Ordinary Differential Equations

Fabian Immler

March 17, 2025

Abstract

Session `Ordinary-Differential-Equations` formalizes ordinary differential equations (ODEs) and initial value problems. This work comprises proofs for local and global existence of unique solutions (Picard-Lindelöf theorem). Moreover, it contains a formalization of the (continuous or even differentiable) dependency of the flow on initial conditions as the *flow* of ODEs.

Not in the generated document are the following sessions:

- `HOL-ODE-Numerics`: Rigorous numerical algorithms for computing enclosures of solutions based on Runge-Kutta methods and affine arithmetic. Reachability analysis with splitting and reduction at hyperplanes.
- `HOL-ODE-Examples`: Applications of the numerical algorithms to concrete systems of ODEs (e.g., van der Pol and Lorenz attractor).

Contents

1 Auxiliary Lemmas

theory *ODE-Auxiliarities*
imports
  HOL−Analysis.Analysis
  HOL−Library.Float
  List−Index.List-Index
  Affine-Arithmetic.Affine-Arithmetic-Auxiliarities
  Affine-Arithmetic.Executable-Euclidean-Space
begin

instantiation *prod* :: (*zero-neq-one*, *zero-neq-one*) *zero-neq-one*
begin

definition *1* = (*1*, *1*)

instance by *standard* (*simp add*: *zero-prod-def one-prod-def*)
end

1.1 there is no inner product for type $'a \Rightarrow_L\ 'b$

lemma (**in** *real-inner*) *parallelogram-law*: $(norm\ (x + y))^2 + (norm\ (x - y))^2 = 2 * (norm\ x)^2 + 2 * (norm\ y)^2$
proof −
  have $(norm\ (x + y))^2 + (norm\ (x - y))^2 = inner\ (x + y)\ (x + y) + inner\ (x - y)\ (x - y)$
    by (*simp add*: *norm-eq-sqrt-inner*)
  also have ... $= 2 * (norm\ x)^2 + 2 * (norm\ y)^2$
    by (*simp add*: *algebra-simps norm-eq-sqrt-inner*)
  finally show *?thesis* **.**
qed

locale *no-real-inner*
begin

lift-definition *fstzero*::(*real*∗*real*) $\Rightarrow_L$ (*real*∗*real*) **is** λ(*x*, *y*). (*x*, *0*)
  by (*auto intro*!: *bounded-linearI′*)

lemma [*simp*]: *fstzero* (*a*, *b*) = (*a*, *0*)
  by *transfer simp*

lift-definition *zerosnd*::(*real*∗*real*) $\Rightarrow_L$ (*real*∗*real*) **is** λ(*x*, *y*). (*0*, *y*)
  by (*auto intro*!: *bounded-linearI′*)

lemma [*simp*]: *zerosnd* (*a*, *b*) = (*0*, *b*)
  by *transfer simp*

lemma *fstzero-add-zerosnd*: *fstzero* + *zerosnd* = *id-blinfun*
  by *transfer auto*

lemma *norm-fstzero-zerosnd*: *norm fstzero* = *1* *norm zerosnd* = *1* *norm* (*fstzero* − *zerosnd*) = *1*
by (*rule norm-blinfun-eqI*[**where** *x*=(*1*, *0*)]) (*auto simp*: *norm-Pair blinfun.bilinear-simps* *intro*: *norm-blinfun-eqI*[**where** *x*=(*0*, *1*)] *norm-blinfun-eqI*[**where** *x*=(*1*, *0*)])

compare with $(norm\ (?x + ?y))^2 + (norm\ (?x - ?y))^2 = 2 * (norm\ ?x)^2 + 2 * (norm\ ?y)^2$

lemma $(norm\ (fstzero + zerosnd))^2 + (norm\ (fstzero - zerosnd))^2 \neq 2 * (norm\ fstzero)^2 + 2 * (norm\ zerosnd)^2$
by (*simp add*: *fstzero-add-zerosnd norm-fstzero-zerosnd*)

end

1.2 Topology

1.3 Vector Spaces

lemma *ex-norm-eq-1*: ∃ *x*. *norm* (*x*::'*a*::{*real-normed-vector*, *perfect-space*}) = *1*
by (*metis vector-choose-size zero-le-one*)

1.4 Reals

1.5 Balls

sometimes (*?y* ∈ *ball ?x ?e*) = (*dist ?x ?y* < *?e*) etc. are not good [*simp*] rules (although they are often useful): not sure that inequalities are "simpler" than set membership (distorts automatic reasoning when only sets are involved)

lemmas [*simp del*] = *mem-ball mem-cball mem-sphere mem-ball-0 mem-cball-0*

1.6 Boundedness

lemma *bounded-subset-cboxE*:
assumes ⋀*i*. *i* ∈ *Basis* ⟹ *bounded* ((λ*x*. *x* · *i*) ‘ *X*)
obtains *a b* **where** *X* ⊆ *cbox a b*
proof −
have ⋀*i*. *i* ∈ *Basis* ⟹ ∃ *a b*. ((λ*x*. *x* · *i*) ‘ *X*) ⊆ {*a*..*b*}
by (*metis box-real*(*2*) *box-subset-cbox subset-trans bounded-subset-box-symmetric*[*OF assms*])
then obtain *a b* **where** *bnds*: ⋀*i*. *i* ∈ *Basis* ⟹ ((λ*x*. *x* · *i*) ‘ *X*) ⊆ {*a i* .. *b i*}
by *metis*
then have *X* ⊆ {*x*. ∀ *i*∈*Basis*. *x* · *i* ∈ {*a i* .. *b i*}}
by *force*
also have ... = *cbox* (∑ *i*∈*Basis*. *a i* $*_R$ *i*) (∑ *i*∈*Basis*. *b i* $*_R$ *i*)
by (*auto simp*: *cbox-def*)
finally show *?thesis* **..**
qed

lemma
bounded-euclideanI:

assumes $\bigwedge i.\ i \in \mathit{Basis} \Longrightarrow \mathit{bounded}\ ((\lambda x.\ x \cdot i)\ \text{`}\ X)$
shows *bounded X*
proof −
from *bounded-subset-cboxE*[*OF assms*] **obtain** *a b* **where** $X \subseteq \mathit{cbox}\ a\ b$ **.**
with *bounded-cbox* **show** *?thesis* **by** (*rule bounded-subset*)
qed

1.7 Intervals

notation *closed-segment* (‹(1{---})›)
notation *open-segment* (‹(1{-<--<-})›)

lemma *min-zero-mult-nonneg-le*: $0 \le h' \Longrightarrow h' \le h \Longrightarrow \min 0\ (h * k{::}\mathit{real}) \le h' * k$
by (*metis dual-order.antisym le-cases min-le-iff-disj mult-eq-0-iff mult-le-0-iff mult-right-mono-neg*)

lemma *max-zero-mult-nonneg-le*: $0 \le h' \Longrightarrow h' \le h \Longrightarrow h' * k \le \max 0\ (h * k{::}\mathit{real})$
by (*metis dual-order.antisym le-cases le-max-iff-disj mult-eq-0-iff mult-right-mono zero-le-mult-iff*)

lemmas *closed-segment-eq-real-ivl* = *closed-segment-eq-real-ivl*

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

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

1.8 Extended Real Intervals

1.9 Euclidean Components

1.10 Operator Norm

1.11 Limits

lemma *eventually-open-cball*:
assumes *open X*
assumes $x \in X$
shows *eventually* $(\lambda e.\ \mathit{cball}\ x\ e \subseteq X)$ (*at-right 0*)
proof −
from *open-contains-cball-eq*[*OF assms*(*1*)] *assms*(*2*)
obtain *e* **where** $e > 0$ $\mathit{cball}\ x\ e \subseteq X$ **by** *auto*
thus *?thesis*
by (*auto simp*: *eventually-at dist-real-def mem-cball intro*!: *exI*[**where** *x*=*e*])
qed

1.12 Continuity

1.13 Derivatives

lemma
if-eventually-has-derivative:
assumes $(f\ \textit{has-derivative}\ F')\ (\textit{at}\ x\ \textit{within}\ S)$
assumes $\forall_F\ x\ \textit{in at}\ x\ \textit{within}\ S.\ P\ x\ P\ x\ x \in S$
shows $((\lambda x.\ \textit{if}\ P\ x\ \textit{then}\ f\ x\ \textit{else}\ g\ x)\ \textit{has-derivative}\ F')\ (\textit{at}\ x\ \textit{within}\ S)$
using *assms*(*1*)
apply (*rule has-derivative-transform-eventually*)
subgoal using *assms*(*2*) **by** *eventually-elim auto*
by (*auto simp*: *assms*)

lemma *norm-le-in-cubeI*: $\textit{norm}\ x \le \textit{norm}\ y$
if $\bigwedge i.\ i \in \textit{Basis} \Longrightarrow \textit{abs}\ (x \cdot i) \le \textit{abs}\ (y \cdot i)$ **for** $x\ y$
unfolding *norm-eq-sqrt-inner*
apply (*subst euclidean-inner*)
apply (*subst* (*3*) *euclidean-inner*)
using *that*
by (*auto intro*!: *sum-mono simp*: *abs-le-square-iff power2-eq-square*[*symmetric*])

lemma *has-derivative-partials-euclidean-convexI*:
fixes f::'*a*::*euclidean-space* ⇒ '*b*::*real-normed-vector*
assumes f': $\bigwedge i\ x\ xi.\ i \in \textit{Basis} \Longrightarrow (\forall j{\in}\textit{Basis}.\ x \cdot j \in X\ j) \Longrightarrow xi = x \cdot i \Longrightarrow$
$((\lambda p.\ f\ (x + (p - x \cdot i) *_R\ i))\ \textit{has-vector-derivative}\ f'\ i\ x)\ (\textit{at}\ xi\ \textit{within}\ X\ i)$
assumes *df-cont*: $\bigwedge i.\ i \in \textit{Basis} \Longrightarrow (f'\ i \longrightarrow (f'\ i\ x))\ (\textit{at}\ x\ \textit{within}\ \{x.\ \forall j{\in}\textit{Basis}.\ x \cdot j \in X\ j\})$
assumes $\bigwedge i.\ i \in \textit{Basis} \Longrightarrow x \cdot i \in X\ i$
assumes $\bigwedge i.\ i \in \textit{Basis} \Longrightarrow \textit{convex}\ (X\ i)$
shows $(f\ \textit{has-derivative}\ (\lambda h.\ \sum j{\in}\textit{Basis}.\ (h \cdot j) *_R\ f'\ j\ x))\ (\textit{at}\ x\ \textit{within}\ \{x.\ \forall j{\in}\textit{Basis}.\ x \cdot j \in X\ j\})$
(**is** - (*at x within ?S*))
proof (*rule has-derivativeI*)
show *bounded-linear* $(\lambda h.\ \sum j{\in}\textit{Basis}.\ (h \cdot j) *_R\ f'\ j\ x)$
by (*auto intro*!: *bounded-linear-intros*)

obtain *E* **where** [*simp*]: *set E* = (*Basis*::'*a set*) *distinct E*
using *finite-distinct-list*[*OF finite-Basis*] **by** *blast*

have [*simp*]: *length E* = *DIM*('*a*)
using ‹*distinct E*› *distinct-card* **by** *fastforce*
have [*simp*]: *E* ! *j* ∈ *Basis* **if** *j* < *DIM*('*a*) **for** *j*
by (*metis* ‹*length E* = *DIM*('*a*)› ‹*set E* = *Basis*› *nth-mem that*)
have *convex ?S*
by (*rule convex-prod*) (*use assms* **in** *auto*)

have *sum-Basis-E*: *sum g Basis* = (∑*j*<*DIM*('*a*). *g* (*E* ! *j*)) **for** *g*
apply (*rule sum.reindex-cong*[*OF* - - *refl*])

apply (*auto simp*: *inj-on-nth*)
by (*metis* ‹*distinct E*› ‹*length E* = *DIM*(′*a*)› ‹*set E* = *Basis*› *bij-betw-def bij-betw-nth*)

have *segment*: $\forall_F$ *x′ in at x within ?S*. *x′* ∈ *?S* $\forall_F$ *x′ in at x within ?S*. *x′* ≠ *x*
unfolding *eventually-at-filter* **by** *auto*

show ((λ*y*. (*f y* − *f x* − (∑*j*∈*Basis*. ((*y* − *x*) · *j*) $*_R$ *f′ j x*)) $/_R$ *norm* (*y* − *x*)) ⟶ *0*) (*at x within* {*x*. ∀*j*∈*Basis*. *x* · *j* ∈ *X j*})
apply (*rule tendstoI*)
unfolding *norm-conv-dist*[*symmetric*]
proof −
fix *e*::*real*
assume *e* > *0*
define *B* **where** *B* = *e* / *norm* (*2*∗*DIM*(′*a*) + *1*)
with ‹*e* > *0*› **have** *B-thms*: *B* > *0 2* ∗ *DIM*(′*a*) ∗ *B* < *e B* ≥ *0*
by (*auto simp*: *divide-simps algebra-simps B-def*)
define *B′* **where** *B′* = *B* / *2*
have *B′* > *0* **by** (*simp add*: *B′-def* ‹*0* < *B*›)
have ∀ *i* ∈ *Basis*. $\forall_F$ *xa in at x within* {*x*. ∀*j*∈*Basis*. *x* · *j* ∈ *X j*}. *dist* (*f′ i xa*) (*f′ i x*) < *B′*
apply (*rule ballI*)
subgoal premises *prems* **using** *df-cont*[*OF prems*, *THEN tendstoD*, *OF* ‹*0* < *B′*›] **.**
done
from *eventually-ball-finite*[*OF finite-Basis this*]
have $\forall_F$ *x′ in at x within* {*x*. ∀*j*∈*Basis*. *x* · *j* ∈ *X j*}. ∀*j*∈*Basis*. *dist* (*f′ j x′*) (*f′ j x*) < *B′* **.**
then obtain *d* **where** *d* > *0*
and ⋀*x′ j*. *x′* ∈ {*x*. ∀*j*∈*Basis*. *x* · *j* ∈ *X j*} ⟹ *x′* ≠ *x* ⟹ *dist x′ x* < *d* ⟹ *j* ∈ *Basis* ⟹ *dist* (*f′ j x′*) (*f′ j x*) < *B′*
using ‹*0* < *B′*›
by (*auto simp*: *eventually-at*)
then have *B′*: *x′* ∈ {*x*. ∀*j*∈*Basis*. *x* · *j* ∈ *X j*} ⟹ *dist x′ x* < *d* ⟹ *j* ∈ *Basis* ⟹ *dist* (*f′ j x′*) (*f′ j x*) < *B′* **for** *x′ j*
by (*cases x′* = *x*, *auto simp add*: ‹*0* < *B′*›)
then have *B*: *norm* (*f′ j x′* − *f′ j y*) < *B* **if**
(⋀*j*. *j* ∈ *Basis* ⟹ *x′* · *j* ∈ *X j*)
(⋀*j*. *j* ∈ *Basis* ⟹ *y* · *j* ∈ *X j*)
dist x′ x < *d*
dist y x < *d*
j ∈ *Basis*
for *x′ y j*
proof −
have *dist* (*f′ j x′*) (*f′ j x*) < *B′ dist* (*f′ j y*) (*f′ j x*) < *B′*
using *that*
by (*auto intro*!: *B′*)
then have *dist* (*f′ j x′*) (*f′ j y*) < *B′* + *B′* **by** *norm*

also have … = B **by** (*simp add*: *B′-def*)
finally show *?thesis* **by** (*simp add*: *dist-norm*)
qed
have $\forall_F$ *x′ in at x within* {*x*. ∀ *j*∈*Basis*. *x* · *j* ∈ *X j*}. *dist x′ x* < *d*
by (*rule tendstoD*[*OF tendsto-ident-at* ‹*d* > *0*›])
with *segment*
show $\forall_F$ *x′ in at x within* {*x*. ∀ *j*∈*Basis*. *x* · *j* ∈ *X j*}.
norm ((*f x′* − *f x* − (∑ *j*∈*Basis*. ((*x′* − *x*) · *j*) $*_R$ *f′ j x*)) $/_R$ *norm* (*x′* − *x*)) < *e*
proof *eventually-elim*
case (*elim x′*)
then have *os-subset*: *open-segment x x′* ⊆ *?S*
using ‹*convex ?S*› *assms*(*3*)
unfolding *convex-contains-open-segment*
by *auto*
then have *cs-subset*: *closed-segment x x′* ⊆ *?S*
using *elim assms*(*3*) **by** (*auto simp add*: *open-segment-def*)
have *csc-subset*: *closed-segment* (*x′* · *i*) (*x* · *i*) ⊆ *X i* **if** *i*: *i* ∈ *Basis* **for** *i*
apply (*rule closed-segment-subset*)
using *cs-subset elim assms*(*3*,*4*) *that*
by (*auto*)
have *osc-subset*: *open-segment* (*x′* · *i*) (*x* · *i*) ⊆ *X i* **if** *i*: *i* ∈ *Basis* **for** *i*
using *segment-open-subset-closed csc-subset*[*OF i*]
by (*rule order-trans*)

define *h* **where** *h* = *x′* − *x*
define *z* **where** *z j* = (∑ *k*<*j*. (*h* · *E* ! *k*) $*_R$ (*E* ! *k*)) **for** *j*
define *g* **where** *g j t* = (*f* (*x* + *z j* + (*t* − *x* · *E* ! *j*) $*_R$ *E* ! *j*)) **for** *j t*
have *z*: *z j* · *E* ! *j* = *0* **if** *j* < *DIM*(*′a*) **for** *j*
using *that*
by (*auto simp*: *z-def h-def algebra-simps inner-sum-left inner-Basis if-distrib*
nth-eq-iff-index-eq
sum.delta
intro!: *euclidean-eqI*[**where** *′a*=*′a*]
cong: *if-cong*)
from *distinct-Ex1*[*OF* ‹*distinct E*›, *unfolded* ‹*set E* = *Basis*› *Ex1-def* ‹*length E* = -›]
obtain *index* **where**
index: ⋀*i*. *i* ∈ *Basis* ⟹ *i* = *E* ! *index i* ⋀*i*. *i* ∈ *Basis* ⟹ *index i* < *DIM*(*′a*)
and *unique*: ⋀*i j*. *i* ∈ *Basis* ⟹ *j* < *DIM*(*′a*) ⟹ *E* ! *j* = *i* ⟹ *j* = *index i*
by *metis*
have *nth-eq-iff-index*: *E* ! *k* = *i* ⟷ *index i* = *k* **if** *i* ∈ *Basis k* < *DIM*(*′a*) **for** *k i*
using *unique*[*OF that*] *index*[*OF* ‹*i* ∈ *Basis*›]
by *auto*
have *z-inner*: *z j* · *i* = (*if j* ≤ *index i then 0 else h* · *i*) **if** *j* < *DIM*(*′a*) *i* ∈ *Basis* **for** *j i*
using *that index*[*of i*]

by (*auto simp*: *z-def h-def algebra-simps inner-sum-left inner-Basis if-distrib sum.delta nth-eq-iff-index*
intro!: *euclidean-eqI*[**where** *'a='a*]
cong: *if-cong*)
have *z-mem*: $j < DIM('a) \Longrightarrow ja \in Basis \Longrightarrow x \cdot ja + z\ j \cdot ja \in X\ ja$ **for** *j ja*
using *csc-subset*
by (*auto simp*: *z-inner h-def algebra-simps*)
have *norm* $(x' - x) < d$
using *elim* **by** (*simp add*: *dist-norm*)
have *norm-z'*: $y \in$ *closed-segment* $(x \cdot E\ !\ j)\ (x' \cdot E\ !\ j) \Longrightarrow$ *norm* $(z\ j + y *_R E\ !\ j - (x \cdot E\ !\ j) *_R E\ !\ j) < d$
if $j < DIM('a)$
for *j y*
apply (*rule le-less-trans*[*OF* - ‹*norm* $(x' - x) < d$›])
apply (*rule norm-le-in-cubeI*)
apply (*auto simp*: *inner-diff-left inner-add-left inner-Basis that z*)
subgoal by (*auto simp*: *closed-segment-eq-real-ivl split*: *if-splits*)
subgoal for *i*
using *that*
by (*auto simp*: *z-inner h-def algebra-simps*)
done
have *norm-z*: *norm* $(z\ j) < d$ **if** $j < DIM('a)$ **for** *j*
using *norm-z'*[*OF that ends-in-segment*(*1*)]
by (*auto simp*: *z-def*)
{
fix *j*
assume *j*: $j < DIM('a)$
have *eq*: $(x + z\ j + ((y - (x + z\ j)) \cdot E\ !\ j) *_R E\ !\ j +$
$(p - (x + z\ j + ((y - (x + z\ j)) \cdot E\ !\ j) *_R E\ !\ j) \cdot E\ !\ j) *_R$
$E\ !\ j) = (x + z\ j + (p - (x \cdot E\ !\ j)) *_R E\ !\ j)$ **for** *y p*
by (*auto simp*: *algebra-simps j z*)
have *f-has-derivative*: $((\lambda p.\ f\ (x + z\ j + (p - x \cdot E\ !\ j) *_R E\ !\ j))$ *has-derivative* $(\lambda xa.\ xa *_R f'\ (E\ !\ j)\ (x + z\ j + ((y *_R E\ !\ j - (x + z\ j)) \cdot E\ !\ j) *_R E\ !\ j)))$
(*at y within closed-segment* $(x \cdot E\ !\ j)\ (x' \cdot E\ !\ j)$)
if $y \in$ *closed-segment* $(x \cdot E\ !\ j)\ (x' \cdot E\ !\ j)$
for *y*
apply (*rule has-derivative-subset*)
apply (*rule f'*[*unfolded has-vector-derivative-def*,
where $x= x + z\ j + ((y *_R E!j - (x + z\ j)) \cdot E!j) *_R E\ !\ j$ **and** $i=E\ !\ j$, *unfolded eq*])
subgoal by (*simp add*: *j*)
subgoal
using *that*
apply (*auto simp*: *algebra-simps z j inner-Basis*)
using *closed-segment-commute* ‹$E\ !\ j \in Basis$› *csc-subset* **apply** *blast*
by (*simp add*: *z-mem j*)
subgoal by (*auto simp*: *algebra-simps z j inner-Basis*)

subgoal
apply (*auto simp*: *algebra-simps z j inner-Basis*)
using *closed-segment-commute* ‹$\bigwedge j.\ j < DIM('a) \Longrightarrow E\ !\ j \in Basis$› *csc-subset j* **apply** *blast*
done
done
have ∗: $((xa *_R E\ !\ j - (x + z\ j)) \cdot E\ !\ j) = xa - x \cdot E\ !\ j$ **for** *xa*
by (*auto simp*: *algebra-simps z j*)
have g': (*g j has-vector-derivative* (f' (*E* ! *j*) $(x + z\ j + (xa - x \cdot E\ !\ j) *_R E\ !\ j)$))
(*at xa within* (*closed-segment* ($x \cdot E!j$) ($x' \cdot E!j$)))
(**is** (- *has-vector-derivative* ?g' *j xa*) -)
if $xa \in$ *closed-segment* ($x \cdot E!j$) ($x' \cdot E!j$) **for** *xa*
using *that*
by (*auto simp*: *has-vector-derivative-def g-def*[*abs-def*] ∗
intro!: *derivative-eq-intros f-has-derivative*[*THEN has-derivative-eq-rhs*])
define g' **where** $g'\ j$ = ?$g'\ j$ **for** *j*
with g' **have** g': (*g j has-vector-derivative* g' *j t*) (*at t within closed-segment* ($x \cdot E!j$) ($x' \cdot E!j$))
if $t \in$ *closed-segment* ($x \cdot E!j$) ($x' \cdot E!j$)
for *t*
by (*simp add*: *that*)
have *cont-bound*: $\bigwedge y.\ y \in$*closed-segment* $(x \cdot E\ !\ j)\ (x' \cdot E\ !\ j) \Longrightarrow$ *norm* $(g'\ j\ y - g'\ j\ (x \cdot E\ !\ j)) \le B$
apply (*auto simp add*: g'-*def j algebra-simps inner-Basis z dist-norm*
intro!: *less-imp-le B z-mem norm-z norm-z'*)
using *closed-segment-commute* ‹$\bigwedge j.\ j < DIM('a) \Longrightarrow E\ !\ j \in Basis$› *csc-subset j* **apply** *blast*
done
from *vector-differentiable-bound-linearization*[*OF* g' *order-refl cont-bound ends-in-segment*(*1*)]
have *n*: *norm* $(g\ j\ (x' \cdot E\ !\ j) - g\ j\ (x \cdot E\ !\ j) - (x' \cdot E\ !\ j - x \cdot E\ !\ j) *_R g'\ j\ (x \cdot E\ !\ j)) \le$ *norm* $(x' \cdot E\ !\ j - x \cdot E\ !\ j) * B$
.
have $z\ (Suc\ j) = z\ j + (x' \cdot E\ !\ j - x \cdot E\ !\ j) *_R E\ !\ j$
by (*auto simp*: *z-def h-def algebra-simps*)
then have $f\ (x + z\ (Suc\ j)) = f\ (x + z\ j + (x' \cdot E\ !\ j - x \cdot E\ !\ j) *_R E\ !\ j)$
by (*simp add*: *ac-simps*)
with *n* **have** *norm* $(f\ (x + z\ (Suc\ j)) - f\ (x + z\ j) - (x' \cdot E\ !\ j - x \cdot E\ !\ j) *_R f'\ (E\ !\ j)\ (x + z\ j)) \le |x' \cdot E\ !\ j - x \cdot E\ !\ j| * B$
by (*simp add*: *g-def* g'-*def*)
} **note** *B-le* = *this*
have B': *norm* $(f'\ (E\ !\ j)\ (x + z\ j) - f'\ (E\ !\ j)\ x) \le B$ **if** $j < DIM('a)$ **for** *j*
using *that assms*(*3*)
by (*auto simp add*: *algebra-simps inner-Basis z dist-norm* ‹$0 < d$›
intro!: *less-imp-le B z-mem norm-z*)
have $(\sum j \le DIM('a) - 1.\ f\ (x + z\ j) - f\ (x + z\ (Suc\ j))) = f\ (x + z\ 0) - f\ (x + z\ (Suc\ (DIM('a) - 1)))$

by (*rule sum-telescope*)
moreover have *z DIM('a) = h*
using *index*
by (*auto simp*: *z-def h-def algebra-simps inner-sum-left inner-Basis if-distrib*
nth-eq-iff-index
sum.delta
intro!: *euclidean-eqI*[**where** *'a='a*]
cong: *if-cong*)
moreover have *z 0 = 0*
by (*auto simp*: *z-def*)
moreover have {..*DIM('a) − 1*} = {..<*DIM('a)*}
using *le-imp-less-Suc* **by** *fastforce*
ultimately have *f x − f (x + h) = (∑j<DIM('a). f (x + z j) − f (x + z (Suc j)))*
by *auto*
then have *norm (f (x + h) − f x − (∑j∈Basis. ((x' − x) · j) *ₐ f' j x)) =*
norm(
(∑*j<DIM('a). f (x + z (Suc j)) − f (x + z j) − (x' · E ! j − x · E ! j) $*_R$ f' (E ! j) (x + z j)) +*
(∑*j<DIM('a). (x' · E ! j − x · E ! j) $*_R$ (f' (E ! j) (x + z j) − f' (E ! j) x)))*
(**is** - = *norm (sum ?a ?E + sum ?b ?E)*)
by (*intro arg-cong*[**where** *f=norm*]) (*simp add*: *sum-negf sum-subtractf sum.distrib algebra-simps sum-Basis-E*)
also have ... ≤ *norm (sum ?a ?E) + norm (sum ?b ?E)* **by** (*norm*)
also have *norm (sum ?a ?E) ≤ sum (λx. norm (?a x)) ?E*
by (*rule norm-sum*)
also have ... ≤ *sum (λj. norm |x' · E ! j − x · E ! j| * B) ?E*
by (*auto intro*!: *sum-mono B-le*)
also have ... ≤ *sum (λj. norm (x' − x) * B) ?E*
apply (*rule sum-mono*)
apply (*auto intro*!: *mult-right-mono* ‹*0 ≤ B*›)
by (*metis (full-types)* ‹⋀*j. j < DIM('a) ⟹ E ! j ∈ Basis*› *inner-diff-left norm-bound-Basis-le order-refl*)
also have ... = *norm (x' − x) * DIM('a) * B*
by *simp*
also have *norm (sum ?b ?E) ≤ sum (λx. norm (?b x)) ?E*
by (*rule norm-sum*)
also have ... ≤ *sum (λj. norm (x' − x) * B) ?E*
apply (*intro sum-mono*)
apply (*auto intro*!: *mult-mono B'*)
apply (*metis (full-types)* ‹⋀*j. j < DIM('a) ⟹ E ! j ∈ Basis*› *inner-diff-left norm-bound-Basis-le order-refl*)
done
also have ... = *norm (x' − x) * DIM('a) * B*
by *simp*
finally have *norm (f (x + h) − f x − (∑j∈Basis. ((x' − x) · j) $*_R$ f' j x))* ≤
*norm (x' − x) * real DIM('a) * B + norm (x' − x) * real DIM('a) * B*

by *arith*
also have ... $/_R$ *norm* $(x' - x) \leq$ *norm* $(2 * DIM('a) * B)$
using ‹$B \geq 0$›
by (*simp add*: *divide-simps abs-mult*)
also have ... $< e$ **using** *B-thms* **by** *simp*
finally show *?case* **by** (*auto simp*: *divide-simps abs-mult h-def*)
qed
qed
qed

lemma
frechet-derivative-equals-partial-derivative:
fixes f::$'a$::*euclidean-space* $\Rightarrow$ $'a$
assumes *Df*: $\bigwedge x$. (f *has-derivative Df* x) (*at* x)
assumes f': ((λp. f $(x + (p - x \cdot i) *_R i) \cdot b$) *has-real-derivative* f' x i b) (*at* $(x \cdot i)$)
shows *Df* x $i \cdot b = f'$ x i b
proof −
define *Dfb* **where** *Dfb* x = *Blinfun* (*Df* x) **for** x
have *Dfb-apply*: *blinfun-apply* (*Dfb* x) = *Df* x **for** x
unfolding *Dfb-def*
apply (*rule bounded-linear-Blinfun-apply*)
apply (*rule has-derivative-bounded-linear*)
apply (*rule assms*)
done
have *Dfb* x = *blinfun-of-matrix* (λi b. *Dfb* x $b \cdot i$) **for** x
using *blinfun-of-matrix-works*[*of Dfb x*] **by** *auto*
have *Dfb*: $\bigwedge x$. (f *has-derivative Dfb* x) (*at* x)
by (*auto simp*: *Dfb-apply Df*)
note [*derivative-intros*] = *diff-chain-at*[*OF - Dfb*, *unfolded o-def*]
have ((λp. f $(x + (p - x \cdot i) *_R i) \cdot b$) *has-real-derivative Dfb* x $i \cdot b$) (*at* $(x \cdot i)$)
by (*auto intro*!: *derivative-eq-intros ext simp*: *has-field-derivative-def blinfun.bilinear-simps*)
from *DERIV-unique*[*OF* f' *this*]
show *?thesis* **by** (*simp add*: *Dfb-apply*)
qed

1.14 Integration

lemmas *content-real*[*simp*]
lemmas *integrable-continuous*[*intro*, *simp*]
and *integrable-continuous-real*[*intro*, *simp*]

lemma *integral-eucl-le*:
fixes f g::$'a$::*euclidean-space* $\Rightarrow$ $'b$::*ordered-euclidean-space*
assumes f *integrable-on* s
and g *integrable-on* s
and $\bigwedge x$. $x \in s \Longrightarrow f$ $x \leq g$ x

```
  shows integral s f ≤ integral s g
  using assms
  by (auto intro!: integral-component-le simp: eucl-le[where 'a='b])

lemma
  integral-ivl-bound:
  fixes l u::'a::ordered-euclidean-space
  assumes ⋀x h'. h' ∈ {t0 .. h} ⟹ x ∈ {t0 .. h} ⟹ (h' − t0) *R f x ∈ {l .. u}
  assumes t0 ≤ h
  assumes f-int: f integrable-on {t0 .. h}
  shows integral {t0 .. h} f ∈ {l .. u}
proof −
  from assms(1)[of t0 t0] assms(2) have 0 ∈ {l .. u} by auto
  have integral {t0 .. h} f = integral {t0 .. h} (λt. if t ∈ {t0, h} then 0 else f t)
    by (rule integral-spike[where S={t0, h}]) auto
  also
  {
    fix x
    assume 1: x ∈ {t0 <..< h}
    with assms have (h − t0) *R f x ∈ {l .. u} by auto
    then have (if x ∈ {t0, h} then 0 else f x) ∈ {l /R (h − t0) .. u /R (h − t0)}
      using ‹x ∈ -›
      by (auto simp: inverse-eq-divide
        simp: eucl-le[where 'a='a] field-simps algebra-simps)
  }
  then have . . . ∈ {integral {t0..h} (λ-. l /R (h − t0)) .. integral {t0..h} (λ-. u
/R (h − t0))}
    unfolding atLeastAtMost-iff
    apply (safe intro!: integral-eucl-le)
    using ‹0 ∈ {l .. u}›
    apply (auto intro!: assms
      intro: integrable-continuous-real  integrable-spike[where S={t0, h}, OF f-int]
      simp: eucl-le[where 'a='a] divide-simps
      split: if-split-asm)
    done
  also have . . . ⊆ {l .. u}
    using assms ‹0 ∈ {l .. u}›
    by (auto simp: inverse-eq-divide)
  finally show ?thesis .
qed

lemma
  add-integral-ivl-bound:
  fixes l u::'a::ordered-euclidean-space
  assumes ⋀x h'. h' ∈ {t0 .. h} ⟹ x ∈ {t0 .. h} ⟹ (h' − t0) *R f x ∈ {l − x0
.. u − x0}
  assumes t0 ≤ h
  assumes f-int: f integrable-on {t0 .. h}
  shows x0 + integral {t0 .. h} f ∈ {l .. u}
```

using *integral-ivl-bound*[*OF assms*]
by (*auto simp*: *algebra-simps*)

1.15 conditionally complete lattice

1.16 Lists

lemma
Ball-set-Cons[*simp*]: (∀ *a*∈*set-Cons x y*. *P a*) ⟷ (∀ *a*∈*x*. ∀ *b*∈*y*. *P* (*a*#*b*))
by (*auto simp*: *set-Cons-def*)

lemma *set-cons-eq-empty*[*iff*]: *set-Cons a b* = {} ⟷ *a* = {} ∨ *b* = {}
by (*auto simp*: *set-Cons-def*)

lemma *listset-eq-empty-iff*[*iff*]: *listset XS* = {} ⟷ {} ∈ *set XS*
by (*induction XS*) *auto*

lemma *sing-in-sings*[*simp*]: [*x*] ∈ (λ*x*. [*x*]) ' *xd* ⟷ *x* ∈ *xd*
by *auto*

lemma *those-eq-None-set-iff*: *those xs* = *None* ⟷ *None* ∈ *set xs*
by (*induction xs*) (*auto split*: *option.split*)

lemma *those-eq-Some-lengthD*: *those xs* = *Some ys* ⟹ *length xs* = *length ys*
by (*induction xs arbitrary*: *ys*) (*auto split*: *option.splits*)

lemma *those-eq-Some-map-Some-iff*: *those xs* = *Some ys* ⟷ (*xs* = *map Some ys*) (**is** *?l* ⟷ *?r*)
proof *safe*
assume *?l*
then have *length xs* = *length ys*
by (*rule those-eq-Some-lengthD*)
then show *?r* **using** ‹*?l*›
by (*induction xs ys rule*: *list-induct2*) (*auto split*: *option.splits*)
next
assume *?r*
then have *length xs* = *length ys*
by *simp*
then show *those* (*map Some ys*) = *Some ys* **using** ‹*?r*›
by (*induction xs ys rule*: *list-induct2*) (*auto split*: *option.splits*)
qed

1.17 Set(sum)

1.18 Max

1.19 Uniform Limit

1.20 Bounded Linear Functions

lift-definition *comp3*::— TODO: name?

$('c::real\text{-}normed\text{-}vector \Rightarrow_L 'd::real\text{-}normed\text{-}vector) \Rightarrow ('b::real\text{-}normed\text{-}vector \Rightarrow_L 'c) \Rightarrow_L 'b \Rightarrow_L 'd$ **is**
$\lambda(cd::('c \Rightarrow_L 'd))\ (bc::'b \Rightarrow_L 'c).\ (cd\ o_L\ bc)$
by (*rule bounded-bilinear.bounded-linear-right*[*OF bounded-bilinear-blinfun-compose*])

lemma *blinfun-apply-comp3*[*simp*]: *blinfun-apply* (*comp3 a*) *b* = (*a* o_L *b*)
by (*simp add*: *comp3.rep-eq*)

lemma *bounded-linear-comp3*[*bounded-linear*]: *bounded-linear comp3*
by *transfer* (*rule bounded-bilinear-blinfun-compose*)

lift-definition *comp12*::— TODO: name?
$('a::real\text{-}normed\text{-}vector \Rightarrow_L 'c::real\text{-}normed\text{-}vector) \Rightarrow ('b::real\text{-}normed\text{-}vector \Rightarrow_L 'c) \Rightarrow ('a \times 'b) \Rightarrow_L 'c$
is $\lambda f\ g\ (a,\ b).\ f\ a + g\ b$
by (*auto intro*!: *bounded-linear-intros*
intro: *bounded-linear-compose*
simp: *split-beta'*)

lemma *blinfun-apply-comp12*[*simp*]: *blinfun-apply* (*comp12 f g*) *b* = *f* (*fst b*) + *g* (*snd b*)
by (*simp add*: *comp12.rep-eq split-beta*)

1.21 Order Transitivity Attributes

attribute-setup *le* = ‹*Scan.succeed* (*Thm.rule-attribute* [] (*fn context* => *fn thm* => *thm RS* @{*thm order-trans*}))›
transitive version of inequality (*useful for intro*)
attribute-setup *ge* = ‹*Scan.succeed* (*Thm.rule-attribute* [] (*fn context* => *fn thm* => *thm RS* @{*thm order-trans*[*rotated*]}))›
transitive version of inequality (*useful for intro*)

1.22 point reflection

definition *preflect*::'*a*::*real-vector* ⇒ '*a* ⇒ '*a* **where** *preflect* ≡ $\lambda t0\ t.\ 2 *_R t0 - t$

lemma *preflect-preflect*[*simp*]: *preflect t0* (*preflect t0 t*) = *t*
by (*simp add*: *preflect-def algebra-simps*)

lemma *preflect-preflect-image*[*simp*]: *preflect t0* ‘ *preflect t0* ‘ *S* = *S*
by (*simp add*: *image-image*)

lemma *is-interval-preflect*[*simp*]: *is-interval* (*preflect t0* ‘ *S*) ⟷ *is-interval S*
by (*auto simp*: *preflect-def*[*abs-def*])

lemma *iv-in-preflect-image*[*intro*, *simp*]: *t0* ∈ *T* ⟹ *t0* ∈ *preflect t0* ‘ *T*
by (*auto intro*!: *image-eqI simp*: *preflect-def algebra-simps scaleR-2*)

lemma *preflect-tendsto*[*tendsto-intros*]:
fixes *l*::'*a*::*real-normed-vector*

shows $(g \longrightarrow l)\ F \Longrightarrow (h \longrightarrow m)\ F \Longrightarrow ((\lambda x.\ \textit{preflect}\ (g\ x)\ (h\ x)) \longrightarrow \textit{preflect}\ l\ m)\ F$
by (*auto intro*!: *tendsto-eq-intros simp*: *preflect-def*)

lemma *continuous-preflect*[*continuous-intros*]:
fixes *a*::'*a*::*real-normed-vector*
shows *continuous* (*at a within A*) (*preflect t0*)
by (*auto simp*: *continuous-within intro*!: *tendsto-intros*)

lemma
fixes *t0*::'*a*::*ordered-real-vector*
shows *preflect-le*[*simp*]: $t0 \le \textit{preflect}\ t0\ b \longleftrightarrow b \le t0$
and *le-preflect*[*simp*]: $\textit{preflect}\ t0\ b \le t0 \longleftrightarrow t0 \le b$
and *antimono-preflect*: *antimono* (*preflect t0*)
and *preflect-le-preflect*[*simp*]: $\textit{preflect}\ t0\ a \le \textit{preflect}\ t0\ b \longleftrightarrow b \le a$
and *preflect-eq-cancel*[*simp*]: $\textit{preflect}\ t0\ a = \textit{preflect}\ t0\ b \longleftrightarrow a = b$
by (*auto intro*!: *antimonoI simp*: *preflect-def scaleR-2*)

lemma *preflect-eq-point-iff*[*simp*]: $t0 = \textit{preflect}\ t0\ s \longleftrightarrow t0 = s$ $\textit{preflect}\ t0\ s = t0 \longleftrightarrow t0 = s$
by (*auto simp*: *preflect-def algebra-simps scaleR-2*)

lemma *preflect-minus-self*[*simp*]: $\textit{preflect}\ t0\ s - t0 = t0 - s$
by (*simp add*: *preflect-def scaleR-2*)

end
theory *MVT-Ex*
imports
HOL−Analysis.Analysis
HOL−Decision-Procs.Approximation
../ODE-Auxiliarities
begin

1.23 (Counter)Example of Mean Value Theorem in Euclidean Space

There is no exact analogon of the mean value theorem in the multivariate case!

lemma *MVT-wrong*: **assumes**
$\bigwedge J\ a\ u\ (f::real*real \Rightarrow real*real).$
$(\bigwedge x.\ \textit{FDERIV}\ f\ x :> J\ x) \Longrightarrow$
$(\exists t \in \{0<..<1\}.\ f\ (a + u) - f\ a = J\ (a + t *_R u)\ u)$
shows *False*
proof −
have $\bigwedge t::real*real.\ \textit{FDERIV}\ (\lambda t.\ (\cos\ (\textit{fst}\ t),\ \sin\ (\textit{fst}\ t)))\ t :> (\lambda h.\ (-\ ((\textit{fst}\ h) * \sin\ (\textit{fst}\ t)),\ (\textit{fst}\ h) * \cos\ (\textit{fst}\ t))))$
by (*auto intro*!: *derivative-eq-intros*)
from *assms*[*OF this*, *of* (*pi*, *pi*) (*pi*, *pi*)] **obtain** *t*::*real* **where** *t*: $0 < t$ $t < 1$

and
pi ∗ sin (t ∗ pi) = 2 cos (t ∗ pi) = 0
by *auto*
then obtain *n* **where** *tpi*: *t ∗ pi = real-of-int n ∗ (pi / 2)* **and** *odd n*
by (*auto simp*: *cos-zero-iff-int*)
then have *teq*: *t = real-of-int n / 2* **by** *auto*
then have *n = 1* **using** *t ‹odd n›* **by** *arith*
then have *t = 1/2* **using** *teq* **by** *simp*
have *sin (t ∗ pi) = 1*
by (*simp add*: *‹t = 1/2› sin-eq-1*)
with *‹pi ∗ sin (t ∗ pi) = 2›*
have *pi = 2* **by** *simp*
moreover have *pi > 2* **using** *pi-approx* **by** *simp*
ultimately show *False* **by** *simp*
qed

lemma *MVT-corrected*:
fixes *f::'a::ordered-euclidean-space⇒'b::euclidean-space*
assumes *fderiv*: *⋀x. x ∈ D ⟹ (f has-derivative J x) (at x within D)*
assumes *line-in*: *⋀x. ⟦0 ≤ x; x ≤ 1⟧ ⟹ a + x ∗R u ∈ D*
shows *(∃ t∈Basis→{0<..<1}. (f (a + u) − f a) = (∑ i∈Basis. (J (a + t i ∗R u) u · i) ∗R i))*
proof −
{
fix *i::'b*
assume *i ∈ Basis*
have *subset*: *((λx. a + x ∗R u) ‘ {0..1}) ⊆ D*
using *line-in* **by** *force*
have *⋀x. ⟦0 ≤ x; x ≤ 1⟧ ⟹ ((λb. f (a + b ∗R u) · i) has-derivative (λb. b ∗R J (a + x ∗R u) u · i)) (at x within {0..1})*
using *line-in*
by (*auto intro*!: *derivative-eq-intros*
has-derivative-subset[OF - subset]
*has-derivative-in-compose[***where** *f=λx. a + x ∗R u]*
fderiv line-in
simp add: *linear.scaleR[OF has-derivative-linear[OF fderiv]]*)
with *zero-less-one*
have *∃ x∈{0<..<1}. f (a + 1 ∗R u) · i − f (a + 0 ∗R u) · i = (1 − 0) ∗R J (a + x ∗R u) u · i*
by (*rule mvt-simple*)
}
then obtain *t* **where** *∀ i∈Basis. t i ∈ {0<..<1} ∧ f (a + u) · i − f a · i = J (a + t i ∗R u) u · i*
by *atomize-elim* (*force intro*!: *bchoice*)
hence *t ∈ Basis → {0<..<1} ⋀i. i ∈ Basis ⟹ (f (a + u) − f a) · i = J (a + t i ∗R u) u · i*
by (*auto simp*: *inner-diff-left*)
moreover hence *(f (a + u) − f a) = (∑ i∈Basis. (J (a + t i ∗R u) u · i) ∗R i)*

```
    by (intro euclidean-eqI[where 'a='b]) simp
  ultimately show ?thesis by blast
qed

lemma MVT-ivl:
  fixes f::'a::ordered-euclidean-space⇒'b::ordered-euclidean-space
  assumes fderiv: ⋀x. x ∈ D ⟹ (f has-derivative J x) (at x within D)
  assumes J-ivl: ⋀x. x ∈ D ⟹ J x u ∈ {J0 .. J1}
  assumes line-in: ⋀x. x ∈ {0..1} ⟹ a + x *R u ∈ D
  shows f (a + u) − f a ∈ {J0..J1}
proof −
  from MVT-corrected[OF fderiv line-in] obtain t where
    t: t∈Basis → {0<..<1} and
    mvt: f (a + u) − f a = (∑ i∈Basis. (J (a + t i *R u) u · i) *R i)
    by auto
  note mvt
  also have ... ∈ {J0 .. J1}
  proof −
    have J: ⋀i. i ∈ Basis ⟹ J0 ≤ J (a + t i *R u) u
            ⋀i. i ∈ Basis ⟹ J (a + t i *R u) u ≤ J1
      using J-ivl t line-in by (auto simp: Pi-iff)
    show ?thesis
      using J
      unfolding atLeastAtMost-iff eucl-le[where 'a='b]
      by auto
  qed
  finally show ?thesis .
qed

lemma MVT:
  shows
  ⋀J J0 J1 a u (f::real*real⇒real*real).
    (⋀x. FDERIV f x :> J x) ⟹
    (⋀x. J x u ∈ {J0 .. J1}) ⟹
    f (a + u) − f a ∈ {J0 .. J1}
  by (rule-tac J = J in MVT-ivl[where D=UNIV]) auto

lemma MVT-ivl':
  fixes f::'a::ordered-euclidean-space⇒'b::ordered-euclidean-space
  assumes fderiv: (⋀x. x ∈ D ⟹ (f has-derivative J x) (at x within D))
  assumes J-ivl: ⋀x. x ∈ D ⟹ J x (a − b) ∈ {J0..J1}
  assumes line-in: ⋀x. x ∈ {0..1} ⟹ b + x *R (a − b) ∈ D
  shows f a ∈ {f b + J0..f b + J1}
proof −
  have f (b + (a − b)) − f b ∈ {J0 .. J1}
    using J-ivl MVT-ivl fderiv line-in by blast
  thus ?thesis
    by (auto simp: diff-le-eq le-diff-eq ac-simps)
qed
```

end
theory
Vector-Derivative-On
imports
HOL−Analysis.Analysis
begin

1.24 Vector derivative on a set

definition
has-vderiv-on :: (*real* ⇒ ′*a*::*real-normed-vector*) ⇒ (*real* ⇒ ′*a*) ⇒ *real set* ⇒ *bool*
(**infix** ‹(*has*′*-vderiv*′*-on*)› *50*)
where
(*f has-vderiv-on f*′) *S* ⟷ (∀ *x* ∈ *S*. (*f has-vector-derivative f*′ *x*) (*at x within S*))

lemma *has-vderiv-on-empty*[*intro*, *simp*]: (*f has-vderiv-on f*′) {}
by (*auto simp*: *has-vderiv-on-def*)

lemma *has-vderiv-on-subset*:
assumes (*f has-vderiv-on f*′) *S*
assumes *T* ⊆ *S*
shows (*f has-vderiv-on f*′) *T*
by (*meson assms*(*1*) *assms*(*2*) *contra-subsetD has-vderiv-on-def has-vector-derivative-within-subset*)

lemma *has-vderiv-on-compose*:
assumes (*f has-vderiv-on f*′) (*g* ‘ *T*)
assumes (*g has-vderiv-on g*′) *T*
shows (*f o g has-vderiv-on* (λ*x*. *g*′ *x* $*_R$ *f*′ (*g x*))) *T*
using *assms*
unfolding *has-vderiv-on-def*
by (*auto intro*!: *vector-diff-chain-within*)

lemma *has-vderiv-on-open*:
assumes *open T*
shows (*f has-vderiv-on f*′) *T* ⟷ (∀ *t* ∈ *T*. (*f has-vector-derivative f*′ *t*) (*at t*))
by (*auto simp*: *has-vderiv-on-def at-within-open*[*OF* - ‹*open T*›])

lemma *has-vderiv-on-eq-rhs*:— TODO: integrate intro *derivative-eq-intros*
(*f has-vderiv-on g*′) *T* ⟹ (⋀*x*. *x* ∈ *T* ⟹ *g*′ *x* = *f*′ *x*) ⟹ (*f has-vderiv-on f*′)
T
by (*auto simp*: *has-vderiv-on-def*)

lemma [*THEN has-vderiv-on-eq-rhs*, *derivative-intros*]:
shows *has-vderiv-on-id*: ((λ*x*. *x*) *has-vderiv-on* (λ*x*. *1*)) *T*
and *has-vderiv-on-const*: ((λ*x*. *c*) *has-vderiv-on* (λ*x*. *0*)) *T*
by (*auto simp*: *has-vderiv-on-def intro*!: *derivative-eq-intros*)

lemma [*THEN has-vderiv-on-eq-rhs*, *derivative-intros*]:

fixes f::*real* ⇒ *'a*::*real-normed-vector*
assumes (f *has-vderiv-on* f') T
shows *has-vderiv-on-uminus*: (($\lambda x. - f\ x$) *has-vderiv-on* ($\lambda x. - f'\ x$)) T
using *assms*
by (*auto simp*: *has-vderiv-on-def intro*!: *derivative-eq-intros*)

lemma [*THEN has-vderiv-on-eq-rhs*, *derivative-intros*]:
fixes $f\ g$::*real* ⇒ *'a*::*real-normed-vector*
assumes (f *has-vderiv-on* f') T
assumes (g *has-vderiv-on* g') T
shows *has-vderiv-on-add*: (($\lambda x.\ f\ x + g\ x$) *has-vderiv-on* ($\lambda x.\ f'\ x + g'\ x$)) T
and *has-vderiv-on-diff*: (($\lambda x.\ f\ x - g\ x$) *has-vderiv-on* ($\lambda x.\ f'\ x - g'\ x$)) T
using *assms*
by (*auto simp*: *has-vderiv-on-def intro*!: *derivative-eq-intros*)

lemma [*THEN has-vderiv-on-eq-rhs*, *derivative-intros*]:
fixes f::*real* ⇒ *real* **and** g::*real* ⇒ *'a*::*real-normed-vector*
assumes (f *has-vderiv-on* f') T
assumes (g *has-vderiv-on* g') T
shows *has-vderiv-on-scaleR*: (($\lambda x.\ f\ x *_R g\ x$) *has-vderiv-on* ($\lambda x.\ f\ x *_R g'\ x + f'\ x *_R g\ x$)) T
using *assms*
by (*auto simp*: *has-vderiv-on-def has-real-derivative-iff-has-vector-derivative*
intro!: *derivative-eq-intros*)

lemma [*THEN has-vderiv-on-eq-rhs*, *derivative-intros*]:
fixes $f\ g$::*real* ⇒ *'a*::*real-normed-algebra*
assumes (f *has-vderiv-on* f') T
assumes (g *has-vderiv-on* g') T
shows *has-vderiv-on-mult*: (($\lambda x.\ f\ x * g\ x$) *has-vderiv-on* ($\lambda x.\ f\ x * g'\ x + f'\ x * g\ x$)) T
using *assms*
by (*auto simp*: *has-vderiv-on-def intro*!: *derivative-eq-intros*)

lemma *has-vderiv-on-ln*[*THEN has-vderiv-on-eq-rhs*, *derivative-intros*]:
fixes g::*real* ⇒ *real*
assumes $\bigwedge x.\ x \in s \Longrightarrow 0 < g\ x$
assumes (g *has-vderiv-on* g') s
shows (($\lambda x.\ ln\ (g\ x)$) *has-vderiv-on* ($\lambda x.\ g'\ x\ /\ g\ x$)) s
using *assms*
unfolding *has-vderiv-on-def*
by (*auto simp*: *has-vderiv-on-def has-real-derivative-iff-has-vector-derivative*[*symmetric*]
intro!: *derivative-eq-intros*)

lemma *fundamental-theorem-of-calculus'*:
fixes f :: *real* ⇒ *'a*::*banach*
shows $a \le b \Longrightarrow$ (f *has-vderiv-on* f') $\{a\ ..\ b\} \Longrightarrow$ (f' *has-integral* ($f\ b - f\ a$)) $\{a\ ..\ b\}$

by (*auto intro*!: *fundamental-theorem-of-calculus simp*: *has-vderiv-on-def*)

lemma *has-vderiv-on-If*:
assumes $U = S \cup T$
assumes (f *has-vderiv-on* f') ($S \cup$ (*closure* $T \cap$ *closure* S))
assumes (g *has-vderiv-on* g') ($T \cup$ (*closure* $T \cap$ *closure* S))
assumes $\bigwedge x.\ x \in$ *closure* $T \Longrightarrow x \in$ *closure* $S \Longrightarrow f\ x = g\ x$
assumes $\bigwedge x.\ x \in$ *closure* $T \Longrightarrow x \in$ *closure* $S \Longrightarrow f'\ x = g'\ x$
shows (($\lambda t.$ *if* $t \in S$ *then* $f\ t$ *else* $g\ t$) *has-vderiv-on* ($\lambda t.$ *if* $t \in S$ *then* $f'\ t$ *else* g' t)) U
using *assms*
by (*auto simp*: *has-vderiv-on-def ac-simps*
intro!: *has-vector-derivative-If-within-closures*
split del: *if-split*)

lemma *mvt-very-simple-closed-segmentE*:
fixes *f*::*real*⇒*real*
assumes (f *has-vderiv-on* f') (*closed-segment* a b)
obtains y **where** $y \in$ *closed-segment* a b $f\ b - f\ a = (b - a) * f'\ y$
proof *cases*
assume $a \leq b$
with *mvt-very-simple*[*of* a b f $\lambda x\ i.\ i *_R f'\ x$] *assms*
obtain y **where** $y \in$ *closed-segment* a b $f\ b - f\ a = (b - a) * f'\ y$
by (*auto simp*: *has-vector-derivative-def closed-segment-eq-real-ivl has-vderiv-on-def*)
thus *?thesis* **..**
next
assume $\neg\ a \leq b$
with *mvt-very-simple*[*of* b a f $\lambda x\ i.\ i *_R f'\ x$] *assms*
obtain y **where** $y \in$ *closed-segment* a b $f\ b - f\ a = (b - a) * f'\ y$
by (*force simp*: *has-vector-derivative-def has-vderiv-on-def closed-segment-eq-real-ivl algebra-simps*)
thus *?thesis* **..**
qed

lemma *mvt-simple-closed-segmentE*:
fixes *f*::*real*⇒*real*
assumes (f *has-vderiv-on* f') (*closed-segment* a b)
assumes $a \neq b$
obtains y **where** $y \in$ *open-segment* a b $f\ b - f\ a = (b - a) * f'\ y$
proof *cases*
assume $a \leq b$
with *assms* **have** $a < b$ **by** *simp*
with *mvt-simple*[*of* a b f $\lambda x\ i.\ i *_R f'\ x$] *assms*
obtain y **where** $y \in$ *open-segment* a b $f\ b - f\ a = (b - a) * f'\ y$
by (*auto simp*: *has-vector-derivative-def closed-segment-eq-real-ivl has-vderiv-on-def open-segment-eq-real-ivl*)
thus *?thesis* **..**
next
assume $\neg\ a \leq b$

then have $b < a$ **by** *simp*
with *mvt-simple*[*of b a f* $\lambda x\ i.\ i *_R f'\ x$] *assms*
obtain *y* **where** $y \in$ *open-segment a b* $f\ b - f\ a = (b - a) * f'\ y$
by (*force simp*: *has-vector-derivative-def has-vderiv-on-def closed-segment-eq-real-ivl algebra-simps*
open-segment-eq-real-ivl)
thus *?thesis* **..**
qed

lemma *differentiable-bound-general-open-segment*:
fixes *a* :: *real*
and *b* :: *real*
and *f* :: *real* ⇒ *'a*::*real-normed-vector*
and *f'* :: *real* ⇒ *'a*
assumes *continuous-on* (*closed-segment a b*) *f*
assumes *continuous-on* (*closed-segment a b*) *g*
and (*f has-vderiv-on f'*) (*open-segment a b*)
and (*g has-vderiv-on g'*) (*open-segment a b*)
and $\bigwedge x.\ x \in$ *open-segment a b* $\Longrightarrow$ *norm* $(f'\ x) \le g'\ x$
shows *norm* $(f\ b - f\ a) \le$ *abs* $(g\ b - g\ a)$
proof −
{
assume $a = b$
hence *?thesis* **by** *simp*
} moreover {
assume $a < b$
with *assms*
have *continuous-on* $\{a\ ..\ b\}$ *f*
and *continuous-on* $\{a\ ..\ b\}$ *g*
and $\bigwedge x.\ x \in \{a <..< b\} \Longrightarrow$ (*f has-vector-derivative* $f'\ x$) (*at x*)
and $\bigwedge x.\ x \in \{a <..< b\} \Longrightarrow$ (*g has-vector-derivative* $g'\ x$) (*at x*)
and $\bigwedge x.\ x \in \{a <..< b\} \Longrightarrow$ *norm* $(f'\ x) \le g'\ x$
by (*auto simp*: *open-segment-eq-real-ivl closed-segment-eq-real-ivl has-vderiv-on-def at-within-open*[**where** $S=\{a<..<b\}$])
from *differentiable-bound-general*[*OF* ‹$a < b$› *this*]
have *?thesis* **by** *auto*
} moreover {
assume $b < a$
with *assms*
have *continuous-on* $\{b\ ..\ a\}$ *f*
and *continuous-on* $\{b\ ..\ a\}$ *g*
and $\bigwedge x.\ x \in \{b <..< a\} \Longrightarrow$ (*f has-vector-derivative* $f'\ x$) (*at x*)
and $\bigwedge x.\ x \in \{b <..< a\} \Longrightarrow$ (*g has-vector-derivative* $g'\ x$) (*at x*)
and $\bigwedge x.\ x \in \{b <..< a\} \Longrightarrow$ *norm* $(f'\ x) \le g'\ x$
by (*auto simp*: *open-segment-eq-real-ivl closed-segment-eq-real-ivl has-vderiv-on-def at-within-open*[**where** $S=\{b<..<a\}$])
from *differentiable-bound-general*[*OF* ‹$b < a$› *this*]
have *norm* $(f\ a - f\ b) \le g\ a - g\ b$ **by** *simp*
also have $\ldots \le$ *abs* $(g\ b - g\ a)$ **by** *simp*

finally have *?thesis* **by** (*simp add*: *norm-minus-commute*)
} **ultimately show** *?thesis* **by** *arith*
qed

lemma *has-vderiv-on-union*:
assumes (*f has-vderiv-on g*) ($s \cup$ *closure* $s \cap$ *closure* t)
assumes (*f has-vderiv-on g*) ($t \cup$ *closure* $s \cap$ *closure* t)
shows (*f has-vderiv-on g*) ($s \cup t$)
unfolding *has-vderiv-on-def*
proof
fix x **assume** $x \in s \cup t$
with *has-vector-derivative-If-within-closures*[*of x s t s* $\cup$ *t f g f g*] *assms*
show (*f has-vector-derivative g x*) (*at x within s* $\cup$ *t*)
by (*auto simp*: *has-vderiv-on-def*)
qed

lemma *has-vderiv-on-union-closed*:
assumes (*f has-vderiv-on g*) *s*
assumes (*f has-vderiv-on g*) *t*
assumes *closed s closed t*
shows (*f has-vderiv-on g*) ($s \cup t$)
using *has-vderiv-on-If*[*OF refl*, *of f g s t f g*] *assms*
by (*auto simp*: *has-vderiv-on-subset*)

lemma *vderiv-on-continuous-on*: (*f has-vderiv-on f'*) $S \Longrightarrow$ *continuous-on S f*
by (*auto intro*!: *continuous-on-vector-derivative simp*: *has-vderiv-on-def*)

lemma *has-vderiv-on-cong*[*cong*]:
assumes $\bigwedge x.\ x \in S \Longrightarrow f\ x = g\ x$
assumes $\bigwedge x.\ x \in S \Longrightarrow f'\ x = g'\ x$
assumes $S = T$
shows (*f has-vderiv-on f'*) S = (*g has-vderiv-on g'*) T
using *assms*
by (*metis has-vector-derivative-transform has-vderiv-on-def*)

lemma *has-vderiv-eq*:
assumes (*f has-vderiv-on f'*) S
assumes $\bigwedge x.\ x \in S \Longrightarrow f\ x = g\ x$
assumes $\bigwedge x.\ x \in S \Longrightarrow f'\ x = g'\ x$
assumes $S = T$
shows (*g has-vderiv-on g'*) T
using *assms* **by** *simp*

lemma *has-vderiv-on-compose'*:
assumes (*f has-vderiv-on f'*) (*g* ‘ *T*)
assumes (*g has-vderiv-on g'*) T
shows (($\lambda x.\ f\ (g\ x)$) *has-vderiv-on* ($\lambda x.\ g'\ x *_R f'\ (g\ x)$)) T
using *has-vderiv-on-compose*[*OF assms*]
by *simp*

lemma *has-vderiv-on-compose2*:
assumes (*f has-vderiv-on f′*) *S*
assumes (*g has-vderiv-on g′*) *T*
assumes $\bigwedge t.\ t \in T \Longrightarrow g\ t \in S$
shows ((λ*x*. *f* (*g x*)) *has-vderiv-on* (λ*x*. *g′ x* $*_R$ *f′* (*g x*))) *T*
using *has-vderiv-on-compose*[*OF has-vderiv-on-subset*[*OF assms*(*1*)] *assms*(*2*)] *assms*(*3*)
by *force*

lemma *has-vderiv-on-singleton*: (*y has-vderiv-on y′*) {*t0*}
by (*auto simp*: *has-vderiv-on-def has-vector-derivative-def has-derivative-within-singleton-iff bounded-linear-scaleR-left*)

lemma
has-vderiv-on-zero-constant:
assumes *convex s*
assumes (*f has-vderiv-on* (λ*h*. *0*)) *s*
obtains *c* **where** $\bigwedge x.\ x \in s \Longrightarrow f\ x = c$
using *has-vector-derivative-zero-constant*[*of s f*] *assms*
by (*auto simp*: *has-vderiv-on-def*)

lemma *bounded-vderiv-on-imp-lipschitz*:
assumes (*f has-vderiv-on f′*) *X*
assumes *convex*: *convex X*
assumes $\bigwedge x.\ x \in X \Longrightarrow norm\ (f'\ x) \le C\ 0 \le C$
shows *C*−*lipschitz-on X f*
using *assms*
by (*auto simp*: *has-vderiv-on-def has-vector-derivative-def onorm-scaleR-left onorm-id intro*!: *bounded-derivative-imp-lipschitz*[**where** *f′* = λ*x d*. *d* $*_R$ *f′ x*])

end
theory *Interval-Integral-HK*
imports *Vector-Derivative-On*
begin

1.25 interval integral

definition *has-ivl-integral* ::
(*real* ⇒ ′*b*::*real-normed-vector*) ⇒ ′*b* ⇒ *real* ⇒ *real* ⇒ *bool*— TODO: generalize?
(**infixr** ‹*has′-ivl′-integral*› *46*)
where (*f has-ivl-integral y*) *a b* ⟷ (*if a* ≤ *b then* (*f has-integral y*) {*a* .. *b*} *else* (*f has-integral* − *y*) {*b* .. *a*})

definition *ivl-integral*::*real* ⇒ *real* ⇒ (*real* ⇒ ′*a*) ⇒ ′*a*::*real-normed-vector*
where *ivl-integral a b f* = *integral* {*a* .. *b*} *f* − *integral* {*b* .. *a*} *f*

lemma *integral-emptyI*[*simp*]:
fixes *a b*::*real*

shows *a* ≥ *b* ⟹ *integral* {*a*..*b*} *f* = *0* *a* > *b* ⟹ *integral* {*a*..*b*} *f* = *0*
by (*cases a* = *b*) *auto*

lemma *ivl-integral-unique*: (*f has-ivl-integral y*) *a b* ⟹ *ivl-integral a b f* = *y*
using *integral-unique*[*of f y* {*a* .. *b*}] *integral-unique*[*of f* − *y* {*b* .. *a*}]
unfolding *ivl-integral-def has-ivl-integral-def*
by (*auto split*: *if-split-asm*)

lemma *fundamental-theorem-of-calculus-ivl-integral*:
fixes *f* :: *real* ⇒ *'a*::*banach*
shows (*f has-vderiv-on f'*) (*closed-segment a b*) ⟹ (*f' has-ivl-integral f b* − *f a*) *a b*
by (*auto simp*: *has-ivl-integral-def closed-segment-eq-real-ivl intro*!: *fundamental-theorem-of-calculus'*)

lemma
fixes *f* :: *real* ⇒ *'a*::*banach*
assumes *f integrable-on* (*closed-segment a b*)
shows *indefinite-ivl-integral-continuous*:
continuous-on (*closed-segment a b*) (*λx*. *ivl-integral a x f*)
continuous-on (*closed-segment b a*) (*λx*. *ivl-integral a x f*)
using *assms*
by (*auto simp*: *ivl-integral-def closed-segment-eq-real-ivl split*: *if-split-asm*
intro!: *indefinite-integral-continuous-1 indefinite-integral-continuous-1'*
continuous-intros intro: *continuous-on-eq*)

lemma
fixes *f* :: *real* ⇒ *'a*::*banach*
assumes *f integrable-on* (*closed-segment a b*)
assumes *c* ∈ *closed-segment a b*
shows *indefinite-ivl-integral-continuous-subset*:
continuous-on (*closed-segment a b*) (*λx*. *ivl-integral c x f*)
proof −
from *assms* **have** *f integrable-on* (*closed-segment c a*) *f integrable-on* (*closed-segment c b*)
by (*auto simp*: *closed-segment-eq-real-ivl integrable-on-subinterval*
integrable-on-insert-iff split: *if-splits*)
then have *continuous-on* (*closed-segment a c* ∪ *closed-segment c b*) (*λx*. *ivl-integral c x f*)
by (*auto intro*!: *indefinite-ivl-integral-continuous continuous-on-closed-Un*)
also have *closed-segment a c* ∪ *closed-segment c b* = *closed-segment a b*
using *assms* **by** (*auto simp*: *closed-segment-eq-real-ivl*)
finally show *?thesis* **.**
qed

lemma *real-Icc-closed-segment*: **fixes** *a b*::*real* **shows** *a* ≤ *b* ⟹ {*a* .. *b*} = *closed-segment a b*
by (*auto simp*: *closed-segment-eq-real-ivl*)

lemma *ivl-integral-zero*[*simp*]: *ivl-integral a a f* = *0*
  by (*auto simp*: *ivl-integral-def*)

lemma *ivl-integral-cong*:
  assumes ⋀*x*. *x* ∈ *closed-segment a b* ⟹ *g x* = *f x*
  assumes *a* = *c b* = *d*
  shows *ivl-integral a b f* = *ivl-integral c d g*
  using *assms integral-spike*[*of* {} *closed-segment a b f g*]
  by (*auto simp*: *ivl-integral-def closed-segment-eq-real-ivl split*: *if-split-asm*)

lemma *ivl-integral-diff*:
  f integrable-on (*closed-segment s t*) ⟹ *g integrable-on* (*closed-segment s t*) ⟹
    ivl-integral s t (λ*x*. *f x* − *g x*) = *ivl-integral s t f* − *ivl-integral s t g*
  using *Henstock-Kurzweil-Integration.integral-diff*[*of f closed-segment s t g*]
  by (*auto simp*: *ivl-integral-def closed-segment-eq-real-ivl split*: *if-split-asm*)

lemma *ivl-integral-norm-bound-ivl-integral*:
  fixes *f* :: *real* ⇒ *'a*::*banach*
  assumes *f integrable-on* (*closed-segment a b*)
    and *g integrable-on* (*closed-segment a b*)
    and ⋀*x*. *x* ∈ *closed-segment a b* ⟹ *norm* (*f x*) ≤ *g x*
  shows *norm* (*ivl-integral a b f*) ≤ *abs* (*ivl-integral a b g*)
  using *integral-norm-bound-integral*[*OF assms*]
  by (*auto simp*: *ivl-integral-def closed-segment-eq-real-ivl split*: *if-split-asm*)

lemma *ivl-integral-norm-bound-integral*:
  fixes *f* :: *real* ⇒ *'a*::*banach*
  assumes *f integrable-on* (*closed-segment a b*)
    and *g integrable-on* (*closed-segment a b*)
    and ⋀*x*. *x* ∈ *closed-segment a b* ⟹ *norm* (*f x*) ≤ *g x*
  shows *norm* (*ivl-integral a b f*) ≤ *integral* (*closed-segment a b*) *g*
  using *integral-norm-bound-integral*[*OF assms*]
  by (*auto simp*: *ivl-integral-def closed-segment-eq-real-ivl split*: *if-split-asm*)

lemma *norm-ivl-integral-le*:
  fixes *f* :: *real* ⇒ *real*
  assumes *f integrable-on* (*closed-segment a b*)
    and *g integrable-on* (*closed-segment a b*)
    and ⋀*x*. *x* ∈ *closed-segment a b* ⟹ *f x* ≤ *g x*
    and ⋀*x*. *x* ∈ *closed-segment a b* ⟹ *0* ≤ *f x*
  shows *abs* (*ivl-integral a b f*) ≤ *abs* (*ivl-integral a b g*)
proof (*cases a* = *b*)
  case *True* **then show** *?thesis*
    by *simp*
next
  case *False*
  have *0* ≤ *integral* {*a*..*b*} *f 0* ≤ *integral* {*b*..*a*} *f*
   by (*metis le-cases Henstock-Kurzweil-Integration.integral-nonneg assms*(*1*) *assms*(*4*)
closed-segment-eq-real-ivl integral-emptyI(*1*))+

then show *?thesis*
using *integral-le*[*OF assms*(*1−3*)]
unfolding *ivl-integral-def closed-segment-eq-real-ivl*
by (*simp split*: *if-split-asm*)
qed

lemma *ivl-integral-const* [*simp*]:
shows *ivl-integral a b* ($\lambda x.$ *c*) = (*b* − *a*) $*_R$ *c*
by (*auto simp*: *ivl-integral-def algebra-simps*)

lemma *ivl-integral-has-vector-derivative*:
fixes *f* :: *real* ⇒ ′*a*::*banach*
assumes *continuous-on* (*closed-segment a b*) *f*
and *x* ∈ *closed-segment a b*
shows (($\lambda u.$ *ivl-integral a u f*) *has-vector-derivative f x*) (*at x within closed-segment a b*)
proof −
have (($\lambda x.$ *integral* {*x*..*a*} *f*) *has-vector-derivative 0*) (*at x within* {*a*..*b*}) **if** *a* ≤ *x x* ≤ *b*
by (*rule has-vector-derivative-transform*) (*auto simp*: *that*)
moreover
have (($\lambda x.$ *integral* {*a*..*x*} *f*) *has-vector-derivative 0*) (*at x within* {*b*..*a*}) **if** *b* ≤ *x x* ≤ *a*
by (*rule has-vector-derivative-transform*) (*auto simp*: *that*)
ultimately
show *?thesis*
using *assms*
by (*auto simp*: *ivl-integral-def closed-segment-eq-real-ivl*
intro!: *derivative-eq-intros*
integral-has-vector-derivative[*of a b f*] *integral-has-vector-derivative*[*of b a* −*f*]
integral-has-vector-derivative′[*of b a f*])
qed

lemma *ivl-integral-has-vderiv-on*:
fixes *f* :: *real* ⇒ ′*a*::*banach*
assumes *continuous-on* (*closed-segment a b*) *f*
shows (($\lambda u.$ *ivl-integral a u f*) *has-vderiv-on f*) (*closed-segment a b*)
using *ivl-integral-has-vector-derivative*[*OF assms*]
by (*auto simp*: *has-vderiv-on-def*)

lemma *ivl-integral-has-vderiv-on-subset-segment*:
fixes *f* :: *real* ⇒ ′*a*::*banach*
assumes *continuous-on* (*closed-segment a b*) *f*
and *c* ∈ *closed-segment a b*
shows (($\lambda u.$ *ivl-integral c u f*) *has-vderiv-on f*) (*closed-segment a b*)
proof −
have (*closed-segment c a*) ⊆ (*closed-segment a b*) (*closed-segment c b*) ⊆ (*closed-segment a b*)

using *assms* **by** (*auto simp*: *closed-segment-eq-real-ivl split*: *if-splits*)
then have ((λ*u*. *ivl-integral c u f*) *has-vderiv-on f*) ((*closed-segment c a*) ∪ (*closed-segment c b*))
by (*auto intro*!: *has-vderiv-on-union-closed ivl-integral-has-vderiv-on assms*
intro: *continuous-on-subset*)
also have (*closed-segment c a*) ∪ (*closed-segment c b*) = (*closed-segment a b*)
using *assms* **by** (*auto simp*: *closed-segment-eq-real-ivl split*: *if-splits*)
finally show *?thesis* **.**
qed

lemma *ivl-integral-has-vector-derivative-subset*:
fixes *f* :: *real* ⇒ *'a*::*banach*
assumes *continuous-on* (*closed-segment a b*) *f*
and *x* ∈ *closed-segment a b*
and *c* ∈ *closed-segment a b*
shows ((λ*u*. *ivl-integral c u f*) *has-vector-derivative f x*) (*at x within closed-segment a b*)
using *ivl-integral-has-vderiv-on-subset-segment*[*OF assms*(*1*)] *assms*(*2*−)
by (*auto simp*: *has-vderiv-on-def*)

lemma
compact-interval-eq-Inf-Sup:
fixes *A*::*real set*
assumes *is-interval A compact A A* ≠ {}
shows *A* = {*Inf A* .. *Sup A*}
apply (*auto simp*: *closed-segment-eq-real-ivl*
intro!: *cInf-lower cSup-upper bounded-imp-bdd-below bounded-imp-bdd-above*
compact-imp-bounded assms)
by (*metis assms*(*1*) *assms*(*2*) *assms*(*3*) *cInf-eq-minimum cSup-eq-maximum compact-attains-inf*
compact-attains-sup mem-is-interval-1-I)

lemma *ivl-integral-has-vderiv-on-compact-interval*:
fixes *f* :: *real* ⇒ *'a*::*banach*
assumes *continuous-on A f*
and *c* ∈ *A is-interval A compact A*
shows ((λ*u*. *ivl-integral c u f*) *has-vderiv-on f*) *A*
proof −
have *A* = {*Inf A* .. *Sup A*}
by (*rule compact-interval-eq-Inf-Sup*) (*use assms* **in** *auto*)
also have ... = *closed-segment* (*Inf A*) (*Sup A*) **using** *assms*
by (*auto simp add*: *closed-segment-eq-real-ivl*
intro!: *cInf-le-cSup bounded-imp-bdd-below bounded-imp-bdd-above compact-imp-bounded*)
finally have ∗: *A* = *closed-segment* (*Inf A*) (*Sup A*) **.**
show *?thesis*
apply (*subst* ∗)
apply (*rule ivl-integral-has-vderiv-on-subset-segment*)
unfolding ∗[*symmetric*]

by *fact+*
qed

lemma *ivl-integral-has-vector-derivative-compact-interval*:
fixes *f* :: *real* ⇒ *'a*::*banach*
assumes *continuous-on A f*
and *is-interval A compact A x* ∈ *A c* ∈ *A*
shows ((*λu. ivl-integral c u f*) *has-vector-derivative f x*) (*at x within A*)
using *ivl-integral-has-vderiv-on-compact-interval*[*OF assms*(*1*)] *assms*(*2*−)
by (*auto simp*: *has-vderiv-on-def*)

lemma *ivl-integral-combine*:
fixes *f*::*real* ⇒ *'a*::*banach*
assumes *f integrable-on* (*closed-segment a b*)
assumes *f integrable-on* (*closed-segment b c*)
assumes *f integrable-on* (*closed-segment a c*)
shows *ivl-integral a b f* + *ivl-integral b c f* = *ivl-integral a c f*
proof −
show *?thesis*
using *assms*
Henstock-Kurzweil-Integration.integral-combine[*of a b c f*]
Henstock-Kurzweil-Integration.integral-combine[*of a c b f*]
Henstock-Kurzweil-Integration.integral-combine[*of b a c f*]
Henstock-Kurzweil-Integration.integral-combine[*of b c a f*]
Henstock-Kurzweil-Integration.integral-combine[*of c a b f*]
Henstock-Kurzweil-Integration.integral-combine[*of c b a f*]
by (*cases a* ≤ *b*; *cases b* ≤ *c*; *cases a* ≤ *c*)
(*auto simp*: *algebra-simps ivl-integral-def closed-segment-eq-real-ivl*)
qed

lemma *integral-equation-swap-initial-value*:
fixes *x*::*real*⇒*'a*::*banach*
assumes ⋀*t. t* ∈ *closed-segment t0 t1* ⟹ *x t* = *x t0* + *ivl-integral t0 t* (*λt. f t* (*x t*))
assumes *t*: *t* ∈ *closed-segment t0 t1*
assumes *int*: (*λt. f t* (*x t*)) *integrable-on closed-segment t0 t1*
shows *x t* = *x t1* + *ivl-integral t1 t* (*λt. f t* (*x t*))
proof −
from *t int* **have** (*λt. f t* (*x t*)) *integrable-on closed-segment t0 t*
(*λt. f t* (*x t*)) *integrable-on closed-segment t t1*
by (*auto intro*: *integrable-on-subinterval simp*: *closed-segment-eq-real-ivl split*: *if-split-asm*)
with *assms*(*1*)[*of t*] *assms*(*2*−)
have *x t* − *x t0* = *ivl-integral t0 t1* (*λt. f t* (*x t*)) + *ivl-integral t1 t* (*λt. f t* (*x t*))
by (*subst ivl-integral-combine*) (*auto simp*: *closed-segment-commute*)
then have *x t* + *x t1* − (*x t0* + *ivl-integral t0 t1* (*λt. f t* (*x t*))) =
x t1 + *ivl-integral t1 t* (*λt. f t* (*x t*))
by (*simp add*: *algebra-simps*)
also have *x t0* + *ivl-integral t0 t1* (*λt. f t* (*x t*)) = *x t1*

by (*auto simp*: *assms*(*1*)[*symmetric*])
finally show *?thesis* **by** *simp*
qed

lemma *has-integral-nonpos*:
fixes *f* :: *'n*::*euclidean-space* ⇒ *real*
assumes (*f has-integral i*) *s*
and ∀ *x*∈*s*. *f x* ≤ *0*
shows *i* ≤ *0*
by (*rule has-integral-nonneg*[*of* −*f* −*i s*, *simplified*])
(*auto intro*!: *has-integral-neg simp*: *fun-Compl-def assms*)

lemma *has-ivl-integral-nonneg*:
fixes *f* :: *real* ⇒ *real*
assumes (*f has-ivl-integral i*) *a b*
and ⋀*x*. *a* ≤ *x* ⟹ *x* ≤ *b* ⟹ *0* ≤ *f x*
and ⋀*x*. *b* ≤ *x* ⟹ *x* ≤ *a* ⟹ *f x* ≤ *0*
shows *0* ≤ *i*
using *assms has-integral-nonneg*[*of f i* {*a* .. *b*}] *has-integral-nonpos*[*of f* −*i* {*b* .. *a*}]
by (*auto simp*: *has-ivl-integral-def Ball-def not-le split*: *if-split-asm*)

lemma *has-ivl-integral-ivl-integral*:
f integrable-on (*closed-segment a b*) ⟷ (*f has-ivl-integral* (*ivl-integral a b f*)) *a b*
by (*auto simp*: *closed-segment-eq-real-ivl has-ivl-integral-def ivl-integral-def*)

lemma *ivl-integral-nonneg*:
fixes *f* :: *real* ⇒ *real*
assumes *f integrable-on* (*closed-segment a b*)
and ⋀*x*. *a* ≤ *x* ⟹ *x* ≤ *b* ⟹ *0* ≤ *f x*
and ⋀*x*. *b* ≤ *x* ⟹ *x* ≤ *a* ⟹ *f x* ≤ *0*
shows *0* ≤ *ivl-integral a b f*
by (*rule has-ivl-integral-nonneg*[*OF assms*(*1*)[*unfolded has-ivl-integral-ivl-integral*] *assms*(*2*−*3*)])

lemma *ivl-integral-bound*:
fixes *f*::*real* ⇒ *'a*::*banach*
assumes *continuous-on* (*closed-segment a b*) *f*
assumes ⋀*t*. *t* ∈ (*closed-segment a b*) ⟹ *norm* (*f t*) ≤ *B*
shows *norm* (*ivl-integral a b f*) ≤ *B* ∗ *abs* (*b* − *a*)
using *integral-bound*[*of a b f B*]
integral-bound[*of b a f B*]
assms
by (*auto simp*: *closed-segment-eq-real-ivl has-ivl-integral-def ivl-integral-def split*: *if-splits*)

lemma *ivl-integral-minus-sets*:
fixes *f*::*real* ⇒ *'a*::*banach*

shows *f integrable-on* (*closed-segment c a*) ⟹ *f integrable-on* (*closed-segment c b*) ⟹ *f integrable-on* (*closed-segment a b*) ⟹
ivl-integral c a f − *ivl-integral c b f* = *ivl-integral b a f*
using *ivl-integral-combine*[*of f c b a*]
by (*auto simp*: *algebra-simps closed-segment-commute*)

lemma *ivl-integral-minus-sets'*:
fixes *f*::*real* ⇒ *'a*::*banach*
shows *f integrable-on* (*closed-segment a c*) ⟹ *f integrable-on* (*closed-segment b c*) ⟹ *f integrable-on* (*closed-segment a b*) ⟹
ivl-integral a c f − *ivl-integral b c f* = *ivl-integral a b f*
using *ivl-integral-combine*[*of f a b c*]
by (*auto simp*: *algebra-simps closed-segment-commute*)

end
theory *Gronwall*
imports *Vector-Derivative-On*
begin

1.26 Gronwall

lemma *derivative-quotient-bound*:
assumes *g-deriv-on*: (*g has-vderiv-on g'*) {*a* .. *b*}
assumes *frac-le*: ⋀*t*. *t* ∈ {*a* .. *b*} ⟹ *g' t* / *g t* ≤ *K*
assumes *g'-cont*: *continuous-on* {*a* .. *b*} *g'*
assumes *g-pos*: ⋀*t*. *t* ∈ {*a* .. *b*} ⟹ *g t* > *0*
assumes *t-in*: *t* ∈ {*a* .. *b*}
shows *g t* ≤ *g a* ∗ *exp* (*K* ∗ (*t* − *a*))
proof −
have *g-deriv*: ⋀*t*. *t* ∈ {*a* .. *b*} ⟹ (*g has-real-derivative g' t*) (*at t within* {*a* .. *b*})
using *g-deriv-on*
by (*auto simp*: *has-vderiv-on-def has-real-derivative-iff-has-vector-derivative*[*symmetric*])
from *assms* **have** *g-nonzero*: ⋀*t*. *t* ∈ {*a* .. *b*} ⟹ *g t* ≠ *0*
by *fastforce*
have *frac-integrable*: ⋀*t*. *t* ∈ {*a* .. *b*} ⟹ (λ*t*. *g' t* / *g t*) *integrable-on* {*a*..*t*}
by (*force simp*: *g-nonzero intro*: *assms has-field-derivative-subset*[*OF g-deriv*]
continuous-on-subset[*OF g'-cont*] *continuous-intros integrable-continuous-real*
continuous-on-subset[*OF vderiv-on-continuous-on*[*OF g-deriv-on*]])
have ⋀*t*. *t* ∈ {*a*..*b*} ⟹ ((λ*t*. *g' t* / *g t*) *has-integral ln* (*g t*) − *ln* (*g a*)) {*a* .. *t*}
by (*rule fundamental-theorem-of-calculus*)
(*auto intro*!: *derivative-eq-intros assms has-field-derivative-subset*[*OF g-deriv*]
simp: *has-real-derivative-iff-has-vector-derivative*[*symmetric*])
hence ∗: ⋀*t*. *t* ∈ {*a* .. *b*} ⟹ *ln* (*g t*) − *ln* (*g a*) = *integral* {*a* .. *t*} (λ*t*. *g' t* / *g t*)
using *integrable-integral*[*OF frac-integrable*]
by (*rule has-integral-unique*[**where** *f* = λ*t*. *g' t* / *g t*])
from ∗ *t-in* **have** *ln* (*g t*) − *ln* (*g a*) = *integral* {*a* .. *t*} (λ*t*. *g' t* / *g t*) **.**
also have . . . ≤ *integral* {*a* .. *t*} (λ-. *K*)

```
    using ‹t ∈ {a .. b}›
    by (intro integral-le) (auto intro!: frac-integrable frac-le integral-le)
  also have ... = K ∗ (t − a) using ‹t ∈ {a .. b}›
    by simp
  finally have ln (g t) ≤ K ∗ (t − a) + ln (g a) (is ?lhs ≤ ?rhs)
    by simp
  hence exp ?lhs ≤ exp ?rhs
    by simp
  thus ?thesis
    using ‹t ∈ {a .. b}› g-pos
    by (simp add: ac-simps exp-add del: exp-le-cancel-iff)
qed

lemma derivative-quotient-bound-left:
  assumes g-deriv-on: (g has-vderiv-on g') {a .. b}
  assumes frac-ge: ⋀t. t ∈ {a .. b} ⟹ K ≤ g' t / g t
  assumes g'-cont: continuous-on {a .. b} g'
  assumes g-pos: ⋀t. t ∈ {a .. b} ⟹ g t > 0
  assumes t-in: t ∈ {a..b}
  shows g t ≤ g b ∗ exp (K ∗ (t − b))
proof −
  have g-deriv: ⋀t. t ∈ {a .. b} ⟹ (g has-real-derivative g' t) (at t within {a ..
b})
    using g-deriv-on
   by (auto simp: has-vderiv-on-def has-real-derivative-iff-has-vector-derivative[symmetric])
  from assms have g-nonzero: ⋀t. t ∈ {a..b} ⟹ g t ≠ 0
    by fastforce
  have frac-integrable: ⋀t. t ∈ {a .. b} ⟹ (λt. g' t / g t) integrable-on {t..b}
    by (force simp: g-nonzero intro: assms has-field-derivative-subset[OF g-deriv]
      continuous-on-subset[OF g'-cont] continuous-intros integrable-continuous-real
      continuous-on-subset[OF vderiv-on-continuous-on[OF g-deriv-on]])
  have ⋀t. t ∈ {a..b} ⟹ ((λt. g' t / g t) has-integral ln (g b) − ln (g t)) {t..b}
    by (rule fundamental-theorem-of-calculus)
      (auto intro!: derivative-eq-intros assms has-field-derivative-subset[OF g-deriv]
        simp: has-real-derivative-iff-has-vector-derivative[symmetric])
  hence ∗: ⋀t. t ∈ {a..b} ⟹ ln (g b) − ln (g t) = integral {t..b} (λt. g' t / g t)
    using integrable-integral[OF frac-integrable]
    by (rule has-integral-unique[where f = λt. g' t / g t])
  have K ∗ (b − t) = integral {t..b} (λ-. K)
    using ‹t ∈ {a..b}›
    by simp
  also have ... ≤ integral {t..b} (λt. g' t / g t)
    using ‹t ∈ {a..b}›
    by (intro integral-le) (auto intro!: frac-integrable frac-ge integral-le)
  also have ... = ln (g b) − ln (g t)
    using ∗ t-in by simp
  finally have K ∗ (b − t) + ln (g t) ≤ ln (g b) (is ?lhs ≤ ?rhs)
    by simp
  hence exp ?lhs ≤ exp ?rhs
```

```
    by simp
  hence g t * exp (K * (b − t)) ≤ g b
    using ‹t ∈ {a..b}› g-pos
    by (simp add: ac-simps exp-add del: exp-le-cancel-iff)
  hence g t / exp (K * (t − b)) ≤ g b
    by (simp add: algebra-simps exp-diff)
  thus ?thesis
    by (simp add: field-simps)
qed

lemma gronwall-general:
  fixes g K C a b and t::real
  defines G ≡ λt. C + K * integral {a..t} (λs. g s)
  assumes g-le-G: ⋀t. t ∈ {a..b} ⟹ g t ≤ G t
  assumes g-cont: continuous-on {a..b} g
  assumes g-nonneg: ⋀t. t ∈ {a..b} ⟹ 0 ≤ g t
  assumes pos: 0 < C K > 0
  assumes t ∈ {a..b}
  shows g t ≤ C * exp (K * (t − a))
proof −
  have G-pos: ⋀t. t ∈ {a..b} ⟹ 0 < G t
   by (auto simp: G-def intro!: add-pos-nonneg mult-nonneg-nonneg Henstock-Kurzweil-Integration.integral-non
      integrable-continuous-real assms intro: less-imp-le continuous-on-subset)
  have g t ≤ G t using assms by auto
  also
  {
    have (G has-vderiv-on (λt. K * g t)) {a..b}
      by (auto intro!: derivative-eq-intros integral-has-vector-derivative g-cont
        simp add: G-def has-vderiv-on-def)
    moreover
    {
      fix t assume t ∈ {a..b}
      hence K * g t / G t ≤ K * G t / G t
        using pos g-le-G G-pos
        by (intro divide-right-mono mult-left-mono) (auto intro!: less-imp-le)
      also have ... = K
        using G-pos[of t] ‹t ∈ {a .. b}› by simp
      finally have K * g t / G t ≤ K .
    }
    ultimately have G t ≤ G a * exp (K * (t − a))
      apply (rule derivative-quotient-bound)
      using ‹t ∈ {a..b}›
      by (auto intro!: continuous-intros g-cont G-pos simp: field-simps pos)
  }
  also have G a = C
    by (simp add: G-def)
  finally show ?thesis
    by simp
qed
```

```
lemma gronwall-general-left:
  fixes g K C a b and t::real
  defines G ≡ λt. C + K * integral {t..b} (λs. g s)
  assumes g-le-G: ⋀t. t ∈ {a..b} ⟹ g t ≤ G t
  assumes g-cont: continuous-on {a..b} g
  assumes g-nonneg: ⋀t. t ∈ {a..b} ⟹ 0 ≤ g t
  assumes pos: 0 < C K > 0
  assumes t ∈ {a..b}
  shows g t ≤ C * exp (−K * (t − b))
proof −
  have G-pos: ⋀t. t ∈ {a..b} ⟹ 0 < G t
   by (auto simp: G-def intro!: add-pos-nonneg mult-nonneg-nonneg Henstock-Kurzweil-Integration.integral-non
      integrable-continuous-real assms intro: less-imp-le continuous-on-subset)
  have g t ≤ G t using assms by auto
  also
  {
    have (G has-vderiv-on (λt. −K * g t)) {a..b}
      by (auto intro!: derivative-eq-intros g-cont integral-has-vector-derivative′
          simp add: G-def has-vderiv-on-def)
    moreover
    {
      fix t assume t ∈ {a..b}
      hence K * g t / G t ≤ K * G t / G t
        using pos g-le-G G-pos
        by (intro divide-right-mono mult-left-mono) (auto intro!: less-imp-le)
      also have . . . = K
        using G-pos[of t] ‹t ∈ {a .. b}› by simp
      finally have K * g t / G t ≤ K .
      hence −K ≤ −K * g t / G t
        by simp
    }
    ultimately
    have G t ≤ G b * exp (−K * (t − b))
      apply (rule derivative-quotient-bound-left)
      using ‹t ∈ {a..b}›
      by (auto intro!: continuous-intros g-cont G-pos simp: field-simps pos)
  }
  also have G b = C
    by (simp add: G-def)
  finally show ?thesis
    by simp
qed

lemma gronwall-general-segment:
  fixes a b::real
 assumes ⋀t. t ∈ closed-segment a b ⟹ g t ≤ C + K * integral (closed-segment
a t) g
    and continuous-on (closed-segment a b) g
```

and $\bigwedge t.\ t \in$ *closed-segment a b* $\Longrightarrow 0 \le g\ t$
and $0 < C$
and $0 < K$
and $t \in$ *closed-segment a b*
shows $g\ t \le C * exp\ (K * abs\ (t - a))$
proof *cases*
assume $a \le b$
then have ∗: $abs\ (t - a) = t - a$ **using** *assms* **by** (*auto simp*: *closed-segment-eq-real-ivl*)
show *?thesis*
unfolding ∗
using *assms*
by (*intro gronwall-general*[**where** *b*=*b*]) (*auto intro*!: *simp*: *closed-segment-eq-real-ivl* ‹$a \le b$›)
next
assume $\neg a \le b$
then have ∗: $K * abs\ (t - a) = - K * (t - a)$ **using** *assms* **by** (*auto simp*: *closed-segment-eq-real-ivl algebra-simps*)
{
fix *s* :: *real*
assume *a1*: $b \le s$
assume *a2*: $s \le a$
assume *a3*: $\bigwedge t.\ b \le t \wedge t \le a \Longrightarrow g\ t \le C + K * integral$ (*if* $a \le t$ *then* $\{a..t\}$ *else* $\{t..a\}$) *g*
have $s = a \vee s < a$
using *a2* **by** (*meson less-eq-real-def*)
then have $g\ s \le C + K * integral\ \{s..a\}\ g$
using *a3 a1* **by** *fastforce*
} **then show** *?thesis*
unfolding ∗
using *assms* ‹$\neg a \le b$›
by (*intro gronwall-general-left*)
(*auto intro*!: *simp*: *closed-segment-eq-real-ivl*)
qed

lemma *gronwall-more-general-segment*:
fixes *a b c*::*real*
assumes $\bigwedge t.\ t \in$ *closed-segment a b* $\Longrightarrow g\ t \le C + K * integral$ (*closed-segment c t*) *g*
and *cont*: *continuous-on* (*closed-segment a b*) *g*
and $\bigwedge t.\ t \in$ *closed-segment a b* $\Longrightarrow 0 \le g\ t$
and $0 < C$
and $0 < K$
and *t*: $t \in$ *closed-segment a b*
and *c*: $c \in$ *closed-segment a b*
shows $g\ t \le C * exp\ (K * abs\ (t - c))$
proof −
from *t c* **have** $t \in$ *closed-segment c a* $\vee\ t \in$ *closed-segment c b*
by (*auto simp*: *closed-segment-eq-real-ivl split-ifs*)
then show *?thesis*

proof
assume *t* ∈ *closed-segment c a*
moreover
have *subs*: *closed-segment c a* ⊆ *closed-segment a b* **using** *t c*
by (*auto simp*: *closed-segment-eq-real-ivl split-ifs*)
ultimately show *?thesis*
by (*intro gronwall-general-segment*[**where** *b*=*a*])
(*auto intro*!: *assms intro*: *continuous-on-subset*)
next
assume *t* ∈ *closed-segment c b*
moreover
have *subs*: *closed-segment c b* ⊆ *closed-segment a b* **using** *t c*
by (*auto simp*: *closed-segment-eq-real-ivl*)
ultimately show *?thesis*
by (*intro gronwall-general-segment*[**where** *b*=*b*])
(*auto intro*!: *assms intro*: *continuous-on-subset*)
qed
qed

lemma *gronwall*:
fixes *g K C* **and** *t*::*real*
defines $G \equiv \lambda t.\ C + K * integral\ \{0..t\}\ (\lambda s.\ g\ s)$
assumes *g-le-G*: $\bigwedge t.\ 0 \le t \Longrightarrow t \le a \Longrightarrow g\ t \le G\ t$
assumes *g-cont*: *continuous-on* {*0..a*} *g*
assumes *g-nonneg*: $\bigwedge t.\ 0 \le t \Longrightarrow t \le a \Longrightarrow 0 \le g\ t$
assumes *pos*: $0 < C\ 0 < K$
assumes $0 \le t\ t \le a$
shows $g\ t \le C * exp\ (K * t)$
apply(*rule gronwall-general*[**where** *a*=*0*, *simplified*, *OF assms*(*2*−*6*)[*unfolded G-def*]])
using *assms*(*7*,*8*)
by *simp-all*

lemma *gronwall-left*:
fixes *g K C* **and** *t*::*real*
defines $G \equiv \lambda t.\ C + K * integral\ \{t..0\}\ (\lambda s.\ g\ s)$
assumes *g-le-G*: $\bigwedge t.\ a \le t \Longrightarrow t \le 0 \Longrightarrow g\ t \le G\ t$
assumes *g-cont*: *continuous-on* {*a..0*} *g*
assumes *g-nonneg*: $\bigwedge t.\ a \le t \Longrightarrow t \le 0 \Longrightarrow 0 \le g\ t$
assumes *pos*: $0 < C\ 0 < K$
assumes $a \le t\ t \le 0$
shows $g\ t \le C * exp\ (-K * t)$
apply(*simp*, *rule gronwall-general-left*[**where** *b*=*0*, *simplified*, *OF assms*(*2*−*6*)[*unfolded G-def*]])
using *assms*(*7*,*8*)
by *simp-all*

end

2 Initial Value Problems

theory *Initial-Value-Problem*
  imports
    ../ODE-Auxiliarities
    ../Library/Interval-Integral-HK
    ../Library/Gronwall
begin

lemma *clamp-le*[*simp*]: $x \le a \Longrightarrow$ *clamp a b x* $= a$ **for** *x*::′*a*::*ordered-euclidean-space*
 by (*auto simp*: *clamp-def eucl-le*[**where** ′*a*=′*a*] *intro*!: *euclidean-eqI*[**where** ′*a*=′*a*])

lemma *clamp-ge*[*simp*]: $a \le b \Longrightarrow b \le x \Longrightarrow$ *clamp a b x* $= b$ **for** *x*::′*a*::*ordered-euclidean-space*
 by (*force simp*: *clamp-def eucl-le*[**where** ′*a*=′*a*] *not-le not-less intro*!: *euclidean-eqI*[**where** ′*a*=′*a*])

abbreviation *cfuncset* :: ′*a*::*topological-space set* ⇒ ′*b*::*metric-space set* ⇒ (′*a* $\Rightarrow_C$ ′*b*) *set*
  (**infixr** ‹$\rightarrow_C$› *60*)
  where $A \rightarrow_C B \equiv$ *PiC A* (λ-. *B*)

lemma *closed-segment-translation-zero*: $z \in \{z + a{-}{-}z + b\} \longleftrightarrow 0 \in \{a -- b\}$
  by (*metis add.right-neutral closed-segment-translation-eq*)

lemma *closed-segment-subset-interval*: *is-interval T* $\Longrightarrow a \in T \Longrightarrow b \in T \Longrightarrow$ *closed-segment a b* $\subseteq T$
 by (*rule closed-segment-subset*) (*auto intro*!: *closed-segment-subset is-interval-convex*)

definition *half-open-segment*::′*a*::*real-vector* ⇒ ′*a* ⇒ ′*a set* (‹(*1*{-−−<-})›)
  where *half-open-segment a b* $= \{a -- b\} - \{b\}$

lemma *half-open-segment-real*:
  fixes *a b*::*real*
  shows $\{a --< b\} =$ (*if* $a \le b$ *then* $\{a ..< b\}$ *else* $\{b <.. a\}$)
  by (*auto simp*: *half-open-segment-def closed-segment-eq-real-ivl*)

lemma *closure-half-open-segment*:
  fixes *a b*::*real*
  shows *closure* $\{a --< b\} =$ (*if* $a = b$ *then* {} *else* $\{a -- b\}$)
  unfolding *closed-segment-eq-real-ivl if-distrib half-open-segment-real*
  unfolding *if-distribR*
  by *simp*

lemma *half-open-segment-subset*[*intro*, *simp*]:
  $\{t0{-}{-}{<}t1\} \subseteq \{t0 -- t1\}$
  $x \in \{t0{-}{-}{<}t1\} \Longrightarrow x \in \{t0 -- t1\}$
  by (*auto simp*: *half-open-segment-def*)

lemma *half-open-segment-closed-segmentI*:

```
  t ∈ {t0 −− t1} ⟹ t ≠ t1 ⟹ t ∈ {t0 −−< t1}
  by (auto simp: half-open-segment-def)

lemma islimpt-half-open-segment:
  fixes t0 t1 s::real
  assumes t0 ≠ t1 s ∈ {t0−−t1}
  shows s islimpt {t0−−<t1}
proof −
  have s islimpt {t0..<t1} if t0 ≤ s s ≤ t1 for s
  proof −
    have ∗: {t0..<t1} − {s} = {t0..<s} ∪ {s<..<t1}
      using that by auto
    show ?thesis
      using that ‹t0 ≠ t1› ∗
      by (cases t0 = s) (auto simp: islimpt-in-closure)
  qed
  moreover have s islimpt {t1<..t0} if t1 ≤ s s ≤ t0 for s
  proof −
    have ∗: {t1<..t0} − {s} = {t1<..<s} ∪ {s<..t0}
      using that by auto
    show ?thesis
      using that ‹t0 ≠ t1› ∗
      by (cases t0 = s) (auto simp: islimpt-in-closure)
  qed
  ultimately show ?thesis using assms
    by (auto simp: half-open-segment-real closed-segment-eq-real-ivl)
qed

lemma
  mem-half-open-segment-eventually-in-closed-segment:
  fixes t::real
  assumes t ∈ {t0−−<t1'}
  shows ∀F t1' in at t1' within {t0−−<t1'}. t ∈ {t0−−t1'}
  unfolding half-open-segment-real
proof (split if-split, safe)
  assume le: t0 ≤ t1'
  with assms have t: t0 ≤ t t < t1'
    by (auto simp: half-open-segment-real)
  then have ∀F t1' in at t1' within {t0..<t1'}. t0 ≤ t
    by simp
  moreover
  from tendsto-ident-at ‹t < t1'›
  have ∀F t1' in at t1' within {t0..<t1'}. t < t1'
    by (rule order-tendstoD)
  ultimately show ∀F t1' in at t1' within {t0..<t1'}. t ∈ {t0−−t1'}
    by eventually-elim (auto simp add: closed-segment-eq-real-ivl)
next
  assume le: ¬ t0 ≤ t1'
  with assms have t: t ≤ t0 t1' < t
```

by (*auto simp*: *half-open-segment-real*)
then have $\forall_F$ *t1′ in at t1′ within* {*t1′*<..*t0*}. $t \le t0$
by *simp*
moreover
from *tendsto-ident-at* ‹*t1′* < *t*›
have $\forall_F$ *t1′ in at t1′ within* {*t1′*<..*t0*}. *t1′* < *t*
by (*rule order-tendstoD*)
ultimately show $\forall_F$ *t1′ in at t1′ within* {*t1′*<..*t0*}. $t \in$ {*t0*−−*t1′*}
by *eventually-elim* (*auto simp add*: *closed-segment-eq-real-ivl*)
qed

lemma *closed-segment-half-open-segment-subsetI*:
fixes *x*::*real* **shows** *x* ∈ {*t0*−−<*t1*} ⟹ {*t0*−−*x*} ⊆ {*t0*−−<*t1*}
by (*auto simp*: *half-open-segment-real closed-segment-eq-real-ivl split*: *if-split-asm*)

lemma *dist-component-le*:
fixes *x y*::*′a*::*euclidean-space*
assumes *i* ∈ *Basis*
shows *dist* (*x* · *i*) (*y* · *i*) ≤ *dist x y*
using *assms*
by (*auto simp*: *euclidean-dist-l2*[*of x y*] *intro*: *member-le-L2-set*)

lemma *sum-inner-Basis-one*: *i* ∈ *Basis* ⟹ (∑*x*∈*Basis*. *x* · *i*) = *1*
by (*subst sum.mono-neutral-right*[**where** *S*={*i*}])
(*auto simp*: *inner-not-same-Basis*)

lemma *cball-in-cbox*:
fixes *y*::*′a*::*euclidean-space*
shows *cball y r* ⊆ *cbox* (*y* − *r* $*_R$ *One*) (*y* + *r* $*_R$ *One*)
unfolding *scaleR-sum-right interval-cbox cbox-def*
proof *safe*
fix *x i*::*′a* **assume** *i* ∈ *Basis x* ∈ *cball y r*
with *dist-component-le*[*OF* ‹*i* ∈ *Basis*›, *of y x*]
have *dist* (*y* · *i*) (*x* · *i*) ≤ *r* **by** (*simp add*: *mem-cball*)
thus (*y* − *sum* (($*_R$) *r*) *Basis*) · *i* ≤ *x* · *i*
x · *i* ≤ (*y* + *sum* (($*_R$) *r*) *Basis*) · *i*
by (*auto simp add*: *inner-diff-left inner-add-left inner-sum-left*
sum-distrib-left[*symmetric*] *sum-inner-Basis-one* ‹*i*∈*Basis*› *dist-real-def*)
qed

lemma *centered-cbox-in-cball*:
shows *cbox* (− *r* $*_R$ *One*) (*r* $*_R$ *One*::*′a*::*euclidean-space*) ⊆
cball 0 (*sqrt*(*DIM*(*′a*)) ∗ *r*)
proof
fix *x*::*′a*
have *norm x* ≤ *sqrt*(*DIM*(*′a*)) ∗ *infnorm x*
by (*rule norm-le-infnorm*)
also
assume *x* ∈ *cbox* (− *r* $*_R$ *One*) (*r* $*_R$ *One*)

hence *infnorm* $x \le r$
by (*auto simp*: *infnorm-def mem-box intro*!: *cSup-least*)
finally show $x \in$ *cball 0* (*sqrt*(*DIM*('*a*)) $*$ *r*)
by (*auto simp*: *dist-norm mult-left-mono mem-cball*)
qed

2.1 Solutions of IVPs

definition
solves-ode :: (*real* ⇒ '*a*::*real-normed-vector*) ⇒ (*real* ⇒ '*a* ⇒ '*a*) ⇒ *real set* ⇒ '*a set* ⇒ *bool*
(**infix** ‹(*solves'-ode*)› *50*)
where
(*y solves-ode f*) *T X* ⟷ (*y has-vderiv-on* (λ*t*. *f t* (*y t*))) *T* ∧ *y* ∈ *T* → *X*

lemma *solves-odeI*:
assumes *solves-ode-vderivD*: (*y has-vderiv-on* (λ*t*. *f t* (*y t*))) *T*
and *solves-ode-domainD*: ⋀*t*. *t* ∈ *T* ⟹ *y t* ∈ *X*
shows (*y solves-ode f*) *T X*
using *assms*
by (*auto simp*: *solves-ode-def*)

lemma *solves-odeD*:
assumes (*y solves-ode f*) *T X*
shows *solves-ode-vderivD*: (*y has-vderiv-on* (λ*t*. *f t* (*y t*))) *T*
and *solves-ode-domainD*: ⋀*t*. *t* ∈ *T* ⟹ *y t* ∈ *X*
using *assms*
by (*auto simp*: *solves-ode-def*)

lemma *solves-ode-continuous-on*: (*y solves-ode f*) *T X* ⟹ *continuous-on T y*
by (*auto intro*!: *vderiv-on-continuous-on simp*: *solves-ode-def*)

lemma *solves-ode-congI*:
assumes (*x solves-ode f*) *T X*
assumes ⋀*t*. *t* ∈ *T* ⟹ *x t* = *y t*
assumes ⋀*t*. *t* ∈ *T* ⟹ *f t* (*x t*) = *g t* (*x t*)
assumes *T* = *S X* = *Y*
shows (*y solves-ode g*) *S Y*
using *assms*
by (*auto simp*: *solves-ode-def Pi-iff*)

lemma *solves-ode-cong*[*cong*]:
assumes ⋀*t*. *t* ∈ *T* ⟹ *x t* = *y t*
assumes ⋀*t*. *t* ∈ *T* ⟹ *f t* (*x t*) = *g t* (*x t*)
assumes *T* = *S X* = *Y*
shows (*x solves-ode f*) *T X* ⟷ (*y solves-ode g*) *S Y*
using *assms*
by (*auto simp*: *solves-ode-def Pi-iff*)

lemma *solves-ode-on-subset*:
assumes (*x solves-ode f*) *S Y*
assumes $T \subseteq S$ $Y \subseteq X$
shows (*x solves-ode f*) *T X*
using *assms*
by (*auto simp*: *solves-ode-def has-vderiv-on-subset*)

lemma *preflect-solution*:
assumes $t0 \in T$
assumes *sol*: ((λt. *x* (*preflect t0 t*)) *solves-ode* (λt *x*. $-$ *f* (*preflect t0 t*) *x*)) (*preflect t0* ‘ *T*) *X*
shows (*x solves-ode f*) *T X*
proof (*rule solves-odeI*)
from *solves-odeD*[*OF sol*]
have *xm-deriv*: (*x o preflect t0 has-vderiv-on* (λt. $-$ *f* (*preflect t0 t*) (*x* (*preflect t0 t*)))) (*preflect t0* ‘ *T*)
and *xm-mem*: $t \in$ *preflect t0* ‘ $T \Longrightarrow$ *x* (*preflect t0 t*) $\in X$ **for** *t*
by *simp-all*
have (*x o preflect t0 o preflect t0 has-vderiv-on* (λt. *f t* (*x t*))) *T*
apply (*rule has-vderiv-on-eq-rhs*)
apply (*rule has-vderiv-on-compose*)
apply (*rule xm-deriv*)
apply (*auto simp*: *preflect-def intro*!: *derivative-intros*)
done
then show (*x has-vderiv-on* (λt. *f t* (*x t*))) *T*
by (*simp add*: *preflect-def*)
show *x t* $\in X$ **if** $t \in T$ **for** *t*
using *that xm-mem*[*of preflect t0 t*]
by (*auto simp*: *preflect-def*)
qed

lemma *solution-preflect*:
assumes $t0 \in T$
assumes *sol*: (*x solves-ode f*) *T X*
shows ((λt. *x* (*preflect t0 t*)) *solves-ode* (λt *x*. $-$ *f* (*preflect t0 t*) *x*)) (*preflect t0* ‘ *T*) *X*
using *sol* ‹$t0 \in T$›
by (*simp-all add*: *preflect-def image-image preflect-solution*[*of t0*])

lemma *solution-eq-preflect-solution*:
assumes $t0 \in T$
shows (*x solves-ode f*) *T X* $\longleftrightarrow$ ((λt. *x* (*preflect t0 t*)) *solves-ode* (λt *x*. $-$ *f* (*preflect t0 t*) *x*)) (*preflect t0* ‘ *T*) *X*
using *solution-preflect*[*OF* ‹$t0 \in T$›] *preflect-solution*[*OF* ‹$t0 \in T$›]
by *blast*

lemma *shift-autonomous-solution*:
assumes *sol*: (*x solves-ode f*) *T X*
assumes *auto*: $\bigwedge s\ t.\ s \in T \Longrightarrow f\ s\ (x\ s) = f\ t\ (x\ s)$

shows ((λ*t*. *x* (*t* + *t0*)) *solves-ode f*) ((λ*t*. *t* − *t0*) ‘ *T*) *X*
using *solves-odeD*[*OF sol*]
apply (*intro solves-odeI*)
apply (*rule has-vderiv-on-compose′*[*of x*, *THEN has-vderiv-on-eq-rhs*])
apply (*auto simp*: *image-image intro*!: *auto derivative-intros*)
done

lemma *solves-ode-singleton*: *y t0* ∈ *X* ⟹ (*y solves-ode f*) {*t0*} *X*
by (*auto intro*!: *solves-odeI has-vderiv-on-singleton*)

2.1.1 Connecting solutions

lemma *connection-solves-ode*:
assumes *x*: (*x solves-ode f*) *T X*
assumes *y*: (*y solves-ode g*) *S Y*
assumes *conn-T*: *closure S* ∩ *closure T* ⊆ *T*
assumes *conn-S*: *closure S* ∩ *closure T* ⊆ *S*
assumes *conn-x*: ⋀*t*. *t* ∈ *closure S* ⟹ *t* ∈ *closure T* ⟹ *x t* = *y t*
assumes *conn-f*: ⋀*t*. *t* ∈ *closure S* ⟹ *t* ∈ *closure T* ⟹ *f t* (*y t*) = *g t* (*y t*)
shows ((λ*t*. *if t* ∈ *T then x t else y t*) *solves-ode* (λ*t*. *if t* ∈ *T then f t else g t*)) (*T* ∪ *S*) (*X* ∪ *Y*)
proof (*rule solves-odeI*)
from *solves-odeD*(*2*)[*OF x*] *solves-odeD*(*2*)[*OF y*]
show *t* ∈ *T* ∪ *S* ⟹ (*if t* ∈ *T then x t else y t*) ∈ *X* ∪ *Y* **for** *t*
by *auto*
show ((λ*t*. *if t* ∈ *T then x t else y t*) *has-vderiv-on* (λ*t*. (*if t* ∈ *T then f t else g t*) (*if t* ∈ *T then x t else y t*))) (*T* ∪ *S*)
apply (*rule has-vderiv-on-If*[*OF refl*, *THEN has-vderiv-on-eq-rhs*])
unfolding *Un-absorb2*[*OF conn-T*] *Un-absorb2*[*OF conn-S*]
apply (*rule solves-odeD*(*1*)[*OF x*])
apply (*rule solves-odeD*(*1*)[*OF y*])
apply (*simp-all add*: *conn-T conn-S Un-absorb2 conn-x conn-f*)
done
qed

lemma
solves-ode-subset-range:
assumes *x*: (*x solves-ode f*) *T X*
assumes *s*: *x* ‘ *T* ⊆ *Y*
shows (*x solves-ode f*) *T Y*
using *assms*
by (*auto intro*!: *solves-odeI dest*!: *solves-odeD*)

2.2 unique solution with initial value

definition
usolves-ode-from :: (*real* ⇒ ′*a*::*real-normed-vector*) ⇒ (*real* ⇒ ′*a* ⇒ ′*a*) ⇒ *real* ⇒ *real set* ⇒ ′*a set* ⇒ *bool*
(‹((-) *usolves′-ode* (-) *from* (-))› [*10*, *10*, *10*] *10*)

— TODO: no idea about mixfix and precedences, check this!
where
$(y\ \textit{usolves-ode}\ f\ \textit{from}\ t0)\ T\ X \longleftrightarrow (y\ \textit{solves-ode}\ f)\ T\ X \wedge t0 \in T \wedge \textit{is-interval}\ T \wedge$
$(\forall z\ T'.\ t0 \in T' \wedge \textit{is-interval}\ T' \wedge T' \subseteq T \wedge (z\ \textit{solves-ode}\ f)\ T'\ X \longrightarrow z\ t0 = y\ t0 \longrightarrow (\forall t \in T'.\ z\ t = y\ t))$

uniqueness of solution can depend on domain X:

lemma
$((\lambda\text{-}.\ 0{::}\textit{real})\ \textit{usolves-ode}\ (\lambda\text{-}.\ \textit{sqrt})\ \textit{from}\ 0)\ \{0..\}\ \{0\}$
$((\lambda t.\ t^2\ /\ 4)\ \textit{solves-ode}\ (\lambda\text{-}.\ \textit{sqrt}))\ \{0..\}\ \{0..\}$
$(\lambda t.\ t^2\ /\ 4)\ 0 = (\lambda\text{-}.\ 0{::}\textit{real})\ 0$
by (*auto intro*!: *derivative-eq-intros*
simp: *has-vderiv-on-def has-vector-derivative-def usolves-ode-from-def solves-ode-def is-interval-ci real-sqrt-divide*)

TODO: show that if solution stays in interior, then domain can be enlarged! (?)

lemma *usolves-odeD*:
assumes $(y\ \textit{usolves-ode}\ f\ \textit{from}\ t0)\ T\ X$
shows $(y\ \textit{solves-ode}\ f)\ T\ X$
and $t0 \in T$
and $\textit{is-interval}\ T$
and $\bigwedge z\ T'\ t.\ t0 \in T' \Longrightarrow \textit{is-interval}\ T' \Longrightarrow T' \subseteq T \Longrightarrow (z\ \textit{solves-ode}\ f)\ T'\ X \Longrightarrow z\ t0 = y\ t0 \Longrightarrow t \in T' \Longrightarrow z\ t = y\ t$
using *assms*
unfolding *usolves-ode-from-def*
by *blast*+

lemma *usolves-ode-rawI*:
assumes $(y\ \textit{solves-ode}\ f)\ T\ X\ t0 \in T\ \textit{is-interval}\ T$
assumes $\bigwedge z\ T'\ t.\ t0 \in T' \Longrightarrow \textit{is-interval}\ T' \Longrightarrow T' \subseteq T \Longrightarrow (z\ \textit{solves-ode}\ f)\ T'\ X \Longrightarrow z\ t0 = y\ t0 \Longrightarrow t \in T' \Longrightarrow z\ t = y\ t$
shows $(y\ \textit{usolves-ode}\ f\ \textit{from}\ t0)\ T\ X$
using *assms*
unfolding *usolves-ode-from-def*
by *blast*

lemma *usolves-odeI*:
assumes $(y\ \textit{solves-ode}\ f)\ T\ X\ t0 \in T\ \textit{is-interval}\ T$
assumes *usol*: $\bigwedge z\ t.\ \{t0\ {-}{-}\ t\} \subseteq T \Longrightarrow (z\ \textit{solves-ode}\ f)\ \{t0\ {-}{-}\ t\}\ X \Longrightarrow z\ t0 = y\ t0 \Longrightarrow z\ t = y\ t$
shows $(y\ \textit{usolves-ode}\ f\ \textit{from}\ t0)\ T\ X$
proof (*rule usolves-ode-rawI*; *fact?*)
fix $z\ T'\ t$
assume T': $t0 \in T'\ \textit{is-interval}\ T'\ T' \subseteq T$
and *z*: $(z\ \textit{solves-ode}\ f)\ T'\ X$ **and** *iv*: $z\ t0 = y\ t0$ **and** *t*: $t \in T'$
have *subset-T'*: $\{t0\ {-}{-}\ t\} \subseteq T'$
by (*rule closed-segment-subset-interval*; *fact*)

with *z* **have** *sol-cs*: (*z solves-ode f*) {*t0* −− *t*} *X*
by (*rule solves-ode-on-subset*[*OF* - - *order-refl*])
from *subset-T′* **have** *subset-T*: {*t0* −− *t*} ⊆ *T*
using ‹*T′* ⊆ *T*› **by** *simp*
from *usol*[*OF subset-T sol-cs iv*]
show *z t* = *y t* **by** *simp*
qed

lemma *is-interval-singleton*[*intro*,*simp*]: *is-interval* {*t0*}
by (*auto simp*: *is-interval-def intro*!: *euclidean-eqI*[**where** ′*a*=′*a*])

lemma *usolves-ode-singleton*: *x t0* ∈ *X* ⟹ (*x usolves-ode f from t0*) {*t0*} *X*
by (*auto intro*!: *usolves-odeI solves-ode-singleton*)

lemma *usolves-ode-congI*:
assumes *x*: (*x usolves-ode f from t0*) *T X*
assumes ⋀*t*. *t* ∈ *T* ⟹ *x t* = *y t*
assumes ⋀*t y*. *t* ∈ *T* ⟹ *y* ∈ *X* ⟹ *f t y* = *g t y*— TODO: weaken this assumption?!
assumes *t0* = *s0*
assumes *T* = *S*
assumes *X* = *Y*
shows (*y usolves-ode g from s0*) *S Y*
proof (*rule usolves-ode-rawI*)
from *assms x* **have** (*y solves-ode f*) *S Y*
by (*auto simp add*: *usolves-ode-from-def*)
then show (*y solves-ode g*) *S Y*
by (*rule solves-ode-congI*) (*use assms* **in** ‹*auto simp*: *usolves-ode-from-def dest*!: *solves-ode-domainD*›)
from *assms* **show** *s0* ∈ *S is-interval S*
by (*auto simp add*: *usolves-ode-from-def*)
next
fix *z T′ t*
assume *hyps*: *s0* ∈ *T′ is-interval T′ T′* ⊆ *S* (*z solves-ode g*) *T′ Y z s0* = *y s0* *t* ∈ *T′*
from ‹(*z solves-ode g*) *T′ Y*›
have *zsol*: (*z solves-ode f*) *T′ Y*
by (*rule solves-ode-congI*) (*use assms hyps* **in** ‹*auto dest*!: *solves-ode-domainD*›)
have *z t* = *x t*
by (*rule x*[*THEN usolves-odeD*(*4*),**where** *T′* = *T′*])
(*use zsol* ‹*s0* ∈ *T′*› ‹*is-interval T′*› ‹*T′* ⊆ *S*› ‹*T* = *S*› ‹*z s0* = *y s0*› ‹*t* ∈ *T′*› *assms* **in** *auto*)
also have *y t* = *x t* **using** *assms* ‹*t* ∈ *T′*› ‹*T′* ⊆ *S*› ‹*T* = *S*› **by** *auto*
finally show *z t* = *y t* **by** *simp*
qed

lemma *usolves-ode-cong*[*cong*]:
assumes ⋀*t*. *t* ∈ *T* ⟹ *x t* = *y t*

assumes $\bigwedge t\ y.\ t \in T \Longrightarrow y \in X \Longrightarrow f\ t\ y = g\ t\ y$— TODO: weaken this assumption?!
assumes $t0 = s0$
assumes $T = S$
assumes $X = Y$
shows $(x$ *usolves-ode f from t0*$)$ $T\ X \longleftrightarrow (y$ *usolves-ode g from s0*$)$ $S\ Y$
apply (*rule iffI*)
subgoal by (*rule usolves-ode-congI*[*OF - assms*]; *assumption*)
subgoal by (*metis assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *assms*(*5*) *usolves-ode-congI*)
done

lemma *shift-autonomous-unique-solution*:
assumes *usol*: $(x$ *usolves-ode f from t0*$)$ $T\ X$
assumes *auto*: $\bigwedge s\ t\ x.\ x \in X \Longrightarrow f\ s\ x = f\ t\ x$
shows $((\lambda t.\ x\ (t + t0 - t1))$ *usolves-ode f from t1*$)$ $((+)\ (t1 - t0)$ ‘ $T)\ X$
proof (*rule usolves-ode-rawI*)
from *usolves-odeD*[*OF usol*]
have *sol*: $(x$ *solves-ode* $f)$ $T\ X$
and $t0 \in T$
and *is-interval* T
and *unique*: $t0 \in T' \Longrightarrow$ *is-interval* $T' \Longrightarrow T' \subseteq T \Longrightarrow (z$ *solves-ode* $f)$ $T'\ X \Longrightarrow z\ t0 = x\ t0 \Longrightarrow t \in T' \Longrightarrow z\ t = x\ t$
for $z\ T'\ t$
by *blast+*
have $(\lambda t.\ t + t1 - t0) = (+)\ (t1 - t0)$
by (*auto simp add*: *algebra-simps*)
with *shift-autonomous-solution*[*OF sol auto*, *of* $t0 - t1$] *solves-odeD*[*OF sol*]
show $((\lambda t.\ x\ (t + t0 - t1))$ *solves-ode* $f)$ $((+)\ (t1 - t0)$ ‘ $T)\ X$
by (*simp add*: *algebra-simps*)
from ‹$t0 \in T$› **show** $t1 \in (+)\ (t1 - t0)$ ‘ T **by** *auto*
from ‹*is-interval* T›
show *is-interval* $((+)\ (t1 - t0)$ ‘ $T)$
by *simp*
fix $z\ T'\ t$
assume z: $(z$ *solves-ode* $f)$ $T'\ X$
and $t0'$: $t1 \in T'\ T' \subseteq (+)\ (t1 - t0)$ ‘ T
and *shift*: $z\ t1 = x\ (t1 + t0 - t1)$
and t: $t \in T'$
and *ivl*: *is-interval* T'

let $?z = (\lambda t.\ z\ (t + (t1 - t0)))$

have $(?z$ *solves-ode* $f)$ $((\lambda t.\ t - (t1 - t0))$ ‘ $T')\ X$
apply (*rule shift-autonomous-solution*[*OF z*, *of* $t1 - t0$])
using *solves-odeD*[*OF z*]
by (*auto intro*!: *auto*)
with - - - **have** $?z\ ((t + (t0 - t1))) = x\ (t + (t0 - t1))$
apply (*rule unique*[**where** $z = ?z$])
using *shift t t0' ivl*

by *auto*
then show $z\ t = x\ (t + t0 - t1)$
by (*simp add*: *algebra-simps*)
qed

lemma *three-intervals-lemma*:
fixes *a b c*::*real*
assumes *a*: $a \in A - B$
and *b*: $b \in B - A$
and *c*: $c \in A \cap B$
and *iA*: *is-interval A* **and** *iB*: *is-interval B*
and *aI*: $a \in I$
and *bI*: $b \in I$
and *iI*: *is-interval I*
shows $c \in I$
apply (*rule mem-is-intervalI*[*OF iI aI bI*])
using *iA iB*
apply (*auto simp*: *is-interval-def*)
apply (*metis Diff-iff Int-iff a b c le-cases*)
apply (*metis Diff-iff Int-iff a b c le-cases*)
done

lemma *connection-usolves-ode*:
assumes *x*: (*x usolves-ode f from tx*) *T X*
assumes *y*: $\bigwedge t.\ t \in closure\ S \cap closure\ T \Longrightarrow$ (*y usolves-ode g from t*) *S X*
assumes *conn-T*: $closure\ S \cap closure\ T \subseteq T$
assumes *conn-S*: $closure\ S \cap closure\ T \subseteq S$
assumes *conn-t*: $t \in closure\ S \cap closure\ T$
assumes *conn-x*: $\bigwedge t.\ t \in closure\ S \Longrightarrow t \in closure\ T \Longrightarrow x\ t = y\ t$
assumes *conn-f*: $\bigwedge t\ x.\ t \in closure\ S \Longrightarrow t \in closure\ T \Longrightarrow x \in X \Longrightarrow f\ t\ x = g\ t\ x$
shows (($\lambda t.$ *if* $t \in T$ *then x t else y t*) *usolves-ode* ($\lambda t.$ *if* $t \in T$ *then f t else g t*) *from tx*) $(T \cup S)$ *X*
apply (*rule usolves-ode-rawI*)
apply (*subst Un-absorb*[*of X*, *symmetric*])
apply (*rule connection-solves-ode*[*OF usolves-odeD*(*1*)[*OF x*] *usolves-odeD*(*1*)[*OF y*[*OF conn-t*]] *conn-T conn-S conn-x conn-f*])
subgoal by *assumption*
subgoal by *assumption*
subgoal by *assumption*
subgoal by *assumption*
subgoal using *solves-odeD*(*2*)[*OF usolves-odeD*(*1*)[*OF x*]] *conn-T* **by** (*auto simp add*: *conn-x*[*symmetric*])
subgoal using *usolves-odeD*(*2*)[*OF x*] **by** *auto*
subgoal using *usolves-odeD*(*3*)[*OF x*] *usolves-odeD*(*3*)[*OF y*]
apply (*rule is-real-interval-union*)
using *conn-T conn-S conn-t* **by** *auto*
subgoal premises *prems* **for** *z TS′ s*
proof −

from ‹(*z solves-ode* -) - -›
have (*z solves-ode* (λt. *if* $t \in T$ *then* $f\ t$ *else* $g\ t$)) ($T \cap TS'$) X
  by (*rule solves-ode-on-subset*) *auto*
then have *z-f*: (*z solves-ode f*) ($T \cap TS'$) X
  by (*subst solves-ode-cong*) *auto*

from *prems*(*4*)
have (*z solves-ode* (λt. *if* $t \in T$ *then* $f\ t$ *else* $g\ t$)) ($S \cap TS'$) X
  by (*rule solves-ode-on-subset*) *auto*
then have *z-g*: (*z solves-ode g*) ($S \cap TS'$) X
  apply (*rule solves-ode-congI*)
  subgoal by *simp*
  subgoal by *clarsimp* (*meson closure-subset conn-f contra-subsetD prems*(*4*) *solves-ode-domainD*)
  subgoal by *simp*
  subgoal by *simp*
  done
have $tx \in T$ **using** *assms* **using** *usolves-odeD*(*2*)[*OF x*] **by** *auto*

have $z\ tx = x\ tx$ **using** *assms prems*
  by (*simp add*: ‹$tx \in T$›)

from *usolves-odeD*(*4*)[*OF x* - - - ‹(*z solves-ode f*) - -›, *of s*] *prems*
have $z\ s = x\ s$ **if** $s \in T$ **using** *that* ‹$tx \in T$› ‹$z\ tx = x\ tx$›
  by (*auto simp*: *is-interval-Int usolves-odeD*(*3*)[*OF x*] ‹*is-interval* TS'›)

moreover

{
  assume $s \notin T$
  then have $s \in S$ **using** *prems assms* **by** *auto*
  {
    assume $tx \notin S$
    then have $tx \in T - S$ **using** ‹$tx \in T$› **by** *simp*
    moreover have $s \in S - T$ **using** ‹$s \notin T$› ‹$s \in S$› **by** *blast*
    ultimately have $t \in TS'$
      apply (*rule three-intervals-lemma*)
      subgoal using *assms* **by** *auto*
      subgoal using *usolves-odeD*(*3*)[*OF x*] **.**
      subgoal using *usolves-odeD*(*3*)[*OF y*[*OF conn-t*]] **.**
      subgoal using ‹$tx \in TS'$› **.**
      subgoal using ‹$s \in TS'$› **.**
      subgoal using ‹*is-interval* TS'› **.**
      done
    with *assms* **have** *t*: $t \in$ *closure* $S \cap$ *closure* $T \cap TS'$ **by** *simp*

    then have $t \in S$ $t \in T$ $t \in TS'$ **using** *assms* **by** *auto*
    have $z\ t = x\ t$
      apply (*rule usolves-odeD*(*4*)[*OF x* - - - *z-f*, *of t*])

using ‹$t \in TS'$› ‹$t \in T$› *prems assms* ‹$tx \in T$› *usolves-odeD*(*3*)[*OF x*]
by (*auto intro*!: *is-interval-Int*)
with *assms* **have** $z\ t = y\ t$ **using** *t* **by** *auto*

from *usolves-odeD*(*4*)[*OF y*[*OF conn-t*] - - - *z-g*, *of s*] *prems*
have $z\ s = y\ s$ **using** ‹$s \notin T$› *assms* ‹$z\ t = y\ t$› *t* ‹$t \in S$›
‹*is-interval TS'*› *usolves-odeD*(*3*)[*OF y*[*OF conn-t*]]
by (*auto simp*: *is-interval-Int*)
} moreover {
assume $tx \in S$
with *prems closure-subset* ‹$tx \in T$›
have *tx*: $tx \in closure\ S \cap closure\ T \cap TS'$ **by** *force*

then have $tx \in S$ $tx \in T$ $tx \in TS'$ **using** *assms* **by** *auto*
have $z\ tx = x\ tx$
apply (*rule usolves-odeD*(*4*)[*OF x* - - - *z-f*, *of tx*])
using ‹$tx \in TS'$› ‹$tx \in T$› *prems assms* ‹$tx \in T$› *usolves-odeD*(*3*)[*OF x*]
by (*auto intro*!: *is-interval-Int*)
with *assms* **have** $z\ tx = y\ tx$ **using** *tx* **by** *auto*

from *usolves-odeD*(*4*)[*OF y*[**where** *t*=*tx*] - - - *z-g*, *of s*] *prems*
have $z\ s = y\ s$ **using** ‹$s \notin T$› *assms* ‹$z\ tx = y\ tx$› *tx* ‹$tx \in S$›
‹*is-interval TS'*› *usolves-odeD*(*3*)[*OF y*]
by (*auto simp*: *is-interval-Int*)
} ultimately have $z\ s = y\ s$ **by** *blast*
}
ultimately
show $z\ s =$ (*if* $s \in T$ *then* $x\ s$ *else* $y\ s$) **by** *simp*
qed
done

lemma *usolves-ode-union-closed*:
assumes *x*: (*x usolves-ode f from tx*) *T X*
assumes *y*: $\bigwedge t.\ t \in closure\ S \cap closure\ T \Longrightarrow$ (*x usolves-ode f from t*) *S X*
assumes *conn-T*: $closure\ S \cap closure\ T \subseteq T$
assumes *conn-S*: $closure\ S \cap closure\ T \subseteq S$
assumes *conn-t*: $t \in closure\ S \cap closure\ T$
shows (*x usolves-ode f from tx*) ($T \cup S$) *X*
using *connection-usolves-ode*[*OF assms*] **by** *simp*

lemma *usolves-ode-solves-odeI*:
assumes (*x usolves-ode f from tx*) *T X*
assumes (*y solves-ode f*) *T X* $y\ tx = x\ tx$
shows (*y usolves-ode f from tx*) *T X*
using *assms*(*1*)
apply (*rule usolves-ode-congI*)
subgoal using *assms* **by** (*metis set-eq-subset usolves-odeD*(*2*) *usolves-odeD*(*3*) *usolves-odeD*(*4*))
by *auto*

lemma *usolves-ode-subset-range*:
 assumes *x*: (*x usolves-ode f from t0*) *T X*
 assumes *r*: *x* ' *T* ⊆ *Y* **and** *Y* ⊆ *X*
 shows (*x usolves-ode f from t0*) *T Y*
proof (*rule usolves-odeI*)
 note *usolves-odeD*[*OF x*]
 show (*x solves-ode f*) *T Y* **by** (*rule solves-ode-subset-range*; *fact*)
 show *t0* ∈ *T is-interval T* **by** *fact*+
 fix *z t*
 assume *s*: {*t0* −− *t*} ⊆ *T* **and** *z*: (*z solves-ode f*) {*t0* −− *t*} *Y* **and** *z0*: *z t0* = *x t0*
 then have *t0* ∈ {*t0* −− *t*} *is-interval* {*t0* −− *t*}
  by *auto*
 moreover note *s*
 moreover have (*z solves-ode f*) {*t0*−−*t*} *X*
  using *solves-odeD*[*OF z*] ‹*Y* ⊆ *X*›
  by (*intro solves-ode-subset-range*[*OF z*]) *force*
 moreover note *z0*
 moreover have *t* ∈ {*t0* −− *t*} **by** *simp*
 ultimately show *z t* = *x t*
  by (*rule usolves-odeD*[*OF x*])
qed

2.3 ivp on interval

context
 fixes *t0 t1*::*real* **and** *T*
 defines *T* ≡ *closed-segment t0 t1*
begin

lemma *is-solution-ext-cont*:
 continuous-on T x ⟹ (*ext-cont x* (*min t0 t1*) (*max t0 t1*) *solves-ode f*) *T X* = (*x solves-ode f*) *T X*
 by (*rule solves-ode-cong*) (*auto simp add*: *T-def min-def max-def closed-segment-eq-real-ivl*)

lemma *solution-fixed-point*:
 fixes *x*:: *real* ⇒ *'a*::*banach*
 assumes *x*: (*x solves-ode f*) *T X* **and** *t*: *t* ∈ *T*
 shows *x t0* + *ivl-integral t0 t* (λ*t*. *f t* (*x t*)) = *x t*
proof −
 from *solves-odeD*(*1*)[*OF x*, *unfolded T-def*]
 have (*x has-vderiv-on* (λ*t*. *f t* (*x t*))) (*closed-segment t0 t*)
  by (*rule has-vderiv-on-subset*) (*insert* ‹*t* ∈ *T*›, *auto simp*: *closed-segment-eq-real-ivl T-def*)
 from *fundamental-theorem-of-calculus-ivl-integral*[*OF this*]
 have ((λ*t*. *f t* (*x t*)) *has-ivl-integral x t* − *x t0*) *t0 t* **.**
 from *this*[*THEN ivl-integral-unique*]
 show *?thesis* **by** *simp*

qed

lemma *solution-fixed-point-left*:
  fixes *x*:: *real* ⇒ *'a*::*banach*
  assumes *x*: (*x solves-ode f*) *T X* **and** *t*: *t* ∈ *T*
  shows *x t1* − *ivl-integral t t1* (λ*t*. *f t* (*x t*)) = *x t*
proof −
  from *solves-odeD*(*1*)[*OF x*, *unfolded T-def*]
  have (*x has-vderiv-on* (λ*t*. *f t* (*x t*))) (*closed-segment t t1*)
   by (*rule has-vderiv-on-subset*) (*insert* ‹*t* ∈ *T*›, *auto simp*: *closed-segment-eq-real-ivl T-def*)
  from *fundamental-theorem-of-calculus-ivl-integral*[*OF this*]
  have ((λ*t*. *f t* (*x t*)) *has-ivl-integral x t1* − *x t*) *t t1* **.**
  from *this*[*THEN ivl-integral-unique*]
  show *?thesis* **by** *simp*
qed

lemma *solution-fixed-pointI*:
  fixes *x*:: *real* ⇒ *'a*::*banach*
  assumes *cont-f*: *continuous-on* (*T* × *X*) (λ(*t*, *x*). *f t x*)
  assumes *cont-x*: *continuous-on T x*
  assumes *defined*: ⋀*t*. *t* ∈ *T* ⟹ *x t* ∈ *X*
  assumes *fp*: ⋀*t*. *t* ∈ *T* ⟹ *x t* = *x t0* + *ivl-integral t0 t* (λ*t*. *f t* (*x t*))
  shows (*x solves-ode f*) *T X*
proof (*rule solves-odeI*)
  note [*continuous-intros*] = *continuous-on-compose-Pair*[*OF cont-f*]
  have ((λ*t*. *x t0* + *ivl-integral t0 t* (λ*t*. *f t* (*x t*))) *has-vderiv-on* (λ*t*. *f t* (*x t*))) *T*
    using *cont-x defined*
    by (*auto intro*!: *derivative-eq-intros ivl-integral-has-vector-derivative*
      continuous-intros
      simp: *has-vderiv-on-def T-def*)
  with *fp* **show** (*x has-vderiv-on* (λ*t*. *f t* (*x t*))) *T* **by** *simp*
qed (*simp add*: *defined*)

end

lemma *solves-ode-half-open-segment-continuation*:
  fixes *f*::*real* ⇒ *'a* ⇒ *'a*::*banach*
  assumes *ode*: (*x solves-ode f*) {*t0* −−< *t1*} *X*
  assumes *continuous*: *continuous-on* ({*t0* −− *t1*} × *X*) (λ(*t*, *x*). *f t x*)
  assumes *compact X*
  assumes *t0* ≠ *t1*
  obtains *l* **where**
    (*x* ⟶ *l*) (*at t1 within* {*t0* −−< *t1*})
    ((λ*t*. *if t* = *t1 then l else x t*) *solves-ode f*) {*t0* −− *t1*} *X*
proof −
  note [*continuous-intros*] = *continuous-on-compose-Pair*[*OF continuous*]
  have *compact* ((λ(*t*, *x*). *f t x*) ‘ ({*t0* −− *t1*} × *X*))
     by (*auto intro*!: *compact-continuous-image continuous-intros compact-Times*

‹*compact X*›
simp: *split-beta*)
then obtain *B* **where** *B* > *0* **and** *B*: ⋀*t x*. *t* ∈ {*t0* −− *t1*} ⟹ *x* ∈ *X* ⟹ *norm* (*f t x*) ≤ *B*
by (*auto dest*!: *compact-imp-bounded simp*: *bounded-pos*)

have *uc*: *uniformly-continuous-on* {*t0* −−< *t1*} *x*
apply (*rule lipschitz-on-uniformly-continuous*[**where** *L*=*B*])
apply (*rule bounded-vderiv-on-imp-lipschitz*)
apply (*rule solves-odeD*[*OF ode*])
using *solves-odeD*(*2*)[*OF ode*] ‹*0* < *B*›
by (*auto simp*: *closed-segment-eq-real-ivl half-open-segment-real subset-iff intro*!: *B split*: *if-split-asm*)

have *t1* ∈ *closure* ({*t0* −−< *t1*})
using *closure-half-open-segment*[*of t0 t1*] ‹*t0* ≠ *t1*›
by *simp*
from *uniformly-continuous-on-extension-on-closure*[*OF uc*]
obtain *g* **where** *uc-g*: *uniformly-continuous-on* {*t0*−−*t1*} *g*
and *xg*: (⋀*t*. *t* ∈ {*t0* −−< *t1*} ⟹ *x t* = *g t*)
using *closure-half-open-segment*[*of t0 t1*] ‹*t0* ≠ *t1*›
by *metis*

from *uc-g*[*THEN uniformly-continuous-imp-continuous*, *unfolded continuous-on-def*]
have (*g* ⟶ *g t*) (*at t within* {*t0*−−*t1*}) **if** *t*∈{*t0*−−*t1*} **for** *t*
using *that* **by** *auto*
then have *g-tendsto*: (*g* ⟶ *g t*) (*at t within* {*t0*−−<*t1*}) **if** *t*∈{*t0*−−*t1*} **for** *t*
using *that* **by** (*auto intro*: *tendsto-within-subset half-open-segment-subset*)
then have *x-tendsto*: (*x* ⟶ *g t*) (*at t within* {*t0*−−<*t1*}) **if** *t*∈{*t0*−−*t1*} **for** *t*
using *that*
by (*subst Lim-cong-within*[*OF refl refl refl xg*]) *auto*
then have (*x* ⟶ *g t1*) (*at t1 within* {*t0* −−< *t1*})
by *auto*
moreover
have *nbot*: *at s within* {*t0*−−<*t1*} ≠ *bot* **if** *s* ∈ {*t0*−−*t1*} **for** *s*
using *that* ‹*t0* ≠ *t1*›
by (*auto simp*: *trivial-limit-within islimpt-half-open-segment*)
have *g-mem*: *s* ∈ {*t0*−−*t1*} ⟹ *g s* ∈ *X* **for** *s*
apply (*rule Lim-in-closed-set*[*OF compact-imp-closed*[*OF* ‹*compact X*›] - - *x-tendsto*])
using *solves-odeD*(*2*)[*OF ode*] ‹*t0* ≠ *t1*›
by (*auto intro*!: *simp*: *eventually-at-filter nbot*)
have (*g solves-ode f*) {*t0* −− *t1*} *X*
apply (*rule solution-fixed-pointI*[*OF continuous*])
subgoal by (*auto intro*!: *uc-g uniformly-continuous-imp-continuous*)
subgoal by (*rule g-mem*)
subgoal premises *prems* **for** *s*

```
  proof –
    {
      fix s
      assume s: s ∈ {t0−−<t1}
      with prems have subs: {t0−−s} ⊆ {t0−−<t1}
        by (auto simp: half-open-segment-real closed-segment-eq-real-ivl)
      with ode have sol: (x solves-ode f) ({t0−−s}) X
        by (rule solves-ode-on-subset) (rule order-refl)
      from subs have inner-eq: t ∈ {t0 −− s} ⟹ x t = g t for t
        by (intro xg) auto
      from solution-fixed-point[OF sol, of s]
      have g t0 + ivl-integral t0 s (λt. f t (g t)) − g s = 0
        using s prems ‹t0 ≠ t1›
        by (auto simp: inner-eq cong: ivl-integral-cong)
    } note fp = this

    from prems have subs: {t0−−s} ⊆ {t0−−t1}
      by (auto simp: closed-segment-eq-real-ivl)
    have int: (λt. f t (g t)) integrable-on {t0−−t1}
      using prems subs
    by (auto intro!: integrable-continuous-closed-segment continuous-intros g-mem
     uc-g[THEN uniformly-continuous-imp-continuous, THEN continuous-on-subset])
    note ivl-tendsto[tendsto-intros] =
         indefinite-ivl-integral-continuous(1)[OF int, unfolded continuous-on-def,
rule-format]

    from subs half-open-segment-subset
    have ((λs. g t0 + ivl-integral t0 s (λt. f t (g t)) − g s) ⟶
      g t0 + ivl-integral t0 s (λt. f t (g t)) − g s) (at s within {t0 −−< t1})
      using subs
      by (auto intro!: tendsto-intros ivl-tendsto[THEN tendsto-within-subset]
        g-tendsto[THEN tendsto-within-subset])
    moreover
    have ((λs. g t0 + ivl-integral t0 s (λt. f t (g t)) − g s) ⟶ 0) (at s within
{t0 −−< t1})
      apply (subst Lim-cong-within[OF refl refl refl, where g=λ-. 0])
      subgoal by (subst fp) auto
      subgoal by simp
      done
    ultimately have g t0 + ivl-integral t0 s (λt. f t (g t)) − g s = 0
      using nbot prems tendsto-unique by blast
    then show g s = g t0 + ivl-integral t0 s (λt. f t (g t)) by simp
  qed
  done
then have ((λt. if t = t1 then g t1 else x t) solves-ode f) {t0−−t1} X
  apply (rule solves-ode-congI)
  using xg ‹t0 ≠ t1›
  by (auto simp: half-open-segment-closed-segmentI)
ultimately show ?thesis ..
```

qed

2.4 Picard-Lindeloef on set of functions into closed set

locale *continuous-rhs* = **fixes** *T X f*
  assumes *continuous*: *continuous-on* ($T \times X$) ($\lambda(t, x)$. *f t x*)
begin

lemma *continuous-rhs-comp*[*continuous-intros*]:
  assumes [*continuous-intros*]: *continuous-on S g*
  assumes [*continuous-intros*]: *continuous-on S h*
  assumes *g* ‘ $S \subseteq T$
  assumes *h* ‘ $S \subseteq X$
  shows *continuous-on S* (λx. *f* (*g x*) (*h x*))
  using *continuous-on-compose-Pair*[*OF continuous assms*(*1*,*2*)] *assms*(*3*,*4*)
  by *auto*

end

locale *global-lipschitz* =
  fixes *T X f* **and** *L*::*real*
  assumes *lipschitz*: $\bigwedge t.\ t \in T \Longrightarrow L$−*lipschitz-on X* ($\lambda x$. *f t x*)

locale *closed-domain* =
  fixes *X* **assumes** *closed*: *closed X*

locale *interval* = **fixes** *T*::*real set*
  assumes *interval*: *is-interval T*
begin

lemma *closed-segment-subset-domain*: $t0 \in T \Longrightarrow t \in T \Longrightarrow$ *closed-segment t0 t* $\subseteq T$
  by (*simp add*: *closed-segment-subset-interval interval*)

lemma *closed-segment-subset-domainI*: $t0 \in T \Longrightarrow t \in T \Longrightarrow s \in$ *closed-segment t0 t* $\Longrightarrow s \in T$
  using *closed-segment-subset-domain* **by** *force*

lemma *convex*[*intro*, *simp*]: *convex T*
  and *connected*[*intro*, *simp*]: *connected T*
  by (*simp-all add*: *interval is-interval-connected is-interval-convex*)

end

locale *nonempty-set* = **fixes** *T* **assumes** *nonempty-set*: $T \neq \{\}$

locale *compact-interval* = *interval* + *nonempty-set T* +
  assumes *compact-time*: *compact T*
begin

definition *tmin* = *Inf T*
definition *tmax* = *Sup T*

lemma
shows *tmin*: $t \in T \Longrightarrow tmin \leq t$ $tmin \in T$
and *tmax*: $t \in T \Longrightarrow t \leq tmax$ $tmax \in T$
using *nonempty-set*
by (*auto intro*!: *cInf-lower cSup-upper bounded-imp-bdd-below bounded-imp-bdd-above compact-imp-bounded compact-time closed-contains-Inf closed-contains-Sup compact-imp-closed*
simp: *tmin-def tmax-def*)

lemma *tmin-le-tmax*[*intro*, *simp*]: $tmin \leq tmax$
using *nonempty-set tmin tmax* **by** *auto*

lemma *T-def*: $T = \{tmin .. tmax\}$
using *closed-segment-subset-interval*[*OF interval tmin*(*2*) *tmax*(*2*)]
by (*auto simp*: *closed-segment-eq-real-ivl subset-iff intro*!: *tmin tmax*)

lemma *mem-T-I*[*intro*, *simp*]: $tmin \leq t \Longrightarrow t \leq tmax \Longrightarrow t \in T$
using *interval mem-is-interval-1-I tmax*(*2*) *tmin*(*2*) **by** *blast*

end

locale *self-mapping* = *interval T* **for** *T* +
fixes *t0*::*real* **and** *x0 f X*
assumes *iv-defined*: $t0 \in T$ $x0 \in X$
assumes *self-mapping*:
$\bigwedge x\ t.\ t \in T \Longrightarrow x\ t0 = x0 \Longrightarrow x \in closed\text{-}segment\ t0\ t \rightarrow X \Longrightarrow$ $continuous\text{-}on\ (closed\text{-}segment\ t0\ t)\ x \Longrightarrow x\ t0 + ivl\text{-}integral\ t0\ t\ (\lambda t.\ f\ t\ (x\ t)) \in X$
begin

sublocale *nonempty-set T* **using** *iv-defined* **by** *unfold-locales auto*

lemma *closed-segment-iv-subset-domain*: $t \in T \Longrightarrow closed\text{-}segment\ t0\ t \subseteq T$
by (*simp add*: *closed-segment-subset-domain iv-defined*)

end

locale *unique-on-closed* =
compact-interval T +
self-mapping T t0 x0 f X +
continuous-rhs T X f +
closed-domain X +
global-lipschitz T X f L **for** *t0*::*real* **and** *T* **and** *x0*::*'a*::*banach* **and** *f X L*
begin

lemma *T-split*: $T = \{tmin .. t0\} \cup \{t0 .. tmax\}$
  by (*metis T-def atLeastAtMost-iff iv-defined*(*1*) *ivl-disj-un-two-touch*(*4*))

lemma *L-nonneg*: $0 \leq L$
  by (*auto intro*!: *lipschitz-on-nonneg*[*OF lipschitz*] *iv-defined*)

Picard Iteration

definition *P-inner* **where** *P-inner x t* = *x0* + *ivl-integral t0 t* (λt. *f t* (*x t*))

definition *P*::(*real* $\Rightarrow_C$ *'a*) $\Rightarrow$ (*real* $\Rightarrow_C$ *'a*)
  where *P x* = (*SOME g*::*real*$\Rightarrow_C$ *'a*.
    ($\forall$ *t* $\in$ *T*. *g t* = *P-inner x t*) $\wedge$
    ($\forall$ *t*$\leq$*tmin*. *g t* = *P-inner x tmin*) $\wedge$
    ($\forall$ *t*$\geq$*tmax*. *g t* = *P-inner x tmax*))

lemma *cont-P-inner-ivl*:
  x $\in$ *T* $\rightarrow_C$ *X* $\Longrightarrow$ *continuous-on* {*tmin..tmax*} (*P-inner* (*apply-bcontfun x*))
  apply (*auto simp*: *real-Icc-closed-segment P-inner-def Pi-iff mem-PiC-iff*
      intro!: *continuous-intros indefinite-ivl-integral-continuous-subset*
      integrable-continuous-closed-segment tmin(*1*) *tmax*(*1*))
  using *closed-segment-subset-domainI tmax*(*2*) *tmin*(*2*) **apply** *blast*
  using *closed-segment-subset-domainI tmax*(*2*) *tmin*(*2*) **apply** *blast*
  using *T-def closed-segment-eq-real-ivl iv-defined*(*1*) **by** *auto*

lemma *P-inner-t0*[*simp*]: *P-inner g t0* = *x0*
  by (*simp add*: *P-inner-def*)

lemma *t0-cs-tmin-tmax*: *t0* $\in$ {*tmin*−−*tmax*} **and** *cs-tmin-tmax-subset*: {*tmin*−−*tmax*} $\subseteq$ *T*
  using *iv-defined T-def closed-segment-eq-real-ivl*
  by *auto*

lemma
  P-eqs:
  assumes *x* $\in$ *T* $\rightarrow_C$ *X*
  shows *P-eq-P-inner*: *t* $\in$ *T* $\Longrightarrow$ *P x t* = *P-inner x t*
    and *P-le-tmin*: *t* $\leq$ *tmin* $\Longrightarrow$ *P x t* = *P-inner x tmin*
    and *P-ge-tmax*: *t* $\geq$ *tmax* $\Longrightarrow$ *P x t* = *P-inner x tmax*
  unfolding *atomize-conj atomize-imp*
proof *goal-cases*
  case *1*
  obtain *g* **where**
    t $\in$ {*tmin* .. *tmax*} $\Longrightarrow$ *apply-bcontfun g t* = *P-inner* (*apply-bcontfun x*) *t*
    apply-bcontfun g t = *P-inner* (*apply-bcontfun x*) (*clamp tmin tmax t*)
    for *t*
   by (*metis continuous-on-cbox-bcontfunE cont-P-inner-ivl*[*OF assms*(*1*)] *cbox-interval*)
  with *T-def* **have** $\exists$ *g*::*real*$\Rightarrow_C$ *'a*.
    ($\forall$ *t* $\in$ *T*. *g t* = *P-inner x t*) $\wedge$
    ($\forall$ *t*$\leq$*tmin*. *g t* = *P-inner x tmin*) $\wedge$

(∀ *t*≥*tmax*. *g t* = *P-inner x tmax*)
by (*auto intro*!: *exI*[**where** *x*=*g*])
then have (∀ *t* ∈ *T*. *P x t* = *P-inner x t*) ∧
(∀ *t*≤*tmin*. *P x t* = *P-inner x tmin*) ∧
(∀ *t*≥*tmax*. *P x t* = *P-inner x tmax*)
unfolding *P-def*
by (*rule someI-ex*)
then show *?case* **using** *T-def* **by** *auto*
qed

lemma *P-if-eq*:
$x \in T \to_C X \Longrightarrow$
P x t = (*if tmin* ≤ *t* ∧ *t* ≤ *tmax then P-inner x t else if t* ≥ *tmax then P-inner x tmax else P-inner x tmin*)
by (*auto simp*: *P-eqs*)

lemma *dist-P-le*:
assumes *y*: $y \in T \to_C X$ **and** *z*: $z \in T \to_C X$
assumes *le*: ⋀*t*. *tmin* ≤ *t* ⟹ *t* ≤ *tmax* ⟹ *dist* (*P-inner y t*) (*P-inner z t*) ≤ *R*
assumes *0* ≤ *R*
shows *dist* (*P y t*) (*P z t*) ≤ *R*
by (*cases t* ≤ *tmin*; *cases t* ≥ *tmax*) (*auto simp*: *P-eqs y z not-le intro*!: *le*)

lemma *P-def'*:
assumes *t* ∈ *T*
assumes *fixed-point* $\in T \to_C X$
shows (*P fixed-point*) *t* = *x0* + *ivl-integral t0 t* (λ*x*. *f x* (*fixed-point x*))
by (*simp add*: *P-eq-P-inner assms P-inner-def*)

definition *iter-space* = *PiC T* ((λ-. *X*)(*t0*:={*x0*}))

lemma *iter-spaceI*:
assumes $g \in T \to_C X$ *g t0* = *x0*
shows *g* ∈ *iter-space*
using *assms*
by (*simp add*: *iter-space-def mem-PiC-iff Pi-iff*)

lemma *iter-spaceD*:
assumes *g* ∈ *iter-space*
shows $g \in T \to_C X$ *apply-bcontfun g t0* = *x0*
using *assms iv-defined*
by (*auto simp add*: *iter-space-def mem-PiC-iff split*: *if-splits*)

lemma *const-in-iter-space*: *const-bcontfun x0* ∈ *iter-space*
by (*auto simp*: *iter-space-def iv-defined mem-PiC-iff*)

lemma *closed-iter-space*: *closed iter-space*
by (*auto simp*: *iter-space-def intro*!: *closed-PiC closed*)

lemma *iter-space-notempty*: *iter-space* ≠ {}
  using *const-in-iter-space* **by** *blast*

lemma *clamp-in-eq*[*simp*]: **fixes** *a x b*::*real* **shows** $a \le x \Longrightarrow x \le b \Longrightarrow$ *clamp a b x* = *x*
  by (*auto simp*: *clamp-def*)

lemma *P-self-mapping*:
  assumes *in-space*: *g* ∈ *iter-space*
  shows *P g* ∈ *iter-space*
proof (*rule iter-spaceI*)
  show *x0*: *P g t0* = *x0*
    by (*auto simp*: *P-def′ iv-defined iter-spaceD*[*OF in-space*])
  from *iter-spaceD*(*1*)[*OF in-space*] **show** *P g* ∈ $T \to_C X$
    unfolding *mem-PiC-iff Pi-iff*
    apply (*auto simp*: *mem-PiC-iff Pi-iff P-def′*)
   apply (*auto simp*: *iter-spaceD*(*2*)[*OF in-space*, *symmetric*] *intro*!: *self-mapping*)
    using *closed-segment-subset-domainI iv-defined*(*1*) **by** *blast*
qed

lemma *continuous-on-T*: *continuous-on* {*tmin* .. *tmax*} *g* ⟹ *continuous-on T g*
  using *T-def* **by** *auto*

lemma *T-closed-segment-subsetI*[*intro*, *simp*]: *t* ∈ {*tmin*−−*tmax*} ⟹ *t* ∈ *T*
  and *T-subsetI*[*intro*, *simp*]: *tmin* ≤ *t* ⟹ *t* ≤ *tmax* ⟹ *t* ∈ *T*
  by (*subst T-def*, *simp add*: *closed-segment-eq-real-ivl*)+

lemma *t0-mem-closed-segment*[*intro*, *simp*]: *t0* ∈ {*tmin*−−*tmax*}
  using *T-def iv-defined*
  by (*simp add*: *closed-segment-eq-real-ivl*)

lemma *tmin-le-t0*[*intro*, *simp*]: *tmin* ≤ *t0*
  and *tmax-ge-t0*[*intro*, *simp*]: *tmax* ≥ *t0*
  using *t0-mem-closed-segment*
  unfolding *closed-segment-eq-real-ivl*
  by *simp-all*

lemma *apply-bcontfun-solution-fixed-point*:
  assumes *ode*: (*apply-bcontfun x solves-ode f*) *T X*
  assumes *iv*: *x t0* = *x0*
  assumes *t*: *t* ∈ *T*
  shows *P x t* = *x t*
proof −
  have *t* ∈ {*t0* −− *t*} **by** *simp*
  have *ode′*: (*apply-bcontfun x solves-ode f*) {*t0*−−*t*} *X t* ∈ {*t0* −− *t*}
    using *ode T-def closed-segment-eq-real-ivl t* **apply** *auto*
    using *closed-segment-iv-subset-domain solves-ode-on-subset* **apply** *fastforce*
    using *closed-segment-iv-subset-domain solves-ode-on-subset* **apply** *fastforce*

```
    done
  from solves-odeD[OF ode]
  have x: x ∈ T →C X by (auto simp: mem-PiC-iff)
  from solution-fixed-point[OF ode'] iv
  show ?thesis
    unfolding P-def'[OF t x]
    by simp
qed

lemma
  solution-in-iter-space:
  assumes ode: (apply-bcontfun z solves-ode f) T X
  assumes iv: z t0 = x0
  shows z ∈ iter-space (is ?z ∈ -)
proof −
  from T-def ode have ode: (z solves-ode f) {tmin −− tmax} X
    by (simp add: closed-segment-eq-real-ivl)
  have (?z solves-ode f) T X
     using is-solution-ext-cont[OF solves-ode-continuous-on[OF ode], of f X] ode
T-def
    by (auto simp: min-def max-def closed-segment-eq-real-ivl)
  then have z ∈ T →C X
    by (auto simp add: solves-ode-def mem-PiC-iff)
  thus ?z ∈ iter-space
    by (auto simp: iv intro!: iter-spaceI)
qed

end

locale unique-on-bounded-closed = unique-on-closed +
  assumes lipschitz-bound: ⋀s t. s ∈ T =⇒ t ∈ T =⇒ abs (s − t) ∗ L < 1
begin

lemma lipschitz-bound-maxmin: (tmax − tmin) ∗ L < 1
  using lipschitz-bound[of tmax tmin]
  by auto

lemma lipschitz-P:
  shows ((tmax − tmin) ∗ L)−lipschitz-on iter-space P
proof (rule lipschitz-onI)
  have t0 ∈ T by (simp add: iv-defined)
  then show 0 ≤ (tmax − tmin) ∗ L
    using T-def
    by (auto intro!: mult-nonneg-nonneg lipschitz lipschitz-on-nonneg[OF lipschitz]
      iv-defined)
  fix y z
  assume y ∈ iter-space and z ∈ iter-space
  hence y-defined: y ∈ (T →C X) and y t0 = x0
    and z-defined: z ∈ (T →C X) and y t0 = x0
```

by (*auto dest*: *iter-spaceD*)
have *defined*: $s \in T$ *y s* $\in X$ *z s* $\in X$ **if** $s \in$ *closed-segment tmin tmax* **for** *s*
using *y-defined z-defined that T-def*
by (*auto simp*: *mem-PiC-iff*)
{
note [*intro*, *simp*] = *integrable-continuous-closed-segment*
fix *t*
assume *t-bounds*: *tmin* $\leq$ *t t* $\leq$ *tmax*
then have *cs-subs*: *closed-segment t0 t* $\subseteq$ *closed-segment tmin tmax*
by (*auto simp*: *closed-segment-eq-real-ivl*)
then have *cs-subs-ext*: $\bigwedge$*ta. ta* $\in$ {*t0*−−*t*} $\Longrightarrow$ *ta* $\in$ {*tmin*−−*tmax*} **by** *auto*

have *norm* (*P-inner y t* − *P-inner z t*) =
norm (*ivl-integral t0 t* (λ*t. f t* (*y t*) − *f t* (*z t*)))
by (*subst ivl-integral-diff*)
(*auto intro*!: *integrable-continuous-closed-segment continuous-intros defined cs-subs-ext simp*: *P-inner-def*)
also have ... $\leq$ *abs* (*ivl-integral t0 t* (λ*t. norm* (*f t* (*y t*) − *f t* (*z t*))))
by (*rule ivl-integral-norm-bound-ivl-integral*)
(*auto intro*!: *ivl-integral-norm-bound-ivl-integral continuous-intros integrable-continuous-closed-segment*
simp: *defined cs-subs-ext*)
also have ... $\leq$ *abs* (*ivl-integral t0 t* (λ*t. L* ∗ *norm* (*y t* − *z t*)))
using *lipschitz t-bounds T-def y-defined z-defined cs-subs*
by (*intro norm-ivl-integral-le*) (*auto intro*!: *continuous-intros integrable-continuous-closed-segment*
simp add: *dist-norm lipschitz-on-def mem-PiC-iff Pi-iff*)
also have ... $\leq$ *abs* (*ivl-integral t0 t* (λ*t. L* ∗ *norm* (*y* − *z*)))
using *norm-bounded*[*of y* − *z*]
L-nonneg
by (*intro norm-ivl-integral-le*) (*auto intro*!: *continuous-intros mult-left-mono*)
also have ... = *L* ∗ *abs* (*t* − *t0*) ∗ *norm* (*y* − *z*)
using *t-bounds L-nonneg* **by** (*simp add*: *abs-mult*)
also have ... $\leq$ *L* ∗ (*tmax* − *tmin*) ∗ *norm* (*y* − *z*)
using *t-bounds zero-le-dist L-nonneg cs-subs tmin-le-t0 tmax-ge-t0*
by (*auto intro*!: *mult-right-mono mult-left-mono simp*: *closed-segment-eq-real-ivl abs-real-def*
simp del: *tmin-le-t0 tmax-ge-t0 split*: *if-split-asm*)
finally
have *dist* (*P-inner y t*) (*P-inner z t*) $\leq$ (*tmax* − *tmin*) ∗ *L* ∗ *dist y z*
by (*simp add*: *dist-norm ac-simps*)
} **note** ∗ = *this*
show *dist* (*P y*) (*P z*) $\leq$ (*tmax* − *tmin*) ∗ *L* ∗ *dist y z*
by (*auto intro*!: *dist-bound dist-P-le* ∗ *y-defined z-defined mult-nonneg-nonneg L-nonneg*)
qed

lemma *fixed-point-unique*: ∃!*x*∈*iter-space. P x* = *x*
using *lipschitz lipschitz-bound-maxmin lipschitz-P T-def*

complete-UNIV iv-defined
by (*intro banach-fix*)
(*auto*
intro: *P-self-mapping split-mult-pos-le*
intro!: *closed-iter-space iter-space-notempty mult-nonneg-nonneg*
simp: *lipschitz-on-def complete-eq-closed*)

definition *fixed-point* **where**
fixed-point = (*THE x. x* ∈ *iter-space* ∧ *P x* = *x*)

lemma *fixed-point′*:
fixed-point ∈ *iter-space* ∧ *P fixed-point* = *fixed-point*
unfolding *fixed-point-def* **using** *fixed-point-unique*
by (*rule theI′*)

lemma *fixed-point*:
fixed-point ∈ *iter-space P fixed-point* = *fixed-point*
using *fixed-point′* **by** *simp-all*

lemma *fixed-point-equality′*: *x* ∈ *iter-space* ∧ *P x* = *x* ⟹ *fixed-point* = *x*
unfolding *fixed-point-def* **using** *fixed-point-unique*
by (*rule the1-equality*)

lemma *fixed-point-equality*: *x* ∈ *iter-space* ⟹ *P x* = *x* ⟹ *fixed-point* = *x*
using *fixed-point-equality′*[*of x*] **by** *auto*

lemma *fixed-point-iv*: *fixed-point t0* = *x0*
and *fixed-point-domain*: *x* ∈ *T* ⟹ *fixed-point x* ∈ *X*
using *fixed-point*
by (*force dest*: *iter-spaceD simp*: *mem-PiC-iff*)+

lemma *fixed-point-has-vderiv-on*: (*fixed-point has-vderiv-on* (λ*t. f t* (*fixed-point t*))) *T*
proof −
have *continuous-on T* (λ*x. f x* (*fixed-point x*))
using *fixed-point-domain*
by (*auto intro*!: *continuous-intros*)
then have ((λ*u. x0* + *ivl-integral t0 u* (λ*x. f x* (*fixed-point x*))) *has-vderiv-on* (λ*t. f t* (*fixed-point t*))) *T*
by (*auto intro*!: *derivative-intros ivl-integral-has-vderiv-on-compact-interval interval compact-time*)
then show *?thesis*
proof (*rule has-vderiv-eq*)
fix *t*
assume *t*: *t* ∈ *T*
have *fixed-point t* = *P fixed-point t*
using *fixed-point* **by** *simp*
also have ... = *x0* + *ivl-integral t0 t* (λ*x. f x* (*fixed-point x*))
using *t fixed-point-domain*

```
      by (auto simp: P-def' mem-PiC-iff)
    finally show x0 + ivl-integral t0 t (λx. f x (fixed-point x)) = fixed-point t by
simp
  qed (insert T-def, auto simp: closed-segment-eq-real-ivl)
qed

lemma fixed-point-solution:
  shows (fixed-point solves-ode f) T X
  using fixed-point-has-vderiv-on fixed-point-domain
  by (rule solves-odeI)
```

2.4.1 Unique solution

```
lemma solves-ode-equals-fixed-point:
  assumes ode: (x solves-ode f) T X
  assumes iv: x t0 = x0
  assumes t: t ∈ T
  shows x t = fixed-point t
proof −
  from solves-ode-continuous-on[OF ode] T-def
  have continuous-on (cbox tmin tmax) x by simp
  from continuous-on-cbox-bcontfunE[OF this]
  obtain g where g:
    t ∈ {tmin .. tmax} ⟹ apply-bcontfun g t = x t
    apply-bcontfun g t = x (clamp tmin tmax t)
    for t
    by (metis interval-cbox)
  with ode T-def have ode-g: (g solves-ode f) T X
    by (metis (no-types, lifting) solves-ode-cong)
  have x t = g t
    using t T-def
    by (intro g[symmetric]) auto
  also
  have g t0 = x0 g ∈ T →C X
    using iv g solves-odeD(2)[OF ode-g]
    unfolding mem-PiC-iff atLeastAtMost-iff
    by blast+
  then have g ∈ iter-space
    by (intro iter-spaceI)
  then have g = fixed-point
    apply (rule fixed-point-equality[symmetric])
    apply (rule bcontfun-eqI)
    subgoal for t
      using apply-bcontfun-solution-fixed-point[OF ode-g ‹g t0 = x0›, of tmin]
        apply-bcontfun-solution-fixed-point[OF ode-g ‹g t0 = x0›, of tmax]
        apply-bcontfun-solution-fixed-point[OF ode-g ‹g t0 = x0›, of t]
      using T-def
      by (fastforce simp: P-eqs not-le ‹g ∈ T →C X› g)
    done
```

finally show *?thesis* **.**
qed

lemma *solves-ode-on-closed-segment-equals-fixed-point*:
assumes *ode*: (*x solves-ode f*) {*t0 −− t1′*} *X*
assumes *iv*: *x t0* = *x0*
assumes *subset*: {*t0*−−*t1′*} ⊆ *T*
assumes *t-mem*: *t* ∈ {*t0*−−*t1′*}
shows *x t* = *fixed-point t*
proof −
have *subsetI*: *t* ∈ {*t0*−−*t1′*} ⟹ *t* ∈ *T* **for** *t*
using *subset* **by** *auto*
interpret *s*: *unique-on-bounded-closed t0* {*t0*−−*t1′*} *x0 f X L*
apply − **apply** *unfold-locales*
subgoal by (*simp add*: *closed-segment-eq-real-ivl*)
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
subgoal using *iv-defined* **by** *simp*
subgoal by (*intro self-mapping subsetI*)
subgoal by (*rule continuous-on-subset*[*OF continuous*]) (*auto simp*: *subsetI*)
subgoal by (*rule lipschitz*) (*auto simp*: *subsetI*)
subgoal by (*auto intro*!: *subsetI lipschitz-bound*)
done
have *x t* = *s.fixed-point t*
by (*rule s.solves-ode-equals-fixed-point*; *fact*)
moreover
have *fixed-point t* = *s.fixed-point t*
by (*intro s.solves-ode-equals-fixed-point solves-ode-on-subset*[*OF fixed-point-solution*]
assms
fixed-point-iv order-refl subset t-mem)
ultimately show *?thesis* **by** *simp*
qed

lemma *unique-solution*:
assumes *ivp1*: (*x solves-ode f*) *T X x t0* = *x0*
assumes *ivp2*: (*y solves-ode f*) *T X y t0* = *x0*
assumes *t* ∈ *T*
shows *x t* = *y t*
using *solves-ode-equals-fixed-point*[*OF ivp1* ‹*t* ∈ *T*›]
solves-ode-equals-fixed-point[*OF ivp2* ‹*t* ∈ *T*›]
by *simp*

lemma *fixed-point-usolves-ode*: (*fixed-point usolves-ode f from t0*) *T X*
apply (*rule usolves-odeI*[*OF fixed-point-solution*])
subgoal by (*simp add*: *iv-defined*(*1*))
subgoal by (*rule interval*)
subgoal
using *fixed-point-iv solves-ode-on-closed-segment-equals-fixed-point*

```
    by auto
  done

end

lemma closed-segment-Un:
  fixes a b c::real
  assumes b ∈ closed-segment a c
  shows closed-segment a b ∪ closed-segment b c = closed-segment a c
  using assms
  by (auto simp: closed-segment-eq-real-ivl)

lemma closed-segment-closed-segment-subset:
  fixes s::real and i::nat
  assumes s ∈ closed-segment a b
  assumes a ∈ closed-segment c d b ∈ closed-segment c d
  shows s ∈ closed-segment c d
  using assms
  by (auto simp: closed-segment-eq-real-ivl split: if-split-asm)


context unique-on-closed begin

context— solution until t1
  fixes t1::real
  assumes mem-t1: t1 ∈ T
begin

lemma subdivide-count-ex: ∃ n. L * abs (t1 − t0) / (Suc n) < 1
  by auto (meson add-strict-increasing less-numeral-extra(1) real-arch-simple)

definition subdivide-count = (SOME n. L * abs (t1 − t0) / Suc n < 1)

lemma subdivide-count: L * abs (t1 − t0) / Suc subdivide-count < 1
  unfolding subdivide-count-def
  using subdivide-count-ex
  by (rule someI-ex)

lemma subdivide-lipschitz:
  assumes |s − t| ≤ abs (t1 − t0) / Suc subdivide-count
  shows |s − t| * L < 1
proof −
  from assms L-nonneg
  have |s − t| * L ≤ abs (t1 − t0) / Suc subdivide-count * L
    by (rule mult-right-mono)
  also have ... < 1
    using subdivide-count
    by (simp add: ac-simps)
  finally show ?thesis .
```

qed

lemma *subdivide-lipschitz-lemma*:
  assumes *st*: $s \in \{a -- b\}$ $t \in \{a -- b\}$
  assumes *abs* $(b - a) \leq$ *abs* $(t1 - t0)$ / *Suc subdivide-count*
  shows $|s - t| * L < 1$
  apply (*rule subdivide-lipschitz*)
  apply (*rule order-trans*[**where** *y*=*abs* $(b - a)$])
  using *assms*
  by (*auto simp*: *closed-segment-eq-real-ivl split*: *if-splits*)

definition *step* = $(t1 - t0)$ / *Suc subdivide-count*

lemma *last-step*: $t0 +$ *real* (*Suc subdivide-count*) $*$ *step* = $t1$
  by (*auto simp*: *step-def*)

lemma *step-in-segment*:
  assumes $0 \leq i$ $i \leq$ *real* (*Suc subdivide-count*)
  shows $t0 + i *$ *step* $\in$ *closed-segment t0 t1*
  unfolding *closed-segment-eq-real-ivl step-def*
proof (*clarsimp*, *safe*)
  assume $t0 \leq t1$
  then have $(t1 - t0) * i \leq (t1 - t0) * (1 +$ *subdivide-count*)
    using *assms*
    by (*auto intro*!: *mult-left-mono*)
  then show $t0 + i * (t1 - t0) / (1 +$ *real subdivide-count*) $\leq t1$
    by (*simp add*: *field-simps*)
next
  assume $\neg t0 \leq t1$
  then have $(1 +$ *subdivide-count*) $* (t0 - t1) \geq i * (t0 - t1)$
    using *assms*
    by (*auto intro*!: *mult-right-mono*)
  then show $t1 \leq t0 + i * (t1 - t0) / (1 +$ *real subdivide-count*)
    by (*simp add*: *field-simps*)
  show $i * (t1 - t0) / (1 +$ *real subdivide-count*) ≤ 0
    using ‹$\neg t0 \leq t1$›
    by (*auto simp*: *divide-simps mult-le-0-iff assms*)
qed (*auto intro*!: *divide-nonneg-nonneg mult-nonneg-nonneg assms*)

lemma *subset-T1*:
  fixes *s*::*real* **and** *i*::*nat*
  assumes $s \in$ *closed-segment t0* $(t0 + i *$ *step*)
  assumes $i \leq$ *Suc subdivide-count*
  shows $s \in \{t0 -- t1\}$
  using *closed-segment-closed-segment-subset assms of-nat-le-iff of-nat-0-le-iff step-in-segment*
  by *blast*

lemma *subset-T*: $\{t0 -- t1\} \subseteq T$ **and** *subset-TI*: $s \in \{t0 -- t1\} \Longrightarrow s \in T$
  using *closed-segment-iv-subset-domain mem-t1* **by** *blast*+

primrec *psolution*::*nat* ⇒ *real* ⇒ *'a* **where**
psolution 0 t = *x0*
| *psolution* (*Suc i*) *t* = *unique-on-bounded-closed.fixed-point*
(*t0* + *real i* ∗ *step*) {*t0* + *real i* ∗ *step* −− *t0* + *real* (*Suc i*) ∗ *step*}
(*psolution i* (*t0* + *real i* ∗ *step*)) *f X t*

definition *psolutions t* = *psolution* (*LEAST i*. *t* ∈ *closed-segment* (*t0* + *real* (*i* − *1*) ∗ *step*) (*t0* + *real i* ∗ *step*)) *t*

lemma *psolutions-usolves-until-step*:
assumes *i-le*: *i* ≤ *Suc subdivide-count*
shows (*psolutions usolves-ode f from t0*) (*closed-segment t0* (*t0* + *real i* ∗ *step*)) *X*
proof *cases*
assume *t0* = *t1*
then have *step* = *0*
unfolding *step-def* **by** *simp*
then show *?thesis* **by** (*simp add*: *psolutions-def iv-defined usolves-ode-singleton*)
next
assume *t0* ≠ *t1*
then have *step* ≠ *0*
by (*simp add*: *step-def*)
define *S* **where** *S* ≡ λ*i*. *closed-segment* (*t0* + *real* (*i* − *1*) ∗ *step*) (*t0* + *real i* ∗ *step*)
have *solution-eq*: *psolutions* ≡ λ*t*. *psolution* (*LEAST i*. *t* ∈ *S i*) *t*
by (*simp add*: *psolutions-def*[*abs-def*] *S-def*)
show *?thesis*
unfolding *solution-eq*
using *i-le*
proof (*induction i*)
case *0* **then show** *?case* **by** (*simp add*: *iv-defined usolves-ode-singleton S-def*)
next
case (*Suc i*)
let *?sol* = λ*t*. *psolution* (*LEAST i*. *t* ∈ *S i*) *t*
let *?pi* = *t0* + *real* (*i* − *Suc 0*) ∗ *step* **and** *?i* = *t0* + *real i* ∗ *step* **and** *?si* = *t0* + (*1* + *real i*) ∗ *step*
from *Suc* **have** *ui*: (*?sol usolves-ode f from t0*) (*closed-segment t0* (*t0* + *real i* ∗ *step*)) *X*
by *simp*

from *usolves-odeD*(*1*)[*OF Suc.IH*] *Suc*
have *IH-sol*: (*?sol solves-ode f*) (*closed-segment t0 ?i*) *X*
by *simp*

have *Least-eq-t0*[*simp*]: (*LEAST n*. *t0* ∈ *S n*) = *0*
by (*rule Least-equality*) (*auto simp add*: *S-def*)
have *Least-eq*[*simp*]: (*LEAST n*. *t0* + *real i* ∗ *step* ∈ *S n*) = *i* **for** *i*
apply (*rule Least-equality*)

subgoal by (*simp add*: *S-def*)
subgoal
using ‹*step* ≠ *0*›
by (*cases step* ≥ *0*)
(*auto simp add*: *S-def closed-segment-eq-real-ivl zero-le-mult-iff split*: *if-split-asm*)
done

have *y* = *t0* + *real i* ∗ *s*
if *t0* + (*1* + *real i*) ∗ *s* ≤ *t t* ≤ *y y* ≤ *t0* + *real i* ∗ *s t0* ≤ *y*
for *y i s t*
proof −
from *that* **have** (*1* + *real i*) ∗ *s* ≤ *real i* ∗ *s 0* ≤ *real i* ∗ *s*
by *arith+*
have *s* + (*t0* + *s* ∗ *real i*) ≤ *t* ⟹ *t* ≤ *y* ⟹ *y* ≤ *t0* + *s* ∗ *real i* ⟹ *t0* ≤ *y* ⟹ *y* = *t0* + *s* ∗ *real i*
by (*metis add-decreasing2 eq-iff le-add-same-cancel2 linear mult-le-0-iff of-nat-0-le-iff order.trans*)
then show *?thesis* **using** *that*
by (*simp add*: *algebra-simps*)
qed
then have *segment-inter*:
xa = *t0* + *real i* ∗ *step*
if
t ∈ {*t0* + *real* (*Suc i* − *1*) ∗ *step*−−*t0* + *real* (*Suc i*) ∗ *step*}
xa ∈ *closed-segment* (*t0* + *real i* ∗ *step*) *t xa* ∈ *closed-segment t0* (*t0* + *real i* ∗ *step*)
for *xa t*
apply (*cases step* > *0*; *cases step* = *0*)
using *that*
by (*auto simp*: *S-def closed-segment-eq-real-ivl split*: *if-split-asm*)

have *right-cond*: *t0* ≤ *t t* ≤ *t1* **if** *t0* + *real i* ∗ *step* ≤ *t t* ≤ *t0* + (*step* + *real i* ∗ *step*) **for** *t*
proof −
from *that* **have** *0* ≤ *step* **by** *simp*
with *last-step* **have** *t0* ≤ *t1*
by (*metis le-add-same-cancel1 of-nat-0-le-iff zero-le-mult-iff*)
from *that* **have** *t0* ≤ *t* − *real i* ∗ *step* **by** *simp*
also have . . . ≤ *t* **using** *that* **by** (*auto intro*!: *mult-nonneg-nonneg*)
finally show *t0* ≤ *t* **.**
have *t* ≤ *t0* + (*real* (*Suc i*) ∗ *step*) **using** *that* **by** (*simp add*: *algebra-simps*)
also have . . . ≤ *t1*
proof −
have *real* (*Suc i*) ∗ (*t1* − *t0*) ≤ *real* (*Suc subdivide-count*) ∗ (*t1* − *t0*)
using *Suc.prems* ‹*t0* ≤ *t1*›
by (*auto intro*!: *mult-mono*)
then show *?thesis* **by** (*simp add*: *divide-simps algebra-simps step-def*)
qed

finally show *t* ≤ *t1* **.**
qed
have *left-cond*: *t1* ≤ *t t* ≤ *t0* **if** *t0* + (*step* + *real i* ∗ *step*) ≤ *t t* ≤ *t0* + *real i* ∗ *step* **for** *t*
proof −
from *that* **have** *step* ≤ *0* **by** *simp*
with *last-step* **have** *t1* ≤ *t0*
by (*metis add-le-same-cancel1 mult-nonneg-nonpos of-nat-0-le-iff*)
from *that* **have** *t0* ≥ *t* − *real i* ∗ *step* **by** *simp*
also have *t* − *real i* ∗ *step* ≥ *t* **using** *that* **by** (*auto intro*!: *mult-nonneg-nonpos*)
finally (*xtrans*) **show** *t* ≤ *t0* **.**
have *t* ≥ *t0* + (*real* (*Suc i*) ∗ *step*) **using** *that* **by** (*simp add*: *algebra-simps*)
also have *t0* + (*real* (*Suc i*) ∗ *step*) ≥ *t1*
proof −
have *real* (*Suc i*) ∗ (*t0* − *t1*) ≤ *real* (*Suc subdivide-count*) ∗ (*t0* − *t1*)
using *Suc.prems* ‹*t0* ≥ *t1*›
by (*auto intro*!: *mult-mono*)
then show *?thesis* **by** (*simp add*: *divide-simps algebra-simps step-def*)
qed
finally (*xtrans*) **show** *t1* ≤ *t* **.**
qed

interpret *l*: *self-mapping S* (*Suc i*) *?i ?sol ?i f X*
proof *unfold-locales*
show *?sol ?i* ∈ *X*
using *solves-odeD*(*2*)[*OF usolves-odeD*(*1*)[*OF ui*], *of ?i*]
by (*simp add*: *S-def*)
fix *x t* **assume** *t*[*unfolded S-def*]: *t* ∈ *S* (*Suc i*)
and *x*: *x ?i* = *?sol ?i x* ∈ *closed-segment ?i t* → *X*
and *cont*: *continuous-on* (*closed-segment ?i t*) *x*

let *?if* = λ*t*. *if t* ∈ *closed-segment t0 ?i then ?sol t else x t*
let *?f* = λ*t*. *f t* (*?if t*)
have *sol-mem*: *?sol s* ∈ *X* **if** *s* ∈ *closed-segment t0 ?i* **for** *s*
by (*auto simp*: *subset-T1 intro*!: *solves-odeD*[*OF IH-sol*] *that*)

from *x*(*1*) **have** *x ?i* + *ivl-integral ?i t* (λ*t*. *f t* (*x t*)) = *?sol ?i* + *ivl-integral ?i t* (λ*t*. *f t* (*x t*))
by *simp*
also have *?sol ?i* = *?sol t0* + *ivl-integral t0 ?i* (λ*t*. *f t* (*?sol t*))
apply (*subst solution-fixed-point*)
apply (*rule usolves-odeD*[*OF ui*])
by *simp-all*
also have *ivl-integral t0 ?i* (λ*t*. *f t* (*?sol t*)) = *ivl-integral t0 ?i ?f*
by (*simp cong*: *ivl-integral-cong*)
also
have *psolution-eq*: *x* (*t0* + *real i* ∗ *step*) = *psolution i* (*t0* + *real i* ∗ *step*) ⟹
ta ∈ {*t0* + *real i* ∗ *step*−−*t*} ⟹

ta ∈ {t0−−t0 + real i ∗ step} ⟹ psolution (LEAST i. ta ∈ S i) ta = x ta
for ta
by (subst segment-inter[OF t], assumption, assumption)+ simp
have ivl-integral ?i t (λt. f t (x t)) = ivl-integral ?i t ?f
by (rule ivl-integral-cong) (simp-all add: x psolution-eq)
also
from t right-cond(1) **have** cs: closed-segment t0 t = closed-segment t0 ?i ∪
closed-segment ?i t
by (intro closed-segment-Un[symmetric])
(auto simp: closed-segment-eq-real-ivl algebra-simps mult-le-0-iff split:
if-split-asm
intro!: segment-inter segment-inter[symmetric])
have cont-if: continuous-on (closed-segment t0 t) ?if
unfolding cs
using x Suc.prems cont t psolution-eq
by (auto simp: subset-T1 T-def intro!: continuous-on-cases solves-ode-continuous-on[OF
IH-sol])
have t-mem: t ∈ closed-segment t0 t1
using x Suc.prems t
apply −
apply (rule closed-segment-closed-segment-subset, assumption)
apply (rule step-in-segment, force, force)
apply (rule step-in-segment, force, force)
done
have segment-subset: ta ∈ {t0 + real i ∗ step−−t} ⟹ ta ∈ {t0−−t1} **for**
ta
using x Suc.prems
apply −
apply (rule closed-segment-closed-segment-subset, assumption)
subgoal by (rule step-in-segment; force)
subgoal by (rule t-mem)
done
have cont-f: continuous-on (closed-segment t0 t) ?f
apply (rule continuous-intros)
apply (rule continuous-intros)
apply (rule cont-if)
unfolding cs
using x Suc.prems
apply (auto simp: subset-T1 segment-subset intro!: sol-mem subset-TI)
done
have ?sol t0 + ivl-integral t0 ?i ?f + ivl-integral ?i t ?f = ?if t0 + ivl-integral
t0 t ?f
by (auto simp: cs intro!: ivl-integral-combine integrable-continuous-closed-segment
continuous-on-subset[OF cont-f])
also have . . . ∈ X
apply (rule self-mapping)
apply (rule subset-TI)
apply (rule t-mem)
using x cont-if

by (*auto simp*: *subset-T1 Pi-iff cs intro*!: *sol-mem*)
finally
have *x ?i* + *ivl-integral ?i t* (λ*t*. *?f t*) ∈ *X* **.**
also have *ivl-integral ?i t* (λ*t*. *?f t*) = *ivl-integral ?i t* (λ*t*. *f t* (*x t*))
apply (*rule ivl-integral-cong*[*OF - refl refl*])
using *x*
by (*auto simp*: *segment-inter psolution-eq*)
finally
show *x ?i* + *ivl-integral ?i t* (λ*t*. *f t* (*x t*)) ∈ *X* **.**
qed (*auto simp add*: *S-def closed-segment-eq-real-ivl*)
have *S* (*Suc i*) ⊆ *T*
unfolding *S-def*
apply (*rule subsetI*)
apply (*rule subset-TI*)
proof (*cases step* = *0*)
case *False*
fix *x* **assume** *x*: *x* ∈ {*t0* + *real* (*Suc i* − *1*) ∗ *step*−−*t0* + *real* (*Suc i*) ∗ *step*}
from *x* **have** *nn*: ((*x* − *t0*) / *step*) ≥ *0*
using *False right-cond*(*1*)[*of x*] *left-cond*(*2*)[*of x*]
by (*auto simp*: *closed-segment-eq-real-ivl divide-simps algebra-simps split*: *if-splits*)
have *t1* < *t0* ⟹ *t1* ≤ *x t1* > *t0* ⟹ *x* ≤ *t1*
using *x False right-cond*(*1*,*2*)[*of x*] *left-cond*(*1*,*2*)[*of x*]
by (*auto simp*: *closed-segment-eq-real-ivl algebra-simps split*: *if-splits*)
then have *le*: (*x* − *t0*) / *step* ≤ *1* + *real subdivide-count*
unfolding *step-def*
by (*auto simp*: *divide-simps*)
have *x* = *t0* + ((*x* − *t0*) / *step*) ∗ *step*
using *False*
by *auto*
also have ... ∈ {*t0* −− *t1*}
by (*rule step-in-segment*) (*auto simp*: *nn le*)
finally show *x* ∈ {*t0* −− *t1*} **by** *simp*
qed *simp*
have *algebra*: (*1* + *real i*) ∗ (*t1* − *t0*) − *real i* ∗ (*t1* − *t0*) = *t1* − *t0*
by (*simp only*: *algebra-simps*)
interpret *l*: *unique-on-bounded-closed ?i S* (*Suc i*) *?sol ?i f X L*
apply *unfold-locales*
subgoal by (*auto simp*: *S-def*)
subgoal using ‹*S* (*Suc i*) ⊆ *T*› **by** (*auto intro*!: *continuous-intros simp*: *split-beta′*)
subgoal using ‹*S* (*Suc i*) ⊆ *T*› **by** (*auto intro*!: *lipschitz*)
subgoal by (*rule subdivide-lipschitz-lemma*) (*auto simp add*: *step-def divide-simps algebra S-def*)
done
note *ui*
moreover
have *mem-SI*: *t* ∈ *closed-segment ?i ?si* ⟹ *t* ∈ *S* (*if t* = *?i then i else Suc i*)

for *t*
by (*auto simp*: *S-def*)
have *min-S*: (*if t* = *t0* + *real i* ∗ *step then i else Suc i*) ≤ *y*
if *t* ∈ *closed-segment* (*t0* + *real i* ∗ *step*) (*t0* + (*1* + *real i*) ∗ *step*)
t ∈ *S y*
for *y t*
apply (*cases t* = *t0* + *real i* ∗ *step*)
subgoal using *that* ‹*step* ≠ *0*›
by (*auto simp add*: *S-def closed-segment-eq-real-ivl algebra-simps zero-le-mult-iff split*: *if-splits*)
subgoal premises *ne*
proof (*cases*)
assume *step* > *0*
with *that* **have** *t0* + *real i* ∗ *step* ≤ *t t* ≤ *t0* + (*1* + *real i*) ∗ *step*
t0 + *real* (*y* − *Suc 0*) ∗ *step* ≤ *t t* ≤ *t0* + *real y* ∗ *step*
by (*auto simp*: *closed-segment-eq-real-ivl S-def*)
then have *real i* ∗ *step* < *real y* ∗ *step* **using** ‹*step* > *0*› *ne*
by *arith*
then show *?thesis* **using** ‹*step* > *0*› *that* **by** (*auto simp add*: *closed-segment-eq-real-ivl S-def*)
next
assume ¬ *step* > *0* **with** ‹*step* ≠ *0*› **have** *step* < *0* **by** *simp*
with *that* **have** *t0* + (*1* + *real i*) ∗ *step* ≤ *t t* ≤ *t0* + *real i* ∗ *step*
t0 + *real y* ∗ *step* ≤ *t t* ≤ *t0* + *real* (*y* − *Suc 0*) ∗ *step* **using** *ne*
by (*auto simp*: *closed-segment-eq-real-ivl S-def diff-Suc zero-le-mult-iff split*: *if-splits nat.splits*)
then have *real y* ∗ *step* < *real i* ∗ *step*
using ‹*step* < *0*› *ne*
by *arith*
then show *?thesis* **using** ‹*step* < *0*› **by** (*auto simp add*: *closed-segment-eq-real-ivl S-def*)
qed
done
have (*?sol usolves-ode f from ?i*) (*closed-segment ?i ?si*) *X*
apply (*subst usolves-ode-cong*)
apply (*subst Least-equality*)
apply (*rule mem-SI*) **apply** *assumption*
apply (*rule min-S*) **apply** *assumption* **apply** *assumption*
apply (*rule refl*)
apply (*rule refl*)
apply (*rule refl*)
apply (*rule refl*)
apply (*rule refl*)
apply (*subst usolves-ode-cong*[**where** *y*=*psolution* (*Suc i*)])
using *l.fixed-point-iv*[*unfolded Least-eq*]
apply (*simp add*: *S-def*; *fail*)
apply (*rule refl*)
apply (*rule refl*)
apply (*rule refl*)

```
      apply (rule refl)
      using l.fixed-point-usolves-ode
      apply –
      apply (simp)
      apply (simp add: S-def)
      done
    moreover have t ∈ {t0 + real i ∗ step−−t0 + (step + real i ∗ step)} =⇒
        t ∈ {t0−−t0 + real i ∗ step} =⇒ t = t0 + real i ∗ step for t
      by (subst segment-inter[rotated], assumption, assumption) (auto simp: alge-
bra-simps)
    ultimately
    have ((λt. if t ∈ closed-segment t0 ?i then ?sol t else ?sol t)
      usolves-ode
      (λt. if t ∈ closed-segment t0 ?i then f t else f t) from t0)
      (closed-segment t0 ?i ∪ closed-segment ?i ?si) X
     by (intro connection-usolves-ode[where t=?i]) (auto simp: algebra-simps split:
if-split-asm)
    also have closed-segment t0 ?i ∪ closed-segment ?i ?si = closed-segment t0 ?si
      apply (rule closed-segment-Un)
      by (cases step < 0)
        (auto simp: closed-segment-eq-real-ivl zero-le-mult-iff mult-le-0-iff
          intro!: mult-right-mono
          split: if-split-asm)
    finally show ?case by simp
  qed
qed

lemma psolutions-usolves-ode: (psolutions usolves-ode f from t0) {t0 −− t1} X
proof –
  let ?T = closed-segment t0 (t0 + real (Suc subdivide-count) ∗ step)
  have (psolutions usolves-ode f from t0) ?T X
    by (rule psolutions-usolves-until-step) simp
  also have ?T = {t0 −− t1} unfolding last-step ..
  finally show ?thesis .
qed

end

definition solution t = (if t ≤ t0 then psolutions tmin t else psolutions tmax t)

lemma solution-eq-left: tmin ≤ t =⇒ t ≤ t0 =⇒ solution t = psolutions tmin t
  by (simp add: solution-def)

lemma solution-eq-right: t0 ≤ t =⇒ t ≤ tmax =⇒ solution t = psolutions tmax t
  by (simp add: solution-def psolutions-def)

lemma solution-usolves-ode: (solution usolves-ode f from t0) T X
proof –
  from psolutions-usolves-ode[OF tmin(2)] tmin-le-t0
```

have *u1*: (*psolutions tmin usolves-ode f from t0*) {*tmin* .. *t0*} *X*
by (*auto simp*: *closed-segment-eq-real-ivl split*: *if-splits*)
from *psolutions-usolves-ode*[*OF tmax*(*2*)] *tmin-le-t0*
have *u2*: (*psolutions tmax usolves-ode f from t0*) {*t0* .. *tmax*} *X*
by (*auto simp*: *closed-segment-eq-real-ivl split*: *if-splits*)
have (*solution usolves-ode f from t0*) ({*tmin* .. *t0*} ∪ {*t0* .. *tmax*}) (*X* ∪ *X*)
apply (*rule usolves-ode-union-closed*[**where** *t*=*t0*])
subgoal by (*subst usolves-ode-cong*[**where** *y*=*psolutions tmin*]) (*auto simp*: *solution-eq-left u1*)
subgoal
using *u2*
by (*rule usolves-ode-congI*) (*auto simp*: *solution-eq-right*)
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
done
also have {*tmin* .. *t0*} ∪ {*t0* .. *tmax*} = *T*
by (*simp add*: *T-split*[*symmetric*])
finally show *?thesis* **by** *simp*
qed

lemma *solution-solves-ode*: (*solution solves-ode f*) *T X*
by (*rule usolves-odeD*[*OF solution-usolves-ode*])

lemma *solution-iv*[*simp*]: *solution t0* = *x0*
by (*auto simp*: *solution-def psolutions-def*)

end

2.5 Picard-Lindeloef for *X* = *UNIV*

locale *unique-on-strip* =
compact-interval T +
continuous-rhs T UNIV f +
global-lipschitz T UNIV f L
for *t0* **and** *T* **and** *f*::*real* ⇒ *'a* ⇒ *'a*::*banach* **and** *L* +
assumes *iv-time*: *t0* ∈ *T*
begin

sublocale *unique-on-closed t0 T x0 f UNIV L* **for** *x0*
by (−, *unfold-locales*) (*auto simp*: *iv-time*)

end

2.6 Picard-Lindeloef on cylindric domain

locale *solution-in-cylinder* =
continuous-rhs T cball x0 b f +
compact-interval T

for *t0 T x0 b* **and** *f*::*real* ⇒ *'a* ⇒ *'a*::*banach* +
fixes *X B*
defines *X* ≡ *cball x0 b*
assumes *initial-time-in*: *t0* ∈ *T*
assumes *norm-f*: ⋀*x t*. *t* ∈ *T* ⟹ *x* ∈ *X* ⟹ *norm* (*f t x*) ≤ *B*
assumes *b-pos*: *b* ≥ *0*
assumes *e-bounded*: ⋀*t*. *t* ∈ *T* ⟹ *dist t t0* ≤ *b* / *B*
begin

lemmas *cylinder* = *X-def*

lemma *B-nonneg*: *B* ≥ *0*
proof −
have *0* ≤ *norm* (*f t0 x0*) **by** *simp*
also from *b-pos norm-f* **have** ... ≤ *B* **by** (*simp add*: *initial-time-in X-def*)
finally show *?thesis* **by** *simp*
qed

lemma *in-bounds-derivativeI*:
assumes *t* ∈ *T*
assumes *init*: *x t0* = *x0*
assumes *cont*: *continuous-on* (*closed-segment t0 t*) *x*
assumes *solves*: (*x has-vderiv-on* (λ*s*. *f s* (*y s*))) (*open-segment t0 t*)
assumes *y-bounded*: ⋀ξ. ξ ∈ *closed-segment t0 t* ⟹ *x* ξ ∈ *X* ⟹ *y* ξ ∈ *X*
shows *x t* ∈ *cball x0* (*B* ∗ *abs* (*t* − *t0*))
proof *cases*
assume *b* = *0* ∨ *B* = *0* **with** *assms e-bounded T-def* **have** *t* = *t0*
by *auto*
thus *?thesis* **using** *b-pos init* **by** *simp*
next
assume ¬(*b* = *0* ∨ *B* = *0*)
hence *b* > *0 B* > *0* **using** *B-nonneg b-pos* **by** *auto*
show *?thesis*
proof *cases*
assume *t0* ≠ *t*
then have *b-less*: *B* ∗ *abs* (*t* − *t0*) ≤ *b*
using *b-pos e-bounded* **using** ‹*b* > *0*› ‹*B* > *0*› ‹*t* ∈ *T*›
by (*auto simp*: *field-simps initial-time-in dist-real-def abs-real-def closed-segment-eq-real-ivl split*: *if-split-asm*)
define *b* **where** *b* ≡ *B* ∗ *abs* (*t* − *t0*)
have *b* > *0* **using** ‹*t0* ≠ *t*› **by** (*auto intro*!: *mult-pos-pos simp*: *algebra-simps b-def* ‹*B* > *0*›)
from *cont*
have *closed*: *closed* (*closed-segment t0 t* ∩ ((λ*s*. *norm* (*x s* − *x t0*)) −‘ {*b*..}))
by (*intro continuous-closed-preimage continuous-intros closed-segment*)
have *exceeding*: {*s* ∈ *closed-segment t0 t*. *norm* (*x s* − *x t0*) ∈ {*b*..}} ⊆ {*t*}
proof (*rule ccontr*)
assume ¬{*s* ∈ *closed-segment t0 t*. *norm* (*x s* − *x t0*) ∈ {*b*..}} ⊆ {*t*}
hence *notempty*: (*closed-segment t0 t* ∩ ((λ*s*. *norm* (*x s* − *x t0*)) −‘ {*b*..}))

≠ {}
    and *not-max*: {*s* ∈ *closed-segment t0 t*. *norm* (*x s* − *x t0*) ∈ {*b*..}} ≠ {*t*}
    by *auto*
  obtain *s* **where** *s-bound*: *s* ∈ *closed-segment t0 t*
    and *exceeds*: *norm* (*x s* − *x t0*) ∈ {*b*..}
    and *min*: ∀ *t2*∈*closed-segment t0 t*.
      norm (*x t2* − *x t0*) ∈ {*b*..} ⟶ *dist t0 s* ≤ *dist t0 t2*
    by (*rule distance-attains-inf*[*OF closed notempty*, *of t0*]) *blast*
  have *s* ≠ *t0* **using** *exceeds* ‹*b* > *0*› **by** *auto*
  have *st*: *closed-segment t0 t* ⊇ *open-segment t0 s* **using** *s-bound*
    by (*auto simp*: *closed-segment-eq-real-ivl open-segment-eq-real-ivl*)
  from *cont* **have** *cont*: *continuous-on* (*closed-segment t0 s*) *x*
    by (*rule continuous-on-subset*)
   (*insert b-pos closed-segment-subset-domain s-bound*, *auto simp*: *closed-segment-eq-real-ivl*)
  have *bnd-cont*: *continuous-on* (*closed-segment t0 s*) ((∗) *B*)
    and *bnd-deriv*: ((∗) *B has-vderiv-on* (λ-. *B*)) (*open-segment t0 s*)
    by (*auto intro*!: *continuous-intros derivative-eq-intros*
      simp: *has-vector-derivative-def has-vderiv-on-def*)
  {
    fix *ss* **assume** *ss*: *ss* ∈ *open-segment t0 s*
    with *st* **have** *ss* ∈ *closed-segment t0 t* **by** *auto*
    have *less-b*: *norm* (*x ss* − *x t0*) < *b*
    proof (*rule ccontr*)
      assume ¬ *norm* (*x ss* − *x t0*) < *b*
      hence *norm* (*x ss* − *x t0*) ∈ {*b*..} **by** *auto*
      from *min*[*rule-format*, *OF* ‹*ss* ∈ *closed-segment t0 t*› *this*]
      show *False* **using** *ss* ‹*s* ≠ *t0*›
        by (*auto simp*: *dist-real-def open-segment-eq-real-ivl split-ifs*)
    qed
    have *norm* (*f ss* (*y ss*)) ≤ *B*
      apply (*rule norm-f*)
      subgoal using *ss st closed-segment-subset-domain*[*OF initial-time-in* ‹*t* ∈
T›] **by** *auto*
      subgoal using *ss st b-less less-b*
        by (*intro y-bounded*)
         (*auto simp*: *X-def dist-norm b-def init norm-minus-commute mem-cball*)
      done
  } **note** *bnd* = *this*
  have *subs*: *open-segment t0 s* ⊆ *open-segment t0 t* **using** *s-bound* ‹*s* ≠ *t0*›
    by (*auto simp*: *closed-segment-eq-real-ivl open-segment-eq-real-ivl*)
  with *differentiable-bound-general-open-segment*[*OF cont bnd-cont has-vderiv-on-subset*[*OF*
solves subs]
    bnd-deriv bnd]
  have *norm* (*x s* − *x t0*) ≤ *B* ∗ |*s* − *t0*|
    by (*auto simp*: *algebra-simps*[*symmetric*] *abs-mult B-nonneg*)
  also
  have *s* ≠ *t*
    using *s-bound exceeds min not-max*
    by (*auto simp*: *dist-norm closed-segment-eq-real-ivl split-ifs*)

```
      hence B ∗ |s − t0| < |t − t0| ∗ B
        using s-bound ‹B > 0›
        by (intro le-neq-trans)
          (auto simp: algebra-simps closed-segment-eq-real-ivl split-ifs
            intro!: mult-left-mono)
      finally have norm (x s − x t0) < |t − t0| ∗ B .
      moreover
      {
        have b ≥ |t − t0| ∗ B by (simp add: b-def algebra-simps)
        also from exceeds have norm (x s − x t0) ≥ b by simp
        finally have |t − t0| ∗ B ≤ norm (x s − x t0) .
      }
      ultimately show False by simp
    qed note mvt-result = this
    from cont assms
    have cont-diff: continuous-on (closed-segment t0 t) (λxa. x xa − x t0)
      by (auto intro!: continuous-intros)
    have norm (x t − x t0) ≤ b
    proof (rule ccontr)
      assume H: ¬ norm (x t − x t0) ≤ b
      hence b ∈ closed-segment (norm (x t0 − x t0)) (norm (x t − x t0))
        using assms T-def ‹0 < b›
        by (auto simp: closed-segment-eq-real-ivl )
      from IVT'-closed-segment-real[OF this continuous-on-norm[OF cont-diff]]
      obtain s where s: s ∈ closed-segment t0 t norm (x s − x t0) = b
        using ‹b > 0› by auto
      have s ∈ {s ∈ closed-segment t0 t. norm (x s − x t0) ∈ {b..}}
        using s ‹t ∈ T› by (auto simp: initial-time-in)
      with mvt-result have s = t by blast
      hence s = t using s ‹t ∈ T› by (auto simp: initial-time-in)
      with s H show False by simp
    qed
    hence x t ∈ cball x0 b using init
      by (auto simp: dist-commute dist-norm[symmetric] mem-cball)
    thus x t ∈ cball x0 (B ∗ abs (t − t0)) unfolding cylinder b-def .
  qed (simp add: init[symmetric])
qed

lemma in-bounds-derivative-globalI:
  assumes t ∈ T
  assumes init: x t0 = x0
  assumes cont: continuous-on (closed-segment t0 t) x
  assumes solves: (x has-vderiv-on (λs. f s (y s))) (open-segment t0 t)
  assumes y-bounded: ⋀ξ. ξ ∈ closed-segment t0 t ⟹ x ξ ∈ X ⟹ y ξ ∈ X
  shows x t ∈ X
proof −
  from in-bounds-derivativeI[OF assms]
  have x t ∈ cball x0 (B ∗ abs (t − t0)) .
  moreover have B ∗ abs (t − t0) ≤ b using e-bounded b-pos B-nonneg ‹t ∈ T›
```

```
    by (cases B = 0)
    (auto simp: field-simps initial-time-in dist-real-def abs-real-def closed-segment-eq-real-ivl
split: if-splits)
  ultimately show ?thesis by (auto simp: cylinder mem-cball)
qed

lemma integral-in-bounds:
  assumes t ∈ T x t0 = x0 x ∈ {t0 -- t} → X
  assumes cont[continuous-intros]: continuous-on ({t0 -- t}) x
  shows x t0 + ivl-integral t0 t (λt. f t (x t)) ∈ X (is - + ?ix t ∈ X)
proof cases
  assume t = t0
  thus ?thesis by (auto simp: cylinder b-pos assms)
next
  assume t ≠ t0
  from closed-segment-subset-domain[OF initial-time-in]
  have cont-f:continuous-on {t0 -- t} (λt. f t (x t))
    using assms
    by (intro continuous-intros)
      (auto intro: cont continuous-on-subset[OF continuous] simp: cylinder split:
if-splits)
  from closed-segment-subset-domain[OF initial-time-in ‹t ∈ T›]
  have subsets: s ∈ {t0--t} ⟹ s ∈ T s ∈ open-segment t0 t ⟹ s ∈ {t0--t}
for s
    by (auto simp: closed-segment-eq-real-ivl open-segment-eq-real-ivl initial-time-in
split: if-split-asm)
  show ?thesis
    unfolding ‹x t0 = -›
    using assms ‹t ≠ t0›
    by (intro in-bounds-derivative-globalI[where y=x and x=λt. x0 + ?ix t])
      (auto simp: initial-time-in subsets cylinder has-vderiv-on-def
        split: if-split-asm
        intro!: cont-f has-vector-derivative-const integrable-continuous-closed-segment
          has-vector-derivative-within-subset[OF ivl-integral-has-vector-derivative]
          has-vector-derivative-add[THEN has-vector-derivative-eq-rhs]
          continuous-intros indefinite-ivl-integral-continuous)
qed

lemma solves-in-cone:
  assumes t ∈ T
  assumes init: x t0 = x0
  assumes cont: continuous-on (closed-segment t0 t) x
  assumes solves: (x has-vderiv-on (λs. f s (x s))) (open-segment t0 t)
  shows x t ∈ cball x0 (B * abs (t - t0))
  using assms
  by (rule in-bounds-derivativeI)

lemma is-solution-in-cone:
  assumes t ∈ T
```

assumes *sol*: (*x solves-ode f*) (*closed-segment t0 t*) *Y* **and** *iv*: *x t0* = *x0*
shows *x t* ∈ *cball x0* (*B* ∗ *abs* (*t* − *t0*))
using *solves-odeD*[*OF sol*] ‹*t* ∈ *T*›
by (*intro solves-in-cone*)
(*auto intro*!: *assms vderiv-on-continuous-on segment-open-subset-closed*
intro: *has-vderiv-on-subset simp*: *initial-time-in*)

lemma *cone-subset-domain*:
assumes *t* ∈ *T*
shows *cball x0* (*B* ∗ |*t* − *t0*|) ⊆ *X*
using *e-bounded*[*OF assms*] *B-nonneg b-pos*
unfolding *cylinder*
by (*intro subset-cball*) (*auto simp*: *dist-real-def divide-simps algebra-simps split*: *if-splits*)

lemma *is-solution-in-domain*:
assumes *t* ∈ *T*
assumes *sol*: (*x solves-ode f*) (*closed-segment t0 t*) *Y* **and** *iv*: *x t0* = *x0*
shows *x t* ∈ *X*
using *is-solution-in-cone*[*OF assms*] *cone-subset-domain*[*OF* ‹*t* ∈ *T*›]
by (*rule rev-subsetD*)

lemma *solves-ode-on-subset-domain*:
assumes *sol*: (*x solves-ode f*) *S Y* **and** *iv*: *x t0* = *x0*
and *ivl*: *t0* ∈ *S is-interval S S* ⊆ *T*
shows (*x solves-ode f*) *S X*
proof (*rule solves-odeI*)
show (*x has-vderiv-on* (λ*t*. *f t* (*x t*))) *S* **using** *solves-odeD*(*1*)[*OF sol*] **.**
show *x s* ∈ *X* **if** *s*: *s* ∈ *S* **for** *s*
proof −
from *s assms* **have** *s* ∈ *T*
by *auto*
moreover
have {*t0*−−*s*} ⊆ *S*
by (*rule closed-segment-subset*) (*auto simp*: *s assms is-interval-convex*)
with *sol* **have** (*x solves-ode f*) {*t0*−−*s*} *Y*
using *order-refl*
by (*rule solves-ode-on-subset*)
ultimately
show *?thesis* **using** *iv*
by (*rule is-solution-in-domain*)
qed
qed

lemma *usolves-ode-on-subset*:
assumes *x*: (*x usolves-ode f from t0*) *T X* **and** *iv*: *x t0* = *x0*
assumes *t0* ∈ *S is-interval S S* ⊆ *T X* ⊆ *Y*
shows (*x usolves-ode f from t0*) *S Y*
proof (*rule usolves-odeI*)

show (*x solves-ode f*) *S Y* **by** (*rule solves-ode-on-subset*[*OF usolves-odeD*(*1*)[*OF x*]]; *fact*)
show *t0* ∈ *S is-interval S* **by** *fact*+
fix *z t* **assume** {*t0* −− *t*} ⊆ *S* **and** *z*: (*z solves-ode f*) {*t0*−−*t*} *Y z t0* = *x t0*
then have *z t0* = *x0 t0* ∈ {*t0*−−*t*} *is-interval* {*t0*−−*t*} {*t0*−−*t*} ⊆ *T*
using *iv* ‹*S* ⊆ *T*› **by** (*auto simp*: *is-interval-convex-1*)
with *z*(*1*) **have** *zX*: (*z solves-ode f*) {*t0* −− *t*} *X*
by (*rule solves-ode-on-subset-domain*)
show *z t* = *x t*
apply (*rule usolves-odeD*(*4*)[*OF x* - - - *zX*])
using ‹{*t0* −− *t*} ⊆ *S*› ‹*S* ⊆ *T*›
by (*auto simp*: *is-interval-convex-1* ‹*z t0* = *x t0*›)
qed

lemma *usolves-ode-on-superset-domain*:
assumes (*x usolves-ode f from t0*) *T X* **and** *iv*: *x t0* = *x0*
assumes *X* ⊆ *Y*
shows (*x usolves-ode f from t0*) *T Y*
using *assms*(*1*,*2*) *usolves-odeD*(*2*,*3*)[*OF assms*(*1*)] *order-refl assms*(*3*)
by (*rule usolves-ode-on-subset*)

end

locale *unique-on-cylinder* =
solution-in-cylinder t0 T x0 b f X B +
global-lipschitz T X f L
for *t0 T x0 b X f B L*
begin

sublocale *unique-on-closed t0 T x0 f X L*
apply *unfold-locales*
subgoal by (*simp add*: *initial-time-in*)
subgoal by (*simp add*: *X-def b-pos*)
subgoal by (*auto intro*!: *integral-in-bounds simp*: *initial-time-in*)
subgoal by (*auto intro*!: *continuous-intros simp*: *split-beta′ X-def*)
subgoal by (*simp add*: *X-def*)
done

end

locale *derivative-on-prod* =
fixes *T X* **and** *f*::*real* ⇒ ′*a*::*banach* ⇒ ′*a* **and** *f* ′:: *real* × ′*a* ⇒ (*real* × ′*a*) ⇒ ′*a*
assumes *f* ′: ⋀*tx*. *tx* ∈ *T* × *X* ⟹ ((λ(*t*, *x*). *f t x*) *has-derivative* (*f* ′ *tx*)) (*at tx within* (*T* × *X*))
begin

lemma *f* ′*-comp*[*derivative-intros*]:
(*g has-derivative g*′) (*at s within S*) ⟹ (*h has-derivative h*′) (*at s within S*) ⟹
s ∈ *S* ⟹ (⋀*x*. *x* ∈ *S* ⟹ *g x* ∈ *T*) ⟹ (⋀*x*. *x* ∈ *S* ⟹ *h x* ∈ *X*) ⟹

$((\lambda x.\ f\ (g\ x)\ (h\ x))$ *has-derivative* $(\lambda y.\ f'\ (g\ s,\ h\ s)\ (g'\ y,\ h'\ y)))$ $(at\ s\ within\ S)$
apply (*rule has-derivative-in-compose2*[*OF f' - - has-derivative-Pair*, *unfolded split-beta' fst-conv snd-conv*, *of g h S s g' h'*])
apply *auto*
done

lemma *derivative-on-prod-subset*:
assumes $X' \subseteq X$
shows *derivative-on-prod* $T\ X'\ f\ f'$
using *assms*
by (*unfold-locales*) (*auto intro*!: *derivative-eq-intros*)

end

end
theory *Picard-Lindeloef-Qualitative*
imports *Initial-Value-Problem*
begin

2.7 Picard-Lindeloef On Open Domains

2.7.1 Local Solution with local Lipschitz

lemma *cball-eq-closed-segment-real*:
fixes *x e*::*real*
shows *cball x e* = (*if* $e \geq 0$ *then* $\{x - e -- x + e\}$ *else* {})
by (*auto simp*: *closed-segment-eq-real-ivl dist-real-def mem-cball*)

lemma *cube-in-cball*:
fixes *x y* :: *'a*::*euclidean-space*
assumes $r > 0$
assumes $\bigwedge i.\ i \in Basis \Longrightarrow dist\ (x \cdot i)\ (y \cdot i) \leq r\ /\ sqrt(DIM('a))$
shows $y \in cball\ x\ r$
unfolding *mem-cball euclidean-dist-l2*[*of x y*] *L2-set-def*
proof –
have $(\sum i \in Basis.\ (dist\ (x \cdot i)\ (y \cdot i))^2) \leq (\sum (i::'a) \in Basis.\ (r\ /\ sqrt(DIM('a)))^2)$
proof (*intro sum-mono*)
fix *i* :: *'a*
assume $i \in Basis$
thus $(dist\ (x \cdot i)\ (y \cdot i))^2 \leq (r\ /\ sqrt(DIM('a)))^2$
using *assms*
by (*auto intro*: *sqrt-le-D*)
qed
moreover
have ... $\leq r^2$
using *assms* **by** (*simp add*: *power-divide*)
ultimately
show $sqrt\ (\sum i \in Basis.\ (dist\ (x \cdot i)\ (y \cdot i))^2) \leq r$
using *assms* **by** (*auto intro*!: *real-le-lsqrt sum-nonneg*)

qed

lemma *cbox-in-cball'*:
  fixes *x*::*'a*::*euclidean-space*
  assumes $0 < r$
  shows $\exists b > 0.\ b \le r \wedge (\exists B.\ B = (\sum i{\in}Basis.\ b *_R i) \wedge (\forall y \in cbox\ (x - B)$
$(x + B).\ y \in cball\ x\ r))$
proof (*rule*, *safe*)
  have $r\ /\ sqrt\ (real\ DIM('a)) \le r\ /\ 1$
    using *assms* **by** (*auto simp*: *divide-simps real-of-nat-ge-one-iff*)
  thus $r\ /\ sqrt\ (real\ DIM('a)) \le r$ **by** *simp*
next
  let $?B = \sum i{\in}Basis.\ (r\ /\ sqrt\ (real\ DIM('a)))\ *_R\ i$
  show $\exists B.\ B = ?B \wedge (\forall y \in cbox\ (x - B)\ (x + B).\ y \in cball\ x\ r)$
  proof (*rule*, *safe*)
    fix *y*::*'a*
    assume $y \in cbox\ (x - ?B)\ (x + ?B)$
    hence *bounds*:
      $\bigwedge i.\ i \in Basis \Longrightarrow (x - ?B) \cdot i \le y \cdot i$
      $\bigwedge i.\ i \in Basis \Longrightarrow y \cdot i \le (x + ?B) \cdot i$
      by (*auto simp*: *mem-box*)
    show $y \in cball\ x\ r$
    proof (*intro cube-in-cball*)
      fix *i* :: *'a*
      assume $i{\in}\ Basis$
      with *bounds*
      have *bounds-comp*:
        $x \cdot i - r\ /\ sqrt\ (real\ DIM('a)) \le y \cdot i$
        $y \cdot i \le x \cdot i + r\ /\ sqrt\ (real\ DIM('a))$
        by (*auto simp*: *algebra-simps*)
      thus $dist\ (x \cdot i)\ (y \cdot i) \le r\ /\ sqrt\ (real\ DIM('a))$
        unfolding *dist-real-def* **by** *simp*
    qed (*auto simp add*: *assms*)
  qed (*rule*)
qed (*auto simp*: *assms*)

lemma *Pair1-in-Basis*: $i \in Basis \Longrightarrow (i,\ 0) \in Basis$
 and *Pair2-in-Basis*: $i \in Basis \Longrightarrow (0,\ i) \in Basis$
  by (*auto simp*: *Basis-prod-def*)

lemma *le-real-sqrt-sumsq'* [*simp*]: $y \le sqrt\ (x * x + y * y)$
  by (*simp add*: *power2-eq-square* [*symmetric*])

lemma *cball-Pair-split-subset*: $cball\ (a,\ b)\ c \subseteq cball\ a\ c \times cball\ b\ c$
  by (*auto simp*: *dist-prod-def mem-cball power2-eq-square*
      intro: *order-trans*[*OF le-real-sqrt-sumsq*] *order-trans*[*OF le-real-sqrt-sumsq'*])

lemma *cball-times-subset*: $cball\ a\ (c/2) \times cball\ b\ (c/2) \subseteq cball\ (a,\ b)\ c$
proof −

```
  {
    fix a' b'
    have sqrt ((dist a a')^2 + (dist b b')^2) ≤ dist a a' + dist b b'
      by (rule real-le-lsqrt) (auto simp: power2-eq-square algebra-simps)
    also assume a' ∈ cball a (c / 2)
    then have dist a a' ≤ c / 2 by (simp add: mem-cball)
    also assume b' ∈ cball b (c / 2)
    then have dist b b' ≤ c / 2 by (simp add: mem-cball)
    finally have sqrt ((dist a a')^2 + (dist b b')^2) ≤ c
      by simp
  } thus ?thesis by (auto simp: dist-prod-def mem-cball)
qed

lemma eventually-bound-pairE:
  assumes isCont f (t0, x0)
  obtains B where
    B ≥ 1
    eventually (λe. ∀ x ∈ cball t0 e × cball x0 e. norm (f x) ≤ B) (at-right 0)
proof −
  from assms[simplified isCont-def, THEN tendstoD, OF zero-less-one]
  obtain d::real where d: d > 0
    ⋀x. x ≠ (t0, x0) ⟹ dist x (t0, x0) < d ⟹ dist (f x) (f (t0, x0)) < 1
    by (auto simp: eventually-at)
  have bound: norm (f (t, x)) ≤ norm (f (t0, x0)) + 1
    if t ∈ cball t0 (d/3) x ∈ cball x0 (d/3) for t x
  proof −
    from that have norm (f (t, x) − f (t0, x0)) < 1
      using ‹0 < d›
      unfolding dist-norm[symmetric]
      apply (cases (t, x) = (t0, x0), force)
      by (rule d) (auto simp: dist-commute dist-prod-def mem-cball
        intro!: le-less-trans[OF sqrt-sum-squares-le-sum-abs])
    then show ?thesis
      by norm
  qed
  have norm (f (t0, x0)) + 1 ≥ 1
    eventually (λe. ∀ x ∈ cball t0 e × cball x0 e.
      norm (f x) ≤ norm (f (t0, x0)) + 1) (at-right 0)
    using d(1) bound
   by (auto simp: eventually-at dist-real-def mem-cball intro!: exI[where x=d/3])
  thus ?thesis ..
qed

lemma
  eventually-in-cballs:
  assumes d > 0 c > 0
  shows eventually (λe. cball t0 (c ∗ e) × (cball x0 e) ⊆ cball (t0, x0) d) (at-right
0)
  using assms
```

by (*auto simp*: *eventually-at dist-real-def field-simps dist-prod-def mem-cball*
intro!: *exI*[**where** *x*=*min d* (*d* / *c*) / *3*]
order-trans[*OF sqrt-sum-squares-le-sum-abs*])

lemma *cball-eq-sing'*:
fixes *x* :: *'a*::{*metric-space*,*perfect-space*}
shows *cball x e* = {*y*} ⟷ *e* = *0* ∧ *x* = *y*
using *cball-eq-sing*[*of x e*]
apply (*cases x* = *y*, *force*)
by (*metis cball-empty centre-in-cball insert-not-empty not-le singletonD*)

locale *ll-on-open* = *interval T* **for** *T* +
fixes *f*::*real* ⇒ *'a*::{*banach*, *heine-borel*} ⇒ *'a* **and** *X*
assumes *local-lipschitz*: *local-lipschitz T X f*
assumes *cont*: ⋀*x*. *x* ∈ *X* ⟹ *continuous-on T* (λ*t*. *f t x*)
assumes *open-domain*[*intro*!, *simp*]: *open T open X*
begin

all flows on closed segments

definition *csols* **where**
csols t0 x0 = {(*x*, *t1*). {*t0*−−*t1*} ⊆ *T* ∧ *x t0* = *x0* ∧ (*x solves-ode f*) {*t0*−−*t1*} *X*}

the maximal existence interval

definition *existence-ivl t0 x0* = (⋃(*x*, *t1*)∈*csols t0 x0* . {*t0*−−*t1*})

witness flow

definition *csol t0 x0* = (*SOME csol*. ∀ *t* ∈ *existence-ivl t0 x0*. (*csol t*, *t*) ∈ *csols t0 x0*)

unique flow

definition *flow* **where** *flow t0 x0* = (λ*t*. *if t* ∈ *existence-ivl t0 x0 then csol t0 x0 t t else 0*)

end

locale *ll-on-open-it* =
general?:— TODO: why is this qualification necessary? It seems only because of *ll-on-open-it T f X*
ll-on-open + **fixes** *t0*::*real*
— if possible, all development should be done with *t0* as explicit parameter for initial time: then it can be instantiated with *0* for autonomous ODEs

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

sublocale *ll-on-open-it* **where** *t0* = *t0* **for** *t0* **..**

sublocale *continuous-rhs T X f*

by *unfold-locales* (*rule continuous-on-TimesI*[*OF local-lipschitz cont*])

end

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

lemma *ll-on-open-rev*[*intro*, *simp*]: *ll-on-open* (*preflect t0* ‘ *T*) (λ*t*. − *f* (*preflect t0 t*)) *X*
  using *local-lipschitz interval*
  by *unfold-locales*
    (*auto intro*!: *continuous-intros cont intro*: *local-lipschitz-compose1*
    simp: *fun-Compl-def local-lipschitz-minus local-lipschitz-subset open-neg-translation*
      image-image preflect-def)

lemma *eventually-lipschitz*:
  assumes $t0 \in T$ $x0 \in X$ $c > 0$
  obtains *L* **where**
    eventually (λ*u*. ∀ *t′* ∈ *cball t0* (*c* ∗ *u*) ∩ *T*.
      L−*lipschitz-on* (*cball x0 u* ∩ *X*) (λ*y*. *f t′ y*)) (*at-right 0*)
proof −
  from *local-lipschitzE*[*OF local-lipschitz*, *OF* ‹*t0* ∈ *T*› ‹*x0* ∈ *X*›]
  obtain *u L* **where**
    u > *0*
    ⋀*t′*. *t′* ∈ *cball t0 u* ∩ *T* ⟹ *L*−*lipschitz-on* (*cball x0 u* ∩ *X*) (λ*y*. *f t′ y*)
    by *auto*
  hence *eventually* (λ*u*. ∀ *t′* ∈ *cball t0* (*c* ∗ *u*) ∩ *T*.
      L−*lipschitz-on* (*cball x0 u* ∩ *X*) (λ*y*. *f t′ y*)) (*at-right 0*)
    using ‹*u* > *0*› ‹*c* > *0*›
    by (*auto simp*: *dist-real-def eventually-at divide-simps algebra-simps*
      intro!: *exI*[**where** *x*=*min u* (*u* / *c*)]
      intro: *lipschitz-on-subset*[**where** *E*=*cball x0 u* ∩ *X*])
  thus *?thesis* **..**
qed

lemmas *continuous-on-Times-f* = *continuous*
lemmas *continuous-on-f* = *continuous-rhs-comp*

lemma
  lipschitz-on-compact:
  assumes *compact K K* ⊆ *T*
  assumes *compact Y Y* ⊆ *X*
  obtains *L* **where** ⋀*t*. *t* ∈ *K* ⟹ *L*−*lipschitz-on Y* (*f t*)
proof −
  have *cont*: ⋀*x*. *x* ∈ *Y* ⟹ *continuous-on K* (λ*t*. *f t x*)
    using ‹*Y* ⊆ *X*› ‹*K* ⊆ *T*›
    by (*auto intro*!: *continuous-on-f continuous-intros*)
  from *local-lipschitz*
  have *local-lipschitz K Y f*
    by (*rule local-lipschitz-subset*[*OF* - ‹*K* ⊆ *T*› ‹*Y* ⊆ *X*›])

from *local-lipschitz-compact-implies-lipschitz*[*OF this* ‹*compact Y*› ‹*compact K*› *cont*] *that*
show *?thesis* **by** *metis*
qed

lemma *csols-empty-iff*: *csols t0 x0* = {} ⟷ *t0* ∉ *T* ∨ *x0* ∉ *X*
proof *cases*
assume *iv-defined*: *t0* ∈ *T* ∧ *x0* ∈ *X*
then have (λ-. *x0*, *t0*) ∈ *csols t0 x0*
by (*auto simp*: *csols-def intro*!: *solves-ode-singleton*)
then show *?thesis* **using** ‹*t0* ∈ *T* ∧ *x0* ∈ *X*› **by** *auto*
qed (*auto simp*: *solves-ode-domainD csols-def*)

lemma *csols-notempty*: *t0* ∈ *T* ⟹ *x0* ∈ *X* ⟹ *csols t0 x0* ≠ {}
by (*simp add*: *csols-empty-iff*)

lemma *existence-ivl-empty-iff*[*simp*]: *existence-ivl t0 x0* = {} ⟷ *t0* ∉ *T* ∨ *x0* ∉ *X*
using *csols-empty-iff*
by (*auto simp*: *existence-ivl-def*)

lemma *existence-ivl-empty1*[*simp*]: *t0* ∉ *T* ⟹ *existence-ivl t0 x0* = {}
and *existence-ivl-empty2*[*simp*]: *x0* ∉ *X* ⟹ *existence-ivl t0 x0* = {}
using *csols-empty-iff*
by (*auto simp*: *existence-ivl-def*)

lemma *flow-undefined*:
shows *t0* ∉ *T* ⟹ *flow t0 x0* = (λ-. *0*)
x0 ∉ *X* ⟹ *flow t0 x0* = (λ-. *0*)
using *existence-ivl-empty-iff*
by (*auto simp*: *flow-def*)

lemma (**in** *ll-on-open*) *flow-eq-in-existence-ivlI*:
assumes ⋀*u*. *x0* ∈ *X* ⟹ *u* ∈ *existence-ivl t0 x0* ⟷ *g u* ∈ *existence-ivl s0 x0*
assumes ⋀*u*. *x0* ∈ *X* ⟹ *u* ∈ *existence-ivl t0 x0* ⟹ *flow t0 x0 u* = *flow s0 x0* (*g u*)
shows *flow t0 x0* = (λ*t*. *flow s0 x0* (*g t*))
apply (*cases x0* ∈ *X*)
subgoal using *assms* **by** (*auto intro*!: *ext simp*: *flow-def*)
subgoal by (*simp add*: *flow-undefined*)
done

2.7.2 Global maximal flow with local Lipschitz

lemma *local-unique-solution*:
assumes *iv-defined*: *t0* ∈ *T x0* ∈ *X*
obtains *et ex B L*
where *et* > *0 0* < *ex cball t0 et* ⊆ *T cball x0 ex* ⊆ *X*

unique-on-cylinder t0 (*cball t0 et*) *x0 ex f B L*
proof −
have ∀$_F$ *e*::*real in at-right 0*. *0* < *e*
by (*auto simp*: *eventually-at-filter*)
moreover

from *open-Times*[*OF open-domain*] **have** *open* (*T* × *X*) **.**
from *at-within-open*[*OF - this*] *iv-defined*
have *isCont* (λ(*t*, *x*). *f t x*) (*t0*, *x0*)
using *continuous* **by** (*auto simp*: *continuous-on-eq-continuous-within*)
from *eventually-bound-pairE*[*OF this*]
obtain *B* **where** *B*:
1 ≤ *B* ∀$_F$ *e in at-right 0*. ∀ *t*∈*cball t0 e*. ∀ *x*∈*cball x0 e*. *norm* (*f t x*) ≤ *B*
by *force*
note *B*(*2*)
moreover

define *t* **where** *t* ≡ *inverse B*
have *te*: ⋀*e*. *e* > *0* ⟹ *t* ∗ *e* > *0*
using ‹*1* ≤ *B*› **by** (*auto simp*: *t-def field-simps*)
have *t-pos*: *t* > *0*
using ‹*1* ≤ *B*› **by** (*auto simp*: *t-def*)

from *B*(*2*) **obtain** *dB* **where** *0* < *dB 0* < *dB* / *2*
and *dB*: ⋀*d t x*. *0* < *d* ⟹ *d* < *dB* ⟹ *t*∈*cball t0 d* ⟹ *x*∈*cball x0 d* ⟹
norm (*f t x*) ≤ *B*
by (*auto simp*: *eventually-at dist-real-def Ball-def*)

hence *dB*′: ⋀*t x*. (*t*, *x*) ∈ *cball* (*t0*, *x0*) (*dB* / *2*) ⟹ *norm* (*f t x*) ≤ *B*
using *cball-Pair-split-subset*[*of t0 x0 dB* / *2*]
by (*auto simp*: *eventually-at dist-real-def*
simp del: *mem-cball*
intro!: *dB*[**where** *d*=*dB*/*2*])
from *eventually-in-cballs*[*OF* ‹*0* < *dB*/*2*› *t-pos*, *of t0 x0*]
have ∀$_F$ *e in at-right 0*. ∀ *t*∈*cball t0* (*t* ∗ *e*). ∀ *x*∈*cball x0 e*. *norm* (*f t x*) ≤ *B*
unfolding *eventually-at-filter*
by *eventually-elim* (*auto intro*!: *dB*′)
moreover

from *eventually-lipschitz*[*OF iv-defined t-pos*] **obtain** *L* **where**
∀$_F$ *u in at-right 0*. ∀ *t*′∈*cball t0* (*t* ∗ *u*) ∩ *T*. *L*−*lipschitz-on* (*cball x0 u* ∩ *X*)
(*f t*′)
by *auto*
moreover
have ∀$_F$ *e in at-right 0*. *cball t0* (*t* ∗ *e*) ⊆ *T*
using *eventually-open-cball*[*OF open-domain*(*1*) *iv-defined*(*1*)]
by (*subst eventually-filtermap*[*symmetric*, **where** *f*=λ*x*. *t* ∗ *x*])
(*simp add*: *filtermap-times-pos-at-right t-pos*)
moreover

have *eventually* (λe. *cball x0 e* $\subseteq X$) (*at-right 0*)
using *open-domain*(*2*) *iv-defined*(*2*)
by (*rule eventually-open-cball*)
ultimately have $\forall_F$ *e in at-right 0*. $0 < e \wedge$ *cball t0* $(t * e) \subseteq T \wedge$ *cball x0 e* $\subseteq X \wedge$
unique-on-cylinder t0 (*cball t0* $(t * e)$) *x0 e f B L*
proof *eventually-elim*
case (*elim e*)
note ‹$0 < e$›
moreover
note T = ‹*cball t0* $(t * e) \subseteq T$›
moreover
note X = ‹*cball x0 e* $\subseteq X$›
moreover
from *elim Int-absorb2*[*OF* ‹*cball x0 e* $\subseteq X$›]
have *L*: $t' \in$ *cball t0* $(t * e) \cap T \Longrightarrow L$*−lipschitz-on* (*cball x0 e*) ($f\ t'$) **for** t'
by *auto*
from *elim* **have** *B*: $\bigwedge t'\ x.\ t' \in$ *cball t0* $(t * e) \Longrightarrow x \in$ *cball x0 e* $\Longrightarrow$ *norm* (f $t'\ x) \leq B$
by *auto*

have $t * e \leq e / B$
by (*auto simp*: *t-def cball-def dist-real-def inverse-eq-divide*)

have $\{t0 -- t0 + t * e\} \subseteq$ *cball t0* $(t * e)$
using ‹$t > 0$› ‹$e > 0$›
by (*auto simp*: *cball-eq-closed-segment-real closed-segment-eq-real-ivl*)
then have *unique-on-cylinder t0* (*cball t0* $(t * e)$) *x0 e f B L*
using *T X* ‹$t > 0$› ‹$e > 0$› ‹$t * e \leq e / B$›
by *unfold-locales*
(*auto intro*!: *continuous-rhs-comp continuous-on-fst continuous-on-snd B L continuous-on-id*
simp: *split-beta′ dist-commute mem-cball*)
ultimately show *?case* **by** *auto*
qed
from *eventually-happens*[*OF this*]
obtain *e* **where** $0 < e$ *cball t0* $(t * e) \subseteq T$ *cball x0 e* $\subseteq X$
unique-on-cylinder t0 (*cball t0* $(t * e)$) *x0 e f B L*
by (*metis trivial-limit-at-right-real*)
with *mult-pos-pos*[*OF* ‹$0 < t$› ‹$0 < e$›] **show** *?thesis* **..**
qed

lemma *mem-existence-ivl-iv-defined*:
assumes $t \in$ *existence-ivl t0 x0*
shows $t0 \in T$ $x0 \in X$
using *assms existence-ivl-empty-iff*
unfolding *atomize-conj*
by *blast*

lemma *csol-mem-csols*:
assumes $t \in$ *existence-ivl t0 x0*
shows (*csol t0 x0 t*, *t*) $\in$ *csols t0 x0*
proof –
have $\exists$ *csol*. $\forall$ *t* $\in$ *existence-ivl t0 x0*. (*csol t*, *t*) $\in$ *csols t0 x0*
proof (*safe intro*!: *bchoice*)
fix *t* **assume** *t* $\in$ *existence-ivl t0 x0*
then obtain *csol t1* **where** *csol*: (*csol t*, *t1*) $\in$ *csols t0 x0 t* $\in$ {*t0* −− *t1*}
by (*auto simp*: *existence-ivl-def*)
then have {*t0*−−*t*} $\subseteq$ {*t0* −− *t1*}
by (*auto simp*: *closed-segment-eq-real-ivl*)
then have (*csol t*, *t*) $\in$ *csols t0 x0* **using** *csol*
by (*auto simp*: *csols-def intro*: *solves-ode-on-subset*)
then show $\exists$ *y*. (*y*, *t*) $\in$ *csols t0 x0* **by** *force*
qed
then have $\forall$ *t* $\in$ *existence-ivl t0 x0*. (*csol t0 x0 t*, *t*) $\in$ *csols t0 x0*
unfolding *csol-def*
by (*rule someI-ex*)
with *assms* **show** *?thesis* **by** *auto*
qed

lemma *csol*:
assumes *t* $\in$ *existence-ivl t0 x0*
shows *t* $\in$ *T* {*t0*−−*t*} $\subseteq$ *T csol t0 x0 t t0* = *x0* (*csol t0 x0 t solves-ode f*) {*t0*−−*t*} *X*
using *csol-mem-csols*[*OF assms*]
by (*auto simp*: *csols-def*)

lemma *existence-ivl-initial-time-iff*[*simp*]: *t0* $\in$ *existence-ivl t0 x0* $\longleftrightarrow$ *t0* $\in$ *T* $\wedge$ *x0* $\in$ *X*
using *csols-empty-iff*
by (*auto simp*: *existence-ivl-def*)

lemma *existence-ivl-initial-time*: *t0* $\in$ *T* $\Longrightarrow$ *x0* $\in$ *X* $\Longrightarrow$ *t0* $\in$ *existence-ivl t0 x0*
by *simp*

lemmas *mem-existence-ivl-subset* = *csol*(*1*)

lemma *existence-ivl-subset*:
existence-ivl t0 x0 $\subseteq$ *T*
using *mem-existence-ivl-subset* **by** *blast*

lemma *is-interval-existence-ivl*[*intro*, *simp*]: *is-interval* (*existence-ivl t0 x0*)
unfolding *is-interval-connected-1*
by (*auto simp*: *existence-ivl-def intro*!: *connected-Union*)

lemma *connected-existence-ivl*[*intro*, *simp*]: *connected* (*existence-ivl t0 x0*)
using *is-interval-connected* **by** *blast*

lemma *in-existence-between-zeroI*:
  t ∈ *existence-ivl t0 x0* ⟹ *s* ∈ {*t0* −− *t*} ⟹ *s* ∈ *existence-ivl t0 x0*
  by (*meson existence-ivl-initial-time interval.closed-segment-subset-domainI inter-val.intro*
    is-interval-existence-ivl mem-existence-ivl-iv-defined(*1*) *mem-existence-ivl-iv-defined*(*2*))

lemma *segment-subset-existence-ivl*:
  assumes *s* ∈ *existence-ivl t0 x0 t* ∈ *existence-ivl t0 x0*
  shows {*s* −− *t*} ⊆ *existence-ivl t0 x0*
  using *assms is-interval-existence-ivl*
  unfolding *is-interval-convex-1*
  by (*rule closed-segment-subset*)

lemma *flow-initial-time-if*: *flow t0 x0 t0* = (*if t0* ∈ *T* ∧ *x0* ∈ *X then x0 else 0*)
  by (*simp add*: *flow-def csol*(*3*))

lemma *flow-initial-time*[*simp*]: *t0* ∈ *T* ⟹ *x0* ∈ *X* ⟹ *flow t0 x0 t0* = *x0*
  by (*auto simp*: *flow-initial-time-if*)

lemma *open-existence-ivl*[*intro*, *simp*]: *open* (*existence-ivl t0 x0*)
proof (*rule openI*)
  fix *t* **assume** *t*: *t* ∈ *existence-ivl t0 x0*
  note *csol* = *csol*[*OF this*]
  note *mem-existence-ivl-iv-defined*[*OF t*]

  have *flow t0 x0 t* ∈ *X* **using** ‹*t* ∈ *existence-ivl t0 x0*›
    using *csol*(*4*) *solves-ode-domainD*
    by (*force simp add*: *flow-def*)

  from *ll-on-open-it.local-unique-solution*[*OF ll-on-open-it-axioms* ‹*t* ∈ *T*› *this*]
  obtain *et ex B L* **where** *lsol*:
    0 < *et*
    0 < *ex*
    cball t et ⊆ *T*
    cball (*flow t0 x0 t*) *ex* ⊆ *X*
    unique-on-cylinder t (*cball t et*) (*flow t0 x0 t*) *ex f B L*
    by *metis*
  then interpret *unique-on-cylinder t cball t et flow t0 x0 t ex cball* (*flow t0 x0 t*) *ex f B L*
    by *auto*
  from *solution-usolves-ode* **have** *lsol-ode*: (*solution solves-ode f*) (*cball t et*) (*cball* (*flow t0 x0 t*) *ex*)
    by (*intro usolves-odeD*)
  show ∃ *e*>*0*. *ball t e* ⊆ *existence-ivl t0 x0*
  proof *cases*
    assume *t* = *t0*
    show *?thesis*
    proof (*safe intro*!: *exI*[**where** *x*=*et*] *mult-pos-pos* ‹*0* < *et*› ‹*0* < *ex*›)

fix *t'* **assume** *t'* ∈ *ball t et*
then have *subset*: {*t0*−−*t'*} ⊆ *ball t et*
by (*intro closed-segment-subset*) (*auto simp*: ‹*0* < *et*› ‹*0* < *ex*› ‹*t* = *t0*›)
also have ... ⊆ *cball t et* **by** *simp*
also note ‹*cball t* - ⊆ *T*›
finally have {*t0*−−*t'*} ⊆ *T* **by** *simp*
moreover have (*solution solves-ode f*) {*t0*−−*t'*} *X*
using *lsol-ode*
apply (*rule solves-ode-on-subset*)
using *subset lsol*
by (*auto simp*: *mem-ball mem-cball*)
ultimately have (*solution*, *t'*) ∈ *csols t0 x0*
unfolding *csols-def*
using *lsol* ‹*t'* ∈ *ball* - -› *lsol* ‹*t* = *t0*› *solution-iv* ‹*x0* ∈ *X*›
by (*auto simp*: *csols-def*)
then show *t'* ∈ *existence-ivl t0 x0*
unfolding *existence-ivl-def*
by *force*
qed
next
assume *t* ≠ *t0*
let *?m* = *min et* (*dist t0 t* / *2*)
show *?thesis*
proof (*safe intro*!: *exI*[**where** *x* = *?m*])
let *?t1'* = *if t0* ≤ *t then t* + *et else t* − *et*
have *lsol-ode*: (*solution solves-ode f*) {*t* −− *?t1'*} (*cball* (*flow t0 x0 t*) *ex*)
by (*rule solves-ode-on-subset*[*OF lsol-ode*])
(*insert* ‹*0* < *et*› ‹*0* < *ex*›, *auto simp*: *mem-cball closed-segment-eq-real-ivl dist-real-def*)
let *?if* = λ*ta*. *if ta* ∈ {*t0*−−*t*} *then csol t0 x0 t ta else solution ta*
let *?iff* = λ*ta*. *if ta* ∈ {*t0*−−*t*} *then f ta else f ta*
have (*?if solves-ode ?iff*) ({*t0*−−*t*} ∪ {*t* −− *?t1'*}) *X*
apply (*rule connection-solves-ode*[*OF csol*(*4*) *lsol-ode*, *unfolded Un-absorb2*[*OF* ‹- ⊆ *X*›]])
using *lsol solution-iv* ‹*t* ∈ *existence-ivl t0 x0*›
by (*auto intro*!: *simp*: *closed-segment-eq-real-ivl flow-def split*: *if-split-asm*)
also have *?iff* = *f* **by** *auto*
also have *Un-eq*: {*t0*−−*t*} ∪ {*t* −− *?t1'*} = {*t0* −− *?t1'*}
using ‹*0* < *et*› ‹*0* < *ex*›
by (*auto simp*: *closed-segment-eq-real-ivl*)
finally have *continuation*: (*?if solves-ode f*) {*t0*−−*?t1'*} *X* **.**
have *subset-T*: {*t0* −− *?t1'*} ⊆ *T*
unfolding *Un-eq*[*symmetric*]
apply (*intro Un-least*)
subgoal using *csol* **by** *force*
subgoal using - *lsol*(*3*)
apply (*rule order-trans*)
using ‹*0* < *et*› ‹*0* < *ex*›
by (*auto simp*: *closed-segment-eq-real-ivl subset-iff mem-cball dist-real-def*)

done
fix t' **assume** $t' \in$ *ball t ?m*
then have *scs*: $\{t0 \ -- \ t'\} \subseteq \{t0--?t1'\}$
using ‹*0 < et*› ‹*0 < ex*›
by (*auto simp*: *closed-segment-eq-real-ivl dist-real-def abs-real-def mem-ball split*: *if-split-asm*)
with *continuation* **have** (*?if solves-ode f*) $\{t0 \ -- \ t'\}$ *X*
by (*rule solves-ode-on-subset*) *simp*
then have (*?if*, t') $\in$ *csols t0 x0*
using *lsol* ‹$t' \in$ *ball - -*› *csol scs subset-T*
by (*auto simp*: *csols-def subset-iff*)
then show $t' \in$ *existence-ivl t0 x0*
unfolding *existence-ivl-def*
by *force*
qed (*insert* ‹$t \neq t0$› ‹*0 < et*› ‹*0 < ex*›, *simp*)
qed
qed

lemma *csols-unique*:
assumes (*x*, *t1*) $\in$ *csols t0 x0*
assumes (*y*, *t2*) $\in$ *csols t0 x0*
shows $\forall$ *t* $\in$ $\{t0 \ -- \ t1\} \cap \{t0 \ -- \ t2\}$. *x t* = *y t*
proof (*rule ccontr*)
let *?S* = $\{t0 \ -- \ t1\} \cap \{t0 \ -- \ t2\}$
let *?Z0* = (λ*t*. *x t* − *y t*) −‘ $\{0\} \cap$ *?S*
let *?Z* = *connected-component-set ?Z0 t0*
from *assms* **have** *t1*: *t1* $\in$ *existence-ivl t0 x0* **and** *t2*: *t2* $\in$ *existence-ivl t0 x0*
and *x*: (*x solves-ode f*) $\{t0 \ -- \ t1\}$ *X*
and *y*: (*y solves-ode f*) $\{t0 \ -- \ t2\}$ *X*
and *sub1*: $\{t0--t1\} \subseteq T$
and *sub2*: $\{t0--t2\} \subseteq T$
and *x0*: *x t0* = *x0*
and *y0*: *y t0* = *x0*
by (*auto simp*: *existence-ivl-def csols-def*)

assume ¬ ($\forall$ *t*$\in$*?S*. *x t* = *y t*)
hence $\exists$ *t*$\in$*?S*. *x t* $\neq$ *y t* **by** *simp*
then obtain *t-ne* **where** *t-ne*: *t-ne* $\in$ *?S x t-ne* $\neq$ *y t-ne* **..**
from *assms* **have** *x*: (*x solves-ode f*) $\{t0--t1\}$ *X*
and *y*:(*y solves-ode f*) $\{t0--t2\}$ *X*
by (*auto simp*: *csols-def*)
have *compact ?S*
by *auto*
have *closed ?Z*
by (*intro closed-connected-component closed-vimage-Int*)
(*auto intro*!: *continuous-intros continuous-on-subset*[*OF solves-ode-continuous-on*[*OF x*]]
continuous-on-subset[*OF solves-ode-continuous-on*[*OF y*]])
moreover

have *t0* ∈ *?Z* **using** *assms*
by (*auto simp*: *csols-def*)
then have *?Z* ≠ {}
by (*auto intro*!: *exI*[**where** *x*=*t0*])
ultimately
obtain *t-max* **where** *max*: *t-max* ∈ *?Z* *y* ∈ *?Z* ⟹ *dist t-ne t-max* ≤ *dist t-ne y* **for** *y*
by (*blast intro*: *distance-attains-inf*)
have *max-equal-flows*: *x t* = *y t* **if** *t* ∈ {*t0* −− *t-max*} **for** *t*
using *max*(*1*) *that*
by (*auto simp*: *connected-component-def vimage-def subset-iff closed-segment-eq-real-ivl split*: *if-split-asm*) (*metis connected-iff-interval*)+
then have *t-ne-outside*: *t-ne* ∉ {*t0* −− *t-max*} **using** *t-ne* **by** *auto*

have *x t-max* = *y t-max*
by (*rule max-equal-flows*) *simp*
have *t-max* ∈ *?S* *t-max* ∈ *T*
using *max sub1 sub2*
by (*auto simp*: *connected-component-def*)
with *solves-odeD*[*OF x*]
have *x t-max* ∈ *X*
by *auto*

from *ll-on-open-it.local-unique-solution*[*OF ll-on-open-it-axioms* ‹*t-max* ∈ *T*› ‹*x t-max* ∈ *X*›]
obtain *et ex B L*
where *0* < *et* *0* < *ex*
and *cball t-max et* ⊆ *T* *cball* (*x t-max*) *ex* ⊆ *X*
and *unique-on-cylinder t-max* (*cball t-max et*) (*x t-max*) *ex f B L*
by *metis*
then interpret *unique-on-cylinder t-max cball t-max et x t-max ex cball* (*x t-max*) *ex f B L*
by *auto*

from *usolves-ode-on-superset-domain*[*OF solution-usolves-ode solution-iv* ‹*cball* - - ⊆ *X*›]
have *solution-usolves-on-X*: (*solution usolves-ode f from t-max*) (*cball t-max et*) *X* **by** *simp*

have *ge-imps*: *t0* ≤ *t1* *t0* ≤ *t2* *t0* ≤ *t-max* *t-max* < *t-ne* **if** *t0* ≤ *t-ne*
using *that t-ne-outside* ‹*0* < *et*› ‹*0* < *ex*› *max*(*1*) ‹*t-max* ∈ *?S*› ‹*t-max* ∈ *T*› *t-ne x0 y0*
by (*auto simp*: *min-def dist-real-def max-def closed-segment-eq-real-ivl split*: *if-split-asm*)
have *le-imps*: *t0* ≥ *t1* *t0* ≥ *t2* *t0* ≥ *t-max* *t-max* > *t-ne* **if** *t0* ≥ *t-ne*
using *that t-ne-outside* ‹*0* < *et*› ‹*0* < *ex*› *max*(*1*) ‹*t-max* ∈ *?S*› ‹*t-max* ∈ *T*› *t-ne x0 y0*
by (*auto simp*: *min-def dist-real-def max-def closed-segment-eq-real-ivl split*: *if-split-asm*)

define *tt* **where** *tt* ≡ *if t0* ≤ *t-ne then min* (*t-max* + *et*) *t-ne else max* (*t-max* − *et*) *t-ne*
have *tt* ∈ *cball t-max et tt* ∈ {*t0* −− *t1*} *tt* ∈ {*t0* −− *t2*}
using *ge-imps le-imps* ‹*0* < *et*› *t-ne*(*1*)
by (*auto simp*: *mem-cball closed-segment-eq-real-ivl tt-def dist-real-def abs-real-def min-def max-def not-less*)

have *segment-unsplit*: {*t0* −− *t-max*} ∪ {*t-max* −− *tt*} = {*t0* −− *tt*}
using *ge-imps le-imps* ‹*0* < *et*›
by (*auto simp*: *tt-def closed-segment-eq-real-ivl min-def max-def split*: *if-split-asm*) *arith*

have *tt* ∈ {*t0* −− *t1*}
using *ge-imps le-imps* ‹*0* < *et*› *t-ne*(*1*)
by (*auto simp*: *tt-def closed-segment-eq-real-ivl min-def max-def split*: *if-split-asm*)

have *tt* ∈ *?Z*
proof (*safe intro*!: *connected-componentI*[**where** *T* = {*t0* −− *t-max*} ∪ {*t-max* −− *tt*}])
fix *s* **assume** *s*: *s* ∈ {*t-max* −− *tt*}
have {*t-max*−−*s*} ⊆ {*t-max* −− *tt*}
by (*rule closed-segment-subset*) (*auto simp*: *s*)
also have . . . ⊆ *cball t-max et*
using ‹*tt* ∈ *cball t-max et*› ‹*0* < *et*›
by (*intro closed-segment-subset*) *auto*
finally have *subset*: {*t-max*−−*s*} ⊆ *cball t-max et* **.**
from *s* **show** *s* ∈ {*t0*−−*t1*} *s* ∈ {*t0*−−*t2*}
using *ge-imps le-imps t-ne* ‹*0* < *et*›
by (*auto simp*: *tt-def min-def max-def closed-segment-eq-real-ivl split*: *if-split-asm*)
have *ivl*: *t-max* ∈ {*t-max* −− *s*} *is-interval* {*t-max*−−*s*}
using ‹*tt* ∈ *cball t-max et*› ‹*0* < *et*› *s*
by (*simp-all add*: *is-interval-convex-1*)
{
note *ivl subset*
moreover
have {*t-max*−−*s*} ⊆ {*t0*−−*t1*}
using ‹*s* ∈ {*t0* −− *t1*}› ‹*t-max* ∈ *?S*›
by (*simp add*: *closed-segment-subset*)
from *x this order-refl* **have** (*x solves-ode f*) {*t-max*−−*s*} *X*
by (*rule solves-ode-on-subset*)
moreover note *solution-iv*[*symmetric*]
ultimately
have *x s* = *solution s*
by (*rule usolves-odeD*(*4*)[*OF solution-usolves-on-X*]) *simp*
} moreover {
note *ivl subset*
moreover
have {*t-max*−−*s*} ⊆ {*t0*−−*t2*}

using ‹$s \in \{t0 \text{ -- } t2\}$› ‹$t\text{-}max \in ?S$›
by (*simp add*: *closed-segment-subset*)
from *y this order-refl* **have** (*y solves-ode f*) {*t-max*−−*s*} *X*
by (*rule solves-ode-on-subset*)
moreover from *solution-iv*[*symmetric*] **have** *y t-max* = *solution t-max*
by (*simp add*: ‹*x t-max* = *y t-max*›)
ultimately
have *y s* = *solution s*
by (*rule usolves-odeD*[*OF solution-usolves-on-X*]) *simp*
} **ultimately show** $s \in (\lambda t.\ x\ t - y\ t)$ −‘ {*0*} **by** *simp*
next
fix *s* **assume** *s*: $s \in \{t0 \text{ -- } t\text{-}max\}$
then show $s \in (\lambda t.\ x\ t - y\ t)$ −‘ {*0*}
by (*auto intro*!: *max-equal-flows*)
show $s \in \{t0\text{--}t1\}$ $s \in \{t0\text{--}t2\}$
by (*metis Int-iff* ‹$t\text{-}max \in ?S$› *closed-segment-closed-segment-subset ends-in-segment*(*1*) *s*)+
qed (*auto simp*: *segment-unsplit*)
then have *dist t-ne t-max* ≤ *dist t-ne tt*
by (*rule max*)
moreover have *dist t-ne t-max* > *dist t-ne tt*
using *le-imps ge-imps* ‹$0 < et$›
by (*auto simp*: *tt-def dist-real-def*)
ultimately show *False* **by** *simp*
qed

lemma *csol-unique*:
assumes *t1*: $t1 \in existence\text{-}ivl\ t0\ x0$
assumes *t2*: $t2 \in existence\text{-}ivl\ t0\ x0$
assumes *t*: $t \in \{t0 \text{ -- } t1\}$ $t \in \{t0 \text{ -- } t2\}$
shows *csol t0 x0 t1 t* = *csol t0 x0 t2 t*
using *csols-unique*[*OF csol-mem-csols*[*OF t1*] *csol-mem-csols*[*OF t2*]] *t*
by *simp*

lemma *flow-vderiv-on-left*:
(*flow t0 x0 has-vderiv-on* ($\lambda x.$ *f x* (*flow t0 x0 x*))) (*existence-ivl t0 x0* ∩ {..*t0*})
unfolding *has-vderiv-on-def*
proof *safe*
fix *t*
assume *t*: $t \in existence\text{-}ivl\ t0\ x0$ $t \le t0$
with *open-existence-ivl*
obtain *e* **where** $e > 0$ **and** *e*: $\bigwedge s.\ s \in cball\ t\ e \Longrightarrow s \in existence\text{-}ivl\ t0\ x0$
by (*force simp*: *open-contains-cball*)
have *csol-eq*: *csol t0 x0* $(t - e)$ *s* = *flow t0 x0 s* **if** $t - e \le s$ $s \le t0$ **for** *s*
unfolding *flow-def*
using *that* ‹$0 < e$› *t e*
by (*auto simp*: *cball-def dist-real-def abs-real-def closed-segment-eq-real-ivl subset-iff*
intro!: *csol-unique in-existence-between-zeroI*[*of* $t - e$ *x0 s*]

split: *if-split-asm*)
from *e*[*of t* − *e*] ‹*0* < *e*› **have** *t* − *e* ∈ *existence-ivl t0 x0* **by** (*auto simp*: *mem-cball*)

let *?l* = *existence-ivl t0 x0* ∩ {..*t0*}
let *?s* = {*t0* −− *t* − *e*}

from *csol*(*4*)[*OF e*[*of t* − *e*]] ‹*0* < *e*›
have *1*: (*csol t0 x0* (*t* − *e*) *solves-ode f*) *?s X*
by (*auto simp*: *mem-cball*)
have *t* ∈ {*t0* −− *t* − *e*} **using** *t* ‹*0* < *e*› **by** (*auto simp*: *closed-segment-eq-real-ivl*)
from *solves-odeD*(*1*)[*OF 1*, *unfolded has-vderiv-on-def*, *rule-format*, *OF this*]
have (*csol t0 x0* (*t* − *e*) *has-vector-derivative f t* (*csol t0 x0* (*t* − *e*) *t*)) (*at t within ?s*) **.**
also have *at t within ?s* = (*at t within ?l*)
using *t* ‹*0* < *e*›
by (*intro at-within-nhd*[**where** *S*={*t* − *e* <..< *t0* + *1*}])
(*auto simp*: *closed-segment-eq-real-ivl intro*!: *in-existence-between-zeroI*[*OF* ‹*t* − *e* ∈ *existence-ivl t0 x0*›])
finally
have (*csol t0 x0* (*t* − *e*) *has-vector-derivative f t* (*csol t0 x0* (*t* − *e*) *t*)) (*at t within existence-ivl t0 x0* ∩ {..*t0*}) **.**
also have *csol t0 x0* (*t* − *e*) *t* = *flow t0 x0 t*
using ‹*0* < *e*› ‹*t* ≤ *t0*› **by** (*auto intro*!: *csol-eq*)
finally
show (*flow t0 x0 has-vector-derivative f t* (*flow t0 x0 t*)) (*at t within existence-ivl t0 x0* ∩ {..*t0*})
apply (*rule has-vector-derivative-transform-within*[**where** *d*=*e*])
using *t* ‹*0* < *e*›
by (*auto intro*!: *csol-eq simp*: *dist-real-def*)
qed

lemma *flow-vderiv-on-right*:
(*flow t0 x0 has-vderiv-on* (λ*x*. *f x* (*flow t0 x0 x*))) (*existence-ivl t0 x0* ∩ {*t0*..})
unfolding *has-vderiv-on-def*
proof *safe*
fix *t*
assume *t*: *t* ∈ *existence-ivl t0 x0 t0* ≤ *t*
with *open-existence-ivl*
obtain *e* **where** *e* > *0* **and** *e*: ⋀*s*. *s* ∈ *cball t e* ⟹ *s* ∈ *existence-ivl t0 x0*
by (*force simp*: *open-contains-cball*)
have *csol-eq*: *csol t0 x0* (*t* + *e*) *s* = *flow t0 x0 s* **if** *s* ≤ *t* + *e t0* ≤ *s* **for** *s*
unfolding *flow-def*
using *e that* ‹*0* < *e*›
by (*auto simp*: *cball-def dist-real-def abs-real-def closed-segment-eq-real-ivl subset-iff*
intro!: *csol-unique in-existence-between-zeroI*[*of t* + *e x0 s*]
split: *if-split-asm*)
from *e*[*of t* + *e*] ‹*0* < *e*› **have** *t* + *e* ∈ *existence-ivl t0 x0* **by** (*auto simp*:

mem-cball dist-real-def)

let *?l* = *existence-ivl t0 x0* ∩ {*t0*..}
let *?s* = {*t0* −− *t* + *e*}

from *csol*(*4*)[*OF e*[*of t* + *e*]] ‹*0* < *e*›
have *1*: (*csol t0 x0* (*t* + *e*) *solves-ode f*) *?s X*
by (*auto simp*: *dist-real-def mem-cball*)
have *t* ∈ {*t0* −− *t* + *e*} **using** *t* ‹*0* < *e*› **by** (*auto simp*: *closed-segment-eq-real-ivl*)
from *solves-odeD*(*1*)[*OF 1*, *unfolded has-vderiv-on-def*, *rule-format*, *OF this*]
have (*csol t0 x0* (*t* + *e*) *has-vector-derivative f t* (*csol t0 x0* (*t* + *e*) *t*)) (*at t within ?s*) **.**
also have *at t within ?s* = (*at t within ?l*)
using *t* ‹*0* < *e*›
by (*intro at-within-nhd*[**where** *S*={*t0* − *1* <..< *t* + *e*}])
(*auto simp*: *closed-segment-eq-real-ivl intro*!: *in-existence-between-zeroI*[*OF* ‹*t* + *e* ∈ *existence-ivl t0 x0*›])
finally
have (*csol t0 x0* (*t* + *e*) *has-vector-derivative f t* (*csol t0 x0* (*t* + *e*) *t*)) (*at t within ?l*) **.**
also have *csol t0 x0* (*t* + *e*) *t* = *flow t0 x0 t*
using ‹*0* < *e*› ‹*t0* ≤ *t*› **by** (*auto intro*!: *csol-eq*)
finally
show (*flow t0 x0 has-vector-derivative f t* (*flow t0 x0 t*)) (*at t within ?l*)
apply (*rule has-vector-derivative-transform-within*[**where** *d*=*e*])
using *t* ‹*0* < *e*›
by (*auto intro*!: *csol-eq simp*: *dist-real-def*)
qed

lemma *flow-usolves-ode*:
assumes *iv-defined*: *t0* ∈ *T x0* ∈ *X*
shows (*flow t0 x0 usolves-ode f from t0*) (*existence-ivl t0 x0*) *X*
proof (*rule usolves-odeI*)
let *?l* = *existence-ivl t0 x0* ∩ {..*t0*} **and** *?r* = *existence-ivl t0 x0* ∩ {*t0*..}
let *?split* = *?l* ∪ *?r*
have *insert-idem*: *insert t0 ?l* = *?l insert t0 ?r* = *?r* **using** *iv-defined*
by *auto*
from *existence-ivl-initial-time* **have** *cl-inter*: *closure ?l* ∩ *closure ?r* = {*t0*}
proof *safe*
from *iv-defined* **have** *t0* ∈ *?l* **by** *simp* **also note** *closure-subset* **finally show** *t0* ∈ *closure ?l* **.**
from *iv-defined* **have** *t0* ∈ *?r* **by** *simp* **also note** *closure-subset* **finally show** *t0* ∈ *closure ?r* **.**
fix *x*
assume *xl*: *x* ∈ *closure ?l*
assume *x* ∈ *closure ?r*
also have *closure ?r* ⊆ *closure* {*t0*..}
by (*rule closure-mono*) *simp*
finally have *t0* ≤ *x* **by** *simp*

moreover
{
note *xl*
also have *cl*: *closure ?l* ⊆ *closure* {..*t0*}
by (*rule closure-mono*) *simp*
finally have $x \leq t0$ **by** *simp*
} **ultimately show** $x = t0$ **by** *simp*
qed
have (*flow t0 x0 has-vderiv-on* (λ*t*. *f t* (*flow t0 x0 t*))) *?split*
by (*rule has-vderiv-on-union*)
(*auto simp*: *cl-inter insert-idem flow-vderiv-on-right flow-vderiv-on-left*)
also have *?split* = *existence-ivl t0 x0*
by *auto*
finally have (*flow t0 x0 has-vderiv-on* (λ*t*. *f t* (*flow t0 x0 t*))) (*existence-ivl t0 x0*) **.**
moreover
have *flow t0 x0 t* ∈ *X* **if** *t* ∈ *existence-ivl t0 x0* **for** *t*
using *solves-odeD*(*2*)[*OF csol*(*4*)[*OF that*]] *that*
by (*simp add*: *flow-def*)
ultimately show (*flow t0 x0 solves-ode f*) (*existence-ivl t0 x0*) *X*
by (*rule solves-odeI*)
show *t0* ∈ *existence-ivl t0 x0* **using** *iv-defined* **by** *simp*
show *is-interval* (*existence-ivl t0 x0*) **by** (*simp add*: *is-interval-existence-ivl*)
fix *z t*
assume *z*: {*t0* −− *t*} ⊆ *existence-ivl t0 x0* (*z solves-ode f*) {*t0* −− *t*} *X z t0* = *flow t0 x0 t0*
then have *t* ∈ *existence-ivl t0 x0* **by** *auto*
moreover
from *csol*[*OF this*] *z* **have** (*z*, *t*) ∈ *csols t0 x0* **by** (*auto simp*: *csols-def*)
moreover have (*csol t0 x0 t*, *t*) ∈ *csols t0 x0*
by (*rule csol-mem-csols*) *fact*
ultimately
show *z t* = *flow t0 x0 t*
unfolding *flow-def*
by (*auto intro*: *csols-unique*[*rule-format*])
qed

lemma *flow-solves-ode*: *t0* ∈ *T* ⟹ *x0* ∈ *X* ⟹ (*flow t0 x0 solves-ode f*) (*existence-ivl t0 x0*) *X*
by (*rule usolves-odeD*[*OF flow-usolves-ode*])

lemma *equals-flowI*:
assumes *t0* ∈ *T'*
is-interval T'
T' ⊆ *existence-ivl t0 x0*
(*z solves-ode f*) *T' X*
z t0 = *flow t0 x0 t0 t* ∈ *T'*
shows *z t* = *flow t0 x0 t*
proof −

from *assms* **have** *iv-defined*: *t0* ∈ *T x0* ∈ *X*
  unfolding *atomize-conj*
  using *assms existence-ivl-subset mem-existence-ivl-iv-defined*
  by *blast*
show *?thesis*
  using *assms*
  by (*rule usolves-odeD*[*OF flow-usolves-ode*[*OF iv-defined*]])
qed

lemma *existence-ivl-maximal-segment*:
assumes (*x solves-ode f*) {*t0* −− *t*} *X x t0* = *x0*
assumes {*t0* −− *t*} ⊆ *T*
shows *t* ∈ *existence-ivl t0 x0*
using *assms*
by (*auto simp*: *existence-ivl-def csols-def*)

lemma *existence-ivl-maximal-interval*:
assumes (*x solves-ode f*) *S X x t0* = *x0*
assumes *t0* ∈ *S is-interval S S* ⊆ *T*
shows *S* ⊆ *existence-ivl t0 x0*
proof
fix *t* **assume** *t* ∈ *S*
with *assms* **have** *subset1*: {*t0*−−*t*} ⊆ *S*
  by (*intro closed-segment-subset*) (*auto simp*: *is-interval-convex-1*)
with ‹*S* ⊆ *T*› **have** *subset2*: {*t0* −− *t*} ⊆ *T* **by** *auto*
have (*x solves-ode f*) {*t0* −− *t*} *X*
  using *assms*(*1*) *subset1 order-refl*
  by (*rule solves-ode-on-subset*)
from *this* ‹*x t0* = *x0*› *subset2* **show** *t* ∈ *existence-ivl t0 x0*
  by (*rule existence-ivl-maximal-segment*)
qed

lemma *maximal-existence-flow*:
assumes *sol*: (*x solves-ode f*) *K X* **and** *iv*: *x t0* = *x0*
assumes *is-interval K*
assumes *t0* ∈ *K*
assumes *K* ⊆ *T*
shows *K* ⊆ *existence-ivl t0 x0* ⋀*t*. *t* ∈ *K* ⟹ *flow t0 x0 t* = *x t*
proof −
from *assms* **have** *iv-defined*: *t0* ∈ *T x0* ∈ *X*
  unfolding *atomize-conj*
  using *solves-ode-domainD* **by** *blast*
show *exivl*: *K* ⊆ *existence-ivl t0 x0*
  by (*rule existence-ivl-maximal-interval*; *rule assms*)
show *flow t0 x0 t* = *x t* **if** *t* ∈ *K* **for** *t*
  apply (*rule sym*)
  apply (*rule equals-flowI*[*OF* ‹*t0* ∈ *K*› ‹*is-interval K*› *exivl sol* - *that*])
  by (*simp add*: *iv iv-defined*)
qed

lemma *maximal-existence-flowI*:
  assumes (*x has-vderiv-on* (λ*t*. *f t* (*x t*))) *K*
  assumes ⋀*t*. *t* ∈ *K* ⟹ *x t* ∈ *X*
  assumes *x t0* = *x0*
  assumes *K*: *is-interval K t0* ∈ *K K* ⊆ *T*
  shows *K* ⊆ *existence-ivl t0 x0* ⋀*t*. *t* ∈ *K* ⟹ *flow t0 x0 t* = *x t*
proof −
  from *assms*(*1*,*2*) **have** *sol*: (*x solves-ode f*) *K X* **by** (*rule solves-odeI*)
  from *maximal-existence-flow*[*OF sol assms*(*3*) *K*]
  show *K* ⊆ *existence-ivl t0 x0* ⋀*t*. *t* ∈ *K* ⟹ *flow t0 x0 t* = *x t*
    by *auto*
qed

lemma *flow-in-domain*: *t* ∈ *existence-ivl t0 x0* ⟹ *flow t0 x0 t* ∈ *X*
  using *flow-solves-ode solves-ode-domainD local.mem-existence-ivl-iv-defined*
  by *blast*

lemma (**in** *ll-on-open*)
  assumes *t* ∈ *existence-ivl s x*
  assumes *x* ∈ *X*
  assumes *auto*: ⋀*s t x*. *x* ∈ *X* ⟹ *f s x* = *f t x*
  assumes *T* = *UNIV*
  shows *mem-existence-ivl-shift-autonomous1*: *t* − *s* ∈ *existence-ivl 0 x*
    and *flow-shift-autonomous1*: *flow s x t* = *flow 0 x* (*t* − *s*)
proof −
  have *na*: *s* ∈ *T x* ∈ *X* **and** *a*: *0* ∈ *T x* ∈ *X*
    by (*auto simp*: *assms*)

  have *tI*[*simp*]: *t* ∈ *T* **for** *t* **by** (*simp add*: *assms*)
  let *?T* = ((+) (− *s*) ‘ *existence-ivl s x*)
  have *shifted*: *is-interval ?T 0* ∈ *?T*
    by (*auto simp*: ‹*x* ∈ *X*›)

  have (λ*t*. *t* − *s*) = (+) (− *s*) **by** *auto*
  with *shift-autonomous-solution*[*OF flow-solves-ode*[*OF na*], *of s*] *flow-in-domain*
  have *sol*: ((λ*t*. *flow s x* (*t* + *s*)) *solves-ode f*) *?T X*
    by (*auto simp*: *auto* ‹*x* ∈ *X*›)

  have *flow s x* (*0* + *s*) = *x* **using** ‹*x* ∈ *X*› *flow-initial-time* **by** *simp*
  from *maximal-existence-flow*[*OF sol this shifted*]
  have ∗: *?T* ⊆ *existence-ivl 0 x*
    and ∗∗: ⋀*t*. *t* ∈ *?T* ⟹ *flow 0 x t* = *flow s x* (*t* + *s*)
    by (*auto simp*: *subset-iff*)

  have *t* − *s* ∈ *?T*
    using ‹*t* ∈ *existence-ivl s x*›
    by *auto*
  also note ∗

finally show $t - s \in$ *existence-ivl 0 x* **.**

show *flow s x t* $=$ *flow 0 x* $(t - s)$
  using ‹$t \in$ *existence-ivl s x*›
  by (*auto simp*: $**$)
qed

lemma (**in** *ll-on-open*)
  assumes $t - s \in$ *existence-ivl 0 x*
  assumes $x \in X$
  assumes *auto*: $\bigwedge s\ t\ x.\ x \in X \Longrightarrow f\ s\ x = f\ t\ x$
  assumes $T = UNIV$
  shows *mem-existence-ivl-shift-autonomous2*: $t \in$ *existence-ivl s x*
    and *flow-shift-autonomous2*: *flow s x t* $=$ *flow 0 x* $(t - s)$
proof $-$
  have *na*: $s \in T$ $x \in X$ **and** *a*: $0 \in T$ $x \in X$
    by (*auto simp*: *assms*)

  let $?T = ((+)\ s$ ‘ *existence-ivl 0 x*$)$
  have *shifted*: *is-interval* $?T$ $s \in ?T$
    by (*auto simp*: *a*)

  have $(\lambda t.\ t + s) = (+)\ s$
    by *auto*
  with *shift-autonomous-solution*[*OF flow-solves-ode*[*OF a*], *of* $-s$]
    flow-in-domain
  have *sol*: $((\lambda t.$ *flow 0 x* $(t - s))$ *solves-ode f*$)$ $?T$ X
    by (*auto simp*: *auto algebra-simps*)

  have *flow 0 x* $(s - s) = x$
    by (*auto simp*: *a*)
  from *maximal-existence-flow*[*OF sol this shifted*]
  have $*$: $?T \subseteq$ *existence-ivl s x*
    and $**$: $\bigwedge t.\ t \in ?T \Longrightarrow$ *flow s x t* $=$ *flow 0 x* $(t - s)$
    by (*auto simp*: *subset-iff assms*)

  have $t \in ?T$
    using ‹$t - s \in$ *existence-ivl 0 x*›
    by *force*
  also note $*$
  finally show $t \in$ *existence-ivl s x* **.**

  show *flow s x t* $=$ *flow 0 x* $(t - s)$
    using ‹$t - s \in$ *existence-ivl - -*›
    by (*subst* $**$; *force*)
qed

lemma
  flow-eq-rev:

assumes *t* ∈ *existence-ivl t0 x0*
shows *preflect t0 t* ∈ *ll-on-open.existence-ivl* (*preflect t0* ‘ *T*) (λ*t*. − *f* (*preflect t0 t*)) *X t0 x0*
flow t0 x0 t = *ll-on-open.flow* (*preflect t0* ‘ *T*) (λ*t*. − *f* (*preflect t0 t*)) *X t0 x0* (*preflect t0 t*)
proof −
from *mem-existence-ivl-iv-defined*[*OF assms*] **have** *mt0*: *t0* ∈ *preflect t0* ‘ *existence-ivl t0 x0*
by (*auto simp*: *preflect-def*)
have *subset*: *preflect t0* ‘ *existence-ivl t0 x0* ⊆ *preflect t0* ‘ *T*
using *existence-ivl-subset*
by (*rule image-mono*)
from *mt0 subset* **have** *t0* ∈ *preflect t0* ‘ *T* **by** *auto*

have *sol*: ((λ*t*. *flow t0 x0* (*preflect t0 t*)) *solves-ode* (λ*t*. − *f* (*preflect t0 t*))) (*preflect t0* ‘ *existence-ivl t0 x0*) *X*
using *mt0*
by (*rule preflect-solution*) (*auto simp*: *image-image flow-solves-ode mem-existence-ivl-iv-defined*[*OF assms*])

have *flow0*: *flow t0 x0* (*preflect t0 t0*) = *x0* **and** *ivl*: *is-interval* (*preflect t0* ‘ *existence-ivl t0 x0*)
by (*auto simp*: *preflect-def mem-existence-ivl-iv-defined*[*OF assms*])

interpret *rev*: *ll-on-open* (*preflect t0* ‘ *T*) (λ*t*. − *f* (*preflect t0 t*)) *X* **..**
from *rev.maximal-existence-flow*[*OF sol flow0 ivl mt0 subset*]
show *preflect t0 t* ∈ *rev.existence-ivl t0 x0 flow t0 x0 t* = *rev.flow t0 x0* (*preflect t0 t*)
using *assms* **by** (*auto simp*: *preflect-def*)
qed

lemma (**in** *ll-on-open*)
shows *rev-flow-eq*: *t* ∈ *ll-on-open.existence-ivl* (*preflect t0* ‘ *T*) (λ*t*. − *f* (*preflect t0 t*)) *X t0 x0* ⟹
ll-on-open.flow (*preflect t0* ‘ *T*) (λ*t*. − *f* (*preflect t0 t*)) *X t0 x0 t* = *flow t0 x0* (*preflect t0 t*)
and *mem-rev-existence-ivl-eq*:
t ∈ *ll-on-open.existence-ivl* (*preflect t0* ‘ *T*) (λ*t*. − *f* (*preflect t0 t*)) *X t0 x0* ⟷ *preflect t0 t* ∈ *existence-ivl t0 x0*
proof −
interpret *rev*: *ll-on-open* (*preflect t0* ‘ *T*) (λ*t*. − *f* (*preflect t0 t*)) *X* **..**
from *rev.flow-eq-rev*[*of - t0 x0*] *flow-eq-rev*[*of 2* ∗ *t0* − *t t0 x0*]
show *t* ∈ *rev.existence-ivl t0 x0* ⟹ *rev.flow t0 x0 t* = *flow t0 x0* (*preflect t0 t*)
(*t* ∈ *rev.existence-ivl t0 x0*) = (*preflect t0 t* ∈ *existence-ivl t0 x0*)
by (*auto simp*: *preflect-def fun-Compl-def image-image dest*: *mem-existence-ivl-iv-defined rev.mem-existence-ivl-iv-defined*)
qed

lemma

shows *rev-existence-ivl-eq*: *ll-on-open.existence-ivl* (*preflect t0* ‘ *T*) (λ*t*. − *f* (*preflect t0 t*)) *X t0 x0* = *preflect t0* ‘ *existence-ivl t0 x0*
and *existence-ivl-eq-rev*: *existence-ivl t0 x0* = *preflect t0* ‘ *ll-on-open.existence-ivl* (*preflect t0* ‘ *T*) (λ*t*. − *f* (*preflect t0 t*)) *X t0 x0*
apply *safe*
subgoal by (*force simp*: *mem-rev-existence-ivl-eq*)
subgoal by (*force simp*: *mem-rev-existence-ivl-eq*)
subgoal for *x* **by** (*force intro*!: *image-eqI*[**where** *x*=*preflect t0 x*] *simp*: *mem-rev-existence-ivl-eq*)
subgoal by (*force simp*: *mem-rev-existence-ivl-eq*)
done

end

end

3 Bounded Linear Operator

theory *Bounded-Linear-Operator*
imports
HOL−Analysis.Analysis
begin

typedef (**overloaded**) ′*a blinop* = *UNIV*::(′*a*, ′*a*) *blinfun set*
by *simp*

setup-lifting *type-definition-blinop*

lift-definition *blinop-apply*::(′*a*::*real-normed-vector*) *blinop* ⇒ ′*a* ⇒ ′*a* **is** *blinfun-apply* **.**
lift-definition *Blinop*::(′*a*::*real-normed-vector* ⇒ ′*a*) ⇒ ′*a blinop* **is** *Blinfun* **.**

no-notation *vec-nth* (**infixl** ‹$› *90*)
notation *blinop-apply* (**infixl** ‹$› *999*)
declare [[*coercion blinop-apply* :: (′*a*::*real-normed-vector*) *blinop* ⇒ ′*a* ⇒ ′*a*]]

instantiation *blinop* :: (*real-normed-vector*) *real-normed-vector*
begin

lift-definition *norm-blinop* :: ′*a blinop* ⇒ *real* **is** *norm* **.**

lift-definition *minus-blinop* :: ′*a blinop* ⇒ ′*a blinop* ⇒ ′*a blinop* **is** *minus* **.**

lift-definition *dist-blinop* :: ′*a blinop* ⇒ ′*a blinop* ⇒ *real* **is** *dist* **.**

definition *uniformity-blinop* :: (′*a blinop* × ′*a blinop*) *filter* **where**
uniformity-blinop = (*INF e*∈{*0*<..}. *principal* {(*x*, *y*). *dist x y* < *e*})

definition *open-blinop* :: ′*a blinop set* ⇒ *bool* **where**
open-blinop U = (∀ *x*∈*U*. ∀$_F$ (*x*′, *y*) *in uniformity*. *x*′ = *x* ⟶ *y* ∈ *U*)

lift-definition *uminus-blinop* :: *'a blinop* ⇒ *'a blinop* **is** *uminus* **.**

lift-definition *zero-blinop* :: *'a blinop* **is** *0* **.**

lift-definition *plus-blinop* :: *'a blinop* ⇒ *'a blinop* ⇒ *'a blinop* **is** *plus* **.**

lift-definition *scaleR-blinop*::*real* ⇒ *'a blinop* ⇒ *'a blinop* **is** *scaleR* **.**

lift-definition *sgn-blinop* :: *'a blinop* ⇒ *'a blinop* **is** *sgn* **.**

instance
  apply *standard*
  apply (*transfer'*, *simp add*: *algebra-simps sgn-div-norm open-uniformity norm-triangle-le*
    uniformity-blinop-def dist-norm
    open-blinop-def)+
  done
end

lemma *bounded-bilinear-blinop-apply*: *bounded-bilinear* ($)
  unfolding *bounded-bilinear-def*
  by *transfer* (*simp add*: *blinfun.bilinear-simps blinfun.bounded*)

interpretation *blinop*: *bounded-bilinear* ($)
  by (*rule bounded-bilinear-blinop-apply*)

lemma *blinop-eqI*: ($\bigwedge i.\ x\ \$\ i = y\ \$\ i$) $\Longrightarrow x = y$
  by *transfer* (*rule blinfun-eqI*)

lemmas *bounded-linear-apply-blinop*[*intro*, *simp*] = *blinop.bounded-linear-left*
declare *blinop.tendsto*[*tendsto-intros*]
declare *blinop.FDERIV*[*derivative-intros*]
declare *blinop.continuous*[*continuous-intros*]
declare *blinop.continuous-on*[*continuous-intros*]

instance *blinop* :: (*banach*) *banach*
  apply *standard*
  unfolding *convergent-def LIMSEQ-def Cauchy-def*
  apply *transfer*
  unfolding *convergent-def*[*symmetric*] *LIMSEQ-def*[*symmetric*] *Cauchy-def*[*symmetric*]
    Cauchy-convergent-iff
  .

instance *blinop* :: (*euclidean-space*) *heine-borel*
  apply *standard*
  unfolding *LIMSEQ-def bounded-def*
  apply *transfer*
  unfolding *LIMSEQ-def*[*symmetric*] *bounded-def*[*symmetric*]

```
  apply (rule bounded-imp-convergent-subsequence)
  .

instantiation blinop::({real-normed-vector, perfect-space}) real-normed-algebra-1
begin

lift-definition one-blinop::'a blinop is id-blinfun .
lemma blinop-apply-one-blinop[simp]: 1 $ x = x
  by transfer simp

lift-definition times-blinop :: 'a blinop ⇒ 'a blinop ⇒ 'a blinop is blinfun-compose
.

lemma blinop-apply-times-blinop[simp]: (f * g) $ x = f $ (g $ x)
  by transfer simp

instance
proof
  from not-open-singleton[of 0::'a] have {0::'a} ≠ UNIV by force
  then obtain x :: 'a where x ≠ 0 by auto
  show 0 ≠ (1::'a blinop)
    apply transfer
    apply transfer
    apply (auto dest!: fun-cong[where x=x] simp: ‹x ≠ 0›)
    done
qed (transfer, transfer,
  simp add: o-def linear-simps onorm-compose onorm-id onorm-compose[simplified
o-def])+
end

lemmas bounded-bilinear-bounded-uniform-limit-intros[uniform-limit-intros] =
  bounded-bilinear.bounded-uniform-limit[OF Bounded-Linear-Operator.bounded-bilinear-blinop-apply]
  bounded-bilinear.bounded-uniform-limit[OF Bounded-Linear-Function.bounded-bilinear-blinfun-apply]
  bounded-bilinear.bounded-uniform-limit[OF Bounded-Linear-Operator.blinop.flip]
  bounded-bilinear.bounded-uniform-limit[OF Bounded-Linear-Function.blinfun.flip]
  bounded-linear.uniform-limit[OF blinop.bounded-linear-right]
  bounded-linear.uniform-limit[OF blinop.bounded-linear-left]
  bounded-linear.uniform-limit[OF bounded-linear-apply-blinop]

no-notation
  blinop-apply (infixl ‹$› 999)
notation vec-nth (infixl ‹$› 90)

end
```

4 Multivariate Taylor

```
theory Multivariate-Taylor
imports
```

```
  HOL−Analysis.Analysis
  ../ODE-Auxiliarities
begin

no-notation vec-nth (infixl ‹$› 90)
notation blinfun-apply (infixl ‹$› 999)

lemma
  fixes f::'a::real-normed-vector ⇒ 'b::banach
    and Df::'a ⇒ nat ⇒ 'a ⇒ 'a ⇒ 'b
  assumes n > 0
  assumes Df-Nil: ⋀a x. Df a 0 H H = f a
  assumes Df-Cons: ⋀a i d. a ∈ closed-segment X (X + H) ⟹ i < n ⟹
      ((λa. Df a i H H) has-derivative (Df a (Suc i) H)) (at a within G)
  assumes cs: closed-segment X (X + H) ⊆ G
  defines i ≡ λx.
      ((1 − x) ^ (n − 1) / fact (n − 1)) *R Df (X + x *R H) n H H
  shows multivariate-Taylor-has-integral:
    (i has-integral f (X + H) − (∑ i<n. (1 / fact i) *R Df X i H H)) {0..1}
  and multivariate-Taylor:
    f (X + H) = (∑ i<n. (1 / fact i) *R Df X i H H) + integral {0..1} i
  and multivariate-Taylor-integrable:
    i integrable-on {0..1}
proof goal-cases
  case 1
  let ?G = closed-segment X (X + H)
  define line where line t = X + t *R H for t
  have segment-eq: closed-segment X (X + H) = line ‘ {0 .. 1}
    by (auto simp: line-def closed-segment-def algebra-simps)
  have line-deriv: ⋀x. (line has-derivative (λt. t *R H)) (at x)
    by (auto intro!: derivative-eq-intros simp: line-def [abs-def])
  define g where g = f o line
  define Dg where Dg n t = Df (line t) n H H for n :: nat and t :: real
  note ‹n > 0›
  moreover
  have Dg0: Dg 0 = g by (auto simp add: Dg-def Df-Nil g-def)
  moreover
  have DgSuc: (Dg m has-vector-derivative Dg (Suc m) t) (at t within {0..1})
    if m < n 0 ≤ t t ≤ 1 for m::nat and t::real
  proof −
    from that have [intro]: line t ∈ ?G using assms
      by (auto simp: segment-eq)
   note [derivative-intros] = has-derivative-in-compose[OF - has-derivative-subset[OF
Df-Cons]]
    interpret Df: linear (λd. Df (line t) (Suc m) H d)
      by (auto intro!: has-derivative-linear derivative-intros ‹m < n›)
    note [derivative-intros] =
      has-derivative-compose[OF - line-deriv]
    show ?thesis
```

using *Df.scaleR* ‹*m* < *n*›
by (*auto simp*: *Dg-def* [*abs-def*] *has-vector-derivative-def g-def segment-eq intro*!: *derivative-eq-intros subsetD*[*OF cs*])
qed
ultimately
have *g-Taylor*: (*i has-integral g 1* − ($\sum$ *i*<*n*. ((*1* − *0*) ^ *i* / *fact i*) $*_R$ *Dg i 0*)) {*0* .. *1*}
unfolding *i-def Dg-def* [*abs-def*] *line-def*
by (*rule Taylor-has-integral*) *auto*
then show *c*: *?case* **using** ‹*n* > *0*› **by** (*auto simp*: *g-def line-def Dg-def*)
case *2* **show** *?case* **using** *c*
by (*simp add*: *integral-unique add.commute*)
case *3* **show** *?case* **using** *c* **by** *force*
qed

4.1 Symmetric second derivative

lemma *symmetric-second-derivative-aux*:
assumes *first-fderiv*[*derivative-intros*]:
$\bigwedge$*a*. *a* ∈ *G* ⟹ (*f has-derivative* (*f′ a*)) (*at a within G*)
assumes *second-fderiv*[*derivative-intros*]:
$\bigwedge$*i*. ((λ*x*. *f′ x i*) *has-derivative* (λ*j*. *f″ j i*)) (*at a within G*)
assumes *i* ≠ *j i* ≠ *0 j* ≠ *0*
assumes *a* ∈ *G*
assumes $\bigwedge$*s t*. *s* ∈ {*0*..*1*} ⟹ *t* ∈ {*0*..*1*} ⟹ *a* + *s* $*_R$ *i* + *t* $*_R$ *j* ∈ *G*
shows *f″ j i* = *f″ i j*
proof −
let *?F* = *at-right* (*0*::*real*)
define *B* **where** *B i j* = {*a* + *s* $*_R$ *i* + *t* $*_R$ *j* |*s t*. *s* ∈ {*0*..*1*} ∧ *t* ∈ {*0*..*1*}} **for** *i j*
have *B i j* ⊆ *G* **using** *assms* **by** (*auto simp*: *B-def*)
{
fix *e*::*real* **and** *i j*::*′a*
assume *e* > *0*
assume *i* ≠ *j i* ≠ *0 j* ≠ *0*
assume *B i j* ⊆ *G*
let *?ij′* = λ*s t*. λ*u*. *a* + (*s* ∗ *u*) $*_R$ *i* + (*t* ∗ *u*) $*_R$ *j*
let *?ij* = λ*t*. λ*u*. *a* + (*t* ∗ *u*) $*_R$ *i* + *u* $*_R$ *j*
let *?i* = λ*t*. λ*u*. *a* + (*t* ∗ *u*) $*_R$ *i*
let *?g* = λ*u t*. *f* (*?ij t u*) − *f* (*?i t u*)
have *filter-ij′I*: $\bigwedge$*P*. *P a* ⟹ *eventually P* (*at a within G*) ⟹
eventually (λ*x*. ∀ *s*∈{*0*..*1*}. ∀ *t*∈{*0*..*1*}. *P* (*?ij′ s t x*)) *?F*
proof −
fix *P*
assume *P a*
assume *eventually P* (*at a within G*)
hence *eventually P* (*at a within B i j*) **by** (*rule filter-leD*[*OF at-le*[*OF* ‹*B i j* ⊆ *G*›]])
then obtain *d* **where** *d*: *d* > *0* **and** $\bigwedge$*x d2*. *x* ∈ *B i j* ⟹ *x* ≠ *a* ⟹ *dist x*

$a < d \Longrightarrow P\ x$
by (*auto simp*: *eventually-at*)
with ‹*P a*› **have** *P*: $\bigwedge x\ d2.\ x \in B\ i\ j \Longrightarrow dist\ x\ a < d \Longrightarrow P\ x$ **by** (*case-tac* $x = a$) *auto*
let $?d = min\ (min\ (d/norm\ i)\ (d/norm\ j)\ /\ 2)\ 1$
show *eventually* ($\lambda x.\ \forall s{\in}\{0..1\}.\ \forall t{\in}\{0..1\}.\ P\ (?ij'\ s\ t\ x)$) (*at-right 0*)
unfolding *eventually-at*
proof (*rule exI*[**where** *x=?d*], *safe*)
show $0 < ?d$ **using** ‹$0 < d$› ‹$i \neq 0$› ‹$j \neq 0$› **by** *simp*
fix *x s t* :: *real* **assume** ∗: $s \in \{0..1\}\ t \in \{0..1\}\ 0 < x\ dist\ x\ 0 < ?d$
show $P\ (?ij'\ s\ t\ x)$
proof (*rule P*)
have $\bigwedge x\ y{::}real.\ x \in \{0..1\} \Longrightarrow y \in \{0..1\} \Longrightarrow x * y \in \{0..1\}$
by (*auto intro*!: *order-trans*[*OF mult-left-le-one-le*])
hence $s * x \in \{0..1\}\ t * x \in \{0..1\}$ **using** ∗ **by** (*auto simp*: *dist-norm*)
thus $?ij'\ s\ t\ x \in B\ i\ j$ **by** (*auto simp*: *B-def*)
have $norm\ (s *_R x *_R i + t *_R x *_R j) \leq norm\ (s *_R x *_R i) + norm\ (t *_R x *_R j)$
by (*rule norm-triangle-ineq*)
also have $\ldots < d\ /\ 2 + d\ /\ 2$ **using** ∗ ‹$i \neq 0$› ‹$j \neq 0$›
by (*intro add-strict-mono*) (*auto simp*: *ac-simps dist-norm*
pos-less-divide-eq le-less-trans[*OF mult-left-le-one-le*])
finally show $dist\ (?ij'\ s\ t\ x)\ a < d$ **by** (*simp add*: *dist-norm*)
qed
qed
qed
have *filter-ijI*: *eventually* ($\lambda x.\ \forall t{\in}\{0..1\}.\ P\ (?ij\ t\ x)$) *?F*
if *P a eventually P* (*at a within G*) **for** *P*
using *filter-ij'I*[*OF that*]
by *eventually-elim* (*force dest*: *bspec*[**where** *x=1*])
have *filter-iI*: *eventually* ($\lambda x.\ \forall t{\in}\{0..1\}.\ P\ (?i\ t\ x)$) *?F*
if *P a eventually P* (*at a within G*) **for** *P*
using *filter-ij'I*[*OF that*] **by** *eventually-elim force*
{
from *second-fderiv*[*of i*, *simplified has-derivative-iff-norm*, *THEN conjunct2*, *THEN tendstoD*, *OF* ‹$0 < e$›]
have *eventually* ($\lambda x.\ norm\ (f'\ x\ i - f'\ a\ i - f''\ (x - a)\ i)\ /\ norm\ (x - a) \leq e$)
(*at a within G*)
by *eventually-elim* (*simp add*: *dist-norm*)
from *filter-ijI*[*OF - this*] *filter-iI*[*OF - this*] ‹$0 < e$›
have
eventually ($\lambda ij.\ \forall t{\in}\{0..1\}.\ norm\ (f'\ (?ij\ t\ ij)\ i - f'\ a\ i - f''\ (?ij\ t\ ij - a)\ i)\ /$
$norm\ (?ij\ t\ ij - a) \leq e$) *?F*
eventually ($\lambda ij.\ \forall t{\in}\{0..1\}.\ norm\ (f'\ (?i\ t\ ij)\ i - f'\ a\ i - f''\ (?i\ t\ ij - a)\ i)\ /$
$norm\ (?i\ t\ ij - a) \leq e$) *?F*
by *auto*

moreover
have *eventually* ($\lambda x.\ x \in G$) (*at a within G*) **unfolding** *eventually-at-filter* **by** *simp*
hence *eventually-in-ij*: *eventually* ($\lambda x.\ \forall\, t\in\{0..1\}.\ ?ij\ t\ x \in G$) *?F* **and**
eventually-in-i: *eventually* ($\lambda x.\ \forall\, t\in\{0..1\}.\ ?i\ t\ x \in G$) *?F*
using ‹$a \in G$› **by** (*auto dest*: *filter-ijI filter-iI*)
ultimately
have *eventually* ($\lambda u.\ norm\ (?g\ u\ 1 - ?g\ u\ 0 - (u * u) *_R f''\ j\ i) \leq$
$u * u * e * (2 * norm\ i + 3 * norm\ j))$ *?F*
proof *eventually-elim*
case (*elim u*)
hence *ijsub*: ($\lambda t.\ ?ij\ t\ u$) ‘ $\{0..1\} \subseteq G$ **and** *isub*: ($\lambda t.\ ?i\ t\ u$) ‘ $\{0..1\} \subseteq G$ **by** *auto*
note *has-derivative-subset*[*OF - ijsub*, *derivative-intros*]
note *has-derivative-subset*[*OF - isub*, *derivative-intros*]
let $?g' = \lambda t.\ (\lambda ua.\ u *_R ua *_R (f'\ (?ij\ t\ u)\ i - (f'\ (?i\ t\ u)\ i)))$
have g': ((*?g u*) *has-derivative* $?g'\ t$) (*at t within* $\{0..1\}$) **if** $t \in \{0..1\}$ **for** *t*::*real*
proof −
from *elim that* **have** *linear-f'*: $\bigwedge c\ x.\ f'\ (?ij\ t\ u)\ (c *_R x) = c *_R f'\ (?ij\ t\ u)\ x$
$\bigwedge c\ x.\ f'\ (?i\ t\ u)\ (c *_R x) = c *_R f'\ (?i\ t\ u)\ x$
using *linear-cmul*[*OF has-derivative-linear*, *OF first-fderiv*] **by** *auto*
show *?thesis*
using *elim* ‹$t \in \{0..1\}$›
by (*auto intro*!: *derivative-eq-intros has-derivative-in-compose*[*of* $\lambda t.\ ?ij\ t\ u$ *- - - f*]
has-derivative-in-compose[*of* $\lambda t.\ ?i\ t\ u$ *- - - f*]
simp: *linear-f' scaleR-diff-right mult.commute*)
qed
from *elim*(*1*) ‹$i \neq 0$› ‹$j \neq 0$› ‹$0 < e$› **have** *f'ij*: $\bigwedge t.\ t \in \{0..1\} \Longrightarrow$
$norm\ (f'\ (a + (t * u) *_R i + u *_R j)\ i - f'\ a\ i - f''\ ((t * u) *_R i + u *_R j)\ i) \leq$
$e * norm\ ((t * u) *_R i + u *_R j)$
using *linear-0*[*OF has-derivative-linear*, *OF second-fderiv*]
by (*case-tac* $u *_R j + (t * u) *_R i = 0$) (*auto simp*: *field-simps*
simp del: *pos-divide-le-eq simp add*: *pos-divide-le-eq*[*symmetric*])
from *elim*(*2*) **have** *f'i*: $\bigwedge t.\ t \in \{0..1\} \Longrightarrow norm\ (f'\ (a + (t * u) *_R i)\ i - f'\ a\ i -$
$f''\ ((t * u) *_R i)\ i) \leq e * abs\ (t * u) * norm\ i$
using ‹$i \neq 0$› ‹$j \neq 0$› *linear-0*[*OF has-derivative-linear*, *OF second-fderiv*]
by (*case-tac* $t * u = 0$) (*auto simp*: *field-simps simp del*: *pos-divide-le-eq*
simp add: *pos-divide-le-eq*[*symmetric*])
have $norm\ (?g\ u\ 1 - ?g\ u\ 0 - (u * u) *_R f''\ j\ i) =$
$norm\ ((?g\ u\ 1 - ?g\ u\ 0 - u *_R (f'\ (a + u *_R j)\ i - (f'\ a\ i)))$
$+ u *_R (f'\ (a + u *_R j)\ i - f'\ a\ i - u *_R f''\ j\ i))$
(**is** - = *norm* (*?g10* + *?f'i*))
by (*simp add*: *algebra-simps linear-cmul*[*OF has-derivative-linear*, *OF second-fderiv*]

linear-add[*OF has-derivative-linear*, *OF second-fderiv*])
also have ... ≤ *norm ?g10* + *norm ?f'i*
by (*blast intro*: *order-trans add-mono norm-triangle-le*)
also
have *0* ∈ {*0..1*::*real*} **by** *simp*
have ∀ *t* ∈ {*0..1*}. *onorm* ((λ*ua*. (*u* ∗ *ua*) ∗$_R$ (*f'* (*?ij t u*) *i* − *f'* (*?i t u*) *i*)) −
(λ*ua*. (*u* ∗ *ua*) ∗$_R$ (*f'* (*a* + *u* ∗$_R$ *j*) *i* − *f'* *a i*)))
≤ *2* ∗ *u* ∗ *u* ∗ *e* ∗ (*norm i* + *norm j*) (**is** ∀ *t* ∈ -. *onorm* (*?d t*) ≤ -)
proof
fix *t*::*real* **assume** *t* ∈ {*0..1*}
show *onorm* (*?d t*) ≤ *2* ∗ *u* ∗ *u* ∗ *e* ∗ (*norm i* + *norm j*)
proof (*rule onorm-le*)
fix *x*
have *norm* (*?d t x*) =
norm ((*u* ∗ *x*) ∗$_R$ (*f'* (*?ij t u*) *i* − *f'* (*?i t u*) *i* − *f'* (*a* + *u* ∗$_R$ *j*) *i* + *f'* *a i*))
by (*simp add*: *algebra-simps*)
also have ... =
abs (*u* ∗ *x*) ∗ *norm* (*f'* (*?ij t u*) *i* − *f'* (*?i t u*) *i* − *f'* (*a* + *u* ∗$_R$ *j*) *i* + *f'* *a i*)
by *simp*
also have ... = *abs* (*u* ∗ *x*) ∗ *norm* (
f' (*?ij t u*) *i* − *f'* *a i* − *f''* ((*t* ∗ *u*) ∗$_R$ *i* + *u* ∗$_R$ *j*) *i*
− (*f'* (*?i t u*) *i* − *f'* *a i* − *f''* ((*t* ∗ *u*) ∗$_R$ *i*) *i*)
− (*f'* (*a* + *u* ∗$_R$ *j*) *i* − *f'* *a i* − *f''* (*u* ∗$_R$ *j*) *i*))
(**is** - = - ∗ *norm* (*?dij* − *?di* − *?dj*))
using ‹*a* ∈ *G*›
by (*simp add*: *algebra-simps*
linear-add[*OF has-derivative-linear*[*OF second-fderiv*]])
also have ... ≤ *abs* (*u* ∗ *x*) ∗ (*norm ?dij* + *norm ?di* + *norm ?dj*)
by (*rule mult-left-mono*[*OF - abs-ge-zero*]) *norm*
also have ... ≤ *abs* (*u* ∗ *x*) ∗
(*e* ∗ *norm* ((*t* ∗ *u*) ∗$_R$ *i* + *u* ∗$_R$ *j*) + *e* ∗ *abs* (*t* ∗ *u*) ∗ *norm i* + *e* ∗ (|*u*| ∗ *norm j*))
using *f'ij f'i f'ij*[*OF* ‹*0* ∈ {*0..1*}›] ‹*t* ∈ {*0..1*}›
by (*auto intro*!: *add-mono mult-left-mono*)
also have ... = *abs u* ∗ *abs x* ∗ *abs u* ∗
(*e* ∗ *norm* (*t* ∗$_R$ *i* + *j*) + *e* ∗ *norm* (*t* ∗$_R$ *i*) + *e* ∗ (*norm j*))
by (*simp add*: *algebra-simps norm-scaleR*[*symmetric*] *abs-mult del*: *norm-scaleR*)
also have ... =
u ∗ *u* ∗ *abs x* ∗ (*e* ∗ *norm* (*t* ∗$_R$ *i* + *j*) + *e* ∗ *norm* (*t* ∗$_R$ *i*) + *e* ∗ (*norm j*))
by (*simp add*: *ac-simps*)
also have ... = *u* ∗ *u* ∗ *e* ∗ *abs x* ∗ (*norm* (*t* ∗$_R$ *i* + *j*) + *norm* (*t* ∗$_R$ *i*) + *norm j*)
by (*simp add*: *algebra-simps*)
also have ... ≤ *u* ∗ *u* ∗ *e* ∗ *abs x* ∗ ((*norm* (*1* ∗$_R$ *i*) + *norm j*) + *norm*

(*1* $*_R$ *i*) + *norm j*)
using ‹*t* ∈ {*0..1*}› ‹*0* < *e*›
by (*intro mult-left-mono add-mono*) (*auto intro*!: *norm-triangle-le add-right-mono*
mult-left-le-one-le zero-le-square)
finally show *norm* (*?d t x*) ≤ *2* ∗ *u* ∗ *u* ∗ *e* ∗ (*norm i* + *norm j*) ∗ *norm x*
by (*simp add*: *ac-simps*)
qed
qed
with *differentiable-bound-linearization*[**where** *f*=*?g u* **and** *f′*=*?g′*, *of 0 1 - 0*, *OF - g′*]
have *norm ?g10* ≤ *2* ∗ *u* ∗ *u* ∗ *e* ∗ (*norm i* + *norm j*) **by** *simp*
also have *norm ?f′i* ≤ *abs u* ∗
norm ((*f′* (*a* + (*u*) $*_R$ *j*) *i* − *f′ a i* − *f″* (*u* $*_R$ *j*) *i*))
using *linear-cmul*[*OF has-derivative-linear*, *OF second-fderiv*]
by *simp*
also have ... ≤ *abs u* ∗ (*e* ∗ *norm* ((*u*) $*_R$ *j*))
using *f′ij*[*OF* ‹*0* ∈ {*0..1*}›] **by** (*auto intro*: *mult-left-mono*)
also have ... = *u* ∗ *u* ∗ *e* ∗ *norm j* **by** (*simp add*: *algebra-simps abs-mult*)
finally show *?case* **by** (*simp add*: *algebra-simps*)
qed
}
} **note** *wlog* = *this*
have *e′*: *norm* (*f″ j i* − *f″ i j*) ≤ *e* ∗ (*5* ∗ *norm j* + *5* ∗ *norm i*) **if** *0* < *e* **for** *e t*::*real*
proof −
have *B i j* = *B j i* **using** ‹*i* ≠ *j*› **by** (*force simp*: *B-def*)+
with *assms* ‹*B i j* ⊆ *G*› **have** *j* ≠ *i B j i* ⊆ *G* **by** (*auto simp*:)
from *wlog*[*OF* ‹*0* < *e*› ‹*i* ≠ *j*› ‹*i* ≠ *0*› ‹*j* ≠ *0*› ‹*B i j* ⊆ *G*›]
wlog[*OF* ‹*0* < *e*› ‹*j* ≠ *i*› ‹*j* ≠ *0*› ‹*i* ≠ *0*› ‹*B j i* ⊆ *G*›]
have *eventually* (λ*u*. *norm* ((*u* ∗ *u*) $*_R$ *f″ j i* − (*u* ∗ *u*) $*_R$ *f″ i j*)
≤ *u* ∗ *u* ∗ *e* ∗ (*5* ∗ *norm j* + *5* ∗ *norm i*)) *?F*
proof *eventually-elim*
case (*elim u*)
have *norm* ((*u* ∗ *u*) $*_R$ *f″ j i* − (*u* ∗ *u*) $*_R$ *f″ i j*) =
norm (*f* (*a* + *u* $*_R$ *j* + *u* $*_R$ *i*) − *f* (*a* + *u* $*_R$ *j*) −
(*f* (*a* + *u* $*_R$ *i*) − *f a*) − (*u* ∗ *u*) $*_R$ *f″ i j*
− (*f* (*a* + *u* $*_R$ *i* + *u* $*_R$ *j*) − *f* (*a* + *u* $*_R$ *i*) −
(*f* (*a* + *u* $*_R$ *j*) − *f a*) −
(*u* ∗ *u*) $*_R$ *f″ j i*)) **by** (*simp add*: *field-simps*)
also have ... ≤ *u* ∗ *u* ∗ *e* ∗ (*2* ∗ *norm j* + *3* ∗ *norm i*) + *u* ∗ *u* ∗ *e* ∗ (*3* ∗ *norm j* + *2* ∗ *norm i*)
using *elim* **by** (*intro order-trans*[*OF norm-triangle-ineq4*]) (*auto simp*: *ac-simps intro*: *add-mono*)
finally show *?case* **by** (*simp add*: *algebra-simps*)
qed
hence *eventually* (λ*u*. *norm* ((*u* ∗ *u*) $*_R$ (*f″ j i* − *f″ i j*)) ≤
u ∗ *u* ∗ *e* ∗ (*5* ∗ *norm j* + *5* ∗ *norm i*)) *?F*

```
      by (simp add: algebra-simps)
    hence eventually (λu. (u * u) * norm ((f'' j i − f'' i j)) ≤
        (u * u) * (e * (5 * norm j + 5 * norm i))) ?F
      by (simp add: ac-simps)
    hence eventually (λu. norm ((f'' j i − f'' i j)) ≤ e * (5 * norm j + 5 * norm
i)) ?F
      unfolding mult-le-cancel-left eventually-at-filter
      by eventually-elim auto
    then show ?thesis
      by (auto simp add:eventually-at dist-norm dest!: bspec[where x=d/2 for d])
  qed
  have e: norm (f'' j i − f'' i j) < e if 0 < e for e::real
  proof −
    let ?e = e/2/(5 * norm j + 5 * norm i)
    have ?e > 0 using ‹0 < e› ‹i ≠ 0› ‹j ≠ 0› by (auto intro!: divide-pos-pos
add-pos-pos)
    from e'[OF this] have norm (f'' j i − f'' i j) ≤ ?e * (5 * norm j + 5 * norm
i) .
    also have ... = e / 2 using ‹i ≠ 0› ‹j ≠ 0› by (auto simp: ac-simps
add-nonneg-eq-0-iff)
    also have ... < e using ‹0 < e› by simp
    finally show ?thesis .
  qed
  have norm (f'' j i − f'' i j) = 0
  proof (rule ccontr)
    assume norm (f'' j i − f'' i j) ≠ 0
    hence norm (f'' j i − f'' i j) > 0 by simp
    from e[OF this] show False by simp
  qed
  thus ?thesis by simp
qed

locale second-derivative-within =
  fixes f f' f'' a G
  assumes first-fderiv[derivative-intros]:
    ⋀a. a ∈ G ⟹ (f has-derivative blinfun-apply (f' a)) (at a within G)
  assumes in-G: a ∈ G
  assumes second-fderiv[derivative-intros]:
    (f' has-derivative blinfun-apply f'') (at a within G)
begin

lemma symmetric-second-derivative-within:
  assumes a ∈ G
  assumes ⋀s t. s ∈ {0..1} ⟹ t ∈ {0..1} ⟹ a + s *R i + t *R j ∈ G
  shows f'' i j = f'' j i
  apply (cases i = j ∨ i = 0 ∨ j = 0)
    apply (force simp add: blinfun.zero-right blinfun.zero-left)
  using first-fderiv - - - - assms
  by (rule symmetric-second-derivative-aux[symmetric])
```

```
  (auto intro!: derivative-eq-intros simp: blinfun.bilinear-simps assms)

end

locale second-derivative =
  fixes f::'a::real-normed-vector ⇒ 'b::banach
    and f' :: 'a ⇒ 'a ⇒L 'b
    and f'' :: 'a ⇒L 'a ⇒L 'b
    and a :: 'a
    and G :: 'a set
  assumes first-fderiv[derivative-intros]:
    ⋀a. a ∈ G ⟹ (f has-derivative f' a) (at a)
  assumes in-G: a ∈ interior G
  assumes second-fderiv[derivative-intros]:
    (f' has-derivative f'') (at a)
begin

lemma symmetric-second-derivative:
  assumes a ∈ interior G
  shows f'' i j = f'' j i
proof −
  from assms have a ∈ G
    using interior-subset by blast
  interpret second-derivative-within
    by unfold-locales
      (auto intro!: derivative-intros intro: has-derivative-at-withinI ‹a ∈ G›)
  from assms open-interior[of G] interior-subset[of G]
  obtain e where e: e > 0 ⋀y. dist y a < e ⟹ y ∈ G
    by (force simp: open-dist)
  define e' where e' = e / 3
  define i' j' where i' = e' *R i /R norm i and j' = e' *R j /R norm j
  hence norm i' ≤ e' norm j' ≤ e'
    by (auto simp: field-simps e'-def ‹0 < e› less-imp-le)
  hence |s| ≤ 1 ⟹ |t| ≤ 1 ⟹ norm (s *R i' + t *R j') ≤ e' + e' for s t
    by (intro norm-triangle-le[OF add-mono])
      (auto intro!: order-trans[OF mult-left-le-one-le])
  also have ... < e by (simp add: e'-def ‹0 < e›)
  finally
  have f'' $ i' $ j' = f'' $ j' $ i'
    by (intro symmetric-second-derivative-within ‹a ∈ G› e)
      (auto simp add: dist-norm)
  thus ?thesis
    using e(1)
    by (auto simp: i'-def j'-def e'-def
      blinfun.zero-right blinfun.zero-left
      blinfun.scaleR-left blinfun.scaleR-right algebra-simps)
qed

end
```

lemma
uniform-explicit-remainder-Taylor-1:
fixes *f*::*'a*::{*banach*,*heine-borel*,*perfect-space*} ⇒ *'b*::*banach*
assumes *f'*[*derivative-intros*]: ⋀*x*. *x* ∈ *G* ⟹ (*f has-derivative blinfun-apply* (*f'* *x*)) (*at x*)
assumes *f'-cont*: ⋀*x*. *x* ∈ *G* ⟹ *isCont f' x*
assumes *open G*
assumes *J* ≠ {} *compact J J* ⊆ *G*
assumes *e* > *0*
obtains *d R*
where *d* > *0*
⋀*x z*. *f z* = *f x* + *f' x* (*z* − *x*) + *R x z*
⋀*x y*. *x* ∈ *J* ⟹ *y* ∈ *J* ⟹ *dist x y* < *d* ⟹ *norm* (*R x y*) ≤ *e* ∗ *dist x y*
continuous-on (*G* × *G*) (λ(*a*, *b*). *R a b*)
proof −
from *assms* **have** *continuous-on G f'* **by** (*auto intro*!: *continuous-at-imp-continuous-on*)
note [*continuous-intros*] = *continuous-on-compose2*[*OF this*]
define *R* **where** *R x z* = *f z* − *f x* − *f' x* (*z* − *x*) **for** *x z*
from *compact-in-open-separated*[*OF* ‹*J* ≠ {}› ‹*compact J*› ‹*open G*› ‹*J* ⊆ *G*›]
obtain η **where** η: *0* < η {*x*. *infdist x J* ≤ η} ⊆ *G* (**is** *?J'* ⊆ -)
by *auto*
hence *infdist-in-G*: *infdist x J* ≤ η ⟹ *x* ∈ *G* **for** *x*
by *auto*
have *dist-in-G*: ⋀*y*. *dist x y* < η ⟹ *y* ∈ *G* **if** *x* ∈ *J* **for** *x*
by (*auto intro*!: *infdist-in-G infdist-le2 that simp*: *dist-commute*)

have *compact ?J'* **by** (*rule compact-infdist-le*; *fact*)
let *?seg* = *?J'*
from ‹*continuous-on G f'*›
have *ucont*: *uniformly-continuous-on ?seg f'*
using ‹*?seg* ⊆ *G*›
by (*auto intro*!: *compact-uniformly-continuous* ‹*compact ?seg*› *intro*: *continuous-on-subset*)

define *e'* **where** *e'* = *e* / *2*
have *e'* > *0* **using** ‹*e* > *0*› **by** (*simp add*: *e'-def*)
from *ucont*[*unfolded uniformly-continuous-on-def*, *rule-format*, *OF* ‹*0* < *e'*›]
obtain *du* **where** *du*:
du > *0*
⋀*x y*. *x* ∈ *?seg* ⟹ *y* ∈ *?seg* ⟹ *dist x y* < *du* ⟹ *norm* (*f' x* − *f' y*) < *e'*
by (*auto simp*: *dist-norm*)
have *min* η *du* > *0* **using** ‹*du* > *0*› ‹η > *0*› **by** *simp*
moreover
have *f z* = *f x* + *f' x* (*z* − *x*) + *R x z* **for** *x z*
by (*auto simp*: *R-def*)
moreover
{
fix *x z*::*'a*

assume $x \in J$ $z \in J$
hence $x \in G$ $z \in G$ **using** *assms* **by** *auto*

assume *dist x z* $<$ *min* η *du*
hence *d-eta*: *dist x z* $< \eta$ **and** *d-du*: *dist x z* $<$ *du*
by (*auto simp add*: *min-def split*: *if-split-asm*)

from ‹*dist x z* $< \eta$› **have** *line-in*:
$\bigwedge xa.\ 0 \le xa \Longrightarrow xa \le 1 \Longrightarrow x + xa *_R (z - x) \in G$
$(\lambda xa.\ x + xa *_R (z - x))$ ‘ $\{0..1\} \subseteq G$
by (*auto intro*!: *dist-in-G* ‹$x \in J$› *le-less-trans*[*OF mult-left-le-one-le*]
simp: *dist-norm norm-minus-commute*)

have $R\ x\ z = f\ z - f\ x - f'\ x\ (z - x)$
by (*simp add*: *R-def*)
also have $f\ z - f\ x = f\ (x + (z - x)) - f\ x$ **by** *simp*
also have $f\ (x + (z - x)) - f\ x =$ *integral* $\{0..1\}$ $(\lambda t.\ (f'\ (x + t *_R (z - x)))$
$(z - x))$
using ‹*dist x z* $< \eta$›
by (*intro mvt-integral*[*of ball x* η *f f' x z* $-$ *x*])
(*auto simp*: *dist-norm norm-minus-commute at-within-ball* ‹$0 < \eta$› *mem-ball*
intro!: *le-less-trans*[*OF mult-left-le-one-le*] *derivative-eq-intros dist-in-G* ‹x
$\in J$›)
also have
(*integral* $\{0..1\}$ $(\lambda t.\ (f'\ (x + t *_R (z - x)))\ (z - x)) - (f'\ x)\ (z - x)) =$
integral $\{0..1\}$ $(\lambda t.\ f'\ (x + t *_R (z - x)) - f'\ x)\ (z - x)$
by (*simp add*: *Henstock-Kurzweil-Integration.integral-diff integral-linear*[**where**
$h=\lambda y.$ *blinfun-apply y* $(z - x)$, *simplified o-def*]
integrable-continuous-real continuous-intros line-in
blinfun.bilinear-simps[*symmetric*])
finally have $R\ x\ z =$ *integral* $\{0..1\}$ $(\lambda t.\ f'\ (x + t *_R (z - x)) - f'\ x)\ (z -$
$x)$
.
also have *norm* $\ldots \le$ *norm* (*integral* $\{0..1\}$ $(\lambda t.\ f'\ (x + t *_R (z - x)) - f'$
$x)) *$ *norm* $(z - x)$
by (*auto intro*!: *order-trans*[*OF norm-blinfun*])
also have $\ldots \le e' * (1 - 0) *$ *norm* $(z - x)$
using *d-eta d-du* ‹$0 < \eta$›
by (*intro mult-right-mono integral-bound*)
(*auto simp*: *dist-norm norm-minus-commute*
intro!: *line-in du*[*THEN less-imp-le*] *infdist-le2*[*OF* ‹$x \in J$›] *line-in contin-*
uous-intros
order-trans[*OF mult-left-le-one-le*] *le-less-trans*[*OF mult-left-le-one-le*])
also have $\ldots \le e *$ *dist x z* **using** ‹$0 < e$› **by** (*simp add*: *e'-def norm-minus-commute*
dist-norm)
finally have *norm* $(R\ x\ z) \le e *$ *dist x z* **.**
}
moreover
{

from f' **have** *f-cont*: *continuous-on G f*
by (*rule has-derivative-continuous-on*[*OF has-derivative-at-withinI*])
note [*continuous-intros*] = *continuous-on-compose2*[*OF this*]
from f'*-cont* **have** f'*-cont*: *continuous-on G* f'
by (*auto intro*!: *continuous-at-imp-continuous-on*)

note *continuous-on-diff2*=*continuous-on-diff*[*OF continuous-on-compose*[*OF continuous-on-snd*] *continuous-on-compose*[*OF continuous-on-fst*], **where** s=$G \times G$, *simplified*]
have *continuous-on* $(G \times G)$ $(\lambda(a, b).\ f\ b - f\ a)$
by (*auto intro*!: *continuous-intros simp*: *split-beta*)
moreover have *continuous-on* $(G \times G)$ $(\lambda(a, b).\ f'\ a\ (b - a))$
by (*auto intro*!: *continuous-intros simp*: *split-beta'*)
ultimately have *continuous-on* $(G \times G)$ $(\lambda(a, b).\ R\ a\ b)$
by (*rule iffD1*[*OF continuous-on-cong*[*OF refl*] *continuous-on-diff*, *rotated*], *auto simp*: *R-def*)
}
ultimately
show *thesis* **..**
qed

TODO: rename, duplication?

locale *second-derivative-within'* =
fixes $f\ f'\ f''\ a\ G$
assumes *first-fderiv*[*derivative-intros*]:
$\bigwedge a.\ a \in G \Longrightarrow$ (*f has-derivative* f' *a*) (*at a within G*)
assumes *in-G*: $a \in G$
assumes *second-fderiv*[*derivative-intros*]:
$\bigwedge i.$ $((\lambda x.\ f'\ x\ i)$ *has-derivative* $f''\ i)$ (*at a within G*)
begin

lemma *symmetric-second-derivative-within*:
assumes $a \in G$ *open G*
assumes $\bigwedge s\ t.\ s \in \{0..1\} \Longrightarrow t \in \{0..1\} \Longrightarrow a + s *_R i + t *_R j \in G$
shows $f''\ i\ j = f''\ j\ i$
proof (*cases* $i = j \lor i = 0 \lor j = 0$)
case *True*
interpret *bounded-linear* $f''\ k$ **for** k
by (*rule has-derivative-bounded-linear*) (*rule second-fderiv*)
have *z1*: $f''\ j\ 0 = 0$ $f''\ i\ 0 = 0$ **by** (*simp-all add*: *zero*)
have *f'z*: $f'\ x\ 0 = 0$ **if** $x \in G$ **for** x
proof −
interpret *bounded-linear* $f'\ x$
by (*rule has-derivative-bounded-linear*) (*rule first-fderiv that*)+
show *?thesis* **by** (*simp add*: *zero*)
qed
note *aw* = *at-within-open*[*OF* ‹$a \in G$› ‹*open G*›]
have $((\lambda x.\ f'\ x\ 0)$ *has-derivative* $(\lambda$-. $0))$ (*at a within G*)
apply (*rule has-derivative-transform-within*)

apply (*rule has-derivative-const*[**where** *c=0*])
apply (*rule zero-less-one*)
apply *fact*
by (*simp add*: *f′z*)
from *has-derivative-unique*[*OF second-fderiv*[*unfolded aw*] *this*[*unfolded aw*]]
have $f''\ 0 = (\lambda\text{-}.\ 0)$ **.**
with *True z1* **show** *?thesis*
by (*auto*)
next
case *False*
show *?thesis*
using *first-fderiv - - - - assms*(*1*,*3*−)
by (*rule symmetric-second-derivative-aux*[])
(*use False* **in** ‹*auto intro*!: *derivative-eq-intros simp*: *blinfun.bilinear-simps assms*›)
qed

end

locale *second-derivative-on-open* =
fixes *f*::′*a*::*real-normed-vector* ⇒ ′*b*::*banach*
and *f′* :: ′*a* ⇒ ′*a* ⇒ ′*b*
and *f″* :: ′*a* ⇒ ′*a* ⇒ ′*b*
and *a* :: ′*a*
and *G* :: ′*a set*
assumes *first-fderiv*[*derivative-intros*]:
$\bigwedge a.\ a \in G \Longrightarrow$ (*f has-derivative f′ a*) (*at a*)
assumes *in-G*: $a \in G$ **and** *open-G*: *open G*
assumes *second-fderiv*[*derivative-intros*]:
((λ*x*. *f′ x i*) *has-derivative f″ i*) (*at a*)
begin

lemma *symmetric-second-derivative*:
assumes $a \in G$
shows $f''\ i\ j = f''\ j\ i$
proof −
interpret *second-derivative-within′*
by *unfold-locales*
(*auto intro*!: *derivative-intros intro*: *has-derivative-at-withinI* ‹$a \in G$›)
from ‹$a \in G$› *open-G*
obtain *e* **where** *e*: $e > 0$ $\bigwedge y.\ dist\ y\ a < e \Longrightarrow y \in G$
by (*force simp*: *open-dist*)
define *e′* **where** $e' = e\ /\ 3$
define *i′ j′* **where** $i' = e' *_R i /_R norm\ i$ **and** $j' = e' *_R j /_R norm\ j$
hence $norm\ i' \le e'$ $norm\ j' \le e'$
by (*auto simp*: *field-simps e′-def* ‹$0 < e$› *less-imp-le*)
hence $|s| \le 1 \Longrightarrow |t| \le 1 \Longrightarrow norm\ (s *_R i' + t *_R j') \le e' + e'$ **for** *s t*
by (*intro norm-triangle-le*[*OF add-mono*])
(*auto intro*!: *order-trans*[*OF mult-left-le-one-le*])

also have ... < *e* **by** (*simp add*: *e′-def* ‹*0* < *e*›)
finally
have *f″ i′ j′* = *f″ j′ i′*
  by (*intro symmetric-second-derivative-within* ‹*a* ∈ *G*› *e*)
    (*auto simp add*: *dist-norm open-G*)
moreover
interpret *f″*: *bounded-linear f″ k* **for** *k*
  by (*rule has-derivative-bounded-linear*) (*rule second-fderiv*)
note *aw* = *at-within-open*[*OF* ‹*a* ∈ *G*› ‹*open G*›]
have *z1*: *f″ j 0* = *0 f″ i 0* = *0* **by** (*simp-all add*: *f″.zero*)
have *f′z*: *f′ x 0* = *0* **if** *x* ∈ *G* **for** *x*
proof −
  interpret *bounded-linear f′ x*
    by (*rule has-derivative-bounded-linear*) (*rule first-fderiv that*)+
  show *?thesis* **by** (*simp add*: *zero*)
qed
have ((λ*x*. *f′ x 0*) *has-derivative* (λ-. *0*)) (*at a within G*)
  apply (*rule has-derivative-transform-within*)
     apply (*rule has-derivative-const*[**where** *c*=*0*])
    apply (*rule zero-less-one*)
   apply *fact*
  by (*simp add*: *f′z*)
from *has-derivative-unique*[*OF second-fderiv*[*unfolded aw*] *this*[*unfolded aw*]]
have *z2*: *f″ 0* = (λ-. *0*) .
have ((λ*a*. *f′ a* (*r* $*_R$ *x*)) *has-derivative f″* (*r* $*_R$ *x*)) (*at a within G*)
  ((λ*a*. *f′ a* (*r* $*_R$ *x*)) *has-derivative* (λ*y*. *r* $*_R$ *f″ x y*)) (*at a within G*)
  for *r x*
  subgoal by (*rule second-fderiv*)
  subgoal
  proof −
    have ((λ*a*. *r* $*_R$ *f′ a* (*x*)) *has-derivative* (λ*y*. *r* $*_R$ *f″ x y*)) (*at a within G*)
      by (*auto intro*!: *derivative-intros*)
    then show *?thesis*
      apply (*rule has-derivative-transform*[*rotated 2*])
       apply (*rule in-G*)
      subgoal premises *prems* **for** *a′*
      proof −
        interpret *bounded-linear f′ a′*
          apply (*rule has-derivative-bounded-linear*)
          by (*rule first-fderiv*[*OF prems*])
        show *?thesis*
          by (*simp add*: *scaleR*)
      qed
      done
  qed
  done
then have ((λ*a*. *f′ a* (*r* $*_R$ *x*)) *has-derivative f″* (*r* $*_R$ *x*)) (*at a*)
  ((λ*a*. *f′ a* (*r* $*_R$ *x*)) *has-derivative* (λ*y*. *r* $*_R$ *f″ x y*)) (*at a*) **for** *r x*
  unfolding *aw* **by** *auto*

then have $f'z$: f'' $(r *_R x) = (\lambda y.\ r *_R f''\ x\ y)$ **for** *r x*
  by (*rule has-derivative-unique*[**where** $f{=}(\lambda a.\ f'\ a\ (r *_R x))$])
  ultimately show *?thesis*
    using *e*(*1*)
    by (*auto simp*: *i′-def j′-def e′-def f″.scaleR z1 z2*
      blinfun.zero-right blinfun.zero-left
      blinfun.scaleR-left blinfun.scaleR-right algebra-simps)
qed

end

no-notation
  blinfun-apply (**infixl** ‹$› *999*)
notation *vec-nth* (**infixl** ‹$› *90*)

end

5 Flow

theory *Flow*
imports
  Picard-Lindeloef-Qualitative
  HOL−Library.Diagonal-Subsequence
  ../Library/Bounded-Linear-Operator
  ../Library/Multivariate-Taylor
  ../Library/Interval-Integral-HK
begin

TODO: extend theorems for dependence on initial time

5.1 simp rules for integrability (TODO: move)

lemma *blinfun-ext*: $x = y \longleftrightarrow (\forall i.$ *blinfun-apply x i* = *blinfun-apply y i*)
  by *transfer auto*

notation *id-blinfun* (‹1_L›)

lemma *blinfun-inverse-left*:
  fixes f::$'a$::*euclidean-space* $\Rightarrow_L$ $'a$ **and** f'
  shows $f\ o_L\ f' = 1_L \longleftrightarrow f'\ o_L\ f = 1_L$
  by *transfer*
    (*auto dest*!: *bounded-linear.linear simp*: *id-def*[*symmetric*]
      linear-inverse-left)

lemma *onorm-zero-blinfun*[*simp*]: *onorm* (*blinfun-apply 0*) = *0*
  by *transfer* (*simp add*: *onorm-zero*)

lemma *blinfun-compose-1-left*[*simp*]: $x\ o_L\ 1_L = x$
  and *blinfun-compose-1-right*[*simp*]: $1_L\ o_L\ y = y$

by (*auto intro*!: *blinfun-eqI*)

named-theorems *integrable-on-simps*

lemma *integrable-on-refl-ivl*[*intro*, *simp*]: *g integrable-on* {*b* .. (*b*::'*b*::*ordered-euclidean-space*)}
and *integrable-on-refl-closed-segment*[*intro*, *simp*]: *h integrable-on closed-segment a a*
using *integrable-on-refl* **by** *auto*

lemma *integrable-const-ivl-closed-segment*[*intro*, *simp*]: (λ*x*. *c*) *integrable-on closed-segment a* (*b*::*real*)
by (*auto simp*: *closed-segment-eq-real-ivl*)

lemma *integrable-ident-ivl*[*intro*, *simp*]: (λ*x*. *x*) *integrable-on closed-segment a* (*b*::*real*)
and *integrable-ident-cbox*[*intro*, *simp*]: (λ*x*. *x*) *integrable-on cbox a* (*b*::*real*)
by (*auto simp*: *closed-segment-eq-real-ivl ident-integrable-on*)

lemma *content-closed-segment-real*:
fixes *a b*::*real*
shows *content* (*closed-segment a b*) = *abs* (*b* − *a*)
by (*auto simp*: *closed-segment-eq-real-ivl*)

lemma *integral-const-closed-segment*:
fixes *a b*::*real*
shows *integral* (*closed-segment a b*) (λ*x*. *c*) = *abs* (*b* − *a*) $*_R$ *c*
by (*auto simp*: *closed-segment-eq-real-ivl content-closed-segment-real*)

lemmas [*integrable-on-simps*] =
integrable-on-empty — empty
integrable-on-refl integrable-on-refl-ivl integrable-on-refl-closed-segment — singleton
integrable-const integrable-const-ivl integrable-const-ivl-closed-segment — constant
ident-integrable-on integrable-ident-ivl integrable-ident-cbox — identity

lemma *integrable-cmul-real*:
fixes *K*::*real*
shows *f integrable-on X* ⟹ (λ*x*. *K* * *f x*) *integrable-on X*
unfolding *real-scaleR-def*[*symmetric*]
by (*rule integrable-cmul*)

lemmas [*integrable-on-simps*] =
integrable-0
integrable-neg
integrable-cmul
integrable-cmul-real
integrable-on-cmult-iff
integrable-on-cmult-left
integrable-on-cmult-right

integrable-on-cmult-iff
integrable-on-cmult-left-iff
integrable-on-cmult-right-iff
integrable-on-cdivide-iff
integrable-diff
integrable-add
integrable-sum

lemma *dist-cancel-add1*: $dist\ (t0 + et)\ t0 = norm\ et$
by (*simp add*: *dist-norm*)

lemma *double-nonneg-le*:
fixes *a*::*real*
shows $a * 2 \le b \Longrightarrow a \ge 0 \Longrightarrow a \le b$
by *arith*

5.2 Nonautonomous IVP on maximal existence interval

context *ll-on-open-it*
begin

context
fixes *x0*
assumes *iv-defined*: $t0 \in T\ x0 \in X$
begin

lemmas *closed-segment-iv-subset-domain* = *closed-segment-subset-domainI*[*OF iv-defined*(*1*)]

lemma
local-unique-solutions:
obtains *t u L*
where
$0 < t\ 0 < u$
$cball\ t0\ t \subseteq existence\text{-}ivl\ t0\ x0$
$cball\ x0\ (2 * u) \subseteq X$
$\bigwedge t'.\ t' \in cball\ t0\ t \Longrightarrow L{-}lipschitz\text{-}on\ (cball\ x0\ (2 * u))\ (f\ t')$
$\bigwedge x.\ x \in cball\ x0\ u \Longrightarrow (flow\ t0\ x\ usolves\text{-}ode\ f\ from\ t0)\ (cball\ t0\ t)\ (cball\ x\ u)$
$\bigwedge x.\ x \in cball\ x0\ u \Longrightarrow cball\ x\ u \subseteq X$
proof −
from *local-unique-solution*[*OF iv-defined*] **obtain** *et ex B L*
where $0 < et\ 0 < ex\ cball\ t0\ et \subseteq T\ cball\ x0\ ex \subseteq X$
unique-on-cylinder t0 (*cball t0 et*) *x0 ex f B L*
by *metis*
then interpret *cyl*: *unique-on-cylinder t0 cball t0 et x0 ex cball x0 ex f B L*
by *auto*

from *cyl.solution-solves-ode order-refl* ‹$cball\ x0\ ex \subseteq X$›
have (*cyl.solution solves-ode f*) (*cball t0 et*) *X*
by (*rule solves-ode-on-subset*)

then have *cball t0 et* ⊆ *existence-ivl t0 x0*
by (*rule existence-ivl-maximal-interval*) (*insert* ‹*cball t0 et* ⊆ *T*› ‹*0* < *et*›, *auto*)

have *cball t0 et* = {*t0* − *et* .. *t0* + *et*}
using ‹*et* > *0*› **by** (*auto simp*: *dist-real-def*)
then have *cylbounds*[*simp*]: *cyl.tmin* = *t0* − *et* *cyl.tmax* = *t0* + *et*
unfolding *cyl.tmin-def cyl.tmax-def*
using ‹*0* < *et*›
by *auto*

define *et'* **where** *et'* ≡ *et* / *2*
define *ex'* **where** *ex'* ≡ *ex* / *2*

have *et'* > *0* *ex'* > *0* **using** ‹*0* < *et*› ‹*0* < *ex*› **by** (*auto simp*: *et'-def ex'-def*)
moreover
from ‹*cball t0 et* ⊆ *existence-ivl t0 x0*› **have** *cball t0 et'* ⊆ *existence-ivl t0 x0*
by (*force simp*: *et'-def dest*!: *double-nonneg-le*)
moreover
from *this* **have** *cball t0 et'* ⊆ *T* **using** *existence-ivl-subset*[*of x0*] **by** *simp*
have *cball x0* (*2* ∗ *ex'*) ⊆ *X* ⋀*t'*. *t'* ∈ *cball t0 et'* ⟹ *L*−*lipschitz-on* (*cball x0* (*2* ∗ *ex'*)) (*f t'*)
using *cyl.lipschitz* ‹*0* < *et*› ‹*cball x0 ex* ⊆ *X*›
by (*auto simp*: *ex'-def et'-def intro*!:)
moreover
{
fix *x0'*::*'a*
assume *x0'*: *x0'* ∈ *cball x0 ex'*
{
fix *b*
assume *d*: *dist x0' b* ≤ *ex'*
have *dist x0 b* ≤ *dist x0 x0'* + *dist x0' b*
by (*rule dist-triangle*)
also have . . . ≤ *ex'* + *ex'*
using *x0' d* **by** *simp*
also have . . . ≤ *ex* **by** (*simp add*: *ex'-def*)
finally have *dist x0 b* ≤ *ex* **.**
} **note** *triangle* = *this*
have *subs1*: *cball t0 et'* ⊆ *cball t0 et*
and *subs2*: *cball x0' ex'* ⊆ *cball x0 ex*
and *subs*: *cball t0 et'* × *cball x0' ex'* ⊆ *cball t0 et* × *cball x0 ex*
using ‹*0* < *ex*› ‹*0* < *et*› *x0'*
by (*auto simp*: *ex'-def et'-def triangle dest*!: *double-nonneg-le*)

have *subset-X*: *cball x0' ex'* ⊆ *X*
using ‹*cball x0 ex* ⊆ *X*› *subs2* ‹*0* < *ex'*› **by** *force*
then have *x0'* ∈ *X* **using** ‹*0* < *ex'*› **by** *force*
have *x0'*: *t0* ∈ *T* *x0'* ∈ *X* **by** *fact*+
have *half-intros*: *a* ≤ *ex'* ⟹ *a* ≤ *ex* *a* ≤ *et'* ⟹ *a* ≤ *et*
and *halfdiv-intro*: *a* ∗ *2* ≤ *ex* / *B* ⟹ *a* ≤ *ex'* / *B* **for** *a*

using ‹*0 < ex*› ‹*0 < et*›
by (*auto simp*: *ex′-def et′-def*)

interpret *cyl′*: *solution-in-cylinder t0 cball t0 et′ x0′ ex′ f cball x0′ ex′ B*
using ‹*0 < et′*› ‹*0 < ex′*› ‹*0 < et*› *cyl.norm-f cyl.continuous subs1* ‹*cball t0 et ⊆ T*›
apply *unfold-locales*
apply (*auto simp*: *split-beta′ dist-cancel-add1 intro*!: *triangle continuous-intros cyl.norm-f order-trans*[*OF - cyl.e-bounded*] *halfdiv-intro*)
by (*simp add*: *ex′-def et′-def dist-commute*)

interpret *cyl′*: *unique-on-cylinder t0 cball t0 et′ x0′ ex′ cball x0′ ex′ f B L*
using *cyl.lipschitz*[*simplified*] *subs subs1*
by (*unfold-locales*)
(*auto simp*: *triangle intro*!: *half-intros lipschitz-on-subset*[*OF - subs2*])
from *cyl′.solution-usolves-ode*
have (*flow t0 x0′ usolves-ode f from t0*) (*cball t0 et′*) (*cball x0′ ex′*)
apply (*rule usolves-ode-solves-odeI*)
subgoal
apply (*rule cyl′.solves-ode-on-subset-domain*[**where** *Y=X*])
subgoal
apply (*rule solves-ode-on-subset*[**where** *S=existence-ivl t0 x0′* **and** *Y=X*])
subgoal by (*rule flow-solves-ode*[*OF x0′*])
subgoal
using *subs2* ‹*cball x0 ex ⊆ X*› ‹*0 < et′*› ‹*cball t0 et′ ⊆ T*›
by (*intro existence-ivl-maximal-interval*[*OF solves-ode-on-subset*[*OF cyl′.solution-solves-ode*]])
auto
subgoal by *force*
done
subgoal by (*force simp*: ‹*x0′ ∈ X*› *iv-defined*)
subgoal using ‹*0 < et′*› **by** *force*
subgoal by *force*
subgoal by *force*
done
subgoal by (*force simp*: ‹*x0′ ∈ X*› *iv-defined cyl′.solution-iv*)
done
note *this subset-X*
} **ultimately show** *thesis* **..**
qed

lemma *Picard-iterate-mem-existence-ivlI*:
assumes *t ∈ T*
assumes *compact C x0 ∈ C C ⊆ X*
assumes ⋀*y s. s ∈ {t0 −− t} ⟹ y t0 = x0 ⟹ y ∈ {t0−−s} → C ⟹ continuous-on {t0−−s} y ⟹*
x0 + ivl-integral t0 s (λt. f t (y t)) ∈ C
shows *t ∈ existence-ivl t0 x0* ⋀*s. s ∈ {t0 −− t} ⟹ flow t0 x0 s ∈ C*
proof −

have $\{t0 \;--\; t\} \subseteq T$
by (*intro closed-segment-subset-domain iv-defined assms*)
from *lipschitz-on-compact*[*OF compact-segment* ‹$\{t0 \;--\; t\} \subseteq T$› ‹*compact C*› ‹$C \subseteq X$›]
obtain *L* **where** *L*: $\bigwedge s.\ s \in \{t0 \;--\; t\} \Longrightarrow L{-}lipschitz\text{-}on\ C\ (f\ s)$ **by** *metis*
interpret *uc*: *unique-on-closed t0* $\{t0 \;--\; t\}$ *x0 f C L*
using *assms closed-segment-iv-subset-domain*
by *unfold-locales*
(*auto intro*!: *L compact-imp-closed* ‹*compact C*› *continuous-on-f continuous-intros*
simp: *split-beta*)
have $\{t0 \;--\; t\} \subseteq$ *existence-ivl t0 x0*
using *assms closed-segment-iv-subset-domain*
by (*intro maximal-existence-flow*[*OF solves-ode-on-subset*[*OF uc.solution-solves-ode*]])
auto
thus $t \in$ *existence-ivl t0 x0*
using *assms* **by** *auto*
show *flow t0 x0 s* $\in C$ **if** $s \in \{t0 \;--\; t\}$ **for** *s*
proof −
have *flow t0 x0 s* = *uc.solution s uc.solution s* $\in C$
using *solves-odeD*[*OF uc.solution-solves-ode*] *that assms*
by (*auto simp*: *closed-segment-iv-subset-domain*
intro!: *maximal-existence-flowI*(*2*)[**where** *K*=$\{t0 \;--\; t\}$])
thus *?thesis* **by** *simp*
qed
qed

lemma *flow-has-vderiv-on*: (*flow t0 x0 has-vderiv-on* ($\lambda t.\ f\ t$ (*flow t0 x0 t*))) (*existence-ivl t0 x0*)
by (*rule solves-ode-vderivD*[*OF flow-solves-ode*[*OF iv-defined*]])

lemmas *flow-has-vderiv-on-compose*[*derivative-intros*] =
has-vderiv-on-compose2[*OF flow-has-vderiv-on*, *THEN has-vderiv-on-eq-rhs*]

end

lemma *unique-on-intersection*:
assumes *sols*: (*x solves-ode f*) *U X* (*y solves-ode f*) *V X*
assumes *iv-mem*: $t0 \in U\ t0 \in V$ **and** *subs*: $U \subseteq T\ V \subseteq T$
assumes *ivls*: *is-interval U is-interval V*
assumes *iv*: *x t0* = *y t0*
assumes *mem*: $t \in U\ t \in V$
shows *x t* = *y t*
proof −
from
maximal-existence-flow(*2*)[*OF sols*(*1*) *refl* *ivls*(*1*) *iv-mem*(*1*) *subs*(*1*) *mem*(*1*)]
maximal-existence-flow(*2*)[*OF sols*(*2*) *iv*[*symmetric*] *ivls*(*2*) *iv-mem*(*2*) *subs*(*2*) *mem*(*2*)]

show *?thesis* **by** *simp*
qed

lemma *unique-solution*:
assumes *sols*: (*x solves-ode f*) *U X* (*y solves-ode f*) *U X*
assumes *iv-mem*: *t0* ∈ *U* **and** *subs*: *U* ⊆ *T*
assumes *ivls*: *is-interval U*
assumes *iv*: *x t0* = *y t0*
assumes *mem*: *t* ∈ *U*
shows *x t* = *y t*
by (*metis unique-on-intersection assms*)

lemma
assumes *s*: *s* ∈ *existence-ivl t0 x0*
assumes *t*: *t* + *s* ∈ *existence-ivl s* (*flow t0 x0 s*)
shows *flow-trans*: *flow t0 x0* (*s* + *t*) = *flow s* (*flow t0 x0 s*) (*s* + *t*)
and *existence-ivl-trans*: *s* + *t* ∈ *existence-ivl t0 x0*
proof −
note *ll-on-open-it-axioms*
moreover
from *ll-on-open-it-axioms*
have *iv-defined*: *t0* ∈ *T x0* ∈ *X*
and *iv-defined′*: *s* ∈ *T flow t0 x0 s* ∈ *X*
using *ll-on-open-it.mem-existence-ivl-iv-defined s t*
by *blast+*

have {*t0*−−*s*} ⊆ *existence-ivl t0 x0*
by (*simp add*: *s segment-subset-existence-ivl iv-defined*)

have *s* ∈ *existence-ivl s* (*flow t0 x0 s*)
by (*rule ll-on-open-it.existence-ivl-initial-time*; *fact*)
have {*s*−−*t* + *s*} ⊆ *existence-ivl s* (*flow t0 x0 s*)
by (*rule ll-on-open-it.segment-subset-existence-ivl*; *fact*)

have *unique*: *flow t0 x0 u* = *flow s* (*flow t0 x0 s*) *u*
if *u* ∈ {*s*−−*t* + *s*} *u* ∈ {*t0*−−*s*} **for** *u*
using
ll-on-open-it-axioms
ll-on-open-it.flow-solves-ode[*OF ll-on-open-it-axioms iv-defined*]
ll-on-open-it.flow-solves-ode[*OF ll-on-open-it-axioms iv-defined′*]
s
apply (*rule ll-on-open-it.unique-on-intersection*)
using ‹*s* ∈ *existence-ivl s* (*flow t0 x0 s*)› *existence-ivl-subset*
‹*flow t0 x0 s* ∈ *X*› ‹*s* ∈ *T*› *iv-defined s t ll-on-open-it.in-existence-between-zeroI*
that ll-on-open-it-axioms ll-on-open-it.mem-existence-ivl-subset
by (*auto simp*: *is-interval-existence-ivl*)

let *?un* = {*t0* −− *s*} ∪ {*s* −− *t* + *s*}
let *?if* = λ*t*. *if t* ∈ {*t0* −− *s*} *then flow t0 x0 t else flow s* (*flow t0 x0 s*) *t*

have (*?if solves-ode* (λt. *if* $t \in \{t0 \ {-}{-}\ s\}$ *then f t else f t*)) *?un* $(X \cup X)$
apply (*rule connection-solves-ode*)
subgoal by (*rule solves-ode-on-subset*[*OF flow-solves-ode*[*OF iv-defined*] ‹$\{t0{-}{-}s\}$ $\subseteq$ -› *order-refl*])
subgoal
by (*rule solves-ode-on-subset*[*OF ll-on-open-it.flow-solves-ode*[*OF ll-on-open-it-axioms iv-defined'*]
‹$\{s{-}{-}t + s\} \subseteq$ -› *order-refl*])
subgoal by *simp*
subgoal by *simp*
subgoal by (*rule unique*) *auto*
subgoal by *simp*
done
then have *ifsol*: (*?if solves-ode f*) *?un X*
by *simp*
moreover
have *?un* $\subseteq$ *existence-ivl t0 x0*
using *existence-ivl-subset*[*of x0*]
ll-on-open-it.existence-ivl-subset[*OF ll-on-open-it-axioms*, *of s flow t0 x0 s*]
‹$\{t0 \ {-}{-}\ s\} \subseteq$ -› ‹$\{s{-}{-}t + s\} \subseteq$ -›
by (*intro existence-ivl-maximal-interval*[*OF ifsol*]) (*auto intro*!: *is-real-interval-union*)
then show $s + t \in$ *existence-ivl t0 x0*
by (*auto simp*: *ac-simps*)
have (*flow t0 x0 solves-ode f*) *?un X*
using ‹$\{t0{-}{-}s\} \subseteq$ -› ‹$\{s \ {-}{-}\ t + s\} \subseteq$ -›
by (*intro solves-ode-on-subset*[*OF flow-solves-ode* ‹*?un* $\subseteq$ -› *order-refl*] *iv-defined*)
moreover have $s \in$ *?un*
by *simp*
ultimately have *?if* $(s + t)$ = *flow t0 x0* $(s + t)$
apply (*rule ll-on-open-it.unique-solution*)
using *existence-ivl-subset*[*of x0*]
ll-on-open-it.existence-ivl-subset[*OF ll-on-open-it-axioms*, *of s flow t0 x0 s*]
‹$\{t0 \ {-}{-}\ s\} \subseteq$ -› ‹$\{s{-}{-}t + s\} \subseteq$ -›
by (*auto intro*!: *is-real-interval-union simp*: *ac-simps*)
with *unique*[*of* $s + t$]
show *flow t0 x0* $(s + t)$ = *flow s* (*flow t0 x0 s*) $(s + t)$
by (*auto split*: *if-splits simp*: *ac-simps*)
qed

lemma
assumes *t*: $t \in$ *existence-ivl t0 x0*
shows *flows-reverse*: *flow t* (*flow t0 x0 t*) *t0* = *x0*
and *existence-ivl-reverse*: *t0* $\in$ *existence-ivl t* (*flow t0 x0 t*)
proof −
have *iv-defined*: *t0* $\in$ *T x0* $\in$ *X*
using *mem-existence-ivl-iv-defined t* **by** *blast+*
show *t0* $\in$ *existence-ivl t* (*flow t0 x0 t*)
using *assms*
by (*metis* (*no-types*, *opaque-lifting*) *closed-segment-commute closed-segment-subset-interval*

ends-in-segment(*2*) *general.csol*(*2*−*4*)
general.existence-ivl-maximal-segment general.is-interval-existence-ivl
is-interval-closed-segment-1 iv-defined ll-on-open-it.equals-flowI
local.existence-ivl-initial-time local.flow-initial-time local.ll-on-open-it-axioms)
then have *flow t* (*flow t0 x0 t*) (*t* + (*t0* − *t*)) = *flow t0 x0* (*t* + (*t0* − *t*))
by (*intro flow-trans*[*symmetric*]) (*auto simp*: *t iv-defined*)
then show *flow t* (*flow t0 x0 t*) *t0* = *x0*
by (*simp add*: *iv-defined*)
qed

lemma *flow-has-derivative*:
assumes *t* ∈ *existence-ivl t0 x0*
shows (*flow t0 x0 has-derivative* (λ*i*. *i* $*_R$ *f t* (*flow t0 x0 t*))) (*at t*)
proof −
have (*flow t0 x0 has-derivative* (λ*i*. *i* $*_R$ *f t* (*flow t0 x0 t*))) (*at t within existence-ivl t0 x0*)
using *flow-has-vderiv-on*
by (*auto simp*: *has-vderiv-on-def has-vector-derivative-def assms mem-existence-ivl-iv-defined*[*OF assms*])
then show *?thesis*
by (*simp add*: *at-within-open*[*OF assms open-existence-ivl*])
qed

lemma *flow-has-vector-derivative*:
assumes *t* ∈ *existence-ivl t0 x0*
shows (*flow t0 x0 has-vector-derivative f t* (*flow t0 x0 t*)) (*at t*)
using *flow-has-derivative*[*OF assms*]
by (*simp add*: *has-vector-derivative-def*)

lemma *flow-has-vector-derivative-at-0*:
assumes*t* ∈ *existence-ivl t0 x0*
shows ((λ*h*. *flow t0 x0* (*t* + *h*)) *has-vector-derivative f t* (*flow t0 x0 t*)) (*at 0*)
proof −
from *flow-has-vector-derivative*[*OF assms*]
have
((+) *t has-vector-derivative 1*) (*at 0*)
(*flow t0 x0 has-vector-derivative f t* (*flow t0 x0 t*)) (*at* (*t* + *0*))
by (*auto intro*!: *derivative-eq-intros*)
from *vector-diff-chain-at*[*OF this*]
show *?thesis* **by** (*simp add*: *o-def*)
qed

lemma
assumes *t* ∈ *existence-ivl t0 x0*
shows *closed-segment-subset-existence-ivl*: *closed-segment t0 t* ⊆ *existence-ivl t0 x0*
and *ivl-subset-existence-ivl*: {*t0* .. *t*} ⊆ *existence-ivl t0 x0*
and *ivl-subset-existence-ivl'*: {*t* .. *t0*} ⊆ *existence-ivl t0 x0*

using *assms in-existence-between-zeroI*
by (*auto simp*: *closed-segment-eq-real-ivl*)

lemma *flow-fixed-point*:
assumes *t*: *t* ∈ *existence-ivl t0 x0*
shows *flow t0 x0 t* = *x0* + *ivl-integral t0 t* (λ*t*. *f t* (*flow t0 x0 t*))
proof −
have (*flow t0 x0 has-vderiv-on* (λ*s*. *f s* (*flow t0 x0 s*))) {*t0* −− *t*}
using *closed-segment-subset-existence-ivl*[*OF t*]
by (*auto intro*!: *has-vector-derivative-at-within flow-has-vector-derivative*
simp: *has-vderiv-on-def*)
from *fundamental-theorem-of-calculus-ivl-integral*[*OF this*]
have ((λ*t*. *f t* (*flow t0 x0 t*)) *has-ivl-integral flow t0 x0 t* − *x0*) *t0 t*
by (*simp add*: *mem-existence-ivl-iv-defined*[*OF assms*])
from *this*[*THEN ivl-integral-unique*]
show *?thesis* **by** *simp*
qed

lemma *flow-continuous*: *t* ∈ *existence-ivl t0 x0* ⟹ *continuous* (*at t*) (*flow t0 x0*)
by (*metis has-derivative-continuous flow-has-derivative*)

lemma *flow-tendsto*: *t* ∈ *existence-ivl t0 x0* ⟹ (*ts* ⟶ *t*) *F* ⟹
((λ*s*. *flow t0 x0* (*ts s*)) ⟶ *flow t0 x0 t*) *F*
by (*rule isCont-tendsto-compose*[*OF flow-continuous*])

lemma *flow-continuous-on*: *continuous-on* (*existence-ivl t0 x0*) (*flow t0 x0*)
by (*auto intro*!: *flow-continuous continuous-at-imp-continuous-on*)

lemma *flow-continuous-on-intro*:
continuous-on s g ⟹
(⋀*xa*. *xa* ∈ *s* ⟹ *g xa* ∈ *existence-ivl t0 x0*) ⟹
continuous-on s (λ*xa*. *flow t0 x0* (*g xa*))
by (*auto intro*!: *continuous-on-compose2*[*OF flow-continuous-on*])

lemma *f-flow-continuous*:
assumes *t* ∈ *existence-ivl t0 x0*
shows *isCont* (λ*t*. *f t* (*flow t0 x0 t*)) *t*
by (*rule continuous-on-interior*)
(*insert existence-ivl-subset assms*,
auto intro!: *flow-in-domain flow-continuous-on continuous-intros*
simp: *interior-open open-existence-ivl*)

lemma *exponential-initial-condition*:
assumes *y0*: *t* ∈ *existence-ivl t0 y0*
assumes *z0*: *t* ∈ *existence-ivl t0 z0*
assumes *Y* ⊆ *X*
assumes *remain*: ⋀*s*. *s* ∈ *closed-segment t0 t* ⟹ *flow t0 y0 s* ∈ *Y*
⋀*s*. *s* ∈ *closed-segment t0 t* ⟹ *flow t0 z0 s* ∈ *Y*
assumes *lipschitz*: ⋀*s*. *s* ∈ *closed-segment t0 t* ⟹ *K*−*lipschitz-on Y* (*f s*)

```
  shows norm (flow t0 y0 t − flow t0 z0 t) ≤ norm (y0 − z0) ∗ exp ((K + 1) ∗
abs (t − t0))
proof cases
  assume y0 = z0
  thus ?thesis
    by simp
next
  assume ne: y0 ≠ z0
  define K' where K' ≡ K + 1
  from lipschitz have K'−lipschitz-on Y (f s) if s ∈ {t0 −− t} for s
    using that
    by (auto simp: lipschitz-on-def K'-def
      intro!: order-trans[OF - mult-right-mono[of K K + 1]])

  from mem-existence-ivl-iv-defined[OF y0] mem-existence-ivl-iv-defined[OF z0]
  have t0 ∈ T and inX: y0 ∈ X z0 ∈ X by auto

  from remain[of t0] inX ‹t0 ∈ T › have y0 ∈ Y z0 ∈ Y by auto

  define v where v ≡ λt. norm (flow t0 y0 t − flow t0 z0 t)
  {
    fix s
    assume s: s ∈ {t0 −− t}
    with s
      closed-segment-subset-existence-ivl[OF y0]
      closed-segment-subset-existence-ivl[OF z0]
    have
      y0': s ∈ existence-ivl t0 y0 and
      z0': s ∈ existence-ivl t0 z0
      by (auto simp: closed-segment-eq-real-ivl)
    have integrable:
      (λt. f t (flow t0 y0 t)) integrable-on {t0−−s}
      (λt. f t (flow t0 z0 t)) integrable-on {t0−−s}
      using closed-segment-subset-existence-ivl[OF y0']
        closed-segment-subset-existence-ivl[OF z0']
        ‹y0 ∈ X› ‹z0 ∈ X› ‹t0 ∈ T›
      by (auto intro!: continuous-at-imp-continuous-on f-flow-continuous
        integrable-continuous-closed-segment)
    hence int: flow t0 y0 s − flow t0 z0 s =
        y0 − z0 + ivl-integral t0 s (λt. f t (flow t0 y0 t) − f t (flow t0 z0 t))
      unfolding v-def
      using flow-fixed-point[OF y0'] flow-fixed-point[OF z0']
        s
      by (auto simp: algebra-simps ivl-integral-diff)
    have v s ≤ v t0 + K' ∗  integral {t0 −− s} (λt. v t)
     using closed-segment-subset-existence-ivl[OF y0'] closed-segment-subset-existence-ivl[OF
z0'] s
        using closed-segment-closed-segment-subset[OF - - s, of - t0, simplified]
      by (subst integral-mult)
```

```
      (auto simp: integral-mult v-def int inX ‹t0 ∈ T›
        simp del: Henstock-Kurzweil-Integration.integral-mult-right
        intro!: norm-triangle-le ivl-integral-norm-bound-integral
          integrable-continuous-closed-segment continuous-intros
          continuous-at-imp-continuous-on flow-continuous f-flow-continuous
          lipschitz-on-normD[OF ‹- ⟹ K′−lipschitz-on - -›] remain)
  } note le = this
  have cont: continuous-on {t0 −− t} v
   using closed-segment-subset-existence-ivl[OF y0] closed-segment-subset-existence-ivl[OF
z0] inX
    by (auto simp: v-def ‹t0 ∈ T›
      intro!: continuous-at-imp-continuous-on continuous-intros flow-continuous)
  have nonneg: ⋀t. v t ≥ 0
    by (auto simp: v-def)
  from ne have pos: v t0 > 0
    by (auto simp: v-def ‹t0 ∈ T› inX)
  have lippos: K′ > 0
  proof −
    have 0 ≤ dist (f t0 y0) (f t0 z0) by simp
    also from lipschitz-onD[OF lipschitz ‹y0 ∈ Y› ‹z0 ∈ Y›, of t0]ne
    have ... ≤ K ∗ dist y0 z0
      by simp
    finally have 0 ≤ K
      by (metis dist-le-zero-iff ne zero-le-mult-iff)
    thus ?thesis by (simp add: K′-def)
  qed
  from le cont nonneg pos ‹0 < K′›
  have v t ≤ v t0 ∗ exp (K′ ∗ abs (t − t0))
    by (rule gronwall-general-segment) simp-all
  thus ?thesis
    by (simp add: v-def K′-def ‹t0 ∈ T› inX)
qed

lemma
  existence-ivl-cballs:
  assumes iv-defined: t0 ∈ T x0 ∈ X
  obtains t u L
  where
    ⋀y. y ∈ cball x0 u ⟹ cball t0 t ⊆ existence-ivl t0 y
    ⋀s y. y ∈ cball x0 u ⟹ s ∈ cball t0 t ⟹ flow t0 y s ∈ cball y u
    L−lipschitz-on (cball t0 t×cball x0 u) (λ(t, x). flow t0 x t)
    ⋀y. y ∈ cball x0 u ⟹ cball y u ⊆ X
    0 < t 0 < u
proof −
  note iv-defined
  from local-unique-solutions[OF this]
  obtain t u L where tu: 0 < t 0 < u
    and subsT: cball t0 t ⊆ existence-ivl t0 x0
    and subs′: cball x0 (2 ∗ u) ⊆ X
```

and *lipschitz*: $\bigwedge s.\ s \in cball\ t0\ t \Longrightarrow L{-}lipschitz\text{-}on\ (cball\ x0\ (2{*}u))\ (f\ s)$
and *usol*: $\bigwedge y.\ y \in cball\ x0\ u \Longrightarrow (flow\ t0\ y\ usolves\text{-}ode\ f\ from\ t0)\ (cball\ t0\ t)\ (cball\ y\ u)$
and *subs*: $\bigwedge y.\ y \in cball\ x0\ u \Longrightarrow cball\ y\ u \subseteq X$
by *metis*
{
fix *y* **assume** *y*: *y* ∈ *cball x0 u*
from *subs*[*OF y*] ‹*0* < *u*› **have** *y* ∈ *X* **by** *auto*
note *iv′* = ‹*t0* ∈ *T*› ‹*y* ∈ *X*›
from *usol*[*OF y*, *THEN usolves-odeD*(*1*)]
have *sol1*: (*flow t0 y solves-ode f*) (*cball t0 t*) (*cball y u*) **.**
from *sol1 order-refl subs*[*OF y*]
have *sol*: (*flow t0 y solves-ode f*) (*cball t0 t*) *X*
by (*rule solves-ode-on-subset*)
note ∗ = *maximal-existence-flow*[*OF sol flow-initial-time*
is-interval-cball-1 - *order-trans*[*OF subsT existence-ivl-subset*],
unfolded centre-in-cball, *OF iv′ less-imp-le*[*OF* ‹*0* < *t*›]]
have *eivl*: *cball t0 t* ⊆ *existence-ivl t0 y*
by (*rule* ∗)
have *flow t0 y s* ∈ *cball y u* **if** *s* ∈ *cball t0 t* **for** *s*
by (*rule solves-odeD*(*2*)[*OF sol1 that*])
note *eivl this*
} **note** ∗ = *this*
note ∗
moreover
have *cont-on-f-flow*:
$\bigwedge x1\ S.\ S \subseteq cball\ t0\ t \Longrightarrow x1 \in cball\ x0\ u \Longrightarrow continuous\text{-}on\ S\ (\lambda t.\ f\ t\ (flow\ t0\ x1\ t))$
using *subs*[*of x0*] ‹*u* > *0*› ∗(*1*) *iv-defined*
by (*auto intro*!: *continuous-at-imp-continuous-on f-flow-continuous*)
have *bounded* ((λ(*t*, *x*). *f t x*) ‘ (*cball t0 t* × *cball x0* (*2* ∗ *u*)))
using *subs′ subsT existence-ivl-subset*[*of x0*]
by (*auto intro*!: *compact-imp-bounded compact-continuous-image compact-Times*
continuous-intros simp: *split-beta′*)
then obtain *B* **where** *B*: $\bigwedge s\ y.\ s \in cball\ t0\ t \Longrightarrow y \in cball\ x0\ (2 * u) \Longrightarrow norm\ (f\ s\ y) \le B\ B > 0$
by (*auto simp*: *bounded-pos cball-def*)
have *flow-in-cball*: *flow t0 x1 s* ∈ *cball x0* (*2* ∗ *u*)
if *s*: *s* ∈ *cball t0 t* **and** *x1*: *x1* ∈ *cball x0 u*
for *s*::*real* **and** *x1*
proof −
from ∗(*2*)[*OF x1 s*] **have** *flow t0 x1 s* ∈ *cball x1 u* **.**
also have . . . ⊆ *cball x0* (*2* ∗ *u*)
using *x1*
by (*auto intro*!: *dist-triangle-le*[*OF add-mono*, *of* - *x1 u* - *u*, *simplified*]
simp: *dist-commute*)
finally show *?thesis* **.**
qed
have $(B + exp\ ((L + 1) * |t|)){-}lipschitz\text{-}on\ (cball\ t0\ t{\times}cball\ x0\ u)\ (\lambda(t, x).\ flow$

```
t0 x t)
  proof (rule lipschitz-onI, safe)
    fix t1 t2 :: real and x1 x2
    assume t1: t1 ∈ cball t0 t and t2: t2 ∈ cball t0 t
      and x1: x1 ∈ cball x0 u and x2: x2 ∈ cball x0 u
    have t1-ex: t1 ∈ existence-ivl t0 x1
      and t2-ex: t2 ∈ existence-ivl t0 x1 t2 ∈ existence-ivl t0 x2
      and x1 ∈ cball x0 (2*u) x2 ∈ cball x0 (2*u)
      using *(1)[OF x1] *(1)[OF x2] t1 t2 x1 x2 tu by auto
    have dist (flow t0 x1 t1) (flow t0 x2 t2) ≤
      dist (flow t0 x1 t1) (flow t0 x1 t2) + dist (flow t0 x1 t2) (flow t0 x2 t2)
      by (rule dist-triangle)
    also have dist (flow t0 x1 t2) (flow t0 x2 t2) ≤ dist x1 x2 * exp ((L + 1) *
|t2 − t0|)
      unfolding dist-norm
    proof (rule exponential-initial-condition[where Y = cball x0 (2 * u)])
      fix s assume s ∈ closed-segment t0 t2 hence s: s ∈ cball t0 t
        using t2
        by (auto simp: dist-real-def closed-segment-eq-real-ivl split: if-split-asm)
      show flow t0 x1 s ∈ cball x0 (2 * u)
        by (rule flow-in-cball[OF s x1])
      show flow t0 x2 s ∈ cball x0 (2 * u)
        by (rule flow-in-cball[OF s x2])
      show L−lipschitz-on (cball x0 (2 * u)) (f s) if s ∈ closed-segment t0 t2 for s
        using that centre-in-cball convex-contains-segment less-imp-le t2 tu(1)
        by (blast intro!: lipschitz)
    qed (fact)+
    also have ... ≤ dist x1 x2 * exp ((L + 1) * |t|)
      using ‹u > 0› t2
      by (auto
      intro!: mult-left-mono add-nonneg-nonneg lipschitz[THEN lipschitz-on-nonneg]
        simp: cball-eq-empty cball-eq-sing' dist-real-def)
    also
    have x1 ∈ X
      using x1 subs[of x0] ‹u > 0›
      by auto
    have *: |t0 − t1| ≤ t ⟹ x ∈ {t0−−t1} ⟹ |t0 − x| ≤ t
      |t0 − t2| ≤ t ⟹ x ∈ {t0−−t2} ⟹ |t0 − x| ≤ t
      |t0 − t1| ≤ t ⟹ |t0 − t2| ≤ t ⟹ x ∈ {t1−−t2} ⟹ |t0 − x| ≤ t
      for x
      using t1 t2 t1-ex x1 flow-in-cball[OF - x1]
      by (auto simp: closed-segment-eq-real-ivl split: if-splits)

    have integrable:
      (λt. f t (flow t0 x1 t)) integrable-on {t0−−t1}
      (λt. f t (flow t0 x1 t)) integrable-on {t0−−t2}
      (λt. f t (flow t0 x1 t)) integrable-on {t1−−t2}
      using t1 t2 t1-ex x1 flow-in-cball[OF - x1]
     by (auto intro!: order-trans[OF integral-bound[where B=B]] cont-on-f-flow B
```

integrable-continuous-closed-segment
intro: ∗
simp: *dist-real-def integral-minus-sets′*)

have ∗: $|t0 - t1| \leq t \Longrightarrow |t0 - t2| \leq t \Longrightarrow s \in \{t1{-}{-}t2\} \Longrightarrow |t0 - s| \leq t$ **for** *s*
by (*auto simp*: *closed-segment-eq-real-ivl split*: *if-splits*)
note [*simp*] = *t1-ex t2-ex* ‹*x1* ∈ *X*› *integrable*
have *dist* (*flow t0 x1 t1*) (*flow t0 x1 t2*) ≤ *dist t1 t2* ∗ *B*
using *t1 t2 x1 flow-in-cball*[*OF - x1*] ‹*t0* ∈ *T*›
ivl-integral-combine[*of* λ*t*. *f t* (*flow t0 x1 t*) *t2 t0 t1*]
ivl-integral-combine[*of* λ*t*. *f t* (*flow t0 x1 t*) *t1 t0 t2*]
by (*auto simp*: *flow-fixed-point dist-norm add.commute closed-segment-commute norm-minus-commute ivl-integral-minus-sets′ ivl-integral-minus-sets*
intro!: *order-trans*[*OF ivl-integral-bound*[**where** *B*=*B*]] *cont-on-f-flow B dest*: ∗)
finally
have *dist* (*flow t0 x1 t1*) (*flow t0 x2 t2*) ≤
dist t1 t2 ∗ *B* + *dist x1 x2* ∗ *exp* ((*L* + *1*) ∗ |*t*|)
by *arith*
also have . . . ≤ *dist* (*t1*, *x1*) (*t2*, *x2*) ∗ *B* + *dist* (*t1*, *x1*) (*t2*, *x2*) ∗ *exp* ((*L* + *1*) ∗ |*t*|)
using ‹*B* > *0*›
by (*auto intro*!: *add-mono mult-right-mono simp*: *dist-prod-def*)
finally show *dist* (*flow t0 x1 t1*) (*flow t0 x2 t2*)
≤ (*B* + *exp* ((*L* + *1*) ∗ |*t*|)) ∗ *dist* (*t1*, *x1*) (*t2*, *x2*)
by (*simp add*: *algebra-simps*)
qed (*simp add*: ‹*0* < *B*› *less-imp-le*)
ultimately
show *thesis* **using** *subs tu* **..**
qed

context
fixes *x0*
assumes *iv-defined*: *t0* ∈ *T x0* ∈ *X*
begin

lemma *existence-ivl-notempty*: *existence-ivl t0 x0* ≠ {}
using *existence-ivl-initial-time iv-defined*
by *auto*

lemma *initial-time-bounds*:
shows *bdd-above* (*existence-ivl t0 x0*) ⟹ *t0* < *Sup* (*existence-ivl t0 x0*) (**is** *?a* ⟹ *-*)
and *bdd-below* (*existence-ivl t0 x0*) ⟹ *Inf* (*existence-ivl t0 x0*) < *t0* (**is** *?b* ⟹ *-*)
proof −
from *local-unique-solutions*[*OF iv-defined*]
obtain *te* **where** *te*: *te* > *0 cball t0 te* ⊆ *existence-ivl t0 x0*

by *metis*
then
show *t0* < *Sup* (*existence-ivl t0 x0*) **if** *bdd*: *bdd-above* (*existence-ivl t0 x0*)
using *less-cSup-iff*[*OF existence-ivl-notempty bdd*, *of t0*] *iv-defined*
by (*auto simp*: *dist-real-def intro*!: *bexI*[**where** *x*=*t0* + *te*])

from *te* **show** *Inf* (*existence-ivl t0 x0*) < *t0* **if** *bdd*: *bdd-below* (*existence-ivl t0 x0*)
unfolding *cInf-less-iff*[*OF existence-ivl-notempty bdd*, *of t0*]
by (*auto simp*: *dist-real-def iv-defined intro*!: *bexI*[**where** *x*=*t0* − *te*])
qed

lemma
flow-leaves-compact-ivl-right:
assumes *bdd*: *bdd-above* (*existence-ivl t0 x0*)
defines *b* ≡ *Sup* (*existence-ivl t0 x0*)
assumes *b* ∈ *T*
assumes *compact K*
assumes *K* ⊆ *X*
obtains *t* **where** *t* ≥ *t0 t* ∈ *existence-ivl t0 x0 flow t0 x0 t* ∉ *K*
proof (*atomize-elim*, *rule ccontr*, *auto*)
note *iv-defined*
note *ne* = *existence-ivl-notempty*
assume *K*[*rule-format*]: ∀ *t*. *t* ∈ *existence-ivl t0 x0* ⟶ *t0* ≤ *t* ⟶ *flow t0 x0 t* ∈ *K*
have *b-upper*: *t* ≤ *b* **if** *t* ∈ *existence-ivl t0 x0* **for** *t*
unfolding *b-def*
by (*rule cSup-upper*[*OF that bdd*])

have *less-b-iff*: *y* < *b* ⟷ (∃ *x*∈*existence-ivl t0 x0*. *y* < *x*) **for** *y*
unfolding *b-def less-cSup-iff*[*OF ne bdd*] **..**
have *t0* ≤ *b*
by (*simp add*: *iv-defined b-upper*)
then have *geI*: *t* ∈ {*t0*−−<*b*} ⟹ *t0* ≤ *t* **for** *t*
by (*auto simp*: *half-open-segment-real*)
have *subset*: {*t0* −−< *b*} ⊆ *existence-ivl t0 x0*
using ‹*t0* ≤ *b*› *in-existence-between-zeroI*
by (*auto simp*: *half-open-segment-real iv-defined less-b-iff*)
have *sol*: (*flow t0 x0 solves-ode f*) {*t0* −−< *b*} *K*
apply (*rule solves-odeI*)
apply (*rule has-vderiv-on-subset*[*OF solves-odeD*(*1*)[*OF flow-solves-ode*] *subset*])
using *subset iv-defined*
by (*auto intro*!: *K geI*)
have *cont*: *continuous-on* ({*t0*−−*b*} × *K*) (λ(*t*, *x*). *f t x*)
using ‹*K* ⊆ *X*› *closed-segment-subset-domainI*[*OF iv-defined*(*1*) ‹*b* ∈ *T*›]
by (*auto simp*: *split-beta intro*!: *continuous-intros*)

from *initial-time-bounds*(*1*)[*OF bdd*] **have** *t0* ≠ *b* **by** (*simp add*: *b-def*)
from *solves-ode-half-open-segment-continuation*[*OF sol cont* ‹*compact K*› ‹*t0* ≠

b›]
obtain *l* **where** *lim*: (*flow t0 x0* ⟶ *l*) (*at b within* {*t0*−−<*b*})
and *limsol*: ((λ*t*. *if t* = *b then l else flow t0 x0 t*) *solves-ode f*) {*t0*−−*b*} *K* **.**
have *b* ∈ *existence-ivl t0 x0*
using ‹*t0* ≠ *b*› *closed-segment-subset-domainI*[*OF* ‹*t0* ∈ *T*› ‹*b* ∈ *T*›]
by (*intro existence-ivl-maximal-segment*[*OF solves-ode-on-subset*[*OF limsol order-refl* ‹*K* ⊆ *X*›]])
(*auto simp*: *iv-defined*)

have *flow t0 x0 b* ∈ *X*
by (*simp add*: ‹*b* ∈ *existence-ivl t0 x0*› *flow-in-domain iv-defined*)

from *ll-on-open-it.local-unique-solutions*[*OF ll-on-open-it-axioms* ‹*b* ∈ *T*› ‹*flow t0 x0 b* ∈ *X*›]
obtain *e* **where** *e* > *0 cball b e* ⊆ *existence-ivl b* (*flow t0 x0 b*)
by *metis*
then have *e* + *b* ∈ *existence-ivl b* (*flow t0 x0 b*)
by (*auto simp*: *dist-real-def*)
from *existence-ivl-trans*[*OF* ‹*b* ∈ *existence-ivl t0 x0*› ‹*e* + *b* ∈ *existence-ivl - -*›]
have *b* + *e* ∈ *existence-ivl t0 x0* **.**
from *b-upper*[*OF this*] ‹*e* > *0*›
show *False*
by *simp*
qed

lemma
flow-leaves-compact-ivl-left:
assumes *bdd*: *bdd-below* (*existence-ivl t0 x0*)
defines *b* ≡ *Inf* (*existence-ivl t0 x0*)
assumes *b* ∈ *T*
assumes *compact K*
assumes *K* ⊆ *X*
obtains *t* **where** *t* ≤ *t0 t* ∈ *existence-ivl t0 x0 flow t0 x0 t* ∉ *K*
proof −
interpret *rev*: *ll-on-open* (*preflect t0* ‘ *T*) (λ*t*. − *f* (*preflect t0 t*)) *X* **..**
from *antimono-preflect bdd* **have** *bdd-rev*: *bdd-above* (*rev.existence-ivl t0 x0*)
unfolding *rev-existence-ivl-eq*
by (*rule bdd-above-image-antimono*)
note *ne* = *existence-ivl-notempty*
have *Sup* (*rev.existence-ivl t0 x0*) = *preflect t0 b*
using *continuous-at-Inf-antimono*[*OF antimono-preflect - ne bdd*]
by (*simp add*: *continuous-preflect b-def rev-existence-ivl-eq*)
then have *Sup-mem*: *Sup* (*rev.existence-ivl t0 x0*) ∈ *preflect t0* ‘ *T*
using ‹*b* ∈ *T*› **by** *auto*

have *rev-iv*: *t0* ∈ *preflect t0* ‘ *T x0* ∈ *X* **using** *iv-defined* **by** *auto*
from *rev.flow-leaves-compact-ivl-right*[*OF rev-iv bdd-rev Sup-mem* ‹*compact K*› ‹*K* ⊆ *X*›]
obtain *t* **where** *t0* ≤ *t t* ∈ *rev.existence-ivl t0 x0 rev.flow t0 x0 t* ∉ *K* **.**

then have *preflect t0 t* $\leq$ *t0 preflect t0 t* $\in$ *existence-ivl t0 x0 flow t0 x0* (*preflect t0 t*) $\notin$ *K*
by (*auto simp*: *rev-existence-ivl-eq rev-flow-eq*)
thus *?thesis* **..**
qed

lemma
sup-existence-maximal:
assumes $\bigwedge t$. *t0* $\leq$ *t* $\Longrightarrow$ *t* $\in$ *existence-ivl t0 x0* $\Longrightarrow$ *flow t0 x0 t* $\in$ *K*
assumes *compact K K* $\subseteq$ *X*
assumes *bdd-above* (*existence-ivl t0 x0*)
shows *Sup* (*existence-ivl t0 x0*) $\notin$ *T*
using *flow-leaves-compact-ivl-right*[*of K*] *assms* **by** *force*

lemma
inf-existence-minimal:
assumes $\bigwedge t$. *t* $\leq$ *t0* $\Longrightarrow$ *t* $\in$ *existence-ivl t0 x0* $\Longrightarrow$ *flow t0 x0 t* $\in$ *K*
assumes *compact K K* $\subseteq$ *X*
assumes *bdd-below* (*existence-ivl t0 x0*)
shows *Inf* (*existence-ivl t0 x0*) $\notin$ *T*
using *flow-leaves-compact-ivl-left*[*of K*] *assms*
by *force*

end

lemma
subset-mem-compact-implies-subset-existence-interval:
assumes *ivl*: *t0* $\in$ *T'* *is-interval T' T'* $\subseteq$ *T*
assumes *iv-defined*: *x0* $\in$ *X*
assumes *mem-compact*: $\bigwedge t$. *t* $\in$ *T'* $\Longrightarrow$ *t* $\in$ *existence-ivl t0 x0* $\Longrightarrow$ *flow t0 x0 t* $\in$ *K*
assumes *K*: *compact K K* $\subseteq$ *X*
shows *T'* $\subseteq$ *existence-ivl t0 x0*
proof (*rule ccontr*)
assume $\neg$ *T'* $\subseteq$ *existence-ivl t0 x0*
then obtain *t'* **where** *t'*: *t'* $\notin$ *existence-ivl t0 x0 t'* $\in$ *T'*
by *auto*
from *assms* **have** *iv-defined*: *t0* $\in$ *T x0* $\in$ *X* **by** *auto*
show *False*
proof (*cases rule*: *not-in-connected-cases*[*OF connected-existence-ivl t'*(*1*) *existence-ivl-notempty*[*OF iv-defined*]])
assume *bdd*: *bdd-below* (*existence-ivl t0 x0*)
assume *t'-lower*: *t'* $\leq$ *y* **if** *y* $\in$ *existence-ivl t0 x0* **for** *y*
have *i*: *Inf* (*existence-ivl t0 x0*) $\in$ *T'*
using *initial-time-bounds*[*OF iv-defined*] *iv-defined*
apply $-$
by (*rule mem-is-intervalI*[*of - t' t0*])
(*auto simp*: *ivl t' bdd intro*!: *t'-lower cInf-greatest*[*OF existence-ivl-notempty*[*OF*

iv-defined]])
  have ∗: $t \in T'$ **if** $t \leq t0$ $t \in$ *existence-ivl t0 x0* **for** *t*
    by (*rule mem-is-intervalI*[*OF* ‹*is-interval T'*› *i* ‹*t0* ∈ *T'*›]) (*auto intro*!: *cInf-lower that bdd*)
  from *inf-existence-minimal*[*OF iv-defined mem-compact K bdd*, *OF* ∗]
  show *False* **using** *i ivl* **by** *auto*
 next
  assume *bdd*: *bdd-above* (*existence-ivl t0 x0*)
  assume *t'-upper*: $y \leq t'$ **if** $y \in$ *existence-ivl t0 x0* **for** *y*
  have *s*: *Sup* (*existence-ivl t0 x0*) ∈ *T'*
   using *initial-time-bounds*[*OF iv-defined*]
   apply −
   apply (*rule mem-is-intervalI*[*of - t0 t'*])
  by (*auto simp*: *ivl t' bdd intro*!: *t'-upper cSup-least*[*OF existence-ivl-notempty*[*OF iv-defined*]])
  have ∗: $t \in T'$ **if** $t0 \leq t$ $t \in$ *existence-ivl t0 x0* **for** *t*
    by (*rule mem-is-intervalI*[*OF* ‹*is-interval T'*› ‹*t0* ∈ *T'*› *s*]) (*auto intro*!: *cSup-upper that bdd*)
  from *sup-existence-maximal*[*OF iv-defined mem-compact K bdd*, *OF* ∗]
  show *False* **using** *s ivl* **by** *auto*
 qed
qed

lemma
 mem-compact-implies-subset-existence-interval:
 assumes *iv-defined*: *t0* ∈ *T* *x0* ∈ *X*
 assumes *mem-compact*: ⋀*t*. *t* ∈ *T* ⟹ *t* ∈ *existence-ivl t0 x0* ⟹ *flow t0 x0 t* ∈ *K*
 assumes *K*: *compact K* *K* ⊆ *X*
 shows *T* ⊆ *existence-ivl t0 x0*
  by (*rule subset-mem-compact-implies-subset-existence-interval*; (*fact* | *rule order-refl interval iv-defined*))

lemma
 global-right-existence-ivl-explicit:
 assumes $b \geq t0$
 assumes *b*: *b* ∈ *existence-ivl t0 x0*
 obtains *d K* **where** $d > 0$ $K > 0$
  ball x0 d ⊆ *X*
  ⋀*y*. *y* ∈ *ball x0 d* ⟹ *b* ∈ *existence-ivl t0 y*
  ⋀*t y*. *y* ∈ *ball x0 d* ⟹ *t* ∈ {*t0* .. *b*} ⟹
   dist (*flow t0 x0 t*) (*flow t0 y t*) ≤ *dist x0 y* ∗ *exp* (*K* ∗ *abs* (*t* − *t0*))
proof −
 note *iv-defined* = *mem-existence-ivl-iv-defined*[*OF b*]
 define *seg* **where** *seg* ≡ (λ*t*. *flow t0 x0 t*) ‘ (*closed-segment t0 b*)
 have [*simp*]: *x0* ∈ *seg*
  by (*auto simp*: *seg-def intro*!: *image-eqI*[**where** *x*=*t0*] *simp*: *closed-segment-eq-real-ivl iv-defined*)
 have *seg* ≠ {} **by** (*auto simp*: *seg-def closed-segment-eq-real-ivl*)

```
moreover
have compact seg
  using iv-defined b
  by (auto simp: seg-def closed-segment-eq-real-ivl
    intro!: compact-continuous-image continuous-at-imp-continuous-on flow-continuous;
    metis (erased, opaque-lifting) atLeastAtMost-iff closed-segment-eq-real-ivl
      closed-segment-subset-existence-ivl contra-subsetD order.trans)
moreover note open-domain(2)
moreover have seg ⊆ X
  using closed-segment-subset-existence-ivl b
  by (auto simp: seg-def intro!: flow-in-domain iv-defined)
ultimately
obtain e where e: 0 < e {x. infdist x seg ≤ e} ⊆ X
  thm compact-in-open-separated
  by (rule compact-in-open-separated)
define A where A ≡ {x. infdist x seg ≤ e}

have A ⊆ X using e by (simp add: A-def)

have mem-existence-ivlI: ⋀s. t0 ≤ s ⟹ s ≤ b ⟹ s ∈ existence-ivl t0 x0
 by (rule in-existence-between-zeroI[OF b]) (auto simp: closed-segment-eq-real-ivl)

have compact A
  unfolding A-def
  by (rule compact-infdist-le) fact+
have compact {t0 .. b} {t0 .. b} ⊆ T
  subgoal by simp
  subgoal
   using mem-existence-ivlI mem-existence-ivl-subset[of - x0] iv-defined b ivl-subset-existence-ivl
    by blast
  done
from lipschitz-on-compact[OF this ‹compact A› ‹A ⊆ X›]
obtain K' where K': ⋀t. t ∈ {t0 .. b} ⟹ K'−lipschitz-on A (f t)
  by metis
define K where K ≡ K' + 1
have 0 < K 0 ≤ K
  using assms lipschitz-on-nonneg[OF K', of t0]
  by (auto simp: K-def)
have K: ⋀t. t ∈ {t0 .. b} ⟹ K−lipschitz-on A (f t)
  unfolding K-def
  using ‹- ⟹ lipschitz-on K' A -›
  by (rule lipschitz-on-mono) auto

have [simp]: x0 ∈ A using ‹0 < e› by (auto simp: A-def)


define d where d ≡ min e (e ∗ exp (−K ∗ (b − t0)))
hence d: 0 < d d ≤ e d ≤ e ∗ exp (−K ∗ (b − t0))
  using e by auto
```

```
have d-times-exp-le: d ∗ exp (K ∗ (t − t0)) ≤ e if t0 ≤ t t ≤ b for t
proof −
  from that have d ∗ exp (K ∗ (t − t0)) ≤ d ∗ exp (K ∗ (b − t0))
    using ‹0 ≤ K› ‹0 < d›
    by (auto intro!: mult-left-mono)
  also have d ∗ exp (K ∗ (b − t0)) ≤ e
    using d by (auto simp: exp-minus divide-simps)
  finally show ?thesis .
qed
have ball x0 d ⊆ X using d ‹A ⊆ X›
  by (auto simp: A-def dist-commute intro!: infdist-le2[where a=x0])
note iv-defined
{
  fix y
  assume y: y ∈ ball x0 d
  hence y ∈ A using d
    by (auto simp: A-def dist-commute intro!: infdist-le2[where a=x0])
  hence y ∈ X using ‹A ⊆ X› by auto
  note y-iv = ‹t0 ∈ T› ‹y ∈ X›
  have in-A: flow t0 y t ∈ A if t: t0 ≤ t t ∈ existence-ivl t0 y t ≤ b for t
  proof (rule ccontr)
    assume flow-out: flow t0 y t ∉ A
    obtain t′ where t′:
      t0 ≤ t′
      t′ ≤ t
      ⋀t. t ∈ {t0 .. t′} ⟹ flow t0 x0 t ∈ A
      infdist (flow t0 y t′) seg ≥ e
      ⋀t. t ∈ {t0 .. t′} ⟹ flow t0 y t ∈ A
    proof −
      let ?out = ((λt. infdist (flow t0 y t) seg) −‘ {e..}) ∩ {t0..t}
      have compact ?out
        unfolding compact-eq-bounded-closed
      proof safe
        show bounded ?out by (auto intro!: bounded-closed-interval)
        have continuous-on {t0 .. t} ((λt. infdist (flow t0 y t) seg))
          using closed-segment-subset-existence-ivl t iv-defined
          by (force intro!: continuous-at-imp-continuous-on
            continuous-intros flow-continuous
            simp: closed-segment-eq-real-ivl)
        thus closed ?out
          by (simp add: continuous-on-closed-vimage)
      qed
      moreover
      have t ∈ (λt. infdist (flow t0 y t) seg) −‘ {e..} ∩ {t0..t}
        using flow-out ‹t0 ≤ t›
        by (auto simp: A-def)
      hence ?out ≠ {}
        by blast
```

ultimately have ∃ *s*∈*?out*. ∀ *t*∈*?out*. *s* ≤ *t*
  by (*rule compact-attains-inf*)
then obtain *t'* **where** *t'*:
  ⋀*s*. *e* ≤ *infdist* (*flow t0 y s*) *seg* ⟹ *t0* ≤ *s* ⟹ *s* ≤ *t* ⟹ *t'* ≤ *s*
  e ≤ *infdist* (*flow t0 y t'*) *seg*
  t0 ≤ *t'* *t'* ≤ *t*
  by (*auto simp*: *vimage-def Ball-def*) *metis*
have *flow-in*: *flow t0 x0 s* ∈ *A* **if** *s*: *s* ∈ {*t0* .. *t'*} **for** *s*
proof −
  from *s* **have** *s* ∈ *closed-segment t0 b*
    using ‹*t* ≤ *b*› *t'* **by** (*auto simp*: *closed-segment-eq-real-ivl*)
  then show *?thesis*
    using *s* ‹*e* > *0*› **by** (*auto simp*: *seg-def A-def*)
qed
have *flow t0 y t'* ∈ *A* **if** *t'* = *t0*
  using *y d iv-defined that*
  by (*auto simp*: *A-def* ‹*y* ∈ *X*› *infdist-le2*[**where** *a*=*x0*] *dist-commute*)
moreover
have *flow t0 y s* ∈ *A* **if** *s*: *s* ∈ {*t0* ..< *t'*} **for** *s*
proof −
  from *s* **have** *s* ∈ *closed-segment t0 b*
    using ‹*t* ≤ *b*› *t'* **by** (*auto simp*: *closed-segment-eq-real-ivl*)
  from *t'*(*1*)[*of s*]
  have *t'* > *s* ⟹ *t0* ≤ *s* ⟹ *s* ≤ *t* ⟹ *e* > *infdist* (*flow t0 y s*) *seg*
    by *force*
  then show *?thesis*
    using *s t'* ‹*e* > *0*› **by** (*auto simp*: *seg-def A-def*)
qed
moreover
note *left-of-in* = *this*
have *closed A* **using** ‹*compact A*› **by** (*auto simp*: *compact-eq-bounded-closed*)
have ((λ*s*. *flow t0 y s*) ⟶ *flow t0 y t'*) (*at-left t'*)
  using *closed-segment-subset-existence-ivl*[*OF t*(*2*)] *t'* ‹*y* ∈ *X*› *iv-defined*
by (*intro flow-tendsto*) (*auto intro*!: *tendsto-intros simp*: *closed-segment-eq-real-ivl*)
with ‹*closed A*› - - **have** *t'* ≠ *t0* ⟹ *flow t0 y t'* ∈ *A*
proof (*rule Lim-in-closed-set*)
  assume *t'* ≠ *t0*
  hence *t'* > *t0* **using** *t'* **by** *auto*
  hence *eventually* (λ*x*. *x* ≥ *t0*) (*at-left t'*)
    by (*metis eventually-at-left less-imp-le*)
  thus *eventually* (λ*x*. *flow t0 y x* ∈ *A*) (*at-left t'*)
    unfolding *eventually-at-filter*
    by *eventually-elim* (*auto intro*!: *left-of-in*)
qed *simp*
ultimately have *flow-y-in*: *s* ∈ {*t0* .. *t'*} ⟹ *flow t0 y s* ∈ *A* **for** *s*
  by (*cases s* = *t'*; *fastforce*)
have
  t0 ≤ *t'*
  t' ≤ *t*

$\bigwedge t.\ t \in \{t0\ ..\ t'\} \Longrightarrow$ *flow t0 x0 t* $\in A$
infdist (*flow t0 y t'*) *seg* $\geq e$
$\bigwedge t.\ t \in \{t0\ ..\ t'\} \Longrightarrow$ *flow t0 y t* $\in A$
by (*auto intro*!: *flow-in flow-y-in*) *fact+*
thus *?thesis* **..**
qed
{
fix *s* **assume** *s*: $s \in \{t0\ ..\ t'\}$
hence $t0 \leq s$ **by** *simp*
have $s \leq b$
using *t t' s b*
by *auto*
hence *sx0*: $s \in$ *existence-ivl t0 x0*
by (*simp add*: ‹$t0 \leq s$› *mem-existence-ivlI*)
have *sy*: $s \in$ *existence-ivl t0 y*
by (*meson atLeastAtMost-iff contra-subsetD s t'*(*1*) *t'*(*2*) *that*(*2*) *ivl-subset-existence-ivl*)
have *int*: *flow t0 y s* − *flow t0 x0 s* =
y − *x0* + (*integral* {*t0* .. *s*} (λ*t*. *f t* (*flow t0 y t*)) −
integral {*t0* .. *s*} (λ*t*. *f t* (*flow t0 x0 t*)))
using *iv-defined s*
unfolding *flow-fixed-point*[*OF sx0*] *flow-fixed-point*[*OF sy*]
by (*simp add*: *algebra-simps ivl-integral-def*)
have *norm* (*flow t0 y s* − *flow t0 x0 s*) ≤ *norm* (*y* − *x0*) +
norm (*integral* {*t0* .. *s*} (λ*t*. *f t* (*flow t0 y t*)) −
integral {*t0* .. *s*} (λ*t*. *f t* (*flow t0 x0 t*)))
unfolding *int*
by (*rule norm-triangle-ineq*)
also
have *norm* (*integral* {*t0* .. *s*} (λ*t*. *f t* (*flow t0 y t*)) −
integral {*t0* .. *s*} (λ*t*. *f t* (*flow t0 x0 t*))) =
norm (*integral* {*t0* .. *s*} (λ*t*. *f t* (*flow t0 y t*) − *f t* (*flow t0 x0 t*)))
using *closed-segment-subset-existence-ivl*[*of s x0*] *sx0 closed-segment-subset-existence-ivl*[*of s y*] *sy*
by (*subst Henstock-Kurzweil-Integration.integral-diff*)
(*auto intro*!: *integrable-continuous-real continuous-at-imp-continuous-on f-flow-continuous*
simp: *closed-segment-eq-real-ivl*)
also have ... ≤ (*integral* {*t0* .. *s*} (λ*t*. *norm* (*f t* (*flow t0 y t*) − *f t* (*flow t0 x0 t*))))
using *closed-segment-subset-existence-ivl*[*of s x0*] *sx0 closed-segment-subset-existence-ivl*[*of s y*] *sy*
by (*intro integral-norm-bound-integral*)
(*auto intro*!: *integrable-continuous-real continuous-at-imp-continuous-on f-flow-continuous continuous-intros*
simp: *closed-segment-eq-real-ivl*)
also have ... ≤ (*integral* {*t0* .. *s*} (λ*t*. *K* ∗ *norm* ((*flow t0 y t*) − (*flow t0 x0 t*))))
using *closed-segment-subset-existence-ivl*[*of s x0*] *sx0 closed-segment-subset-existence-ivl*[*of s y*] *sy*

iv-defined s t'(3,5) ‹*s* ≤ *b*›
by (*auto simp del*: *Henstock-Kurzweil-Integration.integral-mult-right intro*!: *integral-le integrable-continuous-real*
continuous-at-imp-continuous-on lipschitz-on-normD[*OF K*]
flow-continuous f-flow-continuous continuous-intros
simp: *closed-segment-eq-real-ivl*)
also have ... = *K* ∗ *integral* {*t0* .. *s*} (λ*t*. *norm* (*flow t0 y t* − *flow t0 x0 t*))
using *closed-segment-subset-existence-ivl*[*of s x0*] *sx0 closed-segment-subset-existence-ivl*[*of s y*] *sy*
by (*subst integral-mult*)
(*auto intro*!: *integrable-continuous-real continuous-at-imp-continuous-on*
lipschitz-on-normD[*OF K*] *flow-continuous f-flow-continuous continuous-intros*
simp: *closed-segment-eq-real-ivl*)
finally
have *norm*: *norm* (*flow t0 y s* − *flow t0 x0 s*) ≤
norm (*y* − *x0*) + *K* ∗ *integral* {*t0* .. *s*} (λ*t*. *norm* (*flow t0 y t* − *flow t0 x0 t*))
by *arith*
note *norm* ‹*s* ≤ *b*› *sx0 sy*
} **note** *norm-le* = *this*
from *norm-le*(*2*) *t'* **have** *t'* ∈ *closed-segment t0 b*
by (*auto simp*: *closed-segment-eq-real-ivl*)
hence *infdist* (*flow t0 y t'*) *seg* ≤ *dist* (*flow t0 y t'*) (*flow t0 x0 t'*)
by (*auto simp*: *seg-def infdist-le*)
also have ... ≤ *norm* (*flow t0 y t'* − *flow t0 x0 t'*)
by (*simp add*: *dist-norm*)
also have ... ≤ *norm* (*y* − *x0*) ∗ *exp* (*K* ∗ |*t'* − *t0*|)
unfolding *K-def*
apply (*rule exponential-initial-condition*[*OF - - - - - K'*])
subgoal by (*metis atLeastAtMost-iff local.norm-le*(*4*) *order-refl t'*(*1*))
subgoal by (*metis atLeastAtMost-iff local.norm-le*(*3*) *order-refl t'*(*1*))
subgoal using *e* **by** (*simp add*: *A-def*)
subgoal by (*metis closed-segment-eq-real-ivl t'*(*1*,*5*))
subgoal by (*metis closed-segment-eq-real-ivl t'*(*1*,*3*))
subgoal by (*simp add*: *closed-segment-eq-real-ivl local.norm-le*(*2*) *t'*(*1*))
done
also have ... < *d* ∗ *exp* (*K* ∗ (*t* − *t0*))
using *y d t' t*
by (*intro mult-less-le-imp-less*)
(*auto simp*: *dist-norm*[*symmetric*] *dist-commute intro*!: *mult-mono* ‹*0* ≤ *K*›)
also have ... ≤ *e*
by (*rule d-times-exp-le*; *fact*)
finally
have *infdist* (*flow t0 y t'*) *seg* < *e* **.**
with ‹*infdist* (*flow t0 y t'*) *seg* ≥ *e*› **show** *False*
by (*auto simp*: *frontier-def*)

qed

have $\{t0..b\} \subseteq$ *existence-ivl t0 y*
by (*rule subset-mem-compact-implies-subset-existence-interval*[*OF*
- *is-interval-cc* ‹$\{t0..b\} \subseteq T$› ‹$y \in X$› *in-A* ‹*compact A*› ‹$A \subseteq X$›])
(*auto simp*: ‹$t0 \leq b$›)
with ‹$t0 \leq b$› **have** *b-in*: $b \in$ *existence-ivl t0 y*
by *force*
{
fix *t* **assume** *t*: $t \in \{t0 .. b\}$
also have $\{t0 .. b\} = \{t0 -- b\}$
by (*auto simp*: *closed-segment-eq-real-ivl assms*)
also note *closed-segment-subset-existence-ivl*[*OF b-in*]
finally have *t-in*: $t \in$ *existence-ivl t0 y* **.**

note *t*
also note ‹$\{t0 .. b\} = \{t0 -- b\}$›
also note *closed-segment-subset-existence-ivl*[*OF assms*(*2*)]
finally have *t-in'*: $t \in$ *existence-ivl t0 x0* **.**
have *norm* (*flow t0 y t* − *flow t0 x0 t*) ≤ *norm* (*y* − *x0*) ∗ *exp* (*K* ∗ |*t* − *t0*|)
unfolding *K-def*
using *t closed-segment-subset-existence-ivl*[*OF b-in*] ‹$0 < e$›
by (*intro in-A exponential-initial-condition*[*OF t-in t-in'* ‹$A \subseteq X$› - - *K'*])
(*auto simp*: *closed-segment-eq-real-ivl A-def seg-def*)
hence *dist* (*flow t0 x0 t*) (*flow t0 y t*) ≤ *dist x0 y* ∗ *exp* (*K* ∗ |*t* − *t0*|)
by (*auto simp*: *dist-norm*[*symmetric*] *dist-commute*)
}
note *b-in this*
} **from** ‹$d > 0$› ‹$K > 0$› ‹*ball x0 d* ⊆ *X*› *this* **show** *?thesis* **..**
qed

lemma
global-left-existence-ivl-explicit:
assumes $b \leq t0$
assumes *b*: $b \in$ *existence-ivl t0 x0*
assumes *iv-defined*: $t0 \in T$ $x0 \in X$
obtains *d K* **where** $d > 0$ $K > 0$
ball x0 d ⊆ *X*
⋀*y*. *y* ∈ *ball x0 d* ⟹ *b* ∈ *existence-ivl t0 y*
⋀*t y*. *y* ∈ *ball x0 d* ⟹ *t* ∈ {*b* .. *t0*} ⟹ *dist* (*flow t0 x0 t*) (*flow t0 y t*) ≤ *dist x0 y* ∗ *exp* (*K* ∗ *abs* (*t* − *t0*))
proof −
interpret *rev*: *ll-on-open* (*preflect t0* ‘ *T*) (λ*t*. − *f* (*preflect t0 t*)) *X* **..**
have *t0'*: *t0* ∈ *preflect t0* ‘ *T* *x0* ∈ *X*
by (*auto intro*!: *iv-defined*)
from *assms* **have** *preflect t0 b* ≥ *t0* *preflect t0 b* ∈ *rev.existence-ivl t0 x0*
by (*auto simp*: *rev-existence-ivl-eq*)
from *rev.global-right-existence-ivl-explicit*[*OF this*]
obtain *d K* **where** *dK*: $d > 0$ $K > 0$

ball x0 d ⊆ *X*
⋀*y*. *y* ∈ *ball x0 d* ⟹ *preflect t0 b* ∈ *rev.existence-ivl t0 y*
⋀*t y*. *y* ∈ *ball x0 d* ⟹ *t* ∈ {*t0* .. *preflect t0 b*} ⟹ *dist* (*rev.flow t0 x0 t*) (*rev.flow t0 y t*) ≤ *dist x0 y* ∗ *exp* (*K* ∗ *abs* (*t* − *t0*))
by (*auto simp*: *rev-flow-eq* ‹*x0* ∈ *X*›)

have *ex-ivlI*: *dist x0 y* < *d* ⟹ *t* ∈ *existence-ivl t0 y* **if** *b* ≤ *t t* ≤ *t0* **for** *t y*
using *that dK*(*4*)[*of y*] *dK*(*3*) *iv-defined*
by (*auto simp*: *subset-iff rev-existence-ivl-eq*[*of*]
closed-segment-eq-real-ivl iv-defined in-existence-between-zeroI)
have *b* ∈ *existence-ivl t0 y* **if** *dist x0 y* < *d* **for** *y*
using *that dK*
by (*subst existence-ivl-eq-rev*) (*auto simp*: *iv-defined intro*!: *image-eqI*[**where** *x*=*preflect t0 b*])
with *dK* **have** *d* > *0 K* > *0*
ball x0 d ⊆ *X*
⋀*y*. *y* ∈ *ball x0 d* ⟹ *b* ∈ *existence-ivl t0 y*
⋀*t y*. *y* ∈ *ball x0 d* ⟹ *t* ∈ {*b* .. *t0*} ⟹ *dist* (*flow t0 x0 t*) (*flow t0 y t*) ≤ *dist x0 y* ∗ *exp* (*K* ∗ *abs* (*t* − *t0*))
by (*auto simp*: *flow-eq-rev iv-defined ex-ivlI* ‹*x0* ∈ *X*› *subset-iff*
intro!: *order-trans*[*OF dK*(*5*)] *image-eqI*[**where** *x*=*preflect t0 b*])
then show *?thesis* **..**
qed

lemma
global-existence-ivl-explicit:
assumes *a*: *a* ∈ *existence-ivl t0 x0*
assumes *b*: *b* ∈ *existence-ivl t0 x0*
assumes *le*: *a* ≤ *b*
obtains *d K* **where** *d* > *0 K* > *0*
ball x0 d ⊆ *X*
⋀*y*. *y* ∈ *ball x0 d* ⟹ *a* ∈ *existence-ivl t0 y*
⋀*y*. *y* ∈ *ball x0 d* ⟹ *b* ∈ *existence-ivl t0 y*
⋀*t y*. *y* ∈ *ball x0 d* ⟹ *t* ∈ {*a* .. *b*} ⟹
dist (*flow t0 x0 t*) (*flow t0 y t*) ≤ *dist x0 y* ∗ *exp* (*K* ∗ *abs* (*t* − *t0*))
proof −
note *iv-defined* = *mem-existence-ivl-iv-defined*[*OF a*]
define *r* **where** *r* ≡ *Max* {*t0*, *a*, *b*}
define *l* **where** *l* ≡ *Min* {*t0*, *a*, *b*}
have *r*: *r* ≥ *t0 r* ∈ *existence-ivl t0 x0*
using *a b* **by** (*auto simp*: *max-def r-def iv-defined*)
obtain *dr Kr* **where** *right*:
0 < *dr 0* < *Kr ball x0 dr* ⊆ *X*
⋀*y*. *y* ∈ *ball x0 dr* ⟹ *r* ∈ *existence-ivl t0 y*
⋀*y t*. *y* ∈ *ball x0 dr* ⟹ *t* ∈ {*t0*..*r*} ⟹ *dist* (*flow t0 x0 t*) (*flow t0 y t*) ≤ *dist x0 y* ∗ *exp* (*Kr* ∗ |*t* − *t0*|)
by (*rule global-right-existence-ivl-explicit*[*OF r*]) *blast*

have *l*: *l* ≤ *t0 l* ∈ *existence-ivl t0 x0*

using *a b* **by** (*auto simp*: *min-def l-def iv-defined*)
obtain *dl Kl* **where** *left*:
0 < dl 0 < Kl ball x0 dl ⊆ *X*
⋀*y. y* ∈ *ball x0 dl* ⟹ *l* ∈ *existence-ivl t0 y*
⋀*y t. y* ∈ *ball x0 dl* ⟹ *t* ∈ {*l* .. *t0*} ⟹ *dist* (*flow t0 x0 t*) (*flow t0 y t*) ≤ *dist x0 y* ∗ *exp* (*Kl* ∗ |*t* − *t0*|)
by (*rule global-left-existence-ivl-explicit*[*OF l iv-defined*]) *blast*

define *d* **where** *d* ≡ *min dr dl*
define *K* **where** *K* ≡ *max Kr Kl*

note *iv-defined*
have *0 < d 0 < K ball x0 d* ⊆ *X*
using *left right* **by** (*auto simp*: *d-def K-def*)
moreover
{
fix *y* **assume** *y*: *y* ∈ *ball x0 d*
hence *y* ∈ *X* **using** ‹*ball x0 d* ⊆ *X*› **by** *auto*
from *y*
closed-segment-subset-existence-ivl[*OF left*(*4*), *of y*]
closed-segment-subset-existence-ivl[*OF right*(*4*), *of y*]
have *a* ∈ *existence-ivl t0 y b* ∈ *existence-ivl t0 y*
by (*auto simp*: *d-def l-def r-def min-def max-def closed-segment-eq-real-ivl split*: *if-split-asm*)
}
moreover
{
fix *t y*
assume *y*: *y* ∈ *ball x0 d*
and *t*: *t* ∈ {*a* .. *b*}
from *y* **have** *y* ∈ *X* **using** ‹*ball x0 d* ⊆ *X*› **by** *auto*
have *dist* (*flow t0 x0 t*) (*flow t0 y t*) ≤ *dist x0 y* ∗ *exp* (*K* ∗ *abs* (*t* − *t0*))
proof *cases*
assume *t* ≥ *t0*
hence *dist* (*flow t0 x0 t*) (*flow t0 y t*) ≤ *dist x0 y* ∗ *exp* (*Kr* ∗ *abs* (*t* − *t0*))
using *y t*
by (*intro right*) (*auto simp*: *d-def r-def*)
also have *exp* (*Kr* ∗ *abs* (*t* − *t0*)) ≤ *exp* (*K* ∗ *abs* (*t* − *t0*))
by (*auto simp*: *mult-left-mono K-def max-def mult-right-mono*)
finally show *?thesis* **by** (*simp add*: *mult-left-mono*)
next
assume ¬*t* ≥ *t0*
hence *dist* (*flow t0 x0 t*) (*flow t0 y t*) ≤ *dist x0 y* ∗ *exp* (*Kl* ∗ *abs* (*t* − *t0*))
using *y t*
by (*intro left*) (*auto simp*: *d-def l-def*)
also have *exp* (*Kl* ∗ *abs* (*t* − *t0*)) ≤ *exp* (*K* ∗ *abs* (*t* − *t0*))
by (*auto simp*: *mult-left-mono K-def max-def mult-right-mono*)
finally show *?thesis* **by** (*simp add*: *mult-left-mono*)
qed

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

lemma *eventually-exponential-separation*:
assumes *a*: $a \in$ *existence-ivl t0 x0*
assumes *b*: $b \in$ *existence-ivl t0 x0*
assumes *le*: $a \le b$
obtains *K* **where** $K > 0$ $\forall_F$ *y in at x0*. $\forall t \in \{a..b\}$. *dist* (*flow t0 x0 t*) (*flow t0 y t*) $\le$ *dist x0 y* $*$ *exp* ($K * |t - t0|$)
proof −
from *global-existence-ivl-explicit*[*OF assms*]
obtain *d K* **where** $*$: $d > 0$ $K > 0$
ball x0 d $\subseteq X$
$\bigwedge y$. $y \in$ *ball x0 d* $\Longrightarrow a \in$ *existence-ivl t0 y*
$\bigwedge y$. $y \in$ *ball x0 d* $\Longrightarrow b \in$ *existence-ivl t0 y*
$\bigwedge t\ y$. $y \in$ *ball x0 d* $\Longrightarrow t \in \{a .. b\} \Longrightarrow$
dist (*flow t0 x0 t*) (*flow t0 y t*) $\le$ *dist x0 y* $*$ *exp* ($K *$ *abs* ($t - t0$))
by *auto*
note ‹$K > 0$›
moreover
have *eventually* (λy. $y \in$ *ball x0 d*) (*at x0*)
using ‹$d > 0$›[*THEN eventually-at-ball*]
by *eventually-elim simp*
hence *eventually* (λy. $\forall t \in \{a..b\}$. *dist* (*flow t0 x0 t*) (*flow t0 y t*) $\le$ *dist x0 y* $*$ *exp* ($K * |t - t0|$)) (*at x0*)
by *eventually-elim* (*safe intro*!: $*$)
ultimately show *?thesis* **..**
qed

lemma *eventually-mem-existence-ivl*:
assumes *b*: $b \in$ *existence-ivl t0 x0*
shows $\forall_F$ *x in at x0*. $b \in$ *existence-ivl t0 x*
proof −
from *mem-existence-ivl-iv-defined*[*OF b*] **have** *iv-defined*: $t0 \in T$ $x0 \in X$ **by** *simp-all*
note *eiit* = *existence-ivl-initial-time*[*OF iv-defined*]
{
fix *a b*
assume *assms*: $a \in$ *existence-ivl t0 x0* $b \in$ *existence-ivl t0 x0* $a \le b$
from *global-existence-ivl-explicit*[*OF assms*]
obtain *d K* **where** $*$: $d > 0$ $K > 0$
ball x0 d $\subseteq X$
$\bigwedge y$. $y \in$ *ball x0 d* $\Longrightarrow a \in$ *existence-ivl t0 y*
$\bigwedge y$. $y \in$ *ball x0 d* $\Longrightarrow b \in$ *existence-ivl t0 y*
$\bigwedge t\ y$. $y \in$ *ball x0 d* $\Longrightarrow t \in \{a .. b\} \Longrightarrow$
dist (*flow t0 x0 t*) (*flow t0 y t*) $\le$ *dist x0 y* $*$ *exp* ($K *$ *abs* ($t - t0$))
by *auto*
have *eventually* (λy. $y \in$ *ball x0 d*) (*at x0*)
using ‹$d > 0$›[*THEN eventually-at-ball*]

by *eventually-elim simp*
then have $\forall_F$ *x in at x0*. *a* ∈ *existence-ivl t0 x* ∧ *b* ∈ *existence-ivl t0 x*
by (*eventually-elim*) (*auto intro*!: ∗)
} **from** *this*[*OF b eiit*] *this*[*OF eiit b*]
show *?thesis*
by (*cases t0* ≤ *b*) (*auto simp*: *eventually-mono*)
qed

lemma *uniform-limit-flow*:
assumes *a*: *a* ∈ *existence-ivl t0 x0*
assumes *b*: *b* ∈ *existence-ivl t0 x0*
assumes *le*: *a* ≤ *b*
shows *uniform-limit* {*a* .. *b*} (*flow t0*) (*flow t0 x0*) (*at x0*)
proof (*rule uniform-limitI*)
fix *e*::*real* **assume** *0* < *e*
from *eventually-exponential-separation*[*OF assms*] **obtain** *K* **where** *0* < *K*
$\forall_F$ *y in at x0*. ∀ *t*∈{*a*..*b*}. *dist* (*flow t0 x0 t*) (*flow t0 y t*) ≤ *dist x0 y* ∗ *exp* (*K* ∗ |*t* − *t0*|)
by *auto*
note *this*(*2*)
moreover
let *?m* = *max* (*abs* (*b* − *t0*)) (*abs* (*a* − *t0*))
have *eventually* (λ*y*. ∀ *t*∈{*a*..*b*}. *dist x0 y* ∗ *exp* (*K* ∗ |*t* − *t0*|) ≤ *dist x0 y* ∗ *exp* (*K* ∗ *?m*)) (*at x0*)
using ‹*a* ≤ *b*› ‹*0* < *K*›
by (*auto intro*!: *mult-left-mono always-eventually*)
moreover
have *eventually* (λ*y*. *dist x0 y* ∗ *exp* (*K* ∗ *?m*) < *e*) (*at x0*)
using ‹*0* < *e*› **by** (*auto intro*!: *order-tendstoD tendsto-eq-intros*)
ultimately
show *eventually* (λ*y*. ∀ *t*∈{*a*..*b*}. *dist* (*flow t0 y t*) (*flow t0 x0 t*) < *e*) (*at x0*)
by *eventually-elim* (*force simp*: *dist-commute*)
qed

lemma *eventually-at-fst*:
assumes *eventually P* (*at* (*fst x*))
assumes *P* (*fst x*)
shows *eventually* (λ*h*. *P* (*fst h*)) (*at x*)
using *assms*
unfolding *eventually-at-topological*
by (*metis open-vimage-fst rangeI range-fst vimageE vimageI*)

lemma *eventually-at-snd*:
assumes *eventually P* (*at* (*snd x*))
assumes *P* (*snd x*)
shows *eventually* (λ*h*. *P* (*snd h*)) (*at x*)
using *assms*
unfolding *eventually-at-topological*
by (*metis open-vimage-snd rangeI range-snd vimageE vimageI*)

lemma
  shows *open-state-space*: *open* (*Sigma X* (*existence-ivl t0*))
  and *flow-continuous-on-state-space*:
    continuous-on (*Sigma X* (*existence-ivl t0*)) (λ(*x*, *t*). *flow t0 x t*)
proof (*safe intro*!: *topological-space-class.openI continuous-at-imp-continuous-on*)
  fix *t x* **assume** *x* ∈ *X* **and** *t*: *t* ∈ *existence-ivl t0 x*
  have *iv-defined*: *t0* ∈ *T x* ∈ *X*
    using *mem-existence-ivl-iv-defined*[*OF t*] **by** *auto*
  from ‹*x* ∈ *X*› *t open-existence-ivl*
  obtain *e* **where** *e*: *e* > *0 cball t e* ⊆ *existence-ivl t0 x*
    by (*metis open-contains-cball*)
  hence *ivl*: *t* − *e* ∈ *existence-ivl t0 x t* + *e* ∈ *existence-ivl t0 x t* − *e* ≤ *t* + *e*
    by (*auto simp*: *cball-def dist-real-def*)
  obtain *d K* **where** *dK*:
    0 < *d 0* < *K ball x d* ⊆ *X*
    ⋀*y*. *y* ∈ *ball x d* ⟹ *t* − *e* ∈ *existence-ivl t0 y*
    ⋀*y*. *y* ∈ *ball x d* ⟹ *t* + *e* ∈ *existence-ivl t0 y*
    ⋀*y s*. *y* ∈ *ball x d* ⟹ *s* ∈ {*t* − *e*..*t* + *e*} ⟹
      dist (*flow t0 x s*) (*flow t0 y s*) ≤ *dist x y* ∗ *exp* (*K* ∗ |*s* − *t0*|)
    by (*rule global-existence-ivl-explicit*[*OF ivl*]) *blast*
  let *?T* = *ball x d* × *ball t e*
  have *open ?T* **by** (*auto intro*!: *open-Times*)
  moreover have (*x*, *t*) ∈ *?T* **by** (*auto simp*: *dK* ‹*0* < *e*›)
  moreover have *?T* ⊆ *Sigma X* (*existence-ivl t0*)
  proof *safe*
    fix *s y* **assume** *y*: *y* ∈ *ball x d* **and** *s*: *s* ∈ *ball t e*
    with ‹*ball x d* ⊆ *X*› **show** *y* ∈ *X* **by** *auto*
    have *ball t e* ⊆ *closed-segment t0* (*t* − *e*) ∪ *closed-segment t0* (*t* + *e*)
      by (*auto simp*: *closed-segment-eq-real-ivl dist-real-def*)
    with ‹*y* ∈ *X*› *s closed-segment-subset-existence-ivl*[*OF dK*(*4*)[*OF y*]]
      closed-segment-subset-existence-ivl[*OF dK*(*5*)[*OF y*]]
    show *s* ∈ *existence-ivl t0 y*
      by *auto*
  qed
  ultimately show ∃ *T*. *open T* ∧ (*x*, *t*) ∈ *T* ∧ *T* ⊆ *Sigma X* (*existence-ivl t0*)
    by *blast*
  have ∗∗: ∀$_F$ *s in at 0*. *norm* (*flow t0* (*x* + *fst s*) (*t* + *snd s*) − *flow t0 x t*) < *2*
∗ *eps*
    if *eps* > *0* **for** *eps* :: *real*
  proof −
    have ∀$_F$ *s in at 0*. *norm* (*flow t0* (*x* + *fst s*) (*t* + *snd s*) − *flow t0 x t*) =
      norm (*flow t0* (*x* + *fst s*) (*t* + *snd s*) − *flow t0 x* (*t* + *snd s*) +
        (*flow t0 x* (*t* + *snd s*) − *flow t0 x t*))
      by *auto*
    moreover
    have ∀$_F$ *s in at 0*.
        norm (*flow t0* (*x* + *fst s*) (*t* + *snd s*) − *flow t0 x* (*t* + *snd s*) +
          (*flow t0 x* (*t* + *snd s*) − *flow t0 x t*)) ≤

norm (*flow t0* (*x* + *fst s*) (*t* + *snd s*) − *flow t0 x* (*t* + *snd s*)) +
norm (*flow t0 x* (*t* + *snd s*) − *flow t0 x t*)
by *eventually-elim* (*rule norm-triangle-ineq*)
moreover
have ∀ *F s in at 0*. *t* + *snd s* ∈ *ball t e*
by (*auto simp*: *dist-real-def intro*!: *order-tendstoD*[*OF* - ‹*0* < *e*›]
intro!: *tendsto-eq-intros*)
moreover from *uniform-limit-flow*[*OF ivl*, *THEN uniform-limitD*, *OF* ‹*eps* > *0*›]
have ∀ *F* (*h*::(-)) *in at* (*fst* (*0*::′*a*∗*real*)).
∀ *t*∈{*t* − *e*..*t* + *e*}. *dist* (*flow t0 x t*) (*flow t0* (*x* + *h*) *t*) < *eps*
by (*subst* (*asm*) *at-to-0*)
(*auto simp*: *eventually-filtermap dist-commute ac-simps*)
hence ∀ *F* (*h*::(- ∗ *real*)) *in at 0*.
∀ *t*∈{*t* − *e*..*t* + *e*}. *dist* (*flow t0 x t*) (*flow t0* (*x* + *fst h*) *t*) < *eps*
by (*rule eventually-at-fst*) (*simp add*: ‹*eps* > *0*›)
moreover
have ∀ *F h in at* (*snd* (*0*::′*a* ∗ -)). *norm* (*flow t0 x* (*t* + *h*) − *flow t0 x t*) < *eps*
using *flow-continuous*[*OF t*, *unfolded isCont-def*, *THEN tendstoD*, *OF* ‹*eps* > *0*›]
by (*subst* (*asm*) *at-to-0*)
(*auto simp*: *eventually-filtermap dist-norm ac-simps*)
hence ∀ *F h*::(′*a* ∗ -) *in at 0*. *norm* (*flow t0 x* (*t* + *snd h*) − *flow t0 x t*) < *eps*
by (*rule eventually-at-snd*) (*simp add*: ‹*eps* > *0*›)
ultimately
show *?thesis*
proof *eventually-elim*
case (*elim s*)
note *elim*(*1*)
also note *elim*(*2*)
also note *elim*(*5*)
also
from *elim*(*3*) **have** *t* + *snd s* ∈ {*t* − *e*..*t* + *e*}
by (*auto simp*: *dist-real-def algebra-simps*)
from *elim*(*4*)[*rule-format*, *OF this*]
have *norm* (*flow t0* (*x* + *fst s*) (*t* + *snd s*) − *flow t0 x* (*t* + *snd s*)) < *eps*
by (*auto simp*: *dist-commute dist-norm*[*symmetric*])
finally
show *?case* **by** *simp*
qed
qed
have ∗: ∀ *F s in at 0*. *norm* (*flow t0* (*x* + *fst s*) (*t* + *snd s*) − *flow t0 x t*) < *eps*
if *eps* > *0* **for** *eps*::*real*
proof −
from *that* **have** *eps* / *2* > *0* **by** *simp*
from ∗∗[*OF this*] **show** *?thesis* **by** *auto*
qed
show *isCont* (λ(*x*, *y*). *flow t0 x y*) (*x*, *t*)
unfolding *isCont-iff*

by (*rule LIM-zero-cancel*)
(*auto simp*: *split-beta′ norm-conv-dist*[*symmetric*] *intro*!: *tendstoI* ∗)
qed

lemmas *flow-continuous-on-compose*[*continuous-intros*] =
continuous-on-compose-Pair[*OF flow-continuous-on-state-space*]

lemma *flow-isCont-state-space*: *t* ∈ *existence-ivl t0 x0* ⟹ *isCont* (λ(*x*, *t*). *flow t0 x t*) (*x0*, *t*)
using *flow-continuous-on-state-space*[*of*] *mem-existence-ivl-iv-defined*[*of t x0*]
using *continuous-on-eq-continuous-at open-state-space* **by** *fastforce*

lemma
flow-absolutely-integrable-on[*integrable-on-simps*]:
assumes *s* ∈ *existence-ivl t0 x0*
shows (λ*x*. *norm* (*flow t0 x0 x*)) *integrable-on closed-segment t0 s*
using *assms*
by (*auto simp*: *closed-segment-eq-real-ivl intro*!: *integrable-continuous-real continuous-intros*
flow-continuous-on-intro
intro: *in-existence-between-zeroI*)

lemma *existence-ivl-eq-domain*:
assumes *iv-defined*: *t0* ∈ *T x0* ∈ *X*
assumes *bnd*: ⋀*tm tM t x*. *tm* ∈ *T* ⟹ *tM* ∈ *T* ⟹ ∃ *M*. ∃ *L*. ∀ *t* ∈ {*tm* .. *tM*}. ∀ *x* ∈ *X*. *norm* (*f t x*) ≤ *M* + *L* ∗ *norm x*
assumes *is-interval T X* = *UNIV*
shows *existence-ivl t0 x0* = *T*
proof −
from *assms* **have** *XI*: *x* ∈ *X* **for** *x* **by** *auto*
{
fix *tm tM* **assume** *tm*: *tm* ∈ *T* **and** *tM*: *tM* ∈ *T* **and** *tmtM*: *tm* ≤ *t0 t0* ≤ *tM*
from *bnd*[*OF tm tM*] **obtain** *M′ L′*
where *bnd′*: ⋀*x t*. *x* ∈ *X* ⟹ *tm* ≤ *t* ⟹ *t* ≤ *tM* ⟹ *norm* (*f t x*) ≤ *M′* + *L′* ∗ *norm x*
by *force*
define *M* **where** *M* ≡ *norm M′* + *1*
define *L* **where** *L* ≡ *norm L′* + *1*
have *bnd*: ⋀*x t*. *x* ∈ *X* ⟹ *tm* ≤ *t* ⟹ *t* ≤ *tM* ⟹ *norm* (*f t x*) ≤ *M* + *L* ∗ *norm x*
by (*auto simp*: *M-def L-def intro*!: *bnd′*[*THEN order-trans*] *add-mono mult-mono*)
have *M* > *0 L* > *0* **by** (*auto simp*: *L-def M-def*)

let *?r* = (*norm x0* + |*tm* − *tM*| ∗ *M* + *1*) ∗ *exp* (*L* ∗ |*tm* − *tM*|)
define *K* **where** *K* ≡ *cball* (*0*::′*a*) *?r*
have *K*: *compact K K* ⊆ *X*
by (*auto simp*: *K-def* ‹*X* = *UNIV*›)
{

fix *t* **assume** *t*: *t* ∈ *existence-ivl t0 x0* **and** *le*: *tm* ≤ *t t* ≤ *tM*
{
fix *s* **assume** *sc*: *s* ∈ *closed-segment t0 t*
then have *s*: *s* ∈ *existence-ivl t0 x0* **and** *le*: *tm* ≤ *s s* ≤ *tM* **using** *t le sc*
using *closed-segment-subset-existence-ivl*
apply −
subgoal by *force*
subgoal by (*metis* (*full-types*) *atLeastAtMost-iff closed-segment-eq-real-ivl order-trans tmtM*(*1*))
subgoal by (*metis* (*full-types*) *atLeastAtMost-iff closed-segment-eq-real-ivl order-trans tmtM*(*2*))
done
from *sc* **have** *nle*: *norm* (*t0* − *s*) ≤ *norm* (*t0* − *t*) **by** (*auto simp*: *closed-segment-eq-real-ivl split*: *if-split-asm*)
from *flow-fixed-point*[*OF s*]
have *norm* (*flow t0 x0 s*) ≤ *norm x0* + *integral* (*closed-segment t0 s*) (λ*t*. *M* + *L* ∗ *norm* (*flow t0 x0 t*))
using *tmtM*
using *closed-segment-subset-existence-ivl*[*OF s*] *le*
by (*auto simp*:
intro!: *norm-triangle-le norm-triangle-ineq4*[*THEN order.trans*]
ivl-integral-norm-bound-integral bnd
integrable-continuous-closed-segment
integrable-continuous-real
continuous-intros
continuous-on-subset[*OF flow-continuous-on*]
flow-in-domain
mem-existence-ivl-subset)
(*auto simp*: *closed-segment-eq-real-ivl split*: *if-splits*)
also have . . . = *norm x0* + *norm* (*t0* − *s*) ∗ *M* + *L* ∗ *integral* (*closed-segment t0 s*) (λ*t*. *norm* (*flow t0 x0 t*))
by (*simp add*: *integral-add integrable-on-simps* ‹*s* ∈ *existence-ivl* - -›
integral-const-closed-segment abs-minus-commute)
also have *norm* (*t0* − *s*) ∗ *M* ≤ *norm* (*t0* − *t*) ∗ *M*
using *nle* ‹*M* > *0*› **by** *auto*
also have . . . ≤ . . . + *1* **by** *simp*
finally have *norm* (*flow t0 x0 s*) ≤ *norm x0* + *norm* (*t0* − *t*) ∗ *M* + *1* +
L ∗ *integral* (*closed-segment t0 s*) (λ*t*. *norm* (*flow t0 x0 t*)) **by** *simp*
}
then have *norm* (*flow t0 x0 t*) ≤ (*norm x0* + *norm* (*t0* − *t*) ∗ *M* + *1*) ∗ *exp* (*L* ∗ |*t* − *t0*|)
using *closed-segment-subset-existence-ivl*[*OF t*]
by (*intro gronwall-more-general-segment*[**where** *a*=*t0* **and** *b* = *t* **and** *t* = *t*])
(*auto simp*: ‹*0* < *L*› ‹*0* < *M*› *less-imp-le*
intro!: *add-nonneg-pos mult-nonneg-nonneg add-nonneg-nonneg continuous-intros*
flow-continuous-on-intro)
also have . . . ≤ *?r*

```
        using le tmtM
          by (auto simp: less-imp-le ‹0 < M› ‹0 < L› abs-minus-commute intro!:
mult-mono)
      finally
      have flow t0 x0 t ∈ K by (simp add: dist-norm K-def)
    } note flow-compact = this

    have {tm..tM} ⊆ existence-ivl t0 x0
        using tmtM tm ‹x0 ∈ X› ‹compact K› ‹K ⊆ X› mem-is-intervalI[OF
‹is-interval T› ‹tm ∈ T› ‹tM ∈ T›]
     by (intro subset-mem-compact-implies-subset-existence-interval[OF - - - -flow-compact])
         (auto simp: tmtM is-interval-cc)
  } note bnds = this

  show existence-ivl t0 x0 = T
  proof safe
    fix x assume x: x ∈ T
    from bnds[OF x iv-defined(1)] bnds[OF iv-defined(1) x]
    show x ∈ existence-ivl t0 x0
      by (cases x ≤ t0) auto
  qed (insert existence-ivl-subset, fastforce)
qed

lemma flow-unique:
  assumes t ∈ existence-ivl t0 x0
  assumes phi t0 = x0
  assumes ⋀t. t ∈ existence-ivl t0 x0 ⟹ (phi has-vector-derivative f t (phi t))
(at t)
  assumes ⋀t. t ∈ existence-ivl t0 x0 ⟹ phi t ∈ X
  shows flow t0 x0 t = phi t
  apply (rule maximal-existence-flow[where K=existence-ivl t0 x0])
  subgoal by (auto intro!: solves-odeI simp: has-vderiv-on-def assms at-within-open[OF
- open-existence-ivl])
  subgoal by fact
  subgoal by simp
  subgoal using mem-existence-ivl-iv-defined[OF ‹t ∈ existence-ivl t0 x0›] by simp
  subgoal by (simp add: existence-ivl-subset)
  subgoal by fact
  done

lemma flow-unique-on:
  assumes t ∈ existence-ivl t0 x0
  assumes phi t0 = x0
  assumes (phi has-vderiv-on (λt. f t (phi t))) (existence-ivl t0 x0)
  assumes ⋀t. t ∈ existence-ivl t0 x0 ⟹ phi t ∈ X
  shows flow t0 x0 t = phi t
  using flow-unique[where phi=phi, OF assms(1,2) - assms(4)] assms(3)
  by (auto simp: has-vderiv-on-open)
```

end — *local-lipschitz T X f*

locale *two-ll-on-open* =
  F: *ll-on-open T1 F X* + *G*: *ll-on-open T2 G X*
  for *F T1 G T2 X J x0* +
  fixes *e*::*real* **and** *K*
  assumes *t0-in-J*: $0 \in J$
  assumes *J-subset*: $J \subseteq$ *F.existence-ivl 0 x0*
  assumes *J-ivl*: *is-interval J*
  assumes *F-lipschitz*: $\bigwedge t.\ t \in J \Longrightarrow K-$*lipschitz-on X* (*F t*)
  assumes *K-pos*: $0 < K$
  assumes *F-G-norm-ineq*: $\bigwedge t\ x.\ t \in J \Longrightarrow x \in X \Longrightarrow$ *norm* $(F\ t\ x - G\ t\ x) < e$
begin

context begin

lemma *F-iv-defined*: $0 \in T1$ $x0 \in X$
  subgoal using *F.existence-ivl-initial-time-iff J-subset t0-in-J* **by** *blast*
  subgoal using *F.mem-existence-ivl-iv-defined*(*2*) *J-subset t0-in-J* **by** *blast*
  done

lemma *e-pos*: $0 < e$
  using *le-less-trans*[*OF norm-ge-zero F-G-norm-ineq*[*OF t0-in-J F-iv-defined*(*2*)]]
  by *assumption*

qualified definition *flow0 t* = *F.flow 0 x0 t*
qualified definition *Y t* = *G.flow 0 x0 t*

lemma *norm-X-Y-bound*:
shows $\forall t \in J \cap$ *G.existence-ivl 0 x0*. *norm* (*flow0 t* − *Y t*) ≤ *e* / *K* ∗ (*exp*(*K* ∗ |*t*|) − *1*)
proof(*safe*)
  fix *t* **assume** $t \in J$
  assume *tG*: $t \in$ *G.existence-ivl 0 x0*
  have $0 \in J$ **by** (*simp add*: *t0-in-J*)

  let *?u*=λ*t*. *norm* (*flow0 t* − *Y t*)
  show *norm* (*flow0 t* − *Y t*) ≤ *e* / *K* ∗ (*exp* (*K* ∗ |*t*|) − *1*)
  proof(*cases* $0 \le t$)
    assume $0 \le t$
    hence [*simp*]: $|t| = t$ **by** *simp*

    have *t0-t-in-J*: $\{0..t\} \subseteq J$
      using ‹$t \in J$› ‹$0 \in J$› *J-ivl*
      using *mem-is-interval-1-I atLeastAtMost-iff subsetI* **by** *blast*

    note *F-G-flow-cont*[*continuous-intros*] =
      continuous-on-subset[*OF F.flow-continuous-on*]
      continuous-on-subset[*OF G.flow-continuous-on*]

have *?u t + e/K ≤ e/K ∗ exp(K ∗ t)*
proof(*rule gronwall*[**where** *g=λt. ?u t + e/K, OF - - - - K-pos ‹0 ≤ t› order.refl*])
fix *s* **assume** *0 ≤ s s ≤ t*
hence *{0..s} ⊆ J* **using** *t0-t-in-J* **by** *auto*

hence *t0-s-in-existence*:
{0..s} ⊆ F.existence-ivl 0 x0
{0..s} ⊆ G.existence-ivl 0 x0
using *J-subset tG ‹0 ≤ s› ‹s ≤ t› G.ivl-subset-existence-ivl*[*OF tG*]
by *auto*

hence *s-in-existence*:
s ∈ F.existence-ivl 0 x0
s ∈ G.existence-ivl 0 x0
using *‹0 ≤ s›* **by** *auto*

note *cont-statements*[*continuous-intros*] =
F-iv-defined
F.flow-in-domain
G.flow-in-domain
F.mem-existence-ivl-subset
G.mem-existence-ivl-subset

have [*integrable-on-simps*]:
continuous-on {0..s} (λs. F s (F.flow 0 x0 s))
continuous-on {0..s} (λs. G s (G.flow 0 x0 s))
continuous-on {0..s} (λs. F s (G.flow 0 x0 s))
continuous-on {0..s} (λs. G s (F.flow 0 x0 s))
using *t0-s-in-existence*
by (*auto intro*!: *continuous-intros integrable-continuous-real*)

have *flow0 s − Y s = integral {0..s} (λs. F s (flow0 s) − G s (Y s))*
using *‹0 ≤ s›*
by (*simp add*: *flow0-def Y-def Henstock-Kurzweil-Integration.integral-diff integrable-on-simps ivl-integral-def*
F.flow-fixed-point[*OF s-in-existence*(*1*)]
G.flow-fixed-point[*OF s-in-existence*(*2*)])
also have ... = *integral {0..s} (λs. (F s (flow0 s) − F s (Y s)) + (F s (Y s) − G s (Y s)))*
by *simp*
also have ... = *integral {0..s} (λs. F s (flow0 s) − F s (Y s)) + integral {0..s} (λs. F s (Y s) − G s (Y s))*
by (*simp add*: *Henstock-Kurzweil-Integration.integral-diff Henstock-Kurzweil-Integration.integral-add flow0-def Y-def integrable-on-simps*)
finally have *?u s ≤ norm (integral {0..s} (λs. F s (flow0 s) − F s (Y s))) + norm (integral {0..s} (λs. F s (Y s) − G s (Y s)))*
by (*simp add*: *norm-triangle-ineq*)

also have ... ≤ *integral* {*0..s*} (λ*s. norm* (*F s* (*flow0 s*) − *F s* (*Y s*))) + *integral* {*0..s*} (λ*s. norm* (*F s* (*Y s*) − *G s* (*Y s*)))
using *t0-s-in-existence*
by (*auto simp add*: *flow0-def Y-def*
intro!: *add-mono continuous-intros continuous-on-imp-absolutely-integrable-on*)
also have ... ≤ *integral* {*0..s*} (λ*s. K* ∗ *?u s*) + *integral* {*0..s*} (λ*s. e*)
proof (*rule add-mono*[*OF integral-le integral-le*])
show *norm* (*F x* (*flow0 x*) − *F x* (*Y x*)) ≤ *K* ∗ *norm* (*flow0 x* − *Y x*) **if** *x* ∈ {*0..s*} **for** *x*
using *F-lipschitz*[*unfolded lipschitz-on-def*, *THEN conjunct2*] *that*
cont-statements(*1*,*2*,*4*)
t0-s-in-existence F-iv-defined
by (*metis F-lipschitz flow0-def Y-def* ‹{*0..s*} ⊆ *J*› *lipschitz-on-normD F.flow-in-domain*
G.flow-in-domain subsetCE)
show ⋀*x. x* ∈ {*0..s*} ⟹ *norm* (*F x* (*Y x*) − *G x* (*Y x*)) ≤ *e*
using *F-G-norm-ineq cont-statements*(*2*,*3*) *t0-s-in-existence*
using *Y-def* ‹{*0..s*} ⊆ *J*› *cont-statements*(*5*) *subset-iff G.flow-in-domain*
by (*metis eucl-less-le-not-le*)
qed (*simp-all add*: *t0-s-in-existence continuous-intros integrable-on-simps flow0-def Y-def*)
also have ... = *K* ∗ *integral* {*0..s*} (λ*s. ?u s* + *e* / *K*)
using *K-pos t0-s-in-existence*
by (*simp-all add*: *algebra-simps Henstock-Kurzweil-Integration.integral-add flow0-def Y-def continuous-intros*
continuous-on-imp-absolutely-integrable-on)
finally show *?u s* + *e* / *K* ≤ *e* / *K* + *K* ∗ *integral* {*0..s*} (λ*s. ?u s* + *e* / *K*)
by *simp*
next
show *continuous-on* {*0..t*} (λ*t. norm* (*flow0 t* − *Y t*) + *e* / *K*)
using *t0-t-in-J J-subset G.ivl-subset-existence-ivl*[*OF tG*]
by (*auto simp add*: *flow0-def Y-def intro*!: *continuous-intros*)
next
fix *s* **assume** *0* ≤ *s s* ≤ *t*
show *0* ≤ *norm* (*flow0 s* − *Y s*) + *e* / *K*
using *e-pos K-pos* **by** *simp*
next
show *0* < *e* / *K* **using** *e-pos K-pos* **by** *simp*
qed
thus *?thesis* **by** (*simp add*: *algebra-simps*)
next
assume ¬*0* ≤ *t*
hence *t* ≤ *0* **by** *simp*
hence [*simp*]: |*t*| = −*t* **by** *simp*

have *t0-t-in-J*: {*t..0*} ⊆ *J*
using ‹*t* ∈ *J*› ‹*0* ∈ *J*› *J-ivl* ‹¬ *0* ≤ *t*› *atMostAtLeast-subset-convex is-interval-convex-1*
by *auto*

note *F-G-flow-cont*[*continuous-intros*] =
continuous-on-subset[*OF F.flow-continuous-on*]
continuous-on-subset[*OF G.flow-continuous-on*]

have *?u t* + *e*/*K* ≤ *e*/*K* ∗ *exp*(− *K* ∗ *t*)
proof(*rule gronwall-left*[**where** *g*=λ*t*. *?u t* + *e*/*K*, *OF - - - - K-pos order.refl* ‹*t* ≤ *0*›])
fix *s* **assume** *t* ≤ *s s* ≤ *0*
hence {*s..0*} ⊆ *J* **using** *t0-t-in-J* **by** *auto*

hence *t0-s-in-existence*:
{*s..0*} ⊆ *F.existence-ivl 0 x0*
{*s..0*} ⊆ *G.existence-ivl 0 x0*
using *J-subset G.ivl-subset-existence-ivl′*[*OF tG*] ‹*s* ≤ *0*› ‹*t* ≤ *s*›
by *auto*

hence *s-in-existence*:
s ∈ *F.existence-ivl 0 x0*
s ∈ *G.existence-ivl 0 x0*
using ‹*s* ≤ *0*› **by** *auto*

note *cont-statements*[*continuous-intros*] =
F-iv-defined
F.flow-in-domain
G.flow-in-domain
F.mem-existence-ivl-subset
G.mem-existence-ivl-subset
then have [*continuous-intros*]:
{*s..0*} ⊆ *T1*
{*s..0*} ⊆ *T2*
F.flow 0 x0 ‘ {*s..0*} ⊆ *X*
G.flow 0 x0 ‘ {*s..0*} ⊆ *X*
s ≤ *x* ⟹ *x* ≤ *0* ⟹ *x* ∈ *F.existence-ivl 0 x0*
s ≤ *x* ⟹ *x* ≤ *0* ⟹ *x* ∈ *G.existence-ivl 0 x0* **for** *x*
using *t0-s-in-existence*
by *auto*
have *flow0 s* − *Y s* = − *integral* {*s..0*} (λ*s*. *F s* (*flow0 s*) − *G s* (*Y s*))
using *t0-s-in-existence* ‹*s* ≤ *0*›
by (*simp add*: *flow0-def Y-def ivl-integral-def*
F.flow-fixed-point[*OF s-in-existence*(*1*)]
G.flow-fixed-point[*OF s-in-existence*(*2*)]
continuous-intros integrable-on-simps Henstock-Kurzweil-Integration.integral-diff)
also have ... = − *integral* {*s..0*} (λ*s*. (*F s* (*flow0 s*) − *F s* (*Y s*)) + (*F s* (*Y s*) − *G s* (*Y s*)))
by *simp*
also have ... = − (*integral* {*s..0*} (λ*s*. *F s* (*flow0 s*) − *F s* (*Y s*)) + *integral* {*s..0*} (λ*s*. *F s* (*Y s*) − *G s* (*Y s*)))
using *t0-s-in-existence*

by (*subst Henstock-Kurzweil-Integration.integral-add*) (*simp-all add*: *integral-add flow0-def Y-def continuous-intros integrable-on-simps*)
finally have *?u s* ≤ *norm* (*integral* {*s..0*} (λ*s. F s* (*flow0 s*) − *F s* (*Y s*))) + *norm* (*integral* {*s..0*} (λ*s. F s* (*Y s*) − *G s* (*Y s*)))
by (*metis* (*no-types*, *lifting*) *norm-minus-cancel norm-triangle-ineq*)
also have ... ≤ *integral* {*s..0*} (λ*s. norm* (*F s* (*flow0 s*) − *F s* (*Y s*))) + *integral* {*s..0*} (λ*s. norm* (*F s* (*Y s*) − *G s* (*Y s*)))
using *t0-s-in-existence*
by (*auto simp add*: *flow0-def Y-def intro*!: *continuous-intros continuous-on-imp-absolutely-integrable-on add-mono*)
also have ... ≤ *integral* {*s..0*} (λ*s. K* ∗ *?u s*) + *integral* {*s..0*} (λ*s. e*)
proof (*rule add-mono*[*OF integral-le integral-le*])
show *norm* (*F x* (*flow0 x*) − *F x* (*Y x*)) ≤ *K* ∗ *norm* (*flow0 x* − *Y x*) **if** *x*∈{*s..0*} **for** *x*
using *F-lipschitz*[*unfolded lipschitz-on-def*, *THEN conjunct2*]
cont-statements(*1*,*2*,*4*) *that*
t0-s-in-existence F-iv-defined
by (*metis F-lipschitz flow0-def Y-def* ‹{*s..0*} ⊆ *J*› *lipschitz-on-normD F.flow-in-domain*
G.flow-in-domain subsetCE)
show ⋀*x. x* ∈ {*s..0*} ⟹ *norm* (*F x* (*Y x*) − *G x* (*Y x*)) ≤ *e*
using *F-G-norm-ineq Y-def* ‹{*s..0*} ⊆ *J*› *cont-statements*(*5*) *subset-iff t0-s-in-existence*(*2*)
using *Y-def* ‹{*s..0*} ⊆ *J*› *cont-statements*(*5*) *subset-iff G.flow-in-domain*
by (*metis eucl-less-le-not-le*)
qed (*simp-all add*: *t0-s-in-existence continuous-intros integrable-on-simps flow0-def Y-def*)
also have ... = *K* ∗ *integral* {*s..0*} (λ*s. ?u s* + *e* / *K*)
using *K-pos t0-s-in-existence*
by (*simp-all add*: *algebra-simps Henstock-Kurzweil-Integration.integral-add t0-s-in-existence continuous-intros integrable-on-simps flow0-def Y-def*)
finally show *?u s* + *e* / *K* ≤ *e* / *K* + *K* ∗ *integral* {*s..0*} (λ*s. ?u s* + *e* / *K*)
by *simp*
next
show *continuous-on* {*t..0*} (λ*t. norm* (*flow0 t* − *Y t*) + *e* / *K*)
using *t0-t-in-J J-subset G.ivl-subset-existence-ivl'*[*OF tG*] *F-iv-defined*
by (*auto simp add*: *flow0-def Y-def intro*!: *continuous-intros*)
next
fix *s* **assume** *t* ≤ *s s* ≤ *0*
show *0* ≤ *norm* (*flow0 s* − *Y s*) + *e* / *K*
using *e-pos K-pos* **by** *simp*
next
show *0* < *e* / *K* **using** *e-pos K-pos* **by** *simp*
qed
thus *?thesis* **by** (*simp add*: *algebra-simps*)
qed
qed

end

end

locale *auto-ll-on-open* =
  fixes *f*::'*a*::{*banach*, *heine-borel*} ⇒ '*a* **and** *X*
  assumes *auto-local-lipschitz*: *local-lipschitz UNIV X* (λ-::*real*. *f*)
  assumes *auto-open-domain*[*intro*!, *simp*]: *open X*
begin

autonomous flow and existence interval

definition *flow0 x0 t* = *ll-on-open.flow UNIV* (λ-. *f*) *X 0 x0 t*

definition *existence-ivl0 x0* = *ll-on-open.existence-ivl UNIV* (λ-. *f*) *X 0 x0*

sublocale *ll-on-open-it UNIV* λ-. *f X 0*
  rewrites *flow* = (λ*t0 x0 t*. *flow0 x0* (*t* − *t0*))
    and *existence-ivl* = (λ*t0 x0*. (+) *t0* ' *existence-ivl0 x0*)
    and (+) *0* = (λ*x*::*real*. *x*)
    and *s* − *0* = *s*
    and (λ*x*. *x*) ' *S* = *S*
    and *s* ∈ (+) *t* ' *S* ⟷ *s* − *t* ∈ (*S*::*real set*)
    and *P* (*s* + *t* − *s*) = *P* (*t*::*real*)— TODO: why does just the equation not work?
    and *P* (*t* + *s* − *s*) = *P t*— TODO: why does just the equation not work?
proof −
  interpret *ll-on-open UNIV* λ-. *f X*
    by *unfold-locales* (*auto intro*!: *continuous-on-const auto-local-lipschitz*)
  show *ll-on-open-it UNIV* (λ-. *f*) *X* ..
  show (+) *0* = (λ*x*::*real*. *x*) (λ*x*. *x*) ' *S* = *S s* − *0* = *s P* (*t* + *s* − *s*) = *P t P* (*s* + *t* − *s*) = *P* (*t*::*real*)
    by *auto*
  show *flow* = (λ*t0 x0 t*. *flow0 x0* (*t* − *t0*))
    unfolding *flow0-def*
    by (*metis flow-def flow-shift-autonomous1 flow-shift-autonomous2 mem-existence-ivl-iv-defined*(*2*))
  show *existence-ivl* = (λ*t0 x0*. (+) *t0* ' *existence-ivl0 x0*)
    unfolding *existence-ivl0-def*
    apply (*safe intro*!: *ext*)
    subgoal using *image-iff mem-existence-ivl-shift-autonomous1* **by** *fastforce*
    subgoal premises *prems* **for** *t0 x0 x s*
    proof −
      have *f2*: ∀ *x1 x2*. (*x2*::*real*) − *x1* = − *1* ∗ *x1* + *x2*
        by *auto*
      have − *1* ∗ *t0* + (*t0* + *s*) = *s*
        by *auto*
      then show *?thesis*
      using *f2 prems mem-existence-ivl-iv-defined*(*2*) *mem-existence-ivl-shift-autonomous2*
        by *presburger*
    qed

done
show $(s \in (+)\ t\ `\ S) = (s - t \in S)$ **by** *force*
qed
— at this point, there should be no theorems about *existence-ivl*, only *existence-ivl0*. Moreover, (+) - ‘ - and - + - − - etc should have been removed

lemma *existence-ivl-zero*: $x0 \in X \Longrightarrow 0 \in$ *existence-ivl0 x0* **by** *simp*

lemmas [*continuous-intros del*] = *continuous-on-f*
lemmas *continuous-on-f-comp*[*continuous-intros*] = *continuous-on-f*[*OF continuous-on-const - subset-UNIV*]

lemma
flow-in-compact-right-existence:
assumes $\bigwedge t.\ 0 \le t \Longrightarrow t \in$ *existence-ivl0 x* $\Longrightarrow$ *flow0 x t* $\in K$
assumes *compact K* $K \subseteq X$
assumes $x \in X$ $t \ge 0$
shows $t \in$ *existence-ivl0 x*
proof (*rule ccontr*)
assume $t \notin$ *existence-ivl0 x*
have *bdd-above* (*existence-ivl0 x*)
by (*rule bdd-above-is-intervalI*[*OF is-interval-existence-ivl* ‹$0 \le t$› *existence-ivl-zero*]) *fact*+
from *sup-existence-maximal*[*OF UNIV-I* ‹$x \in X$› *assms*(*1*−*3*) *this*]
show *False* **by** *auto*
qed

lemma
flow-in-compact-left-existence:
assumes $\bigwedge t.\ t \le 0 \Longrightarrow t \in$ *existence-ivl0 x* $\Longrightarrow$ *flow0 x t* $\in K$
assumes *compact K* $K \subseteq X$
assumes $x \in X$ $t \le 0$
shows $t \in$ *existence-ivl0 x*
proof (*rule ccontr*)
assume $t \notin$ *existence-ivl0 x*
have *bdd-below* (*existence-ivl0 x*)
by (*rule bdd-below-is-intervalI*[*OF is-interval-existence-ivl* ‹$t \le 0$› - *existence-ivl-zero*]) *fact*+
from *inf-existence-minimal*[*OF UNIV-I* ‹$x \in X$› *assms*(*1*−*3*) *this*]
show *False* **by** *auto*
qed

end

locale *compact-continuously-diff* =
derivative-on-prod T X f $\lambda(t, x).\ f'\ x\ o_L$ *snd-blinfun*
for *T X* **and** *f*::*real* ⇒ ′*a*::{*banach*,*perfect-space*,*heine-borel*} ⇒ ′*a*
and *f*′::′*a* ⇒ (′*a*, ′*a*) *blinfun* +
assumes *compact-domain*: *compact X*

assumes *convex*: *convex X*
assumes *nonempty-domains*: $T \neq \{\}$ $X \neq \{\}$
assumes *continuous-derivative*: *continuous-on X f'*
begin

lemma *ex-onorm-bound*:
$\exists B. \forall x \in X.$ *norm* $(f'\ x) \leq B$
proof −
from - *compact-domain* **have** *compact* $(f'$ ‘ $X)$
by (*intro compact-continuous-image continuous-derivative*)
hence *bounded* $(f'$ ‘ $X)$ **by** (*rule compact-imp-bounded*)
thus *?thesis*
by (*auto simp add*: *bounded-iff cball-def norm-blinfun.rep-eq*)
qed

definition *onorm-bound* = (*SOME B*. $\forall x \in X.$ *norm* $(f'\ x) \leq B$)

lemma *onorm-bound*: **assumes** $x \in X$ **shows** *norm* $(f'\ x) \leq$ *onorm-bound*
unfolding *onorm-bound-def*
using *someI-ex*[*OF ex-onorm-bound*] *assms*
by *blast*

sublocale *closed-domain X*
using *compact-domain* **by** *unfold-locales* (*rule compact-imp-closed*)

sublocale *global-lipschitz T X f onorm-bound*
proof (*unfold-locales*, *rule lipschitz-onI*)
fix *t z y*
assume $t \in T$ $y \in X$ $z \in X$
then have *norm* $(f\ t\ y - f\ t\ z) \leq$ *onorm-bound* $*$ *norm* $(y - z)$
using *onorm-bound*
by (*intro differentiable-bound*[**where** *f'=f'*, *OF convex*])
(*auto intro*!: *derivative-eq-intros simp*: *norm-blinfun.rep-eq*)
thus *dist* $(f\ t\ y)$ $(f\ t\ z) \leq$ *onorm-bound* $*$ *dist y z*
by (*auto simp*: *dist-norm norm-Pair*)
next
from *nonempty-domains* **obtain** *x* **where** *x*: $x \in X$ **by** *auto*
show $0 \leq$ *onorm-bound*
using *dual-order.trans local.onorm-bound norm-ge-zero x* **by** *blast*
qed

end — *compact X*

locale *unique-on-compact-continuously-diff* = *self-mapping* +
compact-interval T +
compact-continuously-diff T X f
begin

sublocale *unique-on-closed t0 T x0 f X onorm-bound*

by *unfold-locales* (*auto intro*!: *f′ has-derivative-continuous-on*)

end

locale *c1-on-open* =
fixes *f*::*′a*::{*banach*, *perfect-space*, *heine-borel*} ⇒ *′a* **and** *f′ X*
assumes *open-dom*[*simp*]: *open X*
assumes *derivative-rhs*:
$\bigwedge x.\ x \in X \Longrightarrow$ (*f has-derivative blinfun-apply* (*f′ x*)) (*at x*)
assumes *continuous-derivative*: *continuous-on X f′*
begin

lemmas *continuous-derivative-comp*[*continuous-intros*] =
continuous-on-compose2[*OF continuous-derivative*]

lemma *derivative-tendsto*[*tendsto-intros*]:
assumes [*tendsto-intros*]: $(g \longrightarrow l)\ F$
and $l \in X$
shows $((\lambda x.\ f'\ (g\ x)) \longrightarrow f'\ l)\ F$
using *continuous-derivative*[*simplified continuous-on*] *assms*
by (*auto simp*: *at-within-open*[*OF - open-dom*]
intro!: *tendsto-eq-intros*
intro: *tendsto-compose*)

lemma *c1-on-open-rev*[*intro*, *simp*]: *c1-on-open* (−*f*) (−*f′*) *X*
using *derivative-rhs continuous-derivative*
by *unfold-locales*
(*auto intro*!: *continuous-intros derivative-eq-intros*
simp: *fun-Compl-def blinfun.bilinear-simps*)

lemma *derivative-rhs-compose*[*derivative-intros*]:
((*g has-derivative g′*) (*at x within s*)) ⟹ *g x* ∈ *X* ⟹
((λ*x*. *f* (*g x*)) *has-derivative*
(λ*xa*. *blinfun-apply* (*f′* (*g x*)) (*g′ xa*)))
(*at x within s*)
by (*metis has-derivative-compose*[*of g g′ x s f f′* (*g x*)] *derivative-rhs*)

sublocale *auto-ll-on-open*
proof (*standard*, *rule local-lipschitzI*)
fix *x* **and** *t*::*real*
assume *x* ∈ *X*
with *open-contains-cball*[*of UNIV*::*real set*] *open-UNIV*
open-contains-cball[*of X*] *open-dom*
obtain *u v* **where** *uv*: *cball t u* ⊆ *UNIV cball x v* ⊆ *X u* > *0 v* > *0*
by *blast*
let *?T* = *cball t u* **and** *?X* = *cball x v*
have *bounded ?X* **by** *simp*
have *compact* (*cball x v*)
by *simp*

interpret *compact-continuously-diff ?T ?X λ-. f f′*
  using *uv*
  by *unfold-locales*
   (*auto simp*: *convex-cball cball-eq-empty split-beta′*
   intro!: *derivative-eq-intros continuous-on-compose2*[*OF continuous-derivative*]
     continuous-intros)
 have *onorm-bound−lipschitz-on ?X f*
  using *lipschitz*[*of t*] *uv*
  by *auto*
 thus ∃ *u*>*0*. ∃ *L*. ∀ *t* ∈ *cball t u* ∩ *UNIV*. *L*−*lipschitz-on* (*cball x u* ∩ *X*) *f*
  by (*intro exI*[**where** *x*=*v*])
   (*auto intro*!: *exI*[**where** *x*=*onorm-bound*] ‹*0* < *v*› *simp*: *Int-absorb2 uv*)
qed (*auto intro*!: *continuous-intros*)

end — *?x* ∈ *X* ⟹ (*f has-derivative blinfun-apply* (*f′ ?x*)) (*at ?x*)

locale *c1-on-open-euclidean* = *c1-on-open f f′ X*
 for *f*::′*a*::*euclidean-space* ⇒ - **and** *f′ X*
begin
lemma *c1-on-open-euclidean-anchor*: *True* **..**

definition *vareq x0 t* = *f′* (*flow0 x0 t*)

interpretation *var*: *ll-on-open existence-ivl0 x0 vareq x0 UNIV*
 apply *standard*
 apply (*auto intro*!: *c1-implies-local-lipschitz*[**where** *f′* = λ(*t*, *x*). *vareq x0 t*] *continuous-intros*
   derivative-eq-intros
   simp: *split-beta′ blinfun.bilinear-simps vareq-def*)
 using *local.mem-existence-ivl-iv-defined*(*2*) **apply** *blast*
 using *local.existence-ivl-reverse local.mem-existence-ivl-iv-defined*(*2*) **apply** *blast*
 using *local.mem-existence-ivl-iv-defined*(*2*) **apply** *blast*
 using *local.existence-ivl-reverse local.mem-existence-ivl-iv-defined*(*2*) **apply** *blast*
 done

context begin

lemma *continuous-on-A*[*continuous-intros*]:
 assumes *continuous-on S a*
 assumes *continuous-on S b*
 assumes ⋀*s*. *s* ∈ *S* ⟹ *a s* ∈ *X*
 assumes ⋀*s*. *s* ∈ *S* ⟹ *b s* ∈ *existence-ivl0* (*a s*)
 shows *continuous-on S* (λ*s*. *vareq* (*a s*) (*b s*))
proof −
 have *continuous-on S* (λ*x*. *f′* (*flow0* (*a x*) (*b x*)))
  by (*auto intro*!: *continuous-intros assms flow-in-domain*)
 then show *?thesis*
  by (*rule continuous-on-eq*) (*auto simp*: *assms vareq-def*)
qed

lemmas [*intro*] = *mem-existence-ivl-iv-defined*

context
  fixes *x0*::′*a*
begin

lemma *flow0-defined*: *xa* ∈ *existence-ivl0 x0* ⟹ *flow0 x0 xa* ∈ *X*
  by (*auto simp*: *flow-in-domain*)

lemma *continuous-on-flow0*: *continuous-on* (*existence-ivl0 x0*) (*flow0 x0*)
  by (*auto simp*: *intro*!: *continuous-intros*)

lemmas *continuous-on-flow0-comp*[*continuous-intros*] = *continuous-on-compose2*[*OF continuous-on-flow0*]

lemma *varexivl-eq-exivl*:
  assumes *t* ∈ *existence-ivl0 x0*
  shows *var.existence-ivl x0 t a* = *existence-ivl0 x0*
proof (*rule var.existence-ivl-eq-domain*)
  fix *s t x*
  assume *s*: *s* ∈ *existence-ivl0 x0* **and** *t*: *t* ∈ *existence-ivl0 x0*
  then have {*s* .. *t*} ⊆ *existence-ivl0 x0*
   by (*metis atLeastatMost-empty-iff2 empty-subsetI real-Icc-closed-segment var.closed-segment-subset-domain*)
  then have *continuous-on* {*s* .. *t*} (*vareq x0*)
   by (*auto simp*: *closed-segment-eq-real-ivl intro*!: *continuous-intros flow0-defined*)
  then have *compact* ((*vareq x0*) ‘ {*s* .. *t*})
    using *compact-Icc*
    by (*rule compact-continuous-image*)
  then obtain *B* **where** *B*: ⋀*u*. *u* ∈ {*s* .. *t*} ⟹ *norm* (*vareq x0 u*) ≤ *B*
    by (*force dest*!: *compact-imp-bounded simp*: *bounded-iff*)
  show ∃ *M L*. ∀ *t*∈{*s*..*t*}. ∀ *x*∈*UNIV*. *norm* (*blinfun-apply* (*vareq x0 t*) *x*) ≤ *M* + *L* ∗ *norm x*
    by (*rule exI*[**where** *x*=*0*], *rule exI*[**where** *x*=*B*])
      (*auto intro*!: *order-trans*[*OF norm-blinfun*] *mult-right-mono B simp*:)
qed (*auto intro*: *assms*)

definition *vector-Dflow u0 t* ≡ *var.flow x0 0 u0 t*

qualified abbreviation *Y z t* ≡ *flow0* (*x0* + *z*) *t*

Linearity of the solution to the variational equation. TODO: generalize this and some other things for arbitrary linear ODEs

lemma *vector-Dflow-linear*:
assumes *t* ∈ *existence-ivl0 x0*
shows *vector-Dflow* ($\alpha *_R$ *a* + $\beta *_R$ *b*) *t* = $\alpha *_R$ *vector-Dflow a t* + $\beta *_R$ *vector-Dflow b t*
proof −
  note *mem-existence-ivl-iv-defined*[*OF assms*, *intro*, *simp*]

have ((*λc. α* $*_R$ *var.flow x0 0 a c* + *β* $*_R$ *var.flow x0 0 b c*) *solves-ode* (*λx. vareq x0 x*)) (*existence-ivl0 x0*) *UNIV*
by (*auto intro*!: *derivative-intros var.flow-has-vector-derivative solves-odeI simp*: *blinfun.bilinear-simps varexivl-eq-exivl vareq-def*[*symmetric*])
moreover
have *α* $*_R$ *var.flow x0 0 a 0* + *β* $*_R$ *var.flow x0 0 b 0* = *α* $*_R$ *a* + *β* $*_R$ *b* **by** *simp*
moreover note *is-interval-existence-ivl*[*of x0*]
ultimately show *?thesis*
unfolding *vareq-def*[*symmetric*] *vector-Dflow-def*
by (*rule var.maximal-existence-flow*) (*auto simp*: *assms*)
qed

lemma *linear-vector-Dflow*:
assumes *t* ∈ *existence-ivl0 x0*
shows *linear* (*λz. vector-Dflow z t*)
using *vector-Dflow-linear*[*OF assms*, *of 1 - 1*] *vector-Dflow-linear*[*OF assms*, *of - - 0*]
by (*auto intro*!: *linearI*)

lemma *bounded-linear-vector-Dflow*:
assumes *t* ∈ *existence-ivl0 x0*
shows *bounded-linear* (*λz. vector-Dflow z t*)
by (*simp add*: *linear-linear linear-vector-Dflow assms*)

lemma *vector-Dflow-continuous-on-time*: *x0* ∈ *X* ⟹ *continuous-on* (*existence-ivl0 x0*) (*λt. vector-Dflow z t*)
using *var.flow-continuous-on*[*of x0 0 z*] *varexivl-eq-exivl*
unfolding *vector-Dflow-def*
by (*auto simp*:)

proposition *proposition-17-6-weak*:
— from "Differential Equations, Dynamical Systems, and an Introduction to Chaos", Hirsch/Smale/Devaney
assumes *t* ∈ *existence-ivl0 x0*
shows (*λy.* (*Y* (*y* − *x0*) *t* − *flow0 x0 t* − *vector-Dflow* (*y* − *x0*) *t*) $/_R$ *norm* (*y* − *x0*)) − *x0* → *0*
proof−
note *x0-def* = *mem-existence-ivl-iv-defined*[*OF assms*]
have *0* ∈ *existence-ivl0 x0*
by (*simp add*: *x0-def*)

Find some *J* ⊆ *existence-ivl0 x0* with *0* ∈ *J* and *t* ∈ *J*.

define *t0* **where** *t0* ≡ *min 0 t*
define *t1* **where** *t1* ≡ *max 0 t*
define *J* **where** *J* ≡ {*t0..t1*}

have *t0* ≤ *0 0* ≤ *t1 0* ∈ *J J* ≠ {} *t* ∈ *J compact J*
and *J-in-existence*: *J* ⊆ *existence-ivl0 x0*

using *ivl-subset-existence-ivl ivl-subset-existence-ivl′ x0-def assms*
by (*auto simp add*: *J-def t0-def t1-def min-def max-def*)

{
fix *z S*
assume *assms*: $x0 + z \in X$ $S \subseteq$ *existence-ivl0* $(x0 + z)$
have *continuous-on S* $(Y\ z)$
using *flow-continuous-on assms*(*1*)
by (*intro continuous-on-subset*[*OF - assms*(*2*)]) (*simp add*:)
}
note [*continuous-intros*] = *this integrable-continuous-real blinfun.continuous-on*

have *U-continuous*[*continuous-intros*]: $\bigwedge z.$ *continuous-on J* (*vector-Dflow z*)
by(*rule continuous-on-subset*[*OF vector-Dflow-continuous-on-time*[*OF* ‹$x0 \in X$›] *J-in-existence*])

from ‹$t \in J$›
have $t0 \le t$
and $t \le t1$
and $t0 \le t1$
and $t0 \in$ *existence-ivl0 x0*
and $t \in$ *existence-ivl0 x0*
and $t1 \in$ *existence-ivl0 x0*
using *J-def J-in-existence* **by** *auto*
from *global-existence-ivl-explicit*[*OF* ‹$t0 \in$ *existence-ivl0 x0*› ‹$t1 \in$ *existence-ivl0 x0*› ‹$t0 \le t1$›]
obtain *u K* **where** *uK-def*:
$0 < u$
$0 < K$
ball x0 u $\subseteq X$
$\bigwedge y.\ y \in$ *ball x0 u* $\Longrightarrow t0 \in$ *existence-ivl0 y*
$\bigwedge y.\ y \in$ *ball x0 u* $\Longrightarrow t1 \in$ *existence-ivl0 y*
$\bigwedge t\ y.\ y \in$ *ball x0 u* $\Longrightarrow t \in J \Longrightarrow$ *dist* (*flow0 x0 t*) $(Y\ (y - x0)\ t) \le$ *dist x0 y* $*$ *exp* $(K * |t|)$
by (*auto simp add*: *J-def*)

have *J-in-existence-ivl*: $\bigwedge y.\ y \in$ *ball x0 u* $\Longrightarrow J \subseteq$ *existence-ivl0 y*
unfolding *J-def*
using *uK-def*
by (*simp add*: *real-Icc-closed-segment segment-subset-existence-ivl t0-def t1-def*)
have *ball-in-X*: $\bigwedge z.\ z \in$ *ball 0 u* $\Longrightarrow x0 + z \in X$
using *uK-def*(*3*)
by (*auto simp*: *dist-norm*)

have *flow0-J-props*: *flow0 x0* ‘ $J \neq \{\}$ *compact* (*flow0 x0* ‘ *J*) *flow0 x0*‘ $J \subseteq X$
using ‹$t0 \le t1$›
using *J-def*(*1*) *J-in-existence*
by (*auto simp add*: *J-def intro*!:
compact-continuous-image continuous-intros flow-in-domain)

have [*continuous-intros*]: *continuous-on J* ($\lambda s.\ f'$ (*flow0 x0 s*))
using *J-in-existence*
by (*auto intro*!: *continuous-intros flow-in-domain simp*:)

Show the thesis via cases $t = 0$, $0 < t$ and $t < 0$.

show *?thesis*
proof(*cases* $t = 0$)
assume $t = 0$
show *?thesis*
unfolding ‹$t = 0$› *Lim-at*
proof(*simp add*: *dist-norm*[*of - 0*] *del*: *zero-less-dist-iff*, *safe*, *rule exI*, *rule conjI*[*OF* ‹$0 < u$›], *safe*)
fix *e*::*real* **and** *x* **assume** $0 < e$ *dist x x0* $< u$
hence $x \in X$
using *uK-def*(*3*)
by (*auto simp*: *dist-commute*)
hence *inverse* (*norm* ($x - x0$)) $*$ *norm* (Y ($x - x0$) 0 $-$ *flow0 x0 0* $-$ *vector-Dflow* ($x - x0$) 0) $= 0$
using *x0-def*
by (*simp add*: *vector-Dflow-def*)
thus *inverse* (*norm* ($x - x0$)) $*$ *norm* (*flow0 x 0* $-$ *flow0 x0 0* $-$ *vector-Dflow* ($x - x0$) 0) $< e$
using ‹$0 < e$› **by** *auto*
qed
next
assume $t \neq 0$
show *?thesis*
proof(*unfold Lim-at*, *safe*)
fix *e*::*real* **assume** $0 < e$
then obtain e' **where** $0 < e'$ $e' < e$
using *dense* **by** *auto*

obtain *N*
where *N-ge-SupS*: *Sup* { *norm* (f' (*flow0 x0 s*)) |*s*. $s \in J$ } $\leq N$ (**is** *Sup ?S* $\leq N$)
and *N-gr-0*: $0 < N$
— We need N to be an upper bound of {*norm* (f' (*flow0 x0 s*)) |*s*. $s \in J$}, but also larger than zero.
by (*meson le-cases less-le-trans linordered-field-no-ub*)
have *N-ineq*: $\bigwedge s.\ s \in J \Longrightarrow$ *norm* (f' (*flow0 x0 s*)) $\leq N$
proof−
fix *s* **assume** $s \in J$
have *?S* = (*norm o* f' *o flow0 x0*) ‘ *J* **by** *auto*
moreover have *continuous-on J* (*norm o* f' *o flow0 x0*)
using *J-in-existence*
by (*auto intro*!: *continuous-intros*)
ultimately have $\exists\, a\ b.$ *?S* = {*a*..*b*} $\wedge$ $a \leq b$
using *continuous-image-closed-interval*[*OF* ‹$t0 \leq t1$›]

by (*simp add*: *J-def*)
then obtain *a b* **where** *?S* = {*a..b*} **and** *a* ≤ *b* **by** *auto*
hence *bdd-above ?S* **by** *simp*
from ‹*s* ∈ *J*› *cSup-upper*[*OF - this*]
have *norm* (*f'* (*flow0 x0 s*)) ≤ *Sup ?S*
by *auto*
thus *norm* (*f'* (*flow0 x0 s*)) ≤ *N*
using *N-ge-SupS* **by** *simp*
qed

Define a small region around *flow0* ‘ *J*, that is a subset of the domain *X*.

from *compact-in-open-separated*[*OF flow0-J-props*(*1*,*2*) *auto-open-domain* *flow0-J-props*(*3*)]
obtain *e-domain* **where** *e-domain-def*: *0* < *e-domain* {*x. infdist x* (*flow0* *x0* ‘ *J*) ≤ *e-domain*} ⊆ *X*
by *auto*
define *G* **where** *G* ≡ {*x*∈*X*. *infdist x* (*flow0 x0* ‘ *J*) < *e-domain*}
have *G-vimage*: *G* = ((λ*x*. *infdist x* (*flow0 x0* ‘ *J*)) −‘ {..<*e-domain*}) ∩ *X*
by (*auto simp*: *G-def*)
have *open G G* ⊆ *X*
unfolding *G-vimage*
by (*auto intro*!: *open-Int open-vimage continuous-intros continuous-at-imp-continuous-on*)

Define a compact subset H of G. Inside H, we can guarantee an upper bound on the Taylor remainder.

define *e-domain2* **where** *e-domain2* ≡ *e-domain* / *2*
have *e-domain2* > *0* *e-domain2* < *e-domain* **using** ‹*e-domain* > *0*›
by (*simp-all add*: *e-domain2-def*)
define *H* **where** *H* ≡ {*x*. *infdist x* (*flow0 x0* ‘ *J*) ≤ *e-domain2*}
have *H-props*: *H* ≠ {} *compact H H* ⊆ *G*
proof−
have *x0* ∈ *flow0 x0* ‘ *J*
unfolding *image-iff*
using ‹*0* ∈ *J*› *x0-def*
by *force*

hence *x0* ∈ *H*
using ‹*0* < *e-domain2*›
by (*simp add*: *H-def x0-def*)
thus *H* ≠ {}
by *auto*
next
show *compact H*
unfolding *H-def*
using ‹*0* < *e-domain2*› *flow0-J-props*
by (*intro compact-infdist-le*) *simp-all*
next
show *H* ⊆ *G*
proof

fix *x* **assume** $x \in H$
then have ∗: *infdist x* (*flow0 x0* ' *J*) < *e-domain*
using ‹*0* < *e-domain*›
by (*simp add*: *H-def e-domain2-def*)
then have $x \in X$
using *e-domain-def*(*2*)
by *auto*
with ∗ **show** $x \in G$
unfolding *G-def*
by *auto*
qed
qed

have *f′-cont-on-G*: (⋀*x*. $x \in G \Longrightarrow$ *isCont f′ x*)
using *continuous-on-interior*[*OF continuous-on-subset*[*OF continuous-derivative* ‹$G \subseteq X$›]]
by (*simp add*: *interior-open*[*OF* ‹*open G*›])

define *e1* **where** *e1* ≡ *e′* / (|*t*| ∗ *exp* (*K* ∗ |*t*|) ∗ *exp* (*N* ∗ |*t*|))
— *e1* is the bounding term for the Taylor remainder.
have *0* < |*t*|
using ‹*t* ≠ *0*›
by *simp*
hence *0* < *e1*
using ‹*0* < *e′*›
by (*simp add*: *e1-def*)

Taylor expansion of f on set G.

from *uniform-explicit-remainder-Taylor-1*[**where** *f*=*f* **and** *f′*=*f′*,
OF derivative-rhs[*OF subsetD*[*OF* ‹$G \subseteq X$›]] *f′-cont-on-G* ‹*open G*› *H-props* ‹*0* < *e1*›]
obtain *d-Taylor R*
where *Taylor-expansion*:
0 < *d-Taylor*
⋀*x z*. *f z* = *f x* + (*f′ x*) (*z* − *x*) + *R x z*
⋀*x y*. $x \in H \Longrightarrow y \in H \Longrightarrow$ *dist x y* < *d-Taylor* ⟹ *norm* (*R x y*) ≤ *e1* ∗ *dist x y*
continuous-on (*G* × *G*) (λ(*a*, *b*). *R a b*)
by *auto*

Find d, such that solutions are always at least *min* (*e-domain*/*2*) *d-Taylor* apart, i.e. always in H. This later gives us the bound on the remainder.

have *0* < *min* (*e-domain*/*2*) *d-Taylor*
using ‹*0* < *d-Taylor*› ‹*0* < *e-domain*›
by *auto*
from *uniform-limit-flow*[*OF* ‹*t0* ∈ *existence-ivl0 x0*› ‹*t1* ∈ *existence-ivl0 x0*› ‹*t0* ≤ *t1*›,
THEN uniform-limitD, *OF this*, *unfolded eventually-at*]
obtain *d-ivl* **where** *d-ivl-def*:

0 < d-ivl
⋀x. 0 < dist x x0 ⟹ dist x x0 < d-ivl ⟹
(∀ t∈J. dist (flow0 x0 t) (Y (x − x0) t) < min (e-domain / 2) d-Taylor)
by (*auto simp*: *dist-commute J-def*)

define *d* **where** *d ≡ min u d-ivl*
have *0 < d* **using** ‹*0 < u*› ‹*0 < d-ivl*›
by (*simp add*: *d-def*)
hence *d ≤ u d ≤ d-ivl*
by (*auto simp*: *d-def*)

Therefore, any flow0 starting in *ball x0 d* will be in G.

have *Y-in-G*: *⋀y. y ∈ ball x0 d ⟹ (λs. Y (y − x0) s) ‘ J ⊆ G*
proof
fix *x y* **assume** *assms*: *y ∈ ball x0 d x ∈ (λs. Y (y − x0) s) ‘ J*
show *x ∈ G*
proof(*cases*)
assume *y = x0*
from *assms*(*2*)
have *x ∈ flow0 x0 ‘ J*
by (*simp add*: ‹*y = x0*›)
thus *x ∈ G*
using ‹*0 < e-domain*› ‹*flow0 x0 ‘ J ⊆ X*›
by (*auto simp*: *G-def*)
next
assume *y ≠ x0*
hence *0 < dist y x0*
by (*simp add*: *dist-norm*)
from *d-ivl-def*(*2*)[*OF this*] ‹*d ≤ d-ivl*› ‹*0 < e-domain*› *assms*(*1*)
have *dist-flow0-Y*: *⋀t. t ∈ J ⟹ dist (flow0 x0 t) (Y (y − x0) t) <
e-domain*
by (*auto simp*: *dist-commute*)

from *assms*(*2*)
obtain *t* **where** *t-def*: *t ∈ J x = Y (y − x0) t*
by *auto*
have *x ∈ X*
unfolding *t-def*(*2*)
using *uK-def*(*3*) *assms*(*1*) ‹*d ≤ u*› *subsetD*[*OF J-in-existence-ivl
t-def*(*1*)]
by (*auto simp*: *intro*!: *flow-in-domain*)

have *flow0 x0 t ∈ flow0 x0 ‘ J* **using** *t-def* **by** *auto*
from *dist-flow0-Y*[*OF t-def*(*1*)]
have *dist x (flow0 x0 t) < e-domain*
by (*simp add*: *t-def*(*2*) *dist-commute*)
from *le-less-trans*[*OF infdist-le*[*OF* ‹*flow0 x0 t ∈ flow0 x0 ‘ J*›] *this*] ‹*x
∈ X*›
show *x ∈ G*

```
        by (auto simp: G-def)
      qed
    qed
  from this[of x0] ‹0 < d›
  have X-in-G: flow0 x0 ‘ J ⊆ G
    by simp

  show ∃d>0. ∀x. 0 < dist x x0 ∧ dist x x0 < d ⟶
            dist ((Y (x − x0) t − flow0 x0 t − vector-Dflow (x − x0) t) /R
norm (x − x0)) 0 < e
  proof(rule exI, rule conjI[OF ‹0 < d›], safe, unfold norm-conv-dist[symmetric])
    fix x assume x-x0-dist: 0 < dist x x0 dist x x0 < d
    hence x-in-ball': x ∈ ball x0 d
      by (simp add: dist-commute)
    hence x-in-ball: x ∈ ball x0 u
      using ‹d ≤ u›
      by simp
```

First, some prerequisites.

```
    from x-in-ball
    have z-in-ball: x − x0 ∈ ball 0 u
      using ‹0 < u›
      by (simp add: dist-norm)
    hence [continuous-intros]: dist x0 x < u
      by (auto simp: dist-norm)

    from J-in-existence-ivl[OF x-in-ball]
    have J-in-existence-ivl-x: J ⊆ existence-ivl0 x .
    from ball-in-X[OF z-in-ball]
    have x-in-X[continuous-intros]: x ∈ X
      by simp
```

On all of J, we can find upper bounds for the distance of *flow0* and Y.

```
    have dist-flow0-Y: ⋀s. s ∈ J ⟹ dist (flow0 x0 s) (Y (x − x0) s) ≤ dist
x0 x ∗ exp (K ∗ |t|)
      using t0-def t1-def uK-def(2)
      by (intro order-trans[OF uK-def(6)[OF x-in-ball] mult-left-mono])
        (auto simp add: J-def intro!: mult-mono)
    from d-ivl-def x-x0-dist ‹d ≤ d-ivl›
    have dist-flow0-Y2: ⋀t. t ∈ J ⟹ dist (flow0 x0 t) (Y (x − x0) t) < min
(e-domain2) d-Taylor
      by (auto simp: e-domain2-def)

    let ?g = λt. norm (Y (x − x0) t − flow0 x0 t − vector-Dflow (x − x0) t)
    let ?C = |t| ∗ dist x0 x ∗ exp (K ∗ |t|) ∗ e1
```

Find an upper bound to *?g*, i.e. show that *?g s* ≤ *?C* + *N* ∗ *integral* {*a..b*} *?g* for {*a..b*} = {*0..s*} or {*a..b*} = {*s..0*} for some $s \in J$. We can then apply Grönwall's inequality to obtain a true bound for *?g*.

have *g-bound*: *?g s* ≤ *?C* + *N* ∗ *integral* {*a..b*} *?g*
if *s-def*: *s* ∈ {*a..b*}
and *J′-def*: {*a..b*} ⊆ *J*
and *ab-cases*: (*a* = *0* ∧ *b* = *s*) ∨ (*a* = *s* ∧ *b* = *0*)
for *s a b*
proof −
from *that* **have** *s* ∈ *J* **by** *auto*

have *s-in-existence-ivl-x0*: *s* ∈ *existence-ivl0 x0*
using *J-in-existence* ‹*s* ∈ *J*› **by** *auto*
have *s-in-existence-ivl*: ⋀*y*. *y* ∈ *ball x0 u* ⟹ *s* ∈ *existence-ivl0 y*
using *J-in-existence-ivl* ‹*s* ∈ *J*› **by** *auto*
have *s-in-existence-ivl2*: ⋀*z*. *z* ∈ *ball 0 u* ⟹ *s* ∈ *existence-ivl0* (*x0* + *z*)
using *s-in-existence-ivl*
by (*simp add*: *dist-norm*)

Prove continuities beforehand.

note *continuous-on-0-s*[*continuous-intros*] = *continuous-on-subset*[*OF* - ‹{*a..b*} ⊆ *J*›]

have[*continuous-intros*]: *continuous-on J* (*flow0 x0*)
using *J-in-existence*
by (*auto intro*!: *continuous-intros simp*:)
{
fix *z S*
assume *assms*: *x0* + *z* ∈ *X S* ⊆ *existence-ivl0* (*x0* + *z*)
have *continuous-on S* (λ*s*. *f* (*Y z s*))
proof(*rule continuous-on-subset*[*OF* - *assms*(*2*)])
show *continuous-on* (*existence-ivl0* (*x0* + *z*)) (λ*s*. *f* (*Y z s*))
using *assms*
by (*auto intro*!: *continuous-intros flow-in-domain flow-continuous-on simp*:)
qed
}
note [*continuous-intros*] = *this*

have [*continuous-intros*]: *continuous-on J* (λ*s*. *f* (*flow0 x0 s*))
by(*rule continuous-on-subset*[*OF* - *J-in-existence*])
(*auto intro*!: *continuous-intros flow-continuous-on flow-in-domain simp*: *x0-def*)

have [*continuous-intros*]: ⋀*z*. *continuous-on J* (λ*s*. *f′* (*flow0 x0 s*) (*vector-Dflow z s*))
proof−
fix *z*
have *a1*: *continuous-on J* (*flow0 x0*)
by (*auto intro*!: *continuous-intros*)

have *a2*: (λ*s*. (*flow0 x0 s*, *vector-Dflow z s*)) ‘ *J* ⊆ (*flow0 x0* ‘ *J*) × ((λ*s*.

vector-Dflow z s) ‘ *J*)
by *auto*
have *a3*: *continuous-on* ((λ*s*. (*flow0 x0 s*, *vector-Dflow z s*)) ‘ *J*) (λ(*x*, *u*). *f′ x u*)
using *assms flow0-J-props*
by (*auto intro*!: *continuous-intros simp*: *split-beta′*)
from *continuous-on-compose*[*OF continuous-on-Pair*[*OF a1 U-continuous*] *a3*]
show *continuous-on J* (λ*s*. *f′* (*flow0 x0 s*) (*vector-Dflow z s*))
by *simp*
qed

have [*continuous-intros*]: *continuous-on J* (λ*s*. *R* (*flow0 x0 s*) (*Y* (*x* − *x0*) *s*))
using *J-in-existence J-in-existence-ivl*[*OF x-in-ball*] *X-in-G* ‹{*a*..*b*} ⊆ *J*› *Y-in-G*
x-x0-dist
by (*auto intro*!: *continuous-intros continuous-on-compose-Pair*[*OF Taylor-expansion*(*4*)]
simp: *dist-commute subset-iff*)
hence [*continuous-intros*]:
(λ*s*. *R* (*flow0 x0 s*) (*Y* (*x* − *x0*) *s*)) *integrable-on J*
unfolding *J-def*
by (*rule integrable-continuous-real*)

have *i1*: *integral* {*a*..*b*} (λ*s*. *f* (*flow0 x s*)) − *integral* {*a*..*b*} (λ*s*. *f* (*flow0 x0 s*)) =
integral {*a*..*b*} (λ*s*. *f* (*flow0 x s*) − *f* (*flow0 x0 s*))
using *J-in-existence-ivl*[*OF x-in-ball*]
apply (*intro Henstock-Kurzweil-Integration.integral-diff*[*symmetric*])
by (*auto intro*!: *continuous-intros existence-ivl-reverse*)
have *i2*:
integral {*a*..*b*} (λ*s*. *f* (*flow0 x s*) − *f* (*flow0 x0 s*) − (*f′* (*flow0 x0 s*)) (*vector-Dflow* (*x* − *x0*) *s*)) =
integral {*a*..*b*} (λ*s*. *f* (*flow0 x s*) − *f* (*flow0 x0 s*)) −
integral {*a*..*b*} (λ*s*. *f′* (*flow0 x0 s*) (*vector-Dflow* (*x* − *x0*) *s*))
using *J-in-existence-ivl*[*OF x-in-ball*]
by (*intro Henstock-Kurzweil-Integration.integral-diff*[*OF Henstock-Kurzweil-Integration.integrable-diff*])
(*auto intro*!: *continuous-intros existence-ivl-reverse*)
from *ab-cases*
have *?g s* = *norm* (*integral* {*a*..*b*} (λ*s′*. *f* (*Y* (*x* − *x0*) *s′*)) −
integral {*a*..*b*} (λ*s′*. *f* (*flow0 x0 s′*)) −
integral {*a*..*b*} (λ*s′*. (*f′* (*flow0 x0 s′*)) (*vector-Dflow* (*x* − *x0*) *s′*)))
proof(*safe*)
assume *a* = *0 b* = *s*
hence *0* ≤ *s* **using** ‹*s* ∈ {*a*..*b*}› **by** *simp*

Integral equations for flow0, Y and U.

have *flow0-integral-eq*: *flow0 x0 s* = *x0* + *ivl-integral 0 s* (λ*s*. *f* (*flow0*

x0 s))
    by (*rule flow-fixed-point*[*OF s-in-existence-ivl-x0*])
   have *Y-integral-eq*: *flow0 x s* = *x0* + (*x* − *x0*) + *ivl-integral 0 s* (λ*s*. *f* (*Y* (*x* − *x0*) *s*))
      using *flow-fixed-point* ‹*0* ≤ *s*› *s-in-existence-ivl2*[*OF z-in-ball*] *ball-in-X*[*OF z-in-ball*]
    by (*simp add*:)
   have *U-integral-eq*: *vector-Dflow* (*x* − *x0*) *s* = (*x* − *x0*) + *ivl-integral 0 s* (λ*s*. *vareq x0 s* (*vector-Dflow* (*x* − *x0*) *s*))
    unfolding *vector-Dflow-def*
    by (*rule var.flow-fixed-point*)
     (*auto simp*: ‹*0* ≤ *s*› *x0-def varexivl-eq-exivl s-in-existence-ivl-x0*)
   show *?g s* = *norm* (*integral* {*0*..*s*} (λ*s'*. *f* (*Y* (*x* − *x0*) *s'*)) −
     integral {*0*..*s*} (λ*s'*. *f* (*flow0 x0 s'*)) −
     integral {*0*..*s*} (λ*s'*. *blinfun-apply* (*f'* (*flow0 x0 s'*)) (*vector-Dflow* (*x* − *x0*) *s'*)))
    using ‹*0* ≤ *s*›
    unfolding *vareq-def*[*symmetric*]
   by (*simp add*: *flow0-integral-eq Y-integral-eq U-integral-eq ivl-integral-def*)
  next
   assume *a* = *s b* = *0*
   hence *s* ≤ *0* **using** ‹*s* ∈ {*a*..*b*}› **by** *simp*

   have *flow0-integral-eq-left*: *flow0 x0 s* = *x0* + *ivl-integral 0 s* (λ*s*. *f* (*flow0 x0 s*))
    by (*rule flow-fixed-point*[*OF s-in-existence-ivl-x0*])
   have *Y-integral-eq-left*: *Y* (*x* − *x0*) *s* = *x0* + (*x* − *x0*) + *ivl-integral 0 s* (λ*s*. *f* (*Y* (*x* − *x0*) *s*))
      using *flow-fixed-point* ‹*s* ≤ *0*› *s-in-existence-ivl2*[*OF z-in-ball*] *ball-in-X*[*OF z-in-ball*]
    by *simp*
   have *U-integral-eq-left*: *vector-Dflow* (*x* − *x0*) *s* = (*x* − *x0*) + *ivl-integral 0 s* (λ*s*. *vareq x0 s* (*vector-Dflow* (*x* − *x0*) *s*))
    unfolding *vector-Dflow-def*
    by (*rule var.flow-fixed-point*)
     (*auto simp*: ‹*s* ≤ *0*› *x0-def varexivl-eq-exivl s-in-existence-ivl-x0*)

   have *?g s* =
    norm (− *integral* {*s*..*0*} (λ*s'*. *f* (*Y* (*x* − *x0*) *s'*)) +
     integral {*s*..*0*} (λ*s'*. *f* (*flow0 x0 s'*)) +
     integral {*s*..*0*} (λ*s'*. *vareq x0 s'* (*vector-Dflow* (*x* − *x0*) *s'*)))
    unfolding *flow0-integral-eq-left Y-integral-eq-left U-integral-eq-left*
    using ‹*s* ≤ *0*›
    by (*simp add*: *ivl-integral-def*)
   also have ... = *norm* (*integral* {*s*..*0*} (λ*s'*. *f* (*Y* (*x* − *x0*) *s'*)) −
     integral {*s*..*0*} (λ*s'*. *f* (*flow0 x0 s'*)) −
     integral {*s*..*0*} (λ*s'*. *vareq x0 s'* (*vector-Dflow* (*x* − *x0*) *s'*)))
    by (*subst norm-minus-cancel*[*symmetric*], *simp*)
   finally show *?g s* =

norm (*integral* {*s*..*0*} (λ*s*′. *f* (*Y* (*x* − *x0*) *s*′)) −
integral {*s*..*0*} (λ*s*′. *f* (*flow0 x0 s*′)) −
integral {*s*..*0*} (λ*s*′. *blinfun-apply* (*f*′ (*flow0 x0 s*′)) (*vector-Dflow* (*x* − *x0*) *s*′)))
unfolding *vareq-def* **.**
qed
also have ... =
norm (*integral* {*a*..*b*} (λ*s*. *f* (*Y* (*x* − *x0*) *s*) − *f* (*flow0 x0 s*) − (*f*′ (*flow0 x0 s*)) (*vector-Dflow* (*x* − *x0*) *s*)))
by (*simp add*: *i1 i2*)
also have ... ≤
integral {*a*..*b*} (λ*s*. *norm* (*f* (*Y* (*x* − *x0*) *s*) − *f* (*flow0 x0 s*) − *f*′ (*flow0 x0 s*) (*vector-Dflow* (*x* − *x0*) *s*)))
using *x-in-X J-in-existence-ivl-x J-in-existence* ‹{*a*..*b*} ⊆ *J*›
by (*auto intro*!: *continuous-intros continuous-on-imp-absolutely-integrable-on existence-ivl-reverse*)
also have ... = *integral* {*a*..*b*}
(λ*s*. *norm* (*f*′ (*flow0 x0 s*) (*Y* (*x* − *x0*) *s* − *flow0 x0 s* − *vector-Dflow* (*x* − *x0*) *s*) + *R* (*flow0 x0 s*) (*Y* (*x* − *x0*) *s*)))
proof (*safe intro*!: *integral-spike*[*OF negligible-empty*, *simplified*] *arg-cong*[**where** *f*=*norm*])
fix *s*′ **assume** *s*′ ∈ {*a*..*b*}
show *f*′ (*flow0 x0 s*′) (*Y* (*x* − *x0*) *s*′ − *flow0 x0 s*′ − *vector-Dflow* (*x* − *x0*) *s*′) + *R* (*flow0 x0 s*′) (*Y* (*x* − *x0*) *s*′) =
f (*Y* (*x* − *x0*) *s*′) − *f* (*flow0 x0 s*′) − *f*′ (*flow0 x0 s*′) (*vector-Dflow* (*x* − *x0*) *s*′)
by (*simp add*: *blinfun.diff-right Taylor-expansion*(*2*)[*of flow0 x s*′ *flow0 x0 s*′])
qed
also have ... ≤ *integral* {*a*..*b*}
(λ*s*. *norm* (*f*′ (*flow0 x0 s*) (*Y* (*x* − *x0*) *s* − *flow0 x0 s* − *vector-Dflow* (*x* − *x0*) *s*)) +
norm (*R* (*flow0 x0 s*) (*Y* (*x* − *x0*) *s*)))
using *J-in-existence-ivl*[*OF x-in-ball*] *norm-triangle-ineq*
using ‹*continuous-on J* (λ*s*. *R* (*flow0 x0 s*) (*Y* (*x* − *x0*) *s*))›
by (*auto intro*!: *continuous-intros integral-le*)
also have ... =
integral {*a*..*b*} (λ*s*. *norm* (*f*′ (*flow0 x0 s*) (*Y* (*x* − *x0*) *s* − *flow0 x0 s* − *vector-Dflow* (*x* − *x0*) *s*))) +
integral {*a*..*b*} (λ*s*. *norm* (*R* (*flow0 x0 s*) (*Y* (*x* − *x0*) *s*)))
using *J-in-existence-ivl*[*OF x-in-ball*]
using ‹*continuous-on J* (λ*s*. *R* (*flow0 x0 s*) (*Y* (*x* − *x0*) *s*))›
by (*auto intro*!: *continuous-intros Henstock-Kurzweil-Integration.integral-add*)
also have ... ≤ *N* ∗ *integral* {*a*..*b*} *?g* + *?C* (**is** *?l1* + *?r1* ≤ -)
proof(*rule add-mono*)
have *?l1* ≤ *integral* {*a*..*b*} (λ*s*. *norm* (*f*′ (*flow0 x0 s*)) ∗ *norm* (*Y* (*x* − *x0*) *s* − *flow0 x0 s* − *vector-Dflow* (*x* − *x0*) *s*))
using *norm-blinfun J-in-existence-ivl*[*OF x-in-ball*]
by (*auto intro*!: *continuous-intros integral-le*)

also have ... ≤ *integral* {*a..b*} (*λs. N* ∗ *norm* (*Y* (*x* − *x0*) *s* − *flow0 x0 s* − *vector-Dflow* (*x* − *x0*) *s*))
using *J-in-existence-ivl*[*OF x-in-ball*] *N-ineq*[*OF* ‹{*a..b*} ⊆ *J*›[*THEN subsetD*]]
by (*intro integral-le*) (*auto intro*!: *continuous-intros mult-right-mono*)

also have ... = *N* ∗ *integral* {*a..b*} (*λs. norm* ((*Y* (*x* − *x0*) *s* − *flow0 x0 s* − *vector-Dflow* (*x* − *x0*) *s*)))
unfolding *real-scaleR-def*[*symmetric*]
by(*rule integral-cmul*)
finally show *?l1* ≤ *N* ∗ *integral* {*a..b*} *?g* **.**
next
have *?r1* ≤ *integral* {*a..b*} (*λs. e1* ∗ *dist* (*flow0 x0 s*) (*Y* (*x* − *x0*) *s*))
using *J-in-existence-ivl*[*OF x-in-ball*] ‹*0* < *e-domain*› *dist-flow0-Y2* ‹*0* < *e-domain2*›
by (*intro integral-le*)
(*force*
intro!: *continuous-intros Taylor-expansion*(*3*) *order-trans*[*OF infdist-le*]
dest!: ‹{*a..b*} ⊆ *J*›[*THEN subsetD*]
intro: *less-imp-le*
simp: *dist-commute H-def*)+
also have ... ≤ *integral* {*a..b*} (*λs. e1* ∗ (*dist x0 x* ∗ *exp* (*K* ∗ |*t*|)))
apply(*rule integral-le*)
subgoal using *J-in-existence-ivl*[*OF x-in-ball*] **by** (*force intro*!: *continuous-intros*)
subgoal by *force*
subgoal by (*force dest*!: ‹{*a..b*} ⊆ *J*›[*THEN subsetD*]
intro!: *less-imp-le*[*OF* ‹*0* < *e1*›] *mult-left-mono*[*OF dist-flow0-Y*])
done
also have ... ≤ *?C*
using ‹*s* ∈ *J*› *x-x0-dist* ‹*0* < *e1*› ‹{*a..b*} ⊆ *J*› ‹*0* < |*t*|› *t0-def t1-def*
by (*auto simp*: *integral-const-real J-def*(*1*))
finally show *?r1* ≤ *?C* **.**
qed
finally show *?thesis*
by *simp*
qed
have *g-continuous*: *continuous-on J ?g*
using *J-in-existence-ivl*[*OF x-in-ball*] *J-in-existence*
using *J-def*(*1*) *U-continuous*
by (*auto simp*: *J-def intro*!: *continuous-intros*)
note [*continuous-intros*] = *continuous-on-subset*[*OF g-continuous*]
have *C-gr-zero*: *0* < *?C*
using ‹*0* < |*t*|› ‹*0* < *e1*› *x-x0-dist*(*1*)
by (*simp add*: *dist-commute*)
have *0* ≤ *t* ∨ *t* ≤ *0* **by** *auto*
then have *?g t* ≤ *?C* ∗ *exp* (*N* ∗ |*t*|)
proof

assume *0* ≤ *t*
moreover
have *continuous-on* {*0*..*t*} (*vector-Dflow* (*x* − *x0*))
using *U-continuous*
by (*rule continuous-on-subset*) (*auto simp*: *J-def t0-def t1-def*)
then have *norm* (*Y* (*x* − *x0*) *t* − *flow0 x0 t* − *vector-Dflow* (*x* − *x0*) *t*)
≤
|*t*| ∗ *dist x0 x* ∗ *exp* (*K* ∗ |*t*|) ∗ *e1* ∗ *exp* (*N* ∗ *t*)
using ‹*t* ∈ *J*› *J-def* ‹*t0* ≤ *0*› *J-in-existence J-in-existence-ivl-x*
by (*intro gronwall*[*OF g-bound* - - *C-gr-zero* ‹*0* < *N*› ‹*0* ≤ *t*› *order*.*refl*])
(*auto intro*!: *continuous-intros simp*:)
ultimately show *?thesis* **by** *simp*
next
assume *t* ≤ *0*
moreover
have *continuous-on* {*t* .. *0*} (*vector-Dflow* (*x* − *x0*))
using *U-continuous*
by (*rule continuous-on-subset*) (*auto simp*: *J-def t0-def t1-def*)
then have *norm* (*Y* (*x* − *x0*) *t* − *flow0 x0 t* − *vector-Dflow* (*x* − *x0*) *t*)
≤
|*t*| ∗ *dist x0 x* ∗ *exp* (*K* ∗ |*t*|) ∗ *e1* ∗ *exp* (− *N* ∗ *t*)
using ‹*t* ∈ *J*› *J-def* ‹*0* ≤ *t1*› *J-in-existence J-in-existence-ivl-x*
by (*intro gronwall-left*[*OF g-bound* - - *C-gr-zero* ‹*0* < *N*› *order*.*refl* ‹*t* ≤
0›])
(*auto intro*!: *continuous-intros*)
ultimately show *?thesis*
by *simp*
qed
also have ... = *dist x x0* ∗ (|*t*| ∗ *exp* (*K* ∗ |*t*|) ∗ *e1* ∗ *exp* (*N* ∗ |*t*|))
by (*auto simp*: *dist-commute*)
also have ... < *norm* (*x* − *x0*) ∗ *e*
unfolding *e1-def*
using ‹*e′* < *e*› ‹*0* < |*t*|› ‹*0* < *e1*› *x-x0-dist*(*1*)
by (*simp add*: *dist-norm*)
finally show *norm* ((*Y* (*x* − *x0*) *t* − *flow0 x0 t* − *vector-Dflow* (*x* − *x0*) *t*) /$_R$ *norm* (*x* − *x0*)) < *e*
by (*simp*, *metis x-x0-dist*(*1*) *dist-norm divide-inverse mult*.*commute pos-divide-less-eq*)
qed
qed
qed
qed

lemma *local-lipschitz-A*:
OT ⊆ *existence-ivl0 x0* ⟹ *local-lipschitz OT* (*OS*::(*′a* ⇒$_L$ *′a*) *set*) (λ*t*. (*o*$_L$) (*vareq x0 t*))
by (*rule local-lipschitz-subset*[*OF* - - *subset-UNIV*, **where** *T*=*existence-ivl0 x0*])
(*auto simp*: *split-beta′ vareq-def*
intro!: *c1-implies-local-lipschitz*[**where** *f′*=λ(*t*, *x*). *comp3* (*f′* (*flow0 x0 t*))]

derivative-eq-intros blinfun-eqI ext
continuous-intros flow-in-domain)

lemma *total-derivative-ll-on-open*:
ll-on-open (*existence-ivl0 x0*) (λt. *blinfun-compose* (*vareq x0 t*)) (*UNIV*::($'a \Rightarrow_L$ $'a$) *set*)
by *standard* (*auto intro*!: *continuous-intros local-lipschitz-A*[*OF order-refl*])

end

end

sublocale *mvar*: *ll-on-open existence-ivl0 x0* λt. *blinfun-compose* (*vareq x0 t*) *UNIV*::($'a$ $\Rightarrow_L$ $'a$) *set* **for** *x0*
by (*rule total-derivative-ll-on-open*)

lemma *mvar-existence-ivl-eq-existence-ivl*[*simp*]:— TODO: unify with *?t* ∈ *existence-ivl0 ?x0.0* ⟹ *var.existence-ivl ?x0.0 ?t ?a* = *existence-ivl0 ?x0.0*
assumes *t* ∈ *existence-ivl0 x0*
shows *mvar.existence-ivl x0 t* = (λ-. *existence-ivl0 x0*)
proof (*rule ext*, *rule mvar.existence-ivl-eq-domain*)
fix *s t x*
assume *s*: *s* ∈ *existence-ivl0 x0* **and** *t*: *t* ∈ *existence-ivl0 x0*
then have {*s* .. *t*} ⊆ *existence-ivl0 x0*
by (*meson atLeastAtMost-iff is-interval-1 is-interval-existence-ivl subsetI*)
then have *continuous-on* {*s* .. *t*} (*vareq x0*)
by (*auto intro*!: *continuous-intros*)
then have *compact* (*vareq x0* ‘ {*s* .. *t*})
using *compact-Icc*
by (*rule compact-continuous-image*)
then obtain *B* **where** *B*: ⋀*u*. *u* ∈ {*s* .. *t*} ⟹ *norm* (*vareq x0 u*) ≤ *B*
by (*force dest*!: *compact-imp-bounded simp*: *bounded-iff*)
show ∃ *M L*. ∀ *t*∈{*s* .. *t*}. ∀ *x*∈*UNIV*. *norm* (*vareq x0 t* o_L *x*) ≤ *M* + *L* ∗ *norm* *x*
unfolding *o-def*
by (*rule exI*[**where** *x=0*], *rule exI*[**where** *x=B*])
(*auto intro*!: *order-trans*[*OF norm-blinfun-compose*] *mult-right-mono B*)
qed (*auto intro*: *assms*)

lemma
assumes *t* ∈ *existence-ivl0 x0*
shows *continuous-on* (*UNIV* × *existence-ivl0 x0*) (λ(*x*, *ta*). *mvar.flow x0 t x ta*)
proof −
from *mvar.flow-continuous-on-state-space*[*of x0 t*, *unfolded mvar-existence-ivl-eq-existence-ivl*[*OF assms*]]
show *continuous-on* (*UNIV* × *existence-ivl0 x0*) (λ(*x*, *ta*). *mvar.flow x0 t x ta*)
.
qed

definition *Dflow x0 = mvar.flow x0 0 id-blinfun*

lemma *var-eq-mvar*:
  assumes *t0* ∈ *existence-ivl0 x0*
  assumes *t* ∈ *existence-ivl0 x0*
  shows *var.flow x0 t0 i t = mvar.flow x0 t0 id-blinfun t i*
  by (*rule var.flow-unique*)
    (*auto intro*!: *assms derivative-eq-intros mvar.flow-has-derivative*
      simp: *varexivl-eq-exivl assms has-vector-derivative-def blinfun.bilinear-simps*)

lemma *Dflow-zero*[*simp*]: *x* ∈ *X* ⟹ *Dflow x 0 = 1_L*
  unfolding *Dflow-def*
  by (*subst mvar.flow-initial-time*) *auto*

5.3 Differentiability of the flow0

U t, i.e. the solution of the variational equation, is the space derivative at the initial value *x0*.

lemma *flow-dx-derivative*:
assumes *t* ∈ *existence-ivl0 x0*
shows ((λ*x0*. *flow0 x0 t*) *has-derivative* (λ*z*. *vector-Dflow x0 z t*)) (*at x0*)
  unfolding *has-derivative-at2*
  using *assms*
  by (*intro iffD1*[*OF LIM-equal proposition-17-6-weak*[*OF assms*]] *conjI*[*OF bounded-linear-vector-Dflow*[*OF assms*]])
    (*simp add*: *diff-diff-add inverse-eq-divide*)

lemma *flow-dx-derivative-blinfun*:
assumes *t* ∈ *existence-ivl0 x0*
shows ((λ*x*. *flow0 x t*) *has-derivative Blinfun* (λ*z*. *vector-Dflow x0 z t*)) (*at x0*)
by (*rule has-derivative-Blinfun*[*OF flow-dx-derivative*[*OF assms*]])

definition *flowderiv x0 t = comp12* (*Dflow x0 t*) (*blinfun-scaleR-left* (*f* (*flow0 x0 t*)))

lemma *flowderiv-eq*: *flowderiv x0 t* (ξ_1, ξ_2) = (*Dflow x0 t*) ξ_1 + ξ_2 $*_R$ *f* (*flow0 x0 t*)
  by (*auto simp*: *flowderiv-def*)

lemma *W-continuous-on*: *continuous-on* (*Sigma X existence-ivl0*) (λ(*x0*, *t*). *Dflow x0 t*)
  — TODO: somewhere here is hidden continuity wrt rhs of ODE, extract it!
  unfolding *continuous-on split-beta′*
proof (*safe intro*!: *tendstoI*)
  fix *e′*::*real* **and** *t x* **assume** *x*: *x* ∈ *X* **and** *tx*: *t* ∈ *existence-ivl0 x* **and** *e′*: *e′* > *0*
  let *?S = Sigma X existence-ivl0*

  have (*x*, *t*) ∈ *?S* **using** *x tx* **by** *auto*
  from *open-prod-elim*[*OF open-state-space this*]

obtain *OX OT* **where** *OXOT*: *open OX open OT* $(x, t) \in OX \times OT$ $OX \times OT \subseteq$ *?S*
by *blast*
then obtain *dx dt*
where *dx*: *dx* $>$ *0 cball x dx* $\subseteq$ *OX*
and *dt*: *dt* $>$ *0 cball t dt* $\subseteq$ *OT*
by (*force simp*: *open-contains-cball*)

from *OXOT dt dx* **have** *cball t dt* $\subseteq$ *existence-ivl0 x cball x dx* $\subseteq$ *X*
apply (*auto simp*: *subset-iff*)
subgoal for *ta*
apply (*drule spec*[**where** *x=ta*])
apply (*drule spec*[**where** *x=t*])+
apply *auto*
done
done

have *one-exivl*: *mvar.existence-ivl x 0* = (λ-. *existence-ivl0 x*)
by (*rule mvar-existence-ivl-eq-existence-ivl*[*OF existence-ivl-zero*[*OF* ‹*x* $\in$ *X*›]])

have $*$: *closed* ({*t* .. *0*} $\cup$ {*0* .. *t*}) {*t* .. *0*} $\cup$ {*0* .. *t*} $\neq$ {}
by *auto*
let *?T* = {*t* .. *0*} $\cup$ {*0* .. *t*} $\cup$ *cball t dt*
have *compact ?T*
by (*auto intro*!: *compact-Un*)
have *?T* $\subseteq$ *existence-ivl0 x*
by (*intro Un-least ivl-subset-existence-ivl′ ivl-subset-existence-ivl* ‹*x* $\in$ *X*›
‹*t* $\in$ *existence-ivl0 x*› ‹*cball t dt* $\subseteq$ *existence-ivl0 x*›)

have *compact* (*mvar.flow x 0 id-blinfun* ‘ *?T*)
using ‹*?T* $\subseteq$ -› ‹*x* $\in$ *X*›
mvar-existence-ivl-eq-existence-ivl[*OF existence-ivl-zero*[*OF* ‹*x* $\in$ *X*›]]
by (*auto intro*!: ‹*0* $<$ *dx*› *compact-continuous-image* ‹*compact ?T*›
continuous-on-subset[*OF mvar.flow-continuous-on*])

let *?line* = *mvar.flow x 0 id-blinfun* ‘ *?T*
let *?X* = {*x*. *infdist x ?line* $\leq$ *dx*}
have *compact ?X*
using ‹*?T* $\subseteq$ -› ‹*x* $\in$ *X*›
mvar-existence-ivl-eq-existence-ivl[*OF existence-ivl-zero*[*OF* ‹*x* $\in$ *X*›]]
by (*auto intro*!: *compact-infdist-le* ‹*0* $<$ *dx*› *compact-continuous-image compact-Un*
continuous-on-subset[*OF mvar.flow-continuous-on*])

from *mvar.local-lipschitz* ‹*?T* $\subseteq$ -›
have *llc*: *local-lipschitz ?T ?X* (λ*t*. (o_L) (*vareq x t*))
by (*rule local-lipschitz-subset*) *auto*

have *cont*: $\bigwedge$*xa*. *xa* $\in$ *?X* $\Longrightarrow$ *continuous-on ?T* (λ*t*. *vareq x t* o_L *xa*)

using ‹?T ⊆ -›
by (*auto intro*!: *continuous-intros* ‹x ∈ X›)

from *local-lipschitz-compact-implies-lipschitz*[*OF llc* ‹*compact ?X*› ‹*compact ?T*› *cont*]
obtain K′ **where** K′: ⋀*ta*. *ta* ∈ ?T ⟹ K′−*lipschitz-on* ?X ((o_L) (*vareq x ta*))
by *blast*
define K **where** K ≡ *abs* K′ + 1
have K > 0
by (*simp add*: *K-def*)
have K: ⋀*ta*. *ta* ∈ ?T ⟹ K−*lipschitz-on* ?X ((o_L) (*vareq x ta*))
by (*auto intro*!: *lipschitz-onI mult-right-mono order-trans*[*OF lipschitz-onD*[*OF K′*]] *simp*: *K-def*)

have *ex-ivlI*: ⋀*y*. *y* ∈ *cball x dx* ⟹ ?T ⊆ *existence-ivl0 y*
using *dx dt OXOT*
by (*intro Un-least ivl-subset-existence-ivl′ ivl-subset-existence-ivl*; *force*)

have *cont*: *continuous-on* ((?T × ?X) × *cball x dx*) (λ((*ta*, *xa*), *y*). (*vareq y ta* o_L *xa*))
using ‹*cball x dx* ⊆ X› *ex-ivlI*
by (*force intro*!: *continuous-intros simp*: *split-beta′*)

have *mvar.flow x 0 id-blinfun t* ∈ *mvar.flow x 0 id-blinfun* ‘ ({*t*..0} ∪ {0..*t*} ∪ *cball t dt*)
by *auto*
then have *mem*: (*t*, *mvar.flow x 0 id-blinfun t*, *x*) ∈ ?T × ?X × *cball x dx*
by (*auto simp*: ‹0 < *dx*› *less-imp-le*)

define e **where** e ≡ *min* e′ (*dx* / 2) / 2
have e > 0 **using** ‹e′ > 0› **by** (*auto simp*: *e-def* ‹0 < *dx*›)
define d **where** d ≡ e ∗ K / (*exp* (K ∗ (*abs t* + *abs dt* + 1)) − 1)
have d > 0 **by** (*auto simp*: *d-def intro*!: *mult-pos-pos divide-pos-pos* ‹0 < e› ‹K > 0›)

have *cmpct*: *compact* ((?T × ?X) × *cball x dx*) *compact* (?T × ?X)
using ‹*compact ?T*› ‹*compact ?X*›
by (*auto intro*!: *compact-cball compact-Times*)

have *compact-line*: *compact ?line*
using ‹{*t*..0} ∪ {0..*t*} ∪ *cball t dt* ⊆ *existence-ivl0 x*› *one-exivl*
by (*force intro*!: *compact-continuous-image* ‹*compact ?T*› *continuous-on-subset*[*OF mvar.flow-continuous-on*] *simp*: ‹x ∈ X›)

from *compact-uniformly-continuous*[*OF cont cmpct*(1), *unfolded uniformly-continuous-on-def*, *rule-format*, *OF* ‹0 < d›]
obtain d′ **where** d′: d′ > 0
⋀*ta xa xa′ y*. *ta* ∈ ?T ⟹ *xa* ∈ ?X ⟹ *xa′*∈*cball x dx* ⟹ *y*∈*cball x dx* ⟹ *dist xa′ y* < d′ ⟹

dist (*vareq xa′ ta* o_L *xa*) (*vareq y ta* o_L *xa*) < *d*
by (*auto simp*: *dist-prod-def*)
{
fix *y*
assume *dxy*: *dist x y* < *d′*
assume *y* ∈ *cball x dx*
then have *y* ∈ *X*
using *dx dt OXOT* **by** *force*+

have *two-exivl*: *mvar.existence-ivl y 0* = (λ-. *existence-ivl0 y*)
by (*rule mvar-existence-ivl-eq-existence-ivl*[*OF existence-ivl-zero*[*OF* ‹*y* ∈ *X*›]])

let *?X′* = ⋃*x* ∈ *?line*. *ball x dx*
have *open ?X′* **by** *auto*
have *?X′* ⊆ *?X*
by (*auto intro*!: *infdist-le2 simp*: *dist-commute*)

interpret *oneR*: *ll-on-open existence-ivl0 x* (λ*t*. (o_L) (*vareq x t*)) *?X′*
by *standard* (*auto intro*!: ‹*x* ∈ *X*› *continuous-intros local-lipschitz-A*[*OF order-refl*])
interpret *twoR*: *ll-on-open existence-ivl0 y* (λ*t*. (o_L) (*vareq y t*)) *?X′*
by *standard* (*auto intro*!: ‹*y* ∈ *X*› *continuous-intros local-lipschitz-A*[*OF order-refl*])
interpret *both*:
two-ll-on-open (λ*t*. (o_L) (*vareq x t*)) *existence-ivl0 x* (λ*t*. (o_L) (*vareq y t*)) *existence-ivl0 y ?X′ ?T id-blinfun d K*
proof *unfold-locales*
show *0* < *K* **by** (*simp add*: ‹*0* < *K*›)
show *iv-defined*: *0* ∈ {*t*..*0*} ∪ {*0*..*t*} ∪ *cball t dt*
by *auto*
show *is-interval* ({*t*..*0*} ∪ {*0*..*t*} ∪ *cball t dt*)
by (*auto simp*: *is-interval-def dist-real-def*)
show {*t*..*0*} ∪ {*0*..*t*} ∪ *cball t dt* ⊆ *oneR.existence-ivl 0 id-blinfun*
apply (*rule oneR.maximal-existence-flow*[**where** *x*=*mvar.flow x 0 id-blinfun*])
subgoal
apply (*rule solves-odeI*)
apply (*rule has-vderiv-on-subset*[*OF solves-odeD*(*1*)[*OF mvar.flow-solves-ode*[*of 0 x id-blinfun*]]])
subgoal using ‹*x* ∈ *X*› ‹*?T* ⊆ -› ‹*0* < *dx*› **by** *simp*
subgoal by *simp*
subgoal by (*simp add*: ‹*cball t dt* ⊆ *existence-ivl0 x*› *ivl-subset-existence-ivl ivl-subset-existence-ivl′ one-exivl tx*)
subgoal using *dx* **by** (*auto*; *force*)
done
subgoal by (*simp add*: ‹*x* ∈ *X*›)
subgoal by *fact*
subgoal using *iv-defined* **by** *blast*
subgoal using ‹{*t*..*0*} ∪ {*0*..*t*} ∪ *cball t dt* ⊆ *existence-ivl0 x*› **by** *blast*
done

```
      fix s assume s: s ∈ ?T
      then show K−lipschitz-on ?X′ ((o_L) (vareq x s))
        by (intro lipschitz-on-subset[OF K ‹?X′ ⊆ ?X›]) auto
      fix j assume j: j ∈ ?X′
      show norm ((vareq x s o_L j) − (vareq y s o_L j)) < d
        unfolding dist-norm[symmetric]
        apply (rule d′)
        subgoal by (rule s)
        subgoal using ‹?X′ ⊆ ?X› j ..
        subgoal using ‹dx > 0› by simp
        subgoal using ‹y ∈ cball x dx› by simp
        subgoal using dxy by simp
        done
    qed
    have less-e: norm (Dflow x s − both.Y s) < e
      if s: s ∈ ?T ∩ twoR.existence-ivl 0 id-blinfun for s
    proof −
      from s have s-less: |s| < |t| + |dt| + 1
        by (auto simp: dist-real-def)
      note both.norm-X-Y-bound[rule-format, OF s]
      also have d / K ∗ (exp (K ∗ |s|) − 1) =
          e ∗ ((exp (K ∗ |s|) − 1) / (exp (K ∗ (|t| + |dt| + 1)) − 1))
        by (simp add: d-def)
      also have . . . < e ∗ 1
        by (rule mult-strict-left-mono[OF - ‹0 < e›])
          (simp add: add-nonneg-pos ‹0 < K› ‹0 < e› s-less)
      also have . . . = e by simp
      also
      from s have s: s ∈ ?T by simp
      have both.flow0 s = Dflow x s
        unfolding both.flow0-def Dflow-def
        apply (rule oneR.maximal-existence-flow[where K=?T])
        subgoal
          apply (rule solves-odeI)
         apply (rule has-vderiv-on-subset[OF solves-odeD(1)[OF mvar.flow-solves-ode[of
0 x id-blinfun]]])
          subgoal using ‹x ∈ X› ‹0 < dx› by simp
          subgoal by simp
         subgoal by (simp add: ‹cball t dt ⊆ existence-ivl0 x› ivl-subset-existence-ivl
ivl-subset-existence-ivl′ one-exivl tx)
          subgoal using dx by (auto; force)
          done
        subgoal by (simp add: ‹x ∈ X›)
        subgoal by (rule both.J-ivl)
        subgoal using both.t0-in-J by blast
        subgoal using ‹{t..0} ∪ {0..t} ∪ cball t dt ⊆ existence-ivl0 x› by blast
        subgoal using s by blast
        done
      finally show ?thesis .
```

```
  qed

  have e < dx using ‹dx > 0› by (auto simp: e-def)

  let ?i = {y. infdist y (mvar.flow x 0 id-blinfun ` ?T) ≤ e}
  have 1: ?i ⊆ (⋃x∈mvar.flow x 0 id-blinfun ` ?T. ball x dx)
  proof –
    have cl: closed ?line ?line ≠ {} using compact-line
      by (auto simp: compact-imp-closed)
    have ?i ⊆ (⋃y∈mvar.flow x 0 id-blinfun ` ?T. cball y e)
    proof safe
      fix x
      assume H: infdist x ?line ≤ e
      from infdist-attains-inf[OF cl, of x]
      obtain y where y ∈ ?line infdist x ?line = dist x y by auto
      then show x ∈ (⋃x∈?line. cball x e)
        using H
        by (auto simp: dist-commute)
    qed
    also have ... ⊆ (⋃x∈?line. ball x dx)
      using ‹e < dx›
      by auto
    finally show ?thesis .
  qed
  have 2: twoR.flow 0 id-blinfun s ∈ ?i
    if s ∈ ?T s ∈ twoR.existence-ivl 0 id-blinfun for s
  proof –
    from that have sT: s ∈ ?T ∩ twoR.existence-ivl 0 id-blinfun
      by force
    from less-e[OF this]
    have dist (twoR.flow 0 id-blinfun s) (mvar.flow x 0 id-blinfun s) ≤ e
      unfolding Dflow-def both.Y-def dist-commute dist-norm by simp
    then show ?thesis
      using sT by (force intro: infdist-le2)
  qed
  have T-subset: ?T ⊆ twoR.existence-ivl 0 id-blinfun
    apply (rule twoR.subset-mem-compact-implies-subset-existence-interval[
        where K={x. infdist x ?line ≤ e}])
    subgoal using ‹0 < dt› by force
    subgoal by (rule both.J-ivl)
    subgoal using ‹y ∈ cball x dx› ex-ivlI by blast
    subgoal using both.F-iv-defined(2) by blast
    subgoal by (rule 2)
   subgoal using ‹dt > 0› by (intro compact-infdist-le) (auto intro!: compact-line
‹0 < e›)
    subgoal by (rule 1)
    done
  also have twoR.existence-ivl 0 id-blinfun ⊆ existence-ivl0 y
    by (rule twoR.existence-ivl-subset)
```

finally have *?T* ⊆ *existence-ivl0 y* **.**
have *norm* (*Dflow x s* − *Dflow y s*) < *e* **if** *s*: *s* ∈ *?T* **for** *s*
proof −
from *s* **have** *s* ∈ *?T* ∩ *twoR.existence-ivl 0 id-blinfun* **using** *T-subset* **by** *force*
from *less-e*[*OF this*] **have** *norm* (*Dflow x s* − *both.Y s*) < *e* **.**
also have *mvar.flow y 0 id-blinfun s* = *twoR.flow 0 id-blinfun s*
apply (*rule mvar.maximal-existence-flow*[**where** *K*=*?T*])
subgoal
apply (*rule solves-odeI*)
apply (*rule has-vderiv-on-subset*[*OF solves-odeD*(*1*)[*OF twoR.flow-solves-ode*[*of 0 id-blinfun*]]])
subgoal using ‹*y* ∈ *X*› **by** *simp*
subgoal using *both.F-iv-defined*(*2*) **by** *blast*
subgoal using *T-subset* **by** *blast*
subgoal by *simp*
done
subgoal using ‹*y* ∈ *X*› *auto-ll-on-open.existence-ivl-zero auto-ll-on-open-axioms both.F-iv-defined*(*2*) *twoR.flow-initial-time* **by** *blast*
subgoal by (*rule both.J-ivl*)
subgoal using *both.t0-in-J* **by** *blast*
subgoal using ‹{*t..0*} ∪ {*0..t*} ∪ *cball t dt* ⊆ *existence-ivl0 y*› **by** *blast*
subgoal using *s* **by** *blast*
done
then have *both.Y s* = *Dflow y s*
unfolding *both.Y-def Dflow-def*
by *simp*
finally show *?thesis* **.**
qed
} **note** *cont-data* = *this*
have ∀$_F$ (*y*, *s*) *in at* (*x*, *t*) *within ?S*. *dist x y* < *d′*
unfolding *at-within-open*[*OF* ‹(*x*, *t*) ∈ *?S*› *open-state-space*] *UNIV-Times-UNIV*[*symmetric*]
using ‹*d′* > *0*›
by (*intro eventually-at-Pair-within-TimesI1*)
(*auto simp*: *eventually-at less-imp-le dist-commute*)
moreover
have ∀$_F$ (*y*, *s*) *in at* (*x*, *t*) *within ?S*. *y* ∈ *cball x dx*
unfolding *at-within-open*[*OF* ‹(*x*, *t*) ∈ *?S*› *open-state-space*] *UNIV-Times-UNIV*[*symmetric*]
using ‹*dx* > *0*›
by (*intro eventually-at-Pair-within-TimesI1*)
(*auto simp*: *eventually-at less-imp-le dist-commute*)
moreover
have ∀$_F$ (*y*, *s*) *in at* (*x*, *t*) *within ?S*. *s* ∈ *?T*
unfolding *at-within-open*[*OF* ‹(*x*, *t*) ∈ *?S*› *open-state-space*] *UNIV-Times-UNIV*[*symmetric*]
using ‹*dt* > *0*›
by (*intro eventually-at-Pair-within-TimesI2*)
(*auto simp*: *eventually-at less-imp-le dist-commute*)
moreover
have *0* ∈ *existence-ivl0 x* **by** (*simp add*: ‹*x* ∈ *X*›)

have $\forall_F$ *y in at t within existence-ivl0 x. dist* (*mvar.flow x 0 id-blinfun y*) (*mvar.flow x 0 id-blinfun t*) < *e*
using *mvar.flow-continuous-on*[*of x 0 id-blinfun*]
using ‹*0* < *e*› *tx*
by (*auto simp add*: *continuous-on one-exivl dest*!: *tendstoD*)
then have $\forall_F$ (*y*, *s*) *in at* (*x*, *t*) *within ?S*. *dist* (*Dflow x s*) (*Dflow x t*) < *e*
using ‹*0* < *e*›
unfolding *at-within-open*[*OF* ‹(*x*, *t*) ∈ *?S*› *open-state-space*] *UNIV-Times-UNIV*[*symmetric*] *Dflow-def*
by (*intro eventually-at-Pair-within-TimesI2*)
(*auto simp*: *at-within-open*[*OF tx open-existence-ivl*])
ultimately
have $\forall_F$ (*y*, *s*) *in at* (*x*, *t*) *within ?S*. *dist* (*Dflow y s*) (*Dflow x t*) < *e′*
apply *eventually-elim*
proof (*safe del*: *UnE*, *goal-cases*)
case (*1 y s*)
have *dist* (*Dflow y s*) (*Dflow x t*) ≤ *dist* (*Dflow y s*) (*Dflow x s*) + *dist* (*Dflow x s*) (*Dflow x t*)
by (*rule dist-triangle*)
also
have *dist* (*Dflow x s*) (*Dflow x t*) < *e*
by (*rule 1*)
also have *dist* (*Dflow y s*) (*Dflow x s*) < *e*
unfolding *dist-norm norm-minus-commute*
using *1*
by (*intro cont-data*)
also have *e* + *e* ≤ *e′* **by** (*simp add*: *e-def*)
finally show *dist* (*Dflow y s*) (*Dflow x t*) < *e′* **by** *arith*
qed
then show $\forall_F$ *ys in at* (*x*, *t*) *within ?S*. *dist* (*Dflow* (*fst ys*) (*snd ys*)) (*Dflow* (*fst* (*x*, *t*)) (*snd* (*x*, *t*))) < *e′*
by (*simp add*: *split-beta′*)
qed

lemma *W-continuous-on-comp*[*continuous-intros*]:
assumes *h*: *continuous-on S h* **and** *g*: *continuous-on S g*
shows (⋀*s*. *s* ∈ *S* ⟹ *h s* ∈ *X*) ⟹ (⋀*s*. *s* ∈ *S* ⟹ *g s* ∈ *existence-ivl0* (*h s*)) ⟹
continuous-on S (λ*s*. *Dflow* (*h s*) (*g s*))
using *continuous-on-compose*[*OF continuous-on-Pair*[*OF h g*] *continuous-on-subset*[*OF W-continuous-on*]]
by *auto*

lemma *f-flow-continuous-on*: *continuous-on* (*Sigma X existence-ivl0*) (λ(*x0*, *t*). *f* (*flow0 x0 t*))
using *flow-continuous-on-state-space*
by (*auto intro*!: *continuous-on-f flow-in-domain simp*: *split-beta′*)

lemma

flow-has-space-derivative:
assumes *t* ∈ *existence-ivl0 x0*
shows ((*λx0. flow0 x0 t*) *has-derivative Dflow x0 t*) (*at x0*)
by (*rule flow-dx-derivative-blinfun*[*THEN has-derivative-eq-rhs*])
(*simp-all add*: *var-eq-mvar assms blinfun.blinfun-apply-inverse Dflow-def vector-Dflow-def*
mem-existence-ivl-iv-defined[*OF assms*])

lemma
flow-has-flowderiv:
assumes *t* ∈ *existence-ivl0 x0*
shows ((*λ*(*x0*, *t*). *flow0 x0 t*) *has-derivative flowderiv x0 t*) (*at* (*x0*, *t*) *within S*)
proof −
have *Sigma*: (*x0*, *t*) ∈ *Sigma X existence-ivl0*
using *assms* **by** *auto*
from *open-state-space assms* **obtain** *e′* **where** *e′*: *e′* > *0 ball* (*x0*, *t*) *e′* ⊆ *Sigma X existence-ivl0*
by (*force simp*: *open-contains-ball*)
define *e* **where** *e* = *e′* / *sqrt 2*
have *0* < *e* **using** *e′* **by** (*auto simp*: *e-def*)
have *ball x0 e* × *ball t e* ⊆ *ball* (*x0*, *t*) *e′*
by (*auto simp*: *dist-prod-def real-sqrt-sum-squares-less e-def*)
also note *e′*(*2*)
finally have *subs*: *ball x0 e* × *ball t e* ⊆ *Sigma X existence-ivl0* **.**

have *d1*: ((*λx0. flow0 x0 s*) *has-derivative blinfun-apply* (*Dflow y s*)) (*at y within ball x0 e*)
if *y* ∈ *ball x0 e s* ∈ *ball t e* **for** *y s*
using *subs that*
by (*subst at-within-open*; *force intro*!: *flow-has-space-derivative*)
have *d2*: (*flow0 y has-derivative blinfun-apply* (*blinfun-scaleR-left* (*f* (*flow0 y s*)))) (*at s within ball t e*)
if *y* ∈ *ball x0 e s* ∈ *ball t e* **for** *y s*
using *subs that*
unfolding *has-vector-derivative-eq-has-derivative-blinfun*[*symmetric*]
by (*subst at-within-open*; *force intro*!: *flow-has-vector-derivative*)
have ((*λ*(*x0*, *t*). *flow0 x0 t*) *has-derivative flowderiv x0 t*) (*at* (*x0*, *t*) *within ball x0 e* × *ball t e*)
using *subs*
unfolding *UNIV-Times-UNIV*[*symmetric*]
by (*intro has-derivative-partialsI*[*OF d1 d2*, *THEN has-derivative-eq-rhs*])
(*auto intro*!: ‹*0* < *e*› *continuous-intros flow-in-domain*
continuous-on-imp-continuous-within[**where** *s*=*Sigma X existence-ivl0*]
assms
simp: *flowderiv-def split-beta′ flow0-defined assms mem-ball*)
then have ((*λ*(*x0*, *t*). *flow0 x0 t*) *has-derivative flowderiv x0 t*) (*at* (*x0*, *t*) *within Sigma X existence-ivl0*)
by (*auto simp*: *at-within-open*[*OF - open-state-space*] *at-within-open*[*OF - open-Times*]

assms ‹*0* < *e*›
mem-existence-ivl-iv-defined[*OF assms*])
then show *?thesis* **unfolding** *at-within-open*[*OF Sigma open-state-space*]
by (*rule has-derivative-at-withinI*)
qed

lemma *flow0-comp-has-derivative*:
assumes *h*: *h s* ∈ *existence-ivl0* (*g s*)
assumes [*derivative-intros*]: (*g has-derivative g′*) (*at s within S*)
assumes [*derivative-intros*]: (*h has-derivative h′*) (*at s within S*)
shows ((λ*x*. *flow0* (*g x*) (*h x*)) *has-derivative* (λ*x*. *blinfun-apply* (*flowderiv* (*g s*) (*h s*)) (*g′ x*, *h′ x*)))
(*at s within S*)
by (*rule has-derivative-compose*[**where** *f*=λ*x*. (*g x*, *h x*) **and** *s*=*S*,
OF - flow-has-flowderiv[*OF h*], *simplified*])
(*auto intro*!: *derivative-eq-intros*)

lemma *flowderiv-continuous-on*: *continuous-on* (*Sigma X existence-ivl0*) (λ(*x0*, *t*). *flowderiv x0 t*)
unfolding *flowderiv-def split-beta′*
by (*subst blinfun-of-matrix-works*[**where** *f*=*comp12* (*Dflow* (*fst x*) (*snd x*))
(*blinfun-scaleR-left* (*f* (*flow0* (*fst x*) (*snd x*)))) **for** *x*, *symmetric*])
(*auto intro*!: *continuous-intros flow-in-domain*)

lemma *flowderiv-continuous-on-comp*[*continuous-intros*]:
assumes *continuous-on S x*
assumes *continuous-on S t*
assumes ⋀*s*. *s* ∈ *S* ⟹ *x s* ∈ *X* ⋀*s*. *s* ∈ *S* ⟹ *t s* ∈ *existence-ivl0* (*x s*)
shows *continuous-on S* (λ*xa*. *flowderiv* (*x xa*) (*t xa*))
by (*rule continuous-on-compose2*[*OF flowderiv-continuous-on*, **where** *f*=λ*s*. (*x s*, *t s*),
unfolded split-beta′ fst-conv snd-conv])
(*auto intro*!: *continuous-intros assms*)

lemmas [*intro*] = *flow-in-domain*

lemma *vareq-trans*: *t0* ∈ *existence-ivl0 x0* ⟹ *t* ∈ *existence-ivl0* (*flow0 x0 t0*) ⟹
vareq (*flow0 x0 t0*) *t* = *vareq x0* (*t0* + *t*)
by (*auto simp*: *vareq-def flow-trans*)

lemma *diff-existence-ivl-trans*:
t0 ∈ *existence-ivl0 x0* ⟹ *t* ∈ *existence-ivl0 x0* ⟹ *t* − *t0* ∈ *existence-ivl0* (*flow0 x0 t0*) **for** *t*
by (*metis* (*no-types*, *opaque-lifting*) *add.left-neutral diff-add-eq*
local.existence-ivl-reverse local.existence-ivl-trans local.flows-reverse)

lemma *has-vderiv-on-blinfun-compose-right*[*derivative-intros*]:
assumes (*g has-vderiv-on g′*) *T*
assumes ⋀*x*. *x* ∈ *T* ⟹ *gd′ x* = *g′ x* o_L *d*

shows $((\lambda x.\ g\ x\ o_L\ d)$ *has-vderiv-on* $gd')\ T$
using *assms*
by (*auto simp*: *has-vderiv-on-def has-vector-derivative-def blinfun-ext blinfun.bilinear-simps intro*!: *derivative-eq-intros ext*)

lemma *has-vderiv-on-blinfun-compose-left*[*derivative-intros*]:
assumes $(g$ *has-vderiv-on* $g')\ T$
assumes $\bigwedge x.\ x \in T \Longrightarrow gd'\ x = d\ o_L\ g'\ x$
shows $((\lambda x.\ d\ o_L\ g\ x)$ *has-vderiv-on* $gd')\ T$
using *assms*
by (*auto simp*: *has-vderiv-on-def has-vector-derivative-def blinfun-ext blinfun.bilinear-simps intro*!: *derivative-eq-intros ext*)

lemma *mvar-flow-shift*:
assumes *t0* ∈ *existence-ivl0 x0 t1* ∈ *existence-ivl0 x0*
shows *mvar.flow x0 t0 d t1* = *Dflow* (*flow0 x0 t0*) (*t1* − *t0*) o_L *d*
proof −
have *mvar.flow x0 t0 d t1* = *mvar.flow x0 t0 d* (*t0* + (*t1* − *t0*))
by *simp*
also have ... = *mvar.flow x0 t0* (*mvar.flow x0 t0 d t0*) *t1*
by (*subst mvar.flow-trans*) (*auto simp add*: *assms*)
also have ... = *Dflow* (*flow0 x0 t0*) (*t1* − *t0*) o_L *d*
apply (*rule mvar.flow-unique-on*)
apply (*auto simp add*: *assms mvar.flow-initial-time-if blinfun-ext Dflow-def intro*!: *derivative-intros derivative-eq-intros*)
apply (*auto simp*: *assms has-vderiv-on-open has-vector-derivative-def intro*!: *derivative-eq-intros blinfun-eqI*)
apply (*subst mvar-existence-ivl-eq-existence-ivl*)
by (*auto simp add*: *vareq-trans assms diff-existence-ivl-trans*)
finally show *?thesis* **.**
qed

lemma *Dflow-trans*:
assumes *h* ∈ *existence-ivl0 x0*
assumes *i* ∈ *existence-ivl0* (*flow0 x0 h*)
shows *Dflow x0* (*h* + *i*) = *Dflow* (*flow0 x0 h*) *i* o_L (*Dflow x0 h*)
proof −
have [*intro*, *simp*]: *h* + *i* ∈ *existence-ivl0 x0 i* + *h* ∈ *existence-ivl0 x0 x0* ∈ *X*
using *assms*
by (*auto simp add*: *add.commute existence-ivl-trans*)
show *?thesis*
unfolding *Dflow-def*
apply (*subst mvar.flow-trans*[**where** *s*=*h* **and** *t*=*i*])
subgoal by (*auto simp*: *assms*)
subgoal by (*auto simp*: *assms*)
by (*subst mvar-flow-shift*) (*auto simp*: *assms Dflow-def*)
qed

lemma *Dflow-trans-apply*:

```
  assumes h ∈ existence-ivl0 x0
  assumes i ∈ existence-ivl0 (flow0 x0 h)
  shows Dflow x0 (h + i) d0 = Dflow (flow0 x0 h) i (Dflow x0 h d0)
proof −
  have [intro, simp]: h + i ∈ existence-ivl0 x0 i + h ∈ existence-ivl0 x0 x0 ∈ X
    using assms
    by (auto simp add: add.commute existence-ivl-trans)
  show ?thesis
    unfolding Dflow-def
    apply (subst mvar.flow-trans[where s=h and t=i])
    subgoal by (auto simp: assms)
    subgoal by (auto simp: assms)
    by (subst mvar-flow-shift) (auto simp: assms Dflow-def )
qed


end — True

end
```

6 Upper and Lower Solutions

```
theory Upper-Lower-Solution
imports Flow
begin
```

Following Walter [1] in section 9

```
lemma IVT-min:
  fixes f :: real ⇒ 'b :: {linorder-topology,real-normed-vector,ordered-real-vector}
  — generalize?
  assumes y: f a ≤ y y ≤ f b a ≤ b
  assumes ∗: continuous-on {a .. b} f
  notes [continuous-intros] = ∗[THEN continuous-on-subset]
  obtains x where a ≤ x x ≤ b f x = y ⋀x'. a ≤ x' ⟹ x' < x ⟹ f x' < y
proof −
  let ?s = ((λx. f x − y) −‘ {0..}) ∩ {a..b}
  have ?s ≠ {}
    using assms
    by auto
  have closed ?s
    by (rule closed-vimage-Int) (auto intro!: continuous-intros)
  moreover have bounded ?s
    by (rule bounded-Int) (simp add: bounded-closed-interval)
  ultimately have compact ?s
    using compact-eq-bounded-closed by blast
  from compact-attains-inf[OF this ‹?s ≠ {}›]
  obtain x where x: a ≤ x x ≤ b f x ≥ y
    and min: ⋀z. a ≤ z ⟹ z ≤ b ⟹ f z ≥ y ⟹ x ≤ z
    by auto
```

have *f x* ≤ *y*
proof (*rule ccontr*)
assume *n*: ¬ *f x* ≤ *y*
then have ∃ *z*≥*a*. *z* ≤ *x* ∧ (λ*x*. *f x* − *y*) *z* = *0*
using *x* **by** (*intro IVT′*) (*auto intro*!: *continuous-intros simp*: *assms*)
then obtain *z* **where** *z*: *a* ≤ *z z* ≤ *x f z* = *y* **by** *auto*
then have *a* ≤ *z z* ≤ *b f z* ≥ *y* **using** *x* **by** *auto*
from *min* [*OF this*] *z n*
show *False* **by** *auto*
qed
then have *a* ≤ *x x* ≤ *b f x* = *y*
using *x*
by (*auto*)
moreover have *f x′* < *y* **if** *a* ≤ *x′ x′* < *x* **for** *x′*
apply (*rule ccontr*)
using *min*[*of x′*] *that x*
by (*auto simp*: *not-less*)
ultimately show *?thesis* **..**
qed

lemma *filtermap-at-left-shift*: *filtermap* (λ*x*. *x* − *d*) (*at-left a*) = *at-left* (*a* − *d*::*real*)
by (*simp add*: *filter-eq-iff eventually-filtermap eventually-at-filter filtermap-nhds-shift*[*symmetric*])

context
fixes *v v′ w w′*::*real* ⇒ *real* **and** *t0 t1 e*::*real*
assumes *v′*: (*v has-vderiv-on v′*) {*t0* <.. *t1*}
and *w′*: (*w has-vderiv-on w′*) {*t0* <.. *t1*}
assumes *pos-ivl*: *t0* < *t1*
assumes *e-pos*: *e* > *0* **and** *e-in*: *t0* + *e* ≤ *t1*
assumes *less*: ⋀*t*. *t0* < *t* ⟹ *t* < *t0* + *e* ⟹ *v t* < *w t*
begin

lemma *first-intersection-crossing-derivatives*:
assumes *na*: *t0* < *tg tg* ≤ *t1 v tg* ≥ *w tg*
notes [*continuous-intros*] =
vderiv-on-continuous-on[*OF v′*, *THEN continuous-on-subset*]
vderiv-on-continuous-on[*OF w′*, *THEN continuous-on-subset*]
obtains *x0* **where**
t0 < *x0 x0* ≤ *tg*
v′ x0 ≥ *w′ x0*
v x0 = *w x0*
⋀*t*. *t0* < *t* ⟹ *t* < *x0* ⟹ *v t* < *w t*
proof −
have (*v* − *w*) (*min tg* (*t0* + *e* / *2*)) ≤ *0 0* ≤ (*v* − *w*) *tg*
min tg (*t0* + *e* / *2*) ≤ *tg*
continuous-on {*min tg* (*t0* + *e* / *2*)..*tg*} (*v* − *w*)
using *less*[*of t0* + *e* / *2*]
less[*of tg*]*na* ‹*e* > *0*›

by (*auto simp*: *min-def intro*!: *continuous-intros*)
from *IVT-min*[*OF this*]
obtain *x0* **where** *x0*: *min tg* (*t0* + *e* / *2*) ≤ *x0 x0* ≤ *tg v x0* = *w x0*
⋀*x'*. *min tg* (*t0* + *e* / *2*) ≤ *x'* ⟹ *x'* < *x0* ⟹ *v x'* < *w x'*
by *auto*
then have *x0-in*: *t0* < *x0 x0* ≤ *t1*
using ‹*e* > *0*› *na*(*1*,*2*)
by (*auto*)
note ‹*t0* < *x0*› ‹*x0* ≤ *tg*›
moreover
{
from *v' x0-in*
have (*v has-derivative* (λ*x*. *x* ∗ *v' x0*)) (*at x0 within* {*t0*<..<*x0*})
by (*force intro*: *has-derivative-subset simp*: *has-vector-derivative-def has-vderiv-on-def*)
then have *v*: ((λ*y*. (*v y* − (*v x0* + (*y* − *x0*) ∗ *v' x0*)) / *norm* (*y* − *x0*)) ⟶ *0*) (*at x0 within* {*t0*<..<*x0*})
unfolding *has-derivative-within*
by (*simp add*: *ac-simps*)
from *w' x0-in*
have (*w has-derivative* (λ*x*. *x* ∗ *w' x0*)) (*at x0 within* {*t0*<..<*x0*})
by (*force intro*: *has-derivative-subset simp*: *has-vector-derivative-def has-vderiv-on-def*)
then have *w*: ((λ*y*. (*w y* − (*w x0* + (*y* − *x0*) ∗ *w' x0*)) / *norm* (*y* − *x0*)) ⟶ *0*) (*at x0 within* {*t0*<..<*x0*})
unfolding *has-derivative-within*
by (*simp add*: *ac-simps*)

have *evs*: ∀$_F$ *x in at x0 within* {*t0*<..<*x0*}. *min tg* (*t0* + *e* / *2*) < *x*
∀$_F$ *x in at x0 within* {*t0*<..<*x0*}. *t0* < *x* ∧ *x* < *x0*
using *less na*(*1*) *na*(*3*) *x0*(*3*) *x0-in*(*1*)
by (*force simp*: *min-def eventually-at-filter intro*!: *order-tendstoD*[*OF tendsto-ident-at*])+
then have ∀$_F$ *x in at x0 within* {*t0*<..<*x0*}.
(*v x* − (*v x0* + (*x* − *x0*) ∗ *v' x0*)) / *norm* (*x* − *x0*) − (*w x* − (*w x0* + (*x* − *x0*) ∗ *w' x0*)) / *norm* (*x* − *x0*) =
(*v x* − *w x*) / *norm* (*x* − *x0*) + (*v' x0* − *w' x0*)
apply *eventually-elim*
using *x0-in x0 less na* ‹*t0* < *t1*› *sum-sqs-eq*
by (*auto simp*: *divide-simps algebra-simps min-def intro*!: *eventuallyI split*: *if-split-asm*)
from *this tendsto-diff*[*OF v w*]
have *1*: ((λ*x*. (*v x* − *w x*) / *norm* (*x* − *x0*) + (*v' x0* − *w' x0*)) ⟶ *0*) (*at x0 within* {*t0*<..<*x0*})
by (*force intro*: *tendsto-eq-rhs Lim-transform-eventually*)
moreover
from *evs* **have** *2*: ∀$_F$ *x in at x0 within* {*t0*<..<*x0*}. (*v x* − *w x*) / *norm* (*x* − *x0*) + (*v' x0* − *w' x0*) ≤ (*v' x0* − *w' x0*)
by *eventually-elim* (*auto simp*: *divide-simps intro*!: *less-imp-le x0*(*4*))

moreover

```
    have at x0 within {t0<..<x0} ≠ bot
      by (simp add: ‹t0 < x0› at-within-eq-bot-iff less-imp-le)

    ultimately
    have 0 ≤ v′ x0 − w′ x0
      by (rule tendsto-upperbound)
    then have v′ x0 ≥ w′ x0 by simp
  }
  moreover note ‹v x0 = w x0›
  moreover
  have t0 < t ⟹ t < x0 ⟹ v t < w t for t
    by (cases min tg (t0 + e / 2) ≤ t) (auto intro: x0 less)
  ultimately show ?thesis ..
qed

lemma defect-less:
  assumes b: ⋀t. t0 < t ⟹ t ≤ t1 ⟹ v′ t − f t (v t) < w′ t − f t (w t)
  notes [continuous-intros] =
    vderiv-on-continuous-on[OF v′, THEN continuous-on-subset]
    vderiv-on-continuous-on[OF w′, THEN continuous-on-subset]
  shows ∀ t ∈ {t0 <.. t1}. v t < w t
proof (rule ccontr)
  assume ¬ (∀ t∈{t0 <.. t1}. v t < w t)
  then obtain tu where t0 < tu tu ≤ t1 v tu ≥ w tu by auto
  from first-intersection-crossing-derivatives[OF this]
  obtain x0 where t0 < x0 x0 ≤ tu w′ x0 ≤ v′ x0 v x0 = w x0 ⋀t. t0 < t ⟹
t < x0 ⟹ v t < w t
    by metis
  with b[of x0] ‹tu ≤ t1›
  show False
    by simp
qed

end

lemma has-derivatives-less-lemma:
  fixes v v′ ::real ⇒ real
  assumes v′: (v has-vderiv-on v′) T
  assumes y′: (y has-vderiv-on y′) T
  assumes lu: ⋀t. t ∈ T ⟹ t > t0 ⟹ v′ t − f t (v t) < y′ t − f t (y t)
  assumes lower: v t0 ≤ y t0
  assumes eq-imp: v t0 = y t0 ⟹ v′ t0 < y′ t0
  assumes t: t0 < t t0 ∈ T t ∈ T is-interval T
  shows v t < y t
proof −
  have subset: {t0 .. t} ⊆ T
    by (rule atMostAtLeast-subset-convex) (auto simp: assms is-interval-convex)
  obtain d where 0 < d t0 < s ⟹ s ≤ t ⟹ s < t0 + d ⟹ v s < y s for s
  proof cases
```

assume *v t0 = y t0*
from *this*[*THEN eq-imp*]
have ∗: *0 < y′ t0 − v′ t0*
by *simp*
have ((λ*t. y t − v t*) *has-vderiv-on* (λ*t0. y′ t0 − v′ t0*)) {*t0* .. *t*}
by (*auto intro*!: *derivative-intros y′ v′ has-vderiv-on-subset*[*OF - subset*])
with ‹*t0 < t*›
have *d*: ((λ*t. y t − v t*) *has-real-derivative y′ t0 − v′ t0*) (*at t0 within* {*t0* .. *t*})
by (*auto simp*: *has-vderiv-on-def has-real-derivative-iff-has-vector-derivative*)
from *has-real-derivative-pos-inc-right*[*OF d* ∗] ‹*v t0 = y t0*›
obtain *d* **where** *d > 0* **and** *vy*: *h > 0* ⟹ *t0 + h ≤ t* ⟹ *h < d* ⟹ *v* (*t0 + h*) *< y* (*t0 + h*) **for** *h*
by *auto*
have *vy*: *t0 < s* ⟹ *s ≤ t* ⟹ *s < t0 + d* ⟹ *v s < y s* **for** *s*
using *vy*[*of s − t0*] **by** *simp*
with ‹*d > 0*› **show** *?thesis* **..**
next
assume *v t0 ≠ y t0*
then have *v t0 < y t0* **using** *lower* **by** *simp*
moreover
have *continuous-on* {*t0* .. *t*} *v continuous-on* {*t0* .. *t*} *y*
by (*auto intro*!: *vderiv-on-continuous-on assms has-vderiv-on-subset*[*OF - subset*])
then have (*v* ⟶ *v t0*) (*at t0 within* {*t0* .. *t*}) (*y* ⟶ *y t0*) (*at t0 within* {*t0* .. *t*})
by (*auto simp*: *continuous-on*)
ultimately have $\forall_F$ *x in at t0 within* {*t0* .. *t*}. *0 < y x − v x*
by (*intro order-tendstoD*) (*auto intro*!: *tendsto-eq-intros*)
then obtain *d* **where** *d > 0* ⋀*x. t0 < x* ⟹ *x ≤ t* ⟹ *x < t0 + d* ⟹ *v x < y x*
by *atomize-elim* (*auto simp*: *eventually-at algebra-simps dist-real-def*)
then show *?thesis* **..**
qed
with ‹*d > 0*› ‹*t0 < t*›
obtain *e* **where** *e > 0 t0 + e ≤ t t0 < s* ⟹ *s < t0 + e* ⟹ *v s < y s* **for** *s*
by *atomize-elim* (*auto simp*: *min-def divide-simps intro*!: *exI*[**where** *x=min* (*d/2*) ((*t − t0*) / *2*)]
split: *if-split-asm*)
from *defect-less*[
OF has-vderiv-on-subset[*OF v′*]
has-vderiv-on-subset[*OF y′*]
‹*t0 < t*›
this lu]
show *v t < y t* **using** ‹*t0 < t*› *subset*
by (*auto simp*: *subset-iff assms*)
qed

lemma *strict-lower-solution*:

fixes *v v′* ::*real* ⇒ *real*
assumes *sol*: (*y solves-ode f*) *T X*
assumes *v′*: (*v has-vderiv-on v′*) *T*
assumes *lower*: $\bigwedge t.\ t \in T \Longrightarrow t > t0 \Longrightarrow v'\ t < f\ t\ (v\ t)$
assumes *iv*: $v\ t0 \leq y\ t0\ v\ t0 = y\ t0 \Longrightarrow v'\ t0 < f\ t0\ (y\ t0)$
assumes *t*: $t0 < t\ t0 \in T\ t \in T$ *is-interval T*
shows $v\ t < y\ t$
proof −
note *v′*
moreover
note *solves-odeD*(*1*)[*OF sol*]
moreover
have *3*: $v'\ t - f\ t\ (v\ t) < f\ t\ (y\ t) - f\ t\ (y\ t)$ **if** $t \in T\ t > t0$ **for** *t*
using *lower*(*1*)[*OF that*]
by *arith*
moreover note *iv*
moreover note *t*
ultimately
show $v\ t < y\ t$
by (*rule has-derivatives-less-lemma*)
qed

lemma *strict-upper-solution*:
fixes *w w′*::*real* ⇒ *real*
assumes *sol*: (*y solves-ode f*) *T X*
assumes *w′*: (*w has-vderiv-on w′*) *T*
and *upper*: $\bigwedge t.\ t \in T \Longrightarrow t > t0 \Longrightarrow f\ t\ (w\ t) < w'\ t$
and *iv*: $y\ t0 \leq w\ t0\ y\ t0 = w\ t0 \Longrightarrow f\ t0\ (y\ t0) < w'\ t0$
assumes *t*: $t0 < t\ t0 \in T\ t \in T$ *is-interval T*
shows $y\ t < w\ t$
proof −
note *solves-odeD*(*1*)[*OF sol*]
moreover
note *w′*
moreover
have $f\ t\ (y\ t) - f\ t\ (y\ t) < w'\ t - f\ t\ (w\ t)$ **if** $t \in T\ t > t0$ **for** *t*
using *upper*(*1*)[*OF that*]
by *arith*
moreover note *iv*
moreover note *t*
ultimately
show $y\ t < w\ t$
by (*rule has-derivatives-less-lemma*)
qed

lemma *uniform-limit-at-within-subset*:
assumes *uniform-limit S x l* (*at t within T*)
assumes $U \subseteq T$
shows *uniform-limit S x l* (*at t within U*)

by (*metis assms*(*1*) *assms*(*2*) *eventually-within-Un filterlim-iff subset-Un-eq*)

lemma *uniform-limit-le*:
fixes f::$'c \Rightarrow 'a \Rightarrow 'b$::{*metric-space*, *linorder-topology*}
assumes I: $I \neq bot$
assumes u: *uniform-limit* $X\ f\ g\ I$
assumes u': *uniform-limit* $X\ f'\ g'\ I$
assumes $\forall_F\ i\ in\ I.\ \forall x \in X.\ f\ i\ x \leq f'\ i\ x$
assumes $x \in X$
shows $g\ x \leq g'\ x$
proof −
have $\forall_F\ i\ in\ I.\ f\ i\ x \leq f'\ i\ x$ **using** *assms* **by** (*simp add*: *eventually-mono*)
with *I tendsto-uniform-limitI*[*OF u'* ‹$x \in X$›] *tendsto-uniform-limitI*[*OF u* ‹$x \in X$›]
show *?thesis* **by** (*rule tendsto-le*)
qed

lemma *uniform-limit-le-const*:
fixes f::$'c \Rightarrow 'a \Rightarrow 'b$::{*metric-space*, *linorder-topology*}
assumes I: $I \neq bot$
assumes u: *uniform-limit* $X\ f\ g\ I$
assumes $\forall_F\ i\ in\ I.\ \forall x \in X.\ f\ i\ x \leq h\ x$
assumes $x \in X$
shows $g\ x \leq h\ x$
proof −
have $\forall_F\ i\ in\ I.\ f\ i\ x \leq h\ x$ **using** *assms* **by** (*simp add*: *eventually-mono*)
then show *?thesis* **by** (*metis tendsto-upperbound I tendsto-uniform-limitI*[*OF u* ‹$x \in X$›])
qed

lemma *uniform-limit-ge-const*:
fixes f::$'c \Rightarrow 'a \Rightarrow 'b$::{*metric-space*, *linorder-topology*}
assumes I: $I \neq bot$
assumes u: *uniform-limit* $X\ f\ g\ I$
assumes $\forall_F\ i\ in\ I.\ \forall x \in X.\ h\ x \leq f\ i\ x$
assumes $x \in X$
shows $h\ x \leq g\ x$
proof −
have $\forall_F\ i\ in\ I.\ h\ x \leq f\ i\ x$ **using** *assms* **by** (*simp add*: *eventually-mono*)
then show *?thesis* **by** (*metis tendsto-lowerbound I tendsto-uniform-limitI*[*OF u* ‹$x \in X$›])
qed

locale *ll-on-open-real* = *ll-on-open* $T\ f\ X$ **for** $T\ f$ **and** X::*real set*
begin

lemma *lower-solution*:
fixes $v\ v'$ ::*real* $\Rightarrow$ *real*
assumes *sol*: (y *solves-ode* f) $S\ X$

assumes *v'*: (*v has-vderiv-on v'*) *S*
assumes *lower*: $\bigwedge t.\ t \in S \Longrightarrow t > t0 \Longrightarrow v'\ t < f\ t\ (v\ t)$
assumes *iv*: $v\ t0 \le y\ t0$
assumes *t*: $t0 \le t\ t0 \in S\ t \in S$ *is-interval* $S\ S \subseteq T$
shows $v\ t \le y\ t$
proof *cases*
assume $v\ t0 = y\ t0$
have $\{t0\ {-}{-}\ t\} \subseteq S$ **using** *t* **by** (*simp add*: *closed-segment-subset is-interval-convex*)
with *sol* **have** (*y solves-ode f*) $\{t0\ {-}{-}\ t\}$ *X* **using** *order-refl* **by** (*rule solves-ode-on-subset*)
moreover note *refl*
moreover
have $\{t0\ {-}{-}\ t\} \subseteq T$ **using** ‹$\{t0\ {-}{-}\ t\} \subseteq S$› ‹$S \subseteq T$› **by** (*rule order-trans*)
ultimately have *t-ex*: $t \in$ *existence-ivl t0* (*y t0*)
by (*rule existence-ivl-maximal-segment*)

have *t0-ex*: $t0 \in$ *existence-ivl t0* (*y t0*)
using *in-existence-between-zeroI t-ex* **by** *blast*
have $t0 \in T$ **using** *assms*(*9*) *t*(*2*) **by** *blast*

from *uniform-limit-flow*[*OF t0-ex t-ex*] ‹$t0 \le t$›
have *uniform-limit* {*t0..t*} (*flow t0*) (*flow t0* (*y t0*)) (*at* (*y t0*)) **by** *simp*
then have *uniform-limit* {*t0..t*} (*flow t0*) (*flow t0* (*y t0*)) (*at-right* (*y t0*))
by (*rule uniform-limit-at-within-subset*) *simp*
moreover
{
have $\forall_F$ *i in at* (*y t0*). $t \in$ *existence-ivl t0 i*
by (*rule eventually-mem-existence-ivl*) *fact*
then have $\forall_F$ *i in at-right* (*y t0*). $t \in$ *existence-ivl t0 i*
unfolding *eventually-at-filter*
by *eventually-elim simp*
moreover have $\forall_F$ *i in at-right* (*y t0*). $i \in X$
proof −
have *f1*: $\bigwedge r\ ra\ rb.\ r \notin$ *existence-ivl ra rb* $\vee\ rb \in X$
by (*metis existence-ivl-reverse flow-in-domain flows-reverse*)
obtain *rr* :: (*real* ⇒ *bool*) ⇒ (*real* ⇒ *bool*) ⇒ *real* **where**
⋀*p f pa fa*. (¬ *eventually p f* ∨ *eventually pa f* ∨ *p* (*rr p pa*)) ∧
(¬ *eventually p fa* ∨ ¬ *pa* (*rr p pa*) ∨ *eventually pa fa*)
by (*metis* (*no-types*) *eventually-mono*)
then show *?thesis*
using *f1 calculation* **by** *meson*
qed
moreover have $\forall_F$ *i in at-right* (*y t0*). $y\ t0 < i$
by (*simp add*: *eventually-at-filter*)
ultimately have $\forall_F$ *i in at-right* (*y t0*). $\forall x \in \{t0..t\}.\ v\ x \le$ *flow t0 i x*
proof *eventually-elim*
case (*elim y'*)
show *?case*
proof *safe*
fix *s* **assume** *s*: $s \in \{t0..t\}$

show *v s* ≤ *flow t0 y′ s*
proof *cases*
assume *s* = *t0* **with** *elim iv* **show** *?thesis*
by (*simp add*: ‹*t0* ∈ *T*› ‹*y′* ∈ *X*›)
next
assume *s* ≠ *t0* **with** *s* **have** *t0* < *s* **by** *simp*
have {*t0* −− *s*} ⊆ *S* **using** ‹{*t0*−−*t*} ⊆ *S*› *closed-segment-eq-real-ivl s*
by *auto*
from *s elim* **have** {*t0*..*s*} ⊆ *existence-ivl t0 y′*
using *ivl-subset-existence-ivl* **by** *blast*
with *flow-solves-ode* **have** *sol*: (*flow t0 y′ solves-ode f*) {*t0* .. *s*} *X*
by (*rule solves-ode-on-subset*) (*auto intro*!: ‹*y′* ∈ *X*› ‹*t0* ∈ *T*›)
have {*t0* .. *s*} ⊆ *S* **using** ‹{*t0* −− *s*} ⊆ *S*› **by** (*simp add*: *closed-segment-eq-real-ivl*
split: *if-splits*)
with *v′* **have** *v′*: (*v has-vderiv-on v′*) {*t0* .. *s*}
by (*rule has-vderiv-on-subset*)
from ‹*y t0* < *y′*› ‹*v t0* = *y t0*› **have** *less-init*: *v t0* < *flow t0 y′ t0*
by (*simp add*: *flow-initial-time-if* ‹*t0* ∈ *T*› ‹*y′* ∈ *X*›)
from *strict-lower-solution*[*OF sol v′ lower less-imp-le*[*OF less-init*] - ‹*t0* <
s›]
‹{*t0* .. *s*} ⊆ *S*›
less-init ‹*t0* < *s*›
have *v s* < *flow t0 y′ s* **by** (*simp add*: *subset-iff is-interval-cc*)
then show *?thesis* **by** *simp*
qed
qed
qed
}
moreover have *t* ∈ {*t0* .. *t*} **using** ‹*t0* ≤ *t*› **by** *simp*
ultimately have *v t* ≤ *flow t0* (*y t0*) *t*
by (*rule uniform-limit-ge-const*[*OF trivial-limit-at-right-real*])
also have *flow t0* (*y t0*) *t* = *y t*
using *sol t*
by (*intro maximal-existence-flow*) *auto*
finally show *?thesis* **.**
next
assume *v t0* ≠ *y t0* **then have** *less*: *v t0* < *y t0* **using** *iv* **by** *simp*
show *?thesis*
apply (*cases t0* = *t*)
subgoal using *iv* **by** *blast*
subgoal using *strict-lower-solution*[*OF sol v′ lower iv*] *less t* **by** *force*
done
qed

lemma *upper-solution*:
fixes *v v′* ::*real* ⇒ *real*
assumes *sol*: (*y solves-ode f*) *S X*
assumes *v′*: (*v has-vderiv-on v′*) *S*
assumes *upper*: ⋀*t*. *t* ∈ *S* ⟹ *t* > *t0* ⟹ *f t* (*v t*) < *v′ t*

assumes *iv*: *y t0* ≤ *v t0*
assumes *t*: *t0* ≤ *t t0* ∈ *S t* ∈ *S is-interval S S* ⊆ *T*
shows *y t* ≤ *v t*
proof *cases*
assume *v t0* = *y t0*
have {*t0* −− *t*} ⊆ *S* **using** *t* **by** (*simp add*: *closed-segment-subset is-interval-convex*)
with *sol* **have** (*y solves-ode f*) {*t0* −− *t*} *X* **using** *order-refl* **by** (*rule solves-ode-on-subset*)
moreover note *refl*
moreover
have {*t0* −− *t*} ⊆ *T* **using** ‹{*t0* −− *t*} ⊆ *S*› ‹*S* ⊆ *T*› **by** (*rule order-trans*)
ultimately have *t-ex*: *t* ∈ *existence-ivl t0* (*y t0*)
  by (*rule existence-ivl-maximal-segment*)

have *t0-ex*: *t0* ∈ *existence-ivl t0* (*y t0*)
  using *in-existence-between-zeroI t-ex* **by** *blast*
have *t0* ∈ *T* **using** *assms*(*9*) *t*(*2*) **by** *blast*

from *uniform-limit-flow*[*OF t0-ex t-ex*] ‹*t0* ≤ *t*›
have *uniform-limit* {*t0*..*t*} (*flow t0*) (*flow t0* (*y t0*)) (*at* (*y t0*)) **by** *simp*
then have *uniform-limit* {*t0*..*t*} (*flow t0*) (*flow t0* (*y t0*)) (*at-left* (*y t0*))
  by (*rule uniform-limit-at-within-subset*) *simp*
moreover
{
  have $\forall_F$ *i in at* (*y t0*). *t* ∈ *existence-ivl t0 i*
    by (*rule eventually-mem-existence-ivl*) *fact*
  then have $\forall_F$ *i in at-left* (*y t0*). *t* ∈ *existence-ivl t0 i*
    unfolding *eventually-at-filter*
    by *eventually-elim simp*
  moreover have $\forall_F$ *i in at-left* (*y t0*). *i* ∈ *X*
  proof −
    have *f1*: ⋀*r ra rb*. *r* ∉ *existence-ivl ra rb* ∨ *rb* ∈ *X*
      by (*metis existence-ivl-reverse flow-in-domain flows-reverse*)
    obtain *rr* :: (*real* ⇒ *bool*) ⇒ (*real* ⇒ *bool*) ⇒ *real* **where**
      ⋀*p f pa fa*. (¬ *eventually p f* ∨ *eventually pa f* ∨ *p* (*rr p pa*)) ∧
        (¬ *eventually p fa* ∨ ¬ *pa* (*rr p pa*) ∨ *eventually pa fa*)
      by (*metis* (*no-types*) *eventually-mono*)
    then show *?thesis*
      using *f1 calculation* **by** *meson*
  qed
  moreover have $\forall_F$ *i in at-left* (*y t0*). *i* < *y t0*
    by (*simp add*: *eventually-at-filter*)
  ultimately have $\forall_F$ *i in at-left* (*y t0*). ∀ *x*∈{*t0*..*t*}. *flow t0 i x* ≤ *v x*
  proof *eventually-elim*
    case (*elim y′*)
    show *?case*
    proof *safe*
      fix *s* **assume** *s*: *s* ∈ {*t0*..*t*}
      show *flow t0 y′ s* ≤ *v s*
      proof *cases*

assume *s* = *t0* **with** *elim iv* **show** *?thesis*
by (*simp add*: ‹*t0* ∈ *T*› ‹*y'* ∈ *X*›)
next
assume *s* ≠ *t0* **with** *s* **have** *t0* < *s* **by** *simp*
have {*t0* −− *s*} ⊆ *S* **using** ‹{*t0*−−*t*} ⊆ *S*› *closed-segment-eq-real-ivl s*
by *auto*
from *s elim* **have** {*t0*..*s*} ⊆ *existence-ivl t0 y'*
using *ivl-subset-existence-ivl* **by** *blast*
with *flow-solves-ode* **have** *sol*: (*flow t0 y' solves-ode f*) {*t0* .. *s*} *X*
by (*rule solves-ode-on-subset*) (*auto intro*!: ‹*y'* ∈ *X*› ‹*t0* ∈ *T*›)
have {*t0* .. *s*} ⊆ *S* **using** ‹{*t0* −− *s*} ⊆ *S*› **by** (*simp add*: *closed-segment-eq-real-ivl*
split: *if-splits*)
with *v'* **have** *v'*: (*v has-vderiv-on v'*) {*t0* .. *s*}
by (*rule has-vderiv-on-subset*)
from ‹*y'* < *y t0*› ‹*v t0* = *y t0*› **have** *less-init*: *flow t0 y' t0* < *v t0*
by (*simp add*: *flow-initial-time-if* ‹*t0* ∈ *T*› ‹*y'* ∈ *X*›)
from *strict-upper-solution*[*OF sol v' upper less-imp-le*[*OF less-init*] - ‹*t0*
< *s*›]
‹{*t0* .. *s*} ⊆ *S*›
less-init ‹*t0* < *s*›
have *flow t0 y' s* < *v s* **by** (*simp add*: *subset-iff is-interval-cc*)
then show *?thesis* **by** *simp*
qed
qed
qed
}
moreover have *t* ∈ {*t0* .. *t*} **using** ‹*t0* ≤ *t*› **by** *simp*
ultimately have *flow t0* (*y t0*) *t* ≤ *v t*
by (*rule uniform-limit-le-const*[*OF trivial-limit-at-left-real*])
also have *flow t0* (*y t0*) *t* = *y t*
using *sol t*
by (*intro maximal-existence-flow*) *auto*
finally show *?thesis* **.**
next
assume *v t0* ≠ *y t0* **then have** *less*: *y t0* < *v t0* **using** *iv* **by** *simp*
show *?thesis*
apply (*cases t0* = *t*)
subgoal using *iv* **by** *blast*
subgoal using *strict-upper-solution*[*OF sol v' upper iv*] *less t* **by** *force*
done
qed

end

end
theory *Poincare-Map*
imports
Flow
begin

abbreviation *plane n c* ≡ $\{x.\ x \cdot n = c\}$

lemma
eventually-tendsto-compose-within:
assumes *eventually P* (*at l within S*)
assumes *P l*
assumes ($f \longrightarrow l$) (*at x within T*)
assumes *eventually* ($\lambda x.\ f\ x \in S$) (*at x within T*)
shows *eventually* ($\lambda x.\ P\ (f\ x)$) (*at x within T*)
proof −
from *assms*(*1*) *assms*(*2*) **obtain** *U* **where** *U*:
open U $l \in U$ $\bigwedge x.\ x \in U \Longrightarrow x \in S \Longrightarrow P\ x$
by (*force simp*: *eventually-at-topological*)
from *topological-tendstoD*[*OF assms*(*3*) ‹*open U*› ‹$l \in U$›]
have $\forall_F$ *x in at x within T*. $f\ x \in U$ **by** *auto*
then show *?thesis* **using** *assms*(*4*)
by *eventually-elim* (*auto intro*!: *U*)
qed

lemma
eventually-eventually-withinI:— aha...
assumes $\forall_F$ *x in at x within A*. *P x P x*
shows $\forall_F$ *a in at x within S*. $\forall_F$ *x in at a within A*. *P x*
using *assms*
unfolding *eventually-at-topological*
by *force*

lemma *eventually-not-in-closed*:
assumes *closed P*
assumes $f\ t \notin P$ $t \in T$
assumes *continuous-on T f*
shows $\forall_F$ *t in at t within T*. $f\ t \notin P$
using *assms*
unfolding *Compl-iff*[*symmetric*] *closed-def continuous-on-topological eventually-at-topological*
by *metis*

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

lemma
existence-ivl-trans′:
assumes $t + s \in$ *existence-ivl t0 x0*
$t \in$ *existence-ivl t0 x0*
shows $t + s \in$ *existence-ivl t* (*flow t0 x0 t*)
by (*meson assms*(*1*) *assms*(*2*) *general.existence-ivl-reverse general.flow-solves-ode general.is-interval-existence-ivl general.maximal-existence-flow*(*1*) *general.mem-existence-ivl-iv-defined*(*2*) *general.mem-existence-ivl-subset local.existence-ivl-subset subsetD*)

end

context *auto-ll-on-open*— TODO: generalize to continuous systems
begin

definition *returns-to* ::′*a set* ⇒ ′*a* ⇒ *bool*
  where *returns-to P x* ⟷ (∀$_F$ *t in at-right 0. flow0 x t* ∉ *P*) ∧ (∃ *t*>*0. t* ∈ *existence-ivl0 x* ∧ *flow0 x t* ∈ *P*)

definition *return-time* :: ′*a set* ⇒ ′*a* ⇒ *real*
  where *return-time P x* =
    (*if returns-to P x then* (*SOME t.*
      t > *0* ∧
      t ∈ *existence-ivl0 x* ∧
      flow0 x t ∈ *P* ∧
      (∀ *s* ∈ {*0*<..<*t*}. *flow0 x s* ∉ *P*)) *else 0*)

lemma *returns-toI*:
  assumes *t*: *t* > *0 t* ∈ *existence-ivl0 x flow0 x t* ∈ *P*
  assumes *ev*: ∀$_F$ *t in at-right 0. flow0 x t* ∉ *P*
  assumes *closed P*
  shows *returns-to P x*
  using *assms*
  by (*auto simp*: *returns-to-def*)

lemma *returns-to-outsideI*:
  assumes *t*: *t* ≥ *0 t* ∈ *existence-ivl0 x flow0 x t* ∈ *P*
  assumes *ev*: *x* ∉ *P*
  assumes *closed P*
  shows *returns-to P x*
proof *cases*
  assume *t* > *0*
  moreover
  have ∀$_F$ *s in at 0 within* {*0* .. *t*}. *flow0 x s* ∉ *P*
    using *assms mem-existence-ivl-iv-defined ivl-subset-existence-ivl*[*OF* ‹*t* ∈ -›] ‹*0* < *t*›
    by (*auto intro*!: *eventually-not-in-closed flow-continuous-on continuous-intros*
      simp: *eventually-conj-iff*)
  with *order-tendstoD*(*2*)[*OF tendsto-ident-at* ‹*0* < *t*›, *of* {*0*<..}]
  have ∀$_F$ *t in at-right 0. flow0 x t* ∉ *P*
    unfolding *eventually-at-filter*
    by *eventually-elim* (*use* ‹*t* > *0*› **in** *auto*)
  then show *?thesis*
    by (*auto intro*!: *returns-toI assms* ‹*0* < *t*›)
qed (*use assms* **in** *simp*)

lemma *returns-toE*:
  assumes *returns-to P x*
  obtains *t0 t1* **where**

0 < t0
t0 ≤ t1
t1 ∈ existence-ivl0 x
flow0 x t1 ∈ P
$\bigwedge$*t. 0 < t ⟹ t < t0 ⟹ flow0 x t ∉ P*
proof −
obtain *t0 t1* **where** *t0*: *t0 > 0* $\bigwedge$*t. 0 < t ⟹ t < t0 ⟹ flow0 x t ∉ P*
and *t1*: *t1 > 0 t1 ∈ existence-ivl0 x flow0 x t1 ∈ P*
using *assms*
by (*auto simp*: *returns-to-def eventually-at-right*[*OF zero-less-one*])
moreover
have *t0 ≤ t1*
using *t0*(*2*)[*of t1*] *t1 t0*(*1*)
by *force*
ultimately show *?thesis* **by** (*blast intro*: *that*)
qed

lemma *return-time-some*:
assumes *returns-to P x*
shows *return-time P x* =
(*SOME t. t > 0 ∧ t ∈ existence-ivl0 x ∧ flow0 x t ∈ P ∧* (∀ *s ∈* {*0<..<t*}.
flow0 x s ∉ P))
using *assms* **by** (*auto simp*: *return-time-def*)

lemma *return-time-ex1*:
assumes *returns-to P x*
assumes *closed P*
shows ∃!*t. t > 0 ∧ t ∈ existence-ivl0 x ∧ flow0 x t ∈ P ∧* (∀ *s ∈* {*0<..<t*}.
flow0 x s ∉ P)
proof −
from *returns-toE*[*OF* ‹*returns-to P x*›]
obtain *t0 t1* **where**
t1: *t1 ≥ t0 t1 ∈ existence-ivl0 x flow0 x t1 ∈ P*
and *t0*: *t0 > 0* $\bigwedge$*t. 0 < t ⟹ t < t0 ⟹ flow0 x t ∉ P*
by *metis*
from *flow-continuous-on* **have** *cont*: *continuous-on* {*0 .. t1*} (*flow0 x*)
by (*rule continuous-on-subset*) (*intro ivl-subset-existence-ivl t1*)
from *cont* **have** *cont′*: *continuous-on* {*t0 .. t1*} (*flow0 x*)
by (*rule continuous-on-subset*) (*use* ‹*0 < t0*› **in** *auto*)
have *compact* (*flow0 x* −‘ *P* ∩ {*t0 .. t1*})
using ‹*closed P*› *cont′*
by (*auto simp*: *compact-eq-bounded-closed bounded-Int bounded-closed-interval*
intro!: *closed-vimage-Int*)

have *flow0 x* −‘ *P* ∩ {*t0..t1*} ≠ {}
using *t1 t0* **by** *auto*
from *compact-attains-inf*[*OF* ‹*compact -*› *this*] *t0 t1*
obtain *rt* **where** *rt*: *t0 ≤ rt rt ≤ t1 flow0 x rt ∈ P*
and *least*: $\bigwedge$*t′. flow0 x t′ ∈ P ⟹ t0 ≤ t′ ⟹ t′ ≤ t1 ⟹ rt ≤ t′*

by *auto*
have $0 < rt$ *flow0 x rt* $\in$ *P rt* $\in$ *existence-ivl0 x*
and $0 < t' \Longrightarrow t' < rt \Longrightarrow$ *flow0 x t'* $\notin$ *P* **for** t'
using *ivl-subset-existence-ivl*[*OF* ‹*t1* $\in$ *existence-ivl0 x*›] *t0 t1 rt least*[*of t'*]
by *force+*
then show *?thesis*
by (*intro ex-ex1I*) *force+*
qed

lemma
return-time-pos-returns-to:
return-time P x $> 0 \Longrightarrow$ *returns-to P x*
by (*auto simp*: *return-time-def split*: *if-splits*)

lemma
assumes *ret*: *returns-to P x*
assumes *closed P*
shows *return-time-pos*: *return-time P x* > 0
using *someI-ex*[*OF return-time-ex1*[*OF assms*, *THEN ex1-implies-ex*]]
unfolding *return-time-some*[*OF ret*, *symmetric*]
by *auto*

lemma *returns-to-return-time-pos*:
assumes *closed P*
shows *returns-to P x* $\longleftrightarrow$ *return-time P x* > 0
by (*auto intro*!: *return-time-pos assms*) (*auto simp*: *return-time-def split*: *if-splits*)

lemma *return-time*:
assumes *ret*: *returns-to P x*
assumes *closed P*
shows *return-time P x* > 0
and *return-time-exivl*: *return-time P x* $\in$ *existence-ivl0 x*
and *return-time-returns*: *flow0 x* (*return-time P x*) $\in$ *P*
and *return-time-least*: $\bigwedge s.\ 0 < s \Longrightarrow s <$ *return-time P x* $\Longrightarrow$ *flow0 x s* $\notin$ *P*
using *someI-ex*[*OF return-time-ex1*[*OF assms*, *THEN ex1-implies-ex*]]
unfolding *return-time-some*[*OF ret*, *symmetric*]
by *auto*

lemma *returns-to-earlierI*:
assumes *ret*: *returns-to P* (*flow0 x t*) *closed P*
assumes $t \geq 0$ $t \in$ *existence-ivl0 x*
assumes *ev*: $\forall_F$ *t in at-right 0*. *flow0 x t* $\notin$ *P*
shows *returns-to P x*
proof −
from *return-time*[*OF ret*]
have *rt*: $0 <$ *return-time P* (*flow0 x t*) *flow0* (*flow0 x t*) (*return-time P* (*flow0 x t*)) $\in$ *P*
and $0 < s \Longrightarrow s <$ *return-time P* (*flow0 x t*) $\Longrightarrow$ *flow0* (*flow0 x t*) *s* $\notin$ *P* **for** *s*
by *auto*

let *?t* = *t* + *return-time P* (*flow0 x t*)
show *?thesis*
proof (*rule returns-toI*[*of ?t*])
show *0* < *?t* **by** (*auto intro*!: *add-nonneg-pos rt* ‹*t* ≥ *0*›)
show *?t* ∈ *existence-ivl0 x*
by (*intro existence-ivl-trans return-time-exivl assms*)
have *flow0 x* (*t* + *return-time P* (*flow0 x t*)) = *flow0* (*flow0 x t*) (*return-time P* (*flow0 x t*))
by (*intro flow-trans assms return-time-exivl*)
also have ... ∈ *P*
by (*rule return-time-returns*[*OF ret*])
finally show *flow0 x* (*t* + *return-time P* (*flow0 x t*)) ∈ *P* **.**
show *closed P* **by** *fact*
show $\forall_F$ *t in at-right 0*. *flow0 x t* ∉ *P* **by** *fact*
qed
qed

lemma *return-time-gt*:
assumes *ret*: *returns-to P x closed P*
assumes *flow-not*: ⋀*s*. *0* < *s* ⟹ *s* ≤ *t* ⟹ *flow0 x s* ∉ *P*
shows *t* < *return-time P x*
using *flow-not*[*of return-time P x*] *return-time-pos*[*OF ret*] *return-time-returns*[*OF ret*] **by** *force*

lemma *return-time-le*:
assumes *ret*: *returns-to P x closed P*
assumes *flow-not*: *flow0 x t* ∈ *P t* > *0*
shows *return-time P x* ≤ *t*
using *return-time-least*[*OF assms*(*1*,*2*), *of t*] *flow-not*
by *force*

lemma *returns-to-laterI*:
assumes *ret*: *returns-to P x closed P*
assumes *t*: *t* > *0 t* ∈ *existence-ivl0 x*
assumes *flow-not*: ⋀*s*. *0* < *s* ⟹ *s* ≤ *t* ⟹ *flow0 x s* ∉ *P*
shows *returns-to P* (*flow0 x t*)
apply (*rule returns-toI*[*of return-time P x* − *t*])
subgoal using *flow-not* **by** (*auto intro*!: *return-time-gt ret*)
subgoal by (*auto intro*!: *existence-ivl-trans′ return-time-exivl ret t*)
subgoal by (*subst flow-trans*[*symmetric*])
(*auto intro*!: *existence-ivl-trans′ return-time-exivl ret t return-time-returns*)
subgoal
proof −
have $\forall_F$ *y in nhds 0*. *y* ∈ *existence-ivl0* (*flow0 x t*)
apply (*rule eventually-nhds-in-open*[*OF open-existence-ivl*[*of flow0 x t*] *existence-ivl-zero*])
apply (*rule flow-in-domain*)
apply *fact*
done

then have $\forall_F$ *s in at-right 0. s* $\in$ *existence-ivl0* (*flow0 x t*)
unfolding *eventually-at-filter*
by *eventually-elim auto*
moreover
have $\forall_F$ *s in at-right 0. t* + *s* < *return-time P x*
using *return-time-gt*[*OF ret flow-not*, *of t*]
by (*auto simp*: *eventually-at-right*[*OF zero-less-one*] *intro*!: *exI*[*of - return-time P x* − *t*])
moreover
have $\forall_F$ *s in at-right 0. 0* < *t* + *s*
by (*metis* (*mono-tags*) *eventually-at-rightI greaterThanLessThan-iff pos-add-strict t*(*1*))
ultimately show *?thesis*
apply *eventually-elim*
apply (*subst flow-trans*[*symmetric*])
using *return-time-least*[*OF ret*]
by (*auto intro*!: *existence-ivl-trans′ t*)
qed
subgoal by *fact*
done

lemma *never-returns*:
assumes ¬*returns-to P x*
assumes *closed P t* ≥ *0 t* ∈ *existence-ivl0 x*
assumes *ev*: $\forall_F$ *t in at-right 0. flow0 x t* ∉ *P*
shows ¬*returns-to P* (*flow0 x t*)
using *returns-to-earlierI*[*OF - assms*(*2*−*5*)] *assms*(*1*)
by *blast*

lemma *return-time-eqI*:
assumes *closed P*
and *t-pos*: *t* > *0*
and *ex*: *t* ∈ *existence-ivl0 x*
and *ret*: *flow0 x t* ∈ *P*
and *least*: ⋀*s. 0* < *s* ⟹ *s* < *t* ⟹ *flow0 x s* ∉ *P*
shows *return-time P x* = *t*
proof −
from *least t-pos* **have** $\forall_F$ *t in at-right 0. flow0 x t* ∉ *P*
by (*auto simp*: *eventually-at-right*[*OF zero-less-one*])
then have *returns-to P x*
by (*auto intro*!: *returns-toI*[*of t*] *assms*)
then show *?thesis*
using *least*
by (*auto simp*: *return-time-def t-pos ex ret*
intro!: *some1-equality*[*OF return-time-ex1*[*OF* ‹*returns-to - -*› ‹*closed -*›]])
qed

lemma *return-time-step*:
assumes *returns-to P* (*flow0 x t*)

assumes *closed P*
assumes *flow-not*: $\bigwedge s.\ 0 < s \Longrightarrow s \leq t \Longrightarrow$ *flow0 x s* $\notin P$
assumes *t*: $t > 0$ $t \in$ *existence-ivl0 x*
shows *return-time P* (*flow0 x t*) = *return-time P x* − *t*
proof −
from *flow-not t* **have** $\forall_F$ *t in at-right 0. flow0 x t* $\notin P$
by (*auto simp*: *eventually-at-right*[*OF zero-less-one*])
from *returns-to-earlierI*[*OF assms*(*1*,*2*) *less-imp-le*, *OF t this*]
have *ret*: *returns-to P x* **.**
from *return-time-gt*[*OF ret* ‹*closed P*› *flow-not*]
have $t <$ *return-time P x* **by** *simp*
moreover
have $0 < s \Longrightarrow s <$ *return-time P x* − $t \Longrightarrow$ *flow0* (*flow0 x t*) *s* = *flow0 x* (*t* + *s*) **for** *s*
using *ivl-subset-existence-ivl*[*OF return-time-exivl*[*OF ret* ‹*closed* -›]] *t*
by (*subst flow-trans*) (*auto intro*!: *existence-ivl-trans'*)
ultimately show *?thesis*
using *flow-not assms*(*1*) *ret return-time-least t*(*1*)
by (*auto intro*!: *return-time-eqI return-time-returns ret*
simp: *flow-trans*[*symmetric*] ‹*closed P*› *t*(*2*) *existence-ivl-trans' return-time-exivl*)
qed

definition *poincare-map P x* = *flow0 x* (*return-time P x*)

lemma *poincare-map-step-flow*:
assumes *ret*: *returns-to P x closed P*
assumes *flow-not*: $\bigwedge s.\ 0 < s \Longrightarrow s \leq t \Longrightarrow$ *flow0 x s* $\notin P$
assumes *t*: $t > 0$ $t \in$ *existence-ivl0 x*
shows *poincare-map P* (*flow0 x t*) = *poincare-map P x*
unfolding *poincare-map-def*
apply (*subst flow-trans*[*symmetric*])
subgoal by *fact*
subgoal using *flow-not* **by** (*auto intro*!: *return-time-exivl returns-to-laterI t ret*)
subgoal
using *flow-not*
by (*subst return-time-step*) (*auto intro*!: *return-time-exivl returns-to-laterI t ret*)
done

lemma *poincare-map-returns*:
assumes *returns-to P x closed P*
shows *poincare-map P x* $\in P$
by (*auto intro*!: *return-time-returns assms simp*: *poincare-map-def*)

lemma *poincare-map-onto*:
assumes *closed P*
assumes $0 < t$ $t \in$ *existence-ivl0 x* $\forall_F$ *t in at-right 0. flow0 x t* $\notin P$
assumes *flow0 x t* $\in P$
shows *poincare-map P x* $\in$ *flow0 x* ‘ $\{0 <..\ t\} \cap P$
proof (*rule IntI*)

have *returns-to P x*
by (*rule returns-toI*) (*rule assms*)+
then have *return-time P x* ∈ {*0*<..*t*}
by (*auto intro*!: *return-time-pos assms return-time-le*)
then show *poincare-map P x* ∈ *flow0 x* ‘ {*0*<..*t*}
by (*auto simp*: *poincare-map-def*)
show *poincare-map P x* ∈ *P*
by (*auto intro*!: *poincare-map-returns* ‹*returns-to* - -› ‹*closed* -›)
qed

end

lemma *isCont-blinfunD*:
fixes *f'*::*'a*::*metric-space* ⇒ *'b*::*real-normed-vector* $\Rightarrow_L$ *'c*::*real-normed-vector*
assumes *isCont f' a 0* < *e*
shows ∃ *d*>*0*. ∀ *x*. *dist a x* < *d* ⟶ *onorm* (λ*v*. *blinfun-apply* (*f' x*) *v* − *blinfun-apply* (*f' a*) *v*) < *e*
proof −
have $\forall_F$ *x in at a*. *dist* (*f' x*) (*f' a*) < *e*
using *assms isCont-def tendsto-iff* **by** *blast*
then show *?thesis*
using ‹*e* > *0*› *norm-eq-zero*
by (*force simp*: *eventually-at dist-commute dist-norm norm-blinfun.rep-eq*
simp flip: *blinfun.bilinear-simps*)
qed

proposition *has-derivative-locally-injective-blinfun*:
fixes *f* :: *'n*::*euclidean-space* ⇒ *'m*::*euclidean-space*
and *f'*::*'n* ⇒ *'n* $\Rightarrow_L$ *'m*
and *g'*::*'m* $\Rightarrow_L$ *'n*
assumes *a* ∈ *s*
and *open s*
and *g'*: *g'* o_L (*f' a*) = 1_L
and *f'*: ⋀*x*. *x* ∈ *s* ⟹ (*f has-derivative f' x*) (*at x*)
and *c*: *isCont f' a*
obtains *r* **where** *r* > *0 ball a r* ⊆ *s inj-on f* (*ball a r*)
proof −
have *bl*: *bounded-linear* (*blinfun-apply g'*)
by (*auto simp*: *blinfun.bounded-linear-right*)
from *g'* **have** *g'*: *blinfun-apply g'* ∘ *blinfun-apply* (*f' a*) = *id*
by *transfer* (*simp add*: *id-def*)
from *has-derivative-locally-injective*[*OF* ‹*a* ∈ *s*› ‹*open s*› *bl g' f' isCont-blinfunD*[*OF c*]]
obtain *r* **where** *0* < *r ball a r* ⊆ *s inj-on f* (*ball a r*)
by *auto*
then show *?thesis* **..**
qed

lift-definition *embed1-blinfun*::'*a*::*real-normed-vector* $\Rightarrow_L$ ('*a*∗'*b*::*real-normed-vector*)
is $\lambda x.$ (*x*, *0*)
  by *standard* (*auto intro*!: *exI*[**where** *x*=*1*])
lemma *blinfun-apply-embed1-blinfun*[*simp*]: *blinfun-apply embed1-blinfun x* = (*x*, *0*)
  by *transfer simp*

lift-definition *embed2-blinfun*::'*a*::*real-normed-vector* $\Rightarrow_L$ ('*b*::*real-normed-vector*∗'*a*)
is $\lambda x.$ (*0*, *x*)
  by *standard* (*auto intro*!: *exI*[**where** *x*=*1*])
lemma *blinfun-apply-embed2-blinfun*[*simp*]: *blinfun-apply embed2-blinfun x* = (*0*, *x*)
  by *transfer simp*

lemma *blinfun-inverseD*: $f\ o_L\ f' = 1_L \Longrightarrow f\ (f'\ x) = x$
  apply *transfer*
  unfolding *o-def*
  by *meson*

lemmas *continuous-on-open-vimageI* = *continuous-on-open-vimage*[*THEN iffD1*, *rule-format*]
lemmas *continuous-on-closed-vimageI* = *continuous-on-closed-vimage*[*THEN iffD1*, *rule-format*]

lemma *ball-times-subset*: $ball\ a\ (c/2) \times ball\ b\ (c/2) \subseteq ball\ (a, b)\ c$
proof –
  {
    fix $a'\ b'$
    have $sqrt\ ((dist\ a\ a')^2 + (dist\ b\ b')^2) \le dist\ a\ a' + dist\ b\ b'$
      by (*rule real-le-lsqrt*) (*auto simp*: *power2-eq-square algebra-simps*)
    also assume $a' \in ball\ a\ (c\ /\ 2)$
    then have $dist\ a\ a' < c\ /\ 2$ **by** (*simp add*:)
    also assume $b' \in ball\ b\ (c\ /\ 2)$
    then have $dist\ b\ b' < c\ /\ 2$ **by** (*simp add*:)
    finally have $sqrt\ ((dist\ a\ a')^2 + (dist\ b\ b')^2) < c$
      by *simp*
  } thus *?thesis* **by** (*auto simp*: *dist-prod-def mem-cball*)
qed

lemma *linear-inverse-blinop-lemma*:
  fixes *w*::'*a*::{*banach*, *perfect-space*} *blinop*
  assumes *norm w* < *1*
  shows
    $summable\ (\lambda n.\ (-1)\hat{}n *_R w\hat{}n)$ (**is** *?C*)
    $(\sum n.\ (-1)\hat{}n *_R w\hat{}n) * (1 + w) = 1$ (**is** *?I1*)
    $(1 + w) * (\sum n.\ (-1)\hat{}n *_R w\hat{}n) = 1$ (**is** *?I2*)
    $norm\ ((\sum n.\ (-1)\hat{}n *_R w\hat{}n) - 1 + w) \le (norm\ w)^2/(1 - norm\ (w))$ (**is** *?L*)
proof –
  have $summable\ (\lambda n.\ norm\ w\ \hat{}\ n)$

```
    apply (rule summable-geometric)
    using assms by auto
  then have summable (λn. norm (w ^ n))
   by (rule summable-comparison-test'[where N=0]) (auto intro!: norm-power-ineq)
  then show ?C
    by (rule summable-comparison-test'[where N=0]) (auto simp: norm-power )
  {
    fix N
    have 1: (1 + w) * sum (λn. (−1)^n *R w^n) {..<N} = sum (λn. (−1)^n *R
w^n) {..<N} * (1 + w)
        by (auto simp: algebra-simps sum-distrib-left sum-distrib-right sum.distrib
power-commutes)
   also have ... = sum (λn. (−1)^n *R w^n − (−1)^Suc n *R w^Suc n) {..<N}
        by (auto simp: algebra-simps sum-distrib-left sum-distrib-right sum.distrib
power-commutes)
    also have ... = 1 − (−1)^N *R w^N
      by (subst sum-lessThan-telescope') (auto simp: algebra-simps)
    finally have (1 + w) * (∑n<N. (− 1) ^ n *R w ^ n) = 1 − (− 1) ^ N *R
w ^ N .
    note 1 this
  } note nu = this
  show ?I1
    apply (subst suminf-mult2, fact)
    apply (subst suminf-eq-lim)
    apply (subst sum-distrib-right[symmetric])
    apply (rule limI)
    apply (subst nu(1)[symmetric])
    apply (subst nu(2))
    apply (rule tendsto-eq-intros)
      apply (rule tendsto-intros)
     apply (rule tendsto-norm-zero-cancel)
     apply auto
    apply (rule Lim-transform-bound[where g=λi. norm w ^ i])
     apply (rule eventuallyI)
    apply simp apply (rule norm-power-ineq)
    apply (auto intro!: LIMSEQ-power-zero assms)
    done
  show ?I2
    apply (subst suminf-mult[symmetric], fact)
    apply (subst suminf-eq-lim)
    apply (subst sum-distrib-left[symmetric])
    apply (rule limI)
    apply (subst nu(2))
    apply (rule tendsto-eq-intros)
      apply (rule tendsto-intros)
     apply (rule tendsto-norm-zero-cancel)
     apply auto
    apply (rule Lim-transform-bound[where g=λi. norm w ^ i])
     apply (rule eventuallyI)
```

apply *simp* **apply** (*rule norm-power-ineq*)
apply (*auto intro*!: *LIMSEQ-power-zero assms*)
done
have ∗: $(\sum n. (- 1) \widehat{\ } n *_R w \widehat{\ } n) - 1 + w = (w^2 * (\sum n. (- 1) \widehat{\ } n *_R w \widehat{\ } n))$
apply (*subst suminf-split-initial-segment*[**where** *k*=*2*], *fact*)
apply (*subst suminf-mult*[*symmetric*], *fact*)
by (*auto simp*: *power2-eq-square algebra-simps eval-nat-numeral*)
also have *norm* ... $\leq$ $(norm\ w)^2\ /\ (1 - norm\ w)$
proof −
have §: *norm* $(\sum n. (- 1) \widehat{\ } n *_R w \widehat{\ } n) \leq 1\ /\ (1 - norm\ w)$
apply (*rule order-trans*[*OF summable-norm*])
apply *auto*
apply *fact*
apply (*rule order-trans*[*OF suminf-le*])
apply (*rule norm-power-ineq*)
apply *fact*
apply *fact*
by (*auto simp*: *suminf-geometric assms*)
show *?thesis*
apply (*rule order-trans*[*OF norm-mult-ineq*])
apply (*subst divide-inverse*)
apply (*rule mult-mono*)
apply (*auto simp*: *norm-power-ineq inverse-eq-divide assms* §)
done
qed
finally show *?L* **.**
qed

lemma *linear-inverse-blinfun-lemma*:
fixes *w*::$'a \Rightarrow_L 'a$::{*banach*, *perfect-space*}
assumes *norm w* < *1*
obtains *I* **where**
$I\ o_L\ (1_L + w) = 1_L\ (1_L + w)\ o_L\ I = 1_L$
norm $(I - 1_L + w) \leq (norm\ w)^2/(1 - norm\ (w))$
proof −
define *v*::*'a blinop* **where** *v* = *Blinop w*
have *norm v* = *norm w*
unfolding *v-def*
apply *transfer*
by (*simp add*: *bounded-linear-Blinfun-apply norm-blinfun.rep-eq*)
with *assms* **have** *norm v* < *1* **by** *simp*
from *linear-inverse-blinop-lemma*[*OF this*]
have *v*: $(\sum n. (- 1) \widehat{\ } n *_R v \widehat{\ } n) * (1 + v) = 1$
$(1 + v) * (\sum n. (- 1) \widehat{\ } n *_R v \widehat{\ } n) = 1$
norm $((\sum n. (- 1) \widehat{\ } n *_R v \widehat{\ } n) - 1 + v) \leq (norm\ v)^2\ /\ (1 - norm\ v)$
by *auto*
define *J*::*'a blinop* **where** $J = (\sum n. (- 1) \widehat{\ } n *_R v \widehat{\ } n)$
define *I*::$'a \Rightarrow_L 'a$ **where** *I* = *Blinfun J*

have *Blinfun* (*blinop-apply J*) − 1_L + *w* = *Rep-blinop* (*J* − *1* + *Blinop* (*blinfun-apply w*))
  by *transfer′* (*auto simp*: *blinfun-apply-inverse*)
then have *ne*: *norm* (*Blinfun* (*blinop-apply J*) − 1_L + *w*) =
  norm (*J* − *1* + *Blinop* (*blinfun-apply w*))
  by (*auto simp*: *norm-blinfun-def norm-blinop-def*)
from *v* **have**
  I o_L (1_L + *w*) = 1_L (1_L + *w*) o_L *I* = 1_L
  norm (*I* − 1_L + *w*) ≤ (*norm w*)^2/(*1* − *norm* (*w*))
    apply (*auto simp*: *I-def J-def*[*symmetric*])
  unfolding *v-def*
    apply (*auto simp*: *blinop.bounded-linear-right bounded-linear-Blinfun-apply*
      intro!: *blinfun-eqI*)
  subgoal by *transfer*
    (*auto simp*: *blinfun-ext blinfun.bilinear-simps bounded-linear-Blinfun-apply*)
  subgoal
    by *transfer* (*auto simp*: *Transfer.Rel-def*
      blinfun-ext blinfun.bilinear-simps bounded-linear-Blinfun-apply)
  subgoal
    apply (*auto simp*: *ne*)
    apply *transfer*
    by (*auto simp*: *norm-blinfun-def bounded-linear-Blinfun-apply*)
  done
then show *?thesis* **..**
qed

definition *invertibles-blinfun* = {*w*. ∃ *wi*. *w* o_L *wi* = 1_L ∧ *wi* o_L *w* = 1_L}

lemma *blinfun-inverse-open*:— 8.3.2 in Dieudonne, TODO: add continuity and derivative
  shows *open* (*invertibles-blinfun*::
    (*′a*::{*banach*, *perfect-space*} ⇒$_L$ *′b*::*banach*) *set*)
proof (*rule openI*)
  fix *u0*::*′a* ⇒$_L$ *′b*
  assume *u0* ∈ *invertibles-blinfun*
  then obtain *u0i* **where** *u0i*: *u0* o_L *u0i* = 1_L *u0i* o_L *u0* = 1_L
    by (*auto simp*: *invertibles-blinfun-def*)
  then have [*simp*]: *u0i* ≠ *0*
    apply (*auto*)
    by (*metis one-blinop.abs-eq zero-blinop.abs-eq zero-neq-one*)
  let *?e* = *inverse* (*norm u0i*)
  show ∃ *e*>*0*. *ball u0 e* ⊆ *invertibles-blinfun*
    apply (*clarsimp intro*!: *exI*[**where** *x* = *?e*] *simp*: *invertibles-blinfun-def*)
    subgoal premises *prems* **for** *u0s*
    proof −
      define *s* **where** *s* = *u0s* − *u0*
      have *u0s*: *u0s* = *u0* + *s*
        by (*auto simp*: *s-def*)
      have *norm* (*u0i* o_L *s*) < *1*

using *prems* **by** (*auto simp*: *dist-norm u0s*
divide-simps ac-simps intro!: *le-less-trans*[*OF norm-blinfun-compose*])
from *linear-inverse-blinfun-lemma*[*OF this*]
obtain *I* **where** *I*:
$I\ o_L\ 1_L + (u0i\ o_L\ s) = 1_L$
$1_L + (u0i\ o_L\ s)\ o_L\ I = 1_L$
$norm\ (I - 1_L + (u0i\ o_L\ s)) \leq (norm\ (u0i\ o_L\ s))^2\ /\ (1 - norm\ (u0i\ o_L\ s))$
by *auto*
have *u0s-eq*: $u0s = u0\ o_L\ (1_L + (u0i\ o_L\ s))$
using *u0i*
by (*auto simp*: *s-def blinfun.bilinear-simps blinfun-ext*)
show *?thesis*
apply (*rule exI*[**where** *x*=*I* o_L *u0i*])
using *I u0i*
apply (*auto simp*: *u0s-eq*)
by (*auto simp*: *algebra-simps blinfun-ext blinfun.bilinear-simps*)
qed
done
qed

lemma *blinfun-compose-assoc*[*ac-simps*]: $a\ o_L\ b\ o_L\ c = a\ o_L\ (b\ o_L\ c)$
by (*auto intro*!: *blinfun-eqI*)

TODO: move *norm* (− *?x*) = *norm ?x* to class!

lemma (**in** *real-normed-vector*) *norm-minus-cancel* [*simp*]: *norm* (− *x*) = *norm x*
proof −
have *scaleR-minus-left*: $- a *_R x = - (a *_R x)$ **for** *a x*
proof −
have ∀ *x1 x2*. (*x2*::*real*) + *x1* = *x1* + *x2*
by *auto*
then have *f1*: ∀ *r ra a*. $(ra + r) *_R (a::'a) = r *_R a + ra *_R a$
using *local.scaleR-add-left* **by** *presburger*
have *f2*: *a* + *a* = *2* ∗ *a*
by *force*
have *f3*: *2* ∗ *a* + − *1* ∗ *a* = *a*
by *auto*
have − *a* = − *1* ∗ *a*
by *auto*
then show *?thesis*
using *f3 f2 f1* **by** (*metis local.add-minus-cancel local.add-right-imp-eq*)
qed
have *norm* (− *x*) = *norm* (*scaleR* (− *1*) *x*)
by (*simp only*: *scaleR-minus-left scaleR-one*)
also have ... = |− *1*| ∗ *norm x*
by (*rule norm-scaleR*)
finally show *?thesis* **by** *simp*
qed

TODO: move *norm* (*?a* − *?b*) = *norm* (*?b* − *?a*) to class!

lemma (**in** *real-normed-vector*) *norm-minus-commute*: *norm* (*a* − *b*) = *norm* (*b* − *a*)
proof −
  have *norm* (− (*b* − *a*)) = *norm* (*b* − *a*)
    by (*rule norm-minus-cancel*)
  then show *?thesis* **by** *simp*
qed

instance *euclidean-space* ⊆ *banach*
  by *standard*

lemma *blinfun-apply-Pair-split*:
  blinfun-apply g (*a*, *b*) = *blinfun-apply g* (*a*, *0*) + *blinfun-apply g* (*0*, *b*)
  unfolding *blinfun.bilinear-simps*[*symmetric*] **by** *simp*

lemma *blinfun-apply-Pair-add2*: *blinfun-apply f* (*0*, *a* + *b*) = *blinfun-apply f* (*0*, *a*) + *blinfun-apply f* (*0*, *b*)
  unfolding *blinfun.bilinear-simps*[*symmetric*] **by** *simp*

lemma *blinfun-apply-Pair-add1*: *blinfun-apply f* (*a* + *b*, *0*) = *blinfun-apply f* (*a*, *0*) + *blinfun-apply f* (*b*, *0*)
  unfolding *blinfun.bilinear-simps*[*symmetric*] **by** *simp*

lemma *blinfun-apply-Pair-minus2*: *blinfun-apply f* (*0*, *a* − *b*) = *blinfun-apply f* (*0*, *a*) − *blinfun-apply f* (*0*, *b*)
  unfolding *blinfun.bilinear-simps*[*symmetric*] **by** *simp*

lemma *blinfun-apply-Pair-minus1*: *blinfun-apply f* (*a* − *b*, *0*) = *blinfun-apply f* (*a*, *0*) − *blinfun-apply f* (*b*, *0*)
  unfolding *blinfun.bilinear-simps*[*symmetric*] **by** *simp*

lemma *implicit-function-theorem*:
  fixes *f*::′*a*::*euclidean-space* ∗ ′*b*::*euclidean-space* ⇒ ′*c*::*euclidean-space*— TODO: generalize?!
  assumes [*derivative-intros*]: ⋀*x*. *x* ∈ *S* ⟹ (*f has-derivative blinfun-apply* (*f′ x*)) (*at x*)
  assumes *S*: (*x*, *y*) ∈ *S open S*
  assumes *DIM*(′*c*) ≤ *DIM*(′*b*)
  assumes *f′C*: *isCont f′* (*x*, *y*)
  assumes *f* (*x*, *y*) = *0*
  assumes *T2*: *T* o_L (*f′* (*x*, *y*) o_L *embed2-blinfun*) = 1_L
  assumes *T1*: (*f′* (*x*, *y*) o_L *embed2-blinfun*) o_L *T* = 1_L— TODO: reduce?!
  obtains *u e r*
  where *f* (*x*, *u x*) = *0* *u x* = *y*
    ⋀*s*. *s* ∈ *cball x e* ⟹ *f* (*s*, *u s*) = *0*
    continuous-on (*cball x e*) *u*
    (λ*t*. (*t*, *u t*)) ‘ *cball x e* ⊆ *S*
    e > *0*
    (*u has-derivative* − *T* o_L *f′* (*x*, *y*) o_L *embed1-blinfun*) (*at x*)

$r > 0$
$\bigwedge U\ v\ s.\ v\ x = y \Longrightarrow (\bigwedge s.\ s \in U \Longrightarrow f\ (s,\ v\ s) = 0) \Longrightarrow U \subseteq cball\ x\ e \Longrightarrow$
$continuous\text{-}on\ U\ v \Longrightarrow s \in U \Longrightarrow (s,\ v\ s) \in ball\ (x,\ y)\ r \Longrightarrow u\ s = v\ s$
proof −
define H **where** $H \equiv \lambda(x,\ y).\ (x,\ f\ (x,\ y))$
define H' **where** $H' \equiv \lambda x.\ (embed1\text{-}blinfun\ o_L\ fst\text{-}blinfun) + (embed2\text{-}blinfun\ o_L\ (f'\ x))$
have $f'\text{-}inv$: $f'\ (x,\ y)\ o_L\ embed2\text{-}blinfun \in invertibles\text{-}blinfun$
using *T1 T2* **by** (*auto simp*: *invertibles-blinfun-def ac-simps intro*!: *exI*[**where** $x=T$])
from *openE*[*OF blinfun-inverse-open this*]
obtain *d0* **where** *e0*: $0 < d0$
$ball\ (f'\ (x,\ y)\ o_L\ embed2\text{-}blinfun)\ d0 \subseteq invertibles\text{-}blinfun$
by *auto*
have $isCont\ (\lambda s.\ f'\ s\ o_L\ embed2\text{-}blinfun)\ (x,\ y)$
by (*auto intro*!: *continuous-intros f′C*)
from *this*[*unfolded isCont-def*, *THEN tendstoD*, *OF* ‹$0 < d0$›]
have $\forall_F\ s\ in\ at\ (x,\ y).\ f'\ s\ o_L\ embed2\text{-}blinfun \in invertibles\text{-}blinfun$
apply *eventually-elim*
using *e0* **by** (*auto simp*: *subset-iff dist-commute*)
then obtain *e0* **where** $e0 > 0$
$xa \neq (x,\ y) \Longrightarrow dist\ xa\ (x,\ y) < e0 \Longrightarrow$
$f'\ xa\ o_L\ embed2\text{-}blinfun \in invertibles\text{-}blinfun$ **for** *xa*
unfolding *eventually-at*
by *auto*
then have *e0*: $e0 > 0$
$dist\ xa\ (x,\ y) < e0 \Longrightarrow f'\ xa\ o_L\ embed2\text{-}blinfun \in invertibles\text{-}blinfun$ **for** *xa*
apply −
subgoal by *simp*
using *f′-inv*
apply (*cases* $xa = (x,\ y)$)
by *auto*

have H': $x \in S \Longrightarrow (H\ has\text{-}derivative\ H'\ x)\ (at\ x)$ **for** *x*
unfolding *H-def H′-def*
by (*auto intro*!: *derivative-eq-intros ext simp*: *blinfun.bilinear-simps*)
have *cH′*: $isCont\ H'\ (x,\ y)$
unfolding *H′-def*
by (*auto intro*!: *continuous-intros assms*)
have *linear-H′*: $\bigwedge s.\ s \in S \Longrightarrow linear\ (H'\ s)$
using *H′ assms*(*2*) *has-derivative-linear* **by** *blast*
have ∗: $blinfun\text{-}apply\ T\ (blinfun\text{-}apply\ (f'\ (x,\ y))\ (0,\ b)) = b$ **for** *b*
using *blinfun-inverseD*[*OF T2*, *of b*]
by *simp*
have $inj\ (f'\ (x,\ y)\ o_L\ embed2\text{-}blinfun)$
by (*metis* (*no-types*, *lifting*) ∗ *blinfun-apply-blinfun-compose embed2-blinfun.rep-eq injI*)
then have [*simp*]: $blinfun\text{-}apply\ (f'\ (x,\ y))\ (0,\ b) = 0 \Longrightarrow b = 0$ **for** *b*

```
    apply (subst (asm) linear-injective-0)
    subgoal
      apply (rule bounded-linear.linear)
      apply (rule blinfun.bounded-linear-right)
      done
    subgoal by simp
    done
  have inj (H′ (x, y))
    apply (subst linear-injective-0)
     apply (rule linear-H′)
     apply fact
    apply (auto simp: H′-def blinfun.bilinear-simps zero-prod-def)
    done
  define Hi where Hi = (embed1-blinfun o_L fst-blinfun) + ((embed2-blinfun o_L T
o_L (snd-blinfun − (f′ (x, y) o_L embed1-blinfun o_L fst-blinfun))))
   have Hi′: (λu. snd (blinfun-apply Hi (u, 0))) = − T o_L f′ (x, y) o_L em-
bed1-blinfun
    by (auto simp: Hi-def blinfun.bilinear-simps)
  have Hi: Hi o_L H′ (x, y) = 1_L
    apply (auto simp: H′-def fun-eq-iff blinfun.bilinear-simps Hi-def
        intro!: ext blinfun-eqI)
    apply (subst blinfun-apply-Pair-split)
    by (auto simp: ∗ blinfun.bilinear-simps)
  from has-derivative-locally-injective-blinfun[OF S this H′ cH′]
  obtain r0 where r0: 0 < r0 ball (x, y) r0 ⊆ S and inj: inj-on H (ball (x, y)
r0)
    by auto
  define r where r = min r0 e0
  have r: 0 < r ball (x, y) r ⊆ S and inj: inj-on H (ball (x, y) r)
   and r-inv: ⋀s. s ∈ ball (x, y) r ⟹ f′ s o_L embed2-blinfun ∈ invertibles-blinfun
    subgoal using e0 r0 by (auto simp: r-def)
    subgoal using e0 r0 by (auto simp: r-def)
    subgoal using inj apply (rule inj-on-subset)
      using e0 r0 by (auto simp: r-def)
    subgoal for s
      using e0 r0 by (auto simp: r-def dist-commute)
    done
  obtain i::′a where i ∈ Basis
    using nonempty-Basis by blast
  define undef where undef ≡ (x, y) + r ∗_R (i, 0)— really??
  have ud: ¬ dist (x, y) undef < r
    using ‹r > 0› ‹i ∈ Basis› by (auto simp: undef-def dist-norm)
  define G where G ≡ the-inv-into (ball (x, y) r) H
  {
    fix u v
    assume [simp]: (u, v) ∈ H ‘ ball (x, y) r
    note [simp] = inj
    have (u, v) = H (G (u, v))
      unfolding G-def
```

```
      by (subst f-the-inv-into-f[where f=H]) auto
    moreover have ... = H (G (u, v))
      by (auto simp: G-def)
    moreover have ... = (fst (G (u, v)), f (G (u, v)))
      by (auto simp: H-def split-beta')
    ultimately have u = fst (G (u, v)) v = f (G (u, v)) by simp-all
    then have f (u, snd (G(u, v))) = v u = fst (G (u, v))
      by (metis prod.collapse)+
  } note uvs = this
  note uv = uvs(1)
  moreover
  have f (x, snd (G (x, 0))) = 0
    apply (rule uv)
     by (metis (mono-tags, lifting) H-def assms(6) case-prod-beta' centre-in-ball
fst-conv image-iff r(1) snd-conv)
  moreover
  have cH: continuous-on S H
    apply (rule has-derivative-continuous-on)
    apply (subst at-within-open)
      apply (auto intro!: H' assms)
    done
  have inj2: inj-on H (ball (x, y) (r / 2))
    apply (rule inj-on-subset, rule inj)
    using r by auto
  have oH: open (H ` ball (x, y) (r/2))
    apply (rule invariance-of-domain-gen)
       apply (auto simp: assms inj)
    apply (rule continuous-on-subset)
      apply fact
    using r
     apply auto
    using inj2 apply simp
    done
  have (x, f (x, y)) ∈ H ` ball (x, y) (r/2)
    using ‹r > 0› by (auto simp: H-def)
  from open-contains-cball[THEN iffD1, OF oH, rule-format, OF this]
  obtain e' where e': e' > 0 cball (x, f (x, y)) e' ⊆ H ` ball (x, y) (r/2)
    by auto

  have inv-subset: the-inv-into (ball (x, y) r) H a = the-inv-into R H a
    if a ∈ H ` R R ⊆ (ball (x, y) r)
    for a R
    apply (rule the-inv-into-f-eq[OF inj])
     apply (rule f-the-inv-into-f)
      apply (rule inj-on-subset[OF inj])
      apply fact
     apply fact
    apply (rule the-inv-into-into)
      apply (rule inj-on-subset[OF inj])
```

apply *fact*
apply *fact*
apply (*rule order-trans*)
apply *fact*
using *r* **apply** *auto*
done
have *GH*: $G\ (H\ z) = z$ **if** $dist\ (x,\ y)\ z < r$ **for** *z*
by (*auto simp*: *G-def the-inv-into-f-f inj that*)
define *e* **where** $e = min\ (e'\ /\ 2)\ e0$
define *r2* **where** $r2 = r\ /\ 2$
have *r2*: $r2 > 0$ $r2 < r$
using ‹$r > 0$› **by** (*auto simp*: *r2-def*)
have $e > 0$ **using** *e′ e0* **by** (*auto simp*: *e-def*)
from *cball-times-subset*[*of x e′ f* (*x*, *y*)] *e′*
have $cball\ x\ e \times cball\ (f\ (x,\ y))\ e \subseteq H$ ‘ $ball\ (x,\ y)\ (r/2)$
by (*force simp*: *e-def*)
then have *e-r-subset*: $z \in cball\ x\ e \Longrightarrow (z,\ 0) \in H$ ‘ $ball\ (x,\ y)\ (r/2)$ **for** *z*
using ‹$0 < e$› *assms*(*6*)
by (*auto simp*: *H-def subset-iff*)
have *u0*: $(u,\ 0) \in H$ ‘ $ball\ (x,\ y)\ r$ **if** $u \in cball\ x\ e$ **for** *u*
apply (*rule rev-subsetD*)
apply (*rule e-r-subset*)
apply *fact*
unfolding *r2-def* **using** *r2* **by** *auto*
have *G-r*: $G\ (u,\ 0) \in ball\ (x,\ y)\ r$ **if** $u \in cball\ x\ e$ **for** *u*
unfolding *G-def*
apply (*rule the-inv-into-into*)
apply *fact*
apply (*auto*)
apply (*rule u0*, *fact*)
done
note *e-r-subset*
ultimately have *G2*:
$f\ (x,\ snd\ (G\ (x,\ 0))) = 0$ $snd\ (G\ (x,\ 0)) = y$
$\bigwedge u.\ u \in cball\ x\ e \Longrightarrow f\ (u,\ snd\ (G\ (u,\ 0))) = 0$
continuous-on (*cball x e*) ($\lambda u.\ snd\ (G\ (u,\ 0))$)
($\lambda t.\ (t,\ snd\ (G\ (t,\ 0)))$) ‘ $cball\ x\ e \subseteq S$
$e > 0$
(($\lambda u.\ snd\ (G\ (u,\ 0))$) *has-derivative* ($\lambda u.\ snd\ (Hi\ (u,\ 0))$)) (*at x*)
apply (*auto simp*: *G-def split-beta′*
intro!: *continuous-intros continuous-on-compose2*[*OF cH*])
subgoal premises *prems*
proof −
have *the-inv-into* (*ball* (*x*, *y*) *r*) *H* (*x*, *0*) = (*x*, *y*)
apply (*rule the-inv-into-f-eq*)
apply *fact*
by (*auto simp*: *H-def assms* ‹$r > 0$›)
then show *?thesis*
by *auto*

```
qed
using r2(2) r2-def apply fastforce
apply (subst continuous-on-cong[OF refl])
 apply (rule inv-subset[where R=cball (x, y) r2])
subgoal
  using r2
  apply auto
  using r2-def by force
subgoal using r2 by (force simp:)
subgoal
  apply (rule continuous-on-compose2[OF continuous-on-inv-into])
  using r(2) r2(2)
    apply (auto simp: r2-def[symmetric]
      intro!: continuous-on-compose2[OF cH] continuous-intros)
   apply (rule inj-on-subset)
    apply (rule inj)
  using r(2) r2(2) apply force
  apply force
  done
subgoal premises prems for u
proof −
  from prems have u: u ∈ cball x e by auto
  note G-r[OF u]
  also have ball (x, y) r ⊆ S
    using r by simp
  finally have (G (u, 0)) ∈ S .
  then show ?thesis
    unfolding G-def[symmetric]
    using uvs(2)[OF u0, OF u]
    by (metis prod.collapse)
qed
subgoal using ‹e > 0› by simp
subgoal premises prems
proof −
  have (x, y) ∈ cball (x, y) r2
    using r2
    by auto
  moreover
  have H (x, y) ∈ interior (H ‘ cball (x, y) r2)
    apply (rule interiorI[OF oH])
    using r2 by (auto simp: r2-def)
  moreover
  have cball (x, y) r2 ⊆ S
    using r r2 by auto
  moreover have ⋀z. z ∈ cball (x, y) r2 ⟹ G (H z) = z
    using r2 by (auto intro!: GH)
  ultimately have (G has-derivative Hi) (at (H (x, y)))
  proof (rule has-derivative-inverse[where g = G and f = H,
        OF compact-cball - - continuous-on-subset[OF cH] - H′ - -])
```

```
      show blinfun-apply Hi ∘ blinfun-apply (H' (x, y)) = id
        using Hi by transfer auto
    qed (use S blinfun.bounded-linear-right in auto)
    then have g': (G has-derivative Hi) (at (x, 0))
      by (auto simp: H-def assms)
    show ?thesis
      unfolding G-def[symmetric] H-def[symmetric]
      apply (auto intro!: derivative-eq-intros)
       apply (rule has-derivative-compose[where g=G and f=λx. (x, 0)])
       apply (auto intro!: g' derivative-eq-intros)
      done
  qed
  done
moreover
note ‹r > 0›
moreover
define u where u ≡ λx. snd (G (x, 0))
have local-unique: u s = v s
  if solves: (⋀s. s ∈ U ⟹ f (s, v s) = 0)
  and i: v x = y
  and v: continuous-on U v
  and s: s ∈ U
  and s': (s, v s) ∈ ball (x, y) r
  and U: U ⊆ cball x e
  for U v s
proof −
  have H-eq: H (s, v s) = H (s, u s)
    apply (auto simp: H-def solves[OF s])
    unfolding u-def
    apply (rule G2)
    apply (rule subsetD; fact)
    done
  have (s, snd (G (s, 0))) = (G (s, 0))
    using GH H-def s s' solves by fastforce
  also have ... ∈ ball (x, y) r
    unfolding G-def
    apply (rule the-inv-into-into)
      apply fact
     apply (rule u0)
     apply (rule subsetD; fact)
    apply (rule order-refl)
    done
  finally have (s, u s) ∈ ball (x, y) r unfolding u-def .
  from inj-onD[OF inj H-eq s' this]
  show u s = v s
    by auto
qed
ultimately show ?thesis
  unfolding u-def Hi' ..
```

qed

lemma *implicit-function-theorem-unique*:
fixes f::'*a*::*euclidean-space* ∗ '*b*::*euclidean-space* ⇒ '*c*::*euclidean-space*— TODO: generalize?!
assumes f'[*derivative-intros*]: ⋀*x*. *x* ∈ *S* ⟹ (*f has-derivative blinfun-apply* (f' *x*)) (*at x*)
assumes *S*: (*x*, *y*) ∈ *S open S*
assumes *D*: *DIM*('*c*) ≤ *DIM*('*b*)
assumes $f'C$: *continuous-on S* f'
assumes *z*: *f* (*x*, *y*) = *0*
assumes *T2*: *T* o_L (f' (*x*, *y*) o_L *embed2-blinfun*) = 1_L
assumes *T1*: (f' (*x*, *y*) o_L *embed2-blinfun*) o_L *T* = 1_L— TODO: reduce?!
obtains *u e*
where *f* (*x*, *u x*) = *0 u x* = *y*
⋀*s*. *s* ∈ *cball x e* ⟹ *f* (*s*, *u s*) = *0*
continuous-on (*cball x e*) *u*
(λ*t*. (*t*, *u t*)) ‘ *cball x e* ⊆ *S*
e > *0*
(*u has-derivative* (− *T* o_L f' (*x*, *y*) o_L *embed1-blinfun*)) (*at x*)
⋀*s*. *s* ∈ *cball x e* ⟹ f' (*s*, *u s*) o_L *embed2-blinfun* ∈ *invertibles-blinfun*
⋀*U v s*. (⋀*s*. *s* ∈ *U* ⟹ *f* (*s*, *v s*) = *0*) ⟹
u x = *v x* ⟹
continuous-on U v ⟹ *s* ∈ *U* ⟹ *x* ∈ *U* ⟹ *U* ⊆ *cball x e* ⟹ *connected U* ⟹ *open U* ⟹ *u s* = *v s*
proof −
from *T1 T2* **have** $f'I$: f' (*x*, *y*) o_L *embed2-blinfun* ∈ *invertibles-blinfun*
by (*auto simp*: *invertibles-blinfun-def*)
from *assms* **have** $f'Cg$: *s* ∈ *S* ⟹ *isCont* f' *s* **for** *s*
by (*auto simp*: *continuous-on-eq-continuous-at*[*OF* ‹*open S*›])
then have $f'C$: *isCont* f' (*x*, *y*) **by** (*auto simp*: *S*)
obtain *u e1 r*
where *u*: *f* (*x*, *u x*) = *0 u x* = *y*
⋀*s*. *s* ∈ *cball x e1* ⟹ *f* (*s*, *u s*) = *0*
continuous-on (*cball x e1*) *u*
(λ*t*. (*t*, *u t*)) ‘ *cball x e1* ⊆ *S*
e1 > *0*
(*u has-derivative* (− *T* o_L f' (*x*, *y*) o_L *embed1-blinfun*)) (*at x*)
and *unique-u*: *r* > *0*
(⋀*v s U*. *v x* = *y* ⟹
(⋀*s*. *s* ∈ *U* ⟹ *f* (*s*, *v s*) = *0*) ⟹
continuous-on U v ⟹ *s* ∈ *U* ⟹ *U* ⊆ *cball x e1* ⟹ (*s*, *v s*) ∈ *ball* (*x*, *y*) *r* ⟹ *u s* = *v s*)
by (*rule implicit-function-theorem*[*OF* f' *S D* $f'C$ *z T2 T1*]; *blast*)

from *openE*[*OF blinfun-inverse-open* $f'I$] **obtain** *d* **where** *d*:
0 < *d ball* (f' (*x*, *y*) o_L *embed2-blinfun*) *d* ⊆ *invertibles-blinfun*
by *auto*
note [*continuous-intros*] = *continuous-at-compose*[*OF* - $f'Cg$, *unfolded o-def*]

from ‹*continuous-on - u*›
have *continuous-on* (*ball x e1*) *u* **by** (*rule continuous-on-subset*) *auto*
then have $\bigwedge$*s*. *s* ∈ *ball x e1* ⟹ *isCont u s*
unfolding *continuous-on-eq-continuous-at*[*OF open-ball*] **by** *auto*
note [*continuous-intros*] = *continuous-at-compose*[*OF - this*, *unfolded o-def*]
from *assms* **have** *f′Ce*: *isCont* (λ*s*. *f′* (*s*, *u s*) o_L *embed2-blinfun*) *x*
by (*auto simp*: *u intro*!: *continuous-intros*)
from *f′Ce*[*unfolded isCont-def*, *THEN tendstoD*, *OF* ‹*0* < *d*›] *d*
obtain *e0* **where** *e0* > *0* $\bigwedge$*s*. *s* ≠ *x* ⟹ *s* ∈ *ball x e0* ⟹
(*f′* (*s*, *u s*) o_L *embed2-blinfun*) ∈ *invertibles-blinfun*
by (*auto simp*: *eventually-at dist-commute subset-iff u*)
then have *e0*: *s* ∈ *ball x e0* ⟹ (*f′* (*s*, *u s*) o_L *embed2-blinfun*) ∈ *invertibles-blinfun* **for** *s*
by (*cases s* = *x*) (*auto simp*: *f′I* ‹*0* < *d*› *u*)

define *e* **where** *e* = *min* (*e0*/*2*) (*e1*/*2*)
have *e*: *f* (*x*, *u x*) = *0*
u x = *y*
$\bigwedge$*s*. *s* ∈ *cball x e* ⟹ *f* (*s*, *u s*) = *0*
continuous-on (*cball x e*) *u*
(λ*t*. (*t*, *u t*)) ‘ *cball x e* ⊆ *S*
e > *0*
(*u has-derivative* (− *T* o_L *f′* (*x*, *y*) o_L *embed1-blinfun*)) (*at x*)
$\bigwedge$*s*. *s* ∈ *cball x e* ⟹ *f′* (*s*, *u s*) o_L *embed2-blinfun* ∈ *invertibles-blinfun*
using *e0 u* ‹*e0* > *0*› **by** (*auto simp*: *e-def intro*: *continuous-on-subset*)

from *u*(*4*) **have** *continuous-on* (*ball x e1*) *u*
apply (*rule continuous-on-subset*)
using ‹*e1* > *0*›
by (*auto simp*: *e-def*)
then have $\bigwedge$*s*. *s* ∈ *cball x e* ⟹ *isCont u s*
using ‹*e0* > *0*› ‹*e1* > *0*›
unfolding *continuous-on-eq-continuous-at*[*OF open-ball*] **by** (*auto simp*: *e-def Ball-def dist-commute*)
note [*continuous-intros*] = *continuous-at-compose*[*OF - this*, *unfolded o-def*]

have *u s* = *v s*
if *solves*: ($\bigwedge$*s*. *s* ∈ *U* ⟹ *f* (*s*, *v s*) = *0*)
and *i*: *u x* = *v x*
and *v*: *continuous-on U v*
and *s*: *s* ∈ *U* **and** *U*: *x* ∈ *U U* ⊆ *cball x e connected U open U*
for *U v s*
proof −
define *M* **where** *M* = {*s* ∈ *U*. *u s* = *v s*}
have *x* ∈ *M* **using** *i U* **by** (*auto simp*: *M-def*)
moreover
have *continuous-on U* (λ*s*. *u s* − *v s*)
by (*auto intro*!: *continuous-intros v continuous-on-subset*[*OF e*(*4*) *U*(*2*)])

from *continuous-closedin-preimage*[*OF this closed-singleton*[**where** *a=0*]]
have *closedin* (*top-of-set U*) *M*
by (*auto simp*: *M-def vimage-def Collect-conj-eq*)
moreover
have $\bigwedge s.\ s \in U \implies isCont\ v\ s$
using *v*
unfolding *continuous-on-eq-continuous-at*[*OF* ‹*open U*›] **by** *auto*
note [*continuous-intros*] = *continuous-at-compose*[*OF - this*, *unfolded o-def*]
{
fix *a* **assume** $a \in M$
then have *aU*: $a \in U$ **and** *u-v*: $u\ a = v\ a$
by (*auto simp*: *M-def*)
then have *a-ball*: $a \in cball\ x\ e$ **and** *a-dist*: $dist\ x\ a \le e$ **using** *U* **by** *auto*
then have *a-S*: $(a, u\ a) \in S$
using *e* **by** *auto*
have *fa-z*: $f\ (a, u\ a) = 0$
using ‹$a \in cball\ x\ e$› **by** (*auto intro*!: *e*)
from *e*(*8*)[*OF* ‹$a \in cball$ - -›]
obtain *Ta* **where** *Ta*: $Ta\ o_L\ (f'\ (a, u\ a)\ o_L\ embed2\text{-}blinfun) = 1_L\ f'\ (a, u\ a)\ o_L\ embed2\text{-}blinfun\ o_L\ Ta = 1_L$
by (*auto simp*: *invertibles-blinfun-def ac-simps*)
obtain $u'\ e'\ r'$
where $r' > 0\ e' > 0$
and *u'*: $\bigwedge v\ s\ U.\ v\ a = u\ a \implies$
$(\bigwedge s.\ s \in U \implies f\ (s, v\ s) = 0) \implies$
continuous-on $U\ v \implies s \in U \implies U \subseteq cball\ a\ e' \implies (s, v\ s) \in ball\ (a, u\ a)\ r' \implies u'\ s = v\ s$
by (*rule implicit-function-theorem*[*OF f' a-S* ‹*open S*› *D f'Cg*[*OF a-S*] *fa-z Ta*]; *blast*)
from *openE*[*OF* ‹*open U*› *aU*] **obtain** *dU* **where** *dU*: $dU > 0\ \bigwedge s.\ s \in ball\ a\ dU \implies s \in U$
by (*auto simp*: *dist-commute subset-iff*)

have *v-tendsto*: $((\lambda s.\ (s, v\ s)) \longrightarrow (a, u\ a))\ (at\ a)$
unfolding *u-v*
by (*subst continuous-at*[*symmetric*]) (*auto intro*!: *continuous-intros aU*)
from *tendstoD*[*OF v-tendsto* ‹$0 < r'$›, *unfolded eventually-at*]
obtain *dv* **where** $dv > 0\ s \ne a \implies dist\ s\ a < dv \implies (s, v\ s) \in ball\ (a, u\ a)\ r'$ **for** *s*
by (*auto simp*: *dist-commute*)
then have *dv*: $dist\ s\ a < dv \implies (s, v\ s) \in ball\ (a, u\ a)\ r'$ **for** *s*
by (*cases* $s = a$) (*auto simp*: *u-v* ‹$0 < r'$›)

have *v-tendsto*: $((\lambda s.\ (s, u\ s)) \longrightarrow (a, u\ a))\ (at\ a)$
using *a-dist*
by (*subst continuous-at*[*symmetric*]) (*auto intro*!: *continuous-intros*)
from *tendstoD*[*OF v-tendsto* ‹$0 < r'$›, *unfolded eventually-at*]
obtain *du* **where** $du > 0\ s \ne a \implies dist\ s\ a < du \implies (s, u\ s) \in ball\ (a, u\ a)\ r'$ **for** *s*

```
        by (auto simp: dist-commute)
      then have du: dist s a < du ⟹ (s, u s) ∈ ball (a, u a) r' for s
        by (cases s = a) (auto simp: u-v ‹0 < r'›)
      {
        fix s assume s: s ∈ ball a (Min {dU, e', dv, du})
        let ?U = ball a (Min {dU, e', dv, du})
        have balls: ball a (Min {dU, e', dv, du}) ⊆ cball a e' by auto
        have dsadv: dist s a < dv
          using s by (auto simp: dist-commute)
        have dsadu: dist s a < du
          using s by (auto simp: dist-commute)
        have U-U: ⋀s. s ∈ ball a (Min {dU, e', dv, du}) ⟹ s ∈ U
          using dU by auto
        have U-e: ⋀s. s ∈ ball a (Min {dU, e', dv, du}) ⟹ s ∈ cball x e
          using dU U by (auto simp: dist-commute subset-iff)
        have cv: continuous-on ?U v
          using v
          apply (rule continuous-on-subset)
          using dU
          by auto
        have cu: continuous-on ?U u
          using e(4)
          apply (rule continuous-on-subset)
          using dU U(2)
          by auto
         from u'[where v=v, OF u-v[symmetric] solves[OF U-U] cv s balls dv[OF
dsadv]]
          u'[where v=u, OF refl                e(3)[OF U-e]    cu s balls du[OF dsadu]]
        have v s = u s by auto
      } then have ∃dv>0. ∀s ∈ ball a dv. v s = u s
        using ‹0 < dU› ‹0 < e'› ‹0 < dv› ‹0 < du›
        by (auto intro!: exI[where x=(Min {dU, e', dv, du})])
    } note ex = this
    have openin (top-of-set U) M
      unfolding openin-contains-ball
      apply (rule conjI)
      subgoal using U by (auto simp: M-def)
      apply (auto simp:)
      apply (drule ex)
      apply auto
      subgoal for x d
        by (rule exI[where x=d]) (auto simp: M-def)
      done
    ultimately have M = U
      using ‹connected U›
      by (auto simp: connected-clopen)
    with ‹s ∈ U› show ?thesis by (auto simp: M-def)
  qed
  from e this
```

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

lemma *uniform-limit-compose*:
assumes *ul*: *uniform-limit T f l F*
assumes *uc*: *uniformly-continuous-on S s*
assumes *ev*: $\forall_F$ *x in F. f x ‘ T* $\subseteq$ *S*
assumes *subs*: *l ‘ T* $\subseteq$ *S*
shows *uniform-limit T* (λ*i x. s* (*f i x*)) (λ*x. s* (*l x*)) *F*
proof (*rule uniform-limitI*)
fix *e*::*real* **assume** *e* > *0*
from *uniformly-continuous-onE*[*OF uc* ‹*e* > *0*›]
obtain *d* **where** *d*: *0* < *d* $\bigwedge$*t t'*. *t* $\in$ *S* $\Longrightarrow$ *t'* $\in$ *S* $\Longrightarrow$ *dist t' t* < *d* $\Longrightarrow$ *dist* (*s t'*) (*s t*) < *e*
by *auto*
from *uniform-limitD*[*OF ul* ‹*0* < *d*›] **have** $\forall_F$ *n in F*. $\forall$*x*$\in$*T*. *dist* (*f n x*) (*l x*) < *d* **.**
then show $\forall_F$ *n in F*. $\forall$*x*$\in$*T*. *dist* (*s* (*f n x*)) (*s* (*l x*)) < *e*
using *ev*
by *eventually-elim* (*use d subs* **in** *force*)
qed

lemma
uniform-limit-in-open:
fixes *l*::*'a*::*topological-space*⇒*'b*::*heine-borel*
assumes *ul*: *uniform-limit T f l* (*at x*)
assumes *cont*: *continuous-on T l*
assumes *compact*: *compact T* **and** *T-ne*: *T* $\neq$ {}
assumes *B*: *open B*
assumes *mem*: *l ‘ T* $\subseteq$ *B*
shows $\forall_F$ *y in at x*. $\forall$ *t* $\in$ *T*. *f y t* $\in$ *B*
proof −
have *l-ne*: *l ‘ T* $\neq$ {} **using** *T-ne* **by** *auto*
have *compact* (*l ‘ T*)
by (*auto intro*!: *compact-continuous-image cont compact*)
from *compact-in-open-separated*[*OF l-ne this B mem*]
obtain *e* **where** *e* > *0* {*x. infdist x* (*l ‘ T*) $\leq$ *e*} $\subseteq$ *B*
by *auto*
from *uniform-limitD*[*OF ul* ‹*0* < *e*›]
have $\forall_F$ *n in at x*. $\forall$*x*$\in$*T*. *dist* (*f n x*) (*l x*) < *e* **.**
then show *?thesis*
proof *eventually-elim*
case (*elim y*)
show *?case*
proof *safe*
fix *t* **assume** *t* $\in$ *T*
have *infdist* (*f y t*) (*l ‘ T*) $\leq$ *dist* (*f y t*) (*l t*)
by (*rule infdist-le*) (*use* ‹*t* $\in$ *T*› **in** *auto*)
also have ... < *e* **using** *elim* ‹*t* $\in$ *T*› **by** *auto*

```
      finally have infdist (f y t) (l ` T) ≤ e by simp
      then have (f y t) ∈ {x. infdist x (l ` T) ≤ e}
        by (auto )
      also note ‹... ⊆ B›
      finally show f y t ∈ B .
    qed
  qed
qed

lemma
  order-uniform-limitD1:
  fixes l::'a::topological-space⇒real— TODO: generalize?!
  assumes ul: uniform-limit T f l (at x)
  assumes cont: continuous-on T l
  assumes compact: compact T
  assumes less: ⋀t. t ∈ T ⟹ l t < b
  shows ∀F y in at x. ∀ t ∈ T. f y t < b
proof cases
  assume ne: T ≠ {}
  from compact-attains-sup[OF compact-continuous-image[OF cont compact], unfolded image-is-empty, OF ne]
  obtain tmax where tmax: tmax ∈ T ⋀s. s ∈ T ⟹ l s ≤ l tmax
    by auto
  have b − l tmax > 0
    using ne tmax less by auto
  from uniform-limitD[OF ul this]
  have ∀F n in at x. ∀x∈T. dist (f n x) (l x) < b − l tmax
    by auto
  then show ?thesis
    apply eventually-elim
    using tmax
    by (force simp: dist-real-def abs-real-def split: if-splits)
qed auto

lemma
  order-uniform-limitD2:
  fixes l::'a::topological-space⇒real— TODO: generalize?!
  assumes ul: uniform-limit T f l (at x)
  assumes cont: continuous-on T l
  assumes compact: compact T
  assumes less: ⋀t. t ∈ T ⟹ l t > b
  shows ∀F y in at x. ∀ t ∈ T. f y t > b
proof −
  have ∀F y in at x. ∀ t∈T. (− f) y t < − b
    by (rule order-uniform-limitD1[of − f T −l x − b])
       (auto simp: assms fun-Compl-def intro!: uniform-limit-eq-intros continuous-intros)
  then show ?thesis by auto
qed
```

lemma *continuous-on-avoid-cases*:
  fixes *l*::*'b*::*topological-space* ⇒ *'a*::*linear-continuum-topology*— TODO: generalize!
  assumes *cont*: *continuous-on T l* **and** *conn*: *connected T*
  assumes *avoid*: $\bigwedge t.\ t \in T \Longrightarrow l\ t \neq b$
  obtains $\bigwedge t.\ t \in T \Longrightarrow l\ t < b \mid \bigwedge t.\ t \in T \Longrightarrow l\ t > b$
  apply *atomize-elim*
  using *connected-continuous-image*[*OF cont conn*] **using** *avoid*
  unfolding *connected-iff-interval*
  apply (*auto simp*: *image-iff*)
  using *leI* **by** *blast*

lemma
  order-uniform-limit-ne:
  fixes *l*::*'a*::*topological-space*⇒*real*— TODO: generalize?!
  assumes *ul*: *uniform-limit T f l* (*at x*)
  assumes *cont*: *continuous-on T l*
  assumes *compact*: *compact T* **and** *conn*: *connected T*
  assumes *ne*: $\bigwedge t.\ t \in T \Longrightarrow l\ t \neq b$
  shows $\forall_F$ *y in at x*. $\forall\, t \in T.\ f\ y\ t \neq b$
proof −
  from *continuous-on-avoid-cases*[*OF cont conn ne*]
  consider $(\bigwedge t.\ t \in T \Longrightarrow l\ t < b) \mid (\bigwedge t.\ t \in T \Longrightarrow l\ t > b)$
    by *blast*
  then show *?thesis*
  proof *cases*
    case *1*
    from *order-uniform-limitD1*[*OF ul cont compact 1*]
    have $\forall_F$ *y in at x*. $\forall\, t{\in}T.\ f\ y\ t < b$ **by** *simp*
    then show *?thesis*
      by *eventually-elim auto*
  next
    case *2*
    from *order-uniform-limitD2*[*OF ul cont compact 2*]
    have $\forall_F$ *y in at x*. $\forall\, t{\in}T.\ f\ y\ t > b$ **by** *simp*
    then show *?thesis*
      by *eventually-elim auto*
  qed
qed

lemma *open-cballE*:
  assumes *open S x*∈*S*
  obtains *e* **where** *e*>*0 cball x e* ⊆ *S*
  using *assms* **unfolding** *open-contains-cball* **by** *auto*

lemma *pos-half-less*: **fixes** *x*::*real* **shows** $x > 0 \Longrightarrow x\ /\ 2 < x$
  by *auto*

lemma *closed-levelset*: *closed* {*x*. *s x* = (*c*::*'a*::*t1-space*)} **if** *continuous-on UNIV s*

proof −
  have {*x*. *s x* = *c*} = *s* −' {*c*} **by** *auto*
  also have *closed* ...
    apply (*rule closed-vimage*)
     apply (*rule closed-singleton*)
    apply (*rule that*)
    done
  finally show *?thesis* **.**
qed

lemma *closed-levelset-within*: *closed* {*x* ∈ *S*. *s x* = (*c*::*'a*::*t1-space*)} **if** *continuous-on S s closed S*
proof −
  have {*x* ∈ *S*. *s x* = *c*} = *s* −' {*c*} ∩ *S* **by** *auto*
  also have *closed* ...
    apply (*rule continuous-on-closed-vimageI*)
      apply (*rule that*)
     apply (*rule that*)
    apply *simp*
    done
  finally show *?thesis* **.**
qed

context *c1-on-open-euclidean*
begin

lemma *open-existence-ivlE*:
  assumes *t* ∈ *existence-ivl0 x t* ≥ *0*
  obtains *e* **where** *e* > *0 cball x e* × {*0* .. *t* + *e*} ⊆ *Sigma X existence-ivl0*
proof −
  from *assms* **have** (*x*, *t*) ∈ *Sigma X existence-ivl0*
    by *auto*
  from *open-cballE*[*OF open-state-space this*]
  obtain *e0'* **where** *e0*: *0* < *e0' cball* (*x*, *t*) *e0'* ⊆ *Sigma X existence-ivl0*
    by *auto*
  define *e0* **where** *e0* = (*e0'* / *2*)
  from *cball-times-subset*[*of x e0' t*] *pos-half-less*[*OF* ‹*0* < *e0'*›] *half-gt-zero*[*OF* ‹*0* < *e0'*›] *e0*
  have *cball x e0* × *cball t e0* ⊆ *Sigma X existence-ivl0 0* < *e0 e0* < *e0'*
    unfolding *e0-def* **by** *auto*
  then have *e0* > *0 cball x e0* × {*0*..*t* + *e0*} ⊆ *Sigma X existence-ivl0*
    apply (*auto simp*: *subset-iff dest*!: *spec*[**where** *x*=*t*])
    subgoal for *a b*
      apply (*rule in-existence-between-zeroI*)
      apply (*drule spec*[**where** *x*=*a*])
       apply (*drule spec*[**where** *x*=*t* + *e0*])
       apply (*auto simp*: *dist-real-def closed-segment-eq-real-ivl*)
      done
    done

then show *?thesis* **..**
qed

lemmas [*derivative-intros*] = *flow0-comp-has-derivative*

lemma *flow-isCont-state-space-comp*[*continuous-intros*]:
t x ∈ *existence-ivl0* (*s x*) ⟹ *isCont s x* ⟹ *isCont t x* ⟹ *isCont* (λ*x*. *flow0* (*s x*) (*t x*)) *x*
using *continuous-within-compose3*[**where** *g*=λ(*x*, *t*). *flow0 x t*
and *f*=λ*x*. (*s x*, *t x*) **and** *x* = *x* **and** *s* = *UNIV*]
flow-isCont-state-space
by *auto*

lemma *closed-plane*[*simp*]: *closed* {*x*. *x* · *i* = *c*}
using *closed-hyperplane*[*of i c*] **by** (*auto simp*: *inner-commute*)

lemma *flow-tendsto-compose*[*tendsto-intros*]:
assumes (*x* ⟶ *xs*) *F* (*t* ⟶ *ts*) *F*
assumes *ts* ∈ *existence-ivl0 xs*
shows ((λ*s*. *flow0* (*x s*) (*t s*)) ⟶ *flow0 xs ts*) *F*
proof −
have *ev*: $\forall_F$ *s in F*. (*x s*, *t s*) ∈ *Sigma X existence-ivl0*
using *tendsto-Pair*[*OF assms*(*1*,*2*), *THEN topological-tendstoD*, *OF open-state-space*]
assms
by *auto*
show *?thesis*
by (*rule continuous-on-tendsto-compose*[*OF flow-continuous-on-state-space tendsto-Pair*, *unfolded split-beta′ fst-conv snd-conv*])
(*use assms ev* **in** *auto*)
qed

lemma *returns-to-implicit-function*:
fixes *s*::′*a*::*euclidean-space* ⇒ *real*
assumes *rt*: *returns-to* {*x* ∈ *S*. *s x* = *0*} *x* (**is** *returns-to ?P x*)
assumes *cS*: *closed S*
assumes *Ds*: ⋀*x*. (*s has-derivative blinfun-apply* (*Ds x*)) (*at x*)
assumes *DsC*: *isCont Ds* (*poincare-map ?P x*)
assumes *nz*: *Ds* (*poincare-map ?P x*) (*f* (*poincare-map ?P x*)) ≠ *0*
obtains *u e*
where *s* (*flow0 x* (*u x*)) = *0*
u x = *return-time ?P x*
(⋀*y*. *y* ∈ *cball x e* ⟹ *s* (*flow0 y* (*u y*)) = *0*)
continuous-on (*cball x e*) *u*
(λ*t*. (*t*, *u t*)) ‘ *cball x e* ⊆ *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*]
have *cont-s*: *continuous-on UNIV s* **by** (*rule has-derivative-continuous-on*[*OF Ds*])
note *cls*[*simp*, *intro*] = *closed-levelset*[*OF cont-s*]
let *?t1* = *return-time ?P x*
have *cls*[*simp*, *intro*]: *closed* {*x* ∈ *S*. *s x* = *0*}
by (*rule closed-levelset-within*) (*auto intro*!: *cS continuous-on-subset*[*OF cont-s*])
then have *xt1*: (*x*, *?t1*) ∈ *Sigma X existence-ivl0*
by (*auto intro*!: *return-time-exivl rt*)
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*, *?t1*)
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 return-time-exivl rt isCont-tendsto-compose*[*OF DsC*, *unfolded poincare-map-def*]
simp: *split-beta′ isCont-def*)
from *return-time-returns*[*OF rt cls*]
have *Z*: (*case* (*x*, *?t1*) *of* (*x*, *t*) ⇒ *s* (*flow0 x t*)) = *0*
by *auto*
have *I1*: *blinfun-scaleR-left* (*inverse* (*Ds* (*flow0 x* (*?t1*))(*f* (*flow0 x* (*?t1*))))) o_L
((*Ds* (*flow0* (*fst* (*x*, *return-time* {*x* ∈ *S*. *s x* = *0*} *x*))
(*snd* (*x*, *return-time* {*x* ∈ *S*. *s x* = *0*} *x*))) o_L
flowderiv (*fst* (*x*, *return-time* {*x* ∈ *S*. *s x* = *0*} *x*))
(*snd* (*x*, *return-time* {*x* ∈ *S*. *s x* = *0*} *x*))) o_L
embed2-blinfun)
= 1_L
using *nz*
by (*auto intro*!: *blinfun-eqI*
simp: *rt flowderiv-def blinfun.bilinear-simps inverse-eq-divide poincare-map-def*)
have *I2*: ((*Ds* (*flow0* (*fst* (*x*, *return-time* {*x* ∈ *S*. *s x* = *0*} *x*))
(*snd* (*x*, *return-time* {*x* ∈ *S*. *s x* = *0*} *x*))) o_L
flowderiv (*fst* (*x*, *return-time* {*x* ∈ *S*. *s x* = *0*} *x*))
(*snd* (*x*, *return-time* {*x* ∈ *S*. *s x* = *0*} *x*))) o_L
embed2-blinfun) o_L *blinfun-scaleR-left* (*inverse* (*Ds* (*flow0 x* (*?t1*))(*f* (*flow0 x* (*?t1*)))))
= 1_L
using *nz*
by (*auto intro*!: *blinfun-eqI*
simp: *rt 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 (**in** *auto-ll-on-open*) *f-tendsto*[*tendsto-intros*]:
  assumes *g1*: (*g1* ⟶ *b1*) (*at s within S*) **and** *b1* ∈ *X*
  shows ((λ*x*. *f* (*g1 x*)) ⟶ *f b1*) (*at s within S*)
  apply (*rule continuous-on-tendsto-compose*[*OF continuous tendsto-Pair*[*OF tendsto-const*],
    unfolded split-beta fst-conv snd-conv, *OF g1*])
  by (*auto simp*: ‹*b1* ∈ *X*› *intro*!: *topological-tendstoD*[*OF g1*])

lemma *flow-avoids-surface-eventually-at-right-pos*:
  assumes *s x* > *0* ∨ *s x* = *0* ∧ *blinfun-apply* (*Ds x*) (*f x*) > *0*
  assumes *x*: *x* ∈ *X*
  assumes *Ds*: ⋀*x*. (*s has-derivative Ds x*) (*at x*)
  assumes *DsC*: ⋀*x*. *isCont Ds x*
  shows ∀$_F$ *t in at-right 0*. *s* (*flow0 x t*) > (*0*::*real*)
proof −
  have *cont-s*: *continuous-on UNIV s* **by** (*rule has-derivative-continuous-on*[*OF Ds*])
  then have [*THEN continuous-on-compose2*, *continuous-intros*]: *continuous-on S s* **for** *S* **by** (*rule continuous-on-subset*) *simp*
  note [*derivative-intros*] = *has-derivative-compose*[*OF - Ds*]
  note [*tendsto-intros*] = *continuous-on-tendsto-compose*[*OF cont-s*]
    isCont-tendsto-compose[*OF DsC*]
  from *assms*(*1*)
  consider *s x* > *0* | *s x* = *0 blinfun-apply* (*Ds x*) (*f x*) > *0*
    by *auto*
  then show *?thesis*
  proof *cases*
    assume *s*: *s x* > *0*
    then have ((λ*t*. *s* (*flow0 x t*)) ⟶ *s x*) (*at-right 0*)
      by (*auto intro*!: *tendsto-eq-intros simp*: *split-beta' x*)
    from *order-tendstoD*(*1*)[*OF this s*]
    show *?thesis* **.**
  next
    assume *sz*: *s x* = *0* **and** *pos*: *blinfun-apply* (*Ds x*) (*f x*) > *0*
    from *x* **have** *0* ∈ *existence-ivl0 x open* (*existence-ivl0 x*) **by** *simp-all*
    then have *evex*: ∀$_F$ *t in at-right 0*. *t* ∈ *existence-ivl0 x*
      using *eventually-at-topological* **by** *blast*
    moreover
    from *evex* **have** ∀$_F$ *xa in at-right 0*. *flow0 x xa* ∈ *X*
      by (*eventually-elim*) (*auto intro*!:)
    then have ((λ*t*. (*Ds* (*flow0 x t*)) (*f* (*flow0 x t*))) ⟶ *blinfun-apply* (*Ds x*) (*f x*)) (*at-right 0*)
      by (*auto intro*!: *tendsto-eq-intros simp*: *split-beta' x*)

from *order-tendstoD*(*1*)[*OF this pos*]
have $\forall_F$ *z in at-right 0*. *blinfun-apply* (*Ds* (*flow0 x z*)) (*f* (*flow0 x z*)) > *0* **.**
then obtain *t* **where** *t*: *t* > *0* ⋀*z*. *0* < *z* ⟹ *z* < *t* ⟹ *blinfun-apply* (*Ds* (*flow0 x z*)) (*f* (*flow0 x z*)) > *0*
by (*auto simp*: *eventually-at*)
have $\forall_F$ *z in at-right 0*. *z* < *t* **using** ‹*t* > *0*› *order-tendstoD*(*2*)[*OF tendsto-ident-at* ‹*0* < *t*›] **by** *auto*
moreover have $\forall_F$ *z in at-right 0*. *0* < *z* **by** (*simp add*: *eventually-at-filter*)
ultimately show *?thesis*
proof *eventually-elim*
case (*elim z*)
from *closed-segment-subset-existence-ivl*[*OF* ‹*z* ∈ *existence-ivl0 x*›]
have *csi*: {*0*..*z*} ⊆ *existence-ivl0 x* **by** (*auto simp add*: *closed-segment-eq-real-ivl*)
then have *cont*: *continuous-on* {*0*..*z*} (λ*t*. *s* (*flow0 x t*))
by (*auto intro*!: *continuous-intros*)
have ⋀*u*. ⟦*0* < *u*; *u* < *z*⟧ ⟹ ((λ*t*. *s* (*flow0 x t*)) *has-derivative* (λ*t*. *t* ∗ *blinfun-apply* (*Ds* (*flow0 x u*)) (*f* (*flow0 x u*)))) (*at u*)
using *csi*
by (*auto intro*!: *derivative-eq-intros simp*: *flowderiv-def blinfun.bilinear-simps*)
from *mvt*[*OF* ‹*0* < *z*› *cont this*]
obtain *w* **where** *w*: *0* < *w w* < *z* **and** *sDs*: *s* (*flow0 x z*) = *z* ∗ *blinfun-apply* (*Ds* (*flow0 x w*)) (*f* (*flow0 x w*))
using *x sz*
by *auto*
note *sDs*
also have . . . > *0*
using *elim t*(*2*)[*of w*] *w* **by** *simp*
finally show *?case* **.**
qed
qed
qed

lemma *flow-avoids-surface-eventually-at-right-neg*:
assumes *s x* < *0* ∨ *s x* = *0* ∧ *blinfun-apply* (*Ds x*) (*f x*) < *0*
assumes *x*: *x* ∈ *X*
assumes *Ds*: ⋀*x*. (*s has-derivative Ds x*) (*at x*)
assumes *DsC*: ⋀*x*. *isCont Ds x*
shows $\forall_F$ *t in at-right 0*. *s* (*flow0 x t*) < (*0*::*real*)
apply (*rule flow-avoids-surface-eventually-at-right-pos*[*of* −*s x* −*Ds*, *simplified*])
using *assms*
by (*auto intro*!: *derivative-eq-intros simp*: *blinfun.bilinear-simps fun-Compl-def*)

lemma *flow-avoids-surface-eventually-at-right*:
assumes *x* ∉ *S* ∨ *s x* ≠ *0* ∨ *blinfun-apply* (*Ds x*) (*f x*) ≠ *0*
assumes *x*: *x* ∈ *X* **and** *cS*: *closed S*
assumes *Ds*: ⋀*x*. (*s has-derivative Ds x*) (*at x*)
assumes *DsC*: ⋀*x*. *isCont Ds x*
shows $\forall_F$ *t in at-right 0*. (*flow0 x t*) ∉ {*x* ∈ *S*. *s x* = (*0*::*real*)}
proof −

from *assms*(*1*)
consider
s x > *0* ∨ *s x* = *0* ∧ *blinfun-apply* (*Ds x*) (*f x*) > *0*
| *s x* < *0* ∨ *s x* = *0* ∧ *blinfun-apply* (*Ds x*) (*f x*) < *0*
| *x* ∉ *S*
by *arith*
then show *?thesis*
proof *cases*
case *1*
from *flow-avoids-surface-eventually-at-right-pos*[*of s x Ds*, *OF 1 x Ds DsC*]
show *?thesis* **by** *eventually-elim auto*
next
case *2*
from *flow-avoids-surface-eventually-at-right-neg*[*of s x Ds*, *OF 2 x Ds DsC*]
show *?thesis* **by** *eventually-elim auto*
next
case *3*
then have *nS*: *open* (− *S*) *x* ∈ − *S* **using** *cS* **by** *auto*
have $\forall_F$ *t in at-right 0*. (*flow0 x t*) ∈ − *S*
by (*rule topological-tendstoD*[*OF - nS*]) (*auto intro*!: *tendsto-eq-intros simp*: *x*)
then show *?thesis* **by** *eventually-elim auto*
qed
qed

lemma *eventually-returns-to*:
fixes *s*::′*a*::*euclidean-space* ⇒ *real*
assumes *rt*: *returns-to* {*x* ∈ *S*. *s x* = *0*} *x* (**is** *returns-to ?P x*)
assumes *cS*: *closed S*
assumes *Ds*: ⋀*x*. (*s has-derivative blinfun-apply* (*Ds x*)) (*at x*)
assumes *DsC*: ⋀*x*. *isCont Ds x*
assumes *eventually-inside*: $\forall_F$ *x in at* (*poincare-map ?P x*). *s x* = *0* ⟶ *x* ∈ *S*
assumes *nz*: *Ds* (*poincare-map ?P x*) (*f* (*poincare-map ?P x*)) ≠ *0*
assumes *nz0*: *x* ∉ *S* ∨ *s x* ≠ *0* ∨ *Ds x* (*f x*) ≠ *0*
shows $\forall_F$ *x in at x*. *returns-to ?P x*
proof −
let *?t1* = *return-time ?P x*
have *cont-s*: *continuous-on UNIV s* **by** (*rule has-derivative-continuous-on*[*OF Ds*])
have *cont-s*′: *continuous-on S s* **for** *S* **by** (*rule continuous-on-subset*[*OF cont-s subset-UNIV*])
note *s-tendsto*[*tendsto-intros*] = *continuous-on-tendsto-compose*[*OF cont-s*, *THEN tendsto-eq-rhs*]
note *cls*[*simp*, *intro*] = *closed-levelset-within*[*OF cont-s*′ *cS*, *of 0*]
note [*tendsto-intros*] = *continuous-on-tendsto-compose*[*OF cont-s*]
isCont-tendsto-compose[*OF DsC*]
obtain *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*
($\lambda t.\ (t,\ u\ t)$) ‘ $cball\ x\ e \subseteq Sigma\ X\ existence\text{-}ivl0$
$0 < e$
by (*rule returns-to-implicit-function*[*OF rt cS Ds DsC nz*]; *blast*)
then have *u*:
$s\ (flow0\ x\ (u\ x)) = 0\ u\ x = ?t1$
($\bigwedge y.\ y \in cball\ x\ e \Longrightarrow s\ (flow0\ y\ (u\ y)) = 0$)
continuous-on (*cball x e*) *u*
$\bigwedge z.\ z \in cball\ x\ e \Longrightarrow u\ z \in existence\text{-}ivl0\ z$
$e > 0$
by (*force simp*: *split-beta*′)+
have $\forall_F\ y\ in\ at\ x.\ y \in ball\ x\ e$
using *eventually-at-ball*[*OF* ‹$0 < e$›]
by *eventually-elim auto*
then have *ev-cball*: $\forall_F\ y\ in\ at\ x.\ y \in cball\ x\ e$
by *eventually-elim* (*use* ‹$e > 0$› **in** *auto*)
moreover
have *continuous-on* (*ball x e*) *u*
using *u* **by** (*auto simp*: *continuous-on-subset*)
then have [*tendsto-intros*]: ($u \longrightarrow u\ x$) (*at x*)
using ‹$e > 0$› *at-within-open*[*of y ball x e* **for** *y*]
by (*auto simp*: *continuous-on-def*)
then have *flow0-u-tendsto*: ($\lambda x.\ flow0\ x\ (u\ x)$) $-x\rightarrow$ *poincare-map ?P x*
by (*auto intro*!: *tendsto-eq-intros u return-time-exivl rt simp*: *poincare-map-def*)
have *s-imp*: $s\ (poincare\text{-}map\ \{x \in S.\ s\ x = 0\}\ x) = 0 \longrightarrow poincare\text{-}map\ \{x \in S.\ s\ x = 0\}\ x \in S$
using *poincare-map-returns*[*OF rt*]
by *auto*
from *eventually-tendsto-compose-within*[*OF eventually-inside s-imp flow0-u-tendsto*]
have $\forall_F\ x\ in\ at\ x.\ s\ (flow0\ x\ (u\ x)) = 0 \longrightarrow flow0\ x\ (u\ x) \in S$ **by** *auto*
with *ev-cball*
have $\forall_F\ x\ in\ at\ x.\ flow0\ x\ (u\ x) \in S$
by *eventually-elim* (*auto simp*: *u*)
moreover
{
have $x \in X$
using *u*(*5*) *u*(*6*) **by** *force*
from *ev-cball*
have *ev-X*: $\forall_F\ y\ in\ at\ x.\ y \in X$— eigentlich ist das *open X*
apply *eventually-elim*
apply (*rule*)
by (*rule u*)
moreover
{
{
assume *a*: $x \notin S$ **then have** *open* ($-S$) $x \in -\ S$ **using** *cS* **by** *auto*
from *topological-tendstoD*[*OF tendsto-ident-at this*]
have ($\forall_F\ y\ in\ at\ x.\ y \notin S$) **by** *auto*

} **moreover** **{**
assume *a*: *s x* ≠ *0*
have (∀$_F$ *y in at x*. *s y* ≠ *0*)
by (*rule tendsto-imp-eventually-ne*[*OF* - *a*]) (*auto intro*!: *tendsto-eq-intros*)
} **moreover** **{**
assume *a*: (*Ds x*) (*f x*) ≠ *0*
have (∀$_F$ *y in at x*. *blinfun-apply* (*Ds y*) (*f y*) ≠ *0*)
by (*rule tendsto-imp-eventually-ne*[*OF* - *a*]) (*auto intro*!: *tendsto-eq-intros ev-X* ‹*x* ∈ *X*›)
} **ultimately have** (∀$_F$ *y in at x*. *y* ∉ *S*) ∨ (∀$_F$ *y in at x*. *s y* ≠ *0*) ∨ (∀$_F$ *y in at x*. *blinfun-apply* (*Ds y*) (*f y*) ≠ *0*)
using *nz0* **by** *auto*
then have ∀$_F$ *y in at x*. *y* ∉ *S* ∨ *s y* ≠ *0* ∨ *blinfun-apply* (*Ds y*) (*f y*) ≠ *0*
apply −
apply (*erule disjE*)
subgoal by (*rule eventually-elim2*, *assumption*, *assumption*, *blast*)
subgoal
apply (*erule disjE*)
subgoal by (*rule eventually-elim2*, *assumption*, *assumption*, *blast*)
subgoal by (*rule eventually-elim2*, *assumption*, *assumption*, *blast*)
done
done
}
ultimately
have ∀$_F$ *y in at x*. (*y* ∉ *S* ∨ *s y* ≠ *0* ∨ *blinfun-apply* (*Ds y*) (*f y*) ≠ *0*) ∧ *y* ∈ *X*
by *eventually-elim auto*
}
then have ∀$_F$ *y in at x*. ∀$_F$ *t in at-right 0*. *flow0 y t* ∉ {*x* ∈ *S*. *s x* = *0*}
apply *eventually-elim*
by (*rule flow-avoids-surface-eventually-at-right*[**where** *Ds*=*Ds*]) (*auto intro*!: *Ds DsC cS*)
moreover
have *at-eq*: (*at x within cball x e*) = *at x*
apply (*rule at-within-interior*)
apply (*auto simp*: ‹*e* > *0*›)
done
have *u x* > *0*
using *u*(*1*) **by** (*auto simp*: *u rt cont-s*′ *intro*!: *return-time-pos closed-levelset-within cS*)
then have ∀$_F$ *y in at x*. *u y* > *0*
apply (*rule order-tendstoD*[*rotated*])
using *u*(*4*)
apply (*auto simp*: *continuous-on-def*)
apply (*drule bspec*[**where** *x*=*x*])
using ‹*e* > *0*›
by (*auto simp*: *at-eq*)
ultimately
show ∀$_F$ *y in at x*. *returns-to ?P y*

apply *eventually-elim*
subgoal premises *prems* **for** *y*
apply (*rule returns-toI*[**where** *t=u y*])
subgoal using *prems* **by** *auto*
subgoal apply (*rule u*) **apply** (*rule prems*) **done**
subgoal using *u*(*3*)[*of y*] *prems* **by** *auto*
subgoal using *prems*(*3*) **by** *eventually-elim auto*
subgoal by *simp*
done
done
qed

lemma
return-time-isCont-outside:
fixes *s*::*'a*::*euclidean-space* ⇒ *real*
assumes *rt*: *returns-to* {*x* ∈ *S*. *s x* = *0*} *x* (**is** *returns-to ?P x*)
assumes *cS*: *closed S*
assumes *Ds*: ⋀*x*. (*s has-derivative blinfun-apply* (*Ds x*)) (*at x*)
assumes *DsC*: ⋀*x*. *isCont Ds x*
assumes *through*: (*Ds* (*poincare-map ?P x*)) (*f* (*poincare-map ?P x*)) ≠ *0*
assumes *eventually-inside*: ∀$_F$ *x in at* (*poincare-map ?P x*). *s x* = *0* ⟶ *x* ∈ *S*
assumes *outside*: *x* ∉ *S* ∨ *s x* ≠ *0*
shows *isCont* (*return-time ?P*) *x*
unfolding *isCont-def*
proof (*rule tendstoI*)
fix *e-orig*::*real* **assume** *e-orig* > *0*
define *e* **where** *e* = *e-orig* / *2*
have *e* > *0* **using** ‹*e-orig* > *0*› **by** (*simp add*: *e-def*)

have *cont-s*: *continuous-on UNIV s* **by** (*rule has-derivative-continuous-on*[*OF Ds*])
then have *s-tendsto*: (*s* ⟶ *s x*) (*at x*) **for** *x*
by (*auto simp*: *continuous-on-def*)
have *cont-s'*: *continuous-on S s* **by** (*rule continuous-on-subset*[*OF cont-s subset-UNIV*])
note *cls*[*simp*, *intro*] = *closed-levelset-within*[*OF cont-s' cS*(*1*)]
have {*x*. *s x* = *0*} = *s* −‘ {*0*} **by** *auto*
have *ret-exivl*: *return-time ?P x* ∈ *existence-ivl0 x*
by (*rule return-time-exivl*; *fact*)
then have [*intro*, *simp*]: *x* ∈ *X* **by** *auto*
have *isCont-Ds-f*: *isCont* (λ*s*. *Ds s* (*f s*)) (*poincare-map ?P x*)
apply (*auto intro*!: *continuous-intros DsC*)
apply (*rule has-derivative-continuous*)
apply (*rule derivative-rhs*)
by (*auto simp*: *poincare-map-def intro*!: *flow-in-domain return-time-exivl assms*)

obtain *u eu* **where** *u*:
s (*flow0 x* (*u x*)) = *0*
u x = *return-time ?P x*

(⋀*y*. *y* ∈ *cball x eu* ⟹ *s* (*flow0 y* (*u y*)) = *0*)
continuous-on (*cball x eu*) *u*
(λ*t*. (*t*, *u t*)) ‘ *cball x eu* ⊆ *Sigma X existence-ivl0*
0 < *eu*
by (*rule returns-to-implicit-function*[*OF rt cS*(*1*) *Ds DsC through*]; *blast*)
have *u-tendsto*: (*u* ⟶ *u x*) (*at x*)
unfolding *isCont-def*[*symmetric*]
apply (*rule continuous-on-interior*[*OF u*(*4*)])
using ‹*0* < *eu*› **by** *auto*
have *u x* > *0* **by** (*auto simp*: *u intro*!: *return-time-pos rt*)
from *order-tendstoD*(*1*)[*OF u-tendsto this*] **have** ∀$_F$ *x in at x*. *0* < *u x* **.**
moreover have ∀$_F$ *y in at x*. *y* ∈ *cball x eu*
using *eventually-at-ball*[*OF* ‹*0* < *eu*›, *of x*]
by *eventually-elim auto*
moreover
have *x* ∉ *S* ∨ *s x* ≠ *0* ∨ *blinfun-apply* (*Ds x*) (*f x*) ≠ *0* **using** *outside* **by** *auto*
have *returns*: ∀$_F$ *y in at x*. *returns-to ?P y*
by (*rule eventually-returns-to*; *fact*)
moreover
have ∀$_F$ *y in at x*. *y* ∈ *ball x eu*
using *eventually-at-ball*[*OF* ‹*0* < *eu*›]
by *eventually-elim simp*
then have *ev-cball*: ∀$_F$ *y in at x*. *y* ∈ *cball x eu*
by *eventually-elim* (*use* ‹*e* > *0*› **in** *auto*)
have *continuous-on* (*ball x eu*) *u*
using *u* **by** (*auto simp*: *continuous-on-subset*)
then have [*tendsto-intros*]: (*u* ⟶ *u x*) (*at x*)
using ‹*eu* > *0*› *at-within-open*[*of y ball x eu* **for** *y*]
by (*auto simp*: *continuous-on-def*)
then have *flow0-u-tendsto*: (λ*x*. *flow0 x* (*u x*)) −*x*→ *poincare-map ?P x*
by (*auto intro*!: *tendsto-eq-intros u return-time-exivl rt simp*: *poincare-map-def*)
have *s-imp*: *s* (*poincare-map* {*x* ∈ *S*. *s x* = *0*} *x*) = *0* ⟶ *poincare-map* {*x* ∈ *S*. *s x* = *0*} *x* ∈ *S*
using *poincare-map-returns*[*OF rt*]
by *auto*
from *eventually-tendsto-compose-within*[*OF eventually-inside s-imp flow0-u-tendsto*]
have ∀$_F$ *x in at x*. *s* (*flow0 x* (*u x*)) = *0* ⟶ *flow0 x* (*u x*) ∈ *S* **by** *auto*
with *ev-cball*
have ∀$_F$ *x in at x*. *flow0 x* (*u x*) ∈ *S*
by *eventually-elim* (*auto simp*: *u*)
ultimately have *u-returns-ge*: ∀$_F$ *y in at x*. *returns-to ?P y* ∧ *return-time ?P y* ≤ *u y*
proof *eventually-elim*
case (*elim y*)
then show *?case*
using *u elim* **by** (*auto intro*!: *return-time-le*[*OF - cls*])
qed
moreover
have ∀$_F$ *y in at x*. *u y* − *return-time ?P x* < *e*

using *tendstoD*[*OF u-tendsto* ‹*0* < *e*›, *unfolded u*] *u-returns-ge*
by *eventually-elim* (*auto simp*: *dist-real-def*)
moreover
note *1* = *outside*
define *ml* **where** *ml* = *max* (*return-time ?P x* / *2*) (*return-time ?P x* − *e*)
have [*intro*, *simp*, *arith*]: *0* < *ml ml* < *return-time ?P x ml* ≤ *return-time ?P x*
using *return-time-pos*[*OF rt cls*] ‹*0* < *e*›
by (*auto simp*: *ml-def*)
have *mt-in*: *ml* ∈ *existence-ivl0 x*
using ‹*0* < *e*›
by (*auto intro*!: *mem-existence-ivl-iv-defined in-existence-between-zeroI*[*OF ret-exivl*]
simp: *closed-segment-eq-real-ivl ml-def*)
from *open-existence-ivlE*[*OF mt-in*]
obtain *e0* **where** *e0*: *e0* > *0 cball x e0* × {*0*..*ml* + *e0*} ⊆ *Sigma X existence-ivl0*
(**is** *?D* ⊆ -)
by *auto*
have *uc*: *uniformly-continuous-on* ((λ(*x*, *t*). *flow0 x t*) ‘ *?D*) *s*
apply (*auto intro*!: *compact-uniformly-continuous continuous-on-subset*[*OF*
cont-s])
apply (*rule compact-continuous-image*)
apply (*rule continuous-on-subset*)
apply (*rule flow-continuous-on-state-space*)
apply (*rule e0*)
apply (*rule compact-Times*)
apply (*rule compact-cball*)
apply (*rule compact-Icc*)
done
let *?T* = {*0*..*ml*}
have *ul*: *uniform-limit ?T flow0* (*flow0 x*) (*at x*)
using ‹*0* < *e*›
by (*intro uniform-limit-flow*)
(*auto intro*!: *mem-existence-ivl-iv-defined in-existence-between-zeroI*[*OF ret-exivl*]
simp: *closed-segment-eq-real-ivl*)
have $\forall_F$ *y in at x*. ∀ *t*∈{*0*..*ml*}. *flow0 y t* ∈ − {*x* ∈ *S*. *s x* = *0*}
apply (*rule uniform-limit-in-open*)
apply (*rule ul*)
apply (*auto intro*!: *continuous-intros continuous-on-compose2*[*OF cont-s*]
simp:
split: *if-splits*)
apply (*meson atLeastAtMost-iff contra-subsetD local.ivl-subset-existence-ivl*
mt-in)
subgoal for *t*
apply (*cases t* = *0*)
subgoal using *1* **by** (*simp*)
subgoal
using *return-time-least*[*OF rt cls*, *of t*] ‹*ml* < *return-time* {*x* ∈ *S*. *s x* = *0*}
x›
by *auto*
done

```
    done
  then have ∀F y in at x. return-time ?P y ≥ return-time ?P x − e
    using u-returns-ge
  proof eventually-elim
    case (elim y)
    have return-time ?P x − e ≤ ml
      by (auto simp: ml-def)
    also
    have ry: returns-to ?P y return-time ?P y ≤ u y
      using elim
      by auto
    have ml < return-time ?P y
      apply (rule return-time-gt[OF ry(1) cls])
      using elim
      by (auto simp: Ball-def)
    finally show ?case by simp
  qed
  ultimately
  have ∀F y in at x. dist (return-time ?P y) (return-time ?P x) ≤ e
    by eventually-elim (auto simp: dist-real-def abs-real-def algebra-simps)
  then show ∀F y in at x. dist (return-time ?P y) (return-time ?P x) < e-orig
    by eventually-elim (use ‹e-orig > 0› in ‹auto simp: e-def›)
qed

lemma isCont-poincare-map:
  assumes isCont (return-time P) x
    returns-to P x closed P
  shows isCont (poincare-map P) x
  unfolding poincare-map-def
  by (auto intro!: continuous-intros assms return-time-exivl)

lemma poincare-map-tendsto:
  assumes (return-time P ⟶ return-time P x) (at x within S)
    returns-to P x closed P
  shows (poincare-map P ⟶ poincare-map P x) (at x within S)
  unfolding poincare-map-def
  by (rule tendsto-eq-intros refl assms return-time-exivl)+

lemma
  return-time-continuous-below:
  fixes s::'a::euclidean-space ⇒ real
  assumes rt: returns-to {x ∈ S. s x = 0} x (is returns-to ?P x)
  assumes Ds: ⋀x. (s has-derivative blinfun-apply (Ds x)) (at x)
  assumes cS: closed S
  assumes eventually-inside: ∀F x in at (poincare-map ?P x). s x = 0 ⟶ x ∈ S
  assumes DsC: ⋀x. isCont Ds x
  assumes through: (Ds (poincare-map ?P x)) (f (poincare-map ?P x)) ≠ 0
  assumes inside: x ∈ S s x = 0 Ds x (f x) < 0
  shows continuous (at x within {x. s x ≤ 0}) (return-time ?P)
```

unfolding *continuous-within*
proof (*rule tendstoI*)
fix *e-orig*::*real* **assume** *e-orig* > *0*
define *e* **where** *e* = *e-orig* / *2*
have *e* > *0* **using** ‹*e-orig* > *0*› **by** (*simp add*: *e-def*)

note *DsC-tendso*[*tendsto-intros*] = *isCont-tendsto-compose*[*OF DsC*]
have *cont-s*: *continuous-on UNIV s* **by** (*rule has-derivative-continuous-on*[*OF Ds*])
then have *s-tendsto*: (*s* ⟶ *s x*) (*at x*) **for** *x*
by (*auto simp*: *continuous-on-def*)
note [*continuous-intros*] = *continuous-on-compose2*[*OF cont-s - subset-UNIV*]
note [*derivative-intros*] = *has-derivative-compose*[*OF - Ds*]
have *cont-s'*: *continuous-on S s* **by** (*rule continuous-on-subset*[*OF cont-s subset-UNIV*])
note *cls*[*simp*, *intro*] = *closed-levelset-within*[*OF cont-s' cS*(*1*)]
have {*x*. *s x* = *0*} = *s* −‘ {*0*} **by** *auto*
have *ret-exivl*: *return-time ?P x* ∈ *existence-ivl0 x*
by (*rule return-time-exivl*; *fact*)
then have [*intro*, *simp*]: *x* ∈ *X* **by** *auto*
have *isCont-Ds-f*: *isCont* (λ*s*. *Ds s* (*f s*)) (*poincare-map ?P x*)
apply (*auto intro*!: *continuous-intros DsC*)
apply (*rule has-derivative-continuous*)
apply (*rule derivative-rhs*)
by (*auto simp*: *poincare-map-def intro*!: *flow-in-domain return-time-exivl assms*)

have ∀$_F$ *yt in at* (*x*, *0*) *within UNIV* × {*0*<..}. (*Ds* (*flow0* (*fst yt*) (*snd yt*))) (*f* (*flow0* (*fst yt*) (*snd yt*))) < *0*
by (*rule order-tendstoD*) (*auto intro*!: *tendsto-eq-intros inside*)
moreover
have (*x*, *0*) ∈ *Sigma X existence-ivl0* **by** *auto*
from *topological-tendstoD*[*OF tendsto-ident-at open-state-space this*, *of UNIV* × {*0*<..}]
have ∀$_F$ *yt in at* (*x*, *0*) *within UNIV* × {*0*<..}. *snd yt* ∈ *existence-ivl0* (*fst yt*)
by *eventually-elim auto*
moreover
from *topological-tendstoD*[*OF tendsto-ident-at open-Times*[*OF open-dom open-UNIV*], *of* (*x*, *0*) *UNIV* × {*0*<..}]
have ∀$_F$ *yt in at* (*x*, *0*) *within UNIV* × {*0*<..}. *fst yt* ∈ *X*
by (*auto simp*: *mem-Times-iff*)
ultimately
have ∀$_F$ *yt in at* (*x*, *0*) *within UNIV* × {*0*<..}. (*Ds* (*flow0* (*fst yt*) (*snd yt*))) (*f* (*flow0* (*fst yt*) (*snd yt*))) < *0* ∧
snd yt ∈ *existence-ivl0* (*fst yt*) ∧
0 ∈ *existence-ivl0* (*fst yt*)
by *eventually-elim auto*
then obtain *d2* **where** *0* < *d2* **and**
d2-neg: ⋀*y t*. (*y*, *t*) ∈ *cball* (*x*, *0*) *d2* ⟹ *0* < *t* ⟹ (*Ds* (*flow0 y t*)) (*f* (*flow0 y t*)) < *0*

and *d2-ex*: $\bigwedge y\ t.\ (y,\ t) \in cball\ (x,\ 0)\ d2 \Longrightarrow 0 < t \Longrightarrow t \in existence\text{-}ivl0\ y$
and *d2-ex0*: $\bigwedge y\ t.\ (y,\ t{::}real) \in cball\ (x,\ 0)\ d2 \Longrightarrow 0 < t \Longrightarrow y \in X$
by (*auto simp*: *eventually-at-le dist-commute*)
define *d* **where** $d \equiv d2\ /\ 2$
from ‹*0 < d2*› **have** $d > 0$ **by** (*simp add*: *d-def*)
have *d-neg*: $dist\ y\ x< d \Longrightarrow 0 < t \Longrightarrow t \leq d \Longrightarrow (Ds\ (flow0\ y\ t))\ (f\ (flow0\ y\ t)) < 0$ **for** *y t*
using *d2-neg*[*of y t*, *OF subsetD*[*OF cball-times-subset*[*of x d2 0*]]]
by (*auto simp*: *d-def dist-commute*)
have *d-ex*: $t \in existence\text{-}ivl0\ y$ **if** $dist\ y\ x< d\ 0 \leq t\ t \leq d$ **for** *y t*
proof *cases*
assume $t = 0$
have $sqrt\ ((dist\ x\ y)^2 + (d2\ /\ 2)^2) \leq dist\ x\ y + d2/2$
using ‹*0 < d2*›
by (*intro sqrt-sum-squares-le-sum*) *auto*
also have $dist\ x\ y \leq d2\ /\ 2$
using *that* **by** (*simp add*: *d-def dist-commute*)
finally have $sqrt\ ((dist\ x\ y)^2 + (d2\ /\ 2)^2) \leq d2$ **by** *simp*
with ‹*t = 0*› **show** *?thesis*
using *d2-ex*[*of y t*, *OF subsetD*[*OF cball-times-subset*[*of x d2 0*]]] *d2-ex0*[*of y d*] ‹*0 < d2*›
by (*auto simp*: *d-def dist-commute dist-prod-def*)
next
assume $t \neq 0$
then show *?thesis*
using *d2-ex*[*of y t*, *OF subsetD*[*OF cball-times-subset*[*of x d2 0*]]] *that*
by (*auto simp*: *d-def dist-commute*)
qed
have *d-mvt*: $s\ (flow0\ y\ t) < s\ y$ **if** $0 < t\ t \leq d\ dist\ y\ x < d$ **for** *y t*
proof −
have *c*: *continuous-on* $\{0\ ..\ t\}\ (\lambda t.\ s\ (flow0\ y\ t))$
using *that*
by (*auto intro*!: *continuous-intros d-ex*)
have *d*: $\bigwedge x.\ [\![0 < x;\ x < t]\!] \Longrightarrow ((\lambda t.\ s\ (flow0\ y\ t))\ has\text{-}derivative\ (\lambda t.\ t * blinfun\text{-}apply\ (Ds\ (flow0\ y\ x))\ (f\ (flow0\ y\ x))))\ (at\ x)$
using *that*
by (*auto intro*!: *derivative-eq-intros d-ex simp*: *flowderiv-def blinfun.bilinear-simps*)
from *mvt*[*OF* ‹*0 < t*› *c d*]
obtain *xi* **where** *xi*: $0 < xi\ xi < t$ **and** $s\ (flow0\ y\ t) - s\ (flow0\ y\ 0) = t * blinfun\text{-}apply\ (Ds\ (flow0\ y\ xi))\ (f\ (flow0\ y\ xi))$
by *auto*
note *this*(*3*)
also have $\ldots < 0$
using ‹*0 < t*›
apply (*rule mult-pos-neg*)
apply (*rule d-neg*)
using *that xi* **by** *auto*
also have $flow0\ y\ 0 = y$
apply (*rule flow-initial-time*)

apply *auto*
using ‹$0 < d$› *d-ex that*(*3*) **by** *fastforce*
finally show *?thesis*
by *auto*
qed
obtain *u eu* **where** *u*:
s (*flow0 x* (*u x*)) = *0*
u x = *return-time ?P x*
($\bigwedge$*y*. *y* ∈ *cball x eu* ⟹ *s* (*flow0 y* (*u y*)) = *0*)
continuous-on (*cball x eu*) *u*
(λ*t*. (*t*, *u t*)) ' *cball x eu* ⊆ *Sigma X existence-ivl0*
0 < *eu*
by (*rule returns-to-implicit-function*[*OF rt cS*(*1*) *Ds DsC through*]; *blast*)
have *u-tendsto*: (*u* ⟶ *u x*) (*at x*)
unfolding *isCont-def* [*symmetric*]
apply (*rule continuous-on-interior*[*OF u*(*4*)])
using ‹*0* < *eu*› **by** *auto*
have *u x* > *0* **by** (*auto simp*: *u intro*!: *return-time-pos rt*)
from *order-tendstoD*(*1*)[*OF u-tendsto this*] **have** ∀$_F$ *x in at x*. *0* < *u x* **.**
moreover have ∀$_F$ *y in at x*. *y* ∈ *cball x eu*
using *eventually-at-ball*[*OF* ‹*0* < *eu*›, *of x*]
by *eventually-elim auto*
moreover
have *x* ∉ *S* ∨ *s x* ≠ *0* ∨ *blinfun-apply* (*Ds x*) (*f x*) ≠ *0* **using** *inside* **by** *auto*
have *returns*: ∀$_F$ *y in at x*. *returns-to ?P y*
by (*rule eventually-returns-to*; *fact*)
moreover
have ∀$_F$ *y in at x*. *y* ∈ *ball x eu*
using *eventually-at-ball*[*OF* ‹*0* < *eu*›]
by *eventually-elim simp*
then have *ev-cball*: ∀$_F$ *y in at x*. *y* ∈ *cball x eu*
by *eventually-elim* (*use* ‹*e* > *0*› **in** *auto*)
have *continuous-on* (*ball x eu*) *u*
using *u* **by** (*auto simp*: *continuous-on-subset*)
then have [*tendsto-intros*]: (*u* ⟶ *u x*) (*at x*)
using ‹*eu* > *0*› *at-within-open*[*of y ball x eu* **for** *y*]
by (*auto simp*: *continuous-on-def*)
then have *flow0-u-tendsto*: (λ*x*. *flow0 x* (*u x*)) −*x*→ *poincare-map ?P x*
by (*auto intro*!: *tendsto-eq-intros u return-time-exivl rt simp*: *poincare-map-def*)
have *s-imp*: *s* (*poincare-map* {*x* ∈ *S*. *s x* = *0*} *x*) = *0* ⟶ *poincare-map* {*x* ∈ *S*. *s x* = *0*} *x* ∈ *S*
using *poincare-map-returns*[*OF rt*]
by *auto*
from *eventually-tendsto-compose-within*[*OF eventually-inside s-imp flow0-u-tendsto*]
have ∀$_F$ *x in at x*. *s* (*flow0 x* (*u x*)) = *0* ⟶ *flow0 x* (*u x*) ∈ *S* **by** *auto*
with *ev-cball*
have ∀$_F$ *x in at x*. *flow0 x* (*u x*) ∈ *S*
by *eventually-elim* (*auto simp*: *u*)
ultimately have *u-returns-ge*: ∀$_F$ *y in at x*. *returns-to ?P y* ∧ *return-time ?P*

```
y ≤ u y
  proof eventually-elim
    case (elim y)
    then show ?case
      using u elim by (auto intro!: return-time-le[OF - cls])
  qed
  moreover
  have ∀F y in at x. u y − return-time ?P x < e
    using tendstoD[OF u-tendsto ‹0 < e›, unfolded u] u-returns-ge
    by eventually-elim (auto simp: dist-real-def)
  moreover
  have d-less: d < return-time ?P x
    apply (rule return-time-gt)
      apply fact apply fact
    subgoal for t
      using d-mvt[of t x] ‹s x = 0› ‹0 < d›
      by auto
    done
  note 1 = inside
  define ml where ml = Max {return-time ?P x / 2, return-time ?P x − e, d}
  have [intro, simp, arith]: 0 < ml ml < return-time ?P x ml ≤ return-time ?P x
d ≤ ml
    using return-time-pos[OF rt cls] ‹0 < e› d-less
    by (auto simp: ml-def)
  have mt-in: ml ∈ existence-ivl0 x
    using ‹0 < e› ‹0 < d› d-less
   by (auto intro!: mem-existence-ivl-iv-defined in-existence-between-zeroI[OF ret-exivl]
        simp: closed-segment-eq-real-ivl ml-def)
  from open-existence-ivlE[OF mt-in]
  obtain e0 where e0: e0 > 0 cball x e0 × {0..ml + e0} ⊆ Sigma X existence-ivl0
(is ?D ⊆ -)
    by auto
  have uc: uniformly-continuous-on ((λ(x, t). flow0 x t) ‘ ?D) s
      apply (auto intro!: compact-uniformly-continuous continuous-on-subset[OF
cont-s])
    apply (rule compact-continuous-image)
     apply (rule continuous-on-subset)
      apply (rule flow-continuous-on-state-space)
     apply (rule e0)
    apply (rule compact-Times)
     apply (rule compact-cball)
    apply (rule compact-Icc)
    done
  let ?T = {d..ml}
  have ul: uniform-limit ?T flow0 (flow0 x) (at x)
    using ‹0 < e› ‹0 < d› d-less
    by (intro uniform-limit-flow)
    (auto intro!: mem-existence-ivl-iv-defined in-existence-between-zeroI[OF ret-exivl]
        simp: closed-segment-eq-real-ivl )
```

{
have ∀ $_F$ *y in at x within* {*x. s x* ≤ *0*}. *y* ∈ *X*
by (*rule topological-tendstoD*[*OF tendsto-ident-at open-dom* ‹*x* ∈ *X*›])
moreover
have ∀ $_F$ *y in at x within* {*x. s x* ≤ *0*}. *s y* ≤ *0*
by (*auto simp*: *eventually-at*)
moreover
have ∀ $_F$ *y in at x within* {*x. s x* ≤ *0*}. *Ds y* (*f y*) < *0*
by (*rule order-tendstoD*) (*auto intro*!: *tendsto-eq-intros inside*)
moreover
from *tendstoD*[*OF tendsto-ident-at* ‹*0* < *d*›]
have ∀ $_F$ *y in at x within* {*x. s x* ≤ *0*}. *dist y x* < *d*
by *auto*
moreover
have *d* ∈ *existence-ivl0 x*
using *d-ex*[*of x d*] ‹*0* < *d*› **by** *auto*
have *dret*: *returns-to* {*x*∈*S. s x* = *0*} (*flow0 x d*)
apply (*rule returns-to-laterI*)
apply *fact+*
subgoal for *u*
using *d-mvt*[*of u x*] ‹*s x* = *0*›
by *auto*
done
have ∀ $_F$ *y in at x.* ∀ *t*∈{*d..ml*}. *flow0 y t* ∈ − {*x* ∈ *S. s x* = *0*}
apply (*rule uniform-limit-in-open*)
apply (*rule ul*)
apply (*auto intro*!: *continuous-intros continuous-on-compose2*[*OF cont-s*]
simp:
split: *if-splits*)
using ‹*d* ∈ *existence-ivl0 x*› *mem-is-interval-1-I mt-in* **apply** *blast*
subgoal for *t*
using *return-time-least*[*OF rt cls*, *of t*] ‹*ml* < *return-time* {*x* ∈ *S. s x* = *0*}
x› ‹*0* < *d*›
by *auto*
done
then have ∀ $_F$ *y in at x within* {*x. s x* ≤ *0*}. ∀ *t*∈{*d* .. *ml*}. *flow0 y t* ∈ − {*x*
∈ *S. s x* = *0*}
by (*auto simp add*: *eventually-at*; *force*)
ultimately
have ∀ $_F$ *y in at x within* {*x. s x* ≤ *0*}. ∀ *t*∈{*0*<..*ml*}. *flow0 y t* ∈ − {*x* ∈ *S. s*
x = *0*}
apply *eventually-elim*
apply *auto*
using *d-mvt*
by *fastforce*
moreover
have ∀ $_F$ *y in at x. returns-to ?P y*
by *fact*
then have ∀ $_F$ *y in at x within* {*x. s x* ≤ *0*}. *returns-to ?P y*

by (*auto simp*: *eventually-at*)
ultimately
have $\forall_F$ *y in at x within* $\{x.\ s\ x \leq 0\}$. *return-time ?P y* $>$ *ml*
apply *eventually-elim*
apply (*rule return-time-gt*)
by *auto*
}
then have $\forall_F$ *y in at x within* $\{x.\ s\ x \leq 0\}$. *return-time ?P y* $\geq$ *return-time ?P x* $-$ *e*
by *eventually-elim* (*auto simp*: *ml-def*)
ultimately
have $\forall_F$ *y in at x within* $\{x\ .\ s\ x \leq 0\}$. *dist* (*return-time ?P y*) (*return-time ?P x*) $\leq$ *e*
unfolding *eventually-at-filter*
by *eventually-elim* (*auto simp*: *dist-real-def abs-real-def algebra-simps*)
then show $\forall_F$ *y in at x within* $\{x.\ s\ x \leq 0\}$. *dist* (*return-time ?P y*) (*return-time ?P x*) $<$ *e-orig*
by *eventually-elim* (*use* ‹*e-orig* $>$ *0*› **in** ‹*auto simp*: *e-def*›)
qed

lemma
return-time-continuous-below-plane:
fixes *s*::*'a*::*euclidean-space* $\Rightarrow$ *real*
assumes *rt*: *returns-to* $\{x \in R.\ x \cdot n = c\}$ *x* (**is** *returns-to ?P x*)
assumes *cR*: *closed R*
assumes *through*: *f* (*poincare-map ?P x*) $\cdot$ *n* $\neq$ *0*
assumes *R*: *x* $\in$ *R*
assumes *inside*: *x* $\cdot$ *n* $=$ *c f x* $\cdot$ *n* $<$ *0*
assumes *eventually-inside*: $\forall_F$ *x in at* (*poincare-map ?P x*). *x* $\cdot$ *n* $=$ *c* $\longrightarrow$ *x* $\in$ *R*
shows *continuous* (*at x within* $\{x.\ x \cdot n \leq c\}$) (*return-time ?P*)
apply (*rule return-time-continuous-below*[*of R* $\lambda x.\ x \cdot n - c$, *simplified*])
using *through rt inside cR R eventually-inside*
by (*auto intro*!: *derivative-eq-intros blinfun-inner-left.rep-eq*[*symmetric*])

lemma
poincare-map-in-interior-eventually-return-time-equal:
assumes *RP*: *R* $\subseteq$ *P*
assumes *cP*: *closed P*
assumes *cR*: *closed R*
assumes *ret*: *returns-to P x*
assumes *evret*: $\forall_F$ *x in at x within S*. *returns-to P x*
assumes *evR*: $\forall_F$ *x in at x within S*. *poincare-map P x* $\in$ *R*
shows $\forall_F$ *x in at x within S*. *returns-to R x* $\wedge$ *return-time P x* $=$ *return-time R x*
proof $-$
from *evret evR*
show *?thesis*
proof *eventually-elim*

case (*elim x*)
from *return-time-least*[*OF elim*(*1*) *cP*] *RP*
have *rtl*: $\bigwedge s.\ 0 < s \Longrightarrow s < \textit{return-time } P\ x \Longrightarrow \textit{flow0 } x\ s \notin R$
by *auto*
from *elim*(*2*) **have** *pR*: *poincare-map P x* ∈ *R*
by *auto*
have ∀$_F$ *t in at-right 0*. *0* < *t*
by (*simp add*: *eventually-at-filter*)
moreover have ∀$_F$ *t in at-right 0*. *t* < *return-time P x*
using *return-time-pos*[*OF elim*(*1*) *cP*]
by (*rule order-tendstoD*[*OF tendsto-ident-at*])
ultimately have *evR*: ∀$_F$ *t in at-right 0*. *flow0 x t* ∉ *R*
proof *eventually-elim*
case *et*: (*elim t*)
from *return-time-least*[*OF elim*(*1*) *cP et*] **show** *?case* **using** *RP* **by** *auto*
qed
have *rtp*: *0* < *return-time P x* **by** (*intro return-time-pos cP elim*)
have *rtex*: *return-time P x* ∈ *existence-ivl0 x* **by** (*intro return-time-exivl elim cP*)
have *frR*: *flow0 x* (*return-time P x*) ∈ *R*
unfolding *poincare-map-def*[*symmetric*] **by** (*rule pR*)
have *returns-to R x*
by (*rule returns-toI*[**where** *t*=*return-time P x*]; *fact*)
moreover have *return-time R x* = *return-time P x*
by (*rule return-time-eqI*) *fact*+
ultimately show *?case* **by** *auto*
qed
qed

lemma *poincare-map-in-planeI*:
assumes *returns-to* (*plane n c*) *x0*
shows *poincare-map* (*plane n c*) *x0* · *n* = *c*
using *poincare-map-returns*[*OF assms*]
by *fastforce*

lemma *less-return-time-imp-exivl*:
h ∈ *existence-ivl0 x′* **if** *h* ≤ *return-time P x′ returns-to P x′ closed P 0* ≤ *h*
proof −
from *return-time-exivl*[*OF that*(*2*,*3*)]
have *return-time P x′* ∈ *existence-ivl0 x′* **by** *auto*
from *ivl-subset-existence-ivl*[*OF this*] *that* **show** *?thesis*
by *auto*
qed

lemma *eventually-returns-to-continuousI*:
assumes *returns-to P x*
assumes *closed P*
assumes *continuous* (*at x within S*) (*return-time P*)
shows ∀$_F$ *x in at x within S*. *returns-to P x*

proof –
  have *return-time P x* > *0*
    using *assms* **by** (*auto simp*: *return-time-pos*)
  from *order-tendstoD*(*1*)[*OF assms*(*3*)[*unfolded continuous-within*] *this*]
  have $\forall_F$ *x in at x within S*. *0* < *return-time P x* **.**
  then show *?thesis*
    by *eventually-elim* (*auto simp*: *return-time-pos-returns-to*)
qed

lemma *return-time-implicit-functionE*:
  fixes *s*::*'a*::*euclidean-space* ⇒ *real*
  assumes *rt*: *returns-to* {*x* ∈ *S*. *s x* = *0*} *x* (**is** *returns-to ?P* -)
  assumes *cS*: *closed S*
  assumes *Ds*: ⋀*x*. (*s has-derivative blinfun-apply* (*Ds x*)) (*at x*)
  assumes *DsC*: ⋀*x*. *isCont Ds x*
  assumes *Ds-through*: (*Ds* (*poincare-map ?P x*)) (*f* (*poincare-map ?P x*)) ≠ *0*
  assumes *eventually-inside*: $\forall_F$ *x in at* (*poincare-map ?P x*). *s x* = *0* ⟶ *x* ∈ *S*
  assumes *outside*: *x* ∉ *S* ∨ *s x* ≠ *0*
  obtains *e'* **where**
    0 < *e'*
    ⋀*y*. *y* ∈ *ball x e'* ⟹ *returns-to ?P y*
    ⋀*y*. *y* ∈ *ball x e'* ⟹ *s* (*flow0 y* (*return-time ?P y*)) = *0*
    continuous-on (*ball x e'*) (*return-time ?P*)
    (⋀*y*. *y* ∈ *ball x e'* ⟹ *Ds* (*poincare-map ?P y*) o_L *flowderiv y* (*return-time ?P y*) o_L *embed2-blinfun* ∈ *invertibles-blinfun*)
    (⋀*U v sa*.
      (⋀*sa*. *sa* ∈ *U* ⟹ *s* (*flow0 sa* (*v sa*)) = *0*) ⟹
      return-time ?P x = *v x* ⟹
      continuous-on U v ⟹ *sa* ∈ *U* ⟹ *x* ∈ *U* ⟹ *U* ⊆ *ball x e'* ⟹ *connected U* ⟹ *open U* ⟹ *return-time ?P sa* = *v sa*)
    (*return-time ?P has-derivative*
      − *blinfun-scaleR-left* (*inverse* ((*Ds* (*poincare-map ?P x*)) (*f* (*poincare-map ?P x*)))) o_L
        (*Ds* (*poincare-map ?P x*) o_L *Dflow x* (*return-time ?P x*)))
      (*at x*)
proof –
  have *cont-s*: *continuous-on UNIV s* **by** (*rule has-derivative-continuous-on*[*OF Ds*])
  then have *s-tendsto*: (*s* ⟶ *s x*) (*at x*) **for** *x*
    by (*auto simp*: *continuous-on-def*)
  have *cls*[*simp*, *intro*]: *closed* {*x* ∈ *S*. *s x* = *0*}
    by (*rule closed-levelset-within*) (*auto intro*!: *cS continuous-on-subset*[*OF cont-s*])

  have *cont-Ds*: *continuous-on UNIV Ds*
    using *DsC* **by** (*auto simp*: *continuous-on-def isCont-def*)
  note [*tendsto-intros*] = *continuous-on-tendsto-compose*[*OF cont-Ds* - *UNIV-I*, *simplified*]
  note [*continuous-intros*] = *continuous-on-compose2*[*OF cont-Ds* - *subset-UNIV*]

have $\forall_F$ *x in at* (*poincare-map ?P x*). *s x* = *0* ⟶ *x* ∈ *S*
using *eventually-inside*
by *auto*
then obtain *U* **where** *open U poincare-map ?P x* ∈ *U* ⋀*x*. *x* ∈ *U* ⟹ *s x* = *0* ⟹ *x* ∈ *S*
using *poincare-map-returns*[*OF rt cls*]
by (*force simp*: *eventually-at-topological*)
have *s-imp*: *s* (*poincare-map ?P x*) = *0* ⟶ *poincare-map ?P x* ∈ *S*
using *poincare-map-returns*[*OF rt cls*]
by *auto*
have *outside-disj*: *x* ∉ *S* ∨ *s x* ≠ *0* ∨ *blinfun-apply* (*Ds x*) (*f x*) ≠ *0*
using *outside* **by** *auto*
have *pm-tendsto*: (*poincare-map ?P* ⟶ *poincare-map ?P x*) (*at x*)
apply (*rule poincare-map-tendsto*)
unfolding *isCont-def*[*symmetric*]
apply (*rule return-time-isCont-outside*)
using *assms*
by (*auto intro*!: *cls*)
have *evmemS*: $\forall_F$ *x in at x*. *poincare-map ?P x* ∈ *S*
using *eventually-returns-to*[*OF rt cS Ds DsC eventually-inside Ds-through outside-disj*]
apply *eventually-elim*
using *poincare-map-returns*
by *auto*
have $\forall_F$ *x in at x*. $\forall_F$ *x in at* (*poincare-map ?P x*). *s x* = *0* ⟶ *x* ∈ *S*
apply (*rule eventually-tendsto-compose-within*[*OF - - pm-tendsto*])
apply (*rule eventually-eventually-withinI*)
apply (*rule eventually-inside*)
apply (*rule s-imp*)
apply (*rule eventually-inside*)
apply (*rule evmemS*)
done
moreover
have *eventually* (λ*x*. *x* ∈ − *?P*) (*at x*)
apply (*rule topological-tendstoD*)
using *outside*
by (*auto intro*!:)
then have *eventually* (λ*x*. *x* ∉ *S* ∨ *s x* ≠ *0*) (*at x*)
by *auto*
moreover
have *eventually* (λ*x*. (*Ds* (*poincare-map ?P x*)) (*f* (*poincare-map ?P x*)) ≠ *0*) (*at x*)
apply (*rule tendsto-imp-eventually-ne*)
apply (*rule tendsto-intros*)
apply (*rule tendsto-intros*)
unfolding *poincare-map-def*
apply (*rule tendsto-intros*)
apply (*rule tendsto-intros*)
apply (*subst isCont-def*[*symmetric*])

apply (*rule return-time-isCont-outside*[*OF rt cS Ds DsC Ds-through eventually-inside outside*])
apply (*rule return-time-exivl*[*OF rt cls*])
apply (*rule tendsto-intros*)
apply (*rule tendsto-intros*)
apply (*rule tendsto-intros*)
apply (*subst isCont-def*[*symmetric*])
apply (*rule return-time-isCont-outside*[*OF rt cS Ds DsC Ds-through eventually-inside outside*])
apply (*rule return-time-exivl*[*OF rt cls*])
apply (*rule flow-in-domain*)
apply (*rule return-time-exivl*[*OF rt cls*])
unfolding *poincare-map-def*[*symmetric*]
apply (*rule Ds-through*)
done
ultimately
have *eventually* (λ*y. returns-to ?P y* $\wedge$ ($\forall_F$ *x in at* (*poincare-map ?P y*). *s x* = *0* $\longrightarrow$ *x* $\in$ *S*) $\wedge$
(*y* $\notin$ *S* $\vee$ *s y* $\neq$ *0*) $\wedge$ (*Ds* (*poincare-map ?P y*)) (*f* (*poincare-map ?P y*)) $\neq$ *0*) (*at x*)
using *eventually-returns-to*[*OF rt cS Ds DsC eventually-inside Ds-through outside-disj*]
by *eventually-elim auto*
then obtain *Y'* **where** *Y'*: *open Y' x* $\in$ *Y'* $\bigwedge$*y. y* $\in$ *Y'* $\Longrightarrow$ *returns-to ?P y*
$\bigwedge$*y. y* $\in$ *Y'* $\Longrightarrow$ ($\forall_F$ *x in at* (*poincare-map ?P y*). *s x* = *0* $\longrightarrow$ *x* $\in$ *S*)
$\bigwedge$*y. y* $\in$ *Y'* $\Longrightarrow$ *y* $\notin$ *S* $\vee$ *s y* $\neq$ *0*
$\bigwedge$*y. y* $\in$ *Y'* $\Longrightarrow$ *blinfun-apply* (*Ds* (*poincare-map ?P y*)) (*f* (*poincare-map ?P y*)) $\neq$ *0*
apply (*subst* (*asm*) (*3*) *eventually-at-topological*)
using *rt outside Ds-through eventually-inside*
by *fastforce*
from *openE*[*OF* ‹*open Y'*› ‹*x* $\in$ *Y'*›] **obtain** *eY* **where** *eY*: *0* < *eY ball x eY* $\subseteq$ *Y'* **by** *auto*
define *Y* **where** *Y* = *ball x eY*
then have *Y*: *open Y* **and** *x*: *x* $\in$ *Y*
and *Yr*: $\bigwedge$*y. y* $\in$ *Y* $\Longrightarrow$ *returns-to ?P y*
and *Y-mem*: $\bigwedge$*y. y* $\in$ *Y* $\Longrightarrow$ ($\forall_F$ *x in at* (*poincare-map ?P y*). *s x* = *0* $\longrightarrow$ *x* $\in$ *S*)
and *Y-nz*: $\bigwedge$*y. y* $\in$ *Y* $\Longrightarrow$ *y* $\notin$ *S* $\vee$ *s y* $\neq$ *0*
and *Y-fnz*: $\bigwedge$*y. y* $\in$ *Y* $\Longrightarrow$ *Ds* (*poincare-map ?P y*) (*f* (*poincare-map ?P y*)) $\neq$ *0*
and *Y-convex*: *convex Y*
using *Y' eY*
by (*auto simp*: *subset-iff dist-commute*)
have *isCont* (*return-time ?P*) *y* **if** *y* $\in$ *Y* **for** *y*
using *return-time-isCont-outside*[*OF Yr*[*OF that*] *cS Ds DsC Y-fnz Y-mem Y-nz*, *OF that that that*] **.**
then have *cY*: *continuous-on Y* (*return-time ?P*)
by (*auto simp*: *continuous-on-def isCont-def Lim-at-imp-Lim-at-within*)

note [*derivative-intros*] = *has-derivative-compose*[*OF - Ds*]
let *?t1* = *return-time ?P x*
have *t1-exivl*: *?t1* ∈ *existence-ivl0 x*
  by (*auto intro*!: *return-time-exivl rt*)
then have [*simp*]: *x* ∈ *X* **by** *auto*
have *xt1*: (*x*, *?t1*) ∈ *Sigma Y existence-ivl0*
  by (*auto intro*!: *return-time-exivl rt x*)
have *Sigma Y existence-ivl0* = *Sigma X existence-ivl0* ∩ *fst* −' *Y* **by** *auto*
also have *open* ...
  by (*rule open-Int*[*OF open-state-space open-vimage-fst*[*OF* ‹*open Y*›]])
finally have *open* (*Sigma Y existence-ivl0*) .
have *D*: (⋀*x*. *x* ∈ *Sigma Y 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*: *continuous-on* (*Sigma Y existence-ivl0*) (λ*x*. *Ds* (*flow0* (*fst x*) (*snd x*))
o_L *flowderiv* (*fst x*) (*snd x*))
  by (*auto intro*!: *continuous-intros*)
from *return-time-returns*[*OF rt cls*]
have *Z*: (*case* (*x*, *?t1*) *of* (*x*, *t*) ⇒ *s* (*flow0 x t*)) = *0*
  by (*auto simp*: *x*)
have *I1*: *blinfun-scaleR-left* (*inverse* (*Ds* (*flow0 x* (*?t1*))(*f* (*flow0 x* (*?t1*))))) o_L
  ((*Ds* (*flow0* (*fst* (*x*, *return-time ?P x*))
      (*snd* (*x*, *return-time ?P x*))) o_L
    flowderiv (*fst* (*x*, *return-time ?P x*))
     (*snd* (*x*, *return-time ?P x*))) o_L
    embed2-blinfun)
   = 1_L
  using *Ds-through*
  by (*auto intro*!: *blinfun-eqI*
    simp: *rt flowderiv-def blinfun.bilinear-simps inverse-eq-divide poincare-map-def*)
have *I2*: ((*Ds* (*flow0* (*fst* (*x*, *return-time ?P x*))
      (*snd* (*x*, *return-time ?P x*))) o_L
    flowderiv (*fst* (*x*, *return-time ?P x*))
     (*snd* (*x*, *return-time ?P x*))) o_L
    embed2-blinfun) o_L *blinfun-scaleR-left* (*inverse* (*Ds* (*flow0 x* (*?t1*))(*f* (*flow0 x*
(*?t1*)))))
   = 1_L
  using *Ds-through*
  by (*auto intro*!: *blinfun-eqI*
    simp: *rt flowderiv-def blinfun.bilinear-simps inverse-eq-divide poincare-map-def*)
obtain *u e* **where** *u*:
    s (*flow0 x* (*u x*)) = *0*
    u x = *return-time ?P x*
    (⋀*sa*. *sa* ∈ *cball x e* ⟹ *s* (*flow0 sa* (*u sa*)) = *0*)
    continuous-on (*cball x e*) *u*
    (λ*t*. (*t*, *u t*)) ' *cball x e* ⊆ *Sigma Y existence-ivl0*

0 < e
(*u has-derivative*
blinfun-apply
(− *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*)
(⋀*s*. *s* ∈ *cball x e* ⟹
Ds (*flow0 s* (*u s*)) o_L *flowderiv s* (*u s*) o_L *embed2-blinfun* ∈ *invertibles-blinfun*)
and *unique*: (⋀*U v sa*.
(⋀*sa*. *sa* ∈ *U* ⟹ *s* (*flow0 sa* (*v sa*)) = *0*) ⟹
u x = *v x* ⟹
continuous-on U v ⟹ *sa* ∈ *U* ⟹ *x* ∈ *U* ⟹ *U* ⊆ *cball x e* ⟹ *connected U* ⟹ *open U* ⟹ *u sa* = *v sa*)
apply (*rule implicit-function-theorem-unique*[**where** *f*=λ(*x*, *t*). *s* (*flow0 x t*)
and *S*=*Sigma Y existence-ivl0*, *OF D xt1* ‹*open* (*Sigma Y* -)› *order-refl C Z I1 I2*])
apply *blast*
unfolding *split-beta′ fst-conv snd-conv poincare-map-def*[*symmetric*]
apply (*rule*)
by (*assumption*+, *blast*)
have *u-rt*: *u y* = *return-time ?P y* **if** *y* ∈ *ball x e* ∩ *Y* **for** *y*
apply (*rule unique*[*of ball x e* ∩ *Y return-time ?P*])
subgoal for *y*
unfolding *poincare-map-def*[*symmetric*]
using *poincare-map-returns*[*OF Yr cls*]
by *auto*
subgoal by (*auto simp*: *u*)
subgoal using *cY* **by** (*rule continuous-on-subset*) *auto*
subgoal using *that* **by** *auto*
subgoal using *x* ‹*0 < e*› **by** *auto*
subgoal by *auto*
subgoal
apply (*rule convex-connected*)
apply (*rule convex-Int*)
apply *simp*
apply *fact*
done
subgoal by (*auto intro*!: *open-Int* ‹*open Y*›)
done

have ∗: (− *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) =
− *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 *Dflow x* (*return-time ?P x*))
by (*auto intro*!: *blinfun-eqI simp*: *flowderiv-def*)
define *e′* **where** *e′* = *min e eY*
have *e′-eq*: *ball x e′* = *ball x e* ∩ *Y* **by** (*auto simp*: *e′-def Y-def*)
have
0 < *e′*
⋀*y*. *y* ∈ *ball x e′* ⟹ *returns-to ?P y*
⋀*y*. *y* ∈ *ball x e′* ⟹ *s* (*flow0 y* (*return-time ?P y*)) = *0*
continuous-on (*ball x e′*) (*return-time ?P*)
(⋀*y*. *y* ∈ *ball x e′* ⟹ *Ds* (*poincare-map ?P y*) o_L *flowderiv y* (*return-time ?P y*) o_L *embed2-blinfun* ∈ *invertibles-blinfun*)
(⋀*U v sa*.
(⋀*sa*. *sa* ∈ *U* ⟹ *s* (*flow0 sa* (*v sa*)) = *0*) ⟹
return-time ?P x = *v x* ⟹
continuous-on U v ⟹ *sa* ∈ *U* ⟹ *x* ∈ *U* ⟹ *U* ⊆ *ball x e′* ⟹ *connected U* ⟹ *open U* ⟹ *return-time ?P sa* = *v sa*)
(*return-time ?P has-derivative blinfun-apply*
(− *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*)
unfolding *e′-eq*
subgoal by (*auto simp*: *e′-def* ‹*0* < *e*› ‹*0* < *eY*›)
subgoal by (*rule Yr*) *auto*
subgoal for *y*
unfolding *poincare-map-def*[*symmetric*]
using *poincare-map-returns*[*OF Yr cls*]
by *auto*
subgoal using *cY* **by** (*rule continuous-on-subset*) *auto*
subgoal premises *prems* **for** *y*
unfolding *poincare-map-def*
unfolding *u-rt*[*OF prems*, *symmetric*]
apply (*rule u*)
using *prems* **by** *auto*
subgoal premises *prems* **for** *U v t*
apply (*subst u-rt*[*symmetric*])
subgoal using *prems* **by** *force*
apply (*rule unique*[*of U v*])
subgoal by *fact*
subgoal by (*auto simp*: *u prems*)
subgoal by *fact*
subgoal by *fact*
subgoal by *fact*
subgoal using *prems* **by** *auto*
subgoal by *fact*
subgoal by *fact*

done
subgoal
proof −
have $\forall_F$ *x′ in at x. x′ ∈ ball x e′*
using *eventually-at-ball*[*OF* ‹*0* < *e′*›]
by *eventually-elim simp*
then have $\forall_F$ *x′ in at x. u x′* = *return-time ?P x′*
unfolding *e′-eq*
by *eventually-elim* (*rule u-rt*, *auto*)
from *u*(*7*) *this*
show *?thesis*
by (*rule has-derivative-transform-eventually*) (*auto simp*: *u*)
qed
done
then show *?thesis* **unfolding** ∗ ..
qed

lemma *return-time-has-derivative*:
fixes *s*::′*a*::*euclidean-space* ⇒ *real*
assumes *rt*: *returns-to* {*x* ∈ *S*. *s x* = *0*} *x* (**is** *returns-to ?P* -)
assumes *cS*: *closed S*
assumes *Ds*: ⋀*x*. (*s has-derivative blinfun-apply* (*Ds x*)) (*at x*)
assumes *DsC*: ⋀*x*. *isCont Ds x*
assumes *Ds-through*: (*Ds* (*poincare-map ?P x*)) (*f* (*poincare-map ?P x*)) ≠ *0*
assumes *eventually-inside*: $\forall_F$ *x in at* (*poincare-map* {*x* ∈ *S*. *s x* = *0*} *x*). *s x* = *0* ⟶ *x* ∈ *S*
assumes *outside*: *x* ∉ *S* ∨ *s x* ≠ *0*
shows (*return-time ?P has-derivative*
− *blinfun-scaleR-left* (*inverse* ((*Ds* (*poincare-map ?P x*)) (*f* (*poincare-map ?P x*)))) o_L
(*Ds* (*poincare-map ?P x*) o_L *Dflow x* (*return-time ?P x*)))
(*at x*)
using *return-time-implicit-functionE*[*OF assms*] **by** *blast*

lemma *return-time-plane-has-derivative-blinfun*:
assumes *rt*: *returns-to* {*x* ∈ *S*. *x* · *i* = *c*} *x* (**is** *returns-to ?P* -)
assumes *cS*: *closed S*
assumes *fnz*: *f* (*poincare-map ?P x*) · *i* ≠ *0*
assumes *eventually-inside*: $\forall_F$ *x in at* (*poincare-map ?P x*). *x* · *i* = *c* ⟶ *x* ∈ *S*
assumes *outside*: *x* ∉ *S* ∨ *x* · *i* ≠ *c*
shows (*return-time ?P has-derivative*
(− *blinfun-scaleR-left* (*inverse* ((*blinfun-inner-left i*) (*f* (*poincare-map ?P x*)))) o_L
(*blinfun-inner-left i* o_L *Dflow x* (*return-time ?P x*)))) (*at x*)
proof −
have *rt*: *returns-to* {*x* ∈ *S*. *x* · *i* − *c* = *0*} *x*
using *rt* **by** *auto*
have *D*: ((λ*x*. *x* · *i* − *c*) *has-derivative blinfun-inner-left i*) (*at x*) **for** *x*
by (*auto intro*!: *derivative-eq-intros*)

have *DC*: ($\bigwedge$*x*. *isCont* (λ*x*. *blinfun-inner-left i*) *x*)
by (*auto intro*!: *continuous-intros*)
have *nz*: *blinfun-apply* (*blinfun-inner-left i*) (*f* (*poincare-map* {*x* ∈ *S*. *x* · *i* − *c* = *0*} *x*)) ≠ *0*
using *fnz* **by** (*auto*)
from *cS* **have** *cS*: *closed S***by** *auto*
have *out*: *x* ∉ *S* ∨ *x* · *i* − *c* ≠ *0* **using** *outside* **by** *simp*
from *eventually-inside*
have *eventually-inside*: ∀ $_F$ *x in at* (*poincare-map* {*x* ∈ *S*. *x* · *i* − *c* = *0*} *x*). *x* · *i* − *c* = *0* ⟶ *x* ∈ *S*
by *auto*
from *return-time-has-derivative*[*OF rt cS D DC nz eventually-inside out*]
show *?thesis*
by *auto*
qed

lemma *return-time-plane-has-derivative*:
assumes *rt*: *returns-to* {*x* ∈ *S*. *x* · *i* = *c*} *x* (**is** *returns-to ?P* -)
assumes *cS*: *closed S*
assumes *fnz*: *f* (*poincare-map ?P x*) · *i* ≠ *0*
assumes *eventually-inside*: ∀ $_F$ *x in at* (*poincare-map ?P x*). *x* · *i* = *c* ⟶ *x* ∈ *S*
assumes *outside*: *x* ∉ *S* ∨ *x* · *i* ≠ *c*
shows (*return-time ?P has-derivative*
(λ*h*. − (*Dflow x* (*return-time ?P x*)) *h* · *i* / (*f* (*poincare-map ?P x*) · *i*))) (*at x*)
by (*rule return-time-plane-has-derivative-blinfun*[*OF assms*, *THEN has-derivative-eq-rhs*])
(*auto simp*: *blinfun.bilinear-simps flowderiv-def inverse-eq-divide intro*!: *ext*)

definition *Dpoincare-map i c S x* =
(λ*h*. (*Dflow x* (*return-time* {*x* ∈ *S*. *x* · *i* = *c*} *x*)) *h* −
((*Dflow x* (*return-time* {*x* ∈ *S*. *x* · *i* = *c*} *x*)) *h* · *i* /
(*f* (*poincare-map* {*x* ∈ *S*. *x* · *i* = *c*} *x*) · *i*)) $*_R$ *f* (*poincare-map* {*x* ∈ *S*. *x* · *i* = *c*} *x*))

definition *Dpoincare-map′ i c S x* =
Dflow x (*return-time* {*x* ∈ *S*. *x* · *i* − *c* = *0*} *x*) −
(*blinfun-scaleR-left* (*f* (*poincare-map* {*x* ∈ *S*. *x* · *i* = *c*} *x*)) o_L
(*blinfun-scaleR-left* (*inverse* ((*f* (*poincare-map* {*x* ∈ *S*. *x* · *i* = *c*} *x*) · *i*))) o_L
(*blinfun-inner-left i* o_L *Dflow x* (*return-time* {*x* ∈ *S*. *x* · *i* − *c* = *0*} *x*))))

theorem *poincare-map-plane-has-derivative*:
assumes *rt*: *returns-to* {*x* ∈ *S*. *x* · *i* = *c*} *x* (**is** *returns-to ?P* -)
assumes *cS*: *closed S*
assumes *fnz*: *f* (*poincare-map ?P x*) · *i* ≠ *0*
assumes *eventually-inside*: ∀ $_F$ *x in at* (*poincare-map ?P x*). *x* · *i* = *c* ⟶ *x* ∈ *S*
assumes *outside*: *x* ∉ *S* ∨ *x* · *i* ≠ *c*
notes [*derivative-intros*] = *return-time-plane-has-derivative*[*OF rt cS fnz eventually-inside outside*]
shows (*poincare-map ?P has-derivative Dpoincare-map′ i c S x*) (*at x*)
unfolding *poincare-map-def Dpoincare-map′-def*

using *fnz outside*
by (*auto intro*!: *derivative-eq-intros return-time-exivl assms ext closed-levelset-within continuous-intros*
simp: *flowderiv-eq poincare-map-def blinfun.bilinear-simps inverse-eq-divide algebra-simps*)

end

end
theory *Reachability-Analysis*
imports
Flow
Poincare-Map
begin

lemma *not-mem-eq-mem-not*: $a \notin A \longleftrightarrow a \in - A$
by *auto*

lemma *continuous-orderD*:
fixes *g*::*'b*::*t2-space* ⇒ *'c*::*order-topology*
assumes *continuous* (*at x within S*) *g*
shows $g\ x > c \Longrightarrow \forall_F\ y\ in\ at\ x\ within\ S.\ g\ y > c$
$g\ x < c \Longrightarrow \forall_F\ y\ in\ at\ x\ within\ S.\ g\ y < c$
using *order-tendstoD*[*OF assms*[*unfolded continuous-within*]]
by *auto*

lemma *frontier-halfspace-component-ge*: $n \neq 0 \Longrightarrow$ *frontier* $\{x.\ c \le x \cdot n\}$ = *plane n c*
apply (*subst* (*1*) *inner-commute*)
apply (*subst* (*2*) *inner-commute*)
apply (*subst frontier-halfspace-ge*[*of n c*])
by *auto*

lemma *closed-Collect-le-within*:
fixes *f g* :: *'a* :: *topological-space* ⇒ *'b*::*linorder-topology*
assumes *f*: *continuous-on UNIV f*
and *g*: *continuous-on UNIV g*
and *closed R*
shows *closed* $\{x \in R.\ f\ x \le g\ x\}$
proof −
have ∗: $- R \cup \{x.\ g\ x < f\ x\} = - \{x \in R.\ f\ x \le g\ x\}$
by *auto*
have *open* (−*R*) **using** *assms* **by** *auto*
from *open-Un*[*OF this open-Collect-less* [*OF g f*], *unfolded* ∗]
show *?thesis*
by (*simp add*: *closed-open*)
qed

6.1 explicit representation of hyperplanes / halfspaces

datatype $'a$ *sctn* = *Sctn* (*normal*: $'a$) (*pstn*: *real*)

definition *le-halfspace sctn x* $\longleftrightarrow$ *x* $\cdot$ *normal sctn* $\leq$ *pstn sctn*

definition *lt-halfspace sctn x* $\longleftrightarrow$ *x* $\cdot$ *normal sctn* $<$ *pstn sctn*

definition *ge-halfspace sctn x* $\longleftrightarrow$ *x* $\cdot$ *normal sctn* $\geq$ *pstn sctn*

definition *gt-halfspace sctn x* $\longleftrightarrow$ *x* $\cdot$ *normal sctn* $>$ *pstn sctn*

definition *plane-of sctn* = $\{$*x*. *x* $\cdot$ *normal sctn* = *pstn sctn*$\}$

definition *above-halfspace sctn* = *Collect* (*ge-halfspace sctn*)

definition *below-halfspace sctn* = *Collect* (*le-halfspace sctn*)

definition *sbelow-halfspace sctn* = *Collect* (*lt-halfspace sctn*)

definition *sabove-halfspace sctn* = *Collect* (*gt-halfspace sctn*)

6.2 explicit H representation of polytopes (mind *Polytopes.thy*)

definition *below-halfspaces*
where *below-halfspaces sctns* = $\bigcap$(*below-halfspace* ‘ *sctns*)

definition *sbelow-halfspaces*
where *sbelow-halfspaces sctns* = $\bigcap$(*sbelow-halfspace* ‘ *sctns*)

definition *above-halfspaces*
where *above-halfspaces sctns* = $\bigcap$(*above-halfspace* ‘ *sctns*)

definition *sabove-halfspaces*
where *sabove-halfspaces sctns* = $\bigcap$(*sabove-halfspace* ‘ *sctns*)

lemmas *halfspace-simps* =
above-halfspace-def
sabove-halfspace-def
below-halfspace-def
sbelow-halfspace-def
below-halfspaces-def
sbelow-halfspaces-def
above-halfspaces-def
sabove-halfspaces-def
ge-halfspace-def[*abs-def*]
gt-halfspace-def[*abs-def*]
le-halfspace-def[*abs-def*]
lt-halfspace-def[*abs-def*]

6.3 predicates for reachability analysis

context *c1-on-open-euclidean*
begin

definition *flowpipe* ::
$((\prime a{::}euclidean\text{-}space) \times (\prime a \Rightarrow_L \prime a))\ set \Rightarrow real \Rightarrow real \Rightarrow$
$(\prime a \times (\prime a \Rightarrow_L \prime a))\ set \Rightarrow (\prime a \times (\prime a \Rightarrow_L \prime a))\ set \Rightarrow bool$
where $flowpipe\ X0\ hl\ hu\ CX\ X1 \longleftrightarrow 0 \le hl \wedge hl \le hu \wedge fst\ \text{`}\ X0 \subseteq X \wedge fst\ \text{`}\ CX \subseteq X \wedge fst\ \text{`}\ X1 \subseteq X \wedge$
$(\forall (x0,\ d0) \in X0.\ \forall h \in \{hl\ ..\ hu\}.$
$h \in existence\text{-}ivl0\ x0 \wedge (flow0\ x0\ h,\ Dflow\ x0\ h\ o_L\ d0) \in X1 \wedge (\forall h' \in \{0\ ..\ h\}.\ (flow0\ x0\ h',\ Dflow\ x0\ h'\ o_L\ d0) \in CX))$

lemma *flowpipeD*:
assumes *flowpipe X0 hl hu CX X1*
shows *flowpipe-safeD*: $fst\ \text{`}\ X0 \cup fst\ \text{`}\ CX \cup fst\ \text{`}\ X1 \subseteq X$
and *flowpipe-nonneg*: $0 \le hl\ hl \le hu$
and *flowpipe-exivl*: $hl \le h \Longrightarrow h \le hu \Longrightarrow (x0,\ d0) \in X0 \Longrightarrow h \in existence\text{-}ivl0\ x0$
and *flowpipe-discrete*: $hl \le h \Longrightarrow h \le hu \Longrightarrow (x0,\ d0) \in X0 \Longrightarrow (flow0\ x0\ h,\ Dflow\ x0\ h\ o_L\ d0) \in X1$
and *flowpipe-cont*: $hl \le h \Longrightarrow h \le hu \Longrightarrow (x0,\ d0) \in X0 \Longrightarrow 0 \le h' \Longrightarrow h' \le h \Longrightarrow (flow0\ x0\ h',\ Dflow\ x0\ h'\ o_L\ d0) \in CX$
using *assms*
by (*auto simp*: *flowpipe-def*)

lemma *flowpipe-source-subset*: $flowpipe\ X0\ hl\ hu\ CX\ X1 \Longrightarrow X0 \subseteq CX$
apply (*auto dest*: *bspec*[**where** *x=hl*] *bspec*[**where** *x=0*] *simp*: *flowpipe-def*)
apply (*drule bspec*)
apply (*assumption*)
apply *auto*
apply (*drule bspec*[**where** *x=hl*])
apply *auto*
apply (*drule bspec*[**where** *x=0*])
by (*auto simp*: *flow-initial-time-if*)

definition $flowsto\ X0\ T\ CX\ X1 \longleftrightarrow$
$(\forall (x0,\ d0) \in X0.\ \exists h \in T.\ h \in existence\text{-}ivl0\ x0 \wedge (flow0\ x0\ h,\ Dflow\ x0\ h\ o_L\ d0) \in X1 \wedge (\forall h' \in open\text{-}segment\ 0\ h.\ (flow0\ x0\ h',\ Dflow\ x0\ h'\ o_L\ d0) \in CX))$

lemma *flowsto-to-empty-iff* [*simp*]: $flowsto\ a\ t\ b\ \{\} \longleftrightarrow a = \{\}$
by (*auto simp*: *simp*: *flowsto-def*)

lemma *flowsto-from-empty-iff* [*simp*]: $flowsto\ \{\}\ t\ b\ c$
by (*auto simp*: *simp*: *flowsto-def*)

lemma *flowsto-empty-time-iff* [*simp*]: $flowsto\ a\ \{\}\ b\ c \longleftrightarrow a = \{\}$
by (*auto simp*: *simp*: *flowsto-def*)

lemma *flowstoE*:
  assumes *flowsto X0 T CX X1* (*x0*, *d0*) ∈ *X0*
  obtains *h* **where** *h* ∈ *T h* ∈ *existence-ivl0 x0* (*flow0 x0 h*, *Dflow x0 h* o_L *d0*) ∈ *X1*
    ⋀*h'*. *h'* ∈ *open-segment 0 h* ⟹ (*flow0 x0 h'*, *Dflow x0 h'* o_L *d0*) ∈ *CX*
  using *assms*
  by (*auto simp*: *flowsto-def*)

lemma *flowsto-safeD*: *flowsto X0 T CX X1* ⟹ *fst* ' *X0* ⊆ *X*
  by (*auto simp*: *flowsto-def split-beta' mem-existence-ivl-iv-defined*)

lemma *flowsto-union*:
  assumes *1*: *flowsto X0 T CX Y* **and** *2*: *flowsto Z S CZ W*
  shows *flowsto* (*X0* ∪ *Z*) (*T* ∪ *S*) (*CX* ∪ *CZ*) (*Y* ∪ *W*)
  using *assms* **unfolding** *flowsto-def*
  by *force*

lemma *flowsto-subset*:
  assumes *flowsto X0 T CX Y*
  assumes *Z* ⊆ *X0 T* ⊆ *S CX* ⊆ *CZ Y* ⊆ *W*
  shows *flowsto Z S CZ W*
  unfolding *flowsto-def*
  using *assms*
  by (*auto elim*!: *flowstoE*) *blast*

lemmas *flowsto-unionI* = *flowsto-subset*[*OF flowsto-union*]

lemma *flowsto-unionE*:
  assumes *flowsto X0 T CX* (*Y* ∪ *Z*)
  obtains *X1 X2* **where** *X0* = *X1* ∪ *X2 flowsto X1 T CX Y flowsto X2 T CX Z*
proof −
  let *?X1* = {*x*∈*X0*. *flowsto* {*x*} *T CX Y*}
  let *?X2* = {*x*∈*X0*. *flowsto* {*x*} *T CX Z*}
  from *assms* **have** *X0* = *?X1* ∪ *?X2 flowsto ?X1 T CX Y flowsto ?X2 T CX Z*
    by (*auto simp*: *flowsto-def*)
  thus *?thesis* **..**
qed

lemma *flowsto-trans*:
  assumes *A*: *flowsto A S B C* **and** *C*: *flowsto C T D E*
  shows *flowsto A* {*s* + *t* |*s t*. *s* ∈ *S* ∧ *t* ∈ *T*} (*B* ∪ *D* ∪ *C*) *E*
  unfolding *flowsto-def*
proof *safe*
  fix *x0 d0* **assume** *x0*: (*x0*, *d0*) ∈ *A*
  from *flowstoE*[*OF A x0*]
  obtain *h* **where** *h*: *h* ∈ *S h* ∈ *existence-ivl0 x0* (*flow0 x0 h*, (*Dflow x0 h*) o_L *d0*) ∈ *C*
    ⋀*h'*. *h'* ∈ {*0*<−−<*h*} ⟹ (*flow0 x0 h'*, *Dflow x0 h'* o_L *d0*) ∈ *B*
    by *auto*

from *h*(*2*) **have** *x0*[*simp*]: *x0* ∈ *X* **by** *auto*
from *flowstoE*[*OF C* ‹- ∈ *C*›]
obtain *i* **where** *i*: *i* ∈ *T i* ∈ *existence-ivl0* (*flow0 x0 h*)
(*flow0* (*flow0 x0 h*) *i*, *Dflow* (*flow0 x0 h*) *i* o_L *Dflow x0 h* o_L *d0*) ∈ *E*
⋀*h*′. *h*′ ∈ {*0*<−−<*i*} ⟹ (*flow0* (*flow0 x0 h*) *h*′, *Dflow* (*flow0 x0 h*) *h*′ o_L (*Dflow x0 h* o_L *d0*)) ∈ *D*
by (*auto simp*: *ac-simps*)
have *hi*: *h* + *i* ∈ *existence-ivl0 x0*
using ‹*h* ∈ *existence-ivl0 x0*› ‹*i* ∈ *existence-ivl0* (*flow0 x0 h*)› *existence-ivl-trans* **by** *blast*
moreover have (*flow0 x0* (*h* + *i*), *Dflow x0* (*h* + *i*) o_L *d0*) ∈ *E*
apply (*subst flow-trans*)
apply *fact* **apply** *fact*
apply (*subst Dflow-trans*)
apply *fact* **apply** *fact*
apply *fact*
done
moreover have (*flow0 x0 h*′, *Dflow x0 h*′ o_L *d0*) ∈ *B* ∪ *D* ∪ *C* **if** *h*′∈{*0*<−−<*h* + *i*} **for** *h*′
proof *cases*
assume *h*′ ∈ {*0* <−−< *h*}
then show *?thesis* **using** *h* **by** *simp*
next
assume *h*′ ∉ {*0* <−−< *h*}
with *that* **have** *h*′: *h*′ − *h* ∈ {*0* <−−< *i*} **if** *h*′ ≠ *h*
using *that*
by (*auto simp*: *open-segment-eq-real-ivl closed-segment-eq-real-ivl split*: *if-splits*)
from *i*(*4*)[*OF this*]
show *?thesis*
apply (*cases h*′ = *h*)
subgoal using *h* **by** *force*
subgoal
apply *simp*
apply (*subst* (*asm*) *flow-trans*[*symmetric*])
subgoal by (*rule h*)
subgoal using ‹- ⟹ *h*′ − *h* ∈ {*0*<−−<*i*}› *i*(*2*) *local.in-existence-between-zeroI*
apply *auto*
using *open-closed-segment* **by** *blast*
subgoal
unfolding *blinfun-compose-assoc*[*symmetric*]
apply (*subst* (*asm*) *Dflow-trans*[*symmetric*])
apply *auto*
apply *fact*+
done
done
done
qed
ultimately show ∃*h*∈{*s* + *t* |*s t*. *s* ∈ *S* ∧ *t* ∈ *T*}.
h ∈ *existence-ivl0 x0* ∧ (*flow0 x0 h*, *Dflow x0 h* o_L *d0*) ∈ *E* ∧ (∀ *h*′∈{*0*<−−<*h*}.

$(flow0\ x0\ h', Dflow\ x0\ h'\ o_L\ d0) \in B \cup D \cup C)$
  using ‹$h \in S$› ‹$i \in T$›
  by (*auto intro*!: *bexI*[**where** $x=h + i$])
qed

lemma *flowsto-step*:
 assumes *A*: *flowsto A S B C*
 assumes *D*: *flowsto D T E F*
 shows *flowsto A* $(S \cup \{s + t \mid s\ t.\ s \in S \wedge t \in T\})$ $(B \cup E \cup C \cap D)$ $(C - D \cup F)$
proof −
 have $C = (C \cap D) \cup (C - D)$ (**is** - = *?C1* ∪ *?C2*)
  by *auto*
 then have *flowsto A S B* (*?C1* ∪ *?C2*) **using** *A* **by** *simp*
 from *flowsto-unionE*[*OF this*]
 obtain *A1 A2* **where** $A = A1 \cup A2$ **and** *A1*: *flowsto A1 S B ?C1* **and** *A2*: *flowsto A2 S B ?C2*
  by *auto*
 have *flowsto ?C1 T E F*
  using *D* **by** (*rule flowsto-subset*) *auto*
 from *flowsto-union*[*OF flowsto-trans*[*OF A1 this*] *A2*]
 show *?thesis* **by** (*auto simp add*: ‹$A = -$› *ac-simps*)
qed

lemma
 flowsto-stepI:
  flowsto X0 U B C ⟹
  flowsto D T E F ⟹
  $Z \subseteq X0$ ⟹
  $(\bigwedge s.\ s \in U \Longrightarrow s \in S) \Longrightarrow$
  $(\bigwedge s\ t.\ s \in U \Longrightarrow t \in T \Longrightarrow s + t \in S) \Longrightarrow$
  $B \cup E \cup D \cap C \subseteq CZ \Longrightarrow C - D \cup F \subseteq W \Longrightarrow$ *flowsto Z S CZ W*
 by (*rule flowsto-subset*[*OF flowsto-step*]) *auto*

lemma *flowsto-imp-flowsto*:
 flowpipe Y h h CY Z ⟹ *flowsto Y* $\{h\}$ (*CY*) *Z*
 unfolding *flowpipe-def flowsto-def*
 by (*auto simp*: *open-segment-eq-real-ivl split-beta′*)

lemma *connected-below-halfspace*:
 assumes $x \in$ *below-halfspace sctn*
 assumes $x \in S$ *connected S*
 assumes $S \cap$ *plane-of sctn* = {}
 shows $S \subseteq$ *below-halfspace sctn*
proof −
 note ‹*connected S*›
 moreover
 have *open* $\{x.\ x \cdot$ *normal sctn* < *pstn sctn*} (**is** *open ?X*)
  and *open* $\{x.\ x \cdot$ *normal sctn* > *pstn sctn*} (**is** *open ?Y*)

by (*auto intro*!: *open-Collect-less continuous-intros*)
moreover have *?X* ∩ *?Y* ∩ *S* = {} *S* ⊆ *?X* ∪ *?Y*
using *assms* **by** (*auto simp*: *plane-of-def*)
ultimately have *?X* ∩ *S* = {} ∨ *?Y* ∩ *S* = {}
by (*rule connectedD*)
then show *?thesis*
using *assms*
by (*force simp*: *below-halfspace-def le-halfspace-def plane-of-def*)
qed

lemma
inter-Collect-eq-empty:
assumes ⋀*x*. *x* ∈ *X0* ⟹ ¬ *g x* **shows** *X0* ∩ *Collect g* = {}
using *assms* **by** *auto*

6.4 Poincare Map

lemma *closed-plane-of*[*simp*]: *closed* (*plane-of sctn*)
by (*auto simp*: *plane-of-def intro*!: *closed-Collect-eq continuous-intros*)

definition *poincare-mapsto P X0 S CX Y* ⟷ (∀ (*x*, *d*) ∈ *X0*.
returns-to P x ∧ *fst* ‘ *X0* ⊆ *S* ∧
(*return-time P differentiable at x within S*) ∧
(∃ *D*. (*poincare-map P has-derivative blinfun-apply D*) (*at x within S*) ∧
(*poincare-map P x*, *D* o_L *d*) ∈ *Y*) ∧
(∀ *t*∈{*0*<..<*return-time P x*}. *flow0 x t* ∈ *CX*))

lemma *poincare-mapsto-empty*[*simp*]:
poincare-mapsto P {} *S CX Y*
by (*auto simp*: *poincare-mapsto-def*)

lemma *flowsto-eventually-mem-cont*:
assumes *flowsto X0 T CX Y* (*x*, *d*) ∈ *X0 T* ⊆ {*0*<..}
shows $\forall_F$ *t in at-right 0*. (*flow0 x t*, *Dflow x t* o_L *d*) ∈ *CX*
proof −
from *flowstoE*[*OF assms*(*1*,*2*)] *assms*(*3*)
obtain *h* **where** *h*: *0* < *h h* ∈ *T h* ∈ *existence-ivl0 x* (*flow0 x h*, (*Dflow x h*) o_L
d) ∈ *Y* ⋀*h*′. *h*′ ∈ {*0*<−−<*h*} ⟹ (*flow0 x h*′, (*Dflow x h*′) o_L *d*) ∈ *CX*
by (*auto simp*: *subset-iff*)
have $\forall_F$ *x in at-right 0*. *0* < *x* ∧ *x* < *h*
apply (*rule eventually-conj*[*OF eventually-at-right-less*])
using *eventually-at-right h*(*1*) **by** *blast*
then show *?thesis*
by *eventually-elim* (*auto intro*!: *h simp*: *open-segment-eq-real-ivl*)
qed

lemma *frontier-aux-lemma*:
fixes *R* :: ′*n*::*euclidean-space set*
assumes *closed R R* ⊆ {*x*. *x* · *n* = *c*} **and** [*simp*]: *n* ≠ *0*

shows *frontier* $\{x \in R.\ c \leq x \cdot n\} = \{x \in R.\ c = x \cdot n\}$
apply (*auto simp*: *frontier-closures*)
subgoal by (*metis* (*full-types*) *Collect-subset assms*(*1*) *closure-minimal subsetD*)
subgoal premises *prems* **for** *x*
proof −
  note *prems*
  have *closed* $\{x \in R.\ c \leq x \cdot n\}$
    by (*auto intro*!: *closed-Collect-le-within continuous-intros assms*)
  from *closure-closed*[*OF this*] *prems*(*1*)
  have $x \in R$ $c \leq x \cdot n$ **by** *auto*
  with *assms* **show** *?thesis* **by** *auto*
qed
subgoal for *x*
  using *closure-subset* **by** *fastforce*
subgoal premises *prems* **for** *x*
proof −
  note *prems*
  have ∗: $\{xa \in R.\ x \cdot n \leq xa \cdot n\} = R$
    using *assms prems* **by** *auto*
  have *interior* $R \subseteq$ *interior* (*plane n c*)
    by (*rule interior-mono*) (*use assms* **in** *auto*)
  also have ... = {}
    by (*subst inner-commute*) *simp*
  finally have *R*: *interior* $R = \{\}$ **by** *simp*
  have $x \in$ *closure* (− *R*)
    unfolding *closure-complement*
    by (*auto simp*: *R*)
  then show *?thesis*
    unfolding ∗ **by** *simp*
qed
done

lemma *blinfun-minus-comp-distrib*: $(a - b)\ o_L\ c = (a\ o_L\ c) - (b\ o_L\ c)$
  by (*auto intro*!: *blinfun-eqI simp*: *blinfun.bilinear-simps*)

lemma *flowpipe-split-at-above-halfspace*:
  assumes *flowpipe X0 hl t CX Y fst ‘ X0* $\cap \{x.\ x \cdot n \geq c\} = \{\}$ **and** [*simp*]: $n \neq 0$
  assumes *cR*: *closed R* **and** *Rs*: $R \subseteq$ *plane n c*
  assumes *PDP*: $\bigwedge x\ d.\ (x, d) \in CX \Longrightarrow x \cdot n = c \Longrightarrow (x,$
    d − (*blinfun-scaleR-left* (*f* (*x*)) o_L (*blinfun-scaleR-left* (*inverse* (*f x* · *n*)) o_L (*blinfun-inner-left n* o_L *d*)))) ∈ *PDP*
  assumes *PDP-nz*: $\bigwedge x\ d.\ (x, d) \in PDP \Longrightarrow f\,x \cdot n \neq 0$
  assumes *PDP-inR*: $\bigwedge x\ d.\ (x, d) \in PDP \Longrightarrow x \in R$
  assumes *PDP-in*: $\bigwedge x\ d.\ (x, d) \in PDP \Longrightarrow \forall_F$ *x in at x within plane n c*. $x \in R$
  obtains *X1 X2* **where** $X0 = X1 \cup X2$
    flowsto X1 {*0<..t*} ($CX \cap \{x.\ x \cdot n < c\} \times UNIV$) ($CX \cap \{x \in R.\ x \cdot n = c\} \times UNIV$)

flowsto X2 {*hl* .. *t*} (*CX* ∩ {*x*. *x* · *n* < *c*} × *UNIV*) (*Y* ∩ ({*x*. *x* · *n* < *c*} × *UNIV*))
poincare-mapsto {*x* ∈ *R*. *x* · *n* = *c*} *X1 UNIV* (*fst* ‘ *CX* ∩ {*x*. *x* · *n* < *c*}) *PDP*
proof −
let *?sB* = {*x*. *x* · *n* < *c*}
let *?A* = {*x*. *x* · *n* ≥ *c*}
let *?P* = {*x* ∈ *R*. *x* · *n* = *c*}
have [*intro*]: *closed ?A closed ?P*
by (*auto intro*!: *closed-Collect-le-within closed-levelset-within continuous-intros cR*
closed-halfspace-component-ge)
let *?CX* = *CX* ∩ *?sB* × *UNIV*
let *?X1* = {*x*∈*X0*. *flowsto* {*x*} {*0* <.. *t*} *?CX* (*CX* ∩ (*?P* × *UNIV*))}
let *?X2* = {*x*∈*X0*. *flowsto* {*x*} {*hl* .. *t*} *?CX* (*Y* ∩ (*?sB* × *UNIV*))}
have (*x*, *d*) ∈ *?X1* ∨ (*x*, *d*) ∈ *?X2* **if** (*x*, *d*) ∈ *X0* **for** *x d*
proof −
from *that assms* **have**
t: *t* ∈ *existence-ivl0 x* ⋀*s*. *0* ≤ *s* ⟹ *s* ≤ *t* ⟹ (*flow0 x s*, *Dflow x s* o_L *d*) ∈ *CX* (*flow0 x t*, *Dflow x t* o_L *d*) ∈ *Y*
apply (*auto simp*: *flowpipe-def dest*!: *bspec*[**where** *x*=*t*])
apply (*drule bspec*[**where** *x*=(*x*, *d*)], *assumption*)
apply *simp*
apply (*drule bspec*[**where** *x*=*t*], *force*)
apply *auto*
done
show *?thesis*
proof (*cases* ∀ *s*∈{*0*..*t*}. *flow0 x s* ∈ *?sB*)
case *True*
then have (*x*, *d*) ∈ *?X2* **using** *assms t* ‹(*x*, *d*) ∈ *X0*›
by (*auto simp*: *flowpipe-def flowsto-def open-segment-eq-real-ivl dest*!: *bspec*[**where** *x*=(*x*, *d*)])
then show *?thesis* **..**
next
case *False*
then obtain *s* **where** *s*: *0* ≤ *s s* ≤ *t flow0 x s* ∈ *?A*
by (*auto simp*: *not-less*)
let *?I* = *flow0 x* −‘ *?A* ∩ {*0* .. *s*}
from *s* **have** *exivlI*: *0* ≤ *s*′ ⟹ *s*′ ≤ *s* ⟹ *s*′ ∈ *existence-ivl0 x* **for** *s*′
using *ivl-subset-existence-ivl*[*OF* ‹*t* ∈ *existence-ivl0 x*›]
by *auto*
then have *compact ?I*
unfolding *compact-eq-bounded-closed*
by (*intro conjI bounded-Int bounded-closed-interval disjI2 closed-vimage-Int*)
(*auto intro*!: *continuous-intros closed-Collect-le-within cR*)
moreover
from *s* **have** *?I* ≠ {} **by** *auto*
ultimately have ∃ *s*∈*?I*. ∀ *t*∈*?I*. *s* ≤ *t*
by (*rule compact-attains-inf*)
then obtain *s*′ **where** *s*′: ⋀*s*″. *0* ≤ *s*″ ⟹ *s*″ < *s*′ ⟹ *flow0 x s*″ ∉ *?A*

flow0 x s′ ∈ ?A 0 ≤ s′ s′ ≤ s
by (*force simp*: *Ball-def*)
have *flow0 x 0 = x* **using** *local.mem-existence-ivl-iv-defined*(*2*) *t*(*1*) **by** *auto*
also have ... *∉ ?A* **using** *assms* ‹(*x*, *d*) ∈ *X0*› **by** *auto*
finally have *s′ ≠ 0* **using** *s′* **by** *auto*
then have *0 < s′* **using** ‹*s′ ≥ 0*› **by** *simp*
have *False* **if** *flow0 x s′ ∈ interior ?A*
proof −
from *that* **obtain** *e* **where** *e > 0* **and** *subset*: *ball* (*flow0 x s′*) *e ⊆ ?A*
by (*auto simp*: *mem-interior*)
from *subset* **have** ∀$_F$ *s′′ in at-left s′*. *ball* (*flow0 x s′*) *e ⊆ ?A* **by** *simp*
moreover
from *flow-continuous*[*OF exivlI*[*OF* ‹*0 ≤ s′*› ‹*s′ ≤ s*›]]
have *flow0 x −s′→ flow0 x s′* **unfolding** *isCont-def* **.**
from *tendstoD*[*OF this* ‹*0 < e*›]
have ∀$_F$ *xa in at-left s′*. *dist* (*flow0 x xa*) (*flow0 x s′*) *< e*
using *eventually-at-split* **by** *blast*
then have ∀$_F$ *s′′ in at-left s′*. *flow0 x s′′ ∈ ball* (*flow0 x s′*) *e*
by (*simp add*: *dist-commute*)
moreover
have ∀$_F$ *s′′ in at-left s′*. *0 < s′′*
using ‹*0 < s′*›
using *eventually-at-left* **by** *blast*
moreover
have ∀$_F$ *s′′ in at-left s′*. *s′′ < s′*
by (*auto simp*: *eventually-at-filter*)
ultimately
have ∀$_F$ *s′′ in at-left s′*. *False*
by *eventually-elim* (*use s′* **in** *auto*)
then show *False*
by *auto*
qed
then have *flow0 x s′ ∈ frontier ?A*
unfolding *frontier-def*
using ‹*closed ?A*› *s′*
by *auto*
with *s′* **have** (*x*, *d*) ∈ *?X1* **using** *assms that s t* ‹*0 < s′*›
ivl-subset-existence-ivl[*OF* ‹*t ∈ existence-ivl0 x*›]
frontier-subset-closed[*OF* ‹*closed ?A*›]
apply (*auto simp*: *flowsto-def flowpipe-def open-segment-eq-real-ivl frontier-halfspace-component-ge*
intro!:
dest!: *bspec*[**where** *x*=(*x*, *d*)]
intro: *exivlI*)
apply (*safe intro*!: *bexI*[**where** *x*=*s′*])
subgoal by *force*
subgoal premises *prems*
proof −
have *CX*: (*flow0 x s′*, *Dflow x s′* o_L *d*) ∈ *CX*

using *prems*
by (*auto intro*!: *prems*)
have *flow0 x s′ · n = c* **using** *prems* **by** *auto*
from *PDP-inR*[*OF PDP*[*OF CX this*]]
show *flow0 x s′ ∈ R* **.**
qed
subgoal by (*auto simp*: *not-le*)
subgoal by *force*
done
then show *?thesis* **..**
qed
qed
then have *X0 = ?X1 ∪ ?X2* **by** *auto*
moreover
have *X1*: *flowsto ?X1* {*0* <.. *t*} *?CX* (*CX* ∩ (*?P* × *UNIV*))
and *X2*: *flowsto ?X2* {*hl* .. *t*} *?CX* (*Y* ∩ (*?sB* × *UNIV*))
by (*auto simp*: *flowsto-def flowpipe-def*)
moreover
from *assms*(*2*) *X1* **have** *poincare-mapsto ?P ?X1 UNIV* (*fst* ‘ *CX* ∩ {*x*. *x* · *n* < *c*}) *PDP*
unfolding *poincare-mapsto-def flowsto-def*
apply *clarsimp*
subgoal premises *prems* **for** *x d t*
proof −
note *prems*
have *ret*: *returns-to ?P x*
apply (*rule returns-to-outsideI*[**where** *t*=*t*])
using *prems* ‹*closed ?P*›
by *auto*
moreover
have *ret-le*: *return-time ?P x* ≤ *t*
apply (*rule return-time-le*[*OF ret* - - ‹*0* < *t*›])
using *prems* ‹*closed ?P*› **by** *auto*
from *prems* **have** *CX*: (*flow0 x h′*, (*Dflow x h′*) o_L *d*) ∈ *CX* **if** *0* < *h′ h′* ≤ *t* **for** *h′*
using *that* **by** (*auto simp*: *open-segment-eq-real-ivl*)
have *PDP*: (*poincare-map ?P x*, *Dpoincare-map′ n c R x* o_L *d*) ∈ *PDP*
unfolding *poincare-map-def Dpoincare-map′-def*
unfolding *blinfun-compose-assoc blinfun-minus-comp-distrib*
apply (*rule PDP*)
using *poincare-map-returns*[*OF ret* ‹*closed ?P*›] *ret-le*
by (*auto simp*: *poincare-map-def intro*!: *CX return-time-pos ret*)
have *eventually* (*returns-to* ({*x* ∈ *R*. *x* · *n* − *c* = *0*})) (*at x*)
apply (*rule eventually-returns-to*)
using *PDP-nz*[*OF PDP*] *assms*(*2*) ‹(*x*, *d*) ∈ *X0*› *cR PDP-in*[*OF PDP*]
by (*auto intro*!: *ret derivative-eq-intros blinfun-inner-left.rep-eq*[*symmetric*] *simp*: *eventually-at-filter*)
moreover have *return-time ?P differentiable at x*
apply (*rule differentiableI*)

apply (*rule return-time-plane-has-derivative*)
using *prems ret PDP-nz*[*OF PDP*] *PDP cR PDP-in*[*OF PDP*]
by (*auto simp*: *eventually-at-filter*)
moreover
have ($\exists$ *D*. (*poincare-map ?P has-derivative blinfun-apply D*) (*at x*) $\wedge$ (*poincare-map ?P x*, *D* o_L *d*) $\in$ *PDP*)
apply (*intro exI*[**where** *x*=*Dpoincare-map′ n c R x*])
using *prems ret PDP-nz*[*OF PDP*] *PDP cR PDP-in*[*OF PDP*]
by (*auto simp*: *eventually-at-filter intro*!: *poincare-map-plane-has-derivative*)
moreover have
flow0 x h $\in$ *fst* ‘ *CX* $\wedge$ (*c* > *flow0 x h* $\cdot$ *n*)
if *0* < *h h* < *return-time ?P x* **for** *h*
using *CX*[*of h*] *ret that ret-le* ‹*0* < *h*›
apply (*auto simp*: *open-segment-eq-real-ivl intro*!: *image-eqI*[**where** *x*=(*flow0 x h*, (*Dflow x h*) o_L *d*)])
using *prems*
by (*auto simp add*: *open-segment-eq-real-ivl dest*!: *bspec*[**where** *x*=*t*])
ultimately show *?thesis*
unfolding *prems*(*7*)[*symmetric*]
by *force*
qed
done
ultimately show *?thesis* **..**
qed

lemma *poincare-map-has-derivative-step*:
assumes *Deriv*: (*poincare-map P has-derivative blinfun-apply D*) (*at* (*flow0 x0 h*))
assumes *ret*: *returns-to P x0*
assumes *cont*: *continuous* (*at x0 within S*) (*return-time P*)
assumes *less*: *0* $\leq$ *h h* < *return-time P x0*
assumes *cP*: *closed P* **and** *x0*: *x0* $\in$ *S*
shows ((λ*x*. *poincare-map P x*) *has-derivative* (*D* o_L *Dflow x0 h*)) (*at x0 within S*)
proof (*rule has-derivative-transform-eventually*)
note *return-time-tendsto* = *cont*[*unfolded continuous-within*, *rule-format*]
have *return-time P x0* $\in$ *existence-ivl0 x0*
by (*auto intro*!: *return-time-exivl cP ret*)
from *ivl-subset-existence-ivl*[*OF this*] *less*
have *hex*: *h* $\in$ *existence-ivl0 x0* **by** *auto*
from *eventually-mem-existence-ivl*[*OF this*]
have $\forall_F$ *x in at x0 within S*. *h* $\in$ *existence-ivl0 x*
by (*auto simp*: *eventually-at*)
moreover
have $\forall_F$ *x in at x0 within S*. *h* < *return-time P x*
apply (*rule order-tendstoD*)
apply (*rule return-time-tendsto*)
by (*auto intro*!: *x0 less*)
moreover have *evret*: *eventually* (*returns-to P*) (*at x0 within S*)

by (*rule eventually-returns-to-continuousI*; *fact*)
ultimately
show $\forall_F$ *x in at x0 within S*. *poincare-map P* (*flow0 x h*) = *poincare-map P x*
apply *eventually-elim*
apply (*cases h* = *0*)
subgoal by *auto*
subgoal for *x*
apply (*rule poincare-map-step-flow*)
using ‹*0* ≤ *h*› *return-time-least*[*of P x*]
by (*auto simp*: ‹*closed P*›)
done
show *poincare-map P* (*flow0 x0 h*) = *poincare-map P x0*
using *less ret x0 cP hex*
apply (*cases h* = *0*)
subgoal by *auto*
subgoal
apply (*rule poincare-map-step-flow*)
using ‹*0* ≤ *h*› *return-time-least*[*of P x0*] *ret*
by (*auto simp*: ‹*closed P*›)
done
show *x0* ∈ *S* **by** *fact*
show ((λ*x*. *poincare-map P* (*flow0 x h*)) *has-derivative blinfun-apply* (*D* o_L *Dflow x0 h*)) (*at x0 within S*)
apply (*rule has-derivative-compose*[**where** *g*=*poincare-map P* **and** *f*=λ*x*. *flow0 x h*, *OF* - *Deriv*,
THEN has-derivative-eq-rhs])
by (*auto intro*!: *derivative-eq-intros simp*: *hex flowderiv-def*)
qed

lemma *poincare-mapsto-trans*:
assumes *poincare-mapsto p1 X0 S CX P1*
assumes *poincare-mapsto p2 P1 UNIV CY P2*
assumes *CX* ∪ *CY* ∪ *fst* ‘ *P1* ⊆ *CZ*
assumes *p2* ∩ (*CX* ∪ *fst* ‘ *P1*) = {}
assumes [*intro*, *simp*]: *closed p1*
assumes [*intro*, *simp*]: *closed p2*
assumes *cont*: ⋀*x d*. (*x*, *d*) ∈ *X0* ⟹ *continuous* (*at x within S*) (*return-time p2*)
shows *poincare-mapsto p2 X0 S CZ P2*
unfolding *poincare-mapsto-def*
proof (*auto*, *goal-cases*)
fix *x0 d0* **assume** *x0*: (*x0*, *d0*) ∈ *X0*
from *assms*(*1*) *x0* **obtain** *D1 dR1* **where** *1*:
returns-to p1 x0
fst ‘ *X0* ⊆ *S*
(*return-time p1 has-derivative dR1*) (*at x0 within S*)
(*poincare-map p1 has-derivative blinfun-apply D1*) (*at x0 within S*)
(*poincare-map p1 x0*, *D1* o_L *d0*) ∈ *P1*
⋀*t*. *0* < *t* ⟹ *t* < *return-time p1 x0* ⟹ *flow0 x0 t* ∈ *CX*

by (*auto simp*: *poincare-mapsto-def differentiable-def*)
then have *crt1*: *continuous* (*at x0 within S*) (*return-time p1*)
by (*auto intro*!: *has-derivative-continuous*)
show *x0* ∈ *S*
using *1 x0* **by** *auto*
let *?x0* = *poincare-map p1 x0*
from *assms*(*2*) *x0* ‹- ∈ *P1*›
obtain *D2 dR2* **where** *2*:
returns-to p2 ?x0
(*return-time p2 has-derivative dR2*) (*at ?x0*)
(*poincare-map p2 has-derivative blinfun-apply D2*) (*at ?x0*)
(*poincare-map p2 ?x0*, *D2* o_L (*D1* o_L *d0*)) ∈ *P2*
⋀*t*. *t*∈{*0*<..<*return-time p2 ?x0*} ⟹ *flow0 ?x0 t* ∈ *CY*
by (*auto simp*: *poincare-mapsto-def differentiable-def*)

have $\forall_F$ *t in at-right 0*. *t* < *return-time p1 x0*
by (*rule order-tendstoD*) (*auto intro*!: *return-time-pos 1*)
moreover have $\forall_F$ *t in at-right 0*. *0* < *t*
by (*auto simp*: *eventually-at-filter*)
ultimately have *evnotp2*: $\forall_F$ *t in at-right 0*. *flow0 x0 t* ∉ *p2*
by *eventually-elim* (*use assms 1* **in** *auto*)
from *2*(*1*)
show *ret2*: *returns-to p2 x0*
unfolding *poincare-map-def*
by (*rule returns-to-earlierI*)
(*use evnotp2* **in** ‹*auto intro*!: *less-imp-le return-time-pos 1 return-time-exivl*›)
have *not-p2*: *0* < *t* ⟹ *t* ≤ *return-time p1 x0* ⟹ *flow0 x0 t* ∉ *p2* **for** *t*
using *1*(*5*) *1*(*6*)[*of t*] *assms*(*4*)
by (*force simp*: *poincare-map-def set-eq-iff*)
have *pm-eq*: *poincare-map p2 x0* = *poincare-map p2* (*poincare-map p1 x0*)
using *not-p2*
apply (*auto simp*: *poincare-map-def*)
apply (*subst flow-trans*[*symmetric*])
apply (*auto intro*!: *return-time-exivl 1 2*[*unfolded poincare-map-def*])
apply (*subst return-time-step*)
by (*auto simp*: *return-time-step*
intro!: *return-time-exivl 1 2*[*unfolded poincare-map-def*] *return-time-pos*)

have *evret2*: $\forall_F$ *x in at ?x0*. *returns-to p2 x*
by (*auto intro*!: *eventually-returns-to-continuousI 2 has-derivative-continuous*)

have *evret1*: $\forall_F$ *x in at x0 within S*. *returns-to p1 x*
by (*auto intro*!: *eventually-returns-to-continuousI 1 has-derivative-continuous*)
moreover
from *evret2*[*unfolded eventually-at-topological*] *2*(*1*)
obtain *U* **where** *U*: *open U poincare-map p1 x0* ∈ *U* ⋀*x*. *x* ∈ *U* ⟹ *returns-to p2 x*
by *force*
have *continuous* (*at x0 within S*) (*poincare-map p1*)

by (*rule has-derivative-continuous*) (*rule 1*)
note [*tendsto-intros*] = *this*[*unfolded continuous-within*]
have *eventually* (λ*x*. *poincare-map p1 x* ∈ *U*) (*at x0 within S*)
by (*rule topological-tendstoD*) (*auto intro*!: *tendsto-eq-intros U*)
then have *evret-flow*: ∀ $_F$ *x in at x0 within S*. *returns-to p2* (*flow0 x* (*return-time p1 x*))
unfolding *poincare-map-def*[*symmetric*]
apply *eventually-elim*
apply (*rule U*)
apply *auto*
done
moreover
have *h-less-rt*: *return-time p1 x0* < *return-time p2 x0*
by (*rule return-time-gt*; *fact*)
then have *0* < *return-time p2 x0* − *return-time p1 x0*
by (*simp*)
from - *this* **have** ∀ $_F$ *x in at x0 within S*. *0* < *return-time p2 x* − *return-time p1 x*
apply (*rule order-tendstoD*)
using *cont* ‹(*x0*, -) ∈ -›
by (*auto intro*!: *tendsto-eq-intros crt1 simp*: *continuous-within*[*symmetric*] *continuous-on-def*)
then have *evpm2*: ∀ $_F$ *x in at x0 within S*. ∀ *s*. *0* < *s* ⟶ *s* ≤ *return-time p1 x* ⟶ *flow0 x s* ∉ *p2*
apply *eventually-elim*
apply *safe*
subgoal for *x s*
using *return-time-least*[*of p2 x s*]
by (*auto simp add*: *return-time-pos-returns-to*)
done
ultimately
have *pm-eq-at*: ∀ $_F$ *x in at x0 within S*.
poincare-map p2 (*poincare-map p1 x*) = *poincare-map p2 x*
apply (*eventually-elim*)
apply (*auto simp*: *poincare-map-def*)
apply (*subst flow-trans*[*symmetric*])
apply (*auto intro*!: *return-time-exivl*)
apply (*subst return-time-step*)
by (*auto simp*: *return-time-step*
intro!: *return-time-exivl return-time-pos*)
from - *this* **have** (*poincare-map p2 has-derivative blinfun-apply* (*D2* o_L *D1*)) (*at x0 within S*)
apply (*rule has-derivative-transform-eventually*)
apply (*rule has-derivative-compose*[*OF 1*(*4*) *2*(*3*), *THEN has-derivative-eq-rhs*])
by (*auto simp*: ‹*x0* ∈ *S*› *pm-eq*)
moreover have (*poincare-map p2 x0*, (*D2* o_L *D1*) o_L *d0*) ∈ *P2*
using *2*(*4*) **unfolding** *pm-eq blinfun-compose-assoc* **.**
ultimately
show ∃ *D*. (*poincare-map p2 has-derivative blinfun-apply D*) (*at x0 within S*) ∧

$(poincare\text{-}map\ p2\ x0,\ D\ o_L\ d0) \in P2$
by *auto*
show $0 < t \Longrightarrow t < return\text{-}time\ p2\ x0 \Longrightarrow flow0\ x0\ t \in CZ$ **for** *t*
apply (*cases t < return-time p1 x0*)
subgoal
apply (*drule 1*)
using *assms*
by *auto*
subgoal
apply (*cases t = return-time p1 x0*)
subgoal using *1(5) assms* **by** (*auto simp*: *poincare-map-def*)
subgoal premises *prems*
proof −
have *flow0 x0 t = flow0 ?x0 (t − return-time p1 x0)*
unfolding *poincare-map-def*
apply (*subst flow-trans*[*symmetric*])
using *prems*
by (*auto simp*:
intro!: *return-time-exivl 1 diff-existence-ivl-trans*
less-return-time-imp-exivl[*OF - ret2*])
also have ... ∈ *CY*
apply (*rule 2*)
using *prems*
apply *auto*
using *1(1) 2(1) assms poincare-map-def ret2 return-time-exivl*
return-time-least return-time-pos return-time-step
by *auto*
also have ... ⊆ *CZ* **using** *assms* **by** *auto*
finally show *flow0 x0 t* ∈ *CZ*
by *simp*
qed
done
done
have *rt-eq*: *return-time p2 (poincare-map p1 x0) + return-time p1 x0 = return-time p2 x0*
apply (*auto simp*: *poincare-map-def*)
apply (*subst return-time-step*)
by (*auto simp*: *return-time-step poincare-map-def*[*symmetric*] *not-p2*
intro!: *return-time-exivl return-time-pos 1 2*)
have *evrt-eq*: $\forall_F$ *x in at x0 within S*.
return-time p2 (poincare-map p1 x) + return-time p1 x = return-time p2 x
using *evret-flow evret1 evpm2*
apply (*eventually-elim*)
apply (*auto simp*: *poincare-map-def*)
apply (*subst return-time-step*)
by (*auto simp*: *return-time-step*
intro!: *return-time-exivl return-time-pos*)
from - *evrt-eq*
have (*return-time p2 has-derivative* (λ*x*. *dR2* (*blinfun-apply D1 x*) + *dR1 x*))

(*at x0 within S*)
by (*rule has-derivative-transform-eventually*)
(*auto intro*!: *derivative-eq-intros has-derivative-compose*[*OF 1*(*4*) *2*(*2*)] *1*(*3*) ‹*x0* ∈ *S*›
simp: *rt-eq*)
then show *return-time p2 differentiable at x0 within S* **by** (*auto intro*!: *differentiableI*)
qed

lemma *flowsto-poincare-trans*:— TODO: the proof is close to ⟦*poincare-mapsto ?p1.0 ?X0.0 ?S ?CX ?P1.0*; *poincare-mapsto ?p2.0 ?P1.0 UNIV ?CY ?P2.0*; *?CX* ∪ *?CY* ∪ *fst* ‘ *?P1.0* ⊆ *?CZ*; *?p2.0* ∩ (*?CX* ∪ *fst* ‘ *?P1.0*) = {}; *closed ?p1.0*; *closed ?p2.0*; ⋀*x d*. (*x*, *d*) ∈ *?X0.0* ⟹ *continuous* (*at x within ?S*) (*return-time ?p2.0*)⟧ ⟹ *poincare-mapsto ?p2.0 ?X0.0 ?S ?CZ ?P2.0*
assumes *f*: *flowsto X0 T CX P1*
assumes *poincare-mapsto p2 P1 UNIV CY P2*
assumes *nn*: ⋀*t*. *t* ∈ *T* ⟹ *t* ≥ *0*
assumes *fst* ‘ *CX* ∪ *CY* ∪ *fst* ‘ *P1* ⊆ *CZ*
assumes *p2* ∩ (*fst* ‘ *CX* ∪ *fst* ‘ *P1*) = {}
assumes [*intro*, *simp*]: *closed p2*
assumes *cont*: ⋀*x d*. (*x*, *d*) ∈ *X0* ⟹ *continuous* (*at x within S*) (*return-time p2*)
assumes *subset*: *fst* ‘ *X0* ⊆ *S*
shows *poincare-mapsto p2 X0 S CZ P2*
unfolding *poincare-mapsto-def*
proof (*auto*, *goal-cases*)
fix *x0 d0* **assume** *x0*: (*x0*, *d0*) ∈ *X0*
from *flowstoE*[*OF f x0*] **obtain** *h* **where** *1*:
h ∈ *T h* ∈ *existence-ivl0 x0*
(*flow0 x0 h*, *Dflow x0 h* o_L *d0*) ∈ *P1* (**is** (*?x0*, -) ∈ -)
(⋀*h'*. *h'* ∈ {*0*<−−<*h*} ⟹ (*flow0 x0 h'*, *Dflow x0 h'* o_L *d0*) ∈ *CX*)
by *auto*
then have *CX*: (⋀*h'*. *0* < *h'* ⟹ *h'* < *h* ⟹ (*flow0 x0 h'*, *Dflow x0 h'* o_L *d0*) ∈ *CX*)
by (*auto simp*: *nn open-segment-eq-real-ivl*)
from *1* **have** *0* ≤ *h* **by** (*auto simp*: *nn*)
from *assms* **have** *CX-p2D*: *x* ∈ *CX* ⟹ *fst x* ∉ *p2* **for** *x* **by** *auto*
from *assms* **have** *P1-p2D*: *x* ∈ *P1* ⟹ *fst x* ∉ *p2* **for** *x* **by** *auto*
show *x0* ∈ *S*
using *x0 1 subset* **by** *auto*
let *?D1* = *Dflow x0 h*
from *assms*(*2*) *x0* ‹- ∈ *P1*›
obtain *D2 dR2* **where** *2*:
returns-to p2 ?x0
(*return-time p2 has-derivative dR2*) (*at ?x0*)
(*poincare-map p2 has-derivative blinfun-apply D2*) (*at ?x0*)
(*poincare-map p2 ?x0*, *D2* o_L (*?D1* o_L *d0*)) ∈ *P2*
⋀*t*. *t*∈{*0*<..<*return-time p2 ?x0*} ⟹ *flow0 ?x0 t* ∈ *CY*
by (*auto simp*: *poincare-mapsto-def differentiable-def*)

{
  assume *pos*: *h* > *0*
  have ∀ *F* *t in at-right 0*. *t* < *h*
    by (*rule order-tendstoD*) (*auto intro*!: *return-time-pos 1 pos*)
  moreover have ∀ *F* *t in at-right 0*. *0* < *t*
    by (*auto simp*: *eventually-at-filter*)
  ultimately have ∀ *F* *t in at-right 0*. *flow0 x0 t* ∉ *p2*
    by *eventually-elim* (*use assms* **in** ‹*force dest*: *CX CX-p2D*›)
} **note** *evnotp2* = *this*
from *2*(*1*)
show *ret2*: *returns-to p2 x0*
  apply (*cases h* = *0*)
  subgoal using *1* **by** *auto*
  unfolding *poincare-map-def*
  by (*rule returns-to-earlierI*)
      (*use evnotp2* ‹*0* ≤ *h*› **in** ‹*auto intro*!: *less-imp-le return-time-pos 1 return-time-exivl* ›)
have *not-p2*: *0* < *t* ⟹ *t* ≤ *h* ⟹ *flow0 x0 t* ∉ *p2* **for** *t*
  using *1*(*1*−*3*) *CX*[*of t*] *assms*(*4*) *CX-p2D P1-p2D*
  by (*cases h* = *t*) (*auto simp*: *poincare-map-def set-eq-iff subset-iff*)
have *pm-eq*: *poincare-map p2 x0* = *poincare-map p2 ?x0*
  apply (*cases h* = *0*, *use 1* **in** *force*)
  using *not-p2* ‹*0* ≤ *h*›
  apply (*auto simp*: *poincare-map-def*)
  apply (*subst flow-trans*[*symmetric*])
    apply (*auto intro*!: *return-time-exivl 1 2*[*unfolded poincare-map-def*])
  apply (*subst return-time-step*)
  by (*auto simp*: *return-time-step*
      intro!: *return-time-exivl 1 2*[*unfolded poincare-map-def*] *return-time-pos*)

have *evret2*: ∀ *F* *x in at ?x0*. *returns-to p2 x*
  by (*auto intro*!: *eventually-returns-to-continuousI 2 has-derivative-continuous*)

have ∀ *F* *x in at x0*. *h* ∈ *existence-ivl0 x*
  by (*simp add*: *1 eventually-mem-existence-ivl*)
then have *evex*: ∀ *F* *x in at x0 within S*. *h* ∈ *existence-ivl0 x*
  by (*auto simp*: *eventually-at*)
moreover
from *evret2*[*unfolded eventually-at-topological*] *2*(*1*)
obtain *U* **where** *U*: *open U flow0 x0 h* ∈ *U* ⋀*x*. *x* ∈ *U* ⟹ *returns-to p2 x*
  by *force*
note [*tendsto-intros*] = *this*[*unfolded continuous-within*]
have *eventually* (λ*x*. *flow0 x h* ∈ *U*) (*at x0 within S*)
  by (*rule topological-tendstoD*) (*auto intro*!: *tendsto-eq-intros U 1*)
then have *evret-flow*: ∀ *F* *x in at x0 within S*. *returns-to p2* (*flow0 x h*)
  unfolding *poincare-map-def*[*symmetric*]
  apply *eventually-elim*
  apply (*rule U*)

apply *auto*
done
moreover
have *h-less-rt*: $h < return\text{-}time\ p2\ x0$
by (*rule return-time-gt*; *fact*)
then have $0 < return\text{-}time\ p2\ x0 - h$
by (*simp*)
from - *this* **have** $\forall_F\ x\ in\ at\ x0\ within\ S.\ 0 < return\text{-}time\ p2\ x - h$
apply (*rule order-tendstoD*)
using *cont* ‹(*x0*, -) ∈ -›
by (*auto intro*!: *tendsto-eq-intros simp*: *continuous-within*[*symmetric*] *continuous-on-def*)
then have *evpm2*: $\forall_F\ x\ in\ at\ x0\ within\ S.\ \forall s.\ 0 < s \longrightarrow s \le h \longrightarrow flow0\ x\ s \notin p2$
apply *eventually-elim*
apply *safe*
subgoal for *x s*
using *return-time-least*[*of p2 x s*]
by (*auto simp add*: *return-time-pos-returns-to*)
done
ultimately
have *pm-eq-at*: $\forall_F\ x\ in\ at\ x0\ within\ S.$
poincare-map p2 (*flow0 x h*) = *poincare-map p2 x*
apply (*eventually-elim*)
apply (*cases* $h = 0$) **subgoal by** *auto*
apply (*auto simp*: *poincare-map-def*)
apply (*subst flow-trans*[*symmetric*])
apply (*auto intro*!: *return-time-exivl*)
apply (*subst return-time-step*)
using ‹$0 \le h$›
by (*auto simp*: *return-time-step intro*!: *return-time-exivl return-time-pos*)
from - *this* **have** (*poincare-map p2 has-derivative blinfun-apply* ($D2\ o_L\ ?D1$)) (*at x0 within S*)
apply (*rule has-derivative-transform-eventually*)
apply (*rule has-derivative-at-withinI*)
apply (*rule has-derivative-compose*[*OF flow-has-space-derivative 2*(*3*), *THEN has-derivative-eq-rhs*])
by (*auto simp*: ‹$x0 \in S$› *pm-eq 1*)
moreover have (*poincare-map p2 x0*, ($D2\ o_L\ ?D1$) $o_L\ d0$) ∈ *P2*
using *2*(*4*) **unfolding** *pm-eq blinfun-compose-assoc* **.**
ultimately
show ∃ *D*. (*poincare-map p2 has-derivative blinfun-apply D*) (*at x0 within S*) ∧
(*poincare-map p2 x0*, $D\ o_L\ d0$) ∈ *P2*
by *auto*
show $0 < t \Longrightarrow t < return\text{-}time\ p2\ x0 \Longrightarrow flow0\ x0\ t \in CZ$ **for** *t*
apply (*cases* $t < h$)
subgoal
apply (*drule CX*)
using *assms*

```
    by auto
  subgoal
    apply (cases t = h)
    subgoal using 1 assms by (auto simp: poincare-map-def)
    subgoal premises prems
    proof -
      have flow0 x0 t = flow0 ?x0 (t - h)
        unfolding poincare-map-def
        apply (subst flow-trans[symmetric])
        using prems
        by (auto simp:
            intro!: return-time-exivl 1 diff-existence-ivl-trans
            less-return-time-imp-exivl[OF - ret2])
      also have ... ∈ CY
        apply (cases h = 0)
        subgoal using 1(2) 2(5) prems(1) prems(2) by auto
        subgoal
          apply (rule 2)
          using prems
          apply auto
          apply (subst return-time-step)
               apply (rule returns-to-laterI)
          using ret2 ‹0 ≤ h› ‹h ∈ existence-ivl0 x0› not-p2
          by auto
        done
        also have ... ⊆ CZ using assms by auto
      finally show flow0 x0 t ∈ CZ
        by simp
    qed
    done
  done
have rt-eq: return-time p2 ?x0 + h = return-time p2 x0
  apply (cases h = 0)
  subgoal using 1 by auto
  subgoal
    apply (subst return-time-step)
    using ‹0 ≤ h›
    by (auto simp: return-time-step poincare-map-def[symmetric] not-p2
        intro!: return-time-exivl return-time-pos 1 2)
  done
have evrt-eq: ∀F x in at x0 within S.
  return-time p2 (flow0 x h) + h = return-time p2 x
  using evret-flow evpm2 evex
  apply (eventually-elim)
  apply (cases h = 0)
  subgoal using 1 by auto
  subgoal
    apply (subst return-time-step)
    using ‹0 ≤ h›
```

by (*auto simp*: *return-time-step*
intro!: *return-time-exivl return-time-pos*)
done
from - *evrt-eq*
have (*return-time p2 has-derivative* (*λx. dR2* (*blinfun-apply ?D1 x*))) (*at x0 within S*)
apply (*rule has-derivative-transform-eventually*)
apply (*rule has-derivative-at-withinI*)
by (*auto intro*!: *derivative-eq-intros has-derivative-compose*[*OF flow-has-space-derivative 2*(*2*)] *1* ‹*x0* ∈ *S*›
simp: *rt-eq*)
then show *return-time p2 differentiable at x0 within S* **by** (*auto intro*!: *differentiableI*)
qed

6.5 conditions for continuous return time

definition *section s Ds S* ⟷
(∀ *x*. (*s has-derivative blinfun-apply* (*Ds x*)) (*at x*)) ∧
(∀ *x*. *isCont Ds x*) ∧
(∀ *x* ∈ *S*. *s x* = (*0*::*real*) ⟶ *Ds x* (*f x*) ≠ *0*) ∧
closed S ∧ *S* ⊆ *X*

lemma *sectionD*:
assumes *section s Ds S*
shows (*s has-derivative blinfun-apply* (*Ds x*)) (*at x*)
isCont Ds x
x ∈ *S* ⟹ *s x* = *0* ⟹ *Ds x* (*f x*) ≠ *0*
closed S S ⊆ *X*
using *assms* **by** (*auto simp*: *section-def*)

definition *transversal p* ⟷ (∀ *x* ∈ *p*. $\forall_F$ *t in at-right 0*. *flow0 x t* ∉ *p*)

lemma *transversalD*: *transversal p* ⟹ *x* ∈ *p* ⟹ $\forall_F$ *t in at-right 0*. *flow0 x t* ∉ *p*
by (*auto simp*: *transversal-def*)

lemma *transversal-section*:
fixes *c*::*real*
assumes *section s Ds S*
shows *transversal* {*x* ∈ *S*. *s x* = *0*}
using *assms*
unfolding *section-def transversal-def*
proof (*safe*, *goal-cases*)
case (*1 x*)
then have *x* ∈ *X* **by** *auto*
have $\forall_F$ *t in at-right 0*. *flow0 x t* ∉ {*xa* ∈ *S*. *s xa* = *0*}
by (*rule flow-avoids-surface-eventually-at-right*)
(*rule disjI2 assms 1*[*rule-format*] *refl* ‹*x* ∈ *X*›)+

then show *?case*
by *simp*
qed

lemma *section-closed*[*intro*, *simp*]: *section s Ds S* $\Longrightarrow$ *closed* $\{x \in S.\ s\ x = 0\}$
by (*auto intro*!: *closed-levelset-within simp*: *section-def*
intro!: *has-derivative-continuous-on has-derivative-at-withinI*[**where** *s*=*S*])

lemma *return-time-continuous-belowI*:
assumes *ft*: *flowsto X0 T CX X1*
assumes *pos*: $\bigwedge t.\ t \in T \Longrightarrow t > 0$
assumes *X0*: *fst* ‘ *X0* $\subseteq \{x \in S.\ s\ x = 0\}$
assumes *CX*: *fst* ‘ *CX* $\cap \{x \in S.\ s\ x = 0\} = \{\}$
assumes *X1*: *fst* ‘ *X1* $\subseteq \{x \in S.\ s\ x = 0\}$
assumes *sec*: *section s Ds S*
assumes *nz*: $\bigwedge x.\ x \in S \Longrightarrow s\ x = 0 \Longrightarrow Ds\ x\ (f\ x) \neq 0$
assumes *Dneg*: $(\lambda x.\ (Ds\ x)\ (f\ x))$ ‘ *fst* ‘ *X0* $\subseteq \{..<0\}$
assumes *rel-int*: $\bigwedge x.\ x \in$ *fst* ‘ *X1* $\Longrightarrow \forall_F$ *x in at x*. $s\ x = 0 \longrightarrow x \in S$
assumes $(x,\ d) \in X0$
shows *continuous* (*at x within* $\{x.\ s\ x \leq 0\}$) (*return-time* $\{x \in S.\ s\ x = 0\}$)
proof (*rule return-time-continuous-below*)
from *assms* **have** $x \in S$ $s\ x = 0$ $x \in \{x \in S.\ s\ x = 0\}$ **by** *auto*
note *cs* = *section-closed*[*OF sec*]
note *sectionD*[*OF sec*]
from *flowstoE*[*OF ft* ‹$(x,\ d) \in X0$›] **obtain** *h*
where *h*: $h \in T$
$h \in$ *existence-ivl0 x*
(*flow0 x h*, *Dflow x h* o_L *d*) $\in$ *X1*
($\bigwedge h'.\ h' \in \{0<--<h\} \Longrightarrow$ (*flow0 x h'*, *Dflow x h'* o_L *d*) $\in$ *CX*)
by *blast*
show *ret*: *returns-to* $\{x \in S.\ s\ x = 0\}$ *x*
apply (*rule returns-toI*)
apply (*rule pos*)
apply (*rule h*)
subgoal by (*rule h*)
subgoal using *h*(*3*) *X1* **by** *auto*
subgoal apply (*intro transversalD*) **apply** (*rule transversal-section*) **apply** (*rule sec*)
apply *fact*
done
subgoal by *fact*
done
show (*s has-derivative blinfun-apply* (*Ds x*)) (*at x*) **for** *x* **by** *fact*
show *closed S* **by** *fact*
show *isCont Ds x* **for** *x* **by** *fact*
show $x \in S$ $s\ x = 0$ **by** *fact*+
let *?p* = *poincare-map* $\{x \in S.\ s\ x = 0\}$ *x*
have *?p* $\in \{x \in S.\ s\ x = 0\}$ **using** *poincare-map-returns*[*OF ret cs*] **.**

with *nz* **show** *Ds ?p* (*f ?p*) ≠ *0* **by** *auto*
from *Dneg* ‹(*x*, -) ∈ *X0*› **show** *Ds x* (*f x*) < *0* **by** *force*
from ‹- ∈ *X1*› *X1 CX h*
have *return-time* {*x* ∈ *S*. *s x* = *0*} *x* = *h*
by (*fastforce intro*!: *return-time-eqI cs pos h simp*: *open-segment-eq-real-ivl*)
then have *?p* ∈ *fst* ‘ *X1*
using ‹- ∈ *X1*› **by** (*force simp*: *poincare-map-def*)
from *rel-int*[*OF this*] **show** $\forall_F$ *x in at* (*poincare-map* {*x* ∈ *S*. *s x* = *0*} *x*). *s x* = *0* ⟶ *x* ∈ *S*
by *auto*
qed

end

end
theory *Flow-Congs*
imports *Reachability-Analysis*
begin

lemma *lipschitz-on-congI*:
assumes *L*′−*lipschitz-on s*′ *g*′
assumes *s*′ = *s*
assumes *L*′ ≤ *L*
assumes ⋀*x y*. *x* ∈ *s* ⟹ *g*′ *x* = *g x*
shows *L*−*lipschitz-on s g*
using *assms*
by (*auto simp*: *lipschitz-on-def intro*!: *order-trans*[*OF* - *mult-right-mono*[*OF* ‹*L*′ ≤ *L*›]])

lemma *local-lipschitz-congI*:
assumes *local-lipschitz s*′ *t*′ *g*′
assumes *s*′ = *s*
assumes *t*′ = *t*
assumes ⋀*x y*. *x* ∈ *s* ⟹ *y* ∈ *t* ⟹ *g*′ *x y* = *g x y*
shows *local-lipschitz s t g*
proof −
from *assms* **have** *local-lipschitz s t g*′
by (*auto simp*: *local-lipschitz-def*)
then show *?thesis*
apply (*auto simp*: *local-lipschitz-def*)
apply (*drule-tac bspec*, *assumption*)
apply (*drule-tac bspec*, *assumption*)
apply *auto*
subgoal for *x y u L*
apply (*rule exI*[**where** *x*=*u*])
apply (*auto intro*!: *exI*[**where** *x*=*L*])
apply (*drule bspec*)
apply *simp*
apply (*rule lipschitz-on-congI*, *assumption*, *rule refl*, *rule order-refl*)

using *assms*
apply (*auto*)
done
done
qed

context *ll-on-open-it*— TODO: do this more generically for *ll-on-open-it*
begin

context fixes *S Y g* **assumes** *cong*: $X = Y$ $T = S$ $\bigwedge x\ t.\ x \in Y \Longrightarrow t \in S \Longrightarrow f\ t\ x = g\ t\ x$
begin

lemma *ll-on-open-congI*: *ll-on-open S g Y*
proof −
interpret *Y*: *ll-on-open-it S f Y t0*
apply (*subst cong*(*1*)[*symmetric*])
apply (*subst cong*(*2*)[*symmetric*])
by *unfold-locales*
show *?thesis*
apply *standard*
subgoal
using *local-lipschitz*
apply (*rule local-lipschitz-congI*)
using *cong* **by** *simp-all*
subgoal apply (*subst continuous-on-cong*) **prefer** *3* **apply** (*rule cont*)
using *cong* **by** (*auto*)
subgoal using *open-domain* **by** (*auto simp*: *cong*)
subgoal using *open-domain* **by** (*auto simp*: *cong*)
done
qed

lemma *existence-ivl-subsetI*:
assumes *t*: $t \in$ *existence-ivl t0 x0*
shows $t \in$ *ll-on-open.existence-ivl S g Y t0 x0*
proof −
from *assms* **have** ‹$t0 \in T$› $x0 \in X$
by (*rule mem-existence-ivl-iv-defined*)+
interpret *Y*: *ll-on-open S g Y* **by** (*rule ll-on-open-congI*)
have (*flow t0 x0 solves-ode f*) (*existence-ivl t0 x0*) *X*
by (*rule flow-solves-ode*) (*auto simp*: ‹$x0 \in X$› ‹$t0 \in T$›)
then have (*flow t0 x0 solves-ode f*) $\{t0 {-}{-} t\}$ *X*
by (*rule solves-ode-on-subset*)
(*auto simp add*: *t local.closed-segment-subset-existence-ivl*)
then have (*flow t0 x0 solves-ode g*) $\{t0 {-}{-} t\}$ *Y*
apply (*rule solves-ode-congI*)
apply (*auto intro*!: *assms cong*)
using ‹(*flow t0 x0 solves-ode f*) $\{t0 {-}{-} t\}$ *X*› *local.cong*(*1*) *solves-ode-domainD*
apply *blast*

```
   using ‹t0 ∈ T› assms closed-segment-subset-domainI general.mem-existence-ivl-subset
local.cong(2)
    by blast
  then show ?thesis
    apply (rule Y.existence-ivl-maximal-segment)
    subgoal by (simp add: ‹t0 ∈ T› ‹x0 ∈ X›)
    apply (subst cong[symmetric])
   using ‹t0 ∈ T› assms closed-segment-subset-domainI general.mem-existence-ivl-subset
local.cong(2)
    by blast
qed

lemma existence-ivl-cong:
  shows existence-ivl t0 x0 = ll-on-open.existence-ivl S g Y t0 x0
proof −
  interpret Y: ll-on-open S g Y by (rule ll-on-open-congI)
  show ?thesis
    apply (auto )
    subgoal by (rule existence-ivl-subsetI)
    subgoal
      apply (rule Y.existence-ivl-subsetI)
      using cong
      by auto
    done
qed

lemma flow-cong:
  assumes t ∈ existence-ivl t0 x0
  shows flow t0 x0 t = ll-on-open.flow S g Y t0 x0 t
proof −
  interpret Y: ll-on-open S g Y by (rule ll-on-open-congI)
  from assms have t0 ∈ T x0 ∈ X
    by (rule mem-existence-ivl-iv-defined)+
  from cong ‹x0 ∈ X› have x0 ∈ Y by auto
  from cong ‹t0 ∈ T› have t0 ∈ S by auto
  show ?thesis
    apply (rule Y.equals-flowI[where T'=existence-ivl t0 x0])
    subgoal using ‹t0 ∈ T› ‹x0 ∈ X›  by auto
    subgoal using ‹x0 ∈ X› by auto
    subgoal by (auto simp: existence-ivl-cong ‹x0 ∈ X›)
    subgoal
      apply (rule solves-ode-congI)
          apply (rule flow-solves-ode[OF ‹t0 ∈ T› ‹x0 ∈ X›])
      using existence-ivl-subset[of x0]
     by (auto simp: cong(2)[symmetric] cong(1)[symmetric] assms flow-in-domain
intro!: cong)
    subgoal using ‹t0 ∈ S› ‹t0 ∈ T› ‹x0 ∈ X› ‹x0 ∈ Y›
      by (auto simp:)
    subgoal by fact
```

done
qed

end

end

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

context fixes *Y g* **assumes** *cong*: $X = Y \bigwedge x\ t.\ x \in Y \Longrightarrow f\ x = g\ x$
begin

lemma *auto-ll-on-open-congI*: *auto-ll-on-open g Y*
apply *unfold-locales*
subgoal
using *local-lipschitz*
apply (*rule local-lipschitz-congI*)
using *cong* **by** *auto*
subgoal
using *open-domain*
using *cong* **by** *auto*
done

lemma *existence-ivl0-cong*:
shows *existence-ivl0 x0* = *auto-ll-on-open.existence-ivl0 g Y x0*
proof −
interpret *Y*: *auto-ll-on-open g Y* **by** (*rule auto-ll-on-open-congI*)
show *?thesis*
unfolding *Y.existence-ivl0-def*
apply (*rule existence-ivl-cong*)
using *cong* **by** *auto*
qed

lemma *flow0-cong*:
assumes $t \in$ *existence-ivl0 x0*
shows *flow0 x0 t* = *auto-ll-on-open.flow0 g Y x0 t*
proof −
interpret *Y*: *auto-ll-on-open g Y* **by** (*rule auto-ll-on-open-congI*)
show *?thesis*
unfolding *Y.flow0-def*
apply (*rule flow-cong*)
using *cong assms* **by** *auto*
qed

end

end

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

context fixes *Y g* **assumes** *cong*: $X = Y \bigwedge x\ t.\ x \in Y \Longrightarrow f\ x = g\ x$
begin

lemma *f′-cong*: (*g has-derivative blinfun-apply* (*f′ x*)) (*at x*) **if** $x \in Y$
proof −
  from *derivative-rhs*[*of x*] *that cong*
  have (*f has-derivative blinfun-apply* (*f′ x*)) (*at x within Y*)
    by (*auto intro*!: *has-derivative-at-withinI*)
  then have (*g has-derivative blinfun-apply* (*f′ x*)) (*at x within Y*)
    by (*rule has-derivative-transform-within*[*OF - zero-less-one that*])
      (*auto simp*: *cong*)
  then show *?thesis*
    using *at-within-open*[*OF that*] *cong open-dom*
    by *auto*
qed

lemma *c1-on-open-euclidean-congI*: *c1-on-open-euclidean g f′ Y*
proof −
  interpret *Y*: *c1-on-open-euclidean f f′ Y* **unfolding** *cong*[*symmetric*] **by** *unfold-locales*
  show *?thesis*
    apply *standard*
    subgoal using *cong* **by** *simp*
    subgoal by (*rule f′-cong*)
    subgoal by (*simp add*: *cong*[*symmetric*] *continuous-derivative*)
    done
qed

lemma *vareq-cong*: *vareq x0 t* = *c1-on-open-euclidean.vareq g f′ Y x0 t*
  if $t \in$ *existence-ivl0 x0*
proof −
  interpret *Y*: *c1-on-open-euclidean g f′ Y* **by** (*rule c1-on-open-euclidean-congI*)
  show *?thesis*
    unfolding *vareq-def Y.vareq-def*
    apply (*rule arg-cong*[**where** *f*=*f′*])
    apply (*rule flow0-cong*)
    using *cong that* **by** *auto*
qed

lemma *Dflow-cong*:
  assumes $t \in$ *existence-ivl0 x0*
  shows *Dflow x0 t* = *c1-on-open-euclidean.Dflow g f′ Y x0 t*
proof −
  interpret *Y*: *c1-on-open-euclidean g f′ Y* **by** (*rule c1-on-open-euclidean-congI*)
  from *assms* **have** *x0* $\in X$
    by (*rule mem-existence-ivl-iv-defined*)
  from *cong* ‹*x0* $\in X$› **have** *x0* $\in Y$ **by** *auto*

```
  show ?thesis
    unfolding Dflow-def Y.Dflow-def
    apply (rule mvar.equals-flowI[symmetric, OF - - order-refl])
    subgoal using ‹x0 ∈ X› by auto
    subgoal using ‹x0 ∈ X› by auto
    subgoal
      apply (rule solves-ode-congI)
          apply (rule Y.mvar.flow-solves-ode)
           prefer 3 apply (rule refl)
      subgoal using ‹x0 ∈ X› ‹x0 ∈ Y› by auto
      subgoal using ‹x0 ∈ X› ‹x0 ∈ Y› by auto
      subgoal for t
        apply (subst vareq-cong)
         apply (subst (asm) Y.mvar-existence-ivl-eq-existence-ivl)
        subgoal using ‹x0 ∈ Y› by simp
        subgoal
          using cong
          by (subst (asm) existence-ivl0-cong[symmetric]) auto
        subgoal using ‹x0 ∈ Y› by simp
        done
      subgoal using ‹x0 ∈ X› ‹x0 ∈ Y›
        apply (subst mvar-existence-ivl-eq-existence-ivl)
        subgoal by simp
        apply (subst Y.mvar-existence-ivl-eq-existence-ivl)
        subgoal by simp
        using cong
        by (subst existence-ivl0-cong[symmetric]) auto
      subgoal by simp
      done
    subgoal using ‹x0 ∈ X› ‹x0 ∈ Y› by auto
    subgoal
      apply (subst mvar-existence-ivl-eq-existence-ivl)
       apply auto
       apply fact+
      done
    done
qed

lemma flowsto-congI1:
  assumes flowsto A B C D
  shows c1-on-open-euclidean.flowsto g f' Y A B C D
proof −
  interpret Y: c1-on-open-euclidean g f' Y by (rule c1-on-open-euclidean-congI)
  show ?thesis
    using assms
    unfolding flowsto-def Y.flowsto-def
    apply (auto simp: existence-ivl0-cong[OF cong]  flow0-cong[OF cong])
      apply (drule bspec, assumption)
    apply clarsimp
```

apply (*rule bexI*)
apply (*rule conjI*)
apply *assumption*
apply (*subst flow0-cong*[*symmetric*, *OF cong*])
apply *auto*
apply (*subst existence-ivl0-cong*[*OF cong*])
apply *auto*
apply (*subst Dflow-cong*[*symmetric*])
apply *auto*
apply (*subst existence-ivl0-cong*[*OF cong*])
apply *auto*
apply (*drule bspec*, *assumption*)
apply (*subst flow0-cong*[*symmetric*, *OF cong*])
apply *auto*
apply (*subst existence-ivl0-cong*[*OF cong*])
apply *auto* **defer**
apply (*subst Dflow-cong*[*symmetric*])
apply *auto*
apply (*subst existence-ivl0-cong*[*OF cong*])
apply *auto*
apply (*drule Y.closed-segment-subset-existence-ivl*;
auto simp: *open-segment-eq-real-ivl closed-segment-eq-real-ivl split*: *if-splits*)+
done
qed

lemma *flowsto-congI2*:
assumes *c1-on-open-euclidean.flowsto g f′ Y A B C D*
shows *flowsto A B C D*
proof −
interpret *Y*: *c1-on-open-euclidean g f′ Y* **by** (*rule c1-on-open-euclidean-congI*)
show *?thesis*
apply (*rule Y.flowsto-congI1*)
using *assms*
by (*auto simp*: *cong*)
qed

lemma *flowsto-congI*: *flowsto A B C D* = *c1-on-open-euclidean.flowsto g f′ Y A B C D*
using *flowsto-congI1*[*of A B C D*] *flowsto-congI2*[*of A B C D*] **by** *auto*

lemma
returns-to-congI1:
assumes *returns-to A x*
shows *auto-ll-on-open.returns-to g Y A x*
proof −
interpret *Y*: *c1-on-open-euclidean g f′ Y* **by** (*rule c1-on-open-euclidean-congI*)
from *assms* **obtain** *t* **where** *t*:
$\forall_F$ *t in at-right 0. flow0 x t* ∉ *A*
0 < *t t* ∈ *existence-ivl0 x flow0 x t* ∈ *A*

by (*auto simp*: *returns-to-def*)

note *t(1)*
moreover
have ∀ $_F$ *s in at-right 0*. *s* < *t*
  using *tendsto-ident-at* ‹*0* < *t*›
  by (*rule order-tendstoD*)
moreover have ∀ $_F$ *s in at-right 0*. *0* < *s*
  by (*auto simp*: *eventually-at-topological*)
ultimately have ∀ $_F$ *t in at-right 0*. *Y*.*flow0 x t* ∉ *A*
  apply *eventually-elim*
  using *ivl-subset-existence-ivl*[*OF* ‹*t* ∈ -›]
  apply (*subst* (*asm*) *flow0-cong*[*OF cong*])
  by *auto*

moreover have ∃ *t*>*0*. *t* ∈ *Y*.*existence-ivl0 x* ∧ *Y*.*flow0 x t* ∈ *A*
  using *t*
  by (*auto intro*!: *exI*[**where** *x*=*t*] *simp*: *flow0-cong*[*OF cong*] *existence-ivl0-cong*[*OF cong*])
ultimately show *?thesis*
  by (*auto simp*: *Y*.*returns-to-def*)
qed

lemma
returns-to-congI2:
assumes *auto-ll-on-open.returns-to g Y x A*
shows *returns-to x A*
proof −
interpret *Y*: *c1-on-open-euclidean g f′ Y* **by** (*rule c1-on-open-euclidean-congI*)
show *?thesis*
  by (*rule Y*.*returns-to-congI1*) (*auto simp*: *assms cong*)
qed

lemma *returns-to-cong*: *auto-ll-on-open.returns-to g Y A x* = *returns-to A x*
using *returns-to-congI1 returns-to-congI2* **by** *blast*

lemma
return-time-cong:
shows *return-time A x* = *auto-ll-on-open.return-time g Y A x*
proof −
interpret *Y*: *c1-on-open-euclidean g f′ Y* **by** (*rule c1-on-open-euclidean-congI*)
have *P-eq*: *0* < *t* ∧ *t* ∈ *existence-ivl0 x* ∧ *flow0 x t* ∈ *A* ∧ (∀ *s*∈{*0*<..<*t*}. *flow0 x s* ∉ *A*) ⟷
  0 < *t* ∧ *t* ∈ *Y*.*existence-ivl0 x* ∧ *Y*.*flow0 x t* ∈ *A* ∧ (∀ *s*∈{*0*<..<*t*}. *Y*.*flow0 x s* ∉ *A*)
  for *t*
  using *ivl-subset-existence-ivl*[*of t x*]
  apply (*auto simp*: *existence-ivl0-cong*[*OF cong*] *flow0-cong*[*OF cong*])
  apply (*drule bspec*)

apply *force*
apply (*subst* (*asm*) *flow0-cong*[*OF cong*])
apply *auto*
apply (*auto simp*: *existence-ivl0-cong*[*OF cong*, *symmetric*] *flow0-cong*[*OF cong*])
apply (*subst* (*asm*) *flow0-cong*[*OF cong*])
apply *auto*
done
show *?thesis*
unfolding *return-time-def Y.return-time-def*
by (*auto simp*: *returns-to-cong P-eq*)
qed

lemma *poincare-mapsto-congI1*:
assumes *poincare-mapsto A B C D E closed A*
shows *c1-on-open-euclidean.poincare-mapsto g Y A B C D E*
proof −
interpret *Y*: *c1-on-open-euclidean g f′ Y* **by** (*rule c1-on-open-euclidean-congI*)
show *?thesis*
using *assms*
unfolding *poincare-mapsto-def Y.poincare-mapsto-def*
apply *auto*
subgoal for *a b*
by (*rule returns-to-congI1*) *auto*
subgoal for *a b*
by (*subst return-time-cong*[*abs-def*, *symmetric*]) *auto*
subgoal for *a b*
unfolding *poincare-map-def Y.poincare-map-def*
apply (*drule bspec*, *assumption*)
apply *safe*
subgoal for *D*
apply (*auto intro*!: *exI*[**where** *x*=*D*])
subgoal premises *prems*
proof −
have $\forall_F$ *y in at a within C*. *returns-to A y*
apply (*rule eventually-returns-to-continuousI*)
apply *fact* **apply** *fact*
apply (*rule differentiable-imp-continuous-within*)
apply *fact*
done
moreover have $\forall_F$ *y in at a within C*. $y \in C$
by (*auto simp*: *eventually-at-filter*)
ultimately have $\forall_F$ *x′ in at a within C*. *flow0 x′* (*return-time A x′*) = *Y.flow0 x′* (*Y.return-time A x′*)
proof *eventually-elim*
case (*elim x′*)
then show *?case*
apply (*subst flow0-cong*[*OF cong*, *symmetric*], *force*)
apply (*subst return-time-cong*[*symmetric*])

using *prems*
apply (*auto intro*!: *return-time-exivl*)
apply (*subst return-time-cong*[*symmetric*])
apply *auto*
done
qed
with *prems*(*7*)
show *?thesis*
apply (*rule has-derivative-transform-eventually*)
using *prems*
apply (*subst flow0-cong*[*OF cong*, *symmetric*], *force*)
apply (*subst return-time-cong*[*symmetric*])
using *prems*
apply (*auto intro*!: *return-time-exivl*)
apply (*subst return-time-cong*[*symmetric*])
apply *auto*
done
qed
subgoal
apply (*subst flow0-cong*[*OF cong*, *symmetric*], *force*)
apply (*subst return-time-cong*[*symmetric*])
apply (*auto intro*!: *return-time-exivl*)
apply (*subst return-time-cong*[*symmetric*])
apply *auto*
done
done
done
subgoal for *a b t*
apply (*drule bspec*, *assumption*)
apply (*subst flow0-cong*[*OF cong*, *symmetric*])
apply *auto*
apply (*subst* (*asm*) *return-time-cong*[*symmetric*])
apply (*rule less-return-time-imp-exivl*)
apply (*rule less-imp-le*, *assumption*)
apply (*auto simp*: *return-time-cong*)
done
done
qed

lemma *poincare-mapsto-congI2*:
assumes *c1-on-open-euclidean.poincare-mapsto g Y A B C D E closed A*
shows *poincare-mapsto A B C D E*
proof −
interpret *Y*: *c1-on-open-euclidean g f′ Y* **by** (*rule c1-on-open-euclidean-congI*)
show *?thesis*
apply (*rule Y.poincare-mapsto-congI1*)
using *assms*
by (*auto simp*: *cong*)
qed

lemma *poincare-mapsto-cong*: *closed A* $\Longrightarrow$
poincare-mapsto A B C D E = *c1-on-open-euclidean.poincare-mapsto g Y A B C D E*
using *poincare-mapsto-congI1*[*of A B C*] *poincare-mapsto-congI2*[*of A B C*] **by** *auto*

end

end

end
theory *Cones*
imports
HOL−Analysis.Analysis
Triangle.Triangle
../ODE-Auxiliarities
begin

lemma *arcsin-eq-zero-iff*[*simp*]: $-1 \le x \Longrightarrow x \le 1 \Longrightarrow$ *arcsin* $x = 0 \longleftrightarrow x = 0$
using *sin-arcsin* **by** *fastforce*

definition *conemem* :: *'a::real-vector* ⇒ *'a* ⇒ *real* ⇒ *'a* **where** *conemem u v t* = *cos t* $*_R$ *u* + *sin t* $*_R$ *v*
definition *conesegment u v* = *conemem u v* ‘ {*0.. pi / 2*}

lemma
bounded-linear-image-conemem:
assumes *bounded-linear F*
shows *F* (*conemem u v t*) = *conemem* (*F u*) (*F v*) *t*
proof −
from *assms* **interpret** *bounded-linear F* **.**
show *?thesis*
by (*auto simp*: *conemem-def*[*abs-def*] *cone-hull-expl closed-segment-def add scaleR*)
qed

lemma
bounded-linear-image-conesegment:
assumes *bounded-linear F*
shows *F* ‘ *conesegment u v* = *conesegment* (*F u*) (*F v*)
proof −
from *assms* **interpret** *bounded-linear F* **.**
show *?thesis*
apply (*auto simp*: *conesegment-def conemem-def*[*abs-def*] *cone-hull-expl closed-segment-def add scaleR*)
apply (*auto simp*: *add*[*symmetric*] *scaleR*[*symmetric*])
done
qed

```
lemma discriminant: a ∗ x^2 + b ∗ x + c = (0::real) ⟹ 0 ≤ b^2 − 4 ∗ a ∗ c
  by (sos (((A<0 ∗ R<1) + (R<1 ∗ (R<1 ∗ [2∗a∗x + b]^2)))))

lemma quadratic-eq-factoring:
  assumes D: D = b^2 − 4 ∗ a ∗ c
  assumes nn: 0 ≤ D
  assumes x1: x_1 = (−b + sqrt D) / (2 ∗ a)
  assumes x2: x_2 = (−b − sqrt D) / (2 ∗ a)
  assumes a: a ≠ 0
  shows a ∗ x^2 + b ∗ x + c = a ∗ (x − x_1) ∗ (x − x_2)
  using nn
  by (simp add: D x1 x2)
    (simp add: assms algebra-simps power2-eq-square power3-eq-cube divide-simps)

lemma quadratic-eq-zeroes-iff:
  assumes D: D = b^2 − 4 ∗ a ∗ c
  assumes x1: x_1 = (−b + sqrt D) / (2 ∗ a)
  assumes x2: x_2 = (−b − sqrt D) / (2 ∗ a)
  assumes a: a ≠ 0
  shows a ∗ x^2 + b ∗ x + c = 0 ⟷ (D ≥ 0 ∧ (x = x_1 ∨ x = x_2)) (is ?z ⟷ -)
  using quadratic-eq-factoring[OF D - x1 x2 a, of x] discriminant[of a x b c] a
  by (auto simp: D)

lemma quadratic-ex-zero-iff:
  (∃x. a ∗ x^2 + b ∗ x + c = 0) ⟷ (a ≠ 0 ∧ b^2 − 4 ∗ a ∗ c ≥ 0 ∨ a = 0 ∧ (b
= 0 ⟶ c = 0))
  for a b c::real
  apply (cases a = 0)
  subgoal by (auto simp: intro: exI[where x=− c / b])
  subgoal by (subst quadratic-eq-zeroes-iff[OF refl refl refl]) auto
  done

lemma Cauchy-Schwarz-eq-iff:
  shows (inner x y)^2 = inner x x ∗ inner y y ⟷ ((∃k. x = k ∗_R y) ∨ y = 0)
proof safe
  assume eq: (x · y)^2 = x · x ∗ (y · y) and y ≠ 0
  define f where f ≡ λl. inner (x − l ∗_R y) (x − l ∗_R y)
  have f-quadratic: f l = inner y y ∗ l^2 + − 2 ∗ inner x y ∗ l + inner x x for l
    by (auto simp: f-def algebra-simps power2-eq-square inner-commute)
  have ∃l. f l = 0
    unfolding f-quadratic quadratic-ex-zero-iff
    using ‹y ≠ 0›
    by (auto simp: eq)
  then show (∃k. x = k ∗_R y)
    by (auto simp: f-def)
qed (auto simp: power2-eq-square)
```

lemma *Cauchy-Schwarz-strict-ineq*:
  $(inner\ x\ y)^2 < inner\ x\ x * inner\ y\ y$ **if** $y \neq 0$ $\bigwedge k.\ x \neq k *_R y$
  apply (*rule neq-le-trans*)
  subgoal
    using *that*
    unfolding *Cauchy-Schwarz-eq-iff*
    by *auto*
  subgoal by (*rule Cauchy-Schwarz-ineq*)
  done

lemma *Cauchy-Schwarz-eq2-iff*:
  $|inner\ x\ y| = norm\ x * norm\ y \longleftrightarrow ((\exists k.\ x = k *_R y) \vee y = 0)$
  using *Cauchy-Schwarz-eq-iff*[*of x y*]
  by (*subst power-eq-iff-eq-base*[*symmetric*, **where** $n = 2$])
    (*simp-all add*: *dot-square-norm power-mult-distrib*)

lemma *Cauchy-Schwarz-strict-ineq2*:
  $|inner\ x\ y| < norm\ x * norm\ y$ **if** $y \neq 0$ $\bigwedge k.\ x \neq k *_R y$
  apply (*rule neq-le-trans*)
  subgoal
    using *that*
    unfolding *Cauchy-Schwarz-eq2-iff*
    by *auto*
  subgoal by (*rule Cauchy-Schwarz-ineq2*)
  done

lemma *gt-minus-one-absI*: $abs\ k < 1 \Longrightarrow -1 < k$ **for** *k*::*real*
  by *auto*
lemma *gt-one-absI*: $abs\ k < 1 \Longrightarrow k < 1$ **for** *k*::*real*
  by *auto*

lemma *abs-impossible*:
  $|y1| < x1 \Longrightarrow |y2| < x2 \Longrightarrow x1 * x2 + y1 * y2 \neq 0$ **for** *x1 x2*::*real*
proof *goal-cases*
  case *1*
  have $- y1 * y2 \leq abs\ y1 * abs\ y2$
    by (*metis abs-ge-minus-self abs-mult mult.commute mult-minus-right*)
  also have $\ldots < x1 * x2$
    apply (*rule mult-strict-mono*)
    using *1* **by** *auto*
  finally show *?case* **by** *auto*
qed

lemma *vangle-eq-arctan-minus*:— TODO: generalize?!
  assumes *ij*: $i \in Basis$ $j \in Basis$ **and** *ij-neq*: $i \neq j$
  assumes *xy1*: $|y1| < x1$
  assumes *xy2*: $|y2| < x2$
  assumes *less*: $y2\ /\ x2 > y1\ /\ x1$

shows *vangle* (*x1* $*_R$ *i* + *y1* $*_R$ *j*) (*x2* $*_R$ *i* + *y2* $*_R$ *j*) = *arctan* (*y2* / *x2*) − *arctan* (*y1* / *x1*)
(**is** *vangle ?u ?v* = -)
proof −
from *assms* **have** *less2*: *x2* ∗ *y1* − *x1* ∗ *y2* < *0*
by (*auto simp*: *divide-simps abs-real-def algebra-simps split*: *if-splits*)
have *norm-eucl*: *norm* (*x* $*_R$ *i* + *y* $*_R$ *j*) = *sqrt* ((*norm x*)^2 + (*norm y*)^2) **for** *x y*
apply (*subst norm-eq-sqrt-inner*)
using *ij ij-neq*
by (*auto simp*: *inner-simps inner-Basis power2-eq-square*)
have *nonzeroes*: *x1* $*_R$ *i* + *y1* $*_R$ *j* ≠ *0 x2* $*_R$ *i* + *y2* $*_R$ *j* ≠ *0*
apply (*auto simp*: *euclidean-eq-iff* [**where** *'a='a*] *inner-simps intro*!: *bexI*[**where** *x=i*])
using *assms*
by (*auto simp*: *inner-Basis*)
have *indep*: *x1* $*_R$ *i* + *y1* $*_R$ *j* ≠ *k* $*_R$ (*x2* $*_R$ *i* + *y2* $*_R$ *j*) **for** *k*
proof
assume *x1* $*_R$ *i* + *y1* $*_R$ *j* = *k* $*_R$ (*x2* $*_R$ *i* + *y2* $*_R$ *j*)
then have *x1* / *x2* = *k y1* = *k* ∗ *y2*
using *ij ij-neq xy1 xy2*
apply (*auto simp*: *abs-real-def divide-simps algebra-simps euclidean-eq-iff* [**where** *'a='a*] *inner-simps*
split: *if-splits*)
by (*auto simp*: *inner-Basis split*: *if-splits*)
then have *y1* = *x1* / *x2* ∗ *y2* **by** *simp*
with *less* **show** *False* **using** *xy1* **by** (*auto split*: *if-splits*)
qed
have ((*x1*^{2} + *y1*^{2}) ∗ (*x2*^{2} + *y2*^{2}) ∗
(*1* − ((*x1* $*_R$ *i* + *y1* $*_R$ *j*) · (*x2* $*_R$ *i* + *y2* $*_R$ *j*))^2 / ((*x1*^{2} + *y1*^{2}) ∗ (*x2*^{2} + *y2*^{2})))) =
((*x1*^{2} + *y1*^{2}) ∗ (*x2*^{2} + *y2*^{2}) ∗
(*1* − (*x1* ∗ *x2* + *y1* ∗ *y2*)^2 / ((*x1*^{2} + *y1*^{2}) ∗ (*x2*^{2} + *y2*^{2}))))
using *ij-neq ij*
by (*auto simp*: *algebra-simps divide-simps inner-simps inner-Basis*)
also have . . . = (*x1*^{2} + *y1*^{2}) ∗ (*x2*^{2} + *y2*^{2}) − (*x1* ∗ *x2* + *y1* ∗ *y2*)^2
unfolding *right-diff-distrib* **by** *simp*
also have . . . = (*x2* ∗ *y1* − *x1* ∗ *y2*)^*2*
by (*auto simp*: *algebra-simps power2-eq-square*)
also have *sqrt* . . . = |*x2* ∗ *y1* − *x1* ∗ *y2*|
by *simp*
also have . . . = *x1* ∗ *y2* − *x2* ∗ *y1*
using *less2*
by (*simp add*: *abs-real-def*)
finally have *sqrt-eq*: *sqrt* ((*x1*^{2} + *y1*^{2}) ∗ (*x2*^{2} + *y2*^{2}) ∗
(*1* − ((*x1* $*_R$ *i* + *y1* $*_R$ *j*) · (*x2* $*_R$ *i* + *y2* $*_R$ *j*))^2 / ((*x1*^{2} + *y1*^{2}) ∗ (*x2*^{2} + *y2*^{2})))) =
x1 ∗ *y2* − *x2* ∗ *y1*
.

```
  show ?thesis
    using ij xy1 xy2
    unfolding vangle-def
    apply (subst arccos-arctan)
    subgoal
      apply (rule gt-minus-one-absI)
      apply simp
      apply (subst pos-divide-less-eq)
      subgoal
        apply (rule mult-pos-pos)
        using nonzeroes
        by auto
      subgoal
        apply simp
        apply (rule Cauchy-Schwarz-strict-ineq2)
        using nonzeroes indep
        by auto
      done
    subgoal
      apply (rule gt-one-absI)
      apply simp
      apply (subst pos-divide-less-eq)
      subgoal
        apply (rule mult-pos-pos)
        using nonzeroes
        by auto
      subgoal
        apply simp
        apply (rule Cauchy-Schwarz-strict-ineq2)
        using nonzeroes indep
        by auto
      done
    subgoal
      apply (auto simp: nonzeroes)
      apply (subst (3) diff-conv-add-uminus)
      apply (subst arctan-minus[symmetric])
      apply (subst arctan-add)
        apply force
       apply force
      apply (subst arctan-inverse[symmetric])
      subgoal
        apply (rule divide-pos-pos)
        subgoal
          apply (auto simp add: inner-simps inner-Basis algebra-simps )
           apply (thin-tac - ∈ Basis)+ apply (thin-tac j = i)
            apply (sos ((((A<0 * (A<1 * (A<2 * A<3))) * R<1) + ((A<=0 *
(A<0 * (A<2 * R<1))) * (R<1 * [1]^2)))))
          apply (thin-tac - ∈ Basis)+ apply (thin-tac j ≠ i)
           by (sos ((((A<0 * (A<1 * (A<2 * A<3))) * R<1) + (((A<2 * (A<3
```

```
* R<1)) * (R<1/3 * [y1]^2)) + (((A<1 * (A<3 * R<1)) * ((R<1/12 * [x2 +
y1]^2) + (R<1/12 * [x1 + y2]^2))) + (((A<1 * (A<2 * R<1)) * (R<1/12 *
[~1*x1 + x2 + y1 + y2]^2)) + (((A<0 * (A<3 * R<1)) * (R<1/12 * [~1*x1
+ x2 + ~1*y1 + ~1*y2]^2)) + (((A<0 * (A<2 * R<1)) * ((R<1/12 * [x2
+ ~1*y1]^2) + (R<1/12 * [~1*x1 + y2]^2))) + (((A<0 * (A<1 * R<1)) *
(R<1/3 * [y2]^2)) + ((A<=0 * R<1) * (R<1/3 * [x1 + x2]^2))))))))))
        subgoal
          apply (intro mult-pos-pos)
          using nonzeroes indep
            apply auto
          apply (rule gt-one-absI)
          apply (simp add: power-divide power-mult-distrib power2-norm-eq-inner)
          apply (rule Cauchy-Schwarz-strict-ineq)
           apply auto
          done
        done
      subgoal
        apply (rule arg-cong[where f=arctan])
        using nonzeroes ij-neq
        apply (auto simp: norm-eucl)
        apply (subst real-sqrt-mult[symmetric])
        apply (subst real-sqrt-mult[symmetric])
        apply (subst real-sqrt-mult[symmetric])
        apply (subst power-divide)
        apply (subst real-sqrt-pow2)
         apply simp
        apply (subst nonzero-divide-eq-eq)
        subgoal
          apply (auto simp: algebra-simps inner-simps inner-Basis)
          by (auto simp: algebra-simps divide-simps abs-real-def abs-impossible)
        apply (subst sqrt-eq)
        apply (auto simp: algebra-simps inner-simps inner-Basis)
        apply (auto simp: algebra-simps divide-simps abs-real-def abs-impossible)
        by (auto split: if-splits)
      done
    done
qed

lemma vangle-le-pi2: 0 ≤ u · v ⟹ vangle u v ≤ pi/2
  unfolding vangle-def atLeastAtMost-iff
  apply (simp del: le-divide-eq-numeral1)
  apply (intro impI arccos-le-pi2 arccos-lbound)
  using Cauchy-Schwarz-ineq2[of u v]
  by (auto simp: algebra-simps)

lemma inner-eq-vangle: u · v = cos (vangle u v) * (norm u * norm v)
  by (simp add: cos-vangle)

lemma vangle-scaleR-self:
```

vangle $(k *_R v)$ *v* = (*if* *k* = *0* ∨ *v* = *0* *then* *pi* / *2* *else* *if* *k* > *0* *then* *0* *else* *pi*)
vangle *v* $(k *_R v)$ = (*if* *k* = *0* ∨ *v* = *0* *then* *pi* / *2* *else* *if* *k* > *0* *then* *0* *else* *pi*)
by (*auto simp*: *vangle-def dot-square-norm power2-eq-square*)

lemma *vangle-scaleR*:
vangle $(k *_R v)$ *w* = *vangle v w vangle w* $(k *_R v)$ = *vangle w v* **if** *k* > *0*
using *that*
by (*auto simp*: *vangle-def*)

lemma *cos-vangle-eq-zero-iff-vangle*:
cos (*vangle u v*) = *0* ⟷ (*u* = *0* ∨ *v* = *0* ∨ *u* · *v* = *0*)
using *Cauchy-Schwarz-ineq2*[*of u v*]
by (*auto simp*: *vangle-def divide-simps algebra-split-simps split*: *if-splits*)

lemma *ortho-imp-angle-pi-half*: *u* · *v* = *0* ⟹ *vangle u v* = *pi* / *2*
using *orthogonal-iff-vangle*[*of u v*]
by (*auto simp*: *orthogonal-def*)

lemma *arccos-eq-zero-iff*: *arccos x* = *0* ⟷ *x* = *1* **if** −*1* ≤ *x x* ≤ *1*
using *that*
apply *auto*
using *cos-arccos* **by** *fastforce*

lemma *vangle-eq-zeroD*: *vangle u v* = *0* ⟹ (∃ *k*. *v* = $k *_R u$)
apply (*auto simp*: *vangle-def split*: *if-splits*)
apply (*subst* (*asm*) *arccos-eq-zero-iff*)
apply (*auto simp*: *divide-simps mult-less-0-iff split*: *if-splits*)
apply (*metis Real-Vector-Spaces.norm-minus-cancel inner-minus-left minus-le-iff norm-cauchy-schwarz*)
apply (*metis norm-cauchy-schwarz*)
by (*metis Cauchy-Schwarz-eq2-iff abs-of-pos inner-commute mult.commute mult-sign-intros*(*5*) *zero-less-norm-iff*)

lemma *less-one-multI*:— TODO: also in AA!
fixes *e x*::*real*
shows *e* ≤ *1* ⟹ *0* < *x* ⟹ *x* < *1* ⟹ *e* ∗ *x* < *1*
by (*metis* (*erased*, *opaque-lifting*) *less-eq-real-def monoid-mult-class.mult.left-neutral mult-strict-mono zero-less-one*)

lemma *conemem-expansion-estimate*:
fixes *u v u′ v′*::*′a*::*euclidean-space*
assumes *t* ∈ {*0* .. *pi* / *2*}
assumes *angle-pos*: *0* < *vangle u v vangle u v* < *pi* / *2*
assumes *angle-le*: (*vangle u′ v′*) ≤ (*vangle u v*)
assumes *norm u* = *1 norm v* = *1*
shows *norm* (*conemem u′ v′ t*) ≥ *min* (*norm u′*) (*norm v′*) ∗ *norm* (*conemem u v t*)
proof −

define *e-pre* **where** *e-pre* = *min* (*norm u'*) (*norm v'*)
let *?w* = *conemem u v*
let *?w'* = *conemem u' v'*
have *cos-angle-le*: *cos* (*vangle u' v'*) ≥ *cos* (*vangle u v*)
using *angle-pos vangle-bounds*
by (*auto intro*!: *cos-monotone-0-pi-le angle-le*)
have *e-pre-le*: $e\text{-}pre^2$ ≤ *norm u'* ∗ *norm v'*
by (*auto simp*: *e-pre-def min-def power2-eq-square intro*: *mult-left-mono mult-right-mono*)
have *lt*: *0* < *1* + *2* ∗ (*u* · *v*) ∗ *sin t* ∗ *cos t*
proof −
have |*u* · *v*| < *norm u* ∗ *norm v*
apply (*rule Cauchy-Schwarz-strict-ineq2*)
using *assms*
apply *auto*
apply (*subst* (*asm*) *vangle-scaleR-self*)+
by (*auto simp*: *split*: *if-splits*)
then have *abs* (*u* · *v* ∗ *sin* (*2* ∗ *t*)) < *1*
using *assms*
apply (*auto simp add*: *abs-mult*)
apply (*subst mult.commute*)
apply (*rule less-one-multI*)
apply (*auto simp add*: *abs-mult inner-eq-vangle*)
by (*auto simp*: *cos-vangle-eq-zero-iff-vangle dest*!: *ortho-imp-angle-pi-half*)
then show *?thesis*
by (*subst mult.assoc sin-times-cos*)+ *auto*
qed
have *le*: *0* ≤ *1* + *2* ∗ (*u* · *v*) ∗ *sin t* ∗ *cos t*
proof −
have |*u* · *v*| ≤ *norm u* ∗ *norm v*
by (*rule Cauchy-Schwarz-ineq2*)
then have *abs* (*u* · *v* ∗ *sin* (*2* ∗ *t*)) ≤ *1*
by (*auto simp add*: *abs-mult assms intro*!: *mult-le-one*)
then show *?thesis*
by (*subst mult.assoc sin-times-cos*)+ *auto*
qed
have $(norm\ (?w\ t))^2 = (cos\ t)^2 *_R (norm\ u)^2 + (sin\ t)^2 *_R (norm\ v)^2 + 2 * (u \cdot v) * sin\ t * cos\ t$
by (*auto simp*: *conemem-def algebra-simps power2-norm-eq-inner*)
(*auto simp*: *power2-eq-square inner-commute*)
also have . . . = *1* + *2* ∗ (*u* · *v*) ∗ *sin t* ∗ *cos t*
by (*auto simp*: *sin-squared-eq algebra-simps assms*)
finally have $(norm\ (conemem\ u\ v\ t))^2 = 1 + 2 * (u \cdot v) * sin\ t * cos\ t$ **by** *simp*
moreover
have $(norm\ (?w'\ t))^2 = (cos\ t)^2 *_R (norm\ u')^2 + (sin\ t)^2 *_R (norm\ v')^2 + 2 * (u' \cdot v') * sin\ t * cos\ t$
by (*auto simp*: *conemem-def algebra-simps power2-norm-eq-inner*)
(*auto simp*: *power2-eq-square inner-commute*)
ultimately
have $(norm\ (?w'\ t)\ /\ norm\ (?w\ t))^2 =$

((*cos t*)^2 $*_R$ (*norm u′*)^2 + (*sin t*)^2 $*_R$ (*norm v′*)^2 + *2* ∗ (*u′* · *v′*) ∗ *sin t* ∗ *cos t*) /
(*1* + *2* ∗ (*u* · *v*) ∗ *sin t* ∗ *cos t*)
(**is** - = (*?a* + *?b*) / *?c*)
by (*auto simp*: *divide-inverse power-mult-distrib*) (*auto simp*: *inverse-eq-divide power2-eq-square*)
also have . . . ≥ (*e-pre*^{2} + *?b*) / *?c*
apply (*rule divide-right-mono*)
apply (*rule add-right-mono*)
subgoal using *assms e-pre-def*
apply (*auto simp*: *min-def*)
subgoal by (*auto simp*: *algebra-simps cos-squared-eq intro*!: *mult-right-mono power-mono*)
subgoal by (*auto simp*: *algebra-simps sin-squared-eq intro*!: *mult-right-mono power-mono*)
done
subgoal by (*rule le*)
done
also (*xtrans*)
have *inner-nonneg*: *u′* · *v′* ≥ *0*
using *angle-le*(*1*) *angle-pos vangle-bounds*[*of u′ v′*]
by (*auto simp*: *inner-eq-vangle intro*!: *mult-nonneg-nonneg cos-ge-zero*)
from *vangle-bounds*[*of u′ v′*] *vangle-le-pi2*[*OF this*]
have *u′v′e-pre*: *u′* · *v′* ≥ *cos* (*vangle u′ v′*) ∗ *e-pre*^{2}
apply (*subst inner-eq-vangle*)
apply (*rule mult-left-mono*)
apply (*rule e-pre-le*)
apply (*rule cos-ge-zero*)
by *auto*
have (*e-pre*^{2} + *?b*) / *?c* ≥ (*e-pre*^{2} + *2* ∗ (*cos* (*vangle u′ v′*) ∗ *e-pre*^{2}) ∗ *sin t* ∗ *cos t*) / *?c*
(**is** - ≥ *?ddd*)
apply (*intro divide-right-mono add-left-mono mult-right-mono mult-left-mono u′v′e-pre*)
using ‹*t* ∈ -›
by (*auto intro*!: *mult-right-mono sin-ge-zero divide-right-mono le cos-ge-zero*
simp: *sin-times-cos u′v′e-pre*)
also (*xtrans*) **have** *?ddd* = *e-pre*^{2} ∗ ((*1* + *2* ∗ *cos* (*vangle u′ v′*) ∗ *sin t* ∗ *cos t*) / *?c*) (**is** - = *?ddd*)
by (*auto simp add*: *divide-simps algebra-simps*)
also (*xtrans*)
have *sc-ge-0*: *0* ≤ *sin t* ∗ *cos t*
using ‹*t* ∈ -›
by (*auto simp*: *assms cos-angle-le intro*!: *mult-nonneg-nonneg sin-ge-zero cos-ge-zero*)
have *?ddd* ≥ *e-pre*^{2}
apply (*subst mult-le-cancel-left1*)
apply (*auto simp add*: *divide-simps split*: *if-splits*)
apply (*rule mult-right-mono*)
using *lt*

by (*auto simp*: *assms inner-eq-vangle intro*!: *mult-right-mono sc-ge-0 cos-angle-le*)
finally (*xtrans*)
have (*norm* (*conemem* u' v' *t*))^2 $\geq$ (*e-pre* $*$ *norm* (*conemem u v t*))^2
by (*simp add*: *divide-simps power-mult-distrib split*: *if-splits*)
then show *norm* (*conemem* u' v' *t*) $\geq$ *e-pre* $*$ *norm* (*conemem u v t*)
using *norm-imp-pos-and-ge power2-le-imp-le* **by** *blast*
qed

lemma *conemem-commute*: *conemem a b t* = *conemem b a* (*pi* / *2* − *t*) **if** *0* $\leq$ *t*
t $\leq$ *pi* / *2*
using *that* **by** (*auto simp*: *conemem-def cos-sin-eq algebra-simps*)

lemma *conesegment-commute*: *conesegment a b* = *conesegment b a*
apply (*auto simp*: *conesegment-def*)
apply (*subst conemem-commute*)
apply *auto*
apply (*subst conemem-commute*)
apply *auto*
done

definition *conefield u v* = *cone hull* (*conesegment u v*)

lemma *conefield-alt-def*: *conefield u v* = *cone hull* {*u*−−*v*}
apply (*auto simp*: *conesegment-def conefield-def cone-hull-expl in-segment*)
subgoal premises *prems* **for** *c t*
proof −
from *prems*
have *sc-pos*: *sin t* + *cos t* > *0*
apply (*cases t* = *0*)
subgoal
by (*rule add-nonneg-pos*) *auto*
subgoal
by (*auto intro*!: *add-pos-nonneg sin-gt-zero cos-ge-zero*)
done
then have *1*: (*sin t* / (*sin t* + *cos t*) + *cos t* / (*sin t* + *cos t*)) = *1*
by (*auto simp*: *divide-simps*)
have $\exists$ *c x*. *c* > *0* $\wedge$ *0* $\leq$ *x* $\wedge$ *x* $\leq$ *1* $\wedge$ *c* $*_R$ *conemem u v t* = (*1* − *x*) $*_R$ *u* +
x $*_R$ *v*
apply (*auto simp*: *algebra-simps conemem-def*)
apply (*rule exI*[**where** *x*=*1* / (*sin t* + *cos t*)])
using *prems*
by (*auto intro*!: *exI*[**where** *x*=(*1* / (*sin t* + *cos t*) $*$ *sin t*)] *sc-pos*
divide-nonneg-nonneg sin-ge-zero add-nonneg-nonneg cos-ge-zero
simp: *scaleR-add-left*[*symmetric*] *1 divide-le-eq-1*)
then obtain *d x* **where** *dx*: *d* > *0* *conemem u v t* = (*1* / *d*) $*_R$ ((*1* − *x*) $*_R$
u + *x* $*_R$ *v*)
0 $\leq$ *x x* $\leq$ *1*
by (*auto simp*: *eq-vector-fraction-iff*)

```
    show ?thesis
      apply (rule exI[where x=c / d])
      using dx
      by (auto simp: intro!: divide-nonneg-nonneg prems )
  qed
  subgoal premises prems for c t
  proof -
    let ?x = arctan (t / (1 - t))
    let ?s = t / sin ?x
    have *: c *R ((1 - t) *R u + t *R v) = (c * ?s) *R (cos ?x *R u + sin ?x *R
v)
      if 0 < t t < 1
      using that
      by (auto simp: scaleR-add-right sin-arctan cos-arctan divide-simps)
    show ?thesis
      apply (cases t = 0)
      subgoal
        apply simp
        apply (rule exI[where x=c])
        apply (rule exI[where x=u])
        using prems
        by (auto simp: conemem-def[abs-def] intro!: image-eqI[where x=0])
      subgoal apply (cases t = 1)
        subgoal
          apply simp
          apply (rule exI[where x=c])
          apply (rule exI[where x=v])
          using prems
          by (auto simp: conemem-def[abs-def] intro!: image-eqI[where x=pi/2])
        subgoal
          apply (rule exI[where x=(c * ?s)])
          apply (rule exI[where x=(cos ?x *R u + sin ?x *R v)])
          using prems * arctan-ubound[of t / (1 - t)]
          apply (auto simp: conemem-def[abs-def]  intro!: imageI)
          by (auto simp: scaleR-add-right sin-arctan)
        done
      done
  qed
  done

lemma
  bounded-linear-image-cone-hull:
  assumes bounded-linear F
  shows F ` (cone hull T) = cone hull (F ` T)
proof -
  from assms interpret bounded-linear F .
  show ?thesis
    apply (auto simp: conefield-def cone-hull-expl closed-segment-def add scaleR)
     apply auto
```

apply (*auto simp*: *add*[*symmetric*] *scaleR*[*symmetric*])
done
qed

lemma
bounded-linear-image-conefield:
assumes *bounded-linear F*
shows *F ‘ conefield u v* = *conefield* (*F u*) (*F v*)
unfolding *conefield-def*
using *assms*
by (*auto simp*: *bounded-linear-image-conesegment bounded-linear-image-cone-hull*)

lemma *conefield-commute*: *conefield x y* = *conefield y x*
by (*auto simp*: *conefield-def conesegment-commute*)

lemma *convex-conefield*: *convex* (*conefield x y*)
by (*auto simp*: *conefield-alt-def convex-cone-hull*)

lemma *conefield-scaleRI*: *v* ∈ *conefield* ($r *_R x$) *y* **if** *v* ∈ *conefield x y r* > *0*
using *that*
using ‹*r* > *0*›
unfolding *conefield-alt-def cone-hull-expl*
apply (*auto simp*: *in-segment*)
proof *goal-cases*
case (*1 c u*)
let *?d* = *c* ∗ (*1* − *u*) / *r* + *c* ∗ *u*
let *?t* = *c* ∗ *u* / *?d*
have *c* ∗ (*1* − *u*) = *?d* ∗ (*1* − *?t*) ∗ *r* **if** *0* < *u*
using ‹*0* < *r*› *that*(*1*) *1*(*3*,*5*) *mult-pos-pos*
by (*force simp*: *divide-simps ac-simps ring-distribs*[*symmetric*])
then have *eq1*: (*c* ∗ (*1* − *u*)) $*_R$ *x* = (*?d* ∗ (*1* − *?t*) ∗ *r*) $*_R$ *x* **if** *0* < *u*
using *that* **by** *simp*
have *c* ∗ *u* = *?d* ∗ *?t* **if** *u* < *1*
using ‹*0* < *r*› *that*(*1*) *1*(*3*,*4*,*5*) *mult-pos-pos*
apply (*auto simp*: *divide-simps ac-simps ring-distribs*[*symmetric*])
proof −
assume *0* ≤ *u*
0 < *r*
1 − *u* + *r* ∗ *u* = *0*
u < *1*
then have *False*
by (*sos* ((((*A*<*0* ∗ *A*<*1*) ∗ *R*<*1*) + (([~*1*∗*r*] ∗ *A*=*0*) + ((*A*<=*0* ∗ *R*<*1*) ∗
(*R*<*1* ∗ [*r*]^*2*))))))
then show *u* = *0*
by *metis*
qed
then have *eq2*: (*c* ∗ *u*) $*_R$ *y* = (*?d* ∗ *?t*) $*_R$ *y* **if** *u* < *1*
using *that* **by** *simp*
have ∗: *c* $*_R$ ((*1* − *u*) $*_R$ *x* + *u* $*_R$ *y*) = *?d* $*_R$ ((*1* − *?t*) $*_R$ *r* $*_R$ *x* + *?t* $*_R$ *y*)

if *0 < u u < 1*
using *that eq1 eq2*
by (*auto simp*: *algebra-simps*)
show *?case*
apply (*cases u = 0*)
subgoal using *1* **by** (*intro exI*[**where** *x=c / r*] *exI*[**where** *x=r* $*_R$ *x*]) *auto*
apply (*cases u = 1*)
subgoal using *1* **by** (*intro exI*[**where** *x=c*] *exI*[**where** *x=y*]) (*auto intro*!: *exI*[**where** *x=1*])
subgoal
apply (*rule exI*[**where** *x=?d*])
apply (*rule exI*[**where** *x*=((*1 − ?t*) $*_R$ *r* $*_R$ *x + ?t* $*_R$ *y*)])
apply (*subst* ∗)
using *1*
apply (*auto intro*!: *exI*[**where** *x = ?t*])
apply (*auto simp*: *algebra-simps divide-simps*)
defer
proof −
assume *a1*: *c + c* ∗ (*r* ∗ *u*) < *c* ∗ *u*
assume *a2*: *0* ≤ *c*
assume *a3*: *0* ≤ *u*
assume *a4*: *u* ≠ *0*
assume *a5*: *0* < *r*
have *c + c* ∗ (*r* ∗ *u*) ≤ *c* ∗ *u*
using *a1 less-eq-real-def* **by** *blast*
then show *c* ≤ *c* ∗ *u*
using *a5 a4 a3 a2* **by** (*metis* (*no-types*) *less-add-same-cancel1 less-eq-real-def mult-pos-pos order-trans real-scaleR-def real-vector.scale-zero-left*)
next
assume *a1*: *0* ≤ *c*
assume *a2*: *u* ≤ *1*
have *f3*: ∀ *x0*. ((*x0*::*real*) < *1*) = (¬ *1* ≤ *x0*)
by *auto*
have *f4*: ∀ *x0*. ((*1*::*real*) < *x0*) = (¬ *x0* ≤ *1*)
by *fastforce*
have ∀ *x0 x1*. ((*x1*::*real*) < *x1* ∗ *x0*) = (¬ *0* ≤ *x1* + − *1* ∗ (*x1* ∗ *x0*))
by *auto*
then have (∀ *r ra*. ((*r*::*real*) < *r* ∗ *ra*) = ((*0* ≤ *r* ⟶ *1* < *ra*) ∧ (*r* ≤ *0* ⟶ *ra* < *1*))) = (∀ *r ra*. (¬ (*0*::*real*) ≤ *r* + − *1* ∗ (*r* ∗ *ra*)) = ((¬ *0* ≤ *r* ∨ ¬ *ra* ≤ *1*) ∧ (¬ *r* ≤ *0* ∨ ¬ *1* ≤ *ra*)))
using *f4 f3* **by** *presburger*
then have *0* ≤ *c* + − *1* ∗ (*c* ∗ *u*)
using *a2 a1 mult-less-cancel-left1* **by** *blast*
then show *c* ∗ *u* ≤ *c*
by *auto*
qed
done
qed

lemma *conefield-scaleRD*: $v \in$ *conefield* $x\ y$ **if** $v \in$ *conefield* $(r *_R x)\ y\ r > 0$
 using *conefield-scaleRI*[*OF that*(*1*) *positive-imp-inverse-positive*[*OF that*(*2*)]] *that*(*2*)
 by *auto*

lemma *conefield-scaleR*: *conefield* $(r *_R x)\ y =$ *conefield* $x\ y$ **if** $r > 0$
 using *conefield-scaleRD conefield-scaleRI that*
 by *blast*

lemma *conefield-expansion-estimate*:
 fixes *u v*::*'a*::*euclidean-space* **and** *F*::*'a* ⇒ *'a*
 assumes $t \in \{0\ ..\ pi\ /\ 2\}$
 assumes *angle-pos*: $0 <$ *vangle u v vangle u v* $< pi\ /\ 2$
 assumes *angle-le*: *vangle* (*F u*) (*F v*) ≤ *vangle u v*
 assumes *bounded-linear F*
 assumes $x \in$ *conefield u v*
 shows *norm* (*F x*) ≥ *min* (*norm* (*F u*)/*norm u*) (*norm* (*F v*)/*norm v*) ∗ *norm x*
proof *cases*
 assume [*simp*]: $x \neq 0$
 from *assms* **have** [*simp*]: $u \neq 0\ v \neq 0$ **by** *auto*
 interpret *bounded-linear F* **by** *fact*
 define *u1* **where** $u1 = u\ /_R$ *norm u*
 define *v1* **where** $v1 = v\ /_R$ *norm v*
 note ‹$x \in$ *conefield u v*›
 also have ‹*conefield u v* = *conefield u1 v1*›
  by (*auto simp*: *u1-def v1-def conefield-scaleR conefield-commute*[*of u*])
 finally obtain *c t* **where** *x*: $x = c *_R$ *conemem u1 v1 t* $t \in \{0\ ..\ pi\ /\ 2\}\ c \geq 0$
  by (*auto simp*: *conefield-def cone-hull-expl conesegment-def*)
 then have *xc*: $x\ /_R\ c =$ *conemem u1 v1 t*
  by (*auto simp*: *divide-simps*)
 also have *F* ... = *conemem* (*F u1*) (*F v1*) *t*
  by (*simp add*: *bounded-linear-image-conemem assms*)
 also have *norm* ... ≥ *min* (*norm* (*F u1*)) (*norm* (*F v1*)) ∗ *norm* (*conemem u1 v1 t*)
  apply (*rule conemem-expansion-estimate*)
  subgoal by *fact*
  subgoal using *angle-pos* **by** (*simp add*: *u1-def v1-def vangle-scaleR*)
  subgoal using *angle-pos* **by** (*simp add*: *u1-def v1-def vangle-scaleR*)
  subgoal using *angle-le* **by** (*simp add*: *u1-def v1-def scaleR vangle-scaleR*)
  subgoal using *angle-le* **by** (*simp add*: *u1-def v1-def scaleR vangle-scaleR*)
  subgoal using *angle-le* **by** (*simp add*: *u1-def v1-def scaleR vangle-scaleR*)
  done
 finally show *norm* (*F x*) ≥ *min* (*norm* (*F u*)/*norm u*) (*norm* (*F v*)/*norm v*) ∗ *norm x*
  unfolding *xc*[*symmetric*] *scaleR u1-def v1-def norm-scaleR x*
  using ‹$c \geq 0$›
  by (*simp add*: *divide-simps split*: *if-splits*)
qed *simp*

lemma *conefield-rightI*:

assumes *ij*: $i \in Basis$ $j \in Basis$ **and** *ij-neq*: $i \neq j$
assumes $y \in \{y1 .. y2\}$
shows $(i + y *_R j) \in conefield\ (i + y1 *_R j)\ (i + y2 *_R j)$
unfolding *conefield-alt-def*
apply (*rule hull-inc*)
using *assms*
by (*auto simp*: *in-segment divide-simps inner-Basis algebra-simps*
intro!: *exI*[**where** $x=(y - y1) / (y2 - y1)$] *euclidean-eqI*[**where** $'a='a$])

lemma *conefield-right-vangleI*:
assumes *ij*: $i \in Basis$ $j \in Basis$ **and** *ij-neq*: $i \neq j$
assumes $y \in \{y1 .. y2\}$ $y1 < y2$
shows $(i + y *_R j) \in conefield\ (i + y1 *_R j)\ (i + y2 *_R j)$
unfolding *conefield-alt-def*
apply (*rule hull-inc*)
using *assms*
by (*auto simp*: *in-segment divide-simps inner-Basis algebra-simps*
intro!: *exI*[**where** $x=(y - y1) / (y2 - y1)$] *euclidean-eqI*[**where** $'a='a$])

lemma *cone-conefield*[*intro*, *simp*]: *cone* (*conefield x y*)
unfolding *conefield-def*
by (*rule cone-cone-hull*)

lemma *conefield-mk-rightI*:
assumes *ij*: $i \in Basis$ $j \in Basis$ **and** *ij-neq*: $i \neq j$
assumes $(i + (y / x) *_R j) \in conefield\ (i + (y1 / x1) *_R j)\ (i + (y2 / x2) *_R j)$
assumes $x > 0$ $x1 > 0$ $x2 > 0$
shows $(x *_R i + y *_R j) \in conefield\ (x1 *_R i + y1 *_R j)\ (x2 *_R i + y2 *_R j)$
proof −
have *rescale*: $(x *_R i + y *_R j) = x *_R (i + (y / x) *_R j)$ **if** $x > 0$ **for** x y
using *that* **by** (*auto simp*: *algebra-simps*)
show *?thesis*
unfolding *rescale*[*OF* ‹$x > 0$›] *rescale*[*OF* ‹$x1 > 0$›] *rescale*[*OF* ‹$x2 > 0$›]
conefield-scaleR[*OF* ‹$x1 > 0$›]
apply (*subst conefield-commute*)
unfolding *conefield-scaleR*[*OF* ‹$x2 > 0$›]
apply (*rule mem-cone*)
apply *simp*
apply (*subst conefield-commute*)
by (*auto intro*!: *assms less-imp-le*)
qed

lemma *conefield-prod3I*:
assumes $x > 0$ $x1 > 0$ $x2 > 0$
assumes $y1 / x1 \le y / x$ $y / x \le y2 / x2$
shows $(x, y, 0) \in (conefield\ (x1, y1, 0)\ (x2, y2, 0)::(real*real*real)\ set)$
proof −
have $(x *_R (1, 0, 0) + y *_R (0, 1, 0)) \in$

(*conefield* (*x1* $*_R$ (*1*, *0*, *0*) + *y1* $*_R$ (*0*, *1*, *0*)) (*x2* $*_R$ (*1*, *0*, *0*) + *y2* $*_R$ (*0*, *1*, *0*))::(*real*∗*real*∗*real*) *set*)
apply (*rule conefield-mk-rightI*)
subgoal by (*auto simp*: *Basis-prod-def zero-prod-def*)
subgoal by (*auto simp*: *Basis-prod-def zero-prod-def*)
subgoal by (*auto simp*: *Basis-prod-def zero-prod-def*)
subgoal using *assms* **by** (*intro conefield-rightI*) (*auto simp*: *Basis-prod-def zero-prod-def*)
by (*auto intro*: *assms*)
then show *?thesis* **by** *simp*
qed

end

7 Linear ODE

theory *Linear-ODE*
imports
../IVP/Flow
Bounded-Linear-Operator
Multivariate-Taylor
begin

lemma
exp-scaleR-has-derivative-right[*derivative-intros*]:
fixes *f*::*real* ⇒ *real*
assumes (*f has-derivative f′*) (*at x within s*)
shows ((λ*x*. *exp* (*f x* $*_R$ *A*)) *has-derivative* (λ*h*. *f′ h* $*_R$ (*exp* (*f x* $*_R$ *A*) ∗ *A*))) (*at x within s*)
proof −
from *assms* **have** *bounded-linear f′* **by** *auto*
with *real-bounded-linear* **obtain** *m* **where** *f′*: *f′* = (λ*h*. *h* ∗ *m*) **by** *blast*
show *?thesis*
using *vector-diff-chain-within*[*OF - exp-scaleR-has-vector-derivative-right*, *of f m x s A*] *assms f′*
by (*auto simp*: *has-vector-derivative-def o-def*)
qed

context
fixes *A*::*′a*::{*banach*,*perfect-space*} *blinop*
begin

definition *linode-solution t0 x0* = (λ*t*. *exp* ((*t* − *t0*) $*_R$ *A*) *x0*)

lemma *linode-solution-solves-ode*:
(*linode-solution t0 x0 solves-ode* (λ-. *A*)) *UNIV UNIV linode-solution t0 x0 t0* = *x0*
by (*auto intro*!: *solves-odeI derivative-eq-intros*
simp: *has-vector-derivative-def blinop.bilinear-simps exp-times-scaleR-commute*

```
    has-vderiv-on-def linode-solution-def)

lemma (linode-solution t0 x0 usolves-ode (λ-. A) from t0) UNIV UNIV
  using linode-solution-solves-ode(1)
proof (rule usolves-odeI)
  fix s t1
  assume s0: s t0 = linode-solution t0 x0 t0
  assume sol: (s solves-ode (λx. blinop-apply A)) {t0−−t1} UNIV

  then have [derivative-intros]:
    (s has-derivative (λh. h *R A (s t))) (at t within {t0 −− t1}) if t ∈ {t0 −−
t1} for t
    using that
    by (auto dest!: solves-odeD(1) simp: has-vector-derivative-def has-vderiv-on-def)
  have ((λt. exp (−(t − t0) *R A) (s t)) has-derivative (λ-. 0)) (at t within {t0
−− t1})
    (is (?es has-derivative -) -)
    if t ∈ {t0 −− t1} for t
    by (auto intro!: derivative-eq-intros that simp: has-vector-derivative-def
      blinop.bilinear-simps)
  from has-derivative-zero-constant[OF convex-closed-segment this]
  obtain c where c: ⋀t. t ∈ {t0 −− t1} ⟹ ?es t = c by auto
  hence (exp ((t − t0) *R A) * (exp (−((t − t0) *R A)))) (s t) = exp ((t − t0)
*R A) c
    if t ∈ {t0 −− t1} for t
    by (metis (no-types, opaque-lifting) blinop-apply-times-blinop real-vector.scale-minus-left
that)
  then have s-def: s t = exp ((t − t0) *R A) c if t ∈ {t0 −− t1} for t
    by (simp add: exp-minus-inverse that)
  from s0 s-def
  have exp ((t0 − t0) *R A) c = x0
    by (simp add: linode-solution-solves-ode(2))
  hence c = x0 by simp
  then show s t1 = linode-solution t0 x0 t1
    using s-def[of t1] by (simp add: linode-solution-def)
qed auto

end

end
theory ODE-Analysis
imports
  Library/MVT-Ex
  IVP/Flow
  IVP/Upper-Lower-Solution
  IVP/Reachability-Analysis
  IVP/Flow-Congs
  IVP/Cones
  Library/Linear-ODE
```

begin

end

References

[1] W. Walter. *Ordinary Differential Equations.* Springer, 1 edition, 1998.