) *frac-le leD power-less-imp-less-exp that zero-less-power*)
also have ... $< e$ **using** *N*

by (*metis* ‹$0 < e$› *assms*(2) *less-trans mult.commute pos-divide-less-eq zero-less-one zero-less-power*)
finally show *?thesis*
by (*meson assms less-eq-real-def not-le order-trans*)
qed
then show $\exists\, no.\ \forall\, n \geq no.\ dist\ (s\ n)\ 0 < e$
by *auto*
qed

lemma *filterlim-apply-filtermap*:
assumes *g*: *filterlim g G F*
shows *filterlim* ($\lambda x.\ m\ (g\ x)$) (*filtermap m G*) *F*
by (*metis filterlim-def filterlim-filtermap filtermap-mono g*)

lemma *eventually-at-right-field-le*:
eventually P (*at-right x*) $\longleftrightarrow$ ($\exists\, b>x.\ \forall\, y>x.\ y \leq b \longrightarrow P\ y$)
for *x* :: *'a*::{*linordered-field*, *linorder-topology*}
by (*smt dense eventually-at-right-field le-less-trans less-le-not-le order.strict-trans1*)

1.2 indexing euclidean space with natural numbers

definition *nth-eucl* :: *'a*::*executable-euclidean-space* ⇒ *nat* ⇒ *real* **where**
nth-eucl x i = *x* • (*Basis-list* ! *i*)
— TODO: why is that and some sort of *lambda-eucl* nowhere available?
definition *lambda-eucl* :: (*nat* ⇒ *real*) ⇒ *'a*::*executable-euclidean-space* **where**
lambda-eucl (*f*::*nat*⇒*real*) = ($\sum i<DIM('a).\ f\ i *_R Basis\text{-}list\ !\ i$)

lemma *eucl-eq-iff*: $x = y \longleftrightarrow (\forall\, i<DIM('a).\ nth\text{-}eucl\ x\ i = nth\text{-}eucl\ y\ i)$
for *x y*::*'a*::*executable-euclidean-space*
apply (*auto simp*: *nth-eucl-def euclidean-eq-iff*[**where** *'a*=*'a*])
by (*metis eucl-of-list-list-of-eucl list-of-eucl-eq-iff*)

open-bundle *eucl-syntax*
begin
notation *nth-eucl* (**infixl** ‹$\$_e$› *90*)
end

lemma *eucl-of-list-eucl-nth*:
(*eucl-of-list xs*::*'a*) $\$_e$ *i* = *xs* ! *i*
if *length xs* = *DIM*(*'a*::*executable-euclidean-space*)
$i < DIM('a)$
using *that*
apply (*auto simp*: *nth-eucl-def*)
by (*metis list-of-eucl-eucl-of-list list-of-eucl-nth*)

lemma *eucl-of-list-inner*:
(*eucl-of-list xs*::*'a*) • *eucl-of-list ys* = ($\sum (x,y)\leftarrow zip\ xs\ ys.\ x * y$)
if *length xs* = *DIM*(*'a*::*executable-euclidean-space*)
length ys = *DIM*(*'a*::*executable-euclidean-space*)

using *that*
by (*auto simp*: *nth-eucl-def eucl-of-list-inner-eq inner-lv-rel-def*)

lemma *self-eq-eucl-of-list*: $x = eucl\text{-}of\text{-}list\ (map\ (\lambda i.\ x\ \$_e\ i)\ [0..<DIM('a)])$
for $x::'a::executable\text{-}euclidean\text{-}space$
by (*auto simp*: *eucl-eq-iff*[**where** *'a='a*] *eucl-of-list-eucl-nth*)

lemma *inner-nth-eucl*: $x \cdot y = (\sum i<DIM('a).\ x\ \$_e\ i * y\ \$_e\ i)$
for $x\ y::'a::executable\text{-}euclidean\text{-}space$
apply (*subst self-eq-eucl-of-list*[**where** *x=x*])
apply (*subst self-eq-eucl-of-list*[**where** *x=y*])
apply (*subst eucl-of-list-inner*)
by (*auto simp*: *map2-map-map atLeast-upt interv-sum-list-conv-sum-set-nat*)

lemma *norm-nth-eucl*: $norm\ x = L2\text{-}set\ (\lambda i.\ x\ \$_e\ i)\ \{..<DIM('a)\}$
for $x::'a::executable\text{-}euclidean\text{-}space$
unfolding *norm-eq-sqrt-inner inner-nth-eucl L2-set-def*
by (*auto simp*: *power2-eq-square*)

lemma *plus-nth-eucl*: $(x + y)\ \$_e\ i = x\ \$_e\ i + y\ \$_e\ i$
and *minus-nth-eucl*: $(x - y)\ \$_e\ i = x\ \$_e\ i - y\ \$_e\ i$
and *uminus-nth-eucl*: $(-x)\ \$_e\ i = -\ x\ \$_e\ i$
and *scaleR-nth-eucl*: $(c *_R x)\ \$_e\ i = c *_R (x\ \$_e\ i)$
by (*auto simp*: *nth-eucl-def algebra-simps*)

lemma *inf-nth-eucl*: $inf\ x\ y\ \$_e\ i = min\ (x\ \$_e\ i)\ (y\ \$_e\ i)$
if $i < DIM('a)$
for $x::'a::executable\text{-}euclidean\text{-}space$
by (*auto simp*: *nth-eucl-def algebra-simps inner-Basis-inf-left that inf-min*)
lemma *sup-nth-eucl*: $sup\ x\ y\ \$_e\ i = max\ (x\ \$_e\ i)\ (y\ \$_e\ i)$
if $i < DIM('a)$
for $x::'a::executable\text{-}euclidean\text{-}space$
by (*auto simp*: *nth-eucl-def algebra-simps inner-Basis-sup-left that sup-max*)

lemma *le-iff-le-nth-eucl*: $x \le y \longleftrightarrow (\forall i<DIM('a).\ (x\ \$_e\ i) \le (y\ \$_e\ i))$
for $x::'a::executable\text{-}euclidean\text{-}space$
apply (*auto simp*: *nth-eucl-def algebra-simps eucl-le*[**where** *'a='a*])
by (*meson eucl-le eucl-le-Basis-list-iff*)

lemma *eucl-less-iff-less-nth-eucl*: $eucl\text{-}less\ x\ y \longleftrightarrow (\forall i<DIM('a).\ (x\ \$_e\ i) < (y\ \$_e\ i))$
for $x::'a::executable\text{-}euclidean\text{-}space$
apply (*auto simp*: *nth-eucl-def algebra-simps eucl-less-def*[**where** *'a='a*])
by (*metis Basis-zero eucl-eq-iff inner-not-same-Basis inner-zero-left length-Basis-list nth-Basis-list-in-Basis nth-eucl-def*)

lemma *continuous-on-nth-eucl*[*continuous-intros*]:
$continuous\text{-}on\ X\ (\lambda x.\ f\ x\ \$_e\ i)$

if *continuous-on X f*
by (*auto simp*: *nth-eucl-def intro*!: *continuous-intros that*)

1.3 derivatives

lemma *eventually-at-ne*[*intro*, *simp*]: $\forall_F$ *x in at x0*. $x \neq x0$
by (*auto simp*: *eventually-at-filter*)

lemma *has-vector-derivative-withinD*:
fixes *f*::*real* ⇒ *'b*::*euclidean-space*
assumes (*f has-vector-derivative f'*) (*at x0 within S*)
shows ((λx. (*f x* − *f x0*) $/_R$ (*x* − *x0*)) ⟶ *f'*) (*at x0 within S*)
apply (*rule LIM-zero-cancel*)
apply (*rule tendsto-norm-zero-cancel*)
apply (*rule Lim-transform-eventually*)
proof −
show $\forall_F$ *x in at x0 within S*. *norm* ((*f x* − *f x0* − (*x* − *x0*) $*_R$ *f'*) $/_R$ *norm* (*x* − *x0*)) =
norm ((*f x* − *f x0*) $/_R$ (*x* − *x0*) − *f'*)
(**is** $\forall_F$ *x in* -. *?th x*)
unfolding *eventually-at-filter*
proof (*safe intro*!: *eventuallyI*)
fix *x* **assume** *x*: $x \neq x0$
then have *norm* ((*f x* − *f x0*) $/_R$ (*x* − *x0*) − *f'*) = *norm* (*sgn* (*x* − *x0*) $*_R$ ((*f x* − *f x0*) $/_R$ (*x* − *x0*) − *f'*))
by *simp*
also have *sgn* (*x* − *x0*) $*_R$ ((*f x* − *f x0*) $/_R$ (*x* − *x0*) − *f'*) = ((*f x* − *f x0*) $/_R$ *norm* (*x* − *x0*) − (*x* − *x0*) $*_R$ *f'* $/_R$ *norm* (*x* − *x0*))
by (*auto simp add*: *algebra-simps sgn-div-norm divide-simps*)
(*metis add.commute add-divide-distrib diff-add-cancel scaleR-add-left*)
also have ... = (*f x* − *f x0* − (*x* − *x0*) $*_R$ *f'*) $/_R$ *norm* (*x* − *x0*) **by** (*simp add*: *algebra-simps*)
finally show *?th x* **..**
qed
show ((λx. *norm* ((*f x* − *f x0* − (*x* − *x0*) $*_R$ *f'*) $/_R$ *norm* (*x* − *x0*))) ⟶ *0*) (*at x0 within S*)
by (*rule tendsto-norm-zero*)
(*use assms* **in** ‹*auto simp*: *has-vector-derivative-def has-derivative-at-within*›)
qed

A *path-connected* set S entering both T and $-T$ must cross the frontier of T

lemma *path-connected-frontier*:
fixes *S* :: *'a*::*real-normed-vector set*
assumes *path-connected S*
assumes $S \cap T \neq \{\}$
assumes $S \cap -T \neq \{\}$
obtains *s* **where** $s \in S$ $s \in$ *frontier T*
proof −

obtain *st* **where** *st*:*st* $\in S \cap T$ **using** *assms*(*2*) **by** *blast*
obtain *sn* **where** *sn*:*sn* $\in S \cap -T$ **using** *assms*(*3*) **by** *blast*
obtain *g* **where** *g*: *path g path-image g* $\subseteq S$
pathstart g = *st pathfinish g* = *sn*
using *assms*(*1*) *st sn* **unfolding** *path-connected-def* **by** *blast*
have *a1*:*pathstart g* $\in$ *closure T* **using** *st g*(*3*) *closure-Un-frontier* **by** *fastforce*
have *a2*:*pathfinish g* $\notin T$ **using** *sn g*(*4*) **by** *auto*
from *exists-path-subpath-to-frontier*[*OF g*(*1*) *a1 a2*]
obtain *h* **where** *path-image h* $\subseteq$ *path-image g pathfinish h* $\in$ *frontier T* **by** *metis*
thus *?thesis* **using** *g*(*2*)
by (*meson in-mono pathfinish-in-path-image that*)
qed

lemma *path-connected-not-frontier-subset*:
fixes *S* :: *'a*::*real-normed-vector set*
assumes *path-connected S*
assumes $S \cap T \neq \{\}$
assumes $S \cap$ *frontier T* = {}
shows $S \subseteq T$
using *path-connected-frontier assms* **by** *auto*

lemma *compact-attains-bounds*:
fixes *f*::*'a*::*topological-space* ⇒ *'b*::*linorder-topology*
assumes *compact*: *compact S*
assumes *ne*: $S \neq \{\}$
assumes *cont*: *continuous-on S f*
obtains *l u* **where** $l \in S$ $u \in S$ $\bigwedge x.\ x \in S \Longrightarrow f\,x \in \{f\,l\ ..\ f\,u\}$
proof −
from *compact-continuous-image*[*OF cont compact*]
have *compact-image*: *compact* (*f* ‘ *S*) **.**
have *ne-image*: *f* ‘ *S* ≠ {} **using** *ne* **by** *simp*
from *compact-attains-inf*[*OF compact-image ne-image*]
obtain *l* **where** $l \in S$ $\bigwedge x.\ x \in S \Longrightarrow f\,l \le f\,x$ **by** *auto*
moreover
from *compact-attains-sup*[*OF compact-image ne-image*]
obtain *u* **where** $u \in S$ $\bigwedge x.\ x \in S \Longrightarrow f\,x \le f\,u$ **by** *auto*
ultimately
have $l \in S$ $u \in S$ $\bigwedge x.\ x \in S \Longrightarrow f\,x \in \{f\,l\ ..\ f\,u\}$ **by** *auto*
then show *?thesis* **..**
qed

lemma *uniform-limit-const*[*uniform-limit-intros*]:
uniform-limit S ($\lambda x\ y.\ f\,x$) (λ-. *l*) *F* **if** ($f \longrightarrow l$) *F*
apply (*auto simp*: *uniform-limit-iff*)
subgoal for *e*
using *tendstoD*[*OF that*(*1*), *of e*]
by (*auto simp*: *eventually-mono*)
done

1.4 Segments

closed-segment throws away the order that our intuition keeps

definition *line*::$'a$::*real-vector* $\Rightarrow$ $'a$ $\Rightarrow$ *real* $\Rightarrow$ $'a$
($\langle\{$- $--$ -$\}$-$\rangle$)
where $\{a -- b\}_u = a + u *_R (b - a)$

abbreviation *line-image* a b U $\equiv(\lambda u.\ \{a -- b\}_u)$ ' U
notation *line-image* ($\langle\{$- $--$ -$\}$'-$\rangle$)

lemma *in-closed-segment-iff-line*: $x \in \{a -- b\} \longleftrightarrow (\exists\, c\in\{0..1\}.\ x = line\ a\ b\ c)$
by (*auto simp*: *in-segment line-def algebra-simps*)

lemma *in-open-segment-iff-line*: $x \in \{a <--< b\} \longleftrightarrow (\exists\, c\in\{0<..<1\}.\ a \neq b \wedge x = line\ a\ b\ c)$
by (*auto simp*: *in-segment line-def algebra-simps*)

lemma *line-convex-combination1*: $(1 - u) *_R line\ a\ b\ i + u *_R b = line\ a\ b\ (i + u - i * u)$
by (*auto simp*: *line-def algebra-simps*)

lemma *line-convex-combination2*: $(1 - u) *_R a + u *_R line\ a\ b\ i = line\ a\ b\ (i{*}u)$
by (*auto simp*: *line-def algebra-simps*)

lemma *line-convex-combination12*: $(1 - u) *_R line\ a\ b\ i + u *_R line\ a\ b\ j = line\ a\ b\ (i + u * (j - i))$
by (*auto simp*: *line-def algebra-simps*)

lemma *mult-less-one-less-self*: $0 < x \Longrightarrow i < 1 \Longrightarrow i * x < x$ **for** i x::*real*
by *auto*

lemma *plus-times-le-one-lemma*: $i + u - i * u \leq 1$ **if** $i \leq 1$ $u \leq 1$ **for** i u::*real*
by (*simp add*: *diff-le-eq sum-le-prod1 that*)

lemma *plus-times-less-one-lemma*: $i + u - i * u < 1$ **if** $i < 1$ $u < 1$ **for** i u::*real*
proof $-$
have $u * (1 - i) < 1 - i$
using *that* **by** *force*
then show *?thesis* **by** (*simp add*: *algebra-simps*)
qed

lemma *line-eq-endpoint-iff*[*simp*]:
$line\ a\ b\ i = b \longleftrightarrow (a = b \vee i = 1)$
$a = line\ a\ b\ i \longleftrightarrow (a = b \vee i = 0)$
by (*auto simp*: *line-def algebra-simps*)

lemma *line-eq-iff*[*simp*]: $line\ a\ b\ x = line\ a\ b\ y \longleftrightarrow (x = y \vee a = b)$
by (*auto simp*: *line-def*)

```
lemma line-open-segment-iff:
  {line a b i<−−<b} = line a b ‘ {i<..<1}
  if i < 1 a ≠ b
  using that
 apply (auto simp: in-segment line-convex-combination1 plus-times-less-one-lemma)
  subgoal for j
    apply (rule exI[where x=(j − i)/(1 − i)])
    apply (auto simp: divide-simps algebra-simps)
   by (metis add-diff-cancel less-numeral-extra(4) mult-2-right plus-times-less-one-lemma
that(1))
  done

lemma open-segment-line-iff:
  {a<−−<line a b i} = line a b ‘ {0<..<i}
  if 0 < i a ≠ b
  using that
 apply (auto simp: in-segment line-convex-combination2 plus-times-less-one-lemma)
  subgoal for j
    apply (rule exI[where x=j/i])
    by auto
  done

lemma line-closed-segment-iff:
  {line a b i−−b} = line a b ‘ {i..1}
  if i ≤ 1 a ≠ b
  using that
 apply (auto simp: in-segment line-convex-combination1 mult-le-cancel-right2 plus-times-le-one-lemma)
  subgoal for j
    apply (rule exI[where x=(j − i)/(1 − i)])
    apply (auto simp: divide-simps algebra-simps)
   by (metis add-diff-cancel less-numeral-extra(4) mult-2-right plus-times-less-one-lemma
that(1))
  done

lemma closed-segment-line-iff:
  {a−−line a b i} = line a b ‘ {0..i}
  if 0 < i a ≠ b
  using that
 apply (auto simp: in-segment line-convex-combination2 plus-times-less-one-lemma)
  subgoal for j
    apply (rule exI[where x=j/i])
    by auto
  done

lemma closed-segment-line-line-iff: {line a b i1−−line a b i2} = line a b ‘ {i1..i2}
if i1 ≤ i2
  using that
  apply (auto simp: in-segment line-convex-combination12 intro!: imageI)
   apply (smt mult-left-le-one-le)
```

subgoal for *u*
by (*rule exI*[**where** *x*=(*u* − *i1*)/(*i2*−*i1*)]) *auto*
done

lemma *line-line1*: *line* (*line a b c*) *b x* = *line a b* (*c* + *x* − *c* ∗ *x*)
by (*simp add*: *line-def algebra-simps*)

lemma *line-line2*: *line a* (*line a b c*) *x* = *line a b* (*c*∗*x*)
by (*simp add*: *line-def algebra-simps*)

lemma *line-in-subsegment*:
i1 < *1* ⟹ *i2* < *1* ⟹ *a* ≠ *b* ⟹ *line a b i1* ∈ {*line a b i2*<−−<*b*} ⟷ *i2* < *i1*
by (*auto simp*: *line-open-segment-iff intro*!: *imageI*)

lemma *line-in-subsegment2*:
0 < *i2* ⟹ *0* < *i1* ⟹ *a* ≠ *b* ⟹ *line a b i1* ∈ {*a*<−−<*line a b i2*} ⟷ *i1* < *i2*
by (*auto simp*: *open-segment-line-iff intro*!: *imageI*)

lemma *line-in-open-segment-iff* [*simp*]:
line a b i ∈ {*a*<−−<*b*} ⟷ (*a* ≠ *b* ∧ *0* < *i* ∧ *i* < *1*)
by (*auto simp*: *in-open-segment-iff-line*)

1.5 Open Segments

lemma *open-segment-subsegment*:
assumes *x1* ∈ {*x0*<−−<*x3*}
x2 ∈ {*x1*<−−<*x3*}
shows *x1* ∈ {*x0*<−−<*x2*}
using *assms*
proof –– TODO: use *line*
from *assms* **obtain** *u v*::*real* **where**
ne: *x0* ≠ *x3* (*1* − *u*) $*_R$ *x0* + *u* $*_R$ *x3* ≠ *x3*
and *x1-def*: *x1* = (*1* − *u*) $*_R$ *x0* + *u* $*_R$ *x3*
and *x2-def*: *x2* = (*1* − *v*) $*_R$ ((*1* − *u*) $*_R$ *x0* + *u* $*_R$ *x3*) + *v* $*_R$ *x3*
and *uv*: ‹*0* < *u*› ‹*0* < *v*› ‹*u* < *1*› ‹*v* < *1*›
by (*auto simp*: *in-segment*)
let *?d* = (*u* + *v* − *u* ∗ *v*)
have *?d* > *0* **using** *uv*
by (*auto simp*: *add-nonneg-pos pos-add-strict*)
with ‹*x0* ≠ *x3*› **have** *0* ≠ *?d* $*_R$ (*x3* − *x0*) **by** *simp*
moreover
define *ua* **where** *ua* = *u* / *?d*
have *ua* ∗ (*u* ∗ *v* − *u* − *v*) − − *u* = *0*
by (*auto simp*: *ua-def algebra-simps divide-simps*)
(*metis uv add-less-same-cancel1 add-strict-mono mult.right-neutral mult-less-cancel-left-pos not-real-square-gt-zero vector-space-over-itself.scale-zero-left*)
then have (*ua* ∗ (*u* ∗ *v* − *u* − *v*) − − *u*) $*_R$ (*x3* − *x0*) = *0*

by *simp*
moreover
have $0 < ua$ $ua < 1$
using ‹$0 < u$› ‹$0 < v$› ‹$u < 1$› ‹$v < 1$›
by (*auto simp*: *ua-def pos-add-strict intro*!: *divide-pos-pos*)
ultimately show *?thesis*
unfolding *x1-def x2-def*
by (*auto intro*!: *exI*[**where** *x*=*ua*] *simp*: *algebra-simps in-segment*)
qed

1.6 Syntax

abbreviation *sequentially-at-top*::(*nat*⇒*real*)⇒*bool*
(‹- $\longrightarrow$ ∞›) — the is to disambiguate syntax...
where $s \longrightarrow \infty \equiv$ *filterlim s at-top sequentially*

abbreviation *sequentially-at-bot*::(*nat*⇒*real*)⇒*bool*
(‹- $\longrightarrow$ −∞›)
where $s \longrightarrow -\infty \equiv$ *filterlim s at-bot sequentially*

1.7 Paths

lemma *subpath0-linepath*:
shows *subpath 0 u* (*linepath t t'*) = *linepath t* ($t + u * (t' - t)$)
unfolding *subpath-def linepath-def*
apply (*rule ext*)
apply *auto*
proof −
fix *x* :: *real*
have *f1*: $\bigwedge r$ *ra rb rc*. (r::*real*) + *ra* ∗ *rb* − *ra* ∗ *rc* = *r* − *ra* ∗ (*rc* − *rb*)
by (*simp add*: *right-diff-distrib'*)
have *f2*: $\bigwedge r$ *ra*. (r::*real*) − *r* ∗ *ra* = *r* ∗ (*1* − *ra*)
by (*simp add*: *right-diff-distrib'*)
have *f3*: $\bigwedge r$ *ra rb*. (r::*real*) − *ra* + *rb* + *ra* − *r* = *rb*
by *auto*
have *f4*: $\bigwedge r$. (r::*real*) + (*1* − *1*) = *r*
by *linarith*
have *f5*: $\bigwedge r$ *ra*. (r::*real*) + *ra* = *ra* + *r*
by *force*
have *f6*: $\bigwedge r$ *ra*. (r::*real*) + (*1* − (*r* + *1*) + *ra*) = *ra*
by *linarith*
have $t - x * (t - (t + u * (t' - t))) = t' * (u * x) + (t - t * (u * x))$
by (*simp add*: *right-diff-distrib'*)
then show $(1 - u * x) * t + u * x * t' = (1 - x) * t + x * (t + u * (t' - t))$
using *f6 f5 f4 f3 f2 f1* **by** (*metis* (*no-types*) *mult.commute*)
qed

lemma *linepath-image0-right-open-real*:
assumes $t < (t'$::*real*)
shows *linepath t t'* ‘ {*0*..<*1*} = {*t*..<*t'*}

```
  unfolding linepath-def
  apply auto
    apply (metis add.commute add-diff-cancel-left' assms diff-diff-eq2 diff-le-eq
less-eq-real-def mult.commute mult.right-neutral mult-right-mono right-diff-distrib')
  apply (smt assms comm-semiring-class.distrib mult-diff-mult semiring-normalization-rules(2)
zero-le-mult-iff)
proof -
  fix x
  assume t ≤ x x < t'
  let ?u = (x−t)/(t'−t)
  have ?u ≥ 0
    using ‹t ≤ x› assms by auto
  moreover have ?u < 1
    by (simp add: ‹x < t'› assms)
  moreover have x = (1−?u) ∗ t + ?u∗t'
  proof -
    have f1: ∀ r ra. (ra::real) ∗ − r = r ∗ − ra
      by simp
    have t + (t' + − t) ∗ ((x + − t) / (t' + − t)) = x
      using assms by force
    then have t' ∗ ((x + − t) / (t' + − t)) + t ∗ (1 + − ((x + − t) / (t' + −
t))) = x
      using f1 by (metis (no-types) add.left-commute distrib-left mult.commute
mult.right-neutral)
    then show ?thesis
      by (simp add: mult.commute)
  qed
  ultimately show x ∈ (λx. (1 − x) ∗ t + x ∗ t') ‘ {0..<1}
    using atLeastLessThan-iff by blast
qed

lemma oriented-subsegment-scale:
  assumes x1 ∈ {a<−−<b}
  assumes x2 ∈ {x1<−−<b}
  obtains e where e > 0 b−a = e ∗R (x2−x1)
proof -
  from assms(1) obtain u where u : u > 0 u < 1 x1 = (1 − u) ∗R a + u ∗R b
    unfolding in-segment by blast
  from assms(2) obtain v where v: v > 0 v < 1 x2 = (1 − v) ∗R x1 + v ∗R b
    unfolding in-segment by blast
  have x2−x1 = −v ∗R x1 + v ∗R b using v
    by (metis add.commute add-diff-cancel-right diff-minus-eq-add scaleR-collapse
scaleR-left.minus)
  also have ... = (−v) ∗R ((1 − u) ∗R a + u ∗R b) + v ∗R b using u by auto
  also have ... = v ∗R ((1−u)∗R b − (1−u)∗R a )
   by (smt add-diff-cancel diff-diff-add diff-minus-eq-add minus-diff-eq scaleR-collapse
scale-minus-left scale-right-diff-distrib)
  finally have x2x1:x2−x1 = (v ∗(1−u)) ∗R (b − a)
    by (metis scaleR-scaleR scale-right-diff-distrib)
```

```
  have v ∗ (1−u) > 0 using u(2) v(1) by simp
  then have (x2−x1)/R (v ∗ (1−u)) = (b−a) unfolding x2x1
    by (smt field-class.field-inverse scaleR-one scaleR-scaleR)
  thus ?thesis
    using ‹0 < v ∗ (1 − u)› positive-imp-inverse-positive that by fastforce
qed

end
```

2 Additions to the ODE Library

```
theory ODE-Misc
  imports
    Ordinary-Differential-Equations.ODE-Analysis
    Analysis-Misc
begin

lemma local-lipschitz-compact-bicomposeE:
  assumes ll: local-lipschitz T X f
  assumes cf: ⋀x. x ∈ X ⟹ continuous-on I (λt. f t x)
  assumes cI: compact I
  assumes I ⊆ T
  assumes cv: continuous-on I v
  assumes cw: continuous-on I w
  assumes v: v ‘ I ⊆ X
  assumes w: w ‘ I ⊆ X
  obtains L where L > 0 ⋀x. x ∈ I ⟹ dist (f x (v x)) (f x (w x)) ≤ L ∗ dist
(v x) (w x)
proof −
  from v w have v ‘ I ∪ w ‘ I ⊆ X by auto
  with ll ‹I ⊆ T› have llI:local-lipschitz I (v ‘ I ∪ w ‘ I) f
    by (rule local-lipschitz-subset)
  have cvwI: compact (v ‘ I ∪ w ‘ I)
    by (auto intro!: compact-continuous-image cv cw cI)

  from local-lipschitz-compact-implies-lipschitz[OF llI cvwI ‹compact I› cf]
  obtain L where L: ⋀t. t ∈ I ⟹ L−lipschitz-on (v ‘ I ∪ w ‘ I) (f t)
    using v w
    by blast
  define L′ where L′ = max L 1
  with L have L′ > 0 ⋀x. x ∈ I ⟹ dist (f x (v x)) (f x (w x)) ≤ L′ ∗ dist (v x)
(w x)
     apply (auto simp: lipschitz-on-def L′-def)
    by (smt Un-iff image-eqI mult-right-mono zero-le-dist)
  then show ?thesis ..
qed
```

2.1 Comparison Principle

lemma *comparison-principle-le*:
  fixes f::*real* $\Rightarrow$ *real* $\Rightarrow$ *real*
    and φ ψ::*real* $\Rightarrow$ *real*
  assumes *ll*: *local-lipschitz X Y f*
  assumes *cf*: $\bigwedge x.\ x \in Y \Longrightarrow$ *continuous-on* $\{a..b\}$ $(\lambda t.\ f\ t\ x)$
  assumes *abX*: $\{a\ ..\ b\} \subseteq X$
  assumes φ': $\bigwedge x.\ x \in \{a\ ..\ b\} \Longrightarrow (\varphi$ *has-real-derivative* $\varphi'\ x)$ $(at\ x)$
  assumes ψ': $\bigwedge x.\ x \in \{a\ ..\ b\} \Longrightarrow (\psi$ *has-real-derivative* $\psi'\ x)$ $(at\ x)$
  assumes φ-*in*: φ ‘ $\{a..b\} \subseteq Y$
  assumes ψ-*in*: ψ ‘ $\{a..b\} \subseteq Y$
  assumes *init*: $\varphi\ a \le \psi\ a$
  assumes *defect*: $\bigwedge x.\ x \in \{a\ ..\ b\} \Longrightarrow \varphi'\ x - f\ x\ (\varphi\ x) \le \psi'\ x - f\ x\ (\psi\ x)$
  shows $\forall\, x \in \{a\ ..\ b\}.\ \varphi\ x \le \psi\ x$ (**is** *?th1*)

  unfolding *atomize-conj*
  apply (*cases* $a \le b$)
   defer subgoal by *simp*
proof −
  assume $a \le b$
  note φ-*cont* = *has-real-derivative-imp-continuous-on*[*OF* φ']
  note ψ-*cont* = *has-real-derivative-imp-continuous-on*[*OF* ψ']
  from *local-lipschitz-compact-bicomposeE*[*OF ll cf compact-Icc abX* φ-*cont* ψ-*cont* φ-*in* ψ-*in*]
  obtain L **where** L: $L > 0$ $\bigwedge x.\ x \in \{a..b\} \Longrightarrow$ *dist* $(f\ x\ (\varphi\ x))\ (f\ x\ (\psi\ x)) \le L$ $*$ *dist* $(\varphi\ x)\ (\psi\ x)$ **by** *blast*
  define w **where** $w\ x = \psi\ x - \varphi\ x$ **for** x

  have w'[*derivative-intros*]: $\bigwedge x.\ x \in \{a\ ..\ b\} \Longrightarrow (w$ *has-real-derivative* $\psi'\ x - \varphi'\ x)$ $(at\ x)$
    using φ' ψ'
    by (*auto simp*: *has-vderiv-on-def w-def*[*abs-def*] *intro*!: *derivative-eq-intros*)
  note *w-cont*[*continuous-intros*] = *has-real-derivative-imp-continuous-on*[*OF* w', *THEN continuous-on-compose2*]
  have $w\ d \ge 0$ **if** $d \in \{a\ ..\ b\}$ **for** d
  proof (*rule ccontr*, *unfold not-le*)
    assume $w\ d < 0$
    let *?N* = $(w$ −‘ $\{..0\} \cap \{a\ ..\ d\})$
    from ‹$w\ d < 0$› *that* **have** $d \in$ *?N* **by** *auto*
    then have *?N* $\neq \{\}$ **by** *auto*
    have *closed ?N*
      unfolding *compact-eq-bounded-closed*
      using *that*
      by (*intro conjI closed-vimage-Int*) (*auto intro*!: *continuous-intros*)

    let *?N'* = $\{a0 \in \{a\ ..\ d\}.\ w$ ‘ $\{a0\ ..\ d\} \subseteq \{..0\}\}$
    from ‹$w\ d < 0$› *that* **have** $d \in$ *?N'* **by** *simp*
    then have *?N'* $\neq \{\}$ **by** *auto*
    have *compact ?N'*

```
  unfolding compact-eq-bounded-closed
proof
  have ?N′ ⊆ {a .. d} using that by auto
  then show bounded ?N′
    by (rule bounded-subset[rotated]) simp
  have w u ≤ 0 if (∀ n. x n ∈ ?N′) x ⟶ l l ≤ u u ≤ d for x l u
  proof cases
    assume l = u
    have ∀ n. x n ∈ ?N using that(1) by force
    from closed-sequentially[OF ‹closed ?N›] this ‹x ⟶ l›
    show ?thesis
      using ‹l = u› by blast
  next
    assume l ≠ u with that have l < u by auto
   from order-tendstoD(2)[OF ‹x ⟶ l› ‹l < u›] obtain n where x n < u
      by (auto dest: eventually-happens)
    with that show ?thesis using ‹l < u›
      by (auto dest!: spec[where x=n] simp: image-subset-iff)
  qed
  then show closed ?N′
    unfolding closed-sequential-limits
    by (auto simp: Lim-bounded Lim-bounded2)
qed

from compact-attains-inf[OF ‹compact ?N′› ‹?N′ ≠ {}›]
obtain a0 where a0: a ≤ a0 a0 ≤ d w ‘ {a0..d} ⊆ {..0}
  and a0-least: ⋀x. a ≤ x ⟹ x ≤ d ⟹ w ‘ {x..d} ⊆ {..0} ⟹ a0 ≤ x
  by auto
have a0d: {a0 .. d} ⊆ {a .. b} using that a0
  by auto
have L-w-bound: L ∗ w x ≤ ψ′ x − φ′ x if x ∈ {a0 .. d} for x
proof −
  from set-mp[OF a0d that] have x ∈ {a .. b} .
  from defect[OF this]
  have φ′ x − ψ′ x ≤ dist (f x (φ x)) (f x (ψ x))
    by (simp add: dist-real-def)
  also have . . . ≤ L ∗ dist (φ x) (ψ x)
    using ‹x ∈ {a .. b}›
    by (rule L)
  also have . . . ≤ −L ∗ w x
    using ‹0 < L› a0 that
    by (force simp add: dist-real-def abs-real-def w-def algebra-split-simps )
  finally show ?thesis
    by simp
qed
have mono: mono-on {a0..d} (λx. w x ∗ exp(−L∗x))
  apply (rule mono-onI)
  apply (rule DERIV-nonneg-imp-nondecreasing, assumption)
  using a0d
```

```
      by (auto intro!: exI[where x=(ψ' x − φ' x) ∗ exp (− (L ∗ x)) − exp (− (L
∗ x)) ∗ L ∗ w x for x]
          derivative-eq-intros L-w-bound simp: )
    then have w a0 ∗ exp (−L ∗ a0) ≤ w d ∗ exp (−L ∗ d)
      by (rule mono-onD) (use that a0 in auto)
    also have ... < 0 using ‹w d < 0› by (simp add: algebra-split-simps)
    finally have w a0 ∗ exp (− L ∗ a0) < 0 .
    then have w a0 < 0 by (simp add: algebra-split-simps)
    have a0 ≤ a
    proof (rule ccontr, unfold not-le)
      assume a < a0
      have continuous-on {a.. a0} w
        by (rule continuous-intros, assumption) (use a0 a0d in auto)
      from continuous-on-Icc-at-leftD[OF this ‹a < a0›]
      have (w ⟶ w a0) (at-left a0) .
      from order-tendstoD(2)[OF this ‹w a0 < 0›] have ∀F x in at-left a0. w x <
0 .
      moreover have ∀F x in at-left a0. a < x
        by (rule order-tendstoD) (auto intro!: ‹a < a0›)
      ultimately have ∀F x in at-left a0. a < x ∧ w x < 0 by eventually-elim
auto
      then obtain a1' where a1'<a0 and a1-neg: ⋀y. y > a1' ⟹ y < a0 ⟹
a < y ∧ w y < 0
        unfolding eventually-at-left-field by auto
      define a1 where a1 = (a1' + a0)/2
      have a1 < a0 using ‹a1' < a0› by (auto simp: a1-def)
      have a ≤ a1
        using ‹a < a0› a1-neg by (force simp: a1-def)
      moreover have a1 ≤ d
        using ‹a1' < a0› a0(2) by (auto simp: a1-def)
      moreover have w ‘ {a1..a0} ⊆ {..0}
        using ‹w a0 < 0› a1-neg a0(3)
        by (auto simp: a1-def) smt
      moreover have w ‘ {a0..d} ⊆ {..0} using a0 by auto
      ultimately
      have a0 ≤ a1
        apply (intro a0-least) apply assumption apply assumption
        by (smt atLeastAtMost-iff image-subset-iff)
      with ‹a1<a0› show False by simp
    qed
    then have a0 = a using ‹a ≤ a0› by simp
    with ‹w a0 < 0› have w a < 0 by simp
    with init show False
      by (auto simp: w-def)
  qed
  then show ?thesis
    by (auto simp: w-def)
qed
```

lemma *local-lipschitz-mult*:
  shows *local-lipschitz* (*UNIV*::*real set*) (*UNIV*::*real set*) ($*$)
  apply (*auto intro*!: *c1-implies-local-lipschitz*[**where** $f'=\lambda p.$ *blinfun-mult-left* (*fst p*)])
   apply (*simp add*: *has-derivative-mult-right mult-commute-abs*)
  by (*auto intro*!: *continuous-intros*)

lemma *comparison-principle-le-linear*:
  fixes φ :: *real* $\Rightarrow$ *real*
  assumes *continuous-on* $\{a..b\}$ g
  assumes $(\bigwedge t.\ t \in \{a..b\} \Longrightarrow (\varphi$ *has-real-derivative* $\varphi'\ t)$ (*at* t))
  assumes $\varphi\ a \leq 0$
  assumes $(\bigwedge t.\ t \in \{a..b\} \Longrightarrow \varphi'\ t \leq g\ t *_R \varphi\ t)$
  shows $\forall t{\in}\{a..b\}.\ \varphi\ t \leq 0$
proof $-$
  have $*$: $\bigwedge x.$ *continuous-on* $\{a..b\}$ $(\lambda t.\ g\ t * x)$
    using *assms*(*1*) *continuous-on-mult-right* **by** *blast*
  then have *local-lipschitz* (g‘$\{a..b\}$) *UNIV* ($*$)
    using *local-lipschitz-subset*[*OF local-lipschitz-mult*] **by** *blast*
  from *local-lipschitz-compose1*[*OF this assms*(*1*)]
  have *local-lipschitz* $\{a..b\}$ *UNIV* $(\lambda t.\ (*)\ (g\ t))$ **.**
  from *comparison-principle-le*[*OF this - - assms*(*2*) *- - - assms*(*3*), *of b* $\lambda t.0$] $*$ *assms*(*4*)
  show *?thesis* **by** *auto*
qed

2.2 Locally Lipschitz ODEs

context *ll-on-open-it* **begin**

lemma *flow-lipschitzE*:
  assumes $\{a\ ..\ b\} \subseteq$ *existence-ivl t0 x*
  obtains L **where** L–*lipschitz-on* $\{a\ ..\ b\}$ (*flow t0 x*)
proof $-$
  have f': (*flow t0 x has-derivative* $(\lambda i.\ i *_R f\ t$ (*flow t0 x t*)))) (*at t within* $\{a\ ..\ b\}$) **if** $t \in \{a\ ..\ b\}$ **for** t
    using *flow-has-derivative*[*of t x*] *assms that*
    by (*auto simp*: *has-derivative-at-withinI*)

  have *compact* $((\lambda t.\ f\ t$ (*flow t0 x t*)) ‘ $\{a\ ..\ b\})$
    using *assms*
    apply (*auto intro*!: *compact-continuous-image continuous-intros*)
    using *local.existence-ivl-empty2* **apply** *fastforce*
     apply (*meson atLeastAtMost-iff general.existence-ivl-subset in-mono*)
    by (*simp add*: *general.flow-in-domain subset-iff*)
  then obtain C **where** $t \in \{a\ ..\ b\} \Longrightarrow$ *norm* ($f\ t$ (*flow t0 x t*)) $\leq C$ **for** t
    by (*fastforce dest*!: *compact-imp-bounded simp*: *bounded-iff intro*: *that*)
  then have $t \in \{a..b\} \Longrightarrow$ *onorm* $(\lambda i.\ i *_R f\ t$ (*flow t0 x t*))) $\leq$ *max 0 C* **for** t
    apply (*subst onorm-scaleR-left*)

apply (*auto simp*: *onorm-id max-def*)
by (*metis diff-0-right diff-mono diff-self norm-ge-zero*)
from *bounded-derivative-imp-lipschitz*[*OF f′ - this*]
have (*max 0 C*)−*lipschitz-on* {*a*..*b*} (*flow t0 x*)
by *auto*
then show *?thesis* **..**
qed

lemma *flow-undefined0*: $t \notin$ *existence-ivl t0 x* $\Longrightarrow$ *flow t0 x t* = *0*
unfolding *flow-def* **by** *auto*

lemma *csols-undefined*: $x \notin X \Longrightarrow$ *csols t0 x* = {}
apply (*auto simp*: *csols-def*)
using *general.existence-ivl-empty2 general.existence-ivl-maximal-segment*
apply *blast*
done

lemmas *existence-ivl-undefined* = *existence-ivl-empty2*

end

2.3 Reverse flow as Sublocale

lemma *range-preflect-0*[*simp*]: *range* (*preflect 0*) = *UNIV*
by (*auto simp*: *preflect-def*)
lemma *range-uminus*[*simp*]: *range uminus* = (*UNIV*::′*a*::*ab-group-add set*)
by *auto*

context *auto-ll-on-open* **begin**

sublocale *rev*: *auto-ll-on-open* −*f* **rewrites** −(−*f*) = *f*
apply *unfold-locales*
using *auto-local-lipschitz auto-open-domain*
unfolding *fun-Compl-def local-lipschitz-minus*
by *auto*

lemma *existence-ivl-eq-rev0*: *existence-ivl0 y* = *uminus* ‘ *rev.existence-ivl0 y* **for** *y*
by (*auto simp*: *existence-ivl-eq-rev rev.existence-ivl0-def preflect-def*)

lemma *rev-existence-ivl-eq0*: *rev.existence-ivl0 y* = *uminus* ‘ *existence-ivl0 y* **for** *y*
using *uminus-uminus-image*[*of rev.existence-ivl0 y*]
by (*simp add*: *existence-ivl-eq-rev0*)

lemma *flow-eq-rev0*: *flow0 y t* = *rev.flow0 y* (−*t*) **for** *y t*
apply (*cases* $t \in$ *existence-ivl0 y*)
subgoal
apply (*subst flow-eq-rev*(*2*), *assumption*)
apply (*subst rev.flow0-def*)
by (*simp add*: *preflect-def*)

subgoal
apply (*frule flow-undefined0*)
by (*auto simp*: *existence-ivl-eq-rev0 rev.flow-undefined0*)
done

lemma *rev-eq-flow*: *rev.flow0* y t = *flow0* y $(-t)$ **for** y t
apply (*subst flow-eq-rev0*)
using *uminus-uminus-image*[*of rev.existence-ivl0 y*]
apply −
apply (*subst* (*asm*) *existence-ivl-eq-rev0*[*symmetric*])
by *auto*

lemma *rev-flow-image-eq*: *rev.flow0* x ‘ S = *flow0* x ‘ (*uminus* ‘ S)
unfolding *rev-eq-flow*[*abs-def*]
by *force*

lemma *flow-image-eq-rev*: *flow0* x ‘ S = *rev.flow0* x ‘ (*uminus* ‘ S)
unfolding *rev-eq-flow*[*abs-def*]
by *force*

end

context *c1-on-open* **begin**

sublocale *rev*: *c1-on-open* $-f$ $-f'$ **rewrites** $-(-f) = f$ **and** $-(-f') = f'$
by (*rule c1-on-open-rev*) *auto*

end

context *c1-on-open-euclidean* **begin**

sublocale *rev*: *c1-on-open-euclidean* $-f$ $-f'$ **rewrites** $-(-f) = f$ **and** $-(-f') = f'$
by *unfold-locales auto*

end

2.4 Autonomous LL ODE : Existence Interval and trapping on the interval

lemma *bdd-above-is-intervalI*: *bdd-above* I
if *is-interval* I $a \leq b$ $a \in I$ $b \notin I$ **for** I::*real set*
by (*meson bdd-above-def is-interval-1 le-cases that*)

lemma *bdd-below-is-intervalI*: *bdd-below* I
if *is-interval* I $a \leq b$ $a \notin I$ $b \in I$ **for** I::*real set*
by (*meson bdd-below-def is-interval-1 le-cases that*)

context *auto-ll-on-open* **begin**

lemma *open-existence-ivl0*:
  assumes $x : x \in X$
  shows $\exists\, a\ b.\ a < 0 \wedge 0 < b \wedge \{a..b\} \subseteq$ *existence-ivl0 x*
proof −
  have *a1*:$0 \in$ *existence-ivl0 x*
    by (*simp add*: *x*)
  have *a2*: *open* (*existence-ivl0 x*)
    by (*simp add*: *x*)
  from *a1 a2* **obtain** *d* **where** $d > 0$ *ball 0 d* $\subseteq$ *existence-ivl0 x*
    using *openE* **by** *blast*
  have $\{-d/2..d/2\} \subseteq$ *ball 0 d*
    using ‹$0 < d$› *dist-norm mem-ball* **by** *auto*
  thus *?thesis*
    by (*smt* ‹$0 < d$› ‹*ball 0 d* $\subseteq$ *existence-ivl0 x*› *divide-minus-left half-gt-zero order-trans*)
qed

lemma *open-existence-ivl′*:
  assumes $x : x \in X$
  obtains *a* **where** $a > 0$ $\{-a..a\} \subseteq$ *existence-ivl0 x*
proof −
  from *open-existence-ivl0*[*OF assms*(*1*)]
  obtain *a b* **where** *ab*: $a < 0$ $0 < b$ $\{a..b\} \subseteq$ *existence-ivl0 x* **by** *auto*
  then have *min* $(-a)$ $b > 0$ **by** *linarith*
  have $\{-$*min* $(-a)$ b .. *min*$(-a)$ $b\} \subseteq \{a..b\}$ **by** *auto*
  thus *?thesis* **using** *ab*(*3*) *that*[*OF* ‹*min* $(-a)$ $b > 0$›] **by** *blast*
qed

lemma *open-existence-ivl-on-compact*:
  assumes *C*: $C \subseteq X$ **and** *compact* C $C \neq \{\}$
  obtains *a* **where** $a > 0$ $\bigwedge x.\ x \in C \Longrightarrow \{-a..a\} \subseteq$ *existence-ivl0 x*
proof −
  from *existence-ivl-cballs*
  have $\forall x{\in}C.\ \exists e{>}0.\ \exists t{>}0.\ \forall y{\in}$*cball x e*. *cball 0 t*$\subseteq$*existence-ivl0 y*
    by (*metis* (*full-types*) *C Int-absorb1 Int-iff UNIV-I*)
  then
  obtain *d′ t′* **where** ∗:
    $\forall x{\in}C.\ 0 < d'\ x \wedge t'\ x > 0 \wedge (\forall y{\in}$*cball x* $(d'\ x)$. *cball 0* $(t'\ x) \subseteq$ *existence-ivl0 y*)
    by *metis*
  with *compactE-image*[*OF* ‹*compact C*›, *of C* $\lambda x.$ *ball x* $(d'\ x)$]
  obtain C' **where** $C' \subseteq C$ **and** [*simp*]: *finite* C' **and** *C-subset*: $C \subseteq (\bigcup c{\in}C'.$ *ball c* $(d'\ c))$
    by *force*
  from *C-subset* ‹$C \neq \{\}$› **have** [*simp*]: $C' \neq \{\}$ **by** *auto*
  define *d* **where** $d = $ *Min* $(d'$ ‘ $C')$
  define *t* **where** $t = $ *Min* $(t'$ ‘ $C')$
  have $t > 0$ **using** ∗ ‹$C' \subseteq C$›

    by (*auto simp*: *t-def*)
  moreover have $\{-t .. t\} \subseteq$ *existence-ivl0 x* **if** $x \in C$ **for** *x*
  proof −
    from *C-subset that* ‹$C' \subseteq C$›
    obtain *c* **where** *c*: $c \in C'$ $x \in$ *ball c* $(d'\ c)$ $c \in C$ **by** *force*
    then have $\{-t .. t\} \subseteq$ *cball 0* $(t'\ c)$
      by (*auto simp*: *abs-real-def t-def minus-le-iff*)
    also
    from *c* **have** *cball 0* $(t'\ c) \subseteq$ *existence-ivl0 x*
      using *[*rule-format*, *OF* ‹$c \in C$›] **by** *auto*
    finally show *?thesis* **.**
  qed
  ultimately show *?thesis* **..**
qed

definition *trapped-forward x K* $\longleftrightarrow$ (*flow0 x* ‘ (*existence-ivl0 x* $\cap$ $\{0..\}$) $\subseteq K$)
  — TODO: use this for backwards trapped, invariant, and all assumptions

definition *trapped-backward x K* $\longleftrightarrow$ (*flow0 x* ‘ (*existence-ivl0 x* $\cap$ $\{..0\}$) $\subseteq K$)

definition *trapped x K* $\longleftrightarrow$ *trapped-forward x K* $\wedge$ *trapped-backward x K*

lemma *trapped-iff-on-existence-ivl0*:
  trapped x K $\longleftrightarrow$ (*flow0 x* ‘ (*existence-ivl0 x*) $\subseteq K$)
  unfolding *trapped-def trapped-forward-def trapped-backward-def*
  apply (*auto*)
 by (*metis IntI atLeast-iff atMost-iff image-subset-iff less-eq-real-def linorder-not-less*)
end

context *auto-ll-on-open* **begin**

lemma *infinite-rev-existence-ivl0-rewrites*:
  $\{0..\} \subseteq$ *rev.existence-ivl0 x* $\longleftrightarrow$ $\{..0\} \subseteq$ *existence-ivl0 x*
  $\{..0\} \subseteq$ *rev.existence-ivl0 x* $\longleftrightarrow$ $\{0..\} \subseteq$ *existence-ivl0 x*
   apply (*auto simp add*: *rev.rev-existence-ivl-eq0 subset-iff*)
  using *neg-le-0-iff-le* **apply** *fastforce*
  using *neg-0-le-iff-le* **by** *fastforce*

lemma *trapped-backward-iff-rev-trapped-forward*:
  trapped-backward x K $\longleftrightarrow$ *rev.trapped-forward x K*
  unfolding *trapped-backward-def rev.trapped-forward-def*
  by (*auto simp add*: *rev-flow-image-eq existence-ivl-eq-rev0 image-subset-iff*)

If solution is trapped in a compact set at some time on its existence interval then it is trapped forever

lemma *trapped-sol-right*:
  — TODO: when building on afp-devel (??? outdated): https://bitbucket.org/isa-afp/afp-devel/commits/0c3edf9248d5389197f248c723b625c419e4d3eb
  assumes *compact K* $K \subseteq X$

assumes $x \in X$ *trapped-forward* x K
shows $\{0..\} \subseteq$ *existence-ivl0* x
proof (*rule ccontr*)
assume $\neg$ $\{0..\} \subseteq$ *existence-ivl0* x
from *this* **obtain** t **where** $0 \leq t$ $t \notin$ *existence-ivl0* x **by** *blast*
then have *bdd*: *bdd-above* (*existence-ivl0* x)
by (*auto intro*!: *bdd-above-is-intervalI* ‹$x \in X$›)
from *flow-leaves-compact-ivl-right* [*OF UNIV-I* ‹$x \in X$› *bdd UNIV-I assms*(*1*−*2*)]
show *False* **by** (*metis assms*(*4*) *trapped-forward-def IntI atLeast-iff image-subset-iff*)
qed

lemma *trapped-sol-right-gen*:
assumes *compact* K $K \subseteq X$
assumes $t \in$ *existence-ivl0* x *trapped-forward* (*flow0* x t) K
shows $\{t..\} \subseteq$ *existence-ivl0* x
proof −
have $x \in X$
using *assms*(*3*) *local.existence-ivl-empty-iff* **by** *fastforce*
have *xtk*: *flow0* x $t \in X$
by (*simp add*: *assms*(*3*) *local.flow-in-domain*)
from *trapped-sol-right*[*OF assms*(*1*−*2*) *xtk assms*(*4*)] **have** $\{0..\} \subseteq$ *existence-ivl0* (*flow0* x t) **.**
thus $\{t..\} \subseteq$ *existence-ivl0* x
using *existence-ivl-trans*[*OF assms*(*3*)]
by (*metis add.commute atLeast-iff diff-add-cancel le-add-same-cancel1 subset-iff*)
qed

lemma *trapped-sol-left*:
— TODO: when building on afp-devel: https://bitbucket.org/isa-afp/afp-devel/commits/0c3edf9248d5389197f248c723b625c419e4d3eb
assumes *compact* K $K \subseteq X$
assumes $x \in X$ *trapped-backward* x K
shows $\{..0\} \subseteq$ *existence-ivl0* x
proof (*rule ccontr*)
assume $\neg$ $\{..0\} \subseteq$ *existence-ivl0* x
from *this* **obtain** t **where** $t \leq 0$ $t \notin$ *existence-ivl0* x **by** *blast*
then have *bdd*: *bdd-below* (*existence-ivl0* x)
by (*auto intro*!: *bdd-below-is-intervalI* ‹$x \in X$›)
from *flow-leaves-compact-ivl-left* [*OF UNIV-I* ‹$x \in X$› *bdd UNIV-I assms*(*1*−*2*)]
show *False*
by (*metis IntI assms*(*4*) *atMost-iff auto-ll-on-open.trapped-backward-def auto-ll-on-open-axioms image-subset-iff*)
qed

lemma *trapped-sol-left-gen*:
assumes *compact* K $K \subseteq X$
assumes $t \in$ *existence-ivl0* x *trapped-backward* (*flow0* x t) K
shows $\{..t\} \subseteq$ *existence-ivl0* x
proof −

have $x \in X$
using *assms*(*3*) *local.existence-ivl-empty-iff* **by** *fastforce*
have *xtk*: *flow0 x t* $\in X$
by (*simp add*: *assms*(*3*) *local.flow-in-domain*)
from *trapped-sol-left*[*OF assms*(*1*−*2*) *xtk assms*(*4*)] **have** $\{..0\} \subseteq$ *existence-ivl0* (*flow0 x t*) **.**
thus $\{..t\} \subseteq$ *existence-ivl0 x*
using *existence-ivl-trans*[*OF assms*(*3*)]
by (*metis add.commute add-le-same-cancel1 atMost-iff diff-add-cancel subset-eq*)
qed

lemma *trapped-sol*:
assumes *compact K* $K \subseteq X$
assumes $x \in X$ *trapped x K*
shows *existence-ivl0 x* = *UNIV*
by (*metis* (*mono-tags*, *lifting*) *assms existence-ivl-zero image-subset-iff interval local.existence-ivl-initial-time-iff local.existence-ivl-subset local.subset-mem-compact-implies-subset-existence-inte order-refl subset-antisym trapped-iff-on-existence-ivl0*)

lemma *regular-locally-noteq*:— TODO: should be true in *ll-on-open-it*
assumes $x \in X$ $f\ x \neq 0$
shows *eventually* (λt. *flow0 x t* $\neq x$) (*at 0*)
proof −
have *nf*:*norm* (*f x*) > *0* **by** (*simp add*: *assms*(*2*))

obtain *a* **where**
a: *a*>*0*
$\{-a--a\} \subseteq$ *existence-ivl0 x*
$0 \in \{-a--a\}$
$\bigwedge t.\ t \in \{-a--a\} \Longrightarrow$ *norm*(*f* (*flow0 x t*) − *f* (*flow0 x 0*)) $\leq$ *norm*(*f x*)/*2*
proof −
from *open-existence-ivl'*[*OF assms*(*1*)]
obtain *a1* **where** *a1*: *a1* > *0* $\{-a1..a1\} \subseteq$ *existence-ivl0 x* **.**
have *continuous* (*at 0*) (λt. *norm*(*f* (*flow0 x t*) − *f* (*flow0 x 0*)))
apply (*auto intro*!: *continuous-intros*)
by (*simp add*: *assms*(*1*) *local.f-flow-continuous*)
then obtain *a2* **where** *a2*>*0*
$\forall t$. *norm t* < *a2* $\longrightarrow$
norm (*f* (*flow0 x t*) − *f* (*flow0 x 0*)) < *norm*(*f x*)/*2*
unfolding *continuous-at-real-range*
by (*metis abs-norm-cancel cancel-comm-monoid-add-class.diff-cancel diff-zero half-gt-zero nf norm-zero*)
then have
t: $\bigwedge t.\ t \in \{-a2<--<a2\} \Longrightarrow$ *norm*(*f* (*flow0 x t*) − *f* (*flow0 x 0*)) $\leq$ *norm*(*f x*)/*2*
by (*smt open-segment-bound*(*2*) *open-segment-bound1 real-norm-def*)
define *a* **where** *a* = *min a1* (*a2*/*2*)
have *t1*:*a* > *0* **unfolding** *a-def* **using** ‹*a1* > *0*› ‹*a2* > *0*› **by** *auto*

then have *t3*:*0* ∈{−*a*−−*a*}
  using *closed-segment-eq-real-ivl* **by** *auto*
have {−*a*−−*a*} ⊆ {−*a1*..*a1*} **unfolding** *a-def* **using** ‹*a1* > *0*› ‹*a2* > *0*›
  using *ODE-Auxiliarities.closed-segment-eq-real-ivl* **by** *auto*
then have *t2*:{−*a*−−*a*} ⊆ *existence-ivl0 x* **using** *a1* **by** *auto*
have {−*a*−−*a*} ⊆ {−*a2*<−−<*a2*} **unfolding** *a-def* **using** ‹*a1* > *0*› ‹*a2* > *0*›
by (*smt Diff-iff closed-segment-eq-real-ivl atLeastAtMost-iff empty-iff half-gt-zero insert-iff pos-half-less segment*(*1*) *subset-eq*)
then have *t4*:⋀*t*. *t* ∈ {−*a*−−*a*} ⟹ *norm*(*f* (*flow0 x t*) − *f* (*flow0 x 0*)) ≤ *norm*(*f x*)/*2* **using** *t* **by** *auto*
show *?thesis* **using** *t1 t2 t3 t4 that* **by** *auto*
qed
have ⋀*t*. *t* ∈ {−*a*−−*a*} ⟹ (*flow0 x has-vector-derivative f* (*flow0 x t*)) (*at t within* {−*a*−−*a*})
  apply (*rule has-vector-derivative-at-within*)
  using *a*(*2*) **by** (*auto intro*!:*flow-has-vector-derivative*)
from *vector-differentiable-bound-linearization*[*OF this - a*(*4*)]
have *nb*:⋀*c d*. {*c*−−*d*} ⊆ {−*a*−−*a*} ⟹
norm (*flow0 x d* − *flow0 x c* − (*d* − *c*) $*_R$ *f* (*flow0 x 0*)) ≤ *norm* (*d* − *c*) ∗ (*norm* (*f x*) / *2*)
  using *a*(*3*) **by** *blast*
have ⋀*t*. *dist t 0* < *a* ⟹ *t* ≠ *0* ⟹ *flow0 x t* ≠ *x*
proof (*rule ccontr*)
  fix *t*
  assume *dist t 0* < *a t* ≠ *0* ¬ *flow0 x t* ≠ *x*
  then have *tx*:*flow0 x t* = *x* **by** *auto*
  have *t* ∈ {−*a*−−*a*}
    using *closed-segment-eq-real-ivl* ‹*dist t 0* < *a*› **by** *auto*
  have *t* > *0* ∨ *t* < *0* **using** ‹*t* ≠ *0*› **by** *linarith*
  moreover {
    assume *t* > *0*
    then have {*0*−−*t*} ⊆ {−*a*−−*a*}
      by (*simp add*: ‹*t* ∈ {−*a*−−*a*}› *a*(*3*) *subset-closed-segment*)
    from *nb*[*OF this*] **have**
      norm (*flow0 x t* − *x* − *t* $*_R$ *f x*) ≤ *norm t* ∗ (*norm* (*f x*) / *2*)
      by (*simp add*: *assms*(*1*))
    then have *norm* (*t* $*_R$ *f x*) ≤ *norm t* ∗ (*norm* (*f x*) / *2*) **using** *tx* **by** *auto*
    then have *False* **using** *nf*
      using ‹*0* < *t*› **by** *auto*
  }
  moreover {
    assume *t* < *0*
    then have {*t*−−*0*} ⊆ {−*a*−−*a*}
      by (*simp add*: ‹*t* ∈ {−*a*−−*a*}› *a*(*3*) *subset-closed-segment*)
    from *nb*[*OF this*] **have**
      norm (*x* − *flow0 x t* + *t* $*_R$ *f x*) ≤ *norm t* ∗ (*norm* (*f x*) / *2*)
      by (*simp add*: *assms*(*1*))
    then have *norm* (*t* $*_R$ *f x*) ≤ *norm t* ∗ (*norm* (*f x*) / *2*) **using** *tx* **by** *auto*

then have *False* **using** *nf*
using ‹$t < 0$› **by** *auto*
}
ultimately show *False* **by** *blast*
qed
thus *?thesis* **unfolding** *eventually-at*
using *a*(*1*) **by** *blast*
qed

lemma *compact-max-time-flow-in-closed*:
assumes *closed M* **and** *t-ex*: $t \in$ *existence-ivl0 x*
shows *compact* $\{s \in \{0..t\}.$ *flow0 x* ‘ $\{0..s\} \subseteq M\}$ (**is** *compact ?C*)
unfolding *compact-eq-bounded-closed*
proof
have *bounded* $\{0\ ..\ t\}$ **by** *auto*
then show *bounded ?C*
by (*rule bounded-subset*) *auto*
show *closed ?C*
unfolding *closed-def*
proof (*rule topological-space-class.openI*, *clarsimp*)
— TODO: there must be a more abstract argument for this, e.g., with ⟦*closed ?s*; *continuous-on ?s ?f*; *closed ?B*⟧ $\Longrightarrow$ *closed* (*?f* $-$‘ *?B* $\cap$ *?s*) and then reasoning about the connected component around 0?
fix *s*
assume *notM*: $s \le t \longrightarrow 0 \le s \longrightarrow \neg$ *flow0 x* ‘ $\{0..s\} \subseteq M$
consider $0 \le s$ $s \le t$ *flow0 x s* $\notin M$ | $0 \le s$ $s \le t$ *flow0 x s* $\in M$ | $s < 0$ | $s > t$
by *arith*
then show $\exists T.$ *open* $T \wedge s \in T \wedge T \subseteq -\{s.\ 0 \le s \wedge s \le t \wedge$ *flow0 x* ‘ $\{0..s\} \subseteq M\}$
proof *cases*
assume *s*: $0 \le s$ $s \le t$ **and** *sM*: *flow0 x s* $\notin M$
have *isCont* (*flow0 x*) *s*
using *s ivl-subset-existence-ivl*[*OF t-ex*]
by (*auto intro*!: *flow-continuous*)
from *this*[*unfolded continuous-at-open*, *rule-format*, *of* $-M$] *sM* ‹*closed M*›
obtain *S* **where** *open S* $s \in S$ ($\forall x' \in S.$ *flow0 x x'* $\in -M$)
by *auto*
then show *?thesis*
by (*force intro*!: *exI*[**where** *x*=*S*])
next
assume *s*: $0 \le s$ $s \le t$ **and** *sM*: *flow0 x s* $\in M$
from *this notM* **obtain** *s0* **where** *s0*: $0 \le s0$ $s0 < s$ *flow0 x s0* $\notin M$
by *force*
from *order-tendstoD*(*1*)[*OF tendsto-ident-at* ‹$s0 < s$›, *of UNIV*, *unfolded eventually-at-topological*]
obtain *S* **where** *open S* $s \in S$ $\bigwedge x.\ x \in S \Longrightarrow x \neq s \Longrightarrow s0 < x$
by *auto*
then show *?thesis* **using** *s0*

    by (*auto simp*: *intro*!: *exI*[**where** *x*=*S*]) (*smt atLeastAtMost-iff image-subset-iff*)
  qed (*force intro*: *exI*[**where** *x*={*t*<..}] *exI*[**where** *x*={..<*0*}])+
 qed
qed

lemma *flow-in-closed-max-timeE*:
 assumes *closed M* $t \in$ *existence-ivl0 x* $0 \leq t$ $x \in M$
 obtains T **where** $0 \leq T$ $T \leq t$ *flow0 x* ‘ $\{0..T\} \subseteq M$
  $\bigwedge s'. \ 0 \leq s' \Longrightarrow s' \leq t \Longrightarrow$ *flow0 x* ‘ $\{0..s'\} \subseteq M \Longrightarrow s' \leq T$
proof −
 let $?C = \{s \in \{0..t\}.$ *flow0 x* ‘ $\{0..s\} \subseteq M\}$
 have $?C \neq \{\}$
  using *assms*
  using *local.mem-existence-ivl-iv-defined*
  by (*auto intro*!: *exI*[**where** *x*=*0*])
 from *compact-max-time-flow-in-closed*[*OF assms*(*1*,*2*)]
 have *compact ?C* **.**
 from *compact-attains-sup*[*OF this* ‹$?C \neq \{\}$›]
 obtain *s* **where** *s*: $0 \leq s$ $s \leq t$ *flow0 x* ‘ $\{0..s\} \subseteq M$
  and *s-max*: $\bigwedge s'. \ 0 \leq s' \Longrightarrow s' \leq t \Longrightarrow$ *flow0 x* ‘ $\{0..s'\} \subseteq M \Longrightarrow s' \leq s$
  by *auto*
 then show *?thesis* **..**
qed

lemma *flow-leaves-closed-at-frontierE*:
 assumes *closed M* **and** *t-ex*: $t \in$ *existence-ivl0 x* **and** $0 \leq t$ $x \in M$ *flow0 x t* $\notin$ M
 obtains *s* **where** $0 \leq s$ $s < t$ *flow0 x* ‘ $\{0..s\} \subseteq M$
  flow0 x s $\in$ *frontier M*
  $\exists_F$ *s′ in at-right s*. *flow0 x s′* $\notin M$
proof −
 from *flow-in-closed-max-timeE*[*OF assms*(*1*−*4*)] *assms*(*5*)
 obtain *s* **where** *s*: $0 \leq s$ $s < t$ *flow0 x* ‘ $\{0..s\} \subseteq M$
  and *s-max*: $\bigwedge s'. \ 0 \leq s' \Longrightarrow s' \leq t \Longrightarrow$ *flow0 x* ‘ $\{0..s'\} \subseteq M \Longrightarrow s' \leq s$
  by (*smt atLeastAtMost-iff image-subset-iff*)
 note *s*
 moreover
 have *flow0 x s* $\notin$ *interior M*
 proof
  assume *interior*: *flow0 x s* $\in$ *interior M*
  have $s \in$ *existence-ivl0 x* **using** *ivl-subset-existence-ivl*[*OF* ‹$t \in$ -›] *s* **by** *auto*
   from *flow-continuous*[*OF this*, *THEN isContD*, *THEN topological-tendstoD*, *OF open-interior interior*]
  have $\forall_F$ *s′ in at s*. *flow0 x s′* $\in$ *interior M* **by** *auto*
  then have $\forall_F$ *s′ in at-right s*. *flow0 x s′* $\in$ *interior M*
   by (*auto simp*: *eventually-at-split*)
  moreover have $\forall_F$ *s′ in at-right s*. $s' < t$
   using *tendsto-ident-at* ‹$s < t$›
   by (*rule order-tendstoD*)

ultimately have $\forall_F$ *s′ in at-right s. flow0 x s′* $\in$ *M* $\wedge$ *s′* $<$ *t*
by *eventually-elim* (*use interior-subset*[*of M*] **in** *auto*)
then obtain *s′* **where** *s′*: *s* $<$ *s′ s′* $<$ *t* $\bigwedge$*y. y* $>$ *s* $\Longrightarrow$ *y* $\leq$ *s′* $\Longrightarrow$ *flow0 x y* $\in$ *M*
by (*auto simp*: *eventually-at-right-field-le*)
have *s′-ivl*: *flow0 x* ‘ {*0..s′*} $\subseteq$ *M*
proof *safe*
fix *s′′* **assume** *s′′* $\in$ {*0* .. *s′*}
then show *flow0 x s′′* $\in$ *M*
using *s interior-subset*[*of M*] *s′*
by (*cases s′′* $\leq$ *s*) *auto*
qed
with *s-max*[*of s′*] ‹*s′* $<$ *t*› ‹*0* $\leq$ *s*› ‹*s* $<$ *s′*› **show** *False* **by** *auto*
qed
then have *flow0 x s* $\in$ *frontier M*
using *s closure-subset*[*of M*]
by (*force simp*: *frontier-def*)
moreover
have *compact* (*flow0 x* −‘ *M* $\cap$ {*s..t*}) (**is** *compact ?C*)
unfolding *compact-eq-bounded-closed*
proof
have *bounded* {*s* .. *t*} **by** *simp*
then show *bounded ?C*
by (*rule bounded-subset*) *auto*
show *closed ?C*
using ‹*closed M*› *assms mem-existence-ivl-iv-defined*(*2*)[*OF t-ex*] *ivl-subset-existence-ivl*[*OF t-ex*] ‹*0* $\leq$ *s*›
by (*intro closed-vimage-Int*) (*auto intro*!: *continuous-intros*)
qed
have $\exists_F$ *s′ in at-right s. flow0 x s′* $\notin$ *M*
apply (*rule ccontr*)
unfolding *not-frequently*
proof −
assume $\forall_F$ *s′ in at-right s.* $\neg$ *flow0 x s′* $\notin$ *M*
moreover have $\forall_F$ *s′ in at-right s. s′* $<$ *t*
using *tendsto-ident-at* ‹*s* $<$ *t*›
by (*rule order-tendstoD*)
ultimately have $\forall_F$ *s′ in at-right s. flow0 x s′* $\in$ *M* $\wedge$ *s′* $<$ *t* **by** *eventually-elim auto*
then obtain *s′* **where** *s′*: *s* $<$ *s′*
$\bigwedge$*y. y* $>$ *s* $\Longrightarrow$ *y* $<$ *s′* $\Longrightarrow$ *flow0 x y* $\in$ *M*
$\bigwedge$*y. y* $>$ *s* $\Longrightarrow$ *y* $<$ *s′* $\Longrightarrow$ *y* $<$ *t*
by (*auto simp*: *eventually-at-right-field*)
define *s′′* **where** *s′′* = (*s* + *s′*) / *2*
have *0* $\leq$ *s′′ s′′* $\leq$ *t* *s* $<$ *s′′ s′′* $<$ *s′*
using *s s′*
by (*auto simp del*: *divide-le-eq-numeral1 le-divide-eq-numeral1 simp*: *s′′-def*) *fastforce*
then have *flow0 x* ‘ {*0..s′′*} $\subseteq$ *M*

using *s s′*
apply *auto*
subgoal for *u*
by (*cases u≤s*) *auto*
done
from *s-max*[*OF* ‹*0* ≤ *s′′*› ‹*s′′*≤ *t*› *this*] ‹*s′′* > *s*›
show *False* **by** *simp*
qed
ultimately show *?thesis* **..**
qed

2.5 Connectedness

lemma *fcontX*:
shows *continuous-on X f*
using *auto-local-lipschitz local-lipschitz-continuous-on* **by** *blast*

lemma *fcontx*:
assumes *x* ∈ *X*
shows *continuous* (*at x*) *f*
proof −
have *open X* **by** *simp*
from *continuous-on-eq-continuous-at*[*OF this*]
show *?thesis* **using** *fcontX assms*(*1*) **by** *blast*
qed

lemma *continuous-at-imp-cball*:
assumes *continuous* (*at x*) *g*
assumes *g x* > (*0*::*real*)
obtains *r* **where** *r* > *0* ∀ *y* ∈ *cball x r*. *g y* > *0*
proof −
from *assms*(*1*)
obtain *d* **where** *d*>*0 g* ‘ (*ball x d*) ⊆ *ball* (*g x*) ((*g x*)/*2*)
by (*meson assms*(*2*) *continuous-at-ball half-gt-zero*)
then have ∀ *y* ∈ *cball x* (*d*/*2*). *g y* > *0*
by (*smt assms*(*2*) *dist-norm image-subset-iff mem-ball mem-cball pos-half-less real-norm-def*)
thus *?thesis*
using ‹*0* < *d*› *that half-gt-zero* **by** *blast*
qed

flow0 is *path-connected*

lemma *flow0-path-connected-time*:
assumes *ts* ⊆ *existence-ivl0 x path-connected ts*
shows *path-connected* (*flow0 x* ‘ *ts*)
proof −
have *continuous-on ts* (*flow0 x*)
by (*meson assms continuous-on-sequentially flow-continuous-on subsetD*)
from *path-connected-continuous-image*[*OF this assms*(*2*)]

show *?thesis* **.**
qed

lemma *flow0-path-connected*:
  assumes *path-connected D*
    path-connected ts
    $\bigwedge x.\ x \in D \Longrightarrow ts \subseteq$ *existence-ivl0 x*
  shows *path-connected* (($\lambda(x, y)$. *flow0 x y*) ‘ ($D \times ts$))
proof −
  have $D \times ts \subseteq$ *Sigma X existence-ivl0*
    using *assms*(*3*) *subset-iff* **by** *fastforce*
  then have *a1*:*continuous-on* ($D \times ts$) ($\lambda(x, y)$. *flow0 x y*)
    using *flow-continuous-on-state-space continuous-on-subset* **by** *blast*
  have *a2* : *path-connected* ($D \times ts$) **using** *path-connected-Times assms* **by** *auto*
  from *path-connected-continuous-image*[*OF a1 a2*]
  show *?thesis* **.**
qed

end

2.6 Return Time and Implicit Function Theorem

context *c1-on-open-euclidean* **begin**

lemma *flow-implicit-function*:
  — TODO: generalization of ⟦*returns-to* {$x \in$ *?S*. *?s x* = *0*} *?x*; *closed ?S*; $\bigwedge x$. (*?s has-derivative blinfun-apply* (*?Ds x*)) (*at x*); *isCont ?Ds* (*poincare-map* {$x \in$ *?S*. *?s x* = *0*} *?x*); *blinfun-apply* (*?Ds* (*poincare-map* {$x \in$ *?S*. *?s x* = *0*} *?x*)) (*f* (*poincare-map* {$x \in$ *?S*. *?s x* = *0*} *?x*)) ≠ *0*; $\bigwedge u\ e$. ⟦*?s* (*flow0 ?x* (*u ?x*)) = *0*; *u ?x* = *return-time* {$x \in$ *?S*. *?s x* = *0*} *?x*; $\bigwedge y.\ y \in$ *cball ?x e* ⟹ *?s* (*flow0 y* (*u y*)) = *0*; *continuous-on* (*cball ?x e*) *u*; (λt. (*t*, *u t*)) ‘ *cball ?x e* ⊆ *Sigma X existence-ivl0*; *0* < *e*; (*u has-derivative blinfun-apply* (− *blinfun-scaleR-left* (*inverse* (*blinfun-apply* (*?Ds* (*poincare-map* {$x \in$ *?S*. *?s x* = *0*} *?x*)) (*f* (*poincare-map* {$x \in$ *?S*. *?s x* = *0*} *?x*)))) o_L (*?Ds* (*poincare-map* {$x \in$ *?S*. *?s x* = *0*} *?x*) o_L *flowderiv ?x* (*return-time* {$x \in$ *?S*. *?s x* = *0*} *?x*)) o_L *embed1-blinfun*)) (*at ?x*)⟧ ⟹ *?thesis*⟧ ⟹ *?thesis*!
  fixes *s*::*'a*::*euclidean-space* ⇒ *real* **and** *S*::*'a set*
  assumes *t*: *t* ∈ *existence-ivl0 x* **and** *x*: *x* ∈ *X* **and** *st*: *s* (*flow0 x t*) = *0*
  assumes *Ds*: $\bigwedge x$. (*s has-derivative blinfun-apply* (*Ds x*)) (*at x*)
  assumes *DsC*: *isCont Ds* (*flow0 x t*)
  assumes *nz*: *Ds* (*flow0 x t*) (*f* (*flow0 x t*)) ≠ *0*
  obtains *u e*
  where *s* (*flow0 x* (*u x*)) = *0*
    u x = *t*
    ($\bigwedge y$. *y* ∈ *cball x e* ⟹ *s* (*flow0 y* (*u y*)) = *0*)
    continuous-on (*cball x e*) *u*
    (λt. (*t*, *u t*)) ‘ *cball x e* ⊆ *Sigma X existence-ivl0*
    0 < *e* (*u has-derivative* (− *blinfun-scaleR-left*
        (*inverse* (*blinfun-apply* (*Ds* (*flow0 x t*)) (*f* (*flow0 x t*)))) o_L
          (*Ds* (*flow0 x t*) o_L *flowderiv x t*) o_L *embed1-blinfun*)) (*at x*)

proof −
note [*derivative-intros*] = *has-derivative-compose*[*OF* - *Ds*]
have *cont-s*: *continuous-on UNIV s* **by** (*rule has-derivative-continuous-on*[*OF Ds*])
note *cls*[*simp*, *intro*] = *closed-levelset*[*OF cont-s*]
then have *xt1*: (*x*, *t*) ∈ *Sigma X existence-ivl0*
by (*auto simp*: *t x*)
have *D*: (⋀*x*. *x* ∈ *Sigma X existence-ivl0* ⟹
((λ(*x*, *t*). *s* (*flow0 x t*)) *has-derivative*
blinfun-apply (*Ds* (*flow0* (*fst x*) (*snd x*)) o_L (*flowderiv* (*fst x*) (*snd x*))))
(*at x*))
by (*auto intro*!: *derivative-eq-intros*)
have *C*: *isCont* (λ*x*. *Ds* (*flow0* (*fst x*) (*snd x*)) o_L *flowderiv* (*fst x*) (*snd x*))
(*x*, *t*)
using *flowderiv-continuous-on*[*unfolded continuous-on-eq-continuous-within*,
rule-format, *OF xt1*]
using *at-within-open*[*OF xt1 open-state-space*]
by (*auto intro*!: *continuous-intros tendsto-eq-intros x t*
isCont-tendsto-compose[*OF DsC*, *unfolded poincare-map-def*]
simp: *split-beta′ isCont-def*)
have *Z*: (*case* (*x*, *t*) *of* (*x*, *t*) ⇒ *s* (*flow0 x t*)) = *0*
by (*auto simp*: *st*)
have *I1*: *blinfun-scaleR-left* (*inverse* (*Ds* (*flow0 x t*)(*f* (*flow0 x t*)))) o_L
((*Ds* (*flow0* (*fst* (*x*, *t*))
(*snd* (*x*, *t*))) o_L
flowderiv (*fst* (*x*, *t*))
(*snd* (*x*, *t*))) o_L
embed2-blinfun)
= 1_L
using *nz*
by (*auto intro*!: *blinfun-eqI*
simp: *flowderiv-def blinfun.bilinear-simps inverse-eq-divide poincare-map-def*)
have *I2*: ((*Ds* (*flow0* (*fst* (*x*, *t*))
(*snd* (*x*, *t*))) o_L
flowderiv (*fst* (*x*, *t*))
(*snd* (*x*, *t*))) o_L
embed2-blinfun) o_L *blinfun-scaleR-left* (*inverse* (*Ds* (*flow0 x t*)(*f* (*flow0 x t*))))
= 1_L
using *nz*
by (*auto intro*!: *blinfun-eqI*
simp: *flowderiv-def blinfun.bilinear-simps inverse-eq-divide poincare-map-def*)
show *?thesis*
apply (*rule implicit-function-theorem*[**where** *f*=λ(*x*, *t*). *s* (*flow0 x t*)
and *S*=*Sigma X existence-ivl0*, *OF D xt1 open-state-space order-refl C Z I1 I2*])
apply *blast*
unfolding *split-beta′ fst-conv snd-conv poincare-map-def*[*symmetric*]
..
qed

lemma *flow-implicit-function-at*:
fixes *s*::*'a*::*euclidean-space* ⇒ *real* **and** *S*::*'a set*
assumes *x*: *x* ∈ *X* **and** *st*: *s x* = *0*
assumes *Ds*: ⋀*x*. (*s has-derivative blinfun-apply* (*Ds x*)) (*at x*)
assumes *DsC*: *isCont Ds x*
assumes *nz*: *Ds x* (*f x*) ≠ *0*
assumes *pos*: *e* > *0*
obtains *u d*
where
0 < *d*
u x = *0*
⋀*y*. *y* ∈ *cball x d* ⟹ *s* (*flow0 y* (*u y*)) = *0*
⋀*y*. *y* ∈ *cball x d* ⟹ |*u y*| < *e*
⋀*y*. *y* ∈ *cball x d* ⟹ *u y* ∈ *existence-ivl0 y*
continuous-on (*cball x d*) *u*
(*u has-derivative* −*Ds x* $/_R$ (*Ds x*) (*f x*)) (*at x*)
proof −
have *x0*: *flow0 x 0* = *x* **by** (*simp add*: *x*)
from *flow-implicit-function*[*OF existence-ivl-zero*[*OF x*] *x*, *unfolded x0*, *of s*, *OF st Ds DsC nz*]
obtain *u d0* **where**
s0: *s* (*flow0 x* (*u x*)) = *0*
and *u0*: *u x* = *0*
and *u*: ⋀*y*. *y* ∈ *cball x d0* ⟹ *s* (*flow0 y* (*u y*)) = *0*
and *uc*: *continuous-on* (*cball x d0*) *u*
and *uex*: (λ*t*. (*t*, *u t*)) ‘ *cball x d0* ⊆ *Sigma X existence-ivl0*
and *d0*: *0* < *d0*
and *u'*: (*u has-derivative*
blinfun-apply
(− *blinfun-scaleR-left* (*inverse* (*blinfun-apply* (*Ds x*) (*f x*))) o_L (*Ds x* o_L *flowderiv x 0*) o_L *embed1-blinfun*))
(*at x*)
by *blast*
have *at x within cball x d0* = *at x* **by** (*rule at-within-interior*) (*auto simp*: ‹*0* < *d0*›)
then have (*u* ⟶ *0*) (*at x*)
using *uc d0* **by** (*auto simp*: *continuous-on-def u0 dest*!: *bspec*[**where** *x*=*x*])
from *tendstoD*[*OF this* ‹*0* < *e*›] *pos u0*
obtain *d1* **where** *d1*: *0* < *d1* ⋀*xa*. *dist xa x* ≤ *d1* ⟹ |*u xa*| < *e*
unfolding *eventually-at-le*
by *force*
define *d* **where** *d* = *min d0 d1*
have *0* < *d* **by** (*auto simp*: *d-def d0 d1*)
moreover note *u0*
moreover have ⋀*y*. *y* ∈ *cball x d* ⟹ *s* (*flow0 y* (*u y*)) = *0* **by** (*auto intro*!: *u simp*: *d-def*)
moreover have ⋀*y*. *y* ∈ *cball x d* ⟹ |*u y*| < *e* **using** *d1* **by** (*auto simp*: *d-def dist-commute*)

moreover have $\bigwedge y.\ y \in cball\ x\ d \Longrightarrow u\ y \in existence\text{-}ivl0\ y$
using *uex* **by** (*force simp*: *d-def*)
moreover have *continuous-on* (*cball x d*) *u*
using *uc* **by** (*rule continuous-on-subset*) (*auto simp*: *d-def*)
moreover
have (u *has-derivative* $-Ds\ x\ /_R\ (Ds\ x)\ (f\ x))\ (at\ x)$
using u'
by (*rule has-derivative-subst*) (*auto intro*!: *ext simp*: *x x0 flowderiv-def blinfun.bilinear-simps*)
ultimately show *?thesis* **..**
qed

lemma *returns-to-implicit-function-gen*:
— TODO: generalizes proof of ⟦*returns-to* $\{x \in ?S.\ ?s\ x = 0\}$ *?x*; *closed ?S*; $\bigwedge x.$ (*?s has-derivative blinfun-apply* (*?Ds x*)) (*at x*); *isCont ?Ds* (*poincare-map* $\{x \in ?S.\ ?s\ x = 0\}$ *?x*); *blinfun-apply* (*?Ds* (*poincare-map* $\{x \in ?S.\ ?s\ x = 0\}$ *?x*)) (*f* (*poincare-map* $\{x \in ?S.\ ?s\ x = 0\}$ *?x*)) $\neq 0$; $\bigwedge u\ e.$ ⟦*?s* (*flow0 ?x* (*u ?x*)) $= 0$; *u ?x* $=$ *return-time* $\{x \in ?S.\ ?s\ x = 0\}$ *?x*; $\bigwedge y.\ y \in cball\ ?x\ e \Longrightarrow\ ?s\ (flow0\ y\ (u\ y)) = 0$; *continuous-on* (*cball ?x e*) *u*; ($\lambda t.\ (t,\ u\ t)$) ‘ *cball ?x e* $\subseteq$ *Sigma X existence-ivl0*; $0 < e$; (*u has-derivative blinfun-apply* ($-$ *blinfun-scaleR-left* (*inverse* (*blinfun-apply* (*?Ds* (*poincare-map* $\{x \in ?S.\ ?s\ x = 0\}$ *?x*)) (*f* (*poincare-map* $\{x \in ?S.\ ?s\ x = 0\}$ *?x*)))) o_L (*?Ds* (*poincare-map* $\{x \in ?S.\ ?s\ x = 0\}$ *?x*) o_L *flowderiv ?x* (*return-time* $\{x \in ?S.\ ?s\ x = 0\}$ *?x*)) o_L *embed1-blinfun*)) (*at ?x*)⟧ $\Longrightarrow$ *?thesis*⟧ $\Longrightarrow$ *?thesis*!
fixes *s*::*'a*::*euclidean-space* $\Rightarrow$ *real*
assumes *rt*: *returns-to* $\{x \in S.\ s\ x = 0\}$ *x* (**is** *returns-to ?P x*)
assumes *cS*: *closed S*
assumes *Ds*: $\bigwedge x.$ (*s has-derivative blinfun-apply* (*Ds x*)) (*at x*)
isCont Ds (*poincare-map ?P x*)
Ds (*poincare-map ?P x*) (*f* (*poincare-map ?P x*)) $\neq 0$
obtains *u e*
where *s* (*flow0 x* (*u x*)) $= 0$
u x $=$ *return-time ?P x*
($\bigwedge y.\ y \in cball\ x\ e \Longrightarrow s\ (flow0\ y\ (u\ y)) = 0$)
continuous-on (*cball x e*) *u*
($\lambda t.\ (t,\ u\ t)$) ‘ *cball x e* $\subseteq$ *Sigma X existence-ivl0*
$0 < e$ (*u has-derivative* ($-$ *blinfun-scaleR-left*
(*inverse* (*blinfun-apply* (*Ds* (*poincare-map ?P x*)) (*f* (*poincare-map ?P x*)))) o_L
(*Ds* (*poincare-map ?P x*) o_L *flowderiv x* (*return-time ?P x*)) o_L *embed1-blinfun*)) (*at x*)
proof $-$
note [*derivative-intros*] $=$ *has-derivative-compose*[*OF - Ds*(*1*)]
have *cont-s*: *continuous-on UNIV s* **by** (*rule has-derivative-continuous-on*[*OF Ds*(*1*)])
note *cls*[*simp*, *intro*] $=$ *closed-levelset*[*OF cont-s*]
let *?t1* $=$ *return-time ?P x*
have *cls*[*simp*, *intro*]: *closed* $\{x \in S.\ s\ x = 0\}$
by (*rule closed-levelset-within*) (*auto intro*!: *cS continuous-on-subset*[*OF cont-s*])

```
  have ∗: poincare-map ?P x = flow0 x (return-time {x ∈ S. s x = 0} x)
    by (simp add: poincare-map-def)
  have return-time {x ∈ S. s x = 0} x ∈ existence-ivl0 x
    x ∈ X
    s (poincare-map ?P x) = 0
    using poincare-map-returns rt
    by (auto intro!: return-time-exivl rt)
  note E = flow-implicit-function[of return-time ?P x x s Ds, OF this[unfolded ∗]
Ds[unfolded ∗],
      folded ∗]
  show ?thesis
    by (rule E) rule
qed
```

c.f. Perko Section 3.7 Lemma 2 part 1.

```
lemma flow-transversal-surface-finite-intersections:
  fixes s::'a ⇒ 'b::real-normed-vector
    and Ds::'a ⇒ 'a ⇒L 'b
  assumes closed S
  assumes ⋀x. (s has-derivative (Ds x)) (at x)
  assumes ⋀x. x ∈ S ⟹ s x = 0 ⟹ Ds x (f x) ≠ 0
  assumes a ≤ b {a .. b} ⊆ existence-ivl0 x
  shows finite {t∈{a..b}. flow0 x t ∈ {x ∈ S. s x = 0}}
    — TODO: define notion of (compact/closed)-(continuous/differentiable/C1)-
surface?
proof cases
  note Ds = ‹⋀x. (s has-derivative (Ds x)) (at x)›
  note transversal = ‹⋀x. x ∈ S ⟹ s x = 0 ⟹ Ds x (f x) ≠ 0›
  assume a < b
  show ?thesis
  proof (rule ccontr)
    let ?S = {x ∈ S. s x = 0}
    let ?T = {t∈{a..b}. flow0 x t ∈ {x ∈ S. s x = 0}}
    define φ where φ = flow0 x
    have [THEN continuous-on-compose2, continuous-intros]: continuous-on S s
    by (auto simp: intro!: has-derivative-continuous-on Ds intro: has-derivative-at-withinI)
    assume infinite ?T
    from compact-sequentialE[OF compact-Icc[of a b] this]
    obtain t tl where t: t n ∈ ?T flow0 x (t n) ∈ ?S t n ∈ {a .. b} t n ≠ tl
      and tl: t ⟶ tl tl ∈ {a..b}
    for n
      by force
    have tl-ex: tl ∈ existence-ivl0 x using ‹{a .. b} ⊆ existence-ivl0 x› ‹tl ∈ {a ..
b}› by auto
    have closed ?S
      by (auto intro!: closed-levelset-within ‹closed S› continuous-intros)
    moreover
    have ∀ n. flow0 x (t n) ∈ ?S
      using t by auto
```

moreover
have *flow-t*: $(\lambda n.\ \mathit{flow0}\ x\ (t\ n)) \longrightarrow \mathit{flow0}\ x\ tl$
by (*auto intro*!: *tendsto-eq-intros tl-ex tl*)
ultimately have $\mathit{flow0}\ x\ tl \in\ ?S$
by (*metis* (*no-types*, *lifting*) *closed-sequentially*)

let $?qt = \lambda t.\ (\mathit{flow0}\ x\ t - \mathit{flow0}\ x\ tl) /_R (t - tl)$
from *flow-has-vector-derivative*[*OF tl-ex*, *THEN has-vector-derivative-withinD*]
have *qt-tendsto*: $?qt\ {-tl\rightarrow}\ f\ (\mathit{flow0}\ x\ tl)$ **.**
let $?q = \lambda n.\ ?qt\ (t\ n)$
have *filterlim t* (*at tl*) *sequentially*
using *tl*(*1*)
by (*rule filterlim-atI*) (*simp add*: *t*)
with *qt-tendsto* **have** $?q \longrightarrow f\ (\mathit{flow0}\ x\ tl)$
by (*rule filterlim-compose*)
then have $((\lambda n.\ Ds\ (\mathit{flow0}\ x\ tl)\ (?q\ n))) \longrightarrow Ds\ (\mathit{flow0}\ x\ tl)\ (f\ (\mathit{flow0}\ x\ tl))$
by (*auto intro*!: *tendsto-intros*)
moreover

from *flow-lipschitzE*[*OF* ‹$\{a\ ..\ b\} \subseteq$ *existence-ivl0 x*›] **obtain** L' **where** L': L'*−lipschitz-on* $\{a..b\}$ (*flow0 x*) **.**
define L **where** $L = L' + 1$
from *lipschitz-on-le*[*OF* L', *of L*] *lipschitz-on-nonneg*[*OF* L']
have L: L*−lipschitz-on* $\{a\ ..\ b\}$ (*flow0 x*) **and** $L > 0$
by (*auto simp*: *L-def*)
from *flow-lipschitzE*[*OF* ‹$\{a\ ..\ b\} \subseteq$ *existence-ivl0 x*›] **obtain** L' **where** L'*−lipschitz-on* $\{a..b\}$ (*flow0 x*) **.**
— TODO: is this reasoning (below) with this Lipschitz constant really necessary?
have *s*[*simp*]: $s\ (\mathit{flow0}\ x\ (t\ n)) = 0s\ (\mathit{flow0}\ x\ tl) = 0$
for *n*
using *t* ‹$\mathit{flow0}\ x\ tl \in\ ?S$›
by *auto*

from *Ds*(*1*)[*of flow0 x tl*, *unfolded has-derivative-within*]
have $(\lambda y.\ (1\ /\ norm\ (y - \mathit{flow0}\ x\ tl)) *_R (s\ y - (s\ (\mathit{flow0}\ x\ tl) + \mathit{blinfun\text{-}apply}\ (Ds\ (\mathit{flow0}\ x\ tl))\ (y - \mathit{flow0}\ x\ tl))))\ {-\mathit{flow0}\ x\ tl\rightarrow}\ 0$
by *auto*
then have $((\lambda y.\ (1\ /\ norm\ (y - \mathit{flow0}\ x\ tl)) *_R (s\ y - (s\ (\mathit{flow0}\ x\ tl) + \mathit{blinfun\text{-}apply}\ (Ds\ (\mathit{flow0}\ x\ tl))\ (y - \mathit{flow0}\ x\ tl)))) \longrightarrow 0)$
(*nhds* (*flow0 x tl*))
by (*rule tendsto-nhds-continuousI*) *simp*

from *filterlim-compose*[*OF this flow-t*]
have $(\lambda xa.\ (\mathit{blinfun\text{-}apply}\ (Ds\ (\mathit{flow0}\ x\ tl))\ (\mathit{flow0}\ x\ (t\ xa) - \mathit{flow0}\ x\ tl)) /_R\ norm\ (\mathit{flow0}\ x\ (t\ xa) - \mathit{flow0}\ x\ tl))$
$\longrightarrow 0$
using *t*
by (*auto simp*: *inverse-eq-divide tendsto-minus-cancel-right*)

from *tendsto-mult*[*OF tendsto-const*[*of L*] *tendsto-norm*[*OF this*, *simplified*, *simplified divide-inverse-commute*[*symmetric*]]]— TODO: uuugly
have *Ds0*: (λ*xa. norm* (*blinfun-apply* (*Ds* (*flow0 x tl*)) (*flow0 x* (*t xa*) $-$ *flow0 x tl*)) / (*norm* (*flow0 x* (*t xa*) $-$ *flow0 x tl*)/(*L*))) $\longrightarrow$ *0*
by (*auto simp*: *ac-simps*)

from - *Ds0* **have** ((λ*n. Ds* (*flow0 x tl*) (*?q n*)) $\longrightarrow$ *0*)
apply (*rule Lim-null-comparison*)
apply (*rule eventuallyI*)
unfolding *norm-scaleR blinfun.scaleR-right abs-inverse divide-inverse-commute*[*symmetric*]
subgoal for *n*
apply (*cases flow0 x* (*t n*) = *flow0 x tl*)
subgoal by (*simp add*: *blinfun.bilinear-simps*)
subgoal
apply (*rule divide-left-mono*)
using *lipschitz-onD*[*OF L*, *of t n tl*] ‹*0* < *L*› *t*(*3*) *tl*(*2*)
by (*auto simp*: *algebra-split-simps zero-less-divide-iff dist-norm pos-divide-le-eq intro*!: *add-pos-nonneg*)
done
done
ultimately have *Ds* (*flow0 x tl*) (*f* (*flow0 x tl*)) = *0*
by (*rule LIMSEQ-unique*)
moreover have *Ds* (*flow0 x tl*) (*f* (*flow0 x tl*)) $\neq$ *0*
by (*rule transversal*) (*use* ‹*flow0 x tl* $\in$ *?S*› **in** *auto*)
ultimately show *False* **by** *auto*
qed
qed (*use assms* **in** *auto*)

lemma *uniform-limit-flow0-state*:— TODO: is that something more general?
assumes *compact C*
assumes *C* $\subseteq$ *X*
shows *uniform-limit C* (λ*s x. flow0 x s*) (λ*x. flow0 x 0*) (*at 0*)
proof (*cases C* = {})
case *True* **then show** *?thesis* **by** *auto*
next
case *False* **show** *?thesis*
proof (*rule uniform-limitI*)
fix *e*::*real* **assume** *0* < *e*
{
fix *x* **assume** *x* $\in$ *C*
with *assms* **have** *x* $\in$ *X* **by** *auto*
from *existence-ivl-cballs*[*OF UNIV-I* ‹*x* $\in$ *X*›]
obtain *t L u* **where** $\bigwedge$*y. y* $\in$ *cball x u* $\Longrightarrow$ *cball 0 t* $\subseteq$ *existence-ivl0 y*
$\bigwedge$*s y. y* $\in$ *cball x u* $\Longrightarrow$ *s* $\in$ *cball 0 t* $\Longrightarrow$ *flow0 y s* $\in$ *cball y u*
L$-$*lipschitz-on* (*cball 0 t*$\times$*cball x u*) (λ(*t*, *x*). *flow0 x t*)
$\bigwedge$*y. y* $\in$ *cball x u* $\Longrightarrow$ *cball y u* $\subseteq$ *X*
0 < *t 0* < *u*
by *metis*
then have $\exists$*L*. $\exists$*u*>*0*. $\exists$*t*>*0*. *L*$-$*lipschitz-on* (*cball 0 t*$\times$*cball x u*) (λ(*t*, *x*).

flow0 x t) **by** *blast*
  } **then have** $\forall x{\in}C.\ \exists L.\ \exists u{>}0.\ \exists t{>}0.$ *L−lipschitz-on* (*cball 0 t*×*cball x u*) (λ(*t*, *x*). *flow0 x t*) **..**
 then obtain *L d′ u′* **where**
  L: $\bigwedge x.\ x \in C \Longrightarrow$ (*L x*)−*lipschitz-on* (*cball 0* (*d′ x*)×*cball x* (*u′ x*)) (λ(*t*, *x*). *flow0 x t*)
  and *d′*: $\bigwedge x.\ x \in C \Longrightarrow d'\ x > 0$
  and *u′*: $\bigwedge x.\ x \in C \Longrightarrow u'\ x > 0$
  by *metis*
 have $C \subseteq (\bigcup c{\in}C.$ *ball c* (*u′ c*)) **using** *u′* **by** *auto*
 from *compactE-image*[*OF* ‹*compact C*› - *this*]
 obtain *C′* **where** $C' \subseteq C$ **and** [*simp*]: *finite C′* **and** *C′-cover*: $C \subseteq (\bigcup c{\in}C'.$ *ball c* (*u′ c*))
  by *auto*
 from *C′-cover* **obtain** *c′* **where** *c′*: $x \in C \Longrightarrow x \in$ *ball* (*c′ x*) (*u′* (*c′ x*)) $x \in C \Longrightarrow c'\ x \in C'$ **for** *x*
  by (*auto simp*: *subset-iff*) *metis*
 have $\forall_F$ *s in at 0*. $\forall x{\in}$*ball c* (*u′ c*). *dist* (*flow0 x s*) (*flow0 x 0*) < *e* **if** $c \in C'$ **for** *c*
 proof −
  have *cC*: $c \in C$
   using *c′* ‹$c \in C'$› *d′* ‹$C' \subseteq C$›
   by *auto*
  have ∗: *dist* (*flow0 x s*) (*flow0 x 0*) ≤ *L c* ∗ |*s*|
   if $x{\in}$*ball c* (*u′ c*)
    $s \in$ *cball 0* (*d′ c*)
   for *x s*
  proof −
   from *L*[*OF cC*, *THEN lipschitz-onD*, *of* (*0*, *x*) (*s*, *x*)] *d′*[*OF cC*] *that*
   show *?thesis*
    by (*auto simp*: *dist-prod-def dist-commute*)
  qed
  have $\forall_F$ *s in at 0*. *abs s* < *d′ c*
   by (*rule order-tendstoD tendsto-intros*)+ (*use d′ cC* **in** *auto*)
  moreover have $\forall_F$ *s in at 0*. *L c* ∗ |*s*| < *e*
   by (*rule order-tendstoD tendsto-intros*)+ (*use* ‹*0* < *e*› **in** *auto*)
  ultimately show *?thesis*
   apply *eventually-elim*
   apply (*use* ∗ **in** *auto*)
   by *smt*
 qed
 then have $\forall_F$ *s in at 0*. $\forall c{\in}C'.\ \forall x{\in}$*ball c* (*u′ c*). *dist* (*flow0 x s*) (*flow0 x 0*) < *e*
  by (*simp add*: *eventually-ball-finite-distrib*)
 then show $\forall_F$ *s in at 0*. $\forall x{\in}C.$ *dist* (*flow0 x s*) (*flow0 x 0*) < *e*
  apply *eventually-elim*
  apply *auto*
  subgoal for *s x*
   apply (*drule bspec*[**where** *x*=*c′ x*])

```
        apply (simp add: c'(2))
       apply (drule bspec) prefer 2 apply assumption
       apply auto
       using c'(1) by auto
     done
  qed
qed

end
```

2.7 Fixpoints

```
context auto-ll-on-open begin

lemma fixpoint-sol:
  assumes x ∈ X f x = 0
  shows existence-ivl0 x = UNIV flow0 x t = x
proof −
  have sol: ((λt::real. x) solves-ode (λ-. f)) UNIV X
    apply (rule solves-odeI)
    by(auto simp add: assms intro!: derivative-intros)
  from maximal-existence-flow[OF sol] have
    UNIV ⊆ existence-ivl0 x flow0 x t = x by auto
  thus existence-ivl0 x = UNIV flow0 x t = x by auto
qed

end

end
```

3 Invariance

```
theory Invariance
  imports ODE-Misc
begin

context auto-ll-on-open begin

definition invariant M ⟷ (∀ x∈M. trapped x M)

definition positively-invariant M ⟷ (∀ x∈M. trapped-forward x M)

definition negatively-invariant M ⟷ (∀ x∈M. trapped-backward x M)

lemma positively-invariant-iff:
  positively-invariant M ⟷
  (⋃x∈M. flow0 x ` (existence-ivl0 x ∩ {0..})) ⊆ M
  unfolding positively-invariant-def trapped-forward-def
  by auto
```

lemma *negatively-invariant-iff*:
negatively-invariant M $\longleftrightarrow$
$(\bigcup x \in M.\ \mathit{flow0}\ x\ `\ (\mathit{existence\text{-}ivl0}\ x \cap \{..0\})) \subseteq M$
unfolding *negatively-invariant-def trapped-backward-def*
by *auto*

lemma *invariant-iff-pos-and-neg-invariant*:
invariant M $\longleftrightarrow$ *positively-invariant M* $\wedge$ *negatively-invariant M*
unfolding *invariant-def trapped-def positively-invariant-def negatively-invariant-def*
by *blast*

lemma *invariant-iff*:
invariant M $\longleftrightarrow$ $(\bigcup x \in M.\ \mathit{flow0}\ x\ `\ (\mathit{existence\text{-}ivl0}\ x)) \subseteq M$
unfolding *invariant-iff-pos-and-neg-invariant positively-invariant-iff negatively-invariant-iff*
by (*metis* (*mono-tags*) *SUP-le-iff invariant-def invariant-iff-pos-and-neg-invariant negatively-invariant-iff positively-invariant-iff trapped-iff-on-existence-ivl0*)

lemma *positively-invariant-restrict-dom*: *positively-invariant M* = *positively-invariant* ($M \cap X$)
unfolding *positively-invariant-def trapped-forward-def*
by (*auto intro*!: *flow-in-domain dest*: *mem-existence-ivl-iv-defined*)

lemma *negatively-invariant-restrict-dom*: *negatively-invariant M* = *negatively-invariant* ($M \cap X$)
unfolding *negatively-invariant-def trapped-backward-def*
by (*auto intro*!: *flow-in-domain dest*: *mem-existence-ivl-iv-defined*)

lemma *invariant-restrict-dom*: *invariant M* = *invariant* ($M \cap X$)
using *invariant-iff-pos-and-neg-invariant*
negatively-invariant-restrict-dom
positively-invariant-restrict-dom **by** *auto*

end context *auto-ll-on-open* **begin**

lemma *positively-invariant-eq-rev*: *positively-invariant M* = *rev.negatively-invariant M*
unfolding *positively-invariant-def rev.negatively-invariant-def*
by (*simp add*: *rev.trapped-backward-iff-rev-trapped-forward*)

lemma *negatively-invariant-eq-rev*: *negatively-invariant M* = *rev.positively-invariant M*
unfolding *negatively-invariant-def rev.positively-invariant-def*
by (*simp add*: *trapped-backward-iff-rev-trapped-forward*)

lemma *invariant-eq-rev*: *invariant M* = *rev.invariant M*
unfolding *invariant-iff-pos-and-neg-invariant rev.invariant-iff-pos-and-neg-invariant*
positively-invariant-eq-rev negatively-invariant-eq-rev **by** *auto*

lemma *negatively-invariant-complI*: *negatively-invariant* $(X-M)$ **if** *positively-invariant* M
  unfolding *negatively-invariant-def trapped-backward-def*
proof *clarsimp*
  fix $x\ t$
  assume x: $x \in X\ x \notin M\ t \in$ *existence-ivl0* $x\ t \leq 0$
  have *a1*:*flow0* $x\ t \in X$ **using** x
    using *flow-in-domain* **by** *blast*
  have *a2*:*flow0* $x\ t \notin M$
  proof (*rule ccontr*)
    assume $\neg$ *flow0* $x\ t \notin M$
    then have *trapped-forward* (*flow0* $x\ t$) M
      using *positively-invariant-def that* **by** *auto*
    moreover have *flow0* (*flow0* $x\ t$) $(-t) = x$
      using ‹$t \in$ *existence-ivl0* x› *flows-reverse* **by** *auto*
    moreover have $-t \in$ *existence-ivl0* (*flow0* $x\ t$) $\cap \{0..\}$
      using *existence-ivl-reverse* $x(3)\ x(4)$ **by** *auto*
    ultimately have $x \in M$ **unfolding** *trapped-forward-def*
      by (*metis image-subset-iff*)
    thus *False* **using** $x(2)$ **by** *auto*
  qed
  show *flow0* $x\ t \in X \wedge$ *flow0* $x\ t \notin M$ **using** *a1 a2* **by** *auto*
qed

end context *auto-ll-on-open* **begin**

lemma *negatively-invariant-complD*: *positively-invariant* M **if** *negatively-invariant* $(X-M)$
proof $-$
  have *rev.positively-invariant* $(X-M)$ **using** *that*
    by (*simp add*: *negatively-invariant-eq-rev*)
  then have *rev.negatively-invariant* $(X-(X-M))$
    by (*simp add*: *rev.negatively-invariant-complI*)
  then have *positively-invariant* $(X-(X-M))$
    using *rev.negatively-invariant-eq-rev* **by** *auto*
  thus *?thesis* **using** *Diff-Diff-Int*
    by (*metis inf-commute positively-invariant-restrict-dom*)
qed

lemma *pos-invariant-iff-compl-neg-invariant*: *positively-invariant* $M \longleftrightarrow$ *negatively-invariant* $(X - M)$
  by (*safe intro*!: *negatively-invariant-complI dest*!: *negatively-invariant-complD*)

lemma *neg-invariant-iff-compl-pos-invariant*:
  shows *negatively-invariant* $M \longleftrightarrow$ *positively-invariant* $(X - M)$
  by (*simp add*: *auto-ll-on-open.pos-invariant-iff-compl-neg-invariant negatively-invariant-eq-rev positively-invariant-eq-rev rev.auto-ll-on-open-axioms*)

lemma *invariant-iff-compl-invariant*:
shows *invariant M* $\longleftrightarrow$ *invariant* $(X - M)$
using *invariant-iff-pos-and-neg-invariant neg-invariant-iff-compl-pos-invariant pos-invariant-iff-compl-neg-in*
by *blast*

lemma *invariant-iff-pos-invariant-and-compl-pos-invariant*:
shows *invariant M* $\longleftrightarrow$ *positively-invariant M* $\wedge$ *positively-invariant* $(X{-}M)$
by (*simp add*: *invariant-iff-pos-and-neg-invariant neg-invariant-iff-compl-pos-invariant*)

end

3.1 Tools for proving invariance

context *auto-ll-on-open* **begin**

lemma *positively-invariant-left-inter*:
assumes *positively-invariant C*
assumes $\forall x \in C \cap D$. *trapped-forward x D*
shows *positively-invariant* $(C \cap D)$
using *assms positively-invariant-def trapped-forward-def* **by** *auto*

lemma *trapped-forward-le*:
fixes $V :: 'a \Rightarrow real$
assumes $V\ x \leq 0$
assumes *contg*: *continuous-on* (*flow0 x* ‘ (*existence-ivl0 x* $\cap$ $\{0..\}$)) *g*
assumes $\bigwedge x$. (V *has-derivative* V' x) (*at x*)
assumes $\bigwedge s$. $s \in$ *existence-ivl0 x* $\cap$ $\{0..\}$ $\Longrightarrow$ V' (*flow0 x s*) (f (*flow0 x s*)) $\leq$ *g*
(*flow0 x s*) $*$ V (*flow0 x s*)
shows *trapped-forward x* $\{x.\ V\ x \leq 0\}$
unfolding *trapped-forward-def*
proof *clarsimp*
fix *t*
assume *t*: $t \in$ *existence-ivl0 x* $0 \leq t$
then have *ex*:$\{0..t\} \subseteq$ *existence-ivl0 x*
by (*simp add*: *local.ivl-subset-existence-ivl*)
have *contV*: *continuous-on UNIV V*
using *assms*(*3*) *has-derivative-continuous-on* **by** *blast*
have *1*: *continuous-on* $\{0..t\}$ ($g \circ$ *flow0 x*)
apply (*rule continuous-on-compose*)
using *continuous-on-subset ex local.flow-continuous-on* **apply** *blast*
by (*meson Int-subset-iff atLeastAtMost-iff atLeast-iff contg continuous-on-subset*
ex image-mono subsetI)
have *2*: ($\bigwedge s$. $s \in \{0..t\}$ $\Longrightarrow$
($V \circ$ *flow0 x has-real-derivative* (V' (*flow0 x s*) $\circ f \circ$ *flow0 x*) *s*) (*at s*))
apply (*auto simp add*:*o-def has-field-derivative-def*)
proof $-$
fix *s*
assume $0 \leq s$ $s \leq t$
then have $s \in$ *existence-ivl0 x* **using** *ex* **by** *auto*

from *flow-has-derivative*[*OF this*] **have**
$(flow0\ x\ has\text{-}derivative\ (\lambda i.\ i *_R f\ (flow0\ x\ s)))\ (at\ s)$ **.**
from *has-derivative-compose*[*OF this assms*(*3*)]
have $((\lambda t.\ V\ (flow0\ x\ t))\ has\text{-}derivative\ (\lambda t.\ V'\ (flow0\ x\ s)\ \ (t *_R f\ (flow0\ x\ s))))\ (at\ s)$ **.**
moreover have $linear\ (V'\ (flow0\ x\ s))$ **using** *assms*(*3*) *has-derivative-linear* **by** *blast*
ultimately
have $((\lambda t.\ V\ (flow0\ x\ t))\ has\text{-}derivative\ (\lambda t.\ t *_R\ V'\ (flow0\ x\ s)\ (f\ (flow0\ x\ s))))\ (at\ s)$
unfolding *linear-cmul*[*OF* ‹*linear* (*V′* (*flow0 x s*))›] **by** *blast*
thus $((\lambda t.\ V\ (flow0\ x\ t))\ has\text{-}derivative\ (*)\ (V'\ (flow0\ x\ s)\ (f\ (flow0\ x\ s))))\ (at\ s)$
by (*auto intro*!: *derivative-eq-intros simp add*: *mult-commute-abs*)
qed
have *3*: $(\bigwedge s.\ s \in \{0..t\} \Longrightarrow$
$(V'\ (flow0\ x\ s)\ \circ f \circ flow0\ x)\ s \le (g \circ flow0\ x)\ s *_R (V \circ flow0\ x)\ s)$
using *ex* **by** (*auto intro*!:*assms*(*4*))
from *comparison-principle-le-linear*[*OF 1 2 - 3*] *assms*(*1*)
have $\forall s \in \{0..t\}.\ (V \circ flow0\ x)\ s \le 0$
using *local.mem-existence-ivl-iv-defined*(*2*) *t*(*1*) **by** *auto*
thus $V\ (flow0\ x\ t) \le 0$
by (*simp add*: *t*(*2*))
qed

lemma *positively-invariant-le-domain*:
fixes $V :: 'a \Rightarrow real$
assumes *positively-invariant D*
assumes *contg*: *continuous-on D g*
assumes $\bigwedge x.\ (V\ has\text{-}derivative\ V'\ x)\ (at\ x)$
assumes $\bigwedge s.\ s \in D \Longrightarrow V'\ s\ (f\ s) \le g\ s * V\ s$
shows *positively-invariant* $(D \cap \{x.\ V\ x \le 0\})$
apply (*auto intro*!:*positively-invariant-left-inter*[*OF assms*(*1*)])
proof −
fix *x*
assume $x \in D\ V\ x \le 0$
have *continuous-on* (*flow0 x* ‘ (*existence-ivl0 x* ∩ {*0*..})) *g*
by (*meson* ‹$x \in D$› *assms*(*1*) *contg continuous-on-subset positively-invariant-def trapped-forward-def*)
from *trapped-forward-le*[*OF* ‹$V\ x \le 0$› *this assms*(*3*)]
show *trapped-forward x* $\{x.\ V\ x \le 0\}$ **using** *assms*(*4*)
using ‹$x \in D$› *assms*(*1*) *positively-invariant-def trapped-forward-def* **by** *auto*
qed

lemma *positively-invariant-barrier-domain*:
fixes $V :: 'a \Rightarrow real$
assumes *positively-invariant D*
assumes $\bigwedge x.\ (V\ has\text{-}derivative\ V'\ x)\ (at\ x)$
assumes *continuous-on* $D\ (\lambda x.\ V'\ x\ (f\ x))$

```
  assumes ⋀s. s ∈ D ⟹ V s = 0 ⟹ V' s (f s) < 0
  shows positively-invariant (D ∩ {x. V x ≤ 0})
  apply (auto intro!:positively-invariant-left-inter[OF assms(1)])
proof -
  fix x
  assume x ∈ D V x ≤ 0
  have contV: continuous-on UNIV V using assms(2) has-derivative-continuous-on
by blast
  then have *: continuous-on (flow0 x ‘ (existence-ivl0 x ∩ {0..})) V
    using continuous-on-subset by blast
  have sub: flow0 x ‘ (existence-ivl0 x ∩ {0..}) ⊆ D
    using ‹x ∈ D› assms(1) positively-invariant-def trapped-forward-def by auto
  then have contV': continuous-on (flow0 x ‘ (existence-ivl0 x ∩ {0..})) (λx. V'
x (f x))
    by (metis assms(3) continuous-on-subset)
  have nz: ⋀i t. t ∈ existence-ivl0 x ⟹
      0 ≤ t ⟹  max (−V' (flow0 x t) (f (flow0 x t))) ((V (flow0 x t))²) > 0
  proof -
    fix i t
    assume t ∈ existence-ivl0 x 0 ≤ t
    then have flow0 x t ∈ D
      using ‹x ∈ D› assms(1) positively-invariant-def trapped-forward-def by auto
    then have V (flow0 x t) = 0 ⟹ − V' (flow0 x t) (f (flow0 x t)) > 0 using
assms(4) by simp
    then have (V (flow0 x t))^2 > 0 ∨ − V' (flow0 x t) (f (flow0 x t)) > 0 by
simp
    thus max (−V' (flow0 x t) (f (flow0 x t))) ((V (flow0 x t))²) > 0 unfolding
less-max-iff-disj
      by auto
  qed
  have *: continuous-on (flow0 x ‘ (existence-ivl0 x ∩ {0..})) (λx. V' x (f x) * V
x / max (− V' x (f x)) ((V x)^2))
    apply (auto intro!:continuous-intros continuous-on-max simp add: * contV')
    using nz by fastforce
  have (⋀t. t ∈ existence-ivl0 x ∩ {0..} ⟹
       V' (flow0 x t) (f (flow0 x t)) ≤
       (V' (flow0 x t) (f (flow0 x t)) * V (flow0 x t)
       / max (− V' (flow0 x t) (f (flow0 x t))) ((V (flow0 x t))²)) * V (flow0 x t))
  proof clarsimp
    fix t
    assume t ∈ existence-ivl0 x 0 ≤ t
    then have p: max (−V' (flow0 x t) (f (flow0 x t))) ((V (flow0 x t))²) > 0
using nz by auto
    have  V' (flow0 x t) (f (flow0 x t)) * max (− V' (flow0 x t) (f (flow0 x t)))
((V (flow0 x t))²)
      ≤  V' (flow0 x t) (f (flow0 x t)) * (V (flow0 x t))²
    by (smt mult-minus-left mult-minus-right power2-eq-square mult-le-cancel-left-pos)
    then have V' (flow0 x t) (f (flow0 x t))
      ≤  V' (flow0 x t) (f (flow0 x t)) * (V (flow0 x t))²
```

/ *max* ($-$ V' (*flow0 x t*) (*f* (*flow0 x t*))) ((*V* (*flow0 x t*))^2)
using *p pos-le-divide-eq* **by** *blast*
thus V' (*flow0 x t*) (*f* (*flow0 x t*))
$\leq$ V' (*flow0 x t*) (*f* (*flow0 x t*)) $*$ (*V* (*flow0 x t*)) $*$ *V* (*flow0 x t*) /
max ($-$ V' (*flow0 x t*) (*f* (*flow0 x t*))) ((*V* (*flow0 x t*))^2)
by (*simp add*: *power2-eq-square*)
qed
from *trapped-forward-le*[*OF* ‹*V x* $\leq$ *0*› $*$ *assms*(*2*) *this*]
show *trapped-forward x* {*x*. *V x* $\leq$ *0*} **by** *auto*
qed

lemma *positively-invariant-UNIV*:
shows *positively-invariant UNIV*
using *positively-invariant-iff* **by** *blast*

lemma *positively-invariant-conj*:
assumes *positively-invariant C*
assumes *positively-invariant D*
shows *positively-invariant* (*C* $\cap$ *D*)
using *assms positively-invariant-def*
using *positively-invariant-left-inter* **by** *auto*

lemma *positively-invariant-le*:
fixes *V* :: *'a* $\Rightarrow$ *real*
assumes *contg*: *continuous-on UNIV g*
assumes $\bigwedge$*x*. (*V has-derivative* V' *x*) (*at x*)
assumes $\bigwedge$*s*. V' *s* (*f s*) $\leq$ *g s* $*$ *V s*
shows *positively-invariant* {*x*. *V x* $\leq$ *0*}
proof $-$
from *positively-invariant-le-domain*[*OF positively-invariant-UNIV assms*]
have *positively-invariant* (*UNIV* $\cap$ {*x*. *V x* $\leq$ *0*}) **.**
thus *?thesis* **by** *auto*
qed

lemma *positively-invariant-barrier*:
fixes *V* :: *'a* $\Rightarrow$ *real*
assumes $\bigwedge$*x*. (*V has-derivative* V' *x*) (*at x*)
assumes *continuous-on UNIV* (λ*x*. V' *x* (*f x*))
assumes $\bigwedge$*s*. *V s* = *0* $\Longrightarrow$ V' *s* (*f s*) < *0*
shows *positively-invariant* {*x*. *V x* $\leq$ *0*}
proof $-$
from *positively-invariant-barrier-domain*[*OF positively-invariant-UNIV assms*]
have *positively-invariant* (*UNIV* $\cap$ {*x*. *V x* $\leq$ *0*}) **.**
thus *?thesis* **by** *auto*
qed

end

end

4 Limit Sets

theory *Limit-Set*
imports *Invariance*
begin

context *auto-ll-on-open* **begin**

Positive limit point, assuming $\{0..\} \subseteq$ *existence-ivl0 x*

definition *ω-limit-point x p* $\longleftrightarrow$
$\{0..\} \subseteq$ *existence-ivl0 x* $\wedge$
$(\exists s.\ s \longrightarrow \infty \wedge (flow0\ x \circ s) \longrightarrow p)$

Also called the ω-limit set of x

definition *ω-limit-set x* $= \{p.\ \omega\text{-}limit\text{-}point\ x\ p\}$

definition *α-limit-point x p* $\longleftrightarrow$
$\{..0\} \subseteq$ *existence-ivl0 x* $\wedge$
$(\exists s.\ s \longrightarrow -\infty \wedge (flow0\ x \circ s) \longrightarrow p)$

Also called the α-limit set of x

definition *α-limit-set x* $=$
$\{p.\ \alpha\text{-}limit\text{-}point\ x\ p\}$

end context *auto-ll-on-open* **begin**

lemma *α-limit-point-eq-rev*: *α-limit-point x p* = *rev.ω-limit-point x p*
unfolding *α-limit-point-def rev.ω-limit-point-def*
apply (*auto simp*: *rev-eq-flow*[*abs-def*] *o-def filterlim-uminus-at-bot rev-existence-ivl-eq0 subset-iff*
intro: *exI*[**where** *x=uminus o s* **for** *s*])
using *neg-0-le-iff-le* **by** *fastforce*

lemma *α-limit-set-eq-rev*: *α-limit-set x* = *rev.ω-limit-set x*
unfolding *α-limit-set-def rev.ω-limit-set-def α-limit-point-eq-rev* **..**

lemma *ω-limit-pointE*:
assumes *ω-limit-point x p*
obtains *s* **where**
filterlim s at-top sequentially
$(flow0\ x \circ s) \longrightarrow p$
$\forall n.\ b \leq s\ n$
using *assms filterlim-at-top-choose-lower ω-limit-point-def* **by** *blast*

lemma *ω-limit-set-eq*:
assumes $\{0..\} \subseteq$ *existence-ivl0 x*
shows *ω-limit-set x* $= (INF\ \tau \in \{0..\}.\ closure\ (flow0\ x\ `\ \{\tau..\}))$
unfolding *ω-limit-set-def*
proof *safe*

fix *p t*
assume *pt*: $0 \leq (t::real)$ *ω-limit-point x p*
from *ω-limit-pointE*[*OF pt*(*2*)]
obtain *s* **where**
filterlim s at-top sequentially
$(flow0\ x \circ s) \longrightarrow p$
$\forall n.\ t \leq s\ n$ **by** *blast*
thus $p \in closure\ (flow0\ x\ `\ \{t..\})$ **unfolding** *closure-sequential*
by (*metis atLeast-iff comp-apply imageI*)
next
fix *p*
assume $p \in (\bigcap \tau \in \{0..\}.\ closure\ (flow0\ x\ `\ \{\tau..\}))$
then have $\bigwedge t.\ t \geq 0 \Longrightarrow p \in closure\ (flow0\ x\ `\ \{t..\})$ **by** *blast*
then have $\bigwedge t\ e.\ t \geq 0 \Longrightarrow e > 0 \Longrightarrow (\exists\, tt \geq t.\ dist\ (flow0\ x\ tt)\ p < e)$
unfolding *closure-approachable*
by *fastforce*
from *approachable-sequenceE*[*OF this*]
obtain *s* **where** *filterlim s at-top sequentially* $(flow0\ x \circ s) \longrightarrow p$ **by** *auto*
thus *ω-limit-point x p* **unfolding** *ω-limit-point-def* **using** *assms* **by** *auto*
qed

lemma *ω-limit-set-empty*:
assumes $\neg (\{0..\} \subseteq existence\text{-}ivl0\ x)$
shows *ω-limit-set x* = {}
unfolding *ω-limit-set-def ω-limit-point-def*
by (*simp add*: *assms*)

lemma *ω-limit-set-closed*: *closed* (*ω-limit-set x*)
using *ω-limit-set-eq*
by (*metis ω-limit-set-empty closed-INT closed-closure closed-empty*)

lemma *ω-limit-set-positively-invariant*:
shows *positively-invariant* (*ω-limit-set x*)
unfolding *positively-invariant-def trapped-forward-def*
proof *safe*
fix *dummy p t*
assume *xa*: $p \in$ *ω-limit-set x*
$t \in existence\text{-}ivl0\ p$
$0 \leq t$
have $p \in X$ **using** *mem-existence-ivl-iv-defined*(*2*) *xa*(*2*) **by** *blast*
have *exist*: $\{0..\} \subseteq existence\text{-}ivl0\ x$ **using** *xa*(*1*)
unfolding *ω-limit-set-def ω-limit-point-def* **by** *auto*
from *xa*(*1*)
obtain *s* **where** *s*:
filterlim s at-top sequentially
$(flow0\ x \circ s) \longrightarrow p$
$\forall n.\ 0 \leq s\ n$
unfolding *ω-limit-set-def* **by** (*auto elim*!:*ω-limit-pointE*)

define *r* **where** *r* = (λ*n*. *t* + *s n*)
have *rlim*: *filterlim r at-top sequentially* **unfolding** *r-def*
by (*auto intro*: *filterlim-tendsto-add-at-top*[*OF - s*(*1*)])
define *dom* **where** *dom* = *image* (*flow0 x*) {*0*..} ∪ {*p*}
have *domin*: ∀ *n*. (*flow0 x* ∘ *s*) *n* ∈ *dom p* ∈ *dom* **unfolding** *dom-def o-def*
using *exist* **by**(*auto simp add*: *s*(*3*))
have *xt*: ⋀*x*. *x* ∈ *dom* ⟹ *t* ∈ *existence-ivl0 x* **unfolding** *dom-def* **using** *xa*(*2*)
apply *auto*
apply (*rule existence-ivl-trans′*)
using *exist xa*(*3*) **apply** *auto*[*1*]
using *exist* **by** *auto*
have *cont*: *continuous-on dom* (λ*x*. *flow0 x t*)
apply (*rule flow-continuous-on-compose*)
apply *auto*
using ‹*p* ∈ *X*› *exist local.dom-def local.flow-in-domain* **apply** *auto*[*1*]
using *xt* **.**
then have *f1*: ((λ*x*. *flow0 x t*) ∘ (*flow0 x* ∘ *s*)) ⟶ *flow0 p t* **using** *domin* *s*(*2*)
unfolding *continuous-on-sequentially*
by *blast*
have *ff*:⋀*n*. (*flow0 x* ∘ *r*) *n* = ((λ*x*. *flow0 x t*) ∘ (*flow0 x* ∘ *s*)) *n*
unfolding *o-def r-def*
proof −
fix *n*
have *s*:*s n* ∈ *existence-ivl0 x*
using *s*(*3*) *exist* **by** *auto*
then have *t*:*t* ∈ *existence-ivl0* (*flow0 x* (*s n*))
using *domin*(*1*) *xt* **by** *auto*
from *flow-trans*[*OF s t*]
show *flow0 x* (*t* + *s n*) = *flow0* (*flow0 x* (*s n*)) *t*
by (*simp add*: *add.commute*)
qed
have *f2*: (*flow0 x* ∘ *r*) ⟶ *flow0 p t* **using** *f1* **unfolding** *ff* **.**
show *flow0 p t* ∈ *ω-limit-set x* **using** *exist f2 rlim*
unfolding *ω-limit-set-def ω-limit-point-def*
using *flow-in-domain r-def s*(*3*) *xa*(*2*) *xa*(*3*) **by** *auto*
qed

lemma *ω-limit-set-invariant*:
shows *invariant* (*ω-limit-set x*)
unfolding *invariant-iff-pos-invariant-and-compl-pos-invariant*
proof *safe*
show *positively-invariant* (*ω-limit-set x*)
using *ω-limit-set-positively-invariant* **.**
next
show *positively-invariant* (*X* − *ω-limit-set x*)
unfolding *positively-invariant-def trapped-forward-def*
apply *safe*
using *local.flow-in-domain* **apply** *blast*

proof −
  fix *dummy p t*
  assume *xa*: *p* ∈ *X p* ∉ *ω-limit-set x*
    t ∈ *existence-ivl0 p 0* ≤ *t*
    and *f*: *flow0 p t* ∈ *ω-limit-set x*
  have *exist*: {*0*..} ⊆ *existence-ivl0 x* **using** *f*
    unfolding *ω-limit-set-def ω-limit-point-def* **by** *auto*
  from *f*
  obtain *s* **where** *s*:
    filterlim s at-top sequentially
    (*flow0 x* ∘ *s*) ⟶ *flow0 p t*
    ∀ *n*. *t* ≤ *s n*
    unfolding *ω-limit-set-def* **by** (*auto elim*!:*ω-limit-pointE*)
  define *r* **where** *r* = (*λn*. (−*t*) + *s n*)
  have (*λx*. −*t*) ⟶ −*t* **by** *simp*
  from *filterlim-tendsto-add-at-top*[*OF this s*(*1*)]
  have *rlim*: *filterlim r at-top sequentially* **unfolding** *r-def* **by** *simp*
  define *dom* **where** *dom* = *image* (*flow0 x*) {*t*..} ∪ {*flow0 p t*}
  have *domin*: ∀ *n*. (*flow0 x* ∘ *s*) *n* ∈ *dom flow0 p t* ∈ *dom* **unfolding** *dom-def*
o-def
    using *exist* **by**(*auto simp add*: *s*(*3*))
  have *xt*: ⋀*x*. *x* ∈ *dom* ⟹ −*t* ∈ *existence-ivl0 x* **unfolding** *dom-def* **using**
xa(*2*)
    apply *auto*
    using *local.existence-ivl-reverse xa*(*3*) **apply** *auto*[*1*]
  by (*metis exist atLeast-iff diff-conv-add-uminus diff-ge-0-iff-ge linordered-ab-group-add-class.zero-le-double-
local.existence-ivl-trans′ order-trans subset-iff xa(*4*))
  have *cont*: *continuous-on dom* (*λx*. *flow0 x* (−*t*))
    apply (*rule flow-continuous-on-compose*)
      apply *auto*
    using *local.mem-existence-ivl-iv-defined*(*2*) *xt* **apply** *blast*
    by (*simp add*: *xt*)
  then have *f1*: ((*λx*. *flow0 x* (−*t*)) ∘ (*flow0 x* ∘ *s*)) ⟶ *flow0* (*flow0 p t*)
(−*t*) **using** *domin s*(*2*)
    unfolding *continuous-on-sequentially*
    by *blast*
  have *ff*:⋀*n*. (*flow0 x* ∘ *r*) *n* = ((*λx*. *flow0 x* (−*t*)) ∘ (*flow0 x* ∘ *s*)) *n*
    unfolding *o-def r-def*
  proof −
    fix *n*
    have *s*:*s n* ∈ *existence-ivl0 x*
      using *s*(*3*) *exist* ‹*0*≤ *t*› **by** (*meson atLeast-iff order-trans subset-eq*)
    then have *t*:−*t* ∈ *existence-ivl0* (*flow0 x* (*s n*))
      using *domin*(*1*) *xt* **by** *auto*
    from *flow-trans*[*OF s t*]
    show *flow0 x* (−*t* + *s n*) = *flow0* (*flow0 x* (*s n*)) (−*t*)
      by (*simp add*: *add.commute*)
  qed
  have (*flow0 x* ∘ *r*) ⟶ *flow0* (*flow0 p t*) (−*t*) **using** *f1* **unfolding** *ff* **.**

then have *f2*: (*flow0 x* ∘ *r*) ⟶ *p* **using** *flows-reverse xa*(*3*) **by** *auto*
then have *p* ∈ *ω-limit-set x* **unfolding** *ω-limit-set-def ω-limit-point-def*
using *rlim exist* **by** *auto*
thus *False* **using** *xa*(*2*) **by** *auto*
qed
qed

end context *auto-ll-on-open* **begin**

lemma *α-limit-set-eq*:
assumes {*..0*} ⊆ *existence-ivl0 x*
shows *α-limit-set x* = (*INF τ* ∈ {*..0*}. *closure* (*flow0 x* ‘ {*..τ*}))
using *rev.ω-limit-set-eq*[*of x*, *OF assms*[*folded infinite-rev-existence-ivl0-rewrites*]]
unfolding *α-limit-set-eq-rev rev-flow-image-eq image-uminus-atLeast*
by (*smt INT-extend-simps*(*10*) *Sup.SUP-cong image-uminus-atMost*)

lemma *α-limit-set-closed*:
shows *closed* (*α-limit-set x*)
unfolding *α-limit-set-eq-rev* **by** (*rule rev.ω-limit-set-closed*)

lemma *α-limit-set-positively-invariant*:
shows *negatively-invariant* (*α-limit-set x*)
unfolding *negatively-invariant-eq-rev α-limit-set-eq-rev*
by (*simp add*: *rev.ω-limit-set-positively-invariant*)

lemma *α-limit-set-invariant*:
shows *invariant* (*α-limit-set x*)
unfolding *invariant-eq-rev α-limit-set-eq-rev*
by (*simp add*: *rev.ω-limit-set-invariant*)

Fundamental properties of the positive limit set

context
fixes *x K*
assumes *K*: *compact K K* ⊆ *X*
assumes *x*: *x* ∈ *X trapped-forward x K*
begin

Bunch of facts for what's to come

private lemma *props*:
shows {*0..*} ⊆ *existence-ivl0 x seq-compact K*
apply (*rule trapped-sol-right*)
using *x K* **by** (*auto simp add*: *compact-imp-seq-compact*)

private lemma *flowimg*:
shows *flow0 x* ‘ (*existence-ivl0 x* ∩ {*0..*}) = *flow0 x* ‘ {*0..*}
using *props*(*1*) **by** *auto*

lemma *ω-limit-set-in-compact-subset*:
shows *ω-limit-set x* ⊆ *K*

unfolding *ω-limit-set-def*
proof *safe*
fix *p s*
assume *ω-limit-point x p*
from *ω-limit-pointE*[*OF this*]
obtain *s* **where** *s*:
filterlim s at-top sequentially
(*flow0 x* ∘ *s*) ⟶ *p*
∀ *n*. *0* ≤ *s n* **by** *blast*
then have *fin*: ∀ *n*. (*flow0 x* ∘ *s*) *n* ∈ *K* **using** *s*(*3*) *x K props*(*1*)
unfolding *trapped-forward-def*
by (*simp add*: *subset-eq*)
from *seq-compactE*[*OF props*(*2*) *fin*]
show *p* ∈ *K* **using** *s*(*2*)
by (*metis LIMSEQ-subseq-LIMSEQ LIMSEQ-unique*)
qed

lemma *ω-limit-set-in-compact-compact*:
shows *compact* (*ω-limit-set x*)
proof −
from *ω-limit-set-in-compact-subset*
have *bounded* (*ω-limit-set x*)
using *bounded-subset compact-imp-bounded*
using *K*(*1*) **by** *auto*
thus *?thesis* **using** *ω-limit-set-closed*
by (*simp add*: *compact-eq-bounded-closed*)
qed

lemma *ω-limit-set-in-compact-nonempty*:
shows *ω-limit-set x* ≠ {}
proof −
have *fin*: ∀ *n*. (*flow0 x* ∘ *real*) *n* ∈ *K* **using** *x K props*(*1*)
by (*simp add*: *flowimg image-subset-iff trapped-forward-def*)
from *seq-compactE*[*OF props*(*2*) *this*]
obtain *r l* **where** *l* ∈ *K strict-mono r* (*flow0 x* ∘ *real* ∘ *r*) ⟶ *l* **by** *blast*
then have *ω-limit-point x l* **unfolding** *ω-limit-point-def* **using** *props*(*1*)
by (*smt comp-def filterlim-sequentially-iff-filterlim-real filterlim-subseq tendsto-at-top-eq-left*)
thus *?thesis* **unfolding** *ω-limit-set-def* **by** *auto*
qed

lemma *ω-limit-set-in-compact-existence*:
shows ⋀*y*. *y* ∈ *ω-limit-set x* ⟹ *existence-ivl0 y* = *UNIV*
proof −
fix *y*
assume *y*: *y* ∈ *ω-limit-set x*
then have *y* ∈ *X* **using** *ω-limit-set-in-compact-subset K* **by** *blast*
from *ω-limit-set-invariant*
have ⋀*t*. *t* ∈ *existence-ivl0 y* ⟹ *flow0 y t* ∈ *ω-limit-set x*

unfolding *invariant-def trapped-iff-on-existence-ivl0* **using** *y* **by** *blast*
then have *t*: $\bigwedge t.\ t \in$ *existence-ivl0 y* $\Longrightarrow$ *flow0 y t* $\in K$
using *ω-limit-set-in-compact-subset* **by** *blast*
thus *existence-ivl0 y = UNIV*
by (*meson* ‹$y \in X$› *existence-ivl-zero existence-ivl-initial-time-iff existence-ivl-subset mem-compact-implies-subset-existence-interval subset-antisym K*)
qed

lemma *ω-limit-set-in-compact-tendsto*:
shows (($\lambda t.$ *infdist* (*flow0 x t*) (*ω-limit-set x*)) $\longrightarrow$ *0*) *at-top*
proof (*rule ccontr*)
assume ¬ (($\lambda t.$ *infdist* (*flow0 x t*) (*ω-limit-set x*)) $\longrightarrow$ *0*) *at-top*
from *not-tendsto-frequentlyE*[*OF this*]
obtain *S* **where** *S*: *open S 0* $\in$ *S*
$\exists_F$ *t in at-top. infdist* (*flow0 x t*) (*ω-limit-set x*) $\notin$ *S* **.**
then obtain *e* **where** *e* > *0 ball 0 e* $\subseteq$ *S* **using** *openE* **by** *blast*
then have $\bigwedge x.\ x \geq 0 \Longrightarrow x \notin S \Longrightarrow x \geq e$ **by** *force*
then have $\forall$ *xa. infdist* (*flow0 x xa*) (*ω-limit-set x*) $\notin S \longrightarrow$
infdist (*flow0 x xa*) (*ω-limit-set x*) $\geq$ *e* **using** *infdist-nonneg* **by** *blast*
from *frequently-mono*[*OF this S*(*3*)]
have $\exists_F$ *t in at-top. infdist* (*flow0 x t*) (*ω-limit-set x*) $\geq$ *e* **by** *blast*
then have $\forall$ *n.* $\exists_F$ *t in at-top. infdist* (*flow0 x t*) (*ω-limit-set x*) $\geq$ *e* $\wedge$ *real n* $\leq$ *t*
by (*auto intro*!: *eventually-frequently-conj*)
from *frequently-at-topE*[*OF this*]
obtain *s* **where** *s*: $\bigwedge i.$ *e* $\leq$ *infdist* (*flow0 x* (*s i*)) (*ω-limit-set x*)
$\bigwedge i.$ *real i* $\leq$ *s i strict-mono s* **by** *force*
then have *sf*: *filterlim s at-top sequentially*
using *filterlim-at-top-mono filterlim-real-sequentially not-eventuallyD* **by** *blast*
have *fin*: $\forall$ *n.* (*flow0 x* $\circ$ *s*) *n* $\in K$ **using** *x K props*(*1*) *s* **unfolding** *flowimg trapped-forward-def*
by (*metis atLeast-iff comp-apply image-subset-iff of-nat-0-le-iff order-trans*)
from *seq-compactE*[*OF props*(*2*) *this*]
obtain *r l* **where** *r*:*strict-mono r* **and** *l*: *l* $\in K$ (*flow0 x* $\circ$ *s* $\circ$ *r*) $\longrightarrow$ *l* **by** *blast*
moreover from *filterlim-at-top-strict-mono*[*OF s*(*3*) *r*(*1*) *sf*]
have *filterlim* (*s* $\circ$ *r*) *at-top sequentially* **.**
moreover have *ω-limit-point x l* **unfolding** *ω-limit-point-def* **using** *props*(*1*) *calculation*
by (*metis comp-assoc*)
ultimately have *infdist l* (*ω-limit-set x*) = *0* **by** (*simp add*: *ω-limit-set-def*)
then have *c1*:(($\lambda y.$ *infdist y* (*ω-limit-set x*)) $\circ$ (*flow0 x* $\circ$ *s* $\circ$ *r*)) $\longrightarrow$ *0*
by (*auto intro*!: *tendsto-compose-at*[*OF l*(*2*)] *tendsto-eq-intros*)
have *c2*: $\bigwedge i.$ *e* $\leq$ *infdist* (*flow0 x* ((*s* $\circ$ *r*) *i*)) (*ω-limit-set x*) **using** *s*(*1*) **by** *simp*
show *False* **using** *c1 c2* ‹*e* > *0*› **unfolding** *o-def*
using *Lim-bounded2*
by (*metis* (*no-types, lifting*) *ball-eq-empty centre-in-ball empty-iff*)
qed

lemma *ω-limit-set-in-compact-connected*:
shows *connected* (*ω-limit-set x*)
unfolding *connected-closed-set*[*OF ω-limit-set-closed*]
proof *clarsimp*
fix *Apre Bpre*
assume *pre*: *closed Apre Apre* ∪ *Bpre* = *ω-limit-set x closed Bpre*
Apre ≠ {} *Bpre* ≠ {} *Apre* ∩ *Bpre* = {}

then obtain *A B* **where** *Apre* ⊆ *A Bpre* ⊆ *B open A open B* **and** *disj*:*A* ∩ *B* = {}
by (*meson t4-space*)
then have *ω-limit-set x* ⊆ *A* ∪ *B*
ω-limit-set x ∩ *A* ≠ {} *ω-limit-set x* ∩ *B* ≠ {} **using** *pre* **by** *auto*
then obtain *p q* **where**
p: *ω-limit-point x p p* ∈ *A*
and *q*: *ω-limit-point x q q* ∈ *B*
using *ω-limit-set-def* **by** *auto*
from *ω-limit-pointE*[*OF p*(*1*)]
obtain *ps* **where** *ps*: *filterlim ps at-top sequentially*
(*flow0 x* ∘ *ps*) ⟶ *p* ∀ *n*. *0* ≤ *ps n* **by** *blast*
from *ω-limit-pointE*[*OF q*(*1*)]
obtain *qs* **where** *qs*: *filterlim qs at-top sequentially*
(*flow0 x* ∘ *qs*) ⟶ *q* ∀ *n*. *0* ≤ *qs n* **by** *blast*
have ∀ *n*. $\exists_F$ *t in at-top*. *flow0 x t* ∉ *A* ∧ *flow0 x t* ∉ *B* **unfolding** *frequently-at-top*
proof *safe*
fix *dummy mpre*
obtain *m* **where** *m* ≥ (*0*::*real*) *m* > *mpre*
by (*meson approximation-preproc-push-neg*(*1*) *gt-ex le-cases order-trans*)
from *ps* **obtain** *a* **where** *a*:*a* > *m* (*flow0 x a*) ∈ *A*
using ‹*open A*› *p* **unfolding** *tendsto-def filterlim-at-top eventually-sequentially*
by (*metis approximation-preproc-push-neg*(*1*) *comp-apply gt-ex le-cases order-trans*)
from *qs* **obtain** *b* **where** *b*:*b* > *a* (*flow0 x b*) ∈ *B*
using ‹*open B*› *q* **unfolding** *tendsto-def filterlim-at-top eventually-sequentially*
by (*metis approximation-preproc-push-neg*(*1*) *comp-apply gt-ex le-cases order-trans*)
have *continuous-on* {*a*..*b*} (*flow0 x*)
by (*metis Icc-subset-Ici-iff* ‹*0* ≤ *m*› ‹*m* < *a*› *approximation-preproc-push-neg*(*2*) *atMost-iff atMost-subset-iff continuous-on-subset le-cases local.flow-continuous-on props*(*1*) *subset-eq*)
from *connected-continuous-image*[*OF this connected-Icc*]
have *c*:*connected* (*flow0 x* ‘ {*a*..*b*}) **.**
have ∃ *t*∈ {*a*..*b*}. *flow0 x t* ∉ *A* ∧ *flow0 x t* ∉ *B*
proof (*rule ccontr*)
assume ¬ (∃ *t*∈{*a*..*b*}. *flow0 x t* ∉ *A* ∧ *flow0 x t* ∉ *B*)
then have *flow0 x* ‘ {*a*..*b*} ⊆ *A* ∪ *B* **by** *blast*
from *topological-space-class.connectedD*[*OF c* ‹*open A*› ‹*open B*› - *this*]
show *False* **using** *a b disj* **by** *force*

qed
thus $\exists\, n{>}mpre.\ \mathit{flow0}\ x\ n \notin A \wedge \mathit{flow0}\ x\ n \notin B$
by (*smt* ‹$mpre < m$› *a*(*1*) *atLeastAtMost-iff*)
qed
from *frequently-at-topE′*[*OF this filterlim-real-sequentially*]
obtain *s* **where** *s*: $\forall\, i.\ \mathit{flow0}\ x\ (s\ i) \notin A \wedge \mathit{flow0}\ x\ (s\ i) \notin B$
strict-mono s $\bigwedge n.\ \mathit{real}\ n \le s\ n$ **by** *blast*
then have $\forall\, n.\ (\mathit{flow0}\ x \circ s)\ n \in K$
by (*smt atLeast-iff comp-apply flowimg image-subset-iff of-nat-0-le-iff trapped-forward-def x*(*2*))
from *seq-compactE*[*OF props*(*2*) *this*]
obtain *r l* **where** *r*: $l \in K$ *strict-mono r* $(\mathit{flow0}\ x \circ s \circ r) \longrightarrow l$ **by** *blast*
have *filterlim s at-top sequentially*
using *s filterlim-at-top-mono filterlim-real-sequentially not-eventuallyD* **by** *blast*

from *filterlim-at-top-strict-mono*[*OF s*(*2*) *r*(*2*) *this*]
have *filterlim* $(s \circ r)$ *at-top sequentially* **.**
then have *ω-limit-point x l* **unfolding** *ω-limit-point-def* **using** *props*(*1*) *r*
by (*metis comp-assoc*)
moreover have $l \notin A$ **using** *s*(*1*) *r*(*3*) ‹*open A*› **unfolding** *tendsto-def* **by** *auto*
moreover have $l \notin B$ **using** *s*(*1*) *r*(*3*) ‹*open B*› **unfolding** *tendsto-def* **by** *auto*
ultimately show *False* **using** ‹*ω-limit-set x* $\subseteq A \cup B$› **unfolding** *ω-limit-set-def*
by *auto*
qed

lemma *ω-limit-set-in-compact-ω-limit-set-contained*:
shows $\forall\, y \in$ *ω-limit-set x. ω-limit-set y* $\subseteq$ *ω-limit-set x*
proof *safe*
fix *y z*
assume $y \in$ *ω-limit-set x* $z \in$ *ω-limit-set y*
then have *ω-limit-point y z* **unfolding** *ω-limit-set-def* **by** *auto*
from *ω-limit-pointE*[*OF this*]
obtain *s* **where** *s*: $(\mathit{flow0}\ y \circ s) \longrightarrow z$ **.**
have $\forall\, n.\ (\mathit{flow0}\ y \circ s)\ n \in$ *ω-limit-set x*
using ‹$y \in$ *ω-limit-set x*› *invariant-def*
ω-limit-set-in-compact-existence ω-limit-set-invariant trapped-iff-on-existence-ivl0
by *force*
thus $z \in$ *ω-limit-set x* **using** *closed-sequential-limits s ω-limit-set-closed*
by *blast*
qed

lemma *ω-limit-set-in-compact-α-limit-set-contained*:
assumes *zpx*: $z \in$ *ω-limit-set x*
shows *α-limit-set z* $\subseteq$ *ω-limit-set x*
proof
fix *w* **assume** $w \in$ *α-limit-set z*
then obtain *s* **where** *s*: $(\mathit{flow0}\ z \circ s) \longrightarrow w$
unfolding *α-limit-set-def α-limit-point-def*
by *auto*

from *ω-limit-set-invariant* **have** *invariant* (*ω-limit-set x*) **.**
then have $\forall$ *n*. (*flow0 z* $\circ$ *s*) *n* $\in$ *ω-limit-set x*
using *ω-limit-set-in-compact-existence*[*OF zpx*] *zpx*
using *invariant-def trapped-iff-on-existence-ivl0* **by** *fastforce*
from *closed-sequentially*[*OF ω-limit-set-closed*] *this s*
show *w* $\in$ *ω-limit-set x*
by *blast*
qed

end

Fundamental properties of the negative limit set

end context *auto-ll-on-open* **begin**

context
fixes *x K*
assumes *x*: *x* $\in$ *X trapped-backward x K*
assumes *K*: *compact K K* $\subseteq$ *X*
begin

private lemma *xrev*: *x* $\in$ *X rev.trapped-forward x K*
using *trapped-backward-iff-rev-trapped-forward x*(*2*)
by (*auto simp*: *rev-existence-ivl-eq0 rev-eq-flow x*(*1*))

lemma *α-limit-set-in-compact-subset*: *α-limit-set x* $\subseteq$ *K*
and *α-limit-set-in-compact-compact*: *compact* (*α-limit-set x*)
and *α-limit-set-in-compact-nonempty*: *α-limit-set x* $\neq$ {}
and *α-limit-set-in-compact-connected*: *connected* (*α-limit-set x*)
and *α-limit-set-in-compact-α-limit-set-contained*:
$\forall$ *y* $\in$ *α-limit-set x*. *α-limit-set y* $\subseteq$ *α-limit-set x*
and *α-limit-set-in-compact-tendsto*: ((λ*t*. *infdist* (*flow0 x t*) (*α-limit-set x*)) $\longrightarrow$ *0*) *at-bot*
using *rev.ω-limit-set-in-compact-subset*[*OF K xrev*]
using *rev.ω-limit-set-in-compact-compact*[*OF K xrev*]
using *rev.ω-limit-set-in-compact-nonempty*[*OF K xrev*]
using *rev.ω-limit-set-in-compact-connected*[*OF K xrev*]
using *rev.ω-limit-set-in-compact-ω-limit-set-contained*[*OF K xrev*]
using *rev.ω-limit-set-in-compact-tendsto*[*OF K xrev*]
unfolding *invariant-eq-rev α-limit-set-eq-rev existence-ivl-eq-rev flow-eq-rev0 filterlim-at-bot-mirror*
minus-minus
.

lemma *α-limit-set-in-compact-existence*:
shows $\bigwedge$*y*. *y* $\in$ *α-limit-set x* $\Longrightarrow$ *existence-ivl0 y* = *UNIV*
using *rev.ω-limit-set-in-compact-existence*[*OF K xrev*]
unfolding *α-limit-set-eq-rev existence-ivl-eq-rev0*
by *auto*

end
end

end

5 Periodic Orbits

theory *Periodic-Orbit*
  imports
    Ordinary-Differential-Equations.ODE-Analysis
    Analysis-Misc
    ODE-Misc
    Limit-Set
begin

Definition of closed and periodic orbits and their associated properties

context *auto-ll-on-open*
begin

TODO: not sure if the "closed orbit" terminology is standard Closed orbits have some non-zero recurrence time T where the flow returns to the initial state The period of a closed orbit is the infimum of all positive recurrence times Periodic orbits are the subset of closed orbits where the period is non-zero

definition *closed-orbit* $x \longleftrightarrow$
  $(\exists\ T \in \textit{existence-ivl0}\ x.\ T \neq 0 \wedge \textit{flow0}\ x\ T = x)$

definition *period* $x =$
  $\textit{Inf}\ \{T \in \textit{existence-ivl0}\ x.\ T > 0 \wedge \textit{flow0}\ x\ T = x\}$

definition *periodic-orbit* $x \longleftrightarrow$
  $\textit{closed-orbit}\ x \wedge \textit{period}\ x > 0$

lemma *recurrence-time-flip-sign*:
  assumes $T \in \textit{existence-ivl0}\ x\ \textit{flow0}\ x\ T = x$
  shows $-T \in \textit{existence-ivl0}\ x\ \textit{flow0}\ x\ (-T) = x$
  using *assms existence-ivl-reverse* **apply** *fastforce*
  using *assms flows-reverse* **by** *fastforce*

lemma *closed-orbit-recurrence-times-nonempty*:
  assumes *closed-orbit x*
  shows $\{T \in \textit{existence-ivl0}\ x.\ T > 0 \wedge \textit{flow0}\ x\ T = x\} \neq \{\}$
  apply *auto*
  using *assms*(*1*) **unfolding** *closed-orbit-def*
  by (*smt recurrence-time-flip-sign*)

lemma *closed-orbit-recurrence-times-bdd-below*:
  shows $\textit{bdd-below}\ \{T \in \textit{existence-ivl0}\ x.\ T > 0 \wedge \textit{flow0}\ x\ T = x\}$

unfolding *bdd-below-def*
by (*auto*) (*meson le-cases not-le*)

lemma *closed-orbit-period-nonneg*:
assumes *closed-orbit x*
shows *period x* $\geq$ *0*
unfolding *period-def*
using *assms*(*1*) **unfolding** *closed-orbit-def* **apply** (*auto intro*!:*cInf-greatest*)
by (*smt recurrence-time-flip-sign*)

lemma *closed-orbit-in-domain*:
assumes *closed-orbit x*
shows $x \in X$
using *assms* **unfolding** *closed-orbit-def*
using *local.mem-existence-ivl-iv-defined*(*2*) **by** *blast*

lemma *closed-orbit-global-existence*:
assumes *closed-orbit x*
shows *existence-ivl0 x* = *UNIV*
proof −
obtain *Tp* **where** *Tp* ≠ *0 Tp* ∈ *existence-ivl0 x flow0 x Tp* = *x* **using** *assms*
unfolding *closed-orbit-def* **by** *blast*
then obtain *T* **where** *T*: *T* > *0 T* ∈ *existence-ivl0 x flow0 x T* = *x*
by (*smt recurrence-time-flip-sign*)
have *apos*: *real n* ∗ *T* ∈ *existence-ivl0 x* ∧ *flow0 x* (*real n* ∗ *T*) = *x* **for** *n*
proof (*induction n*)
case *0*
then show *?case* **using** *closed-orbit-in-domain assms* **by** *auto*
next
case (*Suc n*)
fix *n*
assume *ih*:*real n* ∗ *T* ∈ *existence-ivl0 x* ∧ *flow0 x* (*real n* ∗ *T*) = *x*
then have *T* ∈ *existence-ivl0* (*flow0 x* (*real n* ∗ *T*)) **using** *T* **by** *metis*
then have *l*:*real n* ∗ *T* + *T* ∈ *existence-ivl0 x* **using** *ih*
using *existence-ivl-trans* **by** *blast*
have *flow0* (*flow0 x* (*real n* ∗ *T*)) *T* = *x* **using** *ih T* **by** *metis*
then have *r*: *flow0 x* (*real n* ∗ *T* + *T*) = *x*
by (*simp add*: *T*(*2*) *ih local.flow-trans*)
show *real* (*Suc n*) ∗ *T* ∈ *existence-ivl0 x* ∧ *flow0 x* (*real* (*Suc n*) ∗ *T*) = *x*
using *l r*
by (*simp add*: *add.commute distrib-left mult.commute*)
qed
then have *aneg*: −*real n* ∗ *T* ∈ *existence-ivl0 x* ∧ *flow0 x* (−*real n* ∗ *T*) = *x*
for *n*
by (*simp add*: *recurrence-time-flip-sign*)
have ∀ *t*. *t* ∈ *existence-ivl0 x*
proof *safe*
fix *t*
have *t* ≥ *0* ∨ *t* ≤ (*0*::*real*) **by** *linarith*

moreover {
assume $t \geq 0$
obtain k **where** *real* $k * T > t$
using $T(1)$ *ex-less-of-nat-mult* **by** *blast*
then have $t \in$ *existence-ivl0* x **using** *apos*
by (*meson* ‹$0 \leq t$› *atLeastAtMost-iff less-eq-real-def local.ivl-subset-existence-ivl subset-eq*)
}
moreover {
assume $t \leq 0$
obtain k **where** $-$ *real* $k * T < t$
by (*metis* $T(1)$ *add.inverse-inverse ex-less-of-nat-mult mult.commute mult-minus-right neg-less-iff-less*)
then have $t \in$ *existence-ivl0* x **using** *aneg*
by (*smt apos atLeastAtMost-iff calculation*(*2*) *local.existence-ivl-trans′ local.ivl-subset-existence-ivl mult-minus-left subset-eq*)
}
ultimately show $t \in$ *existence-ivl0* x **by** *blast*
qed
thus *?thesis* **by** *auto*
qed

lemma *recurrence-time-multiples*:
fixes n::*nat*
assumes $T \in$ *existence-ivl0* x $T \neq 0$ *flow0* x $T = x$
shows $\bigwedge t.$ *flow0* x $(t+T*n) =$ *flow0* x t
proof (*induction* n)
case *0*
then show *?case* **by** *auto*
next
case (*Suc* n)
fix n t
assume *ih* : ($\bigwedge t.$ *flow0* x $(t + T *$ *real* $n) =$ *flow0* x t)
have *closed-orbit* x **using** *assms* **unfolding** *closed-orbit-def* **by** *auto*
from *closed-orbit-global-existence*[*OF this*] **have** *ex*:*existence-ivl0* x = *UNIV* **.**
have *flow0* x $(t + T *$ *real* (*Suc* n)) = *flow0* x $(t+T*$*real* $n + T)$
by (*simp add*: *Groups.add-ac*(*3*) *add.commute distrib-left*)
also have ... = *flow0* (*flow0* x $(t+ T*$*real* n)) T **using** *ex*
by (*simp add*: *local.existence-ivl-trans′ local.flow-trans*)
also have ... = *flow0* (*flow0* x t) T **using** *ih* **by** *auto*
also have ... = *flow0* (*flow0* x T) t **using** *ex*
by (*metis UNIV-I add.commute local.existence-ivl-trans′ local.flow-trans*)
finally show *flow0* x $(t + T *$ *real* (*Suc* n)) = *flow0* x t **using** *assms*(*3*) **by** *simp*
qed

lemma *nasty-arithmetic1*:
fixes t T::*real*
assumes $T > 0$ $t \geq 0$

```
  obtains q r where t = (q::nat) ∗ T + r 0 ≤ r r < T
proof −
  define q where q = floor (t / T)
  have q:q ≥ 0 using assms unfolding q-def by auto
  from floor-divide-lower[OF assms(1), of t]
  have ql: q ∗ T ≤ t unfolding q-def .
  from floor-divide-upper[OF assms(1), of t]
  have qu: t < (q + 1)∗ T unfolding q-def by auto
  define r where r = t − q ∗ T
  have rl:0 ≤ r using ql unfolding r-def by auto
  have ru:r < T using qu unfolding r-def  by (simp add: distrib-right)
  show ?thesis using q r-def rl ru
   by (metis le-add-diff-inverse of-int-of-nat-eq plus-int-code(2) ql that zle-iff-zadd)
qed

lemma nasty-arithmetic2:
  fixes t T::real
  assumes T > 0 t ≤ 0
  obtains q r where t = (q::nat) ∗ (−T) + r 0 ≤ r r < T
proof −
  have −t ≥ 0 using assms(2) by linarith
  from nasty-arithmetic1[OF assms(1) this]
  obtain q r where qr: −t = (q::nat) ∗ T + r 0 ≤ r r < T by blast
  then have t = q ∗ (−T) − r  by auto
  then have t = (q+(1::nat)) ∗ (−T) + (T−r) by (simp add: distrib-right)
  thus ?thesis using qr(2−3)
    by (smt ‹t = real q ∗ − T − r› that)
qed

lemma recurrence-time-restricts-compact-flow:
  assumes T ∈ existence-ivl0 x T > 0 flow0 x T = x
  shows flow0 x ‘ UNIV = flow0 x ‘ {0..T}
  apply auto
proof −
  fix t
  have t ≥ 0 ∨ t ≤ (0::real) by linarith
  moreover {
    assume t ≥ 0
    from nasty-arithmetic1[OF assms(2) this]
    obtain q r where qr:t = (q::nat) ∗ T + r 0 ≤ r r < T by blast
    have T ≠ 0 using assms(2) by auto
    from recurrence-time-multiples[OF assms(1) this assms(3),of r q]
    have flow0 x t = flow0 x r
      by (simp add: qr(1) add.commute mult.commute)
    then have flow0 x t ∈ flow0 x ‘ {0..<T} using qr by auto
  }
  moreover {
    assume t ≤ 0
    from nasty-arithmetic2[OF assms(2) this]
```

obtain *q r* **where** *qr*:$t = (q::nat) * (-T) + r$ $0 \leq r$ $r < T$ **by** *blast*
have $-T \in$ *existence-ivl0 x* $-T \neq 0$ *flow0 x* $(-T) = x$ **using** *recurrence-time-flip-sign assms* **by** *auto*
from *recurrence-time-multiples*[*OF this*, *of r q*]
have *flow0 x t* = *flow0 x r*
by (*simp add*: *mult.commute qr*(*1*))
then have *flow0 x t* $\in$ *flow0 x* ‘ $\{0..<T\}$ **using** *qr* **by** *auto*
}
ultimately show *flow0 x t* $\in$ *flow0 x* ‘ $\{0..T\}$
by *auto*
qed

lemma *closed-orbitI*:
assumes $t \neq t'$ $t \in$ *existence-ivl0 y* $t' \in$ *existence-ivl0 y*
assumes *flow0 y t* = *flow0 y t'*
shows *closed-orbit y*
unfolding *closed-orbit-def*
by (*smt assms local.existence-ivl-reverse local.existence-ivl-trans local.flow-trans local.flows-reverse*)

lemma *flow0-image-UNIV*:
assumes *existence-ivl0 x* = *UNIV*
shows *flow0* (*flow0 x t*) ‘ *S* = *flow0 x* ‘ $(\lambda s.\ s + t)$ ‘ *S*
apply *auto*
apply (*metis UNIV-I add.commute assms image-eqI local.existence-ivl-trans' local.flow-trans*)
by (*metis UNIV-I add.commute assms imageI local.existence-ivl-trans' local.flow-trans*)

lemma *recurrence-time-restricts-compact-flow'*:
assumes $t < t'$ $t \in$ *existence-ivl0 y* $t' \in$ *existence-ivl0 y*
assumes *flow0 y t* = *flow0 y t'*
shows *flow0 y* ‘ *UNIV* = *flow0 y* ‘ $\{t..t'\}$
proof –
have *closed-orbit y*
using *assms*(*1–4*) *closed-orbitI inf.strict-order-iff* **by** *blast*
from *closed-orbit-global-existence*[*OF this*]
have *yex*: *existence-ivl0 y* = *UNIV* **.**
have *a1*:$t' - t \in$ *existence-ivl0* (*flow0 y t*)
by (*simp add*: *assms*(*2–3*) *local.existence-ivl-trans'*)
have *a2*:$t' - t > 0$ **using** *assms*(*1*) **by** *auto*
have *a3*:*flow0* (*flow0 y t*) $(t' - t)$ = *flow0 y t*
using *a1 assms*(*2*) *assms*(*4*) *local.flow-trans* **by** *fastforce*
from *recurrence-time-restricts-compact-flow*[*OF a1 a2 a3*]
have *eq*:*flow0* (*flow0 y t*) ‘ *UNIV* = *flow0* (*flow0 y t*) ‘ $\{0..\ t'-t\}$ **.**
from *flow0-image-UNIV*[*OF yex*, *of - UNIV*]
have *eql*:*flow0* (*flow0 y t*) ‘ *UNIV* = *flow0 y* ‘ *UNIV*
by (*metis* (*no-types*) *add.commute surj-def surj-plus*)
from *flow0-image-UNIV*[*OF yex*, *of -* $\{0..t'-t\}$]

have *eqr*:*flow0* (*flow0 y t*) ' {*0*.. *t'*−*t*} = *flow0 y* ' {*t*..*t'*} **by** *auto*
show *?thesis* **using** *eq eql eqr* **by** *auto*
qed

lemma *closed-orbitE'*:
assumes *closed-orbit x*
obtains *T* **where** *T* > *0* ⋀*t* (*n*::*nat*). *flow0 x* (*t*+*T*∗*n*) = *flow0 x t*
proof −
obtain *Tp* **where** *Tp* ≠ *0 Tp* ∈ *existence-ivl0 x flow0 x Tp* = *x* **using** *assms*
unfolding *closed-orbit-def* **by** *blast*
then obtain *T* **where** *T*: *T* > *0 T* ∈ *existence-ivl0 x flow0 x T* = *x*
by (*smt recurrence-time-flip-sign*)
thus *?thesis* **using** *recurrence-time-multiples T that* **by** *blast*
qed

lemma *closed-orbitE*:
assumes *closed-orbit x*
obtains *T* **where** *T* > *0* ⋀*t*. *flow0 x* (*t*+*T*) = *flow0 x t*
using *closed-orbitE'*
by (*metis assms mult.commute reals-Archimedean3*)

lemma *closed-orbit-flow-compact*:
assumes *closed-orbit x*
shows *compact*(*flow0 x* ' *UNIV*)
proof −
obtain *Tp* **where** *Tp* ≠ *0 Tp* ∈ *existence-ivl0 x flow0 x Tp* = *x* **using** *assms*
unfolding *closed-orbit-def* **by** *blast*
then obtain *T* **where** *T*: *T* ∈ *existence-ivl0 x T* > *0 flow0 x T* = *x*
by (*smt recurrence-time-flip-sign*)
from *recurrence-time-restricts-compact-flow*[*OF this*]
have *feq*: *flow0 x* ' *UNIV* = *flow0 x* ' {*0*..*T*} **.**
have *continuous-on* {*0*..*T*} (*flow0 x*)
by (*meson T*(*1*) *continuous-on-sequentially in-mono local.flow-continuous-on local.ivl-subset-existence-ivl*)
from *compact-continuous-image*[*OF this*]
have *compact* (*flow0 x* ' {*0*..*T*}) **by** *auto*
thus *?thesis* **using** *feq* **by** *auto*
qed

lemma *fixed-point-imp-closed-orbit-period-zero*:
assumes *x* ∈ *X*
assumes *f x* = *0*
shows *closed-orbit x period x* = *0*
proof −
from *fixpoint-sol*[*OF assms*] **have** *fp*:*existence-ivl0 x* = *UNIV* ⋀*t*. *flow0 x t* = *x* **by** *auto*
then have *co*:*closed-orbit x* **unfolding** *closed-orbit-def* **by** *blast*
have *a*: ∀ *y*>*0*. ∃ *a*∈{*T* ∈ *existence-ivl0 x*. *0* < *T* ∧ *flow0 x T* = *x*}. *a* < *y*
apply *auto*

using *fp*
by (*simp add*: *dense*)
from *cInf-le-iff*[*OF closed-orbit-recurrence-times-nonempty*[*OF co*]
closed-orbit-recurrence-times-bdd-below , *of 0*]
have *period x* $\leq$ *0* **unfolding** *period-def* **using** *a* **by** *auto*
from *closed-orbit-period-nonneg*[*OF co*] **have** *period x* $\geq$ *0* **.**
then have *period x* = *0* **using** ‹*period x* $\leq$ *0*› **by** *linarith*
thus *closed-orbit x period x* = *0* **using** *co* **by** *auto*
qed

lemma *closed-orbit-period-zero-fixed-point*:
assumes *closed-orbit x period x* = *0*
shows *f x* = *0*
proof (*rule ccontr*)
assume *f x* $\neq$ *0*
from *regular-locally-noteq*[*OF closed-orbit-in-domain*[*OF assms*(*1*)] *this*]
have $\forall_F$ *t in at 0*. *flow0 x t* $\neq$ *x* **.**
then obtain *r* **where** *r*>*0* $\forall$ *t*. *t* $\neq$ *0* $\wedge$ *dist t 0* < *r* $\longrightarrow$ *flow0 x t* $\neq$ *x* **unfolding** *eventually-at*
by *auto*
then have *period x* $\geq$ *r* **unfolding** *period-def*
apply (*auto intro*!:*cInf-greatest*)
apply (*meson assms*(*1*) *closed-orbit-def linorder-neqE-linordered-idom neg-0-less-iff-less recurrence-time-flip-sign*)
using *not-le* **by** *force*
thus *False* **using** *assms*(*2*) ‹*r* >*0*› **by** *linarith*
qed

lemma *closed-orbit-subset-ω-limit-set*:
assumes *closed-orbit x*
shows *flow0 x* ‘ *UNIV* $\subseteq$ *ω-limit-set x*
unfolding *ω-limit-set-def ω-limit-point-def*
proof *clarsimp*
fix *t*
from *closed-orbitE'*[*OF assms*]
obtain *T* **where** *T*: *0* < *T* $\bigwedge$*t n*. *flow0 x* (*t* + *T*∗ *real n*) = *flow0 x t* **by** *blast*
define *s* **where** *s* = (λ*n*::*nat*. *t* + *T* ∗ *real n*)
have *exist*: {*0*..} $\subseteq$ *existence-ivl0 x*
by (*simp add*: *assms closed-orbit-global-existence*)
have *l*:*filterlim s at-top sequentially* **unfolding** *s-def*
using *T*(*1*)
by (*auto intro*!:*filterlim-tendsto-add-at-top filterlim-tendsto-pos-mult-at-top*
simp add: *filterlim-real-sequentially*)
have *flow0 x* ∘ *s* = (λ*n*. *flow0 x t*) **unfolding** *o-def s-def* **using** *T*(*2*) **by** *simp*
then have *r*:(*flow0 x* ∘ *s*) $\longrightarrow$ *flow0 x t* **by** *auto*
show {*0*..} $\subseteq$ *existence-ivl0 x* $\wedge$ ($\exists$ *s*. *s* $\longrightarrow$ ∞ $\wedge$ (*flow0 x* ∘ *s*) $\longrightarrow$ *flow0 x t*)
using *exist l r* **by** *blast*
qed

lemma *closed-orbit-ω-limit-set*:
  assumes *closed-orbit x*
  shows *flow0 x ‘ UNIV* = *ω-limit-set x*
proof −
  have *ω-limit-set x* ⊆ *flow0 x ‘ UNIV*
    using *closed-orbit-global-existence*[*OF assms*]
    by (*intro ω-limit-set-in-compact-subset*)
      (*auto intro*!: *flow-in-domain*
        simp add: *assms closed-orbit-in-domain image-subset-iff trapped-forward-def*
        closed-orbit-flow-compact)
  thus *?thesis* **using** *closed-orbit-subset-ω-limit-set*[*OF assms*] **by** *auto*
qed

lemma *flow0-inj-on*:
  assumes $t \le t'$
  assumes $\{t..t'\} \subseteq$ *existence-ivl0 x*
  assumes $\bigwedge s.\ t < s \Longrightarrow s \le t' \Longrightarrow$ *flow0 x s* ≠ *flow0 x t*
  shows *inj-on* (*flow0 x*) $\{t..t'\}$
  apply (*rule inj-onI*)
proof (*rule ccontr*)
  fix *u v*
  assume *uv*: $u \in \{t..t'\}$ $v \in \{t..t'\}$ *flow0 x u* = *flow0 x v* $u \ne v$
  have $u < v \lor v < u$ **using** *uv*(*4*) **by** *linarith*
  moreover **{**
    assume $u < v$
    from *recurrence-time-restricts-compact-flow′*[*OF this - - uv*(*3*)]
    have *flow0 x ‘ UNIV* = *flow0 x ‘* $\{u..v\}$ **using** *uv*(*1*−*2*) *assms*(*2*) **by** *blast*
    then have *flow0 x t* ∈ *flow0 x ‘* $\{u..v\}$ **by** *auto*
    moreover have $u = t \lor$ *flow0 x t* ∉ *flow0 x ‘* $\{u..v\}$ **using** *assms*(*3*)
      by (*smt atLeastAtMost-iff image-iff uv*(*1*) *uv*(*2*))
    ultimately have *False* **using** *uv assms*(*3*)
      by *force*
  }
  moreover **{**
    assume $v < u$
    from *recurrence-time-restricts-compact-flow′*[*OF this - -*]
    have *flow0 x ‘ UNIV* = *flow0 x ‘* $\{v..u\}$
      by (*metis assms*(*2*) *subset-iff uv*(*1*) *uv*(*2*) *uv*(*3*))
    then have *flow0 x t* ∈ *flow0 x ‘* $\{v..u\}$ **by** *auto*
    moreover have $v = t \lor$ *flow0 x t* ∉ *flow0 x ‘* $\{v..u\}$ **using** *assms*(*3*)
      by (*smt atLeastAtMost-iff image-iff uv*(*1*) *uv*(*2*))
    ultimately have *False* **using** *uv assms*(*3*) **by** *force*
  }
  ultimately show *False* **by** *blast*
qed

lemma *finite-ω-limit-set-in-compact-imp-unique-fixed-point*:

assumes *compact K K* $\subseteq$ *X*
assumes *x* $\in$ *X trapped-forward x K*
assumes *finite* (ω*-limit-set x*)
obtains *y* **where** ω*-limit-set x* = {*y*} *f y* = *0*
proof −
from *connected-finite-iff-sing*[*OF* ω*-limit-set-in-compact-connected*]
obtain *y* **where** *y*: ω*-limit-set x* = {*y*}
using ω*-limit-set-in-compact-nonempty assms* **by** *auto*
have *f y* = *0*
proof (*rule ccontr*)
assume *fy*:*f y* $\neq$ *0*
from ω*-limit-set-in-compact-existence*[*OF assms*(*1*−*4*)]
have *yex*: *existence-ivl0 y* = *UNIV*
by (*simp add*: *y*)
then have *y* $\in$ *X*
by (*simp add*: *local.mem-existence-ivl-iv-defined*(*2*))
from *regular-locally-noteq*[*OF this fy*]
have $\forall_F$ *t in at 0*. *flow0 y t* $\neq$ *y* **.**
then obtain *r* **where** *r*: *r*>*0* $\forall$ *t*. *t* $\neq$ *0* $\wedge$ *dist t 0* < *r* $\longrightarrow$ *flow0 y t* $\neq$ *flow0 y 0*
unfolding *eventually-at* **using** ‹*y* $\in$ *X*›
by *auto*
then have $\bigwedge$*s*. *0* < *s* $\Longrightarrow$ *s* $\leq$ *r*/*2* $\Longrightarrow$ *flow0 y s* $\neq$ *flow0 y 0* **by** *simp*
from *flow0-inj-on*[*OF - - this*, *of r*/*2*]
obtain *inj-on*(*flow0 y*) {*0*..*r*/*2*} **using** *r yex* **by** *simp*
then have *infinite* (*flow0 y*‘{*0*..*r*/*2*}) **by** (*simp add*: *finite-image-iff r*(*1*))
moreover from ω*-limit-set-invariant*[*of x*]
have *flow0 y* ‘{*0*..*r*/*2*} $\subseteq$ ω*-limit-set x* **using** *y yex*
unfolding *invariant-def trapped-iff-on-existence-ivl0* **by** *auto*
ultimately show *False* **using** *y*
by (*metis assms*(*5*) *finite.emptyI subset-singleton-iff*)
qed
thus *?thesis* **using** *that y* **by** *auto*
qed

lemma *closed-orbit-periodic*:
assumes *closed-orbit x f x* $\neq$ *0*
shows *periodic-orbit x*
unfolding *periodic-orbit-def*
using *assms*(*1*) **apply** *auto*
proof (*rule ccontr*)
assume *closed-orbit x*
from *closed-orbit-period-nonneg*[*OF assms*(*1*)] **have** *nneg*: *period x* $\geq$ *0* **.**
assume $\neg$ *0* < *period x*
then have *period x* = *0* **using** *nneg* **by** *linarith*
from *closed-orbit-period-zero-fixed-point*[*OF assms*(*1*) *this*]
have *f x* = *0* **.**
thus *False* **using** *assms*(*2*) **by** *linarith*
qed

lemma *periodic-orbitI*:
  assumes $t \neq t'$ $t \in$ *existence-ivl0 y* $t' \in$ *existence-ivl0 y*
  assumes *flow0 y t* = *flow0 y t'*
  assumes $f\ y \neq 0$
  shows *periodic-orbit y*
proof –
  have *y*:$y \in X$
    using *assms*(*3*) *local.mem-existence-ivl-iv-defined*(*2*) **by** *blast*
  from *closed-orbitI*[*OF assms*(*1*–*4*)] **have** *closed-orbit y* **.**
  from *closed-orbit-periodic*[*OF this assms*(*5*)]
  show *?thesis* **.**
qed

lemma *periodic-orbit-recurrence-times-closed*:
  assumes *periodic-orbit x*
  shows *closed* $\{T \in$ *existence-ivl0 x*. $T > 0 \wedge$ *flow0 x T* $= x\}$
proof –
  have *a1*:$x \in X$
    using *assms closed-orbit-in-domain periodic-orbit-def* **by** *auto*
  have *a2*:$f\ x \neq 0$
    using *assms closed-orbit-in-domain fixed-point-imp-closed-orbit-period-zero*(*2*)
periodic-orbit-def **by** *auto*
  from *regular-locally-noteq*[*OF a1 a2*] **have**
    $\forall_F$ *t in at 0*. *flow0 x t* $\neq x$ **.**
   then obtain *r* **where** *r*:$r>0$ $\forall t.\ t \neq 0 \wedge$ *dist t 0* $< r \longrightarrow$ *flow0 x t* $\neq x$
unfolding *eventually-at*
    by *auto*
  show *?thesis*
  proof (*auto intro*!:*discrete-imp-closed*[*OF r*(*1*)])
    fix *t1 t2*
    assume *t12*: $t1 > 0$ *flow0 x t1* $= x$ $t2 > 0$ *flow0 x t2* $= x$ *dist t2 t1* $< r$
    then have *fx*: *flow0 x* (*t1*–*t2*) $= x$
    by (*smt a1 assms closed-orbit-global-existence existence-ivl-zero general.existence-ivl-initial-time-iff*
local.flow-trans periodic-orbit-def)
    have *dist* (*t1*–*t2*) $0 < r$ **using** *t12*(*5*)
      by (*simp add*: *dist-norm*)
    thus $t2 = t1$ **using** *r fx*
      by *smt*
  qed
qed

lemma *periodic-orbit-period*:
  assumes *periodic-orbit x*
  shows *period x* > 0 *flow0 x* (*period x*) $= x$
proof –
  from *periodic-orbit-recurrence-times-closed*[*OF assms*(*1*)]
  have *cl*: *closed* $\{T \in$ *existence-ivl0 x*. $T > 0 \wedge$ *flow0 x T* $= x\}$ **.**
  have *closed-orbit x* **using** *assms*(*1*) **unfolding** *periodic-orbit-def* **by** *auto*

from *closed-contains-Inf*[*OF closed-orbit-recurrence-times-nonempty*[*OF this*]
closed-orbit-recurrence-times-bdd-below cl]
have *period* $x \in \{T \in$ *existence-ivl0* $x.\ T > 0 \wedge$ *flow0* $x\ T = x\}$ **unfolding** *period-def* **.**
thus *period* $x > 0$ *flow0* x (*period* x) $= x$ **by** *auto*
qed

lemma *closed-orbit-flow0*:
assumes *closed-orbit x*
shows *closed-orbit* (*flow0 x t*)
proof −
from *closed-orbit-global-existence*[*OF assms*]
have *existence-ivl0* $x = UNIV$ **.**
from *closed-orbitE*[*OF assms*]
obtain T **where** $T > 0$ *flow0* x $(t+T) =$ *flow0 x t*
by *metis*
thus *?thesis* **unfolding** *closed-orbit-def*
by (*metis UNIV-I* ‹*existence-ivl0* $x = UNIV$› *less-irrefl local.existence-ivl-trans'* *local.flow-trans*)
qed

lemma *periodic-orbit-imp-flow0-regular*:
assumes *periodic-orbit x*
shows f (*flow0 x t*) $\neq 0$
by (*metis UNIV-I assms closed-orbit-flow0 closed-orbit-global-existence closed-orbit-in-domain fixed-point-imp-closed-orbit-period-zero*(*2*) *fixpoint-sol*(*2*) *less-irrefl local.flows-reverse periodic-orbit-def*)

lemma *fixed-point-imp-ω-limit-set*:
assumes $x \in X$ $f\ x = 0$
shows *ω-limit-set* $x = \{x\}$
proof −
have *closed-orbit x*
by (*metis assms fixed-point-imp-closed-orbit-period-zero*(*1*))
from *closed-orbit-ω-limit-set*[*OF this*]
have *flow0 x* ‘ *UNIV* = *ω-limit-set x* **.**
thus *?thesis*
by (*metis assms*(*1*) *assms*(*2*) *fixpoint-sol*(*2*) *image-empty image-insert image-subset-iff insertI1 rangeI subset-antisym*)
qed

end

context *auto-ll-on-open* **begin**

lemma *closed-orbit-eq-rev*: *closed-orbit* x = *rev.closed-orbit* x
unfolding *closed-orbit-def rev.closed-orbit-def rev-eq-flow rev-existence-ivl-eq0*
by *auto*

lemma *closed-orbit-α-limit-set*:
  assumes *closed-orbit x*
  shows *flow0 x ‘ UNIV = α-limit-set x*
  using *rev.closed-orbit-ω-limit-set assms*
  unfolding *closed-orbit-eq-rev α-limit-set-eq-rev flow-image-eq-rev range-uminus*
.

lemma *fixed-point-imp-α-limit-set*:
  assumes *x ∈ X f x = 0*
  shows *α-limit-set x = {x}*
  using *rev.fixed-point-imp-ω-limit-set assms*
  unfolding *α-limit-set-eq-rev*
  by *auto*

lemma *finite-α-limit-set-in-compact-imp-unique-fixed-point*:
  assumes *compact K K ⊆ X*
  assumes *x ∈ X trapped-backward x K*
  assumes *finite* (*α-limit-set x*)
  obtains *y* **where** *α-limit-set x = {y} f y = 0*
proof −
  from *rev.finite-ω-limit-set-in-compact-imp-unique-fixed-point*[*OF*
    assms(*1−5*)[*unfolded trapped-backward-iff-rev-trapped-forward α-limit-set-eq-rev*]]
  show *?thesis* **using** *that*
    unfolding *α-limit-set-eq-rev*
    by *auto*
qed
end

end

6 Poincare Bendixson Theory

theory *Poincare-Bendixson*
  imports
    Ordinary-Differential-Equations.ODE-Analysis
    Analysis-Misc ODE-Misc Periodic-Orbit
begin

6.1 Flow to Path

context *auto-ll-on-open* **begin**

definition *flow-to-path x t t′ = flow0 x ∘ linepath t t′*

lemma *pathstart-flow-to-path*[*simp*]:
  shows *pathstart* (*flow-to-path x t t′*) = *flow0 x t*
  unfolding *flow-to-path-def*
  by (*auto simp add*: *pathstart-compose*)

lemma *pathfinish-flow-to-path*[*simp*]:
  shows *pathfinish* (*flow-to-path x t t′*) = *flow0 x t′*
  unfolding *flow-to-path-def*
  by (*auto simp add*: *pathfinish-compose*)

lemma *flow-to-path-unfold*:
  shows *flow-to-path x t t′ s* = *flow0 x* ((*1* − *s*) ∗ *t* + *s* ∗ *t′*)
  unfolding *flow-to-path-def o-def linepath-def* **by** *auto*

lemma *subpath0-flow-to-path*:
  shows (*subpath 0 u* (*flow-to-path x t t′*)) = *flow-to-path x t* (*t* + *u*∗(*t′*−*t*))
  unfolding *flow-to-path-def subpath-image subpath0-linepath*
  by *auto*

lemma *path-image-flow-to-path*[*simp*]:
  assumes $t \le t'$
  shows *path-image* (*flow-to-path x t t′*) = *flow0 x* ‘ {*t*..*t′*}
  unfolding *flow-to-path-def path-image-compose path-image-linepath*
  using *assms real-Icc-closed-segment* **by** *auto*

lemma *flow-to-path-image0-right-open*[*simp*]:
  assumes $t < t'$
  shows *flow-to-path x t t′* ‘ {*0*..<*1*} = *flow0 x* ‘{*t*..<*t′*}
  unfolding *flow-to-path-def image-comp*[*symmetric*] *linepath-image0-right-open-real*[*OF assms*]
  by *auto*

lemma *flow-to-path-path*:
  assumes $t \le t'$
  assumes {*t*..*t′*} ⊆ *existence-ivl0 x*
  shows *path* (*flow-to-path x t t′*)
proof −
  have $x \in X$
    using *assms*(*1*) *assms*(*2*) *subset-empty* **by** *fastforce*
  have ⋀*xa*. *0* ≤ *xa* ⟹ *xa* ≤ *1* ⟹ (*1* − *xa*) ∗ *t* + *xa* ∗ *t′* ≤ *t′*
    by (*simp add*: *assms*(*1*) *convex-bound-le*)
  moreover have ⋀*xa*. *0* ≤ *xa* ⟹ *xa* ≤ *1* ⟹ *t* ≤ (*1* − *xa*) ∗ *t* + *xa* ∗ *t′* **using** *assms*(*1*)
    by (*metis add.commute add-diff-cancel-left′ diff-diff-eq2 diff-le-eq mult.commute mult.right-neutral mult-right-mono right-diff-distrib′*)
  ultimately have ⋀*xa*. *0* ≤ *xa* ⟹ *xa* ≤ *1* ⟹ (*1* − *xa*) ∗ *t* + *xa* ∗ *t′* ∈ *existence-ivl0 x*
    using *assms*(*2*) **by** *auto*
  thus *?thesis* **unfolding** *path-def flow-to-path-def linepath-def*
    by (*auto intro*!:*continuous-intros simp add* :‹*x* ∈ *X*›)
qed

lemma *flow-to-path-arc*:

assumes $t \leq t'$
assumes $\{t..t'\} \subseteq$ *existence-ivl0 x*
assumes $\forall\, s \in \{t<..<t'\}.$ *flow0 x s* $\neq$ *flow0 x t*
assumes *flow0 x t* $\neq$ *flow0 x t'*
shows *arc* (*flow-to-path x t t'*)
unfolding *arc-def*
proof *safe*
from *flow-to-path-path*[*OF assms*(*1*−*2*)]
show *path* (*flow-to-path x t t'*) **.**
next
show *inj-on* (*flow-to-path x t t'*) {*0*..*1*}
unfolding *flow-to-path-def*
apply (*rule comp-inj-on*)
apply (*metis assms*(*4*) *inj-on-linepath*)
using *assms path-image-linepath*[*of t t'*] **apply** (*auto intro*!:*flow0-inj-on*)
using *flow0-inj-on greaterThanLessThan-iff linepath-image-01 real-Icc-closed-segment*
by *fastforce*
qed

end

locale *c1-on-open-R2* = *c1-on-open-euclidean f f' X* **for** *f*::*'a*::*executable-euclidean-space* ⇒ - **and** *f'* **and** *X* +
assumes *dim2*: *DIM*(*'a*) = *2*
begin

6.2 2D Line segments

Line segments are specified by two endpoints The closed line segment from x to y is given by the set x–y and x<–<y for the open segment

Rotates a vector clockwise 90 degrees

definition *rot* (*v*::*'a*) = (*eucl-of-list* [*nth-eucl v 1*, −*nth-eucl v 0*]::*'a*)

lemma *exhaust2-nat*: ($\forall$ *i*<(*2*::*nat*). *P i*) $\longleftrightarrow$ *P 0* $\wedge$ *P 1*
using *less-2-cases* **by** *auto*
lemma *sum2-nat*: ($\sum$ *i*<(*2*::*nat*). *P i*) = *P 0* + *P 1*
by (*simp add*: *eval-nat-numeral*)

lemmas *vec-simps* =
eucl-eq-iff[**where** *'a*=*'a*] *dim2 eucl-of-list-eucl-nth exhaust2-nat*
plus-nth-eucl
minus-nth-eucl
uminus-nth-eucl
scaleR-nth-eucl
inner-nth-eucl
sum2-nat
algebra-simps

lemma *minus-expand*:
  shows $(x::'a)-y = (eucl\text{-}of\text{-}list\ [x\$_e 0\ -\ y\$_e 0,\ x\$_e 1\ -\ y\$_e 1])$
  by (*simp add:vec-simps*)

lemma *dot-ortho*[*simp*]: $x \cdot rot\ x = 0$
  unfolding *rot-def minus-expand*
  by (*simp add*: *vec-simps*)

lemma *nrm-dot*:
  shows $((x::'a)-y) \cdot (rot\ (x-y)) = 0$
  unfolding *rot-def minus-expand*
  by (*simp add*: *vec-simps*)

lemma *nrm-reverse*: $a \cdot (rot\ (x-y)) = -\ a \cdot (rot\ (y-x))$ **for** $x\ y::'a$
  unfolding *rot-def*
  by (*simp add:vec-simps*)

lemma *norm-rot*: $norm\ (rot\ v) = norm\ v$ **for** $v::'a$
  unfolding *rot-def*
  by (*simp add:vec-simps norm-nth-eucl L2-set-def*)

lemma *rot-rot*[*simp*]:
  shows $rot\ (rot\ v) = -v$
  unfolding *rot-def*
  by (*simp add:vec-simps*)

lemma *rot-scaleR*[*simp*]:
  shows $rot\ (\ u *_R v) =\ u *_R (rot\ v)$
  unfolding *rot-def*
  by (*simp add:vec-simps*)

lemma *rot-0*[*simp*]: $rot\ 0 = 0$
  using *rot-scaleR*[*of 0*] **by** *simp*

lemma *rot-eq-0-iff*[*simp*]: $rot\ x = 0 \longleftrightarrow x = 0$
  apply (*auto simp*: *rot-def*)
   apply (*metis One-nat-def norm-eq-zero norm-rot norm-zero rot-def*)
  using *rot-0 rot-def* **by** *auto*

lemma *in-segment-inner-rot*:
  $(x\ -\ a) \cdot rot\ (b\ -\ a) = 0$
  if $x \in \{a{-}{-}b\}$
proof −
  from *that* **obtain** u **where** x: $x = a + u *_R (b\ -\ a)\ 0 \le u\ u \le 1$
    by (*auto simp*: *in-segment algebra-simps*)
  show *?thesis*
    unfolding x
    by (*simp add*: *inner-add-left nrm-dot*)
qed

lemma *inner-rot-in-segment*:
  $x \in range\ (\lambda u.\ a + u *_R (b - a))$
  if $(x - a) \cdot rot\ (b - a) = 0\ a \neq b$
proof –
  from *that* **have**
    x0: $b\ \$_e\ 0 = a\ \$_e\ 0 \Longrightarrow x\ \$_e\ 0 =$
      $(a\ \$_e\ 0 * b\ \$_e\ Suc\ 0 - b\ \$_e\ 0 * a\ \$_e\ Suc\ 0 + (b\ \$_e\ 0 - a\ \$_e\ 0) * x\ \$_e\ Suc\ 0)\ /$
      $(b\ \$_e\ Suc\ 0 - a\ \$_e\ Suc\ 0)$
    and *x1*: $b\ \$_e\ 0 \neq a\ \$_e\ 0 \Longrightarrow x\ \$_e\ Suc\ 0 =$
      $((b\ \$_e\ Suc\ 0 - a\ \$_e\ Suc\ 0) * x\ \$_e\ 0 - a\ \$_e\ 0 * b\ \$_e\ Suc\ 0 + b\ \$_e\ 0 * a\ \$_e\ Suc\ 0)\ /\ (b\ \$_e\ 0 - a\ \$_e\ 0)$
    by (*auto simp*: *rot-def vec-simps divide-simps*)
  define u **where** $u = (if\ b\ \$_e\ 0 - a\ \$_e\ 0 \neq 0$
    $then\ ((x\ \$_e\ 0 - a\ \$_e\ 0)\ /\ (b\ \$_e\ 0 - a\ \$_e\ 0))$
    $else\ ((x\ \$_e\ 1 - a\ \$_e\ 1)\ /\ (b\ \$_e\ 1 - a\ \$_e\ 1)))$

  show *?thesis*
    apply $(cases\ b\ \$_e\ 0 - a\ \$_e\ 0 = 0)$
    subgoal
      using *that*(*2*)
      apply (*auto intro*!: *image-eqI*[**where** $x=((x\ \$_e\ 1 - a\ \$_e\ 1)\ /\ (b\ \$_e\ 1 - a\ \$_e\ 1))$]
        simp: *vec-simps x0 divide-simps algebra-simps*)
      apply (*metis ab-semigroup-mult-class.mult-ac*(*1*) *mult.commute sum-sqs-eq*)
      by (*metis mult.commute mult.left-commute sum-sqs-eq*)
    subgoal
      apply (*auto intro*!: *image-eqI*[**where** $x=((x\ \$_e\ 0 - a\ \$_e\ 0)\ /\ (b\ \$_e\ 0 - a\ \$_e\ 0))$]
        simp: *vec-simps x1 divide-simps algebra-simps*)
      apply (*metis ab-semigroup-mult-class.mult-ac*(*1*) *mult.commute sum-sqs-eq*)
      by (*metis mult.commute mult.left-commute sum-sqs-eq*)
    done
qed

lemma *in-open-segment-iff-rot*:
  $x \in \{a{<}{-}{-}{<}b\} \longleftrightarrow (x - a) \cdot rot\ (b - a) = 0 \land x \cdot (b - a) \in \{a \cdot (b - a) <..< b \cdot (b - a)\}$
  if $a \neq b$
  unfolding *open-segment-line-hyperplanes*[*OF that*]
  by (*auto simp*: *nrm-dot intro*!: *inner-rot-in-segment*)

lemma *in-open-segment-rotD*:
  $x \in \{a{<}{-}{-}{<}b\} \Longrightarrow (x - a) \cdot rot\ (b - a) = 0 \land x \cdot (b - a) \in \{a \cdot (b - a) <..< b \cdot (b - a)\}$
  by (*subst in-open-segment-iff-rot*[*symmetric*]) *auto*

lemma *in-closed-segment-iff-rot*:

$x \in \{a{-}{-}b\} \longleftrightarrow (x - a) \cdot rot\ (b - a) = 0 \wedge x \cdot (b - a) \in \{a{\cdot}(b - a) .. b \cdot (b - a)\}$
if $a \neq b$
unfolding *closed-segment-line-hyperplanes*[*OF that*] **using** *that*
by (*auto simp*: *nrm-dot intro*!: *inner-rot-in-segment*)

lemma *in-segment-inner-rot2*:
$(x - y) \cdot rot\ (a - b) = 0$
if $x \in \{a{-}{-}b\}$ $y \in \{a{-}{-}b\}$
proof −
from *that* **obtain** u **where** x: $x = a + u *_R (b - a)$ $0 \leq u$ $u \leq 1$
by (*auto simp*: *in-segment algebra-simps*)
from *that* **obtain** v **where** y: $y = a + v *_R (b - a)$ $0 \leq v$ $v \leq 1$
by (*auto simp*: *in-segment algebra-simps*)
show *?thesis*
unfolding x y
apply (*auto simp*: *inner-add-left*)
by (*smt add-diff-cancel-left′ in-segment-inner-rot inner-diff-left minus-diff-eq nrm-reverse that*(*1*) *that*(*2*) *x*(*1*) *y*(*1*))
qed

lemma *closed-segment-surface*:
$a \neq b \Longrightarrow \{a{-}{-}b\} = \{ x \in \{x.\ x \cdot (b - a) \in \{a{\cdot}(b - a) .. b \cdot (b - a)\}\}.\ (x - a) \cdot rot\ (b - a) = 0\}$
by (*auto simp*: *in-closed-segment-iff-rot*)

lemma *rot-diff-commute*: $rot\ (b - a) = -rot(a - b)$
apply (*auto simp*: *rot-def algebra-simps*)
by (*metis One-nat-def minus-diff-eq rot-def rot-rot*)

6.3 Bijection Real-Complex for Jordan Curve Theorem

definition *complex-of* $(x{::}'a) = x\$_e 0 + \mathrm{i} * x\$_e 1$

definition *real-of* $(x{::}complex) = (eucl\text{-}of\text{-}list\ [Re\ x,\ Im\ x]{::}'a)$

lemma *complex-of-linear*:
shows *linear complex-of*
unfolding *complex-of-def*
apply (*auto intro*!:*linearI simp add*: *distrib-left plus-nth-eucl*)
by (*simp add*: *of-real-def scaleR-add-right scaleR-nth-eucl*)

lemma *complex-of-bounded-linear*:
shows *bounded-linear complex-of*
unfolding *complex-of-def*
apply (*auto intro*!:*bounded-linearI′ simp add*: *distrib-left plus-nth-eucl*)
by (*simp add*: *of-real-def scaleR-add-right scaleR-nth-eucl*)

lemma *real-of-linear*:

shows *linear real-of*
unfolding *real-of-def*
by (*auto intro*!:*linearI simp add*: *vec-simps*)

lemma *real-of-bounded-linear*:
shows *bounded-linear real-of*
unfolding *real-of-def*
by (*auto intro*!:*bounded-linearI′ simp add*: *vec-simps*)

lemma *complex-of-real-of*:
(*complex-of* ∘ *real-of*) = *id*
unfolding *complex-of-def real-of-def*
using *complex-eq* **by** (*auto simp add*:*vec-simps*)

lemma *real-of-complex-of*:
(*real-of* ∘ *complex-of*) = *id*
unfolding *complex-of-def real-of-def*
using *complex-eq* **by** (*auto simp add*:*vec-simps*)

lemma *complex-of-bij*:
shows *bij* (*complex-of*)
using *o-bij*[*OF real-of-complex-of complex-of-real-of*] **.**

lemma *real-of-bij*:
shows *bij* (*real-of*)
using *o-bij*[*OF complex-of-real-of real-of-complex-of*] **.**

lemma *real-of-inj*:
shows *inj* (*real-of*)
using *real-of-bij*
using *bij-betw-imp-inj-on* **by** *auto*

lemma *Jordan-curve-R2*:
fixes *c* :: *real* ⇒ ′*a*
assumes *simple-path c pathfinish c* = *pathstart c*
obtains *inside outside* **where**
inside ≠ {} *open inside connected inside*
outside ≠ {} *open outside connected outside*
bounded inside ¬ *bounded outside*
inside ∩ *outside* = {}
inside ∪ *outside* = − *path-image c*
frontier inside = *path-image c*
frontier outside = *path-image c*
proof −
from *simple-path-linear-image-eq*[*OF complex-of-linear*]
have *a1*:*simple-path* (*complex-of* ∘ *c*) **using** *assms*(*1*) *complex-of-bij*
using *bij-betw-imp-inj-on* **by** *blast*
have *a2*:*pathfinish* (*complex-of* ∘ *c*) = *pathstart* (*complex-of* ∘ *c*)
using *assms*(*2*) **by** (*simp add*:*pathstart-compose pathfinish-compose*)

from *Jordan-curve*[*OF a1 a2*]
obtain *inside outside* **where** *io*:
inside ≠ {} *open inside connected inside*
outside ≠ {} *open outside connected outside*
bounded inside ¬ *bounded outside inside* ∩ *outside* = {}
inside ∪ *outside* = − *path-image* (*complex-of* ∘ *c*)
frontier inside = *path-image* (*complex-of* ∘ *c*)
frontier outside = *path-image* (*complex-of* ∘ *c*) **by** *blast*
let *?rin* = *real-of* ‘ *inside*
let *?rout* = *real-of* ‘ *outside*
have *i*: *inside* = *complex-of* ‘ *?rin* **using** *complex-of-real-of* **unfolding** *image-comp*
by *auto*
have *o*: *outside* = *complex-of* ‘ *?rout* **using** *complex-of-real-of* **unfolding** *image-comp*
by *auto*
have *c*: *path-image*(*complex-of* ∘ *c*) = *complex-of* ‘ (*path-image c*)
by (*simp add*: *path-image-compose*)
have *?rin* ≠ {} **using** *io* **by** *auto*
moreover from *open-bijective-linear-image-eq*[*OF real-of-linear real-of-bij*]
have *open ?rin* **using** *io* **by** *auto*
moreover from *connected-linear-image*[*OF real-of-linear*]
have *connected ?rin* **using** *io* **by** *auto*
moreover have *?rout* ≠ {} **using** *io* **by** *auto*
moreover from *open-bijective-linear-image-eq*[*OF real-of-linear real-of-bij*]
have *open ?rout* **using** *io* **by** *auto*
moreover from *connected-linear-image*[*OF real-of-linear*]
have *connected ?rout* **using** *io* **by** *auto*
moreover from *bounded-linear-image*[*OF io*(*7*) *real-of-bounded-linear*]
have *bounded ?rin* **.**
moreover from *bounded-linear-image*[*OF - complex-of-bounded-linear*]
have ¬ *bounded ?rout* **using** *io*(*8*) *o*
by *force*
from *image-Int*[*OF real-of-inj*]
have *?rin* ∩ *?rout* = {} **using** *io*(*9*) **by** *auto*
moreover from *bij-image-Compl-eq*[*OF complex-of-bij*]
have *?rin* ∪ *?rout* = − *path-image c* **using** *io*(*10*) **unfolding** *c*
by (*metis id-apply image-Un image-comp image-cong image-ident real-of-complex-of*)
moreover from *closure-injective-linear-image*[*OF real-of-linear real-of-inj*]
have *frontier ?rin* = *path-image c* **using** *io*(*11*)
unfolding *frontier-closures c*
by (*metis* ‹⋀*B A*. *real-of* ‘ (*A* ∩ *B*) = *real-of* ‘ *A* ∩ *real-of* ‘ *B*› *bij-image-Compl-eq c calculation*(*9*) *compl-sup double-compl io*(*10*) *real-of-bij*)
moreover from *closure-injective-linear-image*[*OF real-of-linear real-of-inj*]
have *frontier ?rout* = *path-image c* **using** *io*(*12*)
unfolding *frontier-closures c*
by (*metis* ‹⋀*B A*. *real-of* ‘ (*A* ∩ *B*) = *real-of* ‘ *A* ∩ *real-of* ‘ *B*› *bij-image-Compl-eq c calculation*(*10*) *frontier-closures io*(*11*) *real-of-bij*)

ultimately show *?thesis*
  by (*meson* ‹¬ *bounded* (*real-of* ‘ *outside*)› *that*)
qed

corollary *Jordan-inside-outside-R2*:
  fixes $c :: real \Rightarrow 'a$
  assumes *simple-path c pathfinish c = pathstart c*
  shows *inside*(*path-image c*) $\neq$ {} ∧
      open(*inside*(*path-image c*)) ∧
      connected(*inside*(*path-image c*)) ∧
      outside(*path-image c*) $\neq$ {} ∧
      open(*outside*(*path-image c*)) ∧
      connected(*outside*(*path-image c*)) ∧
      bounded(*inside*(*path-image c*)) ∧
      ¬ *bounded*(*outside*(*path-image c*)) ∧
      inside(*path-image c*) ∩ *outside*(*path-image c*) = {} ∧
      inside(*path-image c*) ∪ *outside*(*path-image c*) =
      − *path-image c* ∧
      frontier(*inside*(*path-image c*)) = *path-image c* ∧
      frontier(*outside*(*path-image c*)) = *path-image c*
proof −
  obtain *inner outer*
    where ∗: *inner* $\neq$ {} *open inner connected inner*
      outer $\neq$ {} *open outer connected outer*
      bounded inner ¬ *bounded outer inner* ∩ *outer* = {}
      inner ∪ *outer* = − *path-image c*
      frontier inner = *path-image c*
      frontier outer = *path-image c*
    using *Jordan-curve-R2* [*OF assms*] **by** *blast*
  then have *inner*: *inside*(*path-image c*) = *inner*
    by (*metis dual-order.antisym inside-subset interior-eq interior-inside-frontier*)
  have *outer*: *outside*(*path-image c*) = *outer*
    using ‹*inner* ∪ *outer* = − *path-image c*› ‹*inside* (*path-image c*) = *inner*›
      outside-inside ‹*inner* ∩ *outer* = {}› **by** *auto*
  show *?thesis*
    using ∗ **by** (*auto simp*: *inner outer*)
qed

lemma *jordan-points-inside-outside*:
  fixes $p :: real \Rightarrow 'a$
  assumes $0 < e$
  assumes *jordan*: *simple-path p pathfinish p = pathstart p*
  assumes *x*: $x \in$ *path-image p*
  obtains *y z* **where** $y \in$ *inside* (*path-image p*) $y \in$ *ball x e*
    $z \in$ *outside* (*path-image p*) $z \in$ *ball x e*
proof −
  from *Jordan-inside-outside-R2*[*OF jordan*]
  have *xi*: $x \in$ *frontier*(*inside* (*path-image p*)) **and**

xo: *x* ∈ *frontier*(*outside* (*path-image p*))
using *x* **by** *auto*
obtain *y* **where** *y*:*y* ∈ *inside* (*path-image p*) *y* ∈ *ball x e* **using** ‹*0* < *e*› *xi*
unfolding *frontier-straddle*
by *auto*
obtain *z* **where** *z*:*z* ∈ *outside* (*path-image p*) *z* ∈ *ball x e* **using** ‹*0* < *e*› *xo*
unfolding *frontier-straddle*
by *auto*
show *?thesis* **using** *y z that* **by** *blast*
qed

lemma *eventually-at-open-segment*:
assumes *x* ∈ {*a*<−−<*b*}
shows ∀ *F* *y in at x*. (*y*−*a*) • *rot*(*a*−*b*) = *0* ⟶ *y* ∈ {*a* <−−< *b*}
proof −
from *assms* **have** *a* ≠ *b* **by** *auto*
from *assms* **have** *x*: (*x* − *a*) • *rot* (*b* − *a*) = *0 x* • (*b* − *a*) ∈ {*a* • (*b* − *a*)<..<*b* • (*b* − *a*)}
unfolding *in-open-segment-iff-rot*[*OF* ‹*a* ≠ *b*›]
by *auto*
then have ∀ *F* *y in at x*. *y* • (*b* − *a*) ∈ {*a* • (*b* − *a*)<..<*b* • (*b* − *a*)}
by (*intro topological-tendstoD*) (*auto intro*!: *tendsto-intros*)
then show *?thesis*
by *eventually-elim* (*auto simp*: *in-open-segment-iff-rot*[*OF* ‹*a* ≠ *b*›] *nrm-reverse*[*of* - *a b*] *algebra-simps dist-commute*)
qed

lemma *linepath-ball*:
assumes *x* ∈ {*a*<−−<*b*}
obtains *e* **where** *e* > *0 ball x e* ∩ {*y*. (*y*−*a*) • *rot*(*a*−*b*) = *0*} ⊆ {*a* <−−< *b*}
proof −
from *eventually-at-open-segment*[*OF assms*] *assms*
obtain *e* **where** *0* < *e ball x e* ∩ {*y*. (*y* − *a*) • *rot* (*a* − *b*) = *0*} ⊆ {*a*<−−<*b*}
by (*force simp*: *eventually-at in-open-segment-iff-rot dist-commute*)
then show *?thesis* **..**
qed

lemma *linepath-ball-inside-outside*:
fixes *p* :: *real* ⇒ ′*a*
assumes *jordan*: *simple-path* (*p* +++ *linepath a b*) *pathfinish p* = *a pathstart p* = *b*
assumes *x*: *x* ∈ {*a*<−−<*b*}
obtains *e* **where** *e* > *0 ball x e* ∩ *path-image p* = {}
ball x e ∩ {*y*. (*y*−*a*) • *rot* (*a*−*b*) > *0*} ⊆ *inside* (*path-image* (*p* +++ *linepath a b*)) ∧
ball x e ∩ {*y*. (*y*−*a*) • *rot* (*a*−*b*) < *0*} ⊆ *outside* (*path-image* (*p* +++ *linepath a b*))
∨
ball x e ∩ {*y*. (*y*−*a*) • *rot* (*a*−*b*) < *0*} ⊆ *inside* (*path-image* (*p* +++ *linepath*

a b)) ∧
  ball x e ∩ {*y*. (*y*−*a*) · *rot* (*a*−*b*) > *0*} ⊆ *outside* (*path-image* (*p* +++ *linepath a b*))
proof −
  let *?lp* = *p* +++ *linepath a b*
  have *a* ≠ *b* **using** *x* **by** *auto*
  have *pp*:*path p* **using** *jordan path-join pathfinish-linepath simple-path-imp-path*
    by *fastforce*
  have *path-image p* ∩ *path-image* (*linepath a b*) ⊆ {*a*,*b*}
    using *jordan simple-path-join-loop-eq*
    by (*metis* (*no-types*, *lifting*) *inf-sup-aci*(*1*) *insert-commute path-join-path-ends path-linepath simple-path-imp-path simple-path-joinE*)
  then have *x* ∉ *path-image p* **using** *x* **unfolding** *path-image-linepath*
    by (*metis DiffE Int-iff le-iff-inf open-segment-def*)
  then have ∀$_F$ *y in at x*. *y* ∉ *path-image p*
    by (*intro eventually-not-in-closed*) (*auto simp*: *closed-path-image* ‹*path p*›)
  moreover
  have ∀$_F$ *y in at x*. (*y* − *a*) · *rot* (*a* − *b*) = *0* ⟶ *y* ∈ {*a*<−−<*b*}
    by (*rule eventually-at-open-segment*[*OF x*])
  ultimately have ∀$_F$ *y in at x*. *y* ∉ *path-image p* ∧ ((*y* − *a*) · *rot* (*a* − *b*) = *0* ⟶ *y* ∈ {*a*<−−<*b*})
    by *eventually-elim auto*
  then obtain *e* **where** *e*: *e* > *0 ball x e* ∩ *path-image p* = {}
    ball x e ∩ {*y*. (*y* − *a*) · *rot* (*a* − *b*) = *0*} ⊆ {*a*<−−<*b*}
    using ‹*x* ∉ *path-image p*› *x in-open-segment-rotD*[*OF x*]
    apply (*auto simp*: *eventually-at subset-iff dist-commute dest*!:)
    by (*metis Int-iff all-not-in-conv dist-commute mem-ball*)
  have *a1*: *pathfinish ?lp* = *pathstart ?lp*
    by (*auto simp add*: *jordan*)
  have *x* ∈ *path-image ?lp*
    using *jordan*(*1*) *open-closed-segment path-image-join path-join-path-ends simple-path-imp-path x* **by** *fastforce*
  from *jordan-points-inside-outside*[*OF e*(*1*) *jordan*(*1*) *a1 this*]
  obtain *y z* **where** *y*: *y* ∈ *inside* (*path-image ?lp*) *y* ∈ *ball x e*
    and *z*: *z* ∈ *outside* (*path-image ?lp*) *z* ∈ *ball x e* **by** *blast*
  have *jordancurve*:
    inside (*path-image ?lp*) ∩ *outside*(*path-image ?lp*) = {}
    frontier (*inside* (*path-image ?lp*)) = *path-image ?lp*
    frontier (*outside* (*path-image ?lp*)) = *path-image ?lp*
    using *Jordan-inside-outside-R2*[*OF jordan*(*1*) *a1*] **by** *auto*
  define *b1* **where** *b1* = *ball x e* ∩ {*y*. (*y*−*a*) · *rot* (*a*−*b*) > *0*}
  define *b2* **where** *b2* = *ball x e* ∩ {*y*. (*y*−*a*) · *rot* (*a*−*b*) < *0*}
  define *b3* **where** *b3* = *ball x e* ∩ {*y*. (*y*−*a*) · *rot* (*a*−*b*) = *0*}
  have *path-connected b1* **unfolding** *b1-def*
   apply (*auto intro*!: *convex-imp-path-connected convex-Int simp add*:*inner-diff-left*)
    using *convex-halfspace-gt*[*of a* · *rot* (*a* − *b*) *rot*(*a*−*b*)] *inner-commute*
    by (*metis* (*no-types*, *lifting*) *Collect-cong*)
  have *path-connected b2* **unfolding** *b2-def*
   apply (*auto intro*!: *convex-imp-path-connected convex-Int simp add*:*inner-diff-left*)

```
  using convex-halfspace-lt[of rot(a−b) a · rot (a − b)] inner-commute
  by (metis (no-types, lifting) Collect-cong)
have b1 ∩ path-image(linepath a b) = {} unfolding path-image-linepath b1-def
  using closed-segment-surface[OF ‹a ≠ b›] in-segment-inner-rot2 by auto
then have b1i:b1 ∩ path-image ?lp = {}
 by (metis IntD2 b1-def disjoint-iff-not-equal e(2) inf-sup-aci(1) not-in-path-image-join)
have b2 ∩ path-image(linepath a b) = {} unfolding path-image-linepath b2-def
  using closed-segment-surface[OF ‹a ≠ b›] in-segment-inner-rot2 by auto
then have b2i:b2 ∩ path-image ?lp = {}
 by (metis IntD2 b2-def disjoint-iff-not-equal e(2) inf-sup-aci(1) not-in-path-image-join)
have bsplit: ball x e = b1 ∪ b2 ∪ b3
  unfolding b1-def b2-def b3-def
  by auto
have z ∉ b3
proof clarsimp
  assume z ∈ b3
  then have z ∈ {a<−−<b} unfolding b3-def using e by blast
  then have z ∈ path-image(linepath a b) by (auto simp add: open-segment-def)
  then have z ∈ path-image ?lp
    by (simp add: jordan(2) path-image-join)
  thus False using z
    using inside-Un-outside by fastforce
qed
then have z12: z ∈ b1 ∨ z ∈ b2 using z bsplit by blast
have y ∉ b3
proof clarsimp
  assume y ∈ b3
  then have y ∈ {a<−−<b} unfolding b3-def using e by auto
  then have y ∈ path-image(linepath a b) by (auto simp add: open-segment-def)
  then have y ∈ path-image ?lp
    by (simp add: jordan(2) path-image-join)
  thus False using y
    using inside-Un-outside by fastforce
qed
then have y ∈ b1 ∨ y ∈ b2 using y bsplit by blast
moreover {
  assume y ∈ b1
  then have b1 ∩ inside (path-image ?lp) ≠ {} using y by blast
  from path-connected-not-frontier-subset[OF ‹path-connected b1› this]
  have 1:b1 ⊆ inside (path-image ?lp) unfolding jordancurve using b1i
    by blast
  then have z ∈ b2 using jordancurve(1) z(1) z12 by blast
  then have b2 ∩ outside (path-image ?lp) ≠ {} using z by blast
  from path-connected-not-frontier-subset[OF ‹path-connected b2› this]
  have 2:b2 ⊆ outside (path-image ?lp) unfolding jordancurve using b2i
    by blast
  note conjI[OF 1 2]
}
moreover {
```

assume $y \in b2$
then have $b2 \cap$ *inside* (*path-image ?lp*) $\neq \{\}$ **using** *y* **by** *blast*
from *path-connected-not-frontier-subset*[*OF* ‹*path-connected b2*› *this*]
have *1*:$b2 \subseteq$ *inside* (*path-image ?lp*) **unfolding** *jordancurve* **using** *b2i*
by *blast*
then have $z \in b1$ **using** *jordancurve*(*1*) *z*(*1*) *z12* **by** *blast*
then have $b1 \cap$ *outside* (*path-image ?lp*) $\neq \{\}$ **using** *z* **by** *blast*
from *path-connected-not-frontier-subset*[*OF* ‹*path-connected b1*› *this*]
have *2*:$b1 \subseteq$ *outside* (*path-image ?lp*) **unfolding** *jordancurve* **using** *b1i*
by *blast*
note *conjI*[*OF 1 2*]
}
ultimately show *?thesis* **unfolding** *b1-def b2-def* **using** *that*[*OF e*(*1*−*2*)] **by** *auto*
qed

6.4 Transversal Segments

definition *transversal-segment a b* $\longleftrightarrow$
$a \neq b \wedge \{a{-}{-}b\} \subseteq X \wedge$
$(\forall z \in \{a{-}{-}b\}.\ f\ z \cdot rot\ (a{-}b) \neq 0)$

lemma *transversal-segment-reverse*:
assumes *transversal-segment x y*
shows *transversal-segment y x*
unfolding *transversal-segment-def*
by (*metis* (*no-types*, *opaque-lifting*) *add.left-neutral add-uminus-conv-diff assms closed-segment-commute inner-diff-left inner-zero-left nrm-reverse transversal-segment-def*)

lemma *transversal-segment-commute*: *transversal-segment x y* $\longleftrightarrow$ *transversal-segment y x*
using *transversal-segment-reverse* **by** *blast*

lemma *transversal-segment-neg*:
assumes *transversal-segment x y*
assumes *w*: $w \in \{x\ {-}{-}\ y\}$ **and** $f\ w \cdot rot\ (x{-}y) < 0$
shows $\forall z \in \{x{-}{-}y\}.\ f(z) \cdot rot\ (x{-}y) < 0$
proof (*rule ccontr*)
assume $\neg\ (\forall z{\in}\{x{-}{-}y\}.\ f\ z \cdot rot\ (x{-}y) < 0)$
then obtain *z* **where** *z*: $z \in \{x{-}{-}y\}$ $f\ z \cdot rot\ (x{-}y) \geq 0$ **by** *auto*
define *ff* **where** $ff = (\lambda s.\ f\ (w + s *_R (z - w)) \cdot rot\ (x{-}y))$
have *f0*:$ff\ 0 \leq 0$ **unfolding** *ff-def* **using** *assms*(*3*)
by *simp*
have *fu*:$ff\ 1 \geq 0$
by (*auto simp*: *ff-def z*)
from *assms*(*2*) **obtain** *u* **where** *u*: $0 \leq u$ $u \leq 1$ $w = (1 - u) *_R x + u *_R y$
unfolding *in-segment* **by** *blast*
have $\{x{-}{-}y\} \subseteq X$ **using** *assms*(*1*) **unfolding** *transversal-segment-def* **by** *blast*

```
  then have continuous-on {0..1} ff unfolding ff-def
    using assms(2)
    by (auto intro!:continuous-intros closed-subsegmentI z elim!: set-mp)
  from IVT'[of ff, OF f0 fu zero-le-one this]
  obtain s where s: s ≥ 0 s ≤ 1 ff s = 0 by blast
  have w + s *R (z − w) ∈ {x −− y}
    by (auto intro!: closed-subsegmentI z s w)
  with ‹ff s = 0› show False
    using s assms(1) unfolding transversal-segment-def ff-def by blast
qed

lemmas transversal-segment-sign-less = transversal-segment-neg[OF - ends-in-segment(1)]

lemma transversal-segment-pos:
  assumes transversal-segment x y
  assumes w: w ∈ {x −− y} f w · rot (x−y) > 0
  shows ∀ z ∈ {x−−y}. f(z) · rot (x−y) > 0
  using transversal-segment-neg[OF transversal-segment-reverse[OF assms(1)], of
w] w
  by (auto simp: rot-diff-commute[of x y] closed-segment-commute)

lemma transversal-segment-posD:
  assumes transversal-segment x y
    and pos: z ∈ {x −− y} f z · rot (x − y) > 0
  shows x ≠ y {x−−y} ⊆ X ⋀z. z ∈ {x−−y} ⟹ f z · rot (x−y) > 0
  using assms(1) transversal-segment-pos[OF assms]
  by (auto simp: transversal-segment-def)

lemma transversal-segment-negD:
  assumes transversal-segment x y
    and pos: z ∈ {x −− y} f z · rot (x − y) < 0
  shows x ≠ y {x−−y} ⊆ X ⋀z. z ∈ {x−−y} ⟹ f z · rot (x−y) < 0
  using assms(1) transversal-segment-neg[OF assms]
  by (auto simp: transversal-segment-def)

lemma transversal-segmentE:
  assumes transversal-segment x y
  obtains x ≠ y {x −− y} ⊆ X ⋀z. z ∈ {x−−y} ⟹ f z · rot (x − y) > 0
  | x ≠ y {x −− y} ⊆ X ⋀z. z ∈ {x−−y} ⟹ f z · rot (y − x) > 0
proof (cases f x · rot (x − y) < 0)
  case True
  from transversal-segment-negD[OF assms ends-in-segment(1) True]
  have x ≠ y {x −− y} ⊆ X ⋀z. z ∈ {x−−y} ⟹ f z · rot (y − x) > 0
    by (auto simp: rot-diff-commute[of x y])
  then show ?thesis ..
next
  case False
  then have f x · rot (x − y) > 0 using assms
   by (auto simp: transversal-segment-def algebra-split-simps not-less order.order-iff-strict)
```

```
  from transversal-segment-posD[OF assms ends-in-segment(1) this]
  show ?thesis ..
qed

lemma dist-add-vec:
  shows dist (x + s *R v) x = abs s * norm v
  by (simp add: dist-cancel-add1)

lemma transversal-segment-exists:
  assumes x ∈ X f x ≠ 0
  obtains a b where x ∈ {a<−−<b}
    transversal-segment a b
proof −

  define l where l = (λs::real. x + (s/norm(f x)) *R rot (f x))
  have norm (f x) > 0 using assms(2) using zero-less-norm-iff by blast
  then have distl: ∀ s. dist (l s) x = abs s unfolding l-def dist-add-vec
    by (auto simp add: norm-rot)
  obtain d where d:d > 0 cball x d ⊆ X
    by (meson UNIV-I assms(1) local.local-unique-solution)
 then have lb: l‘{−d..d} ⊆ cball x d using distl by (simp add: abs-le-iff dist-commute
image-subset-iff)
  from fcontx[OF assms(1)] have continuous (at x) f .
  then have c:continuous (at 0) ((λy. (f y · f x)) ∘ l) unfolding l-def
    by (auto intro!:continuous-intros simp add: assms(2))
  have ((λy. f y · f x) ∘ l) 0 > 0 using assms(2) unfolding l-def o-def by auto
  from continuous-at-imp-cball[OF c this]
  obtain r where r:r > 0  ∀ z∈cball 0 r. 0 < ((λy. f y · f x) ∘ l) z by blast
  then have rc:∀ z ∈ l‘{−r..r}. 0 < f z · f x using real-norm-def by auto
  define dr where dr = min r d
  have t1:l (−dr) ≠ l dr unfolding l-def dr-def
    by (smt ‹0 < d› ‹0 < norm (f x)› ‹0 < r› add-left-imp-eq divide-cancel-right
norm-rot norm-zero scale-cancel-right)
  have x = midpoint (l (−dr)) (l dr) unfolding midpoint-def l-def by auto
  then have xin:x ∈ {l (−dr)<−−<(l dr)} using t1 by auto

  have lsub:{l (−dr)−−l dr} ⊆ l‘{−dr..dr}
  proof safe
    fix z
    assume z ∈ {l (− dr)−−l dr}
   then obtain u where u: 0 ≤ u u ≤ 1 z = (1 − u) *R (l (−dr)) + u *R (l dr)
     unfolding in-segment by blast
   then have z = x − (1−u) *R (dr/norm(f x)) *R rot (f x) + u *R  (dr/norm(f
x)) *R rot (f x)
     unfolding l-def
     by (simp add: l-def scaleR-add-right scale-right-diff-distrib u(3))
    also have ... = x − (1 − 2 * u) *R (dr/norm(f x)) *R rot (f x)
     by (auto simp add: algebra-simps divide-simps simp flip: scaleR-add-left)
    also have ... =  x + (((2 * u − 1) * dr)/norm(f x)) *R rot (f x)
```

by (*smt add-uminus-conv-diff scaleR-scaleR scale-minus-left times-divide-eq-right*)
finally have *zeq*: $z = l\ ((2{*}u{-}1){*}dr)$ **unfolding** *l-def* **.**
have *ub*: $2{*}\ u - 1 \le 1 \wedge -1 \le\ 2{*}\ u - 1$ **using** *u* **by** *linarith*
thus $z \in l$ ' $\{-\ dr..dr\}$ **using** *zeq*
by (*smt atLeastAtMost-iff d*(*1*) *dr-def image-eqI mult.commute mult-left-le mult-minus-left r*(*1*))
qed
have *t2*: $\{l\ (-\ dr){-}{-}l\ dr\} \subseteq X$ **using** *lsub*
by (*smt atLeastAtMost-iff d*(*2*) *dist-commute distl dr-def image-subset-iff mem-cball order-trans*)
have $l\ (-\ dr) - l\ dr = -2 *_R (dr/norm(f\ x)) *_R rot\ (f\ x)$ **unfolding** *l-def*
by (*simp add*: *algebra-simps flip*: *scaleR-add-left*)
then have *req*: $rot\ (l\ (-\ dr) - l\ dr) = (2 * dr/norm(f\ x)) *_R f\ x$
by *auto* (*metis add.inverse-inverse rot-rot rot-scaleR*)
have l'$\{-dr..dr\} \subseteq l$ ' $\{-r\ ..r\}$
by (*simp add*: *dr-def image-mono*)
then have $\{l\ (-\ dr){-}{-}l\ dr\} \subseteq l$ ' $\{-r\ ..\ r\}$ **using** *lsub* **by** *auto*
then have $\forall z \in \{l\ (-\ dr){-}{-}l\ dr\}.\ 0 < f\ z \cdot f\ x$ **using** *rc* **by** *blast*
moreover have $(dr\ /\ norm\ (f\ x)) > 0$
using ‹$0 < norm\ (f\ x)$› *d*(*1*) *dr-def r*(*1*) **by** *auto*
ultimately have *t3*: $\forall z \in \{l\ (-\ dr){-}{-}l\ dr\}.\ f\ z \cdot rot\ (l\ (-\ dr){-}\ l\ dr) > 0$ **unfolding** *req*
by (*smt divide-divide-eq-right inner-scaleR-right mult-2 norm-not-less-zero scaleR-2 times-divide-eq-left times-divide-eq-right zero-less-divide-iff*)
have *transversal-segment* $(l\ (-dr))\ (l\ dr)$ **using** *t1 t2 t3* **unfolding** *transversal-segment-def* **by** *auto*
thus *?thesis* **using** *xin*
using *that* **by** *auto*
qed

Perko Section 3.7 Lemma 2 part 1.

lemma *flow-transversal-segment-finite-intersections*:
assumes *transversal-segment a b*
assumes $t \le t'$ $\{t\ ..\ t'\} \subseteq$ *existence-ivl0 x*
shows *finite* $\{s{\in}\{t..t'\}.\ flow0\ x\ s \in \{a{-}{-}b\}\}$
proof −
from *assms* **have** $a \ne b$ **by** (*simp add*: *transversal-segment-def*)
show *?thesis*
unfolding *closed-segment-surface*[*OF* ‹$a \ne b$›]
apply (*rule flow-transversal-surface-finite-intersections*[**where** *Ds*=λ-. *blinfun-inner-left* (*rot* $(b - a)$)])
by
(*use assms* **in** ‹*auto intro*!: *closed-Collect-conj closed-halfspace-component-ge closed-halfspace-component-le*
derivative-eq-intros
simp: *transversal-segment-def nrm-reverse*[*where x*=*a*] *in-closed-segment-iff-rot*›)
qed

lemma *transversal-bound-posE*:

assumes *transversal*: *transversal-segment a b*
assumes *direction*: $z \in \{a -- b\}$ $f\ z \cdot (rot\ (a - b)) > 0$
obtains $d\ B$ **where** $d > 0$ $0 < B$
$\bigwedge x\ y.\ x \in \{a -- b\} \Longrightarrow dist\ x\ y \leq d \Longrightarrow f\ y \cdot (rot\ (a - b)) \geq B$
proof −
let $?a = (\lambda y.\ (f\ y) \cdot (rot\ (a - b)))$
from *transversal-segment-posD*[*OF transversal direction*]
have *seg*: $a \neq b$ $\{a--b\} \subseteq X$ $z \in \{a--b\} \Longrightarrow 0 < f\ z \cdot rot\ (a - b)$ **for** z
by *auto*
{
fix x
assume $x \in \{a--b\}$
then have $x \in X$ $f\ x \neq 0$ $a \neq b$ **using** *transversal* **by** (*auto simp*: *transversal-segment-def*)
then have $?a\ -x\rightarrow\ ?a\ x$
by (*auto intro*!: *tendsto-eq-intros*)
moreover have $?a\ x > 0$
using *seg* ‹$x \in \{a -- b\}$› ‹$f\ x \neq 0$›
by (*auto simp*: *simp del*: *divide-const-simps*
intro!: *divide-pos-pos mult-pos-pos*)
ultimately have $\forall_F\ x\ in\ at\ x.\ ?a\ x > 0$
by (*rule order-tendstoD*)
moreover have $\forall_F\ x\ in\ at\ x.\ x \in X$
by (*rule topological-tendstoD*[*OF tendsto-ident-at open-dom* ‹$x \in X$›])
moreover have $\forall_F\ x\ in\ at\ x.\ f\ x \neq 0$
by (*rule tendsto-imp-eventually-ne tendsto-intros* ‹$x \in X$› ‹$f\ x \neq 0$›)+
ultimately have $\forall_F\ x\ in\ at\ x.\ ?a\ x>0 \wedge x \in X \wedge f\ x \neq 0$ **by** *eventually-elim auto*
then obtain d **where** d: $0 < d$ $\bigwedge y.\ y \in cball\ x\ d \Longrightarrow ?a\ y > 0 \wedge y \in X \wedge f\ y \neq 0$
using ‹$?a\ x > 0$› ‹$x \in X$›
by (*force simp*: *eventually-at-le dist-commute*)

have *continuous-on* (*cball x d*) $?a$
using d ‹$a \neq b$›
by (*auto intro*!: *continuous-intros*)
from *compact-continuous-image*[*OF this compact-cball*]
have *compact* ($?a$ ‘ *cball x d*) **.**
from *compact-attains-inf*[*OF this*] **obtain** s **where** $s \in cball\ x\ d$ $\forall x{\in}cball\ x\ d.\ ?a\ x \geq ?a\ s$
using ‹$d > 0$›
by *auto*
then have $\exists d{>}0.\ \exists b{>}0.\ \forall x \in cball\ x\ d.\ ?a\ x \geq b$
using d
by (*force simp*: *intro*: *exI*[**where** $x{=}?a\ s$])
} **then obtain** *dx Bx* **where** *dB*:
$\bigwedge x\ y.\ x \in \{a -- b\} \Longrightarrow y{\in}cball\ x\ (dx\ x) \Longrightarrow ?a\ y \geq Bx\ x$
$\bigwedge x.\ x \in \{a -- b\} \Longrightarrow Bx\ x > 0$
$\bigwedge x.\ x \in \{a -- b\} \Longrightarrow dx\ x > 0$

by *metis*
define d' **where** $d' = (\lambda x.\ dx\ x\ /\ 2)$
have d':
$\bigwedge x.\ x \in \{a\ {-}{-}\ b\} \Longrightarrow \forall\, y{\in}cball\ x\ (d'\ x).\ ?a\ y \geq Bx\ x$
$\bigwedge x.\ x \in \{a\ {-}{-}\ b\} \Longrightarrow d'\ x > 0$
using $dB(1,3)$ **by** $(force\ simp{:}\ d'\text{-}def)+$
have $d'B$: $\bigwedge x.\ x \in \{a\ {-}{-}\ b\} \Longrightarrow \forall\, y{\in}cball\ x\ (d'\ x).\ ?a\ y \geq Bx\ x$
using d' **by** *auto*
have $\{a{-}{-}b\} \subseteq \bigcup((\lambda x.\ ball\ x\ (d'\ x))\ `\ \{a\ {-}{-}\ b\})$
using $d'(2)$ **by** *auto*
from *compactE-image*[*OF compact-segment - this*]
obtain X **where** X: $X \subseteq \{a{-}{-}b\}$
and [*simp*]: *finite* X
and *cover*: $\{a{-}{-}b\} \subseteq (\bigcup x{\in}X.\ ball\ x\ (d'\ x))$
by *auto*
have [*simp*]: $X \neq \{\}$ **using** *X cover* **by** *auto*
define d **where** $d = Min\ (d'\ `\ X)$
define B **where** $B = Min\ (Bx\ `\ X)$
have $d > 0$
using $X\ d'$
by $(auto\ simp{:}\ d\text{-}def\ d'\text{-}def)$
moreover have $B > 0$
using *X dB*
by (*auto simp*: *B-def simp del*: *divide-const-simps*)
moreover have $B \leq\ ?a\ y$ **if** $x \in \{a\ {-}{-}\ b\}$ $dist\ x\ y \leq d$ **for** $x\ y$
proof −
from ‹$x \in \{a\ {-}{-}\ b\}$› **obtain** xc **where** xc: $xc \in X$ $x \in ball\ xc\ (d'\ xc)$
using *cover* **by** *auto*
have $?a\ y \geq Bx\ xc$
proof (*rule dB*)
show $xc \in \{a\ {-}{-}\ b\}$ **using** xc ‹$X \subseteq$ -› **by** *auto*
have $dist\ xc\ y \leq dist\ xc\ x + dist\ x\ y$ **by** *norm*
also have $dist\ xc\ x \leq d'\ xc$ **using** xc **by** *auto*
also note ‹$dist\ x\ y \leq d$›
also have $d \leq d'\ xc$
using xc
by (*auto simp*: *d-def*)
also have $d'\ xc + d'\ xc = dx\ xc$ **by** $(simp\ add{:}\ d'\text{-}def)$
finally show $y \in cball\ xc\ (dx\ xc)$ **by** *simp*
qed
also have $B \leq Bx\ xc$
using xc
unfolding *B-def*
by (*auto simp*: *B-def*)
finally (*xtrans*) **show** *?thesis* **.**
qed
ultimately show *?thesis* **..**
qed

lemma *transversal-bound-negE*:
assumes *transversal*: *transversal-segment a b*
assumes *direction*: $z \in \{a -- b\}$ $f\ z \cdot (rot\ (a - b)) < 0$
obtains *d B* **where** $d > 0$ $0 < B$
$\bigwedge x\ y.\ x \in \{a -- b\} \Longrightarrow dist\ x\ y \le d \Longrightarrow f\ y \cdot (rot\ (b - a)) \ge B$
proof −
from *direction* **have** $z \in \{b -- a\}$ $f\ z \cdot (rot\ (b - a)) > 0$
by (*auto simp*: *closed-segment-commute rot-diff-commute*[*of b a*])
from *transversal-bound-posE*[*OF transversal-segment-reverse*[*OF transversal*] *this*]
obtain *d B* **where** $d > 0$ $0 < B$
$\bigwedge x\ y.\ x \in \{a -- b\} \Longrightarrow dist\ x\ y \le d \Longrightarrow f\ y \cdot (rot\ (b - a)) \ge B$
by (*auto simp*: *closed-segment-commute*)
then show *?thesis* **..**
qed

lemma *leaves-transversal-segmentE*:
assumes *transversal*: *transversal-segment a b*
obtains *T n* **where** $T > 0$ $n = a - b \lor n = b - a$
$\bigwedge x.\ x \in \{a -- b\} \Longrightarrow \{-T..T\} \subseteq existence\text{-}ivl0\ x$
$\bigwedge x\ s.\ x \in \{a -- b\} \Longrightarrow 0 < s \Longrightarrow s \le T \Longrightarrow$
$(flow0\ x\ s - x) \cdot rot\ n > 0$
$\bigwedge x\ s.\ x \in \{a -- b\} \Longrightarrow -T \le s \Longrightarrow s < 0 \Longrightarrow$
$(flow0\ x\ s - x) \cdot rot\ n < 0$
proof −
from *transversal-segmentE*[*OF assms*(*1*)] **obtain** *n*
where *n*: $n = (a - b) \lor n = (b - a)$
and *seg*: $a \ne b$ $\{a -- b\} \subseteq X$ $\bigwedge z.\ z \in \{a--b\} \Longrightarrow f\ z \cdot rot\ n > 0$
by *metis*
from *open-existence-ivl-on-compact*[*OF* ‹$\{a -- b\} \subseteq X$›]
obtain *t* **where** $0 < t$ **and** *t*: $x \in \{a--b\} \Longrightarrow \{- t..t\} \subseteq existence\text{-}ivl0\ x$ **for** *x*
by *auto*
from *n* **obtain** *d B* **where** *B*: $0 < d$ $0 < B$ $(\bigwedge x\ y.\ x \in \{a--b\} \Longrightarrow dist\ x\ y \le d \Longrightarrow B \le f\ y \cdot rot\ n)$
proof
assume *n-def*: $n = a - b$
with *seg* **have** *pos*: $0 < f\ a \cdot rot\ (a - b)$
by *auto*
from *transversal-bound-posE*[*OF transversal ends-in-segment*(*1*) *pos*, *folded n-def*]
show *?thesis* **using** *that* **by** *blast*
next
assume *n-def*: $n = b - a$
with *seg* **have** *pos*: $0 > f\ a \cdot rot\ (a - b)$
by (*auto simp*: *rot-diff-commute*[*of a b*])
from *transversal-bound-negE*[*OF transversal ends-in-segment*(*1*) *this*, *folded n-def*]
show *?thesis* **using** *that* **by** *blast*
qed
define *S* **where** $S = \bigcup ((\lambda x.\ ball\ x\ d)\ \text{`}\ \{a -- b\})$

have S: $x \in S \Longrightarrow B \leq f\ x \cdot rot\ n$ **for** x
  by ($auto\ simp$: S-$def\ intro!$: B)
have $open\ S$ **by** ($auto\ simp$: S-def)
have $\{a\ {-}{-}\ b\} \subseteq S$
  by ($auto\ simp$: S-def ‹$0 < d$›)
have $\forall_F\ (t, x)\ in\ at\ (0, x).\ flow0\ x\ t \in S$ **if** $x \in \{a\ {-}{-}\ b\}$ **for** x
  unfolding $split$-$beta'$
  apply ($rule\ topological$-$tendstoD\ tendsto$-$intros$)+
  using set-mp[OF ‹$\{a\ {-}{-}\ b\} \subseteq X$› $that$] ‹$0 < d$› $that$ ‹$open\ S$› ‹$\{a\ {-}{-}\ b\} \subseteq S$›
  by $force$+
then obtain d' **where** d':
  $\bigwedge x.\ x \in \{a{-}{-}b\} \Longrightarrow d'\ x > 0$
  $\bigwedge x\ y\ s.\ x \in \{a{-}{-}b\} \Longrightarrow (s = 0 \longrightarrow y \neq x) \Longrightarrow dist\ (s, y)\ (0, x) < d'\ x \Longrightarrow flow0\ y\ s \in S$
  by ($auto\ simp$: $eventually$-at) $metis$
define $d2$ **where** $d2\ x = d'\ x\ /\ 2$ **for** x
have $d2$: $\bigwedge x.\ x \in \{a{-}{-}b\} \Longrightarrow d2\ x > 0$ **using** d' **by** ($auto\ simp$: $d2$-def)
have C: $\{a{-}{-}b\} \subseteq \bigcup((\lambda x.\ ball\ x\ (d2\ x))\ `\ \{a\ {-}{-}\ b\})$
  using $d2$ **by** $auto$
from $compactE$-$image$[$OF\ compact$-$segment$ - C]
obtain C' **where** $C' \subseteq \{a{-}{-}b\}$ **and** [$simp$]: $finite\ C'$
  and C'-$cover$: $\{a{-}{-}b\} \subseteq (\bigcup c{\in}C'.\ ball\ c\ (d2\ c))$ **by** $auto$

define T **where** $T = Min\ (insert\ t\ (d2\ `\ C'))$

have $T > 0$
  using ‹$0 < t$› $d2$ ‹$C' \subseteq$ -›
  by ($auto\ simp$: T-def)
moreover
note n
moreover
have T-ex: $\{-T..T\} \subseteq existence$-$ivl0\ x$ **if** $x \in \{a{-}{-}b\}$ **for** x
  by ($rule\ order$-$trans$[OF - t[$OF\ that$]]) ($auto\ simp$: T-def)
moreover
have B-le: $B \leq f\ (flow0\ x\ \xi) \cdot rot\ n$
  if $x \in \{a\ {-}{-}\ b\}$
    and c': $c' \in C'\ x \in ball\ c'\ (d2\ c')$
    and $\xi \neq 0$ **and** ξ-le: $|\xi| < d2\ c'$
  for $x\ c'\ \xi$
proof $-$
  have $c' \in \{a\ {-}{-}\ b\}$ **using** c' ‹$C' \subseteq$ -› **by** $auto$
  moreover have $\xi = 0 \longrightarrow x \neq c'$ **using** ‹$\xi \neq 0$› **by** $simp$
  moreover have $dist\ (\xi, x)\ (0, c') < d'\ c'$
  proof $-$
    have $dist\ (\xi, x)\ (0, c') \leq dist\ (\xi, x)\ (\xi, c') + dist\ (\xi, c')\ (0, c')$
      by $norm$
    also have $dist\ (\xi, x)\ (\xi, c') < d2\ c'$
      using c'

by (*simp add*: *dist-prod-def dist-commute*)
also
have $T \le d2\ c'$ **using** c'
by (*auto simp*: *T-def*)
then have *dist* $(\xi,\ c')\ (0,\ c') < d2\ c'$
using *ξ-le*
by (*simp add*: *dist-prod-def*)
also have $d2\ c' + d2\ c' = d'\ c'$ **by** (*simp add*: *d2-def*)
finally show *?thesis* **by** *simp*
qed
ultimately have *flow0* $x\ \xi \in S$
by (*rule* d')
then show *?thesis*
by (*rule S*)
qed
let $?g = (\lambda x\ t.\ (flow0\ x\ t - x) \cdot rot\ n)$
have *cont*: *continuous-on* $\{-T\ ..\ T\}$ $(?g\ x)$
if $x \in \{a{-}{-}b\}$ **for** x
using *T-ex that*
by (*force intro*!: *continuous-intros*)
have *deriv*: $-T \le s' \Longrightarrow s' \le T \Longrightarrow ((?g\ x)$ *has-derivative*
$(\lambda t.\ t * (f\ (flow0\ x\ s') \cdot rot\ n)))\ (at\ s')$
if $x \in \{a{-}{-}b\}$ **for** $x\ s'$
using *T-ex that*
by (*force intro*!: *derivative-eq-intros simp*: *flowderiv-def blinfun.bilinear-simps*)

have $(flow0\ x\ s - x) \cdot rot\ n > 0$ **if** $x \in \{a\ {-}{-}\ b\}$ $0 < s$ $s \le T$ **for** $x\ s$
proof (*rule ccontr*, *unfold not-less*)
have [*simp*]: $x \in X$ **using** *that* ‹$\{a\ {-}{-}\ b\} \subseteq X$› **by** *auto*
assume *H*: $(flow0\ x\ s - x) \cdot rot\ n \le 0$
have *cont*: *continuous-on* $\{0\ ..\ s\}$ $(?g\ x)$
using *cont* **by** (*rule continuous-on-subset*) (*use that* **in** *auto*)
from *mvt*[*OF* ‹$0 < s$› *cont deriv*] *that*
obtain ξ **where** ξ: $0 < \xi$ $\xi < s$ $(flow0\ x\ s - x) \cdot rot\ n = s * (f\ (flow0\ x\ \xi) \cdot rot\ n)$
by (*auto intro*: *continuous-on-subset*)
note ‹$0 < B$›
also
from *C′-cover that* **obtain** c' **where** c': $c' \in C'$ $x \in ball\ c'\ (d2\ c')$ **by** *auto*
have $B \le f\ (flow0\ x\ \xi) \cdot rot\ n$
proof (*rule B-le*[*OF that*(*1*) c'])
show $\xi \ne 0$ **using** ‹$0 < \xi$› **by** *simp*
have $T \le d2\ c'$ **using** c'
by (*auto simp*: *T-def*)
then show $|\xi| < d2\ c'$
using ‹$0 < \xi$› ‹$\xi < s$› ‹$s \le T$›
by (*simp add*: *dist-prod-def*)
qed
also from ξ *H* **have** $\ldots \le 0$

by (*auto simp add*: *algebra-split-simps not-less split*: *if-splits*)
finally show *False* **by** *simp*
qed
moreover
have (*flow0 x s* − *x*) · *rot n* < *0* **if** $x \in \{a -- b\}$ $-T \le s$ $s < 0$ **for** *x s*
proof (*rule ccontr*, *unfold not-less*)
have [*simp*]: $x \in X$ **using** *that* ‹$\{a -- b\} \subseteq X$› **by** *auto*
assume *H*: (*flow0 x s* − *x*) · *rot n* ≥ *0*
have *cont*: *continuous-on* {*s* .. *0*} (*?g x*)
using *cont* **by** (*rule continuous-on-subset*) (*use that* **in** *auto*)
from *mvt*[*OF* ‹*s* < *0*› *cont deriv*] *that*
obtain ξ **where** ξ: *s* < ξ ξ < *0* (*flow0 x s* − *x*) · *rot n* = *s* ∗ (*f* (*flow0 x* ξ) · *rot n*)
by *auto*
note ‹*0* < *B*›
also
from *C′-cover that* **obtain** *c′* **where** *c′*: $c' \in C'$ $x \in$ *ball c′* (*d2 c′*) **by** *auto*
have *B* ≤ (*f* (*flow0 x* ξ) · *rot n*)
proof (*rule B-le*[*OF that*(*1*) *c′*])
show ξ ≠ *0* **using** ‹*0* > ξ› **by** *simp*
have *T* ≤ *d2 c′* **using** *c′*
by (*auto simp*: *T-def*)
then show |ξ| < *d2 c′*
using ‹*0* > ξ› ‹ξ > *s*› ‹*s* ≥ − *T*›
by (*simp add*: *dist-prod-def*)
qed
also from ξ *H* **have** ... ≤ *0*
by (*simp add*: *algebra-split-simps*)
finally show *False* **by** *simp*
qed
ultimately show *?thesis* **..**
qed

lemma *inner-rot-pos-move-base*: (*x* − *a*) · *rot* (*a* − *b*) > *0*
if (*x* − *y*) · *rot* (*a* − *b*) > *0* $y \in \{a -- b\}$
by (*smt in-segment-inner-rot inner-diff-left inner-minus-right minus-diff-eq rot-rot that*)

lemma *inner-rot-neg-move-base*: (*x* − *a*) · *rot* (*a* − *b*) < *0*
if (*x* − *y*) · *rot* (*a* − *b*) < *0* $y \in \{a -- b\}$
by (*smt in-segment-inner-rot inner-diff-left inner-minus-right minus-diff-eq rot-rot that*)

lemma *inner-pos-move-base*: (*x* − *a*) · *n* > *0*
if (*a* − *b*) · *n* = *0* (*x* − *y*) · *n* > *0* $y \in \{a -- b\}$
proof −
from *that*(*3*) **obtain** *u* **where** *y-def*: $y = (1 - u) *_R a + u *_R b$ **and** *u*: $0 \le u$ $u \le 1$

by (*auto simp*: *in-segment*)
have $(x - a) \cdot n = (x - y) \cdot n - u * ((a - b) \cdot n)$
by (*simp add*: *algebra-simps y-def*)
also have $\ldots = (x - y) \cdot n$
by (*simp add*: *that*)
also note $\langle \ldots > 0 \rangle$
finally show *?thesis* **.**
qed

lemma *inner-neg-move-base*: $(x - a) \cdot n < 0$
if $(a - b) \cdot n = 0$ $(x - y) \cdot n < 0$ $y \in \{a -- b\}$
proof –
from *that*(*3*) **obtain** *u* **where** *y-def*: $y = (1 - u) *_R a + u *_R b$ **and** *u*: $0 \leq u$ $u \leq 1$
by (*auto simp*: *in-segment*)
have $(x - a) \cdot n = (x - y) \cdot n - u * ((a - b) \cdot n)$
by (*simp add*: *algebra-simps y-def*)
also have $\ldots = (x - y) \cdot n$
by (*simp add*: *that*)
also note $\langle \ldots < 0 \rangle$
finally show *?thesis* **.**
qed

lemma *rot-same-dir*:
assumes $x1 \in \{a{<}{-}{-}{<}b\}$
assumes $x2 \in \{x1{<}{-}{-}{<}b\}$
shows $(y \cdot rot\ (a-b) > 0) = (y \cdot rot(x1-x2) > 0)$ $(y \cdot rot\ (a-b) < 0) = (y \cdot rot(x1-x2) < 0)$
using *oriented-subsegment-scale*[*OF assms*]
apply (*smt inner-scaleR-right nrm-reverse rot-scaleR zero-less-mult-iff*)
by (*smt* $\langle \bigwedge thesis.\ (\bigwedge e.\ [\![0 < e;\ b - a = e *_R (x2 - x1)]\!] \Longrightarrow thesis) \Longrightarrow thesis \rangle$ *inner-minus-right inner-scaleR-right rot-diff-commute rot-scaleR zero-less-mult-iff*)

6.5 Monotone Step Lemma

lemma *flow0-transversal-segment-monotone-step*:
assumes *transversal-segment a b*
assumes $t1 \leq t2$ $\{t1..t2\} \subseteq$ *existence-ivl0 x*
assumes *x1*: *flow0 x t1* $\in \{a{<}{-}{-}{<}b\}$
assumes *x2*: *flow0 x t2* $\in \{$*flow0 x t1*${<}{-}{-}{<}b\}$
assumes $\bigwedge t.\ t \in \{t1{<}..{<}t2\} \Longrightarrow$ *flow0 x t* $\notin \{a{<}{-}{-}{<}b\}$
assumes $t > t2$ $t \in$ *existence-ivl0 x*
shows *flow0 x t* $\notin \{a{<}{-}{-}{<}$*flow0 x t2*$\}$
proof –
note *exist* = $\langle \{t1..t2\} \subseteq$ *existence-ivl0 x*$\rangle$
note *t1t2* = $\langle \bigwedge t.\ t \in \{t1{<}..{<}t2\} \Longrightarrow$ *flow0 x t* $\notin \{a{<}{-}{-}{<}b\}\rangle$

have *x1neqx2*: *flow0 x t1* $\neq$ *flow0 x t2*
using *open-segment-def x2* **by** *force*

then have *t1neqt2*: $t1 \neq t2$ **by** *auto*

have [*simp*]: $a \neq b$ **and** ‹$\{a -- b\} \subseteq X$› **using** ‹*transversal-segment a b*›
by (*auto simp*: *transversal-segment-def*)

from *x1* **obtain** *i1* **where** *i1*: $flow0\ x\ t1 = line\ a\ b\ i1\ 0 < i1\ i1 < 1$
by (*auto simp*: *in-open-segment-iff-line*)
from *x2* **obtain** *i2* **where** *i2*: $flow0\ x\ t2 = line\ a\ b\ i2\ 0 < i1\ i1 < i2$
by (*auto simp*: *i1 line-open-segment-iff*)

have $\{a <--< flow0\ x\ t1\} \subseteq \{a<--<b\}$
by (*simp add*: *open-closed-segment subset-open-segment x1*)
have *t12sub*: $\{flow0\ x\ t1--flow0\ x\ t2\} \subseteq \{a<--<b\}$
by (*metis ends-in-segment*(*2*) *open-closed-segment subset-co-segment subset-eq subset-open-segment x1 x2*)
have *subr*: $\{flow0\ x\ t1<--<flow0\ x\ t2\} \subseteq \{flow0\ x\ t1\ <--<b\}$
by (*simp add*: *open-closed-segment subset-open-segment x2*)
have $flow0\ x\ t1 \in \{a\ <--<flow0\ x\ t2\}$ **using** *x1 x2*
by (*rule open-segment-subsegment*)
then have *subl*: $\{flow0\ x\ t1<--<flow0\ x\ t2\} \subseteq \{a\ <--<\ flow0\ x\ t2\}$ **using** *x1 x2*
by (*simp add*: *open-closed-segment subset-open-segment x2*)
then have *subl2*: $\{flow0\ x\ t1--<flow0\ x\ t2\} \subseteq \{a\ <--<\ flow0\ x\ t2\}$ **using** *x1 x2*
by (*smt DiffE DiffI* ‹$flow0\ x\ t1 \in \{a<--<flow0\ x\ t2\}$› *half-open-segment-def insert-iff open-segment-def subset-eq*)

have *sub1b*: $\{flow0\ x\ t1--b\} \subseteq \{a--b\}$
by (*simp add*: *open-closed-segment subset-closed-segment x1*)
have *suba2*: $\{a--flow0\ x\ t2\} \subseteq \{a -- b\}$
using *open-closed-segment subset-closed-segment t12sub* **by** *blast*
then have *suba2o*: $\{a<--<flow0\ x\ t2\} \subseteq \{a -- b\}$
using *open-closed-segment subset-closed-segment t12sub* **by** *blast*
have *x2-notmem*: $flow0\ x\ t2 \notin \{a--flow0\ x\ t1\}$
using *i1 i2*
by (*auto simp*: *closed-segment-line-iff*)
have *suba12*: $\{a--flow0\ x\ t1\} \subseteq \{a--flow0\ x\ t2\}$
by (*simp add*: ‹$flow0\ x\ t1 \in \{a<--<flow0\ x\ t2\}$› *open-closed-segment subset-closed-segment*)
then have *suba12-open*: $\{a<--<flow0\ x\ t1\} \subseteq \{a<--<flow0\ x\ t2\}$
using *x2-notmem*
by (*auto simp*: *open-segment-def*)
have $flow0\ x\ t2 \in \{a--b\}$
using *suba2* **by** *auto*

have *intereq*: $\bigwedge t.\ t1 \leq t \Longrightarrow t \leq t2 \Longrightarrow flow0\ x\ t \in \{a<--<b\} \Longrightarrow\ t = t1 \lor t = t2$
proof (*rule ccontr*)

fix *t*
assume *t*: *t1* ≤ *t* *t* ≤ *t2* *flow0 x t* ∈ {*a*<−−<*b*} ¬(*t*= *t1* ∨ *t* = *t2*)
then have *t* ∈ {*t1*<..<*t2*} **by** *auto*
then have *flow0 x t* ∉ {*a*<−−<*b*} **using** *t1t2* **by** *blast*
thus *False* **using** *t* **by** *auto*
qed
then have *intereqt12*: ⋀*t*. *t1* ≤ *t* ⟹ *t* ≤ *t2* ⟹ *flow0 x t* ∈ {*flow0 x t1*−−*flow0 x t2*} ⟹ *t* = *t1* ∨ *t* = *t2*
using *t12sub* **by** *blast*

define *J1* **where** *J1* = *flow-to-path x t1 t2*
define *J2* **where** *J2* = *linepath* (*flow0 x t2*) (*flow0 x t1*)
define *J* **where** *J* = *J1* +++ *J2*

have *pathfinish J* = *pathstart J* **unfolding** *J-def J1-def J2-def*
by (*auto simp add*: *pathstart-compose pathfinish-compose*)
have *piJ*: *path-image J* = *path-image J1* ∪ *path-image J2*
unfolding *J-def J1-def J2-def*
apply (*rule path-image-join*)
by *auto*
have *flow0 x t1* ∈ *flow0 x* ‘ {*t1*..*t2*} ∧ *flow0 x t2* ∈ *flow0 x* ‘ {*t1*..*t2*}
using *atLeastAtMost-iff* ‹*t1* ≤ *t2*› **by** *blast*
then have *piD*: *path-image J* = *path-image J1* ∪ {*flow0 x t1* <−−<*flow0 x t2*}
unfolding *piJ J1-def J2-def path-image-flow-to-path*[*OF* ‹*t1* ≤ *t2*›]
path-image-linepath open-segment-def
by (*smt Diff-idemp Diff-insert2 Un-Diff-cancel closed-segment-commute mk-disjoint-insert*)
have ∀ *s*∈{*t1*<..<*t2*}. *flow0 x s* ≠ *flow0 x t1*
using *x1 t1t2* **by** *fastforce*
from *flow-to-path-arc*[*OF* ‹*t1* ≤ *t2*› *exist this x1neqx2*]
have *arc J1* **using** *J1-def assms flow-to-path-arc* **by** *auto*
then have *simple-path J* **unfolding** *J-def*
using ‹*arc J1*› *J1-def J2-def assms x1neqx2 t1neqt2* **apply** (*auto intro*!:*simple-path-join-loop*)
using *intereqt12 closed-segment-commute* **by** *blast*

from *Jordan-inside-outside-R2*[*OF this* ‹*pathfinish J* = *pathstart J*›]
obtain *inner outer* **where** *inner-def*: *inner* = *inside* (*path-image J*)
and *outer-def*: *outer* = *outside* (*path-image J*)
and *io*:
inner ≠ {} *open inner connected inner*
outer ≠ {} *open outer connected outer*
bounded inner ¬ *bounded outer inner* ∩ *outer* = {}
inner ∪ *outer* = − *path-image J*
frontier inner = *path-image J*
frontier outer = *path-image J* **by** *metis*
from *io* **have** *io2*: *outer* ∩ *inner* = {} *outer* ∪ *inner* = − *path-image J* **by** *auto*

have *swap-side*: ⋀*y t*. *y* ∈ *side2* ⟹
0 ≤ *t* ⟹ *t* ∈ *existence-ivl0 y* ⟹

```
  flow0 y t ∈ closure side1 ⟹
  ∃ T. 0 < T ∧ T ≤ t ∧ (∀ s ∈{0..<T}. flow0 y s ∈ side2) ∧
      flow0 y T ∈ {flow0 x t1−−<flow0 x t2}
  if side1 ∩ side2 = {}
    open side2
    frontier side1 = path-image J
    frontier side2 = path-image J
    side1 ∪ side2 = − path-image J
  for side1 side2
proof −
  fix y t
  assume yt: y ∈ side2 0 ≤ t t ∈ existence-ivl0 y
    flow0 y t ∈ closure side1
  define fp where fp = flow-to-path y 0 t
  have ex:{0..t} ⊆ existence-ivl0 y
    using ivl-subset-existence-ivl yt(3) by blast
  then have y0:flow0 y 0 = y
    using mem-existence-ivl-iv-defined(2) yt(3) by auto
  then have tpos: t > 0 using yt(2)  ‹side1 ∩ side2 = {}›
    using yt(1) yt(4)
    by (metis closure-iff-nhds-not-empty less-eq-real-def order-refl that(2))
  from flow-to-path-path[OF yt(2) ex]
  have a1: path fp unfolding fp-def .
  have y ∈ closure side2 using yt(1)
    by (simp add: assms closure-def)
 then have a2: pathstart fp ∈ closure side2 unfolding fp-def using y0 by auto
  have a3:pathfinish fp ∉ side2 using yt(4) ‹side1 ∩ side2 = {}›
    unfolding fp-def apply auto
    using closure-iff-nhds-not-empty that(2) by blast
  from subpath-to-frontier-strong[OF a1 a3]
  obtain u where u:0 ≤ u u ≤ 1
    fp u ∉ interior side2
    u = 0 ∨
    (∀ x. 0 ≤ x ∧ x < 1 ⟶
      subpath 0 u fp x ∈ interior side2) ∧ fp u ∈ closure side2 by blast
  have p1:path-image (subpath 0 u fp) =  flow0 y ‘ {0 ..  u∗t}
    unfolding fp-def subpath0-flow-to-path using path-image-flow-to-path
    by (simp add: u(1) yt(2))
  have p2:fp u = flow0 y (u∗t) unfolding fp-def flow-to-path-unfold by simp
  have inout:interior side2 = side2 using ‹open side2›
    by (simp add: interior-eq)
  then have iemp: side2 ∩ path-image J = {}
    using ‹frontier side2 = path-image J›
    by (metis frontier-disjoint-eq inf-sup-aci(1) interior-eq)
  have u ≠ 0 using inout u(3) y0 p2 yt(1) by force
  then have c1:u ∗ t > 0 using tpos u y0  ‹side1 ∩ side2 = {}›
    using frontier-disjoint-eq io(5) yt(1) zero-less-mult-iff by fastforce
  have uim:fp u ∈ path-image J using u ‹u ≠ 0›
    using ‹frontier side2 = path-image J›
```

by (*metis ComplI IntI closure-subset frontier-closures inout subsetD*)
have *c2*:$u * t \leq t$ **using** $u(1-2)$ *tpos* **by** *auto*
have(*flow-to-path y 0* $(u * t)$ ' $\{0..<1\} \subseteq$ *side2*)
using ‹$u \neq 0$› *u inout* **unfolding** *fp-def subpath0-flow-to-path* **by** *auto*
then have *c3*:$\forall s \in \{0..<u*t\}$. *flow0 y s* $\in$ *side2* **by** *auto*
have *c4*: *flow0 y* $(u*t) \in$ *path-image J*
using *uim path-image-join-subset*
by (*simp add*: *p2*)
have *flow0 y* $(u*t) \notin$ *path-image J1* $\vee$ *flow0 y* $(u*t) =$ *flow0 x t1*
proof *clarsimp*
assume *flow0 y* $(u*t) \in$ *path-image J1*
then obtain *s* **where** *s*: $t1 \leq s$ $s \leq t2$ *flow0 x s* = *flow0 y* $(u*t)$
using *J1-def* ‹$t1 \leq t2$› **by** *auto*
have $s = t1$
proof (*rule ccontr*)
assume $s \neq t1$
then have *st1*:$s > t1$ **using** $s(1)$ **by** *linarith*
define *sc* **where** $sc = min\ (s-t1)\ (u*t)$
have *scd*: $s-sc \in \{t1..t2\}$ **unfolding** *sc-def*
using *c1* $s(1)$ $s(2)$ **by** *auto*
then have $*$:*flow0 x* $(s-sc) \in$ *path-image J1* **unfolding** *J1-def path-image-flow-to-path*[*OF* ‹$t1 \leq t2$›]
by *blast*
have *flow0 x* $(s-sc)$ = *flow0* (*flow0 x s*) $(-sc)$
by (*smt exist atLeastAtMost-iff existence-ivl-trans′ flow-trans* $s(1)$ $s(2)$ *scd subsetD*)
then have $**$:*flow0* (*flow0 y* $(u*t)$) $(-sc)$ $\in$ *path-image J1*
using $s(3)$ $*$ **by** *auto*
have *b*:$u*t - sc \in \{0..<u*t\}$ **unfolding** *sc-def* **by** (*simp add*: *st1 c1* $s(1)$)
then have $u*t - sc \in$ *existence-ivl0 y*
using *c2 ex* **by** *auto*
then have *flow0 y* $(u*t - sc) \in$ *path-image J1* **using** $**$
by (*smt atLeastAtMost-iff diff-existence-ivl-trans ex flow-trans mult-left-le-one-le mult-nonneg-nonneg subset-eq* $u(1)$ $u(2)$ $yt(2)$)
thus *False* **using** *b c3 iemp piJ* **by** *blast*
qed
thus *flow0 y* $(u * t)$ = *flow0 x t1* **using** *s* **by** *simp*
qed
thus $\exists\, T>0.\ T \leq t \wedge (\forall s\in\{0..<T\}.$ *flow0 y s* $\in$ *side2*$) \wedge$
flow0 y T $\in$ {*flow0 x t1*$--<$*flow0 x t2*}
using *c1 c2 c3 c4* **unfolding** *piD*
by (*metis DiffE UnE ends-in-segment*(1) *half-open-segment-closed-segmentI insertCI open-segment-def x1neqx2*)
qed
have *outside-in*: $\bigwedge y\ t.\ y \in$ *outer* $\Longrightarrow$
$0 \leq t \Longrightarrow t \in$ *existence-ivl0 y* $\Longrightarrow$
flow0 y t $\in$ *closure inner* $\Longrightarrow$
$\exists\, T.\ 0 < T \wedge T \leq t \wedge (\forall s \in\{0..<T\}.$ *flow0 y s* $\in$ *outer*$) \wedge$
flow0 y T $\in$ {*flow0 x t1*$--<$*flow0 x t2*}

by (*rule swap-side*; (*rule io* | *assumption*))
have *inside-out*: $\bigwedge y\ t.\ y \in inner \Longrightarrow$
$0 \leq t \Longrightarrow t \in existence\text{-}ivl0\ y \Longrightarrow$
$flow0\ y\ t \in closure\ outer \Longrightarrow$
$\exists T.\ 0 < T \wedge T \leq t \wedge (\forall s \in \{0..<T\}.\ flow0\ y\ s \in inner) \wedge$
$flow0\ y\ T \in \{flow0\ x\ t1 {-}{-}{<} flow0\ x\ t2\}$
by (*rule swap-side*; (*rule io2 io* | *assumption*))

from *leaves-transversal-segmentE*[*OF assms*(*1*)]
obtain *d n* **where** *d*: $d > (0{::}real)$
and *n*: $n = a - b \vee n = b - a$
and *d-ex*: $\bigwedge x.\ x \in \{a {-}{-}\ b\} \Longrightarrow \{-d..d\} \subseteq existence\text{-}ivl0\ x$
and *d-above*: $\bigwedge x\ s.\ x \in \{a {-}{-}\ b\} \Longrightarrow 0 < s \Longrightarrow s \leq d \Longrightarrow (flow0\ x\ s - x) \cdot rot\ n > 0$
and *d-below*: $\bigwedge x\ s.\ x \in \{a {-}{-}\ b\} \Longrightarrow -d \leq s \Longrightarrow s < 0 \Longrightarrow (flow0\ x\ s - x) \cdot rot\ n < 0$
by *blast*

have *ortho*: $(a - b) \cdot rot\ n = 0$
using *n* **by** (*auto simp*: *algebra-simps*)

define *r1* **where** $r1 = (\lambda(x, y).\ flow0\ x\ y)\text{`}(\{flow0\ x\ t1{<}{-}{-}{<}b\} \times \{0{<}..{<}d\})$
have *r1a1*: *path-connected* $\{flow0\ x\ t1\ {<}{-}{-}{<}b\}$ **by** *simp*
have *r1a2*: *path-connected* $\{0{<}..{<}d\}$ **by** *simp*
have $\{flow0\ x\ t1{<}{-}{-}{<}b\} \subseteq \{a{-}{-}b\}$
by (*simp add*: *open-closed-segment subset-oc-segment x1*)
then have *r1a3*: $y \in \{flow0\ x\ t1{<}{-}{-}{<}b\} \Longrightarrow \{0{<}..{<}d\} \subseteq existence\text{-}ivl0\ y$ **for** *y*
using *d-ex*[*of y*]
by *force*
from *flow0-path-connected*[*OF r1a1 r1a2 r1a3*]
have *pcr1*:*path-connected r1* **unfolding** *r1-def* **by** *auto*
have *pir1J1*: $r1 \cap path\text{-}image\ J1 = \{\}$
unfolding *J1-def path-image-flow-to-path*[*OF* ‹$t1 \leq t2$›]
proof (*rule ccontr*)
assume $r1 \cap flow0\ x\ \text{`}\ \{t1..t2\} \neq \{\}$
then obtain *xx tt ss* **where**
eq: $flow0\ xx\ tt = flow0\ x\ ss$
and *xx*: $xx \in \{flow0\ x\ t1{<}{-}{-}{<}b\}$
and *ss*: $t1 \leq ss\ ss \leq t2$
and *tt*: $0 < tt\ tt < d$
unfolding *r1-def*
by *force*
have $xx \in \{a {-}{-}\ b\}$
using *sub1b*
apply (*rule set-mp*)
using *xx* **by** (*simp add*: *open-closed-segment*)
then have [*simp*]: $xx \in X$ **using** ‹*transversal-segment a b*› **by** (*auto simp*:

transversal-segment-def)
  from *ss* **have** *ss-ex*: *ss* ∈ *existence-ivl0 x* **using** *exist*
   by *auto*
  from *d-ex*[*OF* ‹*xx* ∈ {*a* −− *b*}›] *tt*
  have *tt-ex*: *tt* ∈ *existence-ivl0 xx* **by** *auto*
  then have *neg-tt-ex*: − *tt* ∈ *existence-ivl0* (*flow0 xx tt*)
   by (*rule existence-ivl-reverse*[*simplified*])
  from *eq* **have** *flow0* (*flow0 xx tt*) (−*tt*) = *flow0* (*flow0 x ss*) (−*tt*)
   by *simp*
  then have *xx* = *flow0 x* (*ss* − *tt*)
   apply (*subst* (*asm*) *flow-trans*[*symmetric*])
    apply (*rule tt-ex*)
   apply (*rule neg-tt-ex*)
   apply (*subst* (*asm*) *flow-trans*[*symmetric*])
    apply (*rule ss-ex*)
   apply (*subst eq*[*symmetric*])
   apply (*rule neg-tt-ex*)
   by *simp*
  moreover
  define *e* **where** *e* = *ss* − *t1*
  consider *e* > *tt* | *e* ≤ *tt* **by** *arith*
  then show *False*
  proof *cases*
   case *1*
   have *flow0* (*flow0 x ss*) (−*tt*) ∉ {*a*<−−<*b*}
    apply (*subst flow-trans*[*symmetric*])
     apply *fact*
    subgoal using *neg-tt-ex eq* **by** *simp*
    apply (*rule t1t2*)
    using *1 ss tt*
    unfolding *e-def*
    by *auto*
   moreover have *flow0* (*flow0 x ss*) (−*tt*) ∈ {*a*<−−<*b*}
    unfolding *eq*[*symmetric*] **using** *tt-ex xx*
    apply (*subst flow-trans*[*symmetric*])
     apply (*auto simp add*: *neg-tt-ex*)
    by (*metis* (*no-types*, *opaque-lifting*) *sub1b subset-eq subset-open-segment*)
   ultimately show *?thesis* **by** *simp*
  next
   case *2*
   have *les*: *0* ≤ *tt* − *e tt* − *e* ≤ *d*
    using *tt ss 2 e-def*
    by *auto*
   have *xxtte*: *flow0 xx* (*tt* − *e*) = *flow0 x t1*
    apply (*simp add*: *e-def*)
    by (*smt* ‹*0* ≤ *tt* − *e*› ‹{− *d*..*d*} ⊆ *existence-ivl0 xx*› *atLeastAtMost-iff e-def eq*
      local.existence-ivl-reverse local.existence-ivl-trans local.flow-trans ss(*1*) *ss-ex subset-iff tt*(*2*))

```
      show False
      proof (cases tt = e)
        case True
        with xxtte have xx = flow0 x t1
          by simp
        with xx show ?thesis
          apply auto
          by (auto simp: open-segment-def)
      next
        case False
        with les have 0 < tt − e by (simp)
        from d-above[OF ‹xx ∈ {a −− b}› this ‹tt − e ≤ d›]
        have flow0 xx (tt − e) ∉ {a −− b}
          apply (simp add: in-closed-segment-iff-rot[OF ‹a ≠ b›]
              not-le )
            by (smt ‹xx ∈ {a−−b}› inner-minus-right inner-rot-neg-move-base in-
ner-rot-pos-move-base n rot-diff-commute)
        with xxtte show ?thesis
          using ‹flow0 x t1 ∈ {a<−−<flow0 x t2}› suba2o by auto
      qed
    qed
  qed

  moreover
  have pir1J2: r1 ∩ path-image J2 = {}
  proof −
    have r1 ⊆ {x. (x − a) · rot n > 0}
      unfolding r1-def
    proof safe
      fix aa ba
      assume aa ∈ {flow0 x t1<−−<b} ba ∈ {0<..<d}
      with sub1b show 0 < (flow0 aa ba − a) · rot n
        using segment-open-subset-closed[of flow0 x t1 b]
        by (intro inner-pos-move-base[OF ortho d-above]) auto
    qed
    also have ... ∩ {a −− b} = {}
      using in-segment-inner-rot in-segment-inner-rot2 n by auto
    finally show ?thesis
      unfolding J2-def path-image-linepath
      using t12sub open-closed-segment
      by (force simp: closed-segment-commute)
  qed
  ultimately have pir1:r1 ∩ (path-image J) = {} unfolding J-def
    by (metis disjoint-iff-not-equal not-in-path-image-join)

  define r2 where r2 =(λ(x, y). flow0 x y)‘({a <−−< flow0 x t2} × {−d<..<0})
  have r2a1:path-connected {a <−−< flow0 x t2} by simp
  have r2a2:path-connected {−d<..<0} by simp
  have {a <−−< flow0 x t2} ⊆ {a −− b}
```

by (*meson ends-in-segment*(*1*) *open-closed-segment subset-co-segment subset-oc-segment t12sub*)
then have *r2a3*: $y \in \{a <\!--\!< \mathit{flow0}\ x\ t2\} \Longrightarrow \{-d<..<0\} \subseteq \mathit{existence\text{-}ivl0}\ y$
for *y*
using *d-ex*[*of y*]
by *force*
from *flow0-path-connected*[*OF r2a1 r2a2 r2a3*]
have *pcr2*:*path-connected r2* **unfolding** *r2-def* **by** *auto*
have *pir2J2*: $r2 \cap \mathit{path\text{-}image}\ J1 = \{\}$
unfolding *J1-def path-image-flow-to-path*[*OF* ‹$t1 \le t2$›]
proof (*rule ccontr*)
assume $r2 \cap \mathit{flow0}\ x\ `\ \{t1..t2\} \neq \{\}$
then obtain *xx tt ss* **where**
eq: $\mathit{flow0}\ xx\ tt = \mathit{flow0}\ x\ ss$
and *xx*: $xx \in \{a<\!--\!<\mathit{flow0}\ x\ t2\}$
and *ss*: $t1 \le ss\ ss \le t2$
and *tt*: $-d < tt\ tt < 0$
unfolding *r2-def*
by *force*
have $xx \in \{a -\!- b\}$
using *suba2*
apply (*rule set-mp*)
using *xx* **by** (*simp add*: *open-closed-segment*)
then have [*simp*]: $xx \in X$ **using** ‹*transversal-segment a b*› **by** (*auto simp*: *transversal-segment-def*)
from *ss* **have** *ss-ex*: $ss \in \mathit{existence\text{-}ivl0}\ x$ **using** *exist*
by *auto*
from *d-ex*[*OF* ‹$xx \in \{a -\!- b\}$›] *tt*
have *tt-ex*: $tt \in \mathit{existence\text{-}ivl0}\ xx$ **by** *auto*
then have *neg-tt-ex*: $-\ tt \in \mathit{existence\text{-}ivl0}\ (\mathit{flow0}\ xx\ tt)$
by (*rule existence-ivl-reverse*[*simplified*])
from *eq* **have** $\mathit{flow0}\ (\mathit{flow0}\ xx\ tt)\ (-tt) = \mathit{flow0}\ (\mathit{flow0}\ x\ ss)\ (-tt)$
by *simp*
then have $xx = \mathit{flow0}\ x\ (ss - tt)$
apply (*subst* (*asm*) *flow-trans*[*symmetric*])
apply (*rule tt-ex*)
apply (*rule neg-tt-ex*)
apply (*subst* (*asm*) *flow-trans*[*symmetric*])
apply (*rule ss-ex*)
apply (*subst eq*[*symmetric*])
apply (*rule neg-tt-ex*)
by *simp*
moreover
define *e* **where** $e = t2 - ss$
consider $e > -\ tt \mid e \le -tt$ **by** *arith*
then show *False*
proof *cases*
case *1*
have $\mathit{flow0}\ (\mathit{flow0}\ x\ ss)\ (-tt) \notin \{a<\!--\!<b\}$

```
        apply (subst flow-trans[symmetric])
          apply fact
        subgoal using neg-tt-ex eq by simp
        apply (rule t1t2)
        using 1 ss tt
        unfolding e-def
        by auto
      moreover have flow0 (flow0 x ss) (−tt) ∈ {a<−−<b}
        unfolding eq[symmetric] using tt-ex xx
        apply (subst flow-trans[symmetric])
          apply (auto simp add: neg-tt-ex)
        by (metis (no-types, opaque-lifting) suba2 subset-eq subset-open-segment)
      ultimately show ?thesis by simp
    next
      case 2
      have les: tt + e ≤ 0 −d ≤ tt + e
        using tt ss 2 e-def
        by auto
      have xxtte: flow0 xx (tt + e) = flow0 x t2
        apply (simp add: e-def)
        by (smt atLeastAtMost-iff calculation eq exist local.existence-ivl-trans' lo-
cal.flow-trans neg-tt-ex ss-ex subset-iff ‹t1 ≤ t2›)
      show False
      proof (cases tt=−e)
        case True
        with xxtte have xx = flow0 x t2
          by simp
        with xx show ?thesis
          apply auto
          by (auto simp: open-segment-def)
      next
        case False
        with les have tt+e < 0 by simp
        from d-below[OF ‹xx ∈ {a −− b}› ‹−d ≤ tt + e› this]
        have flow0 xx (tt + e) ∉ {a −− b}
          apply (simp add: in-closed-segment-iff-rot[OF ‹a ≠ b›]
              not-le )
            by (smt ‹xx ∈ {a−−b}› inner-minus-right inner-rot-neg-move-base in-
ner-rot-pos-move-base n rot-diff-commute)
        with xxtte show ?thesis
          using ‹flow0 x t2 ∈ {a−−b}› by simp
      qed
    qed
  qed
  moreover
  have pir2J2: r2 ∩ path-image J2 = {}
  proof −
    have r2 ⊆ {x. (x − a) · rot n < 0}
      unfolding r2-def
```

proof *safe*
  fix *aa ba*
  assume $aa \in \{a{<}{-}{-}{<}flow0\ x\ t2\}$ $ba \in \{-d{<}..{<}0\}$
  with *suba2* **show** $0 > (flow0\ aa\ ba - a) \cdot rot\ n$
    using *segment-open-subset-closed*[*of a flow0 x t2*]
    by (*intro inner-neg-move-base*[*OF ortho d-below*]) *auto*
qed
also have $\ldots \cap \{a -- b\} = \{\}$
  using *in-segment-inner-rot in-segment-inner-rot2 n* **by** *auto*
finally show *?thesis*
  unfolding *J2-def path-image-linepath*
  using *t12sub open-closed-segment*
  by (*force simp*: *closed-segment-commute*)
qed
ultimately have *pir2*:$r2 \cap (path\text{-}image\ J) = \{\}$
  unfolding *J-def*
  by (*metis disjoint-iff-not-equal not-in-path-image-join*)

define *rp* **where** $rp = midpoint\ (flow0\ x\ t1)\ (flow0\ x\ t2)$
have *rpi*: $rp \in \{flow0\ x\ t1{<}{-}{-}{<}flow0\ x\ t2\}$ **unfolding** *rp-def*
  by (*simp add*: *x1neqx2*)
have $rp \in \{a -- b\}$
  using *rpi suba2o subl* **by** *blast*
then have [*simp*]: $rp \in X$
  using ‹$\{a{-}{-}b\} \subseteq X$› **by** *blast*

have $*$: $pathfinish\ J1 = flow0\ x\ t2$
  $pathstart\ J1 = flow0\ x\ t1$
  $rp \in \{flow0\ x\ t2{<}{-}{-}{<}flow0\ x\ t1\}$
  using *rpi*
  by (*auto simp*: *open-segment-commute J1-def*)
have $\{y.\ 0 < (y - flow0\ x\ t2) \cdot rot\ (flow0\ x\ t2 - flow0\ x\ t1)\} = \{y.\ 0 < (y - rp) \cdot rot\ (flow0\ x\ t2 - flow0\ x\ t1)\}$
  by (*smt Collect-cong in-open-segment-rotD inner-diff-left nrm-dot rpi*)
also have ... $= \{y.\ 0 > (y - rp) \cdot rot\ (flow0\ x\ t1 - flow0\ x\ t2)\}$
  by (*smt Collect-cong inner-minus-left nrm-reverse*)
also have ... $= \{y.\ 0 > (y - rp) \cdot rot\ (a - b)\ \}$
  by (*metis rot-same-dir*(*2*) *x1 x2*)
finally have *side1*: $\{y.\ 0 < (y - flow0\ x\ t2) \cdot rot\ (flow0\ x\ t2 - flow0\ x\ t1)\} = \{y.\ 0 > (y - rp) \cdot rot\ (a - b)\ \}$
  (**is** - = *?lower1*) **.**
have $\{y.\ (y - flow0\ x\ t2) \cdot rot\ (flow0\ x\ t2 - flow0\ x\ t1) < 0\} = \{y.\ (y - rp) \cdot rot\ (flow0\ x\ t2 - flow0\ x\ t1) < 0\}$
  by (*smt Collect-cong in-open-segment-rotD inner-diff-left nrm-dot rpi*)
also have ... $= \{y.\ (y - rp) \cdot rot\ (flow0\ x\ t1 - flow0\ x\ t2) > 0\}$
  by (*smt Collect-cong inner-minus-left nrm-reverse*)
also have ... $= \{y.\ 0 < (y - rp) \cdot rot\ (a - b)\ \}$
  by (*metis rot-same-dir*(*1*) *x1 x2*)

finally have *side2*: $\{y.\ (y - \mathit{flow0}\ x\ t2) \cdot \mathit{rot}\ (\mathit{flow0}\ x\ t2 - \mathit{flow0}\ x\ t1) < 0\} =$ $\{y.\ 0 < (y - rp) \cdot rot\ (a - b)\ \}$
(**is** - = *?upper1*) **.**
from *linepath-ball-inside-outside*[*OF* ‹*simple-path J*›[*unfolded J-def J2-def*] ∗, *folded J2-def J-def*, *unfolded side1 side2*]
obtain *e* **where** *e0*: $0 < e$
ball rp e ∩ *path-image J1* = {}
ball rp e ∩ *?lower1* ⊆ *inner* ∧
ball rp e ∩ *?upper1* ⊆ *outer* ∨
ball rp e ∩ *?upper1* ⊆ *inner* ∧
ball rp e ∩ *?lower1* ⊆ *outer*
by (*auto simp*: *inner-def outer-def*)

let *?lower* = $\{y.\ 0 > (y - rp) \cdot rot\ n\ \}$
let *?upper* = $\{y.\ 0 < (y - rp) \cdot rot\ n\ \}$
have *?lower1* = $\{y.\ 0 < (y - rp) \cdot rot\ n\ \}$ ∧ *?upper1* = $\{y.\ 0 > (y - rp) \cdot rot$ $n\ \}$ ∨
?lower1 = $\{y.\ 0 > (y - rp) \cdot rot\ n\ \}$ ∧ *?upper1* = $\{y.\ 0 < (y - rp) \cdot rot\ n$ $\}$
using *n rot-diff-commute*[*of a b*]
by *auto*
from *this e0* **have** *e*: $0 < e$
ball rp e ∩ *path-image J1* = {}
ball rp e ∩ *?lower* ⊆ *inner* ∧
ball rp e ∩ *?upper* ⊆ *outer* ∨
ball rp e ∩ *?upper* ⊆ *inner* ∧
ball rp e ∩ *?lower* ⊆ *outer*
by *auto*

have $\forall_F$ *t in at-right 0*. $t < d$
by (*auto intro*!: *order-tendstoD* ‹$0 < d$›)
then have *evr*: $\forall_F$ *t in at-right 0*. $0 < (\mathit{flow0}\ rp\ t - rp) \cdot rot\ n$
unfolding *eventually-at-filter*
by *eventually-elim* (*auto intro*!: ‹$rp \in \{a{-}{-}b\}$› *d-above*)
have $\forall_F$ *t in at-left 0*. $t > -d$
by (*auto intro*!: *order-tendstoD* ‹$0 < d$›)
then have *evl*: $\forall_F$ *t in at-left 0*. $0 > (\mathit{flow0}\ rp\ t - rp) \cdot rot\ n$
unfolding *eventually-at-filter*
by *eventually-elim* (*auto intro*!: ‹$rp \in \{a{-}{-}b\}$› *d-below*)
have $\forall_F$ *t in at 0*. *flow0 rp t* ∈ *ball rp e*
unfolding *mem-ball*
apply (*subst dist-commute*)
apply (*rule tendstoD*)
by (*auto intro*!: *tendsto-eq-intros* ‹$0 < e$›)
then have *evl2*: ($\forall_F$ *t in at-left 0*. *flow0 rp t* ∈ *ball rp e*)
and *evr2*: ($\forall_F$ *t in at-right 0*. *flow0 rp t* ∈ *ball rp e*)
unfolding *eventually-at-split* **by** *auto*
have *evl3*: ($\forall_F$ *t in at-left 0*. $t > -d$)
and *evr3*: ($\forall_F$ *t in at-right 0*. $t < d$)

by (*auto intro*!: *order-tendstoD* ‹*0* < *d*›)
have *evl4*: (∀$_F$ *t in at-left 0*. *t* < *0*)
and *evr4*: (∀$_F$ *t in at-right 0*. *t* > *0*)
by (*auto simp*: *eventually-at-filter*)
from *evl evl2 evl3 evl4*
have ∀$_F$ *t in at-left 0*. (*flow0 rp t* − *rp*) · *rot n* < *0* ∧ *flow0 rp t* ∈ *ball rp e* ∧ *t* > −*d* ∧ *t* < *0*
by *eventually-elim auto*
from *eventually-happens*[*OF this*]
obtain *dl* **where** *dl*: (*flow0 rp dl* − *rp*) · *rot n* < *0 flow0 rp dl* ∈ *ball rp e* − *d* < *dl dl* < *0*
by *auto*
from *evr evr2 evr3 evr4*
have ∀$_F$ *t in at-right 0*. (*flow0 rp t* − *rp*) · *rot n* > *0* ∧ *flow0 rp t* ∈ *ball rp e* ∧ *t* < *d* ∧ *t* > *0*
by *eventually-elim auto*
from *eventually-happens*[*OF this*]
obtain *dr* **where** *dr*: (*flow0 rp dr* − *rp*) · *rot n* > *0 flow0 rp dr* ∈ *ball rp e d* > *dr dr* > *0*
by *auto*

have *rp* ∈ {*flow0 x t1*<−−<*b*} **using** *rpi subr* **by** *auto*
then have *rpr1*:*flow0 rp* (*dr*) ∈ *r1* **unfolding** *r1-def* **using** ‹*d* > *dr*› ‹*dr* > *0*›
by *auto*
have *rp* ∈ {*a*<−−<*flow0 x t2*} **using** *rpi subl* **by** *auto*
then have *rpr2*:*flow0 rp* (*dl*) ∈ *r2* **unfolding** *r2-def* **using** ‹−*d* < *dl*› ‹*dl* < *0*›
by *auto*

from *e*(*3*) *dr dl*
have *flow0 rp* (*dr*) ∈ *outer* ∧ *flow0 rp* (*dl*) ∈ *inner* ∨ *flow0 rp* (*dr*) ∈ *inner* ∧ *flow0 rp* (*dl*) ∈ *outer*
by *auto*
moreover {
assume *flow0 rp dr* ∈ *outer flow0 rp dl* ∈ *inner*
then have
r1o: *r1* ∩ *outer* ≠ {} **and**
r2i: *r2* ∩ *inner* ≠ {} **using** *rpr1 rpr2* **by** *auto*
from *path-connected-not-frontier-subset*[*OF pcr1 r1o*]
have *r1* ⊆ *outer* **using** *pir1* **by** (*simp add*: *io*(*12*))
from *path-connected-not-frontier-subset*[*OF pcr2 r2i*]
have *r2* ⊆ *inner* **using** *pir2* **by** (*simp add*: *io*(*11*))
have (λ(*x*, *y*). *flow0 x y*)‘({*flow0 x t2*} × {*0*<..<*d*}) ⊆ *r1* **unfolding** *r1-def*
by (*auto intro*!:*image-mono simp add*: *x2*)
then have ∗:⋀*t*. *0* < *t* ⟹ *t* < *d* ⟹ *flow0* (*flow0 x t2*) *t* ∈ *outer*
by (*smt* ‹*r1* ⊆ *outer*› *greaterThanLessThan-iff mem-Sigma-iff pair-imageI r1-def subset-eq x2*)

then have *t2o*: ⋀*t*. *0* < *t* ⟹ *t* < *d* ⟹ *flow0 x* (*t2* + *t*) ∈ *outer*
using *r1a3*[*OF x2*] *exist flow-trans*

by (*metis* (*no-types*, *opaque-lifting*) *closed-segment-commute ends-in-segment*(*1*) *local.existence-ivl-trans′ local.flow-undefined0 real-Icc-closed-segment subset-eq* ‹*t1* ≤ *t2*›)

have *inner*: {*a* <−−< *flow0 x t2*} ⊆ *closure inner*
proof (*rule subsetI*)
fix *y*
assume *y*: *y* ∈ {*a* <−−< *flow0 x t2*}
have [*simp*]: *y* ∈ *X*
using *y suba12-open suba2o* ‹{*a* −− *b*} ⊆ *X*›
by *auto*
have (∀ *n*. *flow0 y* (− *d* / *real* (*Suc* (*Suc n*))) ∈ *inner*)
using *y*
using *suba12-open* ‹*0* < *d*› *suba2o* ‹{*a* −− *b*} ⊆ *X*›
by (*auto intro*!: *set-mp*[*OF* ‹*r2* ⊆ *inner*›] *image-eqI*[**where** *x*=(*y*, −*d*/*Suc* (*Suc n*)) **for** *n*]
simp: *r2-def divide-simps*)
moreover
have *d-over-0*: (λ*s*. − *d* / *real* (*Suc* (*Suc s*))) ⟶ *0*
by (*rule real-tendsto-divide-at-top*)
(*auto intro*!: *filterlim-tendsto-add-at-top filterlim-real-sequentially*)
have (λ*n*. *flow0 y* (− *d* / *real* (*Suc* (*Suc n*)))) ⟶ *y*
apply (*rule tendsto-eq-intros*)
apply (*rule tendsto-intros*)
apply (*rule d-over-0*)
by *auto*
ultimately show *y* ∈ *closure inner*
unfolding *closure-sequential*
by (*intro exI*[**where** *x*=λ*n*. *flow0 y* (−*d*/*Suc* (*Suc n*))]) (*rule conjI*)
qed
then have {*a* <−−< *flow0 x t1*} ⊆ *closure inner*
using *suba12-open* **by** *blast*
then have {*flow0 x t1* −− *flow0 x t2*} ⊆ *closure inner*
by (*metis* (*no-types*, *lifting*) *closure-closure closure-mono closure-open-segment dual-order.trans inner subl x1neqx2*)
have *outer*:⋀*t*. *t* > *t2* ⟹ *t* ∈ *existence-ivl0 x* ⟹ *flow0 x t* ∈ *outer*
proof (*rule ccontr*)
fix *t*
assume *t*: *t* > *t2 t* ∈ *existence-ivl0 x flow0 x t* ∉ *outer*
have *0* ≤ *t*− (*t2*+*d*) **using** *t2o t* **by** *smt*
then have *a2*:*0* ≤ *t* − (*t2*+*dr*) **using** *d* ‹*0* < *dr*› ‹*dr* < *d*› **by** *linarith*
have *t2d2-ex*: *t2* + *dr* ∈ *existence-ivl0 x*
using ‹*t1* ≤ *t2*› *exist d-ex*[*of flow0 x t2*] ‹*flow0 x t2* ∈ {*a*−−*b*}› ‹*0* < *d*› ‹*0* < *dr*› ‹*dr* < *d*›
by (*intro existence-ivl-trans*) *auto*
then have *a3*: *t* − (*t2* + *dr*) ∈ *existence-ivl0* (*flow0 x* (*t2* + *dr*))
using *t*(*2*)
by (*intro diff-existence-ivl-trans*) *auto*

then have $\mathit{flow0}\ (\mathit{flow0}\ x\ (t2 + dr))\ (t - (t2 + dr)) = \mathit{flow0}\ x\ t$
by (*subst flow-trans*[*symmetric*]) (*auto simp*: *t2d2-ex*)
moreover have $\mathit{flow0}\ x\ t \in \mathit{closure}\ \mathit{inner}$ **using** $t(3)$ *io*
by (*metis ComplI Un-iff closure-Un-frontier*)
ultimately have *a4*: $\mathit{flow0}\ (\mathit{flow0}\ x\ (t2 + dr))\ (t - (t2 + dr)) \in \mathit{closure}\ \mathit{inner}$ **by** *auto*
have *a1*: $\mathit{flow0}\ x\ (t2{+}dr) \in \mathit{outer}$
by (*simp add*: *d t2o* ‹$0 < dr$› ‹$dr < d$›)
from *outside-in*[*OF a1 a2 a3 a4*]
obtain T **where** T: $T > 0$ $T \le t - (t2 + dr)$
$(\forall s\in\{0..<T\}.\ \mathit{flow0}\ (\mathit{flow0}\ x\ (t2 + dr))\ s \in \mathit{outer})$
$\mathit{flow0}\ (\mathit{flow0}\ x\ (t2 + dr))\ T \in \{\mathit{flow0}\ x\ t1\ {-}{-}{<}\ \mathit{flow0}\ x\ t2\}$ **by** *blast*
define y **where** $y = \mathit{flow0}\ (\mathit{flow0}\ x\ (t2 + dr))\ T$
have $y \in \{a\ {<}{-}{-}{<}\ \mathit{flow0}\ x\ t2\}$ **unfolding** *y-def* **using** $T(4)$
using *subl2* **by** *blast*
then have $(\lambda(x, y).\ \mathit{flow0}\ x\ y)`(\{y\} \times \{-d{<}..{<}0\}) \subseteq r2$ **unfolding** *r2-def*
by (*auto intro*!:*image-mono*)
then have $*$:$\bigwedge t.\ -d < t \Longrightarrow t < 0 \Longrightarrow \mathit{flow0}\ y\ t \in r2$
by (*simp add*: *pair-imageI subsetD*)
have $\mathit{max}\ (-T/2)\ dl < 0$ **using** *d T* ‹$0 > dl$› ‹$dl > -d$› **by** *auto*
moreover have $-d < \mathit{max}\ (-T/2)\ dl$ **using** *d T* ‹$0 > dl$› ‹$dl > -d$› **by** *auto*
ultimately have *inner*: $\mathit{flow0}\ y\ (\mathit{max}\ (-T/2)\ dl) \in \mathit{inner}$ **using** $*$ ‹$r2 \subseteq \mathit{inner}$› **by** *blast*
have $0 \le (T{+}(\mathit{max}\ (-T/2)\ dl))$ **using** $T(1)$ **by** *linarith*
moreover have $(T{+}(\mathit{max}\ (-T/2)\ dl)) < T$ **using** $T(1)$ *d* ‹$0 > dl$› ‹$dl > -d$› **by** *linarith*
ultimately have *outer*: $\mathit{flow0}\ (\mathit{flow0}\ x\ (t2 + dr))\ (T{+}(\mathit{max}\ (-T/2)\ dl)) \in \mathit{outer}$
using T **by** *auto*
have *T-ex*: $T \in \mathit{existence\text{-}ivl0}\ (\mathit{flow0}\ x\ (t2 + dr))$
apply (*subst flow-trans*)
using *exist* ‹$t1 \le t2$›
using *d-ex*[*of flow0 x t2*] ‹$\mathit{flow0}\ x\ t2 \in \{a\ {-}{-}\ b\}$› ‹$d > 0$› T ‹$0 < dr$› ‹$dr < d$›
apply *auto*
apply (*rule set-rev-mp*[**where** $A{=}\{0\ ..\ t - (t2 + dr)\}$], *force*)
apply (*rule ivl-subset-existence-ivl*)
apply (*rule existence-ivl-trans′*)
apply (*rule existence-ivl-trans′*)
by (*auto simp*: *t*)
have *T-ex2*: $dr + T \in \mathit{existence\text{-}ivl0}\ (\mathit{flow0}\ x\ t2)$
by (*smt T-ex ends-in-segment*(*2*) *exist local.existence-ivl-trans local.existence-ivl-trans′ real-Icc-closed-segment subset-eq t2d2-ex* ‹$t1 \le t2$›)
thus *False* **using** T ‹$t1 \le t2$› *exist*
by (*smt T-ex diff-existence-ivl-trans disjoint-iff-not-equal inner io*(*9*) *local.flow-trans local.flow-undefined0 outer y-def*)
qed
have $\mathit{closure}\ \mathit{inner} \cap \mathit{outer} = \{\}$

by (*simp add*: *inf-sup-aci*(*1*) *io*(*5*) *io*(*9*) *open-Int-closure-eq-empty*)
then have *flow0 x t* $\notin$ {*a*<−−<*flow0 x t2*}
using ‹*t* > *t2*› ‹*t* ∈ *existence-ivl0 x*› *inner outer* **by** *blast*
}
moreover {
assume *flow0 rp dr* ∈ *inner flow0 rp dl* ∈ *outer*
then have
r1i: *r1* ∩ *inner* ≠ {} **and**
r2o: *r2* ∩ *outer* ≠ {} **using** *rpr1 rpr2* **by** *auto*
from *path-connected-not-frontier-subset*[*OF pcr1 r1i*]
have *r1* ⊆ *inner* **using** *pir1* **by** (*simp add*: *io*(*11*))
from *path-connected-not-frontier-subset*[*OF pcr2 r2o*]
have *r2* ⊆ *outer* **using** *pir2* **by** (*simp add*: *io*(*12*))

have (λ(*x*, *y*). *flow0 x y*)‘({*flow0 x t2*} × {*0*<..<*d*}) ⊆ *r1* **unfolding** *r1-def*
by (*auto intro*!:*image-mono simp add*: *x2*)
then have
∗:⋀*t*. *0* < *t* ⟹ *t* < *d* ⟹ *flow0* (*flow0 x t2*) *t* ∈ *inner*
by (*smt* ‹*r1* ⊆ *inner*› *greaterThanLessThan-iff mem-Sigma-iff pair-imageI r1-def subset-eq x2*)

then have *t2o*: ⋀*t*. *0* < *t* ⟹ *t* < *d* ⟹ *flow0 x* (*t2* + *t*) ∈ *inner*
using *r1a3*[*OF x2*] *exist flow-trans*
by (*metis* (*no-types*, *opaque-lifting*) *closed-segment-commute ends-in-segment*(*1*) *local.existence-ivl-trans*′ *local.flow-undefined0 real-Icc-closed-segment subset-eq* ‹*t1* ≤ *t2*›)

have *outer*: {*a* <−−< *flow0 x t2*} ⊆ *closure outer*
proof (*rule subsetI*)
fix *y*
assume *y*: *y* ∈ {*a* <−−< *flow0 x t2*}
have [*simp*]: *y* ∈ *X*
using *y suba12-open suba2o* ‹{*a* −− *b*} ⊆ *X*›
by *auto*
have (∀ *n*. *flow0 y* (− *d* / *real* (*Suc* (*Suc n*))) ∈ *outer*)
using *y*
using *suba12-open* ‹*0* < *d*› *suba2o* ‹{*a* −− *b*} ⊆ *X*›
by (*auto intro*!: *set-mp*[*OF* ‹*r2* ⊆ *outer*›] *image-eqI*[**where** *x*=(*y*, −*d*/*Suc* (*Suc n*)) **for** *n*]
simp: *r2-def divide-simps*)
moreover
have *d-over-0*: (λ*s*. − *d* / *real* (*Suc* (*Suc s*))) ⟶ *0*
by (*rule real-tendsto-divide-at-top*)
(*auto intro*!: *filterlim-tendsto-add-at-top filterlim-real-sequentially*)
have (λ*n*. *flow0 y* (− *d* / *real* (*Suc* (*Suc n*)))) ⟶ *y*
apply (*rule tendsto-eq-intros*)
apply (*rule tendsto-intros*)
apply (*rule d-over-0*)

by *auto*
ultimately show $y \in$ *closure outer*
unfolding *closure-sequential*
by (*intro exI*[**where** $x{=}\lambda n.$ *flow0 y* ($-d/Suc$ (*Suc n*))]) (*rule conjI*)
qed
then have $\{a <{-}{-}<$ *flow0 x t1*$\} \subseteq$ *closure outer*
using *suba12-open* **by** *blast*
then have $\{$*flow0 x t1* $--$ *flow0 x t2*$\} \subseteq$ *closure outer*
by (*metis* (*no-types*, *lifting*) *closure-closure closure-mono closure-open-segment dual-order.trans outer subl x1neqx2*)

have *inner*:$\bigwedge t.\ t > t2 \Longrightarrow t \in$ *existence-ivl0 x* $\Longrightarrow$ *flow0 x t* $\in$ *inner*
proof (*rule ccontr*)
fix *t*
assume *t*: $t > t2$ $t \in$ *existence-ivl0 x flow0 x t* $\notin$ *inner*
have $0 \le t-$ ($t2{+}d$) **using** *t2o t* **by** *smt*
then have *a2*:$0 \le t -$ ($t2{+}dr$) **using** *d* ‹$0 < dr$› ‹$dr < d$› **by** *linarith*
have *t2d2-ex*: $t2 + dr \in$ *existence-ivl0 x*
using ‹$t1 \le t2$› *exist d-ex*[*of flow0 x t2*] ‹*flow0 x t2* $\in \{a{-}{-}b\}$› ‹$0 < d$› ‹$0 < dr$› ‹$dr < d$›
by (*intro existence-ivl-trans*) *auto*
then have *a3*: $t -$ ($t2 + dr$) $\in$ *existence-ivl0* (*flow0 x* ($t2 + dr$))
using *t*(*2*)
by (*intro diff-existence-ivl-trans*) *auto*
then have *flow0* (*flow0 x* ($t2 + dr$)) ($t -$ ($t2 + dr$)) = *flow0 x t*
by (*subst flow-trans*[*symmetric*]) (*auto simp*: *t2d2-ex*)
moreover have *flow0 x t* $\in$ *closure outer* **using** *t*(*3*) *io*
by (*metis ComplI Un-iff closure-Un-frontier*)
ultimately have *a4*: *flow0* (*flow0 x* ($t2 + dr$)) ($t -$ ($t2 + dr$)) $\in$ *closure outer* **by** *auto*
have *a1*: *flow0 x* ($t2{+}dr$) $\in$ *inner*
by (*simp add*: *d t2o* ‹$0 < dr$› ‹$dr < d$›)
from *inside-out*[*OF a1 a2 a3 a4*]
obtain *T* **where** *T*: $T > 0$ $T \le t -$ ($t2 + dr$)
($\forall s{\in}\{0..{<}T\}$. *flow0* (*flow0 x* ($t2 + dr$)) *s* $\in$ *inner*)
flow0 (*flow0 x* ($t2 + dr$)) $T \in \{$*flow0 x t1* $--<$ *flow0 x t2*$\}$ **by** *blast*
define *y* **where** $y =$ *flow0* (*flow0 x* ($t2 + dr$)) *T*
have $y \in \{a <{-}{-}<$ *flow0 x t2*$\}$ **unfolding** *y-def* **using** *T*(*4*)
using *subl2* **by** *blast*
then have (λ(x, y). *flow0 x y*)‘($\{y\} \times \{-d{<}..{<}0\}$) $\subseteq$ *r2* **unfolding** *r2-def*
by (*auto intro*!:*image-mono*)
then have $*$:$\bigwedge t.\ -d < t \Longrightarrow t < 0 \Longrightarrow$ *flow0 y t* $\in$ *r2*
by (*simp add*: *pair-imageI subsetD*)
have *max* ($-T/2$) $dl < 0$ **using** *d T* ‹$0 > dl$› ‹$dl > -d$› **by** *auto*
moreover have $-d <$ *max* ($-T/2$) *dl* **using** *d T* ‹$0 > dl$› ‹$dl > -d$› **by** *auto*
ultimately have *outer*: *flow0 y* (*max* ($-T/2$) *dl*) $\in$ *outer* **using** $*$ ‹$r2 \subseteq$ *outer*› **by** *blast*
have $0 \le$($T{+}$(*max* ($-T/2$) *dl*)) **using** *T*(*1*) **by** *linarith*

moreover have $(T+(max\ (-T/2)\ dl)) < T$ **using** $T(1)$ d ‹$0 > dl$› ‹$dl > -d$› **by** *linarith*
ultimately have *inner*: *flow0* (*flow0* x ($t2$ + dr)) (T+(*max* ($-T/2$) *dl*)) $\in$ *inner*
using T **by** *auto*
have *T-ex*: $T \in$ *existence-ivl0* (*flow0* x ($t2$ + dr))
apply (*subst flow-trans*)
using *exist* ‹$t1 \le t2$›
using *d-ex*[*of flow0 x t2*] ‹*flow0* x $t2 \in \{a -- b\}$› ‹$d > 0$› T ‹$0 < dr$› ‹$dr < d$›
apply *auto*
apply (*rule set-rev-mp*[**where** $A=\{0\ ..\ t - (t2 + dr)\}$], *force*)
apply (*rule ivl-subset-existence-ivl*)
apply (*rule existence-ivl-trans*′)
apply (*rule existence-ivl-trans*′)
by (*auto simp*: *t*)
have *T-ex2*: $dr + T \in$ *existence-ivl0* (*flow0* x $t2$)
by (*smt T-ex ends-in-segment*(2) *exist local.existence-ivl-trans local.existence-ivl-trans*′ *real-Icc-closed-segment subset-eq t2d2-ex* ‹$t1 \le t2$›)
thus *False* **using** T ‹$t1 \le t2$› *exist*
by (*smt T-ex diff-existence-ivl-trans disjoint-iff-not-equal inner io*(9) *local.flow-trans local.flow-undefined0 outer y-def*)
qed
have *closure outer* $\cap$ *inner* = {}
by (*metis inf-sup-aci*(1) *io*(2) *io2*(1) *open-Int-closure-eq-empty*)
then have *flow0* x $t \notin \{a<--<$*flow0* x $t2\}$
using ‹$t > t2$› ‹$t \in$ *existence-ivl0* x› *inner outer* **by** *blast*
}
ultimately show
flow0 x $t \notin \{a<--<$*flow0* x $t2\}$ **by** *auto*
qed

lemma *open-segment-trichotomy*:
fixes x y a b::′a
assumes x:$x \in \{a<--<b\}$
assumes y:$y \in \{a<--<b\}$
shows $x = y \lor y \in \{x<--<b\} \lor y \in \{a<--<x\}$
proof −
from *Un-open-segment*[*OF y*]
have $\{a<--<y\} \cup \{y\} \cup \{y<--<b\} = \{a<--<b\}$ **.**
then have $x \in \{a<--<y\} \lor x = y \lor x \in \{y <--<b\}$ **using** x **by** *blast*
moreover {
assume $x \in \{a<--<y\}$
then have $y \in \{x<--<b\}$ **using** *open-segment-subsegment*
using *open-segment-commute* y **by** *blast*
}
moreover {
assume $x \in \{y<--<b\}$
from *open-segment-subsegment*[*OF y this*]

have $y \in \{a{<}{-}{-}{<}x\}$ **.**
}
ultimately show *?thesis* **by** *blast*
qed

sublocale *rev*: *c1-on-open-R2* $-f$ $-f'$ **rewrites** $-(-f) = f$ **and** $-(-f') = f'$
by *unfold-locales* (*auto simp*: *dim2*)

lemma *rev-transversal-segment*: *rev.transversal-segment a b* = *transversal-segment a b*
by (*auto simp*: *transversal-segment-def rev.transversal-segment-def*)

lemma *flow0-transversal-segment-monotone-step-reverse*:
assumes *transversal-segment a b*
assumes $t1 \le t2$
assumes $\{t1..t2\} \subseteq$ *existence-ivl0 x*
assumes *x1*: *flow0 x t1* $\in \{a{<}{-}{-}{<}b\}$
assumes *x2*: *flow0 x t2* $\in \{a{<}{-}{-}{<}$*flow0 x t1*$\}$
assumes $\bigwedge t.\ t \in \{t1{<}..{<}t2\} \Longrightarrow$ *flow0 x t* $\notin \{a{<}{-}{-}{<}b\}$
assumes $t < t1$ $t \in$ *existence-ivl0 x*
shows *flow0 x t* $\notin \{a{<}{-}{-}{<}$*flow0 x t1*$\}$
proof –
note *exist* = ‹$\{t1..t2\} \subseteq$ *existence-ivl0 x*›
note *t1t2* = ‹$\bigwedge t.\ t \in \{t1{<}..{<}t2\} \Longrightarrow$ *flow0 x t* $\notin \{a{<}{-}{-}{<}b\}$›
from ‹*transversal-segment a b*› **have** [*simp*]: $a \ne b$ **by** (*simp add*: *transversal-segment-def*)
from *x1* **obtain** *i1* **where** *i1*: *flow0 x t1* = *line a b i1* $0 < i1$ $i1 < 1$
by (*auto simp*: *in-open-segment-iff-line*)
from *x2* **obtain** *i2* **where** *i2*: *flow0 x t2* = *line a b i2* $0 < i2$ $i2 < i1$
by (*auto simp*: *i1 open-segment-line-iff*)

have *t2-exist*[*simp*]: $t2 \in$ *existence-ivl0 x*
using ‹$t1 \le t2$› *exist* **by** *auto*
have *t2-mem*: *flow0 x t2* $\in \{a{<}{-}{-}{<}b\}$
and *x1-mem*: *flow0 x t1* $\in \{$*flow0 x t2*${<}{-}{-}{<}b\}$
using *i1 i2*
by (*auto simp*: *line-in-subsegment line-line1*)

have *transversal'*: *rev.transversal-segment a b*
using ‹*transversal-segment a b*› **unfolding** *rev-transversal-segment* **.**
have *time'*: $0 \le t2 - t1$ **using** ‹$t1 \le t2$› **by** *simp*
have [*simp*, *intro*]: *flow0 x t2* $\in X$
using *exist* ‹$t1 \le t2$›
by *auto*
have *exivl'*: $\{0..t2 - t1\} \subseteq$ *rev.existence-ivl0* (*flow0 x t2*)
using *exist* ‹$t1 \le t2$›
by (*force simp add*: *rev-existence-ivl-eq0 intro*!: *existence-ivl-trans'*)
have *step'*: *rev.flow0* (*flow0 x t2*) $(t2-t)$ $\notin$ $\{a{<}{-}{-}{<}$*rev.flow0* (*flow0 x t2*) $(t2 - t1)\}$

apply (*rule rev.flow0-transversal-segment-monotone-step*[*OF transversal′ time′ exivl′*])
using *exist* ‹*t1* ≤ *t2*› *x1 x2 t2-mem x1-mem t1t2* ‹*t* < *t1*› ‹*t* ∈ *existence-ivl0 x*›
apply (*auto simp*: *rev-existence-ivl-eq0 rev-eq-flow existence-ivl-trans′ flow-trans*[*symmetric*])
by (*subst* (*asm*) *flow-trans*[*symmetric*]) (*auto intro*!: *existence-ivl-trans′*)
then show *?thesis*
unfolding *rev-eq-flow*
using ‹*t1* ≤ *t2*› *exist* ‹*t* < *t1*› ‹*t* ∈ *existence-ivl0 x*›
by (*auto simp*: *flow-trans*[*symmetric*] *existence-ivl-trans′*)
qed

lemma *flow0-transversal-segment-monotone-step-reverse2*:
assumes *transversal*: *transversal-segment a b*
assumes *time*: *t1* ≤ *t2*
assumes *exist*: {*t1*..*t2*} ⊆ *existence-ivl0 x*
assumes *t1*: *flow0 x t1* ∈ {*a*<−−<*b*}
assumes *t2*: *flow0 x t2* ∈ {*flow0 x t1*<−−<*b*}
assumes *t1t2*: ⋀*t*. *t* ∈ {*t1*<..<*t2*} ⟹ *flow0 x t* ∉ {*a*<−−<*b*}
assumes *t*: *t* < *t1 t* ∈ *existence-ivl0 x*
shows *flow0 x t* ∉ {*flow0 x t1*<−−<*b*}
using *flow0-transversal-segment-monotone-step-reverse*[*of b a*, *OF* - *time exist*, *of t*]
assms
by (*auto simp*: *open-segment-commute transversal-segment-commute*)

lemma *flow0-transversal-segment-monotone-step2*:
assumes *transversal*: *transversal-segment a b*
assumes *time*: *t1* ≤ *t2*
assumes *exist*: {*t1*..*t2*} ⊆ *existence-ivl0 x*
assumes *t1*: *flow0 x t1* ∈ {*a*<−−<*b*}
assumes *t2*: *flow0 x t2* ∈ {*a*<−−<*flow0 x t1*}
assumes *t1t2*: ⋀*t*. *t* ∈ {*t1*<..<*t2*} ⟹ *flow0 x t* ∉ {*a*<−−<*b*}
shows ⋀*t*. *t* > *t2* ⟹ *t* ∈ *existence-ivl0 x* ⟹ *flow0 x t* ∉ {*flow0 x t2*<−−<*b*}
using *flow0-transversal-segment-monotone-step*[*of b a*, *OF* - *time exist*]
assms
by (*auto simp*: *transversal-segment-commute open-segment-commute*)

lemma *flow0-transversal-segment-monotone*:
assumes *transversal-segment a b*
assumes *t1* ≤ *t2*
assumes {*t1*..*t2*} ⊆ *existence-ivl0 x*
assumes *x1*: *flow0 x t1* ∈ {*a*<−−<*b*}
assumes *x2*: *flow0 x t2* ∈ {*flow0 x t1*<−−<*b*}
assumes *t* > *t2 t* ∈ *existence-ivl0 x*
shows *flow0 x t* ∉ {*a*<−−<*flow0 x t2*}
proof −
note *exist* = ‹{*t1*..*t2*} ⊆ *existence-ivl0 x*›
note *t* = ‹*t* > *t2*› ‹*t* ∈ *existence-ivl0 x*›

have *x1neqx2*: *flow0 x t1* $\neq$ *flow0 x t2*
  using *open-segment-def x2* **by** *force*
then have *t1neqt2*: *t1* $\neq$ *t2* **by** *auto*
with ‹*t1* $\leq$ *t2*› **have** *t1* $<$ *t2* **by** *simp*

from ‹*transversal-segment a b*› **have** [*simp*]: *a* $\neq$ *b* **by** (*simp add*: *transversal-segment-def*)
from *x1* **obtain** *i1* **where** *i1*: *flow0 x t1* = *line a b i1 0* $<$ *i1 i1* $<$ *1*
  by (*auto simp*: *in-open-segment-iff-line*)
from *x2 i1* **obtain** *i2* **where** *i2*: *flow0 x t2* = *line a b i2 i1* $<$ *i2 i2* $<$ *1*
  by (*auto simp*: *line-open-segment-iff*)
have *t2-in*: *flow0 x t2* $\in$ {*a*<−−<*b*}
  using *i1 i2*
  by *simp*

let *?T* = {*s* $\in$ {*t1*..*t2*}. *flow0 x s* $\in$ {*a*−−*b*}}
let *?T'* = {*s* $\in$ {*t1*..<*t2*}. *flow0 x s* $\in$ {*a*<−−<*b*}}
from *flow-transversal-segment-finite-intersections*[*OF* ‹*transversal-segment a b*› ‹*t1* $\leq$ *t2*› *exist*]
have *finite ?T* **.**
then have *finite ?T'* **by** (*rule finite-subset*[*rotated*]) (*auto simp*: *open-closed-segment*)
have *?T'* $\neq$ {}
  by (*auto intro*!: *exI*[**where** *x*=*t1*] ‹*t1* $<$ *t2*› *x1*)
note *tm-defined* = ‹*finite ?T'*› ‹*?T'* $\neq$ {}›
define *tm* **where** *tm* = *Max ?T'*
have *tm* $\in$ *?T'*
  unfolding *tm-def*
  using *tm-defined* **by** (*rule Max-in*)
have *tm-in*: *flow0 x tm* $\in$ {*a*<−−<*b*}
  using ‹*tm* $\in$ *?T'*›
  by *auto*
have *tm*: *t1* $\leq$ *tm tm* $<$ *t2 tm* $\leq$ *t2*
  using ‹*tm* $\in$ *?T'*› **by** *auto*
have *tm-Max*: *t* $\leq$ *tm* **if** *t* $\in$ *?T'* **for** *t*
  unfolding *tm-def*
  using *tm-defined*(*1*) *that*
  by (*rule Max-ge*)

have *tm-exclude*: *flow0 x t* $\notin$ {*a*<−−<*b*} **if** *t* $\in$ {*tm*<..<*t2*} **for** *t*
  using ‹*tm* $\in$ *?T'*› *tm-Max that*
  by *auto* (*meson approximation-preproc-push-neg*(*2*) *dual-order.strict-trans2 le-cases*)
have {*tm*..*t2*} $\subseteq$ *existence-ivl0 x*
  using *exist tm* **by** *auto*

from *open-segment-trichotomy*[*OF tm-in t2-in*]
consider
  flow0 x t2 $\in$ {*flow0 x tm*<−−<*b*} |
  flow0 x t2 $\in$ {*a*<−−<*flow0 x tm*} |

flow0 x tm = *flow0 x t2*
by *blast*
then show *flow0 x t* ∉ {*a*<−−<*flow0 x t2*}
proof *cases*
case *1*
from *flow0-transversal-segment-monotone-step*[*OF* ‹*transversal-segment a b*› ‹*tm* ≤ *t2*›
‹{*tm..t2*} ⊆ *existence-ivl0 x*› *tm-in 1 tm-exclude t*]
show *?thesis* **.**
next
case *2*
have *t1* ≠ *tm*
using *2 x2 i1 i2*
by (*auto simp*: *line-in-subsegment line-in-subsegment2*)
then have *t1* < *tm* **using** ‹*t1* ≤ *tm*› **by** *simp*
from *flow0-transversal-segment-monotone-step-reverse*[*OF* ‹*transversal-segment a b*› ‹*tm* ≤ *t2*›
‹{*tm..t2*} ⊆ *existence-ivl0 x*› *tm-in 2 tm-exclude* ‹*t1* < *tm*›] *exist* ‹*t1* ≤ *t2*›
have *flow0 x t1* ∉ {*a*<−−<*flow0 x tm*} **by** *auto*
then have *False*
using *x1 x2 2 i1 i2*
apply (*auto simp*: *line-in-subsegment line-in-subsegment2*)
by (*smt greaterThanLessThan-iff in-open-segment-iff-line line-in-subsegment2 tm-in*)
then show *?thesis* **by** *simp*
next
case *3*
have *t1* ≠ *tm*
using *3 x2*
by (*auto simp*: *open-segment-def*)
then have *t1* < *tm* **using** ‹*t1* ≤ *tm*› **by** *simp*
have *range* (*flow0 x*) = *flow0 x* ‘ {*tm..t2*}
apply (*rule recurrence-time-restricts-compact-flow*′[*OF* ‹*tm* < *t2*› - - *3*])
using *exist* ‹*t1* ≤ *t2*› ‹*t1* < *tm*› ‹*tm* < *t2*›
by *auto*
also have ... = *flow0 x* ‘ (*insert t2* {*tm*<..<*t2*})
using ‹*tm* ≤ *t2*› *3*
apply *auto*
by (*smt greaterThanLessThan-iff image-eqI*)
finally have *flow0 x t1* ∈ *flow0 x* ‘ (*insert t2* {*tm*<..<*t2*})
by *auto*
then have *flow0 x t1* ∈ *flow0 x* ‘ {*tm*<..<*t2*} **using** *x1neqx2*
by *auto*
moreover have ... ∩ {*a*<−−<*b*} = {}
using *tm-exclude*
by *auto*
ultimately have *False* **using** *x1* **by** *auto*
then show *?thesis* **by** *blast*
qed

qed

6.6 Straightening

This lemma uses the implicit function theorem

lemma *cross-time-continuous*:
  assumes *transversal-segment a b*
  assumes $x \in \{a{<}{-}{-}{<}b\}$
  assumes $e > 0$
  obtains *d t* **where** $d > 0$ *continuous-on* (*ball x d*) *t*
    $\bigwedge y.\ y \in ball\ x\ d \Longrightarrow flow0\ y\ (t\ y) \in \{a{<}{-}{-}{<}b\}$
    $\bigwedge y.\ y \in ball\ x\ d \Longrightarrow |t\ y| < e$
    continuous-on (*ball x d*) *t*
    $t\ x = 0$
proof −
  have $x \in X$ **using** *assms segment-open-subset-closed*[*of a b*]
    by (*auto simp*: *transversal-segment-def*)
  have $a \neq b$ **using** *assms* **by** *auto*
  define *s* **where** $s\ x = (x - a) \cdot rot\ (b - a)$ **for** *x*
  have $s\ x = 0$
    unfolding *s-def*
    by (*subst in-segment-inner-rot*) (*auto intro*!: *assms open-closed-segment*)
  have *Ds*: (*s has-derivative blinfun-inner-left* (*rot* (*b* − *a*))) (*at x*)
    (**is** (- *has-derivative blinfun-apply* (*?Ds x*)) -)
    for *x*
    unfolding *s-def*
    by (*auto intro*!: *derivative-eq-intros*)
  have *Dsc*: *isCont ?Ds x* **by** (*auto intro*!: *continuous-intros*)
  have *nz*: $?Ds\ x\ (f\ x) \neq 0$
    using *assms* **apply** *auto*
    unfolding *transversal-segment-def*
    by (*smt inner-minus-left nrm-reverse open-closed-segment*)

  from *flow-implicit-function-at*[*OF* ‹$x \in X$›, *of s*, *OF* ‹$s\ x = 0$› *Ds Dsc nz* ‹$e >$
0›]
  obtain *t d1* **where** $0 < d1$
    and *t0*: $t\ x = 0$
    and *d1*: ($\bigwedge y.\ y \in cball\ x\ d1 \Longrightarrow s\ (flow0\ y\ (t\ y)) = 0$)
    ($\bigwedge y.\ y \in cball\ x\ d1 \Longrightarrow |t\ y| < e$)
    ($\bigwedge y.\ y \in cball\ x\ d1 \Longrightarrow t\ y \in existence\text{-}ivl0\ y$)
    and *tc*: *continuous-on* (*cball x d1*) *t*
    and t': (*t has-derivative*
      (− *blinfun-inner-left* (*rot* (*b* − *a*)) $/_R$ (*blinfun-inner-left* (*rot* (*b* − *a*))) (*f*
x)))
      (*at x*)
    by *metis*
  from *tc*
  have $t\ {-}x{\rightarrow}\ 0$
    using ‹$0 < d1$›

```
    by (auto simp: continuous-on-def at-within-interior t0 dest!: bspec[where x=x])
  then have ftc: ((λy. flow0 y (t y)) ⟶ x) (at x)
    by (auto intro!: tendsto-eq-intros simp: ‹x ∈ X›)



  define e2 where e2 = min (dist a x) (dist b x)
  have e2 > 0
    using assms
    by (auto simp: e2-def open-segment-def)

  from tendstoD[OF ftc this] have ∀F y in at x. dist (flow0 y (t y)) x < e2 .
  moreover
  let ?S = {x. a · (b − a) < x · (b − a) ∧ x · (b − a) < b · (b − a)}
  have open ?S x ∈ ?S
    using ‹x ∈ {a<−−<b}›
    by (auto simp add: open-segment-line-hyperplanes ‹a ≠ b›
      intro!: open-Collect-conj open-halfspace-component-gt open-halfspace-component-lt)
  from topological-tendstoD[OF ftc this] have ∀F y in at x. flow0 y (t y) ∈ ?S .
  ultimately
  have ∀F y in at x. flow0 y (t y) ∈ ball x e2 ∩ ?S by eventually-elim (auto simp:
dist-commute)
  then obtain d2 where 0 < d2 and ⋀y. x ≠ y ⟹ dist y x < d2 ⟹ flow0 y
(t y) ∈ ball x e2 ∩ ?S
    by (force simp: eventually-at)
  then have d2: dist y x < d2 ⟹ flow0 y (t y) ∈ ball x e2 ∩ ?S for y
    using ‹0 < e2› ‹x ∈ X› t0 ‹x ∈ ?S›
    by (cases y = x) auto

  define d where d = min d1 d2
  have d > 0 using ‹0 < d1› ‹0 < d2› by (simp add: d-def)
  moreover have continuous-on (ball x d) t
    by (auto intro!:continuous-on-subset[OF tc] simp add: d-def)
  moreover
  have ball x e2 ∩ ?S ∩ {x. s x = 0} ⊆ {a<−−<b}
     by (auto simp add: in-open-segment-iff-rot ‹a ≠ b›) (auto simp: s-def e2-def
in-segment)
  then have ⋀y. y ∈ ball x d ⟹ flow0 y (t y) ∈ {a<−−<b}
    apply (rule set-mp)
    using d1 d2 ‹0 < d2›
    by (auto simp: d-def e2-def dist-commute)
  moreover have ⋀y. y ∈ ball x d ⟹ |t y| < e
    using d1 by (auto simp: d-def)
  moreover have continuous-on (ball x d) t
    using tc by (rule continuous-on-subset) (auto simp: d-def)
  moreover have t x = 0 by (simp add: t0)
  ultimately show ?thesis ..
qed
```

```
lemma ω-limit-crossings:
  assumes transversal-segment a b
  assumes pos-ex: {0..} ⊆ existence-ivl0 x
  assumes ω-limit-point x p
  assumes p ∈ {a<−−<b}
  obtains s where
    s ⟶ ∞
    (flow0 x ∘ s) ⟶ p
    ∀F n in sequentially. flow0 x (s n) ∈ {a<−−<b} ∧ s n ∈ existence-ivl0 x
proof −
 from assms have p ∈ X by (auto simp: transversal-segment-def open-closed-segment)
  from assms(3)
  obtain t where
    t ⟶ ∞ (flow0 x ∘ t) ⟶ p
    by (auto simp: ω-limit-point-def)
  note t = ‹t ⟶ ∞› ‹(flow0 x ∘ t) ⟶ p›
  note [tendsto-intros] = t(2)
  from cross-time-continuous[OF assms(1,4) zero-less-one— TODO ??]
  obtain τ δ
    where 0 < δ continuous-on (ball p δ) τ
      τ p = 0 (⋀y. y ∈ ball p δ ⟹ |τ y| < 1)
      (⋀y. y ∈ ball p δ ⟹ flow0 y (τ y) ∈ {a<−−<b})
    by metis
  note τ =
    ‹(⋀y. y ∈ ball p δ ⟹ flow0 y (τ y) ∈ {a<−−<b})›
    ‹(⋀y. y ∈ ball p δ ⟹ |τ y| < 1)›
    ‹continuous-on (ball p δ) τ› ‹τ p = 0›
  define s where s n = t n + τ (flow0 x (t n)) for n
  have ev-in-ball: ∀F n in at-top. flow0 x (t n) ∈ ball p δ
    apply simp
    apply (subst dist-commute)
    apply (rule tendstoD)
     apply (rule t[unfolded o-def])
    apply (rule ‹0 < δ›)
    done
  have filterlim s at-top sequentially
  proof (rule filterlim-at-top-mono)
    show filterlim (λn. −1 + t n) at-top sequentially
      by (rule filterlim-tendsto-add-at-top) (auto intro!: filterlim-tendsto-add-at-top
t)
    from ev-in-ball show ∀F x in sequentially. −1 + t x ≤ s x
      apply eventually-elim
      using τ
      by (force simp : s-def)
  qed
  moreover
  have τ-cont: τ −p→ τ p
    using τ(3) ‹0 < δ›
    by (auto simp: continuous-on-def at-within-ball dest!: bspec[where x=p])
```

note [*tendsto-intros*] = *tendsto-compose-at*[*OF* - *this*, *simplified*]
have *ev1*: $\forall_F$ *n in sequentially*. *t n* > *1*
  using *filterlim-at-top-dense t*(*1*) **by** *auto*
then have *ev-eq*: $\forall_F$ *n in sequentially*. *flow0* ((*flow0 x o t*) *n*) ((τ *o* (*flow0 x o t*)) *n*) = (*flow0 x o s*) *n*
  using *ev-in-ball*
  apply (*eventually-elim*)
  apply (*drule* τ(*2*))
  unfolding *o-def*
  apply (*subst flow-trans*[*symmetric*])
  using *pos-ex*
   apply (*auto simp*: *s-def*)
  apply (*rule existence-ivl-trans'*)
  by *auto*
then
have $\forall_F$ *n in sequentially*.
(*flow0 x o s*) *n* = *flow0* ((*flow0 x o t*) *n*) ((τ *o* (*flow0 x o t*)) *n*)
  by (*simp add*: *eventually-mono*)
from ‹(*flow0 x* ∘ *t*) ⟶ *p*› **and** ‹τ −*p*→ τ *p*›
have
  (λ*n*. *flow0* ((*flow0 x* ∘ *t*) *n*) ((τ ∘ (*flow0 x* ∘ *t*)) *n*))
⟶
flow0 p (τ *p*)
  using ‹τ *p* = *0*› τ*-cont* ‹*p* ∈ *X*›
  by (*intro tendsto-eq-intros*) *auto*
then have (*flow0 x o s*) ⟶ *flow0 p* (τ *p*)
  using *ev-eq* **by** (*rule Lim-transform-eventually*)
then have (*flow0 x o s*) ⟶ *p*
  using ‹*p* ∈ *X*› ‹τ *p* = *0*›
  by *simp*
moreover
{
  have $\forall_F$ *n in sequentially*. *flow0 x* (*s n*) ∈ {*a*<−−<*b*}
   using *ev-eq ev-in-ball*
   apply *eventually-elim*
   apply (*drule sym*)
   apply *simp*
   apply (*rule* τ) **by** *simp*
  moreover have $\forall_F$ *n in sequentially*. *s n* ∈ *existence-ivl0 x*
   using *ev-in-ball ev1*
   apply (*eventually-elim*)
   apply (*drule* τ(*2*))
   using *pos-ex*
   by (*auto simp*: *s-def*)
  ultimately have $\forall_F$ *n in sequentially*. *flow0 x* (*s n*) ∈ {*a*<−−<*b*} ∧ *s n* ∈ *existence-ivl0 x*
   by *eventually-elim auto*
}
ultimately show *?thesis* **..**

qed

lemma *filterlim-at-top-tendstoE*:
  assumes $e > 0$
  assumes *filterlim s at-top sequentially*
  assumes $(flow0\ x \circ s) \longrightarrow u$
  assumes $\forall_F\ n$ *in sequentially.* $P\ (s\ n)$
  obtains m **where** $m > b$ $P\ m$ *dist* $(flow0\ x\ m)\ u < e$
proof −
  from *assms*(*2*) **have** $\forall_F\ n$ *in sequentially.* $b < s\ n$
    by (*simp add*: *filterlim-at-top-dense*)
  moreover have $\forall_F\ n$ *in sequentially.* *norm* $((flow0\ x \circ s)\ n - u) < e$
    using *assms*(*3*)[*THEN tendstoD*, *OF assms*(*1*)] **by** (*simp add*: *dist-norm*)
  moreover note *assms*(*4*)
  ultimately have $\forall_F\ n$ *in sequentially.* $b < s\ n \wedge$ *norm* $((flow0\ x \circ s)\ n - u)$
$< e \wedge P\ (s\ n)$
    by *eventually-elim auto*
  then obtain m **where** $m > b$ $P\ m$ *dist* $(flow0\ x\ m)\ u < e$
    by (*auto simp add*: *eventually-sequentially dist-norm*)
  then show *?thesis* **..**
qed

lemma *open-segment-separate-left*:
  fixes $u\ v\ x\ a\ b$::*'a*
  assumes u:$u \in \{a <\!\!-\!\!-\!\!< b\}$
  assumes v:$v \in \{u <\!\!-\!\!-\!\!< b\}$
  assumes x: *dist* $x\ u <$ *dist* $u\ v$ $x \in \{a <\!\!-\!\!-\!\!< b\}$
  shows $x \in \{a <\!\!-\!\!-\!\!< v\}$
proof −
  have $v \neq x$
    by (*smt dist-commute x*(*1*))
  moreover have $x \notin \{v<\!\!-\!\!-\!\!<b\}$
    by (*smt dist-commute dist-in-open-segment open-segment-subsegment v x*(*1*))
  moreover have $v \in \{a<\!\!-\!\!-\!\!<b\}$ **using** v
   by (*metis ends-in-segment*(*1*) *segment-open-subset-closed subset-eq subset-segment*(*4*)
u)
  ultimately show *?thesis* **using** *open-segment-trichotomy*[*OF - x*(*2*)]
    by *blast*
qed

lemma *open-segment-separate-right*:
  fixes $u\ v\ x\ a\ b$::*'a*
  assumes u:$u \in \{a <\!\!-\!\!-\!\!< b\}$
  assumes v:$v \in \{a <\!\!-\!\!-\!\!< u\}$
  assumes x: *dist* $x\ u <$ *dist* $u\ v$ $x \in \{a <\!\!-\!\!-\!\!< b\}$
  shows $x \in \{v <\!\!-\!\!-\!\!< b\}$
proof −
  have $v \neq x$

```
    by (smt dist-commute x(1))
  moreover have x ∉ {a<−−<v}
   by (smt dist-commute dist-in-open-segment open-segment-commute open-segment-subsegment
v x(1))
  moreover have v ∈ {a<−−<b} using v
   by (metis ends-in-segment(1) segment-open-subset-closed subset-eq subset-segment(4)
u)
  ultimately show ?thesis using open-segment-trichotomy[OF - x(2)]
    by blast
qed

lemma no-two-ω-limit-points:
  assumes transversal: transversal-segment a b
  assumes ex-pos: {0..} ⊆ existence-ivl0 x
  assumes u: ω-limit-point x u u ∈ {a<−−<b}
  assumes v: ω-limit-point x v v ∈ {a<−−<b}
  assumes uv: v ∈ {u<−−<b}
  shows False
proof −
  have unotv: u ≠ v using uv
    using dist-in-open-segment by blast
  define duv where duv = dist u v / 2
  have duv: duv > 0 unfolding duv-def using unotv by simp
  from ω-limit-crossings[OF transversal ex-pos u]
  obtain su where su: filterlim su at-top sequentially
    (flow0 x ∘ su) ⟶ u
    ∀F n in sequentially. flow0 x (su n) ∈ {a<−−<b} ∧ su n ∈ existence-ivl0 x
by blast
  from ω-limit-crossings[OF transversal ex-pos v]
  obtain sv where sv: filterlim sv at-top sequentially
    (flow0 x ∘ sv) ⟶ v
    ∀F n in sequentially. flow0 x (sv n) ∈ {a<−−<b} ∧ sv n ∈ existence-ivl0 x by
blast
  from filterlim-at-top-tendstoE[OF duv su]
  obtain su1 where su1:su1 > 0 flow0 x su1 ∈ {a<−−<b}
    su1 ∈ existence-ivl0 x dist (flow0 x su1) u < duv by auto
  from filterlim-at-top-tendstoE[OF duv sv, of su1]
  obtain su2 where su2:su2 > su1 flow0 x su2 ∈ {a<−−<b}
    su2 ∈ existence-ivl0 x dist (flow0 x su2) v < duv by auto
  from filterlim-at-top-tendstoE[OF duv su, of su2]
  obtain su3 where su3:su3 > su2 flow0 x su3 ∈ {a<−−<b}
    su3 ∈ existence-ivl0 x dist (flow0 x su3) u < duv by auto
  have ∗: su1 ≤ su2 {su1..su2} ⊆ existence-ivl0 x using su1 su2
     apply linarith
    by (metis atLeastatMost-empty-iff empty-iff mvar.closed-segment-subset-domain
real-Icc-closed-segment su1(3) su2(3) subset-eq)


  have d1: dist (flow0 x su1) v ≥ (dist u v)/2 using su1(4) duv unfolding duv-def
```

by (*smt dist-triangle-half-r*)
have *dist* (*flow0 x su1*) *u* < *dist u v* **using** *su1*(*4*) *duv* **unfolding** *duv-def* **by** *linarith*
from *open-segment-separate-left*[*OF u*(*2*) *uv this su1*(*2*)]
have *su1l*:*flow0 x su1* ∈ {*a*<−−<*v*} **.**
have *dist* (*flow0 x su2*) *v* < *dist v* (*flow0 x su1*) **using** *d1*
by (*smt dist-commute duv-def su2*(*4*))
from *open-segment-separate-right*[*OF v*(*2*) *su1l this su2*(*2*)]
have *su2l*:*flow0 x su2* ∈ {*flow0 x su1*<−−<*b*} **.**
then have *su2ll*:*flow0 x su2* ∈ {*u*<−−<*b*}
by (*smt dist-commute dist-pos-lt duv-def open-segment-subsegment pos-half-less open-segment-separate-right su2*(*2*) *su2*(*4*) *u*(*2*) *uv v*(*2*) *unotv*)

have *dist* (*flow0 x su2*) *u* ≥ (*dist u v*)/*2* **using** *su2*(*4*) *duv* **unfolding** *duv-def*
by (*smt dist-triangle-half-r*)
then have *dist* (*flow0 x su3*) *u* < *dist u* (*flow0 x su2*)
by (*smt dist-commute duv-def su3*(*4*))
from *open-segment-separate-left*[*OF u*(*2*) *su2ll this su3*(*2*)]
have *su3l*:*flow0 x su3* ∈ {*a*<−−<*flow0 x su2*} **.**

from *flow0-transversal-segment-monotone*[*OF transversal* ∗ *su1*(*2*) *su2l su3*(*1*) *su3*(*3*)]
have *flow0 x su3* ∉ {*a* <−−<*flow0 x su2*} **.**
thus *False* **using** *su3l* **by** *auto*
qed

6.7 Unique Intersection

Perko Section 3.7 Remark 2

lemma *unique-transversal-segment-intersection*:
assumes *transversal-segment a b*
assumes {*0*..} ⊆ *existence-ivl0 x*
assumes *u* ∈ *ω-limit-set x* ∩ {*a*<−−<*b*}
shows *ω-limit-set x* ∩ {*a*<−−<*b*} = {*u*}
proof (*rule ccontr*)
assume *ω-limit-set x* ∩ {*a*<−−<*b*} ≠ {*u*}
then
obtain *v* **where** *uv*: *u* ≠ *v*
and *v*: *ω-limit-point x v v* ∈ {*a*<−−<*b*} **using** *assms* **unfolding** *ω-limit-set-def*
by *fastforce*
have *u*:*ω-limit-point x u u* ∈ {*a*<−−<*b*} **using** *assms* **unfolding** *ω-limit-set-def*
by *auto*
show *False* **using** *no-two-ω-limit-points*[*OF* ‹*transversal-segment a b*›]
by (*smt dist-commute dist-in-open-segment open-segment-trichotomy u uv v assms*)
qed

Adapted from Perko Section 3.7 Lemma 4 (+ Chicone)

lemma *periodic-imp-ω-limit-set*:

```
  assumes compact K K ⊆ X
  assumes x ∈ X trapped-forward x K
  assumes periodic-orbit y
    flow0 y ‘ UNIV ⊆ ω-limit-set x
  shows flow0 y ‘UNIV = ω-limit-set x
proof (rule ccontr)
  note y = ‹periodic-orbit y› ‹flow0 y ‘ UNIV ⊆ ω-limit-set x›
  from trapped-sol-right[OF assms(1−4)]
  have ex-pos: {0..} ⊆ existence-ivl0 x by blast
  assume flow0 y ‘UNIV ≠ ω-limit-set x
  obtain p where p: p ∈ ω-limit-set x p ∉ flow0 y ‘ UNIV
    using y(2) apply auto
    using ‹range (flow0 y) ≠ ω-limit-set x› by blast
  from ω-limit-set-in-compact-connected[OF assms(1−4)] have
    wcon: connected (ω-limit-set x) .
  from ω-limit-set-invariant have
    invariant (ω-limit-set x) .
  from ω-limit-set-in-compact-compact[OF assms(1−4)] have
    compact (ω-limit-set x) .
  then have sc: seq-compact (ω-limit-set x)
    using compact-imp-seq-compact by blast
  have y1:closed (flow0 y ‘ UNIV)
   using closed-orbit-ω-limit-set periodic-orbit-def ω-limit-set-closed y(1) by auto
  have y2: flow0 y ‘ UNIV ≠ {} by simp
  let ?py = infdist p (range (flow0 y))
  have 0 < ?py
    using y1 y2 p(2)
    by (rule infdist-pos-not-in-closed)
  have ∀ n::nat. ∃ z. z ∈ ω-limit-set x − flow0 y ‘ UNIV ∧
    infdist z (flow0 y ‘ UNIV) < ?py/2^n
  proof (rule ccontr)
    assume ¬ (∀ n. ∃ z. z ∈ ω-limit-set x − range (flow0 y) ∧
             infdist z (range (flow0 y))
             < infdist p (range (flow0 y)) / 2 ^ n)
    then obtain n where n: (∀ z ∈ ω-limit-set x − range (flow0 y).
      infdist z (range (flow0 y)) ≥ ?py / 2 ^ n)
      using not-less by blast
    define A where A = flow0 y ‘ UNIV
    define B where B = {z. infdist z (range (flow0 y)) ≥ ?py / 2 ^ n}
    have Ac:closed A unfolding A-def using y1 by auto
    have Bc:closed B unfolding B-def using infdist-closed by auto
    have A ∩ B = {}
    proof (rule ccontr)
      assume A ∩ B ≠ {}
      then obtain q where q: q ∈ A q ∈ B by blast
      have qz:infdist q (range(flow0 y)) = 0 using q(1) unfolding A-def
        by simp
      note ‹0 < ?py›
      moreover have 2 ^ n > (0::real) by auto
```

ultimately have *infdist p* (*range* (*flow0 y*)) / *2* ^ *n* > (*0*::*real*)
by *simp*
then have *qnz*: *infdist q*(*range* (*flow0 y*)) > *0* **using** *q*(*2*) **unfolding** *B-def*
by *auto*
show *False* **using** *qz qnz* **by** *auto*
qed
then have *a1*:*A* ∩ *B* ∩ *ω-limit-set x* = {} **by** *auto*
have *ω-limit-set x* − *range*(*flow0 y*) ⊆ *B* **using** *n B-def* **by** *blast*
then have *a2*:*ω-limit-set x* ⊆ *A* ∪ *B* **using** *A-def* **by** *auto*
from *connected-closedD*[*OF wcon a1 a2 Ac Bc*]
have *A* ∩ *ω-limit-set x* = {} ∨ *B* ∩ *ω-limit-set x* = {} **.**
moreover {
assume *A* ∩ *ω-limit-set x* = {}
then have *False* **unfolding** *A-def* **using** *y*(*2*) **by** *blast*
}
moreover {
assume *B* ∩ *ω-limit-set x* = {}
then have *False* **unfolding** *B-def* **using** *p*
using *A-def B-def a2* **by** *blast*
}
ultimately show *False* **by** *blast*
qed
then obtain *s* **where** *s*: ∀ *n*::*nat*. (*s*::*nat* ⇒ -) *n* ∈ *ω-limit-set x* − *flow0 y* ‘ *UNIV* ∧
infdist (*s n*) (*flow0 y* ‘ *UNIV*) < *?py*/*2*^*n*
by *metis*
then have ∀ *n*. *s n* ∈ *ω-limit-set x* **by** *blast*
from *seq-compactE*[*OF sc this*]
obtain *l r* **where** *lr*: *l* ∈ *ω-limit-set x strict-mono r* (*s* ∘ *r*) ⟶ *l* **by** *blast*
have ⋀*n*. *infdist* (*s n*) (*range* (*flow0 y*)) ≤ *?py* / *2* ^ *n* **using** *s*
using *less-eq-real-def* **by** *blast*
then have ⋀*n*. *norm*(*infdist* (*s n*) (*range* (*flow0 y*))) ≤ *?py* / *2* ^ *n*
by (*auto simp add*: *infdist-nonneg*)
from *LIMSEQ-norm-0-pow*[*OF* ‹*0* < *?py*› - *this*]
have ((*λz*. *infdist z* (*flow0 y* ‘ *UNIV*)) ∘ *s*) ⟶ *0*
by (*auto simp add*:*o-def*)
from *LIMSEQ-subseq-LIMSEQ*[*OF this lr*(*2*)]
have ((*λz*. *infdist z* (*flow0 y* ‘ *UNIV*)) ∘ (*s* ∘ *r*)) ⟶ *0* **by** (*simp add*: *o-assoc*)
moreover have ((*λz*. *infdist z* (*flow0 y* ‘ *UNIV*)) ∘ (*s* ∘ *r*)) ⟶ *infdist l* (*flow0 y* ‘ *UNIV*)
by (*auto intro*!: *tendsto-eq-intros tendsto-compose-at*[*OF lr*(*3*)])
ultimately have *infdist l* (*flow0 y* ‘*UNIV*) = *0* **using** *LIMSEQ-unique* **by** *auto*
then have *lu*: *l* ∈ *flow0 y* ‘ *UNIV* **using** *in-closed-iff-infdist-zero*[*OF y1 y2*] **by** *auto*
then have *l1*:*l* ∈ *X*
using *closed-orbit-global-existence periodic-orbit-def y*(*1*) **by** *auto*

have *l2*:*f l* ≠ *0*
by (*smt* ‹*l* ∈ *X*› ‹*l* ∈ *range* (*flow0 y*)› *closed-orbit-global-existence fixed-point-imp-closed-orbit-period-zero*(*2*)

fixpoint-sol(*2*) *image-iff local.flows-reverse periodic-orbit-def y*(*1*))
from *transversal-segment-exists*[*OF l1 l2*]
obtain *a b* **where** *ab*: *transversal-segment a b l* ∈ {*a*<−−<*b*} **by** *blast*
then have *l* ∈ *ω-limit-set x* ∩ {*a*<−−<*b*} **using** *lr* **by** *auto*
from *unique-transversal-segment-intersection*[*OF ab*(*1*) *ex-pos this*]
have *luniq*: *ω-limit-set x* ∩ {*a*<−−<*b*} = {*l*} **.**
from *cross-time-continuous*[*OF ab*, *of 1*]
obtain *d t* **where** *dt*: *0* < *d*
(⋀*y*. *y* ∈ *ball l d* ⟹ *flow0 y* (*t y*) ∈ {*a*<−−<*b*})
(⋀*y*. *y* ∈ *ball l d* ⟹ |*t y*| < *1*)
continuous-on (*ball l d*) *t t l* = *0*
by *auto*
obtain *n* **where** (*s* ∘ *r*) *n* ∈ *ball l d* **using** *lr*(*3*) *dt*(*1*) **unfolding** *LIMSEQ-iff-nz*
by (*metis dist-commute mem-ball order-refl*)
then have *flow0* ((*s* ∘ *r*) *n*) (*t* ((*s* ∘ *r*) *n*)) ∈ {*a*<−−<*b*} **using** *dt* **by** *auto*
moreover have *sr*: (*s* ∘ *r*) *n* ∈ *ω-limit-set x* (*s* ∘ *r*) *n* ∉ *flow0 y* ‘ *UNIV*
using *s* **by** *auto*
moreover have *flow0* ((*s* ∘ *r*) *n*) (*t* ((*s* ∘ *r*) *n*)) ∈ *ω-limit-set x*
using ‹*invariant* (*ω-limit-set x*)› *calculation* **unfolding** *invariant-def trapped-def*
by (*smt ω-limit-set-in-compact-subset* ‹*invariant* (*ω-limit-set x*)› *assms*(*1*−*4*)
invariant-def order-trans range-eqI subsetD trapped-iff-on-existence-ivl0 trapped-sol)
ultimately have *flow0* ((*s* ∘ *r*) *n*) (*t* ((*s* ∘ *r*) *n*)) ∈ *ω-limit-set x* ∩ {*a*<−−<*b*}
by *auto*
from *unique-transversal-segment-intersection*[*OF ab*(*1*) *ex-pos this*]
have *flow0* ((*s* ∘ *r*) *n*) (*t* ((*s* ∘ *r*) *n*)) = *l* **using** *luniq* **by** *auto*
then have ((*s* ∘ *r*) *n*) = *flow0 l* (−(*t* ((*s* ∘ *r*) *n*)))
by (*smt UNIV-I* ‹(*s* ∘ *r*) *n* ∈ *ω-limit-set x*› *flows-reverse ω-limit-set-in-compact-existence*
assms(*1*−*4*))
thus *False* **using** *sr*(*2*) *lu*
‹*flow0* ((*s* ∘ *r*) *n*) (*t* ((*s* ∘ *r*) *n*)) = *l*› ‹*flow0* ((*s* ∘ *r*) *n*) (*t* ((*s* ∘ *r*) *n*)) ∈
ω-limit-set x›
closed-orbit-global-existence image-iff local.flow-trans periodic-orbit-def ω-limit-set-in-compact-existence
range-eqI assms y(*1*)
by *smt*
qed

end context *c1-on-open-R2* **begin**

lemma *α-limit-crossings*:
assumes *transversal-segment a b*
assumes *pos-ex*: {..*0*} ⊆ *existence-ivl0 x*
assumes *α-limit-point x p*
assumes *p* ∈ {*a*<−−<*b*}
obtains *s* **where**
s ⟶ −∞
(*flow0 x* ∘ *s*) ⟶ *p*
∀ $_F$ *n in sequentially*.
flow0 x (*s n*) ∈ {*a*<−−<*b*} ∧
s n ∈ *existence-ivl0 x*

proof −
from *pos-ex* **have** $\{0..\} \subseteq$ *uminus* ‘ *existence-ivl0 x* **by** *force*
from *rev.ω-limit-crossings*[*unfolded rev-transversal-segment rev-existence-ivl-eq0 rev-eq-flow*
α-limit-point-eq-rev[*symmetric*], *OF assms*(*1*) *this assms*(*3*,*4*)]
obtain *s* **where** *filterlim s at-top sequentially* $((\lambda t.\ flow0\ x\ (-\ t)) \circ s) \longrightarrow p$
$\forall_F$ *n in sequentially. flow0 x* $(-\ s\ n) \in \{a{<}{-}{-}{<}b\} \wedge s\ n \in$ *uminus* ‘ *existence-ivl0 x* **.**
then have *filterlim* $(-s)$ *at-bot sequentially*
$(flow0\ x \circ (-s)) \longrightarrow p$
$\forall_F$ *n in sequentially. flow0 x* $((-s)\ n) \in \{a{<}{-}{-}{<}b\} \wedge (-s)\ n \in$ *existence-ivl0 x*
by (*auto simp*: *fun-Compl-def o-def filterlim-uminus-at-top*)
then show *?thesis* **..**
qed

If a positive limit point has a regular point in its positive limit set then it is periodic

lemma *ω-limit-point-ω-limit-set-regular-imp-periodic*:
assumes *compact K K* $\subseteq$ *X*
assumes $x \in X$ *trapped-forward x K*
assumes *y*: $y \in$ *ω-limit-set x f y* $\neq 0$
assumes *z*: $z \in$ *ω-limit-set y* $\cup$ *α-limit-set y f z* $\neq 0$
shows *periodic-orbit y* $\wedge$ *flow0 y* ‘ *UNIV* = *ω-limit-set x*
proof −
from *trapped-sol-right*[*OF assms*(*1*−*4*)] **have** *ex-pos*: $\{0..\} \subseteq$ *existence-ivl0 x* **by** *blast*
from *ω-limit-set-in-compact-existence*[*OF assms*(*1*−*4*) *y*(*1*)]
have *yex*: *existence-ivl0 y* = *UNIV* **.**
from *ω-limit-set-invariant*
have *invariant* (*ω-limit-set x*) **.**
then have *yinv*: *flow0 y* ‘ *UNIV* $\subseteq$ *ω-limit-set x* **using** *yex* **unfolding** *invariant-def*
using *trapped-iff-on-existence-ivl0 y*(*1*) **by** *blast*

have *zy*: *ω-limit-point y z* $\vee$ *α-limit-point y z*
using *z* **unfolding** *ω-limit-set-def α-limit-set-def* **by** *auto*

from *ω-limit-set-in-compact-ω-limit-set-contained*[*OF assms*(*1*−*4*)]
ω-limit-set-in-compact-α-limit-set-contained[*OF assms*(*1*−*4*)]
have *zx*:$z \in$ *ω-limit-set x* **using** *zy y*
using *z*(*1*) **by** *blast*
then have $z \in X$
by (*metis UNIV-I local.existence-ivl-initial-time-iff ω-limit-set-in-compact-existence assms*(*1*−*4*))
from *transversal-segment-exists*[*OF this z*(*2*)]
obtain *a b* **where** *ab*: *transversal-segment a b z* $\in \{a{<}{-}{-}{<}b\}$ **by** *blast*

from *zy*

```
 obtain t1 t2 where t1: flow0 y t1 ∈ {a<−−<b} and t2: flow0 y t2 ∈ {a<−−<b}
and t1 ≠ t2
  proof
    assume zy: ω-limit-point y z
    from ω-limit-crossings[OF ab(1) - zy ab(2), unfolded yex]
    obtain s where s: filterlim s at-top sequentially
      (flow0 y ∘ s) ⟶ z
      ∀F n in sequentially. flow0 y (s n) ∈ {a<−−<b}
      by auto
      from eventually-happens[OF this(3)] obtain t1 where t1: flow0 y t1 ∈
{a<−−<b} by auto
    have ∀F n in sequentially. s n > t1
      using filterlim-at-top-dense s(1) by auto
    with s(3) have ∀F n in sequentially. flow0 y (s n) ∈ {a<−−<b} ∧ s n > t1
      by eventually-elim simp
   from eventually-happens[OF this] obtain t2 where t2: flow0 y t2 ∈ {a<−−<b}
and t1 ≠ t2
      by auto
    from t1 this show ?thesis ..
  next
    assume zy: α-limit-point y z
    from α-limit-crossings[OF ab(1) - zy ab(2), unfolded yex]
    obtain s where s: filterlim s at-bot sequentially
      (flow0 y ∘ s) ⟶ z
      ∀F n in sequentially. flow0 y (s n) ∈ {a<−−<b}
      by auto
      from eventually-happens[OF this(3)] obtain t1 where t1: flow0 y t1 ∈
{a<−−<b} by auto
    have ∀F n in sequentially. s n < t1
      using filterlim-at-bot-dense s(1) by auto
    with s(3) have ∀F n in sequentially. flow0 y (s n) ∈ {a<−−<b} ∧ s n < t1
      by eventually-elim simp
   from eventually-happens[OF this] obtain t2 where t2: flow0 y t2 ∈ {a<−−<b}
and t1 ≠ t2
      by auto
    from t1 this show ?thesis ..
  qed
  have flow0 y t1 ∈ ω-limit-set x ∩ {a<−−<b} using t1 UNIV-I yinv by auto
  moreover have flow0 y t2 ∈ ω-limit-set x ∩ {a<−−<b} using t2 UNIV-I yinv
by auto
  ultimately have feq:flow0 y t1 = flow0 y t2
      using unique-transversal-segment-intersection[OF ‹transversal-segment a b›
ex-pos]
    by blast
  have t1 ≠ t2 t1 ∈ existence-ivl0 y t2 ∈ existence-ivl0 y using ‹t1 ≠ t2›
      apply blast
     apply (simp add: yex)
    by (simp add: yex)
  from periodic-orbitI[OF this feq y(2)]
```

```
  have 1: periodic-orbit y .
  from periodic-imp-ω-limit-set[OF assms(1−4) this yinv]
  have 2: flow0 y‘ UNIV = ω-limit-set x .
  show ?thesis using 1 2 by auto
qed
```

6.8 Poincare Bendixson Theorems

Perko Section 3.7 Theorem 1

```
theorem poincare-bendixson:
  assumes compact K K ⊆ X
  assumes x ∈ X trapped-forward x K
  assumes 0 ∉ f ‘ (ω-limit-set x)
  obtains y where periodic-orbit y
    flow0 y ‘ UNIV = ω-limit-set x
proof −
  note f = ‹0 ∉ f ‘ (ω-limit-set x)›
  from ω-limit-set-in-compact-nonempty[OF assms(1−4)]
  obtain y where y: y ∈ ω-limit-set x by fastforce
  from ω-limit-set-in-compact-existence[OF  assms(1−4) y]
  have yex: existence-ivl0 y = UNIV .
  from ω-limit-set-invariant
  have invariant (ω-limit-set x) .
  then have yinv: flow0 y ‘ UNIV ⊆ ω-limit-set x using yex unfolding invari-
ant-def
    using trapped-iff-on-existence-ivl0 y by blast
  from ω-limit-set-in-compact-subset[OF assms(1−4)]
  have ω-limit-set x ⊆ K .
  then have flow0 y ‘ UNIV ⊆ K using yinv by auto
  then have yk:trapped-forward y K
    by (simp add: image-subsetI range-subsetD trapped-forward-def)
  have y ∈ X
    by (simp add: local.mem-existence-ivl-iv-defined(2) yex)

  from ω-limit-set-in-compact-nonempty[OF assms(1−2) this -]
  obtain z where z: z ∈ ω-limit-set y using yk by blast
  from ω-limit-set-in-compact-ω-limit-set-contained[OF assms(1−4)]
  have zx:z ∈ ω-limit-set x using ‹z ∈ ω-limit-set y› y by auto

  have yreg : f y ≠ 0 using f y
    by (metis rev-image-eqI)
  have zreg : f z ≠ 0 using f zx
    by (metis rev-image-eqI)
 from ω-limit-point-ω-limit-set-regular-imp-periodic[OF assms(1−4) y yreg - zreg]
z
  show ?thesis using that by blast
qed

lemma fixed-point-in-ω-limit-set-imp-ω-limit-set-singleton-fixed-point:
```

```
  assumes compact K K ⊆ X
  assumes x ∈ X trapped-forward x K
  assumes fp: yfp ∈ ω-limit-set x f yfp = 0
  assumes zpx: z ∈ ω-limit-set x
  assumes finite-fp: finite {y ∈ K. f y = 0} (is finite ?S)
  shows (∃ p1 ∈ ω-limit-set x. f p1 = 0 ∧ ω-limit-set z = {p1}) ∧
    (∃ p2 ∈ ω-limit-set x. f p2 = 0 ∧ α-limit-set z = {p2})
proof −
  let ?weq = {y ∈ ω-limit-set x. f y = 0}
  from ω-limit-set-in-compact-subset[OF assms(1−4)]
  have wxK: ω-limit-set x ⊆ K .
  from ω-limit-set-in-compact-ω-limit-set-contained[OF assms(1−4)]
  have zx: ω-limit-set z ⊆ ω-limit-set x using zpx by auto
  have zX: z ∈ X using subset-trans[OF wxK assms(2)]
    by (metis subset-iff zpx)
  from ω-limit-set-in-compact-subset[OF assms(1−4)]
  have ?weq ⊆ ?S
    by (smt Collect-mono-iff Int-iff inf.absorb-iff1)
  then have finite ?weq using ‹finite ?S›
    by (blast intro: rev-finite-subset)

  consider f z = 0 | f z ≠ 0 by auto
  then show ?thesis
  proof cases
    assume f z = 0
    from fixed-point-imp-ω-limit-set[OF zX this]
      fixed-point-imp-α-limit-set[OF zX this]
    show ?thesis
      by (metis (mono-tags) ‹f z = 0› zpx)
  next
    assume f z ≠ 0
    have zweq: ω-limit-set z ⊆ ?weq
      apply clarsimp
    proof (rule ccontr)
      fix k assume k: k ∈ ω-limit-set z ¬ (k ∈ ω-limit-set x ∧ f k = 0)
      then have f k ≠ 0 using zx k by auto
       from ω-limit-point-ω-limit-set-regular-imp-periodic[OF assms(1−4) zpx ‹f z
≠ 0› - this] k(1)
      have periodic-orbit z range(flow0 z) = ω-limit-set x by auto
      then have 0 ∉ f ‘ (ω-limit-set x)
        by (metis image-iff periodic-orbit-imp-flow0-regular)
      thus False using fp
        by (metis (mono-tags, lifting) empty-Collect-eq image-eqI)
    qed
    have zweq0: α-limit-set z ⊆ ?weq
      apply clarsimp
    proof (rule ccontr)
      fix k assume k: k ∈ α-limit-set z ¬ (k ∈ ω-limit-set x ∧ f k = 0)
      then have f k ≠ 0 using zx k
```

ω-limit-set-in-compact-α-limit-set-contained[*OF assms*(*1−4*), *of z*] *zpx*
by *auto*
from *ω-limit-point-ω-limit-set-regular-imp-periodic*[*OF assms*(*1−4*) *zpx* ‹*f z* ≠ *0*› *- this*] *k*(*1*)
have *periodic-orbit z range*(*flow0 z*) = *ω-limit-set x* **by** *auto*
then have *0* ∉ *f* ‘ (*ω-limit-set x*)
by (*metis image-iff periodic-orbit-imp-flow0-regular*)
thus *False* **using** *fp*
by (*metis* (*mono-tags*, *lifting*) *empty-Collect-eq image-eqI*)
qed
from *ω-limit-set-in-compact-existence*[*OF assms*(*1−4*) *zpx*]
have *zex*: *existence-ivl0 z* = *UNIV* **.**
from *ω-limit-set-invariant*
have *invariant* (*ω-limit-set x*) **.**
then have *zinv*: *flow0 z* ‘ *UNIV* ⊆ *ω-limit-set x* **using** *zex* **unfolding** *invariant-def*
using *trapped-iff-on-existence-ivl0 zpx* **by** *blast*
then have *flow0 z* ‘ *UNIV* ⊆ *K* **using** *wxK* **by** *auto*
then have *a2*: *trapped-forward z K trapped-backward z K*
using *trapped-def trapped-iff-on-existence-ivl0* **apply** *fastforce*
using ‹*range* (*flow0 z*) ⊆ *K*› *trapped-def trapped-iff-on-existence-ivl0* **by** *blast*
have *a3*: *finite* (*ω-limit-set z*)
by (*metis* ‹*finite ?weq*› *finite-subset zweq*)
from *finite-ω-limit-set-in-compact-imp-unique-fixed-point*[*OF assms*(*1−2*) *zX a2*(*1*) *a3*]
obtain *p1* **where** *p1*: *ω-limit-set z* = {*p1*} *f p1* = *0* **by** *blast*
then have *p1* ∈ *?weq* **using** *zweq* **by** *blast*
moreover
have *finite* (*α-limit-set z*)
by (*metis* ‹*finite ?weq*› *finite-subset zweq0*)
from *finite-α-limit-set-in-compact-imp-unique-fixed-point*[*OF assms*(*1−2*) *zX a2*(*2*) *this*]
obtain *p2* **where** *p2*: *α-limit-set z* = {*p2*} *f p2* = *0* **by** *blast*
then have *p2* ∈ *?weq* **using** *zweq0* **by** *blast*
ultimately show *?thesis*
by (*simp add*: *p1 p2*)
qed
qed

end context *c1-on-open-R2* **begin**

Perko Section 3.7 Theorem 2

theorem *poincare-bendixson-general*:
assumes *compact K K* ⊆ *X*
assumes *x* ∈ *X trapped-forward x K*
assumes *S* = {*y* ∈ *K*. *f y* = *0*} *finite S*
shows
(∃ *y* ∈ *S*. *ω-limit-set x* = {*y*}) ∨
(∃ *y*. *periodic-orbit y* ∧

flow0 y ‘ UNIV = *ω-limit-set x*) ∨
(∃ *P R*. *ω-limit-set x* = *P* ∪ *R* ∧
P ⊆ *S* ∧ *0* ∉ *f* ‘ *R* ∧ *R* ≠ {} ∧
(∀ *z* ∈ *R*.
(∃ *p1* ∈ *P*. *ω-limit-set z* = {*p1*}) ∧
(∃ *p2* ∈ *P*. *α-limit-set z* = {*p2*})))
proof −
note *S* = ‹*S* = {*y* ∈ *K*. *f y* = *0*}›
let *?wreg* = {*y* ∈ *ω-limit-set x*. *f y* ≠ *0*}
let *?weq* = {*y* ∈ *ω-limit-set x*. *f y* = *0*}
have *wreqweq*: *?wreg* ∪ *?weq* = *ω-limit-set x*
by (*smt Collect-cong Collect-disj-eq mem-Collect-eq ω-limit-set-def*)

from *trapped-sol-right*[*OF assms*(*1*−*4*)] **have** *ex-pos*: {*0*..} ⊆ *existence-ivl0 x* **by** *blast*
from *ω-limit-set-in-compact-subset*[*OF assms*(*1*−*4*)]
have *wxK*: *ω-limit-set x* ⊆ *K* **.**
then have *?weq* ⊆ *S* **using** *S*
by (*smt Collect-mono-iff Int-iff inf.absorb-iff1*)
then have *finite ?weq* **using** ‹*finite S*›
by (*metis rev-finite-subset*)
from *ω-limit-set-invariant*
have *xinv*: *invariant* (*ω-limit-set x*) **.**

from *ω-limit-set-in-compact-nonempty*[*OF assms*(*1*−*4*)] *wreqweq*
consider *?wreg* = {} |
?weq = {} |
?weq ≠ {} *?wreg* ≠ {} **by** *auto*
then show *?thesis*
proof *cases*

assume *?wreg* = {}
then have *finite* (*ω-limit-set x*)
by (*metis* (*mono-tags*, *lifting*) ‹{*y* ∈ *ω-limit-set x*. *f y* = *0*} ⊆ *S*› ‹*finite S*› *rev-finite-subset sup-bot.left-neutral wreqweq*)
from *finite-ω-limit-set-in-compact-imp-unique-fixed-point*[*OF assms*(*1*−*4*) *this*]
obtain *y* **where** *y*: *ω-limit-set x* = {*y*} *f y* = *0* **by** *blast*
then have *y* ∈ *S*
by (*metis Un-empty-left* ‹*?weq* ⊆ *S*› ‹*?wreg* = {}› *insert-subset wreqweq*)
then show *?thesis* **using** *y* **by** *auto*
next

assume *?weq* = {}
then have *0* ∉ *f* ‘ *ω-limit-set x*
by (*smt empty-Collect-eq imageE*)
from *poincare-bendixson*[*OF assms*(*1*−*4*) *this*]
have (∃ *y*. *periodic-orbit y* ∧ *flow0 y* ‘ *UNIV* = *ω-limit-set x*)
by *metis*
then show *?thesis* **by** *blast*

next

assume *?weq* $\neq$ {} *?wreg* $\neq$ {}
then obtain *yfp* **where** *yfp*: *yfp* $\in$ *ω-limit-set x f yfp* = *0* **by** *auto*
have *0* $\notin$ *f* ‘ *?wreg* **by** *auto*
have ($\exists$ *p1*$\in$*ω-limit-set x*. *f p1* = *0* $\wedge$ *ω-limit-set z* = {*p1*}) $\wedge$
($\exists$ *p2*$\in$*ω-limit-set x*. *f p2* = *0* $\wedge$ *α-limit-set z* = {*p2*})
if *zpx*: *z* $\in$ *ω-limit-set x* **for** *z*
using *fixed-point-in-ω-limit-set-imp-ω-limit-set-singleton-fixed-point*[
OF assms(*1*−*4*) *yfp zpx* ‹*finite S*›[*unfolded S*]] **by** *auto*
then have *ω-limit-set x* = *?weq* $\cup$ *?wreg* $\wedge$
?weq $\subseteq$ *S* $\wedge$ *0* $\notin$ *f* ‘ *?wreg* $\wedge$ *?wreg* $\neq$ {} $\wedge$
($\forall$ *z* $\in$ *?wreg*.
($\exists$ *p1* $\in$ *?weq*. *ω-limit-set z* = {*p1*}) $\wedge$
($\exists$ *p2* $\in$ *?weq*. *α-limit-set z* = {*p2*}))
using *wreqweq* ‹*?weq* $\subseteq$ *S*› ‹*?wreg* $\neq$ {}› ‹*0* $\notin$ *f* ‘ *?wreg*›
by *blast*
then show *?thesis* **by** *blast*
qed
qed

corollary *poincare-bendixson-applied*:
assumes *compact K K* $\subseteq$ *X*
assumes *K* $\neq$ {} *positively-invariant K*
assumes *0* $\notin$ *f* ‘ *K*
obtains *y* **where** *periodic-orbit y flow0 y* ‘ *UNIV* $\subseteq$ *K*
proof −
from *assms*(*1*−*4*) **obtain** *x* **where** *x* $\in$ *K x* $\in$ *X* **by** *auto*
have ∗: *trapped-forward x K*
using *assms*(*4*) ‹*x* $\in$ *K*›
by (*auto simp*: *positively-invariant-def*)
have *subs*: *ω-limit-set x* $\subseteq$ *K*
by (*rule ω-limit-set-in-compact-subset*[*OF assms*(*1*−*2*) ‹*x* $\in$ *X*› ∗])
with *assms*(*5*) **have** *0* $\notin$ *f* ‘ *ω-limit-set x* **by** *auto*
from *poincare-bendixson*[*OF assms*(*1*−*2*) ‹*x* $\in$ *X*› ∗ *this*]
obtain *y* **where** *periodic-orbit y range* (*flow0 y*) = *ω-limit-set x*
by *force*
then have *periodic-orbit y flow0 y* ‘ *UNIV* $\subseteq$ *K* **using** *subs* **by** *auto*
then show *?thesis* **..**
qed

definition *limit-cycle y* $\longleftrightarrow$
periodic-orbit y $\wedge$
($\exists$ *x*. *x* $\notin$ *flow0 y* ‘ *UNIV* $\wedge$
(*flow0 y* ‘ *UNIV* = *ω-limit-set x* $\vee$ *flow0 y* ‘ *UNIV* = *α-limit-set x*))

corollary *poincare-bendixson-limit-cycle*:
assumes *compact K K* $\subseteq$ *X*

assumes *x* ∈ *K positively-invariant K*
assumes *0* ∉ *f* ‘ *K*
assumes *rev.flow0 x t* ∉ *K*
obtains *y* **where** *limit-cycle y flow0 y* ‘ *UNIV* ⊆ *K*
proof −
have *x* ∈ *X* **using** *assms*(*2*−*3*) **by** *blast*
have ∗: *trapped-forward x K*
using *assms*(*3*−*4*)
by (*auto simp*: *positively-invariant-def*)
have *subs*: *ω-limit-set x* ⊆ *K*
by (*rule ω-limit-set-in-compact-subset*[*OF assms*(*1*−*2*) ‹*x* ∈ *X*› ∗])
with *assms*(*5*) **have** *0* ∉ *f* ‘ *ω-limit-set x* **by** *auto*
from *poincare-bendixson*[*OF assms*(*1*−*2*) ‹*x* ∈ *X*› ∗ *this*]
obtain *y* **where** *y*: *periodic-orbit y range* (*flow0 y*) = *ω-limit-set x*
by *force*
then have *c2*: *flow0 y* ‘ *UNIV* ⊆ *K* **using** *subs* **by** *auto*
have *exy*: *existence-ivl0 y* = *UNIV*
using *closed-orbit-global-existence periodic-orbit-def y*(*1*) **by** *blast*
have *x* ∉ *flow0 y* ‘ *UNIV*
proof *clarsimp*
fix *tt*
assume *x* = *flow0 y tt*
then have *rev.flow0* (*flow0 y tt*) *t* ∉ *K* **using** *assms*(*6*) **by** *auto*
moreover have *rev.flow0* (*flow0 y tt*) *t* ∈ *flow0 y* ‘ *UNIV* **using** *exy* **unfolding** *rev-eq-flow*
using *UNIV-I* ‹*x* = *flow0 y tt*› *closed-orbit-ω-limit-set closed-orbit-flow0 periodic-orbit-def y* **by** *auto*
ultimately show *False* **using** *c2* **by** *blast*
qed
then have *limit-cycle y flow0 y* ‘ *UNIV* ⊆ *K* **using** *y c2* **unfolding** *limit-cycle-def* **by** *auto*
then show *?thesis* **..**
qed

end

end
theory *Affine-Arithmetic-Misc*
imports *HOL*−*ODE*−*Numerics.ODE-Numerics*
begin

7 Branch-And-Bound Arithmetic

primrec *prove-nonneg*::(*nat* ∗ *nat* ∗ *string*) *list* ⇒ *nat* ⇒ *nat* ⇒ *slp* ⇒ *real aform list list* ⇒ *bool* **where**
prove-nonneg prnt 0 p slp X = (*let - = if prnt* ≠ [] *then print* (*STR* ''# *depth limit exceeded*↩'') *else* () *in False*)
| *prove-nonneg prnt* (*Suc i*) *p slp XXS* =
(*case XXS of* [] ⇒ *True* | (*X*#*XS*) ⇒

let RS = approx-slp-outer p 1 slp X
in if RS≠None ∧ Inf-aform′ p (hd (the RS)) ≥ 0
then
let - = if prnt ≠ [] then print (STR ''# Success$\boxed{\hookleftarrow}$'') else ();
- = if prnt ≠ [] then print (String.implode ((shows ''# '' o shows-box-of-aforms-hr X) ''$\boxed{\hookleftarrow}$'')) else ();
- = fold (λ(a, b, c) -. print (String.implode (shows-segments-of-aform a b X c ''$\boxed{\hookleftarrow}$''))) prnt ()
in prove-nonneg prnt i p slp XS
else let - = if prnt ≠ [] then print (STR ''# Split$\boxed{\hookleftarrow}$'') else () in case split-aforms-largest-uncond X of (a, b) ⇒
prove-nonneg prnt i p slp (a#b#XS))

lemma *prove-nonneg-simps*[*simp*]:
prove-nonneg prnt 0 p slp X = False
prove-nonneg prnt (Suc i) p slp XXS =
(*case XXS of* [] ⇒ *True* | (*X#XS*) ⇒
let RS = approx-slp-outer p 1 slp X
in if RS≠None ∧ Inf-aform′ p (hd (the RS)) ≥ 0
then prove-nonneg prnt i p slp XS
else case split-aforms-largest-uncond X of (a, b) ⇒ prove-nonneg prnt i p slp (a#b#XS))
by (*auto simp*: *Let-def split*: *if-splits option.splits list.splits*)

lemmas [*simp del*] = *prove-nonneg.simps*

lemma *split-aforms-lemma*:
fixes *xs*::*real list*
assumes *split-aforms XS i* = (*YS*, *ZS*)
assumes *xs* ∈ *Joints XS*
shows *xs* ∈ *Joints YS* ∪ *Joints ZS*
using *set-rev-mp*[*OF assms*(*2*) *Joints-map-split-aform*[*of XS i*]] *assms*(*1*)
by (*auto simp*: *split-aforms-def o-def*)

lemma *prove-nonneg-empty*[*simp*]: *prove-nonneg prnt* (*Suc i*) *p slp* []
by *simp*

lemma *prove-nonneg-fuel-mono*:
prove-nonneg prnt (*Suc i*) *p* (*slp-of-fas* [*fa*]) *YSS*
if *prove-nonneg prnt i p* (*slp-of-fas* [*fa*]) *YSS*
using *that*
proof (*induction i arbitrary*: *YSS*)
case *0*
then show *?case* **by** *simp*
next
case (*Suc i*)
from *Suc.prems* **show** *?case*
supply [*simp del*] = *prove-nonneg-simps*
apply (*subst prove-nonneg-simps*)

apply (*auto simp*: *Let-def split*: *if-splits option.splits list.splits*)
subgoal apply (*rule Suc.IH*)
apply (*subst* (*asm*) *prove-nonneg-simps*)
by (*auto simp*: *Let-def split*: *if-splits option.splits list.splits*)
subgoal apply (*rule Suc.IH*)
apply (*subst* (*asm*) *prove-nonneg.simps*)
by (*auto simp*: *Let-def split*: *if-splits option.splits list.splits*)
subgoal apply (*rule Suc.IH*)
apply (*subst* (*asm*) *prove-nonneg.simps*)
by (*auto simp*: *Let-def split*: *if-splits option.splits list.splits*)
done
qed

lemma *prove-nonneg-mono*:
prove-nonneg prnt i p (*slp-of-fas* [*fa*]) *YSS* **if** *prove-nonneg prnt i p* (*slp-of-fas* [*fa*]) (*YS # YSS*)
using *that*
proof (*induction i arbitrary*: *YS YSS*)
case *0*
then show *?case* **by** *auto*
next
case (*Suc i*)
from *Suc.prems* **show** *?case*
supply [*simp del*] = *prove-nonneg-simps*
apply (*subst* (*asm*) *prove-nonneg-simps*)
apply (*auto simp*: *Let-def split*: *if-splits option.splits list.splits*)
subgoal by (*rule prove-nonneg-fuel-mono*)
subgoal for *x y* **apply** (*rule prove-nonneg-fuel-mono*)
apply (*rule Suc.IH*[*of y*])
by (*rule Suc.IH*[*of x*])
subgoal for *x y* **apply** (*rule prove-nonneg-fuel-mono*)
apply (*rule Suc.IH*[*of y*])
by (*rule Suc.IH*[*of x*])
done
qed

lemma *prove-nonneg*:
assumes *prove-nonneg prnt i p* (*slp-of-fas* [*fa*]) *XSS*
shows $\forall$ *XS* $\in$ *set XSS*. $\forall$ *xs* $\in$ *Joints XS*. *interpret-floatarith fa xs* $\geq$ *0*
using *assms*
proof (*induction i arbitrary*: *XSS*)
case *0*
then show *?case*
by (*auto*)
next
case (*Suc i*)
show *?case*
proof (*cases XSS*)
case *Nil* **then show** *?thesis* **by** *auto*

```
  next
    case (Cons YS YSS)
    show ?thesis
      unfolding Cons
      apply auto
      subgoal for xs using Suc.prems
        apply (auto simp: Cons Let-def split: if-splits option.splits)
        subgoal for ys
          apply (drule approx-slp-outer-plain)
             apply (rule refl)
            apply force
           apply assumption
          apply simp
          apply (frule Joints-imp-length-eq[where XS=ys])
          apply (auto simp: Suc-length-conv)
          by (smt Inf-aform'-Affine-le)
        subgoal
          apply (simp add: split-aforms-largest-uncond-def split: prod.splits)
          apply (drule Suc.IH)
          apply (drule split-aforms-lemma, assumption)
          by auto
        subgoal
          apply (simp add: split-aforms-largest-uncond-def split: prod.splits)
          apply (drule Suc.IH)
          apply (drule split-aforms-lemma, assumption)
          by auto
        done
      subgoal for XS xs using Suc.prems
        apply (auto simp: Cons Let-def split: if-splits option.splits)
           subgoal for ys by (rule Suc.IH[rule-format], assumption, assumption,
assumption)
        subgoal for ys
          apply (drule prove-nonneg-mono)
          apply (drule prove-nonneg-mono)
          by (rule Suc.IH[rule-format], assumption, assumption, assumption)
        subgoal for ys
          apply (drule prove-nonneg-mono)
          apply (drule prove-nonneg-mono)
          by (rule Suc.IH[rule-format], assumption, assumption, assumption)
        done
      done
  qed
qed

end
```

8 Examples

theory *Examples*

```
  imports Poincare-Bendixson
    HOL−ODE−Numerics.ODE-Numerics
    Affine-Arithmetic-Misc
begin
```

8.1 Simple

```
context
begin
```

coordinate functions

```
definition cx x y = −y + x ∗ (1 − x^2 − y^2)
definition cy x y = x + y ∗ (1 − x^2 − y^2)

lemmas c-defs = cx-def cy-def
```

partial derivatives

```
definition C11::real⇒real⇒real where C11 x y = 1 − 3 ∗ x^2 − y^2
definition C12::real⇒real⇒real where C12 x y = −1 − 2 ∗ x ∗ y
definition C21::real⇒real⇒real where C21 x y = 1 − 2 ∗ x ∗ y
definition C22::real⇒real⇒real where C22 x y = 1 − x^2 − 3 ∗ y^2

lemmas C-partials = C11-def C12-def C21-def C22-def
```

Jacobian as linear map

```
definition C :: real ⇒ real ⇒ (real × real) ⇒L (real × real) where
  C x y = blinfun-of-matrix
    ((λ- -. 0)
      ((1,0) := (λ-. 0)((1, 0):=C11 x y, (0, 1):=C12 x y),
       (0, 1):= (λ-. 0)((1, 0):=C21 x y, (0, 1):=C22 x y)))

lemma C-simp[simp]: blinfun-apply (C x y) (dx, dy) =
  (dx ∗ C11 x y + dy ∗ C12 x y,
   dx ∗ C21 x y + dy ∗ C22 x y)
  by (auto simp: C-def blinfun-of-matrix-apply Basis-prod-def)

lemma C-continuous[continuous-intros]:
  continuous-on S (λx. local.C (f x) (g x))
  if continuous-on S f continuous-on S g
  unfolding C-def
  by (auto intro!: continuous-on-blinfun-of-matrix continuous-intros that
      simp: Basis-prod-def C-partials)

interpretation c: c1-on-open-R2 λ(x::real, y::real). (cx x y, cy x y)::real∗real
  λ(x, y). C x y UNIV
  by unfold-locales
      (auto intro!: derivative-eq-intros ext continuous-intros simp: split-beta alge-
bra-simps
      c-defs C-partials power2-eq-square)
```

definition *trapC* = *cball* (*0*::*real*,*0*::*real*) *2* − *ball* (*0*::*real*,*0*::*real*) (*1*/*2*)

lemma *trapC-eq*:
shows *trapC* = {*p*. (*fst p*)^*2* + (*snd p*)^*2* − *4* ≤ *0*} ∩ {*p*. *1*/*4* − ((*fst p*)^*2* + (*snd p*)^*2*) ≤ *0*}
unfolding *trapC-def*
apply (*auto simp add*: *dist-Pair-Pair*)
using *real-sqrt-le-iff* **apply** *fastforce*
apply (*smt four-x-squared one-le-power real-sqrt-ge-0-iff real-sqrt-pow2*)
using *real-sqrt-le-mono* **apply** *fastforce*
proof −
fix *a* :: *real* **and** *b* :: *real*
assume *a1*: *sqrt* (a^2 + b^2) ∗ *2* < *1*
assume *a2*: *1* ≤ a^2 ∗ *4* + b^2 ∗ *4*
have ∀ *r*. *1* ≤ *sqrt r* ∨ ¬ *1* ≤ *r*
by *simp*
then show *False*
using *a2 a1* **by** (*metis* (*no-types*) *Groups.mult-ac*(*2*) *distrib-left linorder-not-le real-sqrt-four real-sqrt-mult*)
qed

lemma *x-in-trapC*:
shows (*2*,*0*) ∈ *trapC*
unfolding *trapC-def*
by (*auto simp add*: *dist-Pair-Pair*)

lemma *compact-trapC*:
shows *compact trapC*
unfolding *trapC-def*
using *compact-cball compact-diff* **by** *blast*

lemma *nonempty-trapC*:
shows *trapC* ≠ {}
using *x-in-trapC* **by** *auto*

lemma *origin-fixpoint*:
assumes (λ(*x*, *y*). (*cx x y*, *cy x y*)) (*a*,*b*) = *0*
shows *a* = (*0*::*real*) *b* = (*0*::*real*)
using *assms* **unfolding** *cx-def cy-def zero-prod-def* **apply** *auto*
apply (*sos* ((((A<0 ∗ R<1) + ((([28859/65536∗a + 5089/8192∗b + ~1/2] ∗ A=0) + ((([~5089/8192∗a + 17219/65536∗b + ~1/2] ∗ A=1) + (R<1 ∗ ((R<11853/65536 ∗ [~16384/11853∗a^2 + ~11585/11853∗b^2 + 302/1317∗a∗b + a + 1940/3951∗b]^2) + ((R<73630271/776798208 ∗ [a^2 + 64177444/73630271∗b^2 + 44531712/73630271∗a∗b + ~131061126/73630271∗b]^2) + ((R<70211653911/4825433440256 ∗ [~77895776116/70211653911∗b^2 + 5825642465/10030236273∗a∗b + b]^2) + ((R<48375415273/657341564387328 ∗ [~36776393918/48375415273∗b^2 + a∗b]^2) + (R<18852430195/11096159253659648 ∗ [b^2]^2)))))))))) & (((A<0 ∗ (A<0 ∗ R<1)) + (([b] ∗ A=0) + (([~1∗a] ∗ A=1) + (R<1 ∗ (R<1 ∗ [b]^2)))))))

proof −
  assume *a1*: $a * (1 - a^2 - b^2) = b$
  assume *a2*: $a + b * (1 - a^2 - b^2) = 0$
  have *f3*: $\forall r\ ra.\ - (ra::real) * r = ra * - r$
    by *simp*
  have $- b * (1 - a^2 - b^2) = a$
    using *a2* **by** *simp*
  then have $\exists r\ ra.\ b * b - ra * (r * (ra * - r)) = 0$
    using *f3 a1* **by** (*metis* (*no-types*) *c.vec-simps*(*15*) *right-minus-eq*)
  then have $\exists r.\ b * b - r * - r = 0$
    using *f3* **by** (*metis* (*no-types*) *c.vec-simps*(*14*))
  then show $b = 0$
    by *simp*
qed

lemma *origin-not-trapC*:
  shows $0 \notin trapC$
  unfolding *trapC-def zero-prod-def*
  by *auto*

lemma *regular-trapC*:
  shows $0 \notin (\lambda(x, y).\ (cx\ x\ y,\ cy\ x\ y))$ ‘ *trapC*
  using *origin-fixpoint origin-not-trapC*
  by (*smt UNIV-I UNIV-I UNIV-def case-prodE2 imageE c.flow-initial-time-if*
    c.rev.flow-initial-time-if mem-Collect-eq zero-prod-def)

lemma *positively-invariant-outer*:
  shows *c.positively-invariant* $\{p.\ (\lambda p.\ (fst\ p)^2 + (snd\ p)^2 - 4)\ p \leq 0\}$
  apply (*rule c.positively-invariant-le*[*of* λ*p*.−*2*∗((*fst p*)ˆ*2*+(*snd p*)ˆ*2*) - λ*x p*. *2*
∗ *fst x* ∗ *fst p* + *2* ∗ *snd x* ∗ *snd p*])
    apply (*auto intro*!: *continuous-intros derivative-eq-intros*)
  unfolding *cx-def cy-def*
  by (*sos* (((*A*<*0* ∗ *R*<*1*) + (*R*<*1* ∗ ((*R*<*6* ∗ [*a*]ˆ*2*) + (*R*<*6* ∗ [*b*]ˆ*2*))))))

lemma *positively-invariant-inner*:
  shows *c.positively-invariant* $\{p.\ (\lambda p.\ 1/4 - ((fst\ p)^2 + (snd\ p)^2))\ p \leq 0\}$
  apply (*rule c.positively-invariant-le*[*of* λ*p*.−*2*∗((*fst p*)ˆ*2*+(*snd p*)ˆ*2*) - λ*x p*. −
2 ∗ *fst x* ∗ *fst p* − *2* ∗ *snd x* ∗ *snd p*])
    apply (*auto intro*!: *continuous-intros derivative-eq-intros*)
  unfolding *cx-def cy-def*
  by (*sos* (((*A*<*0* ∗ *R*<*1*) + (*R*<*1* ∗ ((*R*<*3/2* ∗ [*a*]ˆ*2*) + (*R*<*3/2* ∗ [*b*]ˆ*2*))))))

lemma *positively-invariant-trapC*:
  shows *c.positively-invariant trapC*
  unfolding *trapC-eq*
  apply (*rule c.positively-invariant-conj*)
  using *positively-invariant-outer*
  apply (*metis* (*no-types*, *lifting*) *Collect-cong case-prodE case-prodI2 case-prod-conv*)

using *positively-invariant-inner*
by (*metis* (*no-types*, *lifting*) *Collect-cong case-prodE case-prodI2 case-prod-conv*)

theorem *c-has-periodic-orbit*:
obtains *y* **where** *c.periodic-orbit y c.flow0 y* ‘ *UNIV* $\subseteq$ *trapC*
proof −
from *c.poincare-bendixson-applied*[*OF compact-trapC - nonempty-trapC positively-invariant-trapC regular-trapC*]
show *?thesis* **using** *that* **by** *blast*
qed

Real-Arithmetic

schematic-goal *c-fas*:
$[-(-(X!1) + (X!0) * (1 - (X!0)\hat{~}2 - (X!1)\hat{~}2)), -((X!0) + (X!1) * (1 - (X!0)\hat{~}2 - (X!1)\hat{~}2))] =$ *interpret-floatariths ?fas X*
by (*reify-floatariths*)

concrete-definition *c-fas* **uses** *c-fas*

interpretation *crev*: *ode-interpretation true-form UNIV c-fas*
−(λ(*x*, *y*). (*cx x y*, *cy x y*)::*real*∗*real*)
d::*2* **for** *d*
by *unfold-locales* (*auto simp*: *c-fas-def less-Suc-eq-0-disj nth-Basis-list-prod Basis-list-real-def*
cx-def cy-def eval-nat-numeral
mk-ode-ops-def eucl-of-list-prod power2-eq-square intro!: *isFDERIV-I*)

lemma *crev*: $t \in \{1/8 .. 1/8\} \longrightarrow (x, y) \in \{(2, 0) .. (2, 0)\} \longrightarrow$
$t \in$ *c.rev.existence-ivl0* $(x, y) \wedge$ *c.rev.flow0* $(x, y)\ t \in \{(5.15, -0.651)..(5.18, -0.647)\}$
by (*tactic* ‹*ode-bnds-tac* @{*thms c-fas-def*} *30 20 7 12* [(*0*, *1*, *0x000000*)] (∗ *crev.out* ∗) @{*context*} *1*›)

theorem *c-has-limit-cycle*:
obtains *y* **where** *c.limit-cycle y range* (*c.flow0 y*) $\subseteq$ *trapC*
proof −
define *E* **where** $E = \{(5.15, -0.651)..(5.18, -0.647)$::*real*∗*real*$\}$
from *crev* **have** *c.rev.flow0* (*2*, *0*) (*1*/*8*) $\in E$
by (*auto simp*: *E-def*)
moreover
have $E \cap$ *trapC* $= \{\}$
proof −
have *norm* $x > 2$ **if** $x \in E$ **for** *x*
using *that*
apply (*auto simp*: *norm-prod-def less-eq-prod-def E-def*)
by (*smt power2-less-eq-zero-iff real-less-rsqrt zero-compare-simps*(*9*))
moreover have *norm* $x \leq 2$ **if** $x \in$ *trapC* **for** *x*
using *that*
by (*auto simp*: *trapC-def dist-prod-def norm-prod-def*)

```
    ultimately show ?thesis by force
  qed
  ultimately have c.rev.flow0 (2, 0) (1 / 8) ∉ trapC by blast
  from c.poincare-bendixson-limit-cycle[OF compact-trapC subset-UNIV x-in-trapC
positively-invariant-trapC regular-trapC this] that
  show ?thesis by blast
qed

end
```

8.2 Glycolysis

Strogatz, Example 7.3.2

```
context
begin
```

coordinate functions

```
definition gx x y = −x + 0.08 ∗ y + x^2 ∗ y
definition gy x y = 0.6 − 0.08 ∗ y − x^2 ∗ y

lemmas g-defs = gx-def gy-def
```

partial derivatives

```
definition A11::real⇒real⇒real where A11 x y = −1 + 2 ∗ x ∗ y
definition A12::real⇒real⇒real where A12 x y = (0.08 + x^2)
definition A21::real⇒real⇒real where A21 x y = −2∗x∗y
definition A22::real⇒real⇒real where A22 x y = −(0.08 + x^2)

lemmas A-partials = A11-def A12-def A21-def A22-def
```

Jacobian as linear map

```
definition A :: real ⇒ real ⇒ (real × real) ⇒_L (real × real) where
  A x y = blinfun-of-matrix
    ((λ- -. 0)
      ((1,0) := (λ-. 0)((1, 0):=A11 x y, (0, 1):=A12 x y),
       (0, 1):= (λ-. 0)((1, 0):=A21 x y, (0, 1):=A22 x y)))

lemma A-simp[simp]: blinfun-apply (A x y) (dx, dy) =
  (dx ∗ A11 x y + dy ∗ A12 x y,
   dx ∗ A21 x y + dy ∗ A22 x y)
  by (auto simp: A-def blinfun-of-matrix-apply Basis-prod-def)

lemma A-continuous[continuous-intros]:
  continuous-on S (λx. local.A (f x) (g x))
  if continuous-on S f continuous-on S g
  unfolding A-def
  by (auto intro!: continuous-on-blinfun-of-matrix continuous-intros that
      simp: Basis-prod-def A-partials)
```

interpretation *g*: *c1-on-open-R2* λ(*x*::*real*, *y*::*real*). (*gx x y*, *gy x y*)::*real*∗*real*
λ(*x*, *y*). *A x y UNIV*
by *unfold-locales*
(*auto intro*!: *derivative-eq-intros ext continuous-intros simp*: *split-beta algebra-simps*
g-defs A-partials)

definition (*pos-quad*::(*real* × *real*) *set*) = {*p* . − *snd p* ≤ *0*} ∩ {*p* . − *fst p* ≤ *0*}

definition (*trapG1*::(*real* × *real*) *set*) = *pos-quad* ∩ ({*p*. (*snd p*) − *751*/*100* ≤ *0*} ∩ {*p*. (*fst p*) + (*snd p*) − *812*/*100* ≤ *0*})

lemma *positively-invariant-y*:
shows *g.positively-invariant* {*p* . − *snd p* ≤ *0*}
apply (*rule g.positively-invariant-le*[*of* λ*p*. −(*0.08* + (*fst p*)^*2*) - λ*x p*. − *snd p*])
apply (*auto intro*!: *continuous-intros derivative-eq-intros*)
unfolding *gy-def*
by (*sos* ())

lemma *positively-invariant-pos-quad*:
shows *g.positively-invariant pos-quad*
unfolding *pos-quad-def*
apply (*rule g.positively-invariant-le-domain*[*OF positively-invariant-y*, *of* λ*p*. *fst p* ∗ *snd p* −*1*])
apply (*auto intro*!: *continuous-intros derivative-eq-intros*)
unfolding *gx-def*
by (*sos* (((*A*<*0* ∗ *R*<*1*) + (((*A*<*0* ∗ *R*<*1*) ∗ (*R*<*11*/*14* ∗ [*1*]^*2*)) + ((*A*<=*0* ∗ *R*<*1*) ∗ (*R*<*1*/*7* ∗ [*1*]^*2*))))))

lemma *positively-invariant-y-upper*:
shows *g.positively-invariant* {*p*. (*snd p*) − *751*/*100* ≤ *0*}
apply (*rule g.positively-invariant-barrier*)
apply (*auto intro*!: *continuous-intros derivative-eq-intros*)
unfolding *gy-def*
by (*sos* ((*R*<*1* + ((*R*<*1* ∗ (*R*<*18775*/*2* ∗ [*a*]^*2*)) + ((*A*<=*0* ∗ *R*<*1*) ∗ (*R*<*1250* ∗ [*1*]^*2*))))))

lemma *arith2*:
shows (*y*::*real*) ≤ *751*/*100* ∧ *x* + (*y*::*real*) = *812*/*100* ⟹ *3*/*5* − (*x*::*real*) < *0*
by *linarith*

lemma *positively-invariant-trapG1*:
shows *g.positively-invariant trapG1*
unfolding *trapG1-def*
apply (*rule g.positively-invariant-conj*[*OF positively-invariant-pos-quad*])
apply (*rule g.positively-invariant-barrier-domain*[*OF positively-invariant-y-upper*])

apply (*auto intro*!: *continuous-intros derivative-eq-intros*)
unfolding *gx-def gy-def* **by** *auto*

definition *p1* (*x*::*real*) (*y*::*real*) = $-(21/34) - (69*x)/38 + (19*x\hat{}2)/15 - (9*x\hat{}3)/28 - (6*x\hat{}4)/43 + (\ 14*y)/29 + (31*x*y)/21 + (182*x\hat{}2*y)/47 - (35*x\hat{}3*y)/16 - (\ 3*y\hat{}2)/17 - (2*x*y\hat{}2)/9 - (31*x\hat{}2*y\hat{}2)/20 + y\hat{}3/102 + (x*y\hat{}3)/59$

definition *p1d x xa* = $38 * (fst\ xa * fst\ x)\ /\ 15 - 69 * fst\ xa\ /\ 38 -$
$27 * (fst\ xa * (fst\ x)^2)\ /\ 28 -$
$24 * (fst\ xa * fst\ x\ \hat{}\ 3)\ /\ 43 +$
$14 * snd\ xa\ /\ 29 +$
$(651 * (fst\ x * snd\ xa) +$
$651 * (fst\ xa * snd\ x))\ /$
$441 +$
$(8554 * ((fst\ x)^2 * snd\ xa) +$
$17108 * (fst\ xa * (fst\ x * snd\ x)))\ /$
$2209 -$
$(560 * (fst\ x\ \hat{}\ 3 * snd\ xa) +$
$1680 * (fst\ xa * ((fst\ x)^2 * snd\ x)))\ /$
$256 -$
$6 * (snd\ xa * snd\ x)\ /\ 17 -$
$(36 * (fst\ x * (snd\ xa * snd\ x)) +$
$18 * (fst\ xa * (snd\ x)^2))\ /$
$81 -$
$(1240 * ((fst\ x)^2 * (snd\ xa * snd\ x)) +$
$1240 * (fst\ xa * (fst\ x * (snd\ x)^2)))\ /$
$400 +$
$snd\ xa * (snd\ x)^2\ /\ 34 +$
$(177 * (fst\ x * (snd\ xa * (snd\ x)^2)) +$
$fst\ xa * snd\ x\ \hat{}\ 3 * 59)\ /$
3481

lemma *p1-has-derivative*:
shows ((λx. *p1* (*fst x*) (*snd x*)) *has-derivative p1d x*) (*at x*)
unfolding *p1-def p1d-def*
by (*auto intro*!: *continuous-intros derivative-eq-intros*)

lemma *p1-not-equil*:
shows $p1\ x\ y \le 0 \Longrightarrow gx\ x\ y \neq 0 \lor gy\ x\ y \neq 0$
unfolding *gx-def gy-def p1-def*
by (*sos* ())

definition *trapG* = *trapG1* $\cap$ {*p*. *p1* (*fst p*) (*snd p*) ≤ 0}

Real-Arithmetic

definition *g-arith a b* = $(-\ (27\ /\ 25) - a^2 + 2 * a * b) * p1\ a\ b - p1d\ (a,\ b)$

$(gx\ a\ b,\ gy\ a\ b)$

schematic-goal *g-arith-fas*:
  [*g-arith* (*X*!*0*) (*X*!*1*)] = *interpret-floatariths ?fas X*
  unfolding *g-arith-def p1-def p1d-def gx-def gy-def fst-conv snd-conv*
  by (*reify-floatariths*)

concrete-definition *g-arith-fas* **uses** *g-arith-fas*

lemma *list-interval2*: *list-interval* [*a*, *b*] [*c*, *d*] = $\{[x,\ y] \mid x\ y.\ x \in \{a\ ..\ c\} \wedge y \in \{b\ ..\ d\}\}$
  apply (*auto simp*: *list-interval-def*)
  subgoal for *x*
    apply (*cases x*)
    apply *auto*
    subgoal for *y zs*
      apply (*cases zs*)
      by *auto*
    done
  done

lemma *g-arith-nonneg*: *g-arith a b* $\geq$ *0*
  if *a*: *0* $\leq$ *a a* $\leq$ *8.24* **and** *b*: *0* $\leq$ *b b* $\leq$ *7.51*
proof −
  have *prove-nonneg* [(*0*, *1*, ''*0x000000*'')] *1000000 30* (*slp-of-fas* [*hd g-arith-fas*]) [*aforms-of-ivls* [*0*, *0*]
    [*float-divr 30 824 100*, *float-divr 30 751 100*]]
    by *eval*— slow: 60s
  from *prove-nonneg*[*OF this*]
  have *0* $\leq$ *interpret-floatarith* (*hd g-arith-fas*) [*a*, *b*]
    apply (*auto simp*: *g-arith-fas*)
    apply (*subst* (*asm*) *Joints-aforms-of-ivls*)
     apply (*auto*)
      apply (*smt divide-nonneg-nonneg float-divr float-numeral rel-simps*(*27*))
     apply (*smt divide-nonneg-nonneg float-divr float-numeral rel-simps*(*27*))
    apply (*subst* (*asm*) *list-interval2*)
    apply *auto*
    apply (*drule spec*[**where** *x*=[*a*, *b*]])
    using *a b*
    apply *auto*
    subgoal by (*rule order-trans*[*OF - float-divr*]) *simp*
    subgoal by (*rule order-trans*[*OF - float-divr*]) *simp*
    done
  also have ... = *g-arith a b*
    by (*auto simp*: *g-arith-fas-def g-arith-def p1-def p1d-def gx-def gy-def*)
  finally show *?thesis* **.**
qed

lemma *trap-arithmetic*:

```
  p1d (a, b) (gx a b, gy a b) ≤ (− (27 / 25) − a^2 + 2 ∗ a ∗ b) ∗ p1 a b if (a, b)
∈ trapG1
proof −
  from that
  have b: 0 ≤ b b ≤ 7.51
    and a: 0 ≤ a a ≤ 8.24
    by (auto simp: trapG1-def pos-quad-def)
  from g-arith-nonneg[OF a b] show ?thesis
    by (simp add: g-arith-def)
qed

lemma positively-invariant-trapG:
  shows g.positively-invariant trapG
  unfolding trapG-def
 apply (rule g.positively-invariant-le-domain[OF positively-invariant-trapG1 - p1-has-derivative,
        of λp. −1.08 − (fst p)^2 + 2 ∗ fst p ∗ snd p])
 subgoal by (auto intro!: continuous-intros derivative-eq-intros simp add: pos-quad-def)
  apply auto
  by (rule trap-arithmetic)

lemma regular-trapG:
  shows 0 ∉ (λ(x, y). (gx x y, gy x y)) ‘ trapG
  unfolding trapG-def apply auto using p1-not-equil
  by force

lemma arith:
  ⋀a b::real. 0 ≤ b =⇒
          0 ≤ a =⇒
          b ∗ 100 ≤ 751 =⇒
          a ∗ 25 + b ∗ 25 ≤ 203 =⇒ norm a + norm b ≤ 20
  by auto

lemma trapG1-subset:
  shows trapG1 ⊆ cball (0::real × real) 20
  unfolding trapG1-def pos-quad-def
  apply auto
  using arith norm-Pair-le
  by smt

lemma compact-subset-closed:
  assumes compact S closed T
  assumes T ⊆ S
  shows compact T
  using compact-Int-closed[OF assms(1−2)] assms(3)
  by (simp add: inf-absorb2)

lemma compact-trapG1:
  shows compact trapG1
  apply (auto intro!: compact-subset-closed[OF - - trapG1-subset])
```

unfolding *trapG1-def pos-quad-def*
by (*auto intro*!: *closed-Collect-le continuous-intros*)

lemma *compact-trapG*:
shows *compact trapG*
unfolding *trapG-def*
by (*auto intro*!: *compact-Int-closed compact-trapG1 closed-Collect-le continuous-intros simp add*: *p1-def*)

lemma *x-in-trapG*:
shows (*1*,*0*) ∈ *trapG*
unfolding *trapG-def trapG1-def pos-quad-def p1-def*
by (*auto simp add*: *dist-Pair-Pair*)

schematic-goal *g-fas*:
[− (− (*X*!*0*) + *8* / *100* ∗ (*X*!*1*) + (*X*!*0*)^*2* ∗ (*X*!*1*)),−(*6* / *10* − *8* / *100* ∗ (*X*!*1*) − (*X*!*0*)^*2* ∗ (*X*!*1*))] = *interpret-floatariths ?fas X*
by (*reify-floatariths*)

concrete-definition *g-fas* **uses** *g-fas*

interpretation *grev*: *ode-interpretation true-form UNIV g-fas*
−(λ(*x*, *y*). (*gx x y*, *gy x y*)::*real*∗*real*)
d::*2* **for** *d*
by *unfold-locales* (*auto simp*: *g-fas-def less-Suc-eq-0-disj nth-Basis-list-prod Basis-list-real-def*
gx-def gy-def eval-nat-numeral
mk-ode-ops-def eucl-of-list-prod power2-eq-square intro!: *isFDERIV-I*)

lemma *grev*: *t* ∈ {*1*/*8* .. *1*/*8*} ⟶ (*x*, *y*) ∈ {(*1*, *0*) .. (*1*, *0*)} ⟶
t ∈ *g.rev.existence-ivl0* (*x*, *y*) ∧ *g.rev.flow0* (*x*, *y*) *t* ∈
{(*1.1*, −*0.09*) .. (*1.2*, −*0.08*)}
by (*tactic* ‹*ode-bnds-tac* @{*thms g-fas-def*} *30 20 7 12* [(*0*, *1*, *0x000000*)] (∗ *grev.out* ∗) @{*context*} *1*›)

theorem *g-has-limit-cycle*:
obtains *y* **where** *g.limit-cycle y range* (*g.flow0 y*) ⊆ *trapG*
proof −
define *E*::(*real*∗*real*) *set* **where** *E* = {(*1.1*, −*0.09*) .. (*1.2*, −*0.08*)}
from *grev* **have** *g.rev.flow0* (*1*, *0*) (*1*/*8*) ∈ *E*
by (*auto simp*: *E-def*)
moreover
have *E* ∩ *trapG* = {}
by (*auto simp*: *trapG-def E-def trapG1-def pos-quad-def*)
ultimately have *g.rev.flow0* (*1*, *0*) (*1* / *8*) ∉ *trapG* **by** *blast*
from *g.poincare-bendixson-limit-cycle*[*OF compact-trapG subset-UNIV x-in-trapG positively-invariant-trapG regular-trapG this*] *that*
show *?thesis* **by** *blast*
qed

end

end