Formally Verified Computation of Enclosures of Solutions of Ordinary Differential Equations

Fabian Immler

April 17, 2016

Abstract

Ordinary differential equations (ODEs) are ubiquitous when modeling continuous dynamics. Classical numerical methods compute approximations of the solution, however without any guarantees on the quality of the approximation. Nevertheless, methods have been developed that are supposed to compute enclosures of the solution.

In this paper, we demonstrate that enclosures of the solution can be verified with a high level of rigor: We implement a functional algorithm that computes enclosures of solutions of ODEs in the interactive theorem prover Isabelle/HOL, where we formally verify (and have mechanically checked) the safety of the enclosures against the existing theory of ODEs in Isabelle/HOL.

Our algorithm works with dyadic rational numbers with statically fixed precision and is based on the well-known Euler method. We abstract discretization and round-off errors in the domain of affine forms. Code can be extracted from the verified algorithm and experiments indicate that the extracted code exhibits reasonable efficiency.

1 Relations to the paper

Here we relate the contents of our NFM 2014 paper [2] with the sources you find here. In the following list we show which notions and theorems in the paper correspond to which parts of the source code. If you are (still) interested in the relations to our ITP 2012 paper [3], you should take a look at the document of older releases (before Isabelle 2013-1) of this AFP entry.

1. Introduction
2. Background
   (a) Real numbers: Representation of real numbers with dyadic floats is set up in the separate entry Affine Arithmetic [1]
   (b) Euclidean Space: definition in image Multivariate-Analysis
   (c) Derivatives: definition in Multivariate-Analysis

(d) Notes on Taylor Series Expansion in Euclidean Space: A formal proof of a similar problem with just the mean value theorem is given in Section 2.24

(e) Ordinary Differential Equations

- Definition 1: Definition *ivp* in Section 3.3
- Definition 2: Definition *solution* in Section 3.3
- Theorem 3: In Section 3.4 resp. Section 3.4.2
- Theorem 4: In Section 11.4

3. Affine Arithmetic: see the separate entry Affine Arithmetic [1]

4. Approximation of ODEs:
Assumptions are in locales *approximate-ivp* and *approximate-sets* in Section 13

(a) Euler Step: Definitions in locale *approximate-ivp0* in Section 12
Theorem 7 and Theorem 8 are in Lemma *unique-on-euler-step*

(b) Euler Series: Definitions in locale *approximate-ivp0* in Section 12
Theorem 9 is Lemma *intervals-of-accum*

5. Experiments: Oil reservoir problem in Section 16.4, Second example in Section 16.1

6. Conclusion

Contents

1 Relations to the paper **1**

2 Auxiliary Lemmas **4**

2.1 there is no inner product for type $'a \Rightarrow_L 'b$ 5
2.2 bounded linear functions 6
2.3 Topology 10
2.4 Linorder 11
2.5 Reals 12
2.6 Vector Spaces 12
2.7 Intervals 12
2.8 Extended Real Intervals 18
2.9 Euclidean Components 19
2.10 Operator Norm 20
2.11 Limits 21
2.12 Continuity 25
2.13 Differentiability 27
2.14 Derivatives 28

2.15 Integration . . . 37
2.16 conditionally complete lattice . . . 47
2.17 Banach on type class . . . 48
2.18 Float . . . 48
2.19 Lists . . . 49
2.20 Set(sum) . . . 49
2.21 Max . . . 49
2.22 Uniform Limit . . . 49
2.23 Bounded Linear Functions . . . 50
2.24 (Counter)Example of Mean Value Theorem in Euclidean Space 50

3 Initial Value Problems **53**
3.1 Lipschitz continuity . . . 54
3.2 Local Lipschitz continuity (uniformly for a family of functions) 56
3.3 Solutions of IVPs . . . 64
3.3.1 Connecting solutions . . . 72
3.4 Picard-Lindeloef on set of functions into closed set . . . 75
3.4.1 Existence of solution . . . 81
3.4.2 Unique solution . . . 82
3.5 Picard-Lindeloef for $X = (\lambda\text{-. } UNIV)$ . . . 85
3.6 Picard-Lindeloef on cylindric domain . . . 85
3.7 Picard-Lindeloef On Open Domains . . . 95
3.7.1 Local Solution with local Lipschitz . . . 95
3.7.2 Global maximal solution with local Lipschitz . . . 103

4 Sequence of Properties on Subsequences **120**

5 Bounded Linear Operator **122**

6 Multivariate Taylor **125**
6.1 Symmetric second derivative . . . 128

7 Flow **137**
7.1 simp rules for integrability (TODO: move) . . . 137
7.2 Nonautonomous IVP on maximal existence interval . . . 138
7.3 Differentiability of the flow . . . 218

8 Linear ODE **227**

9 Target Language debug messages **229**
9.1 Printing . . . 229

10 One-Step Methods **231**
10.1 Grids . . . 231
10.2 Definition . . . 233
10.3 Consistency . . . 233
10.4 Accumulation of errors . . . 235
10.5 Consistency of order p implies convergence of order p . . . 238
10.6 Stability . . . 240
10.7 Stability via implicit error . . . 243

11 Runge-Kutta methods **245**
11.1 aux . . . 245
11.2 Definitions . . . 252
11.3 Euler method is consistent . . . 252
11.4 Set-Based Consistency of Euler Method . . . 253
11.5 Euler method is convergent . . . 267
11.6 Euler method on Rectangle is convergent . . . 267
11.7 Stability and Convergence of Approximate Euler . . . 268

12 Euler Method on Affine Forms: Code **272**
12.1 Checkpoint: Partition . . . 275

13 Euler method on Affine Forms **284**
13.1 operations on intervals . . . 285

14 Optimizations for Code Integer **312**

15 Optimizations for Code Float **313**

16 Examples **314**
16.1 Example 1 . . . 315
16.2 Example 3 . . . 318
16.2.1 Comparison with bounds analytically obtained by Walter [4] in section 9, Example V. . . . 323
16.3 Example Exponential . . . 323
16.4 Oil reservoir in Affine arithmetic . . . 330
16.5 Van der Pol oscillator . . . 353
16.6 Variational equation for the van der Pol system . . . 355

2 Auxiliary Lemmas

theory *ODE-Auxiliarities*
imports
  ~~/src/HOL/Multivariate-Analysis/Multivariate-Analysis
  ~~/src/HOL/Library/Float
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

2.1 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

2.2 bounded linear functions

locale *blinfun-syntax*
begin
no-notation *vec-nth* (**infixl** $ *90*)
notation *blinfun-apply* (**infixl** $ *999*)
end

lemma *bounded-linear-via-derivative*:
fixes f::'*a*::*real-normed-vector* ⇒ '*b*::*euclidean-space* $\Rightarrow_L$ '*c*::*real-normed-vector*
— TODO: generalize?
assumes ⋀*i*. ((λ*x*. *blinfun-apply* (*f x*) *i*) *has-derivative* (λ*x*. *f' y x i*)) (*at y*)
shows *bounded-linear* (*f' y x*)
proof −
interpret *linear f' y x*
proof (*unfold-locales*, *goal-cases*)
case (*1 v w*)
from *has-derivative-unique*[*OF assms*[*of v* + *w*, *unfolded blinfun.bilinear-simps*]
has-derivative-add[*OF assms*[*of v*] *assms*[*of w*]], *THEN fun-cong*, *of x*]
show *?case* **.**
next
case (*2 r v*)
from *has-derivative-unique*[*OF assms*[*of r* $*_R$ *v*, *unfolded blinfun.bilinear-simps*]
has-derivative-scaleR-right[*OF assms*[*of v*], *of r*], *THEN fun-cong*, *of x*]
show *?case* **.**
qed
let *?bnd* = ∑*i*∈*Basis*. *norm* (*f' y x i*)
{
fix *v*
have *f' y x v* = (∑*i*∈*Basis*. (*v* · *i*) $*_R$ *f' y x i*)
by (*subst euclidean-representation*[*symmetric*]) (*simp add*: *setsum scaleR*)
also have *norm* ... ≤ *norm v* ∗ *?bnd*
by (*auto intro*!: *order.trans*[*OF norm-setsum*] *setsum-mono mult-right-mono*
simp: *setsum-right-distrib Basis-le-norm*)
finally have *norm* (*f' y x v*) ≤ *norm v* ∗ *?bnd* **.**
}
then show *?thesis* **by** *unfold-locales auto*
qed

definition *blinfun-scaleR*::('*a*::*real-normed-vector* $\Rightarrow_L$ *real*) ⇒ '*b*::*real-normed-vector*
⇒ ('*a* $\Rightarrow_L$ '*b*)
where *blinfun-scaleR a b* = *blinfun-scaleR-left b* o_L *a*

lemma *blinfun-scaleR-transfer*[*transfer-rule*]:
rel-fun (*pcr-blinfun op* = *op* =) (*rel-fun op* = (*pcr-blinfun op* = *op* =))

$(\lambda a\ b\ c.\ a\ c *_R b)$ *blinfun-scaleR*
by (*auto simp*: *blinfun-scaleR-def rel-fun-def pcr-blinfun-def cr-blinfun-def OO-def*)

lemma *blinfun-scaleR-rep-eq*[*simp*]:
blinfun-scaleR a b c = $a\ c *_R b$
by (*simp add*: *blinfun-scaleR-def*)

lemma *bounded-linear-blinfun-scaleR*: *bounded-linear* (*blinfun-scaleR a*)
unfolding *blinfun-scaleR-def*[*abs-def*]
by (*auto intro*!: *bounded-linear-intros*)

lemma *blinfun-scaleR-has-derivative*[*derivative-intros*]:
assumes (*f has-derivative f'*) (*at x within s*)
shows ((λx. *blinfun-scaleR a* (*f x*)) *has-derivative* (λx. *blinfun-scaleR a* (*f' x*))) (*at x within s*)
using *bounded-linear-blinfun-scaleR assms*
by (*rule bounded-linear.has-derivative*)

lemma *blinfun-componentwise*:
fixes *f*::'*a*::*real-normed-vector* ⇒ '*b*::*euclidean-space* $\Rightarrow_L$ '*c*::*real-normed-vector*
shows *f* = (λx. $\sum$ *i*∈*Basis*. *blinfun-scaleR* (*blinfun-inner-left i*) (*f x i*))
by (*auto intro*!: *blinfun-eqI*
simp: *blinfun.setsum-left euclidean-representation blinfun.scaleR-right*[*symmetric*]
blinfun.setsum-right[*symmetric*])

lemma
blinfun-has-derivative-componentwiseI:
fixes *f*::'*a*::*real-normed-vector* ⇒ '*b*::*euclidean-space* $\Rightarrow_L$ '*c*::*real-normed-vector*
assumes $\bigwedge$*i*. *i* ∈ *Basis* ⟹ ((λx. *f x i*) *has-derivative blinfun-apply* (*f' i*)) (*at x*)
shows (*f has-derivative* (λx. $\sum$ *i*∈*Basis*. *blinfun-scaleR* (*blinfun-inner-left i*) (*f' i x*))) (*at x*)
by (*subst blinfun-componentwise*) (*force intro*: *derivative-eq-intros assms simp*: *blinfun.bilinear-simps*)

lemma
has-derivative-BlinfunI:
fixes *f*::'*a*::*real-normed-vector* ⇒ '*b*::*euclidean-space* $\Rightarrow_L$ '*c*::*real-normed-vector*
assumes $\bigwedge$*i*. ((λx. *f x i*) *has-derivative* (λx. *f' y x i*)) (*at y*)
shows (*f has-derivative* (λx. *Blinfun* (*f' y x*))) (*at y*)
proof −
have *1*: *f* = (λx. $\sum$ *i*∈*Basis*. *blinfun-scaleR* (*blinfun-inner-left i*) (*f x i*))
by (*rule blinfun-componentwise*)
moreover have *2*: (... *has-derivative* (λx. $\sum$ *i*∈*Basis*. *blinfun-scaleR* (*blinfun-inner-left i*) (*f' y x i*))) (*at y*)
by (*force intro*: *assms derivative-eq-intros*)
moreover
interpret *f'*: *bounded-linear f' y x* **for** *x*
by (*rule bounded-linear-via-derivative*) (*rule assms*)

```
  have 3: (∑ i∈Basis. blinfun-scaleR (blinfun-inner-left i) (f' y x i)) i = f' y x i
for x i
    by (auto simp: if-distrib cond-application-beta blinfun.bilinear-simps
      f'.scaleR[symmetric] f'.setsum[symmetric] euclidean-representation
      intro!: blinfun-euclidean-eqI)
  have 4: blinfun-apply (Blinfun (f' y x)) = f' y x for x
    apply (subst bounded-linear-Blinfun-apply)
    subgoal by unfold-locales
    subgoal by simp
    done
  show ?thesis
    apply (subst 1)
    apply (rule 2[THEN has-derivative-eq-rhs])
    apply (rule ext)
    apply (rule blinfun-eqI)
    apply (subst 3)
    apply (subst 4)
    apply (rule refl)
    done
qed
```

TODO: use this to replace *op has-derivative*

```
lift-definition has-bderivative ::
  ('a::real-normed-vector ⇒ 'b::real-normed-vector) ⇒ ('a ⇒L 'b) ⇒ 'a filter ⇒
bool
  (infix (has'-bderivative) 50)
  is op has-derivative .

lemma has-bderivative-const: ((λx. c) has-bderivative 0) F
  apply transfer'
  apply (rule has-derivative-const)
  done

lemma has-bderivative-id: ((λx. x) has-bderivative id-blinfun) F
  apply transfer'
  apply (rule has-derivative-id)
  done

context bounded-bilinear
begin

lemma bderivative:
  assumes (f has-bderivative f') (at x within s)
    and (g has-bderivative g') (at x within s)
  shows
    ((λx. prod (f x) (g x)) has-bderivative (prod-right (f x) oL g') + (prod-left (g
x) oL f'))
      (at x within s)
  using assms
```

by *transfer* (*auto intro*!: *derivative-eq-intros FDERIV*)

end

lemmas *has-bderivative-eq-rhs* = *has-derivative-eq-rhs*[*Transfer.transferred*]

lemma *has-bderivative-scaleR-left*:
fixes *g*::'*a*::*real-normed-vector* ⇒ *real* **and** *x*::'*b*::*real-normed-vector*
assumes (*g has-bderivative g'*) *F*
shows ((λ*xa*. *g xa* $*_R$ *x*) *has-bderivative blinfun-scaleR g' x*) *F*
using *assms*
by *transfer'* (*auto intro*!: *derivative-eq-intros*)

lemma *has-bderivative-scaleR-right*:
assumes (*g has-bderivative g'*) *F*
shows ((λ*xa*. *x* $*_R$ *g xa*) *has-bderivative x* $*_R$ *g'*) *F*
using *assms*
by *transfer'* (*rule has-derivative-scaleR-right*)

lemma *has-bderivative-scaleR*:
assumes (*f has-bderivative f'*) (*at x within s*)
assumes (*g has-bderivative g'*) (*at x within s*)
shows ((λ*x*. *f x* $*_R$ *g x*) *has-bderivative f x* $*_R$ *g'* + *blinfun-scaleR f'* (*g x*)) (*at x within s*)
using *assms*
by *transfer'* (*auto intro*!: *derivative-eq-intros*)

lemma *has-bderivative-divide*:
assumes (*f has-bderivative f'*) (*at x within s*)
and (*g has-bderivative g'*) (*at x within s*)
and *g x* ≠ *0*
shows
((λ*x*. *f x* / *g x*) *has-bderivative*
(*blinfun-scaleR f'* (*g x*) − *f x* $*_R$ *g'*) $/_R$ (*g x* * *g x*))
(*at x within s*)
using *assms*
by *transfer'* (*auto intro*!: *derivative-eq-intros simp*: *field-simps*)

lemma
has-derivative-Blinfun:
assumes (*f has-derivative f'*) *F*
shows (*f has-derivative Blinfun f'*) *F*
using *assms*
by (*subst bounded-linear-Blinfun-apply*) *auto*

lift-definition *swap2-blinfun*::
('*a*::*real-normed-vector* $\Rightarrow_L$ '*b*::*real-normed-vector* $\Rightarrow_L$ '*c*::*real-normed-vector*) ⇒ '*b* $\Rightarrow_L$ '*a* $\Rightarrow_L$ '*c* **is**

$\lambda f\ x\ y.\ f\ y\ x$
using *bounded-bilinear.bounded-linear-left bounded-bilinear.bounded-linear-right bounded-bilinear.flip*
by *auto*

lemma *swap2-blinfun-apply*[*simp*]: *swap2-blinfun f a b = f b a*
by *transfer simp*

2.3 Topology

lemma *at-within-ball*: *e > 0* ⟹ *dist x y < e* ⟹ *at y within ball x e = at y*
by (*subst at-within-open*) *auto*

lemma
infdist-attains-inf:
fixes *X*::*'a*::*heine-borel set*
assumes *closed X*
assumes *X ≠ {}*
obtains *x* **where** *x ∈ X infdist y X = dist y x*
proof −
have *bdd-below* (*dist y ` X*)
by *auto*
from *distance-attains-inf*[*OF assms*, *of y*]
obtain *x* **where** *INF*: *x ∈ X* ⋀*z. z ∈ X* ⟹ *dist y x ≤ dist y z* **by** *auto*
have *infdist y X = dist y x*
by (*auto simp*: *infdist-def assms*
intro!: *antisym cINF-lower*[*OF - ‹x ∈ X›*] *cINF-greatest*[*OF assms*(*2*) *INF*(*2*)])
with *‹x ∈ X›* **show** *?thesis* **..**
qed

lemma *compact-infdist-le*:
fixes *A*::*'a*::*heine-borel set*
assumes *A ≠ {}*
assumes *compact A*
assumes *e > 0*
shows *compact* {*x. infdist x A ≤ e*}
proof −
from *continuous-closed-vimage*[*of* λ*x. infdist x A* {*0..e*}]
continuous-infdist[*OF continuous-ident*, *of - UNIV A*]
have *closed* {*x. infdist x A ≤ e*} **by** (*auto simp*: *vimage-def infdist-nonneg*)
moreover
from *assms* **obtain** *x0 b* **where** *b*: ⋀*x. x ∈ A* ⟹ *dist x0 x ≤ b closed A*
by (*auto simp*: *compact-eq-bounded-closed bounded-def*)
{
fix *y*
assume *le*: *infdist y A ≤ e*
from *infdist-attains-inf*[*OF ‹closed A› ‹A ≠ {}›*, *of y*]
obtain *z* **where** *z*: *z ∈ A infdist y A = dist y z* **by** *blast*
have *dist x0 y ≤ dist y z + dist x0 z*

by (*metis dist-commute dist-triangle*)
also have *dist y z* ≤ *e* **using** *le z* **by** *simp*
also have *dist x0 z* ≤ *b* **using** *b z* **by** *simp*
finally have *dist x0 y* ≤ *b* + *e* **by** *arith*
} then
have *bounded* {*x*. *infdist x A* ≤ *e*}
by (*auto simp*: *bounded-any-center*[**where** *a*=*x0*] *intro*!: *exI*[**where** *x*=*b* + *e*])
ultimately show *compact* {*x*. *infdist x A* ≤ *e*}
by (*simp add*: *compact-eq-bounded-closed*)
qed

lemma *compact-in-open-separated*:
fixes *A*::′*a*::*heine-borel set*
assumes *A* ≠ {}
assumes *compact A*
assumes *open B*
assumes *A* ⊆ *B*
obtains *e* **where** *e* > *0* {*x*. *infdist x A* ≤ *e*} ⊆ *B*
proof *atomize-elim*
have *closed* (− *B*) *compact A* − *B* ∩ *A* = {}
using *assms* **by** (*auto simp*: *open-Diff compact-eq-bounded-closed*)
from *separate-closed-compact*[*OF this*]
obtain *d*′::*real* **where** *d*′: *d*′>*0* ⋀*x y*. *x* ∉ *B* ⟹ *y* ∈ *A* ⟹ *d*′ ≤ *dist x y*
by *auto*
def *d* ≡ *d*′ / *2*
hence *d*>*0* *d* < *d*′ **using** *d*′ **by** *auto*
with *d*′ **have** *d*: ⋀*x y*. *x* ∉ *B* ⟹ *y* ∈ *A* ⟹ *d* < *dist x y*
by *force*
show ∃ *e*>*0*. {*x*. *infdist x A* ≤ *e*} ⊆ *B*
proof (*rule ccontr*)
assume ∄ *e*. *0* < *e* ∧ {*x*. *infdist x A* ≤ *e*} ⊆ *B*
with ‹*d* > *0*› **obtain** *x* **where** *x*: *infdist x A* ≤ *d x* ∉ *B*
by *auto*
from *assms* **have** *closed A A* ≠ {} **by** (*auto simp*: *compact-eq-bounded-closed*)
from *infdist-attains-inf*[*OF this*]
obtain *y* **where** *y*: *y* ∈ *A infdist x A* = *dist x y*
by *auto*
have *dist x y* ≤ *d* **using** *x y* **by** *simp*
also have ... < *dist x y* **using** *y d x* **by** *auto*
finally show *False* **by** *simp*
qed
qed

2.4 Linorder

context *linordered-idom*
begin

lemma *mult-left-le-one-le*:

$0 \le x \Longrightarrow y \le 1 \Longrightarrow y * x \le x$
by (*auto simp add*: *mult-le-cancel-right2*)

lemma *mult-le-oneI*: $0 \le a \wedge a \le 1 \wedge b \le 1 \Longrightarrow a * b \le 1$
using *local.dual-order.trans local.mult-left-le* **by** *blast*

end

2.5 Reals

2.6 Vector Spaces

lemma *scaleR-dist-distrib-left*:
fixes *b c*::*'a*::*real-normed-vector*
shows *abs a* ∗ *dist b c* = *dist* (*scaleR a b*) (*scaleR a c*)
unfolding *dist-norm scaleR-diff-right*[*symmetric*] *norm-scaleR* **..**

lemma *scaleR-dist-distrib-right*:
fixes *a*::*'a*::*real-normed-vector*
shows *norm a* ∗ *dist b c* = *dist* (*scaleR b a*) (*scaleR c a*)
unfolding *dist-norm scaleR-diff-left*[*symmetric*] *norm-scaleR*
by *simp*

lemma *ex-norm-eq-1*: ∃ *x*. *norm* (*x*::*'a*::*euclidean-space*) = *1*
by (*metis vector-choose-size zero-le-one*)

lemma *open-neg-translation*:
fixes *s* :: *'a*::*real-normed-vector set*
assumes *open s*
shows *open*((λ*x*. *a* − *x*) ' *s*)
using *open-translation*[*OF open-negations*[*OF assms*], *of a*]
by (*auto simp*: *image-image*)

2.7 Intervals

lemma *open-closed-segment-subset*: *open-segment a b* ⊆ *closed-segment a b*
by (*simp add*: *open-closed-segment subsetI*)

lemma *is-interval-real-cball*[*simp*]:
fixes *a b*::*real*
shows *is-interval* (*cball a b*)
by (*auto simp*: *is-interval-convex-1 convex-cball*)

lemma *atLeastAtMost-eq-cball*:
fixes *a b*::*real*
shows {*a* .. *b*} = *cball* ((*a* + *b*)/*2*) ((*b* − *a*)/*2*)
by (*auto simp*: *dist-real-def field-simps*)

lemma *greaterThanLessThan-eq-ball*:
fixes *a b*::*real*

shows $\{a <..< b\} = ball\ ((a + b)/2)\ ((b - a)/2)$
by (*auto simp*: *dist-real-def field-simps*)

lemma *closure-greaterThanLessThan*[*simp*]:
fixes *a b*::*real*
shows $a < b \Longrightarrow closure\ \{a <..< b\} = \{a .. b\}$
by (*simp add*: *closure-ball greaterThanLessThan-eq-ball atLeastAtMost-eq-cball*)

lemma *image-mult-atLeastAtMost*:
$(\lambda x.\ x * c::real)$ ' $\{x..y\} = (if\ x \leq y\ then\ if\ c > 0\ then\ \{x * c\ ..\ y * c\}\ else\ \{y * c\ ..\ x * c\}\ else\ \{\})$
apply (*cases* $c = 0$)
apply *force*
apply (*auto simp*: *field-simps not-less intro*!: *image-eqI*[**where** *x*=*inverse c* ∗ *xa* **for** *xa*])
done

lemma *image-add-atLeastAtMost*:
op + *c* ' $\{x..y::real\} = \{c + x\ ..\ c + y\}$
by (*auto intro*: *image-eqI*[**where** *x*=*xa* − *c* **for** *xa*])

lemma *min-zero-mult-nonneg-le*: $0 \leq h' \Longrightarrow h' \leq h \Longrightarrow min\ 0\ (h * k::real) \leq h' * k$
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*: $0 \leq h' \Longrightarrow h' \leq h \Longrightarrow h' * k \leq max\ 0\ (h * k::real)$
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-real* = *closed-segment-eq-real-ivl*

lemma *open-segment-real-le*:
fixes *a b*::*real*
assumes $a \leq b$
shows *open-segment a b* = $\{a <..< b\}$
using *assms*
unfolding *open-segment-def closed-segment-real*
by *auto*

lemma *open-segment-real*:
fixes *a b*::*real*
shows *open-segment a b* = $(if\ a \leq b\ then\ \{a <..< b\}\ else\ \{b <..< a\})$
using *open-segment-real-le*[*of a b*]
open-segment-real-le[*of b a*]
open-segment-commute[*of b a*]
by *simp*

lemma *linear-compose*: $(\lambda xa.\ a + xa * b) = (\lambda x.\ a + x)\ o\ (\lambda x.\ x * b)$
  by *auto*

lemma *image-linear-atLeastAtMost*:
  $(\lambda xa.\ a + xa * b)$ ‘ $\{c..d::real\}$ =
  ($if\ c \leq d\ then$
    $if\ b > 0\ then\ \{a + c * b\ ..\ a + d * b\}$
    $else\ \{a + d * b\ ..\ a + c * b\}$
  $else\ \{\}$)
  by (*simp add*: *linear-compose image-comp* [*symmetric*] *image-mult-atLeastAtMost*
    image-add-atLeastAtMost)

lemma *insert-atMost*[*simp*]: *insert t* $\{..t::'a::preorder\} = \{..t\}$ **by** *auto*

lemma *insert-atLeastAtMost*[*simp*]:
  $s \geq 0 \Longrightarrow$ *insert t* $\{t..s + t::'a::ordered\text{-}ab\text{-}group\text{-}add\} = \{t\ ..\ s + t\}$ **by** *auto*

lemma *uminus-uminus-image*[*simp*]:
  fixes $x::'a::group\text{-}add\ set$
  shows *uminus* ‘ *uminus* ‘ $x = x$
  by *force*

lemma *Ball-singleton*: *Ball* $\{x\}\ f = f\ x$
  by *simp*

lemma *is-real-interval-union*:
  fixes $X\ Y::real\ set$
  shows *is-interval* $X \Longrightarrow$
    is-interval $Y \Longrightarrow$
    $(X \neq \{\} \Longrightarrow Y \neq \{\} \Longrightarrow X \cap Y \neq \{\}) \Longrightarrow$
    is-interval $(X \cup Y)$
  unfolding *is-interval-def Basis-real-def Ball-singleton real-inner-1-right*
  by (*safe*; *metis* (*mono-tags*) *all-not-in-conv disjoint-iff-not-equal le-cases*)

lemma *is-interval-translationI*:
  assumes *is-interval* X
  shows *is-interval* (*op* + x ‘ X)
  unfolding *is-interval-def*
proof *safe*
  fix $b\ d\ e$
  assume $b \in X\ d \in X$
    $\forall i \in Basis.\ (x + b) \cdot i \leq e \cdot i \wedge e \cdot i \leq (x + d) \cdot i \vee$
      $(x + d) \cdot i \leq e \cdot i \wedge e \cdot i \leq (x + b) \cdot i$
  hence $e - x \in X$
    by (*intro mem-is-intervalI*[*OF assms* ‹$b \in X$› ‹$d \in X$›, *of* $e - x$])
      (*auto simp*: *algebra-simps*)
  thus $e \in$ *op* + x ‘ X **by** *force*
qed

lemma *is-interval-uminusI*:
  assumes *is-interval X*
  shows *is-interval* (*uminus* ‘ *X*)
  unfolding *is-interval-def*
proof *safe*
  fix *b d e*
  assume $b \in X$ $d \in X$
    $\forall\, i{\in}Basis.\ (-\ b) \cdot i \le e \cdot i \wedge e \cdot i \le (-\ d) \cdot i \vee$
      $(-\ d) \cdot i \le e \cdot i \wedge e \cdot i \le (-\ b) \cdot i$
  hence $-\ e \in X$
    by (*intro mem-is-intervalI*[*OF assms* ‹$b \in X$› ‹$d \in X$›, *of* $-\ e$])
      (*auto simp*: *algebra-simps*)
  thus $e \in$ *uminus* ‘ *X* **by** *force*
qed

lemma *is-interval-uminus*[*simp*]: *is-interval* (*uminus* ‘ *x*) = *is-interval x*
  using *is-interval-uminusI*[*of x*] *is-interval-uminusI*[*of uminus* ‘ *x*]
  by *auto*

lemma *is-interval-neg-translationI*:
  assumes *is-interval X*
  shows *is-interval* (*op* − *x* ‘ *X*)
proof −
  have *op* − *x* ‘ *X* = *op* + *x* ‘ *uminus* ‘ *X*
    by (*force simp*: *algebra-simps*)
  also have *is-interval* . . .
    by (*metis is-interval-uminusI is-interval-translationI assms*)
  finally show *?thesis* **.**
qed

lemma *is-interval-translation*[*simp*]:
  is-interval (*op* + *x* ‘ *X*) = *is-interval X*
  using *is-interval-neg-translationI*[*of op* + *x* ‘ *X x*]
  by (*auto intro*!: *is-interval-translationI simp*: *image-image*)

lemma *is-interval-minus-translation*[*simp*]:
  shows *is-interval* (*op* − *x* ‘ *X*) = *is-interval X*
proof −
  have *op* − *x* ‘ *X* = *op* + *x* ‘ *uminus* ‘ *X*
    by (*force simp*: *algebra-simps*)
  also have *is-interval* . . . = *is-interval X*
    by *simp*
  finally show *?thesis* **.**
qed

lemma *is-interval-minus-translation'*[*simp*]:
  shows *is-interval* (($\lambda x.\ x - c$) ‘ *X*) = *is-interval X*
  using *is-interval-translation*[*of* −*c X*]
  by (*metis image-cong uminus-add-conv-diff*)

lemma
  fixes *a*::′*a*::*ordered-euclidean-space*
  shows *is-interval-ci*: *is-interval* {*a*..}
    and *is-interval-ic*: *is-interval* {..*a*}
  by (*force simp*: *is-interval-def eucl-le*[**where** ′*a*=′*a*])+

lemma *image-add-atLeast-real*[*simp*]:
  fixes *a b c*::′*a*::*ordered-real-vector*
  shows *op* + *c* ‘ {*a*..} = {*c* + *a*..}
  by (*auto intro*!: *image-eqI*[**where** *x*=*x* − *c* **for** *x*] *simp*: *algebra-simps*)

lemma *image-add-atMost-real*[*simp*]:
  fixes *a b c*::′*a*::*ordered-real-vector*
  shows *op* + *c* ‘ {..*a*} = {..*c* + *a*}
  by (*auto intro*!: *image-eqI*[**where** *x*=*x* − *c* **for** *x*] *simp*: *algebra-simps*)

lemma *image-add-atLeastLessThan-real*[*simp*]:
  fixes *a b c*::′*a*::*ordered-real-vector*
  shows *op* + *c* ‘ {*a*..<*b*} = {*c* + *a*..<*c* + *b*}
  by (*auto intro*!: *image-eqI*[**where** *x*=*x* − *c* **for** *x*] *simp*: *algebra-simps*)

lemma *image-add-greaterThanAtMost-real*[*simp*]:
  fixes *a b c*::′*a*::*ordered-real-vector*
  shows *op* + *c* ‘ {*a*<..*b*} = {*c* + *a*<..*c* + *b*}
  by (*auto intro*!: *image-eqI*[**where** *x*=*x* − *c* **for** *x*] *simp*: *algebra-simps*)

lemma *image-minus-const-atLeastLessThan-real*[*simp*]:
  fixes *a b c*::′*a*::*ordered-real-vector*
  shows *op* − *c* ‘ {*a*..<*b*} = {*c* − *b*<..*c* − *a*}
proof −
  have *op* − *c* ‘ {*a*..<*b*} = *op* + *c* ‘ *uminus* ‘ {*a* ..<*b*}
    unfolding *image-image* **by** *simp*
  also have . . . = {*c* − *b*<..*c* − *a*} **by** *simp*
  finally show *?thesis* **by** *simp*
qed

lemma *image-minus-const-greaterThanAtMost-real*[*simp*]:
  fixes *a b c*::′*a*::*ordered-real-vector*
  shows *op* − *c* ‘ {*a*<..*b*} = {*c* − *b*..<*c* − *a*}
proof −
  have *op* − *c* ‘ {*a*<..*b*} = *op* + *c* ‘ *uminus* ‘ {*a*<..*b*}
    unfolding *image-image* **by** *simp*
  also have . . . = {*c* − *b*..<*c* − *a*} **by** *simp*
  finally show *?thesis* **by** *simp*
qed

lemma *image-minus-const-atLeast-real*[*simp*]:

fixes *a c*::′*a*::*ordered-real-vector*
shows *op* − *c* ‘ {*a*..} = {..*c* − *a*}
proof −
have *op* − *c* ‘ {*a*..} = *op* + *c* ‘ *uminus* ‘ {*a* ..}
unfolding *image-image* **by** *simp*
also have . . . = {..*c* − *a*} **by** *simp*
finally show *?thesis* **by** *simp*
qed

lemma *image-minus-const-AtMost-real*[*simp*]:
fixes *b c*::′*a*::*ordered-real-vector*
shows *op* − *c* ‘ {..*b*} = {*c* − *b*..}
proof −
have *op* − *c* ‘ {..*b*} = *op* + *c* ‘ *uminus* ‘ {..*b*}
unfolding *image-image* **by** *simp*
also have . . . = {*c* − *b*..} **by** *simp*
finally show *?thesis* **by** *simp*
qed

lemma *interior-atLeastAtMost*:
fixes *a b*::*real*
assumes *a* < *b*
shows *interior* {*a* .. *b*} = {*a* <..< *b*}
by (*metis assms closure-greaterThanLessThan convex-interior-closure convex-real-interval*(*8*) *interior-open open-greaterThanLessThan*)

lemma *is-interval-Ioo*:
fixes *x*::*real* **shows** *is-interval* {*x*<..<*y*}
by (*metis connected-Ioo is-interval-connected-1*)

lemma *is-interval-Ioi*:
fixes *x*::*real* **shows** *is-interval* {*x*<..}
by (*metis connected-Ioi is-interval-connected-1*)

lemma *is-interval-Iio*:
fixes *x*::*real* **shows** *is-interval* {..<*x*}
by (*metis connected-Iio is-interval-connected-1*)

lemma *is-interval-inter*: *is-interval X* ⟹ *is-interval Y* ⟹ *is-interval* (*X* ∩ *Y*)
unfolding *is-interval-def*
by *blast*

lemma *cball-trans*: *y* ∈ *cball z b* ⟹ *x* ∈ *cball y a* ⟹ *x* ∈ *cball z* (*b* + *a*)
unfolding *mem-cball*
proof −
have *dist z x* ≤ *dist z y* + *dist y x*
by (*rule dist-triangle*)
also assume *dist z y* ≤ *b*
also assume *dist y x* ≤ *a*

finally show *dist z x* $\leq$ *b* + *a* **by** *arith*
qed

2.8 Extended Real Intervals

lemma *open-real-image*:
fixes *X*::*ereal set*
assumes *open X*
assumes $\infty \notin X$
assumes $-\infty \notin X$
shows *open* (*real-of-ereal* ‘ *X*)
proof −
have *real-of-ereal* ‘ *X* = *ereal* −‘ *X*
apply (*auto simp*:)
apply (*metis assms*(*2*) *assms*(*3*) *ereal-infinity-cases ereal-real'*)
using *image-iff* **by** *fastforce*
thus *?thesis*
by (*auto intro*!: *open-ereal-vimage assms*)
qed

lemma *real-greaterThanLessThan-infinity-eq*:
real-of-ereal ‘ {*N*::*ereal*<..<∞} =
(*if N* = ∞ *then* {} *else if N* = −∞ *then UNIV else* {*real-of-ereal N*<..})
proof −
{
fix *x*::*real*
have *x* ∈ *real-of-ereal* ‘ {− ∞<..<∞::*ereal*}
by (*auto intro*!: *image-eqI*[**where** *x*=*ereal x*])
} moreover {
fix *x*::*ereal*
assume *N* ≠ − ∞ *N* < *x x* ≠ ∞
then have *real-of-ereal N* < *real-of-ereal x*
by (*cases N*; *cases x*; *simp*)
} moreover {
fix *x*::*real*
assume *N* ≠ ∞ *real-of-ereal N* < *x*
then have *x* ∈ *real-of-ereal* ‘ {*N*<..<∞}
by (*cases N*) (*auto intro*!: *image-eqI*[**where** *x*=*ereal x*])
} ultimately show *?thesis* **by** *auto*
qed

lemma *real-greaterThanLessThan-minus-infinity-eq*:
real-of-ereal ‘ {−∞<..<*N*::*ereal*} =
(*if N* = ∞ *then UNIV else if N* = −∞ *then* {} *else* {..<*real-of-ereal N*})
proof −
have *real-of-ereal* ‘ {−∞<..<*N*::*ereal*} = *uminus* ‘ *real-of-ereal* ‘ {−*N*<..<∞}
by (*auto simp*: *ereal-uminus-less-reorder intro*!: *image-eqI*[**where** *x*=−*x* **for** *x*])
also note *real-greaterThanLessThan-infinity-eq*

finally show *?thesis* **by** (*auto intro*!: *image-eqI*[**where** $x=-x$ **for** x])
qed

lemma *real-greaterThanLessThan-inter*:
real-of-ereal ‘ $\{N<..<M::ereal\}$ = *real-of-ereal* ‘ $\{-\infty<..<M\}$ ∩ *real-of-ereal* ‘ $\{N<..<\infty\}$
apply (*auto intro*!: *image-eqI*)
by (*metis ereal-infinity-cases ereal-infty-less*(*2*) *ereal-less-eq*(*1*)
ereal-real′ less-trans not-le)

lemma *real-atLeastGreaterThan-eq*: *real-of-ereal* ‘ $\{N<..<M::ereal\}$ =
(*if* $N = \infty$ *then* {} *else*
if $N = -\infty$ *then*
(*if* $M = \infty$ *then UNIV*
else if $M = -\infty$ *then* {}
else {..< *real-of-ereal M*})
else if $M = - \infty$ *then* {}
else if $M = \infty$ *then* {*real-of-ereal N*<..}
else {*real-of-ereal N* <..< *real-of-ereal M*})
apply (*subst real-greaterThanLessThan-inter*)
apply (*subst real-greaterThanLessThan-minus-infinity-eq*)
apply (*subst real-greaterThanLessThan-infinity-eq*)
apply *auto*
done

lemma *is-interval-real-ereal-oo*: *is-interval* (*real-of-ereal* ‘ $\{N<..<M::ereal\}$)
by (*auto simp*: *real-atLeastGreaterThan-eq is-interval-empty is-interval-univ*
is-interval-Ioo is-interval-Iio is-interval-Ioi)

lemma *is-interval-ball-real*: **fixes** *a b*::*real* **shows** *is-interval* (*ball a b*)
by (*metis connected-ball is-interval-connected-1*)

lemma *real-image-ereal-ivl*:
fixes *a b*::*ereal*
shows
real-of-ereal ‘ $\{a<..<b\}$ =
(*if* $a < b$ *then* (*if* $a = - \infty$ *then if* $b = \infty$ *then UNIV else* {..<*real-of-ereal b*}
else if $b = \infty$ *then* {*real-of-ereal a*<..} *else* {*real-of-ereal a* <..< *real-of-ereal b*})
else {})
by (*cases a*; *cases b*; *simp add*: *real-atLeastGreaterThan-eq not-less*)

2.9 Euclidean Components

lemma *sqrt-le-rsquare*:
assumes $|x| \leq$ *sqrt y*
shows $x^2 \leq y$
using *assms real-sqrt-le-iff*[*of* x^2] **by** *simp*

lemma *setsum-ge-element*:

```
  fixes f::'a ⇒ ('b::ordered-comm-monoid-add)
  assumes finite s
  assumes i ∈ s
  assumes ⋀i. i ∈ s ⟹ f i ≥ 0
  assumes el = f i
  shows el ≤ setsum f s
proof −
  have el = setsum f {i} by (simp add: assms)
  also have ... ≤ setsum f s using assms by (intro setsum-mono2) auto
  finally show ?thesis .
qed

lemma norm-nth-le:
  fixes x::'a::euclidean-space
  assumes i ∈ Basis
  shows norm (x · i) ≤ norm x
  unfolding norm-conv-dist euclidean-dist-l2[of x] setL2-def
  by (auto intro!: real-le-rsqrt setsum-ge-element assms)

lemma norm-Pair-le:
  shows norm (x, y) ≤ norm x + norm y
  unfolding norm-Pair
  by (metis norm-ge-zero sqrt-sum-squares-le-sum)

lemma norm-Pair-ge1:
  shows norm x ≤ norm (x, y)
  unfolding norm-Pair
  by (metis real-sqrt-sum-squares-ge1)

lemma norm-Pair-ge2:
  shows norm y ≤ norm (x, y)
  unfolding norm-Pair
  by (metis real-sqrt-sum-squares-ge2)
```

2.10 Operator Norm

```
lemma onorm-setsum-le:
  assumes finite S
  assumes ⋀s. s ∈ S ⟹ bounded-linear (f s)
  shows onorm (λx. setsum (λs. f s x) S) ≤ setsum (λs. onorm (f s)) S
  using assms
 by (induction) (auto simp: onorm-zero intro!: onorm-triangle-le bounded-linear-setsum)

lemma onorm-componentwise:
  assumes bounded-linear f
  shows onorm f ≤ (∑i∈Basis. norm (f i))
proof −
  {
    fix i::'a
```

assume *i* ∈ *Basis*
hence *onorm* (λ*x*. (*x* · *i*) $*_R$ *f i*) ≤ *onorm* (λ*x*. (*x* · *i*)) ∗ *norm* (*f i*)
by (*auto intro*!: *onorm-scaleR-left-lemma*)
also have . . . ≤ *norm i* ∗ *norm* (*f i*)
by (*rule mult-right-mono*)
(*auto simp*: *ac-simps Cauchy-Schwarz-ineq2 intro*!: *onorm-le*)
finally have *onorm* (λ*x*. (*x* · *i*) $*_R$ *f i*) ≤ *norm* (*f i*) **using** ‹*i* ∈ *Basis*›
by *simp*
} **hence** *onorm* (λ*x*. ∑ *i*∈*Basis*. (*x* · *i*) $*_R$ *f i*) ≤ (∑ *i*∈*Basis*. *norm* (*f i*))
by (*auto intro*!: *order-trans*[*OF onorm-setsum-le*] *bounded-linear-scaleR-const setsum-mono*)
also have (λ*x*. ∑ *i*∈*Basis*. (*x* · *i*) $*_R$ *f i*) = (λ*x*. *f* (∑ *i*∈*Basis*. (*x* · *i*) $*_R$ *i*))
by (*simp add*: *linear-setsum bounded-linear*.*linear assms linear-simps*)
also have . . . = *f*
by (*simp add*: *euclidean-representation*)
finally show *?thesis* **.**
qed

lemmas *onorm-componentwise-le* = *order-trans*[*OF onorm-componentwise*]

2.11 Limits

lemma *Zfun-ident*: *Zfun* (λ*x*::′*a*::*real-normed-vector*. *x*) (*at 0*)
using *tendsto-ident-at*[*of 0*::′*a UNIV*, *simplified tendsto-Zfun-iff*]
by *simp*

lemma *not-in-closure-trivial-limitI*:
x ∉ *closure s* ⟹ *trivial-limit* (*at x within s*)
using *not-trivial-limit-within*[*of x s*]
apply *auto*
by (*metis Diff-empty Diff-insert0 closure-subset contra-subsetD*)

lemma *tendsto-If*:
assumes *tendsto*:
x ∈ *s* ∪ (*closure s* ∩ *closure t*) ⟹
(*f* ⟶ *l x*) (*at x within s* ∪ (*closure s* ∩ *closure t*))
x ∈ *t* ∪ (*closure s* ∩ *closure t*) ⟹
(*g* ⟶ *l x*) (*at x within t* ∪ (*closure s* ∩ *closure t*))
assumes *x* ∈ *s* ∪ *t*
shows ((λ*x*. *if x* ∈ *s then f x else g x*) ⟶ *l x*) (*at x within s* ∪ *t*)
proof (*rule Lim-Un*, *safe intro*!: *topological-tendstoI*)
fix *S*::′*b set*
assume *S*: *open S*
assume *l*: *l x* ∈ *S*
let *?thesis* =
λ*t*. *eventually* (λ*x*. (*if x* ∈ *s then f x else g x*) ∈ *S*) (*at x within t*)
{
assume *x*: *x* ∈ *s* **hence** *x* ∈ *s* ∪ (*closure s* ∩ *closure t*) **by** *auto*
from *topological-tendstoD*[*OF tendsto*(*1*)[*OF this*] *S l*]

```
    have ?thesis s unfolding eventually-at-filter
      by eventually-elim auto
  } moreover {
    assume x ∉ closure s
    then have ?thesis s
      by (metis (no-types) not-in-closure-trivial-limitI trivial-limit-eventually)
  } moreover {
    assume s: x ∈ closure s x ∉ s
    hence t: x ∈ t x ∈ closure t
      using assms closure-subset[of t] by auto
    from s t have c1: x ∈ s ∪ (closure s ∩ closure t)
      and c2: x ∈ t ∪ (closure s ∩ closure t)by auto
    from topological-tendstoD[OF tendsto(1)[OF c1] S l]
      topological-tendstoD[OF tendsto(2)[OF c2] S l]
    have ?thesis s
      unfolding eventually-at-filter
      by eventually-elim auto
  } ultimately show ?thesis s by blast
  {
    assume x: x ∈ closure s x ∈ closure t
    hence c1: x ∈ s ∪ (closure s ∩ closure t)
      and c2: x ∈ t ∪ (closure s ∩ closure t)
      by auto
    from topological-tendstoD[OF tendsto(1)[OF c1] S l]
      topological-tendstoD[OF tendsto(2)[OF c2] S l]
    have ?thesis t unfolding eventually-at-filter
      by eventually-elim auto
  } moreover {
    assume x ∉ closure t
    then have ?thesis t
      by (metis (no-types) not-in-closure-trivial-limitI trivial-limit-eventually)
  } moreover {
    assume c: x ∉ closure s
    hence c': x ∈ t ∪ (closure s ∩ closure t)
      using assms closure-subset[of s]
      by auto
    from c have eventually (λx. x ∈ − closure s) (at x within t)
      by (intro topological-tendstoD) (auto intro: tendsto-ident-at)
    hence ?thesis t
      using topological-tendstoD[OF tendsto(2)[OF c'] S l] closure-subset[of s]
      unfolding eventually-at-filter
      by eventually-elim (auto; metis closure-subset contra-subsetD)
  } ultimately show ?thesis t by blast
qed

lemma
  tendsto-within-nhd:
  assumes tendsto: (f ⟶ l) (at x within Y)
  assumes nhd: x ∈ T open T T ∩ X ⊆ Y
```

shows $(f \longrightarrow l)$ (*at x within X*)
proof (*rule topological-tendstoI*)
  fix *S* **assume** *S*: *open S l* ∈ *S*
  have $\forall_F$ *x in at x within X*. *x* ∈ *T*
    by (*auto intro*!: *topological-tendstoD nhd*)
  moreover
  have $\forall_F$ *x in at x within X*. *x* ∈ *X*
    by (*simp add*: *eventually-at-filter*)
  ultimately
  have $\forall_F$ *x in at x within X*. *x* ∈ *Y*
    by *eventually-elim* (*insert nhd*, *auto*)
  moreover
  from *topological-tendstoD*[*OF tendsto S*]
  have $\forall_F$ *x in at x within Y*. *f x* ∈ *S* **.**
  ultimately
  show $\forall_F$ *x in at x within X*. *f x* ∈ *S*
    unfolding *eventually-at-filter*
    by *eventually-elim blast*
qed

lemma *eventually-open-cball*:
  assumes *open X*
  assumes *x* ∈ *X*
  shows *eventually* (λ*e*. *cball x e* ⊆ *X*) (*at-right 0*)
proof −
  from *open-contains-cball-eq*[*OF assms*(*1*)] *assms*(*2*)
  obtain *e* **where** *e* > *0 cball x e* ⊆ *X* **by** *auto*
  thus *?thesis*
    by (*auto simp*: *eventually-at dist-real-def intro*!: *exI*[**where** *x*=*e*])
qed

lemma *filterlim-times-real-le*:
  fixes *c*::*real*
  assumes *c* > *0*
  shows *filtermap* (*op* ∗ *c*) (*at-right 0*) ≤ *at-right 0*
  unfolding *filterlim-def*
proof (*rule filter-leI*)
  fix *P*::*real*⇒*bool*
  assume *eventually P* (*at-right 0*)
  then obtain *d* **where** *d*: *d* > *0* ⋀*x*. *x* > *0* ⟹ *x* < *d* ⟹ *P x*
    by (*auto simp*: *eventually-at dist-real-def*)
  then show *eventually P* (*filtermap* (*op* ∗ *c*) (*at-right 0*))
    by (*auto simp*: *eventually-filtermap eventually-at intro*!: *exI*[**where** *x*=*d* / *c*]
      simp: ‹*0* < *c*› *dist-real-def field-simps*)
qed

lemma *filtermap-times-real*:
  assumes (*c*::*real*) > *0*
  shows *filtermap* (*op* ∗ *c*) (*at-right 0*) = *at-right 0*

proof (*rule antisym*)
  have *filtermap* (*op* ∗ (*inverse c*)) (*at-right 0*) ≤ *at-right 0*
    by (*rule filterlim-times-real-le*) (*auto simp*: *assms*)
  also have ... = *filtermap* (*op* ∗ (*inverse c*)) (*filtermap* (*op* ∗ *c*) (*at-right 0*))
    using ‹*c* > *0*›
    by (*simp add*: *filtermap-filtermap field-simps*)
  finally
  show *at-right 0* ≤ *filtermap* (*op* ∗ *c*) (*at-right 0*)
    using *assms*
    by (*subst* (*asm*) *filtermap-mono-strong*) (*auto intro*!: *inj-onI*)
qed (*intro filterlim-times-real-le assms*)

lemma *eventually-at-shift-zero*:
  fixes *x*::′*b*::*real-normed-vector*
  shows *eventually* (λ*h*. *P* (*x* + *h*)) (*at 0*) ⟷ *eventually P* (*at x*)
proof −
  have *eventually* (λ*h*. *P* (*x* + *h*)) (*at 0*) ⟷
    eventually P (*filtermap* (*op* + *x*) (*at 0*))
    by (*simp add*: *eventually-filtermap*)
  also have *filtermap* (*op* + *x*) (*at 0*) = *at x*
    using *filtermap-at-shift*[*of* −*x 0*]
    by (*subst add*.*commute*[*abs-def*]) (*simp add*:)
  finally show *?thesis* **.**
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 *eventually-at-in-ball*: *d* > *0* ⟹ *eventually* (λ*y*. *y* ∈ *ball x0 d*) (*at x0*)
  by (*auto simp*: *eventually-at dist-commute intro*!: *exI*[**where** *x*=*d*])

lemma *seq-harmonic*′: ((λ*n*. *1* / *n*) ⟶ *0*) *sequentially*
  using *seq-harmonic*
  by (*simp add*: *inverse-eq-divide*)

2.12 Continuity

```
lemma continuous-on-fst[continuous-intros]: continuous-on X fst
  unfolding continuous-on-def
  by (intro ballI tendsto-intros)

lemma continuous-on-snd[continuous-intros]: continuous-on X snd
  unfolding continuous-on-def
  by (intro ballI tendsto-intros)

lemma continuous-at-fst[continuous-intros]:
  fixes x::'a::euclidean-space × 'b::euclidean-space
  shows continuous (at x) fst
  unfolding continuous-def netlimit-at
  by (intro tendsto-intros)

lemma continuous-at-snd[continuous-intros]:
  fixes x::'a::euclidean-space × 'b::euclidean-space
  shows continuous (at x) snd
  unfolding continuous-def netlimit-at
  by (intro tendsto-intros)

lemma continuous-at-Pair[continuous-intros]:
  fixes x::'a::euclidean-space × 'b::euclidean-space
  assumes continuous (at x) f
  assumes continuous (at x) g
  shows continuous (at x) (λx. (f x, g x))
  using assms unfolding continuous-def
  by (intro tendsto-intros)

lemma continuous-on-Pair[continuous-intros]:
  assumes continuous-on S f
  assumes continuous-on S g
  shows continuous-on S (λx. (f x, g x))
  using assms unfolding continuous-on-def
  by (auto intro: tendsto-intros)

lemma continuous-Sigma:
  assumes defined: y ∈ Pi T X
  assumes f-cont: continuous-on (Sigma T X) f
  assumes y-cont: continuous-on T y
  shows continuous-on T (λx. f (x, y x))
  using
    defined
    continuous-on-compose2[OF
      continuous-on-subset[where t=(λx. (x, y x)) ` T, OF f-cont]
      continuous-on-Pair[OF continuous-on-id y-cont]]
  by auto

lemma IVT'-closed-segment-real:
```

fixes f :: *real* ⇒ *real*
assumes y: $y \in$ *closed-segment* $(f\ a)$ $(f\ b)$
assumes $*$: *continuous-on* (*closed-segment* a b) f
shows $\exists\, x \in$ *closed-segment* a b. $f\ x = y$
proof −
{
assume $a \le b$
{
assume $f\ a \le f\ b$
hence *?thesis*
using *IVT'*[*of* f a y b] ‹$a \le b$› *assms* **by** (*auto simp*: *closed-segment-real*)
} **moreover** {
assume $f\ b < f\ a$
hence *?thesis*
using *IVT'*[*of* $-f$ a $-y$ b] ‹$a \le b$› *assms*
by (*force simp*: *closed-segment-real intro*!: *continuous-on-minus*)
} **ultimately have** *?thesis* **by** *arith*
} **moreover** {
assume $b < a$
{
assume $f\ b < f\ a$
hence *?thesis*
using *IVT'*[*of* f b y a] ‹$b < a$› *assms* **by** (*auto simp*: *closed-segment-real*)
} **moreover** {
assume $f\ b \ge f\ a$
hence *?thesis*
using *IVT'*[*of* $-f$ b $-y$ a] ‹$b < a$› *assms*
by (*force simp*: *closed-segment-real intro*!: *continuous-on-minus*)
} **ultimately have** *?thesis* **by** *arith*
} **ultimately show** *?thesis* **by** *arith*
qed

lemma *continuous-on-subset-comp*:
continuous-on s f $\Longrightarrow$ *continuous-on* t g $\Longrightarrow$ g ‘ $t \subseteq s$ $\Longrightarrow$ *continuous-on* t (λx. f (g x))
by (*rule continuous-on-compose2*)

lemma
continuous-on-blinfun-componentwise:
fixes f:: $'d$::*t2-space* ⇒ $'e$::*euclidean-space* $\Rightarrow_L$ $'f$::*real-normed-vector*
assumes $\bigwedge i.\ i \in$ *Basis* $\Longrightarrow$ *continuous-on* s (λx. $f\ x\ i$)
shows *continuous-on* s f
using *assms*
by (*auto intro*!: *continuous-at-imp-continuous-on intro*!: *tendsto-componentwise1*
simp: *continuous-on-eq-continuous-within continuous-def*)

lemma *continuous-on-compose-Pair*:
assumes f: *continuous-on* ($A \times B$) ($\lambda(a,\ b)$. $f\ a\ b$)
assumes g: *continuous-on* C g

assumes *h*: *continuous-on C h*
assumes *subset*: *g ' C* ⊆ *A h ' C* ⊆ *B*
shows *continuous-on C* (λ*c*. *f* (*g c*) (*h c*))
using *continuous-on-compose2*[*OF f continuous-on-Pair*[*OF g h*]] *subset*
by *auto*

lemma *continuous-on-compact-product-lemma*:— TODO is this useful? it is just explicit uniform continuity!
  fixes *A*::'*a*::*metric-space set* **and** *B*::'*b*::*metric-space set*
  assumes *continuous-on* (*A* × *X*) (λ(*a*, *x*). *f a x*)
  assumes *compact A compact X*
  assumes *e* > *0*
  shows ∃ *d*>*0*. ∀ *a* ∈ *A*. ∀ *x* ∈ *X*. ∀ *y* ∈ *X*. *dist x y* < *d* ⟶ *dist* (*f a x*) (*f a y*) < *e*
proof −
  have *uniformly-continuous-on* (*A* × *X*) (λ(*a*, *x*). *f a x*)
    by (*intro compact-uniformly-continuous compact-Times assms*)
  then have ∀ *e*>*0*. ∃ *d*>*0*. ∀ *a*∈*A*. ∀ *x*∈*X*. ∀ *b*∈*A*. ∀ *y*∈*X*. *dist* (*b*, *y*) (*a*, *x*) < *d* ⟶ *dist* (*f b y*) (*f a x*) < *e*
    by (*auto simp*: *uniformly-continuous-on-def*)

  from *this*[*rule-format*, *OF* ‹*0* < *e*›]
  obtain *d* **where** *d*: *0* < *d* ⋀*a b x y*. *a*∈*A* ⟹ *x*∈*X* ⟹ *b*∈*A* ⟹ *y*∈*X* ⟹ *dist* (*b*, *y*) (*a*, *x*) < *d* ⟹ *dist* (*f b y*) (*f a x*) < *e*
    by *blast*
  show *?thesis*
    by (*rule exI*[**where** *x*=*d*]) (*auto intro*!: *d simp*: *dist-prod-def*)
qed

2.13 Differentiability

lemma *differentiable-Pair* [*simp*]:
  f differentiable at x within s ⟹ *g differentiable at x within s* ⟹
    (λ*x*. (*f x*, *g x*)) *differentiable at x within s*
  unfolding *differentiable-def* **by** (*blast intro*: *has-derivative-Pair*)

lemma (**in** *bounded-linear*)
  differentiable:
  assumes *g differentiable* (*at x within s*)
  shows (λ*x*. *f* (*g x*)) *differentiable* (*at x within s*)
  using *assms*[*simplified frechet-derivative-works*]
  by (*intro differentiableI*) (*rule has-derivative*)

context begin
private lemmas *diff* = *bounded-linear*.*differentiable*
lemmas *differentiable-mult-right*[*intro*] = *diff*[*OF bounded-linear-mult-right*]
  and *differentiable-mult-left*[*intro*] = *diff*[*OF bounded-linear-mult-left*]
  and *differentiable-inner-right*[*intro*] = *diff*[*OF bounded-linear-inner-right*]
  and *differentiable-inner-left*[*intro*] = *diff*[*OF bounded-linear-inner-left*]

end

lemma (**in** *bounded-bilinear*)
differentiable:
assumes *f*: *f differentiable at x within s* **and** *g*: *g differentiable at x within s*
shows (λx. *prod* (*f x*) (*g x*)) *differentiable at x within s*
using *assms*[*simplified frechet-derivative-works*]
by (*intro differentiableI*) (*rule FDERIV*)

context begin
private lemmas *bdiff* = *bounded-bilinear.differentiable*
lemmas *differentiable-mult*[*intro*] = *bdiff*[*OF bounded-bilinear-mult*]
and *differentiable-scaleR*[*intro*] = *bdiff*[*OF bounded-bilinear-scaleR*]
end

lemma *differentiable-transform-within-weak*:
assumes $x \in s$ $\bigwedge x'$. $x' \in s \Longrightarrow g\ x' = f\ x'$ *f differentiable at x within s*
shows *g differentiable at x within s*
using *assms* **by** (*intro differentiable-transform-within*[*OF - zero-less-one*, **where** *g*=*g*]) *auto*

lemma *differentiable-compose-at*:
f differentiable (*at x*) $\Longrightarrow$ *g differentiable* (*at* (*f x*)) $\Longrightarrow$
(λx. *g* (*f x*)) *differentiable* (*at x*)
unfolding *o-def*[*symmetric*]
by (*rule differentiable-chain-at*)

lemma *differentiable-compose-within*:
f differentiable (*at x within s*) $\Longrightarrow$
g differentiable (*at*(*f x*) *within* (*f* ‘ *s*)) $\Longrightarrow$
(λx. *g* (*f x*)) *differentiable* (*at x within s*)
unfolding *o-def*[*symmetric*]
by (*rule differentiable-chain-within*)

lemma *differentiable-setsum*[*intro*, *simp*]:
assumes *finite s* $\forall a \in s$. (*f a*) *differentiable net*
shows (λx. *setsum* (λa. *f a x*) *s*) *differentiable net*
proof −
from *bchoice*[*OF assms*(*2*)[*unfolded differentiable-def*]]
show *?thesis*
by (*auto intro*!: *has-derivative-setsum simp*: *differentiable-def*)
qed

2.14 Derivatives

lemma *has-derivative-in-compose2*:— TODO: should there be sth like *op has-derivative-on*?

assumes $\bigwedge x$. $x \in t \Longrightarrow$ (*g has-derivative g′ x*) (*at x within t*)
assumes *f* ‘ $s \subseteq t$ $x \in s$

assumes (*f has-derivative f'*) (*at x within s*)
shows ((*λx. g* (*f x*)) *has-derivative* (*λy. g'* (*f x*) (*f' y*))) (*at x within s*)
using *assms*
by (*auto intro*: *has-derivative-within-subset intro*!: *has-derivative-in-compose*[*of f f' x s g*])

lemma *has-derivative-singletonI*:
bounded-linear g ⟹ (*f has-derivative g*) (*at x within* {*x*})
by (*rule has-derivativeI-sandwich*[**where** *e=1*])
(*auto intro*!: *bounded-linear-scaleR-left*)

lemma *vector-derivative-eq-rhs*:
(*f has-vector-derivative f'*) *F* ⟹ *f'* = *g'* ⟹ (*f has-vector-derivative g'*) *F*
by *simp*

lemma *has-derivative-transform*:
assumes *x* ∈ *s* ⋀*x*. *x* ∈ *s* ⟹ *g x* = *f x*
assumes (*f has-derivative f'*) (*at x within s*)
shows (*g has-derivative f'*) (*at x within s*)
using *assms*
by (*intro has-derivative-transform-within*[*OF - zero-less-one*, **where** *g=g*]) *auto*

lemma *has-derivative-within-If-eq*:
((*λx. if P x then f x else g x*) *has-derivative f'*) (*at x within s*) =
(*bounded-linear f'* ∧
((*λy.*(*if P y then* (*f y* − ((*if P x then f x else g x*) + *f'* (*y* − *x*)))/$_R$ *norm* (*y* − *x*)
else (*g y* − ((*if P x then f x else g x*) + *f'* (*y* − *x*)))/$_R$ *norm* (*y* − *x*)))
⟶ *0*) (*at x within s*))
(**is** - = (- ∧ (*?if* ⟶ *0*) -))
proof −
have (*λy.* (*1* / *norm* (*y* − *x*)) *$_R$
((*if P y then f y else g y*) −
((*if P x then f x else g x*) + *f'* (*y* − *x*)))) = *?if*
by (*auto simp*: *inverse-eq-divide*)
thus *?thesis* **by** (*auto simp*: *has-derivative-within*)
qed

lemma *has-derivative-If*:
assumes *f'*: *x* ∈ *s* ∪ (*closure s* ∩ *closure t*) ⟹
(*f has-derivative f' x*) (*at x within s* ∪ (*closure s* ∩ *closure t*))
assumes *g'*: *x* ∈ *t* ∪ (*closure s* ∩ *closure t*) ⟹
(*g has-derivative g' x*) (*at x within t* ∪ (*closure s* ∩ *closure t*))
assumes *connect*: *x* ∈ *closure s* ⟹ *x* ∈ *closure t* ⟹ *f x* = *g x*
assumes *connect'*: *x* ∈ *closure s* ⟹ *x* ∈ *closure t* ⟹ *f' x* = *g' x*
assumes *x-in*: *x* ∈ *s* ∪ *t*
shows ((*λx. if x* ∈ *s then f x else g x*) *has-derivative*
(*if x* ∈ *s then f' x else g' x*)) (*at x within* (*s* ∪ *t*))
proof −

```
  from f' x-in interpret f': bounded-linear if x ∈ s then f' x else (λx. 0)
    by (auto simp add: has-derivative-within)
  from g' interpret g': bounded-linear if x ∈ t then g' x else (λx. 0)
    by (auto simp add: has-derivative-within)
  have bl: bounded-linear (if x ∈ s then f' x else g' x)
    using f'.scaleR f'.bounded f'.add g'.scaleR g'.bounded g'.add x-in
    by (unfold-locales; force)
  show ?thesis
    using f' g' closure-subset[of t] closure-subset[of s]
    unfolding has-derivative-within-If-eq
    by (intro conjI bl tendsto-If x-in)
      (auto simp: has-derivative-within inverse-eq-divide connect connect' set-mp)
qed

lemma has-vector-derivative-If:
  assumes x-in: x ∈ s ∪ t
  assumes u = s ∪ t
  assumes f': x ∈ s ∪ (closure s ∩ closure t) ⟹
    (f has-vector-derivative f' x) (at x within s ∪ (closure s ∩ closure t))
  assumes g': x ∈ t ∪ (closure s ∩ closure t) ⟹
    (g has-vector-derivative g' x) (at x within t ∪ (closure s ∩ closure t))
  assumes connect: x ∈ closure s ⟹ x ∈ closure t ⟹ f x = g x
  assumes connect': x ∈ closure s ⟹ x ∈ closure t ⟹ f' x = g' x
  shows ((λx. if x ∈ s then f x else g x) has-vector-derivative
    (if x ∈ s then f' x else g' x)) (at x within u)
  unfolding has-vector-derivative-def assms
  using x-in
  apply (intro has-derivative-If[THEN has-derivative-eq-rhs])
      apply (rule f'[unfolded has-vector-derivative-def]; assumption)
     apply (rule g'[unfolded has-vector-derivative-def]; assumption)
  by (auto simp: assms)

lemma has-derivative-If-in-closed:
  assumes f':⋀x. x ∈ s ⟹ (f has-derivative f' x) (at x within s)
  assumes g':⋀x. x ∈ t ⟹ (g has-derivative g' x) (at x within t)
  assumes connect: ⋀x. x ∈ s ∩ t ⟹ f x = g x ⋀x. x ∈ s ∩ t ⟹ f' x = g' x
  assumes closed t closed s x ∈ s ∪ t
  shows ((λx. if x ∈ s then f x else g x) has-derivative (if x ∈ s then f' x else g'
x)) (at x within (s ∪ t))
  (is (?if has-derivative ?if') -)
  unfolding has-derivative-within
proof (safe intro!: tendstoI)
  fix e::real assume 0 < e
  let ?D = λx f f' y. (1 / norm (y − x)) *_R (f y − (f x + f' (y − x)))
  have f': x ∈ s ⟹ ((?D x f (f' x)) ⟶ 0) (at x within s)
    and g': x ∈ t ⟹ ((?D x g (g' x)) ⟶ 0) (at x within t)
    using f' g' by (auto simp: has-vector-derivative-def has-derivative-within)
  let ?thesis = eventually (λy. dist (?D x ?if ?if' y) 0 < e) (at x within s ∪ t)
  {
```

assume $x \in s$ $x \in t$
from *tendstoD*[*OF f'*[*OF* ‹$x \in s$›] ‹$0 < e$›] *tendstoD*[*OF g'*[*OF* ‹$x \in t$›] ‹$0 < e$›]
have *?thesis* **unfolding** *eventually-at-filter*
by *eventually-elim* (*insert* ‹$x \in s$› ‹$x \in t$›, *auto simp*: *connect*)
} moreover {
assume $x \in s$ $x \notin t$
hence *eventually* ($\lambda x.\ x \in -\ t$) (*at x within* $s \cup t$) **using** ‹*closed t*›
by (*intro topological-tendstoD*) (*auto intro*: *tendsto-ident-at*)
with *tendstoD*[*OF f'*[*OF* ‹$x \in s$›] ‹$0 < e$›] **have** *?thesis* **unfolding** *eventually-at-filter*
by *eventually-elim* (*insert* ‹$x \in s$› ‹$x \notin t$›, *auto simp*: *connect*)
} moreover {
assume $x \notin s$ **hence** $x \in t$ **using** *assms* **by** *auto*
have *eventually* ($\lambda x.\ x \in -\ s$) (*at x within* $s \cup t$) **using** ‹*closed s*› ‹$x \notin s$›
by (*intro topological-tendstoD*) (*auto intro*: *tendsto-ident-at*)
with *tendstoD*[*OF g'*[*OF* ‹$x \in t$›] ‹$0 < e$›] **have** *?thesis* **unfolding** *eventually-at-filter*
by *eventually-elim* (*insert* ‹$x \in t$› ‹$x \notin s$›, *auto simp*: *connect*)
} ultimately show *?thesis* **by** *blast*
qed (*insert assms*, *auto intro*!: *has-derivative-bounded-linear f' g'*)

lemma *linear-continuation*:
assumes *f'*:$\bigwedge x.\ x \in \{a\ ..\ b\} \Longrightarrow$
(*f has-vector-derivative f' x*) (*at x within* $\{a\ ..\ b\}$)
assumes *g'*:$\bigwedge x.\ x \in \{b\ ..\ c\} \Longrightarrow$
(*g has-vector-derivative g' x*) (*at x within* $\{b\ ..\ c\}$)
assumes *connect*: $f\ b = g\ b$ $f'\ b = g'\ b$
assumes *x*: $x \in \{a\ ..\ c\}$
assumes *abc*:$a \leq b$ $b \leq c$
shows (($\lambda x.$ *if* $x \leq b$ *then f x else g x*) *has-vector-derivative*
($\lambda x.$ *if* $x \leq b$ *then f' x else g' x*) *x*) (*at x within* $\{a\ ..\ c\}$)
(**is** (*?h has-vector-derivative ?h' x*) -)
proof –
have *un*: $\{a\ ..\ b\} \cup \{b\ ..\ c\} = \{a\ ..\ c\}$ **using** *assms* **by** *auto*
note *has-derivative-If-in-closed*[*derivative-intros*]
note *f'*[*simplified has-vector-derivative-def*, *derivative-intros*]
note *g'*[*simplified has-vector-derivative-def*, *derivative-intros*]
have *if'*: (($\lambda x.$ *if* $x \in \{a\ ..\ b\}$ *then f x else g x*) *has-vector-derivative*
($\lambda x.$ *if* $x \leq b$ *then f' x else g' x*) *x*) (*at x within* $\{a\ ..\ b\}\cup\{b\ ..\ c\}$)
unfolding *has-vector-derivative-def*
using *assms*
apply –
apply (*rule derivative-eq-intros refl* | *assumption*)+
by *auto*
show *?thesis*
unfolding *has-vector-derivative-def*
by (*rule has-derivative-transform*[*OF*
x - *if'*[*simplified un has-vector-derivative-def*]])
simp
qed

lemma *exists-linear-continuation*:
assumes f':$\bigwedge x.\ x \in \{a\ ..\ b\} \Longrightarrow$
$(f$ *has-vector-derivative* $f'\ x)$ $(at\ x\ within\ \{a\ ..\ b\})$
shows $\exists fc.\ (\forall x.\ x \in \{a\ ..\ b\} \longrightarrow (fc$ *has-vector-derivative* $f'\ x)\ (at\ x)) \wedge$
$(\forall x.\ x \in \{a\ ..\ b\} \longrightarrow fc\ x = f\ x)$
proof (*rule*, *safe*)
fix x **assume** $x \in \{a\ ..\ b\}$ **hence** $a \leq b$ **by** *simp*
let $?line = \lambda a\ x.\ f\ a + (x - a) *_R f'\ a$
let $?fc = (\lambda x.\ if\ x \in \{a\ ..\ b\}\ then\ f\ x\ else\ if\ x \in \{..a\}\ then\ ?line\ a\ x\ else\ ?line\ b\ x)$
have [*simp*]:
$\bigwedge x.\ x \in \{a\ ..\ b\} \Longrightarrow (b \leq x \longleftrightarrow x = b)$ $\bigwedge x.\ x \in \{a\ ..\ b\} \Longrightarrow (x \leq a \longleftrightarrow x = a)$
$\bigwedge x.\ x \leq a \Longrightarrow (b \leq x \longleftrightarrow x = b)$ **using** ‹$a \leq b$› **by** *auto*
note [*derivative-intros*] =
has-derivative-If-in-closed
f'[*simplified has-vector-derivative-def*]
have $(?fc$ *has-vector-derivative* $f'\ x)$ $(at\ x\ within\ \{a\ ..\ b\} \cup (\{..a\} \cup \{b..\}))$
using ‹$x \in \{a\ ..\ b\}$› ‹$a \leq b$›
by (*auto intro*!: *derivative-eq-intros simp*: *has-vector-derivative-def*
simp del: *atMost-iff atLeastAtMost-iff*)
moreover have $\{a\ ..\ b\} \cup (\{..a\} \cup \{b..\}) = UNIV$ **by** *auto*
ultimately show $(?fc$ *has-vector-derivative* $f'\ x)$ $(at\ x)$ **by** *simp*
show $?fc\ x = f\ x$ **using** ‹$x \in \{a\ ..\ b\}$› **by** *simp*
qed

lemma *Pair-has-vector-derivative*:
assumes $(f$ *has-vector-derivative* $f')$ $(at\ x\ within\ s)$
$(g$ *has-vector-derivative* $g')$ $(at\ x\ within\ s)$
shows $((\lambda x.\ (f\ x,\ g\ x))$ *has-vector-derivative* $(f',\ g'))$ $(at\ x\ within\ s)$
using *assms*
by (*auto simp*: *has-vector-derivative-def intro*!: *derivative-eq-intros*)

lemma *has-vector-derivative-imp*:
assumes $x \in s$
assumes $\bigwedge x.\ x \in s \Longrightarrow f\ x = g\ x$
assumes $f'g'$:$f' = g'$
assumes $x = y$ $s = t$
assumes f': $(f$ *has-vector-derivative* $f')$ $(at\ x\ within\ s)$
shows $(g$ *has-vector-derivative* $g')$ $(at\ y\ within\ t)$
unfolding *has-vector-derivative-def has-derivative-within'*
proof (*safe*)
fix e::*real*
assume $0 < e$
with *assms* f' **have** $\exists d>0.\ \forall x'\in s.$
$0 < norm\ (x' - x) \wedge norm\ (x' - x) < d \longrightarrow$
$norm\ (g\ x' - g\ y - (x' - y) *_R g')\ /\ norm\ (x' - x) < e$

by (*auto simp add*: *has-vector-derivative-def has-derivative-within′*)
with *assms* **show** $\exists\, d{>}0.\ \forall\, x'{\in}t.\ 0 < norm\ (x' - y) \wedge norm\ (x' - y) < d \longrightarrow$
$norm\ (g\ x' - g\ y - (x' - y) *_R g') \ /\ norm\ (x' - y) < e$
by *auto*
next
show *bounded-linear* $(\lambda x.\ x *_R g')$
using
has-derivative-bounded-linear[*OF f′*[*simplified has-vector-derivative-def*],
simplified f′g′]
.
qed

lemma *has-vector-derivative-cong*:
assumes $x \in s$
assumes $\bigwedge x.\ x \in s \Longrightarrow f\ x = g\ x$
assumes $f'g'$:$f' = g'$
assumes $x = y$ $s = t$
shows $(f$ *has-vector-derivative* $f')$ $(at\ x\ within\ s) =$
$(g$ *has-vector-derivative* $g')$ $(at\ y\ within\ t)$
using *has-vector-derivative-imp assms* **by** *metis*

lemma *has-derivative-within-union*:
assumes (*f has-derivative g*) (*at x within s*)
assumes (*f has-derivative g*) (*at x within t*)
shows (*f has-derivative g*) (*at x within* $(s \cup t)$)
proof *cases*
assume *at x within* $(s \cup t) =$ *bot*
thus *?thesis* **using** *assms* **by** (*simp-all add*: *has-derivative-def*)
next
assume *st*: *at x within* $(s \cup t) \neq$ *bot*
thus *?thesis*
using *assms*
by (*cases at x within s* = *bot*;
cases at x within t = *bot*;
auto simp: *Lim-within-union has-derivative-def netlimit-within*)
qed

lemma *has-vector-derivative-within-union*:
assumes (*f has-vector-derivative g*) (*at x within s*)
assumes (*f has-vector-derivative g*) (*at x within t*)
shows (*f has-vector-derivative g*) (*at x within* $(s \cup t)$)
using *assms*
by (*auto simp*: *has-vector-derivative-def intro*: *has-derivative-within-union*)

lemma *vector-derivative-within-closed-interval*:
fixes f::*real* $\Rightarrow$ *′a*::*euclidean-space*
assumes $a < b$ **and** $x \in \{a\ ..\ b\}$
assumes (*f has-vector-derivative f′*) (*at x within* $\{a\ ..\ b\}$)
shows *vector-derivative f* (*at x within* $\{a\ ..\ b\}$) $= f'$

using *assms vector-derivative-within-closed-interval*
by *fastforce*

lemma
has-vector-derivative-at-within-open-subset:
assumes $\bigwedge x.\ x \in T \Longrightarrow$ (*f has-vector-derivative f′ x*) (*at x within T*)
assumes $x \in S$ *open S* $S \subseteq T$
shows (*f has-vector-derivative f′ x*) (*at x*)
proof −
from *at-within-open*[*OF assms*(*2*,*3*), *symmetric*]
show (*f has-vector-derivative f′ x*) (*at x*)
using ‹$S \subseteq T$›
by (*auto intro*!: *has-vector-derivative-within-subset*[*OF* - ‹$S \subseteq T$›] *assms*)
qed

TODO: include this into the attribute *derivative-intros*?

lemma *DERIV-compose-FDERIV*:
fixes *f*::*real*⇒*real*
assumes *DERIV f* (*g x*) :> *f′*
assumes (*g has-derivative g′*) (*at x within s*)
shows ((λ*x*. *f* (*g x*)) *has-derivative* (λ*x*. *g′ x* ∗ *f′*)) (*at x within s*)
using *assms has-derivative-compose*[*of g g′ x s f op* ∗ *f′*]
by (*auto simp*: *has-field-derivative-def ac-simps*)

lemmas *has-derivative-sin*[*derivative-intros*] = *DERIV-sin*[*THEN DERIV-compose-FDERIV*]
and *has-derivative-cos*[*derivative-intros*] = *DERIV-cos*[*THEN DERIV-compose-FDERIV*]
and *has-derivative-exp*[*derivative-intros*] = *DERIV-exp*[*THEN DERIV-compose-FDERIV*]
and *has-derivative-ln*[*derivative-intros*] = *DERIV-ln*[*THEN DERIV-compose-FDERIV*]

lemma *has-derivative-continuous-on*:
($\bigwedge x.\ x \in s \Longrightarrow$ (*f has-derivative f′ x*) (*at x within s*)) ⟹ *continuous-on s f*
by (*auto intro*!: *differentiable-imp-continuous-on differentiableI simp*: *differentiable-on-def*)

lemma *has-vector-derivative-continuous-on*:
($\bigwedge x.\ x \in s \Longrightarrow$ (*f has-vector-derivative f′ x*) (*at x within s*)) ⟹ *continuous-on*
s f
by (*auto intro*!: *differentiable-imp-continuous-on differentiableI simp*: *has-vector-derivative-def*
differentiable-on-def)

lemma *taylor-up-within*:
assumes *INIT*: *n*>*0* $\bigwedge t.\ t \in \{a\ ..\ b\} \Longrightarrow$ *diff 0 t* = *f t*
and *DERIV*: $\bigwedge m\ t.\ m < n \Longrightarrow a \le t \Longrightarrow t \le b \Longrightarrow$
((*diff m*) *has-vector-derivative* (*diff* (*Suc m*) *t*)) (*at t within* {*a* .. *b*})
and *INTERV*: $a \le c$ $c < b$
shows ∃ *t*. *c* < *t* & *t* < *b* &
f b = (∑ *m*<*n*. (*diff m c* / (*fact m*)) ∗ (*b* − *c*) ^*m*) +
(*diff n t* / (*fact n*)) ∗ (*b* − *c*) ^*n*
(**is** *?taylor f diff*)
proof −

from *exists-linear-continuation*[*of a b*, *OF DERIV*]
have ∀ *m*. ∃ *d′*. *m* < *n* ⟶
(∀ *x* ∈ {*a* .. *b*}. (*d′ has-vector-derivative diff* (*Suc m*) *x*) (*at x*) ∧ *d′ x* = *diff m x*)
by (*metis atLeastAtMost-iff*)
then obtain *d′* **where** *d′*:
⋀*m x*. *m*<*n* ⟹ *a* ≤ *x* ⟹ *x* ≤ *b* ⟹ (*d′ m has-vector-derivative diff* (*Suc m*) *x*) (*at x*)
⋀*m x*. *m*<*n* ⟹ *a* ≤ *x* ⟹ *x* ≤ *b* ⟹ *d′ m x* = *diff m x*
by (*metis atLeastAtMost-iff*)
let *?diff* = λ*m*. *if m* = *n then diff m else d′ m*
have *?taylor* (*?diff 0*) *?diff* **using** *d′*
by (*intro taylor-up*[*OF - - -* ‹*a* ≤ *c*›])
(*auto simp*: *has-field-derivative-def has-vector-derivative-def*
INIT INTERV mult-commute-abs)
thus *?taylor f diff* **using** *d′ INTERV INIT* **by** *auto*
qed

lemma *taylor-up-within-vector*:
fixes *f*::*real* ⇒ ′*a*::*euclidean-space*
assumes *INIT*: *n*>*0* ⋀*t*. *t* ∈ {*a* .. *b*} ⟹ *diff 0 t* = *f t*
and *DERIV*: ⋀*m t*. *m* < *n* ⟹ *a* ≤ *t* ⟹ *t* ≤ *b* ⟹
((*diff m*) *has-vector-derivative* (*diff* (*Suc m*) *t*)) (*at t within* {*a* .. *b*})
and *INTERV*: *a* ≤ *c c* < *b*
shows ∃ *t*. (∀ *i*∈*Basis*::′*a set*. *c* < *t i* & *t i* < *b*) ∧
f b = *setsum* (%*m*. (*b* − *c*)^*m* $*_R$ (*diff m c* $/_R$ (*fact m*))) {..<*n*} +
setsum (λ*x*. (((*b* − *c*) ^ *n* $*_R$ *diff n* (*t x*) $/_R$ (*fact n*)) • *x*) $*_R$ *x*) *Basis*
proof −
obtain *t* **where** *t*: ∀ *i*∈*Basis*::′*a set*. *t i* > *c* ∧ *t i* < *b* ∧
f b • *i* =
(∑ *m*<*n*. *diff m c* • *i* / (*fact m*) ∗ (*b* − *c*) ^ *m*) +
diff n (*t i*) • *i* / (*fact n*) ∗ (*b* − *c*) ^ *n*
proof (*atomize-elim*, *rule bchoice*, *safe*)
fix *i*::′*a*
assume *i* ∈ *Basis*
have *DERIV-0*: ⋀*t*. *t* ∈ {*a* .. *b*} ⟹ (*diff 0*) *t* • *i* = *f t* • *i* **using** *INIT* **by** *simp*
have *DERIV-Suc*: ⋀*m t*. *m* < *n* ⟹ *a* ≤ *t* ⟹ *t* ≤ *b* ⟹
((λ*t*. (*diff m*) *t* • *i*) *has-vector-derivative* (*diff* (*Suc m*) *t* • *i*)) (*at t within* {*a* .. *b*})
using *DERIV* **by** (*auto intro*!: *derivative-eq-intros simp*: *has-vector-derivative-def*)
from *taylor-up-within*[*OF INIT*(*1*) *DERIV-0 DERIV-Suc INTERV*]
show ∃ *t*>*c*. *t* < *b* ∧ *f b* • *i* =
(∑ *m*<*n*. *diff m c* • *i* / (*fact m*) ∗ (*b* − *c*) ^ *m*) +
diff n t • *i* / (*fact n*) ∗ (*b* − *c*) ^ *n* **by** *simp*
qed
have *f b* = (∑ *i*∈*Basis*. (*f b* • *i*) $*_R$ *i*) **by** (*rule euclidean-representation*[*symmetric*])
also have ... =
(∑ *i*∈*Basis*. ((∑ *m*<*n*. (*b* − *c*) ^ *m* $*_R$ (*diff m c* $/_R$ (*fact m*))) • *i*) $*_R$ *i*) +

```
    (∑x∈Basis. (((b − c) ˆ n *R diff n (t x) /R (fact n)) · x) *R x)
    using t
    by (simp add: setsum.distrib inner-setsum-left inverse-eq-divide algebra-simps)
  finally show ?thesis using t by (auto simp: euclidean-representation)
qed

lemma mvt-closed-segmentE:
  fixes f::real⇒real
  assumes ⋀x. x ∈ closed-segment a b ⟹
    (f has-vector-derivative f′ x) (at x within closed-segment a b)
  obtains y where y ∈ closed-segment a b  f b − f a = (b − a) * f′ y
proof cases
  assume a ≤ b
  with mvt-very-simple[of a b f λx i. i *R f′ x] assms
  obtain y where y ∈ closed-segment a b  f b − f a = (b − a) * f′ y
    by (auto simp: has-vector-derivative-def closed-segment-real)
  thus ?thesis ..
next
  assume ¬ a ≤ b
  with mvt-very-simple[of b a f λx i. i *R f′ x] assms
  obtain y where y ∈ closed-segment a b  f b − f a = (b − a) * f′ y
    by (force simp: has-vector-derivative-def closed-segment-real algebra-simps)
  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 ⋀x. x ∈ open-segment a b ⟹
      (f has-vector-derivative f′ x) (at x within open-segment a b)
    and ⋀x. x ∈ open-segment a b ⟹
      (g has-vector-derivative g′ x) (at x within open-segment a b)
    and ⋀x. x ∈ open-segment a b ⟹ norm (f′ x) ≤ g′ x
  shows norm (f b − f a) ≤ 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 ⋀x. x∈{a<..<b} ⟹ (f has-vector-derivative f′ x) (at x)
      and ⋀x. x∈{a<..<b} ⟹ (g has-vector-derivative g′ x) (at x)
```

and $\bigwedge x.\ x \in \{a<..<b\} \Longrightarrow$ *norm* $(f'\ x) \leq g'\ x$
by (*auto simp*: *open-segment-real closed-segment-real*
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* $(f'\ x) \leq g'\ x$
by (*auto simp*: *open-segment-real closed-segment-real*
at-within-open[**where** $S = \{b<..<a\}$])
from *differentiable-bound-general*[*OF* ‹$b < a$› *this*]
have *norm* $(f\ a - f\ b) \leq g\ a - g\ b$ **by** *simp*
also have $\ldots \leq$ *abs* $(g\ b - g\ a)$ **by** *simp*
finally have *?thesis* **by** (*simp add*: *norm-minus-commute*)
} ultimately show *?thesis* **by** *arith*
qed

lemma *has-real-derivative-continuous-on*:
($\bigwedge x.\ x \in s \Longrightarrow$ (f *has-real-derivative* $f'\ x$) (*at* x *within* s)) $\Longrightarrow$
continuous-on s f
by (*metis DERIV-continuous continuous-on-eq-continuous-within*)

2.15 Integration

lemma *has-integral-eq-rhs*:
assumes (f *has-integral* J) s
assumes $I = J$
shows (f *has-integral* I) s
using *assms*
by *metis*

lemma *has-integral-id*:
(($\lambda x.\ x$) *has-integral* (*if* $a \leq b$ *then* $b^2/2 - a^2/2$ *else* 0)) $\{a\ ..\ b{::}real\}$
by (*auto intro*!: *fundamental-theorem-of-calculus derivative-eq-intros*
simp: *has-vector-derivative-def*)

lemma *integrable-antiderivative*:
fixes $F{::}real \Rightarrow{}'a{::}banach$
assumes F: $\bigwedge x.\ a \leq x \Longrightarrow x \leq b \Longrightarrow$
(F *has-vector-derivative* $f\ x$) (*at* x *within* $\{a\ ..\ b\}$)
shows f *integrable-on* $\{a\ ..\ b\}$
apply (*cases* $a \leq b$)
apply (*rule has-integral-integrable*)
apply (*rule fundamental-theorem-of-calculus*)

by (*auto intro*!: *F fundamental-theorem-of-calculus*)

lemmas *content-real*[*simp*]

lemma *integral-real-singleton*[*simp*]:
integral {*a*::*real*} *f* = *0*
using *integral-refl*[*of a f*] **by** *simp*

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

lemma *mvt-integral*:
fixes *f*::′*a*::*real-normed-vector*⇒′*b*::*banach*
assumes *f*′[*derivative-intros*]:
⋀*x*. *x* ∈ *S* ⟹ (*f has-derivative f*′ *x*) (*at x within S*)
assumes *line-in*: ⋀*t*. *t* ∈ {*0..1*} ⟹ *x* + *t* $*_R$ *y* ∈ *S*
shows *f* (*x* + *y*) − *f x* = *integral* {*0..1*} (λ*t*. *f*′ (*x* + *t* $*_R$ *y*) *y*) (**is** *?th1*)
proof −
from *assms* **have** *subset*: (λ*xa*. *x* + *xa* $*_R$ *y*) ‘ {*0..1*} ⊆ *S* **by** *auto*
note [*derivative-intros*] =
has-derivative-subset[*OF* - *subset*]
has-derivative-in-compose[**where** *f*=(λ*xa*. *x* + *xa* $*_R$ *y*) **and** *g* = *f*]
note [*continuous-intros*] =
continuous-on-compose2[**where** *f*=(λ*xa*. *x* + *xa* $*_R$ *y*)]
continuous-on-subset[*OF* - *subset*]
have ⋀*t*. *t* ∈ {*0..1*} ⟹
((λ*t*. *f* (*x* + *t* $*_R$ *y*)) *has-vector-derivative f*′ (*x* + *t* $*_R$ *y*) *y*)
(*at t within* {*0..1*})
using *assms*
by (*auto simp*: *has-vector-derivative-def*
linear-cmul[*OF has-derivative-linear*[*OF f*′], *symmetric*]
intro!: *derivative-eq-intros*)
from *fundamental-theorem-of-calculus*[*rule-format*, *OF* - *this*]
show *?th1*
by (*auto intro*!: *integral-unique*[*symmetric*])
qed

lemma *integral-mult*:
fixes *K*::*real*
shows *f integrable-on X* ⟹ *K* ∗ *integral X f* = *integral X* (λ*x*. *K* ∗ *f x*)
unfolding *real-scaleR-def*[*symmetric*]
apply (*subst integral-cmul*)
by *auto*

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

by *auto*

lemma *integrable-continuous-closed-segment*:
fixes f :: *real* ⇒ *'a*::*banach*
assumes *continuous-on* (*closed-segment a b*) f
shows f *integrable-on* (*closed-segment a b*)
using *assms closed-segment-eq-real-ivl*
by *auto*

lemma *continuous-on-imp-absolutely-integrable-on*:
fixes f::*real* ⇒ *'a*::*banach*
shows *continuous-on* {*a*..*b*} f ⟹
norm (*integral* {*a*..*b*} f) ≤ *integral* {*a*..*b*} (λ*x*. *norm* (f x))
by (*rule absolutely-integrable-le*[*OF absolutely-integrable-onI*[*OF*
integrable-continuous-real integrable-continuous-real[*OF continuous-on-norm*]]])

lemma *integral-bound*:
fixes f::*real* ⇒ *'a*::*banach*
assumes $a \le b$
assumes *continuous-on* {*a* .. *b*} f
assumes ⋀*t*. $t \in$ {*a* .. *b*} ⟹ *norm* (f t) ≤ B
shows *norm* (*integral* {*a* .. *b*} f) ≤ $B * (b - a)$
proof −
note *continuous-on-imp-absolutely-integrable-on*[*OF assms*(*2*)]
also have *integral* {*a*..*b*} (λ*x*. *norm* (f x)) ≤ *integral* {*a*..*b*} (λ-. B)
by (*rule integral-le*)
(*auto intro*!: *integrable-continuous-real continuous-intros assms*)
also have ... = $B * (b - a)$ **using** *assms* **by** *simp*
finally show *?thesis* **.**
qed

lemma *integral-minus-sets*:
fixes f::*real* ⇒ *'a*::*banach*
shows $c \le a$ ⟹ $c \le b$ ⟹ f *integrable-on* {*c* .. *max a b*} ⟹
integral {*c* .. *a*} f − *integral* {*c* .. *b*} f =
(*if* $a \le b$ *then* − *integral* {*a* .. *b*} f *else integral* {*b* .. *a*} f)
using *integral-combine*[*of c a b f*] *integral-combine*[*of c b a f*]
by (*auto simp*: *algebra-simps max-def*)

lemma *integral-minus-sets'*:
fixes f::*real* ⇒ *'a*::*banach*
shows $c \ge a$ ⟹ $c \ge b$ ⟹ f *integrable-on* {*min a b* .. *c*} ⟹
integral {*a* .. *c*} f − *integral* {*b* .. *c*} f =
(*if* $a \le b$ *then integral* {*a* .. *b*} f *else* − *integral* {*b* .. *a*} f)
using *integral-combine*[*of b a c f*] *integral-combine*[*of a b c f*]
by (*auto simp*: *algebra-simps min-def*)

lemma *integral-has-real-derivative*:
assumes *continuous-on* {*a*..*b*} g

```
  assumes t ∈ {a..b}
  shows ((λx. integral {a..x} g) has-real-derivative g t) (at t within {a..b})
  using integral-has-vector-derivative[of a b g t] assms
  by (auto simp: has-field-derivative-iff-has-vector-derivative)

lemma derivative-quotient-bound:
  assumes g-deriv: ⋀t. t ∈ {a .. b} ⟹
    (g has-real-derivative g' t) (at t within {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 −
  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 {a..t}
    by (force simp: g-nonzero intro: assms has-field-derivative-subset[OF g-deriv]
      continuous-on-subset[OF g'-cont] continuous-intros
      has-real-derivative-continuous-on)
  have ⋀t. t ∈ {a..b} ⟹ ((λt. 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 assms(1)]
        simp: has-field-derivative-iff-has-vector-derivative[symmetric])
  hence *: ⋀t. t ∈ {a .. b} ⟹ ln (g t) − ln (g a) = integral {a .. t} (λt. g' t /
g t)
    using integrable-integral[OF frac-integrable]
    by (rule has-integral-unique[where f = λt. g' t / g t])
  from * t-in have ln (g t) − ln (g a) = integral {a .. t} (λt. 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: ⋀t. t ∈ {a .. b} ⟹
    (g has-real-derivative g' t) (at t within {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*: $t \in \{a..b\}$
shows $g\ t \le g\ b * exp\ (K * (t - b))$
proof –
from *assms* **have** *g-nonzero*: $\bigwedge t.\ t \in \{a..b\} \Longrightarrow g\ t \neq 0$
by *fastforce*
have *frac-integrable*: $\bigwedge t.\ t \in \{a\ ..\ b\} \Longrightarrow (\lambda t.\ g'\ t\ /\ g\ t)$ *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* *has-real-derivative-continuous-on*)
have $\bigwedge t.\ t \in \{a..b\} \Longrightarrow ((\lambda t.\ g'\ t\ /\ g\ t)$ *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 assms*(*1*)] *simp*: *has-field-derivative-iff-has-vector-derivative*[*symmetric*])
hence ∗: $\bigwedge t.\ t \in \{a..b\} \Longrightarrow$ *ln* $(g\ b) -$ *ln* $(g\ t) =$ *integral* $\{t..b\}\ (\lambda t.\ g'\ t\ /\ g\ t)$
using *integrable-integral*[*OF frac-integrable*]
by (*rule has-integral-unique*[**where** $f = \lambda t.\ g'\ t\ /\ g\ t$])
have $K * (b - t) =$ *integral* $\{t..b\}\ (\lambda\text{-}.\ K)$
using ‹$t \in \{a..b\}$›
by *simp*
also have ... $\le$ *integral* $\{t..b\}\ (\lambda t.\ g'\ t\ /\ g\ t)$
using ‹$t \in \{a..b\}$›
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* $(g\ t) \le$ *ln* $(g\ b)$ (**is** *?lhs* ≤ *?rhs*)
by *simp*
hence *exp ?lhs* ≤ *exp ?rhs*
by *simp*
hence $g\ t * exp\ (K * (b - t)) \le g\ b$
using ‹$t \in \{a..b\}$› *g-pos*
by (*simp add*: *ac-simps exp-add del*: *exp-le-cancel-iff*)
hence $g\ t\ /\ exp\ (K * (t - b)) \le g\ b$
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 $G \equiv \lambda t.\ C + K *$ *integral* $\{a..t\}\ (\lambda s.\ g\ s)$
assumes *g-le-G*: $\bigwedge t.\ t \in \{a..b\} \Longrightarrow g\ t \le G\ t$
assumes *g-cont*: *continuous-on* $\{a..b\}\ g$
assumes *g-nonneg*: $\bigwedge t.\ t \in \{a..b\} \Longrightarrow 0 \le g\ t$
assumes *pos*: $0 < C\ K > 0$
assumes $t \in \{a..b\}$
shows $g\ t \le C * exp\ (K * (t - a))$
proof –
have *G-pos*: $\bigwedge t.\ t \in \{a..b\} \Longrightarrow 0 < G\ t$
by (*auto simp*: *G-def intro*!: *add-pos-nonneg mult-nonneg-nonneg integral-nonneg*

integrable-continuous-real assms intro: *less-imp-le continuous-on-subset*)
have $g\ t \le G\ t$ **using** *assms* **by** *auto*
also
{
have $\bigwedge t.\ t \in \{a..b\} \Longrightarrow (G$ *has-real-derivative* $K * g\ t)$ (*at* t *within* $\{a..b\}$)
by (*auto intro*!: *derivative-eq-intros integral-has-real-derivative g-cont simp add*: *G-def*)
moreover
{
fix t **assume** $t \in \{a..b\}$
hence $K * g\ t\ /\ G\ t \le K * G\ t\ /\ G\ t$
using *pos g-le-G G-pos*
by (*intro divide-right-mono mult-left-mono*) (*auto intro*!: *less-imp-le*)
also have $\ldots = K$
using *G-pos*[*of t*] ‹$t \in \{a\ ..\ b\}$› **by** *simp*
finally have $K * g\ t\ /\ G\ t \le K$ **.**
}
ultimately have $G\ t \le G\ a * exp\ (K * (t - a))$
apply (*rule derivative-quotient-bound*)
using ‹$t \in \{a..b\}$›
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 *indefinite-integral2-continuous*:
fixes f::*real* $\Rightarrow$ *'a*::*banach*
assumes f *integrable-on* $\{a..b\}$
shows *continuous-on* $\{a..b\}$ ($\lambda x.$ *integral* $\{x..b\}$ f)
proof –
have $*$: *integral* $\{x..b\}$ f = *integral* $\{a\ ..\ b\}$ f − *integral* $\{a\ ..\ x\}$ f **if** $a \le x$ $x \le b$ **for** x
using *integral-combine*[*of a x b* **for** *x*, *OF that assms*]
by (*simp add*: *algebra-simps*)
show *?thesis*
by (*subst continuous-on-cong*[*OF refl* $*$])
(*auto intro*!: *continuous-intros indefinite-integral-continuous assms*)
qed

theorem *integral2-has-vector-derivative*:
fixes f :: *real* $\Rightarrow$ *'b*::*banach*
assumes *continuous-on* $\{a..b\}$ f
and $x \in \{a..b\}$
shows (($\lambda u.$ *integral* $\{u..b\}$ f) *has-vector-derivative* $-\ f\ x$) (*at* x *within* $\{a..b\}$)
proof –
have $*$: *integral* $\{x..b\}$ f = *integral* $\{a\ ..\ b\}$ f − *integral* $\{a\ ..\ x\}$ f **if** $a \le x$ $x \le$

b **for** *x*
using *integral-combine*[*of a x b* **for** *x*, *OF that integrable-continuous-real*[*OF assms*(*1*)]]
by (*simp add*: *algebra-simps*)
show *?thesis*
using ‹*x* ∈ -›
by (*subst has-vector-derivative-cong*[*OF* - ∗ *refl refl refl*])
(*auto intro*!: *derivative-eq-intros indefinite-integral-continuous assms integral-has-vector-derivative*)
qed

lemma *integral-has-real-derivative-left*:
assumes *continuous-on* $\{a..b\}$ *g*
assumes $t \in \{a..b\}$
shows ((λx. *integral* $\{x..b\}$ *g*) *has-real-derivative* $-g\ t$) (*at t within* $\{a..b\}$)
using *integral2-has-vector-derivative*[*OF assms*]
by (*auto simp*: *has-field-derivative-iff-has-vector-derivative*)

lemma *gronwall-general-left*:
fixes *g K C a b* **and** *t*::*real*
defines $G \equiv \lambda t.\ C + K * integral\ \{t..b\}\ (\lambda s.\ g\ s)$
assumes *g-le-G*: $\bigwedge t.\ t \in \{a..b\} \Longrightarrow g\ t \le G\ t$
assumes *g-cont*: *continuous-on* $\{a..b\}$ *g*
assumes *g-nonneg*: $\bigwedge t.\ t \in \{a..b\} \Longrightarrow 0 \le g\ t$
assumes *pos*: $0 < C\ K > 0$
assumes $t \in \{a..b\}$
shows $g\ t \le C * exp\ (-K * (t - b))$
proof −
have *G-pos*: $\bigwedge t.\ t \in \{a..b\} \Longrightarrow 0 < G\ t$
by (*auto simp*: *G-def intro*!: *add-pos-nonneg mult-nonneg-nonneg integral-nonneg integrable-continuous-real assms intro*: *less-imp-le continuous-on-subset*)
have $g\ t \le G\ t$ **using** *assms* **by** *auto*
also
{
have *abc*: $\bigwedge t.\ t \in \{a..b\} \Longrightarrow$ (*G has-real-derivative* $-K * g\ t$) (*at t within* $\{a..b\}$)
by (*auto intro*!: *derivative-eq-intros integral-has-real-derivative-left g-cont simp add*: *G-def*)
moreover
{
fix *t* **assume** $t \in \{a..b\}$
hence $K * g\ t\ /\ G\ t \le K * G\ t\ /\ G\ t$
using *pos g-le-G G-pos*
by (*intro divide-right-mono mult-left-mono*) (*auto intro*!: *less-imp-le*)
also have $\ldots = K$
using *G-pos*[*of t*] ‹$t \in \{a\ ..\ b\}$› **by** *simp*
finally have $K * g\ t\ /\ G\ t \le K$ **.**
hence $-K \le -K * g\ t\ /\ G\ t$
by *simp*

}
ultimately
have $G\ t \le G\ b * exp\ (-K * (t - b))$
apply (*rule derivative-quotient-bound-left*)
using ‹$t \in \{a..b\}$›
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 $\bigwedge t.\ t \in closed\text{-}segment\ a\ b \Longrightarrow g\ t \le C + K * integral\ (closed\text{-}segment\ a\ t)\ g$
and *continuous-on* (*closed-segment a b*) *g*
and $\bigwedge t.\ t \in closed\text{-}segment\ a\ b \Longrightarrow 0 \le g\ t$
and $0 < C$
and $0 < K$
and $t \in closed\text{-}segment\ a\ b$
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-real*)
show *?thesis*
unfolding ∗
using *assms*
by (*intro gronwall-general*[**where** *b*=*b*]) (*auto intro*!: *simp*: *closed-segment-real* ‹$a \le b$›)
next
assume $\neg a \le b$
then have ∗: $K * abs\ (t - a) = - K * (t - a)$ **using** *assms* **by** (*auto simp*: *closed-segment-real 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-real)
qed

lemma gronwall-more-general-segment:
  fixes a b c::real
  assumes ⋀t. t ∈ closed-segment a b ⟹ g t ≤ C + K ∗ integral (closed-segment c t) g
    and cont: continuous-on (closed-segment a b) g
    and ⋀t. t ∈ closed-segment a b ⟹ 0 ≤ g t
    and 0 < C
    and 0 < K
    and t: t ∈ closed-segment a b
    and c: c ∈ closed-segment a b
  shows g t ≤ C ∗ exp (K ∗ abs (t − c))
proof −
  from t c have t ∈ closed-segment c a ∨ t ∈ closed-segment c b
    by (auto simp: closed-segment-real 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-real 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-real)
    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 ≡ λt. C + K ∗ integral {0..t} (λs. g s)
  assumes g-le-G: ⋀t. 0 ≤ t ⟹ t ≤ a ⟹ g t ≤ G t
  assumes g-cont: continuous-on {0..a} g
  assumes g-nonneg: ⋀t. 0 ≤ t ⟹ t ≤ a ⟹ 0 ≤ g t
  assumes pos: 0 < C 0 < K
  assumes 0 ≤ t t ≤ a
  shows g t ≤ 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*

lemma
fixes g::*real* $\Rightarrow$ $'a$::*banach*
assumes $a \le b$
assumes *cf*[*continuous-intros*]: *continuous-on* $\{a\ ..\ b\}$ f
assumes *cg*[*continuous-intros*]: *continuous-on* $\{a\ ..\ b\}$ g
assumes f: $\bigwedge x.\ a \le x \Longrightarrow x \le b \Longrightarrow$
(F *has-real-derivative* $f\ x$) (*at* x *within* $\{a\ ..\ b\}$)
assumes g: $\bigwedge x.\ a \le x \Longrightarrow x \le b \Longrightarrow$
(G *has-vector-derivative* $g\ x$) (*at* x *within* $\{a\ ..\ b\}$)
shows *integral-by-parts*: *integral* $\{a\ ..\ b\}$ $(\lambda x.\ F\ x *_R g\ x) =$
$F\ b *_R G\ b - F\ a *_R G\ a -$ *integral* $\{a\ ..\ b\}$ $(\lambda x.\ f\ x *_R G\ x)$ (**is** *?th1*)
and *has-integral-by-parts*: $((\lambda x.\ F\ x *_R g\ x)$ *has-integral*
$F\ b *_R G\ b - F\ a *_R G\ a -$ *integral* $\{a\ ..\ b\}$ $(\lambda x.\ f\ x *_R G\ x))$ $\{a\ ..\ b\}$
(**is** *?th2*)
proof −
have [*continuous-intros*]: *continuous-on* $\{a..b\}$ F *continuous-on* $\{a..b\}$ G
by (*auto intro*!: *has-vector-derivative-continuous-on f g*
simp: *has-field-derivative-iff-has-vector-derivative*[*symmetric*])
have *integrable*:
$(\lambda x.\ F\ x *_R g\ x)$ *integrable-on* $\{a\ ..\ b\}$
$(\lambda x.\ f\ x *_R G\ x)$ *integrable-on* $\{a\ ..\ b\}$
by (*auto intro*!: *integrable-continuous-real continuous-intros*)
hence *integral* $\{a..b\}$ $(\lambda x.\ F\ x *_R g\ x)$ + *integral* $\{a..b\}$ $(\lambda x.\ f\ x *_R G\ x) =$
integral $\{a..b\}$ $(\lambda x.\ F\ x *_R g\ x + f\ x *_R G\ x)$
by (*rule integral-add*[*symmetric*])
also
note *prod* = *has-vector-derivative-scaleR*[*OF f g*, *rule-format*]
have $((\lambda x.\ F\ x *_R g\ x + f\ x *_R G\ x)$ *has-integral* $F\ b *_R G\ b - F\ a *_R G\ a)$ $\{a..b\}$
by (*rule fundamental-theorem-of-calculus*[*rule-format*, *OF* ‹$a \le b$› *prod*]) *auto*
from *integral-unique*[*OF this*]
have *integral* $\{a..b\}$ $(\lambda x.\ F\ x *_R g\ x + f\ x *_R G\ x) = F\ b *_R G\ b - F\ a *_R G\ a$.

finally
show *th1*: *?th1*
by (*simp add*: *algebra-simps*)
show *?th2*
unfolding *th1*[*symmetric*]
by (*auto intro*!: *integrable-integral integrable-continuous-real continuous-intros*)
qed

2.16 conditionally complete lattice

lemma *bounded-imp-bdd-above*:
bounded S $\Longrightarrow$ *bdd-above* (*S* :: *'a*::*ordered-euclidean-space set*)
by (*auto intro*: *bdd-above-mono dest*!: *bounded-subset-cbox*)

lemma *bounded-imp-bdd-below*:
bounded S $\Longrightarrow$ *bdd-below* (*S* :: *'a*::*ordered-euclidean-space set*)
by (*auto intro*: *bdd-below-mono dest*!: *bounded-subset-cbox*)

lemma *bdd-above-cmult*:
$0 \leq$ (*a* :: *'a* :: *ordered-semiring*) $\Longrightarrow$ *bdd-above S* $\Longrightarrow$
bdd-above (($\lambda x.\ a * x$) ‘ *S*)
by (*metis bdd-above-def bdd-aboveI2 mult-left-mono*)

lemma *Sup-real-mult*:
fixes *a*::*real*
assumes $0 \leq a$
assumes $S \neq \{\}$ *bdd-above S*
shows $a *$ *Sup S* = *Sup* (($\lambda x.\ a * x$) ‘ *S*)
using *assms*
proof *cases*
assume $a = 0$ **with** ‹$S \neq \{\}$› **show** *?thesis*
by (*simp add*: *cSUP-const*)
next
assume $a \neq 0$
with ‹$0 \leq a$› **have** $0 < a$
by *simp*
show *?thesis*
proof (*intro antisym*)
have *Sup S* $\leq$ *Sup* (*op* $*$ *a* ‘ *S*) / *a* **using** *assms*
by (*intro cSup-least mult-imp-le-div-pos cSup-upper*)
(*auto simp*: *bdd-above-cmult assms* ‹$0 < a$› *less-imp-le*)
thus $a *$ *Sup S* $\leq$ *Sup* (*op* $*$ *a* ‘ *S*)
by (*simp add*: *ac-simps pos-le-divide-eq*[*OF* ‹$0<a$›])
qed (*insert assms* ‹$0 < a$›, *auto intro*!: *cSUP-least cSup-upper*)
qed

lemma (**in** *conditionally-complete-lattice*) *cInf-insert2*:
$X \neq \{\}$ $\Longrightarrow$ *bdd-below X* $\Longrightarrow$ *Inf* (*insert a* (*insert b X*)) = *inf* (*inf a b*) (*Inf X*)
by (*simp add*: *local.cInf-insert local.inf-assoc*)

lemma (**in** *conditionally-complete-lattice*) *cSup-insert2*:
X ≠ {} ⟹ *bdd-above X* ⟹ *Sup* (*insert a* (*insert b X*)) = *sup* (*sup a b*) (*Sup X*)
by (*simp add*: *local.cSup-insert-If local.sup-assoc*)

lemma (**in** *conditionally-complete-lattice*) *Inf-set-fold-inf*:
shows *Inf* (*set* (*x#xs*)) = *fold inf xs x*
using *local.Inf-fin.set-eq-fold local.cInf-eq-Inf-fin* **by** *auto*

lemma (**in** *conditionally-complete-lattice*) *Sup-set-fold-sup*:
shows *Sup* (*set* (*x#xs*)) = *fold sup xs x*
using *local.Sup-fin.set-eq-fold local.cSup-eq-Sup-fin* **by** *auto*

2.17 Banach on type class

lemma *banach-fix-type*:
fixes *f*::'*a*::*complete-space*⇒'*a*
assumes *c*:*0* ≤ *c c* < *1*
and *lipschitz*:∀ *x*. ∀ *y*. *dist* (*f x*) (*f y*) ≤ *c* ∗ *dist x y*
shows ∃!*x*. (*f x* = *x*)
using *assms banach-fix*[*OF complete-UNIV UNIV-not-empty assms*(*1*,*2*) *subset-UNIV*, *of f*]
by *auto*

2.18 Float

definition *trunc p s* =
(*let d* = *truncate-down p s in*
let u = *truncate-up p s in*
let ed = *abs* (*s* − *d*) *in*
let eu = *abs* (*u* − *s*) *in*
if abs (*s* − *d*) < *abs* (*u* − *s*) *then* (*d*, *truncate-up p ed*) *else* (*u*, *truncate-up p eu*))

lemma *trunc-nonneg*: *0* ≤ *s* ⟹ *0* ≤ *trunc p s*
by (*auto simp*: *trunc-def Let-def zero-prod-def truncate-down-def round-down-nonneg intro*!: *truncate-up-le*)

definition *trunc-err p f* = *f* − (*fst* (*trunc p f*))

lemma *trunc-err-eq*:
fst (*trunc p f*) + (*trunc-err p f*) = *f*
by (*auto simp*: *trunc-err-def*)

lemma *trunc-err-le*:
abs (*trunc-err p f*) ≤ *snd* (*trunc p f*)
apply (*auto simp*: *trunc-err-def trunc-def Let-def*)
apply (*metis truncate-up*)
by (*metis abs-minus-commute truncate-up*)

lemma *trunc-err-eq-zero-iff*:
trunc-err p f = *0* ⟷ *snd* (*trunc p f*) = *0*
apply (*auto simp*: *trunc-err-def trunc-def Let-def*)
apply (*metis abs-le-zero-iff eq-iff-diff-eq-0 truncate-up*)
apply (*metis abs-le-zero-iff eq-iff-diff-eq-0 truncate-up*)
done

lemma *mantissa-Float-0*[*simp*]: *mantissa* (*Float 0 e*) = *0*
by (*metis real-of-float-inverse float-zero mantissa-eq-zero-iff zero-float-def*)

2.19 Lists

lemma *listsum-nonneg*:
assumes *nn*:
($\bigwedge$*x*. *x* ∈ *set xs* ⟹ *f x* ≥ (*0*::'*a*::{*monoid-add*, *ordered-ab-semigroup-add*}))
shows *0* ≤ *listsum* (*map f xs*)
proof −
have *0* = *listsum* (*map* (λ-. *0*) *xs*)
by (*induct xs*) *auto*
also have ... ≤ *listsum* (*map f xs*)
by (*rule listsum-mono*) (*rule assms*)
finally show *?thesis* **.**
qed

2.20 Set(sum)

lemma *setsum-eq-nonzero*: *finite A* ⟹ ($\sum$ *a*∈*A*. *f a*) = ($\sum$ *a*∈{*a*∈*A*. *f a* ≠ *0*}. *f a*)
by (*subst setsum.mono-neutral-cong-right*) *auto*

lemma *singleton-subsetI*:*i* ∈ *B* ⟹ {*i*} ⊆ *B*
by *auto*

2.21 Max

lemma *max-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: (*rel-fun A* (*rel-fun A* (*op* =))) (*op* ≤) (*op* ≤)
shows (*rel-fun A* (*rel-fun A A*)) *max max*
unfolding *max-def*[*abs-def*]
by *transfer-prover*

lemma *max-power2*: **fixes** *a b*::*real* **shows** (*max* (*abs a*) (*abs b*))^2 = *max* (a^2) (b^2)
by (*auto simp*: *max-def abs-le-square-iff*)

2.22 Uniform Limit

lemmas *bounded-linear-uniform-limit-intros*[*uniform-limit-intros*] =
bounded-linear.*uniform-limit*[*OF bounded-linear-blinfun-apply*]

bounded-linear.uniform-limit[*OF blinfun.bounded-linear-right*]
bounded-linear.uniform-limit[*OF bounded-linear-vec-nth*]
bounded-linear.uniform-limit[*OF bounded-linear-component-cart*]
bounded-linear.uniform-limit[*OF bounded-linear-apply-blinfun*]
bounded-linear.uniform-limit[*OF bounded-linear-blinfun-matrix*]

2.23 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*)

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

2.24 (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**
$\bigwedge J\ a\ u\ (f::real*real \Rightarrow real*real).$
$(\bigwedge x.\ FDERIV\ f\ x :> J\ x) \Longrightarrow$

$(\exists\, t \in \{0 <..< 1\}.\ f\ (a + u) - f\ a = J\ (a + t *_R u)\ u)$
shows *False*
proof −
have $\bigwedge t{::}real{*}real.\ FDERIV\ (\lambda t.\ (cos\ (fst\ t),\ sin\ (fst\ t)))\ t :> (\lambda h.\ (-\ ((fst\ h) * sin\ (fst\ t)),\ (fst\ h) * cos\ (fst\ t)))$
by (*auto intro*!: *derivative-eq-intros*)
from *assms*[*OF this*, *of* (*1*, *1*) (*1*, *1*)] **obtain** *t*::*real* **where** *t*: $0 < t\ t < 1$ **and**
$cos\ 1 - cos\ 2 = sin\ (1 + t)\ sin\ 2 - sin\ 1 = cos\ (1 + t)$
by *auto*
moreover have $t \in \{0..0.3\} \longrightarrow cos\ (1 + t) > sin\ 2 - sin\ 1$
$t \in \{0.3..0.7\} \longrightarrow sin\ (1 + t) > cos\ 1 - cos\ 2$
$t \in \{0.7..0.9\} \longrightarrow cos\ (1 + t) < sin\ 2 - sin\ 1$
$t \in \{0.9..1\} \longrightarrow sin\ (1 + t) < cos\ 1 - cos\ 2$
by (*approximation 80*)+
ultimately show *?thesis* **by** *auto*
qed

lemma *MVT-wrong2*: **assumes**
$\bigwedge J\ a\ u\ (f{::}real{*}real \Rightarrow real{*}real).$
$(\bigwedge x.\ FDERIV\ f\ x :> J\ x) \Longrightarrow$
$(\exists\, x \in \{a..a{+}u\}.\ f\ (a + u) - f\ a = J\ x\ u)$
shows *False*
proof −
have $\bigwedge t{::}real{*}real.\ FDERIV\ (\lambda t.\ (cos\ (fst\ t),\ sin\ (fst\ t)))\ t :> (\lambda h.\ (-\ ((fst\ h) * sin\ (fst\ t)),\ (fst\ h) * cos\ (fst\ t)))$
by (*auto intro*!: *derivative-eq-intros*)
from *assms*[*OF this*, *of* (*1*, *1*) (*1*, *1*)] **obtain** *x*::*real* **where** *x*: $1 \le x\ x \le 2$ **and**
$cos\ 2 - cos\ 1 = -\ sin\ x\ sin\ 2 - sin\ 1 = cos\ x$
by *auto*
moreover have
$x \in \{1\ ..\ 1.5\} \longrightarrow cos\ x > sin\ 2 - sin\ 1$
$x \in \{1.5\ ..\ 1.6\} \longrightarrow -\ sin\ x < cos\ 2 - cos\ 1$
$x \in \{1.6\ ..\ 2\} \longrightarrow cos\ x < sin\ 2 - sin\ 1$
by (*approximation 80*)+
ultimately show *?thesis* **by** *auto*
qed

lemma *MVT-corrected*:
fixes *f*::*'a*::*ordered-euclidean-space*⇒*'b*::*euclidean-space*
assumes *fderiv*: $\bigwedge x.\ x \in D \Longrightarrow (f\ has\text{-}derivative\ J\ x)\ (at\ x\ within\ D)$
assumes *line-in*: $\bigwedge x.\ x \in \{0..1\} \Longrightarrow a + x *_R u \in D$
shows $(\exists\, t \in Basis \rightarrow \{0 <..< 1\}.\ (f\ (a + u) - f\ a) = (\sum i \in Basis.\ (J\ (a + t\ i *_R u)\ u \cdot i) *_R i))$
proof −
{
fix *i*::*'b*
assume $i \in Basis$
have *subset*: $((\lambda x.\ a + x *_R u)\ `\ \{0..1\}) \subseteq D$

using *line-in* **by** *force*
have $\forall x \in \{0 .. 1\}$. $((\lambda b.\ f\ (a + b *_R u) \cdot i)$ *has-derivative* $(\lambda b.\ b *_R J\ (a + x *_R u)\ u \cdot i))$ $($*at x within* $\{0..1\})$
using *line-in*
by (*auto intro*!: *derivative-eq-intros*
has-derivative-subset[*OF - subset*]
has-derivative-in-compose[**where** $f{=}\lambda x.\ a + x *_R u$]
fderiv line-in
simp add: *linear.scaleR*[*OF has-derivative-linear*[*OF fderiv*]])
with *zero-less-one*
have $\exists x \in \{0<..<1\}.\ f\ (a + 1 *_R u) \cdot i - f\ (a + 0 *_R u) \cdot i = (1 - 0) *_R J\ (a + x *_R u)\ u \cdot i$
by (*rule mvt-simple*)
}
then obtain t **where** $\forall i \in Basis.\ t\ i \in \{0<..<1\} \wedge f\ (a + u) \cdot i - f\ a \cdot i = J\ (a + t\ i *_R u)\ u \cdot i$
by *atomize-elim* (*force intro*!: *bchoice*)
hence $t \in Basis \rightarrow \{0<..<1\}$ $\bigwedge i.\ i \in Basis \Longrightarrow (f\ (a + u) - f\ a) \cdot i = J\ (a + t\ i *_R u)\ u \cdot i$
by (*auto simp*: *inner-diff-left*)
moreover hence $(f\ (a + u) - f\ a) = (\sum i \in Basis.\ (J\ (a + t\ i *_R u)\ u \cdot i) *_R i)$
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*: $\bigwedge x.\ x \in D \Longrightarrow (f$ *has-derivative* $J\ x)$ (*at x within D*)
assumes *J-ivl*: $\bigwedge x.\ x \in D \Longrightarrow J\ x\ u \in \{J0 .. J1\}$
assumes *line-in*: $\bigwedge x.\ x \in \{0..1\} \Longrightarrow a + x *_R u \in D$
shows $f\ (a + u) - f\ a \in \{J0..J1\}$
proof −
from *MVT-corrected*[*OF fderiv line-in*] **obtain** t **where**
t: $t \in Basis \rightarrow \{0<..<1\}$ **and**
mvt: $f\ (a + u) - f\ a = (\sum i \in Basis.\ (J\ (a + t\ i *_R u)\ u \cdot i) *_R i)$
by *auto*
note *mvt*
also have $\ldots \in \{J0 .. J1\}$
proof −
have *J*: $\bigwedge i.\ i \in Basis \Longrightarrow J0 \le J\ (a + t\ i *_R u)\ u$
$\bigwedge i.\ i \in Basis \Longrightarrow J\ (a + t\ i *_R u)\ u \le J1$
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
  $\bigwedge$*J J0 J1 a u* (*f*::*real*∗*real*⇒*real*∗*real*).
    ($\bigwedge$*x*. *FDERIV f x* :> *J x*) ⟹
    ($\bigwedge$*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*: ($\bigwedge$*x*. *x* ∈ *D* ⟹ (*f has-derivative J x*) (*at x within D*))
  assumes *J-ivl*: $\bigwedge$*x*. *x* ∈ *D* ⟹ *J x* (*a* − *b*) ∈ {*J0*..*J1*}
  assumes *line-in*: $\bigwedge$*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*}
    apply (*rule MVT-ivl*[*OF fderiv*])
    apply *assumption*
    apply (*rule J-ivl*) **apply** *assumption*
    using *line-in*
    apply (*auto simp*: *diff-le-eq le-diff-eq ac-simps*)
    done
  thus *?thesis*
    by (*auto simp*: *diff-le-eq le-diff-eq ac-simps*)
qed

end

3 Initial Value Problems

theory *Initial-Value-Problem*
imports ../*ODE-Auxiliarities*
begin

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-setL2*)

lemma *setsum-inner-Basis-one*: *i* ∈ *Basis* ⟹ ($\sum$*x*∈*Basis*. *x* · *i*) = *1*
  by (*subst setsum.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* $\subseteq$ *cbox* $(y - r *_R One)$ $(y + r *_R One)$
unfolding *scaleR-setsum-right interval-cbox cbox-def*
proof *safe*
fix *x i::'a* **assume** $i \in Basis$ $x \in cball\ y\ r$
with *dist-component-le*[*OF* ‹$i \in Basis$›, *of y x*]
have *dist* $(y \cdot i)$ $(x \cdot i) \leq r$ **by** *simp*
thus $(y - setsum\ (op *_R r)\ Basis) \cdot i \leq x \cdot i$
$x \cdot i \leq (y + setsum\ (op *_R r)\ Basis) \cdot i$
by (*auto simp add*: *inner-diff-left inner-add-left inner-setsum-left setsum-right-distrib*[*symmetric*] *setsum-inner-Basis-one* ‹$i \in Basis$› *dist-real-def*)
qed

lemma *centered-cbox-in-cball*:
shows *cbox* $(- r *_R One)$ $(r *_R One::'a::euclidean\text{-}space) \subseteq$
cball 0 $(sqrt(DIM('a)) * r)$
proof
fix *x::'a*
have *norm x* $\leq sqrt(DIM('a)) *$ *infnorm x*
by (*rule norm-le-infnorm*)
also
assume $x \in cbox\ (- r *_R One)\ (r *_R One)$
hence *infnorm x* $\leq r$
using *assms*
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*)
qed

3.1 Lipschitz continuity

definition *lipschitz*
where *lipschitz t f L* $\longleftrightarrow$ $(0 \leq L \wedge (\forall x \in t.\ \forall y \in t.\ dist\ (f\ x)\ (f\ y) \leq L * dist\ x\ y))$

lemma *lipschitzI*:
assumes $\bigwedge x\ y.\ x \in t \Longrightarrow y \in t \Longrightarrow dist\ (f\ x)\ (f\ y) \leq L * dist\ x\ y$
assumes $0 \leq L$
shows *lipschitz t f L*
using *assms* **unfolding** *lipschitz-def* **by** *auto*

lemma *lipschitzD*:
assumes *lipschitz t f L*
assumes $x \in t$ $y \in t$
shows $dist\ (f\ x)\ (f\ y) \leq L * dist\ x\ y$
using *assms* **unfolding** *lipschitz-def* **by** *auto*

lemma *lipschitz-nonneg*:
assumes *lipschitz t f L*
shows $0 \leq L$

```
using assms unfolding lipschitz-def by auto

lemma lipschitz-subset:
  assumes lipschitz D f L
  assumes D' ⊆ D
  shows lipschitz D' f L
  using lipschitzD[OF assms(1)] lipschitz-nonneg[OF assms(1)] assms(2)
  by (auto intro!: lipschitzI)

lemma lipschitz-imp-continuous:
  assumes lipschitz X f L
  assumes x ∈ X
  shows continuous (at x within X) f
  unfolding continuous-within-eps-delta
proof safe
  fix e::real
  assume 0 < e
  show ∃ d>0. ∀ x'∈X. dist x' x < d ⟶ dist (f x') (f x) < e
  proof (cases L > 0)
    case True
    thus ?thesis
      using ‹0 < e› using assms
      by (force intro!: exI[where x=e / L] divide-pos-pos
        dest!: lipschitzD simp: field-simps)
  next
    case False
    thus ?thesis
    proof (safe intro!: exI[where x=1] zero-less-one)
      fix x' assume x' ∈ X
      note lipschitzD[OF assms(1) ‹x' ∈ X› ‹x ∈ X›]
      also have L ∗ dist x' x ≤ 0
        using False by (auto simp: not-less mult-nonpos-nonneg)
      also note ‹0 < e›
      finally show dist (f x') (f x) < e .
    qed
  qed
qed

lemma lipschitz-imp-continuous-on:
  assumes lipschitz t f L
  shows continuous-on t f
  using lipschitz-imp-continuous[OF assms]
  by (metis continuous-on-eq-continuous-within)

lemma lipschitz-norm-leI:
  assumes lipschitz t f L
  assumes x ∈ t y ∈ t
  shows norm (f x − f y) ≤ L ∗ norm (x − y)
  using lipschitzD[OF assms]
```

by (*simp add*: *dist-norm*)

lemma *lipschitz-uminus*:
fixes *f*::- ⇒ *'b*::*real-normed-vector*
shows *lipschitz t* ($\lambda x. - f\ x$) *L* ⟷ *lipschitz t f L*
by (*auto intro*!: *lipschitzI intro*: *lipschitz-nonneg dest*: *lipschitzD*
simp: *dist-minus*)

lemma *lipschitz-uminus'*:
fixes *f*::- ⇒ *'b*::*real-normed-vector*
shows *lipschitz t* ($- f$) *L* ⟷ *lipschitz t f L*
by (*auto intro*!: *lipschitzI intro*: *lipschitz-nonneg dest*: *lipschitzD*
simp: *dist-minus*)

lemma *nonneg-lipschitz*:
assumes *lipschitz X f L*
shows *lipschitz X f* (*abs L*)
using *assms lipschitz-nonneg* **by** *fastforce*

lemma *pos-lipschitz*:
assumes *lipschitz X f L*
shows *lipschitz X f* (*abs L* + *1*)
using *assms*
proof (*auto simp*: *lipschitz-def*, *goal-cases*)
case (*1 x y*)
hence *dist* (*f x*) (*f y*) $\leq$ *L* $*$ *dist x y*
by *auto*
also have ... $\leq$ (*abs L* + *1*) $*$ *dist x y*
by (*rule mult-right-mono*) *auto*
finally show *?case* **by** (*simp add*: *lipschitz-nonneg*[*OF assms*])
qed

3.2 Local Lipschitz continuity (uniformly for a family of functions)

definition *local-lipschitz*::
'a::*metric-space set* ⇒ *'b*::*metric-space set* ⇒ (*'a* ⇒ *'b* ⇒ *'c*::*metric-space*) ⇒ *bool*
where
local-lipschitz T X f ≡ ∀ *x* ∈ *X*. ∀ *t* ∈ *T*. ∃ *u*>*0*. ∃ *L*. ∀ *t* ∈ *cball t u* ∩ *T*.
lipschitz (*cball x u* ∩ *X*) (*f t*) *L*

lemma *local-lipschitzI*:
assumes ⋀*t x*. *t* ∈ *T* ⟹ *x* ∈ *X* ⟹ ∃ *u*>*0*. ∃ *L*. ∀ *t* ∈ *cball t u* ∩ *T*. *lipschitz* (*cball x u* ∩ *X*) (*f t*) *L*
shows *local-lipschitz T X f*
using *assms*
unfolding *local-lipschitz-def*
by *auto*

lemma *local-lipschitzE*:
  assumes *local-lipschitz*: *local-lipschitz T X f*
  assumes $t \in T$ $x \in X$
  obtains *u L* **where** $u > 0$ $\bigwedge s.\ s \in cball\ t\ u \cap T \Longrightarrow lipschitz\ (cball\ x\ u \cap X)$ $(f\ s)\ L$
  using *assms local-lipschitz local-lipschitz-def*
  by *metis*

lemma *local-lipschitz-continuous-on*:
  assumes *local-lipschitz*: *local-lipschitz T X f*
  assumes $t \in T$
  shows *continuous-on X* $(f\ t)$
  unfolding *continuous-on-def*
proof *safe*
  fix *x* **assume** $x \in X$
  from *local-lipschitzE*[*OF local-lipschitz* ‹$t \in T$› ‹$x \in X$›] **obtain** *u L*
    where $0 < u$
    and *L*: $\bigwedge s.\ s \in cball\ t\ u \cap T \Longrightarrow lipschitz\ (cball\ x\ u \cap X)\ (f\ s)\ L$
    by *metis*
  have $x \in ball\ x\ u$ **using** ‹$0 < u$› **by** *simp*
  from *lipschitz-imp-continuous-on*[*OF L*]
  have *tendsto*: $(f\ t \longrightarrow f\ t\ x)\ (at\ x\ within\ cball\ x\ u \cap X)$
    using ‹$0 < u$› ‹$x \in X$› ‹$t \in T$›
    by (*auto simp*: *continuous-on-def*)
  then show $(f\ t \longrightarrow f\ t\ x)\ (at\ x\ within\ X)$
    using ‹$x \in ball\ x\ u$›
    by (*rule tendsto-within-nhd*) *auto*
qed

lemma
  local-lipschitz-compose1:
  assumes *ll*: *local-lipschitz* $(g\ `\ T)\ X\ (\lambda t.\ f\ t)$
  assumes *g*: *continuous-on T g*
  shows *local-lipschitz T X* $(\lambda t.\ f\ (g\ t))$
proof (*rule local-lipschitzI*)
  fix *t x*
  assume $t \in T$ $x \in X$
  then have $g\ t \in g\ `\ T$ **by** *simp*
  from *local-lipschitzE*[*OF assms*(*1*) *this* ‹$x \in X$›]
  obtain *u L* **where** $0 < u$ **and** *l*: $(\bigwedge s.\ s \in cball\ (g\ t)\ u \cap g\ `\ T \Longrightarrow lipschitz$ $(cball\ x\ u \cap X)\ (f\ s)\ L)$
    by *auto*
  from *g*[*unfolded continuous-on-eq-continuous-within*, *rule-format*, *OF* ‹$t \in T$›,
    unfolded continuous-within-eps-delta, *rule-format*, *OF* ‹$0 < u$›]
  obtain *d* **where** *d*: $d>0$ $\bigwedge x'.\ x'\in T \Longrightarrow dist\ x'\ t < d \Longrightarrow dist\ (g\ x')\ (g\ t) < u$
    by (*auto*)
  show $\exists u>0.\ \exists L.\ \forall t\in cball\ t\ u \cap T.\ lipschitz\ (cball\ x\ u \cap X)\ (f\ (g\ t))\ L$
    using *d* ‹$0 < u$›

```
    by (fastforce intro: exI[where x=(min d u)/2] exI[where x=L]
      intro!: less-imp-le[OF d(2)] lipschitz-subset[OF l] simp: dist-commute)
qed

context
  fixes T::'a::metric-space set and X f
  assumes local-lipschitz: local-lipschitz T X f
begin

lemma continuous-on-TimesI:
  assumes y: ⋀x. x ∈ X ⟹ continuous-on T (λt. f t x)
  shows continuous-on (T × X) (λ(t, x). f t x)
  unfolding continuous-on-iff
proof (safe, simp)
  fix a b and e::real
  assume H: a ∈ T b ∈ X 0 < e
  hence 0 < e/2 by simp
  from y[unfolded continuous-on-iff, OF ‹b ∈ X›, rule-format, OF ‹a ∈ T› ‹0 <
e/2›]
  obtain d where d: d > 0 ⋀t. t ∈ T ⟹ dist t a < d ⟹ dist (f t b) (f a b) <
e/2
    by auto

  from ‹a : T› ‹b ∈ X›
  obtain u L where u: 0 < u
    and L: ⋀t. t ∈ cball a u ∩ T ⟹ lipschitz (cball b u ∩ X) (f t) L
    by (erule local-lipschitzE[OF local-lipschitz])

  have a ∈ cball a u ∩ T by (auto simp: ‹0 < u› ‹a ∈ T› less-imp-le)
  from lipschitz-nonneg[OF L[OF ‹a ∈ cball - - ∩ -›]] have 0 ≤ L .

  let ?d = Min {d, u, (e/2/(L + 1))}
  show ∃d>0. ∀x∈T. ∀y∈X. dist (x, y) (a, b) < d ⟶ dist (f x y) (f a b) < e
  proof (rule exI[where x = ?d], safe)
    show 0 < ?d
      using ‹0 ≤ L› ‹0 < u› ‹0 < e› ‹0 < d›
      by (auto intro!: divide-pos-pos )
    fix x y
    assume x ∈ T y ∈ X
    assume dist-less: dist (x, y) (a, b) < ?d
    have dist y b ≤ dist (x, y) (a, b)
      using dist-snd-le[of (x, y) (a, b)]
      by auto
    also
    note dist-less
    also
    {
      note calculation
      also have ?d ≤ u by simp
```

```
      finally have dist y b < u .
    }
    have ?d ≤ e/2/(L + 1) by simp
    also have (L + 1) ∗ . . . ≤ e / 2
      using ‹0 < e› ‹L ≥ 0›
      by (auto simp: divide-simps)
    finally have le1: (L + 1) ∗ dist y b < e / 2 using ‹L ≥ 0› by simp

    have dist x a ≤ dist (x, y) (a, b)
      using dist-fst-le[of (x, y) (a, b)]
      by auto
    also note dist-less
    finally have dist x a < ?d .
    also have ?d ≤ d by simp
    finally have dist x a < d .
    note ‹dist x a < ?d›
    also have ?d ≤ u by simp
    finally have dist x a < u .
    then have x ∈ cball a u ∩ T
      using ‹x ∈ T›
      by (auto simp: dist-commute)
    have dist (f x y) (f a b) ≤ dist (f x y) (f x b) + dist (f x b) (f a b)
      by (rule dist-triangle)
    also have dist (f x y) (f x b) ≤ (abs L + 1) ∗ dist y b
      apply (rule lipschitzD[OF pos-lipschitz[OF L]])
      subgoal by fact
      subgoal
        using ‹y ∈ X› ‹dist y b < u›
        by (simp add: dist-commute)
      subgoal
        using ‹0 < u› ‹b ∈ X›
        by (simp add: )
      done
    also have (abs L + 1) ∗ dist y b ≤ e / 2
      using le1 ‹0 ≤ L› by simp
    also have dist (f x b) (f a b) < e / 2
      by (rule d; fact)
    also have e / 2 + e / 2 = e by simp
    finally show dist (f x y) (f a b) < e by simp
  qed
qed

lemma local-lipschitz-on-compact-implies-lipschitz:
  assumes compact X compact T
  assumes cont: ⋀x. x ∈ X ⟹ continuous-on T (λt. f t x)
  obtains L where ⋀t. t ∈ T ⟹ lipschitz X (f t) L
proof −
  {
    assume ∗: ⋀n::nat. ¬(∀ t∈T. lipschitz X (f t) n)
```

{
fix *n*::*nat*
from ∗[*of n*] **have** ∃ *x y t*. *t* ∈ *T* ∧ *x* ∈ *X* ∧ *y* ∈ *X* ∧ *dist* (*f t y*) (*f t x*) > *n*
∗ *dist y x*
by (*force simp*: *lipschitz-def*)
} **then obtain** *t* **and** *x y*::*nat* ⇒ ′*b* **where** *xy*: ⋀*n*. *x n* ∈ *X* ⋀*n*. *y n* ∈ *X*
and *t*: ⋀*n*. *t n* ∈ *T*
and *d*: ⋀*n*. *dist* (*f* (*t n*) (*y n*)) (*f* (*t n*) (*x n*)) > *n* ∗ *dist* (*y n*) (*x n*)
by *metis*
from *xy assms* **obtain** *lx rx* **where** *lx*′: *lx* ∈ *X subseq rx* (*x o rx*) ⟶ *lx*
by (*metis compact-def*)
with *xy* **have** ⋀*n*. (*y o rx*) *n* ∈ *X* **by** *auto*
with *assms* **obtain** *ly ry* **where** *ly*′: *ly* ∈ *X subseq ry* ((*y o rx*) *o ry*) ⟶
ly
by (*metis compact-def*)
with *t* **have** ⋀*n*. ((*t o rx*) *o ry*) *n* ∈ *T* **by** *simp*
with *assms* **obtain** *lt rt* **where** *lt*′: *lt* ∈ *T subseq rt* (((*t o rx*) *o ry*) *o rt*)
⟶ *lt*
by (*metis compact-def*)
from *lx*′ *ly*′
have *lx*: (*x o* (*rx o ry o rt*)) ⟶ *lx* (**is** *?x* ⟶ -)
and *ly*: (*y o* (*rx o ry o rt*)) ⟶ *ly* (**is** *?y* ⟶ -)
and *lt*: (*t o* (*rx o ry o rt*)) ⟶ *lt* (**is** *?t* ⟶ -)
apply (*simp add*: *LIMSEQ-subseq-LIMSEQ o-assoc lt*′(*2*))
apply (*simp add*: *LIMSEQ-subseq-LIMSEQ ly*′(*3*) *o-assoc lt*′(*2*))
by (*simp add*: *o-assoc lt*′(*3*))
hence (λ*n*. *dist* (*?y n*) (*?x n*)) ⟶ *dist ly lx*
by (*metis tendsto-dist*)
moreover
let *?S* = (λ(*t*, *x*). *f t x*) ‘ (*T* × *X*)
have *eventually* (λ*n*::*nat*. *n* > *0*) *sequentially*
by (*metis eventually-at-top-dense*)
hence *eventually* (λ*n*. *norm* (*dist* (*?y n*) (*?x n*)) ≤ *norm* (|*diameter ?S*| / *n*)
∗ *1*) *sequentially*
proof *eventually-elim*
case (*elim n*)
have *0* < *rx* (*ry* (*rt n*)) **using** ‹*0* < *n*›
by (*metis dual-order.strict-trans1 lt*′(*2*) *lx*′(*2*) *ly*′(*2*) *seq-suble*)
have *compact*: *compact ?S*
by (*auto intro*!: *compact-continuous-image continuous-on-subset*[*OF continuous-on-TimesI*]
compact-Times ‹*compact X*› ‹*compact T*› *cont*)
have *norm* (*dist* (*?y n*) (*?x n*)) = *dist* (*?y n*) (*?x n*) **by** *simp*
also
with *elim d*[*of rx* (*ry* (*rt n*))]
have ... < *dist* (*f* (*?t n*) (*?y n*)) (*f* (*?t n*) (*?x n*)) / *rx* (*ry* (*rt* (*n*)))
using *lx*′(*2*) *ly*′(*2*) *lt*′(*2*) ‹*0* < *rx* -›
by (*auto simp add*: *divide-simps algebra-simps subseq-def*)
also have ... ≤ *diameter ?S* / *n*
by (*force intro*!: ‹*0* < *n*› *subseq-def xy diameter-bounded-bound frac-le*

compact-imp-bounded compact t
intro: *le-trans*[*OF seq-suble*[*OF lt′*(*2*)]]
le-trans[*OF seq-suble*[*OF ly′*(*2*)]]
le-trans[*OF seq-suble*[*OF lx′*(*2*)]])
also have . . . ≤ *abs* (*diameter ?S*) / *n*
by (*auto intro*!: *divide-right-mono*)
finally show *?case* **by** *simp*
qed
with - **have** (λ*n*. *dist* (*?y n*) (*?x n*)) ⟶ *0*
by (*rule tendsto-0-le*)
(*metis tendsto-divide-0*[*OF tendsto-const*] *filterlim-at-top-imp-at-infinity filterlim-real-sequentially*)
ultimately have *lx* = *ly*
using *LIMSEQ-unique* **by** *fastforce*
with *assms lx′* **have** *lx* ∈ *X* **by** *auto*
from ‹*lt* ∈ *T*› *this* **obtain** *u L* **where** *L*: *u* > *0* ⋀*t*. *t* ∈ *cball lt u* ∩ *T* ⟹ *lipschitz* (*cball lx u* ∩ *X*) (*f t*) *L*
by (*erule local-lipschitzE*[*OF local-lipschitz*])
hence *L* ≥ *0* **by** (*force intro*!: *lipschitz-nonneg* ‹*lt* ∈ *T*›)

from *L lt ly lx* ‹*lx* = *ly*›
have
eventually (λ*n*. *?t n* ∈ *ball lt u*) *sequentially*
eventually (λ*n*. *?y n* ∈ *ball lx u*) *sequentially*
eventually (λ*n*. *?x n* ∈ *ball lx u*) *sequentially*
by (*auto simp*: *dist-commute Lim*)
moreover have *eventually* (λ*n*. *n* > *L*) *sequentially*
by (*metis filterlim-at-top-dense filterlim-real-sequentially*)
ultimately
have *eventually* (λ-. *False*) *sequentially*
proof *eventually-elim*
case (*elim n*)
hence *dist* (*f* (*?t n*) (*?y n*)) (*f* (*?t n*) (*?x n*)) ≤ *L* ∗ *dist* (*?y n*) (*?x n*)
using *assms xy t*
unfolding *dist-norm*[*symmetric*]
by (*intro lipschitzD*[*OF L*(*2*)]) *auto*
also have . . . ≤ *n* ∗ *dist* (*?y n*) (*?x n*)
using *elim* **by** (*intro mult-right-mono*) *auto*
also have . . . ≤ *rx* (*ry* (*rt n*)) ∗ *dist* (*?y n*) (*?x n*)
by (*intro mult-right-mono*[*OF - zero-le-dist*])
(*meson lt′*(*2*) *lx′*(*2*) *ly′*(*2*) *of-nat-le-iff order-trans seq-suble*)
also have . . . < *dist* (*f* (*?t n*) (*?y n*)) (*f* (*?t n*) (*?x n*))
by (*auto intro*!: *d*)
finally show *?case* **by** *simp*
qed
hence *False*
by *simp*
} **then obtain** *L* **where** ⋀*t*. *t* ∈ *T* ⟹ *lipschitz X* (*f t*) *L*
by *metis*

thus *?thesis* **..**
qed

lemma *local-lipschitz-on-subset*:
assumes $S \subseteq T$ $Y \subseteq X$
shows *local-lipschitz S Y f*
proof (*rule local-lipschitzI*)
fix t x **assume** $t \in S$ $x \in Y$
then have $t \in T$ $x \in X$ **using** *assms* **by** *auto*
from *local-lipschitzE*[*OF local-lipschitz*, *OF this*]
obtain u L **where** u: $0 < u$ **and** L: $\bigwedge s.\ s \in cball\ t\ u \cap T \Longrightarrow lipschitz\ (cball\ x\ u \cap X)\ (f\ s)\ L$
by *blast*
show $\exists u>0.\ \exists L.\ \forall t \in cball\ t\ u \cap S.\ lipschitz\ (cball\ x\ u \cap Y)\ (f\ t)\ L$
using *assms*
by (*auto intro*: *exI*[**where** $x=u$] *exI*[**where** $x=L$] *intro*!: *u lipschitz-subset*[*OF* - *Int-mono*[*OF order-refl* ‹$Y \subseteq X$›]] *L*)
qed

end

lemma *local-lipschitz-uminus*:
fixes f::'*a*::*metric-space* ⇒ '*b*::*metric-space* ⇒ '*c*::*real-normed-vector*
shows *local-lipschitz T X* $(\lambda t\ x.\ -\ f\ t\ x)$ = *local-lipschitz T X f*
by (*auto simp*: *local-lipschitz-def lipschitz-uminus*)

lemma *lipschitz-PairI*:
assumes f: *lipschitz A f L*
assumes g: *lipschitz A g M*
shows *lipschitz A* $(\lambda a.\ (f\ a,\ g\ a))$ $(sqrt\ (L^2 + M^2))$
proof (*rule lipschitzI*, *goal-cases*)
case (*1 x y*)
have $dist\ (f\ x,\ g\ x)\ (f\ y,\ g\ y) = sqrt\ ((dist\ (f\ x)\ (f\ y))^2 + (dist\ (g\ x)\ (g\ y))^2)$
by (*auto simp add*: *dist-Pair-Pair real-le-lsqrt*)
also have $\ldots \le sqrt\ ((L * dist\ x\ y)^2 + (M * dist\ x\ y)^2)$
by (*auto intro*!: *real-sqrt-le-mono add-mono power-mono 1 lipschitzD f g*)
also have $\ldots \le sqrt\ (L^2 + M^2) * dist\ x\ y$
by (*auto simp*: *power-mult-distrib ring-distribs*[*symmetric*] *real-sqrt-mult*)
finally show *?case* **.**
qed *simp*

lemma *local-lipschitz-PairI*:
assumes f: *local-lipschitz A B* $(\lambda a\ b.\ f\ a\ b)$
assumes g: *local-lipschitz A B* $(\lambda a\ b.\ g\ a\ b)$
shows *local-lipschitz A B* $(\lambda a\ b.\ (f\ a\ b,\ g\ a\ b))$
proof (*rule local-lipschitzI*)
fix t x **assume** $t \in A$ $x \in B$
from *local-lipschitzE*[*OF f this*] *local-lipschitzE*[*OF g this*]
obtain u L v M **where** $0 < u$ $(\bigwedge s.\ s \in cball\ t\ u \cap A \Longrightarrow lipschitz\ (cball\ x\ u\ \cap$

B) (*f s*) *L*)
0 < *v* (⋀*s*. *s* ∈ *cball t v* ∩ *A* ⟹ *lipschitz* (*cball x v* ∩ *B*) (*g s*) *M*)
by *metis*
then show ∃ *u*>*0*. ∃ *L*. ∀ *t*∈*cball t u* ∩ *A*. *lipschitz* (*cball x u* ∩ *B*) (λ*b*. (*f t b*, *g t b*)) *L*
by (*intro exI*[**where** *x*=*min u v*])
(*force intro*: *lipschitz-subset intro*!: *lipschitz-PairI*)
qed

lemma *lipschitz-constI*: *lipschitz A* (λ*x*. *c*) *0*
by (*auto simp*: *lipschitz-def*)

lemma *local-lipschitz-constI*: *local-lipschitz S T* (λ*t x*. *f t*)
by (*auto simp*: *intro*!: *local-lipschitzI lipschitz-constI intro*: *exI*[**where** *x*=*1*])

lemma (**in** *bounded-linear*) *lipschitz-boundE*:
obtains *B* **where** *lipschitz A f B*
proof −
from *nonneg-bounded*
obtain *B* **where** *B*: *B* ≥ *0* ⋀*x*. *norm* (*f x*) ≤ *B* ∗ *norm x*
by (*auto simp*: *ac-simps*)
have *lipschitz A f B*
by (*auto intro*!: *lipschitzI B simp*: *dist-norm diff*[*symmetric*])
thus *?thesis* **..**
qed

lemma (**in** *bounded-linear*) *local-lipschitzI*:
shows *local-lipschitz A B* (λ-. *f*)
proof (*rule local-lipschitzI*, *goal-cases*)
case (*1 t x*)
from *lipschitz-boundE*[*of* (*cball x 1* ∩ *B*)] **obtain** *C* **where** *lipschitz* (*cball x 1* ∩ *B*) *f C* **by** *auto*
then show *?case*
by (*auto intro*: *exI*[**where** *x*=*1*])
qed

lemma *c1-implies-local-lipschitz*:
fixes *T*::*real set* **and** *X*::′*a*::{*banach*,*heine-borel*} *set*
and *f*::*real* ⇒ ′*a* ⇒ ′*a*
assumes *f*′: ⋀*t x*. *t* ∈ *T* ⟹ *x* ∈ *X* ⟹ (*f t has-derivative blinfun-apply* (*f*′ (*t*, *x*))) (*at x*)
assumes *cont-f*′: *continuous-on* (*T* × *X*) *f*′
assumes *open T*
assumes *open X*
shows *local-lipschitz T X f*
proof (*rule local-lipschitzI*)
fix *t x*
assume *t* ∈ *T x* ∈ *X*
from *open-contains-cball*[*THEN iffD1*, *OF* ‹*open X*›, *rule-format*, *OF* ‹*x* ∈ *X*›]

obtain *u* **where** *u*: *u* > *0* *cball x u* ⊆ *X* **by** *auto*
moreover
from *open-contains-cball*[*THEN iffD1*, *OF* ‹*open T*›, *rule-format*, *OF* ‹*t* ∈ *T*›]
obtain *v* **where** *v*: *v* > *0* *cball t v* ⊆ *T* **by** *auto*
ultimately
have *compact* (*cball t v* × *cball x u*) *cball t v* × *cball x u* ⊆ *T* × *X*
by (*auto intro*!: *compact-Times*)
then have *compact* (*f'* ‘ (*cball t v* × *cball x u*))
by (*auto intro*!: *compact-continuous-image continuous-on-subset*[*OF cont-f'*])
then obtain *B* **where** *B*: *B* > *0* ⋀*s y*. *s* ∈ *cball t v* ⟹ *y* ∈ *cball x u* ⟹ *norm* (*f'* (*s*, *y*)) ≤ *B*
by (*auto dest*!: *compact-imp-bounded simp*: *bounded-pos simp del*: *mem-cball*)

{
fix *s* **assume** *s*: *s* ∈ *cball t v*
also note ‹. . . ⊆ *T*›
finally
have *deriv*: ∀ *y*∈*cball x u*. (*f s has-derivative blinfun-apply* (*f'* (*s*, *y*))) (*at y within cball x u*)
using ‹- ⊆ *X*›
by (*auto intro*!: *has-derivative-at-within*[*OF f'*])
have *norm* (*f s y* − *f s z*) ≤ *B* ∗ *norm* (*y* − *z*)
if *y* ∈ *cball x u z* ∈ *cball x u*
for *y z*
using *s that*
by (*intro differentiable-bound*[*OF convex-cball deriv*])
(*auto intro*!: *B simp*: *norm-blinfun.rep-eq*[*symmetric*])
then have *lipschitz* (*cball x* (*min u v*) ∩ *X*) (*f s*) *B*
using ‹*0* < *B*›
by (*auto intro*!: *lipschitzI simp*: *dist-norm*)
} **note** *lipschitz* = *this*
show ∃ *u*>*0*. ∃ *L*. ∀ *t*∈*cball t u* ∩ *T*. *lipschitz* (*cball x u* ∩ *X*) (*f t*) *L*
by (*force intro*: *exI*[**where** *x*=*min u v*] *exI*[**where** *x*=*B*] *intro*!: *lipschitz simp*: *u v*)
qed

3.3 Solutions of IVPs

record *'a ivp* =
ivp-f :: *real* × *'a* ⇒ *'a*
ivp-t0 :: *real*
ivp-x0 :: *'a*
ivp-T :: *real set*
ivp-X :: *'a set*

locale *ivp* =
fixes *i*::*'a*::*banach ivp*
assumes *iv-defined*: *ivp-t0 i* ∈ *ivp-T i ivp-x0 i* ∈ *ivp-X i*
begin

abbreviation *t0* ≡ *ivp-t0 i*
abbreviation *x0* ≡ *ivp-x0 i*
abbreviation *T* ≡ *ivp-T i*
abbreviation *X* ≡ *ivp-X i*
abbreviation *f* ≡ *ivp-f i*

definition *is-solution* **where** *is-solution x* ⟷
  x t0 = *x0* ∧
  (∀ *t*∈*T*.
    (*x has-vector-derivative f* (*t*, *x t*))
      (*at t within T*) ∧
     x t ∈ *X*)

definition *solution* = (*SOME x*. *is-solution x*)

lemma *is-solutionD*:
  assumes *is-solution x*
  shows
  x t0 = *x0*
  ⋀*t*. *t* ∈ *T* ⟹ (*x has-vector-derivative f* (*t*, *x t*)) (*at t within T*)
  ⋀*t*. *t* ∈ *T* ⟹ *x t* ∈ *X*
  using *assms*
  by (*auto simp*: *is-solution-def*)

lemma *solution-continuous-on*[*intro*, *simp*]:
  assumes *is-solution x*
  shows *continuous-on T x*
using *is-solutionD*[*OF assms*]
by (*auto intro*!: *differentiable-imp-continuous-on*
  simp add: *differentiable-on-def differentiable-def has-vector-derivative-def*)
  blast

lemma *is-solutionI*[*intro*]:
  assumes *x t0* = *x0*
  assumes ⋀*t*. *t* ∈ *T* ⟹
    (*x has-vector-derivative f* (*t*, *x t*)) (*at t within T*)
  assumes ⋀*t*. *t* ∈ *T* ⟹ *x t* ∈ *X*
  shows *is-solution x*
  using *assms*
  unfolding *is-solution-def* **by** *simp*

lemma *is-solution-cong*:
  assumes ⋀*t*. *t* ∈ *T* ⟹ *x t* = *y t*
  shows *is-solution x* = *is-solution y*
proof –
  { fix *t* **assume** *t* ∈ *T*
    hence (*y has-vector-derivative f* (*t*, *y t*)) (*at t within T*) =
      (*x has-vector-derivative f* (*t*, *y t*)) (*at t within T*)

using *assms*
by (*subst has-vector-derivative-cong*) *auto* **}**
thus *?thesis* **using** *assms iv-defined* **by** (*auto simp*: *is-solution-def*)
qed

lemma *solution-on-subset*:
assumes $t0 \in T'$
assumes $T' \subseteq T$
assumes *is-solution x*
shows *ivp.is-solution* (*i*(|*ivp-T* := *T'*|)) *x*
proof −
interpret *ivp'*: *ivp i*(|*ivp-T* := *T'*|) **using** *assms iv-defined*
by *unfold-locales simp-all*
show *?thesis*
using *assms is-solutionD*[*OF* ‹*is-solution x*›]
by (*intro ivp'.is-solutionI*) (*auto intro*:
has-vector-derivative-within-subset[**where** *s*=*T*])
qed

lemma *solution-on-subset'*:
assumes $t0 \in$ *ivp-T i'*
assumes *ivp-T i'* $\subseteq T$
assumes *is-solution x*
assumes *i'* = *i*(|*ivp-T*:=*ivp-T i'*|)
shows *ivp.is-solution i' x*
by (*subst assms*) (*auto intro*!: *solution-on-subset assms*)

lemma *is-solution-on-superset-domain*:
assumes *is-solution y*
assumes $X \subseteq X'$
shows *ivp.is-solution* (*i* (|*ivp-X* := *X'*|)) *y*
proof −
interpret *ivp'*: *ivp i*(|*ivp-X*:=*X'*|) **using** *assms iv-defined*
by *unfold-locales auto*
show *?thesis*
using *assms*
by (*auto simp*: *is-solution-def ivp'.is-solution-def*)
qed

lemma *restriction-of-solution*:
assumes $t1 \in T'$
assumes *x t1* $\in X$
assumes $T' \subseteq T$
assumes *x-sol*: *is-solution x*
shows *ivp.is-solution* (*i*(|*ivp-t0*:=*t1*, *ivp-x0*:=*x t1*, *ivp-T*:=*T'*|)) *x*
proof −
interpret *ivp'*: *ivp i*(|*ivp-t0*:=*t1*, *ivp-x0*:=*x t1*, *ivp-T*:=*T'*|)
using *assms iv-defined is-solutionD*[*OF x-sol*]
by *unfold-locales simp-all*

```
  show ?thesis
    using is-solutionD[OF x-sol] assms
    by (intro ivp'.is-solutionI)
      (auto intro: has-vector-derivative-within-subset[where t=T' and s=T])
qed

lemma mirror-solution:
  defines mirror ≡ λt. 2 * t0 − t
  defines mi ≡ i(|ivp-f := (λ(t, x). − f (mirror t, x)), ivp-T := mirror ‘ T|)
  assumes sol: is-solution x
  shows ivp.is-solution mi (x o mirror)
proof −
  interpret mi: ivp mi
    using iv-defined
    by unfold-locales (auto simp: mi-def mirror-def)
  show ?thesis
    using is-solutionD[OF sol]
  proof (intro mi.is-solutionI)
    fix t
    assume t ∈ mi.T
    from is-solutionD[OF sol]
    have *: ⋀t. t ∈ T ⟹
      (x has-derivative (λa. a *R f (t, x t))) (at t within T)
      by (auto simp: has-vector-derivative-def)
    show (x o mirror has-vector-derivative mi.f (t, (x o mirror) t))
        (at t within mi.T)
      using ‹t ∈ mi.T›
      by (auto simp: mi-def mirror-def has-vector-derivative-def
        intro!: derivative-eq-intros has-derivative-subset[OF *])
  qed (auto simp: mirror-def mi-def)
qed

lemma solution-mirror:
  defines mirror ≡ λt. 2 * t0 − t
  defines mi ≡ i(|ivp-f := (λ(t, x). − f (mirror t, x)), ivp-T := mirror ‘ T|)
  assumes misol: ivp.is-solution mi (x o mirror)
  shows is-solution x
proof −
  interpret mi: ivp mi
    using iv-defined
    by unfold-locales (auto simp: mi-def mirror-def)
  have op − (2 * t0) ‘ op − (2 * t0) ‘ T = T
    x o (λt. 2 * t0 − t) o (λt. 2 * t0 − t) = x
    by force+
  thus ?thesis
    using mi.mirror-solution[of x o mirror] misol
    by (auto simp: mirror-def mi-def)
qed
```

```
lemma solution-mirror-eq:
  defines mirror ≡ λt. 2 * t0 − t
  defines mi ≡ i(|ivp-f := (λ(t, x). − f (mirror t, x)), ivp-T := mirror ‘ T|)
  shows is-solution x ⟷ ivp.is-solution mi (x o mirror)
  using solution-mirror[of x] mirror-solution[of x]
  by (auto simp add: mirror-def mi-def)

lemma shift-autonomous-solution:
  assumes is-solution y
  assumes x = y o (λt.(t + ivp-t0 i − ivp-t0 j))
  assumes ⋀s t x. ivp-f i (s, x) = ivp-f i (t, x)
  assumes ivp-f j = ivp-f i
  assumes ivp-x0 j = ivp-x0 i
  assumes ivp-X j = ivp-X i
  assumes ivp-T j = op + (ivp-t0 j − ivp-t0 i) ‘ ivp-T i
  shows ivp.is-solution j x
proof −
  interpret j: ivp j
    using iv-defined
    by (unfold-locales) (auto simp: assms)
  have image-collapse:
    (λt. t + t0 − ivp-t0 j) ‘ op + (ivp-t0 j − t0) ‘ T = ivp-T i
    by force
  have deriv-id: ⋀x F. ((λt. t + ivp-t0 i − ivp-t0 j) has-vector-derivative 1) F
    by (auto intro!: derivative-eq-intros simp: has-vector-derivative-def)
  show ?thesis
    using is-solutionD[OF assms(1)]
    by (intro j.is-solutionI;
      force
        simp: assms image-collapse
        intro: deriv-id vector-diff-chain-within[THEN vector-derivative-eq-rhs])
qed

lemma shift-initial-value:
  assumes is-solution y
  assumes ivp-t0 j ∈ ivp-T j
  assumes ivp-f j = ivp-f i
  assumes ivp-x0 j = y (ivp-t0 j)
  assumes ivp-X j = ivp-X i
  assumes ivp-T j ⊆ ivp-T i
  shows ivp.is-solution j y
proof −
  interpret j: ivp j
    using iv-defined is-solutionD(3)[OF assms(1)] assms
    by (unfold-locales) auto
  show ?thesis
    using is-solutionD[OF assms(1)] assms
    by (auto intro!: j.is-solutionI
      has-vector-derivative-within-subset[where t=j.T and s = T])
```

qed

end

locale *has-solution* = *ivp* +
  assumes *exists-solution*: $\exists\, x.$ *is-solution x*
begin

lemma *is-solution-solution*[*intro*, *simp*]:
  shows *is-solution solution*
  using *exists-solution* **unfolding** *solution-def* **by** (*rule someI-ex*)

lemma *solution*:
  shows *solution-t0*: *solution t0* = *x0*
    and *solution-has-deriv*: $\bigwedge t.\ t \in T \Longrightarrow$
      (*solution has-vector-derivative f* (*t*, *solution t*)) (*at t within T*)
    and *solution-in-D*: $\bigwedge t.\ t \in T \Longrightarrow$ *solution t* $\in X$
  using *is-solution-solution* **unfolding** *is-solution-def* **by** *auto*

lemma *has-solution-moved*:
  assumes *ivp-t0 j* $\in$ *ivp-T j*
  assumes *ivp-x0 j* = *ivp.solution i* (*ivp-t0 j*)
  assumes *ivp-X j* = *ivp-X i*
  assumes *ivp-T j* $\subseteq$ *ivp-T i*
  assumes *ivp-f j* = *ivp-f i*
  shows *has-solution j*
  by (*metis assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *assms*(*5*) *has-solution-axioms.intro*
has-solution-def
    is-solutionD(*3*) *is-solution-solution ivp.intro set-mp shift-initial-value*)

end

lemma (**in** *ivp*) *singleton-has-solutionI*:
  assumes $T = \{t0\}$
  shows *has-solution i*
  by *unfold-locales* (*auto simp*: *has-vector-derivative-def assms*
    intro!: *has-derivative-singletonI bounded-linear-scaleR-left*
    iv-defined exI[**where** $x{=}\lambda x.\ x0$])

locale *unique-solution* = *has-solution* +
  assumes *unique-solution*: $\bigwedge y\ t.$ *is-solution y* $\Longrightarrow t \in T \Longrightarrow y\ t =$ *solution t*
  — TODO: stronger uniqueness: assume *is-solution* without restriction to X and allow for shorter time intervals

lemma (**in** *ivp*) *unique-solutionI*:
  assumes *is-solution x*
  assumes $\bigwedge y\ t.$ *is-solution y* $\Longrightarrow t \in T \Longrightarrow y\ t = x\ t$
  shows *unique-solution i*
proof

```
  show ∃x. is-solution x using assms by blast
  then interpret has-solution by unfold-locales
  fix y t
  assume is-solution y t∈T
  from assms(2)[OF this] assms(2)[OF is-solution-solution ‹t ∈ T›]
  show y t = solution t by simp
qed

lemma (in ivp) singleton-unique-solutionI:
  assumes T = {t0}
  shows unique-solution i
  by (metis assms has-solution.is-solution-solution is-solutionD(1) singletonD
    singleton-has-solutionI unique-solutionI)

lemma (in unique-solution) shift-autonomous-unique-solution:
  assumes x = y o (λt.(t + ivp-t0 i − ivp-t0 j))
  assumes ⋀s t x. ivp-f i (s, x) = ivp-f i (t, x)
  assumes ivp-f j = ivp-f i
  assumes ivp-x0 j = ivp-x0 i
  assumes ivp-X j = ivp-X i
  assumes ivp-T j = op + (ivp-t0 j − ivp-t0 i) ‘ ivp-T i
  shows unique-solution j
proof
  interpret j: ivp j
    using iv-defined
    by unfold-locales (auto simp: assms)
  show j.t0 ∈ j.T j.x0 ∈ j.X using j.iv-defined by auto
  show ∃x. ivp.is-solution j x
    by (auto simp: assms
      intro!: exI shift-autonomous-solution[OF is-solution-solution])
  then interpret j: has-solution j by unfold-locales
  fix t y assume t: t ∈ j.T and y-sol: j.is-solution y
  from t have ts: t + t0 − j.t0 ∈ T by (auto simp: assms)
  from y-sol have is-solution (y o (op + (j.t0 − t0)))
    by (rule j.shift-autonomous-solution) (force simp: o-def algebra-simps assms)+
  note unique-solution[OF this ts]
  moreover
  from j.is-solution-solution have is-solution (j.solution o (op + (j.t0 − t0)))
    by (rule j.shift-autonomous-solution) (force simp: o-def algebra-simps assms)+
  note unique-solution[OF this ts]
  ultimately
  show y t = j.solution t
    by simp
qed

locale interval = fixes a b assumes interval-notempty: a ≤ b

locale ivp-on-interval = ivp + interval t0 t1 for t1 +
  assumes interval: T = {t0..t1}
```

begin

lemma *is-solution-ext-cont*:
  assumes *continuous-on T x*
  shows *is-solution* (*ext-cont x t0 t1*) = *is-solution x*
  using *assms iv-defined interval* **by** (*intro is-solution-cong*) *simp-all*

lemma *solution-fixed-point*:
  assumes *x*: *is-solution x* **and** *t*: *t* ∈ *T*
  shows *x0* + *integral* {*t0*..*t*} (λ*t*. *f* (*t*, *x t*)) = *x t*
proof −
  from *is-solutionD*(*2*)[*OF x*] *t*
  have ∀ *ta*∈{*t0* .. *t*}.
      (*x has-vector-derivative f* (*ta*, *x ta*))
      (*at ta within* {*t0*..*t*})
    by (*auto simp*: *interval intro*:
      has-vector-derivative-within-subset[**where** *s*=*T*])
  hence ((λ*t*. *f* (*t*, *x t*)) *has-integral x t* − *x t0*)
      {*t0*..*t*}
    using *t* **by** (*auto simp*: *interval*
      intro!: *fundamental-theorem-of-calculus*)
  from *this*[*THEN integral-unique*]
  show *x0* + *integral* {*t0*..*t*} (λ*t*. *f* (*t*, *x t*)) = *x t*
    by (*simp add*: *is-solutionD*[*OF x*])
qed

end

locale *ivp-on-interval-left* = *ivp* + *interval t1 t0* **for** *t1* +
  assumes *interval*: *T* = {*t1*..*t0*}
begin

lemma *is-solution-ext-cont*:
  assumes *continuous-on T x*
  shows *is-solution* (*ext-cont x t1 t0*) = *is-solution x*
  using *assms iv-defined interval* **by** (*intro is-solution-cong*) *simp-all*

lemma *solution-fixed-point*:
  assumes *x*: *is-solution x* **and** *t*: *t* ∈ *T*
  shows *x0* − *integral* {*t*..*t0*} (λ*t*. *f* (*t*, *x t*)) = *x t*
proof −
  from *is-solutionD*(*2*)[*OF x*] *t*
  have ∀ *ta*∈{*t*..*t0*}.
      (*x has-vector-derivative f* (*ta*, *x ta*))
      (*at ta within* {*t*..*t0*})
    by (*auto simp*: *interval intro*:
      has-vector-derivative-within-subset[**where** *s*=*T*])
  hence ((λ*t*. *f* (*t*, *x t*)) *has-integral x t0* − *x t*)
      {*t*..*t0*}

```
    using t by (auto simp: interval
      intro!: fundamental-theorem-of-calculus)
  from this[THEN integral-unique]
  show x0 − integral {t..t0} (λt. f (t, x t)) = x t
    by (simp add: is-solutionD[OF x])
qed

end

sublocale ivp-on-interval ⊆ interval t0 t1 by unfold-locales
sublocale ivp-on-interval-left ⊆ interval t1 t0 by unfold-locales
```

3.3.1 Connecting solutions

```
locale connected-solutions =
  i1?: has-solution i1 + i2?: has-solution i2 + i?: ivp i
  for i::('a::banach) ivp and i1::'a ivp
  and i2::'a ivp +
  fixes y
  assumes sol1: i1.is-solution y
  assumes iv-on:
    i.t0 ∉ i1.T ⟹ i2.solution i.t0 = i.x0
    i.t0 ∈ i1.T ⟹ y i.t0 = i.x0
  assumes conn-x: ⋀t. t ∈ i1.T ∩ i2.T ⟹ y t = i2.solution t
  assumes conn-f: ⋀t. t ∈ i1.T ∩ i2.T ⟹ i1.f (t, y t) = i2.f (t, y t)
  assumes conn-T: closure i1.T ∩ closure i2.T ⊆ i1.T
    closure i1.T ∩ closure i2.T ⊆ i2.T
  assumes f: f = (λ(t, x). if t ∈ i1.T then i1.f (t, x) else i2.f (t, x))
  assumes interval: T = i1.T ∪ i2.T
  assumes dom:X = i1.X X = i2.X
begin

lemma T-subsets:
  shows T1-subset: i1.T ⊆ T
  and T2-subset: i2.T ⊆ T
  subgoal by (metis Un-commute Un-upper2 interval)
  subgoal by (metis inf-sup-ord(4) interval)
  done

definition connection where
  connection t = (if t ∈ i1.T then y t else i2.solution t)

lemma is-solution-connection: is-solution connection
proof standard
  show connection i.t0 = i.x0 ⋀t. t ∈ i.T ⟹ connection t ∈ i.X
    by (auto simp: connection-def iv-on connection-def[abs-def]
      has-vector-derivative-def interval
      i2.is-solutionD[OF i2.is-solution-solution, simplified dom(2)[symmetric]]
      i1.is-solutionD[OF sol1, simplified dom(1)[symmetric]])
```

fix *t*
assume $t \in T$
have *FDERIV-y*:
$\bigwedge t.\ t \in i1.T \Longrightarrow$
$(y\ \textit{has-derivative}\ (\lambda a.\ a *_R i1.f\ (t,\ y\ t)))$
$(\textit{at}\ t\ \textit{within}\ i1.T)$
using *i1.is-solutionD*[*OF sol1*]
by (*auto simp*: *has-vector-derivative-def*)
have *FDERIV-2*:
$\bigwedge t.\ t \in i2.T \Longrightarrow$
$(i2.\textit{solution}\ \textit{has-derivative}\ (\lambda a.\ a *_R i2.f\ (t,\ i2.\textit{solution}\ t)))$
$(\textit{at}\ t\ \textit{within}\ i2.T)$
using *i2.is-solutionD*[*OF i2.is-solution-solution*]
by (*auto simp*: *has-vector-derivative-def*)
show
(*connection has-vector-derivative i.f* (*t*, *connection t*)) (*at t within i.T*)
unfolding *connection-def*[*abs-def*] *interval has-vector-derivative-def*
apply (*rule has-derivative-subset*[**where** $s{=}i1.T \cup i2.T$])
proof (*rule has-derivative-If*[**where** $t{=}i2.T$, *THEN has-derivative-eq-rhs*, *OF has-derivative-subset has-derivative-subset*])
from *FDERIV-y FDERIV-2*
show $t \in i1.T \cup \textit{closure}\ i1.T \cap \textit{closure}\ i2.T \Longrightarrow (y\ \textit{has-derivative}\ (\lambda a.\ a *_R i1.f\ (t,\ y\ t)))\ (\textit{at}\ t\ \textit{within}\ i1.T)$
and $t \in i2.T \cup \textit{closure}\ i1.T \cap \textit{closure}\ i2.T \Longrightarrow (i2.\textit{solution}\ \textit{has-derivative}\ (\lambda a.\ a *_R i2.f\ (t,\ i2.\textit{solution}\ t)))\ (\textit{at}\ t\ \textit{within}\ i2.T)$ **for** *t*
using *conn-T*
by *auto*
qed (*insert conn-T conn-f conn-T* ‹$t \in T$›, *auto simp*: *conn-x f interval*)
qed

lemma *connection-eq-solution2*: $t \in i2.T \Longrightarrow \textit{connection}\ t = i2.\textit{solution}\ t$
by (*auto simp*: *connection-def conn-x*)

end

sublocale *connected-solutions* ⊆ *has-solution* **using** *is-solution-connection*
by *unfold-locales auto*

locale *connected-unique-solutions* =
i1?: *unique-solution i1* + *i2?*: *unique-solution i2* +
connected-solutions i i1 i2 i1.solution
for *i*::*'a*::*banach ivp* **and** *i1*::*'a ivp*
and *i2*::*'a ivp* +
fixes *t1*::*real*
assumes *inter-T*: $i1.T \cap i2.T = \{t1\}$
assumes *initial-times*: $i2.t0 = t1\ i.t0 = i1.t0$
begin

sublocale *unique-solution*

```
proof (intro unique-solutionI)
  show is-solution connection using is-solution-connection .
  fix y t
  assume is-solution y t ∈ T
  have i1.is-solution y
  proof (intro i1.is-solutionI)
    fix ta
    assume ta ∈ i1.T
    hence ta ∈ T using T1-subset by auto
    from is-solutionD(2)[OF ‹is-solution y› this]
    have (y has-vector-derivative i1.f (ta, y ta)) (at ta within T)
      using ‹ta ∈ i1.T› by (simp add: f)
    thus (y has-vector-derivative i1.f (ta, y ta)) (at ta within i1.T)
      using T1-subset
      by (rule has-vector-derivative-within-subset)
    show y ta ∈ i1.X using is-solutionD(3)[OF ‹is-solution y› ‹ta ∈ T›]
      by (simp add: dom)
  next
    have connection i1.t0 = i1.solution i1.t0
      using i1.iv-defined
      by (auto simp: connection-def)
    show y i1.t0 = i1.x0
      using is-solutionD(1)[OF ‹is-solution y›]
      using i1.iv-defined(1) initial-times i1.solution-t0 iv-on(2)
      by auto
  qed
  have i2.is-solution y
  proof (intro i2.is-solutionI)
    show y (i2.t0) = i2.x0
      by (metis Int-lower1 ‹ivp.is-solution i1 y› conn-x i1.unique-solution
        i2.solution-t0 initial-times(1) insertI1 inter-T rev-subsetD)
    fix ta
    assume ta ∈ i2.T
    hence ta ∈ T using T2-subset by auto
    from is-solutionD(2)[OF ‹is-solution y› this]
    have (y has-vector-derivative i2.f (ta, y ta)) (at ta within T)
      using ‹ta ∈ i2.T› conn-f conn-T
      apply (auto simp: f)
      by (metis (poly-guards-query) ‹ivp.is-solution i1 y› i1.unique-solution)
    thus (y has-vector-derivative i2.f (ta, y ta)) (at ta within i2.T)
      using T2-subset
      by (rule has-vector-derivative-within-subset)
    show y ta ∈ i2.X using is-solutionD(3)[OF ‹is-solution y› ‹ta ∈ T›]
      using dom by simp
  qed
  from i1.unique-solution[OF ‹i1.is-solution y›, of t]
    i2.unique-solution[OF ‹i2.is-solution y›, of t]
  show y t = connection t
    using ‹t ∈ T›
```

by (*auto simp*: *connection-def interval*)
qed

lemma *connection-eq-solution*: $\bigwedge t.\ t \in T \Longrightarrow$ *connection t* = *solution t*
by (*rule unique-solution is-solution-connection*)+

lemma *solution1-eq-solution*:
assumes $t \in i1.T$
shows *i1.solution t* = *solution t*
proof −
from *T1-subset assms* **have** $t \in T$ **by** *auto*
from *connection-eq-solution*[*OF* ‹$t \in T$›] *assms*
show *?thesis*
by (*simp add*: *connection-def*)
qed

lemma *solution2-eq-solution*:
assumes $t \in i2.T$
shows *i2.solution t* = *solution t*
proof −
from *T2-subset assms* **have** $t \in T$ **by** *auto*
from *connection-eq-solution*[*OF* ‹$t \in T$›] *assms conn-x i2.solution-t0*
show *?thesis*
by (*simp add*: *connection-def split-ifs*)
qed

end

3.4 Picard-Lindeloef on set of functions into closed set

locale *continuous-rhs* = **fixes** *T X f*
assumes *continuous*: *continuous-on* $(T \times X)$ *f*

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

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

locale *self-mapping* = *ivp-on-interval* +
assumes *self-mapping*:
$\bigwedge x\ t.\ t \in T \Longrightarrow x\ t0 = x0 \Longrightarrow x \in \{t0..t\} \to X \Longrightarrow$ *continuous-on* $\{t0..t\}$ *x*
$\Longrightarrow$
$x0 +$ *integral* $\{t0..t\}$ $(\lambda t.\ f\ (t, x\ t)) \in X$

locale *unique-on-closed* = *self-mapping* + *continuous-rhs T X f* +
closed-domain X +
global-lipschitz T X f L **for** *L*

begin

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

Picard Iteration

definition *P-inner*
  where
  P-inner x t = *x0* + *integral* {*t0..t*} (λ*t*. *f* (*t*, *x t*))

definition *P*::(*real*, ′*a*) *bcontfun* ⇒ (*real*, ′*a*) *bcontfun* **where**
  P x = *ext-cont* (*P-inner x*) *t0 t1*

lemma
  continuous-f:
  assumes *y* ∈ {*t0..t*} → *X*
  assumes *continuous-on* {*t0..t*} *y*
  assumes *t* ∈ *T*
  shows *continuous-on* {*t0..t*} (λ*t*. *f* (*t*, *y t*))
  using ‹*y* ∈ {*t0..t*} → *X*› *assms interval-notempty*
  by (*intro continuous-Sigma*[*of - - λ-. X*])
    (*auto simp*: *interval intro*: *assms continuous-on-subset continuous*)

lemma *P-inner-bcontfun*:
  assumes *y* ∈ *T* → *X*
  assumes *y-cont*: *continuous-on T y*
  shows (λ*x*. *P-inner y* (*clamp t0 t1 x*)) ∈ *bcontfun*
proof −
  show *?thesis* **using** *interval iv-defined assms*
    by (*auto intro*!: *clamp-bcontfun continuous-intros continuous-f*
      indefinite-integral-continuous integrable-continuous-real
      simp: *P-def P-inner-def*)
qed

definition *iter-space* = (*Abs-bcontfun* ‘ ((*T* → *X*) ∩ *bcontfun* ∩ {*x*. *x t0* = *x0*}))

lemma *iter-spaceI*:
  (⋀*x*. *x* ∈ *T* ⟹ *Rep-bcontfun g x* ∈ *X*) ⟹ *g t0* = *x0* ⟹ *g* ∈ *iter-space*
  by (*force simp add*: *assms iter-space-def Rep-bcontfun Rep-bcontfun-inverse*
    intro!: *Rep-bcontfun*)

lemma *const-in-subspace*: (λ-. *x0*) ∈ (*T* → *X*) ∩ *bcontfun* ∩ {*x*. *x t0* = *x0*}
  by (*auto intro*: *const-bcontfun iv-defined*)

lemma *closed-iter-space*: *closed iter-space*
proof −
  have (*T* → *X*) ∩ *bcontfun* ∩ {*x*. *x t0* = *x0*} =
    Pi T (λ*i*. *if i* = *t0 then* {*x0*} *else X*) ∩ *bcontfun*
    using *iv-defined*

by (*force simp*: *Pi-iff split-ifs*)
thus *?thesis* **using** *closed*
by (*auto simp add*: *iter-space-def intro*!: *closed-Pi-bcontfun*)
qed

lemma *iter-space-notempty*: *iter-space* ≠ {}
using *const-in-subspace* **by** (*auto simp*: *iter-space-def*)

lemma *P-self-mapping*:
assumes *in-space*: *g* ∈ *iter-space*
shows *P g* ∈ *iter-space*
proof (*rule iter-spaceI*)
have *cont*: *continuous-on* (*cbox t0 t1*) (*P-inner* (*Rep-bcontfun g*))
using *assms Rep-bcontfun*[*of g*, *simplified bcontfun-def*]
by (*auto simp*: *interval iter-space-def Abs-bcontfun-inverse P-inner-def*
interval-notempty
intro!: *continuous-intros indefinite-integral-continuous*
integrable-continuous-real continuous-f)
from *ext-cont-cancel*[*OF - cont*] *assms*
show *Rep-bcontfun* (*P g*) *t0* = *x0*
⋀*t*. *t* ∈ *T* ⟹ *Rep-bcontfun* (*P g*) *t* ∈ *X*
using *assms Rep-bcontfun*[*of g*, *simplified bcontfun-def*]
by (*auto intro*!: *self-mapping simp*: *interval interval-notempty P-inner-def*
P-def iter-space-def Abs-bcontfun-inverse)
qed

lemma *ext-cont-solution-fixed-point*:
assumes *is-solution x*
shows *P* (*ext-cont x t0 t1*) = *ext-cont x t0 t1*
unfolding *P-def*
proof (*rule ext-cont-cong*)
show *P-inner* (*Rep-bcontfun* (*ext-cont x t0 t1*)) *t* = *x t* **when** *t* ∈ {*t0*..*t1*} **for** *t*
unfolding *P-inner-def*
using *solution-fixed-point solution-continuous-on assms is-solutionD that*
by (*subst integral-spike*[*OF negligible-empty*])
(*auto simp*: *interval P-inner-def integral-spike*[*OF negligible-empty*])
qed (*insert iv-defined solution-continuous-on assms is-solutionD*,
auto simp: *interval P-inner-def continuous-intros*
indefinite-integral-continuous continuous-f)

lemma
solution-in-iter-space:
assumes *is-solution z*
shows *ext-cont z t0 t1* ∈ *iter-space*
proof −
let *?z* = *ext-cont z t0 t1*
have *is-solution ?z*
using *is-solution-ext-cont interval* ‹*is-solution z*› *solution-continuous-on*
by *simp*

```
  hence ⋀t. t ∈ T ⟹ ext-cont z t0 t1 t ∈ X
    by (auto simp add: is-solution-def)
  thus ?z ∈ iter-space using is-solutionD[OF ‹is-solution z›]
    solution-continuous-on[OF ‹is-solution z›]
    by (auto simp: interval interval-notempty intro!: iter-spaceI)
qed

end

locale unique-on-bounded-closed = unique-on-closed +
  assumes lipschitz-bound: (t1 − t0) ∗ L < 1
begin

lemma lipschitz-P:
  shows lipschitz iter-space P ((t1 − t0) ∗ L)
proof (rule lipschitzI)
  have t0 ∈ T by (simp add: iv-defined)
  thus 0 ≤ (t1 − t0) ∗ L
    using interval-notempty interval
    by (auto intro!: mult-nonneg-nonneg lipschitz lipschitz-nonneg[OF lipschitz]
      iv-defined)
  fix y z
  assume y ∈ iter-space and z ∈ iter-space
  hence y-defined: Rep-bcontfun y ∈ (T → X)
    and z-defined: Rep-bcontfun z ∈ (T → X)
    by (auto simp: Abs-bcontfun-inverse iter-space-def)
  {
    fix y z::real⇒'a
    assume y ∈ bcontfun and y-defined: y ∈ (T → X)
    assume z ∈ bcontfun and z-defined: z ∈ (T → X)
    from bcontfunE[OF ‹y ∈ bcontfun›] have y: continuous-on UNIV y by auto
    from bcontfunE[OF ‹z ∈ bcontfun›] have z: continuous-on UNIV z by auto
    {
      fix t
      assume t-bounds: t0 ≤ t t ≤ t1
        — Instances of continuous-on-subset
      have y-cont: continuous-on {t0..t} (λt. y t) using y
        by (auto intro:continuous-on-subset)
      have continuous-on {t0..t1} (λt. f (t, y t))
        using continuous interval interval-notempty y strip y-defined
        by (auto intro!:continuous-f intro: continuous-on-subset)
      hence fy-cont[intro, simp]:
        continuous-on {t0..t} (λt. f (t, y t))
        by (rule continuous-on-subset) (simp add: t-bounds)
      have z-cont: continuous-on {t0..t} (λt. z t) using z
        by (auto intro:continuous-on-subset)
      have continuous-on {t0..t1} (λt. f (t, z t))
      by (metis (no-types) UNIV-I continuous continuous-Sigma continuous-on-subset
interval subsetI z z-defined)
```

```
    hence fz-cont[intro, simp]:
      continuous-on {t0..t} (λt. f (t, z t))
      by (rule continuous-on-subset) (simp add: t-bounds)

    have norm (P-inner y t − P-inner z t) =
      norm (integral {t0..t} (λt. f (t, y t) − f (t, z t)))
      using y
      by (auto simp add: integral-diff P-inner-def)
    also have ... ≤ integral {t0..t} (λt. norm (f (t, y t) − f (t, z t)))
      by (auto intro!: integral-norm-bound-integral continuous-intros)
    also have ... ≤ integral {t0..t} (λt. L ∗ norm (y t − z t))
      using y-cont z-cont lipschitz t-bounds interval y-defined z-defined
      by (intro integral-le)
         (auto intro!: continuous-intros simp add: dist-norm lipschitz-def Pi-iff)
    also have ... ≤ integral {t0..t} (λt. L ∗
      norm (Abs-bcontfun y − Abs-bcontfun  z))
      using norm-bounded[of Abs-bcontfun y − Abs-bcontfun z]
        y-cont z-cont L-nonneg
      by (intro integral-le) (auto intro!: continuous-intros mult-left-mono
        simp add: Abs-bcontfun-inverse[OF ‹y ∈ bcontfun›]
        Abs-bcontfun-inverse[OF ‹z ∈ bcontfun›])
    also have ... =
      L ∗ (t − t0) ∗ norm (Abs-bcontfun y − Abs-bcontfun z)
      using t-bounds by simp
    also have ... ≤ L ∗ (t1 − t0) ∗ norm (Abs-bcontfun y − Abs-bcontfun z)
      using t-bounds zero-le-dist L-nonneg
      by (auto intro!: mult-right-mono mult-left-mono)
    finally
    have norm (P-inner y t − P-inner z t)
      ≤ L ∗ (t1 − t0) ∗ norm (Abs-bcontfun y − Abs-bcontfun z) .
  } note ∗ = this
  have dist (P (Abs-bcontfun y)) (P (Abs-bcontfun z)) ≤
    L ∗ (t1 − t0) ∗ dist (Abs-bcontfun y) (Abs-bcontfun z)
    unfolding P-def dist-norm ext-cont-def
      Abs-bcontfun-inverse[OF ‹y ∈ bcontfun›]
      Abs-bcontfun-inverse[OF ‹z ∈ bcontfun›]
    using interval iv-defined ‹y ∈ bcontfun› ‹z ∈ bcontfun›
      y-defined z-defined
      clamp-in-interval[of t0 t1] interval-notempty
    apply (intro norm-bound)
    unfolding Rep-bcontfun-minus
    apply (subst Abs-bcontfun-inverse)
     defer
     apply (subst Abs-bcontfun-inverse)
      defer
    by (auto intro!: P-inner-bcontfun ∗ elim!: bcontfunE
      intro: continuous-on-subset)
}
from this[OF Rep-bcontfun y-defined Rep-bcontfun z-defined]
```

show *dist* (*P y*) (*P z*) ≤ (*t1* − *t0*) ∗ *L* ∗ *dist y z*
unfolding *Rep-bcontfun-inverse* **by** (*simp add*: *field-simps*)
qed

lemma *fixed-point-unique*: ∃!*x*∈*iter-space*. *P x* = *x*
using *lipschitz lipschitz-bound lipschitz-P interval*
complete-UNIV iv-defined
by (*intro banach-fix*)
(*auto*
intro: *P-self-mapping split-mult-pos-le*
intro!: *closed-iter-space iter-space-notempty*
simp: *lipschitz-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 assms*
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-continuous*: ⋀*t*. *continuous-on I fixed-point*
using *bcontfunE*[*OF Rep-bcontfun*[*of fixed-point*]]
by (*auto intro*: *continuous-on-subset*)

lemma *fixed-point-solution*:
shows *is-solution fixed-point*
proof
have *fixed-point t0* = *P fixed-point t0*
unfolding *fixed-point* **..**
also have ... = *x0*
using *interval iv-defined continuous fixed-point-continuous fixed-point*
unfolding *P-def P-inner-def*[*abs-def*]
by (*subst ext-cont-cancel*)
(*auto simp add*: *iter-space-def Abs-bcontfun-inverse*
intro!: *continuous-intros indefinite-integral-continuous*
integrable-continuous-real continuous-f

```
      intro: continuous-on-subset)
  finally show fixed-point t0 = x0 .
next
  fix t
  have U: Rep-bcontfun fixed-point ∈ Pi T (λ-. X)
    using fixed-point by (auto simp add: iter-space-def Abs-bcontfun-inverse)
  assume t ∈ T hence t-range: t ∈ {t0..t1} by (simp add: interval)
  from has-vector-derivative-const
    integral-has-vector-derivative[OF
      continuous-Sigma[OF U continuous fixed-point-continuous,
        simplified interval]
      t-range]
  have ((λu. x0 + integral {t0..u}
        (λx. f (x, fixed-point x))) has-vector-derivative
    0 + f (t, fixed-point t))
   (at t within {t0..t1})
    by (rule has-vector-derivative-add)
  hence ((P fixed-point) has-vector-derivative
    f (t, fixed-point t)) (at t within {t0..t1})
    unfolding P-def P-inner-def[abs-def]
    using t-range
    apply (subst has-vector-derivative-cong)
    apply (simp-all)
    using fixed-point fixed-point-continuous continuous interval
    by (subst ext-cont-cancel)
      (auto simp: iter-space-def Abs-bcontfun-inverse
        intro!: continuous-intros indefinite-integral-continuous
          integrable-continuous-real continuous-f
        intro: continuous-on-subset)
  moreover
  have fixed-point t ∈ X
   using fixed-point ‹t ∈ T› by (auto simp add: iter-space-def Abs-bcontfun-inverse)
  ultimately
  show (fixed-point has-vector-derivative
      f (t, fixed-point t)) (at t within T)
    fixed-point t ∈ X unfolding fixed-point interval
    by simp-all
qed

end
```

3.4.1 Existence of solution

```
sublocale unique-on-bounded-closed ⊆ has-solution
proof
  from fixed-point-solution
  show ∃x. is-solution x by blast
qed
```

3.4.2 Unique solution

```
sublocale unique-on-bounded-closed ⊆ unique-solution
proof
  fix z t
  assume is-solution z
  with ext-cont-solution-fixed-point ‹is-solution z› is-solution-solution
    solution-in-iter-space fixed-point-equality
  have ext-cont solution t0 t1 t = ext-cont z t0 t1 t by metis
  moreover assume t ∈ T
  ultimately
  show z t = solution t
    using solution-continuous-on[OF ‹is-solution z›]
      solution-continuous-on[OF is-solution-solution]
    by (auto simp: interval)
qed

sublocale unique-on-closed ⊆ unique-solution
proof (cases t1 = t0)
  assume t1 = t0
  then interpret has-solution
    using is-solution-def interval iv-defined
    by unfold-locales (auto intro!: exI[where x=(λt. x0)]
      simp add: has-vector-derivative-def
      has-derivative-within-alt bounded-linear-scaleR-left)
  show unique-solution i
    using ‹t1=t0› interval solution-t0
    by unfold-locales (simp add: is-solution-def)
next
  assume t1 ≠ t0
  with interval iv-defined
  have interval: T = {t0..t1} t0 < t1
    by auto
  obtain n::nat and b where b: b = (t1 − t0) / (Suc n) and bL: L ∗ b < 1
     by (rule, rule) (auto intro: order-le-less-trans real-nat-ceiling-ge simp del:
of-nat-Suc)
  then interpret i': ivp-on-interval i t0 + (Suc n) ∗ b
    using interval by unfold-locales simp-all
  from b have b > 0 using interval iv-defined
    by auto
  hence b ≥ 0 by simp
  from interval have t0 ∗ (real (Suc n) − 1) ≤ t1 ∗ (real (Suc n) − 1)
    by (cases n) auto
  hence ble: t0 + b ≤ t1 unfolding b by (auto simp add: field-simps)
  have subsetbase: t0 + (Suc n) ∗ b ≤ t1 using i'.interval interval by auto

  interpret i': unique-solution i(|ivp-T := {t0..t0 + real (Suc n) ∗ b}|)
    using subsetbase
  proof (induct n)
```

case *0*
then interpret *sol*: *unique-on-bounded-closed i*(|*ivp-T*:={*t0*..*t0*+*b*}|) *t0* + *b*
using *interval iv-defined* ‹*b* > *0*› *bL continuous lipschitz closed self-mapping*
by *unfold-locales* (*auto intro*: *continuous-on-subset simp*: *ac-simps Pi-iff*)
show *?case* **by** *simp unfold-locales*
next
case (*Suc n*)
def *nb* ≡ *real* (*Suc n*) ∗ *b*
def *snb* ≡ *real* (*Suc* (*Suc n*)) ∗ *b*
note *Suc* = *Suc*[*simplified nb-def*[*symmetric*] *snb-def*[*symmetric*]]
from ‹*b* > *0*› *nb-def snb-def* **have** *nbs-nonneg*: *0* < *snb 0* < *nb*
by (*simp-all add*: *zero-less-mult-iff*)
with ‹*b*>*0*› **have** *nb-le-snb*: *nb* < *snb* **using** *nb-def snb-def*
by *auto*
have [*simp*]: *snb* − *nb* = *b*
proof −
have *snb* + − (*nb*) = *b* ∗ *real* (*Suc* (*Suc n*)) + − (*b* ∗ *real* (*Suc n*))
by (*simp add*: *ac-simps snb-def nb-def*)
thus *?thesis* **by** (*simp add*: *field-simps of-nat-Suc*)
qed
def *i1* ≡ *i*(|*ivp-T* := {*t0*..*t0* + *nb*}|)
def *T1* ≡ *t0* + *nb*
interpret *ivp1*: *ivp-on-interval i1 T1*
using *iv-defined* ‹*nb* > *0*› **by** *unfold-locales* (*auto simp*: *i1-def T1-def*)
interpret *ivp1*: *unique-solution i1*
using *nb-le-snb nbs-nonneg Suc continuous lipschitz* **by** (*simp add*: *i1-def*)
interpret *ivp1-cl*: *unique-on-closed i1 t0* + *nb*
using *nb-le-snb nbs-nonneg Suc continuous lipschitz closed self-mapping*
by *unfold-locales* (*auto simp*: *i1-def interval intro*: *continuous-on-subset*)
def *i2* ≡ *i*(|*ivp-t0*:=*t0*+*nb*, *ivp-T*:={*t0* + *nb*..*t0*+*snb*},
ivp-x0:=*ivp1.solution* (*t0* + *nb*)|)
def *T2* ≡ *t0* + *snb*
interpret *ivp2*: *ivp-on-interval i2 T2*
using *nbs-nonneg* ‹*nb* < *snb*› *ivp1.solution-in-D*
by *unfold-locales* (*auto simp*: *i1-def i2-def T2-def*)
interpret *ivp2*: *self-mapping i2 T2*
proof *unfold-locales*
fix *x t* **assume** *t*: *t* ∈ *ivp2.T*
and *x*: *x ivp2.t0* = *ivp2.x0 x* ∈ {*ivp2.t0* .. *t*} → *ivp2.X*
and *cont*: *continuous-on* {*ivp2.t0* .. *t*} *x*
hence *t* ∈ *T*
using *Suc*(*2*) *nbs-nonneg interval*
by (*simp add*: *i2-def*)
let *?un* = (λ*t*. *if t* ≤ *nb* + *t0 then ivp1.solution t else x t*)
let *?fun* = (λ*t*. *f* (*t*, *?un t*))
have *decomp*: {*t0*..*t*} = {*t0*..*nb* + *t0*} ∪ {*nb* + *t0*..*t*}
using *interval-notempty t nbs-nonneg*
by (*auto simp*: *i2-def*)
have *un-space*: *?un* ∈ {*t0*..*t*} → *X*

```
    using x ivp1.solution-in-D
    by (auto simp: i1-def i2-def Pi-iff)
  have cont-un: continuous-on {t0..t} ?un
    using x cont
      ivp1.solution-continuous-on[OF ivp1.is-solution-solution,
        simplified i1-def]
    unfolding decomp
    by (intro continuous-on-If)
      (auto intro: continuous-on-subset simp: i1-def i2-def ac-simps)
  have cont-fun: continuous-on {t0..t} ?fun
    using un-space cont-un ‹t ∈ T› by (rule continuous-f)
  have ivp.solution i1 (nb + t0) + integral {nb + t0..t} (λxa. f (xa, x xa)) =
    x0 + (integral {t0..nb + t0} (λt. f (t, ivp1.solution t)) +
      integral {nb + t0..t} (λxa. f (xa, x xa)))
    using ivp1-cl.solution-fixed-point[OF ivp1.is-solution-solution] nbs-nonneg
      ivp1-cl.P-inner-def
    by (auto simp: i1-def ac-simps)
  also have integral {t0..nb + t0} (λt. f (t, ivp1.solution t)) =
      integral {t0..nb + t0} ?fun
    by (rule integral-spike[OF negligible-empty]) auto
  also have fun2: integral {nb + t0..t} (λt. f (t, x t)) =
      integral {nb + t0..t} ?fun
    using x
    by (intro integral-spike[OF negligible-empty])
      (auto simp: i1-def i2-def ac-simps)
  also have integral {t0..nb + t0} ?fun + integral {nb + t0..t} ?fun =
      integral {t0..t} ?fun
    using t nbs-nonneg
    by (intro integral-combine)
      (auto simp: i2-def less-imp-le intro!: cont-fun)
  also have x0 + ... ∈ X
    using ‹t ∈ T› ‹nb > 0› ivp1.is-solutionD[OF ivp1.is-solution-solution]
    by (intro self-mapping[OF - - un-space cont-un])
      (auto simp: ivp1.iv-defined i1-def)
  also note fun2[symmetric]
  finally
  show ivp2.x0 + integral {ivp2.t0 .. t} (λt. ivp2.f (t, x t)) ∈ ivp2.X
    by (simp add: i1-def i2-def ac-simps)
qed
interpret ivp2: unique-on-bounded-closed i2 T2
  using bL Suc(2) nbs-nonneg interval continuous lipschitz closed
  by unfold-locales
    (auto intro: continuous-on-subset simp: ac-simps i1-def i2-def T2-def)
def i ≡ i(|ivp-T := {t0..t0 + real (Suc (Suc n)) * b}|)
def T ≡ t0 + real (Suc (Suc n)) * b
interpret i: ivp i
proof
  show ivp-t0 i ∈ ivp-T i ivp-x0 i ∈ ivp-X i
    using ivp1.iv-defined ‹0 ≤ b›
```

```
      by (auto simp: i-def i1-def nb-def intro!: mult-nonneg-nonneg)
    qed
    have *: ivp-T i1 ∩ ivp-T i2 = {T1}
      using nbs-nonneg
      by (auto simp: i1-def i2-def nb-def snb-def max-def min-def T1-def not-le
        mult-less-cancel-right sign-simps
        simp del: of-nat-Suc)
    have nb-le-snb: t0 + real (Suc n) * b ≤ t0 + real (Suc (Suc n)) * b
      using ‹b > 0› by auto
    interpret ivp-c: connected-unique-solutions i i1 i2 T1
      apply unfold-locales
      unfolding *
      using ‹b > 0› iv-defined ivp1.is-solutionD[OF ivp1.is-solution-solution]
        ivp2.is-solutionD[OF ivp2.is-solution-solution]
        ivp1.is-solution-solution
        ivp2.is-solution-solution
         nbs-nonneg
        add-increasing2[of real (Suc n) * b t0 + real (Suc n) * b]
      by (auto simp: i1-def i2-def i-def T1-def T2-def T-def snb-def nb-def
        simp del: of-nat-Suc
        intro!: order-trans[OF - nb-le-snb])
    show ?case unfolding i-def[symmetric] by unfold-locales
  qed
  show unique-solution i
   using i'.solution i'.unique-solution interval(1)[symmetric] i'.interval[symmetric]
    by unfold-locales (auto simp del: of-nat-Suc)
qed
```

3.5 Picard-Lindeloef for $X = (\lambda\text{-}.\ UNIV)$

```
locale unique-on-strip = ivp-on-interval + continuous-rhs T X f +
  global-lipschitz T X f L for L +
  assumes strip: X = UNIV

sublocale unique-on-strip < unique-on-closed
  using strip by unfold-locales auto
```

3.6 Picard-Lindeloef on cylindric domain

```
locale cylinder = ivp i for i::'a::banach ivp +
  fixes e b
  assumes e-pos: e > 0
  assumes b-pos: b > 0
  assumes interval: T = {t0 − e .. t0 + e}
  assumes cylinder: X = cball x0 b

locale solution-in-cylinder = cylinder + continuous-rhs T X f +
  fixes B
  assumes norm-f: ⋀x t. t ∈ T ⟹ x ∈ X ⟹ norm (f (t, x)) ≤ B
```

```
  assumes e-bounded: e ≤ b / B
begin

lemma B-nonneg: B ≥ 0
proof –
  have 0 ≤ norm (f (t0, x0)) by simp
  also from iv-defined norm-f have ... ≤ B by simp
  finally show ?thesis by simp
qed

lemma closed-real-closed-segment: ⋀a b. closed (closed-segment a b::real set)
  by (auto simp: closed-segment-real)

lemma in-bounds-derivativeI:
  assumes t ∈ T
  assumes init: x t0 = x0
  assumes cont: continuous-on (closed-segment t0 t) x
  assumes solves: ⋀s. s ∈ open-segment t0 t ⟹ (x has-vector-derivative f (s, y
s)) (at s within 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 interval e-pos have t = t0
    by auto
  thus ?thesis using iv-defined 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 e-pos e-bounded using ‹b > 0› ‹B > 0› ‹t ∈ T›
      by (auto simp: field-simps interval abs-real-def)
        (metis add-right-mono distrib-left mult-le-cancel-left-pos order-trans)+
    def b≡B ∗ abs (t − t0)
    have b > 0 using ‹t0 ≠ t› by (auto intro!: mult-pos-pos simp: algebra-simps
b-def ‹B > 0›)
    have subs: closed-segment t0 t ⊆ {t0 − e..t0 + e}
      using interval ‹t ∈ T› by (auto simp: closed-segment-real)
    from cont
    have closed: closed {s ∈ closed-segment t0 t. norm (x s − x t0) ∈ {b..}}
      by (intro continuous-closed-preimage continuous-intros
        closed-real-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: {s ∈ closed-segment t0 t. 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-real open-segment-real*)
from *cont* **have** *cont*: *continuous-on* (*closed-segment t0 s*) *x*
by (*rule continuous-on-subset*)
(*insert e-pos subs s-bound*, *auto simp*: *closed-segment-real*)
have *bnd-cont*: *continuous-on* (*closed-segment t0 s*) (*op* ∗ *B*)
and *bnd-deriv*: (⋀*x*. *x* ∈ *open-segment t0 s* ⟹
(*op* ∗ *B has-vector-derivative B*) (*at x within open-segment t0 s*))
by (*auto intro*!: *continuous-intros derivative-eq-intros*
simp: *has-vector-derivative-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-real split-ifs*)
qed
have *norm* (*f* (*ss*, *y ss*)) ≤ *B*
apply (*rule norm-f*)
subgoal using *ss st subs interval* **by** *auto*
subgoal using *ss st b-less less-b*
by (*intro y-bounded*)
(*auto simp*: *cylinder dist-norm b-def init norm-minus-commute*)
done
} **note** *bnd* = *this*
have *subs*: *open-segment t0 s* ⊆ *open-segment t0 t* **using** *s-bound* ‹*s* ≠ *t0*›
by (*auto simp*: *closed-segment-real open-segment-real*)
with *differentiable-bound-general-open-segment*[*OF cont bnd-cont*
has-vector-derivative-within-subset[*OF solves subs*] *bnd-deriv bnd*] *st*
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-real 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-real 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 interval ‹0 < b›
        by (auto simp: closed-segment-real )
      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: interval)
      with mvt-result have s = t by blast
      hence s = t using s ‹t ∈ T› by (auto simp: interval)
      with s H show False by simp
    qed
    hence x t ∈ cball x0 b using init
      by (auto simp: dist-commute dist-norm[symmetric])
    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: ⋀s. s ∈ open-segment t0 t =⇒
    (x has-vector-derivative f (s, y s)) (at s within 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›
    apply (cases B = 0, simp)
    subgoal
```

```
    apply (auto simp: field-simps interval abs-real-def)
    subgoal by (metis add-right-mono less-eq-real-def order-trans
       real-mult-le-cancel-iff2 ring-class.ring-distribs(1))
    subgoal by (metis add-less-same-cancel2 add-right-mono le-less-trans
      mult-le-cancel-left-pos mult-left-mono-neg not-less
      ring-class.ring-distribs(1) zero-le-mult-iff)
    done
   done
  ultimately show ?thesis by (auto simp: cylinder)
qed

lemma integral-in-bounds:
  assumes t ≥ t0 t ∈ T x t0 = x0 x ∈ {t0..t} → X
  assumes cont: continuous-on {t0..t} x
  shows x0 + integral {t0..t} (λt. f (t, x t)) ∈ X (is x0 + ?ix t ∈ X)
proof cases
  assume t = t0
  thus ?thesis by (auto simp: iv-defined)
next
  assume t ≠ t0
  have cont-f:continuous-on {t0..t} (λt. f (t, x t))
    using assms
    by (intro continuous-Sigma)
      (auto intro: cont continuous-on-subset[OF continuous] simp: interval)
  show ?thesis
    using assms ‹t ≠ t0›
    by (intro in-bounds-derivative-globalI[where y=x and x=λt. x0 + ?ix t])
      (auto simp: interval closed-segment-real open-segment-real
        intro!: cont-f has-vector-derivative-const
          has-vector-derivative-within-subset[OF integral-has-vector-derivative]
          has-vector-derivative-add[THEN vector-derivative-eq-rhs]
          continuous-intros indefinite-integral-continuous)
qed

lemma integral-in-bounds':
  assumes ¬ t0 ≤ t t ∈ T x t0 = x0 x ∈ {t..t0} → X
  assumes cont: continuous-on {t..t0} x
  shows x0 + integral {t..t0} (λt. − f (t, x t)) ∈ X (is x0 + ?ix t ∈ X)
proof cases
  assume t = t0
  thus ?thesis by (auto simp: iv-defined)
next
  assume t ≠ t0
  have cont-f:continuous-on {t .. t0} (λt. f (t, x t))
    using assms
    by (intro continuous-Sigma continuous-on-minus)
      (auto intro: cont continuous-on-subset[OF continuous] simp: interval)
  show ?thesis
    using assms ‹t ≠ t0›
```

by (*intro in-bounds-derivative-globalI*[**where** *y*=*x* **and** *x*=λ*t*. *x0* + *?ix t*])
(*auto simp*: *interval closed-segment-real open-segment-real*
intro!: *cont-f*
indefinite-integral2-continuous
has-vector-derivative-within-subset[*OF integral2-has-vector-derivative*]
has-vector-derivative-const
has-vector-derivative-diff [*THEN vector-derivative-eq-rhs*]
continuous-intros)
qed

lemma *solves-in-cone*:
assumes *t* ∈ *T*
assumes *init*: *x t0* = *x0*
assumes *cont*: *continuous-on* (*closed-segment t0 t*) *x*
assumes *solves*: ⋀*s*. *s* ∈ (*open-segment t0 t*) ⟹ (*x has-vector-derivative f* (*s*, *x s*)) (*at s within 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*: *is-solution x*
shows *x t* ∈ *cball x0* (*B* ∗ *abs* (*t* − *t0*))
proof *cases*
assume *t* = *t0*
thus *?thesis* **by** (*auto simp*: *is-solutionD*(*1*)[*OF sol*])
next
assume *t* ≠ *t0*
have *subset1*: (*closed-segment t0 t*) ⊆ *T* **using** *assms interval* **by** (*auto simp*: *closed-segment-real*)
have *subset2*: (*open-segment t0 t*) ⊆ *T* **using** *assms* **by** (*auto simp*: *open-segment-real interval*)
from *is-solutionD*(*1*)[*OF sol*]
is-solutionD(*2*)[*OF sol*, *THEN has-vector-derivative-within-subset*[*OF - subset2*]]
is-solutionD(*3*)[*OF sol set-mp*[*OF subset1*]]
solution-continuous-on[*OF sol*, *THEN continuous-on-subset*[*OF - subset1*]]
show *?thesis*
using *assms*(*1*) *subset1 subset2* ‹*t* ≠ *t0*›
by (*intro solves-in-cone*[**where** *x*=*x*]) (*auto simp*: *interval open-segment-real*
at-within-open[**where** *S*=*open-segment t0 t*, *symmetric*])
qed

end

For the numerical approximation, it is necessary that f is lipschitz-continuous outside the actual domain - therefore X'.

locale *unique-on-cylinder* =

```
  solution-in-cylinder + global-lipschitz: global-lipschitz T X′ f L for L X′ +
  assumes lipschitz-on-domain: X ⊆ X′
begin

lemma lipschitz′: t ∈ T ⟹ lipschitz X (λx. f (t, x)) L 0 ≤ L
  using global-lipschitz.lipschitz lipschitz-on-domain
 by (auto intro: lipschitz-subset intro!: lipschitz-nonneg[OF global-lipschitz.lipschitz]
iv-defined)

sublocale unique-pos: ivp-on-interval i (|ivp-T:={t0 .. t0 + e}|) t0 + e
  using e-pos iv-defined
  by unfold-locales auto

sublocale unique-pos: unique-on-closed i (|ivp-T:={t0 .. t0 + e}|) t0 + e L
proof
  show closed unique-pos.X by (simp-all add: cylinder closed-cball)
  show continuous-on (unique-pos.T × unique-pos.X) unique-pos.f
    using continuous interval by (auto intro: continuous-on-subset)
  fix t assume t: t ∈ unique-pos.T with lipschitz′ interval
  show lipschitz unique-pos.X (λx. unique-pos.f (t, x)) L by simp
  fix x
  assume x unique-pos.t0 = unique-pos.x0
    x ∈ {unique-pos.t0 .. t} → unique-pos.X
    continuous-on {unique-pos.t0 .. t} x
  thus unique-pos.x0 + integral {unique-pos.t0..t} (λt. unique-pos.f (t, x t)) ∈
      unique-pos.X
    using t interval
    by (auto intro: integral-in-bounds)
qed

sublocale unique-neg: ivp i (|ivp-T:={t0 − e.. t0}|)
  using e-pos iv-defined
  by unfold-locales auto

sublocale unique-neg: unique-solution i (|ivp-T:={t0 − e.. t0}|)
proof
  let ?mirror = λt. 2 * t0 − t
  have mirror-eq: ((λx. (2 * t0 − fst x, snd x)) ‘ (T × X)) = T × X
    by (auto intro: image-eqI[where x=(?mirror x, y) for x y] simp: interval)
  have mirror-imp: ⋀t. t ∈ T ⟹ ?mirror t ∈ T
    by (auto simp: interval)
  have cont-mirror: continuous-on (T × X) (− f o (λ(t, x). (?mirror t, x)))
    apply (rule continuous-on-compose)
    using continuous
    by (auto simp: split-beta′ mirror-eq
      intro!: continuous-on-Pair continuous-intros)
  interpret rev:
    unique-on-cylinder i (|ivp-f:=(λ(t, x). −f(?mirror t, x))|) e b B L X′
    apply unfold-locales
```

```
    subgoal using iv-defined by simp
    subgoal using iv-defined by simp
    subgoal using e-pos by simp
    subgoal using b-pos by simp
    subgoal using interval by simp
    subgoal using cylinder by simp
    subgoal using cont-mirror by (simp add: split-beta')
    subgoal using norm-f by (simp add: mirror-imp)
    subgoal using e-bounded by simp
    subgoal using global-lipschitz.lipschitz by (simp add: lipschitz-uminus mirror-imp)
    subgoal using global-lipschitz.lipschitz lipschitz-on-domain by simp
    done
  have *: op − (2 * t0) ‘ {t0 − e..t0} = {t0 .. t0 + e}
    by (auto intro!: image-eqI[where x=?mirror x for x])
  have unique-neg.is-solution (rev.unique-pos.solution o ?mirror)
    using rev.unique-pos.is-solution-solution
    by (simp add: unique-neg.solution-mirror-eq o-def *)
  thus ∃ x. unique-neg.is-solution x by blast
  then interpret unique-neg: has-solution i(|ivp-T := {t0 − e..t0}|)
    by unfold-locales
  fix y t assume t ∈ unique-neg.T and y: unique-neg.is-solution y
  hence t: ?mirror t ∈ rev.unique-pos.T by auto
  from unique-neg.mirror-solution[OF y]
    unique-neg.mirror-solution[OF unique-neg.is-solution-solution]
  have **: rev.unique-pos.is-solution (y o ?mirror)
    rev.unique-pos.is-solution (unique-neg.solution o ?mirror)
    by (auto simp: o-def *)
  from rev.unique-pos.unique-solution[OF **(1) t]
    rev.unique-pos.unique-solution[OF **(2) t]
  show y t = unique-neg.solution t
    by simp
qed

sublocale unique-solution
proof −
  interpret
    connected-solutions
      i i(|ivp-T := {t0 − e..t0}|) i(|ivp-T := {t0..t0+e}|) unique-neg.solution
    using e-pos unique-neg.solution-t0 unique-pos.solution-t0
    by unfold-locales (auto simp: interval)
  interpret
    connected-unique-solutions
      i i(|ivp-T := {t0 − e..t0}|) i(|ivp-T := {t0..t0+e}|) t0
    using e-pos unique-neg.solution-t0 unique-pos.solution-t0
    by unfold-locales auto
  show unique-solution i ..
qed

end
```

```
locale unique-on-superset-domain = subset?: unique-solution +
  fixes X''
  assumes superset: X ⊆ X''
  assumes segment-subset: ⋀t. t ∈ T ⟹ (closed-segment t0 t) ⊆ T
  assumes solution-in-subset: ⋀t x. t ∈ T ⟹ x t0 = x0 ⟹
    (⋀s. s ∈ closed-segment t0 t ⟹
    (x has-vector-derivative f (s, x s)) (at s within closed-segment t0 t)) ⟹
    x t ∈ X
begin

sublocale has-solution i(|ivp-X:=X''|)
  using iv-defined superset
 by unfold-locales (auto intro!: exI[where x=solution] is-solution-on-superset-domain)

lemma is-solution-eq-is-solution-on-supersetdomain:
  shows subset.is-solution = ivp.is-solution (i(|ivp-X:=X''|))
proof −
  interpret ivp': ivp i(|ivp-X:=X''|) using iv-defined assms by unfold-locales auto
  show ?thesis using assms
  proof (safe intro!: ext)
   fix x assume is-solution x
    moreover
    from is-solutionD[OF this] solution-continuous-on[OF this]
    have ⋀t. t ∈ subset.T ⟹ x t ∈ subset.X using assms
      using segment-subset
      by (intro solution-in-subset; force intro!: continuous-on-subset
          continuous-on-subset[OF - segment-subset]
          has-vector-derivative-within-subset[OF - segment-subset])
    ultimately show subset.is-solution x
      by (auto intro!: subset.is-solutionI dest: is-solutionD)
  qed (intro subset.is-solution-on-superset-domain superset)
qed

lemma sup-solution-is-solution: is-solution x ⟹ subset.is-solution x
  using assms superset
  by (subst is-solution-eq-is-solution-on-supersetdomain) auto

lemma solutions-eq:
  t ∈ T ⟹ solution t = subset.solution t
  using sup-solution-is-solution
  by (auto intro!: subset.unique-solution)

sublocale unique-solution i(|ivp-X:=X''|)
proof
  fix y t
  assume t ∈ T hence t: t ∈ subset.T by simp
  assume sol': is-solution y
  hence sol: subset.is-solution y
```

by (*rule sup-solution-is-solution*)
from *unique-solution*[*OF sol t*] **have** $y\ t = subset.solution\ t$ **.**
also
note *solutions-eq*[*OF* ‹$t \in T$›, *symmetric*]
finally show $y\ t = ivp.solution\ (i(\!|ivp\text{-}X := X''|\!))\ t$ **.**
qed

end

locale *unique-of-superset* =
sub?: *has-solution* +
super?: *unique-solution* $i(\!|ivp\text{-}X{:=}X'|\!)$ **for** X' +
assumes *subset*: $sub.X \subseteq X'$
begin

lemma *sub-is-solution*: *super.is-solution sub.solution*
using *sub.is-solutionD*[*OF sub.is-solution-solution*] *subset*
by (*intro is-solutionI*) *auto*

lemma *sub-eq-sup-solution*: $\bigwedge t.\ t \in T \Longrightarrow sub.solution\ t = super.solution\ t$
by (*auto intro*!: *super.unique-solution sub-is-solution*)

sublocale *unique-solution*
proof
fix *y t*
assume *sub.is-solution y*
and $t \in sub.T$
from *this* **have** *t*: $t \in super.T$
and *y*: *super.is-solution y*
by (*auto intro*!: *sub.is-solution-on-superset-domain*[*OF - subset*])
show $y\ t = sub.solution\ t$
using *y*
unfolding *sub-eq-sup-solution*[*OF t*]
by (*rule super.unique-solution*[*OF - t*])
qed

end

locale *derivative-on-prod* =
fixes *T X* **and** $f{::}(real \times 'a{::}banach) \Rightarrow 'a$ **and** $f'{::}\ real \times 'a \Rightarrow (real \times 'a) \Rightarrow 'a$
assumes *nonempty*: $T \neq \{\}\ X \neq \{\}$
assumes f': $\bigwedge tx.\ tx \in T \times X \Longrightarrow$
$(f$ *has-derivative* $(f'\ tx))$ (*at tx within* $(T \times X))$

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

3.7 Picard-Lindeloef On Open Domains

3.7.1 Local Solution with local Lipschitz

lemma *cube-in-cball*:
  fixes $x\ y :: 'a::euclidean\text{-}space$
  assumes $r > 0$
  assumes $\bigwedge i.\ i \in Basis \Longrightarrow dist\ (x \cdot i)\ (y \cdot i) \le r\ /\ sqrt(DIM('a))$
  shows $y \in cball\ x\ r$
  unfolding *mem-cball euclidean-dist-l2*[*of x y*] *setL2-def*
proof −
  have $(\sum i{\in}Basis.\ (dist\ (x \cdot i)\ (y \cdot i))^2) \le (\sum (i::'a){\in}Basis.\ (r\ /\ sqrt(DIM('a)))^2)$
  proof (*intro setsum-mono*)
    fix $i :: 'a$
    assume $i \in Basis$
    thus $(dist\ (x \cdot i)\ (y \cdot i))^2 \le (r\ /\ sqrt(DIM('a)))^2$
      using *assms*
      by (*auto intro*: *sqrt-le-rsquare*)
  qed
  moreover
  have $... \le r^2$
    using *assms* **by** (*simp add*: *power-divide*)
  ultimately
  show $sqrt\ (\sum i{\in}Basis.\ (dist\ (x \cdot i)\ (y \cdot i))^2) \le r$
    using *assms* **by** (*auto intro*!: *real-le-lsqrt setsum-nonneg*)
qed

lemma *cbox-in-cball'*:
  fixes $x::'a::euclidean\text{-}space$
  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 DIM-positive* **by** (*intro divide-left-mono*) *auto*
  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∈ Basis
      with bounds
      have bounds-comp:
        x · i − r / sqrt (real DIM('a)) ≤ y · i
        y · i ≤ x · i + r / sqrt (real DIM('a))
        by (auto simp: algebra-simps)
      thus dist (x · i) (y · i) ≤ r / sqrt (real DIM('a))
        unfolding dist-real-def by simp
    qed (auto simp add: assms)
  qed (rule)
qed (auto simp: assms DIM-positive)

locale ivp-open = ivp +
  assumes openT: open T
  assumes openX: open X

lemma Pair1-in-Basis: i ∈ Basis ⟹ (i, 0) ∈ Basis
 and Pair2-in-Basis: i ∈ Basis ⟹ (0, i) ∈ Basis
  by (auto simp: Basis-prod-def)

lemma Basis-prodD:
  assumes (i, j) ∈ Basis
  shows i ∈ Basis ∧ j = 0 ∨ i = 0 ∧ j ∈ Basis
  using assms
  by (auto simp: Basis-prod-def)

lemma cball-Pair-split-subset: cball (a, b) c ⊆ cball a c × cball b c
  apply (auto simp: dist-prod-def)
   apply (metis dual-order.trans le-real-sqrt-sumsq power2-eq-square)
  by (metis add.commute dual-order.trans le-real-sqrt-sumsq power2-eq-square)

lemma cball-times-subset: cball a (c/2) × cball b (c/2) ⊆ 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
    also assume b' ∈ cball b (c / 2)
    then have dist b b' ≤ c / 2 by simp
    finally have sqrt ((dist a a')² + (dist b b')²) ≤ c
      by simp
  } thus ?thesis by (auto simp: dist-prod-def)
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)
  {
    fix t x assume t ∈ cball t0 (d/3) x ∈ cball x0 (d/3)
    hence 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
        intro!: le-less-trans[OF sqrt-sum-squares-le-sum-abs])
    hence norm (f (t, x)) ≤ norm (f (t0, x0)) + 1
      by norm
  } note bound = this
  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 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
    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 unique-on-open = ivp-open + continuous-rhs T X f +
  assumes local-lipschitz: local-lipschitz T X (λt x. f (t, x))
begin

lemma eventually-lipschitz:
  assumes t ∈ T x ∈ X c > 0
```

obtains L **where**
eventually ($\lambda u. \forall t' \in$ *cball* t ($c * u$) $\cap$ T.
lipschitz (*cball* x u $\cap$ X) ($\lambda y.$ f (t', y)) L) (*at-right* 0)
proof −
from *local-lipschitzE*[*OF local-lipschitz*, *OF* ‹$t \in T$› ‹$x \in X$›]
obtain u L **where**
$u > 0$
$\bigwedge t'. t' \in$ *cball* t u $\cap$ $T \Longrightarrow$ *lipschitz* (*cball* x u $\cap$ X) ($\lambda y.$ f (t', y)) L
by *auto*
hence *eventually* ($\lambda u. \forall t' \in$ *cball* t ($c * u$) $\cap$ T.
lipschitz (*cball* x u $\cap$ X) ($\lambda y.$ f (t', y)) L) (*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-subset*[**where** D=*cball* x u $\cap$ X])
thus *?thesis* **..**
qed

lemma *eventually-unique-solution*:
obtains B L t
where $t > 0$ *eventually* ($\lambda e.$ $e > 0 \wedge$ *cball* $t0$ ($t * e$) $\subseteq T \wedge$ *cball* $x0$ $e \subseteq X \wedge$
(*unique-on-cylinder* (i(|*ivp-T*:=*cball* $t0$ ($t * e$), *ivp-X*:=*cball* $x0$ e|)) ($t * e$) e
B L (*cball* $x0$ e)))
(*at-right* 0)
proof −
from *open-Times*[*OF openT openX*] **have** *open* ($T \times X$) **.**
from *at-within-open*[*OF - this*] *iv-defined*
have *isCont* f ($t0$, $x0$)
using *continuous* **by** (*auto simp*: *continuous-on-eq-continuous-within*)
from *eventually-bound-pairE*[*OF this*]
obtain B **where** B:
$1 \leq B$ $\forall_F$ e *in at-right* 0. $\forall x \in$ *cball* $t0$ e $\times$ *cball* $x0$ e. *norm* (f x) $\leq B$
.
moreover

def $t \equiv$ *inverse* B
have *te*: $\bigwedge e.$ $e > 0 \Longrightarrow t * e > 0$
using ‹$1 \leq B$› **by** (*auto simp*: *t-def field-simps*)
have *t-pos*: $t > 0$
using ‹$1 \leq B$› **by** (*auto simp*: *t-def*)

from B(2) **obtain** *dB* **where** $0 <$ *dB* $0 <$ *dB* / 2
and *dB*: $\bigwedge d$ t x. $d > 0 \Longrightarrow d <$ *dB* $\Longrightarrow$ $t \in$*cball* $t0$ $d \Longrightarrow$ $x \in$*cball* $x0$ $d \Longrightarrow$
norm (f (t, x)) $\leq B$
by (*auto simp*: *eventually-at dist-real-def*)
hence *dB'*: $\bigwedge t$ x. (t, x) $\in$ *cball* ($t0$, $x0$) (*dB* / 2) $\Longrightarrow$ *norm* (f (t, x)) $\leq B$
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 *eventually*
(λ*e*. ∀ *x*∈*cball t0* (*t* ∗ *e*) × *cball x0 e*. *norm* (*f x*) ≤ *B*)
(*at-right 0*)
unfolding *eventually-at-filter*
by *eventually-elim* (*auto intro*!: *dB*′)
moreover

from *eventually-lipschitz*[*OF iv-defined t-pos*] **obtain** *L* **where**
eventually (λ*u*. ∀ *t*′∈*cball t0* (*t* ∗ *u*) ∩ *T*.
lipschitz (*cball x0 u* ∩ *X*) (λ*y*. *f* (*t*′, *y*)) *L*) (*at-right 0*)
.
moreover
have *eventually* (λ*e*. *cball t0* (*t* ∗ *e*) ⊆ *T*) (*at-right 0*)
using *eventually-open-cball*[*OF openT iv-defined*(*1*)]
by (*subst eventually-filtermap*[*symmetric*, **where** *f*=*op* ∗ *t*])
(*simp add*: *filtermap-times-real t-pos*)
moreover
have *eventually* (λ*e*. *cball x0 e* ⊆ *X*) (*at-right 0*)
using *openX iv-defined*(*2*)
by (*rule eventually-open-cball*)
ultimately have *eventually* (λ*e*. *e* > *0* ∧ *cball t0* (*t* ∗ *e*) ⊆ *T* ∧ *cball x0 e* ⊆
X ∧
(*unique-on-cylinder* (*i*⦇*ivp-T*:=*cball t0* (*t* ∗ *e*), *ivp-X*:=(*cball x0 e*)⦈) (*t* ∗ *e*)
e B L (*cball x0 e*)))
(*at-right 0*)
unfolding *eventually-at-filter*
proof *eventually-elim*
case (*elim e*)
thus *?case*
proof *safe*
fix *X*′ **assume** ∗: *cball x0 e* ⊆ *X*′
assume *e*: *0* < *e*
assume *L*: ∀ *t*′∈*cball t0* (*t* ∗ *e*) ∩ *T*.
lipschitz (*cball x0 e* ∩ *X*) (λ*y*. *f* (*t*′, *y*)) *L*
assume *B*: ∀ *x*∈*cball t0 e* × *cball x0 e*. *norm* (*f x*) ≤ *B*
assume *B*′: ∀ *x*∈*cball t0* (*t* ∗ *e*) × *cball x0 e*. *norm* (*f x*) ≤ *B*
assume *T*: *cball t0* (*t* ∗ *e*) ⊆ *T*
assume *X*: *cball x0 e* ⊆ *X*
have *t0* ∈ *cball t0* (*t* ∗ *e*) ∩ *T* **using** *T*
by (*force simp*: *e t-pos intro*!: *mult-nonneg-nonneg less-imp-le*)
hence *L*′: *lipschitz* (*cball x0 e* ∩ *X*) (λ*y*. *f* (*t0*, *y*)) *L* **using** *L*
by *simp*
hence *L* ≥ *0*
by (*rule lipschitz-nonneg*)
from *T X* **have** *subset*: *cball t0* (*t* ∗ *e*) × *cball x0 e* ⊆ *T* × *X* **by** *auto*
let *?i* = (*i*⦇*ivp-T* := *cball t0* (*t* ∗ *e*), *ivp-X* := *cball x0 e*⦈)
interpret *i*: *cylinder ?i t* ∗ *e e* **using** ‹*e* > *0*› *T te*[*OF* ‹*e* > *0*›]

```
        by unfold-locales (auto simp: cball-def dist-real-def)
      interpret i: continuous-rhs i.T i.X i.f
        using continuous-on-subset[OF continuous subset]
        by unfold-locales auto
      interpret i: solution-in-cylinder ?i t ∗ e e B
        using B′
        by unfold-locales (auto simp: t-def cball-def dist-real-def inverse-eq-divide)
      show unique-on-cylinder ?i (t ∗ e) e B L (cball x0 e)
        using L ‹L ≥ 0› te T X
        by unfold-locales
          (auto simp: cball-def dist-real-def abs-real-def
            dest!: bspec
            intro: lipschitz-subset)
    qed
  qed
  with t-pos show ?thesis ..
qed

lemma exists-unique-solution-abstracted:
  shows ∃ e>0. ∃ u>0. cball t0 e ⊆ T ∧ cball x0 u ⊆ X ∧
    (∀ X. cball x0 u ⊆ X ⟶ unique-solution (i(|ivp-T:=cball t0 e, ivp-X:=X|)))
proof −
  from eventually-unique-solution obtain B L t
  where ∗: 0 < t
    ∀F e in at-right 0. 0 < e ∧ cball t0 (t ∗ e) ⊆ T ∧ cball x0 e ⊆ X ∧
       unique-on-cylinder (i(|ivp-T := cball t0 (t ∗ e), ivp-X := cball x0 e|))
        (t ∗ e) e B L (cball x0 e) .
  from eventually-happens[OF ∗(2)]
  obtain e where e: 0 < e
        cball t0 (t ∗ e) ⊆ T
        cball x0 e ⊆ X
        unique-on-cylinder (i(|ivp-T := cball t0 (t ∗ e), ivp-X := cball x0 e|))
          (t ∗ e) e B L (cball x0 e)
    by auto
  then
  interpret uc:
    unique-on-cylinder i(|ivp-T := cball t0 (t ∗ e), ivp-X := cball x0 e|)
      t ∗ e e B L cball x0 e
    by simp
  {
    fix s assume s ∈ cball t0 (t ∗ e)
    hence abs (s − t0) ≤ abs (t ∗ e )
      by (auto simp: cball-def dist-real-def)
    hence B ∗ |s − t0| ≤ B ∗ abs (t ∗ e)
      using ∗ e uc.B-nonneg
      by (intro mult-left-mono)
        (auto simp: cball-def dist-real-def abs-real-def algebra-simps)
    also have abs (t ∗ e) = t ∗ e
      using ∗ e by simp
```

```
    also note uc.e-bounded
    finally have B * |s − t0| ≤ e
      using uc.B-nonneg e
      by (cases B = 0) (auto)
  } note cylinder-le = this
  show ?thesis
    apply (rule exI[where x=t * e])
    apply (rule conjI)
    subgoal using *(1) e by simp
    subgoal
    proof (safe intro!: exI[where x=e] e)
      fix X′ assume cball x0 e ⊆ X′
      then interpret us:
        unique-on-superset-domain
          i(|ivp-T := cball t0 (t * e), ivp-X := cball x0 e|) X′
        apply unfold-locales
        subgoal by simp
        subgoal
          using e *(1)
          by (auto simp: dist-real-def abs-real-def closed-segment-real; fail)
        subgoal
          using uc.e-bounded uc.B-nonneg
          by (intro set-rev-mp[OF uc.solves-in-cone])
           (auto intro!: has-vector-derivative-continuous-on subset-cball uc.solves-in-cone
                open-closed-segment cylinder-le
                has-vector-derivative-within-subset[OF - open-closed-segment-subset])
        done
      have unique-solution
          (i(|ivp-T := cball t0 (t * e), ivp-X := cball x0 e, ivp-X := X′|)) ..
      thus unique-solution (i(|ivp-T := cball t0 (t * e), ivp-X := X′|))
        by simp
    qed
    done
qed

lemma
  eventually-less-at-right:
  fixes a b::real shows
  b > a ⟹ eventually (λe. e < b) (at-right a)
  by (auto simp: eventually-at-le dist-real-def intro!: exI[where x=(b − a)/2])

lemma exists-unique-solution-legacy:
  assumes t0 < t-max
  shows ∃ t1∈{t0<..t-max}. ∃ u>0. {t0..t1} ⊆ T ∧ cball x0 u ⊆ X ∧
    (∀ X. cball x0 u ⊆ X ⟶ unique-solution (i(|ivp-T:={t0..t1}, ivp-X:=X|)))
proof −
  from eventually-unique-solution obtain B L t
  where *: 0 < t
    ∀F e in at-right 0. 0 < e ∧ cball t0 (t * e) ⊆ T ∧ cball x0 e ⊆ X ∧
```

unique-on-cylinder (*i*(|*ivp-T* := *cball t0* (*t* ∗ *e*), *ivp-X* := *cball x0 e*|))
(*t* ∗ *e*) *e B L* (*cball x0 e*) **.**
have *eventually* (λ*e*. *e* < *t-max* − *t0*) (*at-right 0*)
using *assms* **by** (*simp add*: *eventually-less-at-right*)
hence *less*: *eventually* (λ*e*. *t* ∗ *e* < *t-max* − *t0*) (*at-right 0*)
apply (*subst eventually-filtermap*[*symmetric*, **where** *f*=*op* ∗ *t*])
apply (*subst filtermap-times-real*[*OF* ∗(*1*)])
apply *assumption*
done
from *eventually-conj*[*OF* ∗(*2*) *less*, *THEN eventually-happens*]
obtain *e* **where** *e*: *0* < *e cball t0* (*t* ∗ *e*) ⊆ *T cball x0 e* ⊆ *X*
unique-on-cylinder (*i*(|*ivp-T* := *cball t0* (*t* ∗ *e*), *ivp-X* := *cball x0 e*|))
(*t* ∗ *e*) *e B L* (*cball x0 e*) *t0* + *t* ∗ *e* < *t-max*
by *auto*
then interpret *uc*:
unique-on-cylinder
i(|*ivp-T* := *cball t0* (*t* ∗ *e*), *ivp-X* := *cball x0 e*|)
t ∗ *e e B L cball x0 e*
by *simp*
have *unique-solution* (*i*(|*ivp-T* := *cball t0* (*t* ∗ *e*), *ivp-X* := *cball x0 e*, *ivp-T* :=
{*uc.t0* .. *uc.t0* + *t* ∗ *e*}|))
(**is** *unique-solution ?i*)
..
also have *?i* = (*i*(|*ivp-T* := {*t0* .. *t0* + *t* ∗ *e*}, *ivp-X* := *cball x0 e*|))
(**is** - = *?j*)
by *simp*
finally interpret *up*: *unique-solution ?j* **.**
{
fix *s*
have *s* − *t0* ≤ *abs* (*s* − *t0*) **by** *simp*
also
assume *s* ∈ *cball t0* (*t* ∗ *e*)
hence *abs* (*s* − *t0*) ≤ *abs* (*t* ∗ *e*)
by (*auto simp*: *cball-def dist-real-def*)
hence *B* ∗ |*s* − *t0*| ≤ *B* ∗ *abs* (*t* ∗ *e*)
using ∗ *e uc.B-nonneg*
by (*intro mult-left-mono*)
(*auto simp*: *cball-def dist-real-def abs-real-def algebra-simps*)
also have *abs* (*t* ∗ *e*) = *t* ∗ *e*
using ∗ *e* **by** *simp*
also note *uc.e-bounded*
finally have *B* ∗ (*s* − *t0*) ≤ *e*
using *uc.B-nonneg e*
by (*cases B* = *0*) (*auto*)
} **note** *cylinder-le* = *this*
show *?thesis*
apply (*rule bexI*[**where** *x*=*t0* + *t* ∗ *e*])
subgoal
proof (*safe intro*!: *exI*[**where** *x*=*e*] *e*)

fix *x* **assume** $x \in \{t0\ ..\ t0 + t * e\}$ **then show** $x \in T$
  using $*(1)$ *e*
  by (*simp add*: *subset-iff dist-real-def*)
next
  fix X' **assume** *cball x0 e* $\subseteq X'$
  then interpret *us*: *unique-on-superset-domain*
    i(|*ivp-T* := {*t0* .. *t0* + *t* ∗ *e*}, *ivp-X* := *cball x0 e*|) X'
  apply *unfold-locales*
    apply (*simp*; *fail*)
   using *e* ∗(*1*)
   apply (*auto simp*: *dist-real-def abs-real-def closed-segment-real*; *fail*)[*1*]
  apply (*simp del*: *mem-cball*)
  apply (*rule set-rev-mp*)
   apply (*rule uc.solves-in-cone*)
  using *uc.e-bounded uc.B-nonneg cylinder-le*
  by (*auto*
    intro!: *has-vector-derivative-continuous-on subset-cball*
      has-vector-derivative-within-subset[*OF* - *open-closed-segment-subset*]
      open-closed-segment cylinder-le
    simp: *dist-real-def*)
  have *unique-solution*
    (*i*(|*ivp-T* := {*t0* .. *t0* + (*t* ∗ *e*)}, *ivp-X* := *cball x0 e*, *ivp-X* := X'|)) **..**
  thus *unique-solution* (*i*(|*ivp-T* := {*t0* .. *t0* + (*t* ∗ *e*)}, *ivp-X* := X'|))
    by *simp*
qed
 subgoal using *e* **by** (*auto simp add*: *dist-real-def cball-def abs-real-def* ‹*t* > *0*›)
done
qed

lemma *exists-unique-solution-legacy*$'$:
  assumes *t0* < *t-max*
  shows ∃ *t1*∈{*t0*<..*t-max*}. {*t0*..*t1*} ⊆ *T* ∧ *unique-solution* (*i*(|*ivp-T*:={*t0*..*t1*}|))
proof−
  from *exists-unique-solution-legacy*[*OF assms*]
  obtain *t1 u* **where** ∗: *t1*∈{*t0*<..*t-max*} *u* > *0*
    {*t0*..*t1*} ⊆ *T cball x0 u* ⊆ *X* (∀ *X*. *cball x0 u* ⊆ *X* ⟶ *unique-solution* (*i*(|*ivp-T* := {*t0*..*t1*}, *ivp-X* := *X*|)))
    by *auto*
  show *?thesis*
    using ∗(*1*−*4*) ∗(*5*)[*rule-format*, *OF* ‹*cball x0 u* ⊆ *X*›]
    by (*auto intro*!: *bexI*[**where** *x*=*t1*])
qed

3.7.2 Global maximal solution with local Lipschitz

definition *PHI* **where**
  PHI = {(*x*, *t1*). *t0* < *t1* ∧ {*t0*..*t1*} ⊆ *T* ∧ *ivp.is-solution* (*i*(|*ivp-T*:={*t0*..*t1*}|)) *x*}

lemma *PHI-notempty*: $PHI \neq \{\}$
proof −
  from *exists-unique-solution-legacy*[**where** *t-max=t0+1*]
  obtain *t1 a* **where**
    $\bigwedge X$. *cball x0 a* ⊆ *X* ⟹ *unique-solution* (*i*(|*ivp-T*:={*t0..t1*}, *ivp-X*:=*X*|))
    t0 < *t1* {*t0..t1*} ⊆ *T cball x0 a* ⊆ *X*
    by *force*
  from *this*(*1*)[*OF this*(*4*)] **interpret** *i*: *unique-solution i*(|*ivp-T*:={*t0..t1*}|)
    by *auto*
  from *i.is-solution-solution* ‹*t0*<*t1*› ‹{*t0..t1*} ⊆ *T*›
  have (*i.solution*, *t1*) ∈ *PHI*
    by (*simp add*: *PHI-def*)
  thus *?thesis* **by** *auto*
qed

lemma *positive-existence-interval*:
  assumes *E*: ∀(*x*, *t1*) ∈ *PHI*. ∀(*y*, *U*) ∈ *PHI*. ∀*t* ∈ {*t0..t1*} ∩ {*t0..U*}. *x t* =
y t
  defines *J* ≡ ⋃(*x*, *t1*)∈*PHI*. {*t0..t1*}
  defines *j* ≡ *i*(|*ivp-T*:=*J*|)
  defines *M* ≡ (*SUP xt* : *PHI*. *ereal* (*snd xt*))
  shows *unique-solution j*
    ⋀*x t1 t*. (*x*, *t1*) ∈ *PHI* ⟹ *t* ∈ {*t0..t1*} ⟹ *x t* = *ivp.solution* (*i*(|*ivp-T*:=*J*|))
t
    J = *real-of-ereal* ‘ {*ereal t0..*<*M*}
    t0 ∈ *J*
proof −
  from *PHI-def* **have** *PHI*: *PHI* = {*xT*. *t0* < *snd xT* ∧ {*t0..snd xT*} ⊆ *T* ∧
    ivp.is-solution (*i*(|*ivp-T*:={*t0..snd xT*}|)) (*fst xT*)}
    by *auto*
  from *PHI-notempty* **obtain** *a b* **where** (*a*, *b*) ∈ *PHI* **by** *auto*
  hence *t0* ≤ *b* **by** (*simp add*: *PHI-def*)
  thus *t0* ∈ *J*
    using ‹(*a*, *b*) ∈ *PHI*›
    by (*auto simp*: *J-def intro*!: *bexI*[**where** *x*=(*a*, *b*)])
  {
    fix *x y t t1*
    assume
      ivp.is-solution (*i*(|*ivp-T*:={*t0..t1*}|)) *x*
      ivp.is-solution (*i*(|*ivp-T*:={*t0..t1*}|)) *y*
      t ∈ {*t0..t1*} *t0*<*t1* {*t0..t1*} ⊆ *T*
    moreover
    hence (*x*, *t1*) ∈ *PHI* (*y*, *t1*) ∈ *PHI*
      by (*auto simp*: *PHI*)
    ultimately have *x t* = *y t* **using** *E* **by** *force*
  } **note** *sol-eq* = *this*
  from *E* **have** *E*: ∀*xT* ∈ *PHI*. ∀*yU* ∈ *PHI*. ∀*t* ∈ {*t0..snd xT*} ∩ {*t0..snd yU*}.
    (*fst xT*) *t* = (*fst yU*) *t* **by** *force*

```
have J: (⋃(x, t1)∈PHI. {t0..t1}) = (⋃xT∈PHI. {t0..snd xT})
  by auto
with j-def J-def have j-def': j = i(|ivp-T:=⋃xT∈PHI. {t0..snd xT}|) by simp
have J ⊆ T unfolding J-def j-def PHI-def by auto
have ∃x. ∀t ∈ J. ∀yT ∈ PHI. t ≤ snd yT ⟶ x t = fst yT t
proof (intro bchoice, safe)
  fix x
  assume xI: x ∈ J
  hence ∃s∈PHI. x ≤ snd s unfolding J-def PHI-def by auto
  then obtain ya where ya: ya ∈ PHI x ≤ snd ya by auto
  with E[simplified Ball-def, THEN spec, THEN mp, OF ya(1)]
  have E':∀zb∈PHI. x ∈ {t0..snd ya} ∩ {t0..snd zb} ⟶ fst ya x = fst zb x
    by (simp add: Ball-def)
  show ∃y. ∀za∈PHI. x ≤ snd za ⟶ y = fst za x
  proof (rule, rule, rule)
    fix zb
    assume zb: zb ∈ PHI x ≤ snd zb
    with E'[simplified Ball-def, THEN spec, THEN mp, OF ‹zb ∈ PHI›]
    have x ∈ {t0..snd ya} ∩ {t0..snd zb} ⟶ fst ya x = fst zb x by (simp add:
Ball-def)
    thus fst ya x = fst zb x using xI ya zb J-def PHI-def by auto
  qed
qed
then obtain y where y: ∀t∈J. ∀yT∈PHI. t ≤ snd yT ⟶ y t = fst yT t
  by auto
hence equal: ∀s∈PHI. ∀t ∈ {t0..snd s}. y t = fst s t using J-def PHI-def
  by simp
{
  fix x
  assume x ∈ J
  have ∃s∈PHI. x < snd s
  proof –
    obtain s where s: s ∈ PHI x ≤ snd s using ‹x ∈ J›
      by (force simp add: PHI-def J-def)
    def i1 ≡ i(|ivp-T:={t0..snd s}|)
    interpret i1: ivp i1
      using s iv-defined ‹x ∈ J›
      by unfold-locales (auto simp: PHI-def J-def i1-def)
    from ‹s ∈ PHI› have t0 < snd s by (simp add: PHI)
    from ‹s ∈ PHI› have {t0..snd s} ⊆ T by (simp add: PHI)
    from ‹s ∈ PHI› have i1.is-solution (fst s) by (simp add: PHI i1-def)
    then interpret i1: unique-solution i1
    proof (intro i1.unique-solutionI, simp)
      fix y t
      assume i1.is-solution y
      assume t ∈ i1.T
      hence t ∈ {t0..snd s} by (simp add: i1-def)
      with sol-eq ‹i1.is-solution (fst s)› ‹i1.is-solution y›
        ‹t0 < snd s› ‹{t0..snd s} ⊆ T›
```

```
      show y t = fst s t by (simp add: i1-def)
    qed
    show ?thesis
    proof (cases x = snd s)
      assume x = snd s
      def i2' ≡ i(|ivp-t0:=snd s, ivp-x0:=fst s (snd s)|)
      interpret i2': unique-on-open i2'
        using iv-defined ‹x ∈ J› continuous openT openX local-lipschitz
          i1.is-solutionD(3)[OF ‹i1.is-solution (fst s)›] ‹s ∈ PHI›
        by unfold-locales (auto simp: PHI i1-def i2'-def)
      from i2'.exists-unique-solution-legacy[where t-max = snd s + 1]
      obtain t1 u where t1u: t1 > snd s {snd s..t1} ⊆ T 0 < u
        cball (fst s (snd s)) u ⊆ ivp-X i2'
        ⋀X. cball (fst s (snd s)) u ⊆ X ⟹
          unique-solution
            (i(|ivp-t0:=snd s, ivp-x0:=fst s (snd s), ivp-T:={snd s..t1},
              ivp-X := X|))
         by (auto simp: i2'-def)
      def i2 ≡ i(|ivp-t0:=snd s, ivp-x0:=fst s (snd s), ivp-T:={snd s..t1}|)
      interpret i2: unique-solution i2 using t1u(5)[OF t1u(4)]
        by (simp add: i2-def i2'-def)
      def ic ≡ i(|ivp-T:={t0..t1}|)
      interpret ic: ivp-on-interval ic t1
        using iv-defined ‹t1 > snd s› ‹snd s > t0›
        by unfold-locales (auto simp: ic-def)
      interpret ic: connected-unique-solutions ic i1 i2 snd s
        using i1.unique-solution[OF ‹i1.is-solution (fst s)›]
          ‹snd s > t0› ‹t1 > snd s›
          i1.is-solution-solution
          i2.is-solution-solution
          i1.is-solutionD[OF i1.is-solution-solution]
          i2.is-solutionD[OF i2.is-solution-solution]
        by unfold-locales (auto simp: i1-def i2-def ic-def)
      have (ic.solution, t1) ∈ PHI
        using ‹t0 < snd s› ‹{t0..snd s} ⊆ T› t1u(1−4) ic.is-solution-solution
        by (force simp add: PHI ic-def)
      thus ?thesis using ‹x = snd s› ‹snd s < t1› by force
    qed (insert s, force)
  qed
} note continuable=this

{
  fix x a b
  assume (a, b) ∈ PHI t0 ≤ x x ≤ b
  hence x ∈ J
    by (force simp: PHI-def J-def)
} note inJ = this
show J = real-of-ereal ‘ {t0..<M}
  unfolding J-def M-def
```

```
    by safe
     (auto simp: ereal-le-real-iff real-le-ereal-iff less-SUP-iff
       intro!: image-eqI[where x=ereal x for x] continuable inJ bexI[where x=(a,
b) for a b])

  interpret j: ivp j
    using iv-defined PHI-notempty
    by (unfold-locales, auto simp: j-def J-def PHI-def) force
  have j.is-solution y
  proof (intro j.is-solutionI)
    from PHI-notempty have ∃ya. ya ∈ PHI unfolding ex-in-conv .
    then obtain ya where ya: ya ∈ PHI ..
    then interpret iya: ivp i(|ivp-T:={t0..(snd ya)}|)
      using iv-defined by unfold-locales (auto simp: PHI)
    from ya have iya.is-solution (fst ya) by (simp add: PHI)
    from ya equal have y t0 = fst ya t0 by (auto simp: PHI)
    thus y j.t0 = j.x0
      using iv-defined iya.iv-defined
      using iya.is-solutionD(1)[OF ‹iya.is-solution (fst ya)›]
      by (auto simp: j-def)
  next
    fix x
    assume x ∈ j.T
    hence x ∈ J by (simp add: j-def)
    note continuable[OF this]
    then obtain ya where ya: ya ∈ PHI x < snd ya ..
    then interpret iya: ivp i(|ivp-T:={t0..snd ya}|)
      using iv-defined by unfold-locales (auto simp: PHI)
    from ya have iya.is-solution (fst ya) by (simp add: PHI)
    from iya.is-solutionD(2)[OF this]
    have deriv:
      (fst ya has-vector-derivative f (x, fst ya x)) (at x within {t0..snd ya})
      using ‹x ∈ j.T› J-def ya by (auto simp add: j-def)
    moreover
    from ‹x ∈ j.T› ya have x∈{t0..<snd ya} by (auto simp add: J-def j-def)
    with equal ya have y-eq-x: y x = fst ya x by simp
    ultimately
    show (y has-vector-derivative j.f (x, y x)) (at x within j.T)
      apply (simp (no-asm) add: j-def J-def)
      unfolding J
      unfolding has-vector-derivative-def
      unfolding has-derivative-within'
    proof safe
      fix e::real
      assume e > 0 ∀e>0. ∃d>0. ∀x'∈{t0..snd ya}.
        0 < norm (x' − x) ∧ norm (x' − x) < d ⟶
        norm (fst ya x' − fst ya x − (x' − x) *R f (x, fst ya x)) / norm (x' − x)
        < e
      then obtain d where d: d > 0
```

```
    ⋀x'. x'∈{t0..snd ya} ⟹ x' ≠ x ⟹ |x' − x| < d ⟹
    norm (fst ya x' − fst ya x − (x' − x) *R f (x, fst ya x)) / |x' − x| < e
    by auto
  show ∃ d>0. ∀ x'∈⋃ s∈PHI. {t0..snd s}.
    0 < norm (x' − x) ∧ norm (x' − x) < d ⟶
    norm (y x' − y x − (x' − x) *R f (x, y x)) / norm (x' − x) < e
  proof (rule, rule)
    show Min {d, snd ya − x} > 0 using d ya by simp
  next
    have ∀ a∈PHI. ∀ x'∈{t0..snd a}.
      x' ≠ x ∧ |x' − x| < Min {d, snd ya − x} ⟶
      norm (y x' − fst ya x − (x' − x) *R f (x, fst ya x)) / |x' − x| < e
    proof (rule, rule, rule)
      fix t and x'
      assume A: t ∈ PHI
        x' ∈ {t0..snd t}
        x' ≠ x ∧ |x' − x| < Min {d, snd ya − x}
      with d
      have x' ≠ x ∧ |x' − x| < d ⟶
        norm (fst ya x' − fst ya x − (x' − x) *R f (x, fst ya x)) / |x' − x| < e
        by auto
      moreover
      from A have x' ≠ x ∧ |x' − x| < d by simp
      moreover
      from A have x'∈{t0..snd ya} by auto
      with A have y x' = fst ya x' using equal ya by fast
      ultimately show
        norm (y x' − fst ya x − (x' − x) *R f (x, fst ya x)) / |x' − x| < e
        by simp
    qed
    thus ∀ x'∈⋃ s∈PHI. {t0..snd s}.
      0 < norm (x' − x) ∧ norm (x' − x) < Min {d, snd ya − x} ⟶
      norm (y x' − y x − (x' − x) *R f (x, y x)) / norm (x' − x) < e
      using y-eq-x by simp
  qed
 qed simp
 from iya.is-solutionD(3)[OF ‹iya.is-solution (fst ya)›]
 have fst ya x ∈ X
   using ‹x ∈ j.T› ya by (auto simp: PHI-def j-def J-def)
 moreover
 from ‹x ∈ j.T› ya have x∈{t0..snd ya} by (auto simp: PHI-def j-def J-def)
 with equal ya have y-eq-x: y x = fst ya x by simp
 ultimately
 show y x ∈ j.X by (auto simp: j-def J-def)
qed
thus unique-solution j
proof (rule j.unique-solutionI)
 fix x t
 assume t ∈ j.T
```

```
    hence t ∈ J by (simp add: j-def)
    note continuable[OF this]
    then obtain x' t1 where x't1: (x', t1) ∈ PHI t < t1 {t0..t1} ⊆ T
      by (auto simp: PHI)
    then interpret ix': ivp i(|ivp-T:={t0..t1}|)
      using iv-defined by unfold-locales (auto simp: PHI)
    havet0 ≤ t using ‹t ∈ J› unfolding J-def by auto
    from x't1 have ix'.is-solution x' by (simp add: PHI)
    assume j.is-solution x
    hence ix'.is-solution x
      using x't1 ‹t ∈ J› ‹{t0..t1} ⊆ T›
      by (intro j.solution-on-subset[where T'={t0..t1}, simplified j-def,
        simplified]) (auto simp: J-def j-def)
    from equal x't1 ‹t∈j.T› have y t = x' t by (auto simp: j-def J-def)
    thus x t = y t
      using sol-eq[OF ‹ix'.is-solution x'› ‹ix'.is-solution x›] ‹t < t1› ‹t ∈ j.T›
      ‹{t0..t1} ⊆ T›
      by (auto simp: j-def J-def)
  qed
  then interpret j: unique-solution j by simp
  fix x t1 t
  assume (x, t1) ∈ PHI t ∈ {t0..t1}
  then interpret i': ivp i(|ivp-T := {t0..t1}|) using iv-defined
    by unfold-locales auto
  from ‹(x, t1) ∈ PHI› have x: i'.is-solution x t0 < t1 {t0..t1} ⊆ T
    by (auto simp add: PHI-def)
  have i'.is-solution j.solution
    apply (rule j.solution-on-subset[simplified j-def, simplified])
    using x ‹(x, t1) ∈ PHI› j.is-solution-solution
    by (auto simp: j-def J-def)
  from sol-eq[OF x(1) this ‹t ∈ {t0..t1}› ‹t0 < t1› ‹{t0..t1} ⊆ T›]
  show x t = ivp.solution (i(|ivp-T:=J|)) t by (simp add: j-def)
qed

lemma E:
  shows ∀(x, t1)∈PHI. ∀(y, t2)∈PHI. ∀t∈{t0..t1} ∩ {t0..t2}. x t = y t
proof safe
  fix a b
  fix y z
  fix t
  assume (y, a) ∈ PHI (z, b) ∈ PHI
  hence bounds: t0 < a t0 < b
    and subsets: {t0..a} ⊆ T {t0..b} ⊆ T
    and y-sol: ivp.is-solution (i(|ivp-T := {t0..a}|)) y
    and z-sol: ivp.is-solution (i(|ivp-T := {t0..b}|)) z
    unfolding PHI-def by auto
  assume t ∈ {t0..a} t ∈ {t0..b}
  interpret i1: ivp i(|ivp-T := {t0..a}|)
    using bounds iv-defined by unfold-locales auto
```

```
interpret i2: ivp i(|ivp-T := {t0..b}|)
  using bounds iv-defined by unfold-locales auto
have ∀ t ∈ {t0..a} ∩ {t0..b}. y t = z t
proof (rule ccontr)
  assume ¬ (∀ x∈{t0..a} ∩ {t0..b}. y x = z x)
  hence ∃ x∈{t0..min a b}. y x ≠ z x by simp
  then obtain x1 where x1: x1 ∈ {t0..min a b} y x1 ≠ z x1 ..

  from i1.solution-continuous-on[OF y-sol]
 have continuous-on {t0..x1} y by (rule continuous-on-subset) (insert x1, simp)
  moreover
  from i2.solution-continuous-on[OF z-sol]
 have continuous-on {t0..x1} z by (rule continuous-on-subset) (insert x1, simp)
  ultimately have continuous-on {t0..x1} (λx. norm (y x − z x))
    by (auto intro: continuous-intros)
  moreover
  have closed {t0..x1} by simp
  ultimately
  have closed {t ∈ {t0..x1}. norm (y t − z t) = 0}
    by (rule continuous-closed-preimage-constant)
  moreover
  have t0 ∈ {t ∈ {t0..x1}. norm (y t − z t) = 0}
    using x1 i1.is-solutionD[OF y-sol] i2.is-solutionD[OF z-sol]
    by simp
  then have {t ∈ {t0..x1}. norm (y t − z t) = 0} ≠ {} by blast
  ultimately
  have ∃ m∈{t ∈ {t0..x1}. norm (y t − z t) = 0}.
    ∀ y∈{t ∈ {t0..x1}. norm (y t − z t) = 0}. dist x1 m ≤ dist x1 y
    by (rule distance-attains-inf) auto
  then guess x-max .. note max = this
  have z x-max = y x-max using max by simp
  have x-max ∈ {t0..min a b} x-max ∈ T
    using x1 z-sol y-sol max subsets by auto
  with x1 i1.is-solutionD[OF y-sol] have y x-max ∈ X
    by (simp add: is-solution-def)
  with max have z x-max ∈ X by simp
  def i3' ≡ i(|ivp-t0:=x-max, ivp-x0:=y x-max|)
  interpret i3': unique-on-open i3'
    using iv-defined continuous openT openX local-lipschitz
      i1.is-solutionD(3)[OF y-sol] ‹x-max ∈ T› ‹y x-max ∈ X›
    by unfold-locales (auto simp: PHI-def i3'-def)
  have x-max < x1 using x1 max by auto
  with i3'.exists-unique-solution-legacy'[where t-max = x1]
  obtain t1 where t1: t1 ∈{x-max<..x1} {x-max..t1} ⊆ T unique-solution
    (i(|ivp-t0:=x-max, ivp-x0:=y x-max, ivp-T:={x-max..t1}|))
    by (auto simp: i3'-def)
  def i3 ≡ i(|ivp-t0:=x-max, ivp-x0:=y x-max, ivp-T:={x-max..t1}|)
  from t1 interpret i3: unique-solution i3
    by (simp add: i3-def)
```

```
    have x-max ∈ {x-max..t1} using t1 by simp
    have i3.is-solution y unfolding i3-def
      using ‹y x-max ∈ X› ‹x-max ∈ {t0..min a b}› y-sol t1 x1(1)
        i1.restriction-of-solution by auto
    have i3.is-solution z unfolding i3-def
      using ‹z x-max ∈ X› ‹x-max ∈ {t0..min a b}› z-sol t1 x1(1)
        i2.restriction-of-solution
      by (auto simp: ‹z x-max = y x-max›[symmetric])
    let ?m = (x-max + t1) / 2
    have xm1: ?m ∈ {t0..t1} using max ‹x-max ∈ {x-max..t1}› by simp
    have xm2: ?m ∈ {x-max..t1} using max ‹x-max ∈ {x-max..t1}› by simp
    from i3.unique-solution[OF ‹i3.is-solution y›, of ?m]
      i3.unique-solution[OF ‹i3.is-solution z›, of ?m]
      ‹x-max ∈ {x-max..t1}›
    have eq: y ?m = z ?m
      by (simp add: i3-def)
    hence ?m ∈ {t ∈ {t0..x1}. norm (y t − z t) = 0} using max x1 t1 by simp
    with max have dist x1 x-max ≤ dist x1 ?m by auto
    moreover have dist x1 x-max = x1 − x-max
      unfolding dist-real-def using x1 max by simp
    moreover
    have x-max ≤ x1 using max by simp
    hence ?m ≤ x1 using max x1 t1 by simp
    hence dist x1 ?m = x1 − ?m
      using x1 max by (auto intro!: abs-of-nonneg simp add: dist-real-def)
    ultimately
    show False using max x1 t1 by simp
  qed
  thus y t = z t using ‹t ∈ {t0..a}› ‹t ∈ {t0..b}› by simp
qed

lemma global-solution:
  obtains J::real set and M::ereal where
  J = real-of-ereal ‘ {t0 ..< M}
  ⋀x. x ∈ J ⟹ t0 ≤ x
  J ⊆ T
  is-interval J
  t0 ∈ J
  unique-solution (i(|ivp-T:=J|))
  ⋀K x. K ⊆ T ⟹ is-interval K ⟹ t0 ∈ K ⟹ (⋀x. x ∈ K ⟹ t0 ≤ x) ⟹
    ivp.is-solution (i(|ivp-T:=K|)) x ⟹
    K ⊆ J ∧ (∀t∈K. x t = ivp.solution (i(|ivp-T:=J|)) t)
proof −
  def M ≡ SUP xt : PHI. ereal (snd xt)
  def J ≡ (⋃(x, t1)∈PHI. {t0..t1})
  show ?thesis
  proof
    show J = real-of-ereal ‘ {ereal t0 ..< M}
      using positive-existence-interval[OF E]
```

```
    by (simp add: J-def M-def)
  show J ⊆ T
    by (auto simp: PHI-def J-def)
  show is-interval J
    unfolding is-interval-def J-def PHI-def
    by auto (metis order.trans)+
  show t0 ∈ J using PHI-notempty
    by (force simp add: PHI-def J-def)
next
  fix x assume x ∈ J thus t0 ≤ x
    by (auto simp add: J-def PHI-def)
next
  show unique-solution (i(|ivp-T := J|))
    using positive-existence-interval[OF E] by (simp add: J-def)
  then interpret j: unique-solution i(|ivp-T := J|) by simp
  fix K z
  assume K ⊆ T
  def y ≡ ivp.solution (i(|ivp-T := J|))
  assume interval: is-interval K
  assume Inf: t0 ∈ K ⋀x. x ∈ K ⟹ t0 ≤ x
  assume z-sol: ivp.is-solution (i(|ivp-T := K|)) z
  then interpret k: has-solution i(|ivp-T := K|)
    using iv-defined Inf
    by unfold-locales auto
  have ∀x ∈ K. x ∈ J ∧ z x = y x
  proof (rule, cases, safe)
    fix xM
    def k1 ≡ i(|ivp-T:={t0..xM}|)
    assume xM-in: xM ∈ K
    assume t0 < xM
    then interpret k1: ivp k1 using iv-defined
      by unfold-locales (auto simp: k1-def)
    have subset: {t0..xM} ⊆ K
    proof
      fix t
      assume t ∈ {t0..xM}
      moreover
      from Inf(1) xM-in interval have (∀i∈Basis.
        t0 · i ≤ t · i ∧ t · i ≤ xM · i) ⟶
        t ∈ K unfolding is-interval-def by blast
      hence t ∈ {t0..xM} ⟶ t ∈ K by simp
      ultimately show t ∈ K by simp
    qed
    have k1.is-solution z
      using k.solution-on-subset z-sol subset ‹t0 < xM› by (simp add: k1-def)
    then interpret k1: has-solution k1 by unfold-locales auto
    interpret k2': unique-on-open i(|ivp-t0:=xM, ivp-x0:=z xM|)
      using ‹t0 < xM› k1.is-solutionD[OF ‹k1.is-solution z›]
        local-lipschitz openT openX continuous ‹K ⊆ T› ‹xM ∈ K›
```

```
      by unfold-locales (auto simp: k1-def)
    from k2'.exists-unique-solution-legacy'[where t-max = xM + 1, simplified]
    obtain t1 where t1: t1 ∈ {xM<..xM+1} {xM..t1} ⊆ T
      unique-solution (i(|ivp-t0 := xM, ivp-x0 := z xM, ivp-T := {xM..t1}|))
      by auto
    def k2≡i(|ivp-t0 := xM, ivp-x0 := z xM, ivp-T := {xM..t1}|)
    from t1 interpret k2: unique-solution k2 by (simp add: k2-def)
    def kc ≡ i(|ivp-T:={t0..t1}|)
    interpret kc: connected-solutions kc k1 k2 z
      using k1.is-solution-solution k2.is-solution-solution iv-defined
        ‹k1.is-solution z› ‹t0<xM› t1 k1.is-solutionD[OF ‹k1.is-solution z›]
        k2.is-solutionD[OF k2.is-solution-solution]
      by unfold-locales (auto simp: k1-def k2-def kc-def)
    have {t0..t1} ⊆ T
    proof −
      have {t0..t1} = {t0..xM} ∪ {xM..t1} using t1 ‹t0 < xM› by auto
      thus ?thesis using ‹{t0..xM} ⊆ K› ‹{xM..t1} ⊆ T› ‹K ⊆ T› by simp
    qed
    hence concrete-sol: (kc.connection, t1) ∈ PHI
      using ‹t0<xM› t1 ‹{t0..xM} ⊆ K› ‹K ⊆ T› kc.is-solution-connection
      by (auto simp add: PHI-def kc-def)
    moreover have xM ∈ {t0..<snd (kc.connection, t1)}
      using ‹t0<xM› t1 by simp
    ultimately
    show xM ∈ J by (force simp: PHI-def J-def)
    have xM ∈ {t0..t1} using t1 ‹t0 < xM› by simp
    from positive-existence-interval[OF E] J-def y-def concrete-sol this
    show z xM = y xM
      by (simp add: kc.connection-def[abs-def]) (simp add: k1-def)
  next
    fix x
    assume x ∈ K ¬ t0 < x
    hence x = t0 using Inf(2)[OF ‹x∈K›] by simp
    thus x ∈ J using PHI-notempty by (force simp: J-def PHI-def)
    from j.solution-t0 k.is-solutionD[OF z-sol]
    show z x = y x by (simp add: y-def ‹x = t0›)
  qed
  thus K ⊆ J ∧ (∀t∈K. z t = ivp.solution (i(|ivp-T := J|)) t)
    by (auto simp: y-def)
 qed
qed

definition
 maximal-existence-interval J =
  (J ⊆ T ∧
   is-interval J ∧
   t0 ∈ J ∧
   open J ∧
   unique-solution (i(|ivp-T:=J|)) ∧
```

(∀ K x. K ⊆ T ⟶ *is-interval* K ⟶ *t0* ∈ K ⟶ *ivp.is-solution* (*i*⦇*ivp-T*:=K⦈) x ⟶
K ⊆ J ∧ (∀ t ∈ K. x t = *ivp.solution* (*i*⦇*ivp-T*:=J⦈) t)))

lemma *maximal-existence-intervalE*:
obtains *M0 M1*::*ereal* **and** J **where**
J = *real-of-ereal* ' {*M0* <..< *M1*}
maximal-existence-interval J
proof −
from *global-solution* **obtain** J M **where** J:
J = *real-of-ereal* ' {*ereal t0* ..< M}
⋀x. x ∈ J ⟹ *t0* ≤ x
J ⊆ T
is-interval J
t0 ∈ J
unique-solution (*i*⦇*ivp-T*:=J⦈)
⋀K x. K ⊆ T ⟹ *is-interval* K ⟹ *t0* ∈ K ⟹ (⋀x. x ∈ K ⟹ *t0* ≤ x) ⟹
ivp.is-solution (*i*⦇*ivp-T*:=K⦈) x ⟹
K ⊆ J ∧ (∀ t∈K. x t = *ivp.solution* (*i*⦇*ivp-T*:=J⦈) t)
by *blast*
from *openT iv-defined*(*1*) **obtain** *dt* **where** *dt*: *dt* > *0 ball t0 dt* ⊆ T
by (*rule openE*)
hence *subs*: {*t0*..} ∩ *ball t0 dt* ⊆ T
by *auto*
have *is-ivl*: *is-interval* ({*t0*..} ∩ *ball t0 dt*)
by (*intro is-interval-inter is-interval-ci is-interval-ball-real*)
have *t0-in*: *t0* ∈ {*t0*..} ∩ *ball t0 dt* **using** *dt* **by** *auto*

let *?mirror* = λt. *2* ∗ *t0* − t
let *?nT* = *?mirror* ' T
let *?ni* = *i*⦇*ivp-T*:=*?nT*, *ivp-f*:=(λ(t, x). − f (*?mirror* t, x))⦈
have *continuous-on* (*op* − (*2* ∗ *t0*) ' T × X) (*uminus o f o* (λ(t, x). (*2*∗*t0* − t, x)))
using *dt*
by (*intro continuous-intros*)
(*auto intro*!: *continuous-intros continuous-on-subset*[*OF continuous*]
simp: *split-beta dist-real-def*)
then
interpret *neg*: *unique-on-open ?ni*
using *local-lipschitz*
by *unfold-locales*
(*auto simp*: *openX open-neg-translation openT iv-defined split-beta*
local-lipschitz-uminus continuous-on-op-minus image-image
intro: *local-lipschitz-compose1*)
from *neg.global-solution* **obtain** J′ M′ **where** J′:
J′ = *real-of-ereal* ' {*ereal* (*ivp-t0 ?ni*) ..< M′}
(⋀x. x ∈ J′ ⟹ *ivp-t0 ?ni* ≤ x)
J′ ⊆ *ivp-T ?ni*
is-interval J′

```
  ivp-t0 ?ni ∈ J'
  unique-solution (?ni(|ivp-T := J'|))
  (⋀K x. K ⊆ ivp-T ?ni ⟹ is-interval K ⟹ ivp-t0 ?ni ∈ K ⟹
    (⋀x. x ∈ K ⟹ ivp-t0 ?ni ≤ x) ⟹
    ivp.is-solution (?ni(|ivp-T := K|)) x ⟹
    K ⊆ J' ∧ (∀t∈K. x t = ivp.solution (?ni(|ivp-T := J'|)) t))
  by blast
interpret neg-unique: unique-solution ?ni(|ivp-T := J'|)
  by fact
let ?mJ' = ?mirror ` J'
let ?mi = i(|ivp-T := ?mJ'|)
interpret mi: ivp ?mi
  using J'(5) iv-defined
  by unfold-locales auto
interpret mi: has-solution ?mi
proof
  show ∃x. mi.is-solution x
    by (rule exI)
      (rule neg-unique.mirror-solution[simplified,
          OF neg-unique.is-solution-solution[simplified]])
qed
interpret mi: unique-solution ?mi
proof
  fix x t assume misol: mi.is-solution x and t: t ∈ mi.T
  have [simp]: op − (2 * t0) ` ?mJ' = J' by force
  from mi.mirror-solution[OF misol]
  have neg-unique.is-solution (x o ?mirror)
    by simp
  from neg-unique.unique-solution[OF this]
  have ⋀t. t ∈ J' ⟹ (x o ?mirror) t = neg-unique.solution t
    by auto
  moreover
  from mi.mirror-solution[OF mi.is-solution-solution, simplified]
  have neg-unique.is-solution (mi.solution o ?mirror)
    by simp
  from neg-unique.unique-solution[OF this, simplified]
  have ⋀t. t ∈ J' ⟹ (mi.solution o ?mirror) t = neg-unique.solution t
    by auto
  ultimately
  have ⋀t. t ∈ J' ⟹ (x o ?mirror) t = (mi.solution o ?mirror) t
    by simp
  thus x t = mi.solution t using t
    by auto
qed
let ?J = J ∪ ?mJ'
show ?thesis
proof
  have t0-in: t0 ∈ J ∩ op − (2 * t0) ` J'
    using ‹t0 ∈ J› J'(5)
```

```
    by auto
  from t0-in have t0 < M′ t0 < M
    by (auto simp: J(1) J′(1))
  have J ∪ ?mJ′ =
   real-of-ereal ‘ {ereal t0..<M} ∪ op − (2 ∗ t0) ‘ real-of-ereal ‘ {ereal t0..<M′}
    unfolding J(1) J′(1) split image-Un
    by simp
  also
  {
    have {ereal t0..<M} = {ereal t0} ∪ {ereal t0 <..< M}
      using ‹t0 ∈ J› J′(5) J(1) by auto
   also have real-of-ereal ‘ . . . = (if M = ∞ then {t0 ..} else {t0 ..<real-of-ereal
M})
      using ‹t0 < M›
      by (cases M) (auto simp add: real-atLeastGreaterThan-eq)
    finally
  have real-of-ereal ‘ {ereal t0..<M} = (if M = ∞ then {t0..} else {t0..<real-of-ereal
M})
      by (simp add: J)
  } note right-ivl = this
  also
  {
    have {ereal t0..<M′} = {ereal t0} ∪ {ereal t0<..<M′}
      using J′(1, 5) by auto
   also have real-of-ereal ‘ . . . = (if M′ = ∞ then {t0 ..} else {t0 ..<real-of-ereal
M′})
      using ‹t0 < M′›
      by (cases M′) (auto simp add: real-atLeastGreaterThan-eq)
    also have op − (2 ∗ t0) ‘ . . . =
      (if M′ = ∞ then {.. t0} else {2 ∗ t0 − real-of-ereal M′ <.. t0})
      by simp
    finally have op − (2 ∗ t0) ‘ real-of-ereal ‘ {ereal t0..<M′} =
      (if M′ = ∞ then {..t0} else {2 ∗ t0 − real-of-ereal M′<..t0})
      .
  } note left-ivl = this
  also have
    (if M = ∞ then {t0..} else {t0..<real-of-ereal M}) ∪
     (if M′ = ∞ then {..t0} else {2 ∗ t0 − real-of-ereal M′<..t0}) =
      real-of-ereal ‘ {2 ∗ t0 − M′ <..< M}
    using ‹t0 < M› ‹t0 < M′›
    by (cases M; cases M′) (auto simp add: real-atLeastGreaterThan-eq)
  finally show ivl: J ∪ ?mJ′ = real-of-ereal ‘ {2 ∗ t0 − M′ <..< M} .
  show maximal-existence-interval (J ∪ op − (2 ∗ t0) ‘ J′)
    unfolding maximal-existence-interval-def
  proof (intro conjI allI impI)
    show ?J ⊆ T t0 ∈ ?J
      using J(3,5) J′(3,5) by auto
    show is-interval (J ∪ op − (2 ∗ t0) ‘ J′)
      using J(4) J′(4) t0-in
```

by (*auto intro*!: *is-real-interval-union*)
show *open* ($J \cup$ *?mJ'*)
unfolding *ivl*
by (*auto intro*!: *open-real-image*)
interpret *pi*: *unique-solution i*(|*ivp-T*:=*J*|)
by *fact*
have *t0-less-M*: $M \neq \infty \Longrightarrow$ *t0* < *real-of-ereal M*
using *J*(*1*) ‹*t0* ∈ *J*› *right-ivl*
by *auto*
have *closure* (*real-of-ereal* ‘ {*ereal t0*..<*M*}) = (*if M* = ∞ *then* {*t0*..} *else* {*t0* .. *real-of-ereal M*})
by (*simp add*: *t0-less-M right-ivl*)
moreover
have *t0* ∈ *J'* **using** *J'* **by** *auto*
have ∗: *?mJ'* = (*if M'* = ∞ *then* {..*t0*} *else* {*2* ∗ *t0* − *real-of-ereal M'*<..*t0*})
by (*simp add*: *J' left-ivl*)
have *M'* ≠ ∞ ⟹ *2* ∗ *t0* − *real-of-ereal M'* < *t0*
using *J'*(*1*) ‹*t0* ∈ *J'*› ‹*t0* < *M'*›
by (*cases M'*; *simp*)
hence *closure ?mJ'* = (*if M'* = ∞ *then* {..*t0*} *else* {*2* ∗ *t0* − *real-of-ereal M'*..*t0*})
by (*simp add*: ∗)
ultimately have *clos*: ⋀*x*. *x* ∈ *closure J* ⟹ *x* ∈ *closure ?mJ'* ⟹ *x* = *t0*
unfolding *J*(*1*) **by** (*auto simp*: *split-ifs*)
have *JJ'*: ⋀*x*. *2* ∗ *t0* − *x* ∈ *J* ⟹ *x* ∈ *J'* ⟹ *x* = *t0*
using *J*(*1*) *J'*(*1*)
apply (*auto simp*: *algebra-simps*)
apply (*rename-tac x y*)
apply (*case-tac x*; *case-tac y*; *simp*)
done
interpret *glob*: *connected-unique-solutions i*(|*ivp-T* := *J* ∪ *?mJ'*|) *i*(|*ivp-T*:=*J*|) *?mi t0*
using ‹*t0* ∈ *J*› ‹*ivp-t0 ?ni* ∈ *J'*› *pi*.*is-solutionD*[*OF pi*.*is-solution-solution*] *pi*.*iv-defined*
mi.*is-solutionD*[*OF mi*.*is-solution-solution*]
by *unfold-locales* (*auto simp*: *dest*!: *clos JJ'*)
show *unique-solution* (*i*(|*ivp-T* := *J* ∪ *?mJ'*|))
by *unfold-locales*
fix *K x*
assume *K*: *K* ⊆ *T is-interval K t0* ∈ *K*
assume *K-sol*: *ivp*.*is-solution* (*i*(|*ivp-T* := *K*|)) *x*
have *mJ'*: *is-interval ?mJ' t0* ∈ *?mJ'*
using *t0-in*
by (*auto simp add*: *J'*(*4*))
from *K* **have** *Kp*: *K* ∩ {*t0*..} ⊆ *T* *is-interval* (*K* ∩ {*t0*..})
t0 ∈ (*K* ∩ {*t0*..}) ⋀*x*. *x* ∈ *K* ∩ {*t0*..} ⟹ *t0* ≤ *x*
by (*auto simp*: *is-interval-ci is-interval-ic intro*!: *is-interval-inter J*)
have *ivp* (*i*(|*ivp-T* := *K*|))
by *unfold-locales* (*auto simp*: *K iv-defined*)

then have *ivp.is-solution* $(i(\!|ivp\text{-}T := K,\ ivp\text{-}T := K \cap \{t0..\}|\!))\ x$
  by (*rule ivp.solution-on-subset*) (*auto intro*!: *K-sol K J*)
hence *Kp-sol*: *ivp.is-solution* $(i(\!|ivp\text{-}T := K \cap \{t0..\}|\!))\ x$
  by *simp*
from $J(7)[OF\ Kp\ Kp\text{-}sol]$
have *Kp-subset-unique*:
  $K \cap \{t0..\} \subseteq J$
  $(\forall t{\in}K \cap \{t0..\}.\ x\ t = ivp.solution\ (i(\!|ivp\text{-}T := J|\!))\ t)$
  by *auto*

let $?mKp = ?mirror\ `\ K \cap \{t0..\}$
have *Km*: $?mKp \subseteq ?mirror\ `\ T$ *is-interval* $(?mirror\ `\ K \cap \{t0..\})$
  $t0 \in ?mKp\ \bigwedge x.\ x \in ?mKp \Longrightarrow t0 \le x$
  using *K*
  by (*auto simp*: *is-interval-ci*
    intro!: *is-interval-inter K*)
let $?mKi = i(\!|ivp\text{-}f := \lambda(t, x).\ - f\ (2 * t0 - t, x),\ ivp\text{-}T := op - (2 * t0)$
$`\ K \cap \{t0..\}|\!)$
interpret *mKi*: *ivp ?mKi*
  using *K* **by** *unfold-locales* (*auto simp*: *iv-defined*)
interpret *Ki*: *ivp* $i(\!|ivp\text{-}T := K|\!)$
  by *unfold-locales* (*auto simp*: *K iv-defined*)
from *Ki.mirror-solution*[*OF K-sol*]
have $**$:
  ivp.is-solution
    $(i(\!|ivp\text{-}f := \lambda(t, x).\ - f\ (?mirror\ t, x),\ ivp\text{-}T := ?mirror\ `\ K|\!))$
    $(x \circ ?mirror)$
  by *simp*
have *ivp* $(i(\!|ivp\text{-}f := \lambda(t, x).\ - f\ (2 * t0 - t, x),\ ivp\text{-}T := op - (2 * t0)\ `$
$K|\!))$
  using $K\ **$
  by *unfold-locales* (*auto simp*: *iv-defined*)
then have
  ivp.is-solution
    $(i(\!|ivp\text{-}f := \lambda(t, x).\ - f\ (?mirror\ t, x),\ ivp\text{-}T := ?mirror\ `\ K,\ ivp\text{-}T :=$
$?mirror\ `\ K \cap \{t0..\}|\!))$
    $(x \circ ?mirror)$
  apply (*rule ivp.solution-on-subset*)
  using $K\ **$
  by *auto*
hence *mKi.is-solution* $(x\ o\ ?mirror)$
  by *simp*
from $J'(7)[simplified,\ OF\ Km\ this]$
have *Km-unique'*: $op - (2 * t0)\ `\ K \cap \{t0..\} \subseteq J'$
  $(\forall t{\in}op - (2 * t0)\ `\ K \cap \{t0..\}.$
    $(x \circ op - (2 * t0))\ t =$
      $ivp.solution\ (i(\!|ivp\text{-}f := \lambda(t, x).\ - f\ (2 * t0 - t, x),\ ivp\text{-}T := J'|\!))\ t)$
  by *auto*
hence *Km-subset*: $K \cap \{..t0\} \subseteq ?mJ'$

by (*auto simp*: *J'* *intro*!: *image-eqI*[**where** $x=2 * t0 - x$ **for** *x*])
have *Km-unique*: ($\forall t \in K \cap \{..t0\}$. $x\ t = ivp.solution\ (i(|ivp\text{-}T := ?mJ'|))\ t$)
proof *safe*
fix *t* **assume** $t \in K$ **assume** $t \leq t0$
{
fix *t'* **assume** *t'*: $t' \in ?mirror$ ‘ $K \cap \{t0..\}$
hence (*x o ?mirror*) *t'* =
ivp.solution
$(i(|ivp\text{-}f := \lambda(t, x).\ - f\ (2 * t0 - t, x),\ ivp\text{-}T := J'|))$
t'
using *Km-unique'* **by** *auto*
moreover
have *mmid*: $\bigwedge X.\ ?mirror$ ‘ ($?mirror$ ‘ $X \cap \{t0..\}) = X \cap \{..t0\}$
by *force*
have *ivp.is-solution* $(i(|ivp\text{-}T := K \cap \{..t0\}|))$ *mi.solution*
by (*rule mi.solution-on-subset'*) (*auto intro*!: *K Km-subset*)
then have *ivp.is-solution ?mKi*
(*ivp.solution* $(i(|ivp\text{-}T := op - (2 * t0)$ ‘ $J'|)) \circ op - (2 * t0))$
by (*intro mKi.solution-mirror*) (*auto simp*: *o-def mmid*)
from *J'*(*7*)[*simplified*, *OF Km this*] *t'*
have (*ivp.solution* $(i(|ivp\text{-}T := ?mJ'|))$ *o ?mirror*) *t'* =
ivp.solution $(i(|ivp\text{-}f := \lambda(t, x).\ - f\ (2 * t0 - t, x),\ ivp\text{-}T := J'|))$ *t'*
by *auto*
ultimately
have (*x o ?mirror*) *t'* = (*ivp.solution* $(i(|ivp\text{-}T := ?mJ'|))$ *o ?mirror*) *t'*
by *simp*
}
with ‹$t \in K$› ‹$t \leq t0$›
show $x\ t = ivp.solution\ (i(|ivp\text{-}T := op - (2 * t0)$ ‘ $J'|))\ t$ **by** *force*
qed
have $\{t0..\} \cup \{..t0\} = UNIV$ **by** *auto*
with *Kp-subset-unique Km-subset* **have** *K-subset*: $K \subseteq J \cup op - (2 * t0)$ ‘
J'
by *auto*
moreover
have ($\forall t \in K.\ x\ t = ivp.solution\ (i(|ivp\text{-}T := J \cup op - (2 * t0)$ ‘ $J'|))\ t$)
proof *safe*
fix *t*
assume $t \in K$
{
assume $t \in J$
with ‹$t \in K$›
have $x\ t = ivp.solution\ (i(|ivp\text{-}T := J|))\ t$
by (*metis Int-Collect J*(*2*) *Kp-subset-unique*(*2*) *atLeast-def*)
} moreover {
assume $t \notin J$
with ‹$t \in K$› *K-subset* **have** $x\ t = ivp.solution\ (i(|ivp\text{-}T := op - (2 * t0)$
‘ $J'|))\ t$
by (*intro Km-unique*[*rule-format*])

(*auto simp*: *glob.connection-def* ∗ *split*: *if-split-asm*)
} **ultimately**
show *x t* = *ivp.solution* (*i*(|*ivp-T* := *J* ∪ *op* − (*2* ∗ *t0*) ' *J*′|)) *t*
using ‹*t* ∈ *K*› *K-subset*
by (*subst glob.connection-eq-solution*[*symmetric*])
(*auto simp add*: *glob.connection-def*)
qed
ultimately show *K* ⊆ *J* ∪ *?mJ*′ (∀ *t*∈*K*. *x t* = *ivp.solution* (*i*(|*ivp-T* := *J* ∪ *?mJ*′|)) *t*)
by *auto*
qed
qed
qed

end

end

4 Sequence of Properties on Subsequences

theory *Diagonal-Subsequence*
imports *Complex-Main*
begin

locale *subseqs* =
fixes *P*::*nat*⇒(*nat*⇒*nat*)⇒*bool*
assumes *ex-subseq*: ⋀*n s*. *subseq s* ⟹ ∃ *r*′. *subseq r*′ ∧ *P n* (*s o r*′)
begin

definition *reduce* **where** *reduce s n* = (*SOME r*′. *subseq r*′ ∧ *P n* (*s o r*′))

lemma *subseq-reduce*[*intro*, *simp*]:
subseq s ⟹ *subseq* (*reduce s n*)
unfolding *reduce-def* **by** (*rule someI2-ex*[*OF ex-subseq*]) *auto*

lemma *reduce-holds*:
subseq s ⟹ *P n* (*s o reduce s n*)
unfolding *reduce-def* **by** (*rule someI2-ex*[*OF ex-subseq*]) (*auto simp*: *o-def*)

primrec *seqseq* **where**
seqseq 0 = *id*
| *seqseq* (*Suc n*) = *seqseq n o reduce* (*seqseq n*) *n*

lemma *subseq-seqseq*[*intro*, *simp*]: *subseq* (*seqseq n*)
proof (*induct n*)
case *0* **thus** *?case* **by** (*simp add*: *subseq-def*)
next
case (*Suc n*) **thus** *?case* **by** (*subst seqseq.simps*) (*auto intro*!: *subseq-o*)
qed

lemma *seqseq-holds*:
  P n (*seqseq* (*Suc n*))
proof −
  have *P n* (*seqseq n o reduce* (*seqseq n*) *n*)
    by (*intro reduce-holds subseq-seqseq*)
  thus *?thesis* **by** *simp*
qed

definition *diagseq* **where** *diagseq i* = *seqseq i i*

lemma *subseq-mono*: *subseq f* ⟹ *a* ≤ *b* ⟹ *f a* ≤ *f b*
  by (*metis le-eq-less-or-eq subseq-mono*)

lemma *subseq-strict-mono*: *subseq f* ⟹ *a* < *b* ⟹ *f a* < *f b*
  by (*simp add*: *subseq-def*)

lemma *diagseq-mono*: *diagseq n* < *diagseq* (*Suc n*)
proof −
  have *diagseq n* < *seqseq n* (*Suc n*)
    using *subseq-seqseq*[*of n*] **by** (*simp add*: *diagseq-def subseq-def*)
  also have ... ≤ *seqseq n* (*reduce* (*seqseq n*) *n* (*Suc n*))
    by (*auto intro*: *subseq-mono seq-suble*)
  also have ... = *diagseq* (*Suc n*) **by** (*simp add*: *diagseq-def*)
  finally show *?thesis* **.**
qed

lemma *subseq-diagseq*: *subseq diagseq*
  using *diagseq-mono* **by** (*simp add*: *subseq-Suc-iff diagseq-def*)

primrec *fold-reduce* **where**
  fold-reduce n 0 = *id*
| *fold-reduce n* (*Suc k*) = *fold-reduce n k o reduce* (*seqseq* (*n* + *k*)) (*n* + *k*)

lemma *subseq-fold-reduce*[*intro*, *simp*]: *subseq* (*fold-reduce n k*)
proof (*induct k*)
  case (*Suc k*) **from** *subseq-o*[*OF this subseq-reduce*] **show** *?case* **by** (*simp add*: *o-def*)
qed (*simp add*: *subseq-def*)

lemma *ex-subseq-reduce-index*: *seqseq* (*n* + *k*) = *seqseq n o fold-reduce n k*
  by (*induct k*) *simp-all*

lemma *seqseq-fold-reduce*: *seqseq n* = *fold-reduce 0 n*
  by (*induct n*) (*simp-all*)

lemma *diagseq-fold-reduce*: *diagseq n* = *fold-reduce 0 n n*
  using *seqseq-fold-reduce* **by** (*simp add*: *diagseq-def*)

```
lemma fold-reduce-add: fold-reduce 0 (m + n) = fold-reduce 0 m o fold-reduce m
n
  by (induct n) simp-all

lemma diagseq-add: diagseq (k + n) = (seqseq k o (fold-reduce k n)) (k + n)
proof -
  have diagseq (k + n) = fold-reduce 0 (k + n) (k + n)
    by (simp add: diagseq-fold-reduce)
  also have ... = (seqseq k o fold-reduce k n) (k + n)
    unfolding fold-reduce-add seqseq-fold-reduce ..
  finally show ?thesis .
qed

lemma diagseq-sub:
  assumes m ≤ n shows diagseq n = (seqseq m o (fold-reduce m (n - m))) n
  using diagseq-add[of m n - m] assms by simp

lemma subseq-diagonal-rest: subseq (λx. fold-reduce k x (k + x))
  unfolding subseq-Suc-iff fold-reduce.simps o-def
proof
  fix n
  have fold-reduce k n (k + n) < fold-reduce k n (k + Suc n) (is ?lhs < -)
    by (auto intro: subseq-strict-mono)
  also have ... ≤ fold-reduce k n (reduce (seqseq (k + n)) (k + n) (k + Suc n))
    by (rule subseq-mono) (auto intro!: seq-suble subseq-mono)
  finally show ?lhs < ... .
qed

lemma diagseq-seqseq: diagseq o (op + k) = (seqseq k o (λx. fold-reduce k x (k +
x)))
  by (auto simp: o-def diagseq-add)

lemma diagseq-holds:
  assumes subseq-stable: ⋀r s n. subseq r ⟹ P n s ⟹ P n (s o r)
  shows P k (diagseq o (op + (Suc k)))
 unfolding diagseq-seqseq by (intro subseq-stable subseq-diagonal-rest seqseq-holds)

end

end
```

5 Bounded Linear Operator

```
theory Bounded-Linear-Operator
imports
  ~~/src/HOL/Multivariate-Analysis/Multivariate-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 op* $
unfolding *bounded-bilinear-def*
by *transfer* (*simp add*: *blinfun.bilinear-simps blinfun.bounded*)

interpretation *blinop*: *bounded-bilinear op* \$
  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** *auto*
  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

6 Multivariate Taylor

theory *Multivariate-Taylor*
imports
~~/src/HOL/Multivariate-Analysis/Multivariate-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* ⇒ ′*a list* ⇒ ′*b*
assumes *n* > *0*
assumes *Df-Nil*: ⋀*a*. *Df a* [] = *f a*
assumes *Df-Cons*: ⋀*a ds*. *a* ∈ *closed-segment X* (*X* + *H*) ⟹ *length ds* < *n* ⟹
((λ*a*. *Df a ds*) *has-derivative* (λ*d*. *Df a* (*d*#*ds*))) (*at a*)
defines *i* ≡ λ*x*.
((*1* − *x*) ˆ (*n* − *1*) / *fact* (*n* − *1*)) $*_R$ *Df* (*X* + *x* $*_R$ *H*) (*replicate n H*)
shows *multivariate-taylor-has-integral*:
(*i has-integral f* (*X* + *H*) − (∑*i*<*n*. (*1* / *fact i*) $*_R$ *Df X* (*replicate i H*))) {*0*..*1*}
and *multivariate-taylor*:
f (*X* + *H*) = (∑*i*<*n*. (*1* / *fact i*) $*_R$ *Df X* (*replicate i 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*)
def *line* ≡ (λ*t*. *X* + *t* $*_R$ *H*)
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* (λ*t*. *t* $*_R$ *H*)) (*at x*)
by (*auto intro*!: *derivative-eq-intros simp*: *line-def*)
def *g* ≡ *f o line*
def *Dg* ≡ λ(*n*::*nat*) (*t*::*real*). *Df* (*line t*) (*replicate n H*)
note ‹*n* > *0*›
moreover
have *Dg0*: *Dg 0* = *g* **by** (*auto simp add*: *Dg-def Df-Nil g-def*)
moreover
{
fix *m*::*nat* **and** *t*::*real*
assume *m* < *n 0* ≤ *t t* ≤ *1*
hence [*intro*]: *line t* ∈ *?G* **using** *assms*
by (*auto simp*: *segment-eq*)
note [*derivative-intros*] = *has-derivative-compose*[*OF* - *Df-Cons*]
interpret *Df*: *linear* (λ*d*. *Df* (*line t*) (*d*#*replicate m H*))
by (*auto intro*!: *has-derivative-linear derivative-intros* ‹*m* < *n*›)
note [*derivative-intros*] =
has-derivative-compose[*OF* - *line-deriv*]
have (*Dg m has-vector-derivative Dg* (*Suc m*) *t*) (*at t within* {*0..1*})
using *Df.scaleR* ‹*m* < *n*›
by (*auto simp*: *Dg-def has-vector-derivative-def g-def*
intro!: *derivative-eq-intros*)
} **note** *DgSuc* = *this*
ultimately
have *g-taylor*: (*i has-integral g 1* − (∑ *i*<*n*. ((*1* − *0*) ˆ *i* / *fact i*) $*_R$ *Dg i 0*))
{*0* .. *1*}
unfolding *i-def Dg-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 integral-unique* **by** *force*
case *3* **show** *?case* **using** *c* **by** *force*
qed

in particular...

lemma
multivariate-taylor2:
fixes *f*::'*a*::*real-normed-vector* ⇒ '*b*::*banach*
assumes *f* '[*derivative-intros*]:
⋀*y*. *y* ∈ *closed-segment a x* ⟹ (*f has-derivative op* $ (*f* ' *y*)) (*at y*)
assumes *f* ''[*derivative-intros*]:
⋀*y*. *y* ∈ *closed-segment a x* ⟹ (*f* ' *has-derivative op* $ (*f* '' *y*)) (*at y*)

shows $((\lambda xa.\ (1 - xa) *_R f''\ (a + xa *_R (x - a))\ (x - a)\ (x - a))$ *has-integral* $f\ x - f\ a - f'\ a\ (x - a))\ \{0\ ..\ 1\}$
proof –
let *?G* = *closed-segment a x*
def $Df \equiv \lambda x\ ds.$ *case ds of* $[] \Rightarrow f\ x$
| $[d] \Rightarrow f'\ x\ d$
| $[d1, d2] \Rightarrow f''\ x\ d1\ d2$
have *Df-Nil*: $\bigwedge a.\ Df\ a\ [] = f\ a$
by (*auto simp*: *Df-def*)
{
fix *a*::*'a* **and** *ds*::*'a list*
assume $a \in$ *?G length ds* < *2*
hence $((\lambda a.\ Df\ a\ ds)$ *has-derivative* $(\lambda d.\ Df\ a\ (d\ \#\ ds)))$ (*at a*)
by (*cases ds*)
(*auto simp add*: *Df-def assms blinfun.zero-right*
intro!: *derivative-eq-intros*)
} note *Df-Cons* = *this*
from *multivariate-taylor-has-integral*[*of 2 Df f a x − a*, *OF - Df-Nil Df-Cons*]
show *?thesis*
by (*simp add*: *assms numeral-eq-Suc Df-def algebra-simps*)
qed

lemma
multivariate-taylor3:
fixes *f*::*'a*::*real-normed-vector* ⇒ *'b*::*banach*
assumes *f'*[*derivative-intros*]:
$\bigwedge y.\ y \in$ *closed-segment a x* $\Longrightarrow$ (*f has-derivative op* \$ $(f'\ y))$ (*at y*)
assumes *f''*[*derivative-intros*]:
$\bigwedge y.\ y \in$ *closed-segment a x* $\Longrightarrow$ (*f' has-derivative op* \$ $(f''\ y))$ (*at y*)
assumes *f'''*[*derivative-intros*]:
$\bigwedge y.\ y \in$ *closed-segment a x* $\Longrightarrow$ (*f'' has-derivative op* \$ $(f'''\ y))$ (*at y*)
shows
$((\lambda xa.\ ((1 - xa)^2/2) *_R f'''\ (a + xa *_R (x - a))\ (x - a)\ (x - a)\ (x - a))$
has-integral
$f\ x - f\ a - f'\ a\ (x - a) - f''\ a\ (x - a)\ (x - a)\ /_R\ 2)\ \{0..1\}$
proof –
let *?G* = *closed-segment a x*
def $Df \equiv \lambda x\ ds.$ *case ds of* $[] \Rightarrow f\ x$
| $[d] \Rightarrow f'\ x\ d$
| $[d1, d2] \Rightarrow f''\ x\ d1\ d2$
| $[d1, d2, d3] \Rightarrow f'''\ x\ d1\ d2\ d3$
have *Df-Nil*: $\bigwedge a.\ Df\ a\ [] = f\ a$
by (*auto simp*: *Df-def*)
{
fix *a*::*'a* **and** *ds*::*'a list*
assume $a \in$ *?G length ds* < *3*
then consider $ds = []\ |\ \exists\ d1.\ ds = [d1]\ |\ \exists\ d1\ d2.\ ds = [d1, d2]$
apply (*cases ds*)
subgoal by *simp*

subgoal for *d ds* **by** (*cases ds*) *auto*
done
then have ((λ*a*. *Df a ds*) *has-derivative* (λ*d*. *Df a* (*d* # *ds*))) (*at a*)
apply *cases*
using ‹*a* ∈ *?G*›
by (*auto simp add*: *Df-def assms blinfun.zero-right*
intro!: *derivative-eq-intros*)
} **note** *Df-Cons* = *this*
from *multivariate-taylor-has-integral*[*of 3 Df f a x* − *a*, *OF* - *Df-Nil Df-Cons*]
show *?thesis*
by (*simp add*: *assms numeral-eq-Suc Df-def algebra-simps*)
qed

6.1 Symmetric second derivative

lemma *symmetric-second-derivative-aux*:
assumes *first-fderiv*[*derivative-intros*]:
$\bigwedge a.\ a \in G \Longrightarrow (f\ \textit{has-derivative}\ (f'\ a))\ (\textit{at}\ a\ \textit{within}\ G)$
assumes *second-fderiv*[*derivative-intros*]:
$\bigwedge i.\ ((\lambda x.\ f'\ x\ i)\ \textit{has-derivative}\ (\lambda j.\ f''\ j\ i))\ (\textit{at}\ a\ \textit{within}\ G)$
assumes $i \neq j\ i \neq 0\ j \neq 0$
assumes $a \in G$
assumes $\bigwedge s\ t.\ s \in \{0..1\} \Longrightarrow t \in \{0..1\} \Longrightarrow a + s *_R i + t *_R j \in G$
shows $f''\ j\ i = f''\ i\ j$
proof −
let $?F = \textit{at-right}\ (0::\textit{real})$
def $B \equiv \lambda i\ j.\ \{a + s *_R i + t *_R j\ |s\ t.\ s \in \{0..1\} \wedge t \in \{0..1\}\}$
have $B\ i\ j \subseteq G$ **using** *assms* **by** (*auto simp*: *B-def*)
{
fix *e*::*real* **and** *i j*::'*a*
assume $e > 0$
assume $i \neq j\ i \neq 0\ j \neq 0$
assume $B\ i\ j \subseteq G$
let $?ij' = \lambda s\ t.\ \lambda u.\ a + (s * u) *_R i + (t * u) *_R j$
let $?ij = \lambda t.\ \lambda u.\ a + (t * u) *_R i + u *_R j$
let $?i = \lambda t.\ \lambda u.\ a + (t * u) *_R i$
let $?g = \lambda u\ t.\ f\ (?ij\ t\ u) - f\ (?i\ t\ u)$
have *filter-ij'I*: $\bigwedge P.\ P\ a \Longrightarrow \textit{eventually}\ P\ (\textit{at}\ a\ \textit{within}\ G) \Longrightarrow$
$\textit{eventually}\ (\lambda x.\ \forall s{\in}\{0..1\}.\ \forall t{\in}\{0..1\}.\ P\ (?ij'\ s\ t\ x))\ ?F$
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** $\bigwedge x\ d2.\ x \in B\ i\ j \Longrightarrow x \neq a \Longrightarrow \textit{dist}\ x\ a < d \Longrightarrow P\ x$
by (*auto simp*: *eventually-at*)
with ‹*P a*› **have** *P*: $\bigwedge x\ d2.\ x \in B\ i\ j \Longrightarrow \textit{dist}\ x\ a < d \Longrightarrow P\ x$ **by** (*case-tac*

```
x = a) auto
      let ?d = min (min (d/norm i) (d/norm j) / 2) 1
      show eventually (λx. ∀s∈{0..1}. ∀t∈{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 ≠ 0› ‹j ≠ 0› by simp
        fix x s t :: real assume *: s ∈ {0..1} t ∈ {0..1} 0 < x dist x 0 < ?d
        show P (?ij' s t x)
        proof (rule P)
          have ⋀x y::real. x ∈ {0..1} ⟹ y ∈ {0..1} ⟹ x * y ∈ {0..1}
            by (auto intro!: order-trans[OF mult-left-le-one-le])
          hence s * x ∈ {0..1} t * x ∈ {0..1} using * by (auto simp: dist-norm)
          thus ?ij' s t x ∈ B i j by (auto simp: B-def)
          have norm (s *R x *R i + t *R x *R j) ≤ norm (s *R x *R i) + norm
(t *R x *R j)
            by (rule norm-triangle-ineq)
          also have ... < d / 2 + d / 2 using * ‹i ≠ 0› ‹j ≠ 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
    {
      fix P
      assume P a eventually P (at a within G)
      from filter-ij'I[OF this] have eventually (λx. ∀t∈{0..1}. P (?ij t x)) ?F
        by eventually-elim (force dest: bspec[where x=1])
    } note filter-ijI = this
    {
      fix P assume P a eventually P (at a within G)
      from filter-ij'I[OF this] have eventually (λx. ∀t∈{0..1}. P (?i t x)) ?F
        by eventually-elim force
    } note filter-iI = this
    {
      from second-fderiv[of i, simplified has-derivative-iff-norm, THEN conjunct2,
        THEN tendstoD, OF ‹0 < e›]
      have eventually (λx. norm (f' x i − f' a i − f'' (x − a) i) / norm (x − a)
≤ 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 (λij. ∀t∈{0..1}. norm (f' (?ij t ij) i − f' a i − f'' (?ij t ij −
a) i) /
          norm (?ij t ij − a) ≤ e) ?F
        eventually (λij. ∀t∈{0..1}. norm (f' (?i t ij) i − f' a i − f'' (?i t ij − a)
i) /
          norm (?i t ij − a) ≤ 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 $?g' = \lambda t.\ (\lambda ua.\ u *_R ua *_R (f'\ (?ij\ t\ u)\ i - (f'\ (?i\ t\ u)\ i)))$
{
fix *t*::*real* **assume** $t \in \{0..1\}$
with *elim* **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*
have ((*?g u*) *has-derivative* *?g′ t*) (*at t within* $\{0..1\}$)
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*)
} **note** $g' = this$
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 ... $\leq$ *norm ?g10* + *norm ?f'i*
by (*blast intro*: *order-trans add-mono norm-triangle-le*)
also
have *0* $\in$ {*0..1*::*real*} **by** *simp*
have $\forall$ *t* $\in$ {*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*)))
$\leq$ *2* $*$ *u* $*$ *u* $*$ *e* $*$ (*norm i* + *norm j*) (**is** $\forall$ *t* $\in$ -. *onorm* (*?d t*) $\leq$ -)
proof
fix *t*::*real* **assume** *t* $\in$ {*0..1*}
show *onorm* (*?d t*) $\leq$ *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* $\in$ *G*›
by (*simp add*: *algebra-simps*
linear-add[*OF has-derivative-linear*[*OF second-fderiv*]])
also have ... $\leq$ *abs* (*u* $*$ *x*) $*$ (*norm ?dij* + *norm ?di* + *norm ?dj*)
by (*rule mult-left-mono*[*OF* - *abs-ge-zero*]) *norm*
also have ... $\leq$ *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* $\in$ {*0..1*}›] ‹*t* $\in$ {*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 ... $\leq$ *u* $*$ *u* $*$ *e* $*$ *abs x* $*$ ((*norm* (*1* $*_R$ *i*) + *norm j*) + *norm* (*1* $*_R$ *i*) + *norm j*)

using ‹$t \in \{0..1\}$› ‹$0 < e$›
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* $(?d\ t\ x) \leq 2 * u * u * e * (norm\ i + norm\ j) * norm\ x$
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* $?g10 \leq 2 * u * u * e * (norm\ i + norm\ j)$ **by** *simp*
also have *norm* $?f'i \leq abs\ u *$
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 $\ldots \leq abs\ u * (e * norm\ ((u) *_R j))$
using *f'ij*[*OF* ‹$0 \in \{0..1\}$›] **by** (*auto intro*: *mult-left-mono*)
also have $\ldots = u * u * e * norm\ j$ **by** (*simp add*: *algebra-simps abs-mult*)
finally show *?case* **by** (*simp add*: *algebra-simps*)
qed
}
} **note** *wlog = this*
{
fix *e t*::*real*
assume $0 < e$
have $B\ i\ j = B\ j\ i$ **using** ‹$i \neq j$› **by** (*force simp*: *B-def*)+
with *assms* ‹$B\ i\ j \subseteq G$› **have** $j \neq i$ $B\ j\ i \subseteq G$ **by** (*auto simp*:)
from *wlog*[*OF* ‹$0 < e$› ‹$i \neq j$› ‹$i \neq 0$› ‹$j \neq 0$› ‹$B\ i\ j \subseteq G$›]
wlog[*OF* ‹$0 < e$› ‹$j \neq i$› ‹$j \neq 0$› ‹$i \neq 0$› ‹$B\ j\ i \subseteq G$›]
have *eventually* $(\lambda u.\ norm\ ((u * u) *_R f''\ j\ i - (u * u) *_R f''\ i\ j)$
$\leq u * u * e * (5 * norm\ j + 5 * norm\ i))$ *?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 $\ldots \leq u * u * e * (2 * norm\ j + 3 * norm\ i) + u * u * e * (3 * norm\ j + 2 * norm\ i)$
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* $(\lambda u.\ norm\ ((u * u) *_R (f''\ j\ i - f''\ i\ j)) \leq$
$u * u * e * (5 * norm\ j + 5 * norm\ i))$ *?F*
by (*simp add*: *algebra-simps*)

hence *eventually* (λ*u*. (*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* (λ*u*. *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*
hence *norm* (*f''* *j* *i* − *f''* *i* *j*) ≤ *e* ∗ (*5* ∗ *norm* *j* + *5* ∗ *norm* *i*)
by (*auto simp add*:*eventually-at dist-norm dest*!: *bspec*[**where** *x*=*d*/*2* **for** *d*])
} **note** *e'* = *this*
{
fix *e*::*real* **assume** *0* < *e*
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 have *norm* (*f''* *j* *i* − *f''* *i* *j*) < *e* **.**
} **note** *e* = *this*
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* $\Rightarrow$ $'b$::*banach*
    and f' :: $'a \Rightarrow 'a \Rightarrow_L 'b$
    and f'' :: $'a \Rightarrow_L 'a \Rightarrow_L 'b$
    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*: $a \in$ *interior* G
  assumes *second-fderiv*[*derivative-intros*]:
    (f' *has-derivative* f'') (*at* a)
begin

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

  def 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 ∈ J z ∈ 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* ‹$0 < \eta$›
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*: *integral-diff integral-linear*[**where** h=λy. *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 continuous-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-within*])
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* × *G*, *simplified*]
have *continuous-on* (*G* × *G*) (λ(*a*, *b*). *f b* − *f a*)
by (*rule iffD1*[*OF continuous-on-cong continuous-on-diff2*[*OF f-cont f-cont*]], *auto*)
moreover have *continuous-on* (*G* × *G*) (λ(*a*, *b*). *f′ a* (*b* − *a*))
by (*auto intro*!: *continuous-intros simp*: *split-beta′*)
ultimately have *continuous-on* (*G* × *G*) (λ(*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

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

end

7 Flow

theory *Flow*
imports
Picard-Lindeloef-Qualitative
~~/src/HOL/Library/Diagonal-Subsequence
../Library/Bounded-Linear-Operator
../Library/Multivariate-Taylor
begin

7.1 simp rules for integrability (TODO: move)

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*[*of g b*]
by (*auto simp*: *cbox-sing*)

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

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

lemma *integral-const-closed-segment*:
fixes *a b*::*real*
shows *integral* (*closed-segment a b*) ($\lambda x.\ c$) = *abs* ($b - a$) $*_R$ *c*
by (*auto simp*: *closed-segment-real 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

lemmas [*integrable-on-simps*] =
integrable-0
integrable-neg
integrable-cmul
integrable-mult
integrable-on-cmult-left
integrable-on-cmult-right
integrable-on-cdivide
integrable-on-cmult-iff
integrable-on-cmult-left-iff
integrable-on-cmult-right-iff
integrable-on-cdivide-iff
integrable-diff
integrable-add
integrable-setsum

7.2 Nonautonomous IVP on maximal existence interval

locale *ll-on-open* =
fixes *f*::*real* ⇒ *'a*::{*banach*, *heine-borel*} ⇒ *'a* **and** *T X*
assumes *local-lipschitz*: *local-lipschitz T X f*
assumes *cont*: $\bigwedge x.\ x \in X \Longrightarrow$ *continuous-on T* ($\lambda t.\ f\ t\ x$)
assumes *open-domain*[*intro*!, *simp*]: *open T open X*
begin

lemma *continuous-on-Times-f*: *continuous-on* ($T \times X$) ($\lambda(t, x).\ f\ t\ x$)

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

lemma *continuous-on-f*[*continuous-intros*]:
assumes *continuous-on S g*
assumes *continuous-on S h*
assumes *h ' S ⊆ X*
assumes *g ' S ⊆ T*
shows *continuous-on S* (*λx. f* (*g x*) (*h x*))
using *assms*
by (*intro continuous-on-compose2*[*OF continuous-on-Times-f* , *of S λx.* (*g x*, *h x*), *simplified*])
(*auto intro*!: *continuous-intros*)

lemma
lipschitz-on-compact:
assumes *compact K K ⊆ T*
assumes *compact Y Y ⊆ X*
obtains *L* **where** $\bigwedge$*t. t ∈ K ⟹ lipschitz Y* (*f t*) *L*
proof −
have *cont*: $\bigwedge$*x. x ∈ Y ⟹ continuous-on K* (*λt. 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-on-subset*[*OF* - ‹*K ⊆ T*› ‹*Y ⊆ X*›])
from *local-lipschitz-on-compact-implies-lipschitz*[*OF this* ‹*compact Y*› ‹*compact K*› *cont*] *that*
show *?thesis* **by** *metis*
qed

lemma *ll-on-open-rev*[*intro*, *simp*]: *ll-on-open* (*λt.* − *f* (*2* ∗ *t0* − *t*)) ((*λt. 2* ∗ *t0* − *t*) ' *T*) *X*
using *local-lipschitz*
by *unfold-locales*
(*auto intro*!: *continuous-intros cont intro*: *local-lipschitz-compose1*
simp: *fun-Compl-def local-lipschitz-uminus local-lipschitz-on-subset open-neg-translation image-image*)

context fixes *t0*::*real* **and** *x0*::*'a* — initial value
begin

definition *outer-ivp* = (|
ivp-f = (*λ*(*t*, *x*). *f t x*),
ivp-t0 = *t0*,
ivp-x0 = *x0*,
ivp-T = *T*,
ivp-X = *X* |)

definition *maximal-existence-bounds* =

(*SOME* (*a*::*ereal*, *b*::*ereal*).
if unique-on-open outer-ivp then
unique-on-open.maximal-existence-interval outer-ivp (*real-of-ereal* ‘ {*a* <..< *b*})
else b < *a*)

definition *inf-existence* = *fst maximal-existence-bounds*

definition *sup-existence* = *snd maximal-existence-bounds*

definition *existence-ivl* = *real-of-ereal* ‘ {*inf-existence* <..< *sup-existence*}

definition *existence-ivp* = (|
ivp-f = (λ(*t*, *x*). *f t x*),
ivp-t0 = *t0*,
ivp-x0 = *x0*,
ivp-T = *existence-ivl*,
ivp-X = *X* |)

lemma *existence-ivp-simps*[*simp*]:
ivp-f existence-ivp = (λ(*t*, *x*). *f t x*)
ivp-t0 existence-ivp = *t0*
ivp-x0 existence-ivp = *x0*
ivp-T existence-ivp = *existence-ivl*
ivp-X existence-ivp = *X*
by (*simp-all add*: *existence-ivp-def*)

lemma *open-existence-ivl*[*simp*]: *open existence-ivl*
by (*simp add*: *existence-ivl-def open-real-image*)

lemma *is-interval-existence-ivl*[*simp*]: *is-interval existence-ivl*
by (*auto simp*: *existence-ivl-def is-interval-real-ereal-oo*)

definition *flow t* = *ivp.solution existence-ivp t*

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

interpretation *outer-ivp*: *ivp outer-ivp*
by *standard* (*auto simp*: *outer-ivp-def iv-in*)

interpretation *outer-ivp*: *ivp-open outer-ivp*
by *standard* (*auto simp*: *outer-ivp-def*)

interpretation *outer-ivp*: *continuous-rhs ivp-T outer-ivp ivp-X outer-ivp ivp-f outer-ivp*
by *standard*
(*auto simp*: *outer-ivp-def split-beta intro*!: *continuous-intros*)

interpretation *outer-ivp*: *unique-on-open outer-ivp*
using *local-lipschitz*
by *unfold-locales* (*simp add*: *outer-ivp-def*)

lemma *maximal-existence-bounds-def'*:
maximal-existence-bounds =
(*SOME* (*a*::*ereal*, *b*::*ereal*). *outer-ivp.maximal-existence-interval* (*real-of-ereal* ‘ {*a* <..< *b*}))
proof −
have *unique-on-open outer-ivp* **..**
thus *?thesis*
by (*simp add*: *maximal-existence-bounds-def*)
qed

lemma *maximal-existence-bounds*:
outer-ivp.maximal-existence-interval
(*real-of-ereal* ‘ {*fst* (*maximal-existence-bounds*)<..<*snd* (*maximal-existence-bounds*)})
proof −
obtain *a b*::*ereal* **where** *outer-ivp.maximal-existence-interval* (*real-of-ereal* ‘ {*a* <..< *b*})
by (*metis outer-ivp.maximal-existence-intervalE*)
hence ∃ *x*. *case x of* (*a*::*ereal*, *b*::*ereal*) ⇒
outer-ivp.maximal-existence-interval (*real-of-ereal* ‘ {*a* <..< *b*})
by (*auto intro*!: *exI*[**where** *x*=(*a*, *b*)])
from *someI-ex*[*OF this*]
show *?thesis*
by (*auto simp*: *maximal-existence-bounds-def'*)
qed

lemma *maximal-existence-interval*:
outer-ivp.maximal-existence-interval existence-ivl
by (*simp add*: *inf-existence-def sup-existence-def maximal-existence-bounds existence-ivl-def*)

lemma *existence-ivl-subset*:
existence-ivl ⊆ *T*
using *maximal-existence-interval*
unfolding *outer-ivp.maximal-existence-interval-def*
by (*auto simp*: *outer-ivp-def*)

lemma *mem-existence-ivl-subset*:
⋀*x*. *x* ∈ *existence-ivl* ⟹ *x* ∈ *T*
using *existence-ivl-subset* **by** *auto*

interpretation *existence-ivp*: *ivp existence-ivp*
using *maximal-existence-interval*[*unfolded outer-ivp.maximal-existence-interval-def*]
by *unfold-locales* (*auto simp*: *iv-in outer-ivp-def*)

lemma *existence-ivl-initial-time*[*intro*, *simp*]: *t0* ∈ *existence-ivl*
using *existence-ivp.iv-defined*
by (*auto simp*: *existence-ivp-def existence-ivl-def*)

lemma *existence-ivp*: *unique-solution* (*existence-ivp*)

using *maximal-existence-interval*[*unfolded outer-ivp.maximal-existence-interval-def*]
by (*simp add*: *outer-ivp-def existence-ivp-def*)

interpretation *existence-ivp*: *unique-solution existence-ivp*
by (*rule existence-ivp*)

interpretation *existence-ivp*: *unique-on-open existence-ivp*
proof *unfold-locales*
have (*existence-ivl* × *X*) ⊆ *ivp-T* (*outer-ivp*) × *ivp-X* (*outer-ivp*)
by (*auto simp*: *outer-ivp-def mem-existence-ivl-subset*)
from *continuous-on-subset*[*OF outer-ivp.continuous this*]
show *continuous-on* (*ivp-T* (*existence-ivp*) × *ivp-X* (*existence-ivp*)) (*ivp-f* (*existence-ivp*))
by (*simp add*: *outer-ivp-def*)
qed (*insert outer-ivp.local-lipschitz outer-ivp.openX*,
auto simp add: *outer-ivp-def local-lipschitz-on-subset existence-ivl-subset*)

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

lemma
local-unique-solutions:
obtains *t u L*
where
$\bigwedge x.\ x \in cball\ x0\ u \Longrightarrow$
unique-solution
(*existence-ivp* (|*ivp-x0* := *x*, *ivp-T* := *cball t0 t*, *ivp-X* := *cball x u*|))
$\bigwedge x.\ x \in cball\ x0\ u \Longrightarrow cball\ x\ u \subseteq X$
$\bigwedge t'.\ t' \in cball\ t0\ t \Longrightarrow lipschitz\ (cball\ x0\ (2 * u))\ (f\ t')\ L$
$cball\ t0\ t \subseteq T$
$cball\ x0\ (2 * u) \subseteq X$
$0 < t\ 0 < u$
proof −
from *existence-ivp.eventually-unique-solution*
obtain *B L t* **where** *t*: $0 < t$
and *ev*:
eventually
(λ*e*. $0 < e$ ∧
cball (*existence-ivp.t0*) ($t * e$) ⊆ *existence-ivp.T* ∧
cball (*existence-ivp.x0*) *e* ⊆ *existence-ivp.X* ∧
unique-on-cylinder (*existence-ivp* (|*ivp-T* := *cball existence-ivp.t0* ($t * e$),
ivp-X := *cball existence-ivp.x0 e*|)) ($t * e$) *e B L* (*cball existence-ivp.x0 e*))
(*at-right 0*) **.**
from *eventually-happens*[*OF ev*] **obtain** *e* **where** *e*:
$e > 0$
cball (*existence-ivp.t0*) ($t * e$) ⊆ *existence-ivp.T*
cball (*existence-ivp.x0*) *e* ⊆ *existence-ivp.X*
unique-on-cylinder (*existence-ivp* (|*ivp-T* := *cball existence-ivp.t0* ($t * e$),

```
      ivp-X := cball existence-ivp.x0 e⦈) (t * e) e B L (cball existence-ivp.x0 e)
    by auto
  then interpret cyl:
    unique-on-cylinder existence-ivp ⦇ivp-T := cball existence-ivp.t0 (t * e),
      ivp-X := cball existence-ivp.x0 e⦈ t * e e B L cball existence-ivp.x0 e
    by− assumption
  def e' ≡ e / 2
  have lips: ⋀t'. t' ∈ cball t0 (t * e') ⟹ lipschitz (cball x0 (2 * e')) (f t') L cball
x0 (2 * e') ⊆ X
    using cyl.global-lipschitz.lipschitz(1) e t
    by (auto simp add: e'-def dist-real-def dest!: double-nonneg-le)
  from e t have e'-pos: e' > 0 by (simp add: e'-def)
  with t have te-pos: t * e' > 0 by simp
  from e existence-ivl-subset have cball t0 (t * e') ⊆ T
    by (force simp: e'-def dest!: double-nonneg-le)
  moreover
  {
    fix x0'::'a
    assume x0': x0' ∈ cball x0 e'
    let ?i' = existence-ivp ⦇ivp-x0 := x0', ivp-T := cball t0 (t * e'),
      ivp-X := cball x0' e'⦈
    {
      fix b
      assume d: dist x0' b ≤ e'
      have dist x0 b ≤ dist x0 x0' + dist x0' b
        by (rule dist-triangle)
      also have ... ≤ e' + e'
        using x0' d by simp
      also have ... ≤ e by (simp add: e'-def)
      finally have dist x0 b ≤ e .
    } note triangle = this
    have subs1: cball t0 (t * e') ⊆ cball t0 (t * e)
      and subs2: cball x0' e' ⊆ cball x0 e
      and subs: cball t0 (t * e') × cball x0' e' ⊆ cball t0 (t * e) × cball x0 e
      using e'-pos x0'
      by (auto simp: e'-def triangle dest!: double-nonneg-le)

    interpret cyl': cylinder ?i' t * e' e'
      using e'-pos t
      by unfold-locales (auto simp: dist-real-def)
    interpret cyl': solution-in-cylinder ?i' t * e'  e' B
      using cyl.norm-f cyl.e-bounded cyl.continuous subs
      by unfold-locales (force simp: e'-def intro: continuous-on-subset)+
    interpret cyl': unique-on-cylinder ?i' t * e' e' B L (cball x0 e)
      using cyl.global-lipschitz.lipschitz(1)[simplified] t
        cyl.global-lipschitz.lipschitz e'-pos x0' subs subs1
      by unfold-locales (auto simp: triangle)
    have un: unique-solution ?i'
      by unfold-locales
```

from *subs2 e* **have** *subs*: *cball x0′ e′* ⊆ *X* **by** *simp*
note *un this*
} **ultimately show** *thesis* **using** *lips te-pos e′-pos*
by (*metis that*)
qed

lemma *in-existence-between-zeroI*:
t ∈ *existence-ivl* ⟹ *s* ∈ {*t* .. *t0*} ∪ {*t0* .. *t*} ⟹ *s* ∈ *existence-ivl*
using *existence-ivl-initial-time*[*simplified existence-ivl-def*]
by (*cases inf-existence*; *cases sup-existence*)
(*auto simp*: *existence-ivl-def real-atLeastGreaterThan-eq*)

lemma *ivl2-subset-existence-ivl*:
assumes *s* ∈ *existence-ivl t* ∈ *existence-ivl*
shows {*s* .. *t*} ⊆ *existence-ivl*
apply (*rule subsetI*)
subgoal for *x*
using *in-existence-between-zeroI*[*OF assms*(*1*), *of x*] *in-existence-between-zeroI*[*OF assms*(*2*), *of x*]
by (*force*)
done

lemma *flow-in-domain*: *t* ∈ *existence-ivl* ⟹ *flow t* ∈ *X*
using *existence-ivp.solution-in-D flow-def* **by** *auto*

lemma *maximal-existence-flow*:
assumes *ivp.is-solution i x*
assumes *i* = (| *ivp-f* = (λ(*t*, *x*). *f t x*), *ivp-t0* = *t0*, *ivp-x0* = *x0*, *ivp-T* = *K*, *ivp-X* = *X* |)
assumes *is-interval K*
assumes *t0* ∈ *K*
assumes *K* ⊆ *T*
shows *K* ⊆ *existence-ivl* ⋀*t*. *t* ∈ *K* ⟹ *flow t* = *x t*
proof −
from *assms* **have** *sol*: *ivp.is-solution* (|*ivp-f* = λ(*t*, *x*). *f t x*, *ivp-t0* = *t0*, *ivp-x0* = *x0*, *ivp-T* = *K*, *ivp-X* = *X*|) *x*
by *auto*
from *maximal-existence-interval*[*unfolded outer-ivp.maximal-existence-interval-def*]
have *m*: ⋀*K x*. *K* ⊆ *T* ⟹
is-interval K ⟹
t0 ∈ *K* ⟹
ivp.is-solution (| *ivp-f* = (λ(*t*, *x*). *f t x*), *ivp-t0* = *t0*, *ivp-x0* = *x0*, *ivp-T* = *K*, *ivp-X* = *X* |) *x* ⟹
K ⊆ *existence-ivl* ∧
(∀ *t*∈*K*. *x t* = *ivp.solution* (| *ivp-f* = (λ(*t*, *x*). *f t x*), *ivp-t0* = *t0*, *ivp-x0* = *x0*, *ivp-T* = *existence-ivl*, *ivp-X* = *X* |) *t*)
by (*auto simp*: *outer-ivp-def*)
have *K* ⊆ *T* **using** *assms existence-ivl-subset* **by** *auto*
from *m*[*OF this* ‹*is-interval K*› ‹*t0* ∈ *K*› *sol*]

show *K* ⊆ *existence-ivl* ⋀*t*. *t* ∈ *K* ⟹ *flow t* = *x t*
by (*auto simp add*: *outer-ivp-def flow-def existence-ivp-def*)
qed

lemma *maximal-existence-flowI*:
assumes ⋀*t*. *t* ∈ *K* ⟹ (*x has-vector-derivative f t* (*x t*)) (*at t within 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* ⋀*t*. *t* ∈ *K* ⟹ *flow t* = *x t*
proof −
have *sol*: *ivp.is-solution* (|*ivp-f* = λ(*t*, *x*). *f t x*, *ivp-t0* = *t0*, *ivp-x0* = *x0*, *ivp-T* = *K*, *ivp-X* = *X*|) *x*
apply (*rule ivp.is-solutionI*)
apply *unfold-locales*
using *assms iv-in*
by *auto*
from *maximal-existence-flow*[*OF sol refl K*]
show *K* ⊆ *existence-ivl* ⋀*t*. *t* ∈ *K* ⟹ *flow t* = *x t*
by *auto*
qed

lemma *Picard-iterate-mem-existence-ivlI*:
assumes *t0* ≤ *t* {*t0* .. *t*} ⊆ *T*
assumes *compact C x0* ∈ *C C* ⊆ *X*
assumes ⋀*y s*. *t0* ≤ *s* ⟹ *s* ≤ *t* ⟹ *y t0* = *x0* ⟹ *y* ∈ {*t0*..*s*} → *C* ⟹ *continuous-on* {*t0*..*s*} *y* ⟹
x0 + *integral* {*t0*..*s*} (λ*t*. *f t* (*y t*)) ∈ *C*
shows *t* ∈ *existence-ivl* ⋀*s*. *t0* ≤ *s* ⟹ *s* ≤ *t* ⟹ *flow s* ∈ *C*
proof −
let *?i* = (|*ivp-f* = λ(*t*, *x*). *f t x*, *ivp-t0* = *t0*, *ivp-x0* = *x0*, *ivp-T* = {*t0* .. *t*}, *ivp-X* = *C*|)
interpret *uc*: *ivp ?i*
using *assms iv-in*
by *unfold-locales auto*
from *lipschitz-on-compact*[*OF compact-Icc* ‹{*t0* .. *t*} ⊆ *T*› ‹*compact C*› ‹*C* ⊆ *X*›]
obtain *L* **where** *L*: ⋀*s*. *s* ∈ {*t0* .. *t*} ⟹ *lipschitz C* (*f s*) *L* **by** *metis*
interpret *uc*: *unique-on-closed ?i t L*
using *assms*
by *unfold-locales*
(*auto intro*!: *L compact-imp-closed* ‹*compact C*› *continuous-on-f continuous-intros*
simp: *split-beta*)
have {*t0* .. *t*} ⊆ *existence-ivl*
using *assms*
apply (*intro maximal-existence-flow*(*1*)[*OF uc.is-solution-on-superset-domain*[*OF uc.is-solution-solution*]])
apply (*auto simp*: *is-interval-closed-interval*)
done
thus *t* ∈ *existence-ivl*

using *assms* **by** *auto*
show *flow s* ∈ *C* **if** *t0* ≤ *s s* ≤ *t* **for** *s*
proof −
have *flow s* = *uc.solution s uc.solution s* ∈ *C*
using *uc.is-solutionD*[*OF uc.is-solution-solution*] *that assms*
by (*auto simp*: *is-interval-closed-interval intro*!: *maximal-existence-flowI* (*2*)[**where** *K*={*t0* .. *t*}])
thus *?thesis* **by** *simp*
qed
qed

lemma *unique-on-intersection*:
assumes *t* ∈ *ivp-T i* ∩ *ivp-T j*
assumes *has-solution i*
assumes *has-solution j*
assumes *ivp-X i* = *ivp-X* (*existence-ivp*)
assumes *ivp-X j* = *ivp-X* (*existence-ivp*)
assumes *ivp-f i* = *ivp-f* (*existence-ivp*)
assumes *ivp-f j* = *ivp-f* (*existence-ivp*)
assumes *ivp-T i* ⊆ *T*
assumes *ivp-T j* ⊆ *T*
assumes *is-interval* (*ivp-T i*)
assumes *is-interval* (*ivp-T j*)
assumes *ti* ∈ *ivp-T i* ∩ *ivp-T j*
assumes *ivp.solution i ti* = *x0*
assumes *ivp.solution j ti* = *x0*
shows *ivp.solution i t* = *ivp.solution j t*
proof −
interpret *i*: *has-solution i* **by** *fact*
let *?i* = *i*(|*ivp-t0* := *ti*, *ivp-x0* := *x0*|)
interpret *i'*: *ivp ?i*
apply *standard*
using ‹*ti* ∈ -› ‹*x0* ∈ -›
by (*auto simp*: ‹*i.X* = -›)
have *i'-sol*: *i'.is-solution i.solution*
apply (*rule i.shift-initial-value*)
using *assms*
apply *auto*
done
interpret *j*: *has-solution j* **by** *fact*
let *?j* = *j*(|*ivp-t0* := *ti*, *ivp-x0* := *x0*|)
interpret *j'*: *ivp ?j*
apply *standard*
using ‹*ti* ∈ -› ‹*x0* ∈ -›
by (*auto simp*: ‹*j.X* = -›)
have *j'-sol*: *j'.is-solution j.solution*
apply (*rule j.shift-initial-value*)
using *assms*
apply *auto*

done

have *ll-on-open.flow f T X ti x0 t* = *ivp.solution i t*
using *assms*
apply (*intro ll-on-open.maximal-existence-flow*[**where** *i*=*i*(|*ivp-t0* := *ti*, *ivp-x0* := *x0*|) **and** *K*=*i.T*])
subgoal by *unfold-locales*
subgoal using *assms* **by** *force*
subgoal by (*rule* ‹*x0* ∈ *X*›)
subgoal by (*rule i'-sol*)
subgoal by (*rule ivp.equality*; *simp add*: *assms*)
subgoal by (*rule* ‹*is-interval i.T*›)
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
done

moreover have *ll-on-open.flow f T X ti x0 t* = *ivp.solution j t*
using *assms*
apply (*intro ll-on-open.maximal-existence-flow*[**where** *i*=*j*(|*ivp-t0* := *ti*, *ivp-x0* := *x0*|) **and** *K*=*j.T*])
subgoal by *unfold-locales*
subgoal using *assms* **by** *force*
subgoal by (*rule* ‹*x0* ∈ *X*›)
subgoal by (*rule j'-sol*)
subgoal by (*rule ivp.equality*; *simp add*: *assms*)
subgoal by (*rule* ‹*is-interval j.T*›)
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
done

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

lemma *flow-initial-time*[*simp*]: *flow t0* = *x0*
using *existence-ivp.solution-t0 flow-def* **by** *auto*

lemma *flow-has-derivative*:
assumes *t* ∈ *existence-ivl*
shows (*flow has-derivative* ($\lambda i.\ i *_R f\ t$ (*flow t*))) (*at t*)
proof −
have (*flow has-derivative* ($\lambda i.\ i *_R f\ t$ (*flow t*))) (*at t within existence-ivl*)
using *existence-ivp.solution-has-deriv*[*of t*] *assms*
unfolding *flow-def*[*abs-def*]
by (*auto simp*: *has-vector-derivative-def*)
thus *?thesis*
by (*simp add*: *at-within-open*[*OF assms open-existence-ivl*])
qed

lemma
flow-eq-rev:
defines *mirror* ≡ λ*t*. *2* ∗ *t0* − *t*
assumes *t* ∈ *existence-ivl*
shows *flow t* = *ll-on-open.flow* (λ*t*. − *f* (*mirror t*)) (*mirror* ' *T*) *X t0 x0* (*2* ∗ *t0* − *t*)
2 ∗ *t0* − *t* ∈ *ll-on-open.existence-ivl* (λ*t*. − *f* (*mirror t*)) (*mirror* ' *T*) *X t0 x0*
proof −
from *iv-in* **have** *mt0*: *t0* ∈ *mirror* ' *T*
by (*auto simp*: *mirror-def*)
have *subset*: *mirror* ' *existence-ivl* ⊆ *mirror* ' *T*
using *existence-ivl-subset*
by (*rule image-mono*)
have [*simp*]: *is-interval* (*mirror* ' *X*) ⟷ *is-interval X* **for** *X*
by (*auto simp*: *mirror-def*)
interpret *rev*: *ll-on-open* λ*t*. − *f* (*mirror t*) *mirror* ' *T*
unfolding *mirror-def* **..**
have *ivp.solution* (*existence-ivp*) ∘ *mirror* = (λ*t*. *flow* (*mirror t*))
by (*auto simp*: *flow-def*)
with *existence-ivp.mirror-solution*[*OF existence-ivp.is-solution-solution*, *simplified*]
have ∗:
ivp.is-solution
(*existence-ivp* (|*ivp-f* := λ(*t*, *x*). − *f* (*mirror t*) *x*, *ivp-T* := *mirror* ' *existence-ivl*|))
(λ*t*. *flow* (*mirror t*))
by (*auto simp*: *mirror-def*)
have *it*: *t0* ∈ *mirror* ' *existence-ivl*
using *existence-ivl-initial-time* **by** (*simp add*: *mirror-def*)
from *rev.maximal-existence-flow*[**where** *K*=*mirror* ' *existence-ivl*, *OF mt0 iv-in*(*2*) ∗ - - *it*]
have *mirror* ' *existence-ivl* ⊆ *ll-on-open.existence-ivl* (λ*t*. − *f* (*mirror t*)) (*mirror* ' *T*) *X t0 x0*
⋀*t*. *t* ∈ *mirror* ' *existence-ivl* ⟹ *rev.flow t0 x0 t* = *flow* (*mirror t*)
by (*auto simp*: *existence-ivp-def subset*)
then show *2* ∗ *t0* − *t* ∈ *rev.existence-ivl t0 x0 flow t* = *rev.flow t0 x0* (*2* ∗ *t0* − *t*)
using *assms* **by** *auto*
qed

lemma *rev-flow-eq*:
defines *mirror* ≡ λ*t*. *2* ∗ *t0* − *t*
shows *t* ∈ *ll-on-open.existence-ivl* (λ*t*. − *f* (*mirror t*)) (*mirror* ' *T*) *X t0 x0* ⟹
ll-on-open.flow (λ*t*. − *f* (*mirror t*)) (*mirror* ' *T*) *X t0 x0 t* = *flow* (*2* ∗ *t0* − *t*)
and *rev-existence-ivl-eq*:
t ∈ *ll-on-open.existence-ivl* (λ*t*. − *f* (*mirror t*)) (*mirror* ' *T*) *X t0 x0* ⟷ *2* ∗ *t0* − *t* ∈ *existence-ivl*
proof −
from *iv-in* **have** *mt0*: *t0* ∈ *mirror* ' *T* **by** (*auto simp*: *mirror-def*)

interpret *rev*: *ll-on-open* (*λt.* − *f* (*mirror t*)) (*mirror* ' *T*)
  unfolding *mirror-def* **..**
from *rev.flow-eq-rev*[*OF mt0 iv-in*(*2*), *of t*] *flow-eq-rev*[*of 2* ∗ *t0* −*t*]
show *t* ∈ *rev.existence-ivl t0 x0* ⟹ *rev.flow t0 x0 t* = *flow* (*2* ∗ *t0* − *t*)
  (*t* ∈ *rev.existence-ivl t0 x0*) = (*2* ∗ *t0* − *t* ∈ *existence-ivl*)
  by (*auto simp*: *mirror-def* ‹*x0* ∈ *X*› *fun-Compl-def image-image*)
qed

end — *t0* ∈ *T*
x0 ∈ *X*

end — *x0*

lemma
  assumes *s*: *s* ∈ *existence-ivl t0 x0*
  assumes *t*: *t* + *s* ∈ *existence-ivl s* (*flow t0 x0 s*)
  assumes *iv-in*[*simp*]: *t0* ∈ *T x0* ∈ *X*
  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 −
  have *s* ∈ *T*
    using *existence-ivl-subset iv-in*(*1*) *iv-in*(*2*) *s* **by** *blast*
  from *existence-ivp*[*OF iv-in*]
  interpret *u0*: *unique-solution existence-ivp t0 x0* **.**
  let *?u0r* = (*existence-ivp t0 x0*)(|*ivp-T*:=*if s* ≥ *t0 then* {*t0* .. *s*} *else* {*s* .. *t0*}|)
  interpret *u0r*: *ivp ?u0r*
    by *unfold-locales auto*
  have *has-solution ?u0r*
    apply *unfold-locales*
    apply (*rule exI*)
    apply (*rule u0.solution-on-subset*[*OF* - - *u0.is-solution-solution*])
    by (*auto intro*!: *in-existence-between-zeroI*[*OF iv-in s*])
  then interpret *u0r*: *has-solution ?u0r* **.**

  have *u0r.T* ⊆ *existence-ivl t0 x0*
    by (*auto intro*!: *in-existence-between-zeroI*[*OF iv-in s*])
  then have *u0r.T* ⊆ *T*
    using *existence-ivl-subset*[*OF iv-in*]
    by *auto*

  note *flow-in-domain*[*OF iv-in s*, *simp*]
  from *existence-ivp*[*OF* ‹*s* ∈ *T*› *this*]
  interpret *u1*: *unique-solution existence-ivp s* (*flow t0 x0 s*) **by** *simp*
  let *?u1* = (*existence-ivp s* (*flow t0 x0 s*))(|*ivp-T*:=*if t* ≥ *0 then* {*s*..*t* + *s*} *else*
{*t* + *s*..*s*}|)
  interpret *u1r*: *ivp ?u1*
    by *unfold-locales auto*
  interpret *u1r*: *has-solution ?u1*
    apply *unfold-locales*

```
    apply (rule exI)
    apply (rule u1.solution-on-subset[OF - - u1.is-solution-solution])
    by (auto intro!: in-existence-between-zeroI[OF ‹s ∈ T› ‹(flow t0 x0 s) ∈ X› t])

  have u1r.T ⊆ existence-ivl s (flow t0 x0 s)
    by (auto intro!: in-existence-between-zeroI[OF ‹s ∈ T› ‹(flow t0 x0 s) ∈ X› t])
  then have u1r.T ⊆ T
    using existence-ivl-subset[OF ‹s ∈ T› ‹(flow t0 x0 s) ∈ X›]
    by auto

  let ?c = (existence-ivp t0 x0)(|ivp-T:=ivp-T ?u0r ∪ ivp-T ?u1|)
  interpret conn: ivp ?c
    by unfold-locales (auto simp: iv-in)
  interpret conn: connected-solutions ?c ?u0r ?u1 u0r.solution
  proof unfold-locales
    show u0r.is-solution u0r.solution by simp
  next
    assume conn.t0 ∉ u0r.T
    thus u1r.solution conn.t0 = conn.x0
      by (simp split: if-split-asm)
  next
    assume conn.t0 ∈ u0r.T
    thus u0r.solution conn.t0 = conn.x0
      using u0r.solution-t0
      by (simp split: )
  next
    fix t assume t: t ∈ u0r.T ∩ u1r.T
    from ‹u0r.T ⊆ T› have fr: flow t0 x0 s = u0r.solution s
     by (intro maximal-existence-flow[where i=?u0r and K=ivp-T ?u0r])
       (auto simp: is-interval-closed-interval)
    hence fs: flow t0 x0 s = u1r.solution s
      using u1r.solution-t0
      by simp
    from t ‹has-solution ?u0r› ‹has-solution ?u1›
    show u0r.solution t = u1r.solution t
      apply (rule unique-on-intersection[OF ‹s ∈ T› ‹flow t0 x0 s ∈ X›])
      using fr[symmetric] fs[symmetric] ‹u0r.T ⊆ T› ‹u1r.T ⊆ T›
      by (auto simp: is-interval-closed-interval s)
  qed auto
  have flow t0 x0 (s + t) = (conn.connection (s + t))
     by (rule maximal-existence-flow[OF iv-in conn.is-solution-connection, where
K=ivp-T ?c])
        (insert ‹u0r.T ⊆ T› ‹u1r.T ⊆ T›, auto simp: is-interval-closed-interval
is-real-interval-union)
  also have conn.connection (s + t) = u1r.solution (s + t)
    by (rule conn.connection-eq-solution2) simp
  also
  from u1r.is-solution-solution
  have u1r.is-solution u1r.solution by simp
```

then have *flow s* (*flow t0 x0 s*) (*s* + *t*) = *u1r.solution* (*s* + *t*)
by (*rule maximal-existence-flow*(*2*)[*OF* ‹*s* ∈ *T*› ‹(*flow t0 x0 s*) ∈ *X*›, **where** *K*=*ivp-T ?u1*])
(*insert* ‹*u1r.T* ⊆ *T*›, *auto simp*: *is-interval-closed-interval is-real-interval-union*)
then have *u1r.solution* (*s* + *t*) = *flow s* (*flow t0 x0 s*) (*s* + *t*)
by (*simp add*: *algebra-simps*)
finally show *flow t0 x0* (*s* + *t*) = *flow s* (*flow t0 x0 s*) (*s* + *t*) **.**
have *s* + *t* ∈ *conn.T*
by *simp*
also have ... ⊆ *existence-ivl t0 x0* **using** *conn.is-solution-connection*
by (*rule maximal-existence-flow*[*OF iv-in*])
(*insert* ‹*u0r.T* ⊆ *T*› ‹*u1r.T* ⊆ *T*›, *auto simp*: *is-interval-closed-interval is-real-interval-union*)
finally show *s* + *t* ∈ *existence-ivl t0 x0* **.**
qed

lemma
assumes *t*: *t* ∈ *existence-ivl t0 x0*
assumes *iv-in*[*simp*]: *t0* ∈ *T x0* ∈ *X*
shows *flows-reverse*: *flow t* (*flow t0 x0 t*) *t0* = *x0*
and *existence-ivl-reverse*: *t0* ∈ *existence-ivl t* (*flow t0 x0 t*)
proof −
have *flow t0 x0 t* ∈ *X*
by (*rule flow-in-domain*; *fact*)
interpret *existence-ivp*: *unique-solution existence-ivp t0 x0*
by (*rule existence-ivp*; *fact*)
have *t0* ∈ {*t* .. *t0*} ∪ {*t0* .. *t*} **by** *force*
also
have ... ⊆ *existence-ivl t* (*flow t0 x0 t*)
apply (*rule maximal-existence-flow*[*OF* - - - *refl*, **where** *x*=*existence-ivp.solution*])
subgoal using *t existence-ivl-subset*[*OF iv-in*] **by** *force*
subgoal by *fact*
subgoal
using *in-existence-between-zeroI*[*OF iv-in t*]
by (*auto simp*: *flow-def*
intro!: *existence-ivp.shift-initial-value*[*OF existence-ivp.is-solution-solution*])
subgoal by (*auto intro*!: *is-real-interval-union is-interval-closed-interval*)
subgoal by *auto*
subgoal using *in-existence-between-zeroI*[*OF iv-in t*] *existence-ivl-subset*[*OF iv-in*] **by** *auto*
done
finally show *t0* ∈ *existence-ivl t* (*flow t0 x0 t*) **.**
with *flow-trans*[*OF t* - - ‹*x0* ∈ *X*›, *of t0* − *t*, *simplified*]
show *flow t* (*flow t0 x0 t*) *t0* = *x0* **by** *simp*
qed

lemma *flow-has-vector-derivative*:
assumes *t0* ∈ *T x* ∈ *X t* ∈ *existence-ivl t0 x*
shows (*flow t0 x has-vector-derivative f t* (*flow t0 x t*)) (*at t*)

using *flow-has-derivative*[*OF assms*]
by (*simp add*: *has-vector-derivative-def*)

lemma *flow-has-vector-derivative-at-0*:
assumes *t0* ∈ *T x* ∈ *X t* ∈ *existence-ivl t0 x*
shows ((λ*h*. *flow t0 x* (*t* + *h*)) *has-vector-derivative f t* (*flow t0 x t*)) (*at 0*)
proof −
from *flow-has-vector-derivative*[*OF assms*]
have
(*op* + *t has-vector-derivative 1*) (*at 0*)
(*flow t0 x has-vector-derivative f t* (*flow t0 x 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 *in-domain*: *t0* ∈ *T x* ∈ *X*
assumes *t* ∈ *existence-ivl t0 x*
shows *ivl-subset-existence-ivl*: {*t0* .. *t*} ⊆ *existence-ivl t0 x*
and *ivl-subset-existence-ivl'*: {*t* .. *t0*} ⊆ *existence-ivl t0 x*
and *closed-segment-subset-existence-ivl*: *closed-segment t0 t* ⊆ *existence-ivl t0 x*
using *assms in-existence-between-zeroI*[*OF in-domain*]
by (*auto simp*: *closed-segment-real*)

lemma *flow-fixed-point*:
assumes *t*: *t0* ≤ *t t* ∈ *existence-ivl t0 x*
assumes *iv-in*: *t0* ∈ *T x* ∈ *X*
shows *flow t0 x t* = *x* + *integral* {*t0*..*t*} (λ*t*. *f t* (*flow t0 x t*))
proof −
have ∀ *s*∈{*t0* .. *t*}. (*flow t0 x has-vector-derivative f s* (*flow t0 x s*)) (*at s within* {*t0* .. *t*})
using *ivl-subset-existence-ivl*[*OF iv-in t*(*2*)]
by (*auto intro*!: *flow-has-vector-derivative*[*OF iv-in*]
intro: *has-vector-derivative-at-within*)
from *fundamental-theorem-of-calculus*[*OF t*(*1*) *this*]
have ((λ*t*. *f t* (*flow t0 x t*)) *has-integral flow t0 x t* − *x*) {*t0*..*t*}
by (*simp add*: *iv-in*)
from *this*[*THEN integral-unique*]
show *?thesis* **by** (*simp add*: ‹*x* ∈ *X*›)
qed

lemma *flow-fixed-point'*:
assumes *t*: *t* ≤ *t0 t* ∈ *existence-ivl t0 x*
assumes *iv-in*: *t0* ∈ *T x* ∈ *X*
shows *flow t0 x t* = *x* − *integral* {*t*..*t0*} (λ*t*. *f t* (*flow t0 x t*))
proof −
have ∀ *s*∈{*t* .. *t0*}. (*flow t0 x has-vector-derivative f s* (*flow t0 x s*)) (*at s within* {*t* .. *t0*})

using *ivl-subset-existence-ivl′*[*OF iv-in t*(*2*)]
by (*auto intro*!: *flow-has-vector-derivative*[*OF iv-in*]
intro: *has-vector-derivative-at-within*)
from *fundamental-theorem-of-calculus*[*OF t*(*1*) *this*]
have ((λ*t*. *f t* (*flow t0 x t*)) *has-integral x* − *flow t0 x t*) {*t* .. *t0*}
by (*simp add*: *iv-in*)
from *this*[*THEN integral-unique*]
show *?thesis* **by** (*simp add*: ‹*x* ∈ *X*› *algebra-simps*)
qed

lemma *flow-fixed-point′′*:
assumes *t*: *t* ∈ *existence-ivl t0 x*
assumes *t0* ∈ *T x* ∈ *X*
shows *flow t0 x t* =
x + (*if t0* ≤ *t then 1 else* −*1*) $*_R$ *integral* (*closed-segment t0 t*) (λ*t*. *f t* (*flow t0 x t*))
using *assms*
by (*auto simp add*: *closed-segment-real flow-fixed-point flow-fixed-point′*)

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

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

lemma *flow-continuous-on*: *t0* ∈ *T* ⟹ *x* ∈ *X* ⟹ *continuous-on* (*existence-ivl t0 x*) (*flow t0 x*)
by (*auto intro*!: *flow-continuous continuous-at-imp-continuous-on*)

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

lemma *f-flow-continuous*:
assumes *t* ∈ *existence-ivl t0 x t0* ∈ *T x* ∈ *X*
shows *isCont* (λ*t*. *f t* (*flow t0 x 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*)

lemma *exponential-initial-condition-nonneg*:
assumes *t* ≥ *t0 t0* ∈ *T*

assumes *y0*: $t \in$ *existence-ivl t0 y0* **and** $y0 \in Y$
assumes *z0*: $t \in$ *existence-ivl t0 z0* **and** $z0 \in Y$
assumes $Y \subseteq X$
assumes *remain*: $\bigwedge s.\ s \in \{t0\ ..\ t\} \Longrightarrow$ *flow t0 y0 s* $\in Y$
$\bigwedge s.\ s \in \{t0\ ..\ t\} \Longrightarrow$ *flow t0 z0 s* $\in Y$
assumes *lipschitz*: $\bigwedge s.\ s \in \{t0\ ..\ t\} \Longrightarrow$ *lipschitz Y (f s) K*
shows *norm* (*flow t0 y0 t* $-$ *flow t0 z0 t*) $\leq$ *norm* ($y0 - z0$) $*$ *exp* (($K + 1$) $*$ ($t - t0$))
proof *cases*
assume $y0 = z0$
thus *?thesis*
by *simp*
next
assume *ne*: $y0 \neq z0$
def $K' \equiv K + 1$
from *lipschitz* **have** *lipschitz Y (f s) K'* **if** $s \in \{t0\ ..\ t\}$ **for** *s*
using *that*
by (*auto simp*: *lipschitz-def K'-def*
intro!: *order-trans*[*OF - mult-right-mono*[*of K K + 1*]])
from *assms* **have** *inX*: $y0 \in X$ $z0 \in X$ **by** *auto*
def $v \equiv \lambda t.$ *norm* (*flow t0 y0 t* $-$ *flow t0 z0 t*)
{
fix *s*
assume *s*: $s \in \{t0\ ..\ t\}$ **hence** $s \geq t0$ **by** *auto*
with *s*
ivl-subset-existence-ivl[*OF* ‹$t0 \in T$› ‹$y0 \in X$› *y0*]
ivl-subset-existence-ivl[*OF* ‹$t0 \in T$› ‹$z0 \in X$› *z0*]
have
y0': $s \in$ *existence-ivl t0 y0* **and**
z0': $s \in$ *existence-ivl t0 z0*
by *auto*
have *integrable*:
($\lambda t.$ *f t* (*ll-on-open.flow f T X t0 y0 t*)) *integrable-on* $\{t0..s\}$
($\lambda t.$ *f t* (*ll-on-open.flow f T X t0 z0 t*)) *integrable-on* $\{t0..s\}$
using *ivl-subset-existence-ivl*[*OF* ‹$t0 \in T$› ‹$y0 \in X$› *y0'*]
ivl-subset-existence-ivl[*OF* ‹$t0 \in T$› ‹$z0 \in X$› *z0'*]
‹$y0 \in X$› ‹$z0 \in X$› ‹$t0 \in T$›
by (*auto intro*!: *continuous-at-imp-continuous-on f-flow-continuous*)
hence *int*: *flow t0 y0 s* $-$ *flow t0 z0 s* $=$
$y0 - z0 +$ *integral* $\{t0\ ..\ s\}$ ($\lambda t.$ *f t* (*flow t0 y0 t*) $-$ *f t* (*flow t0 z0 t*))
unfolding *v-def*
by (*auto simp*: *algebra-simps flow-fixed-point*[*OF* ‹$s \geq t0$› *y0'* ‹$t0 \in T$› ‹$y0 \in X$›]
flow-fixed-point[*OF* ‹$s \geq t0$› *z0'* ‹$t0 \in T$› ‹$z0 \in X$›] *integral-diff*)
have $v\ s \leq v\ t0 + K' *$ *integral* $\{t0\ ..\ s\}$ ($\lambda t.\ v\ t$)
using *ivl-subset-existence-ivl*[*OF* ‹$t0 \in T$› ‹$y0 \in X$› *y0'*]
ivl-subset-existence-ivl[*OF* ‹$t0 \in T$› ‹$z0 \in X$› *z0'*] *s*
by (*subst integral-mult*)
(*auto simp*: *integral-mult v-def int inX* ‹$t0 \in T$› *simp del*: *integral-mult-right*

```
        intro!: norm-triangle-le integral-norm-bound-integral
          integrable-continuous-real continuous-intros
          continuous-at-imp-continuous-on flow-continuous f-flow-continuous
          lipschitz-norm-leI[OF ‹- ⟹ lipschitz - - K'›] remain)
  } note le = this
  have cont: continuous-on {t0..t} v
    using ivl-subset-existence-ivl[OF ‹t0 ∈ T› ‹y0 ∈ X› y0]
      ivl-subset-existence-ivl[OF ‹t0 ∈ T› ‹z0 ∈ X› 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 lipschitzD[OF lipschitz ‹y0 ∈ Y› ‹z0 ∈ Y›, of t0] ‹t0 ≤ t› 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
  have v t ≤ v t0 ∗ exp (K' ∗ (t − t0))
    apply (rule gronwall-general[OF le cont nonneg pos lippos])
    using ‹t0 ≤ t› by simp-all
  thus ?thesis
    by (simp add: v-def K'-def ‹t0 ∈ T› inX)
qed

lemma exponential-initial-condition:
  assumes t0 ∈ T
  assumes y0: t ∈ existence-ivl t0 y0 and y0 ∈ Y
  assumes z0: t ∈ existence-ivl t0 z0 and z0 ∈ Y
  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 ⟹ lipschitz Y (f s) K
  shows norm (flow t0 y0 t − flow t0 z0 t) ≤ norm (y0 − z0) ∗ exp ((K + 1) ∗
abs (t − t0))
  using assms
proof cases
  assume t0 ≤ t
  with assms remain lipschitz
  have norm (flow t0 y0 t − flow t0 z0 t) ≤ norm (y0 − z0) ∗ exp ((K + 1) ∗ (t
− t0))
    by (intro exponential-initial-condition-nonneg)
      (auto simp: closed-segment-real)
```

thus *?thesis*
using ‹*t0* ≤ *t*› **by** *simp*
next
have *y0* ∈ *X* *z0* ∈ *X* **using** *assms* **by** *auto*
let *?m* = λ*t*. *2* ∗ *t0* − *t*
{
fix *s y0 Y* **assume** *y0* ∈ *X*
and *remain*: ⋀*s*. *s* ∈ *closed-segment t0 t* ⟹ *flow t0 y0 s* ∈ *Y*
and *y0*: *t* ∈ *existence-ivl t0 y0*
and *s*: *s* ∈ {*t0* .. *2* ∗ *t0* − *t*}
have *ll-on-open.flow* (λ*t*. − *f* (*2* ∗ *t0* − *t*)) (*?m* ‘ *T*) *X t0 y0 s* =
ll-on-open.flow (λ*t*. − *f* (*2* ∗ *t0* − *t*)) (*?m* ‘ *T*) *X t0 y0* (*2* ∗ *t0* − (*2* ∗ *t0* − *s*))
by *simp*
also have ... = *flow t0 y0* (*2* ∗ *t0* − *s*)
proof (*rule flow-eq-rev*(*1*)[*symmetric*])
have *2* ∗ *t0* + − *1* ∗ *s* ∈ {*t*..*t0*} ∪ {*t0*..*t*}
using *s* **by** *force*
then have *2* ∗ *t0* + − *1* ∗ *s* ∈ *existence-ivl t0 y0*
using ‹*t0* ∈ *T*› ‹*y0* ∈ *X*› *ll-on-open.in-existence-between-zeroI ll-on-open-axioms y0* **by** *blast*
then show *2* ∗ *t0* − *s* ∈ *existence-ivl t0 y0*
by *auto*
qed *fact*+
also have ... ∈ *Y*
using *s* **by** (*simp add*: *closed-segment-real remain*)
finally
have *ll-on-open.flow* (λ*t*. − *f* (*2* ∗ *t0* − *t*)) (*?m* ‘ *T*) *X t0 y0 s* ∈ *Y* **.**
} **note** *remain-rev* = *this*
interpret *rev*: *ll-on-open* (λ*t*. − *f* (*2* ∗ *t0* − *t*)) *?m* ‘ *T* **..**
assume ¬ *t* ≥ *t0*
then have *norm* (*rev.flow t0 y0* (*2* ∗ *t0* − *t*) − *rev.flow t0 z0* (*2* ∗ *t0* − *t*)) ≤
norm (*y0* − *z0*) ∗ *exp* ((*K* + *1*) ∗ (*2* ∗ *t0* − *t* − *t0*))
using *lipschitz* ‹*t0* ∈ *T*› ‹*y0* ∈ *Y*› ‹*z0* ∈ *Y*› ‹*Y* ⊆ *X*›
by (*intro rev.exponential-initial-condition-nonneg*)
(*auto intro*!: *flow-eq-rev*[*OF* ‹*t0* ∈ *T*› ‹*z0* ∈ *X*› *z0*] *flow-eq-rev*[*OF* ‹*t0* ∈ *T*› ‹*y0* ∈ *X*› *y0*]
remain-rev remain y0 z0 lipschitz
simp: *lipschitz-uminus′ closed-segment-real*)
thus *?thesis*
using ‹¬ *t* ≥ *t0*›
by (*simp add*: *flow-eq-rev*[*OF* ‹*t0* ∈ *T*› ‹*y0* ∈ *X*› *y0*] *flow-eq-rev*[*OF* ‹*t0* ∈ *T*› ‹*z0* ∈ *X*› *z0*])
qed

lemma
existence-ivl-cballs:
assumes *iv-in*: *t0* ∈ *T* *x0* ∈ *X*
obtains *t u L*

where
$\bigwedge y.\ y \in cball\ x0\ u \Longrightarrow cball\ t0\ t \subseteq existence\text{-}ivl\ t0\ y$
$\bigwedge s\ y.\ y \in cball\ x0\ u \Longrightarrow s \in cball\ t0\ t \Longrightarrow flow\ t0\ y\ s \in cball\ y\ u$
$lipschitz\ (cball\ t0\ t \times cball\ x0\ u)\ (\lambda(t, x).\ flow\ t0\ x\ t)\ L$
$\bigwedge y.\ y \in cball\ x0\ u \Longrightarrow cball\ y\ u \subseteq X$
$0 < t\ 0 < u$
proof −
from *local-unique-solutions*[*OF iv-in*]
obtain *t u L* **where** *usol*: $\bigwedge y.\ y \in cball\ x0\ u \Longrightarrow$
unique-solution (*ll-on-open.existence-ivp f T X t0 x0* (|*ivp-x0* := *y*, *ivp-T* := *cball t0 t*, *ivp-X* := *cball y u*|))
and *subs*: $\bigwedge y.\ y \in cball\ x0\ u \Longrightarrow cball\ y\ u \subseteq X$
and *lipschitz*: $\bigwedge s.\ s \in cball\ t0\ t \Longrightarrow lipschitz\ (cball\ x0\ (2{*}u))\ (f\ s)\ L$
and *subsT*: $cball\ t0\ t \subseteq T$
and *subs'*: $cball\ x0\ (2 * u) \subseteq X$
and *tu*: $0 < t\ 0 < u$
by *metis*
{
fix *y* **assume** *y*: $y \in cball\ x0\ u$
from *subs*[*OF y*] ‹$0 < u$› **have** $y \in X$ **by** *auto*
from *usol*[*OF y*] **interpret** *unique-solution*
(*ll-on-open.existence-ivp f T X t0 x0* (|*ivp-x0* := *y*, *ivp-T* := *cball t0 t*, *ivp-X* := *cball y u*|))
.
note ∗ = *maximal-existence-flow*[*OF* ‹$t0 \in T$› ‹$y \in X$› *is-solution-on-superset-domain*[*OF is-solution-solution*],
where *K* = *cball t0 t*, *simplified existence-ivp-def*, *simplified*, *OF subs*,
OF y refl less-imp-le[*OF* ‹$0 < t$›] *subsT*]
from ∗ **have** $cball\ t0\ t \subseteq existence\text{-}ivl\ t0\ y$
by *simp*
note *eivl* = *this*
{
fix *s::real* **assume** *s*: $s \in cball\ t0\ t$
from ∗(*2*)[*of s*] *this* **have** *flow t0 y s* = *solution s*
by (*auto simp*: *existence-ivp-def*)
also
from *s is-solutionD*(*3*)[*OF is-solution-solution*]
have ... ∈ *cball y u*
by (*auto simp del*: *mem-cball*)
finally have $flow\ t0\ y\ s \in cball\ y\ u$ **.**
}
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*) ‹$t0 \in T$›

by (*auto intro*!: *continuous-at-imp-continuous-on f-flow-continuous*)
thm *compact-Times*[*OF compact-cball compact-cball*]
have *bounded* ((λ(*t*, *x*). *f t x*) ‘ (*cball t0 t* × *cball x0* (*2* ∗ *u*)))
using *mem-cball subs′ subsT*
by (*auto intro*!: *compact-imp-bounded compact-continuous-image compact-Times continuous-intros*
simp: *split-beta′*)
then obtain *B* **where** *B*: ⋀*s y*. *s* ∈ *cball t0 t* ⟹ *y* ∈ *cball x0* (*2* ∗ *u*) ⟹ *norm* (*f s y*) ≤ *B B* > *0*
by (*auto simp*: *bounded-pos cball-def*)
{
fix *s*::*real* **and** *x1* **assume** *s*: *s* ∈ *cball t0 t* **and** *x1*: *x1* ∈ *cball x0 u*
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 have *flow t0 x1 s* ∈ *cball x0* (*2* ∗ *u*) **.**
} **note** *flow-in-cball* = *this*
have *lipschitz* (*cball t0 t*×*cball x0 u*) (λ(*t*, *x*). *flow t0 x t*) (*B* + *exp* ((*L* + *1*) ∗ |*t*|))
proof (*rule lipschitzI*, *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*[*of t0 t2 x1 cball x0* (*2* ∗ *u*) *x2*])
fix *s* **assume** *s* ∈ *closed-segment t0 t2* **hence** *s*: *s* ∈ *cball t0 t*
using *t2*
by (*auto simp*: *dist-real-def closed-segment-real 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 *lipschitz* (*cball x0* (*2* ∗ *u*)) (*f s*) *L* **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-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 integrable:
      (λt. f t (flow t0 x1 t)) integrable-on {t0..max t1 t2}
      (λt. f t (flow t0 x1 t)) integrable-on {t2..t1}
      (λt. f t (flow t0 x1 t)) integrable-on {t1..t2}
      (λt. f t (flow t0 x1 t)) integrable-on {min t2 t1..t0}
      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-real
        simp: dist-real-def integral-minus-sets')
    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›
        integral-combine[of t2 t0 t1 λt. f t (flow t0 x1 t)]
        integral-combine[of t1 t0 t2 λt. f t (flow t0 x1 t)]
      by (auto simp: flow-fixed-point'' closed-segment-real dist-norm add.commute
          norm-minus-commute integral-minus-sets' integral-minus-sets
        intro!: order-trans[OF integral-bound[where B=B]] cont-on-f-flow B)
    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

lemma filterlim-real-at-infinity-sequentially[tendsto-intros]:
    filterlim real at-infinity sequentially
  by (simp add: filterlim-at-top-imp-at-infinity filterlim-real-sequentially)

lemma existence-ivl-ninfty:
  assumes iv-in: t0 ∈ T x0 ∈ X
  shows inf-existence-ninfty[intro,simp]: inf-existence t0 x0 ≠ ∞
    and sup-existence-ninfty[intro,simp]: sup-existence t0 x0 ≠ −∞
```

using *existence-ivl-initial-time*[*OF iv-in*]
by (*auto simp*: *existence-ivl-def*)

lemma
flow-leaves-compact-ivl: — explosion if the solution exists for only finite time
assumes *iv-in*: $t0 \in T$ $x0 \in X$
assumes *sup-existence t0 x0* $< \infty$
assumes *real-of-ereal* (*sup-existence t0 x0*) $\in T$
assumes *compact K*
assumes $K \subseteq X$
obtains *t* **where** $t \geq t0$ $t \in$ *existence-ivl t0 x0 flow t0 x0 t* $\notin K$
proof (*atomize-elim*, *rule ccontr*, *auto*)
assume $\forall t.\ t \in$ *ll-on-open.existence-ivl f T X t0 x0* $\longrightarrow t0 \leq t \longrightarrow$ *flow t0 x0 t* $\in K$
note *flow-in-K* = *this*[*rule-format*]
with *assms* **obtain** *b* **where** *b*: *sup-existence t0 x0* = *ereal b*
by (*cases sup-existence t0 x0*) *auto*
from *b* **have** *b-gtI*: $b > s$ **if** $s \in$ *existence-ivl t0 x0* **for** *s*
using *that*
by (*auto simp add*: *existence-ivl-def ereal-less-ereal-Ex*)

from *assms b* **have** $b \in T$ **by** *simp*
from *b* **have** $b > t0$
by (*auto intro*!: *b-gtI iv-in*)
from *b* **have** $b >$ *inf-existence t0 x0*
using *existence-ivl-initial-time*[*OF iv-in*]
by (*auto simp add*: *existence-ivl-def assms*)
note *b-gt* = ‹$b >$ *inf-existence t0 x0*› ‹$b > t0$›

have *in-existence-ivlI*: $\bigwedge t.\ t0 \leq t \Longrightarrow t < b \Longrightarrow t \in$ *existence-ivl t0 x0*
using *b existence-ivl-ninfty*[*OF iv-in*] *existence-ivl-initial-time*[*OF iv-in*]
by (*auto simp*: *existence-ivl-def assms real-image-ereal-ivl*
split: *if-split-asm*)

have *ev1*: *eventually* ($\lambda n.\ b - 1 / n >$ *inf-existence t0 x0*) *sequentially*
using - *b-gt*(*1*)
by (*rule order-tendstoD*) (*auto intro*: *tendsto-eq-intros seq-harmonic'*)
have *ev2*: *eventually* ($\lambda n.\ n > 0$) *sequentially*
by (*metis eventually-at-top-dense*)
have *ev3*: *eventually* ($\lambda n.\ t0 + 1 / n < b$) *sequentially*
by (*rule order-tendstoD*) (*auto intro*!: *tendsto-intros tendsto-divide-0* ‹$t0 < b$›)
let *?f* = λn::*nat*. *flow t0 x0* ($b - 1/n$)
from *eventually-conj*[*OF ev1 eventually-conj*[*OF ev2 ev3*]]
obtain *N*::*nat* **where** *N*: $N > 0$ *inf-existence t0 x0* $< (b - 1 / N)$ $t0 + 1 / N < b$
by (*auto dest*!: *eventually-happens*)
let *?fN* = *?f o* (*op* + *N*)

have $\{t0\ ..\ b\} \subseteq T$

```
  proof
    fix x assume x ∈ {t0 .. b}
    then show x ∈ T
       by (cases x = b) (auto simp: ‹b ∈ T› intro!: mem-existence-ivl-subset[OF
iv-in] in-existence-ivlI)
  qed
  then have bounded ((λ(t, x). f t x) ‘ ({t0 .. b} × K))
    using ‹K ⊆ X› ‹compact K› iv-in
    by (auto intro!: compact-imp-bounded compact-continuous-image
      continuous-intros compact-Times
      simp: split-beta subset-iff)
  then obtain M where M: ⋀t x. t ∈ {t0 .. b} ⟹ x ∈ K ⟹ norm (f t x) ≤
M M > 0
    by (force simp: bounded-pos)
  {
    fix t1 t2
    assume H: t1 ∈ existence-ivl t0 x0 t2 ∈ existence-ivl t0 x0 t0 ≤ t1 t0 ≤ t2
    {
      fix t1 t2
      assume t1: t1 ∈ existence-ivl t0 x0
        and t2: t2 ∈ existence-ivl t0 x0
      assume t0 ≤ t1
      assume t1 < t2
      let ?I = λivl. (λt. f t (flow t0 x0 t)) integrable-on ivl
      have I[simp]: ?I {t0 .. t1} ?I {t0 .. t2} ?I {t1 .. t2} ?I {t1 .. t0}
        using closed-segment-subset-existence-ivl[OF iv-in t1]
          closed-segment-subset-existence-ivl[OF iv-in t2] ‹t1 < t2› ‹t0 ∈ T›
        by (force intro!: integrable-continuous-real continuous-at-imp-continuous-on
          f-flow-continuous ‹x0 ∈ X› simp: closed-segment-real split: if-split-asm)+
     hence flow t0 x0 t2 − flow t0 x0 t1 = integral {t1..t2} (λt. f t (flow t0 x0 t))
        unfolding flow-fixed-point''[OF ‹t1 ∈ existence-ivl t0 x0› iv-in]
          flow-fixed-point''[OF ‹t2 ∈ existence-ivl t0 x0› iv-in]
        using ‹t1 < t2› integral-combine[of t1 t0 t2 λt. f t (flow t0 x0 t)]
        by (auto simp: closed-segment-real algebra-simps integral-combine)
      also have norm ... ≤ M ∗ (t2 − t1)
        using closed-segment-subset-existence-ivl[OF iv-in t1]
          closed-segment-subset-existence-ivl[OF iv-in t2] ‹t0 ≤ t1› ‹t1 < t2›
          b-gtI[OF t2]
        by (intro integral-bound)
          (auto intro!: flow-in-K M continuous-at-imp-continuous-on
            f-flow-continuous iv-in
            simp: closed-segment-real)
      finally have dist (flow t0 x0 t2) (flow t0 x0 t1) ≤ M ∗ (t2 − t1)
        by (simp add: dist-norm)
    } from this[of t1 t2] this[of t2 t1] H
    have dist (flow t0 x0 t1) (flow t0 x0 t2) ≤ M ∗ abs (t2 − t1)
      by (auto simp: abs-real-def dist-commute not-less less-eq-real-def)
  } note dist-flow-le = this
    — TODO: Cauchy really needed in the following?
```

```
  have Cauchy ?f
  proof (rule metric-CauchyI)
    fix e::real assume 0 < e
    have (λn. M / n) ⟶ 0
      by (auto intro!: tendsto-divide-0 tendsto-eq-intros
        simp: filterlim-at-top-imp-at-infinity filterlim-real-sequentially)
    hence eventually (λn. M / n < e/2) sequentially
      by (metis (poly-guards-query) ‹0 < e› half-gt-zero-iff order-tendsto-iff)
     from eventually-conj[OF this eventually-conj[OF ev1 eventually-conj[OF ev2
ev3]]]
    obtain N::nat
    where N: N > 0 M / N < e / 2 inf-existence t0 x0 < (b − 1 / N) t0 + 1 /
N < b
      by (auto dest!: eventually-happens)
    {
      fix n m assume n ≥ N m ≥ N
      with N have nm: n > 0 m > 0 b − 1 / N ≤ b − 1 / n
        b − 1 / N ≤ b − 1 / m t0 +  1/ n ≤ t0 + 1 / N
        by (auto intro!: divide-left-mono)
       from le-less-trans[OF ‹t0 + 1 / n ≤ t0 + 1/N› ‹t0 + 1/N < b›] have t0
+ 1/n < b .
      with nm have dist (flow t0 x0 (b − 1 / n)) (flow t0 x0 (b − 1 / m)) ≤
          M ∗ abs (b − 1 / m − (b − 1 / n))
        using b N existence-ivl-ninfty[OF iv-in] b-gt(1) less-ereal.simps(1)
        by (intro dist-flow-le;
          cases inf-existence t0 x0;
          simp add: existence-ivl-def real-image-ereal-ivl)
      also have ... ≤ M ∗ (1 / m + 1 / n)
      using ‹M > 0› by (auto intro!: mult-left-mono order-trans[OF abs-triangle-ineq4])
      also have ... ≤ M / m + M / n by (simp add: algebra-simps)
       also have ... ≤ M / N + M / N using nm ‹n ≥ N› ‹m ≥ N› ‹M>0› ‹0
< N›
        by (intro add-mono) (auto intro!: divide-left-mono mult-pos-pos)
      also have ... < e / 2 + e / 2 using N by (intro add-strict-mono) simp
      also have ... = e by simp
      finally have dist (flow t0 x0 (b − 1 / n)) (flow t0 x0 (b − 1 / m)) < e .
    }
    thus ∃ M::nat. ∀ m≥M. ∀ n≥M.
      dist (flow t0 x0 (b − 1 / real m)) (flow t0 x0 (b − 1 / real n)) < e
      by blast
  qed
  hence Cauchy ?fN
    by (rule Cauchy-subseq-Cauchy) (metis nat-add-left-cancel-less subseq-def)
  moreover
  {
    {
      fix n::nat
      have inf-existence t0 x0 < (b − 1 / N) by fact
```

```
    also have ... ≤ (b − 1 / (N + n))
      using ‹0 < N›
      by (auto intro!: divide-left-mono mult-pos-pos add-pos-nonneg)
    finally have inf-existence t0 x0 < (b − 1 / (N + n)) .
  } moreover {
    fix n::nat
    have t0 + 1 / (real N + real n) ≤ t0 + 1 / N
      by (auto intro!: divide-left-mono mult-pos-pos add-pos-nonneg ‹0 < N›)
    also note ‹... < b›
    finally have t0 < b − 1 / (N + n) by simp
  } ultimately
  have (∀ n. ?fN n ∈ K)
    using existence-ivl-ninfty[OF iv-in] b-gt ‹0 < N› N
    by (cases inf-existence t0 x0)
      (auto intro!: add-pos-nonneg flow-in-K less-imp-le
        simp: existence-ivl-def ‹x0 ∈ X› real-image-ereal-ivl b)
}
ultimately
have ∃ l∈K. ?fN ⟶ l
  using ‹compact K›
 by (auto simp: compact-eq-bounded-closed complete-eq-closed[symmetric] complete-def)
then obtain x1 where x1: x1 ∈ K ?fN ⟶ x1 by metis
hence x1 ∈ X using assms by auto

have flow-at-b: (flow t0 x0 ⟶ x1) (at b within {t0 .. b})
proof (rule tendstoI)
  fix e::real assume 0 < e hence 0 < e / 2 by auto
  from x1(2)[THEN tendstoD, OF this]
  have ev3: eventually (λn. dist ((?fN) n) x1 < e/2) sequentially .
  have eventually (λn. 1 / n < e / (2 ∗ M)) sequentially
    by (rule order-tendstoD[where y = 0])
      (auto intro!: tendsto-divide-0 tendsto-intros divide-pos-pos
        ‹0 < e› ‹0 < M›)
  hence ev4: eventually (λn. 1 / (N + n) < e / (2 ∗ M)) sequentially
    using ev2
  proof eventually-elim
    case (elim n)
    hence 1 / real (N + n) < 1 / n
      by (auto intro!: divide-strict-left-mono ‹0 < N›)
    also have ... < e / (2 ∗ M)  by fact
    finally show ?case .
  qed
  from eventually-conj[OF ev3 eventually-conj [OF ev4 ev2]]
  obtain N′
  where N′: dist (?fN N′) x1 < e / 2 N′ > 0 1 / (N + N′) < e / (2 ∗ M)
    by (auto dest!: eventually-happens)

  have eventually (λx. x < b) (at b within {t0 .. b})
    by (auto simp: eventually-at-filter)
```

moreover
have *eventually* (λ*x*. *x* > *b* − *1* / (*real N′* + *real N*)) (*at b within* {*t0* .. *b*})
using *N′* **by** (*auto intro*!: *order-tendstoD*)
moreover
have *eventually* (λ*x*. *x* < *b* − (*1* / *real* (*N* + *N′*) − *e* / *2* / *M*)) (*at b within* {*t0* .. *b*})
using *N′* **by** (*auto intro*!: *order-tendstoD*)
hence *eventually* (λ*x*. *x* − (*b* − *1* / *real* (*N* + *N′*)) < *e* / *2* / *M*) (*at b within* {*t0* .. *b*})
by (*simp add*: *algebra-simps*)
moreover
have *eventually* (λ*x*. *x* > *t0*) (*at b within* {*t0* .. *b*}) *eventually* (λ*x*. *x* < *b*) (*at b within* {*t0* .. *b*})
using *b-gt*
by (*intro order-tendstoD*)
(*auto simp*: *eventually-at-filter intro*!: *tendsto-intros*)
moreover
hence *eventually* (λ*x*. *x* ∈ *existence-ivl t0 x0*) (*at b within* {*t0* .. *b*})
by *eventually-elim* (*auto simp*: *in-existence-ivlI*)
ultimately have *eventually* (λ*x*. *dist* (*flow t0 x0 x*) (*?fN N′*) < *e* / *2*) (*at b within* {*t0* .. *b*})
proof *eventually-elim*
case (*elim x*)
have *dist* (*flow t0 x0 x*) (*flow t0 x0* (*b* − *1* / *real* (*N* + *N′*))) ≤
M ∗ |*b* − *1* / *real* (*N* + *N′*) − *x*|
proof (*rule dist-flow-le*)
have *t0* + *1* / *real* (*N* + *N′*) ≤ *t0* + *1* / *N*
by (*auto intro*!: *divide-left-mono mult-pos-pos add-pos-nonneg* ‹*0* < *N*›)
also have ... < *b* **by** *fact*
finally
show *t0* ≤ *b* − *1* / *real* (*N* + *N′*) **by** *simp*
then show *b* − *1* / *real* (*N* + *N′*) ∈ *existence-ivl t0 x0*
using *elim* ‹*0* < *N′*›
by (*auto intro*!: *in-existence-ivlI*)
qed (*intro elim less-imp-le*)+
also have |*b* − *1* / *real* (*N* + *N′*) − *x*| = *x* − (*b* − *1* / *real* (*N* + *N′*))
using ‹*N* > *0*› ‹*N′* > *0*› *elim*
by (*auto simp*: *abs-real-def algebra-simps*)
also have *M* ∗ ... < *M* ∗ (*e* / *2* / *M*)
by (*rule mult-strict-left-mono*) *fact*+
also have ... = *e* / *2*
using ‹*0* < *M*› **by** *simp*
finally show *?case* **by** (*simp add*: *o-def*)
qed
thus *eventually* (λ*x*. *dist* (*flow t0 x0 x*) *x1* < *e*) (*at b within* {*t0* .. *b*})
proof *eventually-elim*
case (*elim x*)
have *dist* (*flow t0 x0 x*) *x1* ≤ *dist* (*flow t0 x0 x*) (*?fN N′*) + *dist* (*?fN N′*) *x1*
by (*rule dist-triangle*)

also note *elim*
also note *N'(1)*
finally show *?case* **by** *simp*
qed
qed

def *u* ≡ *λt. if t < b then flow t0 x0 t else x1*
{
fix *s* **assume** *s*: *t0 < s s < b*
hence *s* ∈ *interior {t0 .. b}*
by (*simp add*: *interior-atLeastAtMost*)
hence *at s within {t0 .. b} = at s*
by (*subst at-within-interior*) *auto*
also
have *at s = at s within {t0 <..< b}*
using *s* **by** (*subst* (*2*) *at-within-open*) *auto*
also
have ∀$_F$ *x in at s within {t0<..<b}. flow t0 x0 x = (if x < b then flow t0 x0 x else x1)*
by (*auto simp*: *eventually-at-filter* ‹*x0* ∈ *X*› *intro*!: *in-existence-ivlI*)
hence ((*λt. u t*) ⟶ *u s*) . . .
using *s*
by (*intro filterlim-mono-eventually*[*OF tendsto-eq-rhs*[*OF flow-tendsto*] *order.refl*])
(*auto simp add*: *iv-in in-existence-ivlI u-def*)
finally have (*u* ⟶ *u s*) (*at s within {t0..b}*) **.**
} **note** *u-below-b = this*
have ((*λt. u t*) ⟶ *u b*) (*at b within {t0 .. b}*)
by (*rule filterlim-mono-eventually*[*OF tendsto-eq-rhs*[*OF flow-at-b*] *order.refl*])
(*auto simp*: *eventually-at-filter u-def*)
hence *u-at-b*: ((*λt. u t*) ⟶ *u b*) (*at b within {t0 .. b}*)
by (*rule tendsto-within-subset*) *auto*
have *eventually* (*λx. x < b*) (*at t0 within {t0 .. b}*)
using ‹*t0 < b*›
by (*auto intro*!: *order-tendstoD*)
hence ∀$_F$ *x in at t0 within {t0..b}. flow t0 x0 x = (if x < b then flow t0 x0 x else x1)*
by *eventually-elim auto*
then have *u-at-t0*: ((*λt. u t*) ⟶ *u t0*) (*at t0 within {t0 .. b}*)
using ‹*t0 < b*›
by (*intro filterlim-mono-eventually*[*OF tendsto-eq-rhs*[*OF flow-tendsto*[**where** *ts=λx. x*]]])
(*auto simp add*: *iv-in u-def*)

{
fix *s* **assume** *t0 ≤ s s ≤ b*
with *u-at-b u-below-b u-at-t0* **have** (*u* ⟶ *u s*) (*at s within {t0 .. b}*)
by (*cases s = b*; *cases s = t0*; *simp*)
}

hence *u-cont*: *continuous-on* {*t0* .. *b*} *u*
by (*auto simp*: *continuous-on*)
moreover
{
fix *t* **assume** *t*: *t0* ≤ *t* *t* < *b*
hence *u t* = *x0* + *integral* {*t0* .. *t*} (λ*s*. *f s* (*u s*))
by (*subst integral-spike*[**where** *s*={*b*} **and** *g* = λ*s*. *f s* (*flow t0 x0 s*)])
(*auto simp*: *u-def flow-fixed-point iv-in not-less in-existence-ivlI*)
} **note** *u-fixed-point* = *this*
have *cont*: *continuous-on* {*t0* .. *b*} (λ*s*. *f s* (*u s*))
using ‹{*t0* .. *b*} ⊆ *T*›
by (*safe intro*!: *continuous-intros u-cont*)
(*auto simp*: *u-def intro*!: *flow-in-domain iv-in* ‹*x1* ∈ *X*› *in-existence-ivlI*)

have *fixed-point-tendsto*:
((λ*t*. *x0* + *integral* {*t0* .. *t*} (λ*s*. *f s* (*u s*))) ⟶
x0 + *integral* {*t0* .. *b*} (λ*s*. *f s* (*u s*))) (*at b within* {*t0* .. *b*})
using ‹*t0* < *b*›
by (*auto intro*!: *integrable-continuous-real cont tendsto-intros*
indefinite-integral-continuous[*unfolded continuous-on*, *rule-format*])
have ∀$_F$ *x in at b within* {*t0*..*b*}. *x0* + *integral* {*t0*..*x*} (λ*s*. *f s* (*u s*)) = *u x*
by (*auto simp*: *eventually-at-filter u-fixed-point*)
with *fixed-point-tendsto order.refl order.refl*
have *u-tendsto*: (*u* ⟶ *x0* + *integral* {*t0* .. *b*} (λ*s*. *f s* (*u s*))) (*at b within* {*t0* .. *b*})
by (*rule filterlim-mono-eventually*)
have {*t0*..*b*} − {*b*} = {*t0*..<*b*} **by** *auto*
then have *at b within* {*t0*..*b*} ≠ *bot* **using** ‹*b* > *t0*›
unfolding *trivial-limit-within*
by (*simp add*: *islimpt-in-closure*)
then have *u b* = *x0* + *integral* {*t0*..*b*} (λ*s*. *f s* (*u s*))
using *u-at-b u-tendsto*
by (*rule tendsto-unique*)
with *u-fixed-point* **have** ⋀*s*. *t0* ≤ *s* ⟹ *s* ≤ *b* ⟹ *x0* + *integral* {*t0*..*s*} (λ*s*. *f s* (*u s*)) = *u s*
by (*case-tac s* = *b*) *auto*
with - **have** *u-vderiv*:
⋀*s*. *t0* ≤ *s* ⟹ *s* ≤ *b* ⟹ (*u has-vector-derivative f s* (*u s*)) (*at s within* {*t0* .. *b*})
by (*rule has-vector-derivative-imp*)
(*auto intro*!: *derivative-eq-intros cont integral-has-vector-derivative*)

interpret *i*:
ivp (|*ivp-f* = λ(*t*, *x*). *f t x*, *ivp-t0* = *t0*, *ivp-x0* = *x0*, *ivp-T* = {*t0*..*b*}, *ivp-X* = *X*|)
by *unfold-locales* (*auto simp*: ‹*t0* < *b*› *less-imp-le* ‹*x0* ∈ *X*›)
have *i.is-solution u*
by (*rule i.is-solutionI*; *clarsimp simp add*: *u-vderiv*)
(*auto simp*: *u-def* ‹*x0* ∈ *X*› ‹*x1* ∈ *X*› ‹*t0* < *b*› *iv-in*

intro!: *flow-in-domain in-existence-ivlI*)
with *iv-in* ‹{*t0* .. *b*} ⊆ *T*› ‹*t0* < *b*› *iv-in*
have {*t0* .. *b*} ⊆ *existence-ivl t0 x0*
by (*intro maximal-existence-flow*(*1*)[*OF iv-in*])
(*auto simp*: *is-interval-closed-interval*)
hence *b* ∈ *existence-ivl t0 x0* **using** ‹*t0* < *b*›
by *auto*
thus *False*
using *b-gtI real-less-ereal-iff*
by (*auto simp*: *existence-ivl-def* ‹*x0* ∈ *X*› *b*)
qed

lemma
sup-existence-maximal:
assumes *t0* ∈ *T x0* ∈ *X*
assumes ⋀*t*. *t0* ≤ *t* ⟹ *t* ∈ *existence-ivl t0 x0* ⟹ *flow t0 x0 t* ∈ *K*
assumes *compact K K* ⊆ *X*
assumes *sup-existence t0 x0* ≠ ∞
shows *real-of-ereal* (*sup-existence t0 x0*) ∉ *T*
using *flow-leaves-compact-ivl*[*of t0 x0 K*] *assms* **by** *force*

lemma fixes *a b c*::*ereal*
shows *not-inftyI*: *a* < *b* ⟹ *b* < *c* ⟹ *abs b* ≠ ∞
by *force*

lemma
interval-neqs:
fixes *r s t*::*real*
shows {*r*<..<*s*} ≠ {*t*<..}
and {*r*<..<*s*} ≠ {..<*t*}
and {*r*<..<*ra*} ≠ *UNIV*
and {*r*<..} ≠ {..<*s*}
and {*r*<..} ≠ *UNIV*
and {..<*r*} ≠ *UNIV*
and {} ≠ {*r*<..}
and {} ≠ {..<*r*}
subgoal by (*metis dual-order.strict-trans greaterThanLessThan-iff greaterThan-iff gt-ex not-le order-refl*)
subgoal by (*metis* (*no-types*, *hide-lams*) *greaterThanLessThan-empty-iff greaterThanLessThan-iff gt-ex lessThan-iff minus-minus neg-less-iff-less not-less order-less-irrefl*)
subgoal by *force*
subgoal by (*metis greaterThanLessThan-empty-iff greaterThanLessThan-eq greaterThan-iff inf.idem lessThan-iff lessThan-non-empty less-irrefl not-le*)
subgoal by *force*
subgoal by *force*
subgoal using *greaterThan-non-empty* **by** *blast*
subgoal using *lessThan-non-empty* **by** *blast*
done

lemma *greaterThanLessThan-eq-iff*:
fixes *r s t u::real*
shows $(\{r<..<s\} = \{t<..<u\}) = (r \geq s \wedge u \leq t \vee r = t \wedge s = u)$
by (*metis cInf-greaterThanLessThan cSup-greaterThanLessThan greaterThanLessThan-empty-iff not-le*)

lemma *real-of-ereal-image-greaterThanLessThan-iff*:
real-of-ereal ‘ $\{a <..< b\}$ = *real-of-ereal* ‘ $\{c <..< d\} \longleftrightarrow (a \geq b \wedge c \geq d \vee a = c \wedge b = d)$
unfolding *real-atLeastGreaterThan-eq*
by (*cases a*; *cases b*; *cases c*; *cases d*;
simp add: *greaterThanLessThan-eq-iff interval-neqs interval-neqs*[*symmetric*])

lemma *uminus-image-real-of-ereal-image-greaterThanLessThan*:
uminus ‘ *real-of-ereal* ‘ $\{l <..< u\}$ = *real-of-ereal* ‘ $\{-u <..< -l\}$
by (*force simp*: *algebra-simps ereal-less-uminus-reorder*
ereal-uminus-less-reorder intro: *image-eqI*[**where** $x=-x$ **for** x])

lemma *add-image-real-of-ereal-image-greaterThanLessThan*:
op $+$ *c* ‘ *real-of-ereal* ‘ $\{l <..< u\}$ = *real-of-ereal* ‘ $\{c + l <..< c + u\}$
apply *safe*
subgoal for *x*
using *ereal-less-add*[*of c*]
by (*force simp*: *real-of-ereal-add add.commute*)
subgoal for - *x*
by (*force simp*: *add.commute real-of-ereal-minus ereal-minus-less ereal-less-minus*
intro: *image-eqI*[**where** $x=x - c$])
done

lemma *add2-image-real-of-ereal-image-greaterThanLessThan*:
$(\lambda x.\ x + c)$ ‘ *real-of-ereal* ‘ $\{l <..< u\}$ = *real-of-ereal* ‘ $\{l + c <..< u + c\}$
using *add-image-real-of-ereal-image-greaterThanLessThan*[*of c l u*]
by (*metis add.commute image-cong*)

lemma *minus-image-real-of-ereal-image-greaterThanLessThan*:
op $-$ *c* ‘ *real-of-ereal* ‘ $\{l <..< u\}$ = *real-of-ereal* ‘ $\{c - u <..< c - l\}$
(**is** *?l* = *?r*)
proof −
have *?l* = *op* $+$ *c* ‘ *uminus* ‘ *real-of-ereal* ‘ $\{l <..< u\}$ **by** *auto*
also note *uminus-image-real-of-ereal-image-greaterThanLessThan*
also note *add-image-real-of-ereal-image-greaterThanLessThan*
finally show *?thesis* **by** (*simp add*: *minus-ereal-def*)
qed

lemma
inf-existence-minimal:
assumes *iv-in*: $t0 \in T$ $x0 \in X$
assumes *mem-compact*: $\bigwedge t.\ t \leq t0 \Longrightarrow t \in$ *existence-ivl t0 x0* $\Longrightarrow$ *flow t0 x0 t* $\in K$

assumes *K*: *compact K K* ⊆ *X*
assumes *inf*: *inf-existence t0 x0* ≠ − ∞
shows *real-of-ereal* (*inf-existence t0 x0*) ∉ *T*
proof −
let *?mirror* = λ*t*. *2* ∗ *t0* − *t*
interpret *rev*: *ll-on-open* λ*t*. − *f* (*?mirror t*) *?mirror* ' *T* **..**
have *rev-iv-in*: *?mirror t0* ∈ *?mirror* ' *T x0* ∈ *X* **using** *iv-in* **by** *auto*

from *rev-existence-ivl-eq*[*OF iv-in*, *unfolded rev.existence-ivl-def existence-ivl-def*]
have *real-of-ereal* ' {*rev.inf-existence t0 x0*<..<*rev.sup-existence t0 x0*} =
?mirror ' *real-of-ereal* ' {*inf-existence t0 x0*<..<*sup-existence t0 x0*}
by (*force intro*!: *image-eqI*[**where** *x*=*?mirror* (*real-of-ereal x*) **for** *x*])
also have ... = *real-of-ereal* ' {*2* ∗ *ereal t0* − *sup-existence t0 x0*<..<*2* ∗ *ereal t0* − *inf-existence t0 x0*}
unfolding *minus-image-real-of-ereal-image-greaterThanLessThan*
by *simp*
finally have *rev-bnds*: *rev.inf-existence t0 x0* = *2* ∗ *t0* − (*sup-existence t0 x0*)
rev.sup-existence t0 x0 = *2* ∗ *t0* − (*inf-existence t0 x0*)
unfolding *real-of-ereal-image-greaterThanLessThan-iff*
using *flow-eq-rev*(*2*) *iv-in*(*1*) *rev.existence-ivl-def rev-iv-in*(*2*)
by *force*+

have *rev-mem-compact*: *2* ∗ *t0* − *t0* ≤ *t* ⟹ *t* ∈ *rev.existence-ivl* (*2* ∗ *t0* − *t0*) *x0* ⟹ *rev.flow* (*2* ∗ *t0* − *t0*) *x0 t* ∈ *K* **for** *t*
using *mem-compact*[*of ?mirror t*] *flow-eq-rev*[*OF iv-in*, *of ?mirror t*] *rev-existence-ivl-eq*[*OF iv-in*, *of t*]
by *auto*
have *real-of-ereal* (*rev.sup-existence* (*2* ∗ *t0* − *t0*) *x0*) ∉ *op* − (*2* ∗ *t0*) ' *T*
using *inf*
by (*intro rev.sup-existence-maximal*[*OF rev-iv-in rev-mem-compact K*])
(*auto simp*: *rev-bnds ereal-minus-eq-PInfty-iff*)
then show *real-of-ereal* (*inf-existence t0 x0*) ∉ *T*
using *inf existence-ivl-def iv-in*(*1*) *rev-iv-in*(*2*)
by (*cases inf-existence t0 x0*) (*fastforce simp*: *rev-bnds*)+
qed

lemma *real-ereal-bound-lemma-up*:
assumes *s* ∈ *real-of-ereal* ' {*a*<..<*b*}
assumes *t* ∉ *real-of-ereal* ' {*a*<..<*b*}
assumes *s* ≤ *t*
shows *b* ≠ ∞
using *assms*
apply (*cases b*)
subgoal by *force*
subgoal by (*metis PInfty-neq-ereal*(*2*) *assms dual-order.strict-trans1 ereal-infty-less*(*1*) *ereal-less-ereal-Ex greaterThanLessThan-empty-iff greaterThanLessThan-iff greaterThan-iff image-eqI less-imp-le linordered-field-no-ub not-less order-trans real-greaterThanLessThan-infinity-eq real-image-ereal-ivl real-of-ereal.simps*(*1*))
subgoal by *force*

done
lemma *real-ereal-bound-lemma-down*:
assumes $s \in$ *real-of-ereal* ‘ $\{a<..<b\}$
assumes $t \notin$ *real-of-ereal* ‘ $\{a<..<b\}$
assumes $t \leq s$
shows $a \neq -\infty$
using *assms*
apply (*cases b*)
apply (*auto simp*: *real-greaterThanLessThan-infinity-eq*)
using *assms*(*1*) *real-greaterThanLessThan-minus-infinity-eq*
apply *auto*
done

lemma *mem-is-intervalI*:
fixes *a b c*::*real*
assumes *is-interval S*
assumes $a \in S$ $c \in S$
assumes $a \leq b$ $b \leq c$
shows $b \in S$
using *assms is-interval-1* **by** *blast*

lemma
initial-time-bounds:
assumes *iv-in*: $t0 \in T$ $x0 \in X$
shows *inf-existence t0 x0* $<$ *t0 t0* $<$ *sup-existence t0 x0*
using *existence-ivl-initial-time*[*OF iv-in*]
by (*auto simp*: *existence-ivl-def ereal-real*)

lemma
mem-compact-implies-subset-existence-interval:
assumes *iv-in*: $t0 \in T$ $x0 \in X$
assumes *mem-compact*: $\bigwedge t.\ t \in T \Longrightarrow$ *flow t0 x0 t* $\in K$
assumes *K*: *compact K* $K \subseteq X$
assumes *ivl*: *is-interval T*
shows $T \subseteq$ *existence-ivl t0 x0*
proof
fix *t* **assume** $t \in T$
have *t0* $\in$ *existence-ivl t0 x0*
by (*rule existence-ivl-initial-time*[*OF iv-in*])
have $t <$ *sup-existence t0 x0*
proof (*cases sup-existence t0 x0*)
fix *s*
assume *s*: *sup-existence t0 x0* = *ereal s*
with *sup-existence-maximal*[*OF assms*(*1*−*5*)] *mem-existence-ivl-subset*[*OF iv-in*]
have $s \notin T$
by *auto*
from *initial-time-bounds*[*OF iv-in*] *s*
have *t0* $< s$
by *simp*

then have $t < s$
using ‹$s \notin T$› *iv-in* ‹$t \in T$› *ivl*
by (*meson leI local.mem-is-intervalI not-less-iff-gr-or-eq*)
then show *?thesis* **using** *s* **by** *simp*
qed (*auto simp*: *existence-ivl-ninfty*[*OF iv-in*])
moreover
have *inf-existence t0 x0* $<$ *t*
proof (*cases inf-existence t0 x0*)
fix *i*
assume *i*: *inf-existence t0 x0* = *ereal i*
with *inf-existence-minimal*[*OF assms*(*1*−*5*)] *mem-existence-ivl-subset*[*OF iv-in*]
have $i \notin T$
by *auto*
from *initial-time-bounds*[*OF iv-in*] *i*
have $i <$ *t0* **by** *simp*
then have $i < t$
using ‹$i \notin T$› *iv-in* ‹$t \in T$› *ivl*
by (*meson is-interval-1 less-imp-le not-le*)
then show *?thesis* **using** *i* **by** *simp*
qed (*auto simp*: *existence-ivl-ninfty*[*OF iv-in*])
ultimately show $t \in$ *existence-ivl t0 x0*
by (*simp add*: *rev-image-eqI existence-ivl-def*)
qed

lemma
subset-mem-compact-implies-subset-existence-interval:
assumes *ivl*: *t0* $\in T'$ *is-interval* T' $T' \subseteq T$
assumes *iv-in*: *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': $t' \in T'$ $t' \notin$ *existence-ivl t0 x0*
by *auto*
then have $t' \leq$ *inf-existence t0 x0* $\vee$ $t' \geq$ *sup-existence t0 x0*
by (*cases sup-existence t0 x0*; *cases inf-existence t0 x0*)
(*auto simp*: *existence-ivl-def real-image-ereal-ivl split*: *if-split-asm*)
then show *False*
proof
assume t'*-le*: *ereal* $t' \leq$ *inf-existence t0 x0*
then have *ni*: *inf-existence t0 x0* $\neq -\infty$ **by** *auto*
then obtain *i* **where** *i*: *inf-existence t0 x0* = *ereal i*
using *initial-time-bounds*(*1*) *iv-in ivl*(*1*) *ivl*(*3*)
by (*cases inf-existence t0 x0*; *force*)
from *assms* **have** *t0* $\in T$ **by** *auto*
have $i \in T'$
using t'*-le i initial-time-bounds*[*OF* ‹*t0* $\in T$› *iv-in*]

by (*intro mem-is-intervalI*[*OF ivl*(*2*) *t*′(*1*) *ivl*(*1*)]) *auto*
have ∗: *t* ∈ *T*′ **if** *t* ≤ *t0* *t* ∈ *existence-ivl t0 x0* **for** *t*
using *that*(*2*)
by (*intro mem-is-intervalI*[*OF ivl*(*2*) ‹*i* ∈ *T*′› ‹*t0* ∈ *T*′› - *that*(*1*)])
(*auto simp add*: *existence-ivl-def i less-imp-le less-eq-ereal-def not-inftyI real-of-ereal-ord-simps*)
from *inf-existence-minimal*[*OF* ‹*t0* ∈ *T*› *iv-in mem-compact K ni*, *OF* ∗]
show *False* **using** ‹*i* ∈ *T*′› *ivl* **by** (*auto simp*: *i*)
next
assume *t*′*-le*: *sup-existence t0 x0* ≤ *ereal t*′
then have *ns*: *sup-existence t0 x0* ≠ ∞ **by** *auto*
then obtain *s* **where** *s*: *sup-existence t0 x0* = *ereal s*
using *initial-time-bounds*(*2*) *iv-in ivl*(*1*) *ivl*(*3*)
by (*cases sup-existence t0 x0*; *force*)
from *assms* **have** *t0* ∈ *T* **by** *auto*
have *s* ∈ *T*′
using *t*′*-le s initial-time-bounds*[*OF* ‹*t0* ∈ *T*› *iv-in*]
by (*intro mem-is-intervalI*[*OF ivl*(*2*) *ivl*(*1*) *t*′(*1*)]) *auto*

have ∗: *t* ∈ *T*′ **if** *t0* ≤ *t* *t* ∈ *existence-ivl t0 x0* **for** *t*
using *that*(*2*)
by (*intro mem-is-intervalI*[*OF ivl*(*2*) ‹*t0* ∈ *T*′› ‹*s* ∈ *T*′› *that*(*1*)])
(*auto simp add*: *existence-ivl-def s real-of-ereal-ord-simps*)
from *sup-existence-maximal*[*OF* ‹*t0* ∈ *T*› *iv-in mem-compact K ns*, *OF* ∗] ‹*s* ∈ *T*′› *ivl*
show *False* **by** (*auto simp*: *s*)
qed
qed

lemma
global-right-existence-interval:
assumes *b* ≥ *t0*
assumes *b*: *b* ∈ *existence-ivl t0 x0*
assumes *iv-in*: *t0* ∈ *T* *x0* ∈ *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* ∈ {*t0* .. *b*} ⟹
dist (*flow t0 x0 t*) (*flow t0 y t*) ≤ *dist x0 y* ∗ *exp* (*K* ∗ *abs* (*t* − *t0*))
⋀*e*. *e* > *0* ⟹
eventually (λ*y*. ∀ *t* ∈ {*t0* .. *b*}. *dist* (*flow t0 x0 t*) (*flow t0 y t*) < *e*) (*at x0*)
proof −
def *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-real iv-in*)
have *seg* ≠ {} **by** (*auto simp*: *seg-def closed-segment-real*)
moreover
have *compact seg*

using *iv-in b*
by (*auto simp*: *seg-def closed-segment-real*
intro!: *compact-continuous-image continuous-at-imp-continuous-on flow-continuous*;
metis (*erased*, *hide-lams*) *atLeastAtMost-iff closed-segment-real*
closed-segment-subset-existence-ivl contra-subsetD order.trans)
moreover note *open-domain*(*2*)
moreover have $seg \subseteq X$
using *closed-segment-subset-existence-ivl b*
by (*auto simp*: *seg-def intro*!: *flow-in-domain iv-in*)
ultimately
obtain *e* **where** *e*: $0 < e$ $\{x.\ infdist\ x\ seg \leq e\} \subseteq X$
thm *compact-in-open-separated*
by (*rule compact-in-open-separated*)
def $A \equiv \{x.\ infdist\ x\ seg \leq e\}$

have $A \subseteq X$ **using** *e* **by** (*simp add*: *A-def*)

have *mem-existence-ivlI*: $\bigwedge s.\ t0 \leq s \Longrightarrow s \leq b \Longrightarrow s \in existence\text{-}ivl\ t0\ x0$
by (*rule in-existence-between-zeroI*[*OF iv-in b*]) *auto*

have *compact A*
unfolding *A-def*
by (*rule compact-infdist-le*) *fact*+
have *compact* $\{t0\ ..\ b\}$ $\{t0\ ..\ b\} \subseteq T$
using *mem-existence-ivlI mem-existence-ivl-subset*[*OF iv-in*]
by (*auto simp add*: *compact-Times* ‹*compact A*›)
from *lipschitz-on-compact*[*OF this* ‹*compact A*› ‹$A \subseteq X$›]
obtain K' **where** $\bigwedge t.\ t \in \{t0\ ..\ b\} \Longrightarrow lipschitz\ A\ (f\ t)\ K'$
by *metis*
hence K': $\bigwedge t.\ t \in \{t0\ ..\ b\} \Longrightarrow lipschitz\ A\ (f\ t)\ (abs\ K')$
by (*rule nonneg-lipschitz*)
def $K \equiv abs\ K' + 1$
have $0 < K$ $0 \leq K$
by (*auto simp*: *K-def*)
have *K*: $\bigwedge t.\ t \in \{t0\ ..\ b\} \Longrightarrow lipschitz\ A\ (f\ t)\ K$
unfolding *K-def*
using ‹$- \Longrightarrow lipschitz\ A\ -\ K'$›
by (*rule pos-lipschitz*)

have [*simp*]: $x0 \in A$ **using** ‹$0 < e$› **by** (*auto simp*: *A-def*)

def $d \equiv min\ e\ (e * exp\ (-K * (b - t0)))$
hence *d*: $0 < d$ $d \leq e$ $d \leq e * exp\ (-K * (b - t0))$
using *e* **by** *auto*

{
fix *t* **assume** $t0 \leq t$ $t \leq b$
hence $d * exp\ (K * (t - t0)) \leq d * exp\ (K * (b - t0))$

```
    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 have d ∗ exp (K ∗ (t − t0)) ≤ e .
} note d-times-exp-le = this
have ball x0 d ⊆ X using d ‹A ⊆ X›
  by (auto simp: A-def dist-commute intro!: infdist-le2[where a=x0])
{
  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
  {
    fix t::real assume t: t0 ≤ t t ∈ existence-ivl t0 y t ≤ b
    have flow t0 y t ∈ A
    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 ivl-subset-existence-ivl t iv-in
            by (auto intro!: continuous-at-imp-continuous-on
              continuous-intros flow-continuous ‹y ∈ X›)
          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*
{
fix *s* **assume** *s*: *s* ∈ {*t0* .. *t'*}
hence *s* ∈ *closed-segment t0 b*
using ‹*t* ≤ *b*› *t'* **by** (*auto simp*: *closed-segment-real*)
hence *flow t0 x0 s* ∈ *A*
using *s* ‹*e* > *0*› **by** (*auto simp*: *seg-def A-def*)
} **note** *flow-in* = *this*
{
assume *t'* = *t0*
hence *flow t0 y t'* ∈ *A*
using *y d iv-in*
by (*auto simp*: *A-def* ‹*y* ∈ *X*› *infdist-le2*[**where** *a*=*x0*] *dist-commute*)
} **moreover** **{**
fix *s* **assume** *s*: *s* ∈ {*t0* ..< *t'*}
hence *s* ∈ *closed-segment t0 b*
using ‹*t* ≤ *b*› *t'* **by** (*auto simp*: *closed-segment-real*)
from *t'*(*1*)[*of s*]
have *t'* > *s* ⟹ *t0* ≤ *s* ⟹ *s* ≤ *t* ⟹ *e* > *infdist* (*flow t0 y s*) *seg*
by *force*
hence *flow t0 y s* ∈ *A*
using *s t'* ‹*e* > *0*› **by** (*auto simp*: *seg-def A-def*)
} **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 *ivl-subset-existence-ivl*[*OF* ‹*t0* ∈ *T*› ‹*y* ∈ *X*› *t*(*2*)] *t'* ‹*y* ∈ *X*› *iv-in*
by (*intro flow-tendsto*) (*auto intro*!: *tendsto-intros*)
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*. *s* ∈ {*t0* .. *t'*} ⟹ *flow t0 y s* ∈ *A*
by (*case-tac s* = *t'*) *auto*
have
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*
by (*auto intro*!: *flow-in flow-y-in*) *fact*+
thus *?thesis* **..**

```
qed
{
  fix s assume s: s ∈ {t0 .. t'}
  hence t0 ≤ s by simp
  have s ≤ b
    using  t t' s b
    using ivl-subset-existence-ivl
    by auto
  hence sx0: s ∈ existence-ivl t0 x0
    by (simp add: ‹t0 ≤ s› mem-existence-ivlI)
  have sy: s ∈ existence-ivl t0 y
 by (meson ‹y ∈ X› atLeastAtMost-iff contra-subsetD iv-in(1) ivl-subset-existence-ivl
     order-trans s t'(2) t(2))
  have int: flow t0 y s − flow t0 x0 s =
      y − x0 + (integral {t0 .. s} (λt. f t (flow t0 y t)) −
        integral {t0 .. s} (λt. f t (flow t0 x0 t)))
    using iv-in
    unfolding flow-fixed-point[OF ‹t0 ≤ s› sx0 iv-in]
      flow-fixed-point[OF ‹t0 ≤ s› sy ‹t0 ∈ T› ‹y ∈ X›]
    by (simp add: algebra-simps)
  have norm (flow t0 y s − flow t0 x0 s) ≤ norm (y − x0) +
    norm (integral {t0 .. s} (λt. f t (flow t0 y t)) −
      integral {t0 .. s} (λt. f t (flow t0 x0 t)))
    unfolding int
    by (rule norm-triangle-ineq)
  also
  have norm (integral {t0 .. s} (λt. f t (flow t0 y t)) −
      integral {t0 .. s} (λt. f t (flow t0 x0 t))) =
    norm (integral {t0 .. s} (λt. f t (flow t0 y t) − f t (flow t0 x0 t)))
     using ivl-subset-existence-ivl[of t0 x0 s] sx0 ivl-subset-existence-ivl[of t0
y s] sy
    by (subst integral-diff)
     (auto intro!: integrable-continuous-real continuous-at-imp-continuous-on
       f-flow-continuous ‹y ∈ X› iv-in)
 also have ... ≤ (integral {t0 .. s} (λt. norm (f t (flow t0 y t) − f t (flow
t0 x0 t))))
     using ivl-subset-existence-ivl[OF - - sx0] ivl-subset-existence-ivl[OF - -
sy]
    by (intro integral-norm-bound-integral)
     (auto intro!: integrable-continuous-real continuous-at-imp-continuous-on
       continuous-intros f-flow-continuous ‹y ∈ X› iv-in)
  also have ... ≤ (integral {t0 .. s} (λt. K ∗ norm ((flow t0 y t) − (flow
t0 x0 t))))
     using ivl-subset-existence-ivl[OF - - sx0] ivl-subset-existence-ivl[OF - -
sy]
    s t'(3,5) ‹s ≤ b›
 by (auto simp del: integral-mult-right intro!: integral-le integrable-continuous-real
     continuous-at-imp-continuous-on lipschitz-norm-leI[OF K]
     continuous-intros f-flow-continuous flow-continuous ‹y ∈ X› iv-in)
```

also have ... = *K* ∗ *integral* {*t0* .. *s*} (*λt*. *norm* (*flow t0 y t* − *flow t0 x0 t*))
using *ivl-subset-existence-ivl*[*OF* - - *sx0*] *ivl-subset-existence-ivl*[*OF* - - *sy*]
by (*subst integral-mult*)
(*auto intro*!: *integrable-continuous-real continuous-at-imp-continuous-on lipschitz-norm-leI*[*OF K*] *continuous-intros f-flow-continuous flow-continuous* ‹*y* ∈ *X*› *iv-in*)
finally
have *norm*: *norm* (*flow t0 y s* − *flow t0 x0 s*) ≤
norm (*y* − *x0*) + *K* ∗ *integral* {*t0* .. *s*} (*λt*. *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-real*)
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* ‹*t0* ∈ *T*› - - - - - - - *K′*])
subgoal by (*metis atLeastAtMost-iff local.norm-le*(*4*) *order-refl t′*(*1*))
subgoal by (*metis* ‹*y* ∈ *A*›)
subgoal by (*metis atLeastAtMost-iff local.norm-le*(*3*) *order-refl t′*(*1*))
subgoal using *e* **by** (*simp add*: *A-def*)
subgoal by *fact*
subgoal by (*metis closed-segment-real t′*(*1*,*5*))
subgoal by (*metis closed-segment-real t′*(*1*,*3*))
subgoal by (*simp add*: *closed-segment-real 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
} **note** *in-A* = *this*

have *b-in*: *b* ∈ *existence-ivl t0 y*
proof (*rule ccontr*)

assume *b* ∉ *existence-ivl t0 y*
hence *disj*: *b* ≤ *inf-existence t0 y* ∨ *sup-existence t0 y* ≤ *b*
by (*auto simp*: *existence-ivl-def ereal-infinity-cases*
ereal-less-real-iff not-le real-less-ereal-iff real-image-ereal-ivl
split: *if-split-asm*)
from *existence-ivl-initial-time*[*OF* ‹*t0* ∈ *T*› ‹*y* ∈ *X*›]
have *t0* ≤ *sup-existence t0 y*
using *ereal-le-real-iff*
by (*force simp add*: *real-image-ereal-ivl existence-ivl-def*
split: *if-split-asm*)
with *existence-ivl-initial-time*[*OF* ‹*t0* ∈ *T*› ‹*y* ∈ *X*›] ‹*t0* ≤ *b*› *disj*
have *sup-le*: *sup-existence t0 y* ≤ *b*
by (*meson* ‹*y* ∈ *X*› *ereal-less-eq*(*3*) *initial-time-bounds*(*1*) *iv-in*(*1*) *not-le order-trans*)
{
fix *t*::*real* **assume** *t*: *t0* ≤ *t t* ∈ *existence-ivl t0 y*
hence *t* < *b*
using *sup-le*
by (*auto simp*: *existence-ivl-def real-less-ereal-iff*)
(*metis less-ereal.simps*(*1*) *less-le-trans*)
note *in-A*[*OF t less-imp-le*[*OF this*]]
} **note** *in-A* = *this*
have *sup-existence t0 y* < ∞ *real-of-ereal* (*sup-existence t0 y*) ∈ *T*
subgoal
using ‹*ereal t0* ≤ *sup-existence t0 y*› *ereal-le-real-iff sup-le*
by (*force intro*!: *mem-existence-ivl-subset*[*OF iv-in*] *intro*: *mem-existence-ivlI*)
subgoal
using ‹*ereal t0* ≤ *sup-existence t0 y*› ‹{*t0*..*b*} ⊆ *T*› *ereal-le-real-iff real-le-ereal-iff sup-le*
by *fastforce*
done
from *flow-leaves-compact-ivl*[*OF* ‹*t0* ∈ *T*› ‹*y* ∈ *X*› *this* ‹*compact A*› ‹*A* ⊆ *X*›]
obtain *t* **where** *t*: *t0* ≤ *t t* ∈ *existence-ivl t0 y flow t0 y t* ∉ *A* **by** *auto*
from *in-A*[*OF t*(*1*,*2*)] *t*(*3*)
show *False*
by *simp*
qed
{
fix *t* **assume** *t*: *t* ∈ {*t0* .. *b*}
also note *ivl-subset-existence-ivl*[*OF* ‹*t0* ∈ *T*› ‹*y* ∈ *X*› *b-in*]
finally have *t-in*: *t* ∈ *existence-ivl t0 y* **.**

note *t*
also note *ivl-subset-existence-ivl*[*OF iv-in assms*(*2*)]
finally have *t-in'*: *t* ∈ *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 ivl-subset-existence-ivl*[*OF* ‹*t0* ∈ *T*› ‹*y* ∈ *X*› *b-in*] ‹*0* < *e*›

by (*intro in-A exponential-initial-condition*[*OF* ‹*t0* ∈ *T*› *t-in* ‹*y* ∈ *A*› *t-in'* ‹*x0* ∈ *A*› ‹*A* ⊆ *X*› - - *K'*])
(*auto simp*: *closed-segment-real 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*
} note ∗ = ‹*d* > *0*› ‹*K* > *0*› ‹*ball x0 d* ⊆ *X*› *this*
moreover
{
fix *e*::*real* **assume** *0* < *e*
have *eventually* (λ*y*. *y* ∈ *ball x0 d*) (*at x0*)
using ‹*d* > *0*›
by (*rule eventually-at-in-ball*)
hence *eventually* (λ*y*. ∀ *t*∈{*t0*..*b*}. *dist* (*flow t0 x0 t*) (*flow t0 y t*) ≤ *dist x0 y* ∗ *exp* (*K* ∗ |*t* − *t0*|)) (*at x0*)
by *eventually-elim* (*safe intro*!: ∗)
moreover
have *eventually* (λ*y*. ∀ *t*∈{*t0*..*b*}. *dist x0 y* ∗ *exp* (*K* ∗ |*t* − *t0*|) ≤ *dist x0 y* ∗ *exp* (*K* ∗ (*b* − *t0*))) (*at x0*)
using ‹*t0* ≤ *b*› ‹*0* < *K*›
by (*auto intro*!: *mult-left-mono always-eventually*)
moreover
have *eventually* (λ*y*. *dist x0 y* ∗ *exp* (*K* ∗ (*b* − *t0*)) < *e*) (*at x0*)
using ‹*0* < *e*› **by** (*auto intro*!: *order-tendstoD tendsto-eq-intros*)
ultimately
have *eventually* (λ*y*. ∀ *t*∈{*t0*..*b*}. *dist* (*flow t0 x0 t*) (*flow t0 y t*) < *e*) (*at x0*)
by *eventually-elim force*
}
ultimately show *?thesis* **..**
qed

lemma
global-left-existence-interval:
assumes *b* ≤ *t0*
assumes *b*: *b* ∈ *existence-ivl t0 x0*
assumes *iv-in*: *t0* ∈ *T x0* ∈ *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*))
⋀*e*. *e* > *0* ⟹ *eventually* (λ*y*. ∀ *t* ∈ {*b* .. *t0*}. *dist* (*flow t0 x0 t*) (*flow t0 y t*) < *e*) (*at x0*)
proof −
let *?mirror* = λ*t*. *2* ∗ *t0* − *t*
have *t0'*: *t0* ∈ *?mirror* ‘ *T* **using** *iv-in* **by** *auto*
interpret *rev*: *ll-on-open* (λ*t*. − *f* (*?mirror t*)) *?mirror* ‘ *T* **..**
from *assms* **have** *2* ∗ *t0* − *b* ≥ *t0 2* ∗ *t0* − *b* ∈ *rev.existence-ivl t0 x0*

by (*auto simp*: *flow-eq-rev*)
from *rev.global-right-existence-interval*[*OF this t0′* ‹*x0* ∈ *X*›]
obtain *d K* **where** *dK*: *d* > *0 K* > *0*
ball x0 d ⊆ *X*
⋀*y*. *y* ∈ *ball x0 d* ⟹ *2* ∗ *t0* − *b* ∈ *rev.existence-ivl t0 y*
⋀*t y*. *y* ∈ *ball x0 d* ⟹ *t* ∈ {*t0* .. *2* ∗ *t0* − *b*} ⟹ *dist* (*rev.flow t0 x0 t*) (*rev.flow t0 y t*) ≤ *dist x0 y* ∗ *exp* (*K* ∗ *abs* (*t* − *t0*))
⋀*e*. *e* > *0* ⟹ *eventually* (λ*y*. ∀ *t* ∈ {*t0* .. *2* ∗ *t0* − *b*}. *dist* (*rev.flow t0 x0 t*) (*rev.flow t0 y t*) < *e*) (*at x0*)
by (*auto simp*: *rev-flow-eq* ‹*x0* ∈ *X*›)
from *dK*(*3*,*4*) **have** ⋀*y*. *y* ∈ *ball x0 d* ⟹ *?mirror* (*?mirror b*) ∈ *existence-ivl t0 y*
by (*subst rev-existence-ivl-eq*[*symmetric*]) (*auto simp*: *iv-in*)
then have *4*: ⋀*y*. *y* ∈ *ball x0 d* ⟹ *b* ∈ *existence-ivl t0 y* **by** *simp*
{
fix *t y* **assume** *yt*: *y* ∈ *ball x0 d t* ∈ {*b* .. *t0*}
with *dK*(*3*) **have** *yx0*: *y* ∈ *X x0* ∈ *ball x0 d* **using** ‹*d* > *0*› **by** *auto*
from *yt yx0 rev.closed-segment-subset-existence-ivl*[*OF t0′* - *dK*(*4*)[*OF yt*(*1*)]]
have *2* ∗ *t0* − *t* ∈ *rev.existence-ivl t0 y*
by (*auto simp*: *closed-segment-real*)
moreover
from *yt* ‹*x0* ∈ *X*› *rev.closed-segment-subset-existence-ivl*[*OF t0′* - *dK*(*4*)[*OF* ‹*x0* ∈ *ball x0 d*›]]
have *2* ∗ *t0* − *t* ∈ *rev.existence-ivl t0 x0*
by (*auto simp*: *closed-segment-real*)
ultimately
have *dist* (*flow t0 x0 t*) (*flow t0 y t*) ≤ *dist x0 y* ∗ *exp* (*K* ∗ *abs* (*t* − *t0*))
using *yt dK*(*5*)[*of y 2* ∗ *t0* − *t*] *rev-flow-eq*[*OF iv-in*, *of 2* ∗ *t0* − *t*]
rev-flow-eq[*OF* ‹*t0* ∈ *T*› ‹*y* ∈ *X*›, *of 2* ∗ *t0* − *t*]
by (*auto simp*: *dist-commute closed-segment-real*)
} **note** *5* = *this*
{
fix *e*::*real* **assume** *0* < *e*
have *eventually* (λ*y*. *y* ∈ *ball x0 d*) (*at x0*)
using ‹*d* > *0*› **by** (*rule eventually-at-in-ball*)
hence *eventually* (λ*y*. ∀ *t*∈{*t0*..*2* ∗ *t0* − *b*}. *dist* (*rev.flow t0 x0 t*) (*rev.flow t0 y t*)
= *dist* (*flow t0 x0* (*2* ∗ *t0* − *t*)) (*flow t0 y* (*2* ∗ *t0* − *t*))) (*at x0*)
proof *eventually-elim*
case (*elim y*)
hence *y* ∈ *X 2* ∗ *t0* −*b* ∈ *rev.existence-ivl t0 y* **using** *dK* **by** *auto*
from *rev.closed-segment-subset-existence-ivl*[*OF t0′ this*]
rev.closed-segment-subset-existence-ivl[*OF t0′* ‹*x0* ∈ *X*› ‹*2* ∗ *t0* − *b* ∈ *rev.existence-ivl t0 x0*›]
show *?case*
by (*force simp*: *iv-in* ‹*y* ∈ *X*› *closed-segment-real rev-flow-eq*)
qed
moreover
note *dK*(*6*)[*OF* ‹*0* < *e*›]

ultimately
have *eventually* (λy. $\forall$ $t\in\{b$ .. $t0\}$. *dist* (*flow t0 x0 t*) (*flow t0 y t*) $<$ *e*) (*at x0*)
by *eventually-elim* (*auto simp*: *dest*: *bspec*[**where** $x=2 * t0 - t$ **for** *t*])
} **note** $6 = this$
from *dK*(*1−3*) *4 5 6* **show** *?thesis* **..**
qed

lemma
global-existence-interval:
assumes *a*: $a \in$ *existence-ivl t0 x0*
assumes *b*: $b \in$ *existence-ivl t0 x0*
assumes *le*: $a \leq b$
assumes *iv-in*: $t0 \in T$ $x0 \in X$
obtains *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*) $\leq$ *dist x0 y* $*$ *exp* ($K *$ *abs* ($t - t0$))
$\bigwedge e$. $e > 0 \Longrightarrow$
eventually (λy. $\forall$ $t \in \{a$ .. $b\}$. *dist* (*flow t0 x0 t*) (*flow t0 y t*) $<$ *e*) (*at x0*)
proof −
def $r \equiv$ *Max* {*t0*, *a*, *b*}
def $l \equiv$ *Min* {*t0*, *a*, *b*}
have *r*: $r \geq t0$ $r \in$ *existence-ivl t0 x0*
using *a b* **by** (*auto simp*: *max-def iv-in r-def*)
obtain *dr Kr* **where** *right*:
$0 < dr$ $0 < Kr$ *ball x0 dr* $\subseteq X$
$\bigwedge y$. $y \in$ *ball x0 dr* $\Longrightarrow r \in$ *existence-ivl t0 y*
$\bigwedge y$ t. $y \in$ *ball x0 dr* $\Longrightarrow t \in \{t0..r\} \Longrightarrow$ *dist* (*flow t0 x0 t*) (*flow t0 y t*) $\leq$ *dist x0 y* $*$ *exp* ($Kr * |t - t0|$)
$\bigwedge e$. $0 < e \Longrightarrow \forall_F$ *y in at x0*. $\forall$ $t\in\{t0..r\}$. *dist* (*flow t0 x0 t*) (*flow t0 y t*) $<$ *e*
by (*rule global-right-existence-interval*[*OF r iv-in*]) *blast*

have *l*: $l \leq t0$ $l \in$ *existence-ivl t0 x0*
using *a b* **by** (*auto simp*: *min-def iv-in l-def*)
obtain *dl Kl* **where** *left*:
$0 < dl$ $0 < Kl$ *ball x0 dl* $\subseteq X$
$\bigwedge y$. $y \in$ *ball x0 dl* $\Longrightarrow l \in$ *existence-ivl t0 y*
$\bigwedge y$ t. $y \in$ *ball x0 dl* $\Longrightarrow t \in \{l$ .. $t0\} \Longrightarrow$ *dist* (*flow t0 x0 t*) (*flow t0 y t*) $\leq$ *dist x0 y* $*$ *exp* ($Kl * |t - t0|$)
$\bigwedge e$. $0 < e \Longrightarrow \forall_F$ *y in at x0*. $\forall$ $t\in\{l$ .. $t0\}$. *dist* (*flow t0 x0 t*) (*flow t0 y t*) $<$ *e*
by (*rule global-left-existence-interval*[*OF l iv-in*]) *blast*

def $d \equiv$ *min dr dl*
def $K \equiv$ *max Kr Kl*

have $0 < d$ $0 < K$ *ball x0 d* $\subseteq X$
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*
ivl-subset-existence-ivl'[*OF* ‹*t0* ∈ *T*› *this left*(*4*)]
ivl-subset-existence-ivl[*OF* ‹*t0* ∈ *T*› *this right*(*4*)]
have *a* ∈ *existence-ivl t0 y b* ∈ *existence-ivl t0 y*
by (*auto simp*: *d-def l-def r-def min-def max-def 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
} moreover {
fix *e*::*real* **assume** *0* < *e*
from *left*(*6*)[*OF* ‹*0* < *e*›] *right*(*6*)[*OF* ‹*0* < *e*›]
have *eventually* (λ*y*. ∀ *t* ∈ {*a* .. *b*}. *dist* (*flow t0 x0 t*) (*flow t0 y t*) < *e*) (*at x0*)
by *eventually-elim* (*auto simp*: *l-def r-def min-def max-def*)
} ultimately show *?thesis* **..**
qed

lemma
assumes *t0*: *t0* ∈ *T*
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
with 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|)
  ⋀eps. 0 < eps ⟹
    ∀F y in at x. ∀t∈{t − e..t + e}. dist (flow t0 x t) (flow t0 y t) < eps
  by (rule global-existence-interval[OF ivl t0 ‹x ∈ X›]) 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-real dist-real-def)
  with ‹y ∈ X› s closed-segment-subset-existence-ivl[OF t0 - dK(4)[OF y]]
    closed-segment-subset-existence-ivl[OF t0 - 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
{
  fix eps :: real assume eps > 0
  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 dK(7)[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*) *eventually-at-shift-zero*[*symmetric*]) *simp*
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 t0* ‹*x* ∈ *X*› *t*, *unfolded isCont-def*, *THEN tendstoD*, *OF* ‹*eps* > *0*›]
by (*subst* (*asm*) *eventually-at-shift-zero*[*symmetric*]) (*auto simp*: *dist-norm*)
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
have ∀$_F$ *s in at 0*. *norm* (*flow t0* (*x* + *fst s*) (*t* + *snd s*) − *flow t0 x t*) < *2* ∗ *eps*
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
} **note** ∗∗ = *this*
{
fix *eps*::*real* **assume** *eps* > *0*
hence *eps* / *2* > *0* **by** *simp*
from ∗∗[*OF this*]
have ∗: ∀$_F$ *s in at 0*. *norm* (*flow t0* (*x* + *fst s*) (*t* + *snd s*) − *flow t0 x t*) < *eps*
by *auto*
} **note** ∗ = *this*
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

lemma *flow-isCont-state-space*: *t0* ∈ *T* ⟹ *x* ∈ *X* ⟹ *t* ∈ *existence-ivl t0 x* ⟹ *isCont* (λ(*x*, *t*). *flow t0 x t*) (*x*, *t*)
using *flow-continuous-on-state-space*

by (*auto simp*: *continuous-on-eq-continuous-within at-within-open*[*OF - open-state-space*])

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

lemma *existence-ivl-eq-domain*:
assumes *iv-in*: $t0 \in T$ $x0 \in X$
assumes *bnd*: $\bigwedge tm\ tM\ t\ x.\ tm \in T \Longrightarrow tM \in T \Longrightarrow \exists M.\ \exists L.\ \forall t \in \{tm\ ..\ tM\}.\ \forall x \in X.\ norm\ (f\ t\ x) \le M + L * norm\ x$
assumes *is-interval T X* = *UNIV*
shows *existence-ivl t0 x0* = T
proof −
from *assms* **have** *XI*: $x \in X$ **for** x **by** *auto*
{
fix *tm tM* **assume** *tm*: $tm \in T$ **and** *tM*: $tM \in T$ **and** *tmtM*: $tm \le t0$ $t0 \le tM$
from *bnd*[*OF tm tM*] **obtain** M' L'
where *bnd'*: $\bigwedge x\ t.\ x \in X \Longrightarrow tm \le t \Longrightarrow t \le tM \Longrightarrow norm\ (f\ t\ x) \le M' + L' * norm\ x$
by *force*
def $M \equiv norm\ M' + 1$
def $L \equiv norm\ L' + 1$
have *bnd*: $\bigwedge x\ t.\ x \in X \Longrightarrow tm \le t \Longrightarrow t \le tM \Longrightarrow norm\ (f\ t\ x) \le M + L * norm\ x$
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 $?r = (norm\ x0 + |tm - tM| * M + 1) * exp\ (L * |tm - tM|)$
def $K \equiv cball\ (0::'a)\ ?r$
have *K*: *compact K* $K \subseteq X$
by (*auto simp*: *K-def* ‹X = *UNIV*›)
{
fix t **assume** t: $t \in$ *existence-ivl t0 x0* **and** *le*: $tm \le t$ $t \le tM$
{
fix s **assume** *sc*: $s \in$ *closed-segment t0 t*
then have s: $s \in$ *existence-ivl t0 x0* **and** *le*: $tm \le s$ $s \le tM$ **using** *t le sc*
using *closed-segment-subset-existence-ivl iv-in*(*1*) *iv-in*(*2*)
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-real split*: *if-split-asm*)
from *flow-fixed-point''*[*OF s iv-in*]
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 iv-in s*] *le*
by (*auto simp*: *closed-segment-real*
intro!: *norm-triangle-le norm-triangle-ineq4* [*THEN order.trans*]
integral-norm-bound-integral bnd
integrable-continuous-closed-segment
integrable-continuous-real
continuous-intros
continuous-on-subset[*OF flow-continuous-on*]
iv-in flow-in-domain
mem-existence-ivl-subset[*OF iv-in*(*1*) *XI*])
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 iv-in* ‹*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 iv-in 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 iv-in)
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-closed-interval*)

then have *inf-existence t0 x0* < *tm* ∧ *tM* < *sup-existence t0 x0*
using *tmtM*
by (*cases inf-existence t0 x0*; *cases sup-existence t0 x0*)
(*auto simp*: *existence-ivl-def real-image-ereal-ivl subset-iff split*: *if-split-asm*)
} **note** *bnds* = *this*[*THEN conjunct2*] *this*[*THEN conjunct1*]

show *existence-ivl t0 x0* = *T*
proof *safe*
fix *x* **assume** *x*: *x* ∈ *T*
have *inf-existence t0 x0* < *x*
apply (*cases x* ≤ *t0*)
subgoal by (*rule bnds*[*OF x iv-in*(*1*)]) *simp-all*
subgoal by (*meson XI ereal-less-eq*(*3*) *initial-time-bounds*(*1*) *iv-in*(*1*) *le-cases not-less order-trans*)
done
moreover have *x* < *sup-existence t0 x0*
apply (*cases x* ≥ *t0*)
subgoal by (*rule bnds*[*OF iv-in*(*1*) *x*]) *simp-all*
subgoal by (*meson XI dual-order.strict-trans ereal-less-eq*(*3*) *initial-time-bounds*(*2*) *iv-in*(*1*) *not-less*)
done
ultimately show *x* ∈ *existence-ivl t0 x0*
by (*cases inf-existence t0 x0*; *cases sup-existence t0 x0*)
(*auto simp*: *existence-ivl-def real-atLeastGreaterThan-eq*)
qed (*insert existence-ivl-subset*[*OF iv-in*], *fastforce*)
qed

lemma *flow-unique*:
assumes *iv-in*: *t0* ∈ *T x0* ∈ *X*
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*
proof −
interpret *u*: *unique-solution existence-ivp t0 x0*
using *iv-in* **by** (*rule existence-ivp*)
have *t* ∈ *u.T* **using** *assms* **by** *auto*
show *?thesis*
unfolding *flow-def*
apply (*rule u.unique-solution*[*OF* - ‹*t* ∈ *u.T*›, *symmetric*])
apply (*rule u.is-solutionI*)
subgoal by (*force simp add*: *assms*)
subgoal by (*subst at-within-open*) (*simp-all add*: *assms*)
subgoal by (*simp add*: *assms*)
done
qed

end — *local-lipschitz T X f*

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

lemma *e-pos*: *0* < *e*
using *le-less-trans*[*OF norm-ge-zero F-G-norm-ineq*[*OF t0-in-J x0-in-X*]]
by *assumption*

definition *XX t* = *F.flow 0 x0 t*
definition *Y t* = *G.flow 0 x0 t*

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

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

have *t0-t-in-J*: {*0*..*t*} ⊆ *J*
using ‹*t* ∈ *J*› ‹*0* ∈ *J*› *J-ivl*
using *G.mem-is-intervalI atLeastAtMost-iff subsetI* **by** *blast*

note *F-G-flow-cont*[*continuous-intros*] =
continuous-on-subset[*OF F.flow-continuous-on*[*OF t0-in-T1 x0-in-X*]]
continuous-on-subset[*OF G.flow-continuous-on*[*OF t0-in-T2 x0-in-X*]]

have *?u t* + *e*/*K* ≤ *e*/*K* ∗ *exp*(*K* ∗ *t*)
proof(*rule gronwall*[**where** *g*=λ*t*. *?u t* + *e*/*K*, *OF* - - - - *K-pos* ‹*0* ≤ *t*› *order.refl*])

fix *s* **assume** $0 \leq s$ $s \leq t$
hence $\{0..s\} \subseteq J$ **using** *t0-t-in-J* **by** *auto*

hence *t0-s-in-existence*:
$\{0..s\} \subseteq$ *F.existence-ivl 0 x0*
$\{0..s\} \subseteq$ *G.existence-ivl 0 x0*
using *J-subset tG* ‹$0 \leq s$› ‹$s \leq t$› *G.ivl-subset-existence-ivl*[*OF t0-in-T2 x0-in-X tG*]
by *auto*

hence *s-in-existence*:
$s \in$ *F.existence-ivl 0 x0*
$s \in$ *G.existence-ivl 0 x0*
using ‹$0 \leq s$› **by** *auto*

note *cont-statements*[*continuous-intros*] =
x0-in-X
t0-in-T1 t0-in-T2
F.flow-in-domain[*OF t0-in-T1 x0-in-X*]
G.flow-in-domain[*OF t0-in-T2 x0-in-X*]
F.mem-existence-ivl-subset[*OF t0-in-T1 x0-in-X*]
G.mem-existence-ivl-subset[*OF t0-in-T2 x0-in-X*]

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

have *XX s* $-$ *Y s* = *integral* $\{0..s\}$ ($\lambda s.$ *F s* (*XX s*) $-$ *G s* (*Y s*))
by (*simp add*: *XX-def Y-def integral-diff integrable-on-simps*
F.flow-fixed-point[*OF* ‹$0 \leq s$› *s-in-existence*(*1*) *t0-in-T1 x0-in-X*]
G.flow-fixed-point[*OF* ‹$0 \leq s$› *s-in-existence*(*2*) *t0-in-T2 x0-in-X*])
also have ... = *integral* $\{0..s\}$ ($\lambda s.$ (*F s* (*XX s*) $-$ *F s* (*Y s*)) + (*F s* (*Y s*) $-$ *G s* (*Y s*)))
by *simp*
also have ... = *integral* $\{0..s\}$ ($\lambda s.$ *F s* (*XX s*) $-$ *F s* (*Y s*)) + *integral* $\{0..s\}$ ($\lambda s.$ *F s* (*Y s*) $-$ *G s* (*Y s*))
by (*simp add*: *integral-diff integral-add XX-def Y-def integrable-on-simps*)
finally have *?u s* $\leq$ *norm* (*integral* $\{0..s\}$ ($\lambda s.$ *F s* (*XX s*) $-$ *F s* (*Y s*))) + *norm* (*integral* $\{0..s\}$ ($\lambda s.$ *F s* (*Y s*) $-$ *G s* (*Y s*)))
by (*simp add*: *norm-triangle-ineq*)
also have ... $\leq$ *integral* $\{0..s\}$ ($\lambda s.$ *norm* (*F s* (*XX s*) $-$ *F s* (*Y s*))) + *integral* $\{0..s\}$ ($\lambda s.$ *norm* (*F s* (*Y s*) $-$ *G s* (*Y s*)))
using *t0-s-in-existence*
by (*auto simp add*: *XX-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 ∀ *x*∈{*0..s*}. *norm* (*F x* (*XX x*) − *F x* (*Y x*)) ≤ *K* ∗ *norm* (*XX x* − *Y x*)
using *F-lipschitz*[*unfolded lipschitz-def*, *THEN conjunct2*]
cont-statements(*1*,*2*,*4*)
t0-s-in-existence
by (*metis F-lipschitz XX-def Y-def* ‹{*0..s*} ⊆ *J*› *lipschitz-norm-leI ll-on-open.flow-in-domain subsetCE t0-in-T2 two-ll-on-open-axioms two-ll-on-open-def*)
show ∀ *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* **by** *fastforce*
qed (*simp-all add*: *t0-s-in-existence continuous-intros integrable-on-simps XX-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 integral-add XX-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* (*XX t* − *Y t*) + *e* / *K*)
using *assms t0-t-in-J J-subset G.ivl-subset-existence-ivl*[*OF t0-in-T2 x0-in-X tG*]
by (*auto simp add*: *XX-def Y-def intro*!: *continuous-intros*)
next
fix *s* **assume** *0* ≤ *s s* ≤ *t*
show *0* ≤ *norm* (*XX 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*[*OF t0-in-T1 x0-in-X*]]
continuous-on-subset[*OF G.flow-continuous-on*[*OF t0-in-T2 x0-in-X*]]

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*

‹$t \leq 0$›])
fix s **assume** $t \leq s$ $s \leq 0$
hence $\{s..0\} \subseteq J$ **using** *t0-t-in-J* **by** *auto*

hence *t0-s-in-existence*:
$\{s..0\} \subseteq F.existence\text{-}ivl\ 0\ x0$
$\{s..0\} \subseteq G.existence\text{-}ivl\ 0\ x0$
using *J-subset G.ivl-subset-existence-ivl'*[*OF t0-in-T2 x0-in-X tG*] ‹$s \leq 0$› ‹$t \leq s$›
by *auto*

hence *s-in-existence*:
$s \in F.existence\text{-}ivl\ 0\ x0$
$s \in G.existence\text{-}ivl\ 0\ x0$
using ‹$s \leq 0$› **by** *auto*

note *cont-statements*[*continuous-intros*] =
x0-in-X
t0-in-T1 t0-in-T2
F.flow-in-domain[*OF t0-in-T1 x0-in-X*]
G.flow-in-domain[*OF t0-in-T2 x0-in-X*]
F.mem-existence-ivl-subset[*OF t0-in-T1 x0-in-X*]
G.mem-existence-ivl-subset[*OF t0-in-T2 x0-in-X*]
then have [*continuous-intros*]:
$\{s..0\} \subseteq T1$
$\{s..0\} \subseteq T2$
$F.flow\ 0\ x0$ ‘ $\{s..0\} \subseteq X$
$G.flow\ 0\ x0$ ‘ $\{s..0\} \subseteq X$
$s \leq x \Longrightarrow x \leq 0 \Longrightarrow x \in F.existence\text{-}ivl\ 0\ x0$
$s \leq x \Longrightarrow x \leq 0 \Longrightarrow x \in G.existence\text{-}ivl\ 0\ x0$ **for** x
using *t0-s-in-existence*
by (*auto simp*:)
have $XX\ s - Y\ s = -$ *integral* $\{s..0\}$ ($\lambda s.\ F\ s\ (XX\ s) - G\ s\ (Y\ s)$)
using *t0-s-in-existence*
by (*simp add*: *XX-def Y-def*
F.flow-fixed-point'[*OF* ‹$s \leq 0$› *s-in-existence*(*1*) *t0-in-T1 x0-in-X*]
G.flow-fixed-point'[*OF* ‹$s \leq 0$› *s-in-existence*(*2*) *t0-in-T2 x0-in-X*]
continuous-intros integrable-on-simps integral-diff)
also have ... $= -$ *integral* $\{s..0\}$ ($\lambda s.\ (F\ s\ (XX\ s) - F\ s\ (Y\ s)) + (F\ s\ (Y\ s) - G\ s\ (Y\ s))$)
by *simp*
also have ... $= -$ (*integral* $\{s..0\}$ ($\lambda s.\ F\ s\ (XX\ s) - F\ s\ (Y\ s)$) + *integral* $\{s..0\}$ ($\lambda s.\ F\ s\ (Y\ s) - G\ s\ (Y\ s)$))
using *t0-s-in-existence*
by (*subst integral-add*) (*simp-all add*: *integral-add XX-def Y-def continuous-intros integrable-on-simps*)
finally have *?u* $s \leq$ *norm* (*integral* $\{s..0\}$ ($\lambda s.\ F\ s\ (XX\ s) - F\ s\ (Y\ s)$)) + *norm* (*integral* $\{s..0\}$ ($\lambda s.\ F\ s\ (Y\ s) - G\ s\ (Y\ s)$))
by (*metis* (*no-types*, *lifting*) *norm-minus-cancel norm-triangle-ineq*)

also have ... ≤ *integral* {*s*..*0*} (λ*s*. *norm* (*F s* (*XX 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*: *XX-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 ∀ *x*∈{*s*..*0*}. *norm* (*F x* (*XX x*) − *F x* (*Y x*)) ≤ *K* ∗ *norm* (*XX x* − *Y x*)
by (*metis F-lipschitz XX-def Y-def* ‹{*s*..*0*} ⊆ *J*› *cont-statements*(*4*) *cont-statements*(*5*)
lipschitz-norm-leI subset-iff t0-s-in-existence(*1*) *t0-s-in-existence*(*2*))
show ∀ *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*)
by *fastforce*
qed (*simp-all add*: *t0-s-in-existence continuous-intros integrable-on-simps XX-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 integral-add t0-s-in-existence continuous-intros integrable-on-simps XX-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* (*XX t* − *Y t*) + *e* / *K*)
using *assms t0-t-in-J J-subset G.ivl-subset-existence-ivl′*[*OF t0-in-T2 x0-in-X tG*]
by (*auto simp add*: *XX-def Y-def intro*!: *continuous-intros*)
next
fix *s* **assume** *t* ≤ *s s* ≤ *0*
show *0* ≤ *norm* (*XX 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

locale *auto-ll-on-open* = — TODO: how to guarantee that this theory is always complete?!
fixes *f*::′*a*::{*banach*, *heine-borel*} ⇒ ′*a* **and** *X*
assumes *local-lipschitz*: *local-lipschitz UNIV X* (λ-::*real*. *f*)
assumes *open-domain*[*intro*!, *simp*]: *open X*
begin

sublocale *na*: *ll-on-open* λ-. *f UNIV X*
by *standard* (*auto simp*: *intro*!: *continuous-on-const local-lipschitz*)

lemma *continuous-on-f*[*continuous-intros*]:
assumes *continuous-on S h*
assumes *h ‘ S* $\subseteq$ *X*
shows *continuous-on S* ($\lambda x.$ *f* (*h x*))
by (*rule na.continuous-on-f*[*OF continuous-on-const assms*]) *simp*

lemma *auto-ll-on-open-rev*[*intro*, *simp*]: *auto-ll-on-open* ($-f$) *X*
proof *standard*
have *range uminus* = (*UNIV*::*real set*) **by** (*auto intro*!: *image-eqI*[**where** *x*=− *x* **for** *x*])
with *na.ll-on-open-rev*[*of 0*] **interpret** *rev*: *ll-on-open* $\lambda t.$ − *f UNIV X*
by *auto*
from *rev.local-lipschitz* **show** *local-lipschitz UNIV X* (λ-::*real*. − *f*) **.**
qed *simp*

context fixes *x0*::*'a* — initial value
begin

definition *inf-existence* = *na.inf-existence 0 x0*

definition *sup-existence* = *na.sup-existence 0 x0*

definition *existence-ivl* = *na.existence-ivl 0 x0*

lemma *open-existence-ivl*[*simp*]: *open* (*existence-ivl*)
by (*simp add*: *existence-ivl-def*)

lemma *is-interval-existence-ivl*[*simp*]: *is-interval existence-ivl*
by (*simp add*: *existence-ivl-def*)

definition *flow t* = *na.flow 0 x0 t*

lemma *Picard-iterate-mem-existence-ivlI*:
assumes *0* $\leq$ *t*
assumes *compact C x0* $\in$ *C C* $\subseteq$ *X*
assumes $\bigwedge$*y s*. *0* $\leq$ *s* $\Longrightarrow$ *s* $\leq$ *t* $\Longrightarrow$ *y 0* = *x0* $\Longrightarrow$ *y* $\in$ {*0* .. *s*} $\rightarrow$ *C* $\Longrightarrow$ *continuous-on* {*0* .. *s*} *y* $\Longrightarrow$
x0 + *integral* {*0* .. *s*} ($\lambda t.$ *f* (*y t*)) $\in$ *C*
shows *t* $\in$ *existence-ivl* $\bigwedge$*s*. *0* $\leq$ *s* $\Longrightarrow$ *s* $\leq$ *t* $\Longrightarrow$ *flow s* $\in$ *C*
unfolding *existence-ivl-def flow-def*
by (*blast intro*!: *na.Picard-iterate-mem-existence-ivlI*[*OF*
UNIV-I set-mp[*OF* ‹*C* $\subseteq$ *X*› ‹*x0* $\in$ *C*›] *assms*(*1*) *subset-UNIV assms*(*2*−*5*)])+

context assumes *iv-in*: *x0* $\in$ *X* **begin**

lemma *existence-ivl-zero*[*intro*, *simp*]: *0* ∈ *existence-ivl*
  unfolding *existence-ivl-def*
  by (*rule na.existence-ivl-initial-time*[*OF UNIV-I iv-in*])

lemma *in-existence-between-zeroI*:
  t ∈ *existence-ivl* ⟹ *s* ∈ {*t* .. *0*} ∪ {*0* .. *t*} ⟹ *s* ∈ *existence-ivl*
  unfolding *existence-ivl-def*
  by (*rule na.in-existence-between-zeroI*[*OF UNIV-I iv-in*])

lemma *ivl2-subset-existence-ivl*:
  s ∈ *existence-ivl* ⟹ *t* ∈ *existence-ivl* ⟹ {*s* .. *t*} ⊆ *existence-ivl*
  unfolding *existence-ivl-def*
  by (*rule na.ivl2-subset-existence-ivl*[*OF UNIV-I iv-in*])

lemma *flow-in-domain*: *t* ∈ *existence-ivl* ⟹ *flow t* ∈ *X*
  by (*simp add*: *existence-ivl-def flow-def iv-in na.flow-in-domain*)

lemma *flow-zero*[*simp*]: *flow 0* = *x0*
  by (*simp add*: *flow-def iv-in*)

lemma *flow-has-derivative*:
  assumes *t* ∈ *existence-ivl*
  shows (*flow has-derivative* (λ*i*. *i* $*_R$ *f* (*flow t*))) (*at t*)
  using *assms*
  by (*auto simp add*: *existence-ivl-def flow-def*[*abs-def*] *iv-in intro*!: *na.flow-has-derivative*)

end — *x0* ∈ *X*

end — *x0*

lemma
  assumes *t* ∈ *na.existence-ivl s x*
  assumes *x* ∈ *X*
  shows *mem-existence-ivl-shift-autonomous1*: *t* − *s* ∈ *existence-ivl x*
    and *flow-shift-autonomous1*: *na.flow s x t* = *flow x* (*t* − *s*)
proof −
  from *na.existence-ivp*[*OF UNIV-I* ‹*x* ∈ *X*›]
  interpret *s*: *unique-solution na.existence-ivp s x* **.**

  let *?T* = (*op* + (− *s*) ‘ *na.existence-ivl s x*)
  have *shifted*: *is-interval ?T 0* ∈ *?T*
    using *na.existence-ivl-initial-time*[*OF UNIV-I* ‹*x* ∈ *X*›]
    by (*auto*)

  def *i* ≡ (|*ivp-f* = λ(*t*, *y*). *f y*, *ivp-t0* = *0*, *ivp-x0* = *x*, *ivp-T* = *?T*, *ivp-X* = *X*|)
  interpret *i*: *ivp i*
    by *unfold-locales* (*auto simp*: *i-def* ‹*x* ∈ *X*›)

  from *s.shift-autonomous-solution*[*OF s.is-solution-solution refl*, **where** *j*=*i*]

have *i.is-solution* (λ*x*. *s.solution* (*x* + *s*)) **by** (*simp add*: *i-def o-def*)

from *na.maximal-existence-flow*[*OF UNIV-I* ‹*x* ∈ *X*› *this*, *unfolded i-def*, *OF refl shifted*]
have ∗: *?T* ⊆ *existence-ivl x*
and ∗∗: ⋀*t*. *t* ∈ *op* + (− *s*) ‘ *na.existence-ivl s x* ⟹ *flow x t* = *s.solution* (*t* + *s*)
by (*auto simp*: *existence-ivl-def flow-def*)

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

have *flow x* (*t* − *s*) = *s.solution t*
using ‹*t* ∈ -›
by (*auto simp*: ∗∗ *existence-ivl-def*)
also have ... = *na.flow s x t*
unfolding *na.flow-def* **..**
finally show *na.flow s x t* = *flow x* (*t* − *s*) **..**
qed

lemma
assumes *t* − *s* ∈ *existence-ivl x*
assumes *x* ∈ *X*
shows *mem-existence-ivl-shift-autonomous2*: *t* ∈ *na.existence-ivl s x*
and *flow-shift-autonomous2*: *na.flow s x t* = *flow x* (*t* − *s*)
proof −
from *na.existence-ivp*[*OF UNIV-I* ‹*x* ∈ *X*›]
interpret *s*: *unique-solution na.existence-ivp 0 x* **.**

let *?T* = (*op* + *s* ‘ *na.existence-ivl 0 x*)
have *shifted*: *is-interval ?T s* ∈ *?T*
using *na.existence-ivl-initial-time*[*OF UNIV-I* ‹*x* ∈ *X*›]
by *auto*

def *i* ≡ (|*ivp-f* = λ(*t*, *y*). *f y*, *ivp-t0* = *s*, *ivp-x0* = *x*, *ivp-T* = *?T*, *ivp-X* = *X*|)
interpret *i*: *ivp i*
by *unfold-locales* (*auto simp*: *i-def* ‹*x* ∈ *X*›)

from *s.shift-autonomous-solution*[*OF s.is-solution-solution refl*, **where** *j*=*i*]
have *i.is-solution* (λ*x*. *s.solution* (*x* − *s*)) **by** (*simp add*: *i-def o-def*)

from *na.maximal-existence-flow*[*OF UNIV-I* ‹*x* ∈ *X*› *this*, *unfolded i-def*, *OF refl shifted*]
have ∗: *?T* ⊆ *na.existence-ivl s x*
and ∗∗: ⋀*t*. *t* ∈ *op* + *s* ‘ *existence-ivl x* ⟹ *na.flow s x t* = *s.solution* (*t* − *s*)
by (*auto simp*: *existence-ivl-def flow-def*)

```
  have t ∈ ?T
    using ‹t - s ∈ -›
    by (force simp: existence-ivl-def)
  also note *
  finally show t ∈ na.existence-ivl s x .

  have na.flow s x t = s.solution (t - s)
    using ‹t - s ∈ -›
    by (subst **; force)
  also have ... = flow x (t - s)
    unfolding flow-def na.flow-def ..
  finally show na.flow s x t = flow x (t - s) .
qed

lemma
  assumes s: s ∈ existence-ivl x0
  assumes t: t ∈ existence-ivl (flow x0 s)
  assumes iv-in[simp]: x0 ∈ X
  shows flow-trans: flow x0 (s + t) = flow (flow x0 s) t
    and existence-ivl-trans: s + t ∈ existence-ivl x0
proof -
 from na.flow-trans[OF s[unfolded existence-ivl-def] - UNIV-I iv-in, OF mem-existence-ivl-shift-autonomous2,
of t]
  have flow x0 (s + t) = na.flow s (flow x0 s) (s + t)
    using t na.flow-in-domain[OF UNIV-I iv-in s[unfolded existence-ivl-def]]
    by (auto simp: flow-def existence-ivl-def)
  also have ... = flow (flow x0 s) t
    by (subst flow-shift-autonomous2) (auto intro!: flow-in-domain s t)
  finally show flow x0 (s + t) = flow (flow x0 s) t .

  from na.existence-ivl-trans[OF s[unfolded existence-ivl-def] - UNIV-I iv-in, OF
mem-existence-ivl-shift-autonomous2, of t]
  show s + t ∈ existence-ivl x0
    using assms flow-in-domain
    by (auto simp: flow-def existence-ivl-def)
qed

lemma
  assumes t: t ∈ existence-ivl x0
  assumes [simp]: x0 ∈ X
  shows flows-reverse: flow (flow x0 t) (- t) = x0
    and existence-ivl-reverse: -t ∈ existence-ivl (flow x0 t)
proof -
  from na.existence-ivl-reverse[OF t[unfolded existence-ivl-def] UNIV-I ‹x0 ∈ X›,
 THEN mem-existence-ivl-shift-autonomous1]
    flow-in-domain[OF ‹x0 ∈ X›] t
  show -t ∈ existence-ivl (flow x0 t)
    by (auto simp: existence-ivl-def flow-def)
```

with *na.flows-reverse*[*OF t*[*unfolded existence-ivl-def*] *UNIV-I* ‹*x0* ∈ *X*›] *flow-in-domain*[*OF* ‹*x0* ∈ *X*›]
show *flow* (*flow x0 t*) (− *t*) = *x0*
by (*subst* (*asm*) *flow-shift-autonomous2*) (*auto simp*: *flow-def t*)
qed

lemma *flow-has-vector-derivative*:
assumes *x* ∈ *X t* ∈ *existence-ivl x*
shows (*flow x has-vector-derivative f* (*flow x t*)) (*at t*)
using *na.flow-has-vector-derivative*[*of 0 x t*] *assms*
by (*simp add*: *flow-def*[*abs-def*] *existence-ivl-def*)

lemma *flow-has-vector-derivative-at-0*:
assumes *x* ∈ *X t* ∈ *existence-ivl x*
shows ((λ*h*. *flow x* (*t* + *h*)) *has-vector-derivative f* (*flow x t*)) (*at 0*)
using *na.flow-has-vector-derivative-at-0*[*of 0 x t*] *assms*
by (*simp add*: *flow-def*[*abs-def*] *existence-ivl-def*)

lemma
assumes *in-domain*: *x* ∈ *X*
assumes *t* ∈ *existence-ivl x*
shows *ivl-subset-existence-ivl*: {*0* .. *t*} ⊆ *existence-ivl x*
and *ivl-subset-existence-ivl′*: {*t* .. *0*} ⊆ *existence-ivl x*
and *closed-segment-subset-existence-ivl*: *closed-segment 0 t* ⊆ *existence-ivl x*
using *assms*
by (*auto simp*: *closed-segment-real*
intro!: *in-existence-between-zeroI*[*OF* ‹*x* ∈ *X*› ‹*t* ∈ -›])

lemma *flow-fixed-point*:
assumes *t*: *0* ≤ *t t* ∈ *existence-ivl x*
assumes *x* ∈ *X*
shows *flow x t* = *x* + *integral* {*0*..*t*} (λ*t*. *f* (*flow x t*))
using *assms*
unfolding *flow-def existence-ivl-def*
by (*intro na.flow-fixed-point*; *simp*)

lemma *flow-fixed-point′*:
assumes *t*: *t* ≤ *0 t* ∈ *existence-ivl x*
assumes *x* ∈ *X*
shows *flow x t* = *x* − *integral* {*t*..*0*} (λ*t*. *f* (*flow x t*))
using *assms*
unfolding *flow-def existence-ivl-def*
by (*intro na.flow-fixed-point′*; *simp*)

lemma *flow-fixed-point′′*:
assumes *t*: *t* ∈ *existence-ivl x*
assumes *x* ∈ *X*
shows *flow x t* =
x + (*if 0* ≤ *t then 1 else* −*1*) $*_R$ *integral* (*closed-segment 0 t*) (λ*t*. *f* (*flow x t*))

using *assms*
unfolding *flow-def existence-ivl-def*
by (*intro na.flow-fixed-point''*; *simp*)

lemma *flow-continuous*: $x \in X \Longrightarrow t \in$ *existence-ivl* $x \Longrightarrow$ *continuous* (*at t*) (*flow x*)
by (*metis has-derivative-continuous flow-has-derivative*)

lemma *flow-tendsto*: $x \in X \Longrightarrow t \in$ *existence-ivl* $x \Longrightarrow$ (*ts* $\longrightarrow$ *t*) $F \Longrightarrow$
((λs. *flow x* (*ts s*)) $\longrightarrow$ *flow x t*) F
unfolding *existence-ivl-def flow-def*
by (*metis na.flow-tendsto UNIV-I*)

lemma *flow-continuous-on*: $x \in X \Longrightarrow$ *continuous-on* (*existence-ivl x*) (*flow x*)
unfolding *existence-ivl-def flow-def*[*abs-def*]
by (*metis na.flow-continuous-on UNIV-I*)

lemma *flow-continuous-on-intro*:
$x \in X \Longrightarrow$
continuous-on s g $\Longrightarrow$
($\bigwedge xa$. $xa \in s \Longrightarrow g\ xa \in$ *existence-ivl x*) $\Longrightarrow$
continuous-on s (λxa. *flow x* (*g xa*))
unfolding *existence-ivl-def flow-def*[*abs-def*]
by (*metis na.flow-continuous-on-intro UNIV-I*)

lemma *f-flow-continuous*:
assumes $t \in$ *existence-ivl* x $x \in X$
shows *isCont* (λt. *f* (*flow x t*)) *t*
using *assms*
unfolding *flow-def existence-ivl-def*
by (*intro na.f-flow-continuous*; *simp*)

lemma *exponential-initial-condition*:
assumes *y0*: $t \in$ *existence-ivl y0* **and** *y0* $\in Y$
assumes *z0*: $t \in$ *existence-ivl z0* **and** *z0* $\in Y$
assumes $Y \subseteq X$
assumes *remain*: $\bigwedge s$. $s \in$ *closed-segment 0 t* $\Longrightarrow$ *flow y0 s* $\in Y$
$\bigwedge s$. $s \in$ *closed-segment 0 t* $\Longrightarrow$ *flow z0 s* $\in Y$
assumes *lipschitz*: $\bigwedge s$. $s \in$ *closed-segment 0 t* $\Longrightarrow$ *lipschitz Y f K*
shows *norm* (*flow y0 t* $-$ *flow z0 t*) $\leq$ *norm* (*y0* $-$ *z0*) $*$ *exp* ((*K* + *1*) $*$ *abs t*)
using *assms*
unfolding *flow-def existence-ivl-def*
by (*intro order-trans*[*OF na.exponential-initial-condition*]) *auto*

lemma
existence-ivl-cballs:
fixes *x* **assumes** $x \in X$
obtains *t u L*
where

$\bigwedge y.\ y \in cball\ x\ u \Longrightarrow cball\ 0\ t \subseteq existence\text{-}ivl\ y$
$\bigwedge s\ y.\ y \in cball\ x\ u \Longrightarrow s \in cball\ 0\ t \Longrightarrow flow\ y\ s \in cball\ y\ u$
lipschitz (*cball 0 t*×*cball x u*) (λ(*t*, *x*). *flow x t*) *L*
$\bigwedge y.\ y \in cball\ x\ u \Longrightarrow cball\ y\ u \subseteq X$
$0 < t\ 0 < u$
unfolding *flow-def existence-ivl-def*
using *na.existence-ivl-cballs*[*OF UNIV-I assms*]
by *metis*

lemma
flow-leaves-compact-ivl:
assumes $x0 \in X$
assumes $sup\text{-}existence\ x0 < \infty$
assumes *compact K*
assumes $K \subseteq X$
obtains *t* **where** $t \geq 0$ $t \in existence\text{-}ivl\ x0$ $flow\ x0\ t \notin K$
unfolding *flow-def existence-ivl-def*
using *na.flow-leaves-compact-ivl*[*OF UNIV-I assms*(*1*) *assms*(*2*)[*unfolded sup-existence-def*]
UNIV-I assms(*3*−*4*)]
by *metis*

lemma
global-existence-interval:
assumes *a*: $a \in existence\text{-}ivl\ x0$
assumes *b*: $b \in existence\text{-}ivl\ x0$
assumes *le*: $a \leq b$
assumes *x0*: $x0 \in X$
obtains *d K* **where** $d > 0$ $K > 0$
$ball\ x0\ d \subseteq X$
$\bigwedge y.\ y \in ball\ x0\ d \Longrightarrow a \in existence\text{-}ivl\ y$
$\bigwedge y.\ y \in ball\ x0\ d \Longrightarrow b \in existence\text{-}ivl\ y$
$\bigwedge t\ y.\ y \in ball\ x0\ d \Longrightarrow t \in \{a\ ..\ b\} \Longrightarrow$
$dist\ (flow\ x0\ t)\ (flow\ y\ t) \leq dist\ x0\ y * exp\ (K * abs\ t)$
$\bigwedge e.\ e > 0 \Longrightarrow$
$eventually\ (\lambda y.\ \forall t \in \{a\ ..\ b\}.\ dist\ (flow\ x0\ t)\ (flow\ y\ t) < e)\ (at\ x0)$
unfolding *flow-def existence-ivl-def*
using *na.global-existence-interval*[*OF assms*(*1*−*3*)[*unfolded flow-def existence-ivl-def*]
UNIV-I x0]
by *auto*

lemma *open-state-space*: *open* (*Sigma X existence-ivl*)
and *flow-continuous-on-state-space*:
continuous-on (*Sigma X existence-ivl*) (λ(*x*, *t*). *flow x t*)
using *na.open-state-space na.flow-continuous-on-state-space*
by (*auto simp*: *existence-ivl-def flow-def*)

lemma *flow-isCont-state-space*: $x \in X \Longrightarrow t \in existence\text{-}ivl\ x \Longrightarrow isCont\ (\lambda(x, t).\ flow\ x\ t)\ (x,\ t)$
using *na.flow-isCont-state-space*

by (*auto simp*: *existence-ivl-def flow-def*)

lemma *flow-continuous-on-state-space-comp*[*continuous-intros*]:
assumes *continuous-on Y h continuous-on Y g*
assumes $\bigwedge y.\ y \in Y \Longrightarrow h\ y \in X$
assumes $\bigwedge y.\ y \in Y \Longrightarrow g\ y \in$ *existence-ivl* (*h y*)
shows *continuous-on Y* (λ*y*. *flow* (*h y*) (*g y*))
using *assms continuous-on-compose2*[**where** *f*=λ*y*. (*h y*, *g y*) **and** *s* = *Y*, *OF flow-continuous-on-state-space*]
by (*auto intro*!: *continuous-intros*)

end — *local-lipschitz UNIV X* (λ-. *f*)

locale *compact-continuously-diff* =
derivative-on-prod T X f λ(*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* ≠ {} *X* ≠ {}
assumes *continuous-derivative*: *continuous-on X f′*
begin

lemma
f-comp-derivative[*derivative-intros*]:
assumes *t* ∈ *T x* ∈ *X*
shows ((λ*a*. *f* (*t*, *a*)) *has-derivative blinfun-apply* (*f′ x*)) (*at x within X*)
proof −
have (*f o* (λ*a*. (*t*, *a*)) *has-derivative blinfun-apply* (*f′ x*)) (*at x within X*)
by (*auto intro*!: *derivative-eq-intros refl has-derivative-within-subset*[*OF f′*] *assms simp*: *split-beta′*)
thus *?thesis* **by** (*simp add*: *o-def*)
qed

lemma *ex-onorm-bound*:
∃ *B*. ∀ *x* ∈ *X*. *norm* (*f′ x*) ≤ *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*. ∀ *x* ∈ *X*. *norm* (*f′ x*) ≤ *B*)

lemma *onorm-bound*: **assumes** *x* ∈ *X* **shows** *norm* (*f′ x*) ≤ *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 lipschitzI*)
fix *t z y*
assume $t \in T$ $y \in X$ $z \in X$
then have *norm* (*f* (*t*, *y*) − *f* (*t*, *z*)) ≤ *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*)) ≤ *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* ≤ *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 i* +
compact-continuously-diff T X f
for *i*::*'a*::{*banach*,*perfect-space*,*heine-borel*} *ivp*
begin

sublocale *unique-on-closed i t1 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*:
⋀*x*. *x* ∈ *X* ⟹ (*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* ⟶ *l*) *F*
and *l* ∈ *X*
shows ((λ*x*. *f'* (*g x*)) ⟶ *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 ⟹
    ((λx. f (g x)) has-derivative
      (λxa. 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(t, x). f x 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 lipschitz ?X f onorm-bound
    using lipschitz[of t] uv
    by auto
  thus ∃u>0. ∃L. ∀t ∈ cball t u ∩ UNIV. lipschitz (cball x u ∩ X) f L
    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 *XX x0* = *flow x0*
definition *A x0 t* = *f'* (*XX x0 t*)

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-ivl* (*a s*)
  shows *continuous-on S* (λ*s*. *A* (*a s*) (*b s*))
proof −
  have *continuous-on S* (λ*x*. *f'* (*flow* (*a x*) (*b x*)))
    by (*auto intro*!: *continuous-intros assms flow-in-domain*)
  then show *?thesis*
    by (*rule continuous-on-eq*) (*auto simp*: *assms A-def XX-def*)
qed

context
  fixes *x0*::'*a*
  assumes *x0-def*[*continuous-intros*]: *x0* ∈ *X*
begin

lemma *XX-defined*: *xa* ∈ *existence-ivl x0* ⟹ *XX x0 xa* ∈ *X*
  by (*auto simp*: *XX-def flow-in-domain x0-def*)

lemma *continuous-on-XX*: *continuous-on* (*existence-ivl x0*) (*XX x0*)
  by (*auto simp*: *XX-def intro*!: *continuous-intros*)

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

interpretation *var*: *ll-on-open A x0 existence-ivl x0 UNIV*
  by *standard*
   (*auto intro*!: *c1-implies-local-lipschitz*[**where** *f'* = λ(*t*, *x*). *A x0 t*] *continuous-intros*
      derivative-eq-intros
      simp: *split-beta' blinfun.bilinear-simps*)

lemma *varexivl-eq-exivl*:
  assumes *t* ∈ *existence-ivl x0*
  shows *var.existence-ivl t a* = *existence-ivl x0*
proof (*rule var.existence-ivl-eq-domain*)
  fix *s t x*
  assume *s*: *s* ∈ *existence-ivl x0* **and** *t*: *t* ∈ *existence-ivl x0*
  then have {*s* .. *t*} ⊆ *existence-ivl x0*
    by (*intro ivl2-subset-existence-ivl*[*OF x0-def*])
  then have *continuous-on* {*s* .. *t*} (*A x0*)
    by (*auto simp*: *closed-segment-real intro*!: *continuous-intros*)
  then have *compact* ((*A x0*) ‘ {*s* .. *t*})
    using *compact-Icc*

```
    by (rule compact-continuous-image)
  then obtain B where B: ⋀u. u ∈ {s .. t} ⟹ norm (A x0 u) ≤ B
    by (force dest!: compact-imp-bounded simp: bounded-iff)
  show ∃M L. ∀t∈{s .. t}. ∀x∈UNIV. norm (blinfun-apply (A 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)
qed (auto intro: assms)

definition U u0 t = var.flow 0 u0 t

definition Y z t = flow (x0 + z) t
```

Linearity of the solution to the variational equation. TODO: generalize for arbitrary linear ODEs

```
lemma U-linear:
assumes t ∈ existence-ivl x0
shows U (α ∗R a + β ∗R b) t = α ∗R U a t + β ∗R U b t
  unfolding U-def
proof (rule var.maximal-existence-flow[OF - - - refl is-interval-existence-ivl[of x0]])
  note x0-def[intro, simp]
  interpret c: ivp
    (|ivp-f = λ(t, x). blinfun-apply (A x0 t) x,
      ivp-t0 = 0,
      ivp-x0 = α ∗R a + β ∗R b,
      ivp-T = existence-ivl x0,
      ivp-X = UNIV|)
    by unfold-locales auto
  show c.is-solution (λc. α ∗R var.flow 0 a c + β ∗R var.flow 0 b c)
  proof (rule c.is-solutionI)
    show α ∗R var.flow 0 a c.t0 + β ∗R var.flow 0 b c.t0 = c.x0
      by simp
  next
    fix t assume t ∈ c.T
    hence t ∈ existence-ivl x0 by simp
    with at-within-open[OF this open-existence-ivl]
    show ((λc. α ∗R var.flow 0 a c + β ∗R var.flow 0 b c) has-vector-derivative
          c.f (t, α ∗R var.flow 0 a t + β ∗R var.flow 0 b t))
          (at t within c.T)
      by (auto intro!: derivative-eq-intros var.flow-has-vector-derivative
        simp: blinfun.bilinear-simps varexivl-eq-exivl)
    show α ∗R var.flow 0 a t + β ∗R var.flow 0 b t ∈ c.X
      by simp
  qed
qed (auto intro!: x0-def assms)

lemma linear-U:
assumes t ∈ existence-ivl x0
shows linear (λz. U z t)
```

using *U-linear*[*OF assms*, *of 1 - 1*] *U-linear*[*OF assms*, *of - - 0*]
by (*auto intro*!: *linearI*)

lemma *bounded-linear-U*:
assumes $t \in$ *existence-ivl x0*
shows *bounded-linear* (λz. *U z t*)
by (*simp add*: *linear-linear linear-U assms*)

lemma *U-continuous-on-time*: *continuous-on* (*existence-ivl x0*) (λt. *U z t*)
unfolding *U-def*
using *var.flow-continuous-on*[*of 0 z*]
by (*auto simp*: *x0-def varexivl-eq-exivl*)

lemma *proposition-17-6-weak*:
— from "Differential Equations, Dynamical Systems, and an Introduction to Chaos", Hirsch/Smale/Devaney
assumes $t \in$ *existence-ivl x0*
shows (λy. (*Y* (*y* − *x0*) *t* − *XX x0 t* − *U* (*y* − *x0*) *t*) $/_R$ *norm* (*y* − *x0*)) − *x0* → *0*
proof−
have *0* ∈ *existence-ivl x0*
by (*simp add*: *x0-def*)

Find some $J \subseteq$ *existence-ivl x0* with *0* ∈ *J* and *t* ∈ *J*.

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

have *t0* ≤ *0 0* ≤ *t1 0* ∈ *J J* ≠ {} *t* ∈ *J compact J*
and *J-in-existence*: *J* ⊆ *existence-ivl 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* ∈ *X S* ⊆ *existence-ivl* (*x0* + *z*)
have *continuous-on S* (*Y z*)
using *flow-continuous-on assms*(*1*)
by (*intro continuous-on-subset*[*OF - assms*(*2*)]) (*simp add*: *Y-def*)
}
note [*continuous-intros*] = *this integrable-continuous-real blinfun.continuous-on*

have *U-continuous*[*continuous-intros*]: ⋀*z*. *continuous-on J* (*U z*)
by(*rule continuous-on-subset*[*OF U-continuous-on-time J-in-existence*])

from ‹*t* ∈ *J*›
have *t0* ≤ *t*
and *t* ≤ *t1*
and *t0* ≤ *t1*

and *t0* ∈ *existence-ivl x0*
and *t* ∈ *existence-ivl x0*
and *t1* ∈ *existence-ivl x0*
using *J-def J-in-existence* **by** *auto*
from *global-existence-interval*[*OF* ‹*t0* ∈ *existence-ivl x0*› ‹*t1* ∈ *existence-ivl x0*› ‹*t0* ≤ *t1*› *x0-def*]
obtain *u K* **where** *uK-def*:
0 < *u*
0 < *K*
ball x0 u ⊆ *X*
⋀*y*. *y* ∈ *ball x0 u* ⟹ *t0* ∈ *existence-ivl y*
⋀*y*. *y* ∈ *ball x0 u* ⟹ *t1* ∈ *existence-ivl y*
⋀*t y*. *y* ∈ *ball x0 u* ⟹ *t* ∈ *J* ⟹ *dist* (*XX x0 t*) (*Y* (*y* − *x0*) *t*) ≤ *dist x0 y* ∗ *exp* (*K* ∗ |*t*|)
⋀*e*. *0* < *e* ⟹ ∀$_F$ *y in at x0*. ∀ *t*∈*J*. *dist* (*XX x0 t*) (*Y* (*y* − *x0*) *t*) < *e*
by (*auto simp add*: *J-def XX-def Y-def*)

have *J-in-existence-ivl*: ⋀*y*. *y* ∈ *ball x0 u* ⟹ *J* ⊆ *existence-ivl y*
unfolding *J-def*
using *uK-def*
by (*intro ivl2-subset-existence-ivl*) *auto*
have *ball-in-X*: ⋀*z*. *z* ∈ *ball 0 u* ⟹ *x0* + *z* ∈ *X*
using *uK-def*(*3*)
by (*auto simp*: *dist-norm*)

have *XX-J-props*: *XX x0* ‘ *J* ≠ {} *compact* (*XX x0* ‘ *J*) *XX x0*‘ *J* ⊆ *X*
using ‹*t0* ≤ *t1*›
using *J-def*(*1*) *J-in-existence*
by (*auto simp add*: *J-def XX-def intro*!:
compact-continuous-image continuous-intros flow-in-domain)

have [*continuous-intros*]: *continuous-on J* (λ*s*. *f'* (*XX x0 s*))
using *J-in-existence*
by (*auto intro*!: *continuous-intros flow-in-domain simp*: *XX-def*)

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 0* < *dist x x0 dist x x0* < *u*
hence *x* ∈ *X*
using *uK-def*(*3*)
by (*auto simp*: *dist-commute*)
hence *inverse* (*norm* (*x* − *x0*)) ∗ *norm* (*Y* (*x* − *x0*) *0* − *XX x0 0* − *U* (*x* − *x0*) *0*) = *0*

using *x0-def*
by (*simp add*: *XX-def Y-def U-def*)
thus *inverse* (*norm* (*x* − *x0*)) ∗ *norm* (*Y* (*x* − *x0*) *0* − *XX x0 0* − *U* (*x* − *x0*) *0*) < *e*
using ‹*0* < *e*› **by** *auto*
qed
next
assume *t* ≠ *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′* (*XX x0 s*)) |*s*. *s* ∈ *J* } ≤ *N* (**is** *Sup ?S* ≤ *N*)
and *N-gr-0*: *0* < *N*
— We need N to be an upper bound of {*norm* (*f′* (*XX x0 s*)) |*s*. *s* ∈ *J*}, but also larger than zero.
by (*meson le-cases less-le-trans linordered-field-no-ub*)
have *N-ineq*: ⋀*s*. *s* ∈ *J* ⟹ *norm* (*f′* (*XX x0 s*)) ≤ *N*
proof−
fix *s* **assume** *s* ∈ *J*
have *?S* = (*norm o f′ o XX x0*) ‘ *J* **by** *auto*
moreover have *continuous-on J* (*norm o f′ o XX x0*)
using *J-in-existence*
by (*auto intro*!: *continuous-intros*)
ultimately have ∃ *a b*. *?S* = {*a*..*b*} ∧ *a* ≤ *b*
using *continuous-image-closed-interval*[*OF* ‹*t0* ≤ *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′* (*XX x0 s*)) ≤ *Sup ?S*
by *auto*
thus *norm* (*f′* (*XX x0 s*)) ≤ *N*
using *N-ge-SupS* **by** *simp*
qed

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

from *compact-in-open-separated*[*OF XX-J-props*(*1*,*2*) *open-domain XX-J-props*(*3*)]
obtain *e-domain* **where** *e-domain-def*: *0* < *e-domain* {*x*. *infdist x* (*XX x0* ‘ *J*) ≤ *e-domain*} ⊆ *X*
by *auto*
def *G*≡{*x*∈*X*. *infdist x* (*XX x0* ‘ *J*) < *e-domain*}
have *G-vimage*: *G* = ((λ*x*. *infdist x* (*XX 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.

```
  def e-domain2 ≡ e-domain / 2
  have e-domain2 > 0 e-domain2 < e-domain using ‹e-domain > 0›
    by (simp-all add: e-domain2-def)
  def H≡{x. infdist x (XX x0 ‘ J) ≤ e-domain2}
  have H-props: H ≠ {} compact H H ⊆ G
    proof−
      have x0 ∈ XX x0 ‘ J
        unfolding image-iff
        using XX-def ‹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› XX-J-props
        by (intro compact-infdist-le) simp-all
    next
      show H ⊆ G
      proof
        fix x assume x ∈ H

        from ‹x ∈ H›
        have infdist x (XX x0 ‘ J) < e-domain
          using ‹0 < e-domain›
          by (simp add: H-def e-domain2-def)
        moreover from this have x ∈ X
          using e-domain-def(2)
          by auto
        ultimately show x ∈ G
          unfolding G-def
          by auto
      qed
    qed

  have f'-cont-on-G: (⋀x. x ∈ G ⟹ isCont f' x)
  using continuous-on-interior[OF continuous-on-subset[OF continuous-derivative
‹G ⊆ X›]]
    by (simp add: interior-open[OF ‹open G›])
```

def *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* ⊆ *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* ∈ *H* ⟹ *y* ∈ *H* ⟹ *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 *uK-def*(*7*)[*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* (*XX x0 t*) (*Y* (*x* − *x0*) *t*) < *min* (*e-domain* / *2*) *d-taylor*)
by (*auto simp*: *dist-norm*)

def *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 flow starting in *ball x0 d* will be in G.

have *Y-in-G*: ⋀*y*. *y* ∈ *ball x0 d* ⟹ (λ*s*. *Y* (*y* − *x0*) *s*) ‘ *J* ⊆ *G*
proof
fix *x y* **assume** *assms*: *y* ∈ *ball x0 d x* ∈ (λ*s*. *Y* (*y* − *x0*) *s*) ‘ *J*
show *x* ∈ *G*
proof(*cases*)
assume *y* = *x0*
from *assms*(*2*)
have *x* ∈ *XX x0* ‘ *J*

by (*simp add*: *XX-def Y-def* ‹*y* = *x0*›)
thus *x* ∈ *G*
using ‹*0* < *e-domain*› ‹*XX 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-XX-Y*: ⋀*t*. *t* ∈ *J* ⟹ *dist* (*XX x0 t*) (*Y* (*y* − *x0*) *t*) < *e-domain*
by (*auto simp*: *XX-def Y-def 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*) *Y-def*
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 *XX x0 t* ∈ *XX x0* ‘ *J* **using** *t-def* **by** *auto*
from *dist-XX-Y*[*OF t-def*(*1*)]
have *dist x* (*XX x0 t*) < *e-domain*
by (*simp add*: *t-def*(*2*) *dist-commute*)
from *le-less-trans*[*OF infdist-le*[*OF* ‹*XX x0 t* ∈ *XX 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*: *XX x0* ‘ *J* ⊆ *G*
by (*simp add*: *XX-def Y-def*)

show ∃ *d*>*0*. ∀ *x*. *0* < *dist x x0* ∧ *dist x x0* < *d* ⟶
dist ((*Y* (*x* − *x0*) *t* − *XX x0 t* − *U* (*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-ivl 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 *XX* and *Y*.

have *dist-XX-Y*: ⋀*s*. *s* ∈ *J* ⟹ *dist* (*XX 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*: *XX-def Y-def J-def intro*!: *mult-mono*)
from *d-ivl-def x-x0-dist* ‹*d* ≤ *d-ivl*›
have *dist-XX-Y2*: ⋀*t*. *t* ∈ *J* ⟹ *dist* (*XX x0 t*) (*Y* (*x* − *x0*) *t*) < *min* (*e-domain2*) *d-taylor*
by (*auto simp*: *XX-def Y-def e-domain2-def*)

let *?g* = λ*t*. *norm* (*Y* (*x* − *x0*) *t* − *XX x0 t* − *U* (*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* ∈ *J*. We can then apply Grönwall's inequality to obtain a true bound for *?g*.

{
fix *s a b* **assume** *s-def*: *s* ∈ {*a*..*b*}
and *J'-def*: {*a*..*b*} ⊆ *J*
and *ab-cases*: (*a* = *0* ∧ *b* = *s*) ∨ (*a* = *s* ∧ *b* = *0*)
hence *s* ∈ *J* **by** *auto*

have *s-in-existence-ivl-x0*: *s* ∈ *existence-ivl x0*
using *J-in-existence* ‹*s* ∈ *J*› **by** *auto*
have *s-in-existence-ivl*: ⋀*y*. *y* ∈ *ball x0 u* ⟹ *s* ∈ *existence-ivl y*
using *J-in-existence-ivl* ‹*s* ∈ *J*› **by** *auto*
have *s-in-existence-ivl2*: ⋀*z*. *z* ∈ *ball 0 u* ⟹ *s* ∈ *existence-ivl* (*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* (*XX x0*)

```
    apply(rule continuous-on-subset[OF - J-in-existence])
    using flow-continuous-on[OF x0-def]
    by (simp add: XX-def)

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

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

  have [continuous-intros]: ⋀z. continuous-on J (λs. f′ (XX x0 s) (U z s))
  proof−
    fix z
    have a1: continuous-on J (XX x0)
      unfolding XX-def
          by (rule continuous-on-subset[OF flow-continuous-on[OF x0-def]
J-in-existence])

    have a2: (λs. (XX x0 s, U z s)) ‘ J ⊆ (XX x0 ‘ J) × ((λs. U z s) ‘ J)
      by auto
    have a3: continuous-on ((λs. (XX x0 s, U z s)) ‘ J) (λ(x, u). f′ x u)
      using assms
      by (intro continuous-on-subset[OF - a2])
        (auto intro!: tendsto-eq-intros blinfun.tendsto
            simp: split-beta′ flow-in-domain[OF x0-def J-in-existence[THEN
subsetD]] XX-def
            continuous-on-def)
   from continuous-on-compose[OF continuous-on-Pair[OF a1 U-continuous]
a3]
    show continuous-on J (λs. f′ (XX x0 s) (U z s))
      by simp
  qed

  have [continuous-intros]: continuous-on J (λs. R (XX 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 (*intro continuous-on-compose-Pair*[*OF taylor-expansion*(*4*)])
(*auto intro*!: *continuous-intros simp*: *dist-commute*)
hence [*continuous-intros*]:
(*λs. R* (*XX x0 s*) (*Y* (*x* − *x0*) *s*)) *integrable-on J*
unfolding *J-def*
by (*rule integrable-continuous-real*)

have *i1*: *integral* {*a..b*} (*λs. f* (*Y* (*x* − *x0*) *s*)) − *integral* {*a..b*} (*λs. f* (*XX x0 s*)) =
integral {*a..b*} (*λs. f* (*Y* (*x* − *x0*) *s*) − *f* (*XX x0 s*))
using *J-in-existence-ivl*[*OF x-in-ball*]
by (*intro integral-diff*[*symmetric*]) (*auto intro*!: *continuous-intros*)

have *i2*:
integral {*a..b*} (*λs. f* (*Y* (*x* − *x0*) *s*) − *f* (*XX x0 s*) − (*f'* (*XX x0 s*)) (*U* (*x* − *x0*) *s*)) =
integral {*a..b*} (*λs. f* (*Y* (*x* − *x0*) *s*) − *f* (*XX x0 s*)) −
integral {*a..b*} (*λs. f'* (*XX x0 s*) (*U* (*x* − *x0*) *s*))
using *J-in-existence-ivl*[*OF x-in-ball*]
by (*intro integral-diff*[*OF integrable-diff*]) (*auto intro*!: *continuous-intros*)

from *ab-cases*
have *?g s* = *norm* (*integral* {*a..b*} (*λs'. f* (*Y* (*x* − *x0*) *s'*)) − *integral* {*a..b*} (*λs'. f* (*XX x0 s'*)) − *integral* {*a..b*} (*λs'.* (*f'* (*XX x0 s'*)) (*U* (*x* − *x0*) *s'*)))
proof(*safe*)
assume *a* = *0 b* = *s*
hence *0* ≤ *s* **using** ‹*s* ∈ {*a..b*}› **by** *simp*

Integral equations for XX, Y and U.

have *XX-integral-eq*: *XX x0 s* = *x0* + *integral* {*0..s*} (*λs. f* (*XX x0 s*))
unfolding *XX-def*
by (*rule flow-fixed-point*[*OF* ‹*0* ≤ *s*› *s-in-existence-ivl-x0 x0-def*])
have *Y-integral-eq*: *Y* (*x* − *x0*) *s* = *x0* + (*x* − *x0*) + *integral* {*0..s*} (*λs. 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*: *Y-def*)
have *U-integral-eq*: *U* (*x* − *x0*) *s* = (*x* − *x0*) + *integral* {*0..s*} (*λs. f'* (*XX x0 s*) (*U* (*x* − *x0*) *s*))
unfolding *U-def A-def*[*symmetric*]
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*} (*λs'. f* (*Y* (*x* − *x0*) *s'*)) − *integral* {*0..s*} (*λs'. f* (*XX x0 s'*)) −
integral {*0..s*} (*λs'. blinfun-apply* (*f'* (*XX x0 s'*)) (*U* (*x* − *x0*) *s'*)))
by (*simp add*: *XX-integral-eq Y-integral-eq U-integral-eq*)
next
assume *a* = *s b* = *0*

hence $s \leq 0$ **using** ‹$s \in \{a..b\}$› **by** *simp*

have *XX-integral-eq-left*: $XX\ x0\ s = x0 - integral\ \{s..0\}\ (\lambda s.\ f\ (XX\ x0\ s))$
unfolding *XX-def*
by (*rule flow-fixed-point'*[*OF* ‹$s \leq 0$› *s-in-existence-ivl-x0 x0-def*])
have *Y-integral-eq-left*: $Y\ (x - x0)\ s = x0 + (x - x0) - integral\ \{s..0\}\ (\lambda s.\ f\ (Y\ (x - x0)\ s))$
using *flow-fixed-point'* ‹$s \leq 0$› *s-in-existence-ivl2*[*OF z-in-ball*] *ball-in-X*[*OF z-in-ball*]
by (*simp add*: *Y-def*)
have *U-integral-eq-left*: $U\ (x - x0)\ s = (x - x0) - integral\ \{s..0\}\ (\lambda s.\ f'\ (XX\ x0\ s)\ (U\ (x - x0)\ s))$
unfolding *U-def A-def*[*symmetric*]
by (*rule var.flow-fixed-point'*)
(*auto simp*: ‹$s \leq 0$› *x0-def varexivl-eq-exivl s-in-existence-ivl-x0*)

have *?g* $s =$
$norm\ (- integral\ \{s..0\}\ (\lambda s'.\ f\ (Y\ (x - x0)\ s')) +$
$integral\ \{s..0\}\ (\lambda s'.\ f\ (XX\ x0\ s')) +$
$integral\ \{s..0\}\ (\lambda s'.\ (f'\ (XX\ x0\ s'))\ (U\ (x - x0)\ s')))$
unfolding *XX-integral-eq-left Y-integral-eq-left U-integral-eq-left*
by *simp*
also have ... $= norm\ (integral\ \{s..0\}\ (\lambda s'.\ f\ (Y\ (x - x0)\ s')) -$
$integral\ \{s..0\}\ (\lambda s'.\ f\ (XX\ x0\ s')) -$
$integral\ \{s..0\}\ (\lambda s'.\ (f'\ (XX\ x0\ s'))\ (U\ (x - x0)\ s')))$
by (*subst norm-minus-cancel*[*symmetric*], *simp*)
finally show *?g* $s =$
$norm\ (integral\ \{s..0\}\ (\lambda s'.\ f\ (Y\ (x - x0)\ s')) -$
$integral\ \{s..0\}\ (\lambda s'.\ f\ (XX\ x0\ s')) -$
$integral\ \{s..0\}\ (\lambda s'.\ blinfun\text{-}apply\ (f'\ (XX\ x0\ s'))\ (U\ (x - x0)\ s')))$
.
qed
also have ... $=$
$norm\ (integral\ \{a..b\}\ (\lambda s.\ f\ (Y\ (x - x0)\ s) - f\ (XX\ x0\ s) - (f'\ (XX\ x0\ s))\ (U\ (x - x0)\ s)))$
by (*simp add*: *i1 i2*)
also have ... $\leq$
$integral\ \{a..b\}\ (\lambda s.\ norm\ (f\ (Y\ (x - x0)\ s) - f\ (XX\ x0\ s) - f'\ (XX\ x0\ s)\ (U\ (x - x0)\ s)))$
using *x-in-X J-in-existence-ivl-x J-in-existence* ‹$\{a..b\} \subseteq J$›
by (*auto intro*!: *continuous-intros continuous-on-imp-absolutely-integrable-on*)
also have ... $= integral\ \{a..b\}$
$(\lambda s.\ norm\ (f'\ (XX\ x0\ s)\ (Y\ (x - x0)\ s - XX\ x0\ s - U\ (x - x0)\ s) + R\ (XX\ x0\ s)\ (Y\ (x - x0)\ s)))$
proof (*safe intro*!: *integral-spike*[*OF negligible-empty*, *simplified*] *arg-cong*[**where** *f=norm*])
fix s' **assume** $s' \in \{a..b\}$
show $f'\ (XX\ x0\ s')\ (Y\ (x - x0)\ s' - XX\ x0\ s' - U\ (x - x0)\ s') + R$

(*XX x0 s'*) (*Y* (*x* − *x0*) *s'*) =
f (*Y* (*x* − *x0*) *s'*) − *f* (*XX x0 s'*) − *f'* (*XX x0 s'*) (*U* (*x* − *x0*) *s'*)
by (*simp add*: *blinfun.diff-right taylor-expansion*(*2*)[*of Y* (*x* − *x0*) *s'*
XX x0 s'])
qed
also have ... ≤ *integral* {*a..b*}
(λ*s*. *norm* (*f'* (*XX x0 s*) (*Y* (*x* − *x0*) *s* − *XX x0 s* − *U* (*x* − *x0*) *s*)) +
norm (*R* (*XX x0 s*) (*Y* (*x* − *x0*) *s*)))
using *J-in-existence-ivl*[*OF x-in-ball*] *norm-triangle-ineq*
by (*auto intro*!: *continuous-intros integral-le*)
also have ... =
integral {*a..b*} (λ*s*. *norm* (*f'* (*XX x0 s*) (*Y* (*x* − *x0*) *s* − *XX x0 s* − *U*
(*x* − *x0*) *s*))) +
integral {*a..b*} (λ*s*. *norm* (*R* (*XX x0 s*) (*Y* (*x* − *x0*) *s*)))
using *J-in-existence-ivl*[*OF x-in-ball*]
by (*auto intro*!: *continuous-intros 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'* (*XX x0 s*)) ∗ *norm* (*Y* (*x* −
x0) *s* − *XX x0 s* − *U* (*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*} (λ*s*. *N* ∗ *norm* (*Y* (*x* − *x0*) *s* − *XX x0*
s − *U* (*x* − *x0*) *s*))
using *J-in-existence-ivl*[*OF x-in-ball*]
by (*intro integral-le*)
(*auto intro*!: *continuous-intros mult-right-mono*
dest!: *N-ineq*[*OF* ‹{*a..b*} ⊆ *J*›[*THEN subsetD*]])
also have ... = *N* ∗ *integral* {*a..b*} (λ*s*. *norm* ((*Y* (*x* − *x0*) *s* − *XX x0*
s − *U* (*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*} (λ*s*. *e1* ∗ *dist* (*XX x0 s*) (*Y* (*x* − *x0*) *s*))
using *J-in-existence-ivl*[*OF x-in-ball*] ‹*0* < *e-domain*› *dist-XX-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*} (λ*s*. *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-XX-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 have ?g s ≤ ?C + N * integral {a..b} ?g
    by simp
}
note g-bound = this
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 norm (Y (x − x0) t − XX x0 t − U (x − x0) t) ≤
    |t| * dist x0 x * exp (K * |t|) * e1 * exp (N * t)
    using ‹t ∈ J› J-def ‹t0 ≤ 0›
   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 norm (Y (x − x0) t − XX x0 t − U (x − x0) t) ≤
    |t| * dist x0 x * exp (K * |t|) * e1 * exp (− N * t)
    using ‹t ∈ J› J-def ‹0 ≤ t1›
    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* $((Y\ (x - x0)\ t - XX\ x0\ t - U\ (x - x0)\ t)\ /_R\ norm\ (x - x0)) < e$
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*:
$OT \subseteq existence\text{-}ivl\ x0 \Longrightarrow local\text{-}lipschitz\ OT\ (OS::('a \Rightarrow_L 'a)\ set)\ (\lambda t.\ op\ o_L\ (A\ x0\ t))$
by (*rule local-lipschitz-on-subset*[*OF - - subset-UNIV*, **where** *T*=*existence-ivl x0*])
(*auto simp*: *split-beta′ A-def XX-def*
intro!: *c1-implies-local-lipschitz*[**where** $f'=\lambda(t, x).\ comp3\ (f'\ (flow\ x0\ t))$]
derivative-eq-intros blinfun-eqI ext
continuous-intros flow-in-domain)

lemma *total-derivative-ll-on-open*:
ll-on-open (λt. *blinfun-compose* (*A x0 t*)) (*existence-ivl x0*) $(UNIV::('a \Rightarrow_L 'a)\ set)$
by *standard* (*auto intro*!: *continuous-intros local-lipschitz-A*[*OF order-refl*])

interpretation *mvar*: *ll-on-open* λt. *blinfun-compose* (*A x0 t*) *existence-ivl x0* $UNIV::('a \Rightarrow_L 'a)\ set$
by (*rule total-derivative-ll-on-open*)

lemma *wholevar-existence-ivl-eq-existence-ivl*:— TODO: unify with *?t* ∈ *existence-ivl x0* ⟹ *var*.*existence-ivl ?t ?a* = *existence-ivl x0*
assumes *t* ∈ *existence-ivl x0*
shows *mvar*.*existence-ivl t* = (λ-. *existence-ivl x0*)
proof (*rule ext*, *rule mvar*.*existence-ivl-eq-domain*)
fix *s t x*
assume *s*: *s* ∈ *existence-ivl x0* **and** *t*: *t* ∈ *existence-ivl x0*
then have {*s* .. *t*} ⊆ *existence-ivl x0*
by (*intro ivl2-subset-existence-ivl*[*OF x0-def*])
then have *continuous-on* {*s* .. *t*} (*A x0*)
by (*auto intro*!: *continuous-intros*)
then have *compact* (*A x0* ‘ {*s* .. *t*})
using *compact-Icc*
by (*rule compact-continuous-image*)
then obtain *B* **where** *B*: $\bigwedge u.\ u \in \{s\ ..\ t\} \Longrightarrow norm\ (A\ x0\ u) \le B$
by (*force dest*!: *compact-imp-bounded simp*: *bounded-iff*)
show $\exists M\ L.\ \forall t\in\{s\ ..\ t\}.\ \forall x\in UNIV.\ norm\ (A\ x0\ t\ o_L\ x) \le M + L * norm\ x$
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-ivl x0
  shows continuous-on (UNIV × existence-ivl x0) (λ(x, ta). mvar.flow t x ta)
proof −
  from mvar.flow-continuous-on-state-space[OF assms,
    unfolded wholevar-existence-ivl-eq-existence-ivl[OF assms]]
  show continuous-on (UNIV × existence-ivl x0) (λ(x, ta). mvar.flow t x ta) .
qed

definition W = mvar.flow 0 id-blinfun

lemma var-eq-mvar:
  assumes t0 ∈ existence-ivl x0
  assumes t ∈ existence-ivl x0
  shows var.flow t0 i t = mvar.flow 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
       wholevar-existence-ivl-eq-existence-ivl)

end
```

7.3 Differentiability of the flow

U t, i.e. the solution of the variational equation, is the space derivative at the initial value *x0*.

```
lemma flow-dx-derivative:
assumes x0 ∈ X
assumes t ∈ existence-ivl x0
shows ((λx0. flow x0 t) has-derivative (λz. U x0 z t)) (at x0)
  unfolding has-derivative-at
  apply(rule conjI[OF bounded-linear-U[OF ‹x0 ∈ X›]])
  subgoal using assms by force
  subgoal using assms(1,2)
    by (intro iffD1[OF LIM-equal proposition-17-6-weak[OF assms]])
      (simp add: diff-diff-add XX-def Y-def U-def inverse-eq-divide)
  done

lemma flow-dx-derivative-blinfun:
assumes x0 ∈ X
assumes t ∈ existence-ivl x0
shows ((λx. flow x t) has-derivative Blinfun (λz. U x0 z t)) (at x0)
by (rule has-derivative-Blinfun[OF flow-dx-derivative[OF assms]])

definition flowderiv x0 t = comp12 (W x0 t) (blinfun-scaleR-left (f (flow x0 t)))

lemma flowderiv-eq: flowderiv x0 t (ξ₁, ξ₂) = (W x0 t) ξ₁ + ξ₂ *R f (flow x0 t)
  by (auto simp: flowderiv-def)
```

lemma *W-continuous-on*: *continuous-on* (*Sigma X existence-ivl*) (λ(*x0*, *t*). *W 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*: $x \in X$ **and** *tx*: $t \in$ *existence-ivl x* **and** *e'*: $e' > 0$
let *?S* = *Sigma X existence-ivl*

have (*x*, *t*) $\in$ *?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-ivl x cball x dx* $\subseteq$ *X* **by** *auto*

interpret *one*: *ll-on-open* (λ*t*. *op* o_L (*A x t*)) *existence-ivl x UNIV*::(*'a*$\Rightarrow_L$*'a*) *set*
by (*rule total-derivative-ll-on-open*) *fact*

have *one-exivl*: *one.existence-ivl 0* = (λ-. *existence-ivl x*)
by (*rule wholevar-existence-ivl-eq-existence-ivl*[*OF* ‹*x* $\in$ *X*› *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-ivl x*
by (*intro Un-least ivl-subset-existence-ivl' ivl-subset-existence-ivl* ‹*x* $\in$ *X*›
‹*t* $\in$ *existence-ivl x*› ‹*cball t dt* $\subseteq$ *existence-ivl x*›)

have *compact* (*one.flow 0 id-blinfun* ‘ *?T*)
using ‹*?T* $\subseteq$ -› ‹*x* $\in$ *X*›
wholevar-existence-ivl-eq-existence-ivl[*OF* ‹*x* $\in$ *X*› *existence-ivl-zero*[*OF* ‹*x* $\in$ *X*›]]
by (*auto intro*!: ‹*0* < *dx*› *compact-continuous-image* ‹*compact ?T*›
continuous-on-subset[*OF one.flow-continuous-on*])

let *?line* = *one.flow 0 id-blinfun* ‘ *?T*
let *?X* = {*x*. *infdist x ?line* $\leq$ *dx*}
have *compact ?X*
using ‹*?T* $\subseteq$ -› ‹*x* $\in$ *X*›

wholevar-existence-ivl-eq-existence-ivl[*OF* ‹*x* ∈ *X*› *existence-ivl-zero*[*OF* ‹*x* ∈ *X*›]]
by (*auto intro*!: *compact-infdist-le* ‹*0* < *dx*› *compact-continuous-image compact-Un continuous-on-subset*[*OF one.flow-continuous-on*])

from *one.local-lipschitz* ‹*?T* ⊆ -›
have *llc*: *local-lipschitz ?T ?X* (λ*t*. *op* o_L (*A x t*))
by (*rule local-lipschitz-on-subset*) *auto*

have *cont*: ⋀*xa*. *xa* ∈ *?X* ⟹ *continuous-on ?T* (λ*t*. *A x t* o_L *xa*)
using ‹*?T* ⊆ -›
by (*auto intro*!: *continuous-intros* ‹*x* ∈ *X*›)

from *local-lipschitz-on-compact-implies-lipschitz*[*OF llc* ‹*compact ?X*› ‹*compact ?T*› *cont*]
obtain *K′* **where** *K′*: ⋀*ta*. *ta* ∈ *?T* ⟹ *lipschitz ?X* (*op* o_L (*A x ta*)) *K′*
by *blast*
def *K* ≡ *abs K′* + *1*
have *K* > *0*
by (*simp add*: *K-def*)
have *K*: ⋀*ta*. *ta* ∈ *?T* ⟹ *lipschitz ?X* (*op* o_L (*A x ta*)) *K*
by (*auto intro*!: *lipschitzI mult-right-mono order-trans*[*OF lipschitzD*[*OF K′*]] *simp*: *K-def*)

have *ex-ivlI*: ⋀*y*. *y* ∈ *cball x dx* ⟹ *?T* ⊆ *existence-ivl 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*). (*A y ta* o_L *xa*))
using ‹*cball x dx* ⊆ *X*› *ex-ivlI*
by (*force intro*!: *continuous-intros simp*: *split-beta′*)

have *one.flow 0 id-blinfun t* ∈ *one.flow 0 id-blinfun* ‘ ({*t*..*0*} ∪ {*0*..*t*} ∪ *cball t dt*)
by *auto*
then have *mem*: (*t*, *one.flow 0 id-blinfun t*, *x*) ∈ *?T* × *?X* × *cball x dx*
by (*auto simp*: ‹*0* < *dx*› *less-imp-le*)

def *e* ≡ *min e′* (*dx* / *2*) / *2*
have *e* > *0* **using** ‹*e′* > *0*› **by** (*auto simp*: *e-def* ‹*0* < *dx*›)
def *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 ‹$\{t..0\} \cup \{0..t\} \cup$ *cball t dt* $\subseteq$ *existence-ivl x*› *one-exivl*
by (*force intro*!: *compact-continuous-image* ‹*compact ?T*› *continuous-on-subset*[*OF one.flow-continuous-on*] *simp*: ‹$x \in X$›)

from *continuous-on-compact-product-lemma*[*OF cont cmpct*(*2*) *compact-cball* ‹$0 < d$›]
obtain d' **where** d': $d' > 0$
$\bigwedge$*ta xa xa' y*. *ta* $\in$ *?T* $\Longrightarrow$ *xa* $\in$ *?X* $\Longrightarrow$ *xa'*$\in$*cball x dx* $\Longrightarrow$ *y*$\in$*cball x dx* $\Longrightarrow$ *dist xa' y* $< d' \Longrightarrow$
dist (*A xa' ta* o_L *xa*) (*A y ta* o_L *xa*) $< d$
by *auto*

{
fix *y*
assume *dxy*: *dist x y* $< d'$
assume $y \in$ *cball x dx*
then have $y \in X$
using *dx dt OXOT* **by** *force+*

interpret *two*: *ll-on-open* (λt. *op* o_L (*A y t*)) *existence-ivl y UNIV*::('$a \Rightarrow_L$ 'a) *set*
by (*rule total-derivative-ll-on-open*) *fact*
have *two-exivl*: *two.existence-ivl 0* = (λ-. *existence-ivl y*)
by (*rule wholevar-existence-ivl-eq-existence-ivl*[*OF* ‹$y \in X$› *existence-ivl-zero*[*OF* ‹$y \in X$›]])

let *?X'* = $\bigcup x \in$ *?line*. *ball x dx*
have *open ?X'* **by** *auto*
have *?X'* $\subseteq$ *?X*
by (*auto intro*!: *infdist-le2 simp*: *dist-commute*)

interpret *oneR*: *ll-on-open* (λt. *op* o_L (*A x t*)) *existence-ivl x ?X'*
by *standard* (*auto intro*!: ‹$x \in X$› *continuous-intros local-lipschitz-A*[*OF* ‹$x \in X$› *order-refl*])
interpret *twoR*: *ll-on-open* (λt. *op* o_L (*A y t*)) *existence-ivl y ?X'*
by *standard* (*auto intro*!: ‹$y \in X$› *continuous-intros local-lipschitz-A*[*OF* ‹$y \in X$› *order-refl*])
interpret *both*:
two-ll-on-open (λt. *op* o_L (*A x t*)) *existence-ivl x* (λt. *op* o_L (*A y t*)) *existence-ivl y ?X' ?T id-blinfun d K*
proof *unfold-locales*
show *mem-codom*: *id-blinfun* $\in$ *?X'*
using ‹$0 < dx$› ‹$x \in X$›
by (*auto intro*!: *bexI*[**where** x=*0*])
show *zero-x*: $0 \in$ *existence-ivl x* **and** *zero-y*: $0 \in$ *existence-ivl y* **and** $0 < K$
by (*auto simp*: ‹$x \in X$› ‹$0 < dx$› ‹$0 < K$›
intro!: *existence-ivl-zero* ‹$x \in X$› ‹$y \in X$› *bexI*[**where** x=*0*])
show *iv-in*: $0 \in \{t..0\} \cup \{0..t\} \cup$ *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[OF - - - refl, where x=one.flow
0 id-blinfun])
      subgoal by (simp add: ‹x ∈ X›)
      subgoal by fact
      subgoal apply (rule ivp.is-solutionI)
        subgoal using iv-in mem-codom by unfold-locales auto
        subgoal using ‹x ∈ X› by simp
        subgoal
          using ‹x ∈ X› ‹?T ⊆ -›
          by (auto simp: one-exivl
         intro!: has-vector-derivative-at-within[OF one.flow-has-vector-derivative])
        subgoal using ‹x ∈ X› ‹dx > 0› by simp force
        done
      subgoal by fact
      subgoal by fact
      subgoal by fact
      done
    fix s assume s: s ∈ ?T
    then show lipschitz ?X' (op o_L (A x s)) K
      by (intro lipschitz-subset[OF K ‹?X' ⊆ ?X›]) auto
    fix j assume j: j ∈ ?X'
    show norm ((A x s o_L j) − (A 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
  {
    fix s assume s: s ∈ ?T ∩ twoR.existence-ivl 0 id-blinfun
    then 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.XX s = W x s
    unfolding both.XX-def W-def[OF ‹x ∈ X›]
    apply (rule oneR.maximal-existence-flow[OF - - - refl, where K=?T])
    subgoal by (rule both.t0-in-T1)
   subgoal using ‹0 < dx› by (force simp: ‹x ∈ X› intro!: bexI[where x=0])
    subgoal
      apply (rule ivp.is-solutionI)
      subgoal using ‹0 ∈ ?T›
       by unfold-locales (auto intro!: bexI[where x=0] simp: ‹x ∈ X› ‹0 < dx›)
      subgoal by (simp add: ‹x ∈ X›)
      subgoal
        apply simp
        using ‹cball t dt ⊆ existence-ivl x›  one-exivl tx ‹x ∈ X› x
          ‹?T ⊆ existence-ivl x›
     by (auto intro!: has-vector-derivative-at-within[OF one.flow-has-vector-derivative])
      subgoal using ‹0 < dx› by simp force
      done
    subgoal by (rule both.J-ivl)
    subgoal by (rule both.t0-in-J)
    subgoal using ‹?T ⊆ existence-ivl x› by blast
    subgoal by (rule s)
    done
  finally have norm (W x s − both.Y s) < e .
} note less-e = this

have e < dx using ‹dx > 0› by (auto simp: e-def)

let ?i = {x. infdist x (one.flow 0 id-blinfun ‘ ?T) ≤ e}
have 1: ?i ⊆ (⋃x∈one.flow 0 id-blinfun ‘ ?T. ball x dx)
proof −
  have cl: closed ?line ?line ≠ {} using compact-line
    by (auto simp: compact-imp-closed)
  have ?i ⊆ (⋃x∈one.flow 0 id-blinfun ‘ ?T. cball x 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) (one.flow 0 id-blinfun s) ≤ e
   unfolding W-def[OF ‹x ∈ X›] 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 by (rule both.x0-in-X)
  defer
 subgoal using ‹dt > 0› by (intro compact-infdist-le) (auto intro!: compact-line
‹0 < e›)
  subgoal by (rule 1)
  subgoal by (rule 2)
  done
also have twoR.existence-ivl 0 id-blinfun ⊆ existence-ivl y
  apply (rule twoR.existence-ivl-subset)
  subgoal by (rule both.t0-in-T2)
  subgoal
    using ‹0 < dx›
    by (force simp: ‹x ∈ X› intro!: bexI[where x=0])
  done
finally have ?T ⊆ existence-ivl y .
{
  fix s assume s: s ∈ ?T
  then have s ∈ ?T ∩ twoR.existence-ivl 0 id-blinfun using T-subset by force
  from less-e[OF this] have norm (W x s − both.Y s) < e .
  also have two.flow 0 id-blinfun s = twoR.flow 0 id-blinfun s
    apply (rule two.maximal-existence-flow[OF - - - refl, where K=?T])
    subgoal by (rule both.t0-in-T2)
    subgoal by simp
    subgoal
      apply (rule ivp.is-solutionI)
      unfolding ivp.simps
      subgoal using ‹0 ∈ ?T› by unfold-locales auto
      subgoal unfolding ivp.simps
        by (rule twoR.flow-initial-time)
          (auto intro!: bexI[where x=0] simp: ‹x ∈ X› ‹0 < dx› ‹y ∈ X›)
      subgoal
        apply (rule has-vector-derivative-at-within)
   apply (rule twoR.flow-has-vector-derivative[THEN has-vector-derivative-eq-rhs])
        subgoal by (simp add: ‹y ∈ X›)
```

```
          subgoal by (force intro!: bexI[where x=0] simp: ‹x ∈ X› ‹0 < dx›)
          subgoal using ‹?T ⊆ twoR.existence-ivl - -› by force
          subgoal by simp
          done
        subgoal by simp
        done
      subgoal by fact
      subgoal by fact
      subgoal by fact
      subgoal by fact
      done
    then have both.Y s = W y s
      unfolding both.Y-def W-def[OF ‹y ∈ X›]
      by simp
    finally have norm (W x s − W y s) < e .
  }
} 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-ivl x by (simp add: ‹x ∈ X›)
have ∀F x in at t within existence-ivl x. dist (one.flow 0 id-blinfun x) (one.flow
0 id-blinfun t) < e
  using one.flow-continuous-on[OF ‹0 ∈ existence-ivl x›]
  using ‹0 < e› tx
  by (auto simp add: continuous-on one-exivl dest!: tendstoD)
then have ∀F (y, s) in at (x, t) within ?S. dist (W x s) (W x t) < e
  using ‹0 < e›
 unfolding at-within-open[OF ‹(x, t) ∈ ?S› open-state-space] UNIV-Times-UNIV[symmetric]
W-def[OF ‹x ∈ X›]
  by (intro eventually-at-Pair-within-TimesI2)
    (auto simp: at-within-open[OF tx open-existence-ivl])
ultimately
have ∀F (y, s) in at (x, t) within ?S. dist (W y s) (W x t) < e′
```

apply *eventually-elim*
proof (*safe del*: *UnE*, *goal-cases*)
case (*1 y s*)
have *dist* (*W y s*) (*W x t*) $\leq$ *dist* (*W y s*) (*W x s*) + *dist* (*W x s*) (*W x t*)
by (*rule dist-triangle*)
also
have *dist* (*W x s*) (*W x t*) < *e*
by (*rule 1*)
also have *dist* (*W y s*) (*W x s*) < *e*
unfolding *dist-norm norm-minus-commute*
using *1*
by (*intro cont-data*)
also have *e* + *e* $\leq$ *e'* **by** (*simp add*: *e-def*)
finally show *dist* (*W y s*) (*W x t*) < *e'* **by** *arith*
qed
then show $\forall_F$ *ys in at* (*x*, *t*) *within ?S*. *dist* (*W* (*fst ys*) (*snd ys*)) (*W* (*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 ($\bigwedge$*s*. *s* $\in$ *S* $\Longrightarrow$ *h s* $\in$ *X*) $\Longrightarrow$ ($\bigwedge$*s*. *s* $\in$ *S* $\Longrightarrow$ *g s* $\in$ *existence-ivl* (*h s*)) $\Longrightarrow$
continuous-on S (λ*s*. *W* (*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-ivl*) (λ(*x0*, *t*). *f* (*flow 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* $\in$ *existence-ivl x0 x0* $\in$ *X*
shows ((λ*x0*. *flow x0 t*) *has-derivative W x0 t*) (*at x0*)
by (*rule flow-dx-derivative-blinfun*[*THEN has-derivative-eq-rhs*])
(*simp-all add*: *var-eq-mvar assms U-def blinfun.blinfun-apply-inverse W-def*)

lemma
flow-has-flowderiv:
assumes *t* $\in$ *existence-ivl x0 x0* $\in$ *X*
shows ((λ(*x0*, *t*). *flow x0 t*) *has-derivative flowderiv x0 t*) (*at* (*x0*, *t*) *within Sigma X existence-ivl*)
proof −
from *open-state-space assms* **obtain** *e'* **where** *e'*: *e'* > *0 ball* (*x0*, *t*) *e'* $\subseteq$ *Sigma X existence-ivl*

by (*force simp*: *open-contains-ball*)
def *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-ivl* **.**

have *d1*: ((λ*x0*. *flow x0 s*) *has-derivative blinfun-apply* (*W 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*: (*flow y has-derivative blinfun-apply* (*blinfun-scaleR-left* (*f* (*flow 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*). *flow 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 flow-continuous-on-state-space-comp*
simp: *flowderiv-def split-beta′*)
then show *?thesis*
by (*auto simp*: *at-within-open*[*OF - open-state-space*] *at-within-open*[*OF - open-Times*] *assms* ‹*0* < *e*›)
qed

lemma *flowderiv-continuous-on*: *continuous-on* (*Sigma X existence-ivl*) (λ(*x0*, *t*). *flowderiv x0 t*)
apply (*auto simp*: *flowderiv-def split-beta′ intro*!:)
apply (*subst blinfun-of-matrix-works*[**where** *f*=*comp12* (*W* (*fst x*) (*snd x*))
(*blinfun-scaleR-left* (*f* (*flow* (*fst x*) (*snd x*)))) **for** *x*, *symmetric*])
apply (*auto intro*!: *continuous-intros flow-in-domain*)
done

end — *True*

end

8 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 ((λx. exp (f x *R A)) has-derivative (λh. 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' = (λh. 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

locale linear-ivp = ivp i for i :: 'a::{banach,perfect-space} ivp +
  fixes A::'a blinop and s::real
  assumes rhs: ivp-f i = (λ(t, x). A x)
  assumes time: ivp-T i = UNIV
  assumes domain: ivp-X i = UNIV
  assumes t0: ivp-t0 i = s
begin

lemma exp-is-solution: is-solution (λt. exp ((t − t0) *R A) x0)
  by (auto intro!: is-solutionI derivative-eq-intros
   simp: rhs domain has-vector-derivative-def blinop.bilinear-simps exp-times-scaleR-commute)

sublocale has-solution
  by unfold-locales (rule exI[where P=is-solution, OF exp-is-solution])

sublocale unique-solution
proof(rule unique-solutionI[OF exp-is-solution])
  fix s t assume is-solution s and t ∈ T
  then have [derivative-intros]: (s has-derivative (λh. h *R A (s t))) (at t) for t
    by (auto dest!: is-solutionD(2) simp: has-vector-derivative-def rhs time)
  have ((λt. exp (−(t − t0) *R A) (s t)) has-derivative (λ-. 0)) (at t)
    (is (?es has-derivative -) -)
    for t
    by (auto intro!: derivative-eq-intros simp: has-vector-derivative-def
      blinop.bilinear-simps)
  from has-derivative-zero-constant[OF - this]
  obtain c where c: ?es = (λ-. c)
    by (auto simp: time)
  hence (λt. (exp ((t − t0) *R A) * (exp (−((t − t0) *R A)))) (s t)) = (λt. exp
((t − t0) *R A) c)
```

by (*metis* (*no-types*, *hide-lams*) *blinop-apply-times-blinop real-vector.scale-minus-left*)
then have *s-def*: $s = (\lambda t.\ exp\ ((t - t0) *_R A)\ c)$
by (*simp add*: *exp-minus-inverse*)
from ‹*is-solution s*› *s-def t0 is-solution-def*
have $exp\ ((t0 - t0) *_R A)\ c = x0$ **by** *simp*
hence $c = x0$ **by** (*simp add*:)
thus $s\ t = exp\ ((t - t0) *_R A)\ x0$ **using** *s-def* **by** *simp*
qed

end

end

9 Target Language debug messages

theory *Print*
imports
../../Affine-Arithmetic/Executable-Euclidean-Space
begin

very ad-hoc...

9.1 Printing

Just for debugging purposes

definition *print*::*String.literal* ⇒ *unit* **where** *print x* = ()

definition *int-to-string*::*int* ⇒ *String.literal*
where *int-to-string x* = *STR* ''''

context includes *integer.lifting* **begin**

lift-definition *integer-to-string*::*integer* ⇒ *String.literal*
is *int-to-string* **.**

end

lemma [*code*]: *integer-to-string x* = *STR* ''''
by (*simp add*: *integer-to-string-def int-to-string-def*)

lemma [*code*]: *int-to-string x* = *integer-to-string* (*integer-of-int x*)
by (*simp add*: *integer-to-string-def*)

definition *println x* = (*let* - = *print x in print* (*STR* ''↩''))

code-printing
constant *print* ⇀ (*SML*) *TextIO.print*
| **constant** *integer-to-string* :: *integer* ⇒ *String.literal* ⇀ (*SML*) *Int.toString*

consts *float2-float10*::*int* ⇒ *bool* ⇒ *int* ⇒ *int* ⇒ (*int* ∗ *int*)

context includes *integer.lifting* **begin**

lift-definition *float2-float10-integer*::*integer* ⇒ *bool* ⇒ *integer* ⇒ *integer* ⇒ (*integer* ∗ *integer*)
  is *float2-float10* **.**

lemma *float2-float10-code*[*code*]: *float2-float10 x b m e* =
  (*case float2-float10-integer* (*integer-of-int x*) *b* (*integer-of-int m*) (*integer-of-int e*) *of* (*a*, *b*) ⇒
  (*int-of-integer a*, *int-of-integer b*))
  by *transfer simp*

end

code-printing
  code-module *Float2-Float10* ⇀ (*SML*)
  — this is taken from Approximation.thy — TODO: implement in Isabelle/HOL?

‹

```
fun float2float10integer prec round-down m e = (
  let
    val (m, e) = (if e < 0 then (m,e) else (m * IntInf.pow (2, e), 0))

    fun frac c p 0 digits cnt = (digits, cnt, 0)
      | frac c 0 r digits cnt = (digits, cnt, r)
      | frac c p r digits cnt = (let
        val (d, r) = IntInf.divMod (r * 10, IntInf.pow (2, ~e))
      in frac (c orelse d <> 0) (if d <> 0 orelse c then p − 1 else p) r
              (digits * 10 + d) (cnt + 1)
      end)

    val sgn = Int.sign m
    val m = abs m

    val round-down = (sgn = 1 andalso round-down) orelse
                     (sgn = ~1 andalso not round-down)

    val (x, r) = IntInf.divMod (m, (IntInf.pow (2, ~e)))

    val p = ((if x = 0 then prec else prec − (IntInf.log2 x + 1)) * 3) div 10 + 1

    val (digits, e10, r) = if p > 0 then frac (x <> 0) p r 0 0 else (0,0,0)

    val digits = if round-down orelse r = 0 then digits else digits + 1

  in (sgn * (digits + x * (IntInf.pow (10, e10))), ~e10)
```

  end)
⟩
| **constant** *float2-float10-integer* ⇀ (*SML*) *float2float10integer*
code-reserved *SML float2-float10-integer*

definition *print-real*::*real* ⇒ *unit* **where** *print-real x* = ()

lemma *print-Floatreal*[*code*]:
  print-real (*FloatR a b*) =
    (*let*
      (*m*, *e*) = *float2-float10 25 True a b*;
      - = *print* (*int-to-string m*);
      - = *print* (*STR ''e''*);
      - = *print* (*int-to-string e*)
     in
       ())
  by *simp-all*

definition *print-eucl*::′*a*::*executable-euclidean-space* ⇒ *unit*
  where *print-eucl x* =
    (*let*
      - = *print* (*STR ''(''*);
     - = *map* (λ*i*. *let* - = *print-real* (*x* · *i*); - = *print* (*STR '', ''*) *in* ()) (*Basis-list*::′*a list*);
      - = *print* (*STR '')''*)
    in ())

definition *bind-err*:: *String.literal* ⇒ ′*c option* ⇒ (′*c* ⇒ ′*d option*) ⇒ ′*d option*
  where [*simp*]: *bind-err err* = *Option.bind*

lemma [*code*]:
  bind-err err None f = (*let* - = *println err in None*)
  bind-err err (*Some x*) *f* = *f x*
  by *auto*

end

10 One-Step Methods

theory *One-Step-Method*
imports
  ../IVP/Initial-Value-Problem
begin

10.1 Grids

locale *grid* =
  fixes *t*::*nat* ⇒ *real*

assumes *steps*: $\bigwedge i.\ t\ i \le t\ (Suc\ i)$
begin

lemmas *grid* = *steps*

lemma *grid-ge-min*:
shows $t\ 0 \le t\ j$
using *assms*
proof (*induct j*)
fix *j*
assume $t\ 0 \le t\ j$
also from *grid* **have** $t\ j \le t\ (Suc\ j)$ **.**
finally show $t\ 0 \le t\ (Suc\ j)$ **.**
qed *simp*

lemma *grid-mono*:
assumes $j \le n$
shows $t\ j \le t\ n$
using *assms*
proof (*induct rule*: *inc-induct*)
fix *j*
assume $j < n\ t\ (Suc\ j) \le t\ n$
moreover
with *grid* **have** $t\ j \le t\ (Suc\ j)$ **by** *auto*
ultimately
show $t\ j \le t\ n$ **by** *simp*
qed *simp*

The size of the step from point j to j+1 in grid t

definition *stepsize*
where $stepsize\ j = t\ (Suc\ j) - t\ j$

lemma *grid-stepsize-nonneg*:
shows $stepsize\ j \ge 0$
using *assms grid* **unfolding** *stepsize-def*
by (*simp add*: *field-simps order-less-imp-le*)

lemma *grid-stepsize-sum*:
shows $(\sum i \in \{0..<n\}.\ stepsize\ i) = t\ n - t\ 0$
by (*induct n*) (*simp-all add*: *stepsize-def*)

definition *max-stepsize*
where $max\text{-}stepsize\ n = Max\ (stepsize\ `\ \{0..n\})$

lemma *max-stepsize-ge-stepsize*:
assumes $j \le n$
shows $max\text{-}stepsize\ n \ge stepsize\ j$
using *assms* **by** (*auto simp*: *max-stepsize-def*)

lemma *max-stepsize-nonneg*:
  shows *max-stepsize n* ≥ *0*
  using *grid-stepsize-nonneg*[*of 0*]
    max-stepsize-ge-stepsize[*of 0 n*]
  by *simp*

lemma *max-stepsize-mono*:
  assumes *j* ≤ *n*
  shows *max-stepsize j* ≤ *max-stepsize n*
  using *assms* **by** (*auto intro*!: *Max-mono simp*: *max-stepsize-def*)

definition *min-stepsize*
where *min-stepsize n* = *Min* (*stepsize* ‘ {*0*..*n*})

lemma *min-stepsize-le-stepsize*:
  assumes *j* ≤ *n*
  shows *min-stepsize n* ≤ *stepsize j*
  using *grid assms*
  by (*auto simp add*: *min-stepsize-def*)

end

lemma (**in** *grid*) *grid-interval-notempty*: *t 0* ≤ *t n* **using** *grid-ge-min*[*of n*] **.**

10.2 Definition

Discrete evolution (noted Ψ) using incrementing function *incr*

definition *discrete-evolution*
where *discrete-evolution incr t1 t0 x* = *x* + (*t1* − *t0*) $*_R$ *incr* (*t1* − *t0*) *t0 x*

Using the discrete evolution Ψ between each two points of the grid, define a function over the whole grid

fun *grid-function*
where
  grid-function Ψ *x0 t 0* = *x0*
| *grid-function* Ψ *x0 t* (*Suc j*) = Ψ (*t* (*Suc j*)) (*t j*) (*grid-function* Ψ *x0 t j*)

10.3 Consistency

definition *consistent x t T B p incr* ⟷
  (∀ *h*≥*0*. *t* + *h* ≤ *T* ⟶ *dist* (*x* (*t* + *h*)) (*discrete-evolution incr* (*t* + *h*) *t* (*x t*))
≤ *B* ∗ *h* ^ (*p* + *1*))

lemma *consistentD*:
  assumes *consistent x t T B p incr*
  assumes *h* ≥ *0 t* + *h* ≤ *T*
  shows *dist* (*x* (*t* + *h*)) (*discrete-evolution incr* (*t* + *h*) *t* (*x t*)) ≤ *B* ∗ *h* ^ (*p* + *1*)

```
  using assms
  unfolding consistent-def by auto

lemma consistentI[intro]:
  fixes x::real⇒'a::real-normed-vector
  assumes ⋀h. h > 0 ⟹ t + h ≤ T ⟹
    dist (x (t + h)) (discrete-evolution incr (t + h) t (x t)) ≤ B ∗ h ˆ (p + 1)
  shows consistent x t T B p incr
  using assms unfolding consistent-def
  by safe (case-tac h = 0, auto simp: discrete-evolution-def)

lemma consistent-imp-nonneg-constant:
  assumes consistent x t T B p incr
  assumes t < T
  shows B ≥ 0
proof −
  from assms have T − t > 0 by simp
  have 0 ≤ dist (x T) (discrete-evolution incr T t (x t)) by simp
  also from assms
  have ... ≤ B ∗ (T − t) ˆ (p + 1)
    unfolding consistent-def by (auto dest: spec[where x=T − t])
  finally show ?thesis using zero-less-power[OF ‹T − t > 0›, of p+1]
    by (simp add: zero-le-mult-iff)
qed

lemma stepsize-inverse:
  assumes L ≥ 0 h ≥ 0 B ≥ 0 r ≥ 0 p > 0 T1 ≥ T2 T2 ≥ 0
  assumes max-step: h ≤ root p (r ∗ L / B / (exp (L ∗ T1 + 1) − 1))
  shows  B / L ∗ (exp (L ∗ T2 + 1) − 1) ∗ h ˆ p ≤ r
proof −
  { assume L = 0 hence ?thesis using ‹r ≥ 0› by simp
  } moreover {
    assume B-pos: B > 0 assume L > 0
    from ‹0 ≤ T2› ‹T1 ≥ T2› have T1 ≥ 0 by simp
    hence eg: (exp (L ∗ T1 + 1) − 1) > 0 using ‹L > 0›
      by (auto intro!: add-nonneg-pos)
    have B ∗ (h ˆ p ∗ (exp (L ∗ T2 + 1) − 1)) / L ≤
      B ∗ (root p (r ∗ L / B / (exp (L ∗ T1 + 1) − 1)) ˆ p
      ∗ (exp (L ∗ T2 + 1) − 1)) / L
      using assms
      by (auto simp add: ge-iff-diff-ge-0[symmetric] divide-simps
              intro!: mult-left-mono mult-right-mono power-mono)
    also
    have root p (r ∗ L / B / (exp (L ∗ T1 + 1) − 1)) ˆ p =
      (r ∗ L / B / (exp (L ∗ T1 + 1) − 1))
      using assms B-pos ‹T1 ≥ 0› ‹L > 0› ‹B > 0›
      by (subst real-root-pow-pos2[OF ‹p > 0›])
        (auto intro!: divide-nonneg-pos add-nonneg-pos mult-pos-pos)
    finally
```

```
    have B ∗ (h ˆ p ∗ (exp (L ∗ T2 + 1) − 1)) / L ≤
      r ∗ ((exp (L ∗ T2 + 1) − 1) / (exp (L ∗ T1 + 1) − 1))
      using B-pos ‹L > 0› eg ‹r ≥ 0›
      by (simp add: ac-simps)
    also have ... ≤ r using ‹T1 ≥ T2› ‹0 ≤ T2›
    proof (cases T1 = 0)
      assume T1 ≠ 0 with ‹T1 ≥ T2› ‹0 ≤ T2› have T1 > 0 by simp
      show ?thesis using ‹L > 0› ‹0 ≤ T2› ‹T1 ≥ 0› add-0-left ‹T1 > 0› ‹T1 ≥
T2›
        by (intro mult-right-le-one-le ‹r ≥ 0›)
         (subst pos-le-divide-eq pos-divide-le-eq, auto simp add: intro!: add-pos-pos)+
    qed simp
    finally have ?thesis by (simp add: ac-simps)
  } moreover {
    assume ¬0<B hence B = 0 using ‹B ≥ 0› by simp
    hence ?thesis using ‹r ≥ 0› by simp
  } ultimately show ?thesis using assms by arith
qed
```

10.4 Accumulation of errors

The concept of accumulating errors applies to convergence and stability.

```
lemma (in grid) error-accumulation:
  fixes x::(nat ⇒ real) ⇒ nat ⇒ 'a::euclidean-space
  assumes max-step: max-stepsize j ≤
    root p (|r| ∗ L / B / (exp (L ∗ (T − t 0) + 1) − 1))
  defines K ≡ {(s, y). ∃ i ≤ j. s = t i ∧ y ∈ cball (x t i) r}
  assumes p > 0
  assumes lipschitz: ⋀j. t (Suc j) ≤ T ⟹
  dist (x t j) (grid-function (discrete-evolution ψ) x0 t j) ≤ |r| ⟹
  dist (ψ (stepsize j) (t j) (x t j))
    (ψ (stepsize j) (t j) (grid-function (discrete-evolution ψ) x0 t j))
    ≤ L ∗ dist (x t j) (grid-function (discrete-evolution ψ) x0 t j) and L ≥ 0
  assumes consistence-error: ⋀j. t (Suc j) ≤ T ⟹
    dist (x t (Suc j)) (discrete-evolution ψ (t (Suc j)) (t j) (x t j)) ≤
    B ∗ stepsize j ˆ (p + 1) and B ≥ 0
  assumes initial-error: dist (x t 0) x0 ≤
    B ∗ (exp 1 − 1) ∗ stepsize 0 ˆ p / L
  assumes t j ≤ T
  shows dist (x t j) (grid-function (discrete-evolution ψ) x0 t j) ≤
    B / L ∗ (exp (L ∗ (t j − t 0) + 1) − 1) ∗ max-stepsize j ˆ p
using ‹t j ≤ T› max-step
proof (induct j)
  case 0 note initial-error
  also have B ∗ (exp 1 − 1) ∗ stepsize 0 ˆ p / L ≤
    B ∗ (exp 1 − 1) ∗ max-stepsize 0 ˆ p / L
    using grid-stepsize-nonneg ‹B≥0› ‹L≥0›
   by (auto intro!: max-stepsize-ge-stepsize power-mono mult-left-mono divide-right-mono)
  finally show ?case by simp
```

next
case (*Suc j*)
have *t 0* ≤ *T*
using *Suc grid-interval-notempty*[*of Suc j*] **by** *auto*
from *Suc* **have** *j-in*:*t j* ≤ *T* **using** *grid-mono*[*of j Suc j*] **by** *simp*
moreover
have *max-stepsize j* ≤ *max-stepsize* (*Suc j*)
by (*simp add*: *max-stepsize-mono*)
with *Suc* **have** *IH1*: *max-stepsize j* ≤
root p (|*r*| ∗ *L* / *B* / (*exp* (*L* ∗ (*T* − *t 0*) + *1*) − *1*)) **by** *simp*
ultimately have
IH2: *dist* (*x t j*) (*grid-function* (*discrete-evolution* ψ) *x0 t j*)
≤ *B* / *L* ∗ (*exp* (*L* ∗ (*t j* − *t 0*) + *1*) − *1*) ∗ *max-stepsize j* ˆ *p*
by (*rule Suc*(*1*))
have *dist* (*x t* (*Suc j*)) (*grid-function* (*discrete-evolution* ψ) *x0 t* (*Suc j*)) =
norm (*x t* (*Suc j*) −
(*discrete-evolution* ψ) (*t* (*Suc j*)) (*t j*) (*x t j*) +
((*discrete-evolution* ψ) (*t* (*Suc j*)) (*t j*) (*x t j*) −
(*discrete-evolution* ψ) (*t* (*Suc j*)) (*t j*) (*grid-function* (*discrete-evolution* ψ) *x0*
t j)))
by (*simp add*: *field-simps dist-norm*)
also have ... ≤ *norm* (*x t* (*Suc j*) −
(*discrete-evolution* ψ) (*t* (*Suc j*)) (*t j*) (*x t j*)) +
norm (((*discrete-evolution* ψ) (*t* (*Suc j*)) (*t j*) (*x t j*) −
(*discrete-evolution* ψ) (*t* (*Suc j*)) (*t j*) (*grid-function* (*discrete-evolution* ψ) *x0*
t j)))
(**is** - ≤ - + *?ej*)
by (*rule norm-triangle-ineq*)
also have *?ej* =
norm (*x t j* − *grid-function* (*discrete-evolution* ψ) *x0 t j* + *stepsize j* $*_R$
(ψ (*stepsize j*) (*t j*) (*x t j*) −
ψ (*stepsize j*) (*t j*) (*grid-function* (*discrete-evolution* ψ) *x0 t j*)))
by (*simp add*: *discrete-evolution-def stepsize-def algebra-simps*)
also have ... ≤
norm (*x t j* − *grid-function* (*discrete-evolution* ψ) *x0 t j*) + *norm* (*stepsize j*
$*_R$
(ψ (*stepsize j*) (*t j*) (*x t j*) −
ψ (*stepsize j*) (*t j*) (*grid-function* (*discrete-evolution* ψ) *x0 t j*)))
by (*rule norm-triangle-ineq*)
also have ... = *norm* (*x t j* − *grid-function* (*discrete-evolution* ψ) *x0 t j*) +
stepsize j ∗
dist (ψ (*stepsize j*) (*t j*) (*x t j*))
(ψ (*stepsize j*) (*t j*) (*grid-function* (*discrete-evolution* ψ) *x0 t j*))
(**is** - = - + - ∗ *?dj*)
using *grid-stepsize-nonneg*
by (*simp add*: *dist-norm*)
also
have *?dj* ≤ *L* ∗ *dist* (*x t j*) (*grid-function* (*discrete-evolution* ψ) *x0 t j*)
proof (*intro lipschitz*(*1*))

```
    from IH2 have
      dist (x t j) (grid-function (discrete-evolution ψ) x0 t j)
      ≤ B / L * (exp (L * (t j − t 0) + 1) − 1) * max-stepsize j ^ p
      by (simp add: ac-simps)
    also have ... ≤
      B / L * (exp (L * (T − t 0) + 1) − 1) * max-stepsize j ^ p
      using ‹L≥0› ‹B≥0› ‹t j ≤ T› max-stepsize-nonneg
      by (auto intro!: mult-left-mono mult-right-mono divide-right-mono)
    also have ... ≤ |r|
      using ‹B ≥ 0› max-step max-stepsize-nonneg ‹L ≥ 0› ‹p > 0›
        grid-ge-min using grid-mono[of 0 j] ‹t 0 ≤ T› IH1
      by (intro stepsize-inverse) auto
    finally show
      dist (x t j) (grid-function (discrete-evolution ψ) x0 t j) ≤ |r| .
  qed (insert Suc, simp)
  finally
  have dist (x t (Suc j)) (grid-function (discrete-evolution ψ) x0 t (Suc j))
    ≤ norm (x t (Suc j) − (discrete-evolution ψ) (t (Suc j)) (t j) (x t j)) +
    (1 + stepsize j * L) *
      dist (x t j) (grid-function (discrete-evolution ψ) x0 t j)
    using grid-stepsize-nonneg
    by (auto simp: algebra-simps mult-right-mono dist-norm)
  also
  have norm (x t (Suc j) − (discrete-evolution ψ) (t (Suc j)) (t j) (x t j)) ≤
    B * stepsize j ^ (p + 1)
    using consistence-error[OF ‹t (Suc j) ≤ T›] by (simp add: dist-norm)
  finally have rec:
    dist (x t (Suc j)) (grid-function (discrete-evolution ψ) x0 t (Suc j))
      ≤ B * stepsize j ^ (p + 1) +
        (1 + stepsize j * L) *
        dist (x t j) (grid-function (discrete-evolution ψ) x0 t j)
    by simp
  also have ... ≤ B * stepsize j ^ (p + 1) +
    (1 + stepsize j * L) * (B / L * (exp (L * (t j − t 0) + 1) − 1) * max-stepsize
j ^ p)
    using ‹B ≥ 0› IH1 IH2 ‹t (Suc j) ≤ T› ‹0≤L› grid-stepsize-nonneg
    by (intro add-mono mult-left-mono) auto
  finally
  have dist (x t (Suc j)) (grid-function (discrete-evolution ψ) x0 t (Suc j))
    ≤ B * stepsize j ^ (p + 1) +
    (1 + stepsize j * L) * (B / L * (exp (L * (t j − t 0) + 1) − 1) *
      max-stepsize j ^ p) .
  also have ... ≤ B * stepsize j * max-stepsize j ^ p +
    (1 + stepsize j * L) *
    (B / L * (exp (L * (t j − t 0) + 1) − 1) * max-stepsize j ^ p)
    using grid-stepsize-nonneg ‹B ≥ 0› grid
    by (auto intro!: mult-left-mono power-mono
      simp add: max-stepsize-def field-simps)
  also have ... = max-stepsize j ^ p * B / L * (1 + stepsize j * L) *
```

(*exp* (*L* ∗ (*t j* − *t 0*) + *1*))
− *max-stepsize j* ˆ *p* ∗ *B* / *L*
using ‹*B* ≥ *0*› *grid-stepsize-nonneg* ‹*p* > *0*› ‹*L*≥*0*›
apply (*cases L* ≠ *0*)
apply (*simp add*: *field-simps*)
apply (*cases max-stepsize j* = *0*)
apply *simp*
by (*metis IH1 abs-not-less-zero abs-of-pos divide-zero-left less-eq-real-def max-stepsize-nonneg mult-zero-right real-root-zero*)
also
have *B* ∗ (*max-stepsize j* ˆ *p* ∗ (*exp* (*L* ∗ (*t j* − *t 0*) + *1*) ∗
(*1* + *L* ∗ (*t* (*Suc j*) − *t j*)))) / *L*
≤ *B* ∗ (*max-stepsize j* ˆ *p* ∗ *exp* (*L* ∗ (*t* (*Suc j*) − *t 0*) + *1*)) / *L*
using ‹*L* ≥ *0*› ‹*B* ≥ *0*› *max-stepsize-nonneg*
proof (*intro divide-right-mono mult-left-mono*)
have *exp* (*L* ∗ (*t j* − *t 0*) + *1*) ∗ (*1* + *L* ∗ (*t* (*Suc j*) − *t j*)) ≤
exp (*L* ∗ (*t j* − *t 0*) + *1*) ∗ *exp* (*stepsize j* ∗ *L*)
unfolding *stepsize-def* [*symmetric*] **by** (*auto simp add*: *ac-simps*)
also have ... ≤ *exp* (*L* ∗ (*t* (*Suc j*) − *t 0*) + *1*)
by (*simp add*: *mult-exp-exp stepsize-def algebra-simps*)
finally
show *exp* (*L* ∗ (*t j* − *t 0*) + *1*) ∗ (*1* + *L* ∗ (*t* (*Suc j*) − *t j*)) ≤
exp (*L* ∗ (*t* (*Suc j*) − *t 0*) + *1*) **.**
qed *simp-all*
hence *max-stepsize j* ˆ *p* ∗ *B* / *L* ∗ (*1* + *stepsize j* ∗ *L*) ∗
exp (*L* ∗ (*t j* − *t 0*) + *1*) ≤
max-stepsize j ˆ *p* ∗ *B* / *L* ∗ *exp* (*L* ∗ (*t* (*Suc j*) − *t 0*) + *1*)
by (*simp add*: *stepsize-def ac-simps*)
finally
have *dist* (*x t* (*Suc j*)) (*grid-function* (*discrete-evolution* ψ) *x0 t* (*Suc j*))
≤ *B* / *L* ∗ (*exp* (*L* ∗ (*t* (*Suc j*) − *t 0*) + *1*) − *1*) ∗
max-stepsize j ˆ *p* **by** (*simp add*: *algebra-simps field-simps*)
also have ... ≤ *B* / *L* ∗ (*exp* (*L* ∗ (*t* (*Suc j*) − *t 0*) + *1*) − *1*) ∗
max-stepsize (*Suc j*) ˆ *p*
using ‹*B*≥*0*›‹*L*≥*0*› *max-stepsize-nonneg*
by (*intro mult-left-mono power-mono max-stepsize-mono*)
(*auto intro*!: *divide-nonneg-nonneg mult-nonneg-nonneg add-nonneg-nonneg grid-mono*)
finally show *?case* **.**
qed

10.5 Consistency of order p implies convergence of order p

locale *consistent-one-step* =
fixes *t0 t1* **and** *x*::*real* ⇒ *'a*::*euclidean-space* **and** *incr p B r L*
assumes *order-pos*: *p* > *0*
assumes *consistent-nonneg*: *B* ≥ *0*
assumes *consistent*: ⋀*s*. *s* ∈ {*t0*..*t1*} ⟹ *consistent x s t1 B p incr*
assumes *lipschitz-nonneg*: *L* ≥ *0*

```
  assumes lipschitz-incr: ⋀s h. s ∈ {t0..t1} ⟹ h ∈ {0..t1 − s} ⟹
    lipschitz (cball (x s) |r|) (λx. incr h s x) L

locale max-step = grid +
  fixes t1 p L B r
  assumes max-step: ⋀j. t j ≤ t1 ⟹ max-stepsize j ≤
    root p (|r| ∗ L / B / (exp (L ∗ (t1 − t 0) + 1) − 1))
begin

lemma max-step-mono-r:
  assumes |s| ≥ |r| L ≥ 0 B ≥ 0 t1 ≥ t 0 0 < p t j ≤ t1
  shows max-stepsize j ≤
    root p (|s| ∗ L / B / (exp (L ∗ (t1 − t 0) + 1) − 1))
proof −
  from max-step ‹t j ≤ t1› have max-stepsize j ≤
    root p (|r| ∗ L / B / (exp (L ∗ (t1 − t 0) + 1) − 1)) .
  also
  have ... ≤ root p (|s| ∗ L / B / (exp (L ∗ (t1 − t 0) + 1) − 1))
    using assms
    apply (cases B = 0, simp)
    apply (cases L = 0, simp)
    by (auto simp add: mult-le-cancel-left1
            intro!: divide-right-mono add-increasing mult-left-mono)
  finally
  show max-stepsize j ≤ root p (|s| ∗ L / B / (exp (L ∗ (t1 − t 0) + 1) − 1)) .
qed

end

locale convergent-one-step = consistent-one-step + max-step +
  assumes grid-from: t0 = t 0
begin

lemma (in convergent-one-step) convergence:
  assumes t j ≤ t1
  shows dist (x (t j)) (grid-function (discrete-evolution incr) (x (t 0)) t j) ≤
      B / L ∗ (exp (L ∗ (t1 − t 0) + 1) − 1) ∗ max-stepsize j ˆ p
proof −
  from order-pos consistent-nonneg lipschitz-nonneg
  have p > 0 B ≥ 0 L ≥ 0 by simp-all
  {
    fix j::nat assume t (Suc j) ≤ t1
    from consistent have dist (x (t j + stepsize j))
      (discrete-evolution incr (t j + stepsize j) (t j) (x (t j)))
      ≤ B ∗ (stepsize j ˆ (p + 1))
      apply (rule consistentD [OF - grid-stepsize-nonneg])
      using ‹t (Suc j) ≤ t1› grid-mono[of j Suc j] grid-from grid-interval-notempty

        by (auto simp add: stepsize-def)
```

} **note** *consistence-error* = *this*
{
fix *j*::*nat*
assume *t* (*Suc j*) ≤ *t1*
assume *in-K*:
dist (*x* (*t j*)) (*grid-function* (*discrete-evolution incr*) (*x* (*t 0*)) *t j*) ≤ |*r*|
hence *stepsize j* ∈ {*0..t1* − *t j*}
using *grid-stepsize-nonneg grid-mono* ‹*t* (*Suc j*) ≤ *t1*›
by (*simp add*: *stepsize-def*)
moreover
have *t j* ∈ {*t 0..t1*} **using** *grid*[*of j*] ‹*t* (*Suc j*) ≤ *t1*›
grid-mono[*of j Suc j*] *grid-ge-min* **by** *simp*
moreover
hence *x* (*t j*) ∈ *cball* (*x* (*t j*)) |*r*| **by** *simp*
moreover
hence *grid-function* (*discrete-evolution incr*) (*x* (*t 0*)) *t j* ∈
cball (*x* (*t j*)) |*r*| **using** *in-K* **by** *simp*
ultimately
have *dist* (*incr* (*stepsize j*) (*t j*) (*x* (*t j*)))
(*incr* (*stepsize j*) (*t j*)
(*grid-function* (*discrete-evolution incr*) (*x* (*t 0*)) *t j*))
≤ *L* ∗
dist (*x* (*t j*))
(*grid-function* (*discrete-evolution incr*) (*x* (*t 0*)) *t j*)
using *lipschitz-incr grid-from*
unfolding *lipschitz-def*
by *blast*
} **note** *lipschitz-grid* = *this*
have
dist (*x* (*t j*)) (*grid-function* (*discrete-evolution incr*) (*x* (*t 0*)) *t j*) ≤
(*B* / *L* ∗ (*exp* (*L* ∗ (*t j* − *t 0*) + *1*) − *1*)) ∗ *max-stepsize j* ˆ *p*
using ‹*p* > *0*› ‹*L* ≥ *0*› ‹*B* ≥ *0*› ‹*t j* ≤ *t1*›
max-stepsize-nonneg
consistence-error lipschitz-grid
by (*intro error-accumulation*[*OF max-step*]) (*auto intro*!:
divide-nonneg-nonneg mult-nonneg-nonneg zero-le-power grid-mono
simp add: *lipschitz-def stepsize-def*)
also have ... ≤
(*B* / *L* ∗ (*exp* (*L* ∗ (*t1* − *t 0*) + *1*) − *1*)) ∗ *max-stepsize j* ˆ *p*
using ‹*t j* ≤ *t1*› ‹*0*≤*L*› ‹*0*≤*B*› *max-stepsize-nonneg*
by (*auto intro*!: *divide-right-mono mult-right-mono mult-left-mono*)
finally show *?thesis* **by** *simp*
qed

end

10.6 Stability

locale *disturbed-one-step* = *grid* +

```
  fixes t1 s s0 x incr p B L
  assumes initial-error: norm s0 ≤ B / L * (exp 1 − 1) * stepsize 0 ^ p
  assumes error: ⋀j. t (Suc j) ≤ t1 ⟹
  norm (s (stepsize j) (t j)
  (grid-function (discrete-evolution (λh t x. incr h t x + s h t x))
    (x (t 0) + s0) t j)) ≤ B * stepsize j ^ p

locale stable-one-step =
  consistent-one-step t 0 + disturbed-one-step +
  max-step t t1 p L B r / 2
begin

lemma t0-le: t i ≤ t1 ⟹ t 0 ≤ t1
  by (metis grid-interval-notempty order.trans)

lemma max-step-r:
  assumes t j ≤ t1
  shows max-stepsize j ≤ root p (|r| * L / B / (exp (L * (t1 − t 0) + 1) − 1))
  using consistent-nonneg lipschitz-nonneg grid-interval-notempty order-pos assms
    grid-mono[of 0 j, simplified]
  by (intro max-step-mono-r) (auto simp: t0-le)


lemma stability:
  assumes t j ≤ t1
  defines incrs: incrs ≡ λh t x. incr h t x + s h t x
  shows dist
    (grid-function (discrete-evolution incrs) (x (t 0) + s0) t j)
    (grid-function (discrete-evolution incr) (x (t 0)) t j) ≤
        B / L * (exp (L * (t1 − t 0) + 1) − 1) * max-stepsize j ^ p
proof −
  have t 0 ≤ t1
    by (metis assms(1) grid-ge-min order-trans)
  {
    fix j assume t (Suc j) ≤ t1 from error[OF this]
    have stepsize j * norm (s (stepsize j) (t j)
        (grid-function (discrete-evolution incrs) (x (t 0) + s0) t j))
       ≤ stepsize j * (B * stepsize j ^ p)
      using grid-stepsize-nonneg
      by (auto intro: mult-left-mono simp: incrs)
    hence norm (stepsize j *R s (stepsize j) (t j)
        (grid-function (discrete-evolution incrs) (x (t 0) + s0) t j))
      ≤ B * stepsize j ^ (p + 1)
      using grid-stepsize-nonneg
      by (simp add: field-simps)
  } note error = this
  interpret c1: convergent-one-step t 0 using max-step-r
    by unfold-locales simp-all
  { fix j assume t (Suc j) ≤ t1
```

```
    hence t j ≤ t1 using grid-mono[of j Suc j] by auto
    have dist (x (t j)) (grid-function (discrete-evolution incr) (x (t 0)) t j)
      ≤ B / L * (exp (L * (t1 − t 0) + 1) − 1) * max-stepsize j ^ p
      using ‹t j ≤ t1› by (rule c1.convergence)
   also have ... ≤ |r/2| using max-stepsize-nonneg grid-interval-notempty max-step
      consistent-nonneg lipschitz-nonneg order-pos
      grid-mono ‹t j ≤ t1› t0-le
      apply (cases L = 0, simp)
      by (intro stepsize-inverse) auto
    finally have
      dist (x (t j)) (grid-function (discrete-evolution incr) (x (t 0)) t j) ≤
      |r / 2| .
  } note incr-in = this
  { fix j assume t (Suc j) ≤ t1
    note incr-in[OF this]
    also have |r/2| ≤ |r| by simp
    finally have
      dist (x (t j)) (grid-function (discrete-evolution incr) (x (t 0)) t j) ≤ |r|.
  }
  note incr-in-r = this
  have dist
    (grid-function (discrete-evolution incrs) (x (t 0) + s0) t j)
    (grid-function (discrete-evolution incr) (x (t 0)) t j) ≤
    B / L * (exp (L * (t j − t 0) + 1) − 1) * max-stepsize j ^ p
  proof (intro error-accumulation[OF max-step])
    fix j assume j: t (Suc j) ≤ t1
    show dist (grid-function (discrete-evolution incrs) (x (t 0) + s0) t (Suc j))
        (discrete-evolution incr (t (Suc j)) (t j) (grid-function (discrete-evolution
incrs) (x (t 0) + s0) t j))
      ≤ B * stepsize j ^ (p + 1)
      using error[OF j]
      by (simp add: incrs discrete-evolution-def[abs-def] dist-norm
        stepsize-def scaleR-right-distrib)
  next
    fix j assume t (Suc j) ≤ t1 hence t j ≤ t1 using grid-mono[of j Suc j]
      by simp
    have
      dist (x (t j)) (grid-function (discrete-evolution incrs) (x (t 0) + s0) t j)
      ≤ dist (x (t j)) (grid-function (discrete-evolution incr) (x (t 0)) t j) +
      dist (grid-function (discrete-evolution incrs) (x (t 0) + s0) t j)
      (grid-function (discrete-evolution incr) (x (t 0)) t j)
      by (rule dist-triangle2)
    also have
      dist (x (t j)) (grid-function (discrete-evolution incr) (x (t 0)) t j) ≤
      B / L * (exp (L * (t1 − t 0) + 1) − 1) * max-stepsize j ^ p
      using ‹t j ≤ t1› by (rule c1.convergence)
    also have ... ≤ |r/2|
      using max-stepsize-nonneg grid-interval-notempty max-step
        consistent-nonneg lipschitz-nonneg order-pos
```

```
      grid-mono ‹t j ≤ t1› ‹t 0 ≤ t1›
    by (intro stepsize-inverse) auto
  also
  assume dist (grid-function (discrete-evolution incrs) (x (t 0) + s0) t j)
    (grid-function (discrete-evolution incr) (x (t 0)) t j) ≤ |r / 2|
  finally
  have dist
    (x (t j))
    (grid-function (discrete-evolution incrs) (x (t 0) + s0) t j) ≤ |r| by simp
  thus dist
    (incr (stepsize j) (t j)
    (grid-function (discrete-evolution incrs) (x (t 0) + s0) t j))
    (incr (stepsize j) (t j)
    (grid-function (discrete-evolution incr) (x (t 0)) t j))
    ≤ L *
    dist (grid-function (discrete-evolution incrs) (x (t 0) + s0) t j)
    (grid-function (discrete-evolution incr) (x (t 0)) t j)
    using ‹t j ≤ t1› ‹t (Suc j) ≤ t1› incr-in-r
      max-stepsize-nonneg
      grid-ge-min
      grid-stepsize-nonneg
      grid-mono[of j]
    by (intro lipschitz-incr[THEN lipschitzD]) (auto simp: stepsize-def)
 next
  show dist (grid-function (discrete-evolution incrs) (x (t 0) + s0) t 0)
    (x (t 0))
    ≤ B * (exp 1 − 1) * stepsize 0 ^ p / L using initial-error
    by (simp add: dist-norm)
 qed (simp-all add: consistent-nonneg order-pos lipschitz-nonneg ‹t j ≤ t1›)
 also have ... ≤
  B / L * (exp (L * (t1 − t 0) + 1) − 1) * max-stepsize j ^ p
  using grid lipschitz-nonneg consistent-nonneg
    max-stepsize-nonneg
    grid-ge-min grid-mono ‹t j ≤ t1›
  by (auto simp add: ac-simps intro!: divide-right-mono mult-left-mono)
 finally have dist (grid-function (discrete-evolution incrs) (x (t 0) + s0) t j)
  (grid-function (discrete-evolution incr) (x (t 0)) t j)
 ≤ B / L * (exp (L * (t1 − t 0) + 1) − 1) * max-stepsize j ^ p .
 thus ?thesis by simp
qed

end
```

10.7 Stability via implicit error

```
locale rounded-one-step = consistent-one-step t 0 t1 x incr p B r L +
 max-step t t1 p L B r / 2
 for t::nat⇒real and t1 and x::real⇒('a::ordered-euclidean-space) and incr p B
r L +
```

fixes *incr'*::*real*⇒*real*⇒*'a*⇒*'a*
fixes *x0'*::*'a*
assumes *initial-error*: *dist* (*x* (*t 0*)) *x0'* ≤
B / *L* ∗ (*exp 1* − *1*) ∗ *stepsize 0* ˆ *p*
assumes *incr-approx*: ⋀*h j x*. *t j* ≤ *t1* ⟹ *dist* (*incr h* (*t j*) *x*) (*incr' h* (*t j*) *x*)
≤
B ∗ *stepsize j* ˆ *p*
begin

lemma *stability*:
assumes *t j* ≤ *t1*
shows *dist*
((*grid-function* (*discrete-evolution incr'*) (*x0'*) *t j*))
(*grid-function* (*discrete-evolution incr*) (*x* (*t 0*)) *t j*) ≤
B / *L* ∗ (*exp* (*L* ∗ (*t1* − (*t 0*)) + *1*) − *1*) ∗ *max-stepsize j* ˆ *p*
proof −
note *fg'* = *incr-approx*
def *s0* ≡ *x0'* − *x* (*t 0*)
hence *x0'*: *x0'* = *x* (*t 0*) + *s0*
by *simp*
def *s* ≡ λ*x xa xb*. (*incr' x xa xb*) − *incr x xa xb*
def *incrs* ≡ λ*h t x*. *incr h t x* + *s h t x*
have *s*: *incr'* = *incrs*
by (*simp add*: *s-def incrs-def*)
interpret *c*: *stable-one-step t1 x incr p B r L t s s0*
proof
fix *j*
assume (*t* (*Suc j*)) ≤ *t1*
hence *t j* ≤ *t1* **using** *grid-mono*[*of j Suc j*] **by** *simp*
have *norm* (*s* (*stepsize j*) (*t j*) (*grid-function*
(*discrete-evolution* (λ*h t x*. (*incr' h t x*)))
(*x* (*t 0*) + *s0*) *t j*))
≤ *B* ∗ *stepsize j* ˆ *p*
unfolding *s-def dist-norm*[*symmetric*]
unfolding *dist-commute*
using ‹*t j* ≤ *t1*›
by (*rule fg'*)
thus *norm*
(*s* (*stepsize j*) (*t j*)
(*grid-function* (*discrete-evolution* (λ*h t x*. *incr h t x* + *s h t x*))
(*x* (*t 0*) + *s0*) (λ*x*. (*t x*)) *j*))
≤ *B* ∗ *stepsize j* ˆ *p* **by** (*simp add*: *s incrs-def*)
next
show *norm s0* ≤ *B* / *L* ∗ (*exp 1* − *1*) ∗ *stepsize 0* ˆ *p*
unfolding *s0-def* **using** *initial-error* **by** (*simp add*: *dist-commute dist-norm*)
qed
show *?thesis*
unfolding *s x0'*
using ‹*t j* ≤ *t1*›

by (*rule c.stability*[*simplified incrs-def*[*symmetric*]])
qed

end

end

11 Runge-Kutta methods

theory *Runge-Kutta*
imports
~~/src/HOL/Multivariate-Analysis/Multivariate-Analysis
One-Step-Method
~~/src/HOL/Library/Float
../../Affine-Arithmetic/Executable-Euclidean-Space
../Library/Multivariate-Taylor
~~/src/HOL/Library/Convex
begin

11.1 aux

lemma *scale-back*: $(r,\ r *_R x) = r *_R (1,\ x)$ $(0,\ r *_R x) = r *_R (0,\ x)$
by *simp-all*

lemma *integral-normalize-bounds*:
fixes *s t*::*real*
assumes $t \le s$
assumes *f integrable-on* $\{t\ ..\ s\}$
shows [*symmetric*]: $(s - t) *_R$ *integral* $\{0\ ..\ 1\}$ $(\lambda x.\ f\ ((s - t) *_R x + t))$ = *integral* $\{t..s\}$ *f*
proof *cases*
assume $s > t$
hence $s - t \ne 0$ $0 \le s - t$ **by** *simp-all*
from *assms* **have** (*f has-integral integral* $\{t\ ..\ s\}$ *f*) (*cbox t s*)
by (*auto simp*: *integrable-integral*)
from *has-integral-affinity*[*OF this* ‹$s - t \ne 0$›, *of t*]
have $((\lambda x.\ f\ ((s - t) * x + t))$ *has-integral* $(1 / |s - t|) *_R$ *integral* $\{t..s\}$ *f*)
$((\lambda x.\ (x - t) / (s - t))$ ‘ $\{t..s\})$
using ‹$s > t$›
by (*simp add*: *divide-simps*)
also
have $t < s \Longrightarrow 0 \le x \Longrightarrow x \le 1 \Longrightarrow x * (s - t) + t \le s$ **for** *x*
by (*auto simp add*: *algebra-simps dest*: *mult-left-le-one-le*[*OF* ‹$0 \le s - t$›])
then have $((\lambda x.\ (x - t) / (s - t))$ ‘ $\{t..s\}) = \{0\ ..\ 1\}$
using ‹$s > t$›
by (*auto intro*!: *image-eqI*[**where** $x=x * (s - t) + t$ **for** *x*]
simp: *divide-simps*)
finally
have *integral* $\{0..1\}$ $(\lambda x.\ f\ ((s - t) * x + t)) = (1 / |s - t|) *_R$ *integral* $\{t..s\}$

f
  by (*rule integral-unique*)
 then show *?thesis*
  using ‹$s > t$› **by** *simp*
qed (*insert assms*, *simp*)

lemma
 has-integral-integral-eqI:
 f integrable-on s $\Longrightarrow$ *integral s f* $=$ *k* $\Longrightarrow$ (*f has-integral k*) *s*
 by (*simp add*: *has-integral-integral*)

lemma *convex-scaleR-sum2*:
 assumes $x \in G$ $y \in G$ *convex G*
 assumes $a \geq 0$ $b \geq 0$ $a + b \neq 0$
 shows $(a *_R x + b *_R y) /_R (a + b) \in G$
proof −
 have $(a / (a + b)) *_R x + (b / (a + b)) *_R y \in G$
  using *assms*
  by (*intro convexD*) (*auto simp*: *divide-simps*)
 then show *?thesis*
  by (*auto simp*: *algebra-simps divide-simps*)
qed

lemma *setsum-by-parts-ivt*:
 assumes *finite X*
 assumes *convex G*
 assumes $\bigwedge i.\ i \in X \Longrightarrow g\ i \in G$
 assumes $\bigwedge i.\ i \in X \Longrightarrow 0 \leq c\ i$
 obtains *y* **where** $y \in G$ $(\sum x{\in}X.\ c\ x *_R g\ x) = setsum\ c\ X *_R y \mid G = \{\}$
proof (*atomize-elim*, *cases setsum c X* $= 0$, *goal-cases*)
 case *pos*: *2*
 let $?y = (\sum x{\in}X.\ (c\ x / setsum\ c\ X) *_R g\ x)$
 have $?y \in G$ **using** *pos*
  by (*intro convex-setsum*)
   (*auto simp*: *setsum-divide-distrib*[*symmetric*]
    intro!: *divide-nonneg-nonneg assms setsum-nonneg*)
 thus *?case*
  by (*auto intro*!: *exI*[**where** $x = ?y$] *simp*: *scaleR-right.setsum pos*)
qed (*insert assms*, *auto simp*: *setsum-nonneg-eq-0-iff*)

lemma
 integral-by-parts-near-bounded-convex-set:
 assumes *f*: (*f has-integral I*) (*cbox a b*)
 assumes *s*: $((\lambda x.\ f\ x *_R g\ x)$ *has-integral P*) (*cbox a b*)
 assumes *G*: $\bigwedge x.\ x \in cbox\ a\ b \Longrightarrow g\ x \in G$
 assumes *nonneg*: $\bigwedge x.\ x \in cbox\ a\ b \Longrightarrow f\ x \geq 0$
 assumes *convex*: *convex G*
 assumes *bounded*: *bounded G*
 shows *infdist P* $(op *_R I$ ‘ $G) = 0$

proof (*rule dense-eq0-I*, *cases*)
fix *e*′::*real* **assume** *e0*: *0* < *e*′
assume *G* ≠ {}
from *bounded* **obtain** *bnd* **where** *bnd*: ⋀*y*. *y* ∈ *G* ⟹ *norm y* < *bnd bnd* > *0*
by (*meson bounded-pos gt-ex le-less-trans norm-ge-zero*)
def *e* ≡ *min* (*e*′ / *2*) (*e*′ / *2* / *bnd*)
have *e*: *e* > *0* **using** *e0*
by (*auto simp add*: *e-def intro*!: *divide-pos-pos* ‹*bnd* > *0*›)
from
has-integral[*of f I a b*, *THEN iffD1*, *OF f*, *rule-format*, *OF e*]
has-integral[*of* λ*x*. *f x* $*_R$ *g x P a b*, *THEN iffD1*, *OF s*, *rule-format*, *OF e*]
obtain *d1 d3*
where *d1*: *gauge d1*
⋀*p*. *p tagged-division-of cbox a b* ⟹ *d1 fine p* ⟹
norm ((∑(*x*, *k*)∈*p*. *content k* $*_R$ *f x*) − *I*) < *e*
and *d3*: *gauge d3*
⋀*p*. *p tagged-division-of cbox a b* ⟹ *d3 fine p* ⟹
norm ((∑(*x*, *k*)∈*p*. *content k* $*_R$ *f x* $*_R$ *g x*) − *P*) < *e*
by *auto*
def *d* ≡ λ*x*. *d1 x* ∩ *d3 x*
from *d1*(*1*) *d3*(*1*)
have *gauge d* **by** (*auto simp add*: *d-def*)
from *fine-division-exists*[*OF this*, *of a b*]
obtain *p* **where** *p*: *p tagged-division-of cbox a b d fine p*
by *metis*
from *tagged-division-of-finite*[*OF p*(*1*)]
have *finite p* **.**

from ‹*d fine p*› **have** *d1 fine p d3 fine p*
by (*auto simp*: *d-def fine-inter*)
have *f-less*: *norm* ((∑(*x*, *k*)∈*p*. *content k* $*_R$ *f x*) − *I*) < *e*
(**is** *norm* (*?f* − *I*) < -)
by (*rule d1*(*2*)[*OF p*(*1*)]) *fact*
have *norm* ((∑(*x*, *k*)∈*p*. *content k* $*_R$ *f x* $*_R$ *g x*) − *P*) < *e*
(**is** *norm* (*?s* − *P*) < -)
by (*rule d3*(*2*)[*OF p*(*1*)]) *fact*

hence *dist* (∑(*x*, *k*)∈*p*. *content k* $*_R$ *f x* $*_R$ *g x*) *P* < *e*
by (*simp add*: *dist-norm*)
also
let *?h* = (λ*x k y*. (*content k* ∗ *f x*) $*_R$ *y*)
let *?s*′ = λ*y*. *setsum* (λ(*x*, *k*). *?h x k y*) *p*
let *?g* = λ(*x*, *k*). *g x*
let *?c* = λ(*x*, *k*). *content k* ∗ *f x*
have *Pi*: ⋀*x*. *x* ∈ *p* ⟹ *?g x* ∈ *G* ⋀*x*. *x* ∈ *p* ⟹ *?c x* ≥ *0*
using *nonneg G p*
using *tag-in-interval*[*OF p*(*1*)]
by (*auto simp*: *intro*!: *mult-nonneg-nonneg*)
obtain *y* **where** *y*: *y* ∈ *G ?s* = *?s*′ *y*

by (*rule setsum-by-parts-ivt*[*OF* ‹*finite p*› ‹*convex G*› *Pi*])
(*auto simp*: *split-beta*′ *scaleR-setsum-left* ‹$G \neq \{\}$›)
note *this*(*2*)
also have $(\sum (x, k)\in p.\ (content\ k * f\ x) *_R\ y) = ?f *_R\ y$
by (*auto simp*: *scaleR-left.setsum intro*!: *setsum.cong*)
finally have $dist\ P\ ((\sum (x, k)\in p.\ content\ k *_R f\ x) *_R\ y) \leq e$
by (*simp add*: *dist-commute*)
moreover have $dist\ (I *_R\ y)\ ((\sum (x, k)\in p.\ content\ k *_R f\ x) *_R\ y) \leq norm\ y * e$
using *f-less*
by (*auto simp add*: *scaleR-dist-distrib-right*[*symmetric*] *dist-real-def*
intro!: *mult-left-mono*)
ultimately
have $dist\ P\ (I *_R\ y) \leq e + norm\ y * e$
by (*rule dist-triangle-le*[*OF add-mono*])
with - **have** $infdist\ P\ (op *_R\ I\ `\ G) \leq e + norm\ y * e$
using *y*(*1*)
by (*intro infdist-le2*) *auto*
also have $norm\ y * e < bnd * e$
by (*rule mult-strict-right-mono*)
(*auto simp*: ‹$e > 0$› *less-imp-le intro*!: *bnd* ‹$y \in G$›)
also have $bnd * e \leq e' / 2$
using ‹$e' > 0$› ‹$bnd > 0$›
by (*auto simp*: *e-def min-def divide-simps*)
also have $e \leq e' / 2$ **by** (*simp add*: *e-def*)
also have $e' / 2 + e' / 2 = e'$ **by** *simp*
finally show $|infdist\ P\ (op *_R\ I\ `\ G)| \leq e'$
by (*auto simp*: *infdist-nonneg*)
qed (*simp add*: *infdist-def*)

lemma
integral-by-parts-in-bounded-closed-convex-set:
assumes *f*: (*f has-integral I*) (*cbox a b*)
assumes *s*: $((\lambda x.\ f\ x *_R\ g\ x)\ has\text{-}integral\ P)\ (cbox\ a\ b)$
assumes *G*: $\bigwedge x.\ x \in cbox\ a\ b \Longrightarrow g\ x \in G$
assumes *nonneg*: $\bigwedge x.\ x \in cbox\ a\ b \Longrightarrow f\ x \geq 0$
assumes *bounded*: *bounded G*
assumes *closed*: *closed G*
assumes *convex*: *convex G*
assumes *nonempty*: $cbox\ a\ b \neq \{\}$
shows $P \in op *_R\ I\ `\ G$
proof −
let $?IG = op *_R\ I\ `\ G$
from *bounded closed* **have** *bounded ?IG closed ?IG*
by (*simp-all add*: *bounded-scaling closed-scaling*)
have $G \neq \{\}$ **using** *nonempty G* **by** *auto*
then show *?thesis*
using ‹*closed ?IG*›
by (*subst in-closed-iff-infdist-zero*)

(*auto intro*!: *assms compact-imp-bounded integral-by-parts-near-bounded-convex-set*)
qed

lemma
integral-by-parts-in-bounded-set:
assumes *f*: (*f has-integral I*) (*cbox a b*)
assumes *s*: (($\lambda x.$ *f x* $*_R$ *g x*) *has-integral P*) (*cbox a b*)
assumes *nonneg*: $\bigwedge x.\ x \in$ *cbox a b* $\Longrightarrow$ *f x* $\geq$ *0*
assumes *bounded*: *bounded* (*g* ‘ *cbox a b*)
assumes *nonempty*: *cbox a b* $\neq$ {}
shows *P* $\in$ *op* $*_R$ *I* ‘ *closure* (*convex hull* (*g* ‘ *cbox a b*))
proof −
have *x* $\in$ *cbox a b* $\Longrightarrow$ *g x* $\in$ *closure* (*convex hull g* ‘ *cbox a b*) **for** *x*
by (*meson closure-subset hull-subset imageI subsetCE*)
then show *?thesis*
by (*intro integral-by-parts-in-bounded-closed-convex-set*[*OF f s - nonneg - - - nonempty*])
(*auto intro*!: *bounded-closure bounded-convex-hull bounded convex-closure*
simp: *convex-convex-hull*)
qed

lemma *snd-imageI*: (*a*, *b*) $\in$ *R* $\Longrightarrow$ *b* $\in$ *snd* ‘ *R*
by *force*

lemma
snd-Pair4I:
assumes $\bigwedge t.\ t \in S \Longrightarrow (a,\ d,\ e,\ b\ t) \in R$
assumes $\bigwedge G.\ (\bigwedge t.\ t \in S \Longrightarrow b\ t \in G) \Longrightarrow x \in G$
shows $(a,\ d,\ e,\ x) \in R$
using *assms* **by** *auto*

lemma
snd-Pair5I:
assumes $\bigwedge t.\ t \in S \Longrightarrow (a,\ c,\ d,\ e,\ b\ t) \in R$
assumes $\bigwedge G.\ (\bigwedge t.\ t \in S \Longrightarrow b\ t \in G) \Longrightarrow x \in G$
shows $(a,\ c,\ d,\ e,\ x) \in R$
using *assms* **by** *auto*

lemma *in-minus-Collect*: $a \in A \Longrightarrow b \in B \Longrightarrow a - b \in \{x - y | x\ y.\ x \in A \wedge y \in B\}$
by *blast*

lemma *closure-minus-Collect*:
fixes *A B*::*'a*::*real-normed-vector set*
shows
$\{x - y | x\ y.\ x \in$ *closure* $A \wedge y \in$ *closure* $B\} \subseteq$ *closure* $\{x - y | x\ y.\ x \in A \wedge y \in B\}$
proof −
have *image*: $(\lambda(x,\ y).\ x - y)$ ‘ $(A \times B) = \{x - y | x\ y.\ x \in A \wedge y \in B\}$ **for** *A*

B::$'a$ *set*
    by *auto*
  have $\{x - y|x\ y.\ x \in closure\ A \wedge y \in closure\ B\} = (\lambda(x,\ y).\ x - y)$ ‘ *closure* $(A \times B)$
    unfolding *closure-Times*
    by (*rule image*[*symmetric*])
  also have ... $\subseteq$ *closure* $((\lambda(x,\ y).\ x - y)$ ‘ $(A \times B))$
    by (*rule image-closure-subset*)
      (*auto simp*: *split-beta′ intro*!: *set-mp*[*OF closure-subset*]
        continuous-at-imp-continuous-on)
  also note *image*
  finally show *?thesis* **.**
qed

lemma *convex-hull-minus-Collect*:
  fixes $A\ B$::$'a$::*real-normed-vector set*
  shows
  $\{x - y|x\ y.\ x \in convex\ hull\ A \wedge y \in convex\ hull\ B\} = convex\ hull\ \{x - y|x\ y.\ x \in A \wedge y \in B\}$
proof −
  have *image*: $(\lambda(x,\ y).\ x - y)$ ‘ $(A \times B) = \{x - y|x\ y.\ x \in A \wedge y \in B\}$ **for** A B::$'a$ *set*
    by *auto*
  have $\{x - y|x\ y.\ x \in convex\ hull\ A \wedge y \in convex\ hull\ B\} = (\lambda(x,\ y).\ x - y)$ ‘ (*convex hull* $(A \times B)$)
    unfolding *convex-hull-Times*
    by (*rule image*[*symmetric*])
  also have ... = *convex hull* $((\lambda(x,\ y).\ x - y)$ ‘ $(A \times B))$
    apply (*rule convex-hull-linear-image*)
    by *unfold-locales* (*auto simp*: *algebra-simps*)
  also note *image*
  finally show *?thesis* **.**
qed

lemma *set-minus-subset*:
  $A \subseteq C \Longrightarrow B \subseteq D \Longrightarrow \{a - b\ |a\ b.\ a \in A \wedge b \in B\} \subseteq \{a - b\ |a\ b.\ a \in C \wedge b \in D\}$
  by *auto*

lemma (**in** *bounded-bilinear*) *bounded-image*:
  assumes *bounded* (f ‘ s)
  assumes *bounded* (g ‘ s)
  shows *bounded* $((\lambda x.\ prod\ (f\ x)\ (g\ x))$ ‘ $s)$
proof −
  from *nonneg-bounded* **obtain** K
  where K: $\bigwedge a\ b.\ norm\ (prod\ a\ b) \leq norm\ a * norm\ b * K$ **and** $0 \leq K$
    by *auto*
  from *assms* **obtain** $F\ G$
  where F: $\bigwedge x.\ x \in s \Longrightarrow norm\ (f\ x) \leq F$

and *G*: $\bigwedge x.\ x \in s \Longrightarrow$ *norm* (*g x*) $\le$ *G*
and *nonneg*: *0* $\le$ *F 0* $\le$ *G*
by (*auto simp*: *bounded-pos intro*: *less-imp-le*)
have *norm* (*prod* (*f x*) (*g x*)) $\le F * G * K$ **if** *x*: $x \in s$ **for** *x*
using *F*[*OF x*] *G*[*OF x*] *nonneg* ‹*0* $\le$ *K*›
by (*auto intro*!: *mult-mono mult-nonneg-nonneg order-trans*[*OF K*])
thus *?thesis*
by (*auto simp*: *bounded-iff*)
qed

lemmas *bounded-scaleR-image* = *bounded-bilinear.bounded-image*[*OF bounded-bilinear-scaleR*]
and *bounded-blinfun-apply-image* = *bounded-bilinear.bounded-image*[*OF bounded-bilinear-blinfun-apply*]

lemma *bounded-plus-image*:
fixes *f*::*'a* ⇒ *'b*::*real-normed-vector*
assumes *bounded* (*f* ‘ *s*)
assumes *bounded* (*g* ‘ *s*)
shows *bounded* ((λ*x*. *f x* + *g x*) ‘ *s*)
proof −
from *assms* **obtain** *F G*
where *F*: $\bigwedge x.\ x \in s \Longrightarrow$ *norm* (*f x*) $\le$ *F*
and *G*: $\bigwedge x.\ x \in s \Longrightarrow$ *norm* (*g x*) $\le$ *G*
by (*auto simp*: *bounded-iff*)
have *norm* (*f x* + *g x*) $\le$ *F* + *G* **if** *x*: $x \in s$ **for** *x*
using *F*[*OF x*] *G*[*OF x*]
by *norm*
thus *?thesis*
by (*auto simp*: *bounded-iff*)
qed

lemma *bounded-Pair-image*:
fixes *f*::*'a* ⇒ *'b*::*real-normed-vector*
fixes *g*::*'a* ⇒ *'c*::*real-normed-vector*
assumes *bounded* (*f* ‘ *s*)
assumes *bounded* (*g* ‘ *s*)
shows *bounded* ((λ*x*. (*f x*, *g x*)) ‘ *s*)
proof −
from *assms* **obtain** *F G*
where *F*: $\bigwedge x.\ x \in s \Longrightarrow$ *norm* (*f x*) $\le$ *F*
and *G*: $\bigwedge x.\ x \in s \Longrightarrow$ *norm* (*g x*) $\le$ *G*
by (*auto simp*: *bounded-iff*)
have *norm* (*f x*, *g x*) $\le$ *F* + *G* **if** *x*: $x \in s$ **for** *x*
using *F*[*OF x*] *G*[*OF x*]
by (*intro order-trans*[*OF norm-Pair-le*]) *norm*
thus *?thesis*
by (*auto simp*: *bounded-iff*)
qed

11.2 Definitions

declare *setsum.cong*[*fundef-cong*]
fun *rk-eval* :: (*nat*⇒*nat*⇒*real*) ⇒ (*nat*⇒*real*) ⇒ (*real*×′*a*::*real-vector* ⇒ ′*a*) ⇒ *real* ⇒ *real* ⇒ ′*a* ⇒ *nat* ⇒ ′*a* **where**
rk-eval A c f t h x j =
f (*t* + *h* ∗ *c j*, *x* + *h* $*_R$ (∑ *l*=*1* ..< *j*. *A j l* $*_R$ *rk-eval A c f t h x l*))

primrec *rk-eval-dynamic* :: (*nat*⇒*nat*⇒*real*) ⇒ (*nat*⇒*real*) ⇒ (*real*×′*a*::{*comm-monoid-add*, *scaleR*} ⇒ ′*a*) ⇒ *real* ⇒ *real* ⇒ ′*a* ⇒ *nat* ⇒ (*nat* ⇒ ′*a*) **where**
rk-eval-dynamic A c f t h x 0 = (λ*i*. *0*)
| *rk-eval-dynamic A c f t h x* (*Suc j*) =
(*let K* = *rk-eval-dynamic A c f t h x j in*
K(*Suc j*:=*f* (*t* + *h* ∗ *c* (*Suc j*), *x* + *h* $*_R$ (∑ *l*=*1*..*j*. *A* (*Suc j*) *l* $*_R$ *K l*))))

definition *rk-increment* **where**
rk-increment f s A b c h t x = (∑ *j*=*1*..*s*. *b j* $*_R$ *rk-eval A c f t h x j*)

definition *rk-increment′* **where**
rk-increment′ error f s A b c h t x =
eucl-down error (∑ *j*=*1*..*s*. *b j* $*_R$ *rk-eval A c f t h x j*)

definition *euler-increment* **where**
euler-increment f = *rk-increment f 1* (λ*i j*. *0*) (λ*i*. *1*) (λ*i*. *0*)

definition *euler* **where**
euler f = *grid-function* (*discrete-evolution* (*euler-increment f*))

definition *euler-increment′* **where**
euler-increment′ e f = *rk-increment′ e f 1* (λ*i j*. *0*) (λ*i*. *1*) (λ*i*. *0*)

definition *euler′* **where**
euler′ e f = *grid-function* (*discrete-evolution* (*euler-increment′ e f*))

definition *rk2-increment* **where**
rk2-increment x f = *rk-increment f 2* (λ*i j*. *if i* = *2* ∧ *j* = *1 then x else 0*)
(λ*i*. *if i* = *1 then 1* − *1* / (*2* ∗ *x*) *else 1* / (*2* ∗ *x*)) (λ*i*. *if i* = *2 then x else 0*)

definition *rk2* **where**
rk2 x f = *grid-function* (*discrete-evolution* (*rk2-increment x f*))

11.3 Euler method is consistent

lemma *euler-increment*:
fixes *f*::- ⇒ ′*a*::*real-vector*
shows *euler-increment f h t x* = *f* (*t*, *x*)
unfolding *euler-increment-def rk-increment-def*
by (*subst rk-eval.simps*) (*simp del*: *rk-eval.simps*)

lemma *euler-float-increment*:
  fixes f::- ⇒ *'a*::*executable-euclidean-space*
  shows *euler-increment' e f h t x* = *eucl-down e* (*f* (*t*, *x*))
  unfolding *euler-increment'-def rk-increment'-def*
  by (*subst rk-eval.simps*) (*simp del*: *rk-eval.simps*)

lemma *euler-lipschitz*:
  fixes *x*::*real* ⇒ *real*
  fixes *f*::- ⇒ *'a*::*real-normed-vector*
  assumes *t*: *t* ∈ {*t0*..*T*}
  assumes *lipschitz*: ∀ *t*∈{*t0*..*T*}. *lipschitz D'* (λ*x*. *f* (*t*, *x*)) *L*
  shows *lipschitz D'* (*euler-increment f h t*) *L*
  using *t lipschitz*
  by (*simp add*: *lipschitz-def euler-increment del*: *One-nat-def*)

lemma *rk2-increment*:
  fixes *f*::- ⇒ *'a*::*real-vector*
  shows *rk2-increment p f h t x* =
    (*1* − *1* / (*p* ∗ *2*)) ∗$_R$ *f* (*t*, *x*) +
    (*1* / (*p* ∗ *2*)) ∗$_R$ *f* (*t* + *h* ∗ *p*, *x* + (*h* ∗ *p*) ∗$_R$ *f* (*t*, *x*))
  unfolding *rk2-increment-def rk-increment-def*
  apply (*subst rk-eval.simps*)
  apply (*simp del*: *rk-eval.simps add*: *numeral-2-eq-2*)
  apply (*subst rk-eval.simps*)
  apply (*simp del*: *rk-eval.simps add*: *field-simps*)
  done

11.4 Set-Based Consistency of Euler Method

locale *derivative-set-bounded* =
  derivative-on-prod +
  fixes *F F'*
  assumes *f-set-bounded*: *bounded F* ⋀*t x*. *t*∈*T* ⟹ *x*∈*X* ⟹ (*x*, *f* (*t*, *x*)) ∈ *F*
  assumes *f'-convex-compact*: *convex F' compact F'* ⋀*t x d*. *t*∈*T* ⟹ (*x*, *d*) ∈ *F*
⟹
    f' (*t*,*x*) (*1*, *d*) ∈ *F'*
begin

lemma *F-nonempty*: *F* ≠ {}
  and *F'-nonempty*: *F'* ≠ {}
  using *nonempty*
  unfolding *ex-in-conv*[*symmetric*]
  by (*auto intro*!: *f-set-bounded f'-convex-compact*)

lemma *euler-consistent-traj-set*:
  fixes *t*
  assumes *ht*: *0* ≤ *h t* + *h* ≤ *u*
  assumes *T*: {*t*..*u*} ⊆ *T*
  assumes *x'*: ⋀*s*. *s* ∈ {*t*..*u*} ⟹ (*x has-vector-derivative f* (*s*, *x s*)) (*at s within*

```
{t..u})
  assumes x: ⋀s. s ∈ {t..u} ⟹ x s ∈ X
  shows x (t + h) − discrete-evolution (euler-increment f) (t + h) t (x t) ∈ op
*R (h^2 / 2) ` F'
proof cases
  assume h = 0
  from F'-nonempty obtain f' where f' ∈ F' by auto
  from this ‹h = 0› show ?thesis
    by (auto simp: discrete-evolution-def)
next
  assume h ≠ 0
  from this ht have t < u by simp
  from ht have line-subset: (λta. t + ta * h) ` {0..1} ⊆ {t..u}
    by (auto intro!: order-trans[OF add-left-mono[OF mult-left-le-one-le]])
  hence line-in: ⋀s. 0 ≤ s ⟹ s ≤ 1 ⟹ t + s * h ∈ {t..u}
    by (rule set-mp) auto
  from ht have subset: {t .. t + h} ⊆ {t .. u} by simp
  let ?T = {t..u}
  from ht have subset: {t .. t + h} ⊆ {t .. u} by simp
  from ‹t < u› have t ∈ ?T by auto
  from ‹t < u› have tx: t ∈ T x t ∈ X using assms by auto
  from tx assms have 0 ≤ norm (f (t, x t)) by simp
  have x-diff: ⋀s. s ∈ ?T ⟹ x differentiable at s within ?T
    by (rule differentiableI, rule x'[simplified has-vector-derivative-def])
  have f': ⋀t x. t ∈ ?T ⟹ x ∈ X ⟹ (f has-derivative f' (t, x)) (at (t, x) within
(?T × X))
    using T by (intro has-derivative-subset[OF f']) auto
  let ?p = (λt. f' (t, x t) (1, f (t, x t)))
  def diff ≡ λn::nat. if n = 0 then x else if n = 1 then λt. f (t, x t) else ?p
  have diff-0[simp]: diff 0 = x by (simp add: diff-def)
  {
    fix m::nat and ta::real
    assume mta: m < 2 t ≤ ta ta ≤ t + h
    have image-subset: (λxa. (xa, x xa)) ` {t..u} ⊆ {t..u} × X
      using assms by auto
    note has-derivative-in-compose[where f=(λxa. (xa, x xa)) and g = f, derivative-intros]
    note has-derivative-subset[OF - image-subset, derivative-intros]
    note f'[derivative-intros]
    note x'[simplified has-vector-derivative-def, derivative-intros]
    have [simp]: ⋀c x'. c *R f' (ta, x ta) x' = f' (ta, x ta) (c *R x')
     using mta ht assms by (auto intro!: f' linear-cmul[symmetric] has-derivative-linear)
    have ((λt. f (t, x t)) has-vector-derivative f' (ta, x ta) (1, f (ta, x ta))) (at ta
within {t..u})
      unfolding has-vector-derivative-def
      using assms ht mta by (auto intro!: derivative-eq-intros)
    hence (diff m has-vector-derivative diff (Suc m) ta) (at ta within {t..t + h})
      using mta ht
      by (auto simp: diff-def intro!: has-vector-derivative-within-subset[OF - subset]
x')
```

} **note** *diff* = *this*

from *taylor-has-integral*[*of 2 diff x t t* + *h*, *OF* - - *diff*] ‹*0* ≤ *h*›
have *taylor*: ((λ*xa*. (*t* + *h* − *xa*) $*_R$ *f ′* (*xa*, *x xa*) (*1*, *f* (*xa*, *x xa*))) *has-integral* *x* (*t* + *h*) − (*x t* + *h* $*_R$ *f* (*t*, *x t*))) {*t*..*t* + *h*}
by (*simp add*: *eval-nat-numeral diff-def*)

have ∗: h^2 / *2* = *content* {*t*..*t* + *h*} $*_R$ (*t* + *h*) − (*if t* ≤ *t* + *h then* $(t + h)^2$ / *2* − t^2 / *2 else 0*)
using ‹*0* ≤ *h*›
by (*simp add*: *algebra-simps power2-eq-square divide-simps*)
have *integral*: (*op* − (*t* + *h*) *has-integral* h^2 / *2*) (*cbox t* (*t* + *h*))
unfolding ∗
apply (*rule has-integral-sub*)
unfolding *cbox-interval*
apply (*rule has-integral-const-real*)
apply (*rule has-integral-id*)
done
have *x* (*t* + *h*) − (*x t* + *h* $*_R$ *f* (*t*, *x t*)) ∈ *op* $*_R$ (h^2 / *2*) ‘ *F ′*
apply (*rule integral-by-parts-in-bounded-closed-convex-set*[*OF*
integral taylor[*unfolded interval-cbox*] *f ′-convex-compact*(*3*)
\-
f ′-convex-compact(*2*)[*THEN compact-imp-bounded*]
f ′-convex-compact(*2*)[*THEN compact-imp-closed*]
f ′-convex-compact(*1*)])
using *assms*
by (*auto intro*!: ‹*0* ≤ *h*› *simp*: *f-set-bounded*(*2*) *subset-eq*)
then show *?thesis* **by** (*simp add*: *discrete-evolution-def euler-increment*)
qed

lemma *euler-consistent-traj-set2*:
fixes *t*
assumes *ht*: *0* ≤ *h t1* ≤ *u*
assumes *T*: {*t*..*u*} ⊆ *T*
assumes *x′*: ⋀*s*. *s* ∈ {*t*..*u*} ⟹ (*x has-vector-derivative f* (*s*, *x s*)) (*at s within* {*t*..*u*})
assumes *x*: ⋀*s*. *s* ∈ {*t*..*u*} ⟹ *x s* ∈ *X*
assumes ∗: *t1* = *t* + *h*
shows *x t1* − *discrete-evolution* (*euler-increment f*) *t1 t* (*x t*) ∈ *op* $*_R$ (h^2 / *2*) ‘ *F ′*
using *ht T x′ x*
unfolding ∗
by (*rule euler-consistent-traj-set*)

end

lemma *numeral-6-eq-6*: *6* = *Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc 0*)))))
by *linarith*

context begin
interpretation *blinfun-syntax* **.**
lemma *rk2-consistent-traj-set*:
  fixes *x* ::*real* ⇒ *'a*::*banach* **and** *t*
  assumes *ht*: *0* ≤ *h* *t* + *h* ≤ *u*
  assumes *T*: {*t*..*u*} ⊆ *T* **and** *X0-nonempty*: *X0* ≠ {} **and** *X-nonempty*: *X* ≠ {} **and** *convex-X*: *convex X*
  assumes *x'*: ⋀*s*. *s* ∈ {*t*..*u*} ⟹ (*x has-vector-derivative f* (*s*, *x s*)) (*at s within* {*t*..*u*})
  assumes *f'*: ⋀*tx*. *tx* ∈ *T* × *X* ⟹ (*f has-derivative blinfun-apply* (*f' tx*)) (*at tx*)
  assumes *f''*: ⋀*tx*. *tx* ∈ *T* × *X* ⟹ (*f' has-derivative blinfun-apply* (*f'' tx*)) (*at tx*)
  assumes *f''-bounded*: *bounded* (*f''* ‘ (*T* × *X*))
  assumes *x*: ⋀*s*. *s* ∈ {*t*..*u*} ⟹ *x s* ∈ *X*
  assumes *f-set-bounded*: *bounded F* ⋀*t x x0*. *t*∈*T* ⟹ *x0* ∈ *X0* ⟹ *x*∈*X* ⟹ (*x0*, *x*, *f* (*t*, *x*)) ∈ *F*
  assumes *p*: *0* < *p* *p* ≤ *1*
  assumes *in-X0*: *x t* ∈ *X0*
  assumes *step-in*: *x t* + (*h* ∗ *p*) $*_R$ *f* (*t*, *x t*) ∈ *X*
  assumes *heun-remainder-bounded*:
    ⋀*x0 xt fxt s1 s2*. *s1* ∈ {*0* .. *1*} ⟹ *s2* ∈ {*0* .. *1*} ⟹ (*x0*, *xt*, *fxt*) ∈ *F* ⟹
      (*h* ˆ *3* / *6*) $*_R$
       (*f''* (*h* ∗ *s1* + *t*, *xt*) (*1*, *fxt*) (*1*, *fxt*) +
        f' (*h* ∗ *s1* + *t*, *xt*) (*0*, *f'* (*h* ∗ *s1* + *t*, *xt*) (*1*, *fxt*))) −
       (*h* ˆ *3* ∗ *p* / *4*) $*_R$
       f'' (*t* + *s2* ∗ (*h* ∗ *p*), *x0* + (*s2* ∗ (*h* ∗ *p*)) $*_R$ *f* (*t*, *x0*))
        (*1*, *f* (*t*, *x0*))
        (*1*, *f* (*t*, *x0*))
       ∈ *R*
  assumes *ccR*: *convex R closed R*
  shows *x* (*t* + *h*) − *discrete-evolution* (*rk2-increment p f*) (*t* + *h*) *t* (*x t*) ∈ *R*
proof *cases*
  assume *h* = *0*
  from *T ht* **have** *t* ∈ *T* **by** *auto*
  hence *F-nonempty*: *F* ≠ {} **using** *X-nonempty X0-nonempty*
    unfolding *ex-in-conv*[*symmetric*]
    by (*force intro*!: *f-set-bounded*)
  with *F-nonempty X-nonempty*
    ‹*h* = *0*›
    heun-remainder-bounded
  have *0* ∈ *R* **by** *force*
  from *this* ‹*h* = *0*› **show** *?thesis*
    by (*auto simp*: *discrete-evolution-def*)
next
  assume *h* ≠ *0*
  from *this ht* **have** *t* < *u* **by** *simp*
  have [*simp*]: *p* ≠ *0* **using** *p* **by** *simp*
  from ‹*h* ≥ *0*› ‹*h* ≠ *0*› **have** *h* > *0* **by** *simp*

let $?r = \lambda a.\ f''\ (t + a,\ x\ t + a *_R f\ (t,\ x\ t))\ (1,\ f\ (t,\ x\ t))$
$(1,\ f\ (t,\ x\ t))$
let $?q = \lambda s.\ f''\ (s,\ x\ s)\ (1,\ f\ (s,\ x\ s))\ (1,\ f\ (s,\ x\ s)) +$
$f'\ (s,\ x\ s)\ (0,\ f'\ (s,\ x\ s)\ (1,\ f\ (s,\ x\ s)))$

let $?d = \lambda tq\ tr.\ (h\hat{\ }3) *_R ((1/6) *_R\ ?q\ tq - (p\ /\ 4) *_R\ ?r\ tr)$

from *ht* **have** *line-subset*: $(\lambda ta.\ t + ta * h)$ ‘ $\{0..1\} \subseteq \{t..u\}$
by (*auto intro*!: *order-trans*[*OF add-left-mono*[*OF mult-left-le-one-le*]])
hence *line-in*: $\bigwedge s.\ 0 \leq s \Longrightarrow s \leq 1 \Longrightarrow t + s * h \in \{t..u\}$
by (*rule set-mp*) *auto*
from *ht* **have** *subset*: $\{t\ ..\ t + h\} \subseteq \{t\ ..\ u\}$ **by** *simp*
let $?T = \{t..u\}$
from *ht* **have** *subset*: $\{t\ ..\ t + h\} \subseteq \{t\ ..\ u\}$ **by** *simp*
from ‹$t < u$› **have** $t \in\ ?T$ **by** *auto*
from ‹$t < u$› **have** *tx*: $t \in T\ x\ t \in X$ **using** *T ht x* **by** *auto*

from *tx assms* **have** $0 \leq norm\ (f\ (t,\ x\ t))$ **by** *simp*
have *x-diff*: $\bigwedge s.\ s \in\ ?T \Longrightarrow x$ *differentiable at s within ?T*
by (*rule differentiableI*, *rule x'*[*simplified has-vector-derivative-def*])
let $?p = (\lambda t.\ f'\ (t,\ x\ t)\ (1,\ f\ (t,\ x\ t)))$
note *f'*[*derivative-intros*]
note *f''*[*derivative-intros*]
note *x'*[*simplified has-vector-derivative-def*, *derivative-intros*]

have *x-cont*: *continuous-on* $\{t..u\}$ *x*
by (*rule has-vector-derivative-continuous-on*) (*rule x'*)
have *f-cont*: *continuous-on* $(T \times X)$ *f*
apply (*rule has-derivative-continuous-on*)
apply (*rule has-derivative-at-within*)
by (*rule assms*)
have *f'-cont*: *continuous-on* $(T \times X)$ *f'*
apply (*rule has-derivative-continuous-on*)
apply (*rule has-derivative-at-within*)
by (*rule assms*)
note [*continuous-intros*] =
continuous-on-compose2[*OF x-cont*]
continuous-on-compose2[*OF f-cont*]
continuous-on-compose2[*OF f'-cont*]

from *f' f''*
have *f'-within*: $tx \in T \times X \Longrightarrow$ (*f has-derivative f' tx*) (*at tx within* $T \times X$)
and *f''-within*: $tx \in T \times X \Longrightarrow$ (*f' has-derivative f'' tx*) (*at tx within* $T \times X$) **for** *tx*
by (*auto intro*: *has-derivative-at-within*)

from *f''* **have** *f''-within*: $tx \in T \times X \Longrightarrow$ (*f' has-derivative op* \$ (*f'' tx*)) (*at tx within* $T \times X$) **for** *tx*

by (*auto intro*: *has-derivative-at-within*)
note [*derivative-intros*] =
has-derivative-in-compose2[*OF f'-within*]
has-derivative-in-compose2[*OF f''-within*]
have p': $\bigwedge s.\ s \in \{t\ ..\ u\} \Longrightarrow$ (*?p has-vector-derivative ?q s*) (*at s within ?T*)
unfolding *has-vector-derivative-def*
using *T x*
by (*auto intro*!: *derivative-eq-intros*
simp: *scale-back blinfun.bilinear-simps algebra-simps*
simp del: *scaleR-Pair*)
def *diff* $\equiv \lambda n$::*nat*. *if* $n = 0$ *then* x *else if* $n = 1$ *then* $\lambda t.\ f\ (t,\ x\ t)$ *else if* $n = 2$ *then ?p*
else ?q
have *diff-0*[*simp*]: *diff 0* = *x* **by** (*simp add*: *diff-def*)
{
fix *m*::*nat* **and** *ta*::*real*
assume *mta*: $m < 3$ $t \le ta$ $ta \le t + h$
have *image-subset*: ($\lambda xa.$ (*xa*, *x xa*)) ‘ {*t..u*} ⊆ {*t..u*} × *X*
using *assms* **by** *auto*
note *has-derivative-in-compose*[**where** *f*=($\lambda xa.$ (*xa*, *x xa*)) **and** *g* = *f*, *derivative-intros*]
note *has-derivative-subset*[*OF - image-subset*, *derivative-intros*]
note *f'*[*derivative-intros*]
note *x'*[*simplified has-vector-derivative-def*, *derivative-intros*]
have [*simp*]: $\bigwedge c\ x'.\ c *_R f'\ (ta,\ x\ ta)\ x' = f'\ (ta,\ x\ ta)\ (c *_R x')$
using *mta ht assms T x*
by (*force intro*!: *f' linear-cmul*[*symmetric*] *has-derivative-linear*)
have (($\lambda t.\ f\ (t,\ x\ t)$) *has-vector-derivative* f' (*ta*, *x ta*) (*1*, *f* (*ta*, *x ta*))) (*at ta within* {*t..u*})
unfolding *has-vector-derivative-def*
using *assms ht mta T x*
by (*force intro*!: *derivative-eq-intros has-derivative-within-subset*[*OF f'*])
hence (*diff m has-vector-derivative diff* (*Suc m*) *ta*) (*at ta within* {*t..t* + *h*})
using *mta ht*
by (*auto simp*: *diff-def intro*!: *has-vector-derivative-within-subset*[*OF - subset*] *x' p'*)
} **note** *diff* = *this*

from *taylor-has-integral*[*of 3 diff x t t* + *h*, *OF - - diff*]
have
(($\lambda x.$ (($t + h - x$) ˆ *2* / *2*) $*_R$ *diff 3 x*)
has-integral
$x\ (t + h) - x\ t - h *_R (f\ (t,\ x\ t)) - (h$ ˆ *2* / (*2*::*nat*)) $*_R$ (*?p t*))
(*cbox t* (*t* + *h*))
using *ht* ‹*h*≠*0*›
by (*auto simp*: *field-simps of-nat-Suc Pi-iff numeral-2-eq-2 numeral-3-eq-3*
numeral-6-eq-6 power2-eq-square diff-def scaleR-setsum-right)
from *has-integral-affinity*[*OF this* ‹*h* ≠ *0*›, *of t*, *simplified*]
have (($\lambda x.$ (($h - h * x)^2$ / *2*) $*_R$ *diff 3* ($h * x + t$)) *has-integral*
(*1* / |*h*|) $*_R$ ($x\ (t + h) - x\ t - h *_R f\ (t,\ x\ t) - (h^2$ / *2*) $*_R f'\ (t,\ x\ t)$ \$ (*1*,

f (*t*, *x t*))))
  ((λ*x*. *x* / *h* − *t* / *h*) ‘ {*t*..*t* + *h*})
  by *simp*
 also have ((λ*x*. *x* / *h* − *t* / *h*) ‘ {*t*..*t* + *h*}) = {*0* .. *1*}
  using ‹*h* ≠ *0*› ‹*h* ≥ *0*›
  by (*auto simp*: *divide-simps intro*!: *image-eqI*[**where** *x*=*x* ∗ *h* + *t* **for** *x*])
 finally have ((λ*x*. ((*h* − *h* ∗ *x*)^2 / *2*) ∗$_R$ *diff 3* (*h* ∗ *x* + *t*)) *has-integral*
 (*1* / |*h*|) ∗$_R$ (*x* (*t* + *h*) − *x t* − *h* ∗$_R$ *f* (*t*, *x t*) − (*h*^{2} / *2*) ∗$_R$ *f′* (*t*, *x t*) $ (*1*, *f* (*t*, *x t*))))
  {*0*..*1*} **.**
 from *has-integral-cmul*[*OF this*, *of h*]
 have *taylor*: ((λ*x*. (*1* − *x*)^2 ∗$_R$ ((*h*ˆ*3* / *2*) ∗$_R$ *?q* (*h* ∗ *x* + *t*))) *has-integral*
 (*x* (*t* + *h*) − *x t* − *h* ∗$_R$ *f* (*t*, *x t*) − (*h*^{2} / *2*) ∗$_R$ *f′* (*t*, *x t*) $ (*1*, *f* (*t*, *x t*))))
  {*0*..*1*} (**is** (*?i-taylor has-integral* -) -)
  using ‹*h* ≥ *0*› ‹*h* ≠ *0*›
 by (*simp add*: *diff-def divide-simps algebra-simps power2-eq-square power3-eq-cube*)
 have *line-in′*: *h* ∗ *y* + *t* ∈ *T*
  x (*h* ∗ *y* + *t*) ∈ *X*
  t ≤ *h* ∗ *y* + *t h* ∗ *y* + *t* ≤ *u*
  if *y* ∈ *cbox 0 1* **for** *y*
  using *line-in*[*of y*] *that T*
  by (*auto simp*: *algebra-simps x*)
 let *?integral* = λ*x*. *x*ˆ*3*/*3* − *x*ˆ*2* + *x*
 have *intsquare*: ((λ*x*. (*1* − *x*)^2) *has-integral ?integral 1* − *?integral 0*) (*cbox 0* (*1*::*real*))
  unfolding *cbox-interval*
  by (*rule fundamental-theorem-of-calculus*)
   (*auto intro*!: *derivative-eq-intros*
    simp: *has-vector-derivative-def power2-eq-square algebra-simps*)
 {
  fix *x h*::*real*∗′*a*
  assume *line-in*: (λ*s*. *x* + *s* ∗$_R$ *h*) ‘ {*0*..*1*} ⊆ *T* × *X*
  hence ∗: *y* ∈ *T* × *X* **if** *y* ∈ *closed-segment x* (*x* + *h*) **for** *y*
   using *that*
   by (*force simp*: *closed-segment-def algebra-simps*
    intro: *image-eqI*[**where** *x* = *1* − *x* **for** *x*])
  from *multivariate-taylor2*[*OF f′ f″*, *OF* ∗ ∗, *of x x* + *h*]
  have ((λ*s*. (*1* − *s*) ∗$_R$ *f″* (*x* + *s* ∗$_R$ *h*) *h h*) *has-integral f* (*x* + *h*) − *f x* − *f′ x* $ *h*) {*0*..*1*}
   by *simp*
 } **note** *f-taylor* = *this*

 let *?k* = λ*t*. *f* ((*t*, *x t*) + (*h* ∗ *p*) ∗$_R$ (*1*, *f* (*t*, *x t*)))

 have *line-in*: (λ*s*. (*t*, *x t*) + *s* ∗$_R$ ((*h* ∗ *p*) ∗$_R$ (*1*, *f* (*t*, *x t*)))) ‘ {*0*..*1*} ⊆ *T* × *X*
 proof (*clarsimp*, *safe*)
  fix *s*::*real* **assume** *s*: *0* ≤ *s s* ≤ *1*
  have *t* + *s* ∗ (*h* ∗ *p*) = *t* + *s* ∗ *p* ∗ *h*
   by (*simp add*: *ac-simps*)

```
    also have ... ∈ {t .. u}
      using ‹0 < p› ‹p ≤ 1› s
     by (intro line-in) (auto intro!: mult-nonneg-nonneg mult-left-le-one-le mult-le-oneI)
    also note ‹... ⊆ T›
    finally show t + s * (h * p) ∈ T .
    show x t + (s * (h * p)) *R f (t, x t) ∈ X
      using convexD-alt[OF ‹convex X› tx(2) step-in s]
      by (simp add: algebra-simps)
  qed
  from f-taylor[OF line-in, simplified]
  have k: ((λs. (1 − s) *R ((h^2 * p^2) *R
          f'' (t + s * (h * p), x t + (s * (h * p)) *R f (t, x t)) $
            (1, f (t, x t)) $
            (1, f (t, x t))))
       has-integral ?k t − f (t, x t) − f' (t, x t) $ (h * p, (h * p) *R f (t, x t)))
{0..1}
    (is (?i has-integral -) -)
    unfolding scale-back blinfun.bilinear-simps
    by (simp add: power2-eq-square algebra-simps)
  have rk2: discrete-evolution (rk2-increment p f) (t + h) t (x t) =
    x t + h *R f (t, x t) −
    (h / (2 * p)) *R f (t, x t) +
    (h / (p * 2)) *R ?k t
    (is - = ?rk2 t)
    unfolding rk2-increment-def discrete-evolution-def rk-increment-def
    apply (subst rk-eval.simps)
    supply rk-eval.simps[simp del]
    apply (simp add: eval-nat-numeral)
    apply (subst rk-eval.simps)
    apply (simp add: algebra-simps)
    done
  also have ... =
    x t + h *R f (t, x t) + (h / (2 * p)) *R (f' (t, x t) ((h * p), (h * p) *R f (t,
x t)))
      + (h / (p * 2)) *R integral {0 .. 1} ?i
    unfolding integral-unique[OF k]
    by (simp add: algebra-simps)
  also have (h / (2 * p)) *R f' (t, x t) (h * p, (h * p) *R f (t, x t)) = (h^2 / 2)
*R ?p t
    by (simp add: scale-back blinfun.bilinear-simps power2-eq-square
      del: scaleR-Pair)
  finally
  have integral {0 .. 1} ?i =
      (discrete-evolution (rk2-increment p f) (t + h) t (x t) −
        x t − h *R f (t, x t) −
        (h^2 / 2) *R ?p t) /R (h / (p * 2))
    by (simp add: blinfun.bilinear-simps zero-prod-def[symmetric])
  with - have (?i has-integral
    (discrete-evolution (rk2-increment p f) (t + h) t (x t) −
```

$x\ t - h *_R f\ (t, x\ t) -$
$(h^2\ /\ 2) *_R\ ?p\ t) /_R$
$(h\ /\ (p * 2)))\ \{0\ ..\ 1\}$
using *k*
by (*intro has-integral-integral-eqI*) (*rule has-integral-integrable*)
from *has-integral-cmul*[*OF this*, *of h* / (*p* ∗ *2*)]
have *discrete-taylor*:
((λ*s*. (*1* − *s*) $*_R$ ((*h*ˆ*3* ∗ *p* / *2*) $*_R$
f'' (*t* + *s* ∗ (*h* ∗ *p*), *x t* + (*s* ∗ (*h* ∗ *p*)) $*_R$ *f* (*t*, *x t*)) $
(*1*, *f* (*t*, *x t*)) $
(*1*, *f* (*t*, *x t*)))) *has-integral*
(*discrete-evolution* (*rk2-increment p f*) (*t* + *h*) *t* (*x t*) −
$x\ t - h *_R f\ (t, x\ t) -$
$(h^2\ /\ 2) *_R f'\ (t, x\ t)\ (1, f\ (t, x\ t))))\ \{0\ ..\ 1\}$
(**is** (*?i-dtaylor has-integral* -) -)
using ‹*h* > *0*›
by (*simp add*: *algebra-simps diff-divide-distrib power2-eq-square power3-eq-cube*)
have *integral-minus*: (*op* − *1 has-integral 1*/*2*) (*cbox 0* (*1*::*real*))
by (*auto intro*!: *has-integral-eq-rhs*[*OF has-integral-sub*] *has-integral-id*)

have *bounded-f*: *bounded* ((λ*xa*. *f* (*h* ∗ *xa* + *t*, *x* (*h* ∗ *xa* + *t*))) ‘ {*0*..*1*})
using ‹*0* ≤ *h*›
by (*auto intro*!: *compact-imp-bounded compact-continuous-image continuous-intros mult-nonneg-nonneg*
simp: *line-in'*)
have *bounded-f'*: *bounded* ((λ*xa*. *f'* (*h* ∗ *xa* + *t*, *x* (*h* ∗ *xa* + *t*))) ‘ {*0*..*1*})
using ‹*0* ≤ *h*›
by (*auto intro*!: *compact-imp-bounded compact-continuous-image continuous-intros*
simp: *line-in'*)
have *bounded-f''*: *bounded* ((λ*xa*. *f''* (*h* ∗ *xa* + *t*, *x* (*h* ∗ *xa* + *t*))) ‘ {*0*..*1*})
apply (*subst o-def*[*of f''*, *symmetric*])
apply (*subst image-comp*[*symmetric*])
apply (*rule bounded-subset*[*OF f''-bounded*])
by (*auto intro*!: *image-eqI line-in'*)
have *bounded-f''-2*:
bounded ((λ*xa*. *f''* (*t* + *xa* ∗ (*h* ∗ *p*), *x t* + (*xa* ∗ (*h* ∗ *p*)) $*_R$ *f* (*t*, *x t*))) ‘
{*0*..*1*})
apply (*subst o-def*[*of f''*, *symmetric*])
apply (*subst image-comp*[*symmetric*])
apply (*rule bounded-subset*[*OF f''-bounded*])
using *line-in*
by *auto*
have *1*: $x\ (t + h) - x\ t - h *_R f\ (t, x\ t) - (h^2\ /\ 2) *_R f'\ (t, x\ t)$ $ (*1*, *f* (*t*, *x t*))
∈ *op* $*_R$ (*1* / *3*) ‘
closure
(*convex hull*
(λ*xa*. (*h* ˆ *3* / *2*) $*_R$
(*f''* (*h* ∗ *xa* + *t*, *x* (*h* ∗ *xa* + *t*)) $

$(1,$
$f\ (h * xa + t,\ x\ (h * xa + t)))$ \$
$(1,$
$f\ (h * xa + t,\ x\ (h * xa + t))) +$
$f'\ (h * xa + t,\ x\ (h * xa + t))$ \$
$(0,$
$f'\ (h * xa + t,\ x\ (h * xa + t))$ \$
$(1,$
$f\ (h * xa + t,$
$x\ (h * xa + t))))))$ ‘
cbox 0 1)
by (*rule set-rev-mp*[*OF integral-by-parts-in-bounded-set*[*OF intsquare taylor*[*unfolded interval-cbox*]]])
(*auto intro*!: *bounded-scaleR-image bounded-plus-image*
bounded-blinfun-apply-image bounded-Pair-image
bounded-f'' bounded-f' bounded-f
simp: *image-constant*[*of 0*])
have *2*: *discrete-evolution* (*rk2-increment p f*) $(t + h)\ t\ (x\ t) -$
$x\ t - h *_R f\ (t,\ x\ t) - (h^2\ /\ 2) *_R f'\ (t,\ x\ t)$ \$ $(1,\ f\ (t,\ x\ t)) \in$
op $*_R$ (*1* / *2*) ‘ *closure* (*convex hull*
$(\lambda s.\ (h\ \hat{}\ 3 * p\ /\ 2) *_R$
f''
$(t + s * (h * p),$
$x\ t +$
$(s * (h * p)) *_R f\ (t,\ x\ t))$ \$
$(1,\ f\ (t,\ x\ t))$ \$
$(1,\ f\ (t,\ x\ t)))$ ‘
cbox 0 1)
by (*rule integral-by-parts-in-bounded-set*[*OF integral-minus discrete-taylor*[*unfolded interval-cbox*]])
(*auto intro*!: *bounded-scaleR-image bounded-blinfun-apply-image*
bounded-f''-2 simp: *image-constant*[*of 0*])
have $x\ (t + h) -$ *discrete-evolution* (*rk2-increment p f*) $(t + h)\ t\ (x\ t) \in$
$\{a - b|a\ b.$
$a \in$
closure
(*convex hull op* $*_R$ (*1/3*) ‘
$(\lambda xa.\ (h\ \hat{}\ 3\ /\ 2) *_R$
$(f''\ (h * xa + t,\ x\ (h * xa + t))$ \$
$(1,$
$f\ (h * xa + t,\ x\ (h * xa + t)))$ \$
$(1,$
$f\ (h * xa + t,\ x\ (h * xa + t))) +$
$f'\ (h * xa + t,\ x\ (h * xa + t))$ \$
$(0,$
$f'\ (h * xa + t,\ x\ (h * xa + t))$ \$
$(1,$
$f\ (h * xa + t,$
$x\ (h * xa + t))))))$ ‘

cbox 0 1) ∧
b ∈ *closure* (*convex hull op* $*_R$ (*1* / *2*) ‘
(λ*s*. (*h* ^ *3* ∗ *p* / *2*) $*_R$
f ′′
(*t* + *s* ∗ (*h* ∗ *p*),
x t +
(*s* ∗ (*h* ∗ *p*)) $*_R$ *f* (*t*, *x t*)) $
(*1*, *f* (*t*, *x t*)) $
(*1*, *f* (*t*, *x t*))) ‘
cbox 0 1)}
using *in-minus-Collect*[*OF 1 2*]
unfolding *closure-scaleR convex-hull-scaling*
by *auto*
also note *closure-minus-Collect*
also note *convex-hull-minus-Collect*
also have *closure*
(*convex hull*
{*xa* − *y* |*xa y*.
xa ∈ *op* $*_R$ (*1* / *3*) ‘
(λ*xa*. (*h* ^ *3* / *2*) $*_R$
(*f* ′′ (*h* ∗ *xa* + *t*, *x* (*h* ∗ *xa* + *t*)) $
(*1*, *f* (*h* ∗ *xa* + *t*, *x* (*h* ∗ *xa* + *t*))) $
(*1*, *f* (*h* ∗ *xa* + *t*, *x* (*h* ∗ *xa* + *t*))) +
f ′ (*h* ∗ *xa* + *t*, *x* (*h* ∗ *xa* + *t*)) $
(*0*, *f* ′ (*h* ∗ *xa* + *t*, *x* (*h* ∗ *xa* + *t*)) $
(*1*, *f* (*h* ∗ *xa* + *t*, *x* (*h* ∗ *xa* + *t*)))))) ‘
cbox 0 1 ∧
y ∈ *op* $*_R$ (*1* / *2*) ‘
(λ*s*. (*h* ^ *3* ∗ *p* / *2*) $*_R$
f ′′ (*t* + *s* ∗ (*h* ∗ *p*), *x t* + (*s* ∗ (*h* ∗ *p*)) $*_R$ *f* (*t*, *x t*)) $
(*1*, *f* (*t*, *x t*)) $
(*1*, *f* (*t*, *x t*))) ‘
cbox 0 1}) ⊆ *R*
apply (*rule closure-minimal*)
subgoal
by (*rule hull-minimal*)
(*auto intro*!: *heun-remainder-bounded f-set-bounded ccR line-in*′ *in-X0*)
subgoal by (*rule ccR*)
done
finally
show *x* (*t* + *h*) − *discrete-evolution* (*rk2-increment p f*) (*t* + *h*) *t* (*x t*) ∈ *R* **.**
qed

end

locale *derivative-norm-bounded* = *derivative-on-prod T X f f* ′ **for** *T* **and** *X*::′*a*::*euclidean-space set* **and** *f f* ′ +
fixes *B B*′
assumes *X-bounded*: *bounded X*

```
  assumes convex: convex T convex X
  assumes f-bounded: ⋀t x. t∈T ⟹ x∈X ⟹ norm (f (t,x)) ≤ B
  assumes f'-bounded: ⋀t x. t∈T ⟹ x∈X ⟹ onorm (f' (t,x)) ≤ B'
begin

lemma f-bound-nonneg: 0 ≤ B
proof −
  from nonempty obtain t x where t ∈ T x ∈ X by auto
  have 0 ≤ norm (f (t, x)) by simp
  also have ... ≤ B by (rule f-bounded) fact+
  finally show ?thesis .
qed

lemma f'-bound-nonneg: 0 ≤ B'
proof −
  from nonempty f-bounded ex-norm-eq-1[where 'a=real*'a]
    obtain t x and d::real*'a where tx: t ∈ T x ∈ X norm d = 1 by auto
  have 0 ≤ norm (f' (t, x) d) by simp
  also have ... ≤ B'
    using tx
    by (intro order-trans[OF onorm[OF has-derivative-bounded-linear[OF f']]])
      (auto intro!: f'-bounded f' has-derivative-linear)
  finally show ?thesis .
qed

sublocale g?: global-lipschitz - - - B'
proof
  fix t assume t ∈ T
  show lipschitz X (λx. f (t, x)) B'
  proof (rule lipschitzI)
    show 0 ≤ B' using f'-bound-nonneg .
    fix x y
    let ?I = T × X
    have convex ?I by (intro convex convex-Times)
    moreover have ∀x∈?I. (f has-derivative f' x) (at x within ?I) ∀x∈?I. onorm
(f' x) ≤ B'
      using f' f'-bounded
      by (auto simp add: intro!: f'-bounded has-derivative-linear)
    moreover assume x ∈ X y ∈ X
    with ‹t ∈ T› have (t, x) ∈ ?I (t, y) ∈ ?I by simp-all
    ultimately have norm (f (t, x) − f (t, y)) ≤ B' * norm ((t, x) − (t, y))
      by (rule differentiable-bound)
    thus dist (f (t, x)) (f (t, y)) ≤ B' * dist x y
      by (simp add: dist-norm norm-Pair)
  qed
qed

definition euler-C::real where euler-C = (sqrt DIM('a) * (B' * (B + 1) / 2))
```

lemma *euler-C-nonneg*: *euler-C* $\geq$ *0*
using *f-bounded f-bound-nonneg f'-bound-nonneg*
by (*simp add*: *euler-C-def*)

sublocale *derivative-set-bounded T X f f' X* $\times$ *cball 0 B*
cbox ($-$ (B' $*$ (B + *1*)) $*_R$ *One*) ((B' $*$ (B + *1*)) $*_R$ *One*)
proof
show *bounded* (X $\times$ *cball 0 B*) **using** *X-bounded* **by** (*auto intro*!: *bounded-Times*)
show *convex* (*cbox* ($-$(B' $*$ (B + *1*)) $*_R$ *One*) ((B' $*$ (B + *1*)) $*_R$ *One*::'*a*))
compact (*cbox* ($-$(B' $*$ (B + *1*)) $*_R$ *One*) ((B' $*$ (B + *1*)) $*_R$ *One*::'*a*))
by (*auto intro*!: *compact-cbox convex-box*)
fix *t x* **assume** $t \in T$ $x \in X$
thus (x, f (t, x)) $\in$ X $\times$ *cball 0 B*
by (*auto simp*: *dist-norm f-bounded*)
next
fix *t* **and** *x d*::'*a* **assume** $t \in T$ (x, d) $\in$ X $\times$ *cball 0 B*
hence $x \in X$ *norm* $d \leq B$ **by** (*auto simp*: *dist-norm*)
have *norm* (f' (t, x) (*1*, d)) $\leq$ *onorm* (f' (t, x)) $*$ *norm* (*1*::*real*, d)
by (*auto intro*!: *onorm has-derivative-bounded-linear f'* ‹$t \in T$› ‹$x \in X$›)
also have ... $\leq$ B' $*$ (B + *1*)
by (*auto intro*!: *mult-mono f'-bounded f-bounded* ‹$t \in T$› ‹$x \in X$› *f'-bound-nonneg*
order-trans[*OF norm-Pair-le*] ‹*norm* $d \leq B$›)
finally have f' (t, x) (*1*, d) $\in$ *cball 0* (B' $*$ (B + *1*))
by (*auto simp*: *dist-norm*)
also note *cball-in-cbox*
finally show f' (t, x) (*1*, d) $\in$ *cbox* ($-$ (B' $*$ (B + *1*)) $*_R$ *One*) ((B' $*$ (B + *1*)) $*_R$ *One*)
by *simp*
qed

lemma *euler-consistent-traj*:
fixes *t*
assumes *T*: $\{t..u\} \subseteq T$
assumes *x'*: $\bigwedge s.\ s \in \{t..u\} \Longrightarrow$ (*x has-vector-derivative f* (s, x s)) (*at s within* $\{t..u\}$)
assumes *x*: $\bigwedge s.\ s \in \{t..u\} \Longrightarrow x\ s \in X$
shows *consistent x t u euler-C 1* (*euler-increment f*)
proof
fix *h*::*real*
assume *ht*: $0 < h$ $t + h \leq u$ **hence** $t < u$ $0 < h^2 / 2$ **by** *simp-all*
from *euler-consistent-traj-set ht T x' x*
have x (t + h) $-$ *discrete-evolution* (*euler-increment f*) (t + h) t (x t) $\in$
op $*_R$ (h^2 / *2*) ‘ *cbox* ($-$ (B' $*$ (B + *1*)) $*_R$ *One*) ((B' $*$ (B + *1*)) $*_R$ *One*)
by *auto*
also have ... = *cbox* ($-$ ((h^2 / *2*) $*$ (B' $*$ (B + *1*))) $*_R$ *One*) (((h^2 / *2*) $*$ (B' $*$ (B + *1*))) $*_R$ *One*)
using *f-bound-nonneg f'-bound-nonneg*
by (*auto simp add*: *image-smult-cbox box-eq-empty mult-less-0-iff*)
also

note *centered-cbox-in-cball*
finally show *dist* (*x* (*t* + *h*)) (*discrete-evolution* (*euler-increment f*) (*t* + *h*) *t* (*x t*))
≤ *euler-C* ∗ *h* ^ (*1* + *1*)
by (*auto simp*: *euler-C-def dist-norm algebra-simps norm-minus-commute power2-eq-square*)
qed

end

locale *grid-from* = *grid* +
fixes *t0*
assumes *grid-min*: *t0* = *t 0*

locale *euler-consistent* =
has-solution i +
derivative-norm-bounded T X′ f B f′ B′
for *i*::′*a*::*euclidean-space ivp* **and** *t X′ B f′ B′* +
fixes *r e*
assumes *domain-subset*: *X* ⊆ *X′*
assumes *interval*: *T* = {*t0* − *e* .. *t0* + *e*}
assumes *lipschitz-area*: ⋀*t*. *t* ∈ *T* ⟹ *cball* (*solution t*) |*r*| ⊆ *X′*
begin

lemma *euler-consistent-solution*:
fixes *t′*
assumes *t′*: *t′* ∈ {*t0* .. *t0* + *e*}
shows *consistent solution t′* (*t0* + *e*) *euler-C 1* (*euler-increment f*)
proof (*rule euler-consistent-traj*)
show {*t′*..*t0* + *e*} ⊆ *T* **using** *t′ interval* **by** *simp*
fix *s*
assume *s* ∈ {*t′*..*t0* + *e*} **hence** *s* ∈ *T* **using** ‹{*t′*..*t0* + *e*} ⊆ *T*› **by** *auto*
show (*solution has-vector-derivative f* (*s*, *solution s*)) (*at s within* {*t′*..*t0* + *e*})
by (*rule has-vector-derivative-within-subset*[*OF* - ‹{*t′*..*t0* + *e*} ⊆ *T*›]) (*rule solution*(*2*)[*OF* ‹*s* ∈ *T*›])
have *solution s* ∈ *ivp-X i* **by** (*rule solution*(*3*)[*OF* ‹*s* ∈ *T*›])
thus *solution s* ∈ *X′* **using** *domain-subset* **..**
qed

end

sublocale *euler-consistent* ⊆
consistent-one-step t0 t0 + *e solution euler-increment f 1 euler-C r B′*
proof
show *0* < (*1*::*nat*) **by** *simp*
show *0* ≤ *euler-C* **using** *euler-C-nonneg* **by** *simp*
show *0* ≤ *B′* **using** *lipschitz-nonneg*[*OF lipschitz*] *iv-defined* **by** *simp*
fix *s x* **assume** *s*: *s* ∈ {*t0* .. *t0* + *e*}
show *consistent solution s* (*t0* + *e*) *euler-C 1* (*euler-increment f*)
using *interval s f-bounded f′-bounded f′*

strip
by (*intro euler-consistent-solution*) *auto*
fix *h*
assume *h*∈{*0..t0* + *e* − *s*}
have *lipschitz X′* (*euler-increment f h s*) *B′*
using *s lipschitz interval strip*
by (*auto intro*!: *euler-lipschitz*)
thus *lipschitz* (*cball* (*solution s*) |*r*|) (*euler-increment f h s*) *B′*
using *s interval*
by (*auto intro*: *lipschitz-subset*[*OF* - *lipschitz-area*])
qed

11.5 Euler method is convergent

locale *max-step1* = *grid* +
fixes *t1 L B r*
assumes *max-step*: ⋀*j*. *t j* ≤ *t1* ⟹ *max-stepsize j* ≤ |*r*| ∗ *L* / *B* / (*exp* (*L* ∗ (*t1* − *t 0*) + *1*) − *1*)

sublocale *max-step1* < *max-step?*: *max-step t t1 1 L B r*
using *max-step* **by** *unfold-locales simp-all*

locale *euler-convergent* =
euler-consistent + *max-step1 t t0* + *e B′ euler-C r* +
assumes *grid-from*: *t0* = *t 0*

sublocale *euler-convergent* ⊆
convergent-one-step t0 t0 + *e solution euler-increment f 1 euler-C r B′ t*
by *unfold-locales* (*simp add*: *grid-from*)

11.6 Euler method on Rectangle is convergent

locale *ivp-rectangle-bounded-derivative* = *solution-in-cylinder i*::′*a*::*euclidean-space ivp e b B* +
derivative-norm-bounded T cbox (*x0* − (*b* + |*r*|) ∗$_R$ *One*) (*x0* + (*b* + |*r*|) ∗$_R$ *One*) *f f′ B B′* **for** *i e b r B f′ B′*

sublocale *ivp-rectangle-bounded-derivative* ⊆ *unique-on-cylinder i e b B B′*
cbox (*x0* − (*b* + |*r*|) ∗$_R$ *One*) (*x0* + (*b* + |*r*|) ∗$_R$ *One*)
using *b-pos cball-in-cbox*[*of x0 b* + *abs r*]
by *unfold-locales* (*auto simp*: *cylinder intro*!: *scaleR-mono One-nonneg*)

sublocale *ivp-rectangle-bounded-derivative* ⊆
euler-consistent i t cbox (*x0* − (*b* + |*r*|) ∗$_R$ *One*) (*x0* + (*b* + |*r*|) ∗$_R$ *One*) *f′ B B′ r e*
proof
show *X* ⊆ *cbox* (*x0* − (*b* + |*r*|) ∗$_R$ *One*) (*x0* + (*b* + |*r*|) ∗$_R$ *One*) **using** *lipschitz-on-domain* **.**
fix *t* **assume** *t* ∈ *T*

have *cball* (*solution t*) |*r*| ⊆ *cball x0* (*b* + |*r*|)
  using *solution-in-D*[*of t*] *cylinder* ‹*t* ∈ *T*›
  by (*auto intro*: *cball-trans simp*: *interval*)
also note *cball-in-cbox*
finally show *cball* (*solution t*) |*r*| ⊆ *cbox* (*x0* − (*b* + |*r*|) $*_R$ *One*) (*x0* + (*b* + |*r*|) $*_R$ *One*) **.**
qed (*simp-all add*: *interval*)

locale *euler-on-rectangle* =
ivp-rectangle-bounded-derivative i e b r B f′ B′ +
grid-from t t0 +
max-step1 t t0 + *e B′ euler-C r*
for *i*::′*a*::*euclidean-space ivp* **and** *t e b r B f′ B′*

sublocale *euler-on-rectangle* ⊆
convergent?: *euler-convergent i t cbox* (*x0* − (*b* + |*r*|) $*_R$ *One*) (*x0* + (*b* + |*r*|) $*_R$ *One*) *f′ B B′ r e*
proof *unfold-locales*
qed (*rule grid-min*)

lemma *B* ≥ (*0*::*real*) ⟹ *0* ≤ (*exp* (*B* + *1*) − *1*) **by** (*simp add*: *algebra-simps*)

context *euler-on-rectangle* **begin**

lemma *convergence*:
assumes *t j* ≤ *t0* + *e*
shows *dist* (*solution* (*t j*)) (*euler f x0 t j*)
≤ *sqrt DIM*(′*a*) ∗ (*B* + *1*) / *2* ∗ (*exp* (*B′* ∗ *e* + *1*) − *1*) ∗ *max-stepsize j*
proof −
have *dist* (*solution* (*t j*)) (*euler f x0 t j*)
≤ *sqrt DIM*(′*a*) ∗ (*B* + *1*) / *2* ∗ *B′* / *B′* ∗ ((*exp* (*B′* ∗ *e* + *1*) − *1*) ∗ *max-stepsize j*)
using *assms convergence*[*OF assms*] *f′-bound-nonneg*
unfolding *euler-C-def*
by (*simp add*: *euler-def grid-min*[*symmetric*] *solution-t0 ac-simps*)
also have ... ≤ *sqrt DIM*(′*a*) ∗ (*B* + *1*) / *2* ∗ ((*exp* (*B′* ∗ *e* + *1*) − *1*) ∗ *max-stepsize j*)
using *f-bound-nonneg f′-bound-nonneg*
by (*auto intro*!: *mult-right-mono mult-nonneg-nonneg max-stepsize-nonneg add-nonneg-nonneg simp*: *le-diff-eq*)
finally show *?thesis* **by** *simp*
qed

end

11.7 Stability and Convergence of Approximate Euler

locale *euler-rounded-on-rectangle* =
ivp-rectangle-bounded-derivative i e1′ b r B f′ B′ +

grid?: *grid-from t t0′* +
max-step-r-2?: *max-step1 t t0* + *e2′ B′ euler-C r/2*
for *i*::′*a*::*executable-euclidean-space ivp* **and** *t* :: *nat* ⇒ *real* **and** *t0′ e1′ e2′*::*real*
and *x0′* :: ′*a*
and *b r B f′ B′* +
fixes *g*::(*real*×′*a*)⇒′*a* **and** *e*::*int*
assumes *t0-float*: *t0* = *t0′*
assumes *ordered-bounds*: *e1′* ≤ *e2′*
assumes *approx-f-e*: ⋀*j x*. *t j* ≤ *t0* + *e1′* ⟹ *dist* (*f* (*t j*, *x*)) ((*g* (*t j*, *x*))) ≤ *sqrt* (*DIM*(′*a*)) ∗ *2 powr* −*e*
assumes *initial-error*: *dist x0* (*x0′*) ≤ *euler-C* / *B′* ∗ (*exp 1* − *1*) ∗ *stepsize 0*
assumes *rounding-error*: ⋀*j*. *t j* ≤ *t0* + *e1′* ⟹ *sqrt* (*DIM*(′*a*)) ∗ *2 powr* −*e* ≤ *euler-C* / *2* ∗ *stepsize j*
begin

lemma *approx-f*: *t j* ≤ *t0* + *e1′* ⟹ *dist* (*f* (*t j*, *x*)) ((*g* (*t j*, *x*)))
≤ *euler-C* / *2* ∗ *stepsize j*
using *approx-f-e*[*of j x*] *rounding-error*[*of j*] **by** *auto*

lemma *t0-le*: *t 0* ≤ *t0* + *e1′*
unfolding *grid-min*[*symmetric*] *t0-float*[*symmetric*]
by (*metis atLeastAtMost-iff interval iv-defined*(*1*))

end

sublocale *euler-rounded-on-rectangle* ⊆ *grid′?*: *grid-from t t0′*
using *grid t0-float grid-min* **by** *unfold-locales auto*

sublocale *euler-rounded-on-rectangle* ⊆ *max-step-r?*: *max-step1 t t0* + *e2′ B′ euler-C r*
proof *unfold-locales*
fix *j*
assume (*t j*) ≤ *t0* + *e2′*
moreover with *grid-mono*[*of 0 j*] **have** *t 0* ≤ *t0* + *e2′* **by** (*simp add*: *less-eq-float-def*)
ultimately show *max-stepsize j*
≤ |*r*| ∗ *B′* / *euler-C* / (*exp* (*B′* ∗ (*t0* + *e2′* − (*t 0*)) + *1*) − *1*)
using *max-step-mono-r lipschitz B-nonneg f′-bound-nonneg*
by (*auto simp*: *less-eq-float-def euler-C-def mult-nonneg-nonneg*)
qed

lemma *max-step1-mono*:
assumes *t 0* ≤ *t1*
assumes *t1* ≤ *t2*
assumes *0* ≤ *a*
assumes *0* ≤ *b*
assumes *ms2*: *max-step1 t t2 a b c*
shows *max-step1 t t1 a b c*
proof −
interpret *t2*: *max-step1 t t2 a b c* **using** *ms2* **.**

```
  show ?thesis
  proof
    fix j
    assume t j ≤ t1 hence t j ≤ t2 using assms by simp
    hence t2.max-stepsize j ≤ |c| ∗ a / b / (exp (a ∗ (t2 − t 0) + 1) − 1) (is -
≤ ?x t2)
      by (rule t2.max-step)
    also have . . . ≤ ?x t1
      using assms
    by (cases b = 0) (auto intro!: divide-left-mono mult-mono abs-ge-zero add-increasing
        mult-pos-pos add-strict-increasing2 simp: le-diff-eq less-diff-eq)
    finally show t2.max-stepsize j ≤ ?x t1 .
  qed
qed

sublocale euler-rounded-on-rectangle ⊆ max-step-r1?: max-step1 t t0 + e1' B'
euler-C r
  by (rule max-step1-mono[of t, OF t0-le add-left-mono[OF ordered-bounds] f'-bound-nonneg
euler-C-nonneg])
    unfold-locales

sublocale euler-rounded-on-rectangle ⊆ c?: euler-on-rectangle i t e1' b r B f' B'
  using t0-float grid-min by unfold-locales simp

sublocale euler-rounded-on-rectangle ⊆
  consistent-one-step t 0 t0 + e1' solution euler-increment f 1 euler-C r B'
  using consistent-nonneg consistent lipschitz-nonneg lipschitz-incr t0-float grid-min
  by unfold-locales simp-all

sublocale euler-rounded-on-rectangle ⊆ max-step1 t t0 + e1' B' euler-C r / 2
  by (rule max-step1-mono[of t, OF t0-le add-left-mono[OF ordered-bounds] f'-bound-nonneg
euler-C-nonneg])
    unfold-locales

sublocale euler-rounded-on-rectangle ⊆
  one-step?:
  rounded-one-step t t0 + e1' solution euler-increment f 1 euler-C r B' euler-increment'
e g x0'
proof
  fix h j x assume t j ≤ t0 + e1'
  have dist (euler-increment f (h) (t j) (x))
        ((euler-increment' e g h (t j) x)) =
    dist (f (t j, x)) ((eucl-down e (g (t j, x))))
    by (simp add: euler-increment euler-float-increment)
  also
  have ... ≤
    dist (f (t j, x)) ((g (t j, x))) +
    dist ((g (t j, x))) ((eucl-down e (g (t j, x))))
    by (rule dist-triangle)
```

also
from *approx-f*[*OF* ‹*t j* ≤ *t0* + *e1′*›]
have *dist* (*f* (*t j*, *x*)) ((*g* (*t j*, *x*))) ≤
euler-C / *2* ∗ *stepsize j* **.**
also
from *eucl-truncate-down-correct*[*of g* (*t j*, *x*) *e*]
have *dist* ((*g* (*t j*, *x*))) ((*eucl-down e* (*g* (*t j*, *x*)))) ≤ *sqrt* (*DIM*(′*a*)) ∗ *2 powr* − *e* **by** *simp*
also
have *sqrt* (*DIM*(′*a*)) ∗ *2 powr* −*e* ≤ *euler-C* / *2* ∗ *stepsize j*
using *rounding-error* ‹*t j* ≤ *t0* + *e1′*› **.**
finally
have *dist* (*euler-increment f* (*h*) (*t j*) (*x*)) ((*euler-increment′ e g h* (*t j*) *x*)) ≤ *euler-C* ∗ *stepsize j*
by *arith*
thus *dist* (*euler-increment f h* (*t j*) (*x*)) ((*euler-increment′ e g h* (*t j*) *x*)) ≤ *euler-C* ∗ *stepsize j* ˆ *1*
by *simp*
qed (*insert initial-error grid-min solution-t0*, *simp-all*)

context *euler-rounded-on-rectangle* **begin**

lemma *stability*:
assumes *t j* ≤ *t0* + *e1′*
shows *dist* (*euler′ e g x0′ t j*) (*euler f x0 t j*) ≤
sqrt DIM(′*a*) ∗ (*B* + *1*) / *2* ∗ (*exp* (*B′* ∗ *e1′* + *1*) − *1*) ∗ *max-stepsize j*
proof −
have *dist* ((*euler′ e g x0′ t j*)) (*euler f x0 t j*) ≤
sqrt DIM(′*a*) ∗ (*B* + *1*) / *2* ∗ *B′* / *B′* ∗ (*exp* (*B′* ∗ *e1′* + *1*) − *1*) ∗ *max-stepsize j*
using *assms stability*[*OF assms*]
unfolding *grid-min*[*symmetric*] *solution-t0 euler-C-def*
by (*auto simp add*: *euler-def euler′-def t0-float*)
also have ... ≤ *sqrt DIM*(′*a*) ∗ (*B* + *1*) / *2* ∗ ((*exp* (*B′* ∗ *e1′* + *1*) − *1*) ∗ *max-stepsize j*)
using *f-bound-nonneg f′-bound-nonneg*
by (*auto intro*!: *mult-right-mono mult-nonneg-nonneg max-stepsize-nonneg add-nonneg-nonneg*
simp: *le-diff-eq*)
finally show *?thesis* **by** *simp*
qed

lemma *convergence-float*:
assumes *t j* ≤ *t0* + *e1′*
shows *dist* (*solution* (*t j*)) (*euler′ e g x0′ t j*) ≤
sqrt DIM(′*a*) ∗ (*B* + *1*) ∗ (*exp* (*B′* ∗ *e1′* + *1*) − *1*) ∗ *max-stepsize j*
proof −
have *dist* (*solution* ((*t j*))) ((*euler′ e g x0′ t j*)) ≤
dist (*solution* ((*t j*)))
(*euler f x0 t j*) +

dist ((*euler′ e g x0′ t j*)) (*euler f x0 t j*)
by (*rule dist-triangle2*)
also have *dist* (*solution* ((*t j*)))
(*euler f x0 t j*) ≤
sqrt DIM(′*a*) ∗ (*B* + *1*) / *2* ∗ (*exp* (*B′* ∗ *e1′* + *1*) − *1*) ∗ *max-stepsize j*
using *assms convergence*[*OF assms*] *t0-float* **by** *simp*
also have *dist* ((*euler′ e g x0′ t j*)) (*euler f x0 t j*) ≤
sqrt DIM(′*a*) ∗ (*B* + *1*) / *2* ∗ (*exp* (*B′* ∗ *e1′* + *1*) − *1*) ∗ *max-stepsize j*
using *assms stability* **by** *simp*
finally
have *dist* (*solution* ((*t j*))) ((*euler′ e g x0′ t j*))
≤ *sqrt DIM*(′*a*) ∗ (*B* + *1*) / *2* ∗ (*exp* (*B′* ∗ (*e1′*) + *1*) − *1*) ∗
max-stepsize j +
sqrt DIM(′*a*) ∗ (*B* + *1*) / *2* ∗ (*exp* (*B′* ∗ (*e1′*) + *1*) − *1*) ∗
max-stepsize j **by** *simp*
thus *?thesis* **by** (*simp add*: *field-simps*)
qed

end

end

12 Euler Method on Affine Forms: Code

theory *Euler-Affine-Code*
imports
Print
~~/src/HOL/Library/Monad-Syntax
~~/src/HOL/Library/While-Combinator
../Numerics/Runge-Kutta
../../Affine-Arithmetic/Affine-Arithmetic
begin

record (′*a*, ′*b*, ′*c*) *options* =
precision :: *nat*
tolerance :: *real*
stepsize :: *real*
min-stepsize :: *real*
iterations :: *nat*
halve-stepsizes :: *nat*
widening-mod :: *nat*
max-tdev-thres :: *real*
presplit-summary-tolerance :: *real*
collect-mod :: *nat*
collect-granularity :: *real*
override-section :: ′*a* ⇒ *real* ⇒ ′*a* ⇒ ′*a* ⇒ ′*a* ∗ *real*
global-section :: ′*b* ⇒ (′*a* ∗ *real*) *option*
stop-iteration :: ′*b* ⇒ *bool*
printing-fun :: *nat* ⇒ *real* ⇒ ′*b* ⇒ *unit*

result-fun :: *nat* ∗ *real* ∗ *'b* ∗ (*real* ∗ *'b* ∗ *real* ∗ *'b*) *list* ⇒ *'c*

locale *approximate-sets0* =
fixes *appr-of-ivl*::*'a*::{*ordered-euclidean-space*, *executable-euclidean-space*} ⇒ *'a* ⇒ *'b*
fixes *msum-appr*::*'b* ⇒ *'b* ⇒ *'b*
fixes *set-of-appr*::*'b* ⇒ *'a set*
fixes *set-of-apprs*::*'b list* ⇒ *'a list set*
fixes *inf-of-appr*::*'b* ⇒ *'a*
fixes *sup-of-appr*::*'b* ⇒ *'a*
fixes *add-appr*::(*'a*, *'b*, *'c*) *options* ⇒ *'b* ⇒ *'b* ⇒ *'b list* ⇒ *'b option*
fixes *scale-appr*::(*'a*, *'b*, *'c*) *options* ⇒ *real* ⇒ *real* ⇒ *'b* ⇒ *'b list* ⇒ *'b option*
fixes *scale-appr-ivl*::(*'a*, *'b*, *'c*) *options* ⇒ *real* ⇒ *real* ⇒ *'b* ⇒ *'b list* ⇒ *'b option*
fixes *split-appr*::(*'a*, *'b*, *'c*) *options* ⇒ *'b* ⇒ *'b list*
fixes *disjoint-apprs*::*'b* ⇒ *'b* ⇒ *bool*
fixes *inter-appr-plane*::*'b* ⇒ *'a* ⇒ *real* ⇒ *'b*
begin

TODO: more conceptual refinement?!

definition *ivl-appr-of-appr*::*'b* ⇒ *'b* **where**
ivl-appr-of-appr x = (*appr-of-ivl* (*inf-of-appr x*) (*sup-of-appr x*))

end

declare *approximate-sets0.ivl-appr-of-appr-def*[*code*]

type-synonym *'a enclosure* = *nat* ∗ *real* ∗ *'a* ∗ (*real* ∗ *'a* ∗ *real* ∗ *'a*) *list*

locale *approximate-ivp0* = *approximate-sets0*
appr-of-ivl msum-appr set-of-appr set-of-apprs inf-of-appr sup-of-appr add-appr scale-appr scale-appr-ivl
split-appr disjoint-apprs inter-appr-plane
for *appr-of-ivl msum-appr set-of-appr set-of-apprs inf-of-appr*
and *sup-of-appr*::*'b* ⇒ *'a*::{*ordered-euclidean-space*, *executable-euclidean-space*}
and *add-appr*:: (*'a*, *'b*, *'c*) *options* ⇒ *'b* ⇒ *'b* ⇒ *'b list* ⇒ *'b option*
and *scale-appr scale-appr-ivl split-appr disjoint-apprs inter-appr-plane* +
fixes *ode-approx*::(*'a*, *'b*, *'c*) *options* ⇒ *'b list* ⇒ *'b option*
fixes *ode-d-approx*:: (*'a*, *'b*, *'c*) *options* ⇒ *'b list* ⇒ *'b option*
begin

abbreviation *extend-appr* ≡ *λx l u. msum-appr x* (*appr-of-ivl l u*)

definition *P-appr*::(*'a*, *'b*, *'c*) *options* ⇒ *'b* ⇒ *real* ⇒ *'b* ⇒ *'b option* **where**
P-appr optns X0 h X = *map-option* (*λY*.
extend-appr X0 (*inf 0* ($h *_R$ *inf-of-appr Y*))
(*sup 0* ($h *_R$ *sup-of-appr Y*)))
(*ode-approx optns* [*X*])

fun *P-iter*::(*'a*, *'b*, *'c*) *options* ⇒ *'b* ⇒ *real* ⇒ *nat* ⇒ *'b* ⇒ *'b option* **where**

P-iter optns X0 h 0 X =
(*let* - = *print* (*STR* ''=P-iter failed: '');
- = *print-eucl* (*inf-of-appr X*);
- = *print* (*STR* '' − '');
- = *print-eucl* (*sup-of-appr X*);
- = *println* (*STR* '''') *in None*)
| *P-iter optns X0 h* (*Suc i*) *X* =
bind-err (*STR* ''=P-appr failed'') (*P-appr optns X0 h X*) (λ*X'*.
let (*l'*, *u'*) = (*inf-of-appr X'*, *sup-of-appr X'*) *in*
let (*l*, *u*) = (*inf-of-appr X*, *sup-of-appr X*) *in*
if l ≤ *l'* ∧ *u'* ≤ *u then Some X*
else P-iter optns X0 h i (*appr-of-ivl* (*inf l' l* − (*if i mod* (*widening-mod optns*) = *0 then abs* (*l'* − *l*) *else 0*)) (*sup u' u* + (*if i mod widening-mod optns* = *0 then abs* (*u'* − *u*) *else 0*))))

fun *cert-stepsize*::(*'a*, *'b*, *'c*) *options* ⇒ *'b* ⇒ *real* ⇒ *nat* ⇒ *nat* ⇒ (*real* ∗ *'b*) *option* **where**
cert-stepsize optns X0 h n 0 = (*let* - = *println* (*STR* ''=cert-stepsize failed'') *in None*)
| *cert-stepsize optns X0 h n* (*Suc i*) =
(*case P-iter optns* (*ivl-appr-of-appr X0*) *h n* (*ivl-appr-of-appr X0*) *of*
Some X' ⇒ *Some* (*h*, *X'*)
| *None* ⇒
let
- = *print* (*STR* ''=cert-stepsize failed on: '');
- = *print-eucl* (*inf-of-appr X0*);
- = *print* (*STR* '' − '');
- = *print-eucl* (*sup-of-appr X0*);
- = *println* (*STR* '''')
in cert-stepsize optns X0 (*h* / *2*) *n i*)

lemma *cert-stepsize-pos*: *cert-stepsize optns X0 h n i* = *Some* (*h'*, *cx*) ⟹ *h* > *0* ⟹ *h'* > *0*
by (*induct i arbitrary*: *h h'*) (*auto split*: *option.split-asm*)

definition *euler-step optns X0* =
bind-err (*STR* ''=certify stepsize failed'') (*cert-stepsize optns X0* (*stepsize optns*) (*iterations optns*) (*halve-stepsizes optns*))
(λ(*h*, *CX*).
bind-err (*STR* ''=ode-approx X0 failed'') (*ode-approx optns* [*X0*])
(λ*X0'*.
bind-err (*STR* ''=ode-approx CX failed'') (*ode-approx optns* [*CX*])
(λ*F*.
bind-err (*STR* ''=ode-d-approx failed'') (*ode-d-approx optns* [*CX*, *F*])
(λ*D*.
bind-err (*STR* ''=scale-appr err failed'') (*scale-appr optns* (*h*∗*h*) *2* (*ivl-appr-of-appr D*) [])
(λ*ERR*.
bind-err (*STR* ''=scale-appr euler failed'') (*scale-appr optns h 1 X0'*

```
[X0])
               (λS.
                 bind-err (STR ''=scale-appr-ivl euler failed'') (scale-appr-ivl optns
0 h X0' [X0])
                  (λS'.
                  bind-err (STR ''=add-appr euler failed'') (add-appr optns X0 S [])
                    (λX1.
                     bind-err (STR ''=add-appr euler-ivl failed'') (add-appr optns X0
S' [])
                      (λCX1.
                        let
                          res = msum-appr X1 (ivl-appr-of-appr ERR);
                             res-ivl = msum-appr CX1 (appr-of-ivl (inf 0 (inf-of-appr
ERR)) (sup 0 (sup-of-appr ERR)))
                        in
                        Some (h, res-ivl, res))))))))))
```

```
fun advance-euler::('a, 'b, 'c) options ⇒ 'b enclosure option ⇒ 'b enclosure option
where
  advance-euler optns None = None
| advance-euler optns (Some (i, t, X, XS)) =
    (case euler-step optns X of
      Some (h, CX, X1) ⇒
        let - = printing-fun optns i (t + h) X1
        in Some (Suc i, t + h, X1, (t, CX, t + h, X1)#XS)
    | None ⇒ None)
```

```
primrec euler-series where
  euler-series optns t0 X0 0 = Some (0, t0, X0, [])
| euler-series optns t0 X0 (Suc i) = advance-euler optns (euler-series optns t0 X0
i)
```

12.1 Checkpoint: Partition

TODO: partitioning really needed when we do cancelling?

```
definition width-appr::'b ⇒ real
  where width-appr x = infnorm (sup-of-appr x − inf-of-appr x)
```

```
primrec split-appr-fp-iter where
  split-appr-fp-iter optns XS 0 = XS
| split-appr-fp-iter optns XS (Suc i) =
    (let
      YS = concat (map (λx.
        if width-appr x > max-tdev-thres optns
        then split-appr optns x
        else [x]) XS)
    in
      if length XS = length YS
      then YS
```

```
    else split-appr-fp-iter optns YS i)

definition split-appr-fp optns X = split-appr-fp-iter optns [X] 100

primrec ivl-of-apprs::'b list ⇒ 'b
  where
  ivl-of-apprs (x#xs) = (let
      i = fold (λa b. inf (inf-of-appr a) b) xs (inf-of-appr x);
      s = fold (λa b. sup (sup-of-appr a) b) xs (sup-of-appr x)
    in appr-of-ivl i s)

definition partition
  where
 partition optns r xs =
    (let
      rs = split-appr-fp (optns(|max-tdev-thres := collect-granularity optns|)) r;
      red-rs = map (λr.
        case (filter (λa. ¬ disjoint-apprs a r) xs) of [] ⇒ None
          | ds ⇒ let d = ivl-of-apprs ds
          in Some (appr-of-ivl (sup (inf-of-appr r) (inf-of-appr d)) (inf (sup-of-appr
r) (sup-of-appr d)))) rs
    in map the (filter (−Option.is-none) red-rs))

definition collect-apprs::- ⇒ 'b ⇒ 'b list ⇒ 'b list
  where collect-apprs optns r XS =
    (let
      - = print (STR ''= Collecting: '');
      - = print (int-to-string (length XS));
      - = print (STR ''aforms ... '');
      YS = partition optns r XS;
      - = print (STR ''= Collected to  '');
      - = print (int-to-string (length YS));
      - = println (STR '' aforms!'')
    in YS)

definition collect-cancel-apprs::('a, 'b, 'c) options ⇒ nat ⇒ 'b list ⇒ 'b list
  where
  collect-cancel-apprs optns i xs =
    (if i mod collect-mod optns = 0
    then let
      - = println (STR ''Collect-cancelling:'');
      r = ivl-of-apprs xs;
      checkpoint-grid = collect-apprs optns r xs;
      - = println (STR ''checkpoint grid:'');
      - = map (λx. printing-fun optns i 0 x) checkpoint-grid;
      steps = map (λx. snd (snd (the (euler-step optns x)))) checkpoint-grid;
      - = println (STR ''steps:'');
      - = map (λx. printing-fun optns i 0 x) steps;
      outside-checkpoint =
```

filter (*λx*. ¬ (*inf-of-appr r* ≤ *inf-of-appr x* ∧ *sup-of-appr x* ≤ *sup-of-appr r*)) *steps*;
- = *println* (*STR* ''*outside-checkpoint*:'');
- = *map* (*λx*. *printing-fun optns i 0 x*) *outside-checkpoint*;
inside-checkpoint =
filter (*λx*. (*inf-of-appr r* ≤ *inf-of-appr x* ∧ *sup-of-appr x* ≤ *sup-of-appr r*)) *steps*;
- = *println* (*STR* ''*inside-checkpoint*:'');
- = *map* (*λx*. *printing-fun optns i 0 x*) *inside-checkpoint*;
steps-grid = *collect-apprs optns r steps*;
- = *println* (*STR* ''*steps-grid*'');
- = *map* (*λx*. *printing-fun optns i 0 x*) *steps-grid*;
sg-not-cp-covered = *fold* (*λx xs*. *removeAll x xs*) *checkpoint-grid steps-grid*;
- = *println* (*STR* ''*sg-not-cp-covered*'');
- = *map* (*λx*. *printing-fun optns i 0 x*) *sg-not-cp-covered*;
s-not-covered = *filter* (*λx*. *list-ex* (− *disjoint-apprs x*) *sg-not-cp-covered*) *steps*;
- = *println* (*STR* ''*s-not-covered*'');
- = *map* (*λx*. *printing-fun optns i 0 x*) *s-not-covered*
in remdups (*outside-checkpoint* @ *s-not-covered*)
else xs)

TODO: certify common stepsize first, and establish a common history of disjunctions of zonotopes

fun *map-enclosure-option*::
(*nat* ⇒ *real* ⇒ '*b* ⇒ '*b list*) ⇒ '*b enclosure option list* ⇒ '*b enclosure option list*
where
map-enclosure-option f [] = []
| *map-enclosure-option f* (*None*#*xs*) = (*None* # *map-enclosure-option f xs*)
| *map-enclosure-option f* (*Some* (*i*, *t*, *X*, *XS*)#*xs*) = *map* (*λX*. *Some* (*i*, *t*, *X*, *XS*)) (*f i t X*) @ *map-enclosure-option f xs*

definition *euler-lists optns t0 X0 t1* =
while-option (*list-ex* (*λx*. *case x of Some* (*i*, *t*, *X*, *XS*) ⇒ *t* < *t1* | *None* ⇒ *False*))
((*λx*. *let* - = *print* (*STR* ''*Affine Forms*: ''); - = *println* (*int-to-string* (*length x*)) *in x*) *o*
(*map-enclosure-option* (*λi t x*. *split-appr-fp optns x*)) *o*
(*λx*. *case x of Some* (*i*, *t*, *X*, *XS*)#- ⇒ *map* (*λx*. *Some* (*i*, *t*, *x*, *XS*)) (*collect-cancel-apprs optns i* (*map* (*fst o snd o snd o the*) *x*))
| *None*#- ⇒ []) *o*
map (*advance-euler optns*)) [*Some* (*0*, *t0*, *X0*, [])]

definition *euler-lists-result optns t0 X0 t1* =
map-option (*map* (*map-option* (*result-fun optns*))) (*euler-lists optns t0 X0 t1*)

definition *euler-series-result*::
('*a*, '*b*, '*c*) *options* ⇒ *real* ⇒ '*b* ⇒ *nat* ⇒ '*c option*
where [*simp*]: *euler-series-result optns t0 X0 i* =
map-option (*result-fun optns*) (*euler-series optns t0 X0 i*)

lemma *euler-series-print*:
euler-series optns t0 X0 i =
fold (λ*a b*.
case b of
None ⇒ *None*
| *Some* (*a*′, *t0*′, *X0*′, *ress*) ⇒
(*case euler-step optns X0*′ *of*
None ⇒ *None*
| *Some* (*h*, *CX*, *X1*) ⇒
let
- = *printing-fun optns a* (*t0*′ + *h*) *X1*
in Some (*Suc a*′, *t0*′ + *h*, *X1*, (*t0*′, *CX*, *t0*′ + *h*, *X1*)#*ress*))) [*0*..<*i*]
(*Some* (*0*, *t0*, *X0*, []))
unfolding *Let-def*
by (*induct i*) (*auto split*: *option.split*)

definition *project-rect X b y* =
(*let i* = *inf-of-appr X*; *s* = *sup-of-appr X in*
appr-of-ivl (*i* + (*y* − *i* · *b*) $*_R$ *b*) (*s* + (*y* − *s* · *b*) $*_R$ *b*))

definition *sup-abs-appr X* = *sup* (*abs* (*inf-of-appr X*)) (*abs*(*sup-of-appr X*))

definition *intersects X b y* ⟷ (*inf-of-appr X* · *b* ≤ *y* ∧ *sup-of-appr X* · *b* ≥ *y*)

Precondition: X does not intersect b, but euler-step does!

primrec *intersect*′
where
intersect′ *optns X b y h 0* = *None*
| *intersect*′ *optns X b y h* (*Suc i*) =
(*let*
(*h*, *CX*, *X1*) = *the* (*euler-step* (*optns*(|*stepsize*:=*h*|)) *X*)
in if intersects X1 b y then intersect′ *optns X b y* (*h*∗*2*) *i else Some* (*inter-appr-plane CX b y*))

definition *intersect optns X b y* = *intersect*′ *optns X b y* (*stepsize optns*) *10*

definition *poincares2-step optns X0 b y* =
(*let*
(*h*, *CX*, *X1*) = *the* (*euler-step optns X0*)
in
if intersects CX b y
then the (*intersect optns X0 b y*)
else X1
)

definition *strongest-direction optns f* =
(*let*
af = *sup-abs-appr f*;

(*b*, -) = *fold* (λ*b* (*b'*, *d'*). *if d'* ≤ *af* · *b then* (*b*, *af* · *b*) *else* (*b'*, *d'*)) (*Basis-list*::'*a list*) *0*;
res = (*if inf-of-appr f* · *b* < *0* ∧ *sup-of-appr f* · *b* < *0 then* (*b*, *inf-of-appr f* · *b*)
else if inf-of-appr f · *b* > *0* ∧ *sup-of-appr f* · *b* > *0 then* (*b*, ((*sup-of-appr f* · *b*)))
else let - = *println* (*STR* ''== *ERROR finding next direction*!'') *in* (*0*, *0*))
in res)

definition *next-sections optns d Xs* =
(*let*
set-dir-alist = *map* (λ*X*. (*X*, *apsnd sgn* (*strongest-direction optns* (*the* (*ode-approx optns* [*X*]))))) *Xs*;
dirs = *remdups* (*map snd set-dir-alist*);
dir-set-alist = *map* (λ*bs*. (*bs*, *map fst* (*filter* (λ(*X*, *b*, *s*). (*b*, *s*) = *bs*) *set-dir-alist*))) *dirs*;
sctns = *map* (λ((*b*, *s*), *Xs*). *if s* = −*1 then* (*Xs*, (*b*, *inf-of-appr* (*ivl-of-apprs Xs*) · *b* − *d*))
else (*Xs*, (−*b*, − (*sup-of-appr* (*ivl-of-apprs Xs*) · *b* + *d*)))) *dir-set-alist*
in
map (λ(*Xs*, (*b*, *s*)). (*Xs*, *override-section optns b s* (*inf-of-appr* (*ivl-of-apprs Xs*)) (*sup-of-appr* (*ivl-of-apprs Xs*)))) *sctns*)

definition *poincares2-iter optns X0 b y* =
while (*list-ex* (λ(*X*, *b*, *y*). ¬*stop-iteration optns X*))
(*concat o* (*map* (λ(*X*, *b*, *y*).
let
F = *the* (*ode-approx optns* [*X*]);
(*bs*, *fs*) = *strongest-direction optns F*;
(*b*, *y*) = (*if bs* = *b then* (*b*, *y*)
else if fs ≤ *0* ∧ *fs* ∗ *3* ≤ *4* ∗ ((*inf-of-appr F*) · *b*) *then* (*bs*, *inf-of-appr X* · *b*)
else if fs ≥ *0* ∧ *4* ∗ ((*sup-of-appr F*) · *b*) ≤ *fs* ∗ *3 then* (*bs*, *sup-of-appr X* · *b*)
else (*b*, *y*));
(*b*, *y*) = (*case global-section optns X of None* ⇒ (*b*, *y*)
| *Some* (*b*, *y*) ⇒ (*b*, *y*));
X1 = *poincares2-step optns X b y*;
X1s = *split-appr-fp optns X1*
in map (λ*X*. (*X*, *b*, *y*)) *X1s*)))

definition *poincares optns X0s b y* =
while (λ(*XS*, *PS*, *RS*). *XS* ≠ [])
(λ(*XS*, *PS*, *RS*).
let
- = *print* (*STR* ''=*XS*: '');
- = *print* (*int-to-string* (*length XS*));
- = *print* (*STR* '' *PS*: '');
- = *print* (*int-to-string* (*length PS*));

- = *print* (*STR* '' *RS*: '');
- = *print* (*int-to-string* (*length RS*));
- = *print* (*STR* '' *Flowing towards*: '');
- = *print-eucl b*;
- = *print* (*STR* '' −− '');
- = *print-real y*;
- = *println* (*STR* '''');
XS = *concat* (*map* (λ(*h*, *X*). *map* (*Pair h*) (*split-appr-fp optns X*)) *XS*);
XS = *filter* (λ(*h*, *X*). *inf-of-appr X* · *b* ≥ *y* ∨ *sup-of-appr X* · *b* ≥ *y*) *XS*;
- = *print* (*STR* ''=*XS above*: '');
- = *println* (*int-to-string* (*length XS*));
- = *map* (*printing-fun optns 0 0 o snd*) *XS*;
YS = *map* (λ(*h*, *X*). *case* (*euler-step* (*optns*(|*stepsize*:=*h*|)) *X*) *of Some res* ⇒ (*h*, *X*, *res*) | *None* ⇒ *undefined*) *XS*;
(*IS*, *NIS*) = *List.partition* (λ(*h*, *X0*, *t*, *CX*, *X*). (*inf-of-appr X* · *b* ≤ *y* ∨ *sup-of-appr X* · *b* ≤ *y*)) *YS*;
(*RS'*, *NIS*) = *List.partition* (λ(-, -, -, -, *X*).
list-ex (λ*b'*. *b* ≠ *b'* ∧
(*let sa* = *sup-abs-appr* (*the* (*ode-approx optns* [*X*])) *in abs* (*sa* · *b*) ∗ *4* ≤ *3* ∗ *abs* (*sa* · *b'*)))
Basis-list) *NIS*;
XS' = ''*concat* (*map* (%*X*. *split-appr-fp* (*optns*(|*max-tdev-thres*:=*collect-granularity optns*|)) *X*) (*map fst IS*)'';
(*IS1*, *IS2*) = *List.partition* (λ(*h*, *X0*, *t*, *CX*, *X*). *h* ≤ *min-stepsize optns*) *IS*;
IS2' = (*map* (λ(*h*, *X0*, *t*, *CX*, *X*). (*h* / *2*, *X0*)) *IS2*);
QS = *map* (λ(*h*, *X0*, *t*, *CX*, *X*). *project-rect CX b y*) *IS1*
in (*map* (λ(*h*, *X0*, *t*, *CX*, *X*). (*h*, *X*)) (*NIS* @ *IS1*) @ *IS2'*, *PS*@*QS*, *RS* @ *map* (*snd o snd o snd o snd*)*RS'*)
)
(*map* (λ*X*. (*stepsize optns*, *X*)) *X0s*, [], []) — Verbindung mit Euler, parametrisiert mit h!

definition *poincares-collected optns X0s b y* =
(*case snd* (*poincares optns X0s b y*) *of* ([], *RS*) ⇒ ([], *RS*)
| (*PS*, *RS*) ⇒ (*collect-apprs optns* (*ivl-of-apprs PS*) *PS*, *RS*))

definition *print-poincares optns X0s b y* =
(*let* (*qs*, *rs*) = *poincares-collected optns X0s b y*;
- = *map* (*printing-fun optns 0 0*) *qs*
in (*qs*, *rs*))

definition *poincare-distance-d optns X0s* =
while (*list-ex* (λ(*XS*, *b*, *y*). *b* ≠ *0*))
(λ*groups*. *let* - = *print* (*STR* ''= *Groups*: ''); - = *println* (*int-to-string* (*length groups*)) *in concat* (*map* (λ(*Xs*, *b*, *y*).
if b = *0 then* [(*Xs*, *b*, *y*)] *else*
let
(*Ys*, *Rs*) = *print-poincares optns Xs b y*;

```
        Yss = next-sections optns 2 Ys;
        Rss = next-sections optns 0 Rs;
        - = print (STR ''= Ys: '');
        - = print (int-to-string (length Yss));
        - = print (STR '' Rss: '');
        - = print (int-to-string (length Rss))
      in
        Yss@Rss) groups)
    ) (next-sections optns 2 X0s)
```

definition *poincare-distance-d-print optns X0s* =

```
  (let
    res = poincare-distance-d optns X0s;
    - = print (STR ''= Returning: '');
    - = print (int-to-string (length res));
    - = println (STR '''');
    - = map (printing-fun optns 0 0) (concat (map fst res))
  in res)
```

end

declare *approximate-ivp0.strongest-direction-def*[*code*]
declare *approximate-ivp0.poincares2-iter-def*[*code*]
declare *approximate-ivp0.poincares2-step-def*[*code*]
declare *approximate-ivp0.intersect-def*[*code*]
declare *approximate-ivp0.intersect'.simps*[*code*]
declare *approximate-ivp0.intersects-def*[*code*]
declare *approximate-ivp0.sup-abs-appr-def*[*code*]
declare *approximate-ivp0.project-rect-def*[*code*]
declare *approximate-ivp0.poincares-def*[*code*]
declare *approximate-ivp0.poincare-distance-d-def*[*code*]
declare *approximate-ivp0.poincare-distance-d-print-def*[*code*]
declare *approximate-ivp0.next-sections-def*[*code*]
declare *approximate-ivp0.poincares-collected-def*[*code*]
declare *approximate-ivp0.print-poincares-def*[*code*]
declare *approximate-ivp0.P-appr-def*[*code*]
declare *approximate-ivp0.P-iter.simps*[*code*]
declare *approximate-ivp0.cert-stepsize.simps*[*code*]
declare *approximate-ivp0.euler-step-def*[*code*]
declare *approximate-ivp0.advance-euler.simps*[*code*]
declare *approximate-ivp0.collect-cancel-apprs-def*[*code*]
declare *approximate-ivp0.euler-series-result-def*[*code*]
declare *approximate-ivp0.map-enclosure-option.simps*[*code*]
declare *approximate-ivp0.euler-lists-def*[*code*]
declare *approximate-ivp0.euler-lists-result-def*[*code*]
declare *approximate-ivp0.euler-series-print*[*code*]
declare *approximate-ivp0.collect-apprs-def*[*code*]
declare *approximate-ivp0.ivl-of-apprs.simps*[*code*]
declare *approximate-ivp0.partition-def*[*code*]

declare *approximate-ivp0.split-appr-fp-iter.simps*[*code*]
declare *approximate-ivp0.split-appr-fp-def*[*code*]
declare *approximate-ivp0.width-appr-def*[*code*]

abbreviation *msum-aform'* ≡ λ*X*. *msum-aform* (*degree-aform X*) *X*

abbreviation *uncurry-options* ≡ λ*f x*. *f* (*precision x*) (*tolerance x*)

intersection with plane

definition *inter-aform-plane* **where**
inter-aform-plane X b y = *X*

locale *aform-approximate-sets0* =
approximate-sets0
aform-of-ivl msum-aform' Affine Joints
Inf-aform Sup-aform
uncurry-options add-aform-componentwise::('*a*, '*a*::*executable-euclidean-space aform*, (*real* × ((*real* × '*a* × '*a* × *real* × '*a* × '*a*) *list*))) *options* ⇒ -
uncurry-options scaleQ-aform-componentwise
uncurry-options scaleR-aform-ivl
λ*optns*. *split-aform-largest* (*precision optns*) (*presplit-summary-tolerance optns*)
disjoint-aforms
inter-aform-plane

interpretation *aform-approximate-sets0* **.**

locale *aform-approximate-ivp0* =
approximate-ivp0
aform-of-ivl msum-aform' Affine Joints
Inf-aform Sup-aform
uncurry-options add-aform-componentwise::('*a*, '*a*::*executable-euclidean-space aform*, (*real* × ((*real* × '*a* × '*a* × *real* × '*a* × '*a*) *list*))) *options* ⇒ -
uncurry-options scaleQ-aform-componentwise
uncurry-options scaleR-aform-ivl
λ*optns*. *split-aform-largest* (*precision optns*) (*presplit-summary-tolerance optns*)
disjoint-aforms
inter-aform-plane

interpretation *aform-approximate-ivp0 x y* **for** *x y* **.**

definition *print-rectangle*
where
print-rectangle m i t0 X =
(*let*
- = *print* (*int-to-string i*);
- = *print* (*STR* '': '');
- = *print-real t0*
in
if i mod m = *0 then*

```
    let
      R = Radius X;
      - = print (STR '': '');
      - = print-eucl (fst X − R);
      - = print (STR '' − '');
      - = print-eucl (fst X + R);
      - = print (STR ''; devs: '');
      - = print (int-to-string (length (list-of-pdevs (snd X))));
      - = print (STR ''; width: '');
      - = print-real (infnorm R);
      - = print (STR ''; tdev: '');
      - = print-eucl R;
      - = print (STR ''; maxdev: '');
      - = print-eucl (snd (max-pdev (snd X)));
      - = println (STR '''')
    in ()
  else println (STR ''''))
```

definition *print-aform*::'a::executable-euclidean-space aform ⇒ unit
where

```
  print-aform X =
    (let
      - = print (STR ''aform('');
      - = print (int-to-string (length (Basis-list::'a list)));
      - = print (STR ''): '');
      - = print-eucl (fst X);
      - = print (STR '' −− '');
      - = map (λ(i, x). print-eucl x) (list-of-pdevs (snd X));
      - = println (STR '''')
    in ())
```

definition *ivls-of-aforms p ress = map (λ(t0, CX, t1, X).*
(t0, eucl-truncate-down p (Inf-aform CX), eucl-truncate-up p (Sup-aform CX), t1,
eucl-truncate-down p (Inf-aform X), eucl-truncate-up p (Sup-aform X))) ress

primrec *summarize-ivls* **where**

```
  summarize-ivls [] = None
| summarize-ivls (x#xs) = (case summarize-ivls xs of
      None ⇒ Some x
    | Some (t0', cl', cu', t1', xl', xu') ⇒
      case x of (t0, cl, cu, t1, xl, xu) ⇒
      if t0 = t1' then
        Some (min t0 t0', inf cl cl', sup cu cu', max t1 t1',
          if t1 ≤ t1' then xl' else xl, if t1 ≤ t1' then xu' else xu)
      else None)
```

fun *set-res-of-ivl-res*
where *set-res-of-ivl-res (t0, CXl, CXu, t1 ,Xl, Xu) = (t0, {CXl .. CXu}, t1, {Xl*

.. *Xu*})

```
fun parts::nat⇒'a list⇒'a list list
where
  parts n [] = []
| parts 0 xs = [xs]
| parts n xs = take n xs # parts n (drop n xs)

definition summarize-enclosure
where summarize-enclosure p m xs =
  map the (filter (−Option.is-none) (map summarize-ivls (parts m (ivls-of-aforms
p xs))))

definition ivls-result p m = (apsnd (summarize-enclosure p m o snd)) o snd

definition default-optns =
    (|
    precision = 53,
    tolerance = FloatR 1 (− 8),
    stepsize  = FloatR 1 (− 8),
    min-stepsize = FloatR 1 (− 8),
    iterations = 40,
    halve-stepsizes = 10,
    widening-mod = 40,
    max-tdev-thres = FloatR 1 100,
    presplit-summary-tolerance = FloatR 1 0,
    collect-mod = 0,
    collect-granularity = FloatR 1 100,
    override-section = (λ- - - -. (0, 0)),
    global-section = (λ-. None),
    stop-iteration = (λ-. False),
    printing-fun = (λ- -. print-aform),
    result-fun = ivls-result 23 1
  |)

end
```

13 Euler method on Affine Forms

```
theory Euler-Affine
imports
  ~~/src/HOL/Decision-Procs/Dense-Linear-Order
  ../IVP/Picard-Lindeloef-Qualitative
  ../Library/Linear-ODE
  Euler-Affine-Code
begin

lemma inf-le-sup-same1: inf a (b::'a::ordered-euclidean-space) ≤ sup a d
  by (metis inf.coboundedI1 sup.cobounded1)
```

lemma fixes $a::'a$ *option*
shows *split-option-bind*: P $(a \gg= f) \longleftrightarrow ((a = None \longrightarrow P\ None) \wedge (\forall x.\ a = Some\ x \longrightarrow P\ (f\ x)))$
and *split-option-bind-asm*: P $(a \gg= f) \longleftrightarrow (\neg\ (a = None \wedge \neg\ P\ None \vee (\exists x.\ a = Some\ x \wedge \neg\ P\ (f\ x))))$
unfolding *atomize-conj*
by (*cases a*) (*auto split*: *option.split*)

lemma *msum-subsetI*:
assumes $X \subseteq X'$ $Y \subseteq Y'$
shows $\{(x::'a::group\text{-}add) + y\ |x\ y.\ x \in X \wedge y \in Y\} \subseteq \{x + y\ |x\ y.\ x \in X' \wedge y \in Y'\}$
proof *safe*
fix x y
assume *xy*: $x \in X$ $y \in Y$
show $\exists x'\ y'.\ x + y = x' + y' \wedge x' \in X' \wedge y' \in Y'$
apply (*rule exI*[**where** $x=x$])
apply (*rule exI*[**where** $x=y$])
using *xy assms* **by** *auto*
qed

13.1 operations on intervals

include separate type of intervals in *approximate-sets0*

type-synonym $'a$ *ivl* $= 'a*'a$

definition *set-of-ivl*::$'a$ *ivl* $\Rightarrow$ $'a$::*executable-euclidean-space set*
where *set-of-ivl* $x = \{fst\ x\ ..\ snd\ x\}$

definition *split-ivl*::$'a$ *ivl* $\Rightarrow$ *real* $\Rightarrow$ $'a \Rightarrow$ $'a$ *ivl* $*$ $'a$::*executable-euclidean-space ivl*
where *split-ivl* $x\ s\ i = ((fst\ x,\ snd\ x + (s - snd\ x \cdot i) *_R i),\ (fst\ x + (s - fst\ x \cdot i) *_R i,\ snd\ x))$

lemma *split-ivl*:
assumes $i \in Basis$
assumes $s \in \{fst\ X \cdot i\ ..\ snd\ X \cdot i\}$
shows $x \in$ *set-of-ivl* $X \longleftrightarrow x \in$ (*set-of-ivl* (*fst* (*split-ivl* $X\ s\ i$)) $\cup$ *set-of-ivl* (*snd* (*split-ivl* $X\ s\ i$)))
using *assms*
by (*auto simp*: *set-of-ivl-def split-ivl-def eucl-le*[**where** $'a='a$] *not-le algebra-simps inner-Basis*)

fun *Pair-of-list*::$'a$ *list* $\Rightarrow$ $'a*'a$ **where**
Pair-of-list $[a,\ b] = (a,\ b)$

locale *approximate-sets* = *approximate-sets0* +
assumes *msum-appr-eq*: *set-of-appr* (*msum-appr* X Y) $= \{x + y\ |x\ y.\ x \in$

set-of-appr X ∧ *y* ∈ *set-of-appr Y*}
assumes *inf-of-appr-msum-appr*: *inf-of-appr* (*msum-appr X Y*) = *inf-of-appr X* + *inf-of-appr Y*
assumes *sup-of-appr-msum-appr*: *sup-of-appr* (*msum-appr X Y*) = *sup-of-appr X* + *sup-of-appr Y*
assumes *inf-of-appr-Inf*: *inf-of-appr X* ≤ *Inf* (*set-of-appr X*)
assumes *sup-of-appr-Sup*: *sup-of-appr X* ≥ *Sup* (*set-of-appr X*)
assumes *sup-of-appr-of-ivl*: *l* ≤ *u* ⟹ *sup-of-appr* (*appr-of-ivl l u*) = *u*
assumes *inf-of-appr-of-ivl*: *l* ≤ *u* ⟹ *inf-of-appr* (*appr-of-ivl l u*) = *l*
assumes *set-of-appr-of-ivl*: *l* ≤ *u* ⟹ *set-of-appr* (*appr-of-ivl l u*) = {*l* .. *u*}
assumes *set-of-appr-nonempty*: *set-of-appr X* ≠ {}
assumes *set-of-appr-compact*: *compact* (*set-of-appr X*)
assumes *set-of-appr-convex*: *convex* (*set-of-appr X*)
assumes *set-of-apprs-set-of-appr*: [*x*] ∈ *set-of-apprs* [*X*] ⟷ *x* ∈ *set-of-appr X*
assumes *set-of-apprs-switch*: *x*#*y*#*xs* ∈ *set-of-apprs* (*X*#*Y*#*XS*) ⟹ *y*#*x*#*xs* ∈ *set-of-apprs* (*Y*#*X*#*XS*)
assumes *set-of-apprs-rotate*: *x*#*y*#*xs* ∈ *set-of-apprs* (*X*#*Y*#*XS*) ⟹ *y*#*xs*@[*x*] ∈ *set-of-apprs* (*Y*#*XS*@[*X*])
assumes *set-of-apprs-Nil*: *xs* ∈ *set-of-apprs* [] ⟹ *xs* = []
assumes *length-set-of-apprs*: *xs* ∈ *set-of-apprs XS* ⟹ *length xs* = *length XS*
assumes *set-of-apprs-Cons-ex*: *xs* ∈ *set-of-apprs* (*X*#*XS*) ⟹ (∃ *y ys*. *xs* = *y*#*ys* ∧ *y* ∈ *set-of-appr X* ∧ *ys* ∈ *set-of-apprs XS*)
assumes *in-image-Pair-of-listI*[*simp*, *intro*]:
[*x*, *y*] ∈ *set-of-apprs* [*X*, *Y*] ⟹ (*x*, *y*) ∈ *Pair-of-list* ‘ *set-of-apprs* [*X*, *Y*]
assumes *add-appr*: (*x* # *y* # *ys*) ∈ *set-of-apprs* (*X* # *Y* # *YS*) ⟹ (*add-appr optns X Y YS*) = *Some S* ⟹ (*x* + *y*)#*x*#*y*#*ys* ∈ *set-of-apprs* (*S*#*X*#*Y*#*YS*)
assumes *scale-appr*: (*x*#*xs*) ∈ *set-of-apprs* (*X*#*XS*) ⟹ (*scale-appr optns r s X XS*) = *Some S* ⟹ ((*r*/*s*) $*_R$ *x* # *x* # *xs*) ∈ *set-of-apprs* (*S*#*X*#*XS*)
assumes *scale-appr-ivl*: *s* ∈ {*r*..*t*} ⟹ (*x*#*xs*) ∈ *set-of-apprs* (*X*#*XS*) ⟹ (*scale-appr-ivl optns r t X XS*) = *Some S* ⟹ (*s* $*_R$ *x* # *x* # *xs*) ∈ *set-of-apprs* (*S*#*X*#*XS*)
assumes *split-appr*: *x* ∈ *set-of-appr X* ⟹ *list-ex* (λ*X*. *x* ∈ *set-of-appr X*) (*split-appr optns X*)
assumes *disjoint-apprs*: *disjoint-apprs X Y* ⟹ *set-of-appr X* ∩ *set-of-appr Y* = {}
begin

lemma *set-of-appr-bounded*[*intro*]: *bounded* (*set-of-appr X*)
by (*rule compact-imp-bounded*) (*rule set-of-appr-compact*)

lemma *inf-of-appr*[*simp*]: *x* ∈ *set-of-appr X* ⟹ *inf-of-appr X* ≤ *x*
by (*auto intro*!: *order-trans*[*OF inf-of-appr-Inf*] *cInf-lower bounded-imp-bdd-below*)

lemma *sup-of-appr*[*simp*]: *x* ∈ *set-of-appr X* ⟹ *x* ≤ *sup-of-appr X*
by (*auto intro*!: *order-trans*[*OF - sup-of-appr-Sup*] *cSup-upper bounded-imp-bdd-above*)

lemma *inf-of-appr-le-sup-of-appr*[*simp*]:
inf-of-appr a ≤ *sup-of-appr a*
using *set-of-appr-nonempty*[*of a*] *order-trans*[*OF inf-of-appr sup-of-appr*]

by *auto*

lemma *set-of-apprs-Cons*: $x\#xs \in$ *set-of-apprs* $(X\#XS) \Longrightarrow xs \in$ *set-of-apprs XS*
by (*auto dest*: *set-of-apprs-Cons-ex*)

lemma *set-of-apprsE*:
assumes $xs \in$ *set-of-apprs* $(X\#XS)$
obtains *y ys* **where** $xs = y \# ys$ $y \in$ *set-of-appr X* $ys \in$ *set-of-apprs XS*
using *set-of-apprs-Cons-ex assms* **by** *blast*

lemma *set-of-apprs-rotate3*:
$[x,y, z] \in$ *set-of-apprs* $[X, Y, Z] \Longrightarrow [y, z, x] \in$ *set-of-apprs* $[Y,Z,X]$
by (*metis Cons-eq-appendI eq-Nil-appendI set-of-apprs-rotate*)

end

lemma *tendsto-singleton*[*tendsto-intros*]: $(f \longrightarrow f\ x)$ (*at x within* $\{x\}$)
by (*auto simp*: *tendsto-def eventually-at-filter*)

lemma *continuous-on-singleton*[*continuous-intros*]: *continuous-on* $\{x\}$ *f*
unfolding *continuous-on-def*
by (*auto intro*!: *tendsto-singleton*)

locale *approximate-ivp* = *approximate-ivp0* + *approximate-sets* +
fixes *ode*::$'a \Rightarrow 'a$
fixes *ode-d*::$'a \Rightarrow 'a \Rightarrow 'a$
assumes *ode-approx*:
$x\#xs \in$ *set-of-apprs* $(X'\#XS) \Longrightarrow$
ode-approx optns $(X'\#XS) =$ *Some* $A \Longrightarrow$
(*ode* $x \# x \# xs) \in$ *set-of-apprs* $(A \# X' \# XS)$
assumes *fderiv*[*derivative-intros*]: $x \in X \Longrightarrow$ (*ode has-derivative ode-d x*) (*at x within X*)
assumes *ode-d-approx*:
$x\#dx\#xs \in$ *set-of-apprs* $(X'\#DX'\#XS) \Longrightarrow$
ode-d-approx optns $(X'\#DX'\#XS) =$ (*Some* D') $\Longrightarrow$
(*ode-d x dx* $\# x \# dx \# xs) \in$ *set-of-apprs* $(D' \# X' \# DX' \# XS)$
assumes *cont-fderiv*: *continuous-on UNIV* (λ((*t*::*real*, *x*), (*dt*::*real*, *y*)). *ode-d x y*)
— TODO: get rid of the reals
begin

lemma *fderiv'*[*derivative-intros*]: ((λ(*t*, *y*). *ode y*) *has-derivative* (λ(*t*, *x*) (*dt*, *dx*). *ode-d x dx*) (*t*, *x*)) (*at* (*t*, *x*) *within X*)
by (*auto intro*!: *derivative-eq-intros has-derivative-compose*[*of snd*])

lemma *picard-approx*:
assumes *appr*: *ode-approx optns* $[X] =$ *Some Y*
assumes *bb*: *inf-of-appr* $Y = l$ *sup-of-appr* $Y = u$
assumes *x-in*: $(\bigwedge t.\ t \in \{t0\ ..\ t1\} \Longrightarrow x\ t \in$ *set-of-appr X*)

assumes *cont*: *continuous-on* {*t0* .. *t1*} *x*
assumes *ivl*: *t0* ≤ *t1*
shows *x0* + *integral* {*t0*..*t1*} (λ*t*. *ode* (*x t*)) ∈ {*x0* + (*t1* − *t0*) $*_R$ *l* .. *x0* + (*t1* − *t0*) $*_R$ *u*}
proof −
{
fix *t*::*real*
assume *0* ≤ *t t* ≤ *1*
hence *t* ∗ (*t1* − *t0*) ≤ *t1* − *t0* **using** *ivl*
by (*auto intro*!: *mult-left-le-one-le*)
hence *t0* + *t* ∗ (*t1* − *t0*) ≤ *t1*
by (*simp add*: *algebra-simps*)
} **note** *segment*[*simp*] = *this*
{
fix *t*::*real*
assume *t*: *t* ∈ {*0* .. *1*}
have *ode* (*x* (*t0* + *t* ∗ (*t1* − *t0*))) ∈ *set-of-appr Y*
unfolding *set-of-apprs-set-of-appr*[*symmetric*]
apply (*rule set-of-apprs-Cons*)
apply (*rule set-of-apprs-switch*)
apply (*rule ode-approx*[*OF* - *appr*])
using *t ivl*
by (*auto intro*!: *x-in ode-approx simp*: *set-of-apprs-set-of-appr*)
also from *bb inf-of-appr sup-of-appr* **have** *set-of-appr Y* ⊆ {*l*..*u*} **by** *auto*
finally have *ode* (*x* (*t0* + *t* ∗ (*t1* − *t0*))) ∈ {*l*..*u*} **.**
} **note** *ode-lu* = *this*
have *cont-ode-x*: *continuous-on* {*t0*..*t1*} (λ*xa*. *ode* (*x xa*))
using *ivl*
by (*auto intro*!: *has-derivative-continuous-on*[*OF fderiv*] *continuous-on-compose2*[*of* - *ode* - *x*] *cont*)
have *cmp*: (λ*t*. *ode* (*x* (*t0* + *t* ∗ (*t1* − *t0*)))) = (λ*t*. *ode* (*x t*)) *o* (λ*t*. (*t0* + *t* ∗ (*t1* − *t0*)))
by *auto*
have *cnt*: *continuous-on* {*0* .. *1*}(λ*t*. *ode* (*x* (*t0* + *t* ∗ (*t1* − *t0*))))
unfolding *cmp* **using** *ivl*
by (*intro continuous-on-compose*)
(*auto intro*!: *continuous-intros simp*: *image-linear-atLeastAtMost cont-ode-x not-less*)
have *integral* {*t0*..*t1*} (λ*t*. *ode* (*x t*)) =
(*t1* − *t0*) $*_R$ *integral* {*0*..*1*} (λ*t*. *ode* (*x* (*t0* + *t* ∗ (*t1* − *t0*))))
using *ivl*
by (*intro mvt-integral*[*of* - λ*t1*. *integral* {*t0*..*t1*} (λ*t*. *ode* (*x t*)) λ*t u*. *u* $*_R$ *ode* (*x t*)
t0 t1 − *t0*, *simplified*])
(*auto intro*!: *integral-has-vector-derivative*[*OF cont-ode-x*]
simp: *has-vector-derivative-def* [*symmetric*])
also
{
have *integral* {*0*..*1*} (λ*t*. *ode* (*x* (*t0* + *t* ∗ (*t1* − *t0*)))) ≤ *integral* {*0*..*1*}

(λt::*real* . *u*)
using *ode-lu*
by (*auto simp*: *eucl-le*[**where** $'a='a$] *intro*!: *order-trans*[*OF integral-component-ubound-real*] *cnt*)
moreover have *integral* {*0..1*} (λt::*real* . *l*) $\leq$ *integral* {*0..1*} (λt. *ode* (*x* (*t0* + *t* $*$ (*t1* $-$ *t0*))))
using *ode-lu*
by (*auto simp*: *eucl-le*[**where** $'a='a$] *intro*!: *order-trans*[*OF* - *integral-component-lbound-real*] *cnt*)
ultimately have *integral* {*0..1*} (λt. *ode* (*x* (*t0* + *t* $*$ (*t1* $-$ *t0*)))) $\in$ {*l* .. *u*}
by *simp*
hence (*t1* $-$ *t0*) $*_R$ *integral* {*0..1*} (λt. *ode* (*x* (*t0* + *t* $*$ (*t1* $-$ *t0*)))) $\in$ {(*t1* $-$ *t0*) $*_R$ *l* .. (*t1* $-$ *t0*) $*_R$ *u*}
using *ivl*
by (*auto intro*!: *scaleR-left-mono*)
}
finally show *?thesis* **by** *auto*
qed

lemma *picard-approx-ivl*:
assumes *appr*: *ode-approx optns* [*X*] = *Some Y*
assumes *bb*: *inf-of-appr Y* = *l sup-of-appr Y* = *u*
assumes *x-in*: ($\bigwedge t$. *t* $\in$ {*t0* .. *t1*} $\Longrightarrow$ *x t* $\in$ *set-of-appr X*)
assumes *cont*: *continuous-on* {*t0* .. *t1*} *x*
assumes *ivl*: *t0* $\leq$ *t t* $\leq$ *t1*
shows *x0* + *integral* {*t0..t*} (λt. *ode* (*x t*)) $\in$ {*x0* + *inf 0* ((*t1* $-$ *t0*) $*_R$ *l*) .. *x0* + *sup 0* ((*t1* $-$ *t0*) $*_R$ *u*)}
using *ivl inf-of-appr-le-sup-of-appr*[*of Y*]
by (*intro set-rev-mp*[*OF picard-approx*[*OF appr bb x-in continuous-on-subset*[*OF cont*]]])
(*auto simp*: *eucl-le*[**where** $'a='a$] *inner-Basis-inf-left inner-Basis-sup-left inf-real-def sup-real-def min-def max-def zero-le-mult-iff not-le inner-add-left not-less bb intro*: *mult-right-mono mult-nonneg-nonpos mult-right-mono-neg*)

automatic Picard operator

lemma *P-appr-Some-ode-approxE*:
assumes *P-appr optns X0 h X* = *Some R*
obtains *Y* **where** *ode-approx optns* [*X*] = *Some Y R* = *extend-appr X0* (*inf 0* (*h* $*_R$ *inf-of-appr Y*)) (*sup 0* (*h* $*_R$ *sup-of-appr Y*))
using *assms*
unfolding *P-appr-def*
using *assms* **by** (*auto simp*: *P-appr-def*)

lemma *P-appr*:
assumes *x0*: *x0* $\in$ *set-of-appr X0*
assumes *x*: $\bigwedge t$. *t* $\in$ {*t0..t1*} $\Longrightarrow$ *x t* $\in$ *set-of-appr X*
assumes *cont*: *continuous-on* {*t0..t1*} *x*
assumes *h'*: *0* $\leq$ *t1* $-$ *t0 t1* $-$ *t0* $\leq$ *h*
assumes *P-res*: *P-appr optns X0 h X* = *Some R*

shows *x0* + *integral* {*t0*..*t1*} (λ*t*. *ode* (*x t*)) ∈ *set-of-appr R*
proof −
from *P-res* **obtain** *Y* **where** *Y*: *ode-approx optns* [*X*] = *Some Y*
R = *extend-appr X0* (*inf 0* (*h* $*_R$ *inf-of-appr Y*)) (*sup 0* (*h* $*_R$ *sup-of-appr Y*))
by (*rule P-appr-Some-ode-approxE*)
have *x0* + *integral* {*t0* .. *t1*} (λ*t*. *ode* (*x t*)) ∈
{*x0* + *inf 0* ((*t1* − *t0*) $*_R$ *inf-of-appr Y*) .. *x0* + *sup 0* ((*t1* − *t0*) $*_R$ *sup-of-appr Y*)}
using *assms*
by (*intro picard-approx-ivl*[*OF Y*(*1*) *refl refl x cont*]) *auto*
also have ... ⊆ {*x* + *y* |*x y*. *x* ∈ *set-of-appr X0* ∧
y ∈ {*inf 0* ((*t1* − *t0*) $*_R$ *inf-of-appr Y*) .. *sup 0* ((*t1* − *t0*) $*_R$ *sup-of-appr Y*)}}
apply *safe*
subgoal for *x*
apply (*rule exI*[**where** *x*=*x0*])
apply (*rule exI*[**where** *x*=*x* − *x0*])
using *assms*
apply (*simp add*: *algebra-simps*)
done
done
also have ... ⊆ {*x* + *y* |*x y*. *x* ∈ *set-of-appr X0* ∧ *y* ∈ {*inf 0* (*h* $*_R$ *inf-of-appr Y*) .. *sup 0* (*h* $*_R$ *sup-of-appr Y*)}}
using *assms*
by (*intro msum-subsetI*) (*auto simp*: *eucl-le*[**where** ′*a*=′*a*] *inner-Basis-inf-left inf-real-def*
inner-Basis-sup-left sup-real-def not-le not-less min-zero-mult-nonneg-le max-zero-mult-nonneg-le)
also have ... = *set-of-appr R*
using *assms*
by (*simp add*: *inf-le-sup-same1 scaleR-left-mono set-of-appr-of-ivl Y msum-appr-eq*)
finally show *?thesis* .
qed

lemma *P-iterE*:
assumes *P-iter optns X0 h i X* = *Some X*′
obtains
X′′ **where** *P-appr optns X0 h X*′ = *Some X*′′
{*inf-of-appr X*′′ .. *sup-of-appr X*′′} ⊆ {*inf-of-appr X*′ .. *sup-of-appr X*′}
using *assms*
proof (*induct i arbitrary*: *X*)
case (*Suc i*) **thus** *?case*
by (*cases P-appr optns X0 h X*) (*auto simp*: *split*: *if-split-asm*)
qed *simp*

lemma *extend-appr-ivl*:
assumes *set-of-appr X* = {*inf-of-appr X* .. *sup-of-appr X*}
assumes *le2*: *a* ≤ *0 0* ≤ *b*
assumes *set-of-apprI*: ⋀*x*. *inf-of-appr X* ≤ *x* ⟹ *x* ≤ *sup-of-appr X* ⟹ *x* ∈ *set-of-appr X*

shows *set-of-appr* (*extend-appr X a b*) = {*inf-of-appr X* + *a* .. *sup-of-appr X* + *b*}
proof −
have {*inf-of-appr X* + *a*..*sup-of-appr X* + *b*} = {*x* + *y*|*x y*. *x* ∈ {*inf-of-appr X* ..*sup-of-appr X*} ∧ *y* ∈ {*a* .. *b*}}
proof *safe*
fix *x* **assume** *x*: *x* ∈ {*inf-of-appr X* + *a*..*sup-of-appr X* + *b*}
let *?x′* = ∑ *i*∈*Basis*. (*if* (*x* · *i*) ≤ *inf-of-appr X* · *i then inf-of-appr X* · *i else if* (*x* · *i*) ≤ *sup-of-appr X* · *i then x* · *i else sup-of-appr X* · *i*)$*_R$ *i*
show ∃ *x′ y*. *x* = *x′* + *y* ∧ *x′* ∈ {*inf-of-appr X*..*sup-of-appr X*} ∧ *y* ∈ {*a*..*b*}
apply (*rule exI*[**where** *x* = *?x′*])
apply (*rule exI*[**where** *x* = *x* − *?x′*])
unfolding *assms*
using *le2 x inf-of-appr-le-sup-of-appr*
by (*auto simp*: *eucl-le*[**where** *′a*=*′a*] *algebra-simps intro*!: *set-of-apprI*)
qed (*auto intro*!: *add-mono*)
also have . . . = *set-of-appr* (*extend-appr X a b*)
unfolding *msum-appr-eq* **using** *le2*
by (*intro antisym msum-subsetI*) (*auto simp*: *set-of-appr-of-ivl assms*(*1*))
finally show *?thesis* **by** *simp*
qed

lemma *P-appr-ivl*:
assumes *P-appr optns X0 h X* = *Some X′*
assumes *h* ≥ *0*
assumes *ivl-0*: {*inf-of-appr X0*..*sup-of-appr X0*} = *set-of-appr X0*
shows {*inf-of-appr X′*..*sup-of-appr X′*} = *set-of-appr X′*
proof −
from *assms* **obtain** *z* **where** *z*: *ode-approx optns* [*X*] = *Some z*
and *X′*: *extend-appr X0* (*inf 0* (*h* $*_R$ *inf-of-appr z*)) (*sup 0* (*h* $*_R$ *sup-of-appr z*)) = *X′*
by (*auto simp*: *P-appr-def*)
have [*simp*]: *inf 0* (*h* $*_R$ *inf-of-appr z*) ≤ *sup 0* (*h* $*_R$ *sup-of-appr z*)
by (*metis inf.coboundedI1 sup.coboundedI1*)
show *?thesis*
unfolding *X′*[*symmetric*]
by (*auto simp*: *ivl-0*[*symmetric*] *extend-appr-ivl inf-of-appr-msum-appr sup-of-appr-msum-appr inf-of-appr-of-ivl sup-of-appr-of-ivl*)
qed

lemma *P-iter-ivl*:
assumes *P-iter optns X0 h i X* = *Some X′*
assumes *h* ≥ *0*
assumes {*inf-of-appr X0*..*sup-of-appr X0*} = *set-of-appr X0*
assumes {*inf-of-appr X*..*sup-of-appr X*} = *set-of-appr X*
shows {*inf-of-appr X′*..*sup-of-appr X′*} = *set-of-appr X′*
using *assms*
proof (*induct i arbitrary*: *X X′*)
case (*Suc i*)

thus *?case*
proof (*cases P-appr optns X0 h X*)
fix *a*
assume ∗: *P-appr optns X0 h X = Some a*
show *?thesis*
proof (*cases inf-of-appr X ≤ inf-of-appr a ∧ sup-of-appr a ≤ sup-of-appr X*)
case *True*
with ∗ *Suc*(*2*) **have** *X′ = X* **by** *simp*
with *Suc* **show** *?thesis* **by** *simp*
next
case *False*
with ∗ *Suc*(*2*) **have** *ind-step*: *P-iter optns X0 h i*
(*appr-of-ivl*
(*inf* (*inf-of-appr a*) (*inf-of-appr X*) −
(*if i mod widening-mod optns = 0 then* |*inf-of-appr a − inf-of-appr X*|
else 0))
(*sup* (*sup-of-appr a*) (*sup-of-appr X*) +
(*if i mod widening-mod optns = 0 then* |*sup-of-appr a − sup-of-appr X*|
else 0))) =
Some X′
by (*simp add*: ∗)
have *inf-le-sup*: *inf* (*inf-of-appr a*) (*inf-of-appr X*) ≤ *sup* (*sup-of-appr a*)
(*sup-of-appr X*)
by (*metis inf-of-appr-le-sup-of-appr le-infI2 le-supI2*)
hence *min-le-max*: *inf* (*inf-of-appr a*) (*inf-of-appr X*) − |*inf-of-appr a −
inf-of-appr X*|
≤ *sup* (*sup-of-appr a*) (*sup-of-appr X*) + |*sup-of-appr a − sup-of-appr X*|
unfolding *diff-conv-add-uminus*
by (*rule add-mono*) (*metis abs-ge-zero dual-order.trans neg-le-0-iff-le*)
show {*inf-of-appr X′..sup-of-appr X′*} = *set-of-appr X′*
by (*rule Suc*(*1*)[*OF ind-step*])
(*auto simp add*: *Suc inf-of-appr-of-ivl sup-of-appr-of-ivl min-le-max set-of-appr-of-ivl
inf-le-sup*)
qed
qed *simp*
qed *simp*

lemma *P-iter-mono*:
assumes *P-iter optns X0 h i X = Some X′*
shows *set-of-appr X0* ⊆ {*inf-of-appr X′..sup-of-appr X′*}
proof −
from *P-iterE*[*OF assms*(*1*)] **obtain** *X″* **where** *X″*:
P-appr optns X0 h X′ = Some X″
{*inf-of-appr X″..sup-of-appr X″*} ⊆ {*inf-of-appr X′..sup-of-appr X′*} **.**
from *X″*(*1*) **have** *set-of-appr X0* ⊆ *set-of-appr X″*
by (*force simp*: *P-appr-def msum-appr-eq set-of-appr-of-ivl inf-le-sup-same1*)
also have . . . ⊆ {*inf-of-appr X″..sup-of-appr X″*}
by *auto*
also note *X″*(*2*)

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

lemma *P-iter-eq*:
assumes *P-iter optns X0 h i X = Some X'*
assumes $h \geq 0$
assumes {*inf-of-appr X0..sup-of-appr X0*} = *set-of-appr X0*
assumes {*inf-of-appr X..sup-of-appr X*} = *set-of-appr X*
shows *set-of-appr X'* = {*inf-of-appr X'..sup-of-appr X'*}
using *assms*
by (*simp add*: *P-iter-ivl*[*OF assms*])

lemma *P-iter-cert-stepsize*:
assumes *cert-stepsize optns X0 h n i = Some* (*h'*, *X'*)
shows *P-iter optns* (*ivl-appr-of-appr X0*) *h' n* (*ivl-appr-of-appr X0*) = *Some X'*
using *assms*
by (*induct i arbitrary*: *h*) (*auto split*: *option.split-asm*)

definition *step-ivp*::*real* ⇒ *'a* ⇒ *real* ⇒ *'b* ⇒ *'a ivp* **where**
step-ivp t0 x0 t1 CX =
(|*ivp-f* = (λ(*t*, *x*). *ode x*),
ivp-t0 = *t0*, *ivp-x0* = *x0*,
ivp-T = {*t0* .. *t1*},
ivp-X = *set-of-appr CX*|)

lemma *step-ivp-simps*[*simp*]:
ivp-f (*step-ivp t0 x0 t1 CX*) = (λ(*t*, *x*). *ode x*)
ivp-t0 (*step-ivp t0 x0 t1 CX*) = *t0*
ivp-x0 (*step-ivp t0 x0 t1 CX*) = *x0*
ivp-T (*step-ivp t0 x0 t1 CX*) = {*t0* .. *t1*}
ivp-X (*step-ivp t0 x0 t1 CX*) = *set-of-appr CX*
by (*simp-all add*: *step-ivp-def*)

definition *euler-ivp*::*real* ⇒ *'a* ⇒ *real* ⇒ *'a ivp*
where
euler-ivp t0 x0 t1 =
(|*ivp-f* = (λ(*t*, *x*). *ode x*),
ivp-t0 = *t0*, *ivp-x0* = *x0*,
ivp-T = {*t0* .. *t1*},
ivp-X = *UNIV*|)

lemma *euler-ivp-simps*[*simp*]:
ivp-f (*euler-ivp t0 x0 t1*) = (λ(*t*, *x*). *ode x*)
ivp-t0 (*euler-ivp t0 x0 t1*) = *t0*
ivp-x0 (*euler-ivp t0 x0 t1*) = *x0*
ivp-T (*euler-ivp t0 x0 t1*) = {*t0* .. *t1*}
ivp-X (*euler-ivp t0 x0 t1*) = *UNIV*
by (*simp-all add*: *euler-ivp-def*)

definition *global-ivp*::*real* ⇒ *'a* ⇒ *'a ivp* **where**
global-ivp t0 x0 =
⦇*ivp-f* = (λ(*t*, *x*). *ode x*),
ivp-t0 = *t0*, *ivp-x0* = *x0*,
ivp-T = *UNIV*,
ivp-X = *UNIV*⦈

lemma *global-ivp-simps*[*simp*]:
ivp-f (*global-ivp t0 x0*) = (λ(*t*, *x*). *ode x*)
ivp-t0 (*global-ivp t0 x0*) = *t0*
ivp-x0 (*global-ivp t0 x0*) = *x0*
ivp-T (*global-ivp t0 x0*) = *UNIV*
ivp-X (*global-ivp t0 x0*) = *UNIV*
by (*simp-all add*: *global-ivp-def*)

execution of *local.euler-step*

context
fixes *optns x0 X0 h RES-ivl RES*
assumes *x0*: *x0* ∈ *set-of-appr X0*
assumes *pos-prestep*: *0* < *stepsize optns*
assumes *euler-step-returns*: *euler-step optns X0* = *Some* (*h*, *RES-ivl*, *RES*)
begin

intermediate results

context
fixes *n i CX X0' F D ERR S S' X1 CX1 t0 t1*
assumes *pos-step*: *0* < *h*
assumes *step-eq*: *t0* + *h* = *t1*
assumes *certified-stepsize*: *cert-stepsize optns X0* (*stepsize optns*) *n i* = *Some* (*h*, *CX*)
assumes *bounded-ode*: *ode-approx optns* [*X0*] = *Some X0'*
assumes *bounded-total-ode*: *ode-approx optns* [*CX*] = *Some F*
assumes *bounded-ode-d*: *ode-d-approx optns* [*CX*, *F*] = *Some D*
assumes *bounded-err*: *scale-appr optns* (*h*∗*h*) *2* (*ivl-appr-of-appr D*) [] = *Some ERR*
assumes *bounded-scale-euler*: *scale-appr optns h 1 X0'* [*X0*] = *Some S*
assumes *bounded-scale-ivl-euler*: *scale-appr-ivl optns 0 h X0'* [*X0*] = *Some S'*
assumes *bounded-add-euler*: *add-appr optns X0 S* [] = *Some X1*
assumes *bounded-add-euler-ivl*: *add-appr optns X0 S'* [] = *Some CX1*
assumes *RES-ivl*: *RES-ivl* = *msum-appr CX1* (*appr-of-ivl* (*inf 0* (*inf-of-appr ERR*)) (*sup 0* (*sup-of-appr ERR*)))
assumes *RES*: *RES* = *msum-appr X1* (*ivl-appr-of-appr ERR*)
begin

lemma *nonneg-step*: *0* ≤ *h* **using** *pos-step* **by** *auto*
lemma *step-less*: *t0* < *t1* **using** *step-eq pos-step* **by** *auto*

lemma *set-of-appr-eq*: *set-of-appr CX* = {*inf-of-appr CX* .. *sup-of-appr CX*}
by (*subst P-iter-eq*[*OF P-iter-cert-stepsize*[*OF certified-stepsize*]])

(*auto simp*: *ivl-appr-of-appr-def sup-of-appr-of-ivl inf-of-appr-of-ivl set-of-appr-of-ivl nonneg-step*)

lemma *x0-in-CX1*: *x0* ∈ *set-of-appr CX1*
proof −
  from *nonneg-step* **have** *0* ∈ {*0* .. *h*} **by** *auto*
  from *scale-appr-ivl*[*OF this ode-approx*[*OF x0*[*simplified set-of-apprs-set-of-appr*[*symmetric*]] *bounded-ode*] *bounded-scale-ivl-euler*]
  have [*x0*, *0*] ∈ *set-of-apprs* [*X0*, *S'*]
    by (*metis pth-4*(*1*) *set-of-apprs-Cons set-of-apprs-rotate3*)
  from *add-appr*[*OF this bounded-add-euler-ivl*]
  show *x0* ∈ *set-of-appr CX1*
    by (*metis monoid-add-class.add.right-neutral set-of-apprs-Cons set-of-apprs-rotate3 set-of-apprs-set-of-appr*)
qed

interpretation *ivp-on-interval step-ivp t0 x0 t1 CX t1*
  using *nonneg-step step-eq*
proof (*unfold-locales*, *simp-all add*: *step-ivp-def*)
  have *x0* ∈ *set-of-appr* (*ivl-appr-of-appr X0*)
    by (*auto simp*: *ivl-appr-of-appr-def set-of-appr-of-ivl x0*)
  also have ... ⊆ {*inf-of-appr CX*..*sup-of-appr CX*}
    by (*metis P-iter-mono P-iter-cert-stepsize certified-stepsize*)
  also have ... = *set-of-appr CX*
    by (*rule set-of-appr-eq*[*symmetric*])
  finally show *x0* ∈ *set-of-appr CX* **.**
qed

interpretation *continuous-rhs T X f*
  using *iv-defined*
  by *unfold-locales* (*auto simp add*: *step-ivp-def split-beta intro*!: *continuous-on-compose2*[*of - ode - snd*] *has-derivative-continuous-on*[*OF fderiv*] *continuous-intros*)

lemma *Blinfun-ode-d*[*simp*]: *blinfun-apply* (*Blinfun* (λ(*dt*, *y*). *ode-d b y*)) = (λ(*dt*, *y*). *ode-d b y*)
  by (*subst bounded-linear-Blinfun-apply*)
    (*auto intro*!: *has-derivative-bounded-linear fderiv'*[*THEN has-derivative-eq-rhs*])

interpretation *derivative-set-bounded T X f* λ(*t*, *x*) (*dt*, *dx*). *ode-d x dx Pair-of-list* ‘ *set-of-apprs* [*CX*, *F*]
  {*inf-of-appr D* .. *sup-of-appr D*}
proof
  have *Pair-of-list* ‘ *set-of-apprs* [*CX*, *F*] ⊆ *set-of-appr CX* × *set-of-appr F*
    by (*auto elim*!: *set-of-apprsE dest*!: *set-of-apprs-Nil*)
  moreover have *bounded* (...)
    by (*rule set-of-appr-compact compact-imp-bounded bounded-Times*)+
  ultimately show *bounded* (*Pair-of-list* ‘ *set-of-apprs* [*CX*, *F*])
    by (*blast intro*: *bounded-subset*)

show *compact* {*inf-of-appr D* .. *sup-of-appr D*} *convex* {*inf-of-appr D*..*sup-of-appr D*}
by (*simp-all add*: *compact-interval convex-closed-interval*)
fix *t x*
assume *t* ∈ *T x* ∈ *X*
hence *x*: [*x*] ∈ *set-of-apprs* [*CX*] **by** (*auto simp*: *step-ivp-def set-of-apprs-set-of-appr*)
with *ode-approx*
have [*x*, *ode x*] ∈ *set-of-apprs* [*CX*, *F*]
by (*auto intro*!: *ode-approx bounded-total-ode intro*: *set-of-apprs-switch*)
thus (*x*, *ivp-f* (*step-ivp t0 x0 t1 CX*) (*t*, *x*)) ∈ *Pair-of-list* ' *set-of-apprs* [*CX*, *F*]
by (*auto simp*: *step-ivp-def*)
next
fix *t x d*
assume *t* ∈ *T*
assume (*x*, *d*) ∈ *Pair-of-list* ' *set-of-apprs* [*CX*, *F*]
then obtain *xs* **where** *xs*: *Pair-of-list xs* = (*x*, *d*) *xs* ∈ *set-of-apprs* [*CX*, *F*]
by *auto*
hence *xs* = [*x*, *d*]
by (*auto elim*!: *set-of-apprsE dest*!: *set-of-apprs-Nil*)
with *xs* **have** [*x*, *d*] ∈ *set-of-apprs* [*CX*, *F*] **by** *simp*
hence [*x*, *d*, *ode-d x d*] ∈ *set-of-apprs* [*CX*, *F*, *D*]
by (*auto intro*!: *ode-d-approx bounded-ode-d intro*: *set-of-apprs-switch set-of-apprs-rotate3*)
hence *ode-d x d* ∈ *set-of-appr D*
unfolding *set-of-apprs-set-of-appr*[*symmetric*]
by (*blast intro*: *set-of-apprs-Cons*)
thus (*case* (*t*, *x*) *of* (*t*, *x*) ⇒ λ(*dt*, *dx*). *ode-d x dx*) (*1*, *d*) ∈ {*inf-of-appr D* .. *sup-of-appr D*}
by *auto*
next
show *T* ≠ {} *X* ≠ {} **using** *iv-defined* **by** *auto*
show (*f has-derivative* (*case tx of* (*t*, *x*) ⇒ λ(*dt*, *dx*). *ode-d x dx*)) (*at tx within* *T* × *X*)
if *tx* ∈ *T* × *X* **for** *tx*
using *that*
by (*auto intro*!: *derivative-eq-intros simp*: *split-beta*)
qed

lemma *t0'*: *ivp-t0* (*step-ivp t0 x0 t1 CX*) = *t0*
by (*simp add*: *step-ivp-def*)

lemma *interval'*: *T* = {*t0*..*t1*}
by (*auto simp*: *step-ivp-def*)

lemma *blinfun-of-matrix-works'*:
fixes *f*::*'d*::*euclidean-space* ⇒ *'e*::*euclidean-space*
assumes *bounded-linear f*
shows *blinfun-of-matrix* (λ*i j*. (*f j*) · *i*) *x* = *f x*
using *blinfun-of-matrix-works*[*of Blinfun f*]
by (*auto simp*: *bounded-linear-Blinfun-apply assms*)

lemma *bounded-linear-ode-d*: *bounded-linear* (*ode-d x*)
by (*auto intro*!: *has-derivative-bounded-linear derivative-eq-intros*)

lemma *continuous-on-ode-d*[*continuous-intros*]:
assumes *continuous-on s f1*
assumes *continuous-on s f2*
shows *continuous-on s* (λ*x*. *ode-d* (*f1 x*) (*f2 x*))
by (*rule continuous-on-compose2*[*OF cont-fderiv*, **where** *f*=λ*x*. ((*0*, *f1 x*), (*0*, *f2 x*)),
simplified split-beta′ fst-conv snd-conv])
(*auto intro*!: *continuous-intros assms*)

lemma *local-lipschitz-ode*: *local-lipschitz UNIV UNIV* (λ*t*::*real*. *ode*)
apply (*rule c1-implies-local-lipschitz*[**where** *f′*=λ(*t*, *x*). *blinfun-of-matrix* (λ*i j*. *ode-d x j* · *i*)])
subgoal
by (*auto intro*!: *derivative-eq-intros ext simp*: *blinfun-of-matrix-works′ bounded-linear-ode-d*)
subgoal
by (*force simp*: *split-beta′ blinfun-of-matrix-apply*
intro: *has-derivative-bounded-linear fderiv continuous-on-blinfun-componentwise*
continuous-intros)
subgoal by *simp*
subgoal by *simp*
done

definition *L-CX* = (*SOME L*. ∀ *t*. *lipschitz X* (λ*x*. *f* (*t*, *x*)) *L*)

lemma *L-CX*: *lipschitz X* (λ*x*. *f* (*t*, *x*)) *L-CX*
proof −
from *local-lipschitz-ode* **have** *local-lipschitz* {*t0*} (*set-of-appr CX*) (λ*t*::*real*. *ode*)
by (*rule local-lipschitz-on-subset*) *auto*
from *local-lipschitz-on-compact-implies-lipschitz*[*OF this*]
obtain *L* **where** ∀ *t*. *lipschitz X* (λ*x*. *f* (*t*, *x*)) *L*
by (*force simp*: *set-of-appr-compact*)
then have ∀ *t*. *lipschitz X* (λ*x*. *f* (*t*, *x*)) *L-CX*
unfolding *L-CX-def*
by (*rule someI*)
then show *?thesis* **..**
qed

interpretation *unique-on-closed step-ivp t0 x0 t1 CX t1 L-CX*
proof *unfold-locales*
let *?step* = *step-ivp t0 x0 t1 CX*
fix *t x*
assume *xt0*: *x* (*ivp-t0 ?step*) = *ivp-x0 ?step*
from *this* **have** *x t0* ∈ *set-of-appr* (*ivl-appr-of-appr X0*) **using** *x0*
by (*auto simp*: *ivl-appr-of-appr-def set-of-appr-of-ivl*)
moreover

assume $x \in \{ivp\text{-}t0\ ?step..t\} \to ivp\text{-}X\ ?step$
from *this* have $\bigwedge ta.\ ta \in \{t0..t0 + (t - t0)\} \Longrightarrow x\ ta \in set\text{-}of\text{-}appr\ CX$ by *auto*
moreover
assume *continuous-on* $\{ivp\text{-}t0\ ?step..t\}\ x$
from *this* have *continuous-on* $\{t0..t0 + (t - t0)\}\ x$ by *simp*
moreover
assume $t \in ivp\text{-}T\ ?step$
from *this step-eq* have $0 \le t0 + (t - t0) - t0\ t0 + (t - t0) - t0 \le h$
by *simp-all*
moreover
from *P-iter-cert-stepsize*[*OF certified-stepsize*, *THEN P-iterE*]
obtain X'' where *P-appr optns* (*ivl-appr-of-appr X0*) *h CX* = *Some* X''
and *subset*: $\{inf\text{-}of\text{-}appr\ X''..sup\text{-}of\text{-}appr\ X''\} \subseteq \{inf\text{-}of\text{-}appr\ CX..sup\text{-}of\text{-}appr\ CX\}$.
note *this*(*1*)
ultimately have $x\ t0 + integral\ \{t0..t0 + (t - t0)\}\ (\lambda t.\ ode\ (x\ t)) \in set\text{-}of\text{-}appr\ X''$
by (*rule P-appr*)
also have $\ldots \subseteq \{inf\text{-}of\text{-}appr\ X''..sup\text{-}of\text{-}appr\ X''\}$ by *auto*
also note *subset*
also note *set-of-appr-eq*[*symmetric*]
finally show $ivp\text{-}x0\ ?step + integral\ \{ivp\text{-}t0\ ?step..t\}\ (\lambda t.\ ivp\text{-}f\ ?step\ (t, x\ t)) \in ivp\text{-}X\ ?step$
using *xt0* by *simp*
next
show *closed X*
using *compact-eq-bounded-closed set-of-appr-compact*
by *auto*
next
show *lipschitz X* $(\lambda x.\ f\ (t, x))$ *L-CX* for *t*
by (*rule L-CX*)
qed

lemma *solution-t0'*: *solution t0* = *x0*
using *solution-t0* by (*simp add*: *step-ivp-def*)

lemma *euler-consistent-solution'*:
assumes $t1' \in \{t0\ ..\ t1\}$
shows $solution\ (t0 + (t1' - t0)) - discrete\text{-}evolution\ (euler\text{-}increment\ f)\ (t0 + (t1' - t0))\ t0\ (solution\ t0) \in$
$op *_R ((t1' - t0)^2 / 2)$ ‘ $\{inf\text{-}of\text{-}appr\ D..sup\text{-}of\text{-}appr\ D\}$
using *pos-step step-less assms solution-in-D solution-has-deriv*
by (*intro euler-consistent-traj-set*[where *u*=*t1*]) (*auto intro*!: *solution-has-deriv simp*:)

lemma *euler-consistent-solution*:
assumes $t1' \in \{t0\ ..\ t1\}$
shows $solution\ (t0 + (t1' - t0)) - discrete\text{-}evolution\ (euler\text{-}increment\ f)\ (t0 +$

(*t1′* − *t0*)) *t0 x0* ∈
op $*_R$ ((*t1′* − *t0*)^2 / *2*) ‘ {*inf-of-appr D*..*sup-of-appr D*}
using *euler-consistent-solution′*[*simplified solution-t0′*, *OF assms*] **.**

lemma *error-overapproximation*:
shows *solution* (*t0* + *h*) ∈ *set-of-appr RES*
proof −
def *euler-res* ≡ *discrete-evolution* (*euler-increment f*) (*t0* + *h*) *t0 x0*
have *step-ok*: *t0* + *h* ∈ {*t0* .. *t1*} **using** *step-eq pos-step* **by** *auto*
from *this* **have** *solution* (*t0* + *h*) ∈ {*euler-res* + (*h*ˆ*2* / *2*) $*_R$ *inf-of-appr D* .. *euler-res* + (*h*ˆ*2* / *2*) $*_R$ *sup-of-appr D*}
using *euler-consistent-solution*[*OF step-ok*] *step-eq*
by (*auto simp*: *euler-res-def algebra-simps intro*!: *scaleR-left-mono*)
also have ... = {*x* + *y* |*x y*. *x* ∈ {*euler-res*} ∧ *y* ∈ {(*h* ∗ *h* / *2*) $*_R$ *inf-of-appr D* .. (*h* ∗ *h* / *2*) $*_R$ *sup-of-appr D*}}
by (*auto intro*!: *exI*[**where** *x*=*x* − *euler-res* **for** *x*] *simp*: *algebra-simps power2-eq-square*)
also have ... ⊆ *set-of-appr* (*msum-appr X1* (*ivl-appr-of-appr ERR*))
unfolding *msum-appr-eq*
proof (*rule msum-subsetI*)
have *ode-x0*: [*ode x0*, *x0*] ∈ *set-of-apprs* [*X0′*, *X0*]
by (*metis bounded-ode ode-approx x0 set-of-apprs-set-of-appr*)
note *scale-appr*[**where** *r*=*h* **and** *s* = *1* **and** *X* = *X0′* **and** *XS* = [*X0*] **and** *x* = *ode x0*
and *xs* = [*x0*] **and** *optns* = *optns*,
THEN set-of-apprs-rotate, *simplified append-Cons append-Nil*,
THEN set-of-apprs-Cons]
from *add-appr*[*OF this* , *of - optns* , *THEN set-of-apprs-switch*, *THEN set-of-apprs-Cons*,
THEN set-of-apprs-switch, *THEN set-of-apprs-Cons*,
simplified set-of-apprs-set-of-appr, *OF ode-x0 bounded-scale-euler bounded-add-euler*]
show {*euler-res*} ⊆ *set-of-appr X1*
using *x0*
unfolding *euler-res-def discrete-evolution-def euler-increment*
by (*simp add*: *step-ivp-def*)
next
from
scale-appr[**where** *r*=*h* ∗ *h* **and** *s* = *2* **and** *X* = *ivl-appr-of-appr D* **and** *XS*=[] **and** *xs*=[]
and *x*=*inf-of-appr D* **and** *optns*=*optns*,
THEN set-of-apprs-switch, *THEN set-of-apprs-Cons*, *OF - bounded-err*]
scale-appr[**where** *r*=*h* ∗ *h* **and** *s* = *2* **and** *X* = *ivl-appr-of-appr D* **and** *XS*=[] **and** *xs*=[]
and *x*=*sup-of-appr D* **and** *optns*=*optns*,
THEN set-of-apprs-switch, *THEN set-of-apprs-Cons*, *OF - bounded-err*]
show {(*h* ∗ *h* / *2*) $*_R$ *inf-of-appr D*..(*h* ∗ *h* / *2*) $*_R$ *sup-of-appr D*} ⊆ *set-of-appr* (*ivl-appr-of-appr ERR*)
by (*simp-all add*: *set-of-apprs-set-of-appr ivl-appr-of-appr-def set-of-appr-of-ivl*)
qed
finally show *?thesis* **unfolding** *RES* **.**
qed

lemma *unique-solution-step-ivp*: *unique-solution* (*step-ivp t0 x0 t1 CX*) **..**

lemma *error-overapproximation-ivl*:
  assumes *h'*: *h'* ∈ {*0..h*}
  shows *solution* (*t0* + *h'*) ∈ *set-of-appr RES-ivl*
proof −
  def *euler-res* ≡ *discrete-evolution* (*euler-increment f*) (*t0* + *h'*) *t0 x0*
  have *step-ok*: *t0* + *h'* ∈ {*t0* .. *t1*} **using** *step-eq pos-step assms* **by** *auto*

  have *solution* (*t0* + *h'*) ∈ {*euler-res* + (*h'*ˆ*2* / *2*) $*_R$ *inf-of-appr D* .. *euler-res* + (*h'*ˆ*2* / *2*) $*_R$ *sup-of-appr D*}
    using *euler-consistent-solution*[*OF step-ok*] *step-eq*
    by (*auto simp*: *euler-res-def algebra-simps intro*!: *scaleR-left-mono*)
  also have ... = {*x* + *y* |*x y*. *x* ∈ {*euler-res*} ∧ *y* ∈ {(*h'* ∗ *h'* / *2*) $*_R$ *inf-of-appr D* .. (*h'* ∗ *h'* / *2*) $*_R$ *sup-of-appr D*}}
    by (*auto intro*!: *exI*[**where** *x*=*x* − *euler-res* **for** *x*] *simp*: *algebra-simps power2-eq-square*)
  also have ... ⊆ *set-of-appr* (*msum-appr CX1* (*appr-of-ivl* (*inf 0* (*inf-of-appr ERR*)) (*sup 0* (*sup-of-appr ERR*))))
    unfolding *msum-appr-eq*
  proof (*rule msum-subsetI*)
    have *ode-x0*: [*ode x0*, *x0*] ∈ *set-of-apprs* [*X0'*, *X0*]
      by (*metis bounded-ode ode-approx x0 set-of-apprs-set-of-appr*)
    note *scale-appr*[**where** *r*=*h* **and** *X* = *X0'* **and** *XS* = [*X0*] **and** *x* = *ode x0* **and** *xs* = [*x0*] **and** *optns* = *optns*,
      THEN set-of-apprs-rotate, *simplified append-Cons append-Nil*,
      THEN set-of-apprs-Cons]
    note *scale-appr-ivl*[*OF h'*, **where** *X* = *X0'* **and** *XS* = [*X0*] **and** *x* = *ode x0* **and** *xs* = [*x0*] **and** *optns* = *optns*,
      THEN set-of-apprs-rotate, *simplified append-Cons append-Nil*,
      THEN set-of-apprs-Cons]
    from *add-appr*[*OF this* , *of - optns* , *THEN set-of-apprs-switch*, *THEN set-of-apprs-Cons*,
      THEN set-of-apprs-switch, *THEN set-of-apprs-Cons*,
      simplified set-of-apprs-set-of-appr, *OF ode-x0 bounded-scale-ivl-euler bounded-add-euler-ivl*]
    show {*euler-res*} ⊆ *set-of-appr CX1*
      using *x0*
      unfolding *euler-res-def discrete-evolution-def euler-increment*
      by (*simp add*: *step-ivp-def*)
  next
    have *infsup*[*simp*]: *inf 0* (*inf-of-appr ERR*) ≤ (*sup 0* (*sup-of-appr ERR*))
      by (*metis inf-sup-ord*(*1*) *le-supI1*)
    have {(*h'* ∗ *h'* / *2*) $*_R$ *inf-of-appr D* .. (*h'* ∗ *h'* / *2*) $*_R$ *sup-of-appr D*} ⊆
        {*inf 0* ((*h* ∗ *h* / *2*) $*_R$ *inf-of-appr D*) .. *sup 0* ((*h* ∗ *h* / *2*) $*_R$ *sup-of-appr D*)}
      unfolding *interval-cbox*
    proof (*rule subset-box-imp*, *safe*)
      fix *i*::*'a* **assume** *i* ∈ *Basis*
      show *inf 0* ((*h* ∗ *h* / *2*) $*_R$ *inf-of-appr D*) · *i* ≤ (*h'* ∗ *h'* / *2*) $*_R$ *inf-of-appr D* · *i*

using *assms*
unfolding *inner-Basis-inf-left*[*OF* ‹*i* ∈ *Basis*›] *inner-zero-left inf-real-def inner-scaleR-left*
by (*intro min-zero-mult-nonneg-le*) (*auto intro*!: *mult-mono*)
show $(h' * h' / 2) *_R$ *sup-of-appr D* $\cdot$ *i* ≤ *sup 0* $((h * h / 2) *_R$ *sup-of-appr D*) $\cdot$ *i*
using *assms*
unfolding *inner-Basis-sup-left*[*OF* ‹*i* ∈ *Basis*›] *inner-zero-left sup-real-def inner-scaleR-left*
by (*intro max-zero-mult-nonneg-le*) (*auto intro*!: *mult-mono*)
qed
also
from
scale-appr[**where** *r*=*h* ∗ *h* **and** *s* = *2* **and** *X* = *ivl-appr-of-appr D* **and** *XS*=[] **and** *xs*=[]
and *x*=*inf-of-appr D* **and** *optns*=*optns*,
THEN set-of-apprs-switch, *THEN set-of-apprs-Cons*, *OF - bounded-err*]
scale-appr[**where** *r*=*h* ∗ *h* **and** *s* = *2* **and** *X* = *ivl-appr-of-appr D* **and** *XS*=[] **and** *xs*=[]
and *x*=*sup-of-appr D* **and** *optns*=*optns*,
THEN set-of-apprs-switch, *THEN set-of-apprs-Cons*, *OF - bounded-err*]
have . . . ⊆ *set-of-appr* (*appr-of-ivl* (*inf 0* (*inf-of-appr ERR*)) (*sup 0* (*sup-of-appr ERR*)))
by (*auto simp add*: *set-of-apprs-set-of-appr ivl-appr-of-appr-def set-of-appr-of-ivl*
intro!: *le-infI2 le-supI2*)
finally show $\{(h' * h' / 2) *_R$ *inf-of-appr D*..$(h' * h' / 2) *_R$ *sup-of-appr D*} ⊆
set-of-appr ((*appr-of-ivl* (*inf 0* (*inf-of-appr ERR*)) (*sup 0* (*sup-of-appr ERR*))))
by (*auto simp add*: *ivl-appr-of-appr-def set-of-appr-of-ivl*)
qed
finally show *?thesis* **unfolding** *RES-ivl* **.**
qed

lemma *unique-on-open-global*: *unique-on-open* (*global-ivp t0 x0*)
proof (*unfold-locales*)
let *?ivp* = (*global-ivp t0 x0*)
show *ivp-t0 ?ivp* ∈ *ivp-T ?ivp ivp-x0 ?ivp* ∈ *ivp-X ?ivp*
by (*simp-all add*: *global-ivp-def*)
show *open* (*ivp-T ?ivp*) *open* (*ivp-X ?ivp*)
by (*auto simp*: *global-ivp-def*)
show *continuous-on* (*ivp-T ?ivp* × *ivp-X ?ivp*) (*ivp-f ?ivp*)
by (*auto simp*: *global-ivp-def intro*!: *continuous-intros fderiv′ has-derivative-continuous-on*)
fix *I t x*
assume *t* ∈ (*ivp-T ?ivp*) *x* ∈ (*ivp-X ?ivp*)
— TODO: make local lipschitz based on open sets
with *open-contains-cball*[*of* (*ivp-T ?ivp*)] ‹*open* (*ivp-T ?ivp*)›
open-contains-cball[*of* (*ivp-X ?ivp*)] ‹*open* (*ivp-X ?ivp*)›
obtain *u v* **where** *uv*: *cball t u* ⊆ (*ivp-T ?ivp*) *cball x v* ⊆ (*ivp-X ?ivp*) *u* > *0 v*

> 0
  by *blast*
 def $w \equiv$ *min u v*
 have *cball t w* $\subseteq$ (*ivp-T ?ivp*) *cball x w* $\subseteq$ (*ivp-X ?ivp*) $w > 0$ **using** *uv* **by** (*auto simp*: *w-def*)
 have *cball t w* = $\{t - w .. t + w\}$ **by** (*auto simp*: *dist-real-def*)
 from *cbox-in-cball'*[*OF* ‹$w > 0$›] **obtain** w' **where** w':
  $w'>0$ $w' \leq w$ $\bigwedge y.\ y \in \{x - setsum\ (op *_R\ w')\ Basis..x + setsum\ (op *_R\ w')\ Basis\} \Longrightarrow y \in cball\ x\ w$
  by (*metis cbox-interval*)
next
 show *local-lipschitz* (*ivp-T* (*global-ivp t0 x0*)) (*ivp-X* (*global-ivp t0 x0*)) (λt *x*. *ivp-f* (*global-ivp t0 x0*) (*t*, *x*))
  using *local-lipschitz-ode* **by** *simp*
qed

lemma *unique-on-intermediate-euler-step*:
 shows
  unique-solution (*euler-ivp t0 x0 t1*) **and**
  $\bigwedge t.\ t \in \{t0 .. t1\} \Longrightarrow$ *ivp.solution* (*euler-ivp t0 x0 t1*) $t \in$ *set-of-appr RES-ivl* **and**
  ivp.solution (*euler-ivp t0 x0 t1*) *t1* $\in$ *set-of-appr RES*
proof −
 from *unique-solution-step-ivp*
 interpret *step*: *unique-solution* (*step-ivp t0 x0 t1 CX*) **.**
 from *iv-defined* **have** $t0 \leq t1$ **by** (*auto simp*: *step-ivp-def*)
 interpret *euler*: *ivp* (*euler-ivp t0 x0 t1*)
  using ‹$t0 \leq t1$›
  by *unfold-locales auto*
 have *euler-ivp-step-ivp*: *euler-ivp t0 x0 t1* = *step-ivp t0 x0 t1 CX*(|*ivp-X* := *UNIV*|)
  by (*simp add*: *step-ivp-def*)
 have *step-solves-euler*: *euler.is-solution solution*
  unfolding *euler-ivp-step-ivp*
  by (*auto intro*!: *is-solution-on-superset-domain*)
 interpret *euler*: *has-solution* (*euler-ivp t0 x0 t1*)
  by *unfold-locales* (*rule exI step-solves-euler*)+
 from *unique-on-open-global*
 interpret *uo*: *unique-on-open global-ivp t0 x0* **.**
 from *uo.global-solution* **guess** *J* **. note** *J*=*this*
 def *max-ivp* $\equiv$
  (|*ivp-f* = (λ(*t*, *x*). *ode x*),
  ivp-t0 = *t0*, *ivp-x0* = *x0*,
  ivp-T = *J*,
  ivp-X = *UNIV*|)
 from *J*(*6*) **interpret** *max-ivp*: *unique-solution max-ivp*
  by (*auto simp*: *global-ivp-def max-ivp-def*)
 {
  fix *t1 x*

assume *ivp.is-solution* (*euler-ivp t0 x0 t1*) *x*
hence $\bigwedge t.\ t \in \{t0\ ..\ t1\} \Longrightarrow x\ t = ivp.solution\ max\text{-}ivp\ t$
using *J*(*7*)[**where** *K2*={*t0* .. *t1*}]
by (*auto simp*: *euler-ivp-def global-ivp-def max-ivp-def is-interval-closed-interval*)
} **note** *solution-eqI* = *this*
interpret *euler*: *unique-solution* (*euler-ivp t0 x0 t1*)
proof
fix *y t*
assume *y*: *euler.is-solution y* **and** *t* ∈ *euler.T*
hence *t* ∈ {*t0* .. *t1*} **by** (*simp add*: *euler-ivp-def*)
thus *y t* = *ivp.solution* (*euler-ivp t0 x0 t1*) *t*
by (*simp add*: *solution-eqI*[*OF y*] *solution-eqI*[*OF euler.is-solution-solution*])
qed
show *unique-solution* (*euler-ivp t0 x0 t1*) **proof qed**
have *step-eq-euler*: $\bigwedge t.\ t \in \{t0\ ..\ t1\} \Longrightarrow solution\ t = euler.solution\ t$
by (*auto intro*!: *euler.unique-solution step-solves-euler*)
{
fix *t* **assume** *t* ∈ {*t0* .. *t1*}
thus *euler.solution t* ∈ *set-of-appr RES-ivl*
using *error-overapproximation-ivl*[*of t* − *t0*] ‹*t0* ≤ *t1*› *step-eq step-eq-euler*
by *auto*
}
show *euler.solution t1* ∈ *set-of-appr RES*
using *error-overapproximation* ‹*t0* ≤ *t1*› *step-eq step-eq-euler*
by (*auto simp add*: *step-ivp-def*)
qed

end

lemma *unique-on-euler-step*:
assumes *t0* + *h* = *t1*
shows
unique-solution (*euler-ivp t0 x0 t1*) (**is** *?th1*) **and**
$\bigwedge t.\ t \in \{t0\ ..\ t1\} \Longrightarrow ivp.solution\ (euler\text{-}ivp\ t0\ x0\ t1)\ t \in set\text{-}of\text{-}appr\ RES\text{-}ivl$
(**is** $\bigwedge t.\ ?ass2\ t \Longrightarrow ?th2\ t$) **and**
ivp.solution (*euler-ivp t0 x0 t1*) *t1* ∈ *set-of-appr RES* (**is** *?th3*)
proof −
from *euler-step-returns*
obtain *X0′ CX F D ERR S S′ X1′ X1″* **where** *intermediate-results*:
cert-stepsize optns X0 (*stepsize optns*) (*iterations optns*) (*halve-stepsizes optns*)
= *Some* (*h*, *CX*)
ode-approx optns [*X0*] = *Some X0′*
ode-approx optns [*CX*] = *Some F*
ode-d-approx optns [*CX*, *F*] = *Some D*
scale-appr optns (*h* ∗ *h*) *2* (*ivl-appr-of-appr D*) [] = *Some ERR*
scale-appr optns h 1 X0′ [*X0*] = *Some S*
scale-appr-ivl optns 0 h X0′ [*X0*] = *Some S′*
add-appr optns X0 S [] = *Some X1′*
add-appr optns X0 S′ [] = *Some X1″*

RES-ivl = *extend-appr X1''* (*inf 0* (*inf-of-appr ERR*)) (*sup 0* (*sup-of-appr ERR*))
RES = *msum-appr X1'* (*ivl-appr-of-appr ERR*)
using *pos-prestep euler-step-returns*
by (*auto simp*: *euler-step-def split*: *split-option-bind-asm*)
from *cert-stepsize-pos*[*OF intermediate-results*(*1*) *pos-prestep*] **have** *0* < *h* **.**
from *unique-on-intermediate-euler-step*[*OF* ‹*0* < *h*› *assms intermediate-results*(*1*−*11*)]
show *?th1* ⋀*t*. *?ass2 t* ⟹ *?th2 t ?th3* **by** −
qed

end

fun *set-res-of-appr-res*
where *set-res-of-appr-res* (*t0'*, *CX*, *t1'*, *X*) = (*t0'*, *set-of-appr CX*, *t1'*, *set-of-appr X*)

definition
enclosure f t0 t1 xs = *list-all* (λ(*t0'*, *CX*, *t1'*, *X*).
f t1' ∈ *X* ∧ (∀ *t* ∈ {*t0'*::*real* .. *t1'*}. *f t* ∈ *CX*) ∧
t0 ≤ *t0'* ∧ *t0'* ≤ *t1'* ∧ *t1'* ≤ *t1*) *xs*

lemma *enclosure-ConsI*:
assumes *enclosure f t0 t1 ress0*
assumes *f* (*fst* (*snd* (*snd r*))) ∈ *snd* (*snd* (*snd r*))
assumes ⋀*t*. *t* ∈ {*fst r* .. *fst* (*snd* (*snd r*))} ⟹ *f t* ∈ *fst* (*snd r*)
assumes *t0* ≤ *fst r fst r* ≤ *fst* (*snd* (*snd r*)) *fst* (*snd* (*snd r*)) ≤ *t1*
shows *enclosure f t0 t1* (*r* # *ress0*)
using *assms* **by** (*auto simp*: *enclosure-def*)

lemma *enclosure-Nil-iff*[*simp*]: *enclosure f t0 t1* [] ⟷ *True* **by** (*auto simp*: *enclosure-def*)

lemma *enclosure-Cons-iff*:
shows *enclosure f t0 t1* ((*t0'*, *CX*, *t1'*, *X1*) # *ress0*) ⟷
(*f t1'* ∈ *X1* ∧ (∀ *t*∈{*t0'* .. *t1'*}. *f t* ∈ *CX*) ∧
t0 ≤ *t0'* ∧ *t0'* ≤ *t1'* ∧ *t1'* ≤ *t1* ∧ *enclosure f t0 t1 ress0*)
using *assms* **by** (*auto simp*: *enclosure-def*)

lemma *enclosure-subst*:
assumes *enclosure f t0 t1 ress*
assumes ⋀*t*. *t* ∈ {*t0* .. *t1*} ⟹ *f t* = *g t*
shows *enclosure g t0 t1 ress*
using *assms*
by (*induct ress*) (*auto simp*: *enclosure-Cons-iff*)

lemma *enclosure-mono*:
assumes *t1* ≤ *t2*
assumes *enclosure f t0 t1 ress*
shows *enclosure f t0 t2 ress*

using *assms*
by (*induct ress*) (*auto simp*: *enclosure-Cons-iff*)

execution of *local.advance-euler*

lemma *advance-euler-enclosure*:
assumes *pos-prestep*: *0 < stepsize optns*
assumes *encl*: *enclosure* (*ivp.solution* (*euler-ivp t0 x0 t1*)) *t0 t1* (*map set-res-of-appr-res xs*)
assumes *u1*: *unique-solution* (*euler-ivp t0 x0 t1*)
assumes *sol*: *ivp.solution* (*euler-ivp t0 x0 t1*) *t1* ∈ *set-of-appr X1*
assumes *adv*: *advance-euler optns* (*Some* (*i*, *t1*, *X1*, *xs*)) = *Some* (*j*, *t2*, *X2*, *ys*)
shows *enclosure* (*ivp.solution* (*euler-ivp t0 x0 t2*)) *t0 t2* (*map set-res-of-appr-res ys*) (**is** *?encl*)
and *unique-solution* (*euler-ivp t0 x0 t2*) (**is** *?unique*)
and *ivp.solution* (*euler-ivp t0 x0 t2*) *t2* ∈ *set-of-appr X2* (**is** *?sol*)
proof −
from *adv* **obtain** *CX* **where** *step*: *euler-step optns X1* = *Some* (*t2* − *t1*, *CX*, *X2*)
and *ys*: *ys* = (*t1*, *CX*, *t2*, *X2*)#*xs*
by (*auto simp*: *split*: *option.split-asm*)
from *u1* **interpret** *u1*: *unique-solution euler-ivp t0 x0 t1*
by *simp*
have *t0* ≤ *t1* **using** *u1.iv-defined* **by** *simp*
have *t1* + (*t2* − *t1*) = *t2* **by** *simp*
note *sol-step* = *unique-on-euler-step*[*OF sol pos-prestep step this*]
from *sol-step*(*1*)
interpret *u2*: *unique-solution euler-ivp t1* (*ivp.solution* (*euler-ivp t0 x0 t1*) *t1*) *t2* **by** *simp*
have *t1* ≤ *t2* **using** *u2.iv-defined* **by** *simp*
from ‹*t0* ≤ *t1*› ‹*t1* ≤ *t2*›
interpret *connected-unique-solutions*
euler-ivp t0 x0 t2
euler-ivp t0 x0 t1
euler-ivp t1 (*ivp.solution* (*euler-ivp t0 x0 t1*) *t1*) *t2*
t1
using *u1.solution-t0 u2.solution-t0*
by *unfold-locales auto*
have *enclosure* (*ivp.solution* (*euler-ivp t0 x0 t2*)) *t0 t2* (*map set-res-of-appr-res xs*)
by (*auto intro*!: *enclosure-mono*[*OF* ‹*t1* ≤ *t2*›] *enclosure-subst*[*OF encl*]
simp: *solution1-eq-solution*)
thus *?encl ?sol*
using *sol-step* ‹*t0* ≤ *t1*› ‹*t1* ≤ *t2*› *encl*
by (*auto simp*: *ys enclosure-Cons-iff solution2-eq-solution*)
show *?unique* **by** *unfold-locales*
qed

lemma *euler-series-enclosure*:

assumes *pos-prestep*: *0* < *stepsize optns*
assumes *x0*: *x0* ∈ *set-of-appr X0*
assumes *euler-series-returns*: *euler-series optns t0 X0 i* = *Some* (*j*, *t2*, *X2*, *ress*)
shows
unique-solution (*euler-ivp t0 x0 t2*)
enclosure (*ivp.solution* (*euler-ivp t0 x0 t2*)) *t0 t2* (*map set-res-of-appr-res ress*)
ivp.solution (*euler-ivp t0 x0 t2*) *t2* ∈ *set-of-appr X2*
unfolding *atomize-conj*
using *x0 euler-series-returns*
proof (*induct i arbitrary*: *t0 ress t2 X2 j*)
case *0*
let *?triv* = *euler-ivp t2 x0 t2*
interpret *triv*: *ivp ?triv*
by *standard auto*
have *triv*: *unique-solution ?triv*
by (*rule triv.singleton-unique-solutionI*) *auto*
then interpret *triv*: *unique-solution ?triv* **.**
have *triv.solution t2* = *x0*
using *triv.solution-t0* **by** *auto*
with *0* **show** *?case*
by (*auto intro*!: *triv enclosure-ConsI*)
next
case (*Suc i*)
then obtain *t1 X1 r1 j′* **where** *ser*: *euler-series optns t0 X0 i* = *Some* (*j′*, *t1*, *X1*, *r1*)
by (*cases euler-series optns t0 X0 i*) *auto*
with *Suc* **have** *adv*: *advance-euler optns* (*Some* (*j′*, *t1*, *X1*, *r1*)) = *Some* (*j*, *t2*, *X2*, *ress*)
by *simp*
from *Suc.hyps*[*OF Suc.prems*(*1*) *ser*]
have *IH*: *enclosure* (*ivp.solution* (*euler-ivp t0 x0 t1*)) *t0 t1* (*map set-res-of-appr-res r1*)
unique-solution (*euler-ivp t0 x0 t1*)
ivp.solution (*euler-ivp t0 x0 t1*) *t1* ∈ *set-of-appr X1*
by *simp-all*
from *advance-euler-enclosure*[*OF pos-prestep IH adv*]
show *?case* **by** *auto*
qed

end

sublocale *aform-approximate-ivp0* ⊆
approximate-sets
aform-of-ivl msum-aform′ Affine Joints
Inf-aform Sup-aform
uncurry-options add-aform-componentwise::(*′a*, *′a*::*executable-euclidean-space aform*, (*real* × ((*real* × *′a* × *′a* × *real* × *′a* × *′a*) *list*))) *options* ⇒ -
uncurry-options scaleQ-aform-componentwise
uncurry-options scaleR-aform-ivl

λ*optns*. *split-aform-largest* (*precision optns*) (*presplit-summary-tolerance optns*)
disjoint-aforms
inter-aform-plane
proof
fix *x y*::*'a* **and** *X Y* **and** *xs ys*::*'a list* **and** *XS YS* **and** *r s S*
and *optns*::(*'a*, *'a aform*, (*real* × ((*real* × *'a* × *'a* × *real* × *'a* × *'a*) *list*))) *options*
show ([*x*] ∈ *Joints* [*X*]) = (*x* ∈ *Affine X*)
by (*auto simp*: *Affine-def valuate-def Joints-def*)
show *Affine X* ≠ {} **by** (*rule Affine-notempty*)
show *compact* (*Affine X*) **by** (*rule compact-Affine*)
{
assume *x* # *y* # *xs* ∈ *Joints* (*X* # *Y* # *XS*)
thus *y* # *x* # *xs* ∈ *Joints* (*Y* # *X* # *XS*) *y* # *xs* @ [*x*] ∈ *Joints* (*Y* # *XS* @ [*X*])
by (*auto simp*: *Joints-def valuate-def*)
}
{
assume *xs* ∈ *Joints* []
thus *xs* = [] **by** (*auto simp*: *Joints-def valuate-def*)
}
{
assume *xs* ∈ *Joints* (*X* # *XS*)
thus ∃ *y ys*. *xs* = *y* # *ys* ∧ *y* ∈ *Affine X* ∧ *ys* ∈ *Joints XS*
by (*auto simp*: *Joints-def Affine-def valuate-def*)
}
{
assume [*x*, *y*] ∈ *Joints* [*X*, *Y*]
thus (*x*, *y*) ∈ *Pair-of-list* ‘ *Joints* [*X*, *Y*]
by (*auto simp*: *Joints-def valuate-def intro*!: *image-eqI*[**where** *x*=[*aform-val e X*, *aform-val e Y*] **for** *e*])
}
{
assume *uncurry-options add-aform-componentwise optns X Y YS* = *Some S x* # *y* # *ys* ∈ *Joints* (*X* # *Y* # *YS*)
from *add-aform-componentwise*[*OF this*]
show (*x* + *y*) # *x* # *y* # *ys* ∈ *Joints* (*S* # *X* # *Y* # *YS*) **.**
}
{
assume *uncurry-options scaleQ-aform-componentwise optns r s X XS* = *Some S x* # *xs* ∈ *Joints* (*X* # *XS*)
from *scaleQ-aform-componentwise*[*OF this*]
show (*r*/*s*) $*_R$ *x* # *x* # *xs* ∈ *Joints* (*S* # *X* # *XS*) **by** *simp*
}
{
fix *s t*::*real*
assume *uncurry-options scaleR-aform-ivl optns r t X XS* = *Some S x* # *xs* ∈ *Joints* (*X* # *XS*)
s ∈ {*r* .. *t*}

```
    from scaleR-aform-ivl[OF this]
    show s *R x # x # xs ∈ Joints (S # X # XS) .
  }
  {
    assume x ∈ Affine X
    then obtain e where e: e ∈ UNIV → {−1 .. 1} x = aform-val e X
      by (auto simp: Affine-def valuate-def)
    let ?sum = summarize-threshold (precision optns) (presplit-summary-tolerance
optns) (degree-aform X) (snd X)
    obtain e' where e': e' ∈ funcset UNIV {−1 .. 1}
      aform-val e' (fst X, ?sum) = aform-val e X
      by (rule summarize-pdevsE[OF e(1) order-refl, of snd X precision optns
            (λi y. presplit-summary-tolerance optns ∗ infnorm (eucl-truncate-up
(precision optns) (Radius' (precision optns) X)) ≤ infnorm y)])
        (auto simp: summarize-threshold-def aform-val-def)
    from e e' have x: x = aform-val e' (fst X, ?sum)
      by simp
   show list-ex (λX. x ∈ Affine X) (split-aform-largest (precision optns) (presplit-summary-tolerance
optns) X)
    proof (rule split-aformE[OF e'(1) x, where i=fst (max-pdev ?sum)])
      fix err::real
      assume err ∈ {−1 .. 1} x = aform-val (e'(fst (max-pdev ?sum) := err))
          (fst (split-aform (fst X, ?sum) (fst (max-pdev ?sum))))
     thus list-ex (λX. x ∈ Affine X) (split-aform-largest (precision optns) (presplit-summary-tolerance
optns) X)
        using e'(1)
      by (force simp: split-aform-largest-def split-aform-largest-uncond-def Affine-def
valuate-def
          intro!: image-eqI[where x=e' (a := err) for a] split: prod.split)
    next
      fix err::real
      assume err ∈ {−1 .. 1} x = aform-val (e'(fst (max-pdev ?sum) := err))
          (snd (split-aform (fst X, ?sum) (fst (max-pdev ?sum))))
     thus list-ex (λX. x ∈ Affine X) (split-aform-largest (precision optns) (presplit-summary-tolerance
optns) X)
        using e'(1)
      by (force simp: split-aform-largest-def split-aform-largest-uncond-def Affine-def
valuate-def
          intro: image-eqI[where x=e' (a := err) for a err]
          split: prod.split)
    qed
  }
  show disjoint-aforms X Y =⇒ Affine X ∩ Affine Y = {}
    by (rule disjoint-aforms)
  show Affine (msum-aform' X Y) = {x + y |x y. x ∈ Affine X ∧ y ∈ Affine Y}
    by (rule Affine-msum-aform) simp
  show Inf-aform (msum-aform' X Y) = Inf-aform X + Inf-aform Y
    Sup-aform (msum-aform' X Y) = Sup-aform X + Sup-aform Y
    by (auto simp: Inf-aform-msum-aform Sup-aform-msum-aform)
```

show *Inf-aform X* ≤ *Inf* (*Affine X*) *Sup* (*Affine X*) ≤ *Sup-aform X*
by (*auto simp*: *Affine-def valuate-def Inf-aform Sup-aform intro*!: *cINF-greatest cSUP-least*)
{
fix *l u*::*'a* **assume** *le*: *l* ≤ *u*
show *Sup-aform* (*aform-of-ivl l u*) = *u*
Inf-aform (*aform-of-ivl l u*) = *l*
Affine (*aform-of-ivl l u*) = {*l*..*u*}
using *Inf-aform-aform-of-ivl*[*OF le*] *Sup-aform-aform-of-ivl*[*OF le*]
Affine-aform-of-ivl[*OF le*]
by *auto*
}
show *convex* (*Affine X*)
by (*rule convex-Affine*)
show *xs* ∈ *Joints XS* ⟹ *length xs* = *length XS* **by** (*auto simp*: *Joints-def valuate-def*)
qed

locale *aform-approximate-ivp* = *aform-approximate-ivp0* +
approximate-ivp
aform-of-ivl msum-aform′ Affine Joints
Inf-aform Sup-aform
uncurry-options add-aform-componentwise::(*'a*, *'a*::*executable-euclidean-space aform*, (*real* × ((*real* × *'a* × *'a* × *real* × *'a* × *'a*) *list*))) *options* ⇒ -
uncurry-options scaleQ-aform-componentwise
uncurry-options scaleR-aform-ivl
λ*optns*. *split-aform-largest* (*precision optns*) (*presplit-summary-tolerance optns*)
disjoint-aforms
inter-aform-plane
begin

TODO: prove these lemmas generically

lemma *ivls-of-aforms*:
assumes *enclosure f t0 t1* (*map set-res-of-appr-res xs*)
shows *enclosure f t0 t1* (*map set-res-of-ivl-res* (*ivls-of-aforms p xs*))
using *assms*
proof (*induct xs*)
case (*Cons x xs*)
thus *?case*
by (*cases x*) (*auto simp*: *ivls-of-aforms-def o-def enclosure-Cons-iff*
intro: *inf-of-appr eucl-truncate-down-le sup-of-appr eucl-truncate-up-le*)
qed (*simp add*: *ivls-of-aforms-def*)

lemma *summarize-ivls*:
fixes *f*::*real* ⇒ *'a*
assumes *enclosure f t0 t1* (*map set-res-of-ivl-res xs*)
shows *enclosure f t0 t1* (*case summarize-ivls xs of Some x* ⇒ [*set-res-of-ivl-res x*] | *None* ⇒ [])
using *assms*

proof (*induct xs*)
  case *Nil*
  thus *?case* **by** *simp*
next
  case (*Cons x xs*)
  have *inf-cases*: $\bigwedge$*t t0 t1 t2 a b*.
    $\forall$ *t*$\in${*t1*..*t2*}. *a* $\leq$ *f t* $\Longrightarrow$
    $\forall$ *t*$\in${*t0*..*t1*}. *b* $\leq$ *f t* $\Longrightarrow$
    t0 $\leq$ *t* $\Longrightarrow$ *t* $\leq$ *t2* $\Longrightarrow$
    inf a b $\leq$ *f t*
    by (*metis atLeastAtMost-iff le-cases le-infI1 le-infI2*)
  have *sup-cases*: $\bigwedge$*t t0 t1 t2 a b*.
      $\forall$ *t*$\in${*t1*..*t2*}. *f t* $\leq$ *a* $\Longrightarrow$
      $\forall$ *t*$\in${*t0*..*t1*}. *f t* $\leq$ *b* $\Longrightarrow$
      t0 $\leq$ *t* $\Longrightarrow$ *t* $\leq$ *t2* $\Longrightarrow$
      f t $\leq$ *sup a b*
    by (*metis atLeastAtMost-iff le-cases le-supI1 le-supI2*)
  show *?case*
    using *Cons*
    by (*cases x*) (*fastforce simp*: *min-def max-def enclosure-Cons-iff*
      split: *if-split-asm option.split*
       intro: *inf-cases sup-cases inf.coboundedI1 inf.coboundedI2 le-infI2 le-supI1 le-supI2*
        sup.coboundedI1 sup.coboundedI2)
qed

lemma *enclosure-takeD*:
  assumes *enclosure f t0 t1* (*map set-res-of-ivl-res xs*)
  shows *enclosure f t0 t1* (*map set-res-of-ivl-res* (*take m xs*))
using *assms*
proof (*induct xs arbitrary*: *m*)
  case (*Cons x xs*)
  thus *?case*
    by (*cases m*) (*auto simp*: *enclosure-def*)
qed *simp*

lemma *enclosure-dropD*:
  assumes *enclosure f t0 t1* (*map set-res-of-ivl-res xs*)
  shows *enclosure f t0 t1* (*map set-res-of-ivl-res* (*drop m xs*))
using *assms*
proof (*induct xs arbitrary*: *m*)
  case (*Cons x xs*)
  thus *?case*
    by (*cases m*) (*auto simp*: *enclosure-def*)
qed *simp*

lemma *summarize-option-map-filter-aux*: (*case f xs of None* $\Rightarrow$ [] | *Some x* $\Rightarrow$ [*set-res-of-ivl-res x*]) =
    (*map set-res-of-ivl-res* (*map the* (*filter* (− *Option.is-none*) (*map f* [*xs*]))))

by (*auto split*: *option.split simp*: *Option.is-none-def*)

lemma *enclosure-Cons-splitI*:
enclosure f t0 t1 (*map set-res-of-ivl-res* (*map the* (*filter* (− *Option.is-none*) ([*X*])))) ⟹
enclosure f t0 t1 (*map set-res-of-ivl-res* (*map the* (*filter* (− *Option.is-none*) ((*Xs*))))) ⟹
enclosure f t0 t1 (*map set-res-of-ivl-res* (*map the* (*filter* (− *Option.is-none*) ((*X* # *Xs*)))))
by (*case-tac set-res-of-ivl-res* (*the X*)) (*auto simp*: *enclosure-Cons-iff*)

lemma *summarize-enclosure-aux*:
fixes *f*::*real* ⇒ ′*a*
assumes *enclosure f t0 t1* (*map set-res-of-ivl-res xs*)
shows *enclosure f t0 t1* (*map set-res-of-ivl-res* (*map the* (*filter* (−*Option.is-none*) (*map summarize-ivls* (*parts m xs*)))))
using *assms*
proof (*induct m xs rule*: *parts.induct*)
case *1* **thus** *?case* **by** *simp*
next
case (*2 x xs*)
from *summarize-ivls*[*OF 2*]
show *?case* **unfolding** *parts.simps summarize-option-map-filter-aux* **.**
next
case (*3 m x xs*)
have *enclosure f t0 t1* (*map set-res-of-ivl-res* (*map the* (*filter* (− *Option.is-none*) [*summarize-ivls* (*take* (*Suc m*) (*x* # *xs*))])))
using *summarize-ivls summarize-option-map-filter-aux*[*symmetric*, *of summarize-ivls* (*take* (*Suc m*) (*x* # *xs*)), *simplified list.map*]
by (*metis 3.prems enclosure-takeD*)
moreover
from *3* **have** *enclosure f t0 t1*
(*map set-res-of-ivl-res* (*map the* (*filter* (− *Option.is-none*) (*map summarize-ivls* (*parts* (*Suc m*) (*drop* (*Suc m*) (*x* # *xs*)))))))
by (*metis enclosure-dropD*)
ultimately
show *?case*
unfolding *parts.simps list.map*
by (*rule enclosure-Cons-splitI*)
qed

lemma
summarize-enclosure:
enclosure f t0 t1 (*map set-res-of-appr-res xs*) ⟹
enclosure f t0 t1 (*map set-res-of-ivl-res* (*summarize-enclosure p m xs*))
unfolding *summarize-enclosure-def*
by (*intro summarize-enclosure-aux ivls-of-aforms*)

lemma *euler-series-ivls-result*:

assumes *pos-prestep*: *0 < stepsize optns*
assumes *x0*: *x0 ∈ Affine X0*
assumes *ivls-result*: *result-fun optns = ivls-result p m*
assumes *euler-series-returns*: *euler-series-result optns t0 X0 i = Some (t1, xs)*
shows *unique-solution (euler-ivp t0 x0 t1)* (**is** *?th1*)
and *enclosure (ivp.solution (euler-ivp t0 x0 t1)) t0 t1 (map set-res-of-ivl-res xs)*
(**is** *?th2*)
proof −
from *euler-series-returns* **obtain** *j X1 ress*
where *ress*: *euler-series optns t0 X0 i = Some (j, t1, X1, ress)*
and *xs*: *xs = summarize-enclosure p m ress*
by (*auto simp*: *ivls-result ivls-result-def*)
from *euler-series-enclosure*[*OF assms*(*1−2*) *ress*]
show *?th1 ?th2*
by (*auto intro*!: *summarize-enclosure simp*: *xs*)
qed

end

end

14 Optimizations for Code Integer

theory *Optimize-Integer*
imports
Complex-Main
~~/src/HOL/Library/Code-Target-Numeral
begin

TODO: Missing? code post rule?

lemma [*code-post*]: *int-of-integer (− 1) = − 1*
by *simp*

shallowly embed log and power

definition *log2*::*int ⇒ int*
where *log2 a = floor (log 2 (of-int a))*

context includes *integer.lifting* **begin**

lift-definition *log2-integer* :: *integer ⇒ integer*
is *log2* :: *int ⇒ int*
.

end

lemma [*code*]: *log2 (int-of-integer a) = int-of-integer (log2-integer a)*
by (*simp add*: *log2-integer.rep-eq*)

code-printing
  constant *log2-integer* :: *integer* ⇒ - ⇀
    (*SML*) *IntInf.log2*

definition *power-int*::*int* ⇒ *int* ⇒ *int*
  where *power-int a b* = *a* ^ (*nat b*)

context includes *integer.lifting* **begin**

lift-definition *power-integer* :: *integer* ⇒ *integer* ⇒ *integer*
  is *power-int* :: *int* ⇒ *int* ⇒ *int*
  .

end

code-printing
  constant *power-integer* :: *integer* ⇒ - ⇒ - ⇀
    (*SML*) *IntInf.pow* ((-), (-))

lemma [*code*]: *power-int* (*int-of-integer a*) (*int-of-integer b*) = *int-of-integer* (*power-integer a b*)
  by (*simp add*: *power-integer.rep-eq*)

end

15 Optimizations for Code Float

theory *Optimize-Float*
imports
  ../ODE-Auxiliarities
  Optimize-Integer
begin

lemma *compute-bitlen*[*code*]: *bitlen a* = (*if a* > *0 then log2 a* + *1 else 0*)
  by (*simp add*: *bitlen-def log2-def*)

lemma *compute-real-of-float*[*code*]:
  real-of-float (*Float m e*) = (*if e* ≥ *0 then m* ∗ *2* ^ *nat e else m* / *power-int 2* (−*e*))
  unfolding *power-int-def*[*symmetric*, *of 2 e*]
  using *compute-real-of-float power-int-def* **by** *auto*

lemma *compute-float-down*[*code*]:
  float-down p (*Float m e*) =
    (*if p* + *e* < *0 then Float* (*m div power-int 2* (−(*p* + *e*))) (−*p*) *else Float m e*)
  by (*simp add*: *Float.compute-float-down power-int-def*)

lemma *compute-lapprox-posrat*[*code*]:
  fixes *prec*::*nat* **and** *x y*::*nat*

shows *lapprox-posrat prec x y* =
(*let*
l = *rat-precision prec x y*;
d = *if 0* ≤ *l then int x* ∗ *power-int 2 l div y else int x div power-int 2* (− *l*) *div y*
in normfloat (*Float d* (− *l*)))
by (*auto simp add*: *Float.compute-lapprox-posrat power-int-def Let-def zdiv-int of-nat-power of-nat-mult*)

lemma *compute-rapprox-posrat*[*code*]:
fixes *prec x y*
defines *l* ≡ *rat-precision prec x y*
shows *rapprox-posrat prec x y* = (*let*
l = *l* ;
(*r*, *s*) = *if 0* ≤ *l then* (*int x* ∗ *power-int 2 l*, *int y*) *else* (*int x*, *int y* ∗ *power-int 2* (−*l*)) ;
d = *r div s* ;
m = *r mod s*
in normfloat (*Float* (*d* + (*if m* = *0* ∨ *y* = *0 then 0 else 1*)) (− *l*)))
by (*auto simp add*: *l-def Float.compute-rapprox-posrat power-int-def Let-def zdiv-int of-nat-power of-nat-mult*)

lemma *compute-float-truncate-down*[*code*]:
float-round-down prec (*Float m e*) = (*let d* = *bitlen* (*abs m*) − *int prec* − *1 in*
if 0 < *d then let P* = *power-int 2 d* ; *n* = *m div P in Float n* (*e* + *d*)
else Float m e)
by (*simp add*: *Float.compute-float-round-down power-int-def cong*: *if-cong*)

lemma *compute-int-floor-fl*[*code*]:
int-floor-fl (*Float m e*) = (*if 0* ≤ *e then m* ∗ *power-int 2 e else m div* (*power-int 2* (−*e*)))
by (*simp add*: *Float.compute-int-floor-fl power-int-def*)

lemma *compute-floor-fl*[*code*]:
floor-fl (*Float m e*) = (*if 0* ≤ *e then Float m e else Float* (*m div* (*power-int 2* ((−*e*)))) *0*)
by (*simp add*: *Float.compute-floor-fl power-int-def*)

end

16 Examples

theory *Example1*
imports
../Numerics/Euler-Affine
../Numerics/Optimize-Float
begin

16.1 Example 1

approximate-affine *e1* λ(*t*::*real*, *y*::*real*). (*1*::*real*, *y*∗*y* + − *t*)

lemma *e1-fderiv*: ((λ(*t*::*real*, *y*::*real*). (*1*::*real*, *y* ∗ *y* + − *t*)) *has-derivative* (λ(*a*, *b*) (*c*, *d*). (*0*, *2* ∗ (*b* ∗ *d*) + − *c*)) *x*) (*at x within X*)
  by (*auto intro*!: *derivative-eq-intros simp*: *split-beta*)

approximate-affine *e1-d* λ(*a*::*real*, *b*::*real*) (*c*::*real*, *d*::*real*). (*0*::*real*, *2* ∗ (*b* ∗ *d*) + − *c*)

abbreviation *e1-ivp* ≡ λ*optns args*. *uncurry-options e1 optns* (*hd args*) (*tl args*)
abbreviation *e1-d-ivp* ≡ λ*optns args*. *uncurry-options e1-d optns* (*hd args*) (*hd* (*tl args*)) (*tl* (*tl args*))

interpretation *e1*: *aform-approximate-ivp*
  e1-ivp e1-d-ivp
  λ(*t*::*real*, *y*::*real*). (*1*::*real*, *y*∗*y* + − *t*)
  λ(*a*::*real*, *b*::*real*) (*c*::*real*, *d*::*real*). (*0*::*real*, *2* ∗ (*b* ∗ *d*) + − *c*)
  apply *unfold-locales*
  apply (*rule e1*[*THEN Joints2-JointsI*])
  unfolding *list.sel* **apply** *assumption* **apply** *assumption*
  apply (*drule length-set-of-apprs*, *simp*)— TODO: prove in affine-approximation
  apply (*rule e1-fderiv*)
  apply (*rule e1-d*[*THEN Joints2-JointsI*]) **apply** *assumption* **apply** *assumption*
  apply (*drule length-set-of-apprs*, *simp*)— TODO: prove in affine-approximation
  apply (*auto intro*!: *continuous-intros simp*: *split-beta′*)
  done

definition *e1-optns* = *default-optns*
  (| *precision* := *30*,
   tolerance := *FloatR 1* (− *4*),
   stepsize := *FloatR 1* (− *5*),
   result-fun := *ivls-result 23 4*,
   printing-fun := (λ- - -. ()) |)

definition *e1test* = (λ-::*unit*. *euler-series-result e1-ivp e1-d-ivp e1-optns 0* (*aform-of-point* (*real-of-float 0*, *FloatR 23* (− *5*))) (*2* ˆ *7*))

lemma *e1test-result*: *e1test* () =
  Some (*FloatR 128* (− *5*),
    [(*FloatR 124* (− *5*), (*FloatR 248* (− *6*), *FloatR* (− *16128666*) (− *23*)),
(*FloatR 256* (− *6*), *FloatR* (− *15740142*) (− *23*)),
    FloatR 128 (− *5*), (*FloatR 128* (− *5*), *FloatR* (− *16125211*) (− *23*)),
FloatR 128 (− *5*), *FloatR* (− *16091195*) (− *23*)),
    (*FloatR 120* (− *5*), (*FloatR 240* (− *6*), *FloatR* (− *15790851*) (− *23*)),
(*FloatR 248* (− *6*), *FloatR* (− *15351306*) (− *23*)),
    FloatR 124 (− *5*), (*FloatR 124* (− *5*), *FloatR* (− *15785979*) (− *23*)),
FloatR 124 (− *5*), *FloatR* (− *15744397*) (− *23*)),

(*FloatR 116* (− *5*), (*FloatR 232* (− *6*), *FloatR* (− *15418032*) (− *23*)), (*FloatR 240* (− *6*), *FloatR* (− *14902020*) (− *23*)),
FloatR 120 (− *5*), (*FloatR 120* (− *5*), *FloatR* (− *15410566*) (− *23*)), *FloatR 120* (− *5*), *FloatR* (− *15356755*) (− *23*)),
(*FloatR 112* (− *5*), (*FloatR 224* (− *6*), *FloatR* (− *14993301*) (− *23*)), (*FloatR 232* (− *6*), *FloatR* (− *14368805*) (− *23*)),
FloatR 116 (− *5*), (*FloatR 116* (− *5*), *FloatR* (− *14983525*) (− *23*)), *FloatR 116* (− *5*), *FloatR* (− *14909318*) (− *23*)),
(*FloatR 108* (− *5*), (*FloatR 216* (− *6*), *FloatR* (− *14493268*) (− *23*)), (*FloatR 224* (− *6*), *FloatR* (− *13713394*) (− *23*)),
FloatR 112 (− *5*), (*FloatR 112* (− *5*), *FloatR* (− *14479328*) (− *23*)), *FloatR 112* (− *5*), *FloatR* (− *14378493*) (− *23*)),
(*FloatR 104* (− *5*), (*FloatR 208* (− *6*), *FloatR* (− *13887534*) (− *23*)), (*FloatR 216* (− *6*), *FloatR* (− *12895863*) (− *23*)),
FloatR 108 (− *5*), (*FloatR 108* (− *5*), *FloatR* (− *13868008*) (− *23*)), *FloatR 108* (− *5*), *FloatR* (− *13726401*) (− *23*)),
(*FloatR 100* (− *5*), (*FloatR 200* (− *6*), *FloatR* (− *13122628*) (− *23*)), (*FloatR 208* (− *6*), *FloatR* (− *11861626*) (− *23*)),
FloatR 104 (− *5*), (*FloatR 104* (− *5*), *FloatR* (− *13100837*) (− *23*)), *FloatR 104* (− *5*), *FloatR* (− *12912482*) (− *23*)),
(*FloatR 96* (− *5*), (*FloatR 192* (− *6*), *FloatR* (− *12153253*) (− *23*)), (*FloatR 200* (− *6*), *FloatR* (− *10545074*) (− *23*)),
FloatR 100 (− *5*), (*FloatR 100* (− *5*), *FloatR* (− *12123546*) (− *23*)), *FloatR 100* (− *5*), *FloatR* (− *11881819*) (− *23*)),
(*FloatR 92* (− *5*), (*FloatR 184* (− *6*), *FloatR* (− *10917605*) (− *23*)), (*FloatR 192* (− *6*), *FloatR* (− *8897159*) (− *23*)),
FloatR 96 (− *5*), (*FloatR 96* (− *5*), *FloatR* (− *10883344*) (− *23*)), *FloatR 96* (− *5*), *FloatR* (− *10568667*) (− *23*)),
(*FloatR 88* (− *5*), (*FloatR 176* (− *6*), *FloatR* (− *9366037*) (− *23*)), (*FloatR 184* (− *6*), *FloatR* (− *13810922*) (− *24*)),
FloatR 92 (− *5*), (*FloatR 92* (− *5*), *FloatR* (− *9327991*) (− *23*)), *FloatR 92* (− *5*), *FloatR* (− *8922623*) (− *23*)),
(*FloatR 84* (− *5*), (*FloatR 168* (− *6*), *FloatR* (− *14921322*) (− *24*)), (*FloatR 176* (− *6*), *FloatR* (− *9216939*) (− *24*)),
FloatR 88 (− *5*), (*FloatR 88* (− *5*), *FloatR* (− *14853547*) (− *24*)), *FloatR 88* (− *5*), *FloatR* (− *13859369*) (− *24*)),
(*FloatR 80* (− *5*), (*FloatR 160* (− *6*), *FloatR* (− *10454837*) (− *24*)), (*FloatR 168* (− *6*), *FloatR* (− *8491289*) (− *25*)),
FloatR 84 (− *5*), (*FloatR 84* (− *5*), *FloatR* (− *10409521*) (− *24*)), *FloatR 84* (− *5*), *FloatR* (− *9255556*) (− *24*)),
(*FloatR 76* (− *5*), (*FloatR 152* (− *6*), *FloatR* (− *11027878*) (− *25*)), (*FloatR 160* (− *6*), *FloatR 12605199* (− *28*)),
FloatR 80 (− *5*), (*FloatR 80* (− *5*), *FloatR* (− *10996003*) (− *25*)), *FloatR 80* (− *5*), *FloatR* (− *8530561*) (− *25*)),
(*FloatR 72* (− *5*), (*FloatR 144* (− *6*), *FloatR* (− *14225580*) (− *29*)), (*FloatR 152* (− *6*), *FloatR 11079600* (− *25*)),
FloatR 76 (− *5*), (*FloatR 76* (− *5*), *FloatR* (− *14225580*) (− *29*)), *FloatR 76* (− *5*), *FloatR 12587625* (− *28*)),
(*FloatR 68* (− *5*), (*FloatR 136* (− *6*), *FloatR 8760369* (− *25*)), (*FloatR

144 (− 6), FloatR 9674778 (− 24)),
FloatR 72 (− 5), (FloatR 72 (− 5), FloatR 8760369 (− 25)), FloatR 72 (− 5), FloatR 11057176 (− 25)),
(FloatR 64 (− 5), (FloatR 128 (− 6), FloatR 8648812 (− 24)), (FloatR 136 (− 6), FloatR 13022332 (− 24)),
FloatR 68 (− 5), (FloatR 68 (− 5), FloatR 8648812 (− 24)), FloatR 68 (− 5), FloatR 9657115 (− 24)),
(FloatR 60 (− 5), (FloatR 120 (− 6), FloatR 12157652 (− 24)), (FloatR 128 (− 6), FloatR 15549315 (− 24)),
FloatR 64 (− 5), (FloatR 64 (− 5), FloatR 12157652 (− 24)), FloatR 64 (− 5), FloatR 13002210 (− 24)),
(FloatR 56 (− 5), (FloatR 112 (− 6), FloatR 14848386 (− 24)), (FloatR 120 (− 6), FloatR 8661118 (− 23)),
FloatR 60 (− 5), (FloatR 60 (− 5), FloatR 14848386 (− 24)), FloatR 60 (− 5), FloatR 15529820 (− 24)),
(FloatR 52 (− 5), (FloatR 104 (− 6), FloatR 16774461 (− 24)), (FloatR 112 (− 6), FloatR 9231099 (− 23)),
FloatR 56 (− 5), (FloatR 56 (− 5), FloatR 16774461 (− 24)), FloatR 56 (− 5), FloatR 8652629 (− 23)),
(FloatR 48 (− 5), (FloatR 96 (− 6), FloatR 9022516 (− 23)), (FloatR 104 (− 6), FloatR 9549818 (− 23)),
FloatR 52 (− 5), (FloatR 52 (− 5), FloatR 9022516 (− 23)), FloatR 52 (− 5), FloatR 9224207 (− 23)),
(FloatR 44 (− 5), (FloatR 88 (− 6), FloatR 9393521 (− 23)), (FloatR 96 (− 6), FloatR 9675737 (− 23)),
FloatR 48 (− 5), (FloatR 48 (− 5), FloatR 9393521 (− 23)), FloatR 48 (− 5), FloatR 9544472 (− 23)),
(FloatR 40 (− 5), (FloatR 80 (− 6), FloatR 9559617 (− 23)), (FloatR 88 (− 6), FloatR 9683077 (− 23)),
FloatR 44 (− 5), (FloatR 44 (− 5), FloatR 9559617 (− 23)), FloatR 44 (− 5), FloatR 9671710 (− 23)),
(FloatR 36 (− 5), (FloatR 72 (− 6), FloatR 9459075 (− 23)), (FloatR 80 (− 6), FloatR 9656348 (− 23)),
FloatR 40 (− 5), (FloatR 40 (− 5), FloatR 9570310 (− 23)), FloatR 40 (− 5), FloatR 9653261 (− 23)),
(FloatR 32 (− 5), (FloatR 64 (− 6), FloatR 9266075 (− 23)), (FloatR 72 (− 6), FloatR 9527557 (− 23)),
FloatR 36 (− 5), (FloatR 36 (− 5), FloatR 9464260 (− 23)), FloatR 36 (− 5), FloatR 9525296 (− 23)),
(FloatR 28 (− 5), (FloatR 56 (− 6), FloatR 9004983 (− 23)), (FloatR 64 (− 6), FloatR 9316288 (− 23)),
FloatR 32 (− 5), (FloatR 32 (− 5), FloatR 9270001 (− 23)), FloatR 32 (− 5), FloatR 9314613 (− 23)),
(FloatR 24 (− 5), (FloatR 48 (− 6), FloatR 8690219 (− 23)), (FloatR 56 (− 6), FloatR 9041602 (− 23)),
FloatR 28 (− 5), (FloatR 28 (− 5), FloatR 9007999 (− 23)), FloatR 28 (− 5), FloatR 9040328 (− 23)),
(FloatR 20 (− 5), (FloatR 40 (− 6), FloatR 16663210 (− 24)), (FloatR 48 (− 6), FloatR 8716734 (− 23)),

FloatR 24 (− 5), (FloatR 24 (− 5), FloatR 8692586 (− 23)), FloatR 24 (− 5), FloatR 8715727 (− 23)),
(FloatR 16 (− 5), (FloatR 32 (− 6), FloatR 15871119 (− 24)), (FloatR 40 (− 6), FloatR 16701213 (− 24)),
FloatR 20 (− 5), (FloatR 20 (− 5), FloatR 16667039 (− 24)), FloatR 20 (− 5), FloatR 16699530 (− 24)),
(FloatR 12 (− 5), (FloatR 24 (− 6), FloatR 15011935 (− 24)), (FloatR 32 (− 6), FloatR 15897932 (− 24)),
FloatR 16 (− 5), (FloatR 16 (− 5), FloatR 15874331 (− 24)), FloatR 16 (− 5), FloatR 15896432 (− 24)),
(FloatR 8 (− 5), (FloatR 16 (− 6), FloatR 14089570 (− 24)), (FloatR 24 (− 6), FloatR 15030402 (− 24)),
FloatR 12 (− 5), (FloatR 12 (− 5), FloatR 15014747 (− 24)), FloatR 12 (− 5), FloatR 15028973 (− 24)),
(FloatR 4 (− 5), (FloatR 8 (− 6), FloatR 13104879 (− 24)), (FloatR 16 (− 6), FloatR 14101814 (− 24)),
FloatR 8 (− 5), (FloatR 8 (− 5), FloatR 14092148 (− 24)), FloatR 8 (− 5), FloatR 14100364 (− 24)),
(FloatR 0 0, (FloatR 0 (− 6), FloatR 12056138 (− 24)), (FloatR 8 (− 6), FloatR 13112497 (− 24)), FloatR 4 (− 5),
(FloatR 4 (− 5), FloatR 13107355 (− 24)), FloatR 4 (− 5), FloatR 13110949 (− 24))])
by *eval*

end
theory *Example3*
imports
../Numerics/Euler-Affine
../Numerics/Optimize-Float
~~/src/HOL/Decision-Procs/Approximation
begin

16.2 Example 3

approximate-affine *e3* λ(*t*, *x*). (*1::real*, *x*∗*x* + *t*∗*t::real*)

lemma *e3-fderiv*: ((λ(*t*, *x*). (*1::real*, *x*∗*x* + *t*∗*t::real*)) *has-derivative*
(λ(*x*, *y*) (*h*, *j*). (*0*, *2* ∗ (*j* ∗ *y*) + *2* ∗ (*h* ∗ *x*))) *x*) (*at x within X*)
by (*auto intro*!: *derivative-eq-intros simp*: *split-beta′*)

approximate-affine *e3-d* λ(*x*, *y*) (*h*, *j*). (*0::real*, *2* ∗ (*j* ∗ *y*) + *2* ∗ (*h* ∗ *x*)*::real*)

abbreviation *e3-ivp* ≡ λ*optns args*. *uncurry-options e3 optns* (*hd args*) (*tl args*)
abbreviation *e3-d-ivp* ≡ λ*optns args*. *uncurry-options e3-d optns* (*hd args*) (*hd* (*tl args*)) (*tl* (*tl args*))

interpretation *e3*: *aform-approximate-ivp*
e3-ivp
e3-d-ivp

λ(t, x). (1::real, x∗x + t∗t::real)
λ(x, y) (h, j). (0, 2 ∗ (j ∗ y) + 2 ∗ (h ∗ x))
apply *unfold-locales*
apply (*rule e3*[*THEN Joints2-JointsI*])
unfolding *list.sel* **apply** *assumption* **apply** *assumption*
apply (*drule length-set-of-apprs*, *simp*)— TODO: prove in affine-approximation
apply (*rule e3-fderiv*)
apply (*rule e3-d*[*THEN Joints2-JointsI*]) **apply** *assumption* **apply** *assumption*
apply (*drule length-set-of-apprs*, *simp*)— TODO: prove in affine-approximation
apply (*auto intro*!: *continuous-intros simp*: *split-beta′*)
done

definition *e3-optns = default-optns*
(| *precision := 30*,
tolerance := FloatR 1 (− 4),
stepsize := FloatR 1 (− 8),
result-fun := ivls-result 23 1,
printing-fun := (*λ- - -.* ()) |)

definition *e3test* = (*λ-::unit. euler-series-result e3-ivp e3-d-ivp e3-optns 0* (*aform-of-point* (*0*, *1*)) (*2 ˆ 5*))

lemma *e3test*: *e3test* () =
Some (*FloatR 32* (− *8*),
[(*FloatR 31* (− *8*), (*FloatR 62* (− *9*), *FloatR 9549658* (− *23*)), (*FloatR 64* (− *9*), *FloatR 9592906* (− *23*)),
FloatR 32 (− *8*), (*FloatR 32* (− *8*), *FloatR 9592812* (− *23*)), *FloatR 32* (− *8*), *FloatR 9592906* (− *23*)),
(*FloatR 30* (− *8*), (*FloatR 60* (− *9*), *FloatR 9506917* (− *23*)), (*FloatR 62* (− *9*), *FloatR 9549748* (− *23*)),
FloatR 31 (− *8*), (*FloatR 31* (− *8*), *FloatR 9549658* (− *23*)), *FloatR 31* (− *8*), *FloatR 9549748* (− *23*)),
(*FloatR 29* (− *8*), (*FloatR 58* (− *9*), *FloatR 9464583* (− *23*)), (*FloatR 60* (− *9*), *FloatR 9507004* (− *23*)),
FloatR 30 (− *8*), (*FloatR 30* (− *8*), *FloatR 9506918* (− *23*)), *FloatR 30* (− *8*), *FloatR 9507004* (− *23*)),
(*FloatR 28* (− *8*), (*FloatR 56* (− *9*), *FloatR 9422650* (− *23*)), (*FloatR 58* (− *9*), *FloatR 9464666* (− *23*)),
FloatR 29 (− *8*), (*FloatR 29* (− *8*), *FloatR 9464584* (− *23*)), *FloatR 29* (− *8*), *FloatR 9464666* (− *23*)),
(*FloatR 27* (− *8*), (*FloatR 54* (− *9*), *FloatR 9381111* (− *23*)), (*FloatR 56* (− *9*), *FloatR 9422729* (− *23*)),
FloatR 28 (− *8*), (*FloatR 28* (− *8*), *FloatR 9422650* (− *23*)), *FloatR 28* (− *8*), *FloatR 9422729* (− *23*)),
(*FloatR 26* (− *8*), (*FloatR 52* (− *9*), *FloatR 9339960* (− *23*)), (*FloatR 54* (− *9*), *FloatR 9381186* (− *23*)),
FloatR 27 (− *8*), (*FloatR 27* (− *8*), *FloatR 9381111* (− *23*)), *FloatR 27* (− *8*), *FloatR 9381186* (− *23*)),
(*FloatR 25* (− *8*), (*FloatR 50* (− *9*), *FloatR 9299191* (− *23*)), (*FloatR 52*

(− 9), FloatR 9340031 (− 23)),
FloatR 26 (− 8), (FloatR 26 (− 8), FloatR 9339960 (− 23)), FloatR 26 (− 8), FloatR 9340031 (− 23)),
(FloatR 24 (− 8), (FloatR 48 (− 9), FloatR 9258799 (− 23)), (FloatR 50 (− 9), FloatR 9299259 (− 23)),
FloatR 25 (− 8), (FloatR 25 (− 8), FloatR 9299191 (− 23)), FloatR 25 (− 8), FloatR 9299259 (− 23)),
(FloatR 23 (− 8), (FloatR 46 (− 9), FloatR 9218777 (− 23)), (FloatR 48 (− 9), FloatR 9258863 (− 23)),
FloatR 24 (− 8), (FloatR 24 (− 8), FloatR 9258799 (− 23)), FloatR 24 (− 8), FloatR 9258863 (− 23)),
(FloatR 22 (− 8), (FloatR 44 (− 9), FloatR 9179120 (− 23)), (FloatR 46 (− 9), FloatR 9218838 (− 23)),
FloatR 23 (− 8), (FloatR 23 (− 8), FloatR 9218777 (− 23)), FloatR 23 (− 8), FloatR 9218838 (− 23)),
(FloatR 21 (− 8), (FloatR 42 (− 9), FloatR 9139823 (− 23)), (FloatR 44 (− 9), FloatR 9179178 (− 23)),
FloatR 22 (− 8), (FloatR 22 (− 8), FloatR 9179121 (− 23)), FloatR 22 (− 8), FloatR 9179178 (− 23)),
(FloatR 20 (− 8), (FloatR 40 (− 9), FloatR 9100880 (− 23)), (FloatR 42 (− 9), FloatR 9139878 (− 23)),
FloatR 21 (− 8), (FloatR 21 (− 8), FloatR 9139824 (− 23)), FloatR 21 (− 8), FloatR 9139878 (− 23)),
(FloatR 19 (− 8), (FloatR 38 (− 9), FloatR 9062286 (− 23)), (FloatR 40 (− 9), FloatR 9100932 (− 23)),
FloatR 20 (− 8), (FloatR 20 (− 8), FloatR 9100880 (− 23)), FloatR 20 (− 8), FloatR 9100932 (− 23)),
(FloatR 18 (− 8), (FloatR 36 (− 9), FloatR 9024034 (− 23)), (FloatR 38 (− 9), FloatR 9062334 (− 23)),
FloatR 19 (− 8), (FloatR 19 (− 8), FloatR 9062286 (− 23)), FloatR 19 (− 8), FloatR 9062334 (− 23)),
(FloatR 17 (− 8), (FloatR 34 (− 9), FloatR 8986121 (− 23)), (FloatR 36 (− 9), FloatR 9024080 (− 23)),
FloatR 18 (− 8), (FloatR 18 (− 8), FloatR 9024035 (− 23)), FloatR 18 (− 8), FloatR 9024080 (− 23)),
(FloatR 16 (− 8), (FloatR 32 (− 9), FloatR 8948541 (− 23)), (FloatR 34 (− 9), FloatR 8986164 (− 23)),
FloatR 17 (− 8), (FloatR 17 (− 8), FloatR 8986121 (− 23)), FloatR 17 (− 8), FloatR 8986164 (− 23)),
(FloatR 15 (− 8), (FloatR 30 (− 9), FloatR 8911288 (− 23)), (FloatR 32 (− 9), FloatR 8948580 (− 23)),
FloatR 16 (− 8), (FloatR 16 (− 8), FloatR 8948541 (− 23)), FloatR 16 (− 8), FloatR 8948580 (− 23)),
(FloatR 14 (− 8), (FloatR 28 (− 9), FloatR 8874358 (− 23)), (FloatR 30 (− 9), FloatR 8911325 (− 23)),
FloatR 15 (− 8), (FloatR 15 (− 8), FloatR 8911288 (− 23)), FloatR 15 (− 8), FloatR 8911325 (− 23)),
(FloatR 13 (− 8), (FloatR 26 (− 9), FloatR 8837747 (− 23)), (FloatR 28 (− 9), FloatR 8874392 (− 23)),

FloatR 14 (− 8), (FloatR 14 (− 8), FloatR 8874359 (− 23)), FloatR 14 (− 8), FloatR 8874392 (− 23)),
(FloatR 12 (− 8), (FloatR 24 (− 9), FloatR 8801448 (− 23)), (FloatR 26 (− 9), FloatR 8837778 (− 23)),
FloatR 13 (− 8), (FloatR 13 (− 8), FloatR 8837747 (− 23)), FloatR 13 (− 8), FloatR 8837778 (− 23)),
(FloatR 11 (− 8), (FloatR 22 (− 9), FloatR 8765457 (− 23)), (FloatR 24 (− 9), FloatR 8801476 (− 23)),
FloatR 12 (− 8), (FloatR 12 (− 8), FloatR 8801448 (− 23)), FloatR 12 (− 8), FloatR 8801476 (− 23)),
(FloatR 10 (− 8), (FloatR 20 (− 9), FloatR 8729770 (− 23)), (FloatR 22 (− 9), FloatR 8765483 (− 23)),
FloatR 11 (− 8), (FloatR 11 (− 8), FloatR 8765457 (− 23)), FloatR 11 (− 8), FloatR 8765483 (− 23)),
(FloatR 9 (− 8), (FloatR 18 (− 9), FloatR 8694382 (− 23)), (FloatR 20 (− 9), FloatR 8729794 (− 23)),
FloatR 10 (− 8), (FloatR 10 (− 8), FloatR 8729770 (− 23)), FloatR 10 (− 8), FloatR 8729794 (− 23)),
(FloatR 8 (− 8), (FloatR 16 (− 9), FloatR 8659289 (− 23)), (FloatR 18 (− 9), FloatR 8694403 (− 23)),
FloatR 9 (− 8), (FloatR 9 (− 8), FloatR 8694382 (− 23)), FloatR 9 (− 8), FloatR 8694403 (− 23)),
(FloatR 7 (− 8), (FloatR 14 (− 9), FloatR 8624485 (− 23)), (FloatR 16 (− 9), FloatR 8659307 (− 23)),
FloatR 8 (− 8), (FloatR 8 (− 8), FloatR 8659289 (− 23)), FloatR 8 (− 8), FloatR 8659307 (− 23)),
(FloatR 6 (− 8), (FloatR 12 (− 9), FloatR 8589966 (− 23)), (FloatR 14 (− 9), FloatR 8624501 (− 23)),
FloatR 7 (− 8), (FloatR 7 (− 8), FloatR 8624485 (− 23)), FloatR 7 (− 8), FloatR 8624501 (− 23)),
(FloatR 5 (− 8), (FloatR 10 (− 9), FloatR 8555729 (− 23)), (FloatR 12 (− 9), FloatR 8589980 (− 23)),
FloatR 6 (− 8), (FloatR 6 (− 8), FloatR 8589966 (− 23)), FloatR 6 (− 8), FloatR 8589980 (− 23)),
(FloatR 4 (− 8), (FloatR 8 (− 9), FloatR 8521768 (− 23)), (FloatR 10 (− 9), FloatR 8555740 (− 23)),
FloatR 5 (− 8), (FloatR 5 (− 8), FloatR 8555729 (− 23)), FloatR 5 (− 8), FloatR 8555740 (− 23)),
(FloatR 3 (− 8), (FloatR 6 (− 9), FloatR 8488080 (− 23)), (FloatR 8 (− 9), FloatR 8521777 (− 23)),
FloatR 4 (− 8), (FloatR 4 (− 8), FloatR 8521768 (− 23)), FloatR 4 (− 8), FloatR 8521777 (− 23)),
(FloatR 2 (− 8), (FloatR 4 (− 9), FloatR 8454659 (− 23)), (FloatR 6 (− 9), FloatR 8488087 (− 23)),
FloatR 3 (− 8), (FloatR 3 (− 8), FloatR 8488080 (− 23)), FloatR 3 (− 8), FloatR 8488087 (− 23)),
(FloatR 1 (− 8), (FloatR 2 (− 9), FloatR 8421503 (− 23)), (FloatR 4 (− 9), FloatR 8454665 (− 23)),
FloatR 2 (− 8), (FloatR 2 (− 8), FloatR 8454660 (− 23)), FloatR 2 (−

8), *FloatR 8454665* (− *23*)),
(*FloatR 0 0*, (*FloatR 0* (− *9*), *FloatR 8388608* (− *23*)), (*FloatR 2* (− *9*), *FloatR 8421507* (− *23*)), *FloatR 1* (− *8*),
(*FloatR 1* (− *8*), *FloatR 8421503* (− *23*)), *FloatR 1* (− *8*), *FloatR 8421507* (− *23*))])
by *eval*

lemma *x0*: (*0*, *1*) ∈ *Affine* (*aform-of-point* (*0*::*real*, *1*::*real*))
by (*rule Affine-aform-of-point*)

lemma *stepsize*: *0* < *stepsize e3-optns*
by (*auto simp*: *e3-optns-def*)

lemma *result-fun*: *result-fun e3-optns* = *ivls-result 23 1*
by (*auto simp*: *e3-optns-def*)

lemmas *certification* = *e3.euler-series-ivls-result*[*OF stepsize x0 result-fun e3test*[*simplified e3test-def*],
simplified e3.euler-ivp-def]

lemma *last-enclosure*: *e3.enclosure*
(*ivp.solution*
(|*ivp-f* = λ(*t*, *x*). *case x of* (*t*, *x*) ⇒ (*1*, *x* ∗ *x* + *t* ∗ *t*), *ivp-t0* = *0*, *ivp-x0* = (*0*, *1*), *ivp-T* = {*0*..*FloatR 32* (− *8*)},
ivp-X = *UNIV*|))
0 (*FloatR 32* (− *8*))
(*map set-res-of-ivl-res*
[(*FloatR 31* (− *8*), (*FloatR 62* (− *9*), *FloatR 9549658* (− *23*)), (*FloatR 64* (− *9*), *FloatR 9592906* (− *23*)),
FloatR 32 (− *8*), (*FloatR 32* (− *8*), *FloatR 9592812* (− *23*)), *FloatR 32* (− *8*), *FloatR 9592906* (− *23*))])
using *certification*
unfolding *e3.enclosure-def*
apply (*subst* (*asm*) *list.map*)
apply (*subst* (*asm*) *list-all-simps*)
apply (*drule conjunct1*)
apply (*simp*)
done

lemma
unique-solution (|*ivp-f* = λ(*s*::*real*, *t*::*real*, *x*::*real*). (*1*, *x* ∗ *x* + *t* ∗ *t*), *ivp-t0* = *0*,
ivp-x0 = (*0*, *1*), *ivp-T* = {*0*..*1* / *8*}, *ivp-X* = *UNIV*|)
ivp.solution (|*ivp-f* = λ(*s*::*real*, *t*::*real*, *x*::*real*). (*1*, *x* ∗ *x* + *t* ∗ *t*), *ivp-t0* = *0*,
ivp-x0 = (*0*, *1*), *ivp-T* = {*0*..*1* / *8*}, *ivp-X* = *UNIV*|) (*1* / *8*) ∈
{(*1* / *8*, *2398203* / *2097152*) .. (*1* / *8*, *4796453* / *4194304*)}
using *certification*(*1*) *last-enclosure*

by (*simp-all add*: *e3.enclosure-def*)

16.2.1 Comparison with bounds analytically obtained by Walter [4] in section 9, Example V.

First approximation.

notepad begin
  fix *solution*
  assume *Walter*: $\bigwedge x$. *solution x* ∈ {*1/(1 − x)..tan(x + pi/4)*}
  let *?x = 0.125::real*
  value *1 / (1 − 0.125)*
  have *1/(1−?x)* ∈ {*1.142857139 .. 1.142857146*} **by** *simp*
  moreover
  approximate *tan (0.125 + pi/4)*
  have *tan(?x + pi/4)* ∈ {*1.287426935 .. 1.287426955*}
    by (*approximation 40*)
  ultimately
  have {*1/(1−?x)..tan(?x + pi/4)*} ⊆ {*1.142857139 .. 1.287426955*} **by** *simp*
  with *Walter* **have** *solution ?x* ∈ {*1.142857139 .. 1.287426955*} **by** *blast*
end

Better approximation.

notepad begin
  fix *solution::real⇒real*
  assume *Walter*: $\bigwedge x$. *solution x* ∈ {*1/(1−x)..16 / (16 − 17∗x)*}
  let *?x = 0.125::real*
  approximate *1 / (1 − ?x)*
  have *1/(1−?x)* ∈ {*1.142857139 .. 1.142857146*} **by** *simp*
  moreover
  approximate *16 / (16 − 17∗?x)*
  have *16 / (16 − 17∗?x)* ∈ {*1.153153151 .. 1.153153155*}
    by (*approximation 40*)
  ultimately
  have {*1/(1−?x)..16 / (16 − 17∗?x)*} ⊆ {*1.142857139 .. 1.153153155*} **by** *simp*
  with *Walter* **have** *solution ?x* ∈ {*1.142857139 .. 1.153153155*} **by** *blast*
  have *error*: *16 / (16 − 17∗?x) − 1 / (1 − ?x)* ≥ *1/10^2* **by** (*approximation 20*)
end

end
theory *Example-Exp*
imports
  ../Numerics/Euler-Affine
  ../Numerics/Optimize-Float
begin

16.3 Example Exponential

TODO: why not exp-ivp "lambda x::real. x"?

approximate-affine *exp-affine* $\lambda(x::real,\ y::real).\ (x,\ y)$

lemma *exp-ivp-fderiv*: $((\lambda(x::real,\ y::real).\ (x,\ y))$ *has-derivative* $(\lambda(a,\ b)\ (h_1,\ h_2).\ (h_1,\ h_2 + 0 * a * b))\ x)$ (*at x within X*)
  by (*auto intro*!: *derivative-eq-intros simp*: *split-beta id-def*)

approximate-affine *exp-d* $(\lambda(a::real,\ b::real)\ (h_1::real,\ h_2::real).\ (h_1,\ h_2 + 0 * a * b))$

abbreviation *exp-ivp* ≡ λ*optns args*. *uncurry-options exp-affine optns* (*hd args*) (*tl args*)
abbreviation *exp-d-ivp* ≡ λ*optns args*. *uncurry-options exp-d optns* (*hd args*) (*hd* (*tl args*)) (*tl* (*tl args*))

interpretation *exp-ivp*: *aform-approximate-ivp*
  exp-ivp
  exp-d-ivp
  $\lambda(y_1,\ y_2).\ (y_1,\ y_2)$
  $\lambda(a,\ b)\ (h_1,\ h_2).\ (h_1,\ h_2 + 0 * a * b)$
  apply *standard*
  apply (*rule exp-affine*[*THEN Joints2-JointsI*])
  unfolding *list.sel*
  apply *assumption* **apply** *assumption*
  apply (*drule length-set-of-apprs*, *simp*)— TODO: prove in affine-approximation
  apply (*rule exp-ivp-fderiv*)
  apply (*rule exp-d*[*THEN Joints2-JointsI*]) **apply** *assumption* **apply** *assumption*
  apply (*drule length-set-of-apprs*, *simp*)— TODO: prove in affine-approximation
  apply (*auto intro*!: *continuous-intros simp*: *split-beta*)
  done

definition *exp-optns* = *default-optns*
  (| *precision* := *40*,
   tolerance := *FloatR 1* (− *9*),
   stepsize := *FloatR 1* (− *6*),
   result-fun := *ivls-result 23 1*,
   iterations := *40*,
   widening-mod := *10*,
   printing-fun := (λ- - -. ())|)

definition *exptest* = (λ-::*unit*. *euler-series-result exp-ivp exp-d-ivp exp-optns 0*
  (*aform-of-point* (*1*, *1*)) (*2* ˆ *6*))

lemma *exptest* () = *Some* (*FloatR 64* (− *6*),
  [(*FloatR 63* (− *6*), (*FloatR 11224084* (− *22*), *FloatR 11224084* (− *22*)), (*FloatR 11402234* (− *22*), *FloatR 11402234* (− *22*)),
  FloatR 64 (− *6*), (*FloatR 11400841* (− *22*), *FloatR 11400841* (− *22*)), *FloatR 11402234* (− *22*), *FloatR 11402234* (− *22*)),
  (*FloatR 62* (− *6*), (*FloatR 11050078* (− *22*), *FloatR 11050078* (− *22*)), (*FloatR 11225445* (− *22*), *FloatR 11225445* (− *22*)),

FloatR 63 (− *6*), (*FloatR 11224095* (− *22*), *FloatR 11224095* (− *22*)), *FloatR 11225445* (− *22*), *FloatR 11225445* (− *22*)),
(*FloatR 61* (− *6*), (*FloatR 10878769* (− *22*), *FloatR 10878769* (− *22*)), (*FloatR 11051396* (− *22*), *FloatR 11051396* (− *22*)),
FloatR 62 (− *6*), (*FloatR 11050088* (− *22*), *FloatR 11050088* (− *22*)), *FloatR 11051396* (− *22*), *FloatR 11051396* (− *22*)),
(*FloatR 60* (− *6*), (*FloatR 10710117* (− *22*), *FloatR 10710117* (− *22*)), (*FloatR 10880046* (− *22*), *FloatR 10880046* (− *22*)),
FloatR 61 (− *6*), (*FloatR 10878779* (− *22*), *FloatR 10878779* (− *22*)), *FloatR 10880046* (− *22*), *FloatR 10880046* (− *22*)),
(*FloatR 59* (− *6*), (*FloatR 10544078* (− *22*), *FloatR 10544078* (− *22*)), (*FloatR 10711353* (− *22*), *FloatR 10711353* (− *22*)),
FloatR 60 (− *6*), (*FloatR 10710126* (− *22*), *FloatR 10710126* (− *22*)), *FloatR 10711353* (− *22*), *FloatR 10711353* (− *22*)),
(*FloatR 58* (− *6*), (*FloatR 10380614* (− *22*), *FloatR 10380614* (− *22*)), (*FloatR 10545275* (− *22*), *FloatR 10545275* (− *22*)),
FloatR 59 (− *6*), (*FloatR 10544088* (− *22*), *FloatR 10544088* (− *22*)), *FloatR 10545275* (− *22*), *FloatR 10545275* (− *22*)),
(*FloatR 57* (− *6*), (*FloatR 10219684* (− *22*), *FloatR 10219684* (− *22*)), (*FloatR 10381773* (− *22*), *FloatR 10381773* (− *22*)),
FloatR 58 (− *6*), (*FloatR 10380623* (− *22*), *FloatR 10380623* (− *22*)), *FloatR 10381773* (− *22*), *FloatR 10381773* (− *22*)),
(*FloatR 56* (− *6*), (*FloatR 10061249* (− *22*), *FloatR 10061249* (− *22*)), (*FloatR 10220805* (− *22*), *FloatR 10220805* (− *22*)),
FloatR 57 (− *6*), (*FloatR 10219693* (− *22*), *FloatR 10219693* (− *22*)), *FloatR 10220805* (− *22*), *FloatR 10220805* (− *22*)),
(*FloatR 55* (− *6*), (*FloatR 9905270* (− *22*), *FloatR 9905270* (− *22*)), (*FloatR 10062334* (− *22*), *FloatR 10062334* (− *22*)),
FloatR 56 (− *6*), (*FloatR 10061258* (− *22*), *FloatR 10061258* (− *22*)), *FloatR 10062334* (− *22*), *FloatR 10062334* (− *22*)),
(*FloatR 54* (− *6*), (*FloatR 9751710* (− *22*), *FloatR 9751710* (− *22*)), (*FloatR 9906319* (− *22*), *FloatR 9906319* (− *22*)),
FloatR 55 (− *6*), (*FloatR 9905279* (− *22*), *FloatR 9905279* (− *22*)), *FloatR 9906319* (− *22*), *FloatR 9906319* (− *22*)),
(*FloatR 53* (− *6*), (*FloatR 9600530* (− *22*), *FloatR 9600530* (− *22*)), (*FloatR 9752723* (− *22*), *FloatR 9752723* (− *22*)),
FloatR 54 (− *6*), (*FloatR 9751718* (− *22*), *FloatR 9751718* (− *22*)), *FloatR 9752723* (− *22*), *FloatR 9752723* (− *22*)),
(*FloatR 52* (− *6*), (*FloatR 9451693* (− *22*), *FloatR 9451693* (− *22*)), (*FloatR 9601509* (− *22*), *FloatR 9601509* (− *22*)),
FloatR 53 (− *6*), (*FloatR 9600537* (− *22*), *FloatR 9600537* (− *22*)), *FloatR 9601509* (− *22*), *FloatR 9601509* (− *22*)),
(*FloatR 51* (− *6*), (*FloatR 9305164* (− *22*), *FloatR 9305164* (− *22*)), (*FloatR 9452639* (− *22*), *FloatR 9452639* (− *22*)),
FloatR 52 (− *6*), (*FloatR 9451701* (− *22*), *FloatR 9451701* (− *22*)), *FloatR 9452639* (− *22*), *FloatR 9452639* (− *22*)),
(*FloatR 50* (− *6*), (*FloatR 9160907* (− *22*), *FloatR 9160907* (− *22*)), (*FloatR 9306078* (− *22*), *FloatR 9306078* (− *22*)),
FloatR 51 (− *6*), (*FloatR 9305171* (− *22*), *FloatR 9305171* (− *22*)), *FloatR*

9306078 (− *22*), *FloatR 9306078* (− *22*)),
(*FloatR 49* (− *6*), (*FloatR 9018886* (− *22*), *FloatR 9018886* (− *22*)), (*FloatR 9161789* (− *22*), *FloatR 9161789* (− *22*)),
FloatR 50 (− *6*), (*FloatR 9160914* (− *22*), *FloatR 9160914* (− *22*)), *FloatR 9161789* (− *22*), *FloatR 9161789* (− *22*)),
(*FloatR 48* (− *6*), (*FloatR 8879067* (− *22*), *FloatR 8879067* (− *22*)), (*FloatR 9019737* (− *22*), *FloatR 9019737* (− *22*)),
FloatR 49 (− *6*), (*FloatR 9018893* (− *22*), *FloatR 9018893* (− *22*)), *FloatR 9019737* (− *22*), *FloatR 9019737* (− *22*)),
(*FloatR 47* (− *6*), (*FloatR 8741415* (− *22*), *FloatR 8741415* (− *22*)), (*FloatR 8879887* (− *22*), *FloatR 8879887* (− *22*)),
FloatR 48 (− *6*), (*FloatR 8879073* (− *22*), *FloatR 8879073* (− *22*)), *FloatR 8879887* (− *22*), *FloatR 8879887* (− *22*)),
(*FloatR 46* (− *6*), (*FloatR 8605898* (− *22*), *FloatR 8605898* (− *22*)), (*FloatR 8742206* (− *22*), *FloatR 8742206* (− *22*)),
FloatR 47 (− *6*), (*FloatR 8741422* (− *22*), *FloatR 8741422* (− *22*)), *FloatR 8742206* (− *22*), *FloatR 8742206* (− *22*)),
(*FloatR 45* (− *6*), (*FloatR 8472481* (− *22*), *FloatR 8472481* (− *22*)), (*FloatR 8606660* (− *22*), *FloatR 8606660* (− *22*)),
FloatR 46 (− *6*), (*FloatR 8605904* (− *22*), *FloatR 8605904* (− *22*)), *FloatR 8606660* (− *22*), *FloatR 8606660* (− *22*)),
(*FloatR 44* (− *6*), (*FloatR 16682266* (− *23*), *FloatR 16682266* (− *23*)), (*FloatR 8473215* (− *22*), *FloatR 8473215* (− *22*)),
FloatR 45 (− *6*), (*FloatR 8472487* (− *22*), *FloatR 8472487* (− *22*)), *FloatR 8473215* (− *22*), *FloatR 8473215* (− *22*)),
(*FloatR 43* (− *6*), (*FloatR 16423642* (− *23*), *FloatR 16423642* (− *23*)), (*FloatR 16683679* (− *23*), *FloatR 16683679* (− *23*)),
FloatR 44 (− *6*), (*FloatR 16682277* (− *23*), *FloatR 16682277* (− *23*)), *FloatR 16683679* (− *23*), *FloatR 16683679* (− *23*)),
(*FloatR 42* (− *6*), (*FloatR 16169028* (− *23*), *FloatR 16169028* (− *23*)), (*FloatR 16425001* (− *23*), *FloatR 16425001* (− *23*)),
FloatR 43 (− *6*), (*FloatR 16423653* (− *23*), *FloatR 16423653* (− *23*)), *FloatR 16425001* (− *23*), *FloatR 16425001* (− *23*)),
(*FloatR 41* (− *6*), (*FloatR 15918360* (− *23*), *FloatR 15918360* (− *23*)), (*FloatR 16170334* (− *23*), *FloatR 16170334* (− *23*)),
FloatR 42 (− *6*), (*FloatR 16169038* (− *23*), *FloatR 16169038* (− *23*)), *FloatR 16170334* (− *23*), *FloatR 16170334* (− *23*)),
(*FloatR 40* (− *6*), (*FloatR 15671579* (− *23*), *FloatR 15671579* (− *23*)), (*FloatR 15919616* (− *23*), *FloatR 15919616* (− *23*)),
FloatR 41 (− *6*), (*FloatR 15918370* (− *23*), *FloatR 15918370* (− *23*)), *FloatR 15919616* (− *23*), *FloatR 15919616* (− *23*)),
(*FloatR 39* (− *6*), (*FloatR 15428624* (− *23*), *FloatR 15428624* (− *23*)), (*FloatR 15672785* (− *23*), *FloatR 15672785* (− *23*)),
FloatR 40 (− *6*), (*FloatR 15671589* (− *23*), *FloatR 15671589* (− *23*)), *FloatR 15672785* (− *23*), *FloatR 15672785* (− *23*)),
(*FloatR 38* (− *6*), (*FloatR 15189435* (− *23*), *FloatR 15189435* (− *23*)), (*FloatR 15429782* (− *23*), *FloatR 15429782* (− *23*)),
FloatR 39 (− *6*), (*FloatR 15428633* (− *23*), *FloatR 15428633* (− *23*)), *FloatR 15429782* (− *23*), *FloatR 15429782* (− *23*)),

(*FloatR 37* (− *6*), (*FloatR 14953954* (− *23*), *FloatR 14953954* (− *23*)),
(*FloatR 15190546* (− *23*), *FloatR 15190546* (− *23*)),
FloatR 38 (− *6*), (*FloatR 15189444* (− *23*), *FloatR 15189444* (− *23*)),
FloatR 15190546 (− *23*), *FloatR 15190546* (− *23*)),
(*FloatR 36* (− *6*), (*FloatR 14722124* (− *23*), *FloatR 14722124* (− *23*)),
(*FloatR 14955019* (− *23*), *FloatR 14955019* (− *23*)),
FloatR 37 (− *6*), (*FloatR 14953962* (− *23*), *FloatR 14953962* (− *23*)),
FloatR 14955019 (− *23*), *FloatR 14955019* (− *23*)),
(*FloatR 35* (− *6*), (*FloatR 14493888* (− *23*), *FloatR 14493888* (− *23*)),
(*FloatR 14723144* (− *23*), *FloatR 14723144* (− *23*)),
FloatR 36 (− *6*), (*FloatR 14722132* (− *23*), *FloatR 14722132* (− *23*)),
FloatR 14723144 (− *23*), *FloatR 14723144* (− *23*)),
(*FloatR 34* (− *6*), (*FloatR 14269190* (− *23*), *FloatR 14269190* (− *23*)),
(*FloatR 14494864* (− *23*), *FloatR 14494864* (− *23*)),
FloatR 35 (− *6*), (*FloatR 14493896* (− *23*), *FloatR 14493896* (− *23*)),
FloatR 14494864 (− *23*), *FloatR 14494864* (− *23*)),
(*FloatR 33* (− *6*), (*FloatR 14047976* (− *23*), *FloatR 14047976* (− *23*)),
(*FloatR 14270124* (− *23*), *FloatR 14270124* (− *23*)),
FloatR 34 (− *6*), (*FloatR 14269198* (− *23*), *FloatR 14269198* (− *23*)),
FloatR 14270124 (− *23*), *FloatR 14270124* (− *23*)),
(*FloatR 32* (− *6*), (*FloatR 13830191* (− *23*), *FloatR 13830191* (− *23*)),
(*FloatR 14048868* (− *23*), *FloatR 14048868* (− *23*)),
FloatR 33 (− *6*), (*FloatR 14047983* (− *23*), *FloatR 14047983* (− *23*)),
FloatR 14048868 (− *23*), *FloatR 14048868* (− *23*)),
(*FloatR 31* (− *6*), (*FloatR 13615783* (− *23*), *FloatR 13615783* (− *23*)),
(*FloatR 13831043* (− *23*), *FloatR 13831043* (− *23*)),
FloatR 32 (− *6*), (*FloatR 13830198* (− *23*), *FloatR 13830198* (− *23*)),
FloatR 13831043 (− *23*), *FloatR 13831043* (− *23*)),
(*FloatR 30* (− *6*), (*FloatR 13404698* (− *23*), *FloatR 13404698* (− *23*)),
(*FloatR 13616595* (− *23*), *FloatR 13616595* (− *23*)),
FloatR 31 (− *6*), (*FloatR 13615789* (− *23*), *FloatR 13615789* (− *23*)),
FloatR 13616595 (− *23*), *FloatR 13616595* (− *23*)),
(*FloatR 29* (− *6*), (*FloatR 13196886* (− *23*), *FloatR 13196886* (− *23*)),
(*FloatR 13405472* (− *23*), *FloatR 13405472* (− *23*)),
FloatR 30 (− *6*), (*FloatR 13404704* (− *23*), *FloatR 13404704* (− *23*)),
FloatR 13405472 (− *23*), *FloatR 13405472* (− *23*)),
(*FloatR 28* (− *6*), (*FloatR 12992296* (− *23*), *FloatR 12992296* (− *23*)),
(*FloatR 13197623* (− *23*), *FloatR 13197623* (− *23*)),
FloatR 29 (− *6*), (*FloatR 13196892* (− *23*), *FloatR 13196892* (− *23*)),
FloatR 13197623 (− *23*), *FloatR 13197623* (− *23*)),
(*FloatR 27* (− *6*), (*FloatR 12790877* (− *23*), *FloatR 12790877* (− *23*)),
(*FloatR 12992996* (− *23*), *FloatR 12992996* (− *23*)),
FloatR 28 (− *6*), (*FloatR 12992301* (− *23*), *FloatR 12992301* (− *23*)),
FloatR 12992996 (− *23*), *FloatR 12992996* (− *23*)),
(*FloatR 26* (− *6*), (*FloatR 12592581* (− *23*), *FloatR 12592581* (− *23*)),
(*FloatR 12791542* (− *23*), *FloatR 12791542* (− *23*)),
FloatR 27 (− *6*), (*FloatR 12790882* (− *23*), *FloatR 12790882* (− *23*)),
FloatR 12791542 (− *23*), *FloatR 12791542* (− *23*)),
(*FloatR 25* (− *6*), (*FloatR 12397359* (− *23*), *FloatR 12397359* (− *23*)),

(*FloatR 12593211* (− *23*), *FloatR 12593211* (− *23*)),
FloatR 26 (− *6*), (*FloatR 12592586* (− *23*), *FloatR 12592586* (− *23*)),
FloatR 12593211 (− *23*), *FloatR 12593211* (− *23*)),
(*FloatR 24* (− *6*), (*FloatR 12205164* (− *23*), *FloatR 12205164* (− *23*)),
(*FloatR 12397956* (− *23*), *FloatR 12397956* (− *23*)),
FloatR 25 (− *6*), (*FloatR 12397364* (− *23*), *FloatR 12397364* (− *23*)),
FloatR 12397956 (− *23*), *FloatR 12397956* (− *23*)),
(*FloatR 23* (− *6*), (*FloatR 12015948* (− *23*), *FloatR 12015948* (− *23*)),
(*FloatR 12205728* (− *23*), *FloatR 12205728* (− *23*)),
FloatR 24 (− *6*), (*FloatR 12205168* (− *23*), *FloatR 12205168* (− *23*)),
FloatR 12205728 (− *23*), *FloatR 12205728* (− *23*)),
(*FloatR 22* (− *6*), (*FloatR 11829665* (− *23*), *FloatR 11829665* (− *23*)),
(*FloatR 12016480* (− *23*), *FloatR 12016480* (− *23*)),
FloatR 23 (− *6*), (*FloatR 12015952* (− *23*), *FloatR 12015952* (− *23*)),
FloatR 12016480 (− *23*), *FloatR 12016480* (− *23*)),
(*FloatR 21* (− *6*), (*FloatR 11646271* (− *23*), *FloatR 11646271* (− *23*)),
(*FloatR 11830167* (− *23*), *FloatR 11830167* (− *23*)),
FloatR 22 (− *6*), (*FloatR 11829669* (− *23*), *FloatR 11829669* (− *23*)),
FloatR 11830167 (− *23*), *FloatR 11830167* (− *23*)),
(*FloatR 20* (− *6*), (*FloatR 11465720* (− *23*), *FloatR 11465720* (− *23*)),
(*FloatR 11646742* (− *23*), *FloatR 11646742* (− *23*)),
FloatR 21 (− *6*), (*FloatR 11646275* (− *23*), *FloatR 11646275* (− *23*)),
FloatR 11646742 (− *23*), *FloatR 11646742* (− *23*)),
(*FloatR 19* (− *6*), (*FloatR 11287967* (− *23*), *FloatR 11287967* (− *23*)),
(*FloatR 11466162* (− *23*), *FloatR 11466162* (− *23*)),
FloatR 20 (− *6*), (*FloatR 11465723* (− *23*), *FloatR 11465723* (− *23*)),
FloatR 11466162 (− *23*), *FloatR 11466162* (− *23*)),
(*FloatR 18* (− *6*), (*FloatR 11112971* (− *23*), *FloatR 11112971* (− *23*)),
(*FloatR 11288381* (− *23*), *FloatR 11288381* (− *23*)),
FloatR 19 (− *6*), (*FloatR 11287971* (− *23*), *FloatR 11287971* (− *23*)),
FloatR 11288381 (− *23*), *FloatR 11288381* (− *23*)),
(*FloatR 17* (− *6*), (*FloatR 10940687* (− *23*), *FloatR 10940687* (− *23*)),
(*FloatR 11113356* (− *23*), *FloatR 11113356* (− *23*)),
FloatR 18 (− *6*), (*FloatR 11112974* (− *23*), *FloatR 11112974* (− *23*)),
FloatR 11113356 (− *23*), *FloatR 11113356* (− *23*)),
(*FloatR 16* (− *6*), (*FloatR 10771075* (− *23*), *FloatR 10771075* (− *23*)),
(*FloatR 10941046* (− *23*), *FloatR 10941046* (− *23*)),
FloatR 17 (− *6*), (*FloatR 10940690* (− *23*), *FloatR 10940690* (− *23*)),
FloatR 10941046 (− *23*), *FloatR 10941046* (− *23*)),
(*FloatR 15* (− *6*), (*FloatR 10604091* (− *23*), *FloatR 10604091* (− *23*)),
(*FloatR 10771407* (− *23*), *FloatR 10771407* (− *23*)),
FloatR 16 (− *6*), (*FloatR 10771077* (− *23*), *FloatR 10771077* (− *23*)),
FloatR 10771407 (− *23*), *FloatR 10771407* (− *23*)),
(*FloatR 14* (− *6*), (*FloatR 10439697* (− *23*), *FloatR 10439697* (− *23*)),
(*FloatR 10604398* (− *23*), *FloatR 10604398* (− *23*)),
FloatR 15 (− *6*), (*FloatR 10604094* (− *23*), *FloatR 10604094* (− *23*)),
FloatR 10604398 (− *23*), *FloatR 10604398* (− *23*)),
(*FloatR 13* (− *6*), (*FloatR 10277851* (− *23*), *FloatR 10277851* (− *23*)),
(*FloatR 10439979* (− *23*), *FloatR 10439979* (− *23*)),

FloatR 14 (− 6), (FloatR 10439699 (− 23), FloatR 10439699 (− 23)), FloatR 10439979 (− 23), FloatR 10439979 (− 23)),

(FloatR 12 (− 6), (FloatR 10118514 (− 23), FloatR 10118514 (− 23)), (FloatR 10278109 (− 23), FloatR 10278109 (− 23)),

FloatR 13 (− 6), (FloatR 10277853 (− 23), FloatR 10277853 (− 23)), FloatR 10278109 (− 23), FloatR 10278109 (− 23)),

(FloatR 11 (− 6), (FloatR 9961647 (− 23), FloatR 9961647 (− 23)), (FloatR 10118749 (− 23), FloatR 10118749 (− 23)),

FloatR 12 (− 6), (FloatR 10118516 (− 23), FloatR 10118516 (− 23)), FloatR 10118749 (− 23), FloatR 10118749 (− 23)),

(FloatR 10 (− 6), (FloatR 9807213 (− 23), FloatR 9807213 (− 23)), (FloatR 9961859 (− 23), FloatR 9961859 (− 23)),

FloatR 11 (− 6), (FloatR 9961649 (− 23), FloatR 9961649 (− 23)), FloatR 9961859 (− 23), FloatR 9961859 (− 23)),

(FloatR 9 (− 6), (FloatR 9655172 (− 23), FloatR 9655172 (− 23)), (FloatR 9807402 (− 23), FloatR 9807402 (− 23)),

FloatR 10 (− 6), (FloatR 9807214 (− 23), FloatR 9807214 (− 23)), FloatR 9807402 (− 23), FloatR 9807402 (− 23)),

(FloatR 8 (− 6), (FloatR 9505489 (− 23), FloatR 9505489 (− 23)), (FloatR 9655340 (− 23), FloatR 9655340 (− 23)),

FloatR 9 (− 6), (FloatR 9655174 (− 23), FloatR 9655174 (− 23)), FloatR 9655340 (− 23), FloatR 9655340 (− 23)),

(FloatR 7 (− 6), (FloatR 9358126 (− 23), FloatR 9358126 (− 23)), (FloatR 9505636 (− 23), FloatR 9505636 (− 23)),

FloatR 8 (− 6), (FloatR 9505490 (− 23), FloatR 9505490 (− 23)), FloatR 9505636 (− 23), FloatR 9505636 (− 23)),

(FloatR 6 (− 6), (FloatR 9213047 (− 23), FloatR 9213047 (− 23)), (FloatR 9358253 (− 23), FloatR 9358253 (− 23)),

FloatR 7 (− 6), (FloatR 9358127 (− 23), FloatR 9358127 (− 23)), FloatR 9358253 (− 23), FloatR 9358253 (− 23)),

(FloatR 5 (− 6), (FloatR 9070218 (− 23), FloatR 9070218 (− 23)), (FloatR 9213155 (− 23), FloatR 9213155 (− 23)),

FloatR 6 (− 6), (FloatR 9213048 (− 23), FloatR 9213048 (− 23)), FloatR 9213155 (− 23), FloatR 9213155 (− 23)),

(FloatR 4 (− 6), (FloatR 8929603 (− 23), FloatR 8929603 (− 23)), (FloatR 9070306 (− 23), FloatR 9070306 (− 23)),

FloatR 5 (− 6), (FloatR 9070219 (− 23), FloatR 9070219 (− 23)), FloatR 9070306 (− 23), FloatR 9070306 (− 23)),

(FloatR 3 (− 6), (FloatR 8791168 (− 23), FloatR 8791168 (− 23)), (FloatR 8929673 (− 23), FloatR 8929673 (− 23)),

FloatR 4 (− 6), (FloatR 8929604 (− 23), FloatR 8929604 (− 23)), FloatR 8929673 (− 23), FloatR 8929673 (− 23)),

(FloatR 2 (− 6), (FloatR 8654879 (− 23), FloatR 8654879 (− 23)), (FloatR 8791220 (− 23), FloatR 8791220 (− 23)),

FloatR 3 (− 6), (FloatR 8791169 (− 23), FloatR 8791169 (− 23)), FloatR 8791220 (− 23), FloatR 8791220 (− 23)),

(FloatR 1 (− 6), (FloatR 8520703 (− 23), FloatR 8520703 (− 23)), (FloatR 8654914 (− 23), FloatR 8654914 (− 23)),

FloatR 2 (− 6), (FloatR 8654880 (− 23), FloatR 8654880 (− 23)), FloatR

8654914 (− *23*), *FloatR 8654914* (− *23*)),
(*FloatR 0 0*, (*FloatR 8388608* (− *23*), *FloatR 8388608* (− *23*)), (*FloatR 8520721* (− *23*), *FloatR 8520721* (− *23*)), *FloatR 1* (− *6*),
(*FloatR 8520703* (− *23*), *FloatR 8520703* (− *23*)), *FloatR 8520721* (− *23*), *FloatR 8520721* (− *23*))])
by *eval*

end
theory *Example-Oil*
imports
../Numerics/Euler-Affine
../Numerics/Optimize-Float
begin

16.4 Oil reservoir in Affine arithmetic

approximate-affine *oil* λ(*y*::*real*, *z*::*real*). (*z*, *z*∗*z* + −*3* ∗ *inverse* (*inverse 1000* + *y*∗*y*))

lemma *oil-deriv-ok*: **fixes** *y*::*real*
shows *1* / *1000* + *y*∗*y* = *0* ⟷ *False*
proof −
have *1* / *1000* + *y*∗*y* > *0*
by (*auto intro*!: *add-pos-nonneg*)
thus *?thesis* **by** *auto*
qed

lemma *oil-fderiv*: ((λ(*y*::*real*, *z*::*real*). (*z*, *z* ∗ *z* + −*3* ∗ *inverse* (*inverse 1000* + *y* ∗ *y*))) *has-derivative*
(*case x of* (*y*, *z*) ⇒ λ(*dy*, *dz*). (*dz*, *2* ∗ *dz* ∗ *z* + *6* ∗ (*inverse* (*inverse 1000* + *y* ∗ *y*) ∗ (*dy* ∗ (*y* ∗ *inverse* (*inverse 1000* + *y* ∗ *y*)))))))
(*at x within X*)
by (*auto intro*!: *derivative-eq-intros simp*: *oil-deriv-ok split-beta inverse-eq-divide*)

approximate-affine *oil-d* λ(*y*::*real*, *z*) (*dy*, *dz*). (*dz*, *2* ∗ *dz* ∗ *z* + *6* ∗ (*inverse* (*inverse 1000* + *y*∗*y*) ∗ (*dy* ∗ (*y* ∗ *inverse* (*inverse 1000* + *y*∗*y*)))))

abbreviation *oil-ivp* ≡ λ*optns args*. *uncurry-options oil optns* (*hd args*) (*tl args*)
abbreviation *oil-d-ivp* ≡ λ*optns args*. *uncurry-options oil-d optns* (*hd args*) (*hd* (*tl args*)) (*tl* (*tl args*))

interpretation *oil*: *aform-approximate-ivp*
oil-ivp oil-d-ivp
λ(*y*::*real*, *z*::*real*). (*z*, *z*∗*z* + −*3* ∗ *inverse* (*inverse 1000* + *y*∗*y*))
λ(*y*::*real*, *z*) (*dy*, *dz*).
(*dz*, *2* ∗ *dz* ∗ *z* + *6* ∗ (*inverse* (*inverse 1000* + *y*∗*y*) ∗ (*dy* ∗ (*y* ∗ *inverse* (*inverse 1000* + *y*∗*y*)))))
apply *standard*
apply (*rule oil*[*THEN Joints2-JointsI*])

unfolding *list.sel*
apply *assumption* **apply** *assumption*
apply (*drule length-set-of-apprs*, *simp*)— TODO: prove in affine-approximation
apply (*rule oil-fderiv*)
apply (*rule oil-d*[*THEN Joints2-JointsI*]) **apply** *assumption* **apply** *assumption*
apply (*drule length-set-of-apprs*, *simp*)— TODO: prove in affine-approximation
apply (*auto intro*!: *continuous-intros simp*: *split-beta oil-deriv-ok*)
done

definition *rough-optns* = *default-optns*
(| *precision* := *50*,
tolerance := *FloatR 1* (− *3*),
stepsize := *FloatR 1* (− *9*),
result-fun := *ivls-result 20 110*,
iterations := *20*,
widening-mod := *20*,
printing-fun := (λ- - -. ())|)

definition *oiltest-rough* =
(λ-::*unit*. *euler-series-result oil-ivp oil-d-ivp rough-optns 0*
(*aform-of-point* (*10*, *0*)) *22000*)

lemma *oiltest-rough* () =
Some (*FloatR 175831* (− *12*),
[(*FloatR 174951* (− *12*), (*FloatR* (− *899741*) (− *17*), *FloatR* (− *666307*) (− *21*)),
(*FloatR* (− *793547*) (− *17*), *FloatR* (− *591743*) (− *21*)), *FloatR 175831* (− *12*),
(*FloatR* (− *899741*) (− *17*), *FloatR* (− *658317*) (− *21*)), *FloatR* (− *801547*) (− *17*),
FloatR (− *591821*) (− *21*)),
(*FloatR 174071* (− *12*), (*FloatR* (− *890848*) (− *17*), *FloatR* (− *674567*) (− *21*)),
(*FloatR* (− *785451*) (− *17*), *FloatR* (− *599104*) (− *21*)), *FloatR 174951* (− *12*),
(*FloatR* (− *890848*) (− *17*), *FloatR* (− *666230*) (− *21*)), *FloatR* (− *793551*) (− *17*),
FloatR (− *599183*) (− *21*)),
(*FloatR 173191* (− *12*), (*FloatR* (− *881846*) (− *17*), *FloatR* (− *683203*) (− *21*)),
(*FloatR* (− *777248*) (− *17*), *FloatR* (− *606801*) (− *21*)), *FloatR 174071* (− *12*),
(*FloatR* (− *881846*) (− *17*), *FloatR* (− *674489*) (− *21*)), *FloatR* (− *785455*) (− *17*),
FloatR (− *606882*) (− *21*)),
(*FloatR 172311* (− *12*), (*FloatR* (− *872731*) (− *17*), *FloatR* (− *692248*) (− *21*)),
(*FloatR* (− *768935*) (− *17*), *FloatR* (− *614865*) (− *21*)), *FloatR 173191* (− *12*),

(*FloatR* (− *872731*) (− *17*), *FloatR* (− *683123*) (− *21*)), *FloatR* (− *777252*) (− *17*),
FloatR (− *614947*) (− *21*)),
(*FloatR* *171431* (− *12*), (*FloatR* (− *863498*) (− *17*), *FloatR* (− *701740*) (− *21*)),
(*FloatR* (− *760505*) (− *17*), *FloatR* (− *623329*) (− *21*)), *FloatR* *172311* (− *12*),
(*FloatR* (− *863498*) (− *17*), *FloatR* (− *692166*) (− *21*)), *FloatR* (− *768939*) (− *17*),
FloatR (− *623413*) (− *21*)),
(*FloatR* *170551* (− *12*), (*FloatR* (− *854140*) (− *17*), *FloatR* (− *711722*) (− *21*)),
(*FloatR* (− *751953*) (− *17*), *FloatR* (− *632231*) (− *21*)), *FloatR* *171431* (− *12*),
(*FloatR* (− *854140*) (− *17*), *FloatR* (− *701656*) (− *21*)), *FloatR* (− *760510*) (− *17*),
FloatR (− *632318*) (− *21*)),
(*FloatR* *169671* (− *12*), (*FloatR* (− *844651*) (− *17*), *FloatR* (− *722245*) (− *21*)),
(*FloatR* (− *743271*) (− *17*), *FloatR* (− *641616*) (− *21*)), *FloatR* *170551* (− *12*),
(*FloatR* (− *844651*) (− *17*), *FloatR* (− *711637*) (− *21*)), *FloatR* (− *751957*) (− *17*),
FloatR (− *641705*) (− *21*)),
(*FloatR* *168791* (− *12*), (*FloatR* (− *835025*) (− *17*), *FloatR* (− *733366*) (− *21*)),
(*FloatR* (− *734452*) (− *17*), *FloatR* (− *651534*) (− *21*)), *FloatR* *169671* (− *12*),
(*FloatR* (− *835025*) (− *17*), *FloatR* (− *722158*) (− *21*)), *FloatR* (− *743275*) (− *17*),
FloatR (− *651624*) (− *21*)),
(*FloatR* *167911* (− *12*), (*FloatR* (− *825253*) (− *17*), *FloatR* (− *745150*) (− *21*)),
(*FloatR* (− *725487*) (− *17*), *FloatR* (− *662041*) (− *21*)), *FloatR* *168791* (− *12*),
(*FloatR* (− *825253*) (− *17*), *FloatR* (− *733276*) (− *21*)), *FloatR* (− *734456*) (− *17*),
FloatR (− *662134*) (− *21*)),
(*FloatR* *167031* (− *12*), (*FloatR* (− *815328*) (− *17*), *FloatR* (− *757674*) (− *21*)),
(*FloatR* (− *716368*) (− *17*), *FloatR* (− *673205*) (− *21*)), *FloatR* *167911* (− *12*),
(*FloatR* (− *815328*) (− *17*), *FloatR* (− *745058*) (− *21*)), *FloatR* (− *725491*) (− *17*),
FloatR (− *673299*) (− *21*)),
(*FloatR* *166151* (− *12*), (*FloatR* (− *805240*) (− *17*), *FloatR* (− *771028*) (− *21*)),
(*FloatR* (− *707083*) (− *17*), *FloatR* (− *685101*) (− *21*)), *FloatR* *167031* (− *12*),

(*FloatR* (− *805240*) (− *17*), *FloatR* (− *757580*) (− *21*)), *FloatR* (− *716372*) (− *17*),
FloatR (− *685198*) (− *21*)),
(*FloatR* *165271* (− *12*), (*FloatR* (− *794978*) (− *17*), *FloatR* (− *785316*) (− *21*)),
(*FloatR* (− *697622*) (− *17*), *FloatR* (− *697820*) (− *21*)), *FloatR* *166151* (− *12*),
(*FloatR* (− *794978*) (− *17*), *FloatR* (− *770931*) (− *21*)), *FloatR* (− *707088*) (− *17*),
FloatR (− *697920*) (− *21*)),
(*FloatR* *164391* (− *12*), (*FloatR* (− *784531*) (− *17*), *FloatR* (− *800661*) (− *21*)),
(*FloatR* (− *687970*) (− *17*), *FloatR* (− *711467*) (− *21*)), *FloatR* *165271* (− *12*),
(*FloatR* (− *784531*) (− *17*), *FloatR* (− *785216*) (− *21*)), *FloatR* (− *697626*) (− *17*),
FloatR (− *711570*) (− *21*)),
(*FloatR* *163511* (− *12*), (*FloatR* (− *773884*) (− *17*), *FloatR* (− *817211*) (− *21*)),
(*FloatR* (− *678114*) (− *17*), *FloatR* (− *726165*) (− *21*)), *FloatR* *164391* (− *12*),
(*FloatR* (− *773884*) (− *17*), *FloatR* (− *800559*) (− *21*)), *FloatR* (− *687975*) (− *17*),
FloatR (− *726271*) (− *21*)),
(*FloatR* *162631* (− *12*), (*FloatR* (− *763024*) (− *17*), *FloatR* (− *835141*) (− *21*)),
(*FloatR* (− *668034*) (− *17*), *FloatR* (− *742062*) (− *21*)), *FloatR* *163511* (− *12*),
(*FloatR* (− *763024*) (− *17*), *FloatR* (− *817105*) (− *21*)), *FloatR* (− *678118*) (− *17*),
FloatR (− *742171*) (− *21*)),
(*FloatR* *161751* (− *12*), (*FloatR* (− *751933*) (− *17*), *FloatR* (− *854663*) (− *21*)),
(*FloatR* (− *657713*) (− *17*), *FloatR* (− *759332*) (− *21*)), *FloatR* *162631* (− *12*),
(*FloatR* (− *751933*) (− *17*), *FloatR* (− *835032*) (− *21*)), *FloatR* (− *668039*) (− *17*),
FloatR (− *759444*) (− *21*)),
(*FloatR* *160871* (− *12*), (*FloatR* (− *740591*) (− *17*), *FloatR* (− *876032*) (− *21*)),
(*FloatR* (− *647126*) (− *17*), *FloatR* (− *778186*) (− *21*)), *FloatR* *161751* (− *12*),
(*FloatR* (− *740591*) (− *17*), *FloatR* (− *854550*) (− *21*)), *FloatR* (− *657717*) (− *17*),
FloatR (− *778302*) (− *21*)),
(*FloatR* *159991* (− *12*), (*FloatR* (− *728974*) (− *17*), *FloatR* (− *899562*) (− *21*)),
(*FloatR* (− *636247*) (− *17*), *FloatR* (− *798881*) (− *21*)), *FloatR* *160871* (− *12*),

(*FloatR* (− *728974*) (− *17*), *FloatR* (− *875915*) (− *21*)), *FloatR* (− *647131*) (− *17*),
FloatR (− *799001*) (− *21*)),
(*FloatR* *159111* (− *12*), (*FloatR* (− *717056*) (− *17*), *FloatR* (− *925642*) (− *21*)),
(*FloatR* (− *625045*) (− *17*), *FloatR* (− *821726*) (− *21*)), *FloatR* *159991* (− *12*),
(*FloatR* (− *717056*) (− *17*), *FloatR* (− *899441*) (− *21*)), *FloatR* (− *636252*) (− *17*),
FloatR (− *821851*) (− *21*)),
(*FloatR* *158231* (− *12*), (*FloatR* (− *704806*) (− *17*), *FloatR* (− *954755*) (− *21*)),
(*FloatR* (− *613482*) (− *17*), *FloatR* (− *847108*) (− *21*)), *FloatR* *159111* (− *12*),
(*FloatR* (− *704806*) (− *17*), *FloatR* (− *925515*) (− *21*)), *FloatR* (− *625050*) (− *17*),
FloatR (− *847237*) (− *21*)),
(*FloatR* *157351* (− *12*), (*FloatR* (− *692186*) (− *17*), *FloatR* (− *987512*) (− *21*)),
(*FloatR* (− *601515*) (− *17*), *FloatR* (− *875505*) (− *21*)), *FloatR* *158231* (− *12*),
(*FloatR* (− *692186*) (− *17*), *FloatR* (− *954623*) (− *21*)), *FloatR* (− *613487*) (− *17*),
FloatR (− *875640*) (− *21*)),
(*FloatR* *156471* (− *12*), (*FloatR* (− *679151*) (− *17*), *FloatR* (− *1024697*) (− *21*)),
(*FloatR* (− *589089*) (− *17*), *FloatR* (− *907522*) (− *21*)), *FloatR* *157351* (− *12*),
(*FloatR* (− *679151*) (− *17*), *FloatR* (− *987375*) (− *21*)), *FloatR* (− *601520*) (− *17*),
FloatR (− *907663*) (− *21*)),
(*FloatR* *155591* (− *12*), (*FloatR* (− *665648*) (− *17*), *FloatR* (− *533662*) (− *20*)),
(*FloatR* (− *576139*) (− *17*), *FloatR* (− *943929*) (− *21*)), *FloatR* *156471* (− *12*),
(*FloatR* (− *665648*) (− *17*), *FloatR* (− *1024553*) (− *21*)), *FloatR* (− *589094*) (− *17*),
FloatR (− *944077*) (− *21*)),
(*FloatR* *154711* (− *12*), (*FloatR* (− *651610*) (− *17*), *FloatR* (− *558366*) (− *20*)),
(*FloatR* (− *562584*) (− *17*), *FloatR* (− *985725*) (− *21*)), *FloatR* *155591* (− *12*),
(*FloatR* (− *651610*) (− *17*), *FloatR* (− *533586*) (− *20*)), *FloatR* (− *576144*) (− *17*),
FloatR (− *985881*) (− *21*)),
(*FloatR* *153831* (− *12*), (*FloatR* (− *636956*) (− *17*), *FloatR* (− *587361*) (− *20*)),
(*FloatR* (− *548324*) (− *17*), *FloatR* (− *1034224*) (− *21*)), *FloatR* *154711* (− *12*),

(*FloatR* (− *636956*) (− *17*), *FloatR* (− *558286*) (− *20*)), *FloatR* (− *562589*) (− *17*),
FloatR (− *1034389*) (− *21*)),
(*FloatR* *152951* (− *12*), (*FloatR* (− *621583*) (− *17*), *FloatR* (− *621886*) (− *20*)),
(*FloatR* (− *533229*) (− *17*), *FloatR* (− *545594*) (− *20*)), *FloatR* *153831* (− *12*),
(*FloatR* (− *621583*) (− *17*), *FloatR* (− *587276*) (− *20*)), *FloatR* (− *548329*) (− *17*),
FloatR (− *545681*) (− *20*)),
(*FloatR* *152071* (− *12*), (*FloatR* (− *605361*) (− *17*), *FloatR* (− *663687*) (− *20*)),
(*FloatR* (− *1034263*) (− *18*), *FloatR* (− *579517*) (− *20*)), *FloatR* *152951* (− *12*),
(*FloatR* (− *605361*) (− *17*), *FloatR* (− *621794*) (− *20*)), *FloatR* (− *533234*) (− *17*),
FloatR (− *579611*) (− *20*)),
(*FloatR* *151191* (− *12*), (*FloatR* (− *588119*) (− *17*), *FloatR* (− *715306*) (− *20*)),
(*FloatR* (− *999617*) (− *18*), *FloatR* (− *620582*) (− *20*)), *FloatR* *152071* (− *12*),
(*FloatR* (− *588119*) (− *17*), *FloatR* (− *663587*) (− *20*)), *FloatR* (− *1034274*) (− *18*),
FloatR (− *620685*) (− *20*)),
(*FloatR* *150311* (− *12*), (*FloatR* (− *569629*) (− *17*), *FloatR* (− *780586*) (− *20*)),
(*FloatR* (− *961911*) (− *18*), *FloatR* (− *671256*) (− *20*)), *FloatR* *151191* (− *12*),
(*FloatR* (− *569629*) (− *17*), *FloatR* (− *715197*) (− *20*)), *FloatR* (− *999628*) (− *18*),
FloatR (− *671370*) (− *20*)),
(*FloatR* *149431* (− *12*), (*FloatR* (− *549580*) (− *17*), *FloatR* (− *865610*) (− *20*)),
(*FloatR* (− *920285*) (− *18*), *FloatR* (− *735257*) (− *20*)), *FloatR* *150311* (− *12*),
(*FloatR* (− *549580*) (− *17*), *FloatR* (− *780464*) (− *20*)), *FloatR* (− *961922*) (− *18*),
FloatR (− *735385*) (− *20*)),
(*FloatR* *148551* (− *12*), (*FloatR* (− *527531*) (− *17*), *FloatR* (− *980602*) (− *20*)),
(*FloatR* (− *873468*) (− *18*), *FloatR* (− *818447*) (− *20*)), *FloatR* *149431* (− *12*),
(*FloatR* (− *527530*) (− *17*), *FloatR* (− *865471*) (− *20*)), *FloatR* (− *920297*) (− *18*),
FloatR (− *818595*) (− *20*)),
(*FloatR* *147671* (− *12*), (*FloatR* (− *1005650*) (− *18*), *FloatR* (− *572069*) (− *19*)),
(*FloatR* (− *819476*) (− *18*), *FloatR* (− *930637*) (− *20*)), *FloatR* *148551* (− *12*),

(*FloatR* (− *1005649*) (− *18*), *FloatR* (− *980439*) (− *20*)), *FloatR* (− *873480*) (− *18*),
FloatR (− *930815*) (− *20*)),
(*FloatR* *146791* (− *12*), (*FloatR* (− *948872*) (− *18*), *FloatR* (− *696902*) (− *19*)),
(*FloatR* (− *754961*) (− *18*), *FloatR* (− *544773*) (− *19*)), *FloatR* *147671* (− *12*),
(*FloatR* (− *948871*) (− *18*), *FloatR* (− *571968*) (− *19*)), *FloatR* (− *819489*) (− *18*),
FloatR (− *544887*) (− *19*)),
(*FloatR* *145911* (− *12*), (*FloatR* (− *881238*) (− *18*), *FloatR* (− *909366*) (− *19*)),
(*FloatR* (− *673607*) (− *18*), *FloatR* (− *665379*) (− *19*)), *FloatR* *146791* (− *12*),
(*FloatR* (− *881238*) (− *18*), *FloatR* (− *696766*) (− *19*)), *FloatR* (− *754977*) (− *18*),
FloatR (− *665541*) (− *19*)),
(*FloatR* *145031* (− *12*), (*FloatR* (− *796106*) (− *18*), *FloatR* (− *676194*) (− *18*)),
(*FloatR* (− *561314*) (− *18*), *FloatR* (− *868852*) (− *19*)), *FloatR* *145911* (− *12*),
(*FloatR* (− *796105*) (− *18*), *FloatR* (− *909153*) (− *19*)), *FloatR* (− *673628*) (− *18*),
FloatR (− *869126*) (− *19*)),
(*FloatR* *144151* (− *12*), (*FloatR* (− *678028*) (− *18*), *FloatR* (− *697052*) (− *17*)),
(*FloatR* (− *747200*) (− *19*), *FloatR* (− *637694*) (− *18*)), *FloatR* *145031* (− *12*),
(*FloatR* (− *678025*) (− *18*), *FloatR* (− *675934*) (− *18*)), *FloatR* (− *561354*) (− *18*),
FloatR (− *638108*) (− *18*)),
(*FloatR* *35859* (− *10*), (*FloatR* (− *949024*) (− *19*), *FloatR* (− *782952*) (− *15*)),
(*FloatR* *809103* (− *23*), *FloatR* (− *621866*) (− *17*)), *FloatR* *144151* (− *12*),
(*FloatR* (− *949003*) (− *19*), *FloatR* (− *696167*) (− *17*)), *FloatR* (− *747511*) (− *19*),
FloatR (− *623437*) (− *17*)),
(*FloatR* *17820* (− *9*), (*FloatR* *794796* (− *23*), *FloatR* (− *863739*) (− *17*)),
(*FloatR* *704638* (− *20*), *FloatR* (− *923149*) (− *19*)), *FloatR* *35859* (− *10*),
(*FloatR* *794796* (− *23*), *FloatR* (− *863739*) (− *17*)), *FloatR* *808606* (− *23*),
FloatR (− *847811*) (− *17*)),
(*FloatR* *17710* (− *9*), (*FloatR* *704583* (− *20*), *FloatR* (− *923233*) (− *19*)),
(*FloatR* *1040854* (− *20*), *FloatR* (− *680113*) (− *19*)), *FloatR* *17820* (− *9*),
(*FloatR* *704583* (− *20*), *FloatR* (− *923232*) (− *19*)), *FloatR* *704638* (− *20*),
FloatR (− *923150*) (− *19*)),
(*FloatR* *17600* (− *9*), (*FloatR* *1040826* (− *20*), *FloatR* (− *680135*) (− *19*)),

(*FloatR 653079* (− *19*), *FloatR* (− *565606*) (− *19*)), *FloatR 17710* (− *9*),
(*FloatR 1040826* (− *20*), *FloatR* (− *680135*) (− *19*)), *FloatR 1040854* (− *20*),
FloatR (− *680113*) (− *19*)),
(*FloatR 17490* (− *9*), (*FloatR 653069* (− *19*), *FloatR* (− *565616*) (− *19*)),
(*FloatR 766522* (− *19*), *FloatR* (− *990470*) (− *20*)), *FloatR 17600* (− *9*),
(*FloatR 653069* (− *19*), *FloatR* (− *565616*) (− *19*)), *FloatR 653079* (− *19*),
FloatR (− *565606*) (− *19*)),
(*FloatR 17380* (− *9*), (*FloatR 766515* (− *19*), *FloatR* (− *990482*) (− *20*)),
(*FloatR 867382* (− *19*), *FloatR* (− *892557*) (− *20*)), *FloatR 17490* (− *9*),
(*FloatR 766515* (− *19*), *FloatR* (− *990482*) (− *20*)), *FloatR 766522* (− *19*),
FloatR (− *990470*) (− *20*)),
(*FloatR 17270* (− *9*), (*FloatR 867376* (− *19*), *FloatR* (− *892565*) (− *20*)),
(*FloatR 959155* (− *19*), *FloatR* (− *819290*) (− *20*)), *FloatR 17380* (− *9*),
(*FloatR 867376* (− *19*), *FloatR* (− *892565*) (− *20*)), *FloatR 867382* (− *19*),
FloatR (− *892557*) (− *20*)),
(*FloatR 17160* (− *9*), (*FloatR 959150* (− *19*), *FloatR* (− *819295*) (− *20*)),
(*FloatR 1043960* (− *19*), *FloatR* (− *761757*) (− *20*)), *FloatR 17270* (− *9*),
(*FloatR 959150* (− *19*), *FloatR* (− *819295*) (− *20*)), *FloatR 959155* (− *19*),
FloatR (− *819290*) (− *20*)),
(*FloatR 17050* (− *9*), (*FloatR 1043956* (− *19*), *FloatR* (− *761761*) (− *20*)),
(*FloatR 561599* (− *18*), *FloatR* (− *715002*) (− *20*)), *FloatR 17160* (− *9*),
(*FloatR 1043956* (− *19*), *FloatR* (− *761761*) (− *20*)), *FloatR 1043960* (− *19*),
FloatR (− *761757*) (− *20*)),
(*FloatR 16940* (− *9*), (*FloatR 561596* (− *18*), *FloatR* (− *715005*) (− *20*)),
(*FloatR 598926* (− *18*), *FloatR* (− *676015*) (− *20*)), *FloatR 17050* (− *9*),
(*FloatR 561596* (− *18*), *FloatR* (− *715005*) (− *20*)), *FloatR 561599* (− *18*),
FloatR (− *715002*) (− *20*)),
(*FloatR 16830* (− *9*), (*FloatR 598923* (− *18*), *FloatR* (− *676018*) (− *20*)),
(*FloatR 634322* (− *18*), *FloatR* (− *642850*) (− *20*)), *FloatR 16940* (− *9*),
(*FloatR 598923* (− *18*), *FloatR* (− *676018*) (− *20*)), *FloatR 598926* (− *18*),
FloatR (− *676015*) (− *20*)),
(*FloatR 16720* (− *9*), (*FloatR 634319* (− *18*), *FloatR* (− *642853*) (− *20*)),
(*FloatR 668062* (− *18*), *FloatR* (− *614182*) (− *20*)), *FloatR 16830* (− *9*),
(*FloatR 634319* (− *18*), *FloatR* (− *642853*) (− *20*)), *FloatR 634322* (− *18*),
FloatR (− *642850*) (− *20*)),
(*FloatR 16610* (− *9*), (*FloatR 668060* (− *18*), *FloatR* (− *614184*) (− *20*)),
(*FloatR 700362* (− *18*), *FloatR* (− *589074*) (− *20*)), *FloatR 16720* (− *9*),
(*FloatR 668060* (− *18*), *FloatR* (− *614184*) (− *20*)), *FloatR 668062* (− *18*),
FloatR (− *614182*) (− *20*)),

(*FloatR 16500* (− *9*), (*FloatR 700360* (− *18*), *FloatR* (− *589076*) (− *20*)),
(*FloatR 731393* (− *18*), *FloatR* (− *566842*) (− *20*)), *FloatR 16610* (− *9*),
(*FloatR 700360* (− *18*), *FloatR* (− *589076*) (− *20*)), *FloatR 700362* (−
18),
FloatR (− *589074*) (− *20*)),
(*FloatR 16390* (− *9*), (*FloatR 731391* (− *18*), *FloatR* (− *566844*) (− *20*)),
(*FloatR 761295* (− *18*), *FloatR* (− *546974*) (− *20*)), *FloatR 16500* (− *9*),
(*FloatR 731391* (− *18*), *FloatR* (− *566844*) (− *20*)), *FloatR 731393* (−
18),
FloatR (− *566842*) (− *20*)),
(*FloatR 16280* (− *9*), (*FloatR 761293* (− *18*), *FloatR* (− *546976*) (− *20*)),
(*FloatR 790185* (− *18*), *FloatR* (− *529077*) (− *20*)), *FloatR 16390* (− *9*),
(*FloatR 761293* (− *18*), *FloatR* (− *546976*) (− *20*)), *FloatR 761295* (−
18),
FloatR (− *546974*) (− *20*)),
(*FloatR 16170* (− *9*), (*FloatR 790183* (− *18*), *FloatR* (− *529078*) (− *20*)),
(*FloatR 818160* (− *18*), *FloatR* (− *1025688*) (− *21*)), *FloatR 16280* (− *9*),
(*FloatR 790183* (− *18*), *FloatR* (− *529078*) (− *20*)), *FloatR 790185* (−
18),
FloatR (− *529077*) (− *20*)),
(*FloatR 16060* (− *9*), (*FloatR 818158* (− *18*), *FloatR* (− *1025690*) (− *21*)),
(*FloatR 845301* (− *18*), *FloatR* (− *996061*) (− *21*)), *FloatR 16170* (− *9*),
(*FloatR 818158* (− *18*), *FloatR* (− *1025690*) (− *21*)), *FloatR 818160* (−
18),
FloatR (− *1025688*) (− *21*)),
(*FloatR 15950* (− *9*), (*FloatR 845299* (− *18*), *FloatR* (− *996064*) (− *21*)),
(*FloatR 871681* (− *18*), *FloatR* (− *968883*) (− *21*)), *FloatR 16060* (− *9*),
(*FloatR 845299* (− *18*), *FloatR* (− *996064*) (− *21*)), *FloatR 845301* (−
18),
FloatR (− *996061*) (− *21*)),
(*FloatR 15840* (− *9*), (*FloatR 871679* (− *18*), *FloatR* (− *968885*) (− *21*)),
(*FloatR 897360* (− *18*), *FloatR* (− *943831*) (− *21*)), *FloatR 15950* (− *9*),
(*FloatR 871679* (− *18*), *FloatR* (− *968885*) (− *21*)), *FloatR 871681* (−
18),
FloatR (− *968883*) (− *21*)),
(*FloatR 15730* (− *9*), (*FloatR 897358* (− *18*), *FloatR* (− *943833*) (− *21*)),
(*FloatR 922392* (− *18*), *FloatR* (− *920641*) (− *21*)), *FloatR 15840* (− *9*),
(*FloatR 897358* (− *18*), *FloatR* (− *943833*) (− *21*)), *FloatR 897360* (−
18),
FloatR (− *943831*) (− *21*)),
(*FloatR 15620* (− *9*), (*FloatR 922390* (− *18*), *FloatR* (− *920643*) (− *21*)),
(*FloatR 946823* (− *18*), *FloatR* (− *899093*) (− *21*)), *FloatR 15730* (− *9*),
(*FloatR 922390* (− *18*), *FloatR* (− *920643*) (− *21*)), *FloatR 922392* (−
18),
FloatR (− *920641*) (− *21*)),
(*FloatR 15510* (− *9*), (*FloatR 946821* (− *18*), *FloatR* (− *899094*) (− *21*)),
(*FloatR 970696* (− *18*), *FloatR* (− *879000*) (− *21*)), *FloatR 15620* (− *9*),
(*FloatR 946821* (− *18*), *FloatR* (− *899094*) (− *21*)), *FloatR 946823* (−
18),

FloatR (− *899093*) (− *21*)),
(*FloatR 15400* (− *9*), (*FloatR 970694* (− *18*), *FloatR* (− *879002*) (− *21*)),
(*FloatR 994047* (− *18*), *FloatR* (− *860207*) (− *21*)), *FloatR 15510* (− *9*),
(*FloatR 970694* (− *18*), *FloatR* (− *879002*) (− *21*)), *FloatR 970696* (−
18),
FloatR (− *879000*) (− *21*)),
(*FloatR 15290* (− *9*), (*FloatR 994045* (− *18*), *FloatR* (− *860208*) (− *21*)),
(*FloatR 1016909* (− *18*), *FloatR* (− *842577*) (− *21*)), *FloatR 15400* (− *9*),
(*FloatR 994045* (− *18*), *FloatR* (− *860208*) (− *21*)), *FloatR 994047* (−
18),
FloatR (− *860207*) (− *21*)),
(*FloatR 15180* (− *9*), (*FloatR 1016907* (− *18*), *FloatR* (− *842578*) (− *21*)),
(*FloatR 1039312* (− *18*), *FloatR* (− *825995*) (− *21*)), *FloatR 15290* (− *9*),
(*FloatR 1016907* (− *18*), *FloatR* (− *842578*) (− *21*)), *FloatR 1016909* (−
18),
FloatR (− *842577*) (− *21*)),
(*FloatR 15070* (− *9*), (*FloatR 1039310* (− *18*), *FloatR* (− *825997*) (− *21*)),
(*FloatR 530641* (− *17*), *FloatR* (− *810362*) (− *21*)), *FloatR 15180* (− *9*),
(*FloatR 1039310* (− *18*), *FloatR* (− *825997*) (− *21*)), *FloatR 1039312* (−
18),
FloatR (− *825995*) (− *21*)),
(*FloatR 14960* (− *9*), (*FloatR 530640* (− *17*), *FloatR* (− *810363*) (− *21*)),
(*FloatR 541423* (− *17*), *FloatR* (− *795589*) (− *21*)), *FloatR 15070* (− *9*),
(*FloatR 530640* (− *17*), *FloatR* (− *810363*) (− *21*)), *FloatR 530641* (−
17),
FloatR (− *810362*) (− *21*)),
(*FloatR 14850* (− *9*), (*FloatR 541421* (− *17*), *FloatR* (− *795590*) (− *21*)),
(*FloatR 552011* (− *17*), *FloatR* (− *781600*) (− *21*)), *FloatR 14960* (− *9*),
(*FloatR 541421* (− *17*), *FloatR* (− *795590*) (− *21*)), *FloatR 541423* (−
17),
FloatR (− *795589*) (− *21*)),
(*FloatR 14740* (− *9*), (*FloatR 552010* (− *17*), *FloatR* (− *781602*) (− *21*)),
(*FloatR 562416* (− *17*), *FloatR* (− *768329*) (− *21*)), *FloatR 14850* (− *9*),
(*FloatR 552010* (− *17*), *FloatR* (− *781602*) (− *21*)), *FloatR 552011* (−
17),
FloatR (− *781600*) (− *21*)),
(*FloatR 14630* (− *9*), (*FloatR 562415* (− *17*), *FloatR* (− *768330*) (− *21*)),
(*FloatR 572648* (− *17*), *FloatR* (− *755715*) (− *21*)), *FloatR 14740* (− *9*),
(*FloatR 562415* (− *17*), *FloatR* (− *768330*) (− *21*)), *FloatR 562416* (−
17),
FloatR (− *768329*) (− *21*)),
(*FloatR 14520* (− *9*), (*FloatR 572646* (− *17*), *FloatR* (− *755716*) (− *21*)),
(*FloatR 582714* (− *17*), *FloatR* (− *743706*) (− *21*)), *FloatR 14630* (− *9*),
(*FloatR 572646* (− *17*), *FloatR* (− *755716*) (− *21*)), *FloatR 572648* (−
17),
FloatR (− *755715*) (− *21*)),
(*FloatR 14410* (− *9*), (*FloatR 582713* (− *17*), *FloatR* (− *743707*) (− *21*)),
(*FloatR 592623* (− *17*), *FloatR* (− *732255*) (− *21*)), *FloatR 14520* (− *9*),
(*FloatR 582713* (− *17*), *FloatR* (− *743707*) (− *21*)), *FloatR 582714* (−

17),
FloatR (− *743706*) (− *21*)),
(*FloatR 14300* (− *9*), (*FloatR 592621* (− *17*), *FloatR* (− *732256*) (− *21*)),
(*FloatR 602381* (− *17*), *FloatR* (− *721319*) (− *21*)), *FloatR 14410* (− *9*),
(*FloatR 592621* (− *17*), *FloatR* (− *732256*) (− *21*)), *FloatR 592623* (−
17),
FloatR (− *732255*) (− *21*)),
(*FloatR 14190* (− *9*), (*FloatR 602380* (− *17*), *FloatR* (− *721321*) (− *21*)),
(*FloatR 611996* (− *17*), *FloatR* (− *710862*) (− *21*)), *FloatR 14300* (− *9*),
(*FloatR 602380* (− *17*), *FloatR* (− *721321*) (− *21*)), *FloatR 602381* (−
17),
FloatR (− *721319*) (− *21*)),
(*FloatR 14080* (− *9*), (*FloatR 611995* (− *17*), *FloatR* (− *710864*) (− *21*)),
(*FloatR 621474* (− *17*), *FloatR* (− *700849*) (− *21*)), *FloatR 14190* (− *9*),
(*FloatR 611995* (− *17*), *FloatR* (− *710864*) (− *21*)), *FloatR 611996* (−
17),
FloatR (− *710862*) (− *21*)),
(*FloatR 13970* (− *9*), (*FloatR 621473* (− *17*), *FloatR* (− *700851*) (− *21*)),
(*FloatR 630820* (− *17*), *FloatR* (− *691250*) (− *21*)), *FloatR 14080* (− *9*),
(*FloatR 621473* (− *17*), *FloatR* (− *700851*) (− *21*)), *FloatR 621474* (−
17),
FloatR (− *700849*) (− *21*)),
(*FloatR 13860* (− *9*), (*FloatR 630818* (− *17*), *FloatR* (− *691251*) (− *21*)),
(*FloatR 640039* (− *17*), *FloatR* (− *682037*) (− *21*)), *FloatR 13970* (− *9*),
(*FloatR 630818* (− *17*), *FloatR* (− *691251*) (− *21*)), *FloatR 630820* (−
17),
FloatR (− *691250*) (− *21*)),
(*FloatR 13750* (− *9*), (*FloatR 640038* (− *17*), *FloatR* (− *682038*) (− *21*)),
(*FloatR 649138* (− *17*), *FloatR* (− *673184*) (− *21*)), *FloatR 13860* (− *9*),
(*FloatR 640038* (− *17*), *FloatR* (− *682038*) (− *21*)), *FloatR 640039* (−
17),
FloatR (− *682037*) (− *21*)),
(*FloatR 13640* (− *9*), (*FloatR 649137* (− *17*), *FloatR* (− *673185*) (− *21*)),
(*FloatR 658119* (− *17*), *FloatR* (− *664668*) (− *21*)), *FloatR 13750* (− *9*),
(*FloatR 649137* (− *17*), *FloatR* (− *673185*) (− *21*)), *FloatR 649138* (−
17),
FloatR (− *673184*) (− *21*)),
(*FloatR 13530* (− *9*), (*FloatR 658118* (− *17*), *FloatR* (− *664669*) (− *21*)),
(*FloatR 666989* (− *17*), *FloatR* (− *656469*) (− *21*)), *FloatR 13640* (− *9*),
(*FloatR 658118* (− *17*), *FloatR* (− *664669*) (− *21*)), *FloatR 658119* (−
17),
FloatR (− *664668*) (− *21*)),
(*FloatR 13420* (− *9*), (*FloatR 666988* (− *17*), *FloatR* (− *656470*) (− *21*)),
(*FloatR 675751* (− *17*), *FloatR* (− *648567*) (− *21*)), *FloatR 13530* (− *9*),
(*FloatR 666988* (− *17*), *FloatR* (− *656470*) (− *21*)), *FloatR 666989* (−
17),
FloatR (− *656469*) (− *21*)),
(*FloatR 13310* (− *9*), (*FloatR 675749* (− *17*), *FloatR* (− *648569*) (− *21*)),
(*FloatR 684408* (− *17*), *FloatR* (− *640946*) (− *21*)), *FloatR 13420* (− *9*),

(*FloatR 675749* (− *17*), *FloatR* (− *648569*) (− *21*)), *FloatR 675751* (−
17),
FloatR (− *648567*) (− *21*)),
(*FloatR 13200* (− *9*), (*FloatR 684407* (− *17*), *FloatR* (− *640947*) (− *21*)),
(*FloatR 692965* (− *17*), *FloatR* (− *633587*) (− *21*)), *FloatR 13310* (− *9*),
(*FloatR 684407* (− *17*), *FloatR* (− *640947*) (− *21*)), *FloatR 684408* (−
17),
FloatR (− *640946*) (− *21*)),
(*FloatR 13090* (− *9*), (*FloatR 692964* (− *17*), *FloatR* (− *633588*) (− *21*)),
(*FloatR 701424* (− *17*), *FloatR* (− *626477*) (− *21*)), *FloatR 13200* (− *9*),
(*FloatR 692964* (− *17*), *FloatR* (− *633588*) (− *21*)), *FloatR 692965* (−
17),
FloatR (− *633587*) (− *21*)),
(*FloatR 12980* (− *9*), (*FloatR 701423* (− *17*), *FloatR* (− *626479*) (− *21*)),
(*FloatR 709790* (− *17*), *FloatR* (− *619603*) (− *21*)), *FloatR 13090* (− *9*),
(*FloatR 701423* (− *17*), *FloatR* (− *626479*) (− *21*)), *FloatR 701424* (−
17),
FloatR (− *626477*) (− *21*)),
(*FloatR 12870* (− *9*), (*FloatR 709789* (− *17*), *FloatR* (− *619604*) (− *21*)),
(*FloatR 718065* (− *17*), *FloatR* (− *612950*) (− *21*)), *FloatR 12980* (− *9*),
(*FloatR 709789* (− *17*), *FloatR* (− *619604*) (− *21*)), *FloatR 709790* (−
17),
FloatR (− *619603*) (− *21*)),
(*FloatR 12760* (− *9*), (*FloatR 718064* (− *17*), *FloatR* (− *612951*) (− *21*)),
(*FloatR 726252* (− *17*), *FloatR* (− *606508*) (− *21*)), *FloatR 12870* (− *9*),
(*FloatR 718064* (− *17*), *FloatR* (− *612951*) (− *21*)), *FloatR 718065* (−
17),
FloatR (− *612950*) (− *21*)),
(*FloatR 12650* (− *9*), (*FloatR 726251* (− *17*), *FloatR* (− *606509*) (− *21*)),
(*FloatR 734354* (− *17*), *FloatR* (− *600265*) (− *21*)), *FloatR 12760* (− *9*),
(*FloatR 726251* (− *17*), *FloatR* (− *606509*) (− *21*)), *FloatR 726252* (−
17),
FloatR (− *606508*) (− *21*)),
(*FloatR 12540* (− *9*), (*FloatR 734353* (− *17*), *FloatR* (− *600266*) (− *21*)),
(*FloatR 742373* (− *17*), *FloatR* (− *594211*) (− *21*)), *FloatR 12650* (− *9*),
(*FloatR 734353* (− *17*), *FloatR* (− *600266*) (− *21*)), *FloatR 734354* (−
17),
FloatR (− *600265*) (− *21*)),
(*FloatR 12430* (− *9*), (*FloatR 742372* (− *17*), *FloatR* (− *594212*) (− *21*)),
(*FloatR 750313* (− *17*), *FloatR* (− *588337*) (− *21*)), *FloatR 12540* (− *9*),
(*FloatR 742372* (− *17*), *FloatR* (− *594212*) (− *21*)), *FloatR 742373* (−
17),
FloatR (− *594211*) (− *21*)),
(*FloatR 12320* (− *9*), (*FloatR 750312* (− *17*), *FloatR* (− *588339*) (− *21*)),
(*FloatR 758174* (− *17*), *FloatR* (− *582635*) (− *21*)), *FloatR 12430* (− *9*),
(*FloatR 750312* (− *17*), *FloatR* (− *588339*) (− *21*)), *FloatR 750313* (−
17),
FloatR (− *588337*) (− *21*)),
(*FloatR 12210* (− *9*), (*FloatR 758173* (− *17*), *FloatR* (− *582636*) (− *21*)),

(*FloatR 765960* (− *17*), *FloatR* (− *577095*) (− *21*)), *FloatR 12320* (− *9*),
(*FloatR 758173* (− *17*), *FloatR* (− *582636*) (− *21*)), *FloatR 758174* (−
17),
FloatR (− *582635*) (− *21*)),
(*FloatR 12100* (− *9*), (*FloatR 765959* (− *17*), *FloatR* (− *577096*) (− *21*)),
(*FloatR 773673* (− *17*), *FloatR* (− *571710*) (− *21*)), *FloatR 12210* (− *9*),
(*FloatR 765959* (− *17*), *FloatR* (− *577096*) (− *21*)), *FloatR 765960* (−
17),
FloatR (− *577095*) (− *21*)),
(*FloatR 11990* (− *9*), (*FloatR 773672* (− *17*), *FloatR* (− *571711*) (− *21*)),
(*FloatR 781314* (− *17*), *FloatR* (− *566473*) (− *21*)), *FloatR 12100* (− *9*),
(*FloatR 773672* (− *17*), *FloatR* (− *571711*) (− *21*)), *FloatR 773673* (−
17),
FloatR (− *571710*) (− *21*)),
(*FloatR 11880* (− *9*), (*FloatR 781313* (− *17*), *FloatR* (− *566474*) (− *21*)),
(*FloatR 788887* (− *17*), *FloatR* (− *561378*) (− *21*)), *FloatR 11990* (− *9*),
(*FloatR 781313* (− *17*), *FloatR* (− *566474*) (− *21*)), *FloatR 781314* (−
17),
FloatR (− *566473*) (− *21*)),
(*FloatR 11770* (− *9*), (*FloatR 788886* (− *17*), *FloatR* (− *561379*) (− *21*)),
(*FloatR 796391* (− *17*), *FloatR* (− *556417*) (− *21*)), *FloatR 11880* (− *9*),
(*FloatR 788886* (− *17*), *FloatR* (− *561379*) (− *21*)), *FloatR 788887* (−
17),
FloatR (− *561378*) (− *21*)),
(*FloatR 11660* (− *9*), (*FloatR 796390* (− *17*), *FloatR* (− *556418*) (− *21*)),
(*FloatR 803830* (− *17*), *FloatR* (− *551585*) (− *21*)), *FloatR 11770* (− *9*),
(*FloatR 796390* (− *17*), *FloatR* (− *556418*) (− *21*)), *FloatR 796391* (−
17),
FloatR (− *556417*) (− *21*)),
(*FloatR 11550* (− *9*), (*FloatR 803829* (− *17*), *FloatR* (− *551586*) (− *21*)),
(*FloatR 811205* (− *17*), *FloatR* (− *546877*) (− *21*)), *FloatR 11660* (− *9*),
(*FloatR 803829* (− *17*), *FloatR* (− *551586*) (− *21*)), *FloatR 803830* (−
17),
FloatR (− *551585*) (− *21*)),
(*FloatR 11440* (− *9*), (*FloatR 811204* (− *17*), *FloatR* (− *546878*) (− *21*)),
(*FloatR 818517* (− *17*), *FloatR* (− *542286*) (− *21*)), *FloatR 11550* (− *9*),
(*FloatR 811204* (− *17*), *FloatR* (− *546878*) (− *21*)), *FloatR 811205* (−
17),
FloatR (− *546877*) (− *21*)),
(*FloatR 11330* (− *9*), (*FloatR 818516* (− *17*), *FloatR* (− *542287*) (− *21*)),
(*FloatR 825769* (− *17*), *FloatR* (− *537808*) (− *21*)), *FloatR 11440* (− *9*),
(*FloatR 818516* (− *17*), *FloatR* (− *542287*) (− *21*)), *FloatR 818517* (−
17),
FloatR (− *542286*) (− *21*)),
(*FloatR 11220* (− *9*), (*FloatR 825768* (− *17*), *FloatR* (− *537809*) (− *21*)),
(*FloatR 832961* (− *17*), *FloatR* (− *533439*) (− *21*)), *FloatR 11330* (− *9*),
(*FloatR 825768* (− *17*), *FloatR* (− *537809*) (− *21*)), *FloatR 825769* (−
17),
FloatR (− *537808*) (− *21*)),

(*FloatR 11110* (− *9*), (*FloatR 832960* (− *17*), *FloatR* (− *533440*) (− *21*)),
(*FloatR 840095* (− *17*), *FloatR* (− *529174*) (− *21*)), *FloatR 11220* (− *9*),
(*FloatR 832960* (− *17*), *FloatR* (− *533440*) (− *21*)), *FloatR 832961* (−
17),
FloatR (− *533439*) (− *21*)),
(*FloatR 11000* (− *9*), (*FloatR 840094* (− *17*), *FloatR* (− *529175*) (− *21*)),
(*FloatR 847172* (− *17*), *FloatR* (− *525008*) (− *21*)), *FloatR 11110* (− *9*),
(*FloatR 840094* (− *17*), *FloatR* (− *529175*) (− *21*)), *FloatR 840095* (−
17),
FloatR (− *529174*) (− *21*)),
(*FloatR 10890* (− *9*), (*FloatR 847171* (− *17*), *FloatR* (− *525009*) (− *21*)),
(*FloatR 854195* (− *17*), *FloatR* (− *1041875*) (− *22*)), *FloatR 11000* (− *9*),
(*FloatR 847171* (− *17*), *FloatR* (− *525009*) (− *21*)), *FloatR 847172* (−
17),
FloatR (− *525008*) (− *21*)),
(*FloatR 10780* (− *9*), (*FloatR 854194* (− *17*), *FloatR* (− *1041876*) (− *22*)),
(*FloatR 861163* (− *17*), *FloatR* (− *1033918*) (− *22*)), *FloatR 10890* (− *9*),
(*FloatR 854194* (− *17*), *FloatR* (− *1041876*) (− *22*)), *FloatR 854195* (−
17),
FloatR (− *1041875*) (− *22*)),
(*FloatR 10670* (− *9*), (*FloatR 861162* (− *17*), *FloatR* (− *1033919*) (− *22*)),
(*FloatR 868078* (− *17*), *FloatR* (− *1026137*) (− *22*)), *FloatR 10780* (− *9*),
(*FloatR 861162* (− *17*), *FloatR* (− *1033919*) (− *22*)), *FloatR 861163* (−
17),
FloatR (− *1033918*) (− *22*)),
(*FloatR 10560* (− *9*), (*FloatR 868077* (− *17*), *FloatR* (− *1026138*) (− *22*)),
(*FloatR 874942* (− *17*), *FloatR* (− *1018525*) (− *22*)), *FloatR 10670* (− *9*),
(*FloatR 868077* (− *17*), *FloatR* (− *1026138*) (− *22*)), *FloatR 868078* (−
17),
FloatR (− *1026137*) (− *22*)),
(*FloatR 10450* (− *9*), (*FloatR 874941* (− *17*), *FloatR* (− *1018526*) (− *22*)),
(*FloatR 881755* (− *17*), *FloatR* (− *1011076*) (− *22*)), *FloatR 10560* (− *9*),
(*FloatR 874941* (− *17*), *FloatR* (− *1018526*) (− *22*)), *FloatR 874942* (−
17),
FloatR (− *1018525*) (− *22*)),
(*FloatR 10340* (− *9*), (*FloatR 881754* (− *17*), *FloatR* (− *1011077*) (− *22*)),
(*FloatR 888519* (− *17*), *FloatR* (− *1003783*) (− *22*)), *FloatR 10450* (− *9*),
(*FloatR 881754* (− *17*), *FloatR* (− *1011077*) (− *22*)), *FloatR 881755* (−
17),
FloatR (− *1011076*) (− *22*)),
(*FloatR 10230* (− *9*), (*FloatR 888518* (− *17*), *FloatR* (− *1003784*) (− *22*)),
(*FloatR 895234* (− *17*), *FloatR* (− *996641*) (− *22*)), *FloatR 10340* (− *9*),
(*FloatR 888518* (− *17*), *FloatR* (− *1003784*) (− *22*)), *FloatR 888519* (−
17),
FloatR (− *1003783*) (− *22*)),
(*FloatR 10120* (− *9*), (*FloatR 895233* (− *17*), *FloatR* (− *996642*) (− *22*)),
(*FloatR 901902* (− *17*), *FloatR* (− *989643*) (− *22*)), *FloatR 10230* (− *9*),
(*FloatR 895233* (− *17*), *FloatR* (− *996642*) (− *22*)), *FloatR 895234* (−
17),

FloatR (− 996641) (− 22)),
(FloatR 10010 (− 9), (FloatR 901901 (− 17), FloatR (− 989644) (− 22)),
(FloatR 908523 (− 17), FloatR (− 982784) (− 22)), FloatR 10120 (− 9),
(FloatR 901901 (− 17), FloatR (− 989644) (− 22)), FloatR 901902 (−
17),
FloatR (− 989643) (− 22)),
(FloatR 9900 (− 9), (FloatR 908522 (− 17), FloatR (− 982785) (− 22)),
(FloatR 915099 (− 17), FloatR (− 976058) (− 22)), FloatR 10010 (− 9),
(FloatR 908522 (− 17), FloatR (− 982785) (− 22)), FloatR 908523 (−
17),
FloatR (− 982784) (− 22)),
(FloatR 9790 (− 9), (FloatR 915098 (− 17), FloatR (− 976059) (− 22)),
(FloatR 921629 (− 17), FloatR (− 969461) (− 22)), FloatR 9900 (− 9),
(FloatR 915098 (− 17), FloatR (− 976059) (− 22)), FloatR 915099 (−
17),
FloatR (− 976058) (− 22)),
(FloatR 9680 (− 9), (FloatR 921628 (− 17), FloatR (− 969462) (− 22)),
(FloatR 928117 (− 17), FloatR (− 962987) (− 22)), FloatR 9790 (− 9),
(FloatR 921628 (− 17), FloatR (− 969462) (− 22)), FloatR 921629 (−
17),
FloatR (− 969461) (− 22)),
(FloatR 9570 (− 9), (FloatR 928115 (− 17), FloatR (− 962988) (− 22)),
(FloatR 934560 (− 17), FloatR (− 956631) (− 22)), FloatR 9680 (− 9),
(FloatR 928115 (− 17), FloatR (− 962988) (− 22)), FloatR 928117 (−
17),
FloatR (− 962987) (− 22)),
(FloatR 9460 (− 9), (FloatR 934559 (− 17), FloatR (− 956632) (− 22)),
(FloatR 940962 (− 17), FloatR (− 950389) (− 22)), FloatR 9570 (− 9),
(FloatR 934559 (− 17), FloatR (− 956632) (− 22)), FloatR 934560 (−
17),
FloatR (− 956631) (− 22)),
(FloatR 9350 (− 9), (FloatR 940961 (− 17), FloatR (− 950391) (− 22)),
(FloatR 947322 (− 17), FloatR (− 944257) (− 22)), FloatR 9460 (− 9),
(FloatR 940961 (− 17), FloatR (− 950391) (− 22)), FloatR 940962 (−
17),
FloatR (− 950389) (− 22)),
(FloatR 9240 (− 9), (FloatR 947321 (− 17), FloatR (− 944258) (− 22)),
(FloatR 953642 (− 17), FloatR (− 938228) (− 22)), FloatR 9350 (− 9),
(FloatR 947321 (− 17), FloatR (− 944258) (− 22)), FloatR 947322 (−
17),
FloatR (− 944257) (− 22)),
(FloatR 9130 (− 9), (FloatR 953641 (− 17), FloatR (− 938229) (− 22)),
(FloatR 959921 (− 17), FloatR (− 932300) (− 22)), FloatR 9240 (− 9),
(FloatR 953641 (− 17), FloatR (− 938229) (− 22)), FloatR 953642 (−
17),
FloatR (− 938228) (− 22)),
(FloatR 9020 (− 9), (FloatR 959920 (− 17), FloatR (− 932301) (− 22)),
(FloatR 966161 (− 17), FloatR (− 926467) (− 22)), FloatR 9130 (− 9),
(FloatR 959920 (− 17), FloatR (− 932301) (− 22)), FloatR 959921 (−

17),
FloatR (− *932300*) (− *22*)),
(*FloatR 8910* (− *9*), (*FloatR 966159* (− *17*), *FloatR* (− *926468*) (− *22*)),
(*FloatR 972361* (− *17*), *FloatR* (− *920726*) (− *22*)), *FloatR 9020* (− *9*),
(*FloatR 966159* (− *17*), *FloatR* (− *926468*) (− *22*)), *FloatR 966161* (−
17),
FloatR (− *926467*) (− *22*)),
(*FloatR 8800* (− *9*), (*FloatR 972360* (− *17*), *FloatR* (− *920727*) (− *22*)),
(*FloatR 978524* (− *17*), *FloatR* (− *915072*) (− *22*)), *FloatR 8910* (− *9*),
(*FloatR 972360* (− *17*), *FloatR* (− *920727*) (− *22*)), *FloatR 972361* (−
17),
FloatR (− *920726*) (− *22*)),
(*FloatR 8690* (− *9*), (*FloatR 978523* (− *17*), *FloatR* (− *915073*) (− *22*)),
(*FloatR 984649* (− *17*), *FloatR* (− *909501*) (− *22*)), *FloatR 8800* (− *9*),
(*FloatR 978523* (− *17*), *FloatR* (− *915073*) (− *22*)), *FloatR 978524* (−
17),
FloatR (− *915072*) (− *22*)),
(*FloatR 8580* (− *9*), (*FloatR 984648* (− *17*), *FloatR* (− *909502*) (− *22*)),
(*FloatR 990737* (− *17*), *FloatR* (− *904009*) (− *22*)), *FloatR 8690* (− *9*),
(*FloatR 984648* (− *17*), *FloatR* (− *909502*) (− *22*)), *FloatR 984649* (−
17),
FloatR (− *909501*) (− *22*)),
(*FloatR 8470* (− *9*), (*FloatR 990736* (− *17*), *FloatR* (− *904010*) (− *22*)),
(*FloatR 996788* (− *17*), *FloatR* (− *898591*) (− *22*)), *FloatR 8580* (− *9*),
(*FloatR 990736* (− *17*), *FloatR* (− *904010*) (− *22*)), *FloatR 990737* (−
17),
FloatR (− *904009*) (− *22*)),
(*FloatR 8360* (− *9*), (*FloatR 996787* (− *17*), *FloatR* (− *898593*) (− *22*)),
(*FloatR 1002803* (− *17*), *FloatR* (− *893244*) (− *22*)), *FloatR 8470* (− *9*),
(*FloatR 996787* (− *17*), *FloatR* (− *898593*) (− *22*)), *FloatR 996788* (−
17),
FloatR (− *898591*) (− *22*)),
(*FloatR 8250* (− *9*), (*FloatR 1002802* (− *17*), *FloatR* (− *893245*) (− *22*)),
(*FloatR 1008782* (− *17*), *FloatR* (− *887963*) (− *22*)), *FloatR 8360* (− *9*),
(*FloatR 1002802* (− *17*), *FloatR* (− *893245*) (− *22*)), *FloatR 1002803* (−
17),
FloatR (− *893244*) (− *22*)),
(*FloatR 8140* (− *9*), (*FloatR 1008781* (− *17*), *FloatR* (− *887964*) (− *22*)),
(*FloatR 1014726* (− *17*), *FloatR* (− *882744*) (− *22*)), *FloatR 8250* (− *9*),
(*FloatR 1008781* (− *17*), *FloatR* (− *887964*) (− *22*)), *FloatR 1008782* (−
17),
FloatR (− *887963*) (− *22*)),
(*FloatR 8030* (− *9*), (*FloatR 1014725* (− *17*), *FloatR* (− *882745*) (− *22*)),
(*FloatR 1020636* (− *17*), *FloatR* (− *877582*) (− *22*)), *FloatR 8140* (− *9*),
(*FloatR 1014725* (− *17*), *FloatR* (− *882745*) (− *22*)), *FloatR 1014726* (−
17),
FloatR (− *882744*) (− *22*)),
(*FloatR 7920* (− *9*), (*FloatR 1020635* (− *17*), *FloatR* (− *877583*) (− *22*)),
(*FloatR 1026510* (− *17*), *FloatR* (− *872474*) (− *22*)), *FloatR 8030* (− *9*),

(*FloatR 1020635 (− 17), FloatR (− 877583) (− 22)), FloatR 1020636 (− 17),*
FloatR (− 877582) (− 22)),
(FloatR 7810 (− 9), (FloatR 1026509 (− 17), FloatR (− 872475) (− 22)),
(FloatR 1032351 (− 17), FloatR (− 867414) (− 22)), FloatR 7920 (− 9),
(FloatR 1026509 (− 17), FloatR (− 872475) (− 22)), FloatR 1026510 (− 17),
FloatR (− 872474) (− 22)),
(FloatR 7700 (− 9), (FloatR 1032350 (− 17), FloatR (− 867415) (− 22)),
(FloatR 1038158 (− 17), FloatR (− 862398) (− 22)), FloatR 7810 (− 9),
(FloatR 1032350 (− 17), FloatR (− 867415) (− 22)), FloatR 1032351 (− 17),
FloatR (− 867414) (− 22)),
(FloatR 7590 (− 9), (FloatR 1038157 (− 17), FloatR (− 862399) (− 22)),
(FloatR 1043931 (− 17), FloatR (− 857421) (− 22)), FloatR 7700 (− 9),
(FloatR 1038157 (− 17), FloatR (− 862399) (− 22)), FloatR 1038158 (− 17),
FloatR (− 862398) (− 22)),
(FloatR 7480 (− 9), (FloatR 1043930 (− 17), FloatR (− 857422) (− 22)),
(FloatR 524836 (− 16), FloatR (− 852479) (− 22)), FloatR 7590 (− 9),
(FloatR 1043930 (− 17), FloatR (− 857422) (− 22)), FloatR 1043931 (− 17),
FloatR (− 857421) (− 22)),
(FloatR 7370 (− 9), (FloatR 524835 (− 16), FloatR (− 852480) (− 22)),
(FloatR 527689 (− 16), FloatR (− 847567) (− 22)), FloatR 7480 (− 9),
(FloatR 524835 (− 16), FloatR (− 852480) (− 22)), FloatR 524836 (− 16),
FloatR (− 852479) (− 22)),
(FloatR 7260 (− 9), (FloatR 527688 (− 16), FloatR (− 847568) (− 22)),
(FloatR 530526 (− 16), FloatR (− 842678) (− 22)), FloatR 7370 (− 9),
(FloatR 527688 (− 16), FloatR (− 847568) (− 22)), FloatR 527689 (− 16),
FloatR (− 847567) (− 22)),
(FloatR 7150 (− 9), (FloatR 530525 (− 16), FloatR (− 842679) (− 22)),
(FloatR 533347 (− 16), FloatR (− 837808) (− 22)), FloatR 7260 (− 9),
(FloatR 530525 (− 16), FloatR (− 842679) (− 22)), FloatR 530526 (− 16),
FloatR (− 842678) (− 22)),
(FloatR 7040 (− 9), (FloatR 533346 (− 16), FloatR (− 837809) (− 22)),
(FloatR 536151 (− 16), FloatR (− 832951) (− 22)), FloatR 7150 (− 9),
(FloatR 533346 (− 16), FloatR (− 837809) (− 22)), FloatR 533347 (− 16),
FloatR (− 837808) (− 22)),
(FloatR 6930 (− 9), (FloatR 536150 (− 16), FloatR (− 832952) (− 22)),
(FloatR 538939 (− 16), FloatR (− 828102) (− 22)), FloatR 7040 (− 9),
(FloatR 536150 (− 16), FloatR (− 832952) (− 22)), FloatR 536151 (− 16),
FloatR (− 832951) (− 22)),
(FloatR 6820 (− 9), (FloatR 538938 (− 16), FloatR (− 828103) (− 22)),

(*FloatR 541711* (− *16*), *FloatR* (− *823254*) (− *22*)), *FloatR 6930* (− *9*),
(*FloatR 538938* (− *16*), *FloatR* (− *828103*) (− *22*)), *FloatR 538939* (−
16),
FloatR (− *828102*) (− *22*)),
(*FloatR 6710* (− *9*), (*FloatR 541710* (− *16*), *FloatR* (− *823255*) (− *22*)),
(*FloatR 544467* (− *16*), *FloatR* (− *818400*) (− *22*)), *FloatR 6820* (− *9*),
(*FloatR 541710* (− *16*), *FloatR* (− *823255*) (− *22*)), *FloatR 541711* (−
16),
FloatR (− *823254*) (− *22*)),
(*FloatR 6600* (− *9*), (*FloatR 544466* (− *16*), *FloatR* (− *818401*) (− *22*)),
(*FloatR 547206* (− *16*), *FloatR* (− *813534*) (− *22*)), *FloatR 6710* (− *9*),
(*FloatR 544466* (− *16*), *FloatR* (− *818401*) (− *22*)), *FloatR 544467* (−
16),
FloatR (− *818400*) (− *22*)),
(*FloatR 6490* (− *9*), (*FloatR 547205* (− *16*), *FloatR* (− *813535*) (− *22*)),
(*FloatR 549929* (− *16*), *FloatR* (− *808649*) (− *22*)), *FloatR 6600* (− *9*),
(*FloatR 547205* (− *16*), *FloatR* (− *813535*) (− *22*)), *FloatR 547206* (−
16),
FloatR (− *813534*) (− *22*)),
(*FloatR 6380* (− *9*), (*FloatR 549927* (− *16*), *FloatR* (− *808650*) (− *22*)),
(*FloatR 552635* (− *16*), *FloatR* (− *803737*) (− *22*)), *FloatR 6490* (− *9*),
(*FloatR 549927* (− *16*), *FloatR* (− *808650*) (− *22*)), *FloatR 549929* (−
16),
FloatR (− *808649*) (− *22*)),
(*FloatR 6270* (− *9*), (*FloatR 552634* (− *16*), *FloatR* (− *803738*) (− *22*)),
(*FloatR 555325* (− *16*), *FloatR* (− *798791*) (− *22*)), *FloatR 6380* (− *9*),
(*FloatR 552634* (− *16*), *FloatR* (− *803738*) (− *22*)), *FloatR 552635* (−
16),
FloatR (− *803737*) (− *22*)),
(*FloatR 6160* (− *9*), (*FloatR 555324* (− *16*), *FloatR* (− *798792*) (− *22*)),
(*FloatR 557998* (− *16*), *FloatR* (− *793801*) (− *22*)), *FloatR 6270* (− *9*),
(*FloatR 555324* (− *16*), *FloatR* (− *798792*) (− *22*)), *FloatR 555325* (−
16),
FloatR (− *798791*) (− *22*)),
(*FloatR 6050* (− *9*), (*FloatR 557997* (− *16*), *FloatR* (− *793802*) (− *22*)),
(*FloatR 560654* (− *16*), *FloatR* (− *788760*) (− *22*)), *FloatR 6160* (− *9*),
(*FloatR 557997* (− *16*), *FloatR* (− *793802*) (− *22*)), *FloatR 557998* (−
16),
FloatR (− *793801*) (− *22*)),
(*FloatR 5940* (− *9*), (*FloatR 560653* (− *16*), *FloatR* (− *788761*) (− *22*)),
(*FloatR 563293* (− *16*), *FloatR* (− *783659*) (− *22*)), *FloatR 6050* (− *9*),
(*FloatR 560653* (− *16*), *FloatR* (− *788761*) (− *22*)), *FloatR 560654* (−
16),
FloatR (− *788760*) (− *22*)),
(*FloatR 5830* (− *9*), (*FloatR 563292* (− *16*), *FloatR* (− *783660*) (− *22*)),
(*FloatR 565915* (− *16*), *FloatR* (− *778487*) (− *22*)), *FloatR 5940* (− *9*),
(*FloatR 563292* (− *16*), *FloatR* (− *783660*) (− *22*)), *FloatR 563293* (−
16),
FloatR (− *783659*) (− *22*)),

(*FloatR 5720* (− *9*), (*FloatR 565914* (− *16*), *FloatR* (− *778488*) (− *22*)),
(*FloatR 568520* (− *16*), *FloatR* (− *773234*) (− *22*)), *FloatR 5830* (− *9*),
(*FloatR 565914* (− *16*), *FloatR* (− *778488*) (− *22*)), *FloatR 565915* (−
16),
FloatR (− *778487*) (− *22*)),
(*FloatR 5610* (− *9*), (*FloatR 568519* (− *16*), *FloatR* (− *773235*) (− *22*)),
(*FloatR 571107* (− *16*), *FloatR* (− *767889*) (− *22*)), *FloatR 5720* (− *9*),
(*FloatR 568519* (− *16*), *FloatR* (− *773235*) (− *22*)), *FloatR 568520* (−
16),
FloatR (− *773234*) (− *22*)),
(*FloatR 5500* (− *9*), (*FloatR 571106* (− *16*), *FloatR* (− *767890*) (− *22*)),
(*FloatR 573675* (− *16*), *FloatR* (− *762441*) (− *22*)), *FloatR 5610* (− *9*),
(*FloatR 571106* (− *16*), *FloatR* (− *767890*) (− *22*)), *FloatR 571107* (−
16),
FloatR (− *767889*) (− *22*)),
(*FloatR 5390* (− *9*), (*FloatR 573674* (− *16*), *FloatR* (− *762442*) (− *22*)),
(*FloatR 576225* (− *16*), *FloatR* (− *756878*) (− *22*)), *FloatR 5500* (− *9*),
(*FloatR 573674* (− *16*), *FloatR* (− *762442*) (− *22*)), *FloatR 573675* (−
16),
FloatR (− *762441*) (− *22*)),
(*FloatR 5280* (− *9*), (*FloatR 576224* (− *16*), *FloatR* (− *756880*) (− *22*)),
(*FloatR 578757* (− *16*), *FloatR* (− *751188*) (− *22*)), *FloatR 5390* (− *9*),
(*FloatR 576224* (− *16*), *FloatR* (− *756880*) (− *22*)), *FloatR 576225* (−
16),
FloatR (− *756878*) (− *22*)),
(*FloatR 5170* (− *9*), (*FloatR 578756* (− *16*), *FloatR* (− *751189*) (− *22*)),
(*FloatR 581269* (− *16*), *FloatR* (− *745356*) (− *22*)), *FloatR 5280* (− *9*),
(*FloatR 578756* (− *16*), *FloatR* (− *751189*) (− *22*)), *FloatR 578757* (−
16),
FloatR (− *751188*) (− *22*)),
(*FloatR 5060* (− *9*), (*FloatR 581268* (− *16*), *FloatR* (− *745357*) (− *22*)),
(*FloatR 583761* (− *16*), *FloatR* (− *739369*) (− *22*)), *FloatR 5170* (− *9*),
(*FloatR 581268* (− *16*), *FloatR* (− *745357*) (− *22*)), *FloatR 581269* (−
16),
FloatR (− *745356*) (− *22*)),
(*FloatR 4950* (− *9*), (*FloatR 583760* (− *16*), *FloatR* (− *739370*) (− *22*)),
(*FloatR 586233* (− *16*), *FloatR* (− *733212*) (− *22*)), *FloatR 5060* (− *9*),
(*FloatR 583760* (− *16*), *FloatR* (− *739370*) (− *22*)), *FloatR 583761* (−
16),
FloatR (− *739369*) (− *22*)),
(*FloatR 4840* (− *9*), (*FloatR 586232* (− *16*), *FloatR* (− *733213*) (− *22*)),
(*FloatR 588683* (− *16*), *FloatR* (− *726869*) (− *22*)), *FloatR 4950* (− *9*),
(*FloatR 586232* (− *16*), *FloatR* (− *733213*) (− *22*)), *FloatR 586233* (−
16),
FloatR (− *733212*) (− *22*)),
(*FloatR 4730* (− *9*), (*FloatR 588682* (− *16*), *FloatR* (− *726870*) (− *22*)),
(*FloatR 591112* (− *16*), *FloatR* (− *720325*) (− *22*)), *FloatR 4840* (− *9*),
(*FloatR 588682* (− *16*), *FloatR* (− *726870*) (− *22*)), *FloatR 588683* (−
16),

FloatR (− *726869*) (− *22*)),
(*FloatR 4620* (− *9*), (*FloatR 591111* (− *16*), *FloatR* (− *720326*) (− *22*)),
(*FloatR 593519* (− *16*), *FloatR* (− *713561*) (− *22*)), *FloatR 4730* (− *9*),
(*FloatR 591111* (− *16*), *FloatR* (− *720326*) (− *22*)), *FloatR 591112* (−
16),
FloatR (− *720325*) (− *22*)),
(*FloatR 4510* (− *9*), (*FloatR 593518* (− *16*), *FloatR* (− *713562*) (− *22*)),
(*FloatR 595903* (− *16*), *FloatR* (− *706560*) (− *22*)), *FloatR 4620* (− *9*),
(*FloatR 593518* (− *16*), *FloatR* (− *713562*) (− *22*)), *FloatR 593519* (−
16),
FloatR (− *713561*) (− *22*)),
(*FloatR 4400* (− *9*), (*FloatR 595902* (− *16*), *FloatR* (− *706561*) (− *22*)),
(*FloatR 598263* (− *16*), *FloatR* (− *699304*) (− *22*)), *FloatR 4510* (− *9*),
(*FloatR 595902* (− *16*), *FloatR* (− *706561*) (− *22*)), *FloatR 595903* (−
16),
FloatR (− *706560*) (− *22*)),
(*FloatR 4290* (− *9*), (*FloatR 598262* (− *16*), *FloatR* (− *699305*) (− *22*)),
(*FloatR 600598* (− *16*), *FloatR* (− *691773*) (− *22*)), *FloatR 4400* (− *9*),
(*FloatR 598262* (− *16*), *FloatR* (− *699305*) (− *22*)), *FloatR 598263* (−
16),
FloatR (− *699304*) (− *22*)),
(*FloatR 4180* (− *9*), (*FloatR 600597* (− *16*), *FloatR* (− *691774*) (− *22*)),
(*FloatR 602907* (− *16*), *FloatR* (− *683947*) (− *22*)), *FloatR 4290* (− *9*),
(*FloatR 600597* (− *16*), *FloatR* (− *691774*) (− *22*)), *FloatR 600598* (−
16),
FloatR (− *691773*) (− *22*)),
(*FloatR 4070* (− *9*), (*FloatR 602906* (− *16*), *FloatR* (− *683948*) (− *22*)),
(*FloatR 605189* (− *16*), *FloatR* (− *675805*) (− *22*)), *FloatR 4180* (− *9*),
(*FloatR 602906* (− *16*), *FloatR* (− *683948*) (− *22*)), *FloatR 602907* (−
16),
FloatR (− *683947*) (− *22*)),
(*FloatR 3960* (− *9*), (*FloatR 605188* (− *16*), *FloatR* (− *675806*) (− *22*)),
(*FloatR 607444* (− *16*), *FloatR* (− *667325*) (− *22*)), *FloatR 4070* (− *9*),
(*FloatR 605188* (− *16*), *FloatR* (− *675806*) (− *22*)), *FloatR 605189* (−
16),
FloatR (− *675805*) (− *22*)),
(*FloatR 3850* (− *9*), (*FloatR 607443* (− *16*), *FloatR* (− *667326*) (− *22*)),
(*FloatR 609669* (− *16*), *FloatR* (− *658486*) (− *22*)), *FloatR 3960* (− *9*),
(*FloatR 607443* (− *16*), *FloatR* (− *667326*) (− *22*)), *FloatR 607444* (−
16),
FloatR (− *667325*) (− *22*)),
(*FloatR 3740* (− *9*), (*FloatR 609668* (− *16*), *FloatR* (− *658487*) (− *22*)),
(*FloatR 611864* (− *16*), *FloatR* (− *649264*) (− *22*)), *FloatR 3850* (− *9*),
(*FloatR 609668* (− *16*), *FloatR* (− *658487*) (− *22*)), *FloatR 609669* (−
16),
FloatR (− *658486*) (− *22*)),
(*FloatR 3630* (− *9*), (*FloatR 611863* (− *16*), *FloatR* (− *649265*) (− *22*)),
(*FloatR 614028* (− *16*), *FloatR* (− *639637*) (− *22*)), *FloatR 3740* (− *9*),
(*FloatR 611863* (− *16*), *FloatR* (− *649265*) (− *22*)), *FloatR 611864* (−

16),
FloatR (− *649264*) (− *22*)),
(*FloatR 3520* (− *9*), (*FloatR 614027* (− *16*), *FloatR* (− *639638*) (− *22*)),
(*FloatR 616158* (− *16*), *FloatR* (− *629580*) (− *22*)), *FloatR 3630* (− *9*),
(*FloatR 614027* (− *16*), *FloatR* (− *639638*) (− *22*)), *FloatR 614028* (−
16),
FloatR (− *639637*) (− *22*)),
(*FloatR 3410* (− *9*), (*FloatR 616157* (− *16*), *FloatR* (− *629581*) (− *22*)),
(*FloatR 618254* (− *16*), *FloatR* (− *619070*) (− *22*)), *FloatR 3520* (− *9*),
(*FloatR 616157* (− *16*), *FloatR* (− *629581*) (− *22*)), *FloatR 616158* (−
16),
FloatR (− *629580*) (− *22*)),
(*FloatR 3300* (− *9*), (*FloatR 618253* (− *16*), *FloatR* (− *619071*) (− *22*)),
(*FloatR 620314* (− *16*), *FloatR* (− *608083*) (− *22*)), *FloatR 3410* (− *9*),
(*FloatR 618253* (− *16*), *FloatR* (− *619071*) (− *22*)), *FloatR 618254* (−
16),
FloatR (− *619070*) (− *22*)),
(*FloatR 3190* (− *9*), (*FloatR 620313* (− *16*), *FloatR* (− *608084*) (− *22*)),
(*FloatR 622336* (− *16*), *FloatR* (− *596594*) (− *22*)), *FloatR 3300* (− *9*),
(*FloatR 620313* (− *16*), *FloatR* (− *608084*) (− *22*)), *FloatR 620314* (−
16),
FloatR (− *608083*) (− *22*)),
(*FloatR 3080* (− *9*), (*FloatR 622335* (− *16*), *FloatR* (− *596595*) (− *22*)),
(*FloatR 624319* (− *16*), *FloatR* (− *584581*) (− *22*)), *FloatR 3190* (− *9*),
(*FloatR 622335* (− *16*), *FloatR* (− *596595*) (− *22*)), *FloatR 622336* (−
16),
FloatR (− *596594*) (− *22*)),
(*FloatR 2970* (− *9*), (*FloatR 624318* (− *16*), *FloatR* (− *584582*) (− *22*)),
(*FloatR 626260* (− *16*), *FloatR* (− *572019*) (− *22*)), *FloatR 3080* (− *9*),
(*FloatR 624318* (− *16*), *FloatR* (− *584582*) (− *22*)), *FloatR 624319* (−
16),
FloatR (− *584581*) (− *22*)),
(*FloatR 2860* (− *9*), (*FloatR 626259* (− *16*), *FloatR* (− *572020*) (− *22*)),
(*FloatR 628159* (− *16*), *FloatR* (− *558886*) (− *22*)), *FloatR 2970* (− *9*),
(*FloatR 626259* (− *16*), *FloatR* (− *572020*) (− *22*)), *FloatR 626260* (−
16),
FloatR (− *572019*) (− *22*)),
(*FloatR 2750* (− *9*), (*FloatR 628158* (− *16*), *FloatR* (− *558888*) (− *22*)),
(*FloatR 630012* (− *16*), *FloatR* (− *545161*) (− *22*)), *FloatR 2860* (− *9*),
(*FloatR 628158* (− *16*), *FloatR* (− *558888*) (− *22*)), *FloatR 628159* (−
16),
FloatR (− *558886*) (− *22*)),
(*FloatR 2640* (− *9*), (*FloatR 630011* (− *16*), *FloatR* (− *545162*) (− *22*)),
(*FloatR 631818* (− *16*), *FloatR* (− *530822*) (− *22*)), *FloatR 2750* (− *9*),
(*FloatR 630011* (− *16*), *FloatR* (− *545162*) (− *22*)), *FloatR 630012* (−
16),
FloatR (− *545161*) (− *22*)),
(*FloatR 2530* (− *9*), (*FloatR 631817* (− *16*), *FloatR* (− *530823*) (− *22*)),
(*FloatR 633575* (− *16*), *FloatR* (− *1031701*) (− *23*)), *FloatR 2640* (− *9*),

(*FloatR 631817* (− *16*), *FloatR* (− *530823*) (− *22*)), *FloatR 631818* (−
16),
FloatR (− *530822*) (− *22*)),
(*FloatR 2420* (− *9*), (*FloatR 633574* (− *16*), *FloatR* (− *1031702*) (− *23*)),
(*FloatR 635281* (− *16*), *FloatR* (− *1000458*) (− *23*)), *FloatR 2530* (− *9*),
(*FloatR 633574* (− *16*), *FloatR* (− *1031702*) (− *23*)), *FloatR 633575* (−
16),
FloatR (− *1031701*) (− *23*)),
(*FloatR 2310* (− *9*), (*FloatR 635280* (− *16*), *FloatR* (− *1000459*) (− *23*)),
(*FloatR 636933* (− *16*), *FloatR* (− *967884*) (− *23*)), *FloatR 2420* (− *9*),
(*FloatR 635280* (− *16*), *FloatR* (− *1000459*) (− *23*)), *FloatR 635281* (−
16),
FloatR (− *1000458*) (− *23*)),
(*FloatR 2200* (− *9*), (*FloatR 636932* (− *16*), *FloatR* (− *967885*) (− *23*)),
(*FloatR 638529* (− *16*), *FloatR* (− *933955*) (− *23*)), *FloatR 2310* (− *9*),
(*FloatR 636932* (− *16*), *FloatR* (− *967885*) (− *23*)), *FloatR 636933* (−
16),
FloatR (− *967884*) (− *23*)),
(*FloatR 2090* (− *9*), (*FloatR 638528* (− *16*), *FloatR* (− *933956*) (− *23*)),
(*FloatR 640067* (− *16*), *FloatR* (− *898650*) (− *23*)), *FloatR 2200* (− *9*),
(*FloatR 638528* (− *16*), *FloatR* (− *933956*) (− *23*)), *FloatR 638529* (−
16),
FloatR (− *933955*) (− *23*)),
(*FloatR 1980* (− *9*), (*FloatR 640066* (− *16*), *FloatR* (− *898651*) (− *23*)),
(*FloatR 641545* (− *16*), *FloatR* (− *861955*) (− *23*)), *FloatR 2090* (− *9*),
(*FloatR 640066* (− *16*), *FloatR* (− *898651*) (− *23*)), *FloatR 640067* (−
16),
FloatR (− *898650*) (− *23*)),
(*FloatR 1870* (− *9*), (*FloatR 641544* (− *16*), *FloatR* (− *861957*) (− *23*)),
(*FloatR 642960* (− *16*), *FloatR* (− *823865*) (− *23*)), *FloatR 1980* (− *9*),
(*FloatR 641544* (− *16*), *FloatR* (− *861957*) (− *23*)), *FloatR 641545* (−
16),
FloatR (− *861955*) (− *23*)),
(*FloatR 1760* (− *9*), (*FloatR 642959* (− *16*), *FloatR* (− *823866*) (− *23*)),
(*FloatR 644310* (− *16*), *FloatR* (− *784380*) (− *23*)), *FloatR 1870* (− *9*),
(*FloatR 642959* (− *16*), *FloatR* (− *823866*) (− *23*)), *FloatR 642960* (−
16),
FloatR (− *823865*) (− *23*)),
(*FloatR 1650* (− *9*), (*FloatR 644309* (− *16*), *FloatR* (− *784382*) (− *23*)),
(*FloatR 645592* (− *16*), *FloatR* (− *743511*) (− *23*)), *FloatR 1760* (− *9*),
(*FloatR 644309* (− *16*), *FloatR* (− *784381*) (− *23*)), *FloatR 644310* (−
16),
FloatR (− *784380*) (− *23*)),
(*FloatR 1540* (− *9*), (*FloatR 645591* (− *16*), *FloatR* (− *743512*) (− *23*)),
(*FloatR 646805* (− *16*), *FloatR* (− *701274*) (− *23*)), *FloatR 1650* (− *9*),
(*FloatR 645591* (− *16*), *FloatR* (− *743512*) (− *23*)), *FloatR 645592* (−
16),
FloatR (− *743511*) (− *23*)),
(*FloatR 1430* (− *9*), (*FloatR 646804* (− *16*), *FloatR* (− *701275*) (− *23*)),

(*FloatR 647946* (− *16*), *FloatR* (− *657699*) (− *23*)), *FloatR 1540* (− *9*),
(*FloatR 646804* (− *16*), *FloatR* (− *701275*) (− *23*)), *FloatR 646805* (−
16),
FloatR (− *701274*) (− *23*)),
(*FloatR 1320* (− *9*), (*FloatR 647945* (− *16*), *FloatR* (− *657700*) (− *23*)),
(*FloatR 649012* (− *16*), *FloatR* (− *612824*) (− *23*)), *FloatR 1430* (− *9*),
(*FloatR 647945* (− *16*), *FloatR* (− *657700*) (− *23*)), *FloatR 647946* (−
16),
FloatR (− *657699*) (− *23*)),
(*FloatR 1210* (− *9*), (*FloatR 649011* (− *16*), *FloatR* (− *612825*) (− *23*)),
(*FloatR 650002* (− *16*), *FloatR* (− *566695*) (− *23*)), *FloatR 1320* (− *9*),
(*FloatR 649011* (− *16*), *FloatR* (− *612825*) (− *23*)), *FloatR 649012* (−
16),
FloatR (− *612824*) (− *23*)),
(*FloatR 1100* (− *9*), (*FloatR 650001* (− *16*), *FloatR* (− *566696*) (− *23*)),
(*FloatR 650914* (− *16*), *FloatR* (− *1038743*) (− *24*)), *FloatR 1210* (− *9*),
(*FloatR 650001* (− *16*), *FloatR* (− *566696*) (− *23*)), *FloatR 650002* (−
16),
FloatR (− *566695*) (− *23*)),
(*FloatR 990* (− *9*), (*FloatR 650913* (− *16*), *FloatR* (− *1038744*) (− *24*)),
(*FloatR 651745* (− *16*), *FloatR* (− *941842*) (− *24*)), *FloatR 1100* (− *9*),
(*FloatR 650913* (− *16*), *FloatR* (− *1038744*) (− *24*)), *FloatR 650914* (−
16),
FloatR (− *1038743*) (− *24*)),
(*FloatR 880* (− *9*), (*FloatR 651744* (− *16*), *FloatR* (− *941843*) (− *24*)),
(*FloatR 652494* (− *16*), *FloatR* (− *842845*) (− *24*)), *FloatR 990* (− *9*),
(*FloatR 651744* (− *16*), *FloatR* (− *941843*) (− *24*)), *FloatR 651745* (−
16),
FloatR (− *941842*) (− *24*)),
(*FloatR 770* (− *9*), (*FloatR 652493* (− *16*), *FloatR* (− *842846*) (− *24*)),
(*FloatR 653159* (− *16*), *FloatR* (− *741927*) (− *24*)), *FloatR 880* (− *9*),
(*FloatR 652493* (− *16*), *FloatR* (− *842846*) (− *24*)), *FloatR 652494* (−
16),
FloatR (− *842845*) (− *24*)),
(*FloatR 660* (− *9*), (*FloatR 653158* (− *16*), *FloatR* (− *741928*) (− *24*)),
(*FloatR 653739* (− *16*), *FloatR* (− *639284*) (− *24*)), *FloatR 770* (− *9*),
(*FloatR 653158* (− *16*), *FloatR* (− *741928*) (− *24*)), *FloatR 653159* (−
16),
FloatR (− *741927*) (− *24*)),
(*FloatR 550* (− *9*), (*FloatR 653738* (− *16*), *FloatR* (− *639285*) (− *24*)),
(*FloatR 654232* (− *16*), *FloatR* (− *535127*) (− *24*)), *FloatR 660* (− *9*),
(*FloatR 653738* (− *16*), *FloatR* (− *639285*) (− *24*)), *FloatR 653739* (−
16),
FloatR (− *639284*) (− *24*)),
(*FloatR 440* (− *9*), (*FloatR 654231* (− *16*), *FloatR* (− *535128*) (− *24*)),
(*FloatR 654637* (− *16*), *FloatR* (− *859366*) (− *25*)), *FloatR 550* (− *9*),
(*FloatR 654231* (− *16*), *FloatR* (− *535128*) (− *24*)), *FloatR 654232* (−
16),
FloatR (− *535127*) (− *24*)),

(*FloatR 330* (− *9*), (*FloatR 654636* (− *16*), *FloatR* (− *859367*) (− *25*)),
(*FloatR 654953* (− *16*), *FloatR* (− *646385*) (− *25*)), *FloatR 440* (− *9*),
(*FloatR 654636* (− *16*), *FloatR* (− *859367*) (− *25*)), *FloatR 654637* (−
16),
FloatR (− *859366*) (− *25*)),
(*FloatR 220* (− *9*), (*FloatR 654952* (− *16*), *FloatR* (− *646387*) (− *25*)),
(*FloatR 655179* (− *16*), *FloatR* (− *863632*) (− *26*)), *FloatR 330* (− *9*),
(*FloatR 654952* (− *16*), *FloatR* (− *646387*) (− *25*)), *FloatR 654953* (−
16),
FloatR (− *646386*) (− *25*)),
(*FloatR 110* (− *9*), (*FloatR 655178* (− *16*), *FloatR* (− *863633*) (− *26*)),
(*FloatR 655315* (− *16*), *FloatR* (− *864707*) (− *27*)), *FloatR 220* (− *9*),
(*FloatR 655178* (− *16*), *FloatR* (− *863633*) (− *26*)), *FloatR 655179* (−
16),
FloatR (− *863632*) (− *26*)),
(*FloatR 0 0*, (*FloatR 655314* (− *16*), *FloatR* (− *864708*) (− *27*)),
(*FloatR 655360* (− *16*), *FloatR 869715* (− *57*)), *FloatR 110* (− *9*),
(*FloatR 655314* (− *16*), *FloatR* (− *864708*) (− *27*)), *FloatR 655315* (−
16),
FloatR (− *864707*) (− *27*))])
oops — by eval

end
theory *Example-van-der-Pol*
imports
../Numerics/Euler-Affine
../Numerics/Optimize-Float
begin

16.5 Van der Pol oscillator

abbreviation *vanderpol-real* ≡ λ(*x*::*real*, *y*::*real*). (*y*, *y* ∗ (*1* + − *x*∗*x*) + − *x*)

approximate-affine *vanderpol vanderpol-real*

abbreviation *of-matrix22* ≡ λ*a b c d*. λ(*e*, *f*). (*a* ∗ *e* + *b* ∗ *f*, *c* ∗ *e* + *d* ∗ *f*)::*real*∗*real*

abbreviation *vanderpol-d-real* ≡ λ(*x*, *y*). *of-matrix22 0 1* (− (*1* + *2* ∗ *x* ∗ *y*)) (− *x* ∗ *x* + *1*)

lemma *vanderpol-fderiv*:
(*vanderpol-real has-derivative vanderpol-d-real x*) (*at x within X*)
by (*auto intro*!: *derivative-eq-intros ext simp*: *split-beta inverse-eq-divide algebra-simps*)

approximate-affine *vanderpol-d vanderpol-d-real*

abbreviation *vanderpol-ivp* ≡ λ*optns args*. *uncurry-options vanderpol optns* (*hd args*) (*tl args*)

abbreviation *vanderpol-d-ivp* ≡ *λoptns args. uncurry-options vanderpol-d optns* (*hd args*) (*hd* (*tl args*)) (*tl* (*tl args*))

interpretation *vanderpol*: *aform-approximate-ivp*
vanderpol-ivp vanderpol-d-ivp
vanderpol-real
vanderpol-d-real
apply *unfold-locales*
unfolding *list.sel*
apply (*rule Joints2-JointsI*)
apply (*rule vanderpol*, *assumption*, *assumption*)
apply (*drule length-set-of-apprs*, *simp*)— TODO: prove in affine-approximation
apply (*rule vanderpol-fderiv*)
apply (*rule vanderpol-d*[*THEN Joints2-JointsI*]) **apply** *assumption* **apply** *assumption*
apply (*drule length-set-of-apprs*, *simp*)— TODO: prove in affine-approximation
apply (*auto intro*!: *continuous-intros simp*: *split-beta*)
apply *intro-locales*
done

definition *vanderpoltest* =
(*poincare-distance-d vanderpol-ivp vanderpol-d-ivp*
(|
precision = *30*,
tolerance = *FloatR 1* (− *5*),
stepsize = *FloatR 1* (− *6*),
min-stepsize = *FloatR 1* (− *7*),
iterations = *40*,
halve-stepsizes = *10*,
widening-mod = *40*,
max-tdev-thres = *FloatR 1* (− *3*),
presplit-summary-tolerance = *FloatR 1* (− *1*),
collect-mod = *30*,
collect-granularity = *FloatR 1* (− *4*),
override-section = (*λb y i s. if snd i* > *FloatR 149* (− *6*) *then* ((*0*, *1*), *FloatR 149* (− *6*)) *else*
if snd i = *FloatR 149* (− *6*) ∧ *snd s* = *FloatR 149* (− *6*) *then* (*0*, *0*) *else* (*b*, *y*)),
global-section = (*λX. None*),
stop-iteration = (*λX. False*),
printing-fun = (*λ- -. print-aform*),
result-fun = *ivls-result 20 40*
|))

vanderpoltest [*aform-of-ivl* (*FloatR 5* (− *2*), *FloatR 146* (− *6*)) (*FloatR 49* (− *5*), *FloatR 149* (− *6*))] proves a stable limit-cycle.

value *vanderpoltest* [*aform-of-ivl* (*FloatR 5* (− *2*), *FloatR 146* (− *6*)) (*FloatR 49* (− *5*), *FloatR 149* (− *6*))]

end

theory *Example-Variational-Equation*
imports
../Library/Linear-ODE
Example-van-der-Pol
begin

16.6 Variational equation for the van der Pol system

lift-definition *blinfun-of-matrix22::real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ (*real* × *real*) $\Rightarrow_L$ (*real* × *real*)
is *of-matrix22*
by (*auto intro*!: *bounded-linearI′ simp*: *algebra-simps*)

definition *vanderpol-d-blinfun* ≡ λ(*x*, *y*). *blinfun-of-matrix22 0 1* (− (*1* + *2* ∗ *x* ∗ *y*)) (− *x* ∗ *x* + *1*)

interpretation *vanderpol*: *c1-on-open-euclidean vanderpol-real vanderpol-d-blinfun UNIV*
apply *unfold-locales*
apply (*force intro*!: *derivative-eq-intros continuous-on-blinfun-componentwise*
intro: *continuous-intros*
simp: *vanderpol-d-blinfun-def blinfun-of-matrix22.rep-eq algebra-simps split-beta′*)+
done

abbreviation *vareq-real::real* ∗ *real* ∗ *real* ∗ *real* ∗ *real* ∗ *real* ⇒ *real* ∗ *real* ∗ *real* ∗ *real* ∗ *real* ∗ *real*
where *vareq-real* ≡ λ(*x*, *y*, *a*, *b*, *c*, *d*).
(*y*, *y* ∗ (*1* + − *x*∗*x*) + − *x*,
c, *d*,
− (*a* ∗ (*1* + *2* ∗ *x* ∗ *y*)) + *c* ∗ (− *x* ∗ *x* + *1*),
− (*b* ∗ (*1* + *2* ∗ *x* ∗ *y*)) + *d* ∗ (− *x* ∗ *x* + *1*))

approximate-affine *vareq vareq-real*

abbreviation *vareq-d-real::real* ∗ *real* ∗ *real* ∗ *real* ∗ *real* ∗ *real* ⇒
real ∗ *real* ∗ *real* ∗ *real* ∗ *real* ∗ *real* ⇒ *real* ∗ *real* ∗ *real* ∗ *real* ∗ *real* ∗ *real*
where *vareq-d-real* ≡ λ(*x*, *y*, *a*, *b*, *c*, *d*). λ(*d1*, *d2*, *d3*, *d4*, *d5*, *d6*).
(*d2*, *d2* ∗ (*1* +− *x* ∗ *x*) +− *y* ∗ (*2* ∗ (*d1* ∗ *x*))+− *d1*, *d5*, *d6*,
d5 ∗ (*1* +− *x* ∗ *x*) +− *c* ∗ (*2* ∗ (*d1* ∗ *x*)) +− (*a* ∗ (*2* ∗ *x* ∗ *d2* + *2* ∗ *d1* ∗ *y*) + *d3* ∗ (*1* + *2* ∗ *x* ∗ *y*)),
d6 ∗ (*1* +− *x* ∗ *x*) +− *d* ∗ (*2* ∗ (*d1* ∗ *x*)) +− (*b* ∗ (*2* ∗ *x* ∗ *d2* + *2* ∗ *d1* ∗ *y*) + *d4* ∗ (*1* + *2* ∗ *x* ∗ *y*)))

approximate-affine *vareq-d vareq-d-real*

lift-definition *vareq-d-blinfun*::
(*real* × *real* × *real* × *real* × *real* × *real*) ⇒
(*real* × *real* × *real* × *real* × *real* × *real*) $\Rightarrow_L$ (*real* × *real* × *real* × *real* × *real* × *real*) **is**

vareq-d-real
by (*auto intro*!: *bounded-linearI′ simp*: *algebra-simps*)

lemma *vareq-fderiv*:
(*vareq-real has-derivative vareq-d-real x*) (*at x within X*)
by (*auto intro*!: *derivative-eq-intros ext simp*: *split-beta′*)

interpretation *vareq*: *c1-on-open-euclidean vareq-real vareq-d-blinfun UNIV*
by *unfold-locales*
(*force intro*!: *derivative-eq-intros continuous-on-blinfun-componentwise*
intro: *continuous-intros*
simp: *vareq-d-blinfun.rep-eq algebra-simps split-beta′*)+

abbreviation *vareq-ivp* ≡ *λoptns args. uncurry-options vareq optns* (*hd args*) (*tl args*)
abbreviation *vareq-d-ivp* ≡ *λoptns args. uncurry-options vareq-d optns* (*hd args*) (*hd* (*tl args*)) (*tl* (*tl args*))

interpretation *vareq*: *aform-approximate-ivp*
vareq-ivp vareq-d-ivp
vareq-real
vareq-d-real
apply *unfold-locales*
unfolding *list.sel*
apply (*rule Joints2-JointsI*)
apply (*rule vareq*, *assumption*, *assumption*)
apply (*drule length-set-of-apprs*, *simp*)— TODO: prove in affine-approximation
apply (*rule vareq-fderiv*)
apply (*rule vareq-d*[*THEN Joints2-JointsI*]) **apply** *assumption* **apply** *assumption*
apply (*drule length-set-of-apprs*, *simp*)— TODO: prove in affine-approximation
apply (*auto intro*!: *continuous-intros simp*: *split-beta*)
apply *intro-locales*
done

definition *vareqtest* =
(*euler-series-result vareq-ivp vareq-d-ivp*
(|
precision = *30*,
tolerance = *FloatR 1* (− *5*),
stepsize = *FloatR 1* (− *4*),
min-stepsize = *FloatR 1* (− *8*),
iterations = *40*,
halve-stepsizes = *10*,
widening-mod = *40*,
max-tdev-thres = *FloatR 1* (− *8*),
presplit-summary-tolerance = *FloatR 1* (− *1*),
collect-mod = *30*,
collect-granularity = *FloatR 1* (− *1*),
override-section = (*λb y i s.* ((*0*, *1*, *0*, *0*, *0*, *0*), *FloatR 4* (−*1*))),

global-section = (λ*X*. *Some* (((*0*, *1*, *0*, *0*, *0*, *0*), *FloatR 4* (−*1*)))),
stop-iteration = (λ*X*. *True*),
printing-fun = (λ*i t*. *print-rectangle i i t*),
result-fun = *ivls-result 20 40*
⦈ *0*)

value[*code*] *vareqtest* (*aform-of-point* (*FloatR 5* (− *2*), *FloatR 146* (− *6*), *1*, *0*, *0*, *1*)) *10*

lemma *blinfun-apply-vanderpol-d-blinfun*: *blinfun-apply* (*vanderpol-d-blinfun x*) *y* =
(*snd y*, (− *1* − *2* ∗ *fst x* ∗ *snd x*) ∗ *fst y* + (*1* − *fst x* ∗ *fst x*) ∗ *snd y*)
by (*auto simp*: *vanderpol-d-blinfun-def blinfun-of-matrix22*.*rep-eq split-beta′*)

TODO: generalize?

lemma *vareq-encoding*:
notes [*simp del*] = *add-uminus-conv-diff*
assumes *t* ∈ *vanderpol*.*existence-ivl* (*x0*, *y0*)
shows
vareq.*flow*(*x0*, *y0*, *1*, *0*, *0*, *1*) *t* =
(*let*
xy = *vanderpol*.*flow* (*x0*, *y0*) *t*;
M = *vanderpol*.*W* (*x0*, *y0*) *t*;
ac = *M* (*1*, *0*);
bd = *M* (*0*, *1*)
in (*fst xy*, *snd xy*, *fst ac*, *fst bd*, *snd ac*, *snd bd*))
(**is** *?l* = *?r*)
proof −
from *vanderpol*.*total-derivative-ll-on-open*[*of* (*x0*, *y0*)]
interpret *mvar*: *ll-on-open* (λ*t*. *op* o_L (*vanderpol*.*A* (*x0*, *y0*) *t*)) (*vanderpol*.*existence-ivl* (*x0*, *y0*)) *UNIV*::((*real* × *real*) $\Rightarrow_L$ (*real* × *real*)) *set*
by *auto*
have *W-eq*: *vanderpol*.*W* (*x0*, *y0*) = *mvar*.*flow 0 id-blinfun*
by (*subst vanderpol*.*W-def*) *auto*
have *mvar-existence-ivlI*: *t* ∈ *vanderpol*.*existence-ivl* (*x0*, *y0*) ⟹ *t* ∈ *mvar*.*existence-ivl 0 id-blinfun* **for** *t*
using *vanderpol*.*existence-ivl-zero*
by (*subst vanderpol*.*wholevar-existence-ivl-eq-existence-ivl*)
(*auto*)
have *?l* = *vareq*.*na*.*flow 0* (*x0*, *y0*, *1*, *0*, *0*, *1*) *t*
unfolding *vareq*.*flow-def* **..**
also have . . . = *?r*
apply (*rule vareq*.*na*.*maximal-existence-flowI*[**where** *K*=*vanderpol*.*existence-ivl* (*x0*, *y0*)])
unfolding *vareq*.*flow-def*[*symmetric*] *W-eq*
subgoal by *simp*
subgoal by *simp*
subgoal for *t*
unfolding *Let-def*

proof *goal-cases*
case *hyps*: *1*
have *eq*: *vanderpol.A* (*x0*, *y0*) *t* = *vanderpol-d-blinfun* (*vanderpol.flow* (*x0*, *y0*) *t*)
unfolding *vanderpol.A-def vanderpol.XX-def*
by *auto*
show *?case*
unfolding *at-within-open*[*OF hyps vanderpol.open-existence-ivl*] *has-vector-derivative-def*
apply (*rule derivative-eq-intros vanderpol.flow-has-derivative UNIV-I hyps refl*
mvar.flow-has-derivative vanderpol.existence-ivl-zero mvar-existence-ivlI)+
unfolding *blinfun.bilinear-simps eq blinfun-apply-vanderpol-d-blinfun*
blinfun-apply-blinfun-compose
by (*auto simp*: *algebra-simps prod-eq-iff*
intro!: *ext simp*: *blinfun.bilinear-simps split*: *prod.split*)
qed
subgoal by *simp*
subgoal by (*simp only*: *vanderpol.existence-ivl-zero mvar.flow-initial-time UNIV-I*
vanderpol.flow-zero blinfun-apply-id-blinfun fst-conv snd-conv Let-def)
subgoal by (*rule vanderpol.is-interval-existence-ivl*)
subgoal by (*rule vanderpol.existence-ivl-zero*) *simp*
subgoal by *simp*
subgoal by (*rule assms*)
done
finally show *?thesis* **.**
qed

lemma *blinfun-of-matrix22-works*:
fixes *W*::(*real* × *real*) $\Rightarrow_L$ (*real* × *real*)
shows *blinfun-of-matrix22*
(*fst* (*W* (*1*, *0*)))
(*fst* (*W* (*0*, *1*)))
(*snd* (*W* (*1*, *0*)))
(*snd* (*W* (*0*, *1*))) = *W*
apply (*auto intro*!: *blinfun-eqI*)
apply (*auto simp*: *blinfun-of-matrix22.rep-eq blinfun.bilinear-simps*[*symmetric*])
proof *goal-cases*
case (*1 a b*)
have (*fst* (*W* (*1*, *0*)) ∗ *a* + *fst* (*W* (*0*, *1*)) ∗ *b*, *snd* (*W* (*1*, *0*)) ∗ *a* + *snd* (*W* (*0*, *1*)) ∗ *b*) =
(*fst* (*a* $*_R$ *W* (*1*, *0*)) + *fst* (*b* $*_R$ *W* (*0*, *1*)), *snd* (*a* $*_R$ *W* (*1*, *0*)) + *snd* (*b* $*_R$ *W* (*0*, *1*)))
by *simp*
also have ... = (*fst* (*W* (*a* $*_R$ (*1*, *0*))) + *fst* (*W* (*b* $*_R$ (*0*, *1*))),
snd (*W* (*a* $*_R$ (*1*, *0*))) + *snd* (*W* (*b* $*_R$ (*0*, *1*))))
unfolding *blinfun.scaleR-right scaleR-blinfun.rep-eq*[*symmetric*] **..**
also have ... = (*fst* (*W* ((*a*, *0*))) + *fst* (*W* ((*0*, *b*))), *snd* (*W* ((*a*, *0*))) + *snd* (*W* ((*0*, *b*))))

by *auto*
also have ... = (*fst* (*W* ((*a*, *0*)) + *W* ((*0*, *b*))), *snd* (*W* ((*a*, *0*)) + *W* ((*0*, *b*))))
by *auto*
also have ... = (*fst* (*W* (*a*, *b*)), *snd* (*W* (*a*, *b*)))
unfolding *blinfun.add-right*[*symmetric*]
by *auto*
finally show *?case* **by** *simp*
qed

lemma *compute-vareq*:
assumes *t* ∈ *vanderpol.existence-ivl* (*x0*, *y0*)
shows
(*vanderpol.flow* (*x0*, *y0*) *t*, *vanderpol.W* (*x0*, *y0*) *t*) =
(*let*
(*x*, *y*, *a*, *b*, *c*, *d*) = *vareq.flow* (*x0*, *y0*, *1*, *0*, *0*, *1*) *t*
in ((*x*, *y*), *blinfun-of-matrix22 a b c d*))
using *vareq-encoding*[*OF assms*]
by (*auto simp*: *Let-def blinfun-of-matrix22.rep-eq blinfun.bilinear-simps*
blinfun-of-matrix22-works
intro!: *blinfun-eqI*)

end
theory *Examples*
imports
Example1
Example3
Example-Exp
Example-Oil
Example-van-der-Pol
Example-Variational-Equation
begin

end
theory *Ordinary-Differential-Equations*
imports
Library/MVT-Ex
Library/Linear-ODE
Ex/Examples
begin

end

References

[1] F. Immler. Affine arithmetic. *Archive of Formal Proofs*, Feb. 2014. http://afp.sf.net/devel-entries/Affine_Arithmetic.shtml, Formal proof development.

[2] F. Immler and J. Hölzl. Formally Verified Enclosures of Solutions of Ordinary Differential Equations. In *NASA Formal Methods Symposium 2014*, LNCS.

[3] F. Immler and J. Hölzl. Numerical Analysis of Ordinary Differential Equations in Isabelle/HOL. In *ITP 2012*, LNCS.

[4] W. Walter. *Ordinary Differential Equations*. Springer, 1 edition, 1998.