An Isabelle/HOL formalisation of Green's Theorem

Mohammad Abdulaziz and Lawrence C. Paulson

March 17, 2025

Abstract

We formalise a statement of Greens theorem—the first formalisation to our knowledge—in Isabelle/HOL. The theorem statement that we formalise is enough for most applications, especially in physics and engineering. Our formalisation is made possible by a novel proof that avoids the ubiquitous line integral cancellation argument. This eliminates the need to formalise orientations and region boundaries explicitly with respect to the outwards-pointing normal vector. Instead we appeal to a homological argument about equivalences between paths.

1 Acknowledgements

Paulson was supported by the ERC Advanced Grant ALEXANDRIA (Project 742178) funded by the European Research Council at the University of Cambridge, UK.

theory *General-Utils*
  imports *HOL−Analysis.Analysis*
begin

lemma *lambda-skolem-gen*: $(\forall i.\ \exists f'::('a\ \hat{}\ 'n) \Rightarrow 'a.\ P\ i\ f') \longleftrightarrow$
  $(\exists f'::('a\ \hat{}\ 'n) \Rightarrow ('a\ \hat{}\ 'n).\ \forall i.\ P\ i\ ((\lambda x.\ (f'\ x)\ \$\ i)))$ (**is** *?lhs* $\longleftrightarrow$ *?rhs*)

proof −
  { assume *H*: *?rhs*
    then have *?lhs* **by** *auto* **}**
  moreover
  { assume *H*: *?lhs*
    then obtain f'' **where** f'':$\forall i.\ P\ i\ (f''\ i)$ **unfolding** *choice-iff*
      by *metis*
    let $?x = (\lambda x.\ (\chi\ i.\ (f''\ \ i)\ x))$
    { fix *i*
      from f'' **have** $P\ i\ (f''\ i)$ **by** *metis*
      then have $P\ i\ (\lambda x.(?x\ x)\ \$\ i)$ **by** *auto*
    }
    hence $\forall i.\ P\ i\ (\lambda x.(?x\ x)\ \$\ i)$ **by** *metis*
    hence *?rhs* **by** *metis***}**

```
  ultimately show ?thesis by metis
qed

lemma lambda-skolem-euclidean: (∀ i∈Basis. ∃ f'::('a::{euclidean-space}⇒real). P
i f') ⟷
   (∃ f'::('a::euclidean-space⇒'b::euclidean-space). ∀ i∈Basis. P i ((λx. (f' x) · i)))
(is ?lhs ⟷ ?rhs)

proof −
  { assume H: ?rhs
    then have ?lhs by auto }
  moreover
  { assume H: ?lhs
   then obtain f'' where f'':∀ i::('b::euclidean-space)∈Basis. P i (f'' i) unfolding
choice-iff
      by metis
    let ?x = (λx. (∑ i∈Basis. ((f''  i) x) *R i))
    { fix i::'b::euclidean-space
      assume ass: i∈Basis
      then have P i (f'' i)
        using f''
        by metis
      then have P i (λx.(?x x) · i) using ass by auto
    }
    hence *: ∀ i∈Basis. P i (λx.(?x x) · i) by auto
    then have ?rhs
      apply auto
    proof
      let ?f'6 = ?x
      show  ∀ i∈Basis. P i (λx. ?f'6 x · i) using * by auto
    qed}
  ultimately show ?thesis by metis
qed

lemma lambda-skolem-euclidean-explicit: (∀ i∈Basis. ∃ f'::('a::{euclidean-space}⇒real).
P i f') ⟷
   (∃ f'::('a::{euclidean-space}⇒'a). ∀ i∈Basis. P i ((λx. (f' x) · i))) (is ?lhs ⟷
?rhs)

proof −
  { assume H: ?rhs
    then have ?lhs by auto }
  moreover
  { assume H: ?lhs
   then obtain f'' where f'':∀ i::('a::euclidean-space)∈Basis. P i (f'' i) unfolding
choice-iff
      by metis
    let ?x = (λx. (∑ i∈Basis. ((f''  i) x) *R i))
    { fix i::'a::euclidean-space
```

```
      assume ass: i∈Basis
      then have P i (f'' i)
        using f''
        by metis
      then have P i (λx.(?x x) • i) using ass by auto
    }
    hence *: ∀ i∈Basis. P i (λx.(?x x) • i) by auto
    then have ?rhs
      apply auto
    proof
      let ?f'6 = ?x
      show  ∀ i∈Basis. P i (λx. ?f'6 x • i) using * by auto
    qed}
  ultimately show ?thesis by metis
qed

lemma indic-ident:
  ⋀ (f::'a ⇒ real) s. (λx. (f x) * indicator s x) = (λx. if x ∈ s then f x else 0)
proof
  fix f::'a ⇒ real
  fix s::'a set
  fix x:: 'a
  show  f x * indicator s x = (if x ∈ s then f x else 0)
    by (simp add: indicator-def)
qed

lemma real-pair-basis: Basis = {(1::real,0::real), (0::real,1::real)}
  by (simp add: Basis-prod-def insert-commute)

lemma real-singleton-in-borel:
  shows {a::real} ∈ sets borel
  using Borel-Space.cbox-borel[of a a]
  apply auto
  done

lemma real-singleton-in-lborel:
  shows {a::real} ∈ sets lborel
  using real-singleton-in-borel
  apply auto
  done

lemma cbox-diff:
  shows {0::real..1} − {0,1} = box 0 1
  by (auto simp add: cbox-def)

lemma sum-bij:
```

assumes *bij F*
$\forall x{\in}s.\ f\ x = g\ (F\ x)$
shows $\bigwedge t.\ F\ `\ s = t \Longrightarrow sum\ f\ s = sum\ g\ t$
by (*metis assms bij-betw-def bij-betw-subset subset-UNIV sum.reindex-cong*)

abbreviation *surj-on* **where**
$surj\text{-}on\ s\ f \equiv s \subseteq range\ f$

lemma *surj-on-image-vimage-eq*: $surj\text{-}on\ s\ f \Longrightarrow f\ `\ (f\ -`\ s) = s$
by *fastforce*

end
theory *Derivs*
imports *General-Utils*
begin

lemma *field-simp-has-vector-derivative* [*derivative-intros*]:
$(f\ has\text{-}field\text{-}derivative\ y)\ F \Longrightarrow (f\ has\text{-}vector\text{-}derivative\ y)\ F$
by (*simp add*: *has-real-derivative-iff-has vector-derivative*)

lemma *continuous-on-cases-empty* [*continuous-intros*]:
$[\![closed\ S;\ continuous\text{-}on\ S\ f;\ \bigwedge x.\ [\![x \in S;\ \neg\ P\ x]\!] \Longrightarrow f\ x = g\ x]\!] \Longrightarrow$
$continuous\text{-}on\ S\ (\lambda x.\ if\ P\ x\ then\ f\ x\ else\ g\ x)$
using *continuous-on-cases* [*of* - {}] **by** *force*

lemma *inj-on-cases*:
assumes $inj\text{-}on\ f\ (Collect\ P \cap S)\ inj\text{-}on\ g\ (Collect\ (Not \circ P) \cap S)$
$f\ `\ (Collect\ P \cap S) \cap g\ `\ (Collect\ (Not \circ P) \cap S) = \{\}$
shows $inj\text{-}on\ (\lambda x.\ if\ P\ x\ then\ f\ x\ else\ g\ x)\ S$
using *assms* **by** (*force simp*: *inj-on-def*)

lemma *inj-on-arccos*: $S \subseteq \{-1..1\} \Longrightarrow inj\text{-}on\ arccos\ S$
by (*metis atLeastAtMost-iff cos-arccos inj-onI subsetCE*)

lemma *has-vector-derivative-componentwise-within*:
$(f\ has\text{-}vector\text{-}derivative\ f')\ (at\ a\ within\ S) \longleftrightarrow$
$(\forall i \in Basis.\ ((\lambda x.\ f\ x \cdot i)\ has\text{-}vector\text{-}derivative\ (f' \cdot i))\ (at\ a\ within\ S))$
apply (*simp add*: *has-vector-derivative-def*)
apply (*subst has-derivative-componentwise-within*)
apply *simp*
done

lemma *has-vector-derivative-pair-within*:
fixes $f :: real \Rightarrow 'a::euclidean\text{-}space$ **and** $g :: real \Rightarrow 'b::euclidean\text{-}space$
assumes $\bigwedge u.\ u \in Basis \Longrightarrow ((\lambda x.\ f\ x \cdot u)\ has\text{-}vector\text{-}derivative\ f' \cdot u)\ (at\ x\ within\ S)$
$\bigwedge u.\ u \in Basis \Longrightarrow ((\lambda x.\ g\ x \cdot u)\ has\text{-}vector\text{-}derivative\ g' \cdot u)\ (at\ x\ within\ S)$
shows $((\lambda x.\ (f\ x,\ g\ x))\ has\text{-}vector\text{-}derivative\ (f',g'))\ (at\ x\ within\ S)$
apply (*subst has-vector-derivative-componentwise-within*)

apply (*auto simp*: *assms Basis-prod-def*)
done

lemma *piecewise-C1-differentiable-const*:
shows (λ*x*. *c*) *piecewise-C1-differentiable-on s*
using *continuous-on-const*
by (*auto simp add*: *piecewise-C1-differentiable-on-def*)

declare *piecewise-C1-differentiable-const* [*simp*, *derivative-intros*]
declare *piecewise-C1-differentiable-neg* [*simp*, *derivative-intros*]
declare *piecewise-C1-differentiable-add* [*simp*, *derivative-intros*]
declare *piecewise-C1-differentiable-diff* [*simp*, *derivative-intros*]

lemma *piecewise-C1-differentiable-on-ident* [*simp*, *derivative-intros*]:
fixes *f* :: *real* ⇒ *'a*::*real-normed-vector*
shows (λ*x*. *x*) *piecewise-C1-differentiable-on S*
unfolding *piecewise-C1-differentiable-on-def* **using** *C1-differentiable-on-ident*
by (*blast intro*: *continuous-on-id C1-differentiable-on-ident*)

lemma *piecewise-C1-differentiable-on-mult* [*simp*, *derivative-intros*]:
fixes *f* :: *real* ⇒ *'a*::*real-normed-algebra*
assumes *f piecewise-C1-differentiable-on S g piecewise-C1-differentiable-on S*
shows (λ*x*. *f x* ∗ *g x*) *piecewise-C1-differentiable-on S*
using *assms*
unfolding *piecewise-C1-differentiable-on-def*
apply *safe*
apply (*blast intro*: *continuous-intros*)
apply (*rename-tac A B*)
apply (*rule-tac x*=*A* ∪ *B* **in** *exI*)
apply (*auto intro*: *C1-differentiable-on-mult C1-differentiable-on-subset*)
done

lemma *C1-differentiable-on-cdiv* [*simp*, *derivative-intros*]:
fixes *f* :: *real* ⇒ *'a* :: *real-normed-field*
shows *f C1-differentiable-on S* ⟹ (λ*x*. *f x* / *c*) *C1-differentiable-on S*
by (*simp add*: *divide-inverse*)

lemma *piecewise-C1-differentiable-on-cdiv* [*simp*, *derivative-intros*]:
fixes *f* :: *real* ⇒ *'a*::*real-normed-field*
assumes *f piecewise-C1-differentiable-on S*
shows (λ*x*. *f x* / *c*) *piecewise-C1-differentiable-on S*
by (*simp add*: *divide-inverse piecewise-C1-differentiable-const piecewise-C1-differentiable-on-mult assms*)

lemma *sqrt-C1-differentiable* [*simp*, *derivative-intros*]:
assumes *f*: *f C1-differentiable-on S* **and** *fim*: *f* ‘ *S* ⊆ {*0*<..}
shows (λ*x*. *sqrt* (*f x*)) *C1-differentiable-on S*

```
proof -
  have contf: continuous-on S f
    by (simp add: C1-differentiable-imp-continuous-on f)
  show ?thesis
    using assms
     unfolding C1-differentiable-on-def has-real-derivative-iff-has-vector-derivative
[symmetric]
    by (fastforce intro!: contf continuous-intros derivative-intros)
qed

lemma sqrt-piecewise-C1-differentiable [simp, derivative-intros]:
  assumes f: f piecewise-C1-differentiable-on S and fim: f ‘ S ⊆ {0<..}
  shows (λx. sqrt (f x)) piecewise-C1-differentiable-on S
  using assms
  unfolding piecewise-C1-differentiable-on-def
  by (fastforce intro!: continuous-intros derivative-intros)

lemma
  fixes f :: real ⇒ 'a::{banach,real-normed-field}
  assumes f: f C1-differentiable-on S
 shows sin-C1-differentiable [simp, derivative-intros]: (λx. sin (f x)) C1-differentiable-on
S
 and   cos-C1-differentiable [simp, derivative-intros]: (λx. cos (f x)) C1-differentiable-on
S
proof -
  have contf: continuous-on S f
    by (simp add: C1-differentiable-imp-continuous-on f)
  note df-sin = field-vector-diff-chain-at [where g=sin, unfolded o-def]
  note df-cos = field-vector-diff-chain-at [where g=cos, unfolded o-def]
  show  (λx. sin (f x)) C1-differentiable-on S (λx. cos (f x)) C1-differentiable-on
S
    using assms
     unfolding C1-differentiable-on-def has-real-derivative-iff-has-vector-derivative
[symmetric]
    apply auto
     by (rule contf continuous-intros df-sin df-cos derivative-intros exI conjI ballI |
force)+
qed

lemma has-derivative-abs:
  fixes a::real
  assumes a ≠ 0
  shows (abs has-derivative ((*) (sgn a))) (at a)
proof -
  have [simp]: norm = abs
    using real-norm-def by force
  show ?thesis
    using has-derivative-norm [where 'a=real, simplified] assms
    by (simp add: mult-commute-abs)
```

qed

lemma *abs-C1-differentiable* [*simp*, *derivative-intros*]:
  fixes *f* :: *real* ⇒ *real*
  assumes *f*: *f C1-differentiable-on S* **and** *0* ∉ *f* ‘ *S*
  shows (λ*x*. *abs* (*f x*)) *C1-differentiable-on S*
proof −
  have *contf*: *continuous-on S f*
    by (*simp add*: *C1-differentiable-imp-continuous-on f*)
  note *df* = *DERIV-chain* [**where** *f*=*abs* **and** *g*=*f*, *unfolded o-def*]
  show *?thesis*
    using *assms*
    unfolding *C1-differentiable-on-def has-real-derivative-iff-has-vector-derivative* [*symmetric*]
    apply *clarify*
    apply (*rule df exI conjI ballI*)+
    apply (*force simp*: *has-field-derivative-def intro*: *has-derivative-abs continuous-intros contf*)+
    done
qed

lemma *C1-differentiable-on-pair* [*simp*, *derivative-intros*]:
  fixes *f* :: *real* ⇒ ′*a*::*euclidean-space* **and** *g* :: *real* ⇒ ′*b*::*euclidean-space*
  assumes *f C1-differentiable-on S g C1-differentiable-on S*
  shows (λ*x*. (*f x*, *g x*)) *C1-differentiable-on S*
  using *assms* **unfolding** *C1-differentiable-on-def*
  apply *safe*
  apply (*rename-tac A B*)
  apply (*intro exI ballI conjI*)
  apply (*rule-tac f*′=*A x* **and** *g*′=*B x* **in** *has-vector-derivative-pair-within*)
  using *has-vector-derivative-componentwise-within*
  by (*blast intro*: *continuous-on-Pair*)+

lemma *piecewise-C1-differentiable-on-pair* [*simp*, *derivative-intros*]:
  fixes *f* :: *real* ⇒ ′*a*::*euclidean-space* **and** *g* :: *real* ⇒ ′*b*::*euclidean-space*
  assumes *f piecewise-C1-differentiable-on S g piecewise-C1-differentiable-on S*
  shows (λ*x*. (*f x*, *g x*)) *piecewise-C1-differentiable-on S*
  using *assms* **unfolding** *piecewise-C1-differentiable-on-def*
  by (*blast intro*!: *continuous-intros C1-differentiable-on-pair del*: *continuous-on-discrete intro*: *C1-differentiable-on-subset*)

lemma *test2*:
  assumes *s*: ⋀*x*. *x* ∈ {*0..1*} − *s* ⟹ *g differentiable at x*
    and *fs*: *finite s* **and** *uv*: *u* ∈ {*0..1*} *v* ∈ {*0..1*} *u* ≤ *v*
    and *x* ∈ {*0..1*} *x* ∉ (λ*t*. (*v*−*u*) $*_R$ *t* + *u*) −‘ *s*
  shows *vector-derivative* (λ*x*. *g* ((*v*−*u*) ∗ *x* + *u*)) (*at x within* {*0..1*}) = (*v*−*u*) $*_R$ *vector-derivative g* (*at* ((*v*−*u*) ∗ *x* + *u*) *within*{*0..1*})
proof −
  have *i*:(*g has-vector-derivative vector-derivative g* (*at* ((*v* − *u*) ∗ *x* + *u*))) (*at*

((v−u) ∗ x + u))
  using *assms s* [*of* (v − u) ∗ x + u] *uv mult-left-le* [*of x v−u*]
  by (*auto simp*: *vector-derivative-works*)
 have *ii*:((λx. g ((v − u) ∗ x + u)) *has-vector-derivative* (v − u) $*_R$ *vector-derivative g* (*at* ((v − u) ∗ x + u))) (*at x*)
  by (*intro vector-diff-chain-at* [*simplified o-def*] *derivative-eq-intros* | *simp add*: *i*)+
 have *0*: *0* ≤ (v − u) ∗ x + u
  using *assms uv* **by** *auto*
 have *1*: (v − u) ∗ x + u ≤ *1*
  using *assms uv*
 by *simp* (*metis add.commute atLeastAtMost-iff atLeastatMost-empty-iff diff-ge-0-iff-ge empty-iff le-diff-eq mult-left-le*)
 have *iii*: *vector-derivative g* (*at* ((v − u) ∗ x + u) *within* {*0..1*}) = *vector-derivative g* (*at* ((v − u) ∗ x + u))
  using *Derivative.vector-derivative-at-within-ivl*[*OF i*, *of 0 1*, *OF 0 1*]
  by *auto*
 have *iv*: *vector-derivative* (λx. g ((v − u) ∗ x + u)) (*at x within* {*0..1*}) = (v − u) $*_R$ *vector-derivative g* (*at* ((v − u) ∗ x + u))
  using *Derivative.vector-derivative-at-within-ivl*[*OF ii*, *of 0 1*] *assms*
  by *auto*
 show *?thesis*
  using *iii iv* **by** *auto*
qed

lemma *C1-differentiable-on-components*:
 assumes ⋀i. i ∈ *Basis* ⟹ (λx. f x · i) *C1-differentiable-on s*
 shows *f C1-differentiable-on s*
proof (*clarsimp simp add*: *C1-differentiable-on-def has-vector-derivative-def*)
 have ∗:∀ f i x. x $*_R$ (f · i) = (x $*_R$ f) · i **by** *auto*
 have ∃ f′. ∀ i∈*Basis*. ∀ x∈s. ((λx. f x · i) *has-derivative* (λz. z $*_R$ f′ x · i)) (*at x*) ∧ *continuous-on s f′*
 using *assms lambda-skolem-euclidean*[*of* λi D. (∀ x∈s. ((λx. f x · i) *has-derivative* (λz. z $*_R$ D x)) (*at x*)) ∧ *continuous-on s D*]
  apply (*simp only*: *C1-differentiable-on-def has-vector-derivative-def* ∗)
  using *continuous-on-componentwise*[*of s*]
  by *metis*
 then obtain f′ **where** f′: ∀ i∈*Basis*. ∀ x∈s. ((λx. f x · i) *has-derivative* (λz. z $*_R$ f′ x · i)) (*at x*) ∧ *continuous-on s f′*
  by *auto*
 then have *0*: (∀ x∈s. (f *has-derivative* (λz. z $*_R$ f′ x)) (*at x*)) ∧ *continuous-on s f′*
  using f′ *has-derivative-componentwise-within*[*of f*, **where** S= *UNIV*]
  by *auto*
 then show ∃ D. (∀ x∈s. (f *has-derivative* (λz. z $*_R$ D x)) (*at x*)) ∧ *continuous-on s D* **by** *metis*
qed

lemma *piecewise-C1-differentiable-on-components*:

assumes *finite t*
$\bigwedge i.\ i \in Basis \Longrightarrow (\lambda x.\ f\ x \cdot i)$ *C1-differentiable-on* $s - t$
$\bigwedge i.\ i \in Basis \Longrightarrow$ *continuous-on* $s\ (\lambda x.\ f\ x \cdot i)$
shows *f piecewise-C1-differentiable-on s*
using *C1-differentiable-on-components assms continuous-on-componentwise piecewise-C1-differentiable-on-def* **by** *blast*

lemma *all-components-smooth-one-pw-smooth-is-pw-smooth*:
assumes $\bigwedge i.\ i \in Basis - \{j\} \Longrightarrow (\lambda x.\ f\ x \cdot i)$ *C1-differentiable-on s*
assumes $(\lambda x.\ f\ x \cdot j)$ *piecewise-C1-differentiable-on s*
shows *f piecewise-C1-differentiable-on s*
proof −
have *is-cont*: $\forall i \in Basis.$ *continuous-on* $s\ (\lambda x.\ f\ x \cdot i)$
using *assms C1-differentiable-imp-continuous-on piecewise-C1-differentiable-on-def*
by *fastforce*
obtain *t* **where** *t*:(*finite t* $\wedge$ $(\lambda x.\ f\ x \cdot j)$ *C1-differentiable-on* $s - t$) **using** *assms*(*2*) *piecewise-C1-differentiable-on-def* **by** *auto*
show *?thesis*
using *piecewise-C1-differentiable-on-components*[**where** *?f* = *f*]
using *assms*(*2*) *piecewise-C1-differentiable-on-def*
C1-differentiable-on-subset[*OF assms*(*1*) *Diff-subset*, **where** *?B1* =*t*] *t is-cont*
by *fastforce*
qed

lemma *derivative-component-fun-component*:
fixes *i*::*'a*::*euclidean-space*
assumes *f differentiable* (*at x*)
shows ((*vector-derivative f* (*at x*)) $\cdot$ *i*) = ((*vector-derivative* ($\lambda x.$ (*f x*) $\cdot$ *i*) (*at x*)))
proof −
have (($\lambda x.$ *f x* $\cdot$ *i*) *has-vector-derivative vector-derivative f* (*at x*) $\cdot$ *i*) (*at x*)
using *assms* **and** *bounded-linear.has-vector-derivative*[*of* ($\lambda x.$ *x* $\cdot$ *i*) *f* (*vector-derivative f* (*at x*)) (*at x*)] **and**
bounded-linear-inner-left[*of i*] **and** *vector-derivative-works*[*of f* (*at x*)]
by *blast*
then show ((*vector-derivative f* (*at x*)) $\cdot$ *i*) = ((*vector-derivative* ($\lambda x.$ (*f x*) $\cdot$ *i*) (*at x*)))
using *vector-derivative-works*[*of* ($\lambda x.$ (*f x*) $\cdot$ *i*) (*at x*)] **and**
differentiableI-vector[*of* ($\lambda x.$ (*f x*) $\cdot$ *i*) (*vector-derivative f* (*at x*) $\cdot$ *i*) (*at x*)] **and**
Derivative.vector-derivative-at
by *force*
qed

lemma *gamma-deriv-at-within*:
assumes *a-leq-b*: $a < b$ **and**
x-within-bounds: $x \in \{a..b\}$ **and**
gamma-differentiable: $\forall x \in \{a\ ..\ b\}.\ \gamma$ *differentiable at x*
shows *vector-derivative* γ (*at x within* $\{a..b\}$) = *vector-derivative* γ (*at x*)

using *Derivative.vector-derivative-at-within-ivl*[*of* γ *vector-derivative* γ (*at x*) *x a b*]
gamma-differentiable x-within-bounds a-leq-b
by (*auto simp add*: *vector-derivative-works*)

lemma *islimpt-diff-finite*:
assumes *finite* (*t*::'*a*::*t1-space set*)
shows *x islimpt s* − *t* = *x islimpt s*
proof−
have *iii*: *s* − *t* = *s* − (*t* ∩ *s*) **by** *auto*
have (*t* ∩ *s*) ⊆ *s* **by** *auto*
have *ii*:*finite* (*t* ∩ *s*) **using** *assms*(*1*) **by** *auto*
have *i*: (*t* ∩ *s*) ∪ (*s* − (*t* ∩ *s*)) = (*s*)
using *assms* **by** *auto*
then have *x islimpt s* − (*t* ∩ *s*) = *x islimpt s*
by (*metis ii islimpt-Un-finite*)
then show *?thesis* **using** *iii* **by** *auto*
qed

lemma *ivl-limpt-diff*:
assumes *finite s a* < *b* (*x*::*real*) ∈ {*a*..*b*} − *s*
shows *x islimpt* {*a*..*b*} − *s*
proof−
have *x islimpt* {*a*..*b*}
proof (*cases x* ∈{*a*,*b*})
have *i*: *finite* {*a*,*b*} **and** *ii*: {*a*, *b*} ∪ {*a*<..<*b*} = {*a*..*b*} **using** *assms* **by** *auto*
assume *x* ∈{*a*,*b*}
then show *?thesis*
by (*meson DiffE assms*(*2*) *assms*(*3*) *islimpt-Icc*)
next
assume *x* ∉{*a*,*b*}
then show *x islimpt* {*a*..*b*} **using** *assms* **by** *auto*
qed
then show *x islimpt* {*a*..*b*} − *s* **using** *islimpt-diff-finite*[*OF assms*(*1*)] *assms*
by *fastforce*
qed

lemma *ivl-closure-diff-del*:
assumes *finite s a* < *b* (*x*::*real*) ∈ {*a*..*b*} − *s*
shows *x* ∈ *closure* (({*a*..*b*} − *s*) − {*x*})
using *ivl-limpt-diff islimpt-in-closure assms* **by** *blast*

lemma *ivl-not-trivial-limit-within*:
assumes *finite s*
a < *b*
(*x*::*real*) ∈ {*a*..*b*} − *s*
shows *at x within* {*a*..*b*} − *s* ≠ *bot*
using *assms ivl-closure-diff-del not-trivial-limit-within*
by *blast*

lemma *vector-derivative-at-within-non-trivial-limit*:
  at x within s ≠ *bot* ∧ (*f has-vector-derivative f′*) (*at x*) ⟹
    vector-derivative f (*at x within s*) = *f′*
  using *has-vector-derivative-at-within vector-derivative-within* **by** *fastforce*

lemma *vector-derivative-at-within-ivl-diff*:
  finite s ∧ *a* < *b* ∧ (*x*::*real*) ∈ {*a*..*b*} − *s* ∧ (*f has-vector-derivative f′*) (*at x*) ⟹
    vector-derivative f (*at x within* {*a*..*b*} − *s*) = *f′*
  using *vector-derivative-at-within-non-trivial-limit ivl-not-trivial-limit-within* **by** *fastforce*

lemma *gamma-deriv-at-within-diff*:
  assumes *a-leq-b*: *a* < *b* **and**
    x-within-bounds: *x* ∈ {*a*..*b*} − *s* **and**
    gamma-differentiable: ∀ *x* ∈ {*a* .. *b*} − *s*. *γ differentiable at x* **and**
    s-subset: *s* ⊆ {*a*..*b*} **and**
    finite-s: *finite s*
  shows *vector-derivative γ* (*at x within* {*a*..*b*} − *s*)
        = *vector-derivative γ* (*at x*)
  using *vector-derivative-at-within-ivl-diff* [*of s a b x γ vector-derivative γ* (*at x*)]
    gamma-differentiable
    x-within-bounds a-leq-b s-subset finite-s
  by (*auto simp add*: *vector-derivative-works*)

lemma *gamma-deriv-at-within-gen*:
  assumes *a-leq-b*: *a* < *b* **and**
    x-within-bounds: *x* ∈ *s* **and**
    s-subset: *s* ⊆ {*a*..*b*} **and**
    gamma-differentiable: ∀ *x* ∈ *s*. *γ differentiable at x*
  shows *vector-derivative γ* (*at x within* ({*a*..*b*})) = *vector-derivative γ* (*at x*)
  using *Derivative.vector-derivative-at-within-ivl*[*of γ vector-derivative γ* (*at x*) *x a b*]
    gamma-differentiable x-within-bounds a-leq-b s-subset
  by (*auto simp add*: *vector-derivative-works*)

lemma *derivative-component-fun-component-at-within-gen*:
  assumes *gamma-differentiable*: ∀ *x* ∈ *s*. *γ differentiable at x* **and** *s-subset*: *s* ⊆ {*0*..*1*}
  shows ∀ *x* ∈ *s*. *vector-derivative* (λ*x*. *γ x*) (*at x within* {*0*..*1*}) • (*i*::′*a*:: *euclidean-space*)
    = *vector-derivative* (λ*x*. *γ x* • *i*) (*at x within* {*0*..*1*})
proof −
  have *gamma-i-component-smooth*:
    ∀ *x* ∈ *s*. (λ*x*. *γ x* • *i*) *differentiable at x*
    using *gamma-differentiable*
    by *auto*
  show ∀ *x* ∈ *s*. *vector-derivative* (λ*x*. *γ x*) (*at x within* {*0*..*1*}) • *i*
        = *vector-derivative* (λ*x*. *γ x* • *i*) (*at x within* {*0*..*1*})

proof
fix *x*::*real*
assume *x-within-bounds*: $x \in s$
have *gamma-deriv-at-within*:
vector-derivative ($\lambda x.\ \gamma\ x$) (*at x within* {*0..1*}) = *vector-derivative* ($\lambda x.\ \gamma$ *x*) (*at x*)
using *gamma-deriv-at-within-gen*[*of 0 1*] *x-within-bounds*
gamma-differentiable s-subset
by (*auto simp add*: *vector-derivative-works*)
then have *gamma-component-deriv-at-within*:
vector-derivative ($\lambda x.\ \gamma\ x \cdot i$) (*at x*)
= *vector-derivative* ($\lambda x.\ \gamma\ x \cdot i$) (*at x within* {*0..1*})
using *gamma-deriv-at-within-gen*[*of 0 1*, **where** $\gamma = (\lambda x.\ \gamma\ x \cdot i)$] *x-within-bounds*
gamma-i-component-smooth s-subset
by (*auto simp add*: *vector-derivative-works*)
have *gamma-component-deriv-eq-gamma-deriv-component*:
vector-derivative ($\lambda x.\ \gamma\ x \cdot i$) (*at x*) = *vector-derivative* ($\lambda x.\ \gamma\ x$) (*at x*) $\cdot$ *i*
using *derivative-component-fun-component*[*of* γ *x i*] *gamma-differentiable* *x-within-bounds*
by *auto*
show *vector-derivative* γ (*at x within* {*0..1*}) $\cdot$ *i* = *vector-derivative* ($\lambda x.\ \gamma\ x \cdot$ *i*) (*at x within* {*0..1*})
using *gamma-component-deriv-eq-gamma-deriv-component gamma-component-deriv-at-within* *gamma-deriv-at-within*
by *auto*
qed
qed

lemma *derivative-component-fun-component-at-within*:
assumes *gamma-differentiable*: $\forall x \in \{0\ ..\ 1\}.\ \gamma$ *differentiable at x*
shows $\forall x \in \{0..1\}$. *vector-derivative* ($\lambda x.\ \gamma\ x$) (*at x within* {*0..1*}) $\cdot$ (*i*::*'a*:: *euclidean-space*)
= *vector-derivative* ($\lambda x.\ \gamma\ x \cdot i$) (*at x within* {*0..1*})
proof −
have *gamma-i-component-smooth*:
$\forall x \in \{0\ ..\ 1\}$. ($\lambda x.\ \gamma\ x \cdot i$) *differentiable at x*
using *gamma-differentiable* **by** *auto*
show $\forall x \in \{0..1\}$. *vector-derivative* ($\lambda x.\ \gamma\ x$) (*at x within* {*0..1*}) $\cdot$ *i*
= *vector-derivative* ($\lambda x.\ \gamma\ x \cdot i$) (*at x within* {*0..1*})
proof
fix *x*::*real*
assume *x-within-bounds*: $x \in \{0..1\}$
have *gamma-deriv-at-within*:
vector-derivative ($\lambda x.\ \gamma\ x$) (*at x within* {*0..1*}) = *vector-derivative* ($\lambda x.\ \gamma\ x$) (*at x*)
using *gamma-deriv-at-within*[*of 0 1*] *x-within-bounds*
gamma-differentiable
by (*auto simp add*: *vector-derivative-works*)
have *gamma-component-deriv-at-within*:

vector-derivative (λ*x*. γ *x* · *i*) (*at x*) = *vector-derivative* (λ*x*. γ *x* · *i*) (*at x within* {*0..1*})
using *Derivative.vector-derivative-at-within-ivl*[*of* (λ*x*. (γ *x*) · *i*) *vector-derivative* (λ*x*. (γ *x*) · *i*) (*at x*) *x 0 1*]
has-vector-derivative-at-within[*of* (λ*x*. γ *x* · *i*) *vector-derivative* (λ*x*. γ *x* · *i*) (*at x*) *x* {*0..1*}]
gamma-i-component-smooth x-within-bounds
by (*simp add*: *vector-derivative-works*)
have *gamma-component-deriv-eq-gamma-deriv-component*:
vector-derivative (λ*x*. γ *x* · *i*) (*at x*) = *vector-derivative* (λ*x*. γ *x*) (*at x*) · *i*
using *derivative-component-fun-component*[*of* γ *x i*] *gamma-differentiable x-within-bounds*
by *auto*
show *vector-derivative* γ (*at x within* {*0..1*}) · *i* = *vector-derivative* (λ*x*. γ *x* · *i*) (*at x within* {*0..1*})
using *gamma-component-deriv-eq-gamma-deriv-component gamma-component-deriv-at-within gamma-deriv-at-within*
by *auto*
qed
qed

lemma *straight-path-diffrentiable-x*:
fixes *b* :: *real* **and** *y1* ::*real*
assumes *gamma-def*: γ = (λ*x*. (*b*, *y2* + *y1* ∗ *x*))
shows ∀ *x*. γ *differentiable at x*
unfolding *gamma-def differentiable-def*
by (*fast intro*!: *derivative-intros*)

lemma *straight-path-diffrentiable-y*:
fixes *b* :: *real* **and**
y1 y2 ::*real*
assumes *gamma-def*: γ = (λ*x*. (*y2* + *y1* ∗ *x* , *b*))
shows ∀ *x*. γ *differentiable at x*
unfolding *gamma-def differentiable-def*
by (*fast intro*!: *derivative-intros*)

lemma *piecewise-C1-differentiable-on-imp-continuous-on*:
assumes *f piecewise-C1-differentiable-on s*
shows *continuous-on s f*
using *assms*
by (*auto simp add*: *piecewise-C1-differentiable-on-def*)

lemma *boring-lemma1*:
fixes *f* :: *real*⇒*real*
assumes (*f has-vector-derivative D*) (*at x*)
shows ((λ*x*. (*f x*, *0*)) *has-vector-derivative* ((*D*,*0*::*real*))) (*at x*)
proof−
have ∗: ((λ*x*. (*f x*) $*_R$ (*1*,*0*)) *has-vector-derivative* (*D* $*_R$ (*1*,*0*))) (*at x*)
using *bounded-linear.has-vector-derivative*[*OF Real-Vector-Spaces.bounded-linear-scaleR-left*

```
assms(1),
        of (1,0)] by auto
  have ((λx. (f x) *R (1,0)) has-vector-derivative (D,0)) (at x)
  proof -
    have (D, 0::'a) = D *R (1, 0)
      by simp
    then show ?thesis
      by (metis (no-types) *)
  qed
  then show ?thesis by auto
qed

lemma boring-lemma2:
  fixes f :: real⇒real
  assumes (f has-vector-derivative D) (at x)
  shows ((λx. (0, f x)) has-vector-derivative (0, D)) (at x)
proof−
  have *: ((λx. (f x) *R (0,1)) has-vector-derivative (D *R (0,1))) (at x)
   using bounded-linear.has-vector-derivative[OF Real-Vector-Spaces.bounded-linear-scaleR-left
assms(1),
        of (0,1)] by auto
  then have ((λx. (f x) *R (0,1)) has-vector-derivative ((0,D))) (at x)
    using scaleR-Pair Real-Vector-Spaces.real-scaleR-def
  proof -
    have (0::'b, D) = D *R (0, 1)
      by auto
    then show ?thesis
      by (metis (no-types) *)
  qed
  then show ?thesis by auto
qed

lemma pair-prod-smooth-pw-smooth:
 assumes (f::real⇒real) C1-differentiable-on s (g::real⇒real) piecewise-C1-differentiable-on
s
  shows (λx. (f x, g x)) piecewise-C1-differentiable-on s
proof -
  have f-cont: continuous-on s f
    using assms(1) C1-differentiable-imp-continuous-on
    by fastforce
  have g-cont: continuous-on s g
    using assms(2)  by (auto simp add: piecewise-C1-differentiable-on-def)
  obtain t where t:(finite t ∧ g C1-differentiable-on s − t) using assms(2) piece-
wise-C1-differentiable-on-def by auto
  show ?thesis
    using piecewise-C1-differentiable-on-components[where ?f = (λx. (f x, g x))]
    apply (simp add: real-pair-basis)
    using assms(2) piecewise-C1-differentiable-on-def
      C1-differentiable-on-subset[OF assms(1) Diff-subset, where ?B1 =t] t
```

```
    f-cont g-cont
  by fastforce
qed

lemma scale-shift-smooth:
  shows (λx. a + b * x) C1-differentiable-on s
proof –
  show (λx. a + b * x) C1-differentiable-on s
   using C1-differentiable-on-mult C1-differentiable-on-add C1-differentiable-on-const
        C1-differentiable-on-ident by auto
qed

lemma open-diff:
  assumes finite (t::'a::t1-space set)
    open (s::'a set)
  shows open (s −t)
  using assms
proof(induction t)
  show open s ⟹ open (s − {}) by auto
next
  fix x::'a::t1-space
  fix F::'a::t1-space set
  assume step: finite F   x ∉ F open s
  then have i: (s − insert x F) = (s − F) − {x} by auto
  assume ind-hyp:  (open s ⟹ open (s − F))
  show open (s − insert x F)
    apply (simp only: i)
    using open-delete[of s −F] ind-hyp[OF step(3)] by auto
qed

lemma has-derivative-transform-within:
  assumes 0 < d
    and x ∈ s
    and ∀x'∈s. dist x' x < d ⟶ f x' = g x'
    and (f has-derivative f') (at x within s)
  shows (g has-derivative f') (at x within s)
  using assms
  unfolding has-derivative-within
  by (force simp add: intro: Lim-transform-within)

lemma has-derivative-transform-within-ivl:
  assumes (0::real) < b
    and     ∀x∈{a..b} −s. f x = g x
    and x ∈ {a..b} −s
    and (f has-derivative f') (at x within {a..b} − s)
  shows (g has-derivative f') (at x within {a..b} − s)
  using has-derivative-transform-within[of b x {a..b} − s] assms
  by auto
```

lemma *has-vector-derivative-transform-within-ivl*:
assumes (*0*::*real*) < *b*
and ∀ *x*∈{*a*..*b*} −*s* . *f x* = *g x*
and *x* ∈ {*a*..*b*} − *s*
and (*f has-vector-derivative f′*) (*at x within* {*a*..*b*} − *s*)
shows (*g has-vector-derivative f′*) (*at x within* {*a*..*b*} − *s*)
using *assms has-derivative-transform-within-ivl*
apply (*auto simp add*: *has-vector-derivative-def*)
by *blast*

lemma *has-derivative-transform-at*:
assumes *0* < *d*
and ∀ *x′*. *dist x′ x* < *d* ⟶ *f x′* = *g x′*
and (*f has-derivative f′*) (*at x*)
shows (*g has-derivative f′*) (*at x*)
using *has-derivative-transform-within* [*of d x UNIV f g f′*] *assms*
by *simp*

lemma *has-vector-derivative-transform-at*:
assumes *0* < *d*
and ∀ *x′*. *dist x′ x* < *d* ⟶ *f x′* = *g x′*
and (*f has-vector-derivative f′*) (*at x*)
shows (*g has-vector-derivative f′*) (*at x*)
using *assms*
unfolding *has-vector-derivative-def*
by (*rule has-derivative-transform-at*)

lemma *C1-diff-components-2*:
assumes *b* ∈ *Basis*
assumes *f C1-differentiable-on s*
shows (λ*x*. *f x* · *b*) *C1-differentiable-on s*
proof −
obtain *D* **where** *D*:(∀ *x*∈*s*. (*f has-derivative* (λ*z*. *z* $*_R$ *D x*)) (*at x*)) *continuous-on s D*
using *assms*(*2*) **by** (*fastforce simp add*: *C1-differentiable-on-def has-vector-derivative-def*)
show *?thesis*
proof (*simp add*: *C1-differentiable-on-def has-vector-derivative-def*, *intro exI conjI*)
show *continuous-on s* (λ*x*. *D x* · *b*) **using** *D continuous-on-componentwise assms*(*1*) **by** *fastforce*
show (∀ *x*∈*s*. ((λ*x*. *f x* · *b*) *has-derivative* (λ*y*. *y* ∗ (λ*x*. *D x* · *b*) *x*)) (*at x*))
using *has-derivative-inner-left D*(*1*) **by** *fastforce*
qed
qed

lemma *eq-smooth*:
assumes *0* < *d*
∀ *x*∈*s*. ∀ *y*. *dist x y* < *d* ⟶ *f y* = *g y*
f C1-differentiable-on s

shows *g C1-differentiable-on s*
proof (*simp add*: *C1-differentiable-on-def*)
obtain *D* **where** *D*: (∀ *x*∈*s*. (*f has-vector-derivative D x*) (*at x*)) ∧ *continuous-on s D*
using *assms* **by** (*auto simp add*: *C1-differentiable-on-def*)
then have *f*: (∀ *x*∈*s*. (*g has-vector-derivative D x*) (*at x*))
using *assms*(*1*−*2*)
by (*metis dist-commute has-vector-derivative-transform-at*)
have (∀ *x*∈*s*. (*g has-vector-derivative D x*) (*at x*)) ∧ *continuous-on s D* **using** *D f* **by** *auto*
then show ∃ *D*. (∀ *x*∈*s*. (*g has-vector-derivative D x*) (*at x*)) ∧ *continuous-on s D* **by** *metis*
qed

lemma *eq-pw-smooth*:
assumes *0* < *d*
∀ *x*∈*s*. ∀ *y*. *dist x y* < *d* ⟶ *f y* = *g y*
∀ *x*∈*s*. *f x* = *g x*
f piecewise-C1-differentiable-on s
shows *g piecewise-C1-differentiable-on s*
proof (*simp add*: *piecewise-C1-differentiable-on-def*)
have *g-cont*: *continuous-on s g* **using** *assms piecewise-C1-differentiable-const*
by (*simp add*: *piecewise-C1-differentiable-on-def*)
obtain *t* **where** *t*: *finite t* ∧ *f C1-differentiable-on s* − *t*
using *assms* **by** (*auto simp add*: *piecewise-C1-differentiable-on-def*)
then have *g C1-differentiable-on s* − *t* **using** *assms eq-smooth* **by** *blast*
then show *continuous-on s g* ∧ (∃ *t*. *finite t* ∧ *g C1-differentiable-on s* − *t*)
using *t g-cont* **by** *metis*
qed

lemma *scale-piecewise-C1-differentiable-on*:
assumes *f piecewise-C1-differentiable-on s*
shows (*λx*. (*c*::*real*) ∗ (*f x*)) *piecewise-C1-differentiable-on s*
proof (*simp add*: *piecewise-C1-differentiable-on-def*, *intro conjI*)
show *continuous-on s* (*λx*. *c* ∗ *f x*)
using *assms continuous-on-mult-left*
by (*auto simp add*: *piecewise-C1-differentiable-on-def*)
show ∃ *t*. *finite t* ∧ (*λx*. *c* ∗ *f x*) *C1-differentiable-on s* − *t*
using *assms continuous-on-mult-left*
by (*auto simp add*: *piecewise-C1-differentiable-on-def*)
qed

lemma *eq-smooth-gen*:
assumes *f C1-differentiable-on s*
∀ *x*. *f x* = *g x*
shows *g C1-differentiable-on s*
using *assms* **unfolding** *C1-differentiable-on-def*
by (*metis* (*no-types*, *lifting*) *has-vector-derivative-weaken UNIV-I top-greatest*)

```
lemma subpath-compose:
  shows (subpath a b γ) = γ o (λx. (b − a) ∗ x + a)
  by (auto simp add: subpath-def)

lemma subpath-smooth:
  assumes γ C1-differentiable-on {0..1} 0 ≤ a a < b b ≤ 1
  shows (subpath a b γ) C1-differentiable-on {0..1}
proof−
  have γ C1-differentiable-on {a..b}
    apply (rule C1-differentiable-on-subset)
    using assms by auto
  then have γ C1-differentiable-on (λx. (b − a) ∗ x + a) ‘ {0..1}
    using ‹a < b› closed-segment-eq-real-ivl closed-segment-real-eq by auto
  moreover have finite ({0..1} ∩ (λx. (b − a) ∗ x + a) −‘ {x}) for x
  proof −
    have ((λx. (b − a) ∗ x + a) −‘ {x}) = {(x −a) /(b−a)}
      using assms by (auto simp add: divide-simps)
    then show ?thesis
      by auto
  qed
  ultimately show ?thesis
    by (force simp add: subpath-compose intro: C1-differentiable-compose deriva-
tive-intros)
qed

lemma has-vector-derivative-divide[derivative-intros]:
  fixes a :: 'a::real-normed-field
  shows (f has-vector-derivative x) F ⟹ ((λx. f x / a) has-vector-derivative (x /
a)) F
  unfolding divide-inverse by (fact has-vector-derivative-mult-left)

end
theory Integrals
  imports HOL−Analysis.Analysis General-Utils
begin

lemma gauge-integral-Fubini-universe-x:
  fixes f :: ('a::euclidean-space ∗ 'b::euclidean-space) ⇒ 'c::euclidean-space
  assumes fun-lesbegue-integrable: integrable lborel f and
    x-axis-integral-measurable: (λx. integral UNIV (λy. f(x, y))) ∈ borel-measurable
lborel
  shows integral UNIV f = integral UNIV (λx. integral UNIV (λy. f(x,y)))
    (λx. integral UNIV (λy. f(x,y))) integrable-on UNIV
proof −
  have f-is-measurable: f ∈ borel-measurable lborel
    using fun-lesbegue-integrable and borel-measurable-integrable
    by auto
  have fun-lborel-prod-integrable:
    integrable (lborel ⨂M lborel) f
```

using *fun-lesbegue-integrable*
by (*simp add*: *lborel-prod*)
then have *region-integral-is-one-twoD-integral*:
(*LBINT x. LBINT y. f* (*x*, *y*)) = $integral^L$ (*lborel* $\bigotimes_M$ *lborel*) *f*
using *lborel-pair.integral-fst′*
by *auto*
then have *AE-one-D-integrals-eq*: *AE x in lborel.* (*LBINT y. f* (*x*, *y*)) = *integral UNIV* (*λy. f*(*x*,*y*))
proof −
have *AE x in lborel. integrable lborel* (*λy. f*(*x*,*y*))
using *lborel-pair.AE-integrable-fst′* **and** *fun-lborel-prod-integrable*
by *blast*
then show *?thesis*
using *integral-lborel* **and** *always-eventually*
and *AE-mp*
by *fastforce*
qed
have *one-D-integral-measurable*:
(*λx. LBINT y. f* (*x*, *y*)) ∈ *borel-measurable lborel*
using *f-is-measurable* **and** *lborel.borel-measurable-lebesgue-integral*
by *auto*
then have *second-lesbegue-integral-eq*:
(*LBINT x. LBINT y. f* (*x*, *y*)) = (*LBINT x. integral UNIV* (*λy. f*(*x*,*y*)))
using *x-axis-integral-measurable* **and** *integral-cong-AE* **and** *AE-one-D-integrals-eq*
by *blast*
have *integrable lborel* (*λx. LBINT y. f* (*x*, *y*))
using *fun-lborel-prod-integrable* **and** *lborel-pair.integrable-fst′*
by *auto*
then have *oneD-gauge-integral-lesbegue-integrable*:
integrable lborel (*λx. integral UNIV* (*λy. f*(*x*,*y*)))
using *x-axis-integral-measurable* **and** *AE-one-D-integrals-eq* **and** *integrable-cong-AE-imp*
by *blast*
then show *one-D-gauge-integral-integrable*:
(*λx. integral UNIV* (*λy. f*(*x*,*y*))) *integrable-on UNIV*
using *integrable-on-lborel*
by *auto*
have (*LBINT x. integral UNIV* (*λy. f*(*x*,*y*))) = *integral UNIV* (*λx. integral UNIV* (*λy. f*(*x*, *y*)))
using *integral-lborel oneD-gauge-integral-lesbegue-integrable*
by *fastforce*
then have *twoD-lesbeuge-eq-twoD-gauge*:
(*LBINT x. LBINT y. f* (*x*, *y*)) = *integral UNIV* (*λx. integral UNIV* (*λy. f*(*x*, *y*)))
using *second-lesbegue-integral-eq*
by *auto*
then show *integral UNIV f* = *integral UNIV* (*λx. integral UNIV* (*λy. f*(*x*,*y*)))
using *fun-lesbegue-integrable* **and** *integral-lborel* **and** *region-integral-is-one-twoD-integral*
by (*metis lborel-prod*)
qed

lemma *gauge-integral-Fubini-universe-y*:
fixes f :: ('a::*euclidean-space* $*$ 'b::*euclidean-space*) $\Rightarrow$ 'c::*euclidean-space*
assumes *fun-lesbegue-integrable*: *integrable lborel f* **and**
y-axis-integral-measurable: (λx. *integral UNIV* (λy. $f(y, x)$)) $\in$ *borel-measurable lborel*
shows *integral UNIV f* = *integral UNIV* (λx. *integral UNIV* (λy. $f(y, x)$))
(λx. *integral UNIV* (λy. $f(y, x)$)) *integrable-on UNIV*
proof −
have *f-is-measurable*: $f \in$ *borel-measurable lborel*
using *fun-lesbegue-integrable* **and** *borel-measurable-integrable*
by *auto*
have *fun-lborel-prod-integrable*:
integrable (*lborel* $\bigotimes_M$ *lborel*) *f*
using *fun-lesbegue-integrable*
by (*simp add*: *lborel-prod*)
then have *region-integral-is-one-twoD-integral*:
(*LBINT x. LBINT y. f* (y, x)) = *integral*^{L} (*lborel* $\bigotimes_M$ *lborel*) *f*
by (*simp add*: *lborel-pair.integrable-product-swap-iff lborel-pair.integral-fst lborel-pair.integral-product-swap*)
then have *AE-one-D-integrals-eq*: *AE x in lborel.* (*LBINT y. f* (y, x)) = *integral UNIV* (λy. $f(y,x)$)
proof −
have *AE x in lborel. integrable lborel* (λy. $f(y,x)$)
using *lborel-pair.AE-integrable-fst'* **and** *fun-lborel-prod-integrable*
lborel-pair.AE-integrable-fst lborel-pair.integrable-product-swap
by *blast*
then show *?thesis*
using *integral-lborel always-eventually AE-mp* **by** *fastforce*
qed
have *one-D-integral-measurable*:
(λx. *LBINT y. f* (y, x)) $\in$ *borel-measurable lborel*
using *f-is-measurable* **and** *lborel.borel-measurable-lebesgue-integral*
by *auto*
then have *second-lesbegue-integral-eq*:
(*LBINT x. LBINT y. f* (y, x)) = (*LBINT x. integral UNIV* (λy. $f(y, x)$))
using *y-axis-integral-measurable* **and** *integral-cong-AE* **and** *AE-one-D-integrals-eq*
by *blast*
have *integrable lborel* (λx. *LBINT y. f* (y, x))
using *fun-lborel-prod-integrable* **and** *lborel-pair.integrable-fst'*
by (*simp add*: *lborel-pair.integrable-fst lborel-pair.integrable-product-swap*)
then have *oneD-gauge-integral-lesbegue-integrable*:
integrable lborel (λx. *integral UNIV* (λy. $f(y, x)$))
using *y-axis-integral-measurable* **and** *AE-one-D-integrals-eq* **and** *integrable-cong-AE-imp*
by *blast*
then show *one-D-gauge-integral-integrable*:
(λx. *integral UNIV* (λy. $f(y, x)$)) *integrable-on UNIV*
using *integrable-on-lborel* **by** *auto*
have (*LBINT x. integral UNIV* (λy. $f(y, x)$)) = *integral UNIV* (λx. *integral UNIV* (λy. $f(y, x)$))

using *integral-lborel oneD-gauge-integral-lesbegue-integrable*
by *fastforce*
then have *twoD-lesbeuge-eq-twoD-gauge*:
(*LBINT x. LBINT y. f* (*y*, *x*)) = *integral UNIV* (λ*x. integral UNIV* (λ*y. f*(*y*, *x*)))
using *second-lesbegue-integral-eq* **by** *auto*
then show *integral UNIV f* = *integral UNIV* (λ*x. integral UNIV* (λ*y. f*(*y*, *x*)))
using *fun-lesbegue-integrable* **and** *integral-lborel* **and** *region-integral-is-one-twoD-integral*
by (*metis lborel-prod*)
qed

lemma *gauge-integral-Fubini-curve-bounded-region-x*:
fixes *f g* :: ('*a*::*euclidean-space* ∗ '*b*::*euclidean-space*) ⇒ '*c*::*euclidean-space* **and**
g1 g2:: '*a* ⇒ '*b* **and**
s:: ('*a* ∗ '*b*) *set*
assumes *fun-lesbegue-integrable*: *integrable lborel f* **and**
x-axis-gauge-integrable: ⋀*x*. (λ*y. f*(*x*, *y*)) *integrable-on UNIV* **and**

x-axis-integral-measurable: (λ*x. integral UNIV* (λ*y. f*(*x*, *y*))) ∈ *borel-measurable lborel* **and**
f-is-g-indicator: *f* = (λ*x. if x* ∈ *s then g x else 0*) **and**
s-is-bounded-by-g1-and-g2: *s* = {(*x*,*y*). (∀ *i*∈*Basis. a* · *i* ≤ *x* · *i* ∧ *x* · *i* ≤ *b* · *i*) ∧
(∀ *i*∈*Basis.* (*g1 x*) · *i* ≤ *y* · *i* ∧ *y* · *i* ≤ (*g2 x*) · *i*)}
shows *integral s g* = *integral* (*cbox a b*) (λ*x. integral* (*cbox* (*g1 x*) (*g2 x*)) (λ*y. g*(*x*,*y*)))
proof −
have *two-D-integral-to-one-D*: *integral UNIV f* = *integral UNIV* (λ*x. integral UNIV* (λ*y. f*(*x*,*y*)))
using *gauge-integral-Fubini-universe-x* **and** *fun-lesbegue-integrable* **and** *x-axis-integral-measurable*
by *auto*
have *one-d-integral-integrable*: (λ*x. integral UNIV* (λ*y. f*(*x*,*y*))) *integrable-on UNIV*
using *gauge-integral-Fubini-universe-x*(*2*) **and** *assms*
by *blast*
have *case-x-in-range*:
∀ *x* ∈ *cbox a b. integral* (*cbox* (*g1 x*) (*g2 x*)) (λ*y. g*(*x*,*y*)) = *integral UNIV* (λ*y. f*(*x*,*y*))
proof
fix *x*:: '*a*
assume *within-range*: *x* ∈ (*cbox a b*)
let *?f-one-D-spec* = (λ*y. if y* ∈ (*cbox* (*g1 x*) (*g2 x*)) *then g*(*x*,*y*) *else 0*)
have *f-one-D-region*: (λ*y. f*(*x*,*y*)) = (λ*y. if y* ∈ *cbox* (*g1 x*) (*g2 x*) *then g*(*x*,*y*) *else 0*)
proof
fix *y*::'*b*
show *f* (*x*, *y*) = (*if y* ∈ (*cbox* (*g1 x*) (*g2 x*)) *then g* (*x*, *y*) *else 0*)
using *within-range*
by (*force simp add*: *cbox-def f-is-g-indicator s-is-bounded-by-g1-and-g2*)

qed
have *zero-out-of-bound*: ∀ *y*. *y* ∉ *cbox* (*g1 x*) (*g2 x*) ⟶ *f* (*x*,*y*) = *0*
using *f-is-g-indicator* **and** *s-is-bounded-by-g1-and-g2*
by (*auto simp add*: *cbox-def*)
have (λ*y*. *f*(*x*,*y*)) *integrable-on cbox* (*g1 x*) (*g2 x*)
proof −
have *?f-one-D-spec integrable-on UNIV*
using *f-one-D-region* **and** *x-axis-gauge-integrable*
by *metis*
then have *?f-one-D-spec integrable-on cbox*(*g1 x*) (*g2 x*)
using *integrable-on-subcbox* **by** *blast*
then show *?thesis* **using** *f-one-D-region* **by** *auto*
qed
then have *f-integrale-x*: ((λ*y*. *f*(*x*,*y*)) *has-integral* (*integral* (*cbox* (*g1 x*) (*g2 x*)) (λ*y*. *f*(*x*,*y*)))) (*cbox* (*g1 x*) (*g2 x*))
using *integrable-integral* **and** *within-range* **and** *x-axis-gauge-integrable*
by *auto*
have *integral* (*cbox* (*g1 x*) (*g2 x*)) (λ*y*. *f* (*x*, *y*)) = *integral UNIV* (λ*y*. *f* (*x*, *y*))
using *has-integral-on-superset*[*OF f-integrale-x - Set.subset-UNIV*] *zero-out-of-bound*
by (*simp add*: *integral-unique*)
then have ((λ*y*. *f*(*x*, *y*)) *has-integral integral UNIV* (λ*y*. *f* (*x*, *y*))) (*cbox* (*g1 x*) (*g2 x*))
using *f-integrale-x*
by *simp*
then have ((λ*y*. *g*(*x*, *y*)) *has-integral integral UNIV* (λ*y*. *f* (*x*, *y*))) (*cbox* (*g1 x*)(*g2 x*))
by (*simp add*: *f-one-D-region*)
then show *integral* (*cbox* (*g1 x*) (*g2 x*)) (λ*y*. *g* (*x*, *y*)) = *integral UNIV* (λ*y*. *f* (*x*, *y*))
by *auto*
qed
have *case-x-not-in-range*:
∀ *x*. *x* ∉ *cbox a b* ⟶ *integral UNIV* (λ*y*. *f*(*x*,*y*)) = *0*
proof
fix *x*::′*a*
have *x* ∉ (*cbox a b*) ⟶ (∀ *y*. *f*(*x*,*y*) = *0*)
by (*auto simp add*: *s-is-bounded-by-g1-and-g2 f-is-g-indicator cbox-def*)
then show *x* ∉ *cbox a b* ⟶ *integral UNIV* (λ*y*. *f* (*x*, *y*)) = *0*
by (*simp*)
qed
have *RHS*: *integral UNIV* (λ*x*. *integral UNIV* (λ*y*. *f*(*x*,*y*))) = *integral* (*cbox a b*) (λ*x*. *integral* (*cbox* (*g1 x*) (*g2 x*)) (λ*y*. *g*(*x*,*y*)))
proof −
let *?first-integral* = (λ*x*. *integral* (*cbox* (*g1 x*) (*g2 x*)) (λ*y*. *g*(*x*,*y*)))
let *?x-integral-cases* = (λ*x*. *if x* ∈ *cbox a b then ?first-integral x else 0*)
have *x-integral-cases-integral*: (λ*x*. *integral UNIV* (λ*y*. *f*(*x*,*y*))) = *?x-integral-cases*
using *case-x-in-range* **and** *case-x-not-in-range*
by *auto*

have ((λ*x*. *integral UNIV* (λ*y*. *f*(*x*,*y*))) *has-integral* (*integral UNIV f*)) *UNIV*
using *two-D-integral-to-one-D one-d-integral-integrable* **by** *auto*
then have (*?x-integral-cases has-integral* (*integral UNIV f*)) *UNIV*
using *x-integral-cases-integral* **by** *auto*
then have (*?first-integral has-integral* (*integral UNIV f*)) (*cbox a b*)
using *has-integral-restrict-UNIV*[*of cbox a b ?first-integral integral UNIV f*]
by *auto*
then show *?thesis*
using *two-D-integral-to-one-D* **by** (*simp add*: *integral-unique*)
qed
have *f-integrable*:*f integrable-on UNIV*
using *fun-lesbegue-integrable* **and** *integrable-on-lborel*
by *auto*
then have *LHS*: *integral UNIV f* = *integral s g*
using *assms*(*4*) *integrable-integral* **by** *fastforce*
then show *?thesis*
using *RHS* **and** *two-D-integral-to-one-D*
by *auto*
qed

lemma *gauge-integral-Fubini-curve-bounded-region-y*:
fixes *f g* :: (*'a*::*euclidean-space* ∗ *'b*::*euclidean-space*) ⇒ *'c*::*euclidean-space* **and**
g1 g2:: *'b* ⇒ *'a* **and**
s:: (*'a* ∗ *'b*) *set*
assumes *fun-lesbegue-integrable*: *integrable lborel f* **and**
y-axis-gauge-integrable: ⋀*x*. (λ*y*. *f*(*y*, *x*)) *integrable-on UNIV* **and**

y-axis-integral-measurable: (λ*x*. *integral UNIV* (λ*y*. *f*(*y*, *x*))) ∈ *borel-measurable lborel* **and**
f-is-g-indicator: *f* = (λ*x*. *if x* ∈ *s then g x else 0*) **and**
s-is-bounded-by-g1-and-g2: *s* = {(*y*, *x*). (∀ *i*∈*Basis*. *a* · *i* ≤ *x* · *i* ∧ *x* · *i* ≤ *b* · *i*) ∧
(∀ *i*∈*Basis*. (*g1 x*) · *i* ≤ *y* · *i* ∧ *y* · *i* ≤ (*g2 x*) · *i*)}
shows *integral s g* = *integral* (*cbox a b*) (λ*x*. *integral* (*cbox* (*g1 x*) (*g2 x*)) (λ*y*. *g*(*y*, *x*)))
proof −
have *two-D-integral-to-one-D*: *integral UNIV f* = *integral UNIV* (λ*x*. *integral UNIV* (λ*y*. *f*(*y*, *x*)))
using *gauge-integral-Fubini-universe-y* **and** *fun-lesbegue-integrable* **and** *y-axis-integral-measurable*
by *auto*
have *one-d-integral-integrable*: (λ*x*. *integral UNIV* (λ*y*. *f*(*y*, *x*))) *integrable-on UNIV*
using *gauge-integral-Fubini-universe-y*(*2*) **and** *assms*
by *blast*
have *case-y-in-range*:
∀ *x* ∈ *cbox a b*. *integral* (*cbox* (*g1 x*) (*g2 x*)) (λ*y*. *g*(*y*, *x*)) = *integral UNIV* (λ*y*. *f*(*y*, *x*))
proof

```
    fix x:: 'b
    assume within-range: x ∈ (cbox a b)
    let ?f-one-D-spec = (λy. if y ∈ (cbox (g1 x) (g2 x)) then g(y, x) else 0)
     have f-one-D-region: (λy. f(y, x)) = (λy. if y ∈ cbox (g1 x) (g2 x) then g(y,
x) else 0)
    proof
      fix y::'a
      show f (y, x) = (if y ∈ (cbox (g1 x) (g2 x)) then g (y, x) else 0)
        using within-range
        by (force simp add: cbox-def f-is-g-indicator s-is-bounded-by-g1-and-g2)
    qed
    have zero-out-of-bound: ∀ y.  y ∉ cbox (g1 x) (g2 x) ⟶ f (y, x) = 0
      using f-is-g-indicator and s-is-bounded-by-g1-and-g2
      by (auto simp add: cbox-def)
    have (λy. f(y, x)) integrable-on cbox (g1 x)  (g2 x)
    proof −
      have ?f-one-D-spec integrable-on UNIV
        using f-one-D-region and y-axis-gauge-integrable
        by metis
      then have ?f-one-D-spec integrable-on cbox(g1 x) (g2 x)
        using integrable-on-subcbox
        by blast
      then show ?thesis using f-one-D-region  by auto
    qed
     then have f-integrale-y: ((λy. f(y, x)) has-integral (integral (cbox (g1 x) (g2
x)) (λy. f(y,x)))) (cbox (g1 x) (g2 x))
      using integrable-integral and within-range and y-axis-gauge-integrable
      by auto
     have integral (cbox (g1 x)  (g2 x)) (λy. f (y, x)) = integral UNIV (λy. f (y,
x))
     using has-integral-on-superset[OF f-integrale-y - Set.subset-UNIV] zero-out-of-bound
      by (simp add: integral-unique)
     then have ((λy. f(y, x)) has-integral integral UNIV (λy. f (y, x))) (cbox (g1
x) (g2 x))
      using f-integrale-y
      by simp
     then have ((λy. g(y, x)) has-integral integral UNIV (λy. f (y, x))) (cbox (g1
x)(g2 x))
      using f-one-D-region by fastforce
    then show integral (cbox (g1 x) (g2 x)) (λy. g (y, x)) = integral UNIV (λy. f
(y, x))
      by auto
  qed
  have case-y-not-in-range:
    ∀ x. x ∉ cbox a  b ⟶ integral UNIV (λy. f(y, x)) = 0
  proof
    fix x::'b
    have x ∉ (cbox a b) ⟶ (∀ y. f(y, x) = 0)
      apply  (simp add: s-is-bounded-by-g1-and-g2 f-is-g-indicator cbox-def)
```

```
      by auto
    then show x ∉ cbox a b ⟶ integral UNIV (λy. f (y, x)) = 0
      by (simp)
  qed
  have RHS: integral UNIV (λx. integral UNIV (λy. f(y, x))) = integral (cbox a
b) (λx. integral (cbox (g1 x) (g2 x)) (λy. g(y, x)))
  proof −
    let ?first-integral = (λx. integral (cbox (g1 x) (g2 x)) (λy. g(y, x)))
    let ?x-integral-cases = (λx. if x ∈ cbox a b then  ?first-integral x else 0)
   have y-integral-cases-integral: (λx. integral UNIV (λy. f(y, x))) = ?x-integral-cases
      using case-y-in-range and case-y-not-in-range
      by auto
    have ((λx. integral UNIV (λy. f(y, x))) has-integral (integral UNIV f)) UNIV
      using two-D-integral-to-one-D
        one-d-integral-integrable
      by auto
    then have (?x-integral-cases has-integral (integral UNIV f)) UNIV
      using y-integral-cases-integral by auto
    then have (?first-integral has-integral (integral UNIV f)) (cbox a b)
      using has-integral-restrict-UNIV[of cbox a b ?first-integral integral UNIV f]
      by auto
    then show ?thesis
      using two-D-integral-to-one-D
      by (simp add: integral-unique)
  qed
  have f-integrable:f integrable-on UNIV
    using fun-lesbegue-integrable and integrable-on-lborel
    by auto
  then have LHS: integral UNIV f = integral s g
    apply (simp add: f-is-g-indicator)
    using integrable-restrict-UNIV
      integral-restrict-UNIV
    by auto
  then show ?thesis
    using RHS and two-D-integral-to-one-D
    by auto
qed

lemma gauge-integral-by-substitution:
  fixes f::(real ⇒ real) and
    g::(real ⇒ real) and
    g′::real ⇒ real and
    a::real and
    b::real
  assumes a-le-b: a ≤ b and
    ga-le-gb: g a ≤ g b and
    g′-derivative: ∀ x ∈ {a..b}. (g has-vector-derivative (g′ x)) (at x within {a..b})
and
    g′-continuous: continuous-on {a..b} g′ and
```

f-continuous: *continuous-on* $(g$ ‘ $\{a..b\})$ f
shows *integral* $\{g\ a..g\ b\}$ $(f) =$ *integral* $\{a..b\}$ $(\lambda x.\ f(g\ x) * (g'\ x))$
proof −
have $\forall x \in \{a..b\}$. $(g$ *has-real-derivative* $(g'\ x))$ (*at* x *within* $\{a..b\})$
using *has-real-derivative-iff-has-vector-derivative*[*of* g] **and** g'*-derivative*
by *auto*
then have *2*: *interval-lebesgue-integral lborel* (*ereal* (a)) (*ereal* (b)) $(\lambda x.\ g'\ x *_R f\ (g\ x))$
$=$ *interval-lebesgue-integral lborel* (*ereal* $(g\ a)$) (*ereal* $(g\ b)$) f
using *interval-integral-substitution-finite*[*of* $a\ b\ g\ g'\ f$] **and** g'*-continuous* **and** *a-le-b* **and** *f-continuous*
by *auto*
have *g-continuous*: *continuous-on* $\{a\ ..\ b\}$ g
using *Derivative.differentiable-imp-continuous-on*
apply (*simp add*: *differentiable-on-def differentiable-def*)
by (*metis continuous-on-vector-derivative* g'*-derivative*)
have *set-integrable lborel* $\{a\ ..\ b\}$ $(\lambda x.\ g'\ x *_R f\ (g\ x))$
proof −
have *continuous-on* $\{a\ ..\ b\}$ $(\lambda x.\ g'\ x *_R f\ (g\ x))$
proof −
have *continuous-on* $\{a\ ..\ b\}$ $(\lambda x.\ f\ (g\ x))$
proof −
show *?thesis*
using *Topological-Spaces.continuous-on-compose f-continuous g-continuous*
by *auto*
qed
then show *?thesis*
using *Limits.continuous-on-mult* g'*-continuous*
by *auto*
qed
then show *?thesis*
using *borel-integrable-atLeastAtMost*$'$ **by** *blast*
qed
then have *0*: *interval-lebesgue-integral lborel* (*ereal* (a)) (*ereal* (b)) $(\lambda x.\ g'\ x *_R f\ (g\ x))$
$=$ *integral* $\{a\ ..\ b\}$ $(\lambda x.\ g'\ x *_R f\ (g\ x))$
using *a-le-b* **and** *interval-integral-eq-integral*
by (*metis* (*no-types*))
have *set-integrable lborel* $\{g\ a\ ..\ g\ b\}$ f
proof −
have *continuous-on* $\{g\ a\ ..\ g\ b\}$ f
proof −
have $\{g\ a\ ..\ g\ b\} \subseteq g$ ‘ $\{a\ ..\ b\}$
using *g-continuous*
by (*metis a-le-b atLeastAtMost-iff atLeastatMost-subset-iff continuous-image-closed-interval imageI order-refl*)
then show *continuous-on* $\{g\ a\ ..\ g\ b\}$ f
using *f-continuous continuous-on-subset*
by *blast*

    qed
    then show *?thesis*
      using *borel-integrable-atLeastAtMost′*
      by *blast*
  qed
  then have *1*: *interval-lebesgue-integral lborel* (*ereal* (*g a*)) (*ereal* (*g b*)) *f*
                  = *integral* {*g a* .. *g b*} *f*
    using *ga-le-gb* **and** *interval-integral-eq-integral*
    by (*metis* (*no-types*))
  then show *?thesis*
    using *0* **and** *1* **and** *2*
   by (*metis* (*no-types*, *lifting*) *Henstock-Kurzweil-Integration.integral-cong mult.commute real-scaleR-def*)
qed

lemma *frontier-ic*:
  assumes *a* < (*b*::*real*)
  shows *frontier* {*a*<..*b*} = {*a*,*b*}
  apply(*simp add*: *frontier-def*)
  using *assms*
  by *auto*

lemma *frontier-ci*:
  assumes *a* < (*b*::*real*)
  shows *frontier* {*a*<..<*b*} = {*a*,*b*}
  apply(*simp add*: *frontier-def*)
  using *assms*
  by *auto*

lemma *ic-not-closed*:
  assumes *a* < (*b*::*real*)
  shows ¬ *closed* {*a*<..*b*}
  using *assms frontier-subset-eq frontier-ic greaterThanAtMost-iff* **by** *blast*

lemma *closure-ic-union-ci*:
  assumes *a* < (*b*::*real*) *b* < *c*
  shows *closure* ({*a*..<*b*} ∪ {*b*<..*c*}) = {*a* .. *c*}
  using *frontier-ic*[*OF assms*(*1*)] *frontier-ci*[*OF assms*(*2*)] *closure-Un assms*
  apply(*simp add*: *frontier-def*)
  by *auto*

lemma *interior-ic-ci-union*:
  assumes *a* < (*b*::*real*) *b* < *c*
  shows *b* ∉ (*interior* ({*a*..<*b*} ∪ {*b*<..*c*}))
proof−
  have *b* ∉ ({*a*..<*b*} ∪ {*b*<..*c*}) **by** *auto*
  then show *?thesis*
    using *interior-subset* **by** *blast*
qed

lemma *frontier-ic-union-ci*:
  assumes $a < (b::real)$ $b < c$
  shows $b \in$ *frontier* $(\{a..<b\} \cup \{b<..c\})$
  using *closure-ic-union-ci assms interior-ic-ci-union*
  by(*simp add*: *frontier-def*)

lemma *ic-union-ci-not-closed*:
  assumes $a < (b::real)$ $b < c$
  shows $\neg$ *closed* $(\{a..<b\} \cup \{b<..c\})$
proof−
  have $b \notin (\{a..<b\} \cup \{b<..c\})$ **by** *auto*
  then show *?thesis*
    using *assms frontier-subset-eq frontier-ic-union-ci*[*OF assms*]
    by (*auto simp only*: *subset-iff*)
qed

lemma *integrable-continuous-*:
  fixes $f :: 'b::euclidean\text{-}space \Rightarrow 'a::banach$
  assumes *continuous-on* (*cbox a b*) *f*
  shows *f integrable-on cbox a b*
  by (*simp add*: *assms integrable-continuous*)

lemma *removing-singletons-from-div*:
  assumes $\forall t \in S.\ \exists c\ d::real.\ c < d \wedge \{c..d\} = t$
    $\{x\} \cup \bigcup S = \{a..b\}$ $a < x$ $x < b$
    finite S
  shows $\exists t \in S.\ x \in t$
proof(*rule ccontr*)
  assume $\neg(\exists t \in S.\ x \in t)$
  then have $\forall t \in S.\ x \notin t$ **by** *auto*
  then have $x \notin \bigcup S$ **by** *auto*
  then have *i*: $\bigcup S = \{a..b\} - \{x\}$ **using** *assms* (*2*) **by** *auto*
  have $x \in \{a..b\}$ **using** *assms* **by** *auto*
  then have $\{a\ ..\ b\} - \{x\} = \{a..<x\} \cup \{x<..b\}$ **by** *auto*
  then have *0*: $\bigcup S = \{a..<x\} \cup \{x<..b\}$ **using** *i* **by** *auto*
  have *1*:*closed* $(\bigcup S)$
    apply(*rule closed-Union*)
  proof−
    show *finite S*
      using *assms* **by** *auto*
    show $\forall T \in S.$ *closed T* **using** *assms* **by** *auto*
  qed
  show *False* **using** *0 1 ic-union-ci-not-closed assms* **by** *auto*
qed

lemma *remove-singleton-from-division-of*:
  assumes *A division-of* $\{a::real..b\}$ $a < b$
  assumes $x \in \{a..b\}$

```
  shows  ∃ c d. c < d ∧ {c..d} ∈ A ∧ x ∈ {c..d}
proof −
  from assms have x islimpt {a..b}
    by (intro connected-imp-perfect) auto
  also have {a..b} = {x. {x..x} ∈ A} ∪ ({a..b} − {x. {x..x} ∈ A})
    using assms by auto
  also have x islimpt ... ⟷ x islimpt {a..b} − {x. {x..x} ∈ A}
  proof (intro islimpt-Un-finite)
    have {x. {x..x} ∈ A} ⊆ Inf ` A
    proof safe
      fix x assume {x..x} ∈ A
      thus x ∈ Inf ` A
        by (auto intro!: bexI[of - {x}] simp: image-iff)
    qed
    moreover from assms have finite A by (auto simp: division-of-def)
    hence finite (Inf ` A) by auto
    ultimately show finite {x. {x..x} ∈ A} by (rule finite-subset)
  qed
  also have {a..b} = ⋃ A
    using assms by (auto simp: division-of-def)
  finally have x islimpt ⋃(A − range (λx. {x..x}))
    by (rule islimpt-subset) auto
  moreover have closed (⋃(A − range (λx. {x..x})))
    using assms by (intro closed-Union) auto
  ultimately have x ∈ (⋃(A − range (λx. {x..x})))
    by (auto simp: closed-limpt)
  then obtain X where x ∈ X X ∈ A X ∉ range (λx. {x..x})
    by blast
 moreover from division-ofD(2)[OF assms(1) this(2)] division-ofD(3)[OF assms(1)
this(2)]
    division-ofD(4)[OF assms(1) this(2)]
  obtain c d where X = cbox c d X ⊆ {a..b} X ≠ {} by blast
  ultimately have c ≤ d by auto
  have c ≠ d
  proof
    assume c = d
    with ‹X = cbox c d› have X = {c..c} by auto
    hence X ∈ range (λx. {x..x}) by blast
    with ‹X ∉ range (λx. {x..x})› show False by contradiction
  qed
  with ‹c ≤ d› have c < d by simp
  with ‹X = cbox c d› and ‹x ∈ X› and ‹X ∈ A› show ?thesis
    by auto
qed

lemma remove-singleton-from-tagged-division-of:
  assumes A tagged-division-of {a::real..b} a < b
  assumes x ∈ {a..b}
  shows  ∃ k c d. c < d ∧ (k, {c..d}) ∈ A ∧ x ∈ {c..d}
```

```
 using remove-singleton-from-division-of[OF division-of-tagged-division[OF assms(1)]
assms(2)]
  using assms(3) by fastforce

lemma tagged-div-wo-singlestons:
  assumes p tagged-division-of {a::real..b} a < b
  shows (p − {xk. ∃ x y. xk = (x,{y})}) tagged-division-of cbox a b
  using remove-singleton-from-tagged-division-of[OF assms] assms
  apply(auto simp add: tagged-division-of-def tagged-partial-division-of-def)
     apply blast
    apply blast
   apply blast
  by fastforce

lemma tagged-div-wo-empty:
  assumes p tagged-division-of {a::real..b} a < b
  shows (p − {xk. ∃ x. xk = (x,{})}) tagged-division-of cbox a b
  using remove-singleton-from-tagged-division-of[OF assms] assms
  apply(auto simp add: tagged-division-of-def tagged-partial-division-of-def)
     apply blast
    apply blast
   apply blast
  by fastforce

lemma fine-diff:
  assumes γ fine p
  shows γ fine (p − s)
  apply (auto simp add: fine-def)
  using assms by auto

lemma tagged-div-tage-notin-set:
  assumes finite (s::real set)
    p tagged-division-of {a..b}
    γ fine p (∀ (x, K)∈p. ∃ c d::real. c < d ∧ K = {c..d}) gauge γ
  shows ∃ p′ γ′. p′ tagged-division-of {a..b} ∧
              γ′ fine p′ ∧ (∀ (x, K)∈p′. x ∉ s) ∧ gauge γ′
proof−
  have (∀ (x::real, K)∈p. ∃ x′. x′ ∉ s ∧ x′∈ interior K)
  proof−
    {fix x::real
      fix K
      assume ass: (x::real,K) ∈ p
      have (∀ (x, K)∈p. infinite (interior K))
        using assms(4) infinite-Ioo interior-atLeastAtMost-real
        by (smt (verit) split-beta)
      then have i: infinite (interior K) using ass by auto
      have ∃ x′. x′ ∉ s ∧ x′∈ interior K
      using infinite-imp-nonempty[OF Diff-infinite-finite[OF assms(1) i]] by auto}
    then show ?thesis by auto
```

```
  qed
  then obtain f where f: (∀(x::real, K)∈p. (f (x,K)) ∉ s ∧ (f (x,K)) ∈ interior
K)
     using choice-iff[where ?Q = λ(x,K) x'. (x::real, K)∈p ⟶ x' ∉ s ∧ x' ∈
interior K]
    apply (auto simp add: case-prod-beta)
    by metis
  have f': (∀(x::real, K)∈p. (f (x,K)) ∉ s ∧ (f (x,K)) ∈ K)
    using f interior-subset
    by (auto simp add: case-prod-beta subset-iff)
  let ?p' = {m. (∃xK. m = ((f xK), snd xK) ∧ xK ∈ p)}
  have 0: (∀(x, K)∈?p'. x ∉ s)
    using f
    by (auto simp add: case-prod-beta)
  have i: finite {(f (a, b), b) |a b. (a, b) ∈ p}
  proof−
    have a: {(f (a, b), b) |a b. (a, b) ∈ p} = (%(a,b). (f(a,b),b)) ` p  by auto
    have b: finite p using assms(2) by auto
    show ?thesis using a b by auto
  qed
  have 1: ?p' tagged-division-of {a..b}
    using assms(2) f'
   apply(auto simp add: tagged-division-of-def tagged-partial-division-of-def case-prod-beta)
       apply(metis i)
      apply blast
     apply blast
    by fastforce

  have f-inj: inj-on f p
    unfolding inj-on-def
  proof (intro strip)
    fix x y
    assume x ∈ p y ∈ p f x = f y
    then show x = y
      using f tagged-division-ofD(5)[OF assms(2)]
      by (smt (verit, del-insts) IntI case-prodE empty-iff)
  qed
  let ?γ' = λx. if (∃xK ∈ p. f xK  = x) then (γ o fst o  the-inv-into p f) x else γ
x
  have 2: ?γ' fine ?p' using assms(3)
    by (force simp add: fine-def case-prod-beta the-inv-into-f-f[OF f-inj])
  have 3: gauge ?γ'
    using assms(5) assms(3) f'
    by (force simp add: fine-def gauge-def case-prod-beta the-inv-into-f-f[OF f-inj])
  have ?p' tagged-division-of {a..b} ∧ ?γ' fine ?p' ∧ (∀(x, K)∈?p'. x ∉ s) ∧ gauge
?γ'
    using 0 1 2 3 by auto
  then show ?thesis by meson
qed
```

lemma *has-integral-bound-spike-finite*:
  fixes *f* :: *'a::euclidean-space* ⇒ *'b::real-normed-vector*
  assumes $0 \le B$ **and** *finite S*
    and *f*: (*f has-integral i*) (*cbox a b*)
    and *leB*: $\bigwedge x.\ x \in cbox\ a\ b - S \Longrightarrow norm\ (f\ x) \le B$
  shows $norm\ i \le B * content\ (cbox\ a\ b)$
proof −
  define *g* **where** $g \equiv (\lambda x.\ if\ x \in S\ then\ 0\ else\ f\ x)$
  then have $\bigwedge x.\ x \in cbox\ a\ b - S \Longrightarrow norm\ (g\ x) \le B$
    using *leB* **by** *simp*
  moreover have (*g has-integral i*) (*cbox a b*)
    using *has-integral-spike-finite* [*OF* ‹*finite S*› - *f*]
    by (*simp add*: *g-def*)
  ultimately show *?thesis*
    by (*simp add*: ‹$0 \le B$› *g-def has-integral-bound*)
qed

lemma *has-integral-bound-*:
  fixes *f* :: *real* ⇒ *'a::real-normed-vector*
  assumes $a < b$
    and $0 \le B$
    and *f*: (*f has-integral i*) (*cbox a b*)
    and *finite s*
    and $\forall x \in (cbox\ a\ b) - s.\ norm\ (f\ x) \le B$
  shows $norm\ i \le B * content\ (cbox\ a\ b)$
  using *has-integral-bound-spike-finite assms* **by** *blast*

corollary *has-integral-bound-real'*:
  fixes *f* :: *real* ⇒ *'b::real-normed-vector*
  assumes $0 \le B$
    and *f*: (*f has-integral i*) (*cbox a b*)
    and *finite s*
    and $\forall x \in (cbox\ a\ b) - s.\ norm\ (f\ x) \le B$
  shows $norm\ i \le B * content\ \{a..b\}$
  by (*metis assms*(*1*) *assms*(*3*) *assms*(*4*) *box-real*(*2*) *f has-integral-bound-spike-finite*)

lemma *integral-has-vector-derivative-continuous-at'*:
  fixes *f* :: *real* ⇒ *'a::banach*
  assumes *finite s*
    and *f*: *f integrable-on* {*a..b*}
    and *x*: $x \in \{a..b\} - s$
    and *fx*: *continuous* (*at x within* ({*a..b*} − *s*)) *f*
  shows ((λ*u*. *integral* {*a..u*} *f*) *has-vector-derivative f x*) (*at x within* ({*a..b*} − *s*))
proof −
  let *?I* = λ*a b*. *integral* {*a..b*} *f*
  { **fix** *e::real*
    assume $e > 0$

```
  obtain d where d>0 and d: ⋀x'. ⟦x' ∈ {a..b} − s; |x' − x| < d⟧ ⟹ norm(f x' − f x) ≤ e
    using ‹e>0› fx by (auto simp: continuous-within-eps-delta dist-norm less-imp-le)
    have norm (integral {a..y} f − integral {a..x} f − (y−x) *R f x) ≤ e * |y − x|
      if y: y ∈ {a..b} − s and yx: |y − x| < d for y
    proof (cases y < x)
      case False
      have f integrable-on {a..y}
        using f y by (simp add: integrable-subinterval-real)
      then have Idiff: ?I a y − ?I a x = ?I x y
      using False x by (simp add: algebra-simps Henstock-Kurzweil-Integration.integral-combine)
      have fux-int: ((λu. f u − f x) has-integral integral {x..y} f − (y−x) *R f x) {x..y}
        apply (rule has-integral-diff)
      using x y apply (auto intro: integrable-integral [OF integrable-subinterval-real [OF f]])
        using has-integral-const-real [of f x x y] False
        apply simp
        done
      show ?thesis
        using False
        apply (simp add: abs-eq-content del: content-real-if measure-lborel-Icc)
        apply (rule has-integral-bound-real'[where f=(λu. f u − f x)])
        using yx False d x y ‹e>0› apply (auto simp add: Idiff fux-int)
      proof−
        let ?M48= mset-set s
        show ⋀z. y − x < d ⟹ (⋀x'. a ≤ x' ∧ x' ≤ b ∧ x' ∉ s ⟹ |x' − x| < d ⟹ norm (f x' − f x) ≤ e) ⟹ 0 < e ⟹ z ∉# ?M48 ⟹ a ≤ x ⟹ x ∉ s ⟹ y ≤ b ⟹ y ∉ s ⟹ x ≤ z ⟹ z ≤ y ⟹ norm (f z − f x) ≤ e
          using assms by auto
      qed
    next
      case True
      have f integrable-on {a..x}
        using f x by (simp add: integrable-subinterval-real)
      then have Idiff: ?I a x − ?I a y = ?I y x
      using True x y by (simp add: algebra-simps Henstock-Kurzweil-Integration.integral-combine)
      have fux-int: ((λu. f u − f x) has-integral integral {y..x} f − (x − y) *R f x) {y..x}
        apply (rule has-integral-diff)
      using x y apply (auto intro: integrable-integral [OF integrable-subinterval-real [OF f]])
        using has-integral-const-real [of f x y x] True
        by simp
      have norm (integral {a..x} f − integral {a..y} f − (x − y) *R f x) ≤ e * |y − x|
        using True
        apply (simp add: abs-eq-content del: content-real-if measure-lborel-Icc)
```

apply (*rule has-integral-bound-real′*[**where** $f=(\lambda u.\ f\ u - f\ x)$])
using *yx True d x y* ‹$e>0$› **apply** (*auto simp add*: *Idiff fux-int*)
proof−
let *?M44*= *mset-set s*
show $\bigwedge xa.\ x - y < d \Longrightarrow y < x \Longrightarrow (\bigwedge x'.\ a \le x' \wedge x' \le b \wedge x' \notin s \Longrightarrow |x' - x| < d \Longrightarrow norm\ (f\ x' - f\ x) \le e) \Longrightarrow 0 < e \Longrightarrow xa \notin\# \ ?M44 \Longrightarrow x \le b \Longrightarrow x \notin s \Longrightarrow a \le y \Longrightarrow y \notin s \Longrightarrow y \le xa \Longrightarrow xa \le x \Longrightarrow norm\ (f\ xa - f\ x) \le e$
using *assms* **by** *auto*
qed
then show *?thesis*
by (*simp add*: *algebra-simps norm-minus-commute*)
qed
then have $\exists\ d>0.\ \forall\ y\in\{a..b\} - s.\ |y - x| < d \longrightarrow norm\ (integral\ \{a..y\}\ f - integral\ \{a..x\}\ f - (y-x) *_R f\ x) \le e * |y - x|$
using ‹$d>0$› **by** *blast*
}
then show *?thesis*
by (*simp add*: *has-vector-derivative-def has-derivative-within-alt bounded-linear-scaleR-left*)
qed

lemma *at-within-closed-interval-finite*:
fixes *x*::*real*
assumes $a < x$ $x < b$ $x \notin S$ *finite S*
shows $(at\ x\ within\ \{a..b\} - S) = at\ x$
proof −
have $interior\ (\{a..b\} - S) = \{a<..<b\} - S$
using ‹*finite S*›
by (*simp add*: *interior-diff finite-imp-closed*)
then show *?thesis*
using *at-within-interior assms* **by** *fastforce*
qed

lemma *fundamental-theorem-of-calculus-interior-stronger′*:
fixes f :: *real* ⇒ ′*a*::*banach*
assumes *finite S*
and $a \le b$ $\bigwedge x.\ x \in \{a <..< b\} - S \Longrightarrow (f\ has\text{-}vector\text{-}derivative\ f'(x))\ (at\ x\ within\ \{a..b\} - S)$
and *continuous-on* $\{a .. b\}$ f
shows $(f'$ *has-integral* $(f\ b - f\ a))\ \{a .. b\}$
using *assms fundamental-theorem-of-calculus-interior-strong at-within-cbox-finite*
by (*metis DiffD1 DiffD2 interior-atLeastAtMost-real interior-cbox interval-cbox*)

lemma *has-integral-substitution-general-*:
fixes f :: *real* ⇒ ′*a*::*euclidean-space* **and** g :: *real* ⇒ *real*
assumes *s*: *finite s* **and** *le*: $a \le b$
and *subset*: g ‘ $\{a..b\} \subseteq \{c..d\}$
and *f*: *f integrable-on* $\{c..d\}$ *continuous-on* $(\{c..d\} - (g$ ‘ $s))$ f
and g : *continuous-on* $\{a..b\}$ g *inj-on* g $(\{a..b\} \cup s)$
and *deriv* [*derivative-intros*]:

$\bigwedge x.\ x \in \{a..b\} - s \Longrightarrow$ (*g has-field-derivative g′ x*) (*at x within* $\{a..b\}$)
shows (($\lambda x.\ g'\ x *_R f\ (g\ x)$) *has-integral* (*integral* $\{g\ a..g\ b\}$ *f* − *integral* $\{g\ b..g\ a\}$ *f*)) $\{a..b\}$
proof −
let *?F* = $\lambda x.$ *integral* $\{c..g\ x\}$ *f*
have *cont-int*: *continuous-on* $\{a..b\}$ *?F*
by (*rule continuous-on-compose2*[*OF - g*(*1*) *subset*] *indefinite-integral-continuous-1 f*)+
have *deriv*: $\bigwedge x.\ x \in \{a..b\} - s \Longrightarrow$ ((($\lambda x.$ *integral* $\{c..x\}$ *f*) ∘ *g*) *has-vector-derivative* $g'\ x *_R f\ (g\ x)$)
(*at x within* ($\{a..b\} - s$))
apply (*rule has-vector-derivative-eq-rhs*)
apply (*rule vector-diff-chain-within*)
apply (*subst has-real-derivative-iff-has-vector-derivative* [*symmetric*])
proof−
fix *x*::*real*
assume *ass*: $x \in \{a..b\} - s$
let *?f′3* = *g′ x*
have *i*: $\{a..b\} - s \subseteq \{a..b\}$ **by** *auto*
have *ii*: (*g has-vector-derivative g′ x*) (*at x within* $\{a..b\}$) **using** *deriv*[*OF ass*]
by (*simp only*: *has-real-derivative-iff-has-vector-derivative*)
show (*g has-real-derivative ?f′3*) (*at x within* $\{a..b\} - s$)
using *has-vector-derivative-within-subset*[*OF ii i*]
by (*simp only*: *has-real-derivative-iff-has-vector-derivative*)
next
let *?g′3* = *f o g*
show $\bigwedge x.\ x \in \{a..b\} - s \Longrightarrow$ (($\lambda x.$ *integral* $\{c..x\}$ *f*) *has-vector-derivative ?g′3 x*) (*at* (*g x*) *within g* ‘ ($\{a..b\} - s$))
proof−
fix *x*::*real*
assume *ass*: $x \in \{a..b\} - s$
have *finite* (*g* ‘ *s*) **using** *s* **by** *auto*
then have *i*: (($\lambda x.$ *integral* $\{c..x\}$ *f*) *has-vector-derivative f*(*g x*)) (*at* (*g x*) *within* ($\{c..d\}$ − *g* ‘ *s*))
proof (*rule integral-has-vector-derivative-continuous-at′*)
show *f integrable-on* $\{c..d\}$ **using** *f* **by** *auto*
show *g x* ∈ $\{c..d\}$ − *g* ‘ *s* **using** *ass subset*
by (*smt* (*verit*) *Diff-iff g*(*2*) *inf-sup-ord*(*4*) *inj-on-image-mem-iff subsetD sup-ge1*)
show *continuous* (*at* (*g x*) *within* $\{c..d\}$ − *g* ‘ *s*) *f*
using ‹*g x* ∈ $\{c..d\}$ − *g* ‘ *s*› *continuous-on-eq-continuous-within f*(*2*) **by** *blast*
qed
have *ii*: *g* ‘ ($\{a..b\} - s$) ⊆ ($\{c..d\}$ − *g* ‘ *s*)
using *subset g*(*2*)
by (*simp add*: *image-subset-iff inj-on-image-mem-iff*)
then show (($\lambda x.$ *integral* $\{c..x\}$ *f*) *has-vector-derivative ?g′3 x*) (*at* (*g x*) *within g* ‘ ($\{a..b\} - s$))
using *has-vector-derivative-within-subset i* **by** *fastforce*

qed
show ⋀x. x ∈ {a..b} − s ⟹ g′ x ∗_R ?g′3 x = g′ x ∗_R f (g x) **by** *auto*
qed
have *deriv*: (*?F has-vector-derivative g′ x ∗_R f (g x)*)
(*at x within* {a..b} − s) **if** x ∈ {a<..<b} − (s) **for** x
using *deriv*[*of x*] *that* **by** (*simp add*: *at-within-Icc-at o-def*)
have ((λx. g′ x ∗_R f (g x)) *has-integral* (*?F b* − *?F a*)) {a..b}
using *cont-int*
using *fundamental-theorem-of-calculus-interior-stronger′*[*OF s le deriv*]
by *blast*
also
from *subset* **have** g x ∈ {c..d} **if** x ∈ {a..b} **for** x **using** *that* **by** *blast*
from *this*[*of a*] *this*[*of b*] *le* **have** *cd*: c ≤ g a g b ≤ d c ≤ g b g a ≤ d **by** *auto*
have *integral* {c..g b} f − *integral* {c..g a} f = *integral* {g a..g b} f − *integral* {g b..g a} f
proof *cases*
assume g a ≤ g b
note *le* = *le this*
from *cd* **have** *integral* {c..g a} f + *integral* {g a..g b} f = *integral* {c..g b} f
by (*meson Henstock-Kurzweil-Integration.integral-combine atLeastatMost-subset-iff f(1) integrable-on-subinterval le(2) order-refl*)
with *le* **show** *?thesis*
by (*cases g a = g b*) (*simp-all add*: *algebra-simps*)
next
assume *less*: ¬g a ≤ g b
then have *le*: g a ≥ g b **by** *simp*
from *cd* **have** *integral* {c..g b} f + *integral* {g b..g a} f = *integral* {c..g a} f
by (*meson Henstock-Kurzweil-Integration.integral-combine atLeastatMost-subset-iff f(1) integrable-on-subinterval le order-refl*)
with *less* **show** *?thesis*
by (*simp-all add*: *algebra-simps*)
qed
finally show *?thesis* **.**
qed

lemma *has-integral-substitution-general--*:
fixes f :: *real* ⇒ ′a::*euclidean-space* **and** g :: *real* ⇒ *real*
assumes *s*: *finite s* **and** *le*: a ≤ b **and** *s-subset*: s ⊆ {a..b}
and *subset*: g ‘ {a..b} ⊆ {c..d}
and *f*: *f integrable-on* {c..d} *continuous-on* ({c..d} − (g ‘ s)) f
and *g* : *continuous-on* {a..b} g *inj-on* g {a..b}
and *deriv* [*derivative-intros*]:
⋀x. x ∈ {a..b} − s ⟹ (*g has-field-derivative g′ x*) (*at x within* {a..b})
shows ((λx. g′ x ∗_R f (g x)) *has-integral* (*integral* {g a..g b} f − *integral* {g b..g a} f)) {a..b}
using *s-subset has-integral-substitution-general-*[*OF s le subset f g(1) - deriv*]
by (*simp add*: *g(2) sup-absorb1*)

lemma *has-integral-substitution-general-′*:

fixes *f* :: *real* ⇒ *'a::euclidean-space* **and** *g* :: *real* ⇒ *real*
assumes *s*: *finite s* **and** *le*: $a \leq b$ **and** *s'*: *finite s'*
and *subset*: *g* ‘ {*a..b*} ⊆ {*c..d*}
and *f*: *f integrable-on* {*c..d*} *continuous-on* ({*c..d*} − *s'*) *f*
and *g* : *continuous-on* {*a..b*} *g* ∀ *x*∈*s'*. *finite* (*g* −‘ {*x*}) *surj-on s' g inj-on g* ({*a..b*} ∪ ((*s* ∪ *g* −‘ *s'*)))
and *deriv* [*derivative-intros*]:
⋀*x*. *x* ∈ {*a..b*} − *s* ⟹ (*g has-field-derivative g' x*) (*at x within* {*a..b*})
shows ((λ*x*. *g' x* $*_R$ *f* (*g x*)) *has-integral* (*integral* {*g a..g b*} *f* − *integral* {*g b..g a*} *f*)) {*a..b*}
proof−
have *a*: *g* −‘ *s'* = ⋃{*t*. ∃ *x*. *t* = *g* −‘ {*x*} ∧ *x* ∈ *s'*}
using *s s'* **by** *blast*
have *finite* (⋃{*t*. ∃ *x*. *t* = *g* −‘ {*x*} ∧ *x* ∈ *s'*}) **using** *s'*
by (*metis* (*no-types*, *lifting*) ‹*g* −‘ *s'* = ⋃{*g* −‘ {*x*} |*x*. *x* ∈ *s'*}› *finite-UN-I g*(*2*) *vimage-eq-UN*)
then have *0*: *finite* (*s* ∪ (*g* −‘ *s'*))
using *a s* **by** *simp*
have *1*: *continuous-on* ({*c..d*} − *g* ‘ (*s* ∪ *g* −‘ *s'*)) *f*
using *f*(*2*) *surj-on-image-vimage-eq*
by (*metis Diff-mono Un-upper2 continuous-on-subset equalityE g*(*3*) *image-Un*)
have *2*: (⋀*x*. *x* ∈ {*a..b*} − (*s* ∪ *g* −‘ *s'*) ⟹ (*g has-real-derivative g' x*) (*at x within* {*a..b*}))
using *deriv* **by** *auto*
show *?thesis* **using** *has-integral-substitution-general-*[*OF 0 assms*(*2*) *subset f*(*1*) *1 g*(*1*) *g*(*4*) *2*]
by *auto*
qed

end
theory *Paths*
imports *Derivs General-Utils Integrals*
begin

lemma *reverse-subpaths-join*:
shows *subpath 1* (*1* / *2*) *p* +++ *subpath* (*1* / *2*) *0 p* = *reversepath p*
using *reversepath-subpath join-subpaths-middle pathfinish-subpath pathstart-subpath reversepath-joinpaths*
by (*metis* (*no-types*, *lifting*))

definition *line-integral*:: (*'a::euclidean-space* ⇒ *'a::euclidean-space*) ⇒ ((*'a*) *set*) ⇒ (*real* ⇒ *'a*) ⇒ *real* **where**
line-integral F basis g ≡ *integral* {*0* .. *1*} (λ*x*. ∑ *b*∈*basis*. (*F*(*g x*) · *b*) ∗ (*vector-derivative g* (*at x within* {*0..1*}) · *b*))

definition *line-integral-exists* **where**

line-integral-exists F basis γ ≡ (λx. ∑ b∈basis. F(γ x) · b ∗ (vector-derivative γ (at x within {0..1}) · b)) integrable-on {0..1}

lemma *line-integral-on-pair-straight-path*:
fixes *F*::(*'a*::*euclidean-space*) ⇒ *'a* **and** *g* :: *real* ⇒ *real* **and** *γ*
assumes *gamma-const*: ∀ *x*. *γ*(*x*)· *i* = *a*
and *gamma-smooth*: ∀ *x* ∈ {*0* .. *1*}. *γ differentiable at x*
shows (*line-integral F* {*i*} *γ*) = *0* (*line-integral-exists F* {*i*} *γ*)
proof (*simp add*: *line-integral-def*)
have ∗: *F* (*γ x*) · *i* ∗ (*vector-derivative γ* (*at x within* {*0*..*1*}) · *i*) = *0*
if *0* ≤ *x* ∧ *x* ≤ *1* **for** *x*
proof −
have ((*λx*. *γ*(*x*)· *i*) *has-vector-derivative 0*) (*at x*)
using *vector-derivative-const-at*[*of a x*] **and** *gamma-const*
by *auto*
then have (*vector-derivative γ* (*at x*) · *i*) = *0*
using *derivative-component-fun-component*[*of γ x i*]
and *gamma-smooth* **and** *that*
by (*simp add*: *vector-derivative-at*)
then have (*vector-derivative γ* (*at x within* {*0* .. *1*}) · *i*) = *0*
using *has-vector-derivative-at-within vector-derivative-at-within-ivl that*
by (*metis atLeastAtMost-iff gamma-smooth vector-derivative-works zero-less-one*)
then show *?thesis*
by *auto*
qed
then have ((*λx*. *F* (*γ x*) · *i* ∗ (*vector-derivative γ* (*at x within* {*0*..*1*}) · *i*)) *has-integral 0*) {*0*..*1*}
using *has-integral-is-0*[*of* {*0* .. *1*} (*λx*. *F* (*γ x*) · *i* ∗ (*vector-derivative γ* (*at x within* {*0*..*1*}) · *i*))]
by *auto*
then have ((*λx*. *F* (*γ x*) · *i* ∗ (*vector-derivative γ* (*at x within* {*0*..*1*}) · *i*)) *integrable-on* {*0*..*1*})
by *auto*
then show *line-integral-exists F* {*i*} *γ* **by** (*auto simp add*:*line-integral-exists-def*)
show *integral* {*0*..*1*} (*λx*. *F* (*γ x*) · *i* ∗ (*vector-derivative γ* (*at x within* {*0*..*1*}) · *i*)) = *0*
using ∗ *has-integral-is-0*[*of* {*0* .. *1*} (*λx*. *F* (*γ x*) · *i* ∗ (*vector-derivative γ* (*at x within* {*0*..*1*}) · *i*))]
by *auto*
qed

lemma *line-integral-on-pair-path-strong*:
fixes *F*::(*'a*::*euclidean-space*) ⇒ (*'a*) **and**
g::*real* ⇒ *'a* **and**
γ::(*real* ⇒ *'a*) **and**
i::*'a*
assumes *i-norm-1*: *norm i* = *1* **and**
g-orthogonal-to-i: ∀ *x*. *g*(*x*) · *i* = *0* **and**
gamma-is-in-terms-of-i: *γ* = (*λx*. *f*(*x*) $*_R$ *i* + *g*(*f*(*x*))) **and**

gamma-smooth: γ *piecewise-C1-differentiable-on* $\{0\ ..\ 1\}$ **and**
g-continuous-on-f: *continuous-on* (f ‘ $\{0..1\}$) g **and**
path-start-le-path-end: (*pathstart* γ) $\cdot$ i $\leq$ (*pathfinish* γ) $\cdot$ i **and**
field-i-comp-cont: *continuous-on* (*path-image* γ) ($\lambda x.$ F x $\cdot$ i)
shows *line-integral* F $\{i\}$ γ
= *integral* (*cbox* ((*pathstart* γ) $\cdot$ i) ((*pathfinish* γ) $\cdot$ i)) (λ*f-var*. (F (*f-var* $*_R$ i + g(*f-var*)) $\cdot$ i))
line-integral-exists F $\{i\}$ γ
proof (*simp add*: *line-integral-def*)
obtain s **where** *gamma-differentiable*: *finite* s ($\forall\, x \in \{0\ ..\ 1\} - s.$ γ *differentiable at* x)
using *gamma-smooth*
by (*auto simp add*: *C1-differentiable-on-eq piecewise-C1-differentiable-on-def*)
then have *gamma-i-component-smooth*: $\forall\, x \in \{0\ ..\ 1\} - s.$ ($\lambda x.$ γ x $\cdot$ i) *differentiable at* x
by *auto*
have *field-cont-on-path*: *continuous-on* (($\lambda x.$ γ x $\cdot$ i) ‘ $\{0..1\}$) (λ*f-var*. F (*f-var* $*_R$ i + g *f-var*) $\cdot$ i)
proof −
have *0*: ($\lambda x.$ γ x $\cdot$ i) = f
proof
fix x
show γ x $\cdot$ i = f x
using *g-orthogonal-to-i i-norm-1*
by (*simp only*: *gamma-is-in-terms-of-i real-inner-class.inner-add-left g-orthogonal-to-i inner-scaleR-left inner-same-Basis norm-eq-1*)
qed
show *?thesis*
unfolding *0*
apply (*rule continuous-on-compose2* [*of* - ($\lambda x.$ $F(x)$ $\cdot$ i) f ‘ $\{0..1\}$ ($\lambda x.$ x $*_R$ i + g x)]
field-i-comp-cont g-continuous-on-f field-i-comp-cont continuous-intros)+
by (*auto simp add*: *gamma-is-in-terms-of-i path-image-def*)
qed
have *path-start-le-path-end′*: γ *0* $\cdot$ i $\leq$ γ *1* $\cdot$ i **using** *path-start-le-path-end* **by** (*auto simp add*: *pathstart-def pathfinish-def*)
have *gamm-cont*: *continuous-on* $\{0..1\}$ ($\lambda a.$ γ a $\cdot$ i)
apply(*rule continuous-on-inner*)
using *gamma-smooth*
apply (*simp add*: *piecewise-C1-differentiable-on-def*)
using *continuous-on-const* **by** *auto*
then obtain c d **where** *cd*: $c \leq d$ ($\lambda a.$ γ a $\cdot$ i) ‘ $\{0..1\}$ = $\{c..d\}$
by (*meson continuous-image-closed-interval zero-le-one*)
then have *subset-cd*: ($\lambda a.$ γ a $\cdot$ i) ‘ $\{0..1\}$ $\subseteq$ $\{c..d\}$ **by** *auto*
have *field-cont-on-path-cd*:
continuous-on $\{c..d\}$ (λ*f-var*. F (*f-var* $*_R$ i + g *f-var*) $\cdot$ i)
using *field-cont-on-path cd* **by** *auto*
have *path-vector-deriv-line-integrals*:
$\forall\, x \in \{0..1\} - s.$ (($\lambda x.$ γ x $\cdot$ i) *has-vector-derivative*

vector-derivative (λ*x*. γ *x* · *i*) (*at x*))
(*at x*)
using *gamma-i-component-smooth* **and** *derivative-component-fun-component* **and**
vector-derivative-works
by *blast*
then have ∀ *x*∈{*0..1*} − *s*. ((λ*x*. γ *x* · *i*) *has-vector-derivative*
vector-derivative (λ*x*. γ *x* · *i*) (*at x within* ({*0..1*})))
(*at x within* ({*0..1*}))
using *has-vector-derivative-at-within vector-derivative-at-within-ivl*
by *fastforce*
then have *has-int*:((λ*x*. *vector-derivative* (λ*x*. γ *x* · *i*) (*at x within* {*0..1*}) $*_R$ (*F* ((γ *x* · *i*) $*_R$ *i* + *g* (γ *x* · *i*)) · *i*)) *has-integral*
integral {γ *0* · *i*..γ *1* · *i*} (λ*f-var*. *F* (*f-var* $*_R$ *i* + *g f-var*) · *i*)) {*0..1*}
using *has-integral-substitution-strong*[*OF gamma-differentiable*(*1*) *rel-simps*(*44*)
path-start-le-path-end′ subset-cd field-cont-on-path-cd gamm-cont,
of (λ*x*. *vector-derivative* (λ*x*. γ(*x*) · *i*) (*at x within* ({*0..1*})))]
gamma-is-in-terms-of-i
by (*auto simp only*: *has-real-derivative-iff-has-vector-derivative*)
then have *has-int′*:((λ*x*. (*F*(γ(*x*)) · *i*)∗(*vector-derivative* (λ*x*. γ(*x*) · *i*) (*at x within* ({*0..1*})))) *has-integral*
integral {((*pathstart* γ) · *i*)..((*pathfinish* γ) · *i*)} (λ*f-var*. *F* (*f-var* $*_R$ *i* + *g f-var*) · *i*)) {*0..1*}
using *gamma-is-in-terms-of-i i-norm-1*
apply (*auto simp add*: *pathstart-def pathfinish-def*)
apply (*simp only*: *real-inner-class.inner-add-left inner-not-same-Basis g-orthogonal-to-i inner-scaleR-left norm-eq-1*)
by (*auto simp add*: *algebra-simps*)
have *substitute*:
integral ({((*pathstart* γ) · *i*)..((*pathfinish* γ) · *i*)}) (λ*f-var*. (*F* (*f-var* $*_R$ *i* + *g*(*f-var*)) · *i*))
= *integral* ({*0..1*}) (λ*x*. (*F*(γ(*x*)) · *i*)∗(*vector-derivative* (λ*x*. γ(*x*) · *i*) (*at x within* ({*0..1*}))))
using *gamma-is-in-terms-of-i integral-unique*[*OF has-int*] *i-norm-1*
apply (*auto simp add*: *pathstart-def pathfinish-def*)
apply (*simp only*: *real-inner-class.inner-add-left inner-not-same-Basis g-orthogonal-to-i inner-scaleR-left norm-eq-1*)
by (*auto simp add*: *algebra-simps*)
have *comp-in-eq-comp-out*: ∀ *x* ∈ {*0..1*} − *s*.
(*vector-derivative* (λ*x*. γ(*x*) · *i*) (*at x within* {*0..1*}))
= (*vector-derivative* γ (*at x within* {*0..1*})) · *i*
proof
fix *x*:: *real*
assume *ass*:*x* ∈ {*0..1*} −*s*
then have *x-bounds*:*x* ∈ {*0..1*} **by** *auto*
have γ *differentiable at x* **using** *ass gamma-differentiable* **by** *auto*
then have *dotprod-in-is-out*:
vector-derivative (λ*x*. γ(*x*) · *i*) (*at x*)

= (*vector-derivative* γ (*at x*)) $\cdot$ *i*
using *derivative-component-fun-component*
by *force*
then have *0*: (*vector-derivative* γ (*at x*)) $\cdot$ *i* = (*vector-derivative* γ (*at x within* $\{0..1\}$)) $\cdot$ *i*
proof −
have (γ *has-vector-derivative vector-derivative* γ (*at x*)) (*at x*)
using ‹γ *differentiable at x*› *vector-derivative-works* **by** *blast*
moreover have $0 \leq x \wedge x \leq 1$
using *x-bounds* **by** *presburger*
ultimately show *?thesis*
by (*simp add*: *vector-derivative-at-within-ivl*)
qed
have *1*: *vector-derivative* ($\lambda x.\ \gamma(x) \cdot i$) (*at x*) = *vector-derivative* ($\lambda x.\ \gamma(x) \cdot i$) (*at x within* $\{0..1\}$)
using *path-vector-deriv-line-integrals* **and** *vector-derivative-at-within-ivl* **and** *x-bounds*
by (*metis ass atLeastAtMost-iff zero-less-one*)
show *vector-derivative* ($\lambda x.\ \gamma\ x \cdot i$) (*at x within* $\{0..1\}$) = *vector-derivative* γ (*at x within* $\{0..1\}$) $\cdot$ *i*
using *0* **and** *1* **and** *dotprod-in-is-out*
by *auto*
qed
show *integral* $\{0..1\}$ ($\lambda x.\ F\ (\gamma\ x) \cdot i * (\text{vector-derivative}\ \gamma\ (\text{at}\ x\ \text{within}\ \{0..1\}) \cdot i)$) =
integral $\{\text{pathstart}\ \gamma \cdot i..\text{pathfinish}\ \gamma \cdot i\}$ (λ*f-var*. *F* (*f-var* $*_R$ *i* + *g f-var*) $\cdot$ *i*)
using *substitute* **and** *comp-in-eq-comp-out* **and** *negligible-finite*
Henstock-Kurzweil-Integration.integral-spike
[*of s* $\{0..1\}$ ($\lambda x.\ F\ (\gamma\ x) \cdot i * (\text{vector-derivative}\ \gamma\ (\text{at}\ x\ \text{within}\ \{0..1\}) \cdot i)$)
($\lambda x.\ F\ (\gamma\ x) \cdot i *$ *vector-derivative* ($\lambda x.\ \gamma\ x \cdot i$) (*at x within* $\{0..1\}$))]
by (*metis* (*no-types*, *lifting*) *gamma-differentiable*(*1*))
have (($\lambda x.\ F\ (\gamma\ x) \cdot i * (\text{vector-derivative}\ \gamma\ (\text{at}\ x\ \text{within}\ \{0..1\}) \cdot i)$) *has-integral*
integral $\{\text{pathstart}\ \gamma \cdot i..\text{pathfinish}\ \gamma \cdot i\}$ (λ*f-var*. *F* (*f-var* $*_R$ *i* + *g f-var*) $\cdot$ *i*)) $\{0..1\}$
using *has-int′* **and** *comp-in-eq-comp-out* **and** *negligible-finite*
Henstock-Kurzweil-Integration.has-integral-spike
[*of s* $\{0..1\}$ ($\lambda x.\ F\ (\gamma\ x) \cdot i * (\text{vector-derivative}\ \gamma\ (\text{at}\ x\ \text{within}\ \{0..1\}) \cdot i)$)
($\lambda x.\ F\ (\gamma\ x) \cdot i *$ *vector-derivative* ($\lambda x.\ \gamma\ x \cdot i$) (*at x within* $\{0..1\}$))
integral $\{\text{pathstart}\ \gamma \cdot i..\text{pathfinish}\ \gamma \cdot i\}$ (λ*f-var*. *F* (*f-var* $*_R$ *i* + *g f-var*) $\cdot$ *i*)]
by (*metis* (*no-types*, *lifting*) *gamma-differentiable*(*1*))
then have ($\lambda x.\ F\ (\gamma\ x) \cdot i * (\text{vector-derivative}\ \gamma\ (\text{at}\ x\ \text{within}\ \{0..1\}) \cdot i)$) *integrable-on* $\{0..1\}$
using *integrable-on-def* **by** *auto*
then show *line-integral-exists F* $\{i\}$ γ
by (*auto simp add*: *line-integral-exists-def*)
qed

lemma *line-integral-on-pair-path*:
fixes *F*::(*'a*::*euclidean-space*) ⇒ (*'a*) **and**
g::*real* ⇒ *'a* **and**
γ::(*real* ⇒ *'a*) **and**
i::*'a*
assumes *i-norm-1*: *norm i* = *1* **and**
g-orthogonal-to-i: ∀ *x*. *g*(*x*) · *i* = *0* **and**
gamma-is-in-terms-of-i: *γ* = (*λx*. *f*(*x*) $*_R$ *i* + *g*(*f*(*x*))) **and**
gamma-smooth: *γ C1-differentiable-on* {*0* .. *1*} **and**
g-continuous-on-f: *continuous-on* (*f* ‘ {*0*..*1*}) *g* **and**
path-start-le-path-end: (*pathstart γ*) · *i* ≤ (*pathfinish γ*) · *i* **and**
field-i-comp-cont: *continuous-on* (*path-image γ*) (*λx*. *F x* · *i*)
shows (*line-integral F* {*i*} *γ*)
= *integral* (*cbox* ((*pathstart γ*) · *i*) ((*pathfinish γ*) · *i*)) (*λf-var*. (*F* (*f-var* $*_R$ *i* + *g*(*f-var*)) · *i*))
proof (*simp add*: *line-integral-def*)
have *gamma-differentiable*: ∀ *x* ∈ {*0* .. *1*}. *γ differentiable at x*
using *gamma-smooth C1-differentiable-on-eq* **by** *blast*
then have *gamma-i-component-smooth*:
∀ *x* ∈ {*0* .. *1*}. (*λx*. *γ x* · *i*) *differentiable at x*
by *auto*
have *vec-deriv-i-comp-cont*:
continuous-on {*0*..*1*} (*λx*. *vector-derivative* (*λx*. *γ x* · *i*) (*at x within* {*0*..*1*}))
proof −
have *continuous-on* {*0*..*1*} (*λx*. *vector-derivative* (*λx*. *γ x*) (*at x within* {*0*..*1*}))
using *gamma-smooth C1-differentiable-on-eq*
by (*smt C1-differentiable-on-def atLeastAtMost-iff continuous-on-eq vector-derivative-at-within-ivl*)
then have *deriv-comp-cont*:
continuous-on {*0*..*1*} (*λx*. *vector-derivative* (*λx*. *γ x*) (*at x within* {*0*..*1*}) · *i*)
by (*simp add*: *continuous-intros*)
show *?thesis*
using *derivative-component-fun-component-at-within*[*OF gamma-differentiable*, *of i*]
continuous-on-eq[*OF deriv-comp-cont*, *of* (*λx*. *vector-derivative* (*λx*. *γ x* · *i*) (*at x within* {*0*..*1*}))]
by *fastforce*
qed
have *field-cont-on-path*:
continuous-on ((*λx*. *γ x* · *i*) ‘ {*0*..*1*}) (*λf-var*. *F* (*f-var* $*_R$ *i* + *g f-var*) · *i*)
proof −
have *0*: (*λx*. *γ x* · *i*) = *f*
proof
fix *x*
show *γ x* · *i* = *f x*
using *g-orthogonal-to-i i-norm-1*
by (*simp only*: *gamma-is-in-terms-of-i real-inner-class.inner-add-left g-orthogonal-to-i inner-scaleR-left inner-same-Basis norm-eq-1*)
qed

show *?thesis*
  unfolding *0*
  apply (*rule continuous-on-compose2* [*of* - (λx. $F(x)$ $\cdot$ *i*) *f* ' { *0..1*} (λx. *x* $*_R$ *i* + *g x*)]
    field-i-comp-cont g-continuous-on-f field-i-comp-cont continuous-intros)+
  by (*auto simp add*: *gamma-is-in-terms-of-i path-image-def*)
qed
have *path-vector-deriv-line-integrals*:
  $\forall x\in\{0..1\}$. ((λx. γ *x* $\cdot$ *i*) *has-vector-derivative*
    vector-derivative (λx. γ *x* $\cdot$ *i*) (*at x*))
    (*at x*)
  using *gamma-i-component-smooth* **and** *derivative-component-fun-component* **and**
    vector-derivative-works
  by *blast*
then have $\forall x\in\{0..1\}$. ((λx. γ *x* $\cdot$ *i*) *has-vector-derivative*
    vector-derivative (λx. γ *x* $\cdot$ *i*) (*at x within* {*0..1*}))
    (*at x within* {*0..1*})
  using *has-vector-derivative-at-within vector-derivative-at-within-ivl*
  by *fastforce*
then have *substitute*:
  integral (*cbox* ((*pathstart* γ) $\cdot$ *i*) ((*pathfinish* γ) $\cdot$ *i*)) (λ*f-var*. (*F* (*f-var* $*_R$ *i* + *g*(*f-var*)) $\cdot$ *i*))
    = *integral* (*cbox 0 1*) (λx. (*F*(γ(*x*)) $\cdot$ *i*)$*$(*vector-derivative* (λx. γ(*x*) $\cdot$ *i*) (*at x within* {*0..1*})))
  using *gauge-integral-by-substitution*
    [*of 0 1* (λx. (γ *x*) $\cdot$ *i*)
      (λx. *vector-derivative* (λx. γ(*x*) $\cdot$ *i*) (*at x within* {*0..1*}))
      (λ*f-var*. (*F* (*f-var* $*_R$ *i* + *g*(*f-var*)) $\cdot$ *i*))] **and**
    path-start-le-path-end **and** *vec-deriv-i-comp-cont* **and** *field-cont-on-path* **and**
    gamma-is-in-terms-of-i i-norm-1
  apply (*auto simp add*: *pathstart-def pathfinish-def*)
  apply (*simp only*: *real-inner-class.inner-add-left inner-not-same-Basis g-orthogonal-to-i inner-scaleR-left norm-eq-1*)
  by (*auto*)

have *comp-in-eq-comp-out*: $\forall x \in \{0..1\}$.
    (*vector-derivative* (λx. γ(*x*) $\cdot$ *i*) (*at x within* {*0..1*}))
      = (*vector-derivative* γ (*at x within* {*0..1*})) $\cdot$ *i*
proof
  fix *x*:: *real*
  assume *x-bounds*: $x \in \{0..1\}$
  then have γ *differentiable at x* **using** *gamma-differentiable* **by** *auto*
  then have *dotprod-in-is-out*:
    vector-derivative (λx. γ(*x*) $\cdot$ *i*) (*at x*)
      = (*vector-derivative* γ (*at x*)) $\cdot$ *i*
    using *derivative-component-fun-component*
    by *force*
  then have *0*: (*vector-derivative* γ (*at x*)) $\cdot$ *i*

= (*vector-derivative* γ (*at x within* {*0..1*})) • *i*
using *has-vector-derivative-at-within* **and** *x-bounds* **and** *vector-derivative-at-within-ivl*
by (*smt atLeastAtMost-iff gamma-differentiable inner-commute vector-derivative-works*)
have *1*: *vector-derivative* (λx. $\gamma(x)$ • *i*) (*at x*) = *vector-derivative* (λx. $\gamma(x)$ • *i*) (*at x within* {*0..1*})
using *path-vector-deriv-line-integrals* **and** *vector-derivative-at-within-ivl* **and** *x-bounds*
by *fastforce*
show *vector-derivative* (λx. γ *x* • *i*) (*at x within* {*0..1*}) = *vector-derivative* γ (*at x within* {*0..1*}) • *i*
using *0* **and** *1* **and** *dotprod-in-is-out*
by *auto*
qed
show *integral* {*0..1*} (λx. *F* (γ *x*) • *i* ∗ (*vector-derivative* γ (*at x within* {*0..1*}) • *i*)) =
integral {*pathstart* γ • *i..pathfinish* γ • *i*} (λ*f-var*. *F* (*f-var* $*_R$ *i* + *g f-var*) • *i*)
using *substitute* **and** *comp-in-eq-comp-out* **and** *Henstock-Kurzweil-Integration.integral-cong*
[*of* {*0..1*} (λx. *F* (γ *x*) • *i* ∗ (*vector-derivative* γ (*at x within* {*0..1*}) • *i*))
(λx. *F* (γ *x*) • *i* ∗ *vector-derivative* (λx. γ *x* • *i*) (*at x within* {*0..1*}))]
by *auto*
qed

lemma *content-box-cases*:
content (*box a b*) = (*if* ∀ *i*∈*Basis*. *a*•*i* ≤ *b*•*i then prod* (λi. *b*•*i* − *a*•*i*) *Basis else 0*)
by (*simp add*: *measure-lborel-box-eq inner-diff*)

lemma *content-box-cbox*:
shows *content* (*box a b*) = *content* (*cbox a b*)
by (*simp add*: *content-box-cases content-cbox-cases*)

lemma *content-eq-0*: *content* (*box a b*) = *0* ⟷ (∃ *i*∈*Basis*. *b*•*i* ≤ *a*•*i*)
by (*auto simp*: *content-box-cases not-le intro*: *less-imp-le antisym eq-refl*)

lemma *content-pos-lt-eq*: *0* < *content* (*cbox a* (*b*::′*a*::*euclidean-space*)) ⟷ (∀ *i*∈*Basis*. *a*•*i* < *b*•*i*)
by (*auto simp add*: *content-cbox-cases less-le prod-nonneg*)

lemma *content-lt-nz*: *0* < *content* (*box a b*) ⟷ *content* (*box a b*) ≠ *0*
using *Paths.content-eq-0 zero-less-measure-iff* **by** *blast*

lemma *content-subset*: *cbox a b* ⊆ *box c d* ⟹ *content* (*cbox a b*) ≤ *content* (*box c d*)
unfolding *measure-def*
by (*intro enn2real-mono emeasure-mono*) (*auto simp*: *emeasure-lborel-cbox-eq emeasure-lborel-box-eq*)

lemma *sum-content-null*:
  assumes *content* (*box a b*) = *0*
    and *p tagged-division-of* (*box a b*)
  shows *sum* (λ(*x*,*k*). *content k* $*_R$ *f x*) *p* = (*0*::′*a*::*real-normed-vector*)
proof (*rule sum.neutral*, *rule*)
  fix *y*
  assume *y*: *y* ∈ *p*
  obtain *x k* **where** *xk*: *y* = (*x*, *k*)
    using *surj-pair*[*of y*] **by** *blast*
  note *assm* = *tagged-division-ofD*(*3*−*4*)[*OF assms*(*2*) *y*[*unfolded xk*]]
  from *this*(*2*) **obtain** *c d* **where** *k*: *k* = *cbox c d* **by** *blast*
  have (λ(*x*, *k*). *content k* $*_R$ *f x*) *y* = *content k* $*_R$ *f x*
    unfolding *xk* **by** *auto*
  also have . . . = *0*
    using *content-subset*[*OF assm*(*1*)[*unfolded k*]] *content-pos-le*[*of cbox c d*]
    unfolding *assms*(*1*) *k*
    by *auto*
  finally show (λ(*x*, *k*). *content k* $*_R$ *f x*) *y* = *0* **.**
qed

lemma *has-integral-null* [*intro*]: *content*(*box a b*) = *0* =⇒ (*f has-integral 0*) (*box a b*)
  by (*simp add*: *content-box-cbox content-eq-0-interior*)

lemma *line-integral-distrib*:
  assumes *line-integral-exists f basis g1*
    line-integral-exists f basis g2
    valid-path g1 valid-path g2
  shows *line-integral f basis* (*g1* +++ *g2*) = *line-integral f basis g1* + *line-integral f basis g2*
    line-integral-exists f basis (*g1* +++ *g2*)
proof −
  obtain *s1 s2*
    where *s1*: *finite s1* ∀ *x*∈{*0*..*1*} − *s1*. *g1 differentiable at x*
      and *s2*: *finite s2* ∀ *x*∈{*0*..*1*} − *s2*. *g2 differentiable at x*
    using *assms*
    by (*auto simp*: *valid-path-def piecewise-C1-differentiable-on-def C1-differentiable-on-eq*)
  obtain *i1 i2*
    where *1*: ((λ*x*. ∑ *b*∈*basis*. *f* (*g1 x*) · *b* ∗ (*vector-derivative g1* (*at x within* {*0*..*1*}) · *b*)) *has-integral i1*) {*0*..*1*}
      and *2*: ((λ*x*. ∑ *b*∈*basis*. *f* (*g2 x*) · *b* ∗ (*vector-derivative g2* (*at x within* {*0*..*1*}) · *b*)) *has-integral i2*) {*0*..*1*}
    using *assms*
    by (*auto simp*: *line-integral-exists-def*)
  have *i1*: ((λ*x*. *2* ∗ (∑ *b*∈*basis*. *f* (*g1* (*2* ∗ *x*)) · *b* ∗ (*vector-derivative g1* (*at* (*2* ∗ *x*) *within* {*0*..*1*}) · *b*))) *has-integral i1*) {*0*..*1*/*2*}
   and *i2*: ((λ*x*. *2* ∗ (∑ *b*∈*basis*. *f* (*g2* (*2* ∗ *x* − *1*)) · *b* ∗ (*vector-derivative g2* (*at* ((*2* ∗ *x*) − *1*) *within* {*0*..*1*}) · *b*))) *has-integral i2*) {*1*/*2*..*1*}

using *has-integral-affinity01* [*OF 1*, **where** *m= 2* **and** *c=0*, *THEN has-integral-cmul* [**where** *c=2*]]
has-integral-affinity01 [*OF 2*, **where** *m= 2* **and** *c=−1*, *THEN has-integral-cmul* [**where** *c=2*]]
by (*simp-all only*: *image-affinity-atLeastAtMost-div-diff*, *simp-all add*: *scaleR-conv-of-real mult-ac*)
have *g1*: ⟦*0* ≤ *z*; *z*∗*2* < *1*; *z*∗*2* ∉ *s1*⟧ ⟹
vector-derivative (λ*x*. *if x*∗*2* ≤ *1 then g1* (*2*∗*x*) *else g2* (*2*∗*x* − *1*)) (*at z within* {*0..1*}) =
2 $*_R$ *vector-derivative g1* (*at* (*z*∗*2*) *within* {*0..1*}) **for** *z*
proof −
have *i*:⟦*0* ≤ *z*; *z*∗*2* < *1*; *z*∗*2* ∉ *s1*⟧ ⟹
vector-derivative (λ*x*. *if x* ∗ *2* ≤ *1 then g1* (*2* ∗ *x*) *else g2* (*2* ∗ *x* − *1*)) (*at z within* {*0..1*}) = *2* $*_R$ *vector-derivative g1* (*at* (*z* ∗ *2*)) **for** *z*
proof−
have *g1-at-z*:⟦*0* ≤ *z*; *z*∗*2* < *1*; *z*∗*2* ∉ *s1*⟧ ⟹
((λ*x*. *if x*∗*2* ≤ *1 then g1* (*2*∗*x*) *else g2* (*2*∗*x* − *1*)) *has-vector-derivative*
2 $*_R$ *vector-derivative g1* (*at* (*z*∗*2*))) (*at z*) **for** *z*
apply (*rule has-vector-derivative-transform-at* [*of* |*z* − *1/2*| - (λ*x*. *g1*(*2*∗*x*))])
apply (*simp-all add*: *dist-real-def abs-if split*: *if-split-asm*)
apply (*rule vector-diff-chain-at* [*of* λ*x*. *2*∗*x 2* - *g1*, *simplified o-def*])
apply (*simp add*: *has-vector-derivative-def has-derivative-def bounded-linear-mult-left*)
using *s1*
apply (*auto simp*: *algebra-simps vector-derivative-works*)
done
assume *ass*: *0* ≤ *z z*∗*2* < *1 z*∗*2* ∉ *s1*
then have *z-ge*: *z*≤ *1* **by** *auto*
show *vector-derivative* (λ*x*. *if x* ∗ *2* ≤ *1 then g1* (*2* ∗ *x*) *else g2* (*2* ∗ *x* − *1*)) (*at z within* {*0..1*}) = *2* $*_R$ *vector-derivative g1* (*at* (*z* ∗ *2*))
using *Derivative.vector-derivative-at-within-ivl*[*OF g1-at-z*[*OF ass*] *ass*(*1*) *z-ge*]
by *auto*
qed
assume *ass*: *0* ≤ *z z*∗*2* < *1 z*∗*2* ∉ *s1*
then have (*g1 has-vector-derivative* ((*vector-derivative g1* (*at* (*z*∗*2*))))) (*at* (*z*∗*2*))
using *s1* **by** (*auto simp*: *algebra-simps vector-derivative-works*)
then have *ii*: (*vector-derivative g1* (*at* (*z*∗*2*) *within* {*0..1*})) = (*vector-derivative g1* (*at* (*z*∗*2*)))
using *Derivative.vector-derivative-at-within-ivl ass*
by *force*
show *vector-derivative* (λ*x*. *if x* ∗ *2* ≤ *1 then g1* (*2* ∗ *x*) *else g2* (*2* ∗ *x* − *1*)) (*at z within* {*0..1*}) = *2* $*_R$ *vector-derivative g1* (*at* (*z* ∗ *2*) *within* {*0..1*})
using *i*[*OF ass*] *ii*
by *auto*
qed
have *g2*: ⟦*1* < *z*∗*2*; *z* ≤ *1*; *z*∗*2* − *1* ∉ *s2*⟧ ⟹
vector-derivative (λ*x*. *if x*∗*2* ≤ *1 then g1* (*2*∗*x*) *else g2* (*2*∗*x* − *1*)) (*at z*

within {*0..1*}) =
2 $*_R$ *vector-derivative g2* (*at* (*z**2 − 1*) *within* {*0..1*}) **for** *z* **proof** −
have *i*:⟦*1* < *z***2*; *z* ≤ *1*; *z***2* − *1* ∉ *s2*⟧ ⟹
vector-derivative (λ*x*. *if x* * *2* ≤ *1 then g1* (*2* * *x*) *else g2* (*2* * *x* − *1*)) (*at z within* {*0..1*})
= *2* $*_R$ *vector-derivative g2* (*at* (*z* * *2* − *1*)) **for** *z*
proof−
have *g2-at-z*:⟦*1* < *z***2*; *z* ≤ *1*; *z***2* − *1* ∉ *s2*⟧ ⟹
((λ*x*. *if x***2* ≤ *1 then g1* (*2***x*) *else g2* (*2***x* − *1*)) *has-vector-derivative 2* $*_R$ *vector-derivative g2* (*at* (*z***2* − *1*))) (*at z*) **for** *z*
apply (*rule has-vector-derivative-transform-at* [*of* |*z* − *1/2*| - (λ*x*. *g2* (*2***x* − *1*))])
apply (*simp-all add*: *dist-real-def abs-if split*: *if-split-asm*)
apply (*rule vector-diff-chain-at* [*of* λ*x*. *2***x* − *1 2* - *g2*, *simplified o-def*])
apply (*simp add*: *has-vector-derivative-def has-derivative-def bounded-linear-mult-left*)
using *s2*
apply (*auto simp*: *algebra-simps vector-derivative-works*)
done
assume *ass*: *1* < *z***2 z* ≤ *1 z***2* − *1* ∉ *s2*
then have *z-le*: *z*≤ *1* **by** *auto*
have *z-ge*: *0* ≤ *z* **using** *ass* **by** *auto*
show *vector-derivative* (λ*x*. *if x* * *2* ≤ *1 then g1* (*2* * *x*) *else g2* (*2* * *x* − *1*)) (*at z within* {*0..1*})
= *2* $*_R$ *vector-derivative g2* (*at* (*z* * *2* − *1*))
using *Derivative.vector-derivative-at-within-ivl*[*OF g2-at-z*[*OF ass*] *z-ge z-le*]
by *auto*
qed
assume *ass*: *1* < *z***2 z* ≤ *1 z***2* − *1* ∉ *s2*
then have (*g2 has-vector-derivative* ((*vector-derivative g2* (*at* (*z***2* − *1*))))) (*at* (*z***2* − *1*))
using *s2* **by** (*auto simp*: *algebra-simps vector-derivative-works*)
then have *ii*: (*vector-derivative g2* (*at* (*z***2* − *1*) *within* {*0..1*})) = (*vector-derivative g2* (*at* (*z***2* − *1*)))
using *Derivative.vector-derivative-at-within-ivl ass*
by *force*
show *vector-derivative* (λ*x*. *if x* * *2* ≤ *1 then g1* (*2* * *x*) *else g2* (*2* * *x* − *1*)) (*at z within* {*0..1*}) = *2* $*_R$ *vector-derivative g2* (*at* (*z* * *2* − *1*) *within* {*0..1*})
using *i*[*OF ass*] *ii*
by *auto*
qed
have *lem1*: ((λ*x*. ∑ *b*∈*basis*. *f* ((*g1*+++*g2*) *x*) · *b* * (*vector-derivative* (*g1*+++*g2*) (*at x within* {*0..1*}) · *b*)) *has-integral i1*) {*0..1/2*}
apply (*rule has-integral-spike-finite* [*OF* - - *i1*, *of insert* (*1/2*) ((*)*2* −‘ *s1*)])
using *s1*
apply (*force intro*: *finite-vimageI* [**where** *h* = (*)*2*] *inj-onI*)
apply (*clarsimp simp add*: *joinpaths-def scaleR-conv-of-real mult-ac g1*)
by (*simp add*: *sum-distrib-left*)
moreover have *lem2*: ((λ*x*. ∑ *b*∈*basis*. *f* ((*g1*+++*g2*) *x*) · *b* * (*vector-derivative*

(*g1*+++*g2*) (*at x within* {*0..1*}) · *b*)) *has-integral i2*) {*1/2..1*}
apply (*rule has-integral-spike-finite* [*OF - - i2*, *of insert* (*1/2*) ((λ*x*. *2*∗*x*−*1*) −' *s2*)])
using *s2*
apply (*force intro*: *finite-vimageI* [**where** *h* = λ*x*. *2*∗*x*−*1*] *inj-onI*)
apply (*clarsimp simp add*: *joinpaths-def scaleR-conv-of-real mult-ac g2*)
by (*simp add*: *sum-distrib-left*)
ultimately
show *line-integral f basis* (*g1* +++ *g2*) = *line-integral f basis g1* + *line-integral f basis g2*
apply (*simp add*: *line-integral-def*)
apply (*rule integral-unique* [*OF has-integral-combine* [**where** *c* = *1/2*]])
using *1 2 integral-unique* **apply** *auto*
done
show *line-integral-exists f basis* (*g1* +++ *g2*)
proof (*simp add*: *line-integral-exists-def integrable-on-def*)
have (*1*::*real*) ≤ *1* ∗ *2* ∧ (*0*::*real*) ≤ *1* / *2*
by *simp*
then show ∃ *r*. ((λ*r*. ∑ *a*∈*basis*. *f* ((*g1* +++ *g2*) *r*) · *a* ∗ (*vector-derivative* (*g1* +++ *g2*) (*at r within* {*0..1*}) · *a*)) *has-integral r*) {*0..1*}
using *has-integral-combine* [**where** *c* = *1/2*] *1 2 divide-le-eq-numeral1*(*1*) *lem1 lem2* **by** *blast*
qed
qed

lemma *line-integral-exists-joinD1*:
assumes *line-integral-exists f basis* (*g1* +++ *g2*) *valid-path g1*
shows *line-integral-exists f basis g1*
proof −
obtain *s1*
where *s1*: *finite s1* ∀ *x*∈{*0..1*} − *s1*. *g1 differentiable at x*
using *assms* **by** (*auto simp*: *valid-path-def piecewise-C1-differentiable-on-def C1-differentiable-on-eq*)
have (λ*x*. ∑ *b*∈*basis*. *f* ((*g1* +++ *g2*) (*x/2*)) · *b* ∗ (*vector-derivative* (*g1* +++ *g2*) (*at* (*x/2*) *within* {*0..1*}) · *b*)) *integrable-on* {*0..1*}
using *assms*
apply (*auto simp*: *line-integral-exists-def*)
apply (*drule integrable-on-subcbox* [**where** *a*=*0* **and** *b*=*1/2*])
apply (*auto intro*: *integrable-affinity* [*of - 0 1/2*::*real 1/2 0*, *simplified*])
done
then have ∗:(λ*x*. ∑ *b*∈*basis*. ((*f* ((*g1* +++ *g2*) (*x/2*)) · *b*) / *2*)∗ (*vector-derivative* (*g1* +++ *g2*) (*at* (*x/2*) *within* {*0..1*}) · *b*)) *integrable-on* {*0..1*}
by (*auto simp*: *Groups-Big.sum-distrib-left dest*: *integrable-cmul* [**where** *c*=*1/2*] *simp*: *scaleR-conv-of-real*)
have *g1*: ⟦*0* ≤ *z*; *z*∗*2* < *1*; *z*∗*2* ∉ *s1*⟧ ⟹
vector-derivative (λ*x*. *if x*∗*2* ≤ *1 then g1* (*2*∗*x*) *else g2* (*2*∗*x* − *1*)) (*at z within* {*0..1*}) =
2 $*_R$ *vector-derivative g1* (*at* (*z*∗*2*) *within* {*0..1*}) **for** *z*
proof −

have *i*:⟦*0* ≤ *z*; *z*∗*2* < *1*; *z*∗*2* ∉ *s1*⟧ ⟹
vector-derivative (λ*x*. *if x* ∗ *2* ≤ *1 then g1* (*2* ∗ *x*) *else g2* (*2* ∗ *x* − *1*))
(*at z within* {*0*..*1*}) = *2* ∗$_R$ *vector-derivative g1* (*at* (*z* ∗ *2*)) **for** *z*
proof−
have *g1-at-z*:⟦*0* ≤ *z*; *z*∗*2* < *1*; *z*∗*2* ∉ *s1*⟧ ⟹
((λ*x*. *if x*∗*2* ≤ *1 then g1* (*2*∗*x*) *else g2* (*2*∗*x* − *1*))
has-vector-derivative
2 ∗$_R$ *vector-derivative g1* (*at* (*z*∗*2*))) (*at z*) **for** *z*
apply (*rule has-vector-derivative-transform-at* [*of* |*z* − *1*/*2*| - (λ*x*. *g1*(*2*∗*x*))])
apply (*simp-all add*: *dist-real-def abs-if split*: *if-split-asm*)
apply (*rule vector-diff-chain-at* [*of* λ*x*. *2*∗*x 2* - *g1*, *simplified o-def*])
apply (*simp add*: *has-vector-derivative-def has-derivative-def bounded-linear-mult-left*)
using *s1*
apply (*auto simp*: *algebra-simps vector-derivative-works*)
done
assume *ass*: *0* ≤ *z z*∗*2* < *1 z*∗*2* ∉ *s1*
then have *z-ge*: *z*≤ *1* **by** *auto*
show *vector-derivative* (λ*x*. *if x* ∗ *2* ≤ *1 then g1* (*2* ∗ *x*) *else g2* (*2* ∗ *x* −
1)) (*at z within* {*0*..*1*}) = *2* ∗$_R$ *vector-derivative g1* (*at* (*z* ∗ *2*))
using *Derivative.vector-derivative-at-within-ivl*[*OF g1-at-z*[*OF ass*] *ass*(*1*)
z-ge]
by *auto*
qed
assume *ass*: *0* ≤ *z z*∗*2* < *1 z*∗*2* ∉ *s1*
then have (*g1 has-vector-derivative* ((*vector-derivative g1* (*at* (*z*∗*2*))))) (*at*
(*z*∗*2*))
using *s1* **by** (*auto simp*: *algebra-simps vector-derivative-works*)
then have *ii*: (*vector-derivative g1* (*at* (*z*∗*2*) *within* {*0*..*1*})) = (*vector-derivative*
g1 (*at* (*z*∗*2*)))
using *Derivative.vector-derivative-at-within-ivl ass* **by** *force*
show *vector-derivative* (λ*x*. *if x* ∗ *2* ≤ *1 then g1* (*2* ∗ *x*) *else g2* (*2* ∗ *x* − *1*))
(*at z within* {*0*..*1*}) = *2* ∗$_R$ *vector-derivative g1* (*at* (*z* ∗ *2*) *within* {*0*..*1*})
using *i*[*OF ass*] *ii* **by** *auto*
qed
show *?thesis*
using *s1*
apply (*auto simp*: *line-integral-exists-def*)
apply (*rule integrable-spike-finite* [*of* {*0*,*1*} ∪ *s1*, *OF* - - ∗])
apply (*auto simp*: *joinpaths-def scaleR-conv-of-real g1*)
done
qed

lemma *line-integral-exists-joinD2*:
assumes *line-integral-exists f basis* (*g1* +++ *g2*) *valid-path g2*
shows *line-integral-exists f basis g2*
proof −
obtain *s2*
where *s2*: *finite s2* ∀ *x*∈{*0*..*1*} − *s2*. *g2 differentiable at x*
using *assms* **by** (*auto simp*: *valid-path-def piecewise-C1-differentiable-on-def*

C1-differentiable-on-eq)
have (λ*x*. ∑ *b*∈*basis*. *f* ((*g1* +++ *g2*) (*x*/*2* + *1*/*2*)) · *b* ∗ (*vector-derivative* (*g1* +++ *g2*) (*at* (*x*/*2* + *1*/*2*) *within* {*0*..*1*}) · *b*)) *integrable-on* {*0*..*1*}
using *assms*
apply (*auto simp*: *line-integral-exists-def*)
apply (*drule integrable-on-subcbox* [**where** *a*=*1*/*2* **and** *b*=*1*], *auto*)
apply (*drule integrable-affinity* [*of* - *1*/*2*::*real 1 1*/*2 1*/*2*, *simplified*])
apply (*simp add*: *image-affinity-atLeastAtMost-diff*)
done
then have ∗:(λ*x*. ∑ *b*∈*basis*. ((*f* ((*g1* +++ *g2*) (*x*/*2* + *1*/*2*)) · *b*) / *2*)∗ (*vector-derivative* (*g1* +++ *g2*) (*at* (*x*/*2* + *1*/*2*) *within* {*0*..*1*}) · *b*)) *integrable-on* {*0*..*1*}
by (*auto simp*: *Groups-Big.sum-distrib-left dest*: *integrable-cmul* [**where** *c*=*1*/*2*] *simp*: *scaleR-conv-of-real*)
have *g2*: ⟦*1* < *z*∗*2*; *z* ≤ *1*; *z*∗*2* − *1* ∉ *s2*⟧ ⟹
vector-derivative (λ*x*. *if x*∗*2* ≤ *1 then g1* (*2*∗*x*) *else g2* (*2*∗*x* − *1*)) (*at z within* {*0*..*1*}) =
2 $*_R$ *vector-derivative g2* (*at* (*z*∗*2* − *1*) *within* {*0*..*1*}) **for** *z* **proof** −
have *i*:⟦*1* < *z*∗*2*; *z* ≤ *1*; *z*∗*2* − *1* ∉ *s2*⟧ ⟹
vector-derivative (λ*x*. *if x* ∗ *2* ≤ *1 then g1* (*2* ∗ *x*) *else g2* (*2* ∗ *x* − *1*)) (*at z within* {*0*..*1*})
= *2* $*_R$ *vector-derivative g2* (*at* (*z* ∗ *2* − *1*)) **for** *z*
proof−
have *g2-at-z*:⟦*1* < *z*∗*2*; *z* ≤ *1*; *z*∗*2* − *1* ∉ *s2*⟧ ⟹
((λ*x*. *if x*∗*2* ≤ *1 then g1* (*2*∗*x*) *else g2* (*2*∗*x* − *1*)) *has-vector-derivative 2* $*_R$ *vector-derivative g2* (*at* (*z*∗*2* − *1*))) (*at z*) **for** *z*
apply (*rule has-vector-derivative-transform-at* [*of* |*z* − *1*/*2*| - (λ*x*. *g2* (*2*∗*x* − *1*))])
apply (*simp-all add*: *dist-real-def abs-if split*: *if-split-asm*)
apply (*rule vector-diff-chain-at* [*of* λ*x*. *2*∗*x* − *1 2* - *g2*, *simplified o-def*])
apply (*simp add*: *has-vector-derivative-def has-derivative-def bounded-linear-mult-left*)
using *s2*
apply (*auto simp*: *algebra-simps vector-derivative-works*)
done
assume *ass*: *1* < *z*∗*2 z* ≤ *1 z*∗*2* − *1* ∉ *s2*
then have *z-le*: *z*≤ *1* **by** *auto*
have *z-ge*: *0* ≤ *z* **using** *ass* **by** *auto*
show *vector-derivative* (λ*x*. *if x* ∗ *2* ≤ *1 then g1* (*2* ∗ *x*) *else g2* (*2* ∗ *x* − *1*)) (*at z within* {*0*..*1*})
= *2* $*_R$ *vector-derivative g2* (*at* (*z* ∗ *2* − *1*))
using *Derivative.vector-derivative-at-within-ivl*[*OF g2-at-z*[*OF ass*] *z-ge z-le*]
by *auto*
qed
assume *ass*: *1* < *z*∗*2 z* ≤ *1 z*∗*2* − *1* ∉ *s2*
then have (*g2 has-vector-derivative* ((*vector-derivative g2* (*at* (*z*∗*2* − *1*))))) (*at* (*z*∗*2* − *1*))
using *s2* **by** (*auto simp*: *algebra-simps vector-derivative-works*)
then have *ii*: (*vector-derivative g2* (*at* (*z*∗*2* − *1*) *within* {*0*..*1*})) = (*vector-derivative*

```
g2 (at (z*2 − 1)))
      using Derivative.vector-derivative-at-within-ivl ass
      by force
    show vector-derivative (λx. if x ∗ 2 ≤ 1 then g1 (2 ∗ x) else g2 (2 ∗ x − 1))
(at z within {0..1}) = 2 ∗R vector-derivative g2 (at (z ∗ 2 − 1) within {0..1})
      using i[OF ass] ii
      by auto
  qed
  show ?thesis
    using s2
    apply (auto simp: line-integral-exists-def)
    apply (rule integrable-spike-finite [of {0,1} ∪ s2, OF - - ∗])
     apply (auto simp: joinpaths-def scaleR-conv-of-real g2)
    done
qed

lemma has-line-integral-on-reverse-path:
  assumes g: valid-path g and int:
    ((λx. ∑ b∈basis. F (g x) · b ∗ (vector-derivative g (at x within {0..1}) · b))
has-integral c){0..1}
  shows ((λx. ∑ b∈basis. F ((reversepath g) x) · b ∗ (vector-derivative (reversepath
g) (at x within {0..1}) · b)) has-integral −c){0..1}
proof −
  { fix s x
    assume xs: g C1-differentiable-on ({0..1} − s) x ∉ (−) 1 ‘ s 0 ≤ x x ≤ 1
    have vector-derivative (λx. g (1 − x)) (at x within {0..1}) =
              − vector-derivative g (at (1 − x) within {0..1})
    proof −
      obtain f′ where f′: (g has-vector-derivative f′) (at (1 − x))
        using xs
        by (force simp: has-vector-derivative-def C1-differentiable-on-def)
      have (g o (λx. 1 − x) has-vector-derivative −1 ∗R f′) (at x)
        apply (rule vector-diff-chain-within)
         apply (intro vector-diff-chain-within derivative-eq-intros | simp)+
        apply (rule has-vector-derivative-at-within [OF f′])
        done
      then have mf′: ((λx. g (1 − x)) has-vector-derivative −f′) (at x)
        by (simp add: o-def)
      show ?thesis
        using xs
      by (auto simp: vector-derivative-at-within-ivl [OF mf′] vector-derivative-at-within-ivl
[OF f′])
    qed
  } note ∗ = this
  obtain S where continuous-on {0..1} g finite S g C1-differentiable-on {0..1}
− S
    using g
    by (auto simp: valid-path-def piecewise-C1-differentiable-on-def)
  then show ?thesis
```

```
    using has-integral-affinity01 [OF int, where m= −1 and c=1]
    unfolding reversepath-def
    by (rule-tac S = (λx. 1 − x) ‘ S in has-integral-spike-finite) (auto simp: ∗
has-integral-neg Groups-Big.sum-negf)
qed

lemma line-integral-on-reverse-path:
  assumes valid-path γ line-integral-exists F basis γ
  shows line-integral F basis γ = − (line-integral F basis (reversepath γ))
      line-integral-exists F basis (reversepath γ)
proof −
  obtain i where
    0: ((λx. ∑ b∈basis. F (γ x) · b ∗ (vector-derivative γ (at x within {0..1}) · b))
has-integral i){0..1}
    using assms unfolding integrable-on-def line-integral-exists-def by auto
  then have 1: ((λx. ∑ b∈basis. F ((reversepath γ) x) · b ∗ (vector-derivative
(reversepath γ) (at x within {0..1}) · b)) has-integral −i){0..1}
    using has-line-integral-on-reverse-path assms
    by auto
  then have rev-line-integral:line-integral F basis (reversepath γ) = −i
    using line-integral-def Henstock-Kurzweil-Integration.integral-unique
    by (metis (no-types))
  have line-integral: line-integral F basis γ = i
    using line-integral-def 0 Henstock-Kurzweil-Integration.integral-unique
    by blast
  show line-integral F basis γ = − (line-integral F basis (reversepath γ))
    using line-integral rev-line-integral
    by auto
  show line-integral-exists F basis (reversepath γ)
    using 1 line-integral-exists-def
    by auto
qed

lemma line-integral-exists-on-degenerate-path:
  assumes finite basis
  shows line-integral-exists F basis (λx. c)
proof−
  have every-component-integrable:
   ∀ b∈basis. (λx. F ((λx. c) x) · b ∗ (vector-derivative (λx. c) (at x within {0..1})
· b)) integrable-on {0..1}
  proof
    fix b
    assume b-in-basis: b ∈ basis
    have cont-field-zero-one: continuous-on {0..1} (λx. F ((λx. c) x) · b)
      using continuous-on-const by fastforce
    have cont-path-zero-one:
     continuous-on {0..1} (λx. (vector-derivative (λx. c) (at x within {0..1})) · b)
    proof −
      have ((vector-derivative (λx. c) (at x within {0..1})) · b) = 0 if x ∈ {0..1}
```

for *x*
proof −
have *vector-derivative* (*λx. c*) (*at x within* {*0..1*}) = *0*
using *that gamma-deriv-at-within*[*of 0 1*] *differentiable-const vector-derivative-const-at*
by *fastforce*
then show *vector-derivative* (*λx. c*) (*at x within* {*0..1*}) · *b* = *0*
by *auto*
qed
then show *continuous-on* {*0..1*} (*λx.* (*vector-derivative* (*λx. c*) (*at x within* {*0..1*})) · *b*)
using *continuous-on-const*[*of* {*0..1*} *0*] *continuous-on-eq*[*of* {*0..1*} *λx. 0* (*λx.* (*vector-derivative* (*λx. c*) (*at x within* {*0..1*})) · *b*)]
by *auto*
qed
show (*λx. F* (*c*) · *b* ∗ (*vector-derivative* (*λx. c*) (*at x within* {*0..1*}) · *b*)) *integrable-on* {*0..1*}
using *cont-field-zero-one cont-path-zero-one continuous-on-mult integrable-continuous-real*
by *blast*
qed
have *integrable-sum'*: ⋀*t f s. finite t* ⟹
∀ *a*∈*t. f a integrable-on s* ⟹ (*λx.* ∑ *a*∈*t. f a x*) *integrable-on s*
using *integrable-sum* **by** *metis*
have *field-integrable-on-basis*:
(*λx.* ∑ *b*∈*basis. F* (*c*) · *b* ∗ (*vector-derivative* (*λx. c*) (*at x within* {*0..1*}) · *b*)) *integrable-on* {*0..1*}
using *integrable-sum'*[*OF assms*(*1*) *every-component-integrable*]
by *auto*
then show *?thesis*
using *line-integral-exists-def* **by** *auto*
qed

lemma *degenerate-path-is-valid-path*: *valid-path* (*λx. c*)
by (*auto simp add*: *valid-path-def piecewise-C1-differentiable-on-def continuous-on-const*)

lemma *line-integral-degenerate-path*:
assumes *finite basis*
shows *line-integral F basis* (*λx. c*) = *0*
proof (*simp add*: *line-integral-def*)
have ((*vector-derivative* (*λx. c*) (*at x within* {*0..1*})) · *b*) = *0* **if** *x* ∈ {*0..1*} **for** *x b*
proof −
have *vector-derivative* (*λx. c*) (*at x within* {*0..1*}) = *0*
using *that gamma-deriv-at-within*[*of 0 1*] *differentiable-const vector-derivative-const-at*
by *fastforce*
then show *vector-derivative* (*λx. c*) (*at x within* {*0..1*}) · *b* = *0*
by *auto*
qed
then have *0*: ⋀*x. x* ∈ {*0..1*} ⟹ (*λx.* ∑ *b*∈*basis. F c* · *b* ∗ (*vector-derivative* (*λx. c*) (*at x within* {*0..1*}) · *b*)) *x* = (*λx. 0*) *x*

by *auto*
then show *integral* {*0..1*} (λ*x*. ∑ *b*∈*basis*. *F c* · *b* ∗ (*vector-derivative* (λ*x*. *c*) (*at x within* {*0..1*}) · *b*)) = *0*
using *integral-cong*[*of* {*0..1*}, *OF 0*] *integral-0* **by** *auto*
qed

definition *point-path* **where**
point-path γ ≡ ∃ *c*. γ = (λ*x*. *c*)

lemma *line-integral-point-path*:
assumes *point-path* γ
assumes *finite basis*
shows *line-integral F basis* γ = *0*
using *assms*(*1*) *point-path-def line-integral-degenerate-path*[*OF assms*(*2*)]
by *force*

lemma *line-integral-exists-point-path*:
assumes *finite basis point-path* γ
shows *line-integral-exists F basis* γ
using *assms*
apply(*simp add*: *point-path-def*)
using *line-integral-exists-on-degenerate-path* **by** *auto*

lemma *line-integral-exists-subpath*:
assumes *f*: *line-integral-exists f basis g* **and** *g*: *valid-path g*
and *uv*: *u* ∈ {*0..1*} *v* ∈ {*0..1*} *u* ≤ *v*
shows (*line-integral-exists f basis* (*subpath u v g*))
proof (*cases v*=*u*)
case *tr*: *True*
have *zero*: (∑ *b*∈*basis*. *f* (*g u*) · *b* ∗ (*vector-derivative* (λ*x*. *g u*) (*at x within* {*0..1*}) · *b*)) = *0* **if** *x* ∈ {*0..1*} **for** *x*::*real*
proof −
have (*vector-derivative* (λ*x*. *g u*) (*at x within* {*0..1*})) = *0*
using *Deriv.has-vector-derivative-const that Derivative.vector-derivative-at-within-ivl*
by *fastforce*
then show (∑ *b*∈*basis*. *f* (*g u*) · *b* ∗ (*vector-derivative* (λ*x*. *g u*) (*at x within* {*0..1*}) · *b*)) = *0*
by *auto*
qed
then have ((λ*x*. ∑ *b*∈*basis*. *f* (*g u*) · *b* ∗ (*vector-derivative* (λ*x*. *g u*) (*at x within* {*0..1*}) · *b*)) *has-integral 0*) {*0..1*}
by (*meson has-integral-is-0*)
then show *?thesis*
using *f tr* **by** (*auto simp add*: *line-integral-def line-integral-exists-def subpath-def*)
next
case *False*
obtain *s* **where** *s*: ⋀*x*. *x* ∈ {*0..1*} − *s* ⟹ *g differentiable at x* **and** *fs*: *finite s*
using *g* **unfolding** *piecewise-C1-differentiable-on-def C1-differentiable-on-eq*

valid-path-def **by** *blast*
have ∗: ((λx. ∑ b∈basis. f (g ((v − u) ∗ x + u)) · b ∗ (vector-derivative g (at ((v − u) ∗ x + u) within {0..1}) · b))
has-integral (1 / (v − u)) ∗ integral {u..v} (λx. ∑ b∈basis. f (g (x)) · b ∗ (vector-derivative g (at x within {0..1}) · b)))
{0..1}
using *f uv*
apply (*simp add*: *line-integral-exists-def subpath-def*)
apply (*drule integrable-on-subcbox* [**where** *a*=*u* **and** *b*=*v*, *simplified*])
apply (*simp-all add*: *has-integral-integral*)
apply (*drule has-integral-affinity* [**where** *m*=*v*−*u* **and** *c*=*u*, *simplified*])
apply (*simp-all add*: *False image-affinity-atLeastAtMost-div-diff scaleR-conv-of-real*)
apply (*simp add*: *divide-simps False*)
done
have *vd*:⋀x. x ∈ {0..1} ⟹
x ∉ (λt. (v−u) $*_R$ t + u) −‘ s ⟹
vector-derivative (λx. g ((v−u) ∗ x + u)) (at x within {0..1}) = (v−u) $*_R$ vector-derivative g (at ((v−u) ∗ x + u) within {0..1})
using *test2*[*OF s fs uv*]
by *auto*
have *arg*:⋀x. (∑ n∈basis. (v − u) ∗ (f (g ((v − u) ∗ x + u)) · n) ∗ (vector-derivative g (at ((v − u) ∗ x + u) within {0..1}) · n))
= (∑ b∈basis. f (g ((v − u) ∗ x + u)) · b ∗ (v − u) ∗ (vector-derivative g (at ((v − u) ∗ x + u) within {0..1}) · b))
by (*simp add*: *mult.commute*)
have((λx. ∑ b∈basis. f (g ((v − u) ∗ x + u)) · b ∗ (vector-derivative (λx. g ((v − u) ∗ x + u)) (at x within {0..1}) · b)) has-integral
(integral {u..v} (λx. ∑ b∈basis. f (g (x)) · b ∗ (vector-derivative g (at x within {0..1}) · b)))) {0..1}
apply (*cut-tac Henstock-Kurzweil-Integration.has-integral-mult-right* [*OF* ∗, **where** *c* = *v*−*u*])
using *fs assms*
apply (*simp add*: *False subpath-def line-integral-exists-def*)
apply (*rule-tac S* = (λt. ((v−u) $*_R$ t + u)) −‘ *s* **in** *has-integral-spike-finite*)
apply (*auto simp*: *inj-on-def False vd finite-vimageI scaleR-conv-of-real Groups-Big.sum-distrib-left*
mult.assoc[*symmetric*] *arg*)
done
then show (*line-integral-exists f basis* (*subpath u v g*))
by(*auto simp add*: *line-integral-exists-def subpath-def integrable-on-def*)
qed

type-synonym *path* = *real* ⇒ (*real* ∗ *real*)
type-synonym *one-cube* = (*real* ⇒ (*real* ∗ *real*))
type-synonym *one-chain* = (*int* ∗ *path*) *set*
type-synonym *two-cube* = (*real* ∗ *real*) ⇒ (*real* ∗ *real*)
type-synonym *two-chain* = *two-cube set*

definition *one-chain-line-integral* :: $((real * real) \Rightarrow (real * real))$ => $((real*real)$ $set) \Rightarrow$ *one-chain* $\Rightarrow$ *real* **where**
  one-chain-line-integral F b C $\equiv (\sum (k,g) \in C.\ k * ($*line-integral F b g*$))$

definition *boundary-chain* **where**
  boundary-chain s $\equiv (\forall (k, \gamma) \in s.\ k = 1 \vee k = -1)$

fun *coeff-cube-to-path*::(*int* $*$ *one-cube*) $\Rightarrow$ *path*
 where *coeff-cube-to-path* $(k, \gamma) = ($*if* $k = 1$ *then* γ *else* (*reversepath* γ))

fun *rec-join* :: (*int*$*$*path*) *list* $\Rightarrow$ *path* **where**
 rec-join [] = $(\lambda x.0)$ |
 rec-join [*oneC*] = *coeff-cube-to-path oneC* |
 rec-join (*oneC* # *xs*) = *coeff-cube-to-path oneC* +++ (*rec-join xs*)

fun *valid-chain-list* **where**
 valid-chain-list [] = *True* |
 valid-chain-list [*oneC*] = *True* |
 valid-chain-list (*oneC*#*l*) = (*pathfinish* (*coeff-cube-to-path* (*oneC*)) = *pathstart* (*rec-join l*) $\wedge$ *valid-chain-list l*)

lemma *joined-is-valid*:
 assumes *boundary-chain*: *boundary-chain* (*set l*) **and**
  valid-path: $\bigwedge k\ \gamma.\ (k, \gamma) \in$ *set l* $\Longrightarrow$ *valid-path* γ **and**
  valid-chain-list-ass: *valid-chain-list l*
 shows *valid-path* (*rec-join l*)
 using *assms*
proof(*induction l*)
 case *Nil*
 then show *?case*
  using *C1-differentiable-imp-piecewise*[*OF C1-differentiable-on-const*[*of 0* {*0..1*}]]
  by (*auto simp add*: *valid-path-def*)
next
 case (*Cons a l*)
 have $*$: *valid-path* (*rec-join* ((*k*::*int*, γ) # *l*))
  if *boundary-chain* (*set* (*l*))
   $(\bigwedge k'\ \gamma'.\ (k', \gamma') \in$ *set l* $\Longrightarrow$ *valid-path* $\gamma')$
   valid-chain-list l
   valid-path (*rec-join l*)
   $(\bigwedge k'\ \gamma'.\ (k', \gamma') \in$ *set* $((k, \gamma)$ # *l*) $\Longrightarrow$ *valid-path* $\gamma')$
   valid-chain-list $((k, \gamma)$ # *l*)
   boundary-chain (*set* $((k,\gamma)$ # *l*)) **for** $k\ \gamma\ l$
 proof (*cases l* = [])
  case *True*
  with *that* **show** *valid-path* (*rec-join* $((k, \gamma)$ # *l*))
   by *auto*
 next

```
    case False
    then obtain h l' where l-nempty: l = h#l'
      by (meson rec-join.elims)
    then show valid-path (rec-join ((k, γ) # l))
    proof (simp, intro conjI impI)
      assume k-eq-1: k = 1
      have 0:valid-path γ
        using that by auto
      have 1:pathfinish γ = pathstart (rec-join (h#l'))
        using that(6) k-eq-1 l-nempty by auto
      show valid-path (γ +++ rec-join (h#l'))
        using 0 1 valid-path-join that(4) l-nempty by auto
    next
      assume k ≠ 1
      then have k-eq-neg-1: k = −1
        using that(7)
        by (auto simp add: boundary-chain-def)
      have valid-path γ
        using that by auto
      then have 0: valid-path (reversepath γ)
        using valid-path-imp-reverse
        by auto
      have 1: pathfinish (reversepath γ) = pathstart (rec-join (h#l'))
        using that(6) k-eq-neg-1 l-nempty by auto
      show valid-path ((reversepath γ) +++ rec-join (h#l'))
        using 0 1 valid-path-join that(4) l-nempty by blast
    qed
  qed
  have ∀ps. valid-chain-list ps ∨ (∃i f p psa. ps = (i, f) # p # psa ∧ ((i = 1 ∧
pathfinish f ≠ pathstart (rec-join (p # psa)) ∨ i ≠ 1 ∧ pathfinish (reversepath f)
≠ pathstart (rec-join (p # psa))) ∨ ¬ valid-chain-list (p # psa)))
    by (smt coeff-cube-to-path.elims valid-chain-list.elims(3))
  moreover have boundary-chain (set l)
    by (meson Cons.prems(1) boundary-chain-def set-subset-Cons subset-eq)
  ultimately show ?case
   using ∗ Cons by (metis (no-types) list.set-intros(2) prod.collapse valid-chain-list.simps(3))
qed

lemma pathstart-rec-join-1:
  pathstart (rec-join ((1, γ) # l)) = pathstart γ
proof (cases l = [])
  case True
  then show pathstart (rec-join ((1, γ) # l)) = pathstart γ
    by simp
next
  case False
  then obtain h l' where l = h#l'
    by (meson rec-join.elims)
  then show pathstart (rec-join ((1, γ) # l)) = pathstart γ
```

by *simp*
qed

lemma *pathstart-rec-join-2*:
pathstart (*rec-join* ((−*1*, γ) # *l*)) = *pathstart* (*reversepath* γ)
proof *cases*
assume *l* = []
then show *pathstart* (*rec-join* ((− *1*, γ) # *l*)) = *pathstart* (*reversepath* γ)
by *simp*
next
assume *l* ≠ []
then obtain *h l′* **where** *l* = *h*#*l′*
by (*meson rec-join.elims*)
then show *pathstart* (*rec-join* ((− *1*, γ) # *l*)) = *pathstart* (*reversepath* γ)
by *simp*
qed

lemma *pathstart-rec-join*:
pathstart (*rec-join* ((*1*, γ) # *l*)) = *pathstart* γ
pathstart (*rec-join* ((−*1*, γ) # *l*)) = *pathstart* (*reversepath* γ)
using *pathstart-rec-join-1 pathstart-rec-join-2*
by *auto*

lemma *line-integral-exists-on-rec-join*:
assumes *boundary-chain*: *boundary-chain* (*set l*) **and**
valid-chain-list: *valid-chain-list l* **and**
valid-path: $\bigwedge k\ \gamma.\ (k, \gamma) \in$ *set l* $\Longrightarrow$ *valid-path* γ **and**
line-integral-exists: $\forall (k, \gamma) \in$ *set l*. *line-integral-exists F basis* γ
shows *line-integral-exists F basis* (*rec-join l*)
using *assms*
proof (*induction l*)
case *Nil*
then show *?case*
proof (*simp add*: *line-integral-exists-def*)
have $\forall x$. (*vector-derivative* (λx. *0*) (*at x*)) = *0*
using *Derivative.vector-derivative-const-at*
by *auto*
then have $\forall x$. ((λx. *0*) *has-vector-derivative 0*) (*at x*)
using *Derivative.vector-derivative-const-at*
by *auto*
then have $\forall x$. ((λx. *0*) *has-vector-derivative 0*) (*at x within* {*0*..*1*})
using *Derivative.vector-derivative-const-at*
by *auto*
then have *0*: $\forall x \in$ {*0*..*1*}. (*vector-derivative* (λx. *0*) (*at x within*{*0*..*1*})) = *0*
by (*simp add*: *gamma-deriv-at-within*)
have ($\forall x \in$ {*0*..*1*}. ($\sum b \in$ *basis*. *F 0* · *b* ∗ (*vector-derivative* (λx. *0*) (*at x within* {*0*..*1*}) · *b*)) = *0*)
by (*simp add*: *0*)
then have ((λx. $\sum b \in$ *basis*. *F 0* · *b* ∗ (*vector-derivative* (λx. *0*) (*at x within*

```
{0..1}) · b)) has-integral 0) {0..1}
      by (meson has-integral-is-0)
    then show (λx. ∑ b∈basis. F 0 · b ∗ (vector-derivative (λx. 0) (at x within
{0..1}) · b)) integrable-on {0..1}
      by auto
  qed
next
  case (Cons a l)
  obtain k γ where aeq: a = (k,γ)
    by force
  show ?case
    unfolding aeq
  proof cases
    assume l-empty: l = []
    then show line-integral-exists F basis (rec-join ((k, γ) # l))
      using Cons.prems aeq line-integral-on-reverse-path(2) by fastforce
  next
    assume l ≠ []
    then obtain h l' where l-nempty: l = h#l'
      by (meson rec-join.elims)
    show line-integral-exists F basis (rec-join ((k, γ) # l))
    proof (auto simp add: l-nempty)
      assume k-eq-1: k = 1
      have 0: line-integral-exists F basis γ
        using Cons.prems(4) aeq by auto
      have 1: line-integral-exists F basis (rec-join l)
      by (metis (mono-tags) Cons boundary-chain-def list.set-intros(2) valid-chain-list.elims(3)
valid-chain-list.simps(3))
      have 2: valid-path γ
        using Cons aeq by auto
      have 3:valid-path (rec-join l)
       by (metis (no-types) Cons.prems boundary-chain-def joined-is-valid l-nempty
set-subset-Cons subsetCE valid-chain-list.simps(3))
      show line-integral-exists F basis (γ +++ rec-join (h#l'))
        using line-integral-distrib(2)[OF 0 1 2 3] assms l-nempty by auto
    next
      assume k ≠ 1
      then have k-eq-neg-1: k = −1
        using Cons aeq by (simp add: boundary-chain-def)
      have gamma-valid: valid-path γ
        using Cons aeq by auto
      then have 2: valid-path (reversepath γ)
        using valid-path-imp-reverse by auto
      have line-integral-exists F basis γ
        using Cons aeq by auto
      then have 0: line-integral-exists F basis (reversepath γ)
        using line-integral-on-reverse-path(2) gamma-valid
        by auto
      have 1: line-integral-exists F basis (rec-join l)
```

using *Cons aeq*
by (*metis* (*mono-tags*) *boundary-chain-def insert-iff list.set*(*2*) *list.set-intros*(*2*) *valid-chain-list.elims*(*3*) *valid-chain-list.simps*(*3*))
have *3*:*valid-path* (*rec-join l*)
by (*metis* (*no-types*) *Cons.prems*(*1*) *Cons.prems*(*2*) *Cons.prems*(*3*) *boundary-chain-def joined-is-valid l-nempty set-subset-Cons subsetCE valid-chain-list.simps*(*3*))
show *line-integral-exists F basis* ((*reversepath* γ) +++ *rec-join* ($h\#l'$))
using *line-integral-distrib*(*2*)[*OF 0 1 2 3*] *assms l-nempty*
by *auto*
qed
qed
qed

lemma *line-integral-exists-rec-join-cons*:
assumes *line-integral-exists F basis* (*rec-join* ((*1*,γ) # *l*))
($\bigwedge k'\ \gamma'$. (k', γ') $\in$ *set* ((*1*,γ) # *l*) $\Longrightarrow$ *valid-path* γ')
finite basis
shows *line-integral-exists F basis* (γ +++ (*rec-join l*))
proof *cases*
assume *l-empty*: $l = []$
show *line-integral-exists F basis* (γ +++ *rec-join l*)
using *assms*(*2*) *line-integral-distrib*(*2*)[*OF assms*(*1*) *line-integral-exists-on-degenerate-path*[*OF assms*(*3*)], *of 0*]
using *degenerate-path-is-valid-path*
by (*fastforce simp add*: *l-empty*)
next
assume $l \neq []$
then obtain $h\ l'$ **where** $l = h\#l'$
by (*meson rec-join.elims*)
then show *line-integral-exists F basis* (γ +++ *rec-join l*)
using *assms* **by** *auto*
qed

lemma *line-integral-exists-rec-join-cons-2*:
assumes *line-integral-exists F basis* (*rec-join* ((−*1*,γ) # *l*))
($\bigwedge k'\ \gamma'$. (k', γ') $\in$ *set* ((*1*,γ) # *l*) $\Longrightarrow$ *valid-path* γ')
finite basis
shows *line-integral-exists F basis* ((*reversepath* γ) +++ (*rec-join l*))
proof *cases*
assume *l-empty*: $l = []$
show *line-integral-exists F basis* ((*reversepath* γ) +++ *rec-join l*)
using *assms*(*2*) *line-integral-distrib*(*2*)[*OF assms*(*1*) *line-integral-exists-on-degenerate-path*[*OF assms*(*3*)], *of 0*]
using *degenerate-path-is-valid-path*
by (*auto simp add*: *l-empty*)
next
assume $l \neq []$
then obtain $h\ l'$ **where** $l = h\#l'$
by (*meson rec-join.elims*)

```
  with assms show line-integral-exists F basis ((reversepath γ) +++ rec-join l)
    using assms by auto
qed

lemma line-integral-exists-on-rec-join':
  assumes boundary-chain: boundary-chain (set l) and
    valid-chain-list: valid-chain-list l and
    valid-path: ⋀k γ. (k, γ) ∈ set l ⟹ valid-path γ and
    line-integral-exists: line-integral-exists F basis (rec-join l) and
    finite-basis: finite basis
  shows ∀ (k, γ) ∈ set l. line-integral-exists F basis γ
  using assms
proof (induction l)
  case Nil
  show ?case
    by (simp add: line-integral-exists-def)
next
  case ass: (Cons a l)
  obtain k γ where k-gamma:a = (k,γ)
    by fastforce
  show ?case
    apply (auto simp add: k-gamma)
  proof -
    show line-integral-exists F basis γ
    proof(cases k = 1)
      assume k-eq-1: k = 1
      have 0: line-integral-exists F basis (γ +++ (rec-join l))
        using line-integral-exists-rec-join-cons k-eq-1 k-gamma ass(4) ass(5) ass(6)
        by auto
      have 2: valid-path γ
        using ass k-gamma by auto
      show line-integral-exists F basis γ
        using line-integral-exists-joinD1[OF 0 2]
        by auto
    next
      assume k ≠ 1
      then have k-eq-neg-1: k = −1
        using ass k-gamma
        by (simp add: boundary-chain-def)
      have 0: line-integral-exists F basis ((reversepath γ) +++ (rec-join l))
         using line-integral-exists-rec-join-cons-2[OF ] k-eq-neg-1 k-gamma ass(4)
ass(5) ass(6)
        by fastforce
      have gamma-valid:
        valid-path γ
        using ass k-gamma by auto
      then have 2: valid-path (reversepath γ)
        using valid-path-imp-reverse by auto
      have line-integral-exists F basis (reversepath γ)
```

```
      using line-integral-exists-joinD1[OF 0 2] by auto
    then show line-integral-exists F basis (γ)
      using line-integral-on-reverse-path(2)[OF 2] reversepath-reversepath
      by auto
  qed
next
  have 0:boundary-chain (set l)
    using ass(2)
    by (auto simp add: boundary-chain-def)
  have 1:valid-chain-list l
    using ass(3)
    apply (auto simp add: k-gamma)
    by (metis valid-chain-list.elims(3) valid-chain-list.simps(3))
  have 2:(⋀k γ. (k, γ) ∈ set (l) ⟹ valid-path γ)
    using ass(4) by auto
  have 3: valid-path (rec-join l)
    using joined-is-valid[OF 0] 1 2 by auto
  have 4: line-integral-exists F basis (rec-join l)
  proof(cases k = 1)
    assume k-eq-1: k = 1
    have 0: line-integral-exists F basis (γ +++ (rec-join l))
      using line-integral-exists-rec-join-cons k-eq-1 k-gamma ass(4) ass(5) ass(6)
by auto
    show line-integral-exists F basis (rec-join l)
      using line-integral-exists-joinD2[OF 0 3] by auto
  next
    assume k ≠ 1
    then have k-eq-neg-1: k = -1
      using ass k-gamma
      by (simp add: boundary-chain-def)
    have 0: line-integral-exists F basis ((reversepath γ) +++ (rec-join l))
       using line-integral-exists-rec-join-cons-2[OF ] k-eq-neg-1 k-gamma ass(4)
ass(5) ass(6)
      by fastforce
    show line-integral-exists F basis (rec-join l)
      using line-integral-exists-joinD2[OF 0 3]
      by auto
  qed
  show ⋀a b. (a, b) ∈ set l ⟹ line-integral-exists F basis b
    using 0 1 2 3 4 ass(1)[OF 0 1 2] ass(6)
    by fastforce
qed
qed

inductive chain-subdiv-path
  where I: chain-subdiv-path γ (set l) if distinct l rec-join l = γ valid-chain-list l

lemma valid-path-equiv-valid-chain-list:
  assumes path-eq-chain: chain-subdiv-path γ one-chain
```

and *boundary-chain one-chain* ∀ (*k*, *γ*) ∈ *one-chain*. *valid-path γ*
shows *valid-path γ*
proof −
obtain *l* **where** *l-props*: *set l* = *one-chain distinct l rec-join l* = *γ valid-chain-list l*
using *chain-subdiv-path.cases path-eq-chain* **by** *force*
show *valid-path γ*
using *joined-is-valid assms l-props* **by** *blast*
qed

lemma *line-integral-rec-join-cons*:
assumes *line-integral-exists F basis γ*
line-integral-exists F basis (*rec-join* ((*l*)))
(⋀*k′ γ′*. (*k′*, *γ′*) ∈ *set* ((*1*,*γ*) # *l*) ⟹ *valid-path γ′*)
finite basis
shows *line-integral F basis* (*rec-join* ((*1*,*γ*) # *l*)) = *line-integral F basis* (*γ* +++ (*rec-join l*))
proof *cases*
assume *l-empty*: *l* = []
show *line-integral F basis* (*rec-join* ((*1*,*γ*) # *l*)) = *line-integral F basis* (*γ* +++ (*rec-join l*))
using *assms line-integral-distrib*(*1*)[*OF assms*(*1*) *line-integral-exists-on-degenerate-path*[*OF assms*(*4*)], *of 0*]
apply (*auto simp add*: *l-empty*)
using *degenerate-path-is-valid-path line-integral-degenerate-path*
by *fastforce*
next
assume *l* ≠ []
then obtain *h l′* **where** *l-nempty*: *l* = *h*#*l′*
by (*meson rec-join.elims*)
show *line-integral F basis* (*rec-join* ((*1*,*γ*) # *l*)) = *line-integral F basis* (*γ* +++ (*rec-join l*))
using *assms* **by** (*auto simp add*: *l-nempty*)
qed

lemma *line-integral-rec-join-cons-2*:
assumes *line-integral-exists F basis γ*
line-integral-exists F basis (*rec-join* ((*l*)))
(⋀*k′ γ′*. (*k′*, *γ′*) ∈ *set* ((−*1*,*γ*) # *l*) ⟹ *valid-path γ′*)
finite basis
shows *line-integral F basis* (*rec-join* ((−*1*,*γ*) # *l*)) = *line-integral F basis* ((*reversepath γ*) +++ (*rec-join l*))
proof *cases*
assume *l-empty*: *l* = []
have *0*: *line-integral-exists F basis* (*reversepath γ*)
using *assms line-integral-on-reverse-path*(*2*) **by** *fastforce*
have *1*: *valid-path* (*reversepath γ*)
using *assms* **by** *fastforce*
show *line-integral F basis* (*rec-join* ((−*1*,*γ*) # *l*)) = *line-integral F basis* ((*reversepath*

γ) +++ (*rec-join l*))
  using *assms line-integral-distrib*(*1*)[*OF 0 line-integral-exists-on-degenerate-path*[*OF assms*(*4*)], *of 0*]
    apply (*auto simp add*: *l-empty*)
    using *degenerate-path-is-valid-path line-integral-degenerate-path*
    by *fastforce*
next
  assume $l \neq []$
  then obtain *h l′* **where** *l-nempty*: *l* = *h*#*l′*
    by (*meson rec-join.elims*)
  show *line-integral F basis* (*rec-join* ((−*1*,γ) # *l*)) = *line-integral F basis* ((*reversepath* γ) +++ (*rec-join l*))
    using *assms* **by** (*auto simp add*: *l-nempty*)
qed

lemma *one-chain-line-integral-rec-join*:
  assumes *l-props*: *set l* = *one-chain distinct l valid-chain-list l* **and**
    boundary-chain: *boundary-chain one-chain* **and**
    line-integral-exists: ∀ (*k*::*int*, γ) ∈ *one-chain*. *line-integral-exists F basis* γ **and**
    valid-path: ∀ (*k*::*int*, γ) ∈ *one-chain*. *valid-path* γ **and**
    finite-basis: *finite basis*
  shows *line-integral F basis* (*rec-join l*) = *one-chain-line-integral F basis one-chain*
proof −
  have *0*: *sum-list* (*map* (λ((*k*::*int*), γ). (*k*::*int*) ∗ (*line-integral F basis* γ)) *l*) = *one-chain-line-integral F basis one-chain*
    unfolding *one-chain-line-integral-def*
   using *l-props Groups-List.comm-monoid-add-class.sum.distinct-set-conv-list*[*OF l-props*(*2*), *of* (λ(*k*, γ). (*k*::*int*) ∗ (*line-integral F basis* γ))]
    by *auto*
  have *valid-chain-list l* ⟹
            boundary-chain (*set l*) ⟹
            (∀ (*k*::*int*, γ) ∈ *set l*. *line-integral-exists F basis* γ) ⟹
            (∀ (*k*::*int*, γ) ∈ *set l*. *valid-path* γ) ⟹
            line-integral F basis (*rec-join l*) = *sum-list* (*map* (λ(*k*::*int*, γ). *k* ∗ (*line-integral F basis* γ)) *l*)
  proof (*induction l*)
    case *Nil*
    show *?case*
      unfolding *line-integral-def boundary-chain-def*
      apply (*auto*)
    proof
      have ∀ *x*. (*vector-derivative* (λ*x*. *0*) (*at x*)) = *0*
        using *Derivative.vector-derivative-const-at*
        by *auto*
      then have ∀ *x*. ((λ*x*. *0*) *has-vector-derivative 0*) (*at x*)
        using *Derivative.vector-derivative-const-at*
        by *auto*
      then have ∀ *x*. ((λ*x*. *0*) *has-vector-derivative 0*) (*at x within* {*0*..*1*})
        using *Derivative.vector-derivative-const-at*

by *auto*
then have *0*: ∀ *x*∈{*0..1*}. (*vector-derivative* (λ*x*. *0*) (*at x within*{*0..1*})) = *0*
by (*metis* (*no-types*) *box-real*(*2*) *vector-derivative-within-cbox zero-less-one*)
have (∀ *x*∈{*0..1*}. (∑ *b*∈*basis*. *F 0* · *b* ∗ (*vector-derivative* (λ*x*. *0*) (*at x within* {*0..1*}) · *b*)) = *0*)
by (*simp add*: *0*)
then show ((λ*x*. ∑ *b*∈*basis*. *F 0* · *b* ∗ (*vector-derivative* (λ*x*. *0*) (*at x within* {*0..1*}) · *b*)) *has-integral 0*) {*0..1*}
by (*meson has-integral-is-0*)
qed
next
case *ass*: (*Cons a l*)
obtain *k*::*int* **and**
γ::*one-cube* **where** *props*: *a* = (*k*,γ)
proof
let *?k2* = *fst a*
let *?*γ*2* = *snd a*
show *a* = (*?k2*, *?*γ*2*)
by *auto*
qed
have *line-integral-exists F basis* (*rec-join* (*a* # *l*))
using *line-integral-exists-on-rec-join*[*OF ass*(*3*) *ass*(*2*)] *ass*(*5*) *ass*(*4*)
by *blast*
have *boundary-chain* (*set l*)
by (*meson ass.prems*(*2*) *boundary-chain-def list.set-intros*(*2*))
have *val-l*: ⋀*f i*. (*i*,*f*) ∈ *set l* ⟹ *valid-path f*
using *ass.prems*(*4*) **by** *fastforce*
have *vcl-l*: *valid-chain-list l*
by (*metis* (*no-types*) *ass.prems*(*1*) *valid-chain-list.elims*(*3*) *valid-chain-list.simps*(*3*))
have *line-integral-exists-on-joined*:
line-integral-exists F basis (*rec-join l*)
by (*metis* ‹*boundary-chain* (*set l*)› ‹*line-integral-exists F basis* (*rec-join* (*a* # *l*))› *emptyE val-l vcl-l joined-is-valid line-integral-exists-joinD2 line-integral-exists-on-rec-join list.set*(*1*) *neq-Nil-conv rec-join.simps*(*3*))
have *valid-path* (*rec-join* (*a* # *l*))
using *joined-is-valid ass*(*5*) *ass*(*3*) *ass*(*2*) **by** *blast*
then have *joined-is-valid*: *valid-path* (*rec-join l*)
using ‹*boundary-chain* (*set l*)› *val-l vcl-l joined-is-valid* **by** *blast*
show *?case*
proof (*clarsimp*, *cases*)
assume *k-eq-1*: (*k*::*int*) = *1*
have *line-integral-exists-on-gamma*: *line-integral-exists F basis* γ
using *ass props* **by** *auto*
have *gamma-is-valid*: *valid-path* γ
using *ass props* **by** *auto*
have *line-int-rw*: *line-integral F basis* (*rec-join* ((*k*, γ) # *l*)) = *line-integral F basis* (γ +++ *rec-join l*)
proof −
have *gam-int*: *line-integral-exists F basis* γ **using** *ass props* **by** *auto*

have *rec-join-int*: *line-integral-exists F basis* (*rec-join l*)
using *line-integral-exists-on-rec-join*
using *line-integral-exists-on-joined* **by** *blast*
show *?thesis*
using *line-integral-rec-join-cons*[*OF gam-int rec-join-int*] *ass k-eq-1 finite-basis props* **by** *force*
qed
show *line-integral F basis* (*rec-join* (*a # l*)) =
(*case a of* (*x*, γ) ⇒ *real-of-int x* ∗ *line-integral F basis* γ) + ($\sum$ (*x*, γ)←*l*. *real-of-int x* ∗ *line-integral F basis* γ)
apply (*simp add*: *props line-int-rw*)
using *line-integral-distrib*[*OF line-integral-exists-on-gamma line-integral-exists-on-joined gamma-is-valid joined-is-valid*]
ass k-eq-1 vcl-l
by (*auto simp*: *boundary-chain-def props*)
next
assume $k \neq 1$
then have *k-eq-neg-1*: $k = -1$
using *ass props*
by (*auto simp add*: *boundary-chain-def*)
have *line-integral-exists-on-gamma*:
line-integral-exists F basis (*reversepath* γ)
using *line-integral-on-reverse-path ass props*
by *auto*
have *gamma-is-valid*: *valid-path* (*reversepath* γ)
using *valid-path-imp-reverse ass props* **by** *auto*
have *line-int-rw*: *line-integral F basis* (*rec-join* ((*k*, γ) # *l*)) = *line-integral F basis* ((*reversepath* γ) +++ *rec-join l*)
proof −
have *gam-int*: *line-integral-exists F basis* γ **using** *ass props* **by** *auto*
have *rec-join-int*: *line-integral-exists F basis* (*rec-join l*)
using *line-integral-exists-on-rec-join*
using *line-integral-exists-on-joined* **by** *blast*
show *?thesis*
using *line-integral-rec-join-cons-2*[*OF gam-int rec-join-int*]
using *ass k-eq-neg-1*
using *finite-basis props* **by** *blast*
qed
show *line-integral F basis* (*rec-join* (*a # l*)) =
(*case a of* (*x*, γ) ⇒ *real-of-int x* ∗ *line-integral F basis* γ) + ($\sum$ (*x*, γ)←*l*. *real-of-int x* ∗ *line-integral F basis* γ)
apply (*simp add*: *props line-int-rw*)
using *line-integral-distrib*[*OF line-integral-exists-on-gamma line-integral-exists-on-joined gamma-is-valid joined-is-valid*]
props ass line-integral-on-reverse-path(*1*)[*of* γ *F basis*] *k-eq-neg-1*
using ‹*boundary-chain* (*set l*)› *vcl-l* **by** *auto*
qed
qed
then have *1*:*line-integral F basis* (*rec-join l*) = *sum-list* (*map* (λ(*k*::*int*, γ). *k* ∗

(*line-integral F basis γ*)) *l*)
    using *l-props assms* **by** *auto*
  then show *?thesis*
    using *0 1* **by** *auto*
qed

lemma *line-integral-on-path-eq-line-integral-on-equiv-chain*:
  assumes *path-eq-chain*: *chain-subdiv-path γ one-chain* **and**
    boundary-chain: *boundary-chain one-chain* **and**
    line-integral-exists: ∀ (*k*::*int*, *γ*) ∈ *one-chain. line-integral-exists F basis γ* **and**
    valid-path: ∀ (*k*::*int*, *γ*) ∈ *one-chain. valid-path γ* **and**
    finite-basis: *finite basis*
  shows *one-chain-line-integral F basis one-chain = line-integral F basis γ*
    line-integral-exists F basis γ
    valid-path γ
proof −
  obtain *l* **where** *l-props*: *set l = one-chain distinct l rec-join l = γ valid-chain-list l*
    using *chain-subdiv-path.cases path-eq-chain* **by** *force*
  show *line-integral-exists F basis γ*
    using *line-integral-exists-on-rec-join assms l-props*
    by *blast*
  show *valid-path γ*
    using *joined-is-valid assms l-props*
    by *blast*
  have *line-integral F basis* (*rec-join l*) = *one-chain-line-integral F basis one-chain*
    using *one-chain-line-integral-rec-join l-props assms* **by** *auto*
  then show *one-chain-line-integral F basis one-chain = line-integral F basis γ*
    using *l-props*
    by *auto*
qed

lemma *line-integral-on-path-eq-line-integral-on-equiv-chain′*:
  assumes *path-eq-chain*: *chain-subdiv-path γ one-chain* **and**
    boundary-chain: *boundary-chain one-chain* **and**
    line-integral-exists: *line-integral-exists F basis γ* **and**
    valid-path: ∀ (*k*, *γ*) ∈ *one-chain. valid-path γ* **and**
    finite-basis: *finite basis*
  shows *one-chain-line-integral F basis one-chain = line-integral F basis γ*
    ∀ (*k*, *γ*) ∈ *one-chain. line-integral-exists F basis γ*
proof −
  obtain *l* **where** *l-props*: *set l = one-chain distinct l rec-join l = γ valid-chain-list l*
    using *chain-subdiv-path.cases path-eq-chain* **by** *force*
  show *0*: ∀ (*k*, *γ*) ∈ *one-chain. line-integral-exists F basis γ*
    using *line-integral-exists-on-rec-join′ assms l-props*
    by *blast*
  show *one-chain-line-integral F basis one-chain = line-integral F basis γ*
   using *line-integral-on-path-eq-line-integral-on-equiv-chain*(*1*)[*OF assms*(*1*) *assms*(*2*)

0 assms(*4*) *assms*(*5*)] **by** *auto*
qed

definition *chain-subdiv-chain* **where**
chain-subdiv-chain one-chain1 subdiv
$\equiv \exists f.$ ($\bigcup$(*f* ‘ *one-chain1*)) = *subdiv* ∧
(∀ *c*∈*one-chain1*. *chain-subdiv-path* (*coeff-cube-to-path c*) (*f c*)) ∧
pairwise (λ *p p'*. *f p* ∩ *f p'* = {}) *one-chain1* ∧
(∀ *x* ∈ *one-chain1*. *finite* (*f x*))

lemma *chain-subdiv-chain-character*:
shows *chain-subdiv-chain one-chain1 subdiv* ⟷
(∃ *f*. $\bigcup$(*f* ‘ *one-chain1*) = *subdiv* ∧
(∀ (*k*, *γ*)∈*one-chain1*.
if k = *1*
then chain-subdiv-path γ (*f* (*k*, *γ*))
else chain-subdiv-path (*reversepath γ*) (*f* (*k*, *γ*))) ∧
(∀ *p*∈*one-chain1*.
∀ *p'*∈*one-chain1*. *p* ≠ *p'* ⟶ *f p* ∩ *f p'* = {}) ∧
(∀ *x*∈*one-chain1*. *finite* (*f x*)))
unfolding *chain-subdiv-chain-def*
by (*safe*; *intro exI conjI iffI*; *fastforce simp add*: *pairwise-def*)

lemma *chain-subdiv-chain-imp-finite-subdiv*:
assumes *finite one-chain1*
chain-subdiv-chain one-chain1 subdiv
shows *finite subdiv*
using *assms* **by**(*auto simp add*: *chain-subdiv-chain-def*)

lemma *valid-subdiv-imp-valid-one-chain*:
assumes *chain1-eq-chain2*: *chain-subdiv-chain one-chain1 subdiv* **and**
boundary-chain1: *boundary-chain one-chain1* **and**
boundary-chain2: *boundary-chain subdiv* **and**
valid-path: ∀ (*k*, *γ*) ∈ *subdiv*. *valid-path γ*
shows ∀ (*k*, *γ*) ∈ *one-chain1*. *valid-path γ*
proof −
obtain *f* **where** *f-props*:
(($\bigcup$(*f* ‘ *one-chain1*)) = *subdiv*)
(∀ (*k*,*γ*)∈*one-chain1*. *if k* = *1 then chain-subdiv-path γ* (*f*(*k*,*γ*)) *else chain-subdiv-path* (*reversepath γ*) (*f*(*k*,*γ*)))
(∀ *p*∈*one-chain1*. ∀ *p'*∈*one-chain1*. *p* ≠ *p'* ⟶ *f p* ∩ *f p'* = {})
using *chain1-eq-chain2 chain-subdiv-chain-character* **by** *auto*
have $\bigwedge$ *k γ*. (*k*,*γ*) ∈ *one-chain1* ⟹ *valid-path γ*
proof−
fix *k γ*
assume *ass*: (*k*,*γ*) ∈ *one-chain1*
show *valid-path γ*
proof (*cases k* = *1*)
assume *k-eq-1*: *k* = *1*

then have *i*:*chain-subdiv-path* γ (*f*(*k*,γ))
  using *f-props*(*2*) *ass* **by** *auto*
have *ii*:*boundary-chain* (*f*(*k*,γ))
  using *f-props*(*1*) *ass assms*
  apply (*simp add*: *boundary-chain-def*)
  by *blast*
have *iv*: $\forall$(*k*, γ)$\in$*f* (*k*, γ). *valid-path* γ
  using *f-props*(*1*) *ass assms*
  by *blast*
show *?thesis*
  using *valid-path-equiv-valid-chain-list*[*OF i ii iv*]
  by *auto*
next
assume $k \neq 1$
then have *k-eq-neg1*: $k = -1$
  using *ass boundary-chain1*
  by (*auto simp add*: *boundary-chain-def*)
then have *i*:*chain-subdiv-path* (*reversepath* γ) (*f*(*k*,γ))
  using *f-props*(*2*) *ass* **using** ‹$k \neq 1$› **by** *auto*
have *ii*:*boundary-chain* (*f*(*k*,γ))
  using *f-props*(*1*) *ass assms* **by** (*auto simp add*: *boundary-chain-def*)
have *iv*: $\forall$(*k*, γ)$\in$*f* (*k*, γ). *valid-path* γ
  using *f-props*(*1*) *ass assms*
  by *blast*
have *valid-path* (*reversepath* γ)
  using *valid-path-equiv-valid-chain-list*[*OF i ii iv*]
  by *auto*
then show *?thesis*
  using *reversepath-reversepath valid-path-imp-reverse*
  by *force*
qed
qed
then show *valid-path1*: $\forall$(*k*, γ) $\in$ *one-chain1*. *valid-path* γ
  by *auto*
qed

lemma *one-chain-line-integral-eq-line-integral-on-sudivision*:
assumes *chain1-eq-chain2*: *chain-subdiv-chain one-chain1 subdiv* **and**
boundary-chain1: *boundary-chain one-chain1* **and**
boundary-chain2: *boundary-chain subdiv* **and**
line-integral-exists-on-chain2: $\forall$(*k*, γ) $\in$ *subdiv*. *line-integral-exists F basis* γ **and**
valid-path: $\forall$(*k*, γ) $\in$ *subdiv*. *valid-path* γ **and**
finite-chain1: *finite one-chain1* **and**
finite-basis: *finite basis*
shows *one-chain-line-integral F basis one-chain1* = *one-chain-line-integral F basis subdiv*
$\forall$(*k*, γ) $\in$ *one-chain1*. *line-integral-exists F basis* γ
proof −

obtain *f* **where** *f-props*:
((⋃(*f* ‘ *one-chain1*)) = *subdiv*)
(∀ (*k*,γ)∈*one-chain1*. *if k* = *1 then chain-subdiv-path* γ (*f*(*k*,γ)) *else chain-subdiv-path* (*reversepath* γ) (*f*(*k*,γ)))
(∀ *p*∈*one-chain1*. ∀ *p'*∈*one-chain1*. *p* ≠ *p'* ⟶ *f p* ∩ *f p'* = {})
(∀ *x* ∈ *one-chain1*. *finite* (*f x*))
using *chain1-eq-chain2 chain-subdiv-chain-character* **by** (*auto simp add*: *pairwise-def chain-subdiv-chain-def*)
then have *0*: *one-chain-line-integral F basis subdiv* = *one-chain-line-integral F basis* (⋃(*f* ‘ *one-chain1*))
by *auto*
have *finite-chain2*: *finite subdiv*
using *finite-chain1 f-props*(*1*) *f-props*(*4*)
apply (*simp add*: *image-def*)
using *f-props*(*1*) **by** *auto*
have ⋀ *k* γ. (*k*,γ) ∈ *one-chain1* ⟹ *line-integral-exists F basis* γ
proof−
fix *k* γ
assume *ass*: (*k*,γ) ∈ *one-chain1*
show *line-integral-exists F basis* γ
proof (*cases k* = *1*)
assume *k-eq-1*: *k* = *1*
then have *i*:*chain-subdiv-path* γ (*f*(*k*,γ))
using *f-props*(*2*) *ass* **by** *auto*
have *ii*:*boundary-chain* (*f*(*k*,γ))
using *f-props*(*1*) *ass assms* **by** (*auto simp add*: *boundary-chain-def*)
have *iii*:∀ (*k*, γ)∈*f* (*k*, γ). *line-integral-exists F basis* γ
using *f-props*(*1*) *ass assms*
by *blast*
have *iv*: ∀ (*k*, γ)∈*f* (*k*, γ). *valid-path* γ
using *f-props*(*1*) *ass assms*
by *blast*
show *?thesis*
using *line-integral-on-path-eq-line-integral-on-equiv-chain*(*2*)[*OF i ii iii iv finite-basis*]
by *auto*
next
assume *k* ≠ *1*
then have *k-eq-neg1*: *k* = −*1*
using *ass boundary-chain1*
by (*auto simp add*: *boundary-chain-def*)
then have *i*:*chain-subdiv-path* (*reversepath* γ) (*f*(*k*,γ))
using *f-props*(*2*) *ass* **by** *auto*
have *ii*:*boundary-chain* (*f*(*k*,γ))
using *f-props*(*1*) *ass assms* **by** (*auto simp add*: *boundary-chain-def*)
have *iii*:∀ (*k*, γ)∈*f* (*k*, γ). *line-integral-exists F basis* γ
using *f-props*(*1*) *ass assms*
by *blast*
have *iv*: ∀ (*k*, γ)∈*f* (*k*, γ). *valid-path* γ

using *f-props*(*1*) *ass assms*
by *blast*
have *x*:*line-integral-exists F basis* (*reversepath* γ)
using *line-integral-on-path-eq-line-integral-on-equiv-chain*(*2*)[*OF i ii iii iv finite-basis*]
by *auto*
have *valid-path* (*reversepath* γ)
using *line-integral-on-path-eq-line-integral-on-equiv-chain*(*3*)[*OF i ii iii iv finite-basis*]
by *auto*
then show *?thesis*
using *line-integral-on-reverse-path*(*2*) *reversepath-reversepath x*
by *fastforce*
qed
qed
then show *line-integral-exists-on-chain1*: $\forall (k, \gamma) \in$ *one-chain1*. *line-integral-exists F basis* γ
by *auto*
have *1*: *one-chain-line-integral F basis* ($\bigcup$(*f* ' *one-chain1*)) = *one-chain-line-integral F basis one-chain1*
proof −
have *0*:*one-chain-line-integral F basis* ($\bigcup$(*f* ' *one-chain1*)) =
($\sum$ *one-chain* $\in$ (*f* ' *one-chain1*). *one-chain-line-integral F basis one-chain*)
proof −
have *finite*: $\forall$ *chain* $\in$ (*f* ' *one-chain1*). *finite chain*
using *f-props*(*1*) *finite-chain2*
by (*meson Sup-upper finite-subset*)
have *disj*: $\forall$ *A*$\in$*f* ' *one-chain1*. $\forall$ *B*$\in$*f* ' *one-chain1*. $A \neq B \longrightarrow A \cap B = \{\}$
by (*metis* (*no-types*, *opaque-lifting*) *f-props*(*3*) *image-iff*)
show *one-chain-line-integral F basis* ($\bigcup$(*f* ' *one-chain1*)) =
($\sum$ *one-chain* $\in$ (*f* ' *one-chain1*). *one-chain-line-integral F basis one-chain*)
using *Groups-Big.comm-monoid-add-class.sum.Union-disjoint*[*OF finite disj*]
one-chain-line-integral-def
by *auto*
qed
have *1*:($\sum$ *one-chain* $\in$ (*f* ' *one-chain1*). *one-chain-line-integral F basis one-chain*) =
one-chain-line-integral F basis one-chain1
proof −
have ($\sum$ *one-chain* $\in$ (*f* ' *one-chain1*). *one-chain-line-integral F basis one-chain*) =
($\sum$ (k,γ)$\in$*one-chain1*. $k*$(*line-integral F basis* γ))
proof −
have *i*:($\sum$ *one-chain* $\in$ (*f* ' (*one-chain1* − {*p*. *f p* = {}})). *one-chain-line-integral F basis one-chain*) =
($\sum$ (k,γ)$\in$*one-chain1* − {*p*. *f p* = {}}. $k*$(*line-integral F basis* γ))

proof −
have *inj-on f* (*one-chain1* − {*p*. *f p* = {}})
unfolding *inj-on-def* **using** *f-props*(*3*) **by** *blast*
then have *0*: ($\sum$ *one-chain* ∈ (*f* ‘ (*one-chain1* − {*p*. *f p* = {}})). *one-chain-line-integral F basis one-chain*)
= ($\sum$ (*k*, γ) ∈ (*one-chain1* − {*p*. *f p* = {}}). *one-chain-line-integral F basis* (*f* (*k*, γ)))
using *Groups-Big.comm-monoid-add-class.sum.reindex*
by *auto*
have $\bigwedge$ *k* γ. (*k*, γ) ∈ (*one-chain1* − {*p*. *f p* = {}}) ⟹
one-chain-line-integral F basis (*f*(*k*, γ)) = *k*∗ (*line-integral F basis* γ)
proof−
fix *k* γ
assume *ass*: (*k*, γ) ∈ (*one-chain1* − {*p*. *f p* = {}})
have *bchain*: *boundary-chain* (*f*(*k*,γ))
using *f-props*(*1*) *boundary-chain2 ass*
by (*auto simp add*: *boundary-chain-def*)
have *wexist*: ∀ (*k*, γ)∈(*f*(*k*,γ)). *line-integral-exists F basis* γ
using *f-props*(*1*) *line-integral-exists-on-chain2 ass*
by *blast*
have *vpath*: ∀ (*k*, γ)∈(*f*(*k*, γ)). *valid-path* γ
using *f-props*(*1*) *assms ass*
by *blast*
show *one-chain-line-integral F basis* (*f* (*k*, γ)) = *k* ∗ *line-integral F basis* γ
proof(*cases k* = *1*)
assume *k-eq-1*: *k* = *1*
have *one-chain-line-integral F basis* (*f* (*k*, γ)) = *line-integral F basis* γ
using *f-props*(*2*) *k-eq-1 line-integral-on-path-eq-line-integral-on-equiv-chain bchain wexist vpath ass finite-basis*
by *auto*
then show *one-chain-line-integral F basis* (*f* (*k*, γ)) = *k* ∗ *line-integral F basis* γ
using *k-eq-1* **by** *auto*
next
assume *k* ≠ *1*
then have *k-eq-neg1*: *k* = −*1*
using *ass boundary-chain1*
by (*auto simp add*: *boundary-chain-def*)
have *one-chain-line-integral F basis* (*f* (*k*, γ)) = *line-integral F basis* (*reversepath* γ)
using *f-props*(*2*) *k-eq-neg1 line-integral-on-path-eq-line-integral-on-equiv-chain bchain wexist vpath ass finite-basis*
by *auto*
then have *one-chain-line-integral F basis* (*f* (*k*, γ)) = − (*line-integral F basis* γ)
using *line-integral-on-reverse-path*(*1*) *ass line-integral-exists-on-chain1 valid-subdiv-imp-valid-one-chain*[*OF chain1-eq-chain2 boundary-chain1*

boundary-chain2 valid-path]
by *force*
then show *one-chain-line-integral F basis* $(f\ (k,\ \gamma)) = k * $ *line-integral F basis* γ
using *k-eq-neg1* **by** *auto*
qed
qed
then have $(\sum (k, \gamma) \in (\textit{one-chain1} - \{p.\ f\, p = \{\}\}).$ *one-chain-line-integral F basis* $(f\ (k,\ \gamma)))$
$= (\sum (k,\ \gamma) \in (\textit{one-chain1} - \{p.\ f\ p = \{\}\}).\ k*$ (*line-integral F basis* γ))
by (*auto intro*!: *Finite-Cartesian-Product.sum-cong-aux*)
then show $(\sum$ *one-chain* $\in (f\,\text{`}\,(\textit{one-chain1} - \{p.\ f\, p = \{\}\})).$ *one-chain-line-integral F basis one-chain*) =
$(\sum (k,\ \gamma) \in (\textit{one-chain1} - \{p.\ f\ p = \{\}\}).\ k*$ (*line-integral F basis* γ))
using *0* **by** *auto*
qed
have $\bigwedge (k{::}int)\ \gamma.\ (k,\ \gamma) \in \textit{one-chain1} \Longrightarrow (f\ (k,\ \gamma) = \{\}) \Longrightarrow (k,\ \gamma) \in \{(k',\gamma').\ k'*$ (*line-integral F basis* γ') $= 0\}$
proof−
fix *k*::*int*
fix γ::*one-cube*
assume *ass*:$(k,\ \gamma) \in$ *one-chain1*
$(f\ (k,\ \gamma) = \{\})$
then have *zero-line-integral*:*one-chain-line-integral F basis* $(f\ (k,\ \gamma)) = 0$
using *one-chain-line-integral-def*
by *auto*
have *bchain*: *boundary-chain* $(f(k,\gamma))$
using *f-props*(*1*) *boundary-chain2 ass*
by (*auto simp add*: *boundary-chain-def*)
have *wexist*: $\forall (k,\ \gamma)\in(f(k,\gamma)).$ *line-integral-exists F basis* γ
using *f-props*(*1*) *line-integral-exists-on-chain2 ass*
by *blast*
have *vpath*: $\forall (k,\ \gamma)\in(f(k,\ \gamma)).$ *valid-path* γ
using *f-props*(*1*) *assms ass* **by** *blast*
have *one-chain-line-integral F basis* $(f\ (k,\ \gamma)) = k *$ *line-integral F basis* γ
proof(*cases* $k = 1$)
assume *k-eq-1*: $k = 1$
have *one-chain-line-integral F basis* $(f\ (k,\ \gamma)) =$ *line-integral F basis* γ
using *f-props*(*2*) *k-eq-1 line-integral-on-path-eq-line-integral-on-equiv-chain bchain wexist vpath ass finite-basis*
by *auto*
then show *one-chain-line-integral F basis* $(f\ (k,\ \gamma)) = k *$ *line-integral F basis* γ
using *k-eq-1*
by *auto*
next
assume $k \neq 1$

then have *k-eq-neg1*: $k = -1$
using *ass boundary-chain1*
by (*auto simp add*: *boundary-chain-def*)
have *one-chain-line-integral F basis* (f (k, γ)) = *line-integral F basis* (*reversepath* γ)
using *f-props*(*2*) *k-eq-neg1 line-integral-on-path-eq-line-integral-on-equiv-chain bchain wexist vpath ass finite-basis*
by *auto*
then have *one-chain-line-integral F basis* (f (k, γ)) = − (*line-integral F basis* γ)
using *line-integral-on-reverse-path*(*1*) *ass line-integral-exists-on-chain1 valid-subdiv-imp-valid-one-chain*[*OF chain1-eq-chain2 boundary-chain1 boundary-chain2 valid-path*]
by *force*
then show *one-chain-line-integral F basis* (f (k::*int*, γ)) = k ∗ *line-integral F basis* γ
using *k-eq-neg1* **by** *auto*
qed
then show (k, γ) ∈ {(k'::*int*, γ'). k' ∗ *line-integral F basis* γ' = *0*}
using *zero-line-integral* **by** *auto*
qed
then have *ii*:($\sum$ *one-chain* ∈ (f ‘ (*one-chain1* − {p. f p = {}})). *one-chain-line-integral F basis one-chain*) =
($\sum$ *one-chain* ∈ (f ‘ (*one-chain1*)). *one-chain-line-integral F basis one-chain*)
proof −
have $\bigwedge$*one-chain*. *one-chain* ∈ (f ‘ (*one-chain1*)) − (f ‘ (*one-chain1* − {p. f p = {}})) $\Longrightarrow$
one-chain-line-integral F basis one-chain = *0*
proof −
fix *one-chain*
assume *one-chain* ∈ (f ‘ (*one-chain1*)) − (f ‘ (*one-chain1* − {p. f p = {}}))
then show *one-chain-line-integral F basis one-chain* = *0*
by (*auto simp add*: *one-chain-line-integral-def*)
qed
then have *0*:($\sum$ *one-chain* ∈ f ‘ (*one-chain1*) − (f ‘ (*one-chain1* − {p. f p = {}})). *one-chain-line-integral F basis one-chain*)
= *0*
using *comm-monoid-add-class.sum.neutral* **by** *auto*
then have ($\sum$ *one-chain* ∈ f ‘ (*one-chain1*). *one-chain-line-integral F basis one-chain*)
− ($\sum$ *one-chain* ∈ (f ‘ (*one-chain1* − {p. f p = {}})). *one-chain-line-integral F basis one-chain*)
= *0*
proof −
have *finte*: *finite* (f ‘ *one-chain1*) **using** *finite-chain1* **by** *auto*
show *?thesis*

using *Groups-Big.sum-diff*[*OF finte*, *of* (*f* ‘ (*one-chain1* − {*p*. *f p* = {}}))
one-chain-line-integral F basis]
0
by *auto*
qed
then show ($\sum$ *one-chain* ∈ (*f* ‘ (*one-chain1* − {*p*. *f p* = {}})). *one-chain-line-integral F basis one-chain*) =
($\sum$ *one-chain* ∈ (*f* ‘ (*one-chain1*)). *one-chain-line-integral F basis one-chain*)
by *auto*
qed
have ⋀ (*k*::*int*) γ. (*k*, γ) ∈ *one-chain1* ⟹ (*f* (*k*, γ) = {}) ⟹ *f* (*k*, γ) ∈ {*chain*. *one-chain-line-integral F basis chain* = *0*}
proof−
fix *k*::*int*
fix γ::*one-cube*
assume *ass*:(*k*, γ) ∈ *one-chain1* (*f* (*k*, γ) = {})
then have *one-chain-line-integral F basis* (*f* (*k*, γ)) = *0*
using *one-chain-line-integral-def* **by** *auto*
then show *f* (*k*, γ) ∈ {*p*′. (*one-chain-line-integral F basis p*′ = *0*)}
by *auto*
qed
then have *iii*:($\sum$ (*k*::*int*,γ)∈*one-chain1* − {*p*. *f p* = {}}. *k*∗(*line-integral F basis* γ))
= ($\sum$ (*k*::*int*,γ)∈*one-chain1*. *k*∗(*line-integral F basis* γ))
proof−
have ⋀ *k* γ. (*k*,γ)∈*one-chain1* − (*one-chain1* − {*p*. *f p* = {}})
⟹ *k*∗ (*line-integral F basis* γ) = *0*
proof−
fix *k* γ
assume *ass*: (*k*,γ)∈*one-chain1* − (*one-chain1* − {*p*. *f p* = {}})
then have *f*(*k*, γ) = {}
by *auto*
then have *one-chain-line-integral F basis* (*f*(*k*, γ)) = *0*
by (*auto simp add*: *one-chain-line-integral-def*)
then have *zero-line-integral*:*one-chain-line-integral F basis* (*f* (*k*, γ)) = *0*
using *one-chain-line-integral-def* **by** *auto*
have *bchain*: *boundary-chain* (*f*(*k*,γ))
using *f-props*(*1*) *boundary-chain2 ass*
by (*auto simp add*: *boundary-chain-def*)
have *wexist*: ∀ (*k*, γ)∈(*f*(*k*,γ)). *line-integral-exists F basis* γ
using *f-props*(*1*) *line-integral-exists-on-chain2 ass*
by *blast*
have *vpath*: ∀ (*k*, γ)∈(*f*(*k*, γ)). *valid-path* γ
using *f-props*(*1*) *assms ass*
by *blast*

have *one-chain-line-integral F basis* (f (k, γ)) = k ∗ *line-integral F basis* γ
proof(*cases* k = *1*)
assume *k-eq-1*: k = *1*
have *one-chain-line-integral F basis* (f (k, γ)) = *line-integral F basis* γ
using *f-props*(*2*) *k-eq-1 line-integral-on-path-eq-line-integral-on-equiv-chain bchain wexist vpath ass finite-basis*
by *auto*
then show *one-chain-line-integral F basis* (f (k, γ)) = k ∗ *line-integral F basis* γ
using *k-eq-1*
by *auto*
next
assume k ≠ *1*
then have *k-eq-neg1*: k = −*1*
using *ass boundary-chain1*
by (*auto simp add*: *boundary-chain-def*)
have *one-chain-line-integral F basis* (f (k, γ)) = *line-integral F basis* (*reversepath* γ)
using *f-props*(*2*) *k-eq-neg1 line-integral-on-path-eq-line-integral-on-equiv-chain bchain wexist vpath ass finite-basis*
by *auto*
then have *one-chain-line-integral F basis* (f (k, γ)) = − (*line-integral F basis* γ)
using *line-integral-on-reverse-path*(*1*) *ass line-integral-exists-on-chain1 valid-subdiv-imp-valid-one-chain*[*OF chain1-eq-chain2 boundary-chain1 boundary-chain2 valid-path*]
by *force*
then show *one-chain-line-integral F basis* (f (k, γ)) = k ∗ *line-integral F basis* γ
using *k-eq-neg1*
by *auto*
qed
then show k∗ (*line-integral F basis* γ) = *0*
using *zero-line-integral*
by *auto*
qed
then have ∀ (k::*int*,γ)∈*one-chain1* − (*one-chain1* − {p. f p = {}}). k∗ (*line-integral F basis* γ) = *0* **by** *auto*
then have ($\sum$ (k::*int*,γ)∈*one-chain1* − (*one-chain1* − {p. f p = {}}). k∗(*line-integral F basis* γ)) = *0*
using *Groups-Big.comm-monoid-add-class.sum.neutral*
[*of one-chain1* − (*one-chain1* − {p. f p = {}}) (λ(k::*int*,γ). k∗ (*line-integral F basis* γ))]
by (*simp add*: *split-beta*)
then have ($\sum$ (k::*int*,γ)∈*one-chain1*. k∗(*line-integral F basis* γ)) −
($\sum$ (k::*int*,γ)∈ (*one-chain1* − {p. f p = {}}). k∗(*line-integral F basis* γ)) = *0*
using *Groups-Big.sum-diff*[*OF finite-chain1*]

by (*metis* (*no-types*) *Diff-subset* ‹(∑(k, γ)∈*one-chain1* − (*one-chain1* − {p. f p = {}}). k ∗ *line-integral F basis* γ) = *0*› ‹⋀f B. B ⊆ *one-chain1* ⟹ *sum f* (*one-chain1* − B) = *sum f one-chain1* − *sum f B*›)
then show *?thesis* **by** *auto*
qed
show *?thesis* **using** *i ii iii* **by** *auto*
qed
then show *?thesis* **using** *one-chain-line-integral-def* **by** *auto*
qed
show *?thesis* **using** *0 1* **by** *auto*
qed
show *one-chain-line-integral F basis one-chain1* = *one-chain-line-integral F basis subdiv* **using** *0 1* **by** *auto*
qed

lemma *one-chain-line-integral-eq-line-integral-on-sudivision′*:
assumes *chain1-eq-chain2*: *chain-subdiv-chain one-chain1 subdiv* **and**
boundary-chain1: *boundary-chain one-chain1* **and**
boundary-chain2: *boundary-chain subdiv* **and**
line-integral-exists-on-chain1: ∀(k, γ) ∈ *one-chain1*. *line-integral-exists F basis* γ **and**
valid-path: ∀(k, γ) ∈ *subdiv*. *valid-path* γ **and**
finite-chain1: *finite one-chain1* **and**
finite-basis: *finite basis*
shows *one-chain-line-integral F basis one-chain1* = *one-chain-line-integral F basis subdiv*
∀(k, γ) ∈ *subdiv*. *line-integral-exists F basis* γ
proof −
obtain *f* **where** *f-props*:
((⋃(f ‘ *one-chain1*)) = *subdiv*)
(∀(k,γ)∈*one-chain1*. *if* k = *1* *then chain-subdiv-path* γ (f(k,γ)) *else chain-subdiv-path* (*reversepath* γ) (f(k,γ)))
(∀p∈*one-chain1*. ∀p′∈*one-chain1*. p ≠ p′ ⟶ f p ∩ f p′ = {})
(∀x ∈ *one-chain1*. *finite* (f x))
using *chain1-eq-chain2 chain-subdiv-chain-character* **by** (*auto simp add*: *pairwise-def chain-subdiv-chain-def*)
have *finite-chain2*: *finite subdiv*
using *finite-chain1 f-props*(*1*) *f-props*(*4*) **by** *blast*
have ⋀k γ. (k, γ) ∈ *subdiv* ⟹ *line-integral-exists F basis* γ
proof −
fix k γ
assume *ass*: (k, γ) ∈ *subdiv*
then obtain k′ γ′ **where** *kp-gammap*: (k′,γ′) ∈ *one-chain1* (k,γ) ∈ f(k′,γ′)
using *f-props*(*1*) **by** *fastforce*
show *line-integral-exists F basis* γ
proof (*cases* k′ = *1*)
assume *k-eq-1*: k′ = *1*
then have *i*:*chain-subdiv-path* γ′ (f(k′,γ′))
using *f-props*(*2*) *kp-gammap ass* **by** *auto*

```
      have ii:boundary-chain (f(k',γ'))
      using f-props(1) ass assms kp-gammap  by (meson UN-I boundary-chain-def)
      have iii:line-integral-exists F basis γ'
        using assms kp-gammap by blast
      have iv:  ∀(k, γ)∈f (k', γ'). valid-path γ
        using f-props(1) ass assms kp-gammap by blast
      show ?thesis
         using line-integral-on-path-eq-line-integral-on-equiv-chain'(2)[OF i ii iii iv
finite-basis] kp-gammap
        by auto
    next
      assume k' ≠ 1
      then have k-eq-neg1: k' = -1
        using boundary-chain1 kp-gammap
        by (auto simp add: boundary-chain-def)
      then have i:chain-subdiv-path (reversepath γ') (f(k',γ'))
        using f-props(2) kp-gammap by auto
      have ii:boundary-chain (f(k',γ'))
        using f-props(1) assms kp-gammap  by (meson UN-I boundary-chain-def)
      have iii:  ∀(k, γ)∈f (k', γ'). valid-path γ
        using f-props(1) ass assms kp-gammap by blast
      have iv: valid-path (reversepath γ')
        using valid-path-equiv-valid-chain-list[OF i ii iii]
        by force
      have line-integral-exists F basis γ'
        using assms kp-gammap  by blast
      then have x: line-integral-exists F basis (reversepath γ')
        using iv line-integral-on-reverse-path(2) valid-path-reversepath
        by fastforce
      show ?thesis
         using line-integral-on-path-eq-line-integral-on-equiv-chain'(2)[OF i ii x iii
finite-basis] kp-gammap
        by auto
    qed
  qed
  then show ∀(k, γ)∈subdiv. line-integral-exists F basis γ by auto
  then show one-chain-line-integral F basis one-chain1 = one-chain-line-integral
F basis subdiv
    using one-chain-line-integral-eq-line-integral-on-sudivision(1) assms
    by auto
qed

lemma line-integral-sum-gen:
  assumes finite-basis:
    finite basis and
    line-integral-exists:
    line-integral-exists F basis1 γ
    line-integral-exists F basis2 γ and
    basis-partition:
```

basis1 ∪ *basis2* = *basis basis1* ∩ *basis2* = {}
shows *line-integral F basis* γ = (*line-integral F basis1* γ) + (*line-integral F basis2* γ)
line-integral-exists F basis γ
apply (*simp add*: *line-integral-def*)
proof −
have *0*: *integral* {*0..1*} (λ*x*. (∑ *b*∈*basis1*. *F* (γ *x*) · *b* ∗ (*vector-derivative* γ (*at x within* {*0..1*}) · *b*)) +
(∑ *b*∈*basis2*. *F* (γ *x*) · *b* ∗ (*vector-derivative* γ (*at x within* {*0..1*}) · *b*))) =
integral {*0..1*} (λ*x*. ∑ *b*∈*basis1*. *F* (γ *x*) · *b* ∗ (*vector-derivative* γ (*at x within* {*0..1*}) · *b*)) +
integral {*0..1*} (λ*x*. ∑ *b*∈*basis2*. *F* (γ *x*) · *b* ∗ (*vector-derivative* γ (*at x within* {*0..1*}) · *b*))
using *Henstock-Kurzweil-Integration.integral-add line-integral-exists*
by (*auto simp add*: *line-integral-exists-def*)
have *1*: *integral* {*0..1*} (λ*x*. ∑ *b*∈*basis*. *F* (γ *x*) · *b* ∗ (*vector-derivative* γ (*at x within* {*0..1*}) · *b*)) =
integral {*0..1*} (λ*x*. (∑ *b*∈*basis1*. *F* (γ *x*) · *b* ∗ (*vector-derivative* γ (*at x within* {*0..1*}) · *b*)) +
(∑ *b*∈*basis2*. *F* (γ *x*) · *b* ∗ (*vector-derivative* γ (*at x within* {*0..1*}) · *b*)))
by (*metis* (*mono-tags*, *lifting*) *basis-partition finite-Un finite-basis sum.union-disjoint*)
show *integral* {*0..1*} (λ*x*. ∑ *b*∈*basis*. *F* (γ *x*) · *b* ∗ (*vector-derivative* γ (*at x within* {*0..1*}) · *b*)) =
integral {*0..1*} (λ*x*. ∑ *b*∈*basis1*. *F* (γ *x*) · *b* ∗ (*vector-derivative* γ (*at x within* {*0..1*}) · *b*)) +
integral {*0..1*} (λ*x*. ∑ *b*∈*basis2*. *F* (γ *x*) · *b* ∗ (*vector-derivative* γ (*at x within* {*0..1*}) · *b*))
using *0 1*
by *auto*
have *2*: ((λ*x*. (∑ *b*∈*basis1*. *F* (γ *x*) · *b* ∗ (*vector-derivative* γ (*at x within* {*0..1*}) · *b*)) +
(∑ *b*∈*basis2*. *F* (γ *x*) · *b* ∗ (*vector-derivative* γ (*at x within* {*0..1*}) · *b*))) *has-integral*
integral {*0..1*} (λ*x*. ∑ *b*∈*basis1*. *F* (γ *x*) · *b* ∗ (*vector-derivative* γ (*at x within* {*0..1*}) · *b*)) +
integral {*0..1*} (λ*x*. ∑ *b*∈*basis2*. *F* (γ *x*) · *b* ∗ (*vector-derivative* γ (*at x within* {*0..1*}) · *b*))) {*0..1*}
using *Henstock-Kurzweil-Integration.has-integral-add line-integral-exists has-integral-integral*
apply (*auto simp add*: *line-integral-exists-def*)
by *blast*
have *3*: (λ*x*. ∑ *b*∈*basis*. *F* (γ *x*) · *b* ∗ (*vector-derivative* γ (*at x within* {*0..1*}) · *b*)) =
(λ*x*. (∑ *b*∈*basis1*. *F* (γ *x*) · *b* ∗ (*vector-derivative* γ (*at x within* {*0..1*}) · *b*)) +
(∑ *b*∈*basis2*. *F* (γ *x*) · *b* ∗ (*vector-derivative* γ (*at x within* {*0..1*}) · *b*)))
by (*metis* (*mono-tags*, *lifting*) *basis-partition finite-Un finite-basis sum.union-disjoint*)

show *line-integral-exists F basis* γ
apply(*auto simp add*: *line-integral-exists-def has-integral-integral*)
using *2 3*
using *has-integral-integrable-integral* **by** *fastforce*
qed

definition *common-boundary-sudivision-exists* **where**
common-boundary-sudivision-exists one-chain1 one-chain2 $\equiv$
$\exists$ *subdiv. chain-subdiv-chain one-chain1 subdiv* $\wedge$
chain-subdiv-chain one-chain2 subdiv $\wedge$
($\forall$ (k, γ) $\in$ *subdiv. valid-path* γ)$\wedge$
boundary-chain subdiv

lemma *common-boundary-sudivision-commutative*:
(*common-boundary-sudivision-exists one-chain1 one-chain2*) = (*common-boundary-sudivision-exists one-chain2 one-chain1*)
apply (*simp add*: *common-boundary-sudivision-exists-def*)
by *blast*

lemma *common-subdivision-imp-eq-line-integral*:
assumes (*common-boundary-sudivision-exists one-chain1 one-chain2*)
boundary-chain one-chain1
boundary-chain one-chain2
$\forall$ (k, γ)$\in$*one-chain1*. *line-integral-exists F basis* γ
finite one-chain1
finite one-chain2
finite basis
shows *one-chain-line-integral F basis one-chain1* = *one-chain-line-integral F basis one-chain2*
$\forall$ (k, γ)$\in$*one-chain2*. *line-integral-exists F basis* γ
proof −
obtain *subdiv* **where** *subdiv-props*:
chain-subdiv-chain one-chain1 subdiv
chain-subdiv-chain one-chain2 subdiv
($\forall$ (k, γ) $\in$ *subdiv. valid-path* γ)
(*boundary-chain subdiv*)
using *assms*
by (*auto simp add*: *common-boundary-sudivision-exists-def*)
have *i*: $\forall$ (k, γ)$\in$*subdiv*. *line-integral-exists F basis* γ
using *one-chain-line-integral-eq-line-integral-on-sudivision′*(*2*)[*OF subdiv-props*(*1*) *assms*(*2*) *subdiv-props*(*4*) *assms*(*4*) *subdiv-props*(*3*) *assms*(*5*) *assms*(*7*)]
by *auto*
show *one-chain-line-integral F basis one-chain1* = *one-chain-line-integral F basis one-chain2*
using *one-chain-line-integral-eq-line-integral-on-sudivision′*(*1*)[*OF subdiv-props*(*1*) *assms*(*2*) *subdiv-props*(*4*) *assms*(*4*) *subdiv-props*(*3*) *assms*(*5*) *assms*(*7*)]
one-chain-line-integral-eq-line-integral-on-sudivision(*1*)[*OF subdiv-props*(*2*) *assms*(*3*) *subdiv-props*(*4*) *i subdiv-props*(*3*) *assms*(*6*) *assms*(*7*)]
by *auto*

show ∀ (*k*, *γ*)∈*one-chain2*. *line-integral-exists F basis γ*
using *one-chain-line-integral-eq-line-integral-on-sudivision*(*2*)[*OF subdiv-props*(*2*) *assms*(*3*) *subdiv-props*(*4*) *i subdiv-props*(*3*) *assms*(*6*) *assms*(*7*)]
by *auto*
qed

definition *common-sudiv-exists* **where**
common-sudiv-exists one-chain1 one-chain2 ≡
∃ *subdiv ps1 ps2*. *chain-subdiv-chain* (*one-chain1* − *ps1*) *subdiv* ∧
chain-subdiv-chain (*one-chain2* − *ps2*) *subdiv* ∧
(∀ (*k*, *γ*) ∈ *subdiv*. *valid-path γ*) ∧
(*boundary-chain subdiv*) ∧
(∀ (*k*, *γ*) ∈ *ps1*. *point-path γ*) ∧
(∀ (*k*, *γ*) ∈ *ps2*. *point-path γ*)

lemma *common-sudiv-exists-comm*:
shows *common-sudiv-exists C1 C2* = *common-sudiv-exists C2 C1*
by (*auto simp add*: *common-sudiv-exists-def*)

lemma *line-integral-degenerate-chain*:
assumes (∀ (*k*, *γ*) ∈ *chain*. *point-path γ*)
assumes *finite basis*
shows *one-chain-line-integral F basis chain* = *0*
proof (*simp add*: *one-chain-line-integral-def*)
have ∀ (*k*,*g*)∈*chain*. *line-integral F basis g* = *0*
using *assms line-integral-point-path*
by *blast*
then have ∀ (*k*,*g*)∈*chain*. *real-of-int k* ∗ *line-integral F basis g* = *0* **by** *auto*
then have ⋀*p*. *p* ∈ *chain* ⟹ (*case p of* (*i*, *f*) ⇒ *real-of-int i* ∗ *line-integral F basis f*) = *0*
by *fastforce*
then show (∑ *x*∈*chain*. *case x of* (*k*, *g*) ⇒ *real-of-int k* ∗ *line-integral F basis g*) = *0*
by *simp*
qed

lemma *gen-common-subdiv-imp-common-subdiv*:
shows (*common-sudiv-exists one-chain1 one-chain2*) = (∃ *ps1 ps2*. (*common-boundary-sudivision-exists* (*one-chain1* − *ps1*) (*one-chain2* − *ps2*)) ∧ (∀ (*k*, *γ*)∈*ps1*. *point-path γ*) ∧ (∀ (*k*, *γ*)∈*ps2*. *point-path γ*))
by (*auto simp add*: *common-sudiv-exists-def common-boundary-sudivision-exists-def*)

lemma *common-subdiv-imp-gen-common-subdiv*:
assumes (*common-boundary-sudivision-exists one-chain1 one-chain2*)
shows (*common-sudiv-exists one-chain1 one-chain2*)
using *assms*
apply (*auto simp add*: *common-sudiv-exists-def common-boundary-sudivision-exists-def*)
by (*metis Diff-empty all-not-in-conv*)

lemma *one-chain-line-integral-point-paths*:
assumes *finite one-chain*
assumes *finite basis*
assumes (∀ (*k*, *γ*)∈*ps*. *point-path γ*)
shows *one-chain-line-integral F basis* (*one-chain* − *ps*) = *one-chain-line-integral F basis* (*one-chain*)
proof−
have *0*:(∀ *x*∈*ps*. *case x of* (*k*, *g*) ⇒ (*real-of-int k* ∗ *line-integral F basis g*) = *0*)
using *line-integral-point-path assms*
by *force*
show *one-chain-line-integral F basis* (*one-chain* − *ps*) = *one-chain-line-integral F basis one-chain*
unfolding *one-chain-line-integral-def* **using** *0* ‹*finite one-chain*›
by (*force simp add*: *intro*: *comm-monoid-add-class.sum.mono-neutral-left*)
qed

lemma *boundary-chain-diff*:
assumes *boundary-chain one-chain*
shows *boundary-chain* (*one-chain* − *s*)
using *assms*
by (*auto simp add*: *boundary-chain-def*)

lemma *gen-common-subdivision-imp-eq-line-integral*:
assumes (*common-sudiv-exists one-chain1 one-chain2*)
boundary-chain one-chain1
boundary-chain one-chain2
∀ (*k*, *γ*)∈*one-chain1*. *line-integral-exists F basis γ*
finite one-chain1
finite one-chain2
finite basis
shows *one-chain-line-integral F basis one-chain1* = *one-chain-line-integral F basis one-chain2*
∀ (*k*, *γ*)∈*one-chain2*. *line-integral-exists F basis γ*
proof −
obtain *ps1 ps2* **where** *gen-subdiv*: (*common-boundary-sudivision-exists* (*one-chain1* − *ps1*) (*one-chain2* − *ps2*)) (∀ (*k*, *γ*)∈*ps1*. *point-path γ*) (∀ (*k*, *γ*)∈*ps2*. *point-path γ*)
using *assms*(*1*) *gen-common-subdiv-imp-common-subdiv*
by *blast*
show *one-chain-line-integral F basis one-chain1* = *one-chain-line-integral F basis one-chain2*
using *one-chain-line-integral-point-paths gen-common-subdiv-imp-common-subdiv assms*(*2*−*7*) *gen-subdiv*
common-subdivision-imp-eq-line-integral(*1*)[*OF gen-subdiv*(*1*) *boundary-chain-diff*[*OF assms*(*2*)] *boundary-chain-diff*[*OF assms*(*3*)]]
by *auto*
show ∀ (*k*, *γ*)∈*one-chain2*. *line-integral-exists F basis γ*
proof−
obtain *subdiv* **where** *subdiv-props*:

```
      chain-subdiv-chain (one-chain1−ps1) subdiv
      chain-subdiv-chain (one-chain2−ps2) subdiv
      (∀ (k, γ) ∈ subdiv. valid-path γ)
      (boundary-chain subdiv)
      using gen-subdiv(1)
      by (auto simp add: common-boundary-sudivision-exists-def)
    have ∀ (k, γ)∈subdiv. line-integral-exists F basis γ
     using one-chain-line-integral-eq-line-integral-on-sudivision′(2)[OF subdiv-props(1)
boundary-chain-diff[OF assms(2)] subdiv-props(4)] assms(4) subdiv-props(3) assms(5)
assms(7)
      by blast
    then have i: ∀ (k, γ)∈one-chain2−ps2. line-integral-exists F basis γ
     using one-chain-line-integral-eq-line-integral-on-sudivision(2)[OF subdiv-props(2)
boundary-chain-diff[OF assms(3)] subdiv-props(4)] subdiv-props(3) assms(6) assms(7)
      by blast
    then show ?thesis
      using  gen-subdiv(3) line-integral-exists-point-path[OF assms(7)]
      by blast
  qed
qed

lemma common-sudiv-exists-refl:
      assumes common-sudiv-exists C1 C2
      shows common-sudiv-exists C2 C1
      using assms
      apply(simp add: common-sudiv-exists-def)
      by auto

lemma chain-subdiv-path-singleton:
  shows chain-subdiv-path γ {(1,γ)}
proof −
  have rec-join [(1,γ)] = γ
    by (simp add: joinpaths-def)
 then have set [(1,γ)] = {(1, γ)} distinct [(1,γ)] rec-join [(1,γ)] = γ valid-chain-list
[(1,γ)]
    by auto
  then show ?thesis
    by (metis (no-types) chain-subdiv-path.intros)
qed

lemma chain-subdiv-path-singleton-reverse:
  shows chain-subdiv-path (reversepath γ) {(−1,γ)}
proof −
  have rec-join [(−1,γ)] = reversepath γ
    by (simp add: joinpaths-def)
  then have set [(−1,γ)] = {(− 1, γ)} distinct [(−1,γ)]
            rec-join [(−1,γ)] = reversepath γ valid-chain-list [(−1,γ)]
    by auto
  then  have chain-subdiv-path (reversepath γ) (set [(− 1, γ)])
```

using *chain-subdiv-path.intros* **by** *blast*
then show *?thesis*
by *simp*
qed

lemma *chain-subdiv-chain-refl*:
assumes *boundary-chain C*
shows *chain-subdiv-chain C C*
using *chain-subdiv-path-singleton chain-subdiv-path-singleton-reverse assms*
unfolding *chain-subdiv-chain-def boundary-chain-def pairwise-def* **using** *case-prodI2 coeff-cube-to-path.simps*
by (*rule-tac x=λx. {x}* **in** *exI*) *auto*

definition *reparam-weak* **where**
reparam-weak γ1 γ2 ≡ ∃ φ. (∀ x∈{0..1}. γ1 x = (γ2 o φ) x) ∧ φ *piecewise-C1-differentiable-on* {0..1} ∧ φ(0) = 0 ∧ φ(1) = 1 ∧ φ ‘ {0..1} = {0..1}

definition *reparam* **where**
reparam γ1 γ2 ≡ ∃ φ. (∀ x∈{0..1}. γ1 x = (γ2 o φ) x) ∧ φ *piecewise-C1-differentiable-on* {0..1} ∧ φ(0) = 0 ∧ φ(1) = 1 ∧ *bij-betw* φ {0..1} {0..1} ∧ φ −‘ {0..1} ⊆ {0..1} ∧ (∀ x∈{0..1}. *finite* (φ −‘ {x}))

lemma *reparam-weak-eq-refl*:
shows *reparam-weak γ1 γ1*
unfolding *reparam-weak-def*
apply (*rule-tac x=id* **in** *exI*)
by (*auto simp add*: *id-def piecewise-C1-differentiable-on-def C1-differentiable-on-def continuous-on-id*)

lemma *line-integral-exists-smooth-one-base*:
assumes γ *C1-differentiable-on* {0..1}
continuous-on (*path-image* γ) (λx. F x · b)
shows *line-integral-exists F* {b} γ
proof−
have *gamma2-differentiable*: (∀ x ∈ {0 .. 1}. γ *differentiable at x*)
using *assms*(1)
by (*auto simp add*: *valid-path-def C1-differentiable-on-eq*)
then have *gamma2-b-component-differentiable*: (∀ x ∈ {0 .. 1}. (λx. (γ x) · b) *differentiable at x*)
by *auto*
then have (λx. (γ x) · b) *differentiable-on* {0..1}
using *differentiable-at-withinI*
by (*auto simp add*: *differentiable-on-def*)
then have *gama2-cont-comp*: *continuous-on* {0..1} (λx. (γ x) · b)
using *differentiable-imp-continuous-on*
by *auto*
have *gamma2-cont*:*continuous-on* {0..1} γ

using *assms*(*1*) *C1-differentiable-imp-continuous-on*
by (*auto simp add*: *valid-path-def*)
have *iii*: *continuous-on* {*0..1*} (λx. *F* (γ *x*) $\cdot$ *b* $*$ (*vector-derivative* γ (*at x within* {*0..1*}) $\cdot$ *b*))
proof−
have *0*: *continuous-on* {*0..1*} (λx. *F* (γ *x*) $\cdot$ *b*)
using *assms*(*2*) *continuous-on-compose*[*OF gamma2-cont*]
by (*auto simp add*: *path-image-def*)
obtain *D* **where** *D*: ($\forall x \in \{0..1\}$. (γ *has-vector-derivative D x*) (*at x*)) $\wedge$ *continuous-on* {*0..1*} *D*
using *assms*(*1*)
by (*auto simp add*: *C1-differentiable-on-def*)
then have $*$:$\forall x \in \{0..1\}$. *vector-derivative* γ (*at x within*{*0..1*}) = *D x*
using *vector-derivative-at vector-derivative-at-within-ivl*
by *fastforce*
then have *continuous-on* {*0..1*} (λx. *vector-derivative* γ (*at x within*{*0..1*}))
using *continuous-on-eq D* **by** *force*
then have *1*: *continuous-on* {*0..1*} (λx. (*vector-derivative* γ (*at x within*{*0..1*})) $\cdot$ *b*)
by(*auto intro*!: *continuous-intros*)
show *?thesis*
using *continuous-on-mult*[*OF 0 1*] **by** *auto*
qed
then have (λx. *F* (γ *x*) $\cdot$ *b* $*$ (*vector-derivative* γ (*at x within* {*0..1*}) $\cdot$ *b*)) *integrable-on* {*0..1*}
using *integrable-continuous-real*
by *auto*
then show *line-integral-exists F* {*b*} γ
by(*auto simp add*: *line-integral-exists-def*)
qed

lemma *contour-integral-primitive-lemma*:
fixes *f* :: *complex* $\Rightarrow$ *complex* **and** *g* :: *real* $\Rightarrow$ *complex*
assumes $a \leq b$
and $\bigwedge x$. $x \in s \Longrightarrow$ (*f has-field-derivative* f' *x*) (*at x within s*)
and *g piecewise-differentiable-on* {*a..b*} $\bigwedge x$. $x \in \{a..b\} \Longrightarrow g\ x \in s$
shows ((λx. f'(*g x*) $*$ *vector-derivative g* (*at x within* {*a..b*}))
has-integral (*f*(*g b*) − *f*(*g a*))) {*a..b*}
proof −
obtain *k* **where** *k*: *finite k* $\forall x \in \{a..b\} - k$. *g differentiable* (*at x within* {*a..b*}) **and** *cg*: *continuous-on* {*a..b*} *g*
using *assms* **by** (*auto simp*: *piecewise-differentiable-on-def*)
have *cfg*: *continuous-on* {*a..b*} (λx. *f* (*g x*))
apply (*rule continuous-on-compose* [*OF cg*, *unfolded o-def*])
using *assms*
apply (*metis field-differentiable-def field-differentiable-imp-continuous-at continuous-on-eq-continuous-within continuous-on-subset image-subset-iff*)
done
{ **fix** *x*::*real*

assume *a*: $a < x$ **and** *b*: $x < b$ **and** *xk*: $x \notin k$
then have *g differentiable at x within* $\{a..b\}$
using *k* **by** (*simp add*: *differentiable-at-withinI*)
then have (*g has-vector-derivative vector-derivative g* (*at x within* $\{a..b\}$)) (*at x within* $\{a..b\}$)
by (*simp add*: *vector-derivative-works has-field-derivative-def scaleR-conv-of-real*)
then have *gdiff*: (*g has-derivative* ($\lambda u.\ u *$ *vector-derivative g* (*at x within* $\{a..b\}$))) (*at x within* $\{a..b\}$)
by (*simp add*: *has-vector-derivative-def scaleR-conv-of-real*)
have (*f has-field-derivative* (f' (*g x*))) (*at* (*g x*) *within g* ' $\{a..b\}$)
using *assms* **by** (*metis a atLeastAtMost-iff b DERIV-subset image-subset-iff less-eq-real-def*)
then have *fdiff*: (*f has-derivative* ($*$) (f' (*g x*))) (*at* (*g x*) *within g* ' $\{a..b\}$)
by (*simp add*: *has-field-derivative-def*)
have (($\lambda x.$ *f* (*g x*)) *has-vector-derivative* f' (*g x*) $*$ *vector-derivative g* (*at x within* $\{a..b\}$)) (*at x within* $\{a..b\}$)
using *diff-chain-within* [*OF gdiff fdiff*]
by (*simp add*: *has-vector-derivative-def scaleR-conv-of-real o-def mult-ac*)
} **note** $* = $ *this*
show *?thesis*
apply (*rule fundamental-theorem-of-calculus-interior-strong*)
using *k assms cfg* $*$
apply (*auto simp*: *at-within-Icc-at*)
done
qed

lemma *line-integral-primitive-lemma*:
fixes *f* :: *'a*::{*euclidean-space*,*real-normed-field*} $\Rightarrow$ *'a*::{*euclidean-space*,*real-normed-field*} **and**
g :: *real* $\Rightarrow$ *'a*
assumes $\bigwedge$(*a*::*'a*). $a \in s \Longrightarrow$ (*f has-field-derivative* (f' *a*)) (*at a within s*)
and *g piecewise-differentiable-on* $\{0$::*real*$..1\}$ $\bigwedge x.\ x \in \{0..1\} \Longrightarrow g\ x \in s$
and *base-vec* $\in$ *Basis*
shows (($\lambda x.$ ((f'(*g x*)) $*$ (*vector-derivative g* (*at x within* $\{0..1\}$))) $\cdot$ *base-vec*)
has-integral (((*f*(*g 1*)) $\cdot$ *base-vec* $-$ (*f*(*g 0*)) $\cdot$ *base-vec*))) $\{0..1\}$
proof $-$
obtain *k* **where** *k*: *finite k* $\forall x \in \{0..1\} - k.$ *g differentiable* (*at x within* $\{0..1\}$) **and** *cg*: *continuous-on* $\{0..1\}$ *g*
using *assms* **by** (*auto simp*: *piecewise-differentiable-on-def*)
have *cfg*: *continuous-on* $\{0..1\}$ ($\lambda x.$ *f* (*g x*))
apply (*rule continuous-on-compose* [*OF cg*, *unfolded o-def*])
using *assms*
apply (*metis field-differentiable-def field-differentiable-imp-continuous-at continuous-on-eq-continuous-within continuous-on-subset image-subset-iff*)
done
{ **fix** *x*::*real*
assume *a*: $0 < x$ **and** *b*: $x < 1$ **and** *xk*: $x \notin k$
then have *g differentiable at x within* $\{0..1\}$
using *k* **by** (*simp add*: *differentiable-at-withinI*)

then have (*g has-vector-derivative vector-derivative g* (*at x within* {*0..1*})) (*at x within* {*0..1*})
by (*simp add*: *vector-derivative-works has-field-derivative-def scaleR-conv-of-real*)
then have *gdiff*: (*g has-derivative* (*λu. of-real u* ∗ *vector-derivative g* (*at x within* {*0..1*}))) (*at x within* {*0..1*})
by (*simp add*: *has-vector-derivative-def scaleR-conv-of-real*)
have (*f has-field-derivative* (*f′* (*g x*))) (*at* (*g x*) *within g* ' {*0..1*})
using *assms* **by** (*metis a atLeastAtMost-iff b DERIV-subset image-subset-iff less-eq-real-def*)
then have *fdiff*: (*f has-derivative* (∗) (*f′* (*g x*))) (*at* (*g x*) *within g* ' {*0..1*})
by (*simp add*: *has-field-derivative-def*)
have ((*λx. f* (*g x*)) *has-vector-derivative f′* (*g x*) ∗ *vector-derivative g* (*at x within* {*0..1*})) (*at x within* {*0..1*})
using *diff-chain-within* [*OF gdiff fdiff*]
by (*simp add*: *has-vector-derivative-def scaleR-conv-of-real o-def mult-ac*)
}
then have ∗: ⋀*x. x*∈{*0*<..<*1*} − *k* ⟹ ((*λx. f* (*g x*)) *has-vector-derivative f′* (*g x*) ∗ *vector-derivative g* (*at x within* {*0..1*})) (*at x within* {*0..1*})
by *auto*
have ((*λx.* ((*f′*(*g x*))) ∗ ((*vector-derivative g* (*at x within* {*0..1*}))))
has-integral (((*f*(*g 1*)) − (*f*(*g 0*))))) {*0..1*}
using *fundamental-theorem-of-calculus-interior-strong*[*OF k*(*1*) *zero-le-one* - *cfg*]
using *k assms cfg* ∗ **by** (*auto simp*: *at-within-Icc-at*)
then have ((*λx.* (((*f′*(*g x*))) ∗ ((*vector-derivative g* (*at x within* {*0..1*})))) · *base-vec*)
has-integral (((*f*(*g 1*)) − (*f*(*g 0*)))) · *base-vec*) {*0..1*}
using *has-integral-componentwise-iff assms*(*4*)
by *fastforce*
then show *?thesis* **using** *inner-mult-left′*
by (*simp add*: *inner-mult-left′ inner-diff-left*)
qed

lemma *reparam-eq-line-integrals*:
assumes *reparam*: *reparam γ1 γ2* **and**
pw-smooth: *γ2 piecewise-C1-differentiable-on* {*0..1*} **and**
cont: *continuous-on* (*path-image γ2*) (*λx. F x* · *b*) **and**
line-integral-ex: *line-integral-exists F* {*b*} *γ2*
shows *line-integral F* {*b*} *γ1* = *line-integral F* {*b*} *γ2*
line-integral-exists F {*b*} *γ1*
proof−
obtain *φ* **where** *phi*: (∀ *x*∈{*0..1*}. *γ1 x* = (*γ2 o φ*) *x*) *φ piecewise-C1-differentiable-on* {*0..1*} *φ*(*0*) = *0 φ*(*1*) = *1bij-betw φ* {*0..1*} {*0..1*} *φ* −' {*0..1*} ⊆ {*0..1*} ∀ *x*∈{*0..1*}. *finite* (*φ* −' {*x*})
using *reparam*
by (*auto simp add*: *reparam-def*)
obtain *s* **where** *s*: *finite s φ C1-differentiable-on* {*0..1*} − *s*
using *phi*
by(*auto simp add*: *reparam-def piecewise-C1-differentiable-on-def*)

```
  let ?s = s ∩ {0..1}
  have s-inter: finite ?s φ C1-differentiable-on {0..1} − ?s
    using s
     apply blast
    by (metis Diff-Compl Diff-Diff-Int Diff-eq inf.commute s(2))
  have cont-phi: continuous-on {0..1} φ
    using phi
   by(auto simp add: reparam-def piecewise-C1-differentiable-on-imp-continuous-on)
  obtain s′ D where s′-D: finite s′ (∀ x ∈ {0 .. 1} − s′. γ2 differentiable at x)
(∀ x∈{0..1} − s′. (γ2 has-vector-derivative D x) (at x)) ∧ continuous-on ({0..1}
− s′) D
    using pw-smooth
   apply (auto simp add: valid-path-def piecewise-C1-differentiable-on-def C1-differentiable-on-eq)
    by (simp add: vector-derivative-works)
  let ?s′ = s′ ∩ {0..1}
  have gamma2-differentiable: finite ?s′ (∀ x ∈ {0 .. 1} − ?s′. γ2 differentiable
at x) (∀ x∈{0..1} − ?s′. (γ2 has-vector-derivative D x) (at x)) ∧ continuous-on
({0..1} − ?s′) D
    using s′-D
      apply blast
    using s′-D(2) apply auto[1]
    by (metis Diff-Int2 inf-top.left-neutral s′-D(3))
  then have gamma2-b-component-differentiable: (∀ x ∈ {0 .. 1} − ?s′. (λx. (γ2
x) · b) differentiable at x)
    using differentiable-inner by force
  then have (λx. (γ2 x) · b) differentiable-on {0..1} − ?s′
    using differentiable-at-withinI
    by (auto simp add: differentiable-on-def)
  then have gama2-cont-comp: continuous-on ({0..1} − ?s′) (λx. (γ2 x) · b)
    using differentiable-imp-continuous-on
    by auto

  have s-in01: ?s ⊆ {0..1} by auto
  have s′-in01: ?s′ ⊆ {0..1} by auto
  have phi-backimg-s′: φ −‘ {0..1} ⊆ {0..1} using phi by auto
  have inj-on φ {0..1} using phi(5) by (auto simp add: bij-betw-def)
  have bij-phi: bij-betw φ {0..1} {0..1} using phi(5) by auto
  have finite-bck-img-single: ∀ x∈{0..1}. finite (φ −‘ {x}) using phi by auto
  then have finite-bck-img-single-s′:  ∀ x∈?s′. finite (φ −‘ {x}) by auto
  have gamma2-line-integrable: (λx. F (γ2 x) · b ∗ (vector-derivative γ2 (at x
within {0..1}) · b)) integrable-on {0..1}
    using line-integral-ex
    by (simp add: line-integral-exists-def)

  have finite-neg-img: finite (φ −‘ ?s′)
    using finite-bck-img-single
   by (metis Int-iff finite-Int gamma2-differentiable(1) image-vimage-eq inf-img-fin-dom′)
  have gamma2-cont:continuous-on ({0..1} − ?s′) γ2
    using  gamma2-differentiable
```

by (*meson continuous-at-imp-continuous-on differentiable-imp-continuous-within*)
have *iii*: *continuous-on* ({*0..1*} − *?s'*) (λ*x*. *F* (γ*2 x*) · *b* ∗ (*vector-derivative* γ*2* (*at x within* {*0..1*}) · *b*))
proof−
have *0*: *continuous-on* ({*0..1*} − *?s'*) (λ*x*. *F* (γ*2 x*) · *b*)
using *cont continuous-on-compose*[*OF gamma2-cont*] *continuous-on-compose2 gamma2-cont*
unfolding *path-image-def* **by** *fastforce*
have *D*: (∀ *x*∈{*0..1*} − *?s'*. (γ*2 has-vector-derivative D x*) (*at x*)) ∧ *continuous-on* ({*0..1*} − *?s'*) *D*
using *gamma2-differentiable*
by *auto*
then have ∗:∀ *x*∈{*0..1*} − *?s'*. *vector-derivative* γ*2* (*at x within*{*0..1*}) = *D x*
using *vector-derivative-at vector-derivative-at-within-ivl*
by *fastforce*
then have *continuous-on* ({*0..1*} − *?s'*) (λ*x*. *vector-derivative* γ*2* (*at x within*{*0..1*}))
using *continuous-on-eq D*
by *metis*
then have *1*: *continuous-on* ({*0..1*} − *?s'*) (λ*x*. (*vector-derivative* γ*2* (*at x within*{*0..1*})) · *b*)
by (*auto intro*!: *continuous-intros*)
show *?thesis*
using *continuous-on-mult*[*OF 0 1*]
by *auto*
qed
have *iv*: φ(*0*) ≤ φ(*1*)
using *phi*(*3*) *phi*(*4*)
by (*simp add*: *reparam-def*)
have *v*: φ‘{*0..1*} ⊆ {*0..1*}
using *phi*
by (*auto simp add*: *reparam-def bij-betw-def*)
obtain *D* **where** *D-props*: (∀ *x*∈{*0..1*} − *?s*. (φ *has-vector-derivative D x*) (*at x*))
using *s*
by (*auto simp add*: *C1-differentiable-on-def*)
then have (⋀*x*. *x* ∈ ({*0..1*} − *?s*) ⟹ (φ *has-vector-derivative D x*) (*at x within* {*0..1*}))
using *has-vector-derivative-at-within*
by *blast*
then have *vi*: (⋀*x*. *x* ∈ ({*0..1*} − *?s*) ⟹ (φ *has-real-derivative D x*) (*at x within* {*0..1*}))
using *has-real-derivative-iff-has-vector-derivative*
by *blast*
have *a*:((λ*x*. *D x* ∗ (*F* (γ*2* (φ *x*)) · *b* ∗ (*vector-derivative* γ*2* (*at* (φ *x*) *within* {*0..1*}) · *b*))) *has-integral*
integral {φ *0*..φ *1*} (λ*x*. *F* (γ*2 x*) · *b* ∗ (*vector-derivative* γ*2* (*at x within* {*0..1*}) · *b*)))
({*0..1*})
proof−

have *a*: *integral* {*φ 1..φ 0*} (*λx. F* (*γ2 x*) · *b* ∗ (*vector-derivative γ2* (*at x within* {*0..1*}) · *b*)) = *0* **using** *integral-singleton integral-empty iv*
by (*simp add*: *phi*(*3*) *phi*(*4*))
have *b*: ((*λx. D x* $*_R$ (*F* (*γ2* (*φ x*)) · *b* ∗ (*vector-derivative γ2* (*at* (*φ x*) *within* {*0..1*}) · *b*))) *has-integral*
integral {*φ 0..φ 1*} (*λx. F* (*γ2 x*) · *b* ∗ (*vector-derivative γ2* (*at x within* {*0..1*}) · *b*)) − *integral* {*φ 1..φ 0*} (*λx. F* (*γ2 x*) · *b* ∗ (*vector-derivative γ2* (*at x within* {*0..1*}) · *b*)))
{*0..1*}
apply(*rule has-integral-substitution-general-′*[*OF s-inter*(*1*) *zero-le-one gamma2-differentiable*(*1*) *v gamma2-line-integrable iii cont-phi finite-bck-img-single-s′*])
proof−
have *surj-on* {*0..1*} *φ*
using *bij-phi*
by (*metis* (*full-types*) *bij-betw-def image-subsetI rangeI*)
then show *surj-on ?s′ φ* **using** *bij-phi s′-in01*
by *blast*
show *inj-on φ* ({*0..1*} ∪ (*?s* ∪ *φ* −‘ *?s′*))
proof−
have *i*: *inj-on φ* {*0..1*} **using** *bij-phi*
using *bij-betw-def* **by** *blast*
have *ii*: ({*0..1*} ∪ (*?s* ∪ *φ* −‘ *?s′*)) = {*0..1*} **using** *phi-backimg-s′ s-in01 s′-in01*
by *blast*
show *?thesis* **using** *i ii* **by** *auto*
qed
show ⋀*x. x* ∈ {*0..1*} − *?s* ⟹ (*φ has-real-derivative D x*) (*at x within* {*0..1*})
using *vi* **by** *blast*
qed
show *?thesis* **using** *a b* **by** *auto*
qed
then have *b*: *integral* {*0..1*} (*λx. D x* ∗ (*F* (*γ2* (*φ x*)) · *b* ∗ (*vector-derivative γ2* (*at* (*φ x*) *within* {*0..1*}) · *b*))) =
integral {*φ 0..φ 1*} (*λx. F* (*γ2 x*) · *b* ∗ (*vector-derivative γ2* (*at x within* {*0..1*}) · *b*))
by *auto*
have *gamma2-vec-diffable*: ⋀*x*::*real. x* ∈ {*0..1*} − ((*φ* −‘ *?s′*) ∪ *?s*) ⟹ ((*γ2 o φ*) *has-vector-derivative vector-derivative* (*γ2* ∘ *φ*) (*at x*)) (*at x*)
proof−
fix *x*::*real*
assume *ass*: *x* ∈ {*0..1*} − ((*φ* −‘ *?s′*) ∪ *?s*)
have *zer-le-x-le-1*:*0*≤ *x* ∧ *x* ≤ *1* **using** *ass*
by *simp*
show ((*γ2 o φ*) *has-vector-derivative vector-derivative* (*γ2* ∘ *φ*) (*at x*)) (*at x*)
proof−
have ∗∗: *γ2 differentiable at* (*φ x*)
using *gamma2-differentiable*(*2*) *ass v*
by *blast*
have ∗∗∗: *φ differentiable at x*

using *s ass*
by (*auto simp add*: *C1-differentiable-on-eq*)
then show ((*γ2 o φ*) *has-vector-derivative vector-derivative* (*γ2* ∘ *φ*) (*at x*)) (*at x*)
using *differentiable-chain-at*[*OF* ∗∗∗ ∗∗]
by (*auto simp add*: *vector-derivative-works*)
qed
qed
then have *gamma2-vec-deriv-within*: ⋀*x*::*real*. *x* ∈ {*0*..*1*} − ((*φ* −‘ *?s′*) ∪ *?s*) ⟹ *vector-derivative* (*γ2* ∘ *φ*) (*at x*) = *vector-derivative* (*γ2* ∘ *φ*) (*at x within* {*0*..*1*})
using *vector-derivative-at-within-ivl*[*OF gamma2-vec-diffable*]
by *auto*
have ∀ *x*∈{*0*..*1*} − ((*φ* −‘ *?s′*) ∪ *?s*). *D x* ∗ (*vector-derivative γ2* (*at* (*φ x*) *within* {*0*..*1*}) · *b*) = (*vector-derivative* (*γ2* ∘ *φ*) (*at x within* {*0*..*1*}) · *b*)
proof
fix *x*::*real*
assume *ass*: *x* ∈ {*0*..*1*} −((*φ* −‘ *?s′*) ∪ *?s*)
then have *0*: *φ differentiable* (*at x*)
using *s* **by** (*auto simp add*: *C1-differentiable-on-def differentiable-def has-vector-derivative-def*)
obtain *D2* **where** (*φ has-vector-derivative D2*) (*at x*)
using *D-props ass* **by** *blast*
have *φ x* ∈ {*0*..*1*} − *?s′*
using *phi*(*5*) *ass* **by** (*metis Diff-Un Diff-iff Int-iff bij-betw-def image-eqI vimageI*)
then have *1*: *γ2 differentiable* (*at* (*φ x*))
using *gamma2-differentiable*
by *auto*
have *3*: *vector-derivative γ2* (*at* (*φ x*)) = *vector-derivative γ2* (*at* (*φ x*) *within* {*0*..*1*})
proof−
have ∗:*0*≤ *φ x* ∧ *φ x* ≤ *1* **using** *phi*(*5*) *ass*
using ‹*φ x* ∈ {*0*..*1*} − *s′* ∩ {*0*..*1*}› **by** *auto*
then have ∗∗:(*γ2 has-vector-derivative* (*vector-derivative γ2* (*at* (*φ x*)))) (*at* (*φ x*))
using *1 vector-derivative-works* **by** *auto*
show *?thesis*
using ∗ *vector-derivative-at-within-ivl*[*OF* ∗∗] **by** *auto*
qed
show *D x* ∗ (*vector-derivative γ2* (*at* (*φ x*) *within* {*0*..*1*}) · *b*) = *vector-derivative* (*γ2* ∘ *φ*) (*at x within* {*0*..*1*}) · *b*
using *vector-derivative-chain-at*[*OF 0 1*]
apply (*auto simp add*: *gamma2-vec-deriv-within*[*OF ass*, *symmetric*] *3*[*symmetric*])
using *D-props ass vector-derivative-at*
by *fastforce*
qed
then have *c*:⋀*x*. *x*∈({*0*..*1*} −((*φ* −‘ *?s′*) ∪ *?s*)) ⟹ *D x* ∗ (*F* (*γ2* (*φ x*)) · *b* ∗ (*vector-derivative γ2* (*at* (*φ x*) *within* {*0*..*1*}) · *b*)) =
F (*γ2* (*φ x*)) · *b* ∗ (*vector-derivative* (*γ2* ∘ *φ*) (*at x*

within {*0..1*}) · *b*)
  by *auto*
 then have *d*: *integral* ({*0..1*}) (λ*x*. *D x* ∗ (*F* (γ*2* (φ *x*)) · *b* ∗ (*vector-derivative* γ*2* (*at* (φ *x*) *within* {*0..1*}) · *b*))) =
 integral ({*0..1*}) (λ*x*. *F* (γ*2* (φ *x*)) · *b* ∗ (*vector-derivative* (γ*2* ∘ φ) (*at x within* {*0..1*}) · *b*))
 proof−
  have *negligible* ((φ −' *?s'*) ∪ *?s*) **using** *finite-neg-img s*(*1*) **by** *auto*
  then show *?thesis*
   using *c integral-spike* **by** (*metis*(*no-types,lifting*))
 qed
 have *phi-in-int*: (⋀*x*. *x* ∈ {*0..1*} ⟹ φ *x* ∈ {*0..1*}) **using** *phi*
  using *v* **by** *blast*
 then have *e*: ((λ*x*. *F* (γ*2* (φ *x*)) · *b* ∗ (*vector-derivative* (γ*2* ∘ φ) (*at x within* {*0..1*}) · *b*)) *has-integral*
 integral {φ *0*..φ *1*} (λ*x*. *F* (γ*2 x*) · *b* ∗ (*vector-derivative* γ*2* (*at x within* {*0..1*}) · *b*))){*0..1*}
 proof−
  have *negligible ?s* **using** *s-inter*(*1*) **by** *auto*
  have *0*: *negligible* ((φ −' *?s'*) ∪ *?s*) **using** *finite-neg-img s*(*1*) **by** *auto*
   have *c'*:∀ *x*∈ {*0..1*} − ((φ −' *?s'*) ∪ *?s*). *D x* ∗ (*F* (γ*2* (φ *x*)) · *b* ∗ (*vector-derivative* γ*2* (*at* (φ *x*) *within* {*0..1*}) · *b*)) =
 F (γ*2* (φ *x*)) · *b* ∗ (*vector-derivative* (γ*2* ∘ φ) (*at x within* {*0..1*}) · *b*)
   using *c* **by** *blast*
  have *has-integral-spike-eq'*: ⋀*s t f g y*. *negligible s* ⟹
 ∀ *x*∈*t* − *s*. *g x* = *f x* ⟹ (*f has-integral y*) *t* = (*g has-integral y*) *t*
   using *has-integral-spike-eq* **by** *metis*
  show *?thesis*
   using *a has-integral-spike-eq'*[*OF 0 c'*] **by** *blast*
 qed
 then have *f*: ((λ*x*. *F* (γ*1 x*) · *b* ∗ (*vector-derivative* γ*1* (*at x within* {*0..1*}) · *b*)) *has-integral*
 integral {φ *0*..φ *1*} (λ*x*. *F* (γ*2 x*) · *b* ∗ (*vector-derivative* γ*2* (*at x within* {*0..1*}) · *b*)))
 {*0..1*}
 proof−
  assume *ass*: ((λ*x*. *F* (γ*2* (φ *x*)) · *b* ∗ (*vector-derivative* (γ*2* ∘ φ) (*at x within* {*0..1*}) · *b*)) *has-integral*
 integral {φ *0*..φ *1*} (λ*x*. *F* (γ*2 x*) · *b* ∗ (*vector-derivative* γ*2* (*at x within* {*0..1*}) · *b*)))
 {*0..1*}
  have ∗:∀ *x*∈{*0..1*} − (((φ −' *?s'*) ∪ *?s*) ∪ {*0,1*}). (λ*x*. *F* (γ*2* (φ *x*)) · *b* ∗ (*vector-derivative* (γ*2* ∘ φ) (*at x within* {*0..1*}) · *b*)) *x* =
 (λ*x*. *F* (γ*1 x*) · *b* ∗ (*vector-derivative* γ*1* (*at x within* {*0..1*}) · *b*)) *x*
  proof−
   have ∀ *x*∈{*0*<..<*1*} − (φ −' *?s'* ∪ *?s*). (*vector-derivative* (γ*2* ∘ φ) (*at x*

within $\{0..1\}) \cdot b) = ($*vector-derivative* $(\gamma 1)$ $($*at x within* $\{0..1\}) \cdot b)$

proof

have *i*: *open* $(\{0<..<1\} - ((\varphi -` ?s') \cup ?s))$ **using** *open-diff s*(1) *open-greaterThanLessThan finite-neg-img*

by (*simp add*: *open-diff*)

have *ii*: $\forall x\in\{0<..<1::real\} - (\varphi -` ?s' \cup ?s).\ (\gamma 2 \circ \varphi)\ x = \gamma 1\ x$ **using** *phi*(1)

by *auto*

fix *x::real*

assume *ass*: $x \in \{0<..<1::real\} - ((\varphi -` ?s') \cup ?s)$

then have *iii*: $(\gamma 2 \circ \varphi$ *has-vector-derivative vector-derivative* $(\gamma 2 \circ \varphi)$ (*at x within* $\{0..1\}$)) (*at x*)

by (*metis* (*no-types*) *Diff-iff add.commute add-strict-mono ass atLeast-AtMost-iff gamma2-vec-deriv-within gamma2-vec-diffable greaterThanLessThan-iff less-irrefl not-le*)

then have *iv*:($\gamma 1$ *has-vector-derivative vector-derivative* $(\gamma 2 \circ \varphi)$ (*at x within* $\{0..1\}$)) (*at x*)

using *has-derivative-transform-within-open i ii ass*

apply(*auto simp add*: *has-vector-derivative-def*)

apply (*meson ass has-derivative-transform-within-open i ii*)

apply (*meson ass has-derivative-transform-within-open i ii*)

by (*meson ass has-derivative-transform-within-open i ii*)

have *v*: $0 \le x$ $x \le 1$ **using** *ass* **by** *auto*

have *0*: *vector-derivative* $\gamma 1$ (*at x within* $\{0..1\}$) = *vector-derivative* $(\gamma 2 \circ \varphi)$ (*at x within* $\{0..1\}$)

using *vector-derivative-at-within-ivl*[*OF iv v*(1) *v*(2) *zero-less-one*]

by *force*

have *1*: *vector-derivative* $(\gamma 2 \circ \varphi)$ (*at x within* $\{0..1\}$) = *vector-derivative* $(\gamma 2 \circ \varphi)$ (*at x within* $\{0..1\}$)

using *vector-derivative-at-within-ivl*[*OF iii v*(1) *v*(2) *zero-less-one*]

by *force*

then have *vector-derivative* $(\gamma 2 \circ \varphi)$ (*at x within* $\{0..1\}$) = *vector-derivative* $\gamma 1$ (*at x within* $\{0..1\}$)

using *0 1* **by** *auto*

then show *vector-derivative* $(\gamma 2 \circ \varphi)$ (*at x within* $\{0..1\}$) $\cdot\ b$ = *vector-derivative* $\gamma 1$ (*at x within* $\{0..1\}$) $\cdot\ b$ **by** *auto*

qed

then have *i*: $\forall x\in\{0..1\} - (((\varphi -` ?s') \cup ?s)\cup\{0,1\}).$ (*vector-derivative* $(\gamma 2 \circ \varphi)$ (*at x within* $\{0..1\}$) $\cdot\ b$) = (*vector-derivative* $(\gamma 1)$ (*at x within* $\{0..1\}$) $\cdot\ b$)

by *auto*

have *ii*: $\forall x\in\{0..1\} - (((\varphi -` ?s') \cup ?s)\cup\{0,1\}).\ F\ (\gamma 1\ x) \cdot b = F\ (\gamma 2\ (\varphi\ x)) \cdot b$

using *phi*(1)

by *auto*

show *?thesis*

using *i ii* **by** *metis*

qed

have ∗∗: *negligible* $((\varphi -` ?s') \cup ?s \cup \{0, 1\})$ **using** *s*(*1*) *finite-neg-img* **by** *auto*
have *has-integral-spike-eq'*: $\bigwedge s\ t\ g\ f\ y.$ *negligible s* $\Longrightarrow$
$\forall x \in t - s.\ g\ x = f\ x \Longrightarrow$ (*f has-integral y*) *t* = (*g has-integral y*) *t*
using *has-integral-spike-eq* **by** *metis*
show *?thesis*
using *has-integral-spike-eq'*[*OF* ∗∗ ∗] *ass*
by *blast*
qed
then show *line-integral-exists F* {*b*} *γ1*
using *phi* **by**(*auto simp add*: *line-integral-exists-def*)
have *integral* ({*0..1*}) (λ*x*. *F* (*γ2* (*φ x*)) · *b* ∗ (*vector-derivative* (*γ2* ∘ *φ*) (*at x within* {*0..1*}) · *b*)) =
integral ({*0..1*}) (λ*x*. *F* (*γ1 x*) · *b* ∗ (*vector-derivative γ1* (*at x within* {*0..1*}) · *b*))
using *integral-unique*[*OF e*] *integral-unique*[*OF f*]
by *metis*
moreover have *integral* ({*0..1*}) (λ*x*. *F* (*γ2* (*φ x*)) · *b* ∗ (*vector-derivative* (*γ2* ∘ *φ*) (*at x within* {*0..1*}) · *b*)) =
integral ({*0..1*}) (λ*x*. *F* (*γ2 x*) · *b* ∗ (*vector-derivative γ2* (*at x within* {*0..1*}) · *b*))
using *b d phi* **by** (*auto simp add*:)
ultimately show *line-integral F* {*b*} *γ1* = *line-integral F* {*b*} *γ2*
using *phi* **by**(*auto simp add*: *line-integral-def*)
qed

lemma *reparam-weak-eq-line-integrals*:
assumes *reparam-weak γ1 γ2*
γ2 C1-differentiable-on {*0..1*}
continuous-on (*path-image γ2*) (λ*x*. *F x* · *b*)
shows *line-integral F* {*b*} *γ1* = *line-integral F* {*b*} *γ2*
line-integral-exists F {*b*} *γ1*
proof−
obtain *φ* **where** *phi*: (∀ *x*∈{*0..1*}. *γ1 x* = (*γ2 o φ*) *x*) *φ piecewise-C1-differentiable-on* {*0..1*} *φ*(*0*) = *0* *φ*(*1*) = *1* *φ* ` {*0..1*} = {*0..1*}
using *assms*(*1*)
by (*auto simp add*: *reparam-weak-def*)
obtain *s* **where** *s*: *finite s φ C1-differentiable-on* {*0..1*} − *s*
using *phi*
by(*auto simp add*: *reparam-weak-def piecewise-C1-differentiable-on-def*)
have *cont-phi*: *continuous-on* {*0..1*} *φ*
using *phi*
by(*auto simp add*: *reparam-weak-def piecewise-C1-differentiable-on-imp-continuous-on*)
have *gamma2-differentiable*: (∀ *x* ∈ {*0* .. *1*}. *γ2 differentiable at x*)
using *assms*(*2*)
by (*auto simp add*: *valid-path-def C1-differentiable-on-eq*)
then have *gamma2-b-component-differentiable*: (∀ *x* ∈ {*0* .. *1*}. (λ*x*. (*γ2 x*) · *b*) *differentiable at x*)

by *auto*
then have $(\lambda x.\ (\gamma 2\ x) \cdot b)$ *differentiable-on* $\{0..1\}$
using *differentiable-at-withinI*
by (*auto simp add*: *differentiable-on-def*)
then have *gama2-cont-comp*: *continuous-on* $\{0..1\}$ $(\lambda x.\ (\gamma 2\ x) \cdot b)$
using *differentiable-imp-continuous-on*
by *auto*
have *gamma2-cont*:*continuous-on* $\{0..1\}$ $\gamma 2$
using *assms*(*2*) *C1-differentiable-imp-continuous-on*
by (*auto simp add*: *valid-path-def*)
have *iii*: *continuous-on* $\{0..1\}$ $(\lambda x.\ F\ (\gamma 2\ x) \cdot b * (vector\text{-}derivative\ \gamma 2\ (at\ x\ within\ \{0..1\}) \cdot b))$
proof−
have *0*: *continuous-on* $\{0..1\}$ $(\lambda x.\ F\ (\gamma 2\ x) \cdot b)$
using *assms*(*3*) *continuous-on-compose*[*OF gamma2-cont*]
by (*auto simp add*: *path-image-def*)
obtain *D* **where** *D*: $(\forall x \in \{0..1\}.\ (\gamma 2\ has\text{-}vector\text{-}derivative\ D\ x)\ (at\ x)) \wedge continuous\text{-}on\ \{0..1\}\ D$
using *assms*(*2*) **by** (*auto simp add*: *C1-differentiable-on-def*)
then have $*$:$\forall x \in \{0..1\}.\ vector\text{-}derivative\ \gamma 2\ (at\ x\ within \{0..1\}) = D\ x$
using *vector-derivative-at vector-derivative-at-within-ivl*
by *fastforce*
then have *continuous-on* $\{0..1\}$ $(\lambda x.\ vector\text{-}derivative\ \gamma 2\ (at\ x\ within \{0..1\}))$
using *continuous-on-eq D* **by** *force*
then have *1*: *continuous-on* $\{0..1\}$ $(\lambda x.\ (vector\text{-}derivative\ \gamma 2\ (at\ x\ within \{0..1\})) \cdot b)$
by (*auto intro*!: *continuous-intros*)
show *?thesis*
using *continuous-on-mult*[*OF 0 1*] **by** *auto*
qed
have *iv*: $\varphi(0) \leq \varphi(1)$
using *phi*(*3*) *phi*(*4*) **by** (*simp add*: *reparam-weak-def*)
have *v*: $\varphi`\{0..1\} \subseteq \{0..1\}$
using *phi*(*5*) **by** (*simp add*: *reparam-weak-def*)
obtain *D* **where** *D-props*: $(\forall x \in \{0..1\} - s.\ (\varphi\ has\text{-}vector\text{-}derivative\ D\ x)\ (at\ x))$
using *s*
by (*auto simp add*: *C1-differentiable-on-def*)
then have $(\bigwedge x.\ x \in (\{0..1\} - s) \Longrightarrow (\varphi\ has\text{-}vector\text{-}derivative\ D\ x)\ (at\ x\ within \{0..1\}))$
using *has-vector-derivative-at-within* **by** *blast*
then have *vi*: $(\bigwedge x.\ x \in (\{0..1\} - s) \Longrightarrow (\varphi\ has\text{-}real\text{-}derivative\ D\ x)\ (at\ x\ within \{0..1\}))$
using *has-real-derivative-iff-has-vector-derivative*
by *blast*
have *a*:$((\lambda x.\ D\ x * (F\ (\gamma 2\ (\varphi\ x)) \cdot b * (vector\text{-}derivative\ \gamma 2\ (at\ (\varphi\ x)\ within \{0..1\}) \cdot b)))$ *has-integral*
integral $\{\varphi\ 0..\varphi\ 1\}$ $(\lambda x.\ F\ (\gamma 2\ x) \cdot b * (vector\text{-}derivative\ \gamma 2\ (at\ x\ within\ \{0..1\}) \cdot b)))$
$\{0..1\}$

using *has-integral-substitution-strong*[*OF s*(*1*) *zero-le-one iv v iii cont-phi vi*]
by *simp*
then have *b*: *integral* {*0..1*} (λ*x*. *D x* $*$ (*F* (γ*2* (φ *x*)) $\cdot$ *b* $*$ (*vector-derivative* γ*2* (*at* (φ *x*) *within* {*0..1*}) $\cdot$ *b*))) =
integral {φ *0*..φ *1*} (λ*x*. *F* (γ*2 x*) $\cdot$ *b* $*$ (*vector-derivative* γ*2* (*at x within* {*0..1*}) $\cdot$ *b*))
by *auto*
have *gamma2-vec-diffable*: $\bigwedge$*x*::*real*. *x* $\in$ {*0..1*} $-s \Longrightarrow$ ((γ*2 o* φ) *has-vector-derivative vector-derivative* (γ*2* $\circ$ φ) (*at x*)) (*at x*)
proof−
fix *x*::*real*
assume *ass*: *x* $\in$ {*0..1*} $-s$
have *zer-le-x-le-1*:*0*$\leq$ *x* $\wedge$ *x* $\leq$ *1* **using** *ass* **by** *auto*
show ((γ*2 o* φ) *has-vector-derivative vector-derivative* (γ*2* $\circ$ φ) (*at x*)) (*at x*)
proof−
have $**$: γ*2 differentiable at* (φ *x*)
using *phi gamma2-differentiable*
by (*auto simp add*: *zer-le-x-le-1*)
have $***$: φ *differentiable at x*
using *s ass*
by (*auto simp add*: *C1-differentiable-on-eq*)
then show ((γ*2 o* φ) *has-vector-derivative vector-derivative* (γ*2* $\circ$ φ) (*at x*)) (*at x*)
using *differentiable-chain-at*[*OF* $***$ $**$]
by (*auto simp add*: *vector-derivative-works*)
qed
qed
then have *gamma2-vec-deriv-within*: $\bigwedge$*x*::*real*. *x* $\in$ {*0..1*} $-s \Longrightarrow$ *vector-derivative* (γ*2* $\circ$ φ) (*at x*) = *vector-derivative* (γ*2* $\circ$ φ) (*at x within* {*0..1*})
using *vector-derivative-at-within-ivl*[*OF gamma2-vec-diffable*]
by *auto*
have $\forall$ *x*$\in${*0..1*} − *s*. *D x* $*$ (*vector-derivative* γ*2* (*at* (φ *x*) *within* {*0..1*}) $\cdot$ *b*) = (*vector-derivative* (γ*2* $\circ$ φ) (*at x within* {*0..1*}) $\cdot$ *b*)
proof
fix *x*::*real*
assume *ass*: *x* $\in$ {*0..1*} $-s$
then have *0*: φ *differentiable* (*at x*)
using *s*
by (*auto simp add*: *C1-differentiable-on-def differentiable-def has-vector-derivative-def*)
obtain *D2* **where** (φ *has-vector-derivative D2*) (*at x*)
using *D-props ass*
by *blast*
have φ *x* $\in$ {*0..1*}
using *phi*(*5*) *ass*
by (*auto simp add*: *reparam-weak-def*)
then have *1*: γ*2 differentiable* (*at* (φ *x*))
using *gamma2-differentiable*
by *auto*
have *3*: *vector-derivative* γ*2* (*at* (φ *x*)) = *vector-derivative* γ*2* (*at* (φ *x*) *within*

{0..1})
proof−
have ∗:0≤ φ x ∧ φ x ≤ 1 **using** phi(5) ass **by** auto
then have ∗∗:(γ2 has-vector-derivative (vector-derivative γ2 (at (φ x)))) (at (φ x))
using 1 vector-derivative-works
by auto
show ?thesis
using ∗ vector-derivative-at-within-ivl[OF ∗∗]
by auto
qed
show D x ∗ (vector-derivative γ2 (at (φ x) within {0..1}) · b) = vector-derivative (γ2 ∘ φ) (at x within {0..1}) · b
using vector-derivative-chain-at[OF 0 1]
apply (auto simp add: gamma2-vec-deriv-within[OF ass, symmetric] 3[symmetric])
using D-props ass vector-derivative-at
by fastforce
qed
then have c:⋀x. x∈({0..1} −s) ⟹ D x ∗ (F (γ2 (φ x)) · b ∗ (vector-derivative γ2 (at (φ x) within {0..1}) · b)) =
F (γ2 (φ x)) · b ∗ (vector-derivative (γ2 ∘ φ) (at x within {0..1}) · b)
by auto
then have d: integral ({0..1}) (λx. D x ∗ (F (γ2 (φ x)) · b ∗ (vector-derivative γ2 (at (φ x) within {0..1}) · b))) =
integral ({0..1}) (λx. F (γ2 (φ x)) · b ∗ (vector-derivative (γ2 ∘ φ) (at x within {0..1}) · b))
proof−
have negligible s **using** s(1) **by** auto
then show ?thesis
using c integral-spike **by** (metis(no-types,lifting))
qed
have phi-in-int: (⋀x. x ∈ {0..1} ⟹ φ x ∈ {0..1}) **using** phi
by(auto simp add:)
then have e: ((λx. F (γ2 (φ x)) · b ∗ (vector-derivative (γ2 ∘ φ) (at x within {0..1}) · b)) has-integral
integral {φ 0..φ 1} (λx. F (γ2 x) · b ∗ (vector-derivative γ2 (at x within {0..1}) · b))){0..1}
proof−
have 0:negligible s **using** s(1) **by** auto
have c′:∀ x∈ {0..1} −s. D x ∗ (F (γ2 (φ x)) · b ∗ (vector-derivative γ2 (at (φ x) within {0..1}) · b)) =
F (γ2 (φ x)) · b ∗ (vector-derivative (γ2 ∘ φ) (at x within {0..1}) · b)
using c **by** auto
have has-integral-spike-eq′: ⋀s t f g y. negligible s ⟹
∀ x∈t − s. g x = f x ⟹ (f has-integral y) t = (g has-integral y) t
using has-integral-spike-eq **by** metis

show *?thesis*
using *a has-integral-spike-eq′[OF 0 c′]* **by** *blast*
qed
then have *f*: ((λ*x*. *F* (γ*1 x*) · *b* ∗ (*vector-derivative* γ*1* (*at x within* {*0..1*}) · *b*)) *has-integral*
integral {φ *0*..φ *1*} (λ*x*. *F* (γ*2 x*) · *b* ∗ (*vector-derivative* γ*2* (*at x within* {*0..1*}) · *b*)))
{*0..1*}
proof−
assume *ass*: ((λ*x*. *F* (γ*2* (φ *x*)) · *b* ∗ (*vector-derivative* (γ*2* ∘ φ) (*at x within* {*0..1*}) · *b*)) *has-integral*
integral {φ *0*..φ *1*} (λ*x*. *F* (γ*2 x*) · *b* ∗ (*vector-derivative* γ*2* (*at x within* {*0..1*}) · *b*)))
{*0..1*}
have ∗:∀ *x*∈{*0..1*} − (*s*∪{*0*,*1*}). (λ*x*. *F* (γ*2* (φ *x*)) · *b* ∗ (*vector-derivative* (γ*2* ∘ φ) (*at x within* {*0..1*}) · *b*)) *x* =
(λ*x*. *F* (γ*1 x*) · *b* ∗ (*vector-derivative* γ*1* (*at x within* {*0..1*}) · *b*)) *x*
proof−
have ∀ *x*∈{*0*<..<*1*} − *s*. (*vector-derivative* (γ*2* ∘ φ) (*at x within* {*0..1*}) · *b*) = (*vector-derivative* (γ*1*) (*at x within* {*0..1*}) · *b*)
proof
have *i*: *open* ({*0*<..<*1*} − *s*) **using** *open-diff s open-greaterThanLessThan* **by** *blast*
have *ii*: ∀ *x*∈{*0*<..<*1*::*real*} − *s*. (γ*2* ∘ φ) *x* = γ*1 x* **using** *phi*(*1*)
by *auto*
fix *x*::*real*
assume *ass*: *x* ∈ {*0*<..<*1*::*real*} − *s*
then have *iii*: (γ*2* ∘ φ *has-vector-derivative vector-derivative* (γ*2* ∘ φ) (*at x within* {*0..1*})) (*at x*)
using *has-vector-derivative-at-within gamma2-vec-deriv-within gamma2-vec-diffable*
by *auto*

then have *iv*:(γ*1 has-vector-derivative vector-derivative* (γ*2* ∘ φ) (*at x within* {*0..1*})) (*at x*)
using *has-derivative-transform-within-open i ii ass*
apply(*auto simp add*: *has-vector-derivative-def*)
by *force*
have *v*: *0* ≤ *x x* ≤ *1* **using** *ass* **by** *auto*
have *0*: *vector-derivative* γ*1* (*at x within* {*0..1*}) = *vector-derivative* (γ*2* ∘ φ) (*at x within* {*0..1*})
using *vector-derivative-at-within-ivl*[*OF iv v*(*1*) *v*(*2*) *zero-less-one*]
by *force*
have *1*: *vector-derivative* (γ*2* ∘ φ) (*at x within* {*0..1*}) = *vector-derivative* (γ*2* ∘ φ) (*at x within* {*0..1*})
using *vector-derivative-at-within-ivl*[*OF iii v*(*1*) *v*(*2*) *zero-less-one*]
by *force*
then have *vector-derivative* (γ*2* ∘ φ) (*at x within* {*0..1*}) = *vector-derivative* γ*1* (*at x within* {*0..1*})

```
        using 0 1 by auto
        then show vector-derivative (γ2 ∘ φ) (at x within {0..1}) · b = vector-derivative γ1 (at x within {0..1}) · b by auto
      qed
      then have i: ∀x∈{0..1} − (s∪{0,1}). (vector-derivative (γ2 ∘ φ) (at x within {0..1}) · b) = (vector-derivative (γ1) (at x within {0..1}) · b)
        by auto
      have ii: ∀x∈{0..1} − (s∪{0,1}). F (γ1 x) · b = F (γ2 (φ x)) · b
        using phi(1) by auto
      show ?thesis
        using i ii by auto
    qed
    have **: negligible (s∪{0,1}) using s(1) by auto
    have has-integral-spike-eq′: ⋀s t g f y. negligible s ⟹
                                  ∀x∈t − s. g x = f x ⟹ (f has-integral y) t = (g has-integral y) t
      using has-integral-spike-eq by metis
    show ?thesis
      using has-integral-spike-eq′[OF ** *] ass
      by blast
  qed
  then show line-integral-exists F {b} γ1
    using phi by(auto simp add: line-integral-exists-def)
  have integral ({0..1}) (λx. F (γ2 (φ x)) · b * (vector-derivative (γ2 ∘ φ) (at x within {0..1}) · b)) =
                integral ({0..1}) (λx. F (γ1 x) · b * (vector-derivative γ1 (at x within {0..1}) · b))
    using integral-unique[OF e] integral-unique[OF f]
    by metis
  moreover have integral ({0..1}) (λx. F (γ2 (φ x)) · b * (vector-derivative (γ2 ∘ φ) (at x within {0..1}) · b)) =
                integral ({0..1}) (λx. F (γ2 x) · b * (vector-derivative γ2 (at x within {0..1}) · b))
    using b d phi by (auto simp add:)
  ultimately show line-integral F {b} γ1 = line-integral F {b} γ2
    using phi by(auto simp add: line-integral-def)
qed

lemma line-integral-sum-basis:
  assumes finite (basis::('a::euclidean-space) set) ∀b∈basis. line-integral-exists F {b} γ
  shows line-integral F basis γ = (∑b∈basis. line-integral F {b} γ)
        line-integral-exists F basis γ
  using assms
proof(induction basis)
  show line-integral F {} γ = (∑b∈{}. line-integral F {b} γ)
    by(auto simp add: line-integral-def)
  show ∀b∈{}. line-integral-exists F {b} γ ⟹ line-integral-exists F {} γ
    by(simp add: line-integral-exists-def integrable-0)
```

next
 fix *basis*::(*'a*::*euclidean-space*) *set*
 fix *x*::*'a*::*euclidean-space*
 fix γ
 assume *ind-hyp*: (∀ *b*∈*basis*. *line-integral-exists F* {*b*} γ ⟹ *line-integral-exists F basis* γ)
 (∀ *b*∈*basis*. *line-integral-exists F* {*b*} γ ⟹ *line-integral F basis* γ = (∑ *b*∈*basis*. *line-integral F* {*b*} γ))
 assume *step*: *finite basis*
 x ∉ *basis*
 ∀ *b*∈*insert x basis*. *line-integral-exists F* {*b*} γ
 then have *0*: *line-integral-exists F* {*x*} γ **by** *auto*
 have *1*:*line-integral-exists F basis* γ
 using *ind-hyp step* **by** *auto*
 then show *line-integral-exists F* (*insert x basis*) γ
 using *step*(*1*) *step*(*2*) *line-integral-sum-gen*(*2*)[*OF - 0 1*] **by** *simp*
 have *3*: *finite* (*insert x basis*) **using** *step*(*1*) **by** *auto*
 have *line-integral F basis* γ = (∑ *b*∈*basis*. *line-integral F* {*b*} γ)
 using *ind-hyp step* **by** *auto*
 then show *line-integral F* (*insert x basis*) γ = (∑ *b*∈*insert x basis*. *line-integral F* {*b*} γ)
 using *step*(*1*) *step*(*2*) *line-integral-sum-gen*(*1*)[*OF 3 0 1*]
 by *force*
qed

lemma *reparam-weak-eq-line-integrals-basis*:
 assumes *reparam-weak* $\gamma 1$ $\gamma 2$
 $\gamma 2$ *C1-differentiable-on* {*0..1*}
 ∀ *b*∈*basis*. *continuous-on* (*path-image* $\gamma 2$) (λx. *F x* · *b*)
 finite basis
 shows *line-integral F basis* $\gamma 1$ = *line-integral F basis* $\gamma 2$
 line-integral-exists F basis $\gamma 1$
proof−
 show *line-integral-exists F basis* $\gamma 1$
 using *reparam-weak-eq-line-integrals*(*2*)[*OF assms*(*1*) *assms*(*2*)] *assms*(*3*−*4*) *line-integral-sum-basis*(*2*)[*OF assms*(*4*)]
 by(*simp add*: *subset-iff*)
 show *line-integral F basis* $\gamma 1$ = *line-integral F basis* $\gamma 2$
 using *reparam-weak-eq-line-integrals*[*OF assms*(*1*) *assms*(*2*)] *assms*(*3*−*4*) *line-integral-sum-basis*(*1*)[*OF assms*(*4*)]
 line-integral-exists-smooth-one-base[*OF assms*(*2*)]
 by(*simp add*: *subset-iff*)
qed

lemma *reparam-eq-line-integrals-basis*:
 assumes *reparam* $\gamma 1$ $\gamma 2$
 $\gamma 2$ *piecewise-C1-differentiable-on* {*0..1*}
 ∀ *b*∈*basis*. *continuous-on* (*path-image* $\gamma 2$) (λx. *F x* · *b*)
 finite basis

```
    ∀ b∈basis. line-integral-exists F {b} γ2
  shows line-integral F basis γ1 = line-integral F basis γ2
    line-integral-exists F basis γ1
proof−
  show line-integral-exists F basis γ1
   using reparam-eq-line-integrals(2)[OF assms(1) assms(2)] assms(3−5) line-integral-sum-basis(2)[OF
assms(4)]
    by(simp add: subset-iff)
  show line-integral F basis γ1 = line-integral F basis γ2
   using reparam-eq-line-integrals[OF assms(1) assms(2)] assms(3−5) line-integral-sum-basis(1)[OF
assms(4)]
    by(simp add: subset-iff)
qed

lemma line-integral-exists-smooth:
  assumes γ C1-differentiable-on {0..1}
    ∀ (b::'a::euclidean-space) ∈basis. continuous-on (path-image γ) (λx. F x · b)
    finite basis
  shows line-integral-exists F basis γ
 using reparam-weak-eq-line-integrals-basis(2)[OF reparam-weak-eq-refl[where ?γ1.0
= γ]] assms
  by fastforce

lemma smooth-path-imp-reverse:
  assumes g C1-differentiable-on {0..1}
  shows (reversepath g) C1-differentiable-on {0..1}
  using assms continuous-on-const
  apply (auto simp: reversepath-def)
  apply (rule C1-differentiable-compose [of λx::real. 1−x - g, unfolded o-def])
    apply (auto simp: C1-differentiable-on-eq)
  apply (simp add: finite-vimageI inj-on-def)
  done

lemma piecewise-smooth-path-imp-reverse:
  assumes g piecewise-C1-differentiable-on {0..1}
  shows (reversepath g) piecewise-C1-differentiable-on {0..1}
  using assms valid-path-reversepath
  using valid-path-def by blast

definition chain-reparam-weak-chain where
  chain-reparam-weak-chain one-chain1 one-chain2 ≡
       ∃f. bij f ∧ f ‘ one-chain1 = one-chain2 ∧ (∀ (k,γ)∈one-chain1. if k = fst
(f(k,γ)) then reparam-weak γ (snd (f(k,γ))) else reparam-weak γ (reversepath (snd
(f(k,γ)))))

lemma chain-reparam-weak-chain-line-integral:
  assumes chain-reparam-weak-chain one-chain1 one-chain2
    ∀ (k2,γ2)∈one-chain2. γ2 C1-differentiable-on {0..1}
    ∀ (k2,γ2)∈one-chain2.∀ b∈basis. continuous-on (path-image γ2) (λx. F x · b)
```

finite basis
and *bound1*: *boundary-chain one-chain1*
and *bound2*: *boundary-chain one-chain2*
shows *one-chain-line-integral F basis one-chain1* = *one-chain-line-integral F basis one-chain2*
∀ (*k*, γ)∈*one-chain1*. *line-integral-exists F basis* γ
proof−
obtain *f* **where** *f*: *bij f*
(∀ (*k*,γ)∈*one-chain1*. *if k* = *fst* (*f*(*k*,γ)) *then reparam-weak* γ (*snd* (*f*(*k*,γ))) *else reparam-weak* γ (*reversepath* (*snd* (*f*(*k*,γ)))))
f ' *one-chain1* = *one-chain2*
using *assms*(*1*)
by (*auto simp add*: *chain-reparam-weak-chain-def*)
have *0*: ∀ *x*∈*one-chain1*. (*case x of* (*k*, γ) ⇒ (*real-of-int k* ∗ *line-integral F basis* γ) = (*case f x of* (*k*, γ) ⇒ *real-of-int k* ∗ *line-integral F basis* γ) ∧
line-integral-exists F basis γ)
proof
{**fix** *k1* γ*1*
assume *ass1*: (*k1*,γ*1*) ∈*one-chain1*
have *real-of-int k1* ∗ *line-integral F basis* γ*1* = (*case* (*f* (*k1*,γ*1*)) *of* (*k2*, γ*2*) ⇒ *real-of-int k2* ∗ *line-integral F basis* γ*2*) ∧
line-integral-exists F basis γ*1*
proof(*cases*)
assume *ass2*: *k1* = *1*
let *?k2* = *fst* (*f* (*k1*, γ*1*))
let *?*γ*2* = *snd* (*f* (*k1*, γ*1*))
have *real-of-int k1* ∗ *line-integral F basis* γ*1* = *real-of-int ?k2* ∗ *line-integral F basis ?*γ*2*∧
line-integral-exists F basis γ*1*
proof(*cases*)
assume *ass3*: *?k2* = *1*
then have *0*: *reparam-weak* γ*1 ?*γ*2*
using *ass1 ass2 f*(*2*)
by *auto*
have *1*: *?*γ*2 C1-differentiable-on* {*0*..*1*}
using *f*(*3*) *assms*(*2*) *ass1*
by *force*
have *2*: ∀ *b*∈*basis*. *continuous-on* (*path-image ?*γ*2*) (λ*x*. *F x* · *b*)
using *f*(*3*) *assms*(*3*) *ass1*
by *force*
show *real-of-int k1* ∗ *line-integral F basis* γ*1* = *real-of-int ?k2* ∗ *line-integral F basis ?*γ*2* ∧
line-integral-exists F basis γ*1*
using *assms reparam-weak-eq-line-integrals-basis*[*OF 0 1 2 assms*(*4*)]
ass2 ass3
by *auto*
next
assume *?k2* ≠ *1*
then have *ass3*: *?k2* = −*1*

using *bound2 ass1 f*(*3*) **unfolding** *boundary-chain-def* **by** *force*
then have *0*: *reparam-weak γ1* (*reversepath ?γ2*)
using *ass1 ass2 f*(*2*)
by *auto*
have *1*: (*reversepath ?γ2*) *C1-differentiable-on* {*0..1*}
using *f*(*3*) *assms*(*2*) *ass1 smooth-path-imp-reverse*
by *force*
have *2*: ∀ *b*∈*basis. continuous-on* (*path-image* (*reversepath ?γ2*)) (*λx. F x* · *b*)
using *f*(*3*) *assms*(*3*) *ass1 path-image-reversepath*
by *force*
have *3*: *line-integral F basis ?γ2* = − *line-integral F basis* (*reversepath ?γ2*)
proof−
have *i*:*valid-path* (*reversepath ?γ2*)
using *1 C1-differentiable-imp-piecewise*
by (*auto simp add*: *valid-path-def*)
show *?thesis*
using *line-integral-on-reverse-path*(*1*)[*OF i line-integral-exists-smooth*[*OF 1 2*]] *assms*
by *auto*
qed
show *real-of-int k1* ∗ *line-integral F basis γ1* = *real-of-int ?k2* ∗ *line-integral F basis ?γ2* ∧
line-integral-exists F basis γ1
using *assms reparam-weak-eq-line-integrals-basis*[*OF 0 1 2 assms*(*4*)]
ass2 ass3 3
by *auto*
qed
then show *real-of-int k1* ∗ *line-integral F basis γ1* = (*case f* (*k1*, *γ1*) *of* (*k2*, *γ2*) ⇒ *real-of-int k2* ∗ *line-integral F basis γ2*) ∧
line-integral-exists F basis γ1
by (*simp add*: *case-prod-beta′*)
next
assume *k1* ≠ *1*
then have *ass2*: *k1* = −*1*
using *bound1 ass1 f*(*3*) **unfolding** *boundary-chain-def* **by** *force*
let *?k2* = *fst* (*f* (*k1*, *γ1*))
let *?γ2* = *snd* (*f* (*k1*, *γ1*))
have *real-of-int k1* ∗ *line-integral F basis γ1* = *real-of-int ?k2* ∗ *line-integral F basis ?γ2* ∧
line-integral-exists F basis γ1
proof(*cases*)
assume *ass3*: *?k2* = *1*
then have *0*: *reparam-weak γ1* (*reversepath ?γ2*)
using *ass1 ass2 f*(*2*)
by *auto*
have *1*: (*reversepath ?γ2*) *C1-differentiable-on* {*0..1*}
using *f*(*3*) *assms*(*2*) *ass1 smooth-path-imp-reverse*

by *force*
have *2*: ∀ *b*∈*basis. continuous-on* (*path-image* (*reversepath ?γ2*)) (λ*x. F x* · *b*)
using *f*(*3*) *assms*(*3*) *ass1 path-image-reversepath*
by *force*
have *3*: *line-integral F basis ?γ2* = − *line-integral F basis* (*reversepath ?γ2*)
proof−
have *i*:*valid-path* (*reversepath ?γ2*)
using *1 C1-differentiable-imp-piecewise*
by (*auto simp add*: *valid-path-def*)
show *?thesis*
using *line-integral-on-reverse-path*(*1*)[*OF i line-integral-exists-smooth*[*OF 1 2 assms*(*4*)]]
by *auto*
qed
show *real-of-int k1* ∗ *line-integral F basis γ1* = *real-of-int ?k2* ∗ *line-integral F basis ?γ2* ∧
line-integral-exists F basis γ1
using *assms reparam-weak-eq-line-integrals-basis*[*OF 0 1 2 assms*(*4*)]
ass2 ass3 3
by *auto*
next
assume *?k2* ≠ *1*
then have *ass3*: *?k2* = −*1*
using *bound2 ass1 f*(*3*) **unfolding** *boundary-chain-def* **by** *force*
then have *0*: *reparam-weak γ1 ?γ2*
using *ass1 ass2 f*(*2*) **by** *auto*
have *1*: *?γ2 C1-differentiable-on* {*0*..*1*}
using *f*(*3*) *assms*(*2*) *ass1* **by** *force*
have *2*: ∀ *b*∈*basis. continuous-on* (*path-image ?γ2*) (λ*x. F x* · *b*)
using *f*(*3*) *assms*(*3*) *ass1* **by** *force*
show *real-of-int k1* ∗ *line-integral F basis γ1* = *real-of-int ?k2* ∗ *line-integral F basis ?γ2* ∧
line-integral-exists F basis γ1
using *assms reparam-weak-eq-line-integrals-basis*[*OF 0 1 2 assms*(*4*)] *ass2 ass3*
by *auto*
qed
then show *real-of-int k1* ∗ *line-integral F basis γ1* = (*case f* (*k1*, *γ1*) *of* (*k2*, *γ2*) ⇒ *real-of-int k2* ∗ *line-integral F basis γ2*) ∧
line-integral-exists F basis γ1
by (*simp add*: *case-prod-beta′*)
qed
}
then show ⋀*x. x* ∈ *one-chain1* ⟹
(*case x of* (*k*, *γ*) ⇒ (*real-of-int k* ∗ *line-integral F basis γ*) = (*case f x of* (*k*, *γ*) ⇒ *real-of-int k* ∗ *line-integral F basis γ*) ∧
line-integral-exists F basis γ)

by (*auto simp add*: *case-prod-beta′*)
qed
show *one-chain-line-integral F basis one-chain1* = *one-chain-line-integral F basis one-chain2*
using *0* **by**(*simp add*: *one-chain-line-integral-def sum-bij*[*OF f*(*1*) - *f*(*3*)] *split-beta*)
show ∀(*k*, γ)∈*one-chain1*. *line-integral-exists F basis* γ
using *0* **by** *blast*
qed

definition *chain-reparam-chain* **where**
chain-reparam-chain one-chain1 one-chain2 ≡
∃*f*. *bij f* ∧ *f* ‘ *one-chain1* = *one-chain2* ∧ (∀(*k*,γ)∈*one-chain1*. *if k* = *fst* (*f*(*k*,γ)) *then reparam* γ (*snd* (*f*(*k*,γ))) *else reparam* γ (*reversepath* (*snd* (*f*(*k*,γ)))))

definition *chain-reparam-weak-path*::((*real*) ⇒ (*real* ∗ *real*)) ⇒ ((*int* ∗ ((*real*) ⇒ (*real* ∗ *real*))) *set*) ⇒ *bool* **where**
chain-reparam-weak-path γ *one-chain*
≡ ∃*l*. *set l* = *one-chain* ∧ *distinct l* ∧ *reparam* γ (*rec-join l*) ∧ *valid-chain-list l* ∧ *l* ≠ []

lemma *chain-reparam-chain-line-integral*:
assumes *chain-reparam-chain one-chain1 one-chain2*
∀(*k2*,γ*2*)∈*one-chain2*. γ*2 piecewise-C1-differentiable-on* {*0*..*1*}
∀(*k2*,γ*2*)∈*one-chain2*.∀*b*∈*basis*. *continuous-on* (*path-image* γ*2*) (λ*x*. *F x* · *b*)
finite basis
and *bound1*: *boundary-chain one-chain1*
and *bound2*: *boundary-chain one-chain2*
and *line*: ∀(*k2*,γ*2*)∈*one-chain2*. (∀*b*∈*basis*. *line-integral-exists F* {*b*} γ*2*)
shows *one-chain-line-integral F basis one-chain1* = *one-chain-line-integral F basis one-chain2*
∀(*k*, γ)∈*one-chain1*. *line-integral-exists F basis* γ
proof−
obtain *f* **where** *f*: *bij f*
(∀(*k*,γ)∈*one-chain1*. *if k* = *fst* (*f*(*k*,γ)) *then reparam* γ (*snd* (*f*(*k*,γ))) *else reparam* γ (*reversepath* (*snd* (*f*(*k*,γ)))))
f ‘ *one-chain1* = *one-chain2*
using *assms*(*1*)
by (*auto simp add*: *chain-reparam-chain-def*)
have *integ-exist-b*: ∀(*k1*,γ*1*)∈*one-chain1*. ∀*b*∈*basis*. *line-integral-exists F* {*b*} (*snd* (*f* (*k1*, γ*1*)))
using *line f* **by** *fastforce*
have *valid-cubes*: ∀(*k1*,γ*1*)∈*one-chain1*. *valid-path* (*snd* (*f* (*k1*, γ*1*)))
using *assms*(*2*) *f*(*3*) *valid-path-def* **by** *fastforce*
have *integ-rev-exist-b*: ∀(*k1*,γ*1*)∈*one-chain1*. ∀*b*∈*basis*. *line-integral-exists F* {*b*} (*reversepath* (*snd* (*f* (*k1*, γ*1*))))
using *line-integral-on-reverse-path*(*2*) *integ-exist-b valid-cubes*
by *blast*
have *0*: ∀*x*∈*one-chain1*. (*case x of* (*k*, γ) ⇒ (*real-of-int k* ∗ *line-integral F basis*

γ) = (*case f x of* (k, γ) ⇒ *real-of-int k* ∗ *line-integral F basis* γ) ∧
line-integral-exists F basis γ)
proof
{**fix** *k1 γ1*
assume *ass1*: (*k1*,*γ1*) ∈*one-chain1*
have *real-of-int k1* ∗ *line-integral F basis γ1* = (*case* (*f* (*k1*,*γ1*)) *of* (*k2*, *γ2*) ⇒ *real-of-int k2* ∗ *line-integral F basis γ2*) ∧
line-integral-exists F basis γ1
proof(*cases*)
assume *ass2*: *k1* = *1*
let *?k2* = *fst* (*f* (*k1*, *γ1*))
let *?γ2* = *snd* (*f* (*k1*, *γ1*))
have *real-of-int k1* ∗ *line-integral F basis γ1* = *real-of-int ?k2* ∗ *line-integral F basis ?γ2*∧
line-integral-exists F basis γ1
proof(*cases*)
assume *ass3*: *?k2* = *1*
then have *0*: *reparam γ1 ?γ2*
using *ass1 ass2 f*(*2*)
by *auto*
have *1*: *?γ2 piecewise-C1-differentiable-on* {*0*..*1*}
using *f*(*3*) *assms*(*2*) *ass1*
by *force*
have *2*: ∀ *b*∈*basis*. *continuous-on* (*path-image ?γ2*) (λ*x*. *F x* · *b*)
using *f*(*3*) *assms*(*3*) *ass1*
by *force*
show *real-of-int k1* ∗ *line-integral F basis γ1* = *real-of-int ?k2* ∗ *line-integral F basis ?γ2* ∧
line-integral-exists F basis γ1
using *assms reparam-eq-line-integrals-basis*[*OF 0 1 2 assms*(*4*)] *integ-exist-b ass1 ass2 ass3*
by *auto*
next
assume *?k2* ≠ *1*
then have *ass3*: *?k2* = −*1*
using *bound2 ass1 f*(*3*) **unfolding** *boundary-chain-def* **by** *force*
then have *0*: *reparam γ1* (*reversepath ?γ2*)
using *ass1 ass2 f*(*2*)
by *auto*
have *1*: (*reversepath ?γ2*) *piecewise-C1-differentiable-on* {*0*..*1*}
using *f*(*3*) *assms*(*2*) *ass1 piecewise-smooth-path-imp-reverse*
by *force*
have *2*: ∀ *b*∈*basis*. *continuous-on* (*path-image* (*reversepath ?γ2*)) (λ*x*. *F x* · *b*)
using *f*(*3*) *assms*(*3*) *ass1 path-image-reversepath*
by *force*
have *3*: *line-integral F basis ?γ2* = − *line-integral F basis* (*reversepath ?γ2*)
proof−

have *i*:*valid-path* (*reversepath ?γ2*)
using *1 C1-differentiable-imp-piecewise*
by (*auto simp add*: *valid-path-def*)
have *ii*: *line-integral-exists F basis* (*snd* (*f* (*k1*, *γ1*)))
using *assms*(*4*) *line-integral-sum-basis*(*2*) *integ-exist-b ass1*
by *fastforce*
show *?thesis*
using *i ii line-integral-on-reverse-path*(*1*) *valid-path-reversepath* **by** *blast*
qed
show *real-of-int k1* ∗ *line-integral F basis γ1* = *real-of-int ?k2* ∗ *line-integral F basis ?γ2* ∧
line-integral-exists F basis γ1
using *assms reparam-eq-line-integrals-basis*[*OF 0 1 2 assms*(*4*)] *integ-rev-exist-b*
ass1 ass2 ass3 3
by *auto*
qed
then show *real-of-int k1* ∗ *line-integral F basis γ1* = (*case f* (*k1*, *γ1*) *of* (*k2*, *γ2*) ⇒ *real-of-int k2* ∗ *line-integral F basis γ2*) ∧
line-integral-exists F basis γ1
by (*simp add*: *case-prod-beta′*)
next
assume *k1* ≠ *1*
then have *ass2*: *k1* = −*1*
using *bound1 ass1 f*(*3*) **unfolding** *boundary-chain-def* **by** *force*
let *?k2* = *fst* (*f* (*k1*, *γ1*))
let *?γ2* = *snd* (*f* (*k1*, *γ1*))
have *real-of-int k1* ∗ *line-integral F basis γ1* = *real-of-int ?k2* ∗ *line-integral F basis ?γ2* ∧
line-integral-exists F basis γ1
proof(*cases*)
assume *ass3*: *?k2* = *1*
then have *0*: *reparam γ1* (*reversepath ?γ2*)
using *ass1 ass2 f*(*2*)
by *auto*
have *1*: (*reversepath ?γ2*) *piecewise-C1-differentiable-on* {*0..1*}
using *f*(*3*) *assms*(*2*) *ass1 piecewise-smooth-path-imp-reverse*
by *force*
have *2*: ∀ *b*∈*basis*. *continuous-on* (*path-image* (*reversepath ?γ2*)) (λ*x*. *F x* · *b*)
using *f*(*3*) *assms*(*3*) *ass1 path-image-reversepath*
by *force*
have *3*: *line-integral F basis ?γ2* = − *line-integral F basis* (*reversepath ?γ2*)
proof−
have *i*:*valid-path* (*reversepath ?γ2*)
using *1 C1-differentiable-imp-piecewise*
by (*auto simp add*: *valid-path-def*)
show *?thesis*

using *line-integral-on-reverse-path*(*1*)[*OF i*] *integ-rev-exist-b*
using *ass1 assms*(*4*) *line-integral-sum-basis*(*2*) **by** *fastforce*
qed
show *real-of-int k1* ∗ *line-integral F basis* γ*1* = *real-of-int ?k2* ∗ *line-integral F basis ?*γ*2* ∧
line-integral-exists F basis γ*1*
using *assms reparam-eq-line-integrals-basis*[*OF 0 1 2 assms*(*4*)]
ass2 ass3 3
using *ass1 integ-rev-exist-b* **by** *auto*
next
assume *?k2* ≠ *1*
then have *ass3*: *?k2* = −*1*
using *bound2 ass1 f*(*3*) **unfolding** *boundary-chain-def* **by** *force*
then have *0*: *reparam* γ*1 ?*γ*2*
using *ass1 ass2 f*(*2*) **by** *auto*
have *1*: *?*γ*2 piecewise-C1-differentiable-on* {*0..1*}
using *f*(*3*) *assms*(*2*) *ass1*
by *force*
have *2*: ∀ *b*∈*basis. continuous-on* (*path-image ?*γ*2*) (λ*x. F x* · *b*)
using *f*(*3*) *assms*(*3*) *ass1*
by *force*
show *real-of-int k1* ∗ *line-integral F basis* γ*1* = *real-of-int ?k2* ∗ *line-integral F basis ?*γ*2* ∧
line-integral-exists F basis γ*1*
using *assms reparam-eq-line-integrals-basis*[*OF 0 1 2 assms*(*4*)]
ass2 ass3
using *ass1 integ-exist-b* **by** *auto*
qed
then show *real-of-int k1* ∗ *line-integral F basis* γ*1* = (*case f* (*k1*, γ*1*) *of* (*k2*, γ*2*) ⇒ *real-of-int k2* ∗ *line-integral F basis* γ*2*) ∧
line-integral-exists F basis γ*1*
by (*simp add*: *case-prod-beta′*)
qed
}
then show ⋀*x. x* ∈ *one-chain1* ⟹
(*case x of* (*k*, γ) ⇒ (*real-of-int k* ∗ *line-integral F basis* γ) = (*case f x of* (*k*, γ) ⇒ *real-of-int k* ∗ *line-integral F basis* γ) ∧
line-integral-exists F basis γ)
by (*auto simp add*: *case-prod-beta′*)
qed
show *one-chain-line-integral F basis one-chain1* = *one-chain-line-integral F basis one-chain2*
using *0* **by** (*simp add*: *one-chain-line-integral-def sum-bij*[*OF f*(*1*) - *f*(*3*)] *prod.case-eq-if*)
show ∀ (*k*, γ)∈*one-chain1*. *line-integral-exists F basis* γ
using *0* **by** *blast*
qed

lemma *path-image-rec-join*:

fixes γ::*real* $\Rightarrow$ (*real* $\times$ *real*)
fixes *k*::*int*
fixes *l*
shows $\bigwedge k\ \gamma.\ (k,\ \gamma) \in set\ l \Longrightarrow valid\text{-}chain\text{-}list\ l \Longrightarrow path\text{-}image\ \gamma \subseteq path\text{-}image$ $(rec\text{-}join\ l)$
proof(*induction l*)
case *Nil*
then show *?case* **by** *auto*
next
case *ass*: (*Cons a l*)
obtain $k'\ \gamma'$ **where** *a*: $a = (k',\gamma')$ **by** *force*
have $path\text{-}image\ \gamma \subseteq path\text{-}image\ (rec\text{-}join\ ((k',\gamma')\ \#\ l))$
proof(*cases*)
assume $l=[]$
then show $path\text{-}image\ \gamma \subseteq path\text{-}image\ (rec\text{-}join\ ((k',\gamma')\ \#\ l))$
using *ass*(*2*−*3*) *a*
by(*auto simp add*:)
next
assume $l\neq[]$
then obtain $b\ l'$ **where** *b-l*: $l = b\ \#\ l'$
by (*meson rec-join.elims*)
obtain $k''\ \gamma''$ **where** *b*: $b = (k'',\gamma'')$ **by** *force*
show *?thesis*
using *ass path-image-reversepath b-l path-image-join*
by(*fastforce simp add*: *a*)
qed
then show *?case*
using *a* **by** *auto*
qed

lemma *path-image-rec-join-2*:
fixes *l*
shows $l \neq [] \Longrightarrow valid\text{-}chain\text{-}list\ l \Longrightarrow path\text{-}image\ (rec\text{-}join\ l) \subseteq (\bigcup (k,\ \gamma) \in set$ $l.\ path\text{-}image\ \gamma)$
proof(*induction l*)
case *Nil*
then show *?case* **by** *auto*
next
case *ass*: (*Cons a l*)
obtain $k'\ \gamma'$ **where** *a*: $a = (k',\gamma')$ **by** *force*
have $path\text{-}image\ (rec\text{-}join\ (a\ \#\ l)) \subseteq (\bigcup (k,\ y){\in}set\ (a\ \#\ l).\ path\text{-}image\ y)$
proof(*cases*)
assume $l=[]$
then show $path\text{-}image\ (rec\text{-}join\ (a\ \#\ l)) \subseteq (\bigcup (k,\ y){\in}set\ (a\ \#\ l).\ path\text{-}image$ $y)$
using *step a* **by**(*auto simp add*:)
next
assume $l\neq[]$
then obtain $b\ l'$ **where** *b-l*: $l = b\ \#\ l'$

by (*meson rec-join.elims*)
obtain k'' γ'' **where** *b*: $b = (k'',\gamma'')$ **by** *force*
show *?thesis*
using *ass*
path-image-reversepath b-l path-image-join
apply(*auto simp add*: *a*)
apply *blast*
by *fastforce*
qed
then show *?case*
using *a* **by** *auto*
qed

lemma *continuous-on-closed-UN*:
assumes *finite S*
shows $((\bigwedge s.\ s \in S \Longrightarrow closed\ s) \Longrightarrow (\bigwedge s.\ s \in S \Longrightarrow continuous\text{-}on\ s\ f) \Longrightarrow continuous\text{-}on\ (\bigcup S)\ f)$
using *assms*
proof(*induction S*)
case *empty*
then show *?case* **by** *auto*
next
case (*insert x F*)
then show *?case* **using** *continuous-on-closed-Un closed-Union*
by (*simp add*: *closed-Union continuous-on-closed-Un*)
qed

lemma *chain-reparam-weak-path-line-integral*:
assumes *path-eq-chain*: *chain-reparam-weak-path* γ *one-chain* **and**
boundary-chain: *boundary-chain one-chain* **and**
line-integral-exists: $\forall\, b{\in}basis.\ \forall\,(k{::}int,\ \gamma) \in one\text{-}chain.\ line\text{-}integral\text{-}exists\ F\ \{b\}\ \gamma$ **and**
valid-path: $\forall\,(k{::}int,\ \gamma) \in one\text{-}chain.\ valid\text{-}path\ \gamma$ **and**
finite-basis: *finite basis* **and**
cont: $\forall\, b{\in}basis.\ \forall\,(k,\gamma 2){\in}one\text{-}chain.\ continuous\text{-}on\ (path\text{-}image\ \gamma 2)\ (\lambda x.\ F\ x \cdot b)$ **and**
finite-one-chain: *finite one-chain*
shows *line-integral F basis* γ = *one-chain-line-integral F basis one-chain*
line-integral-exists F basis γ

proof −
obtain *l* **where** *l-props*: *set l* = *one-chain distinct l reparam* γ (*rec-join l*) *valid-chain-list l* $l \neq []$
using *chain-reparam-weak-path-def assms*
by *auto*
have *bchain-l*: *boundary-chain* (*set l*)
using *l-props boundary-chain*
by (*simp add*: *boundary-chain-def*)
have *cont-forall*: $\forall\, b{\in}basis.\ continuous\text{-}on\ (\bigcup\,(k,\ \gamma){\in}one\text{-}chain.\ path\text{-}image\ \gamma)$

($\lambda x.$ *F x* $\cdot$ *b*)
proof
fix *b*
assume *ass*: $b \in basis$
have *continuous-on* ($\bigcup$((*path-image* $\circ$ *snd*) ‘ *one-chain*)) ($\lambda x.$ *F x* $\cdot$ *b*)
apply(*rule continuous-on-closed-UN*[**where** *?S* = (*path-image o snd*) ‘ *one-chain*])
proof
show *finite one-chain* **using** *finite-one-chain* **by** *auto*
show $\bigwedge s.\ s \in$ (*path-image* $\circ$ *snd*) ‘ *one-chain* $\Longrightarrow$ *closed s*
using *closed-valid-path-image*[*OF*] *valid-path*
by *fastforce*
show $\bigwedge s.\ s \in$ (*path-image* $\circ$ *snd*) ‘ *one-chain* $\Longrightarrow$ *continuous-on s* ($\lambda x.$ *F x* $\cdot$ *b*)
using *cont ass* **by** *force*
qed
then show *continuous-on* ($\bigcup$(*k*, γ)$\in$*one-chain. path-image* γ) ($\lambda x.$ *F x* $\cdot$ *b*)
by (*auto simp add*: *Union-eq case-prod-beta*)
qed
show *line-integral-exists F basis* γ
proof (*rule reparam-eq-line-integrals-basis*[*OF l-props*(*3*) - - *finite-basis*])
let *?*γ*2.0*=*rec-join l*
show *?*γ*2.0 piecewise-C1-differentiable-on* {*0..1*}
apply(*simp only*: *valid-path-def*[*symmetric*])
apply(*rule joined-is-valid*)
using *assms l-props* **by** *auto*
show $\forall$ *b*$\in$*basis. continuous-on* (*path-image* (*rec-join l*)) ($\lambda x.$ *F x* $\cdot$ *b*) **using** *path-image-rec-join-2*[*OF l-props*(*5*) *l-props*(*4*)] *l-props*(*1*)
using *cont-forall continuous-on-subset* **by** *blast*
show $\forall$ *b*$\in$*basis. line-integral-exists F* {*b*} (*rec-join l*)
proof
fix *b*
assume *ass*: *b*$\in$*basis*
show *line-integral-exists F* {*b*} (*rec-join l*)
proof (*rule line-integral-exists-on-rec-join*)
show *boundary-chain* (*set l*)
using *l-props boundary-chain* **by** *auto*
show *valid-chain-list l* **using** *l-props* **by** *auto*
show $\bigwedge k\ \gamma.$ (*k*, γ) $\in$ *set l* $\Longrightarrow$ *valid-path* γ **using** *l-props assms* **by** *auto*
show $\forall$ (*k*, γ)$\in$*set l. line-integral-exists F* {*b*} γ **using** *l-props line-integral-exists ass* **by** *blast*
qed
qed
qed
show *line-integral F basis* γ = *one-chain-line-integral F basis one-chain*
proof−
have *i*: *line-integral F basis* (*rec-join l*) = *one-chain-line-integral F basis one-chain*
proof (*rule one-chain-line-integral-rec-join*)

show *set l = one-chain distinct l valid-chain-list l* **using** *l-props* **by** *auto*
show *boundary-chain one-chain* **using** *boundary-chain* **by** *auto*
show ∀ (*k*, γ)∈*one-chain. line-integral-exists F basis* γ
using *line-integral-sum-basis*(*2*)[*OF finite-basis*] *line-integral-exists* **by** *blast*
show ∀ (*k*, γ)∈*one-chain. valid-path* γ **using** *valid-path* **by** *auto*
show *finite basis* **using** *finite-basis* **by** *auto*
qed
have *ii*: *line-integral F basis* γ = *line-integral F basis* (*rec-join l*)
proof (*rule reparam-eq-line-integrals-basis*[*OF l-props*(*3*) - - *finite-basis*])
let *?*γ*2.0*=*rec-join l*
show *?*γ*2.0 piecewise-C1-differentiable-on* {*0..1*}
apply(*simp only*: *valid-path-def*[*symmetric*])
apply(*rule joined-is-valid*)
using *assms l-props* **by** *auto*
show ∀ *b*∈*basis. continuous-on* (*path-image* (*rec-join l*)) (λ*x. F x* · *b*) **using**
path-image-rec-join-2[*OF l-props*(*5*) *l-props*(*4*)] *l-props*(*1*)
using *cont-forall continuous-on-subset* **by** *blast*
show ∀ *b*∈*basis. line-integral-exists F* {*b*} (*rec-join l*)
proof
fix *b*
assume *ass*: *b*∈*basis*
show *line-integral-exists F* {*b*} (*rec-join l*)
proof (*rule line-integral-exists-on-rec-join*)
show *boundary-chain* (*set l*)
using *l-props boundary-chain* **by** *auto*
show *valid-chain-list l* **using** *l-props* **by** *auto*
show ⋀*k* γ. (*k*, γ) ∈ *set l* ⟹ *valid-path* γ **using** *l-props assms* **by** *auto*
show ∀ (*k*, γ)∈*set l. line-integral-exists F* {*b*} γ **using** *l-props line-integral-exists*
ass **by** *blast*
qed
qed
qed
show *line-integral F basis* γ = *one-chain-line-integral F basis one-chain* **using**
i ii **by** *auto*
qed
qed

definition *chain-reparam-chain′* **where**
chain-reparam-chain′ one-chain1 subdiv
≡ ∃ *f*. ((⋃(*f* ‘ *one-chain1*)) = *subdiv*) ∧
(∀ *cube* ∈ *one-chain1*. *chain-reparam-weak-path* (*rec-join* [*cube*]) (*f cube*))
∧
(∀ *p*∈*one-chain1*. ∀ *p′*∈*one-chain1*. *p* ≠ *p′* ⟶ *f p* ∩ *f p′* = {}) ∧
(∀ *x* ∈ *one-chain1*. *finite* (*f x*))

lemma *chain-reparam-chain′-imp-finite-subdiv*:
assumes *finite one-chain1*
chain-reparam-chain′ one-chain1 subdiv
shows *finite subdiv*

using *assms*
by(*auto simp add*: *chain-reparam-chain′-def*)

lemma *chain-reparam-chain′-line-integral*:
assumes *chain1-eq-chain2*: *chain-reparam-chain′ one-chain1 subdiv* **and**
boundary-chain1: *boundary-chain one-chain1* **and**
boundary-chain2: *boundary-chain subdiv* **and**
line-integral-exists-on-chain2: ∀ *b*∈*basis*. ∀ (*k*::*int*, γ) ∈ *subdiv*. *line-integral-exists* *F* {*b*} γ **and**
valid-path: ∀ (*k*, γ) ∈ *subdiv*. *valid-path* γ **and**
valid-path-2: ∀ (*k*, γ) ∈ *one-chain1*. *valid-path* γ **and**
finite-chain1: *finite one-chain1* **and**
finite-basis: *finite basis* **and**
cont-field: ∀ *b*∈*basis*. ∀ (*k*, γ*2*)∈*subdiv*. *continuous-on* (*path-image* γ*2*) (λ*x*. *F* *x* · *b*)
shows *one-chain-line-integral F basis one-chain1* = *one-chain-line-integral F basis subdiv*
∀ (*k*, γ) ∈ *one-chain1*. *line-integral-exists F basis* γ
proof −
obtain *f* **where** *f-props*:
((⋃(*f* ‘ *one-chain1*)) = *subdiv*)
(∀ *cube* ∈ *one-chain1*. *chain-reparam-weak-path* (*rec-join* [*cube*]) (*f cube*))
(∀ *p*∈*one-chain1*. ∀ *p′*∈*one-chain1*. *p* ≠ *p′* ⟶ *f p* ∩ *f p′* = {})
(∀ *x* ∈ *one-chain1*. *finite* (*f x*))
using *chain1-eq-chain2*
by (*auto simp add*: *chain-reparam-chain′-def*)
then have *0*: *one-chain-line-integral F basis subdiv* = *one-chain-line-integral F basis* (⋃(*f* ‘ *one-chain1*))
by *auto*
{**fix** *k* γ
assume *ass*:(*k*, γ) ∈ *one-chain1*
have *line-integral-exists F basis* γ ∧
one-chain-line-integral F basis (*f* (*k*, γ)) = *k* ∗ *line-integral F basis* γ
proof(*cases k* = *1*)
assume *k-eq-1*: *k* = *1*
then have *i*:*chain-reparam-weak-path* γ (*f*(*k*,γ))
using *f-props*(*2*) *ass* **by** *auto*
have *ii*:*boundary-chain* (*f*(*k*,γ))
using *f-props*(*1*) *ass assms* **unfolding** *boundary-chain-def*
by *blast*
have *iii*:∀ *b*∈*basis*. ∀ (*k*, γ)∈*f* (*k*, γ). *line-integral-exists F* {*b*} γ
using *f-props*(*1*) *ass assms*
by *blast*
have *iv*: ∀ (*k*, γ)∈*f* (*k*, γ). *valid-path* γ
using *f-props*(*1*) *ass assms*
by *blast*
have *v*: ∀ *b*∈*basis*. ∀ (*k*, γ*2*)∈*f* (*k*, γ). *continuous-on* (*path-image* γ*2*) (λ*x*. *F* *x* · *b*)
using *f-props*(*1*) *ass assms*

by *blast*
have *line-integral-exists F basis* $\gamma \wedge$ *one-chain-line-integral F basis* $(f\ (k, \gamma))$ $=$ *line-integral F basis* γ
using *chain-reparam-weak-path-line-integral*[*OF i ii iii iv finite-basis v*] *ass f-props*(*4*)
by (*auto*)
then show *line-integral-exists F basis* $\gamma \wedge$ *one-chain-line-integral F basis* $(f\ (k, \gamma)) = k * $ *line-integral F basis* γ
using *k-eq-1* **by** *auto*
next
assume $k \neq 1$
then have *k-eq-neg1*: $k = -1$
using *ass boundary-chain1*
by (*auto simp add*: *boundary-chain-def*)
then have *i*:*chain-reparam-weak-path* (*reversepath* γ) $(f(k,\gamma))$
using *f-props*(*2*) *ass* **by** *auto*
have *ii*:*boundary-chain* $(f(k,\gamma))$
using *f-props*(*1*) *ass assms* **unfolding** *boundary-chain-def*
by *blast*
have *iii*:$\forall\, b \in basis.\ \forall (k, \gamma) \in f\ (k, \gamma).$ *line-integral-exists F* $\{b\}\ \gamma$
using *f-props*(*1*) *ass assms*
by *blast*
have *iv*: $\forall (k, \gamma) \in f\ (k, \gamma).$ *valid-path* γ
using *f-props*(*1*) *ass assms* **by** *blast*
have *v*: $\forall\, b \in basis.\ \forall (k, \gamma 2) \in f\ (k, \gamma).$ *continuous-on* (*path-image* $\gamma 2$) $(\lambda x.\ F\ x \cdot b)$
using *f-props*(*1*) *ass assms* **by** *blast*
have *x*:*line-integral-exists F basis* (*reversepath* γ) $\wedge$ *one-chain-line-integral F basis* $(f\ (k, \gamma))$ $=$ *line-integral F basis* (*reversepath* γ)
using *chain-reparam-weak-path-line-integral*[*OF i ii iii iv finite-basis v*] *ass f-props*(*4*)
by *auto*
have *valid-path* (*reversepath* γ)
using *f-props*(*1*) *ass assms* **by** *auto*
then have *line-integral-exists F basis* $\gamma \wedge$ *one-chain-line-integral F basis* $(f\ (k, \gamma)) = -$ (*line-integral F basis* γ)
using *line-integral-on-reverse-path reversepath-reversepath x ass*
by *metis*
then show *line-integral-exists F basis* $\gamma \wedge$ *one-chain-line-integral F basis* $(f\ (k{::}int, \gamma)) = k *$ *line-integral F basis* γ
using *k-eq-neg1* **by** *auto*
qed}
note *cube-line-integ* = *this*
have *finite-chain2*: *finite subdiv*
using *finite-chain1 f-props*(*1*) *f-props*(*4*) **by** *auto*
show *line-integral-exists-on-chain1*: $\forall (k, \gamma) \in$ *one-chain1*. *line-integral-exists F basis* γ
using *cube-line-integ* **by** *auto*
have *1*: *one-chain-line-integral F basis* $(\bigcup (f$ ‘ *one-chain1*)) = *one-chain-line-integral*

F basis one-chain1
proof −
have *0*:*one-chain-line-integral F basis* ($\bigcup$(*f* ' *one-chain1*)) =
($\sum$ *one-chain* ∈ (*f* ' *one-chain1*). *one-chain-line-integral F basis one-chain*)
proof −
have *finite*: ∀ *chain* ∈ (*f* ' *one-chain1*). *finite chain*
using *f-props*(*1*) *finite-chain2*
by (*meson Sup-upper finite-subset*)
have *disj*: ∀ *A*∈*f* ' *one-chain1*. ∀ *B*∈*f* ' *one-chain1*. $A \neq B \longrightarrow A \cap B = \{\}$
apply(*simp add*:*image-def*)
using *f-props*(*3*)
by *metis*
show *one-chain-line-integral F basis* ($\bigcup$(*f* ' *one-chain1*)) =
($\sum$ *one-chain* ∈ (*f* ' *one-chain1*). *one-chain-line-integral F basis one-chain*)
using *Groups-Big.comm-monoid-add-class.sum.Union-disjoint*[*OF finite disj*]
one-chain-line-integral-def
by *auto*
qed
have *1*:($\sum$ *one-chain* ∈ (*f* ' *one-chain1*). *one-chain-line-integral F basis one-chain*) =
one-chain-line-integral F basis one-chain1
proof −
have ($\sum$ *one-chain* ∈ (*f* ' *one-chain1*). *one-chain-line-integral F basis one-chain*) =
($\sum$ (*k*,γ)∈*one-chain1*. *k*∗(*line-integral F basis* γ))
proof −
have *i*:($\sum$ *one-chain* ∈ (*f* ' (*one-chain1* − {*p*. *f p* = {}})). *one-chain-line-integral F basis one-chain*) =
($\sum$ (*k*,γ)∈*one-chain1* − {*p*. *f p* = {}}. *k*∗(*line-integral F basis* γ))
proof −
have *inj-on f* (*one-chain1* − {*p*. *f p* = {}})
unfolding *inj-on-def*
using *f-props*(*3*) **by** *force*
then have *0*: ($\sum$ *one-chain* ∈ (*f* ' (*one-chain1* − {*p*. *f p* = {}})). *one-chain-line-integral F basis one-chain*)
= ($\sum$ (*k*, γ) ∈ (*one-chain1* − {*p*. *f p* = {}}). *one-chain-line-integral F basis* (*f* (*k*, γ)))
using *Groups-Big.comm-monoid-add-class.sum.reindex*
by *auto*
have $\bigwedge$ *k* γ. (*k*, γ) ∈ (*one-chain1* − {*p*. *f p* = {}}) ⟹
one-chain-line-integral F basis (*f*(*k*, γ)) = *k*∗ (*line-integral F basis* γ)
using *cube-line-integ* **by** *auto*
then have ($\sum$ (*k*, γ) ∈ (*one-chain1* − {*p*. *f p* = {}}). *one-chain-line-integral F basis* (*f* (*k*, γ)))
= ($\sum$ (*k*, γ) ∈ (*one-chain1* − {*p*. *f p* = {}}). *k*∗ (*line-integral F*

basis γ))
by (*auto intro*!: *Finite-Cartesian-Product.sum-cong-aux*)
then show ($\sum$ *one-chain* $\in$ (*f* ‘ (*one-chain1* $-$ {*p*. *f p* = {}})). *one-chain-line-integral F basis one-chain*) =
($\sum$ (*k*, γ) $\in$ (*one-chain1* $-$ {*p*. *f p* = {}}). *k*∗ (*line-integral F basis* γ))
using *0* **by** *auto*
qed
have $\bigwedge$ (*k*::*int*) γ. (*k*, γ) $\in$ *one-chain1* $\Longrightarrow$ (*f* (*k*, γ) = {}) $\Longrightarrow$ (*k*, γ) $\in$ {(*k′*,γ'). *k′*∗ (*line-integral F basis* γ') = *0*}
proof−
fix *k*::*int*
fix γ::*real*⇒*real*∗*real*
assume *ass*:(*k*, γ) $\in$ *one-chain1*
(*f* (*k*, γ) = {})
then have *zero-line-integral*:*one-chain-line-integral F basis* (*f* (*k*, γ)) = *0*
using *one-chain-line-integral-def*
by *auto*
show (*k*, γ) $\in$ {(*k′*::*int*, γ'). *k′* ∗ *line-integral F basis* γ' = *0*}
using *zero-line-integral cube-line-integ ass*
by *force*
qed
then have *ii*:($\sum$ *one-chain* $\in$ (*f* ‘ (*one-chain1* $-$ {*p*. *f p* = {}})). *one-chain-line-integral F basis one-chain*) =
($\sum$ *one-chain* $\in$ (*f* ‘ (*one-chain1*)). *one-chain-line-integral F basis one-chain*)
proof −
have $\bigwedge$*one-chain*. *one-chain* $\in$ (*f* ‘ (*one-chain1*)) $-$ (*f* ‘ (*one-chain1* $-$ {*p*. *f p* = {}})) $\Longrightarrow$
one-chain-line-integral F basis one-chain = *0*
proof −
fix *one-chain*
assume *one-chain* $\in$ (*f* ‘ (*one-chain1*)) $-$ (*f* ‘ (*one-chain1* $-$ {*p*. *f p* = {}}))
then have *one-chain* = {}
by *auto*
then show *one-chain-line-integral F basis one-chain* = *0*
by (*auto simp add*: *one-chain-line-integral-def*)
qed
then have *0*:($\sum$ *one-chain* $\in$ *f* ‘ (*one-chain1*) $-$ (*f* ‘ (*one-chain1* $-$ {*p*. *f p* = {}})). *one-chain-line-integral F basis one-chain*)
= *0*
using *Groups-Big.comm-monoid-add-class.sum.neutral*
by *auto*
then have ($\sum$ *one-chain* $\in$ *f* ‘ (*one-chain1*). *one-chain-line-integral F basis one-chain*)
− ($\sum$ *one-chain* $\in$ (*f* ‘ (*one-chain1* $-$ {*p*. *f p* = {}})). *one-chain-line-integral F basis one-chain*)

= *0*

proof −

have *finte*: *finite* (*f* ‘ *one-chain1*) **using** *finite-chain1* **by** *auto*

show *?thesis*

using *Groups-Big.sum-diff*[*OF finte*, *of* (*f* ‘ (*one-chain1* − {*p*. *f p* = {}}))

one-chain-line-integral F basis]

0

by *auto*

qed

then show (∑ *one-chain* ∈ (*f* ‘ (*one-chain1* − {*p*. *f p* = {}})). *one-chain-line-integral F basis one-chain*) =

(∑ *one-chain* ∈ (*f* ‘ (*one-chain1*)). *one-chain-line-integral F basis one-chain*)

by *auto*

qed

have ⋀ (*k*::*int*) γ. (*k*, γ) ∈ *one-chain1* ⟹ (*f* (*k*, γ) = {}) ⟹ *f* (*k*, γ) ∈ {*chain*. *one-chain-line-integral F basis chain* = *0*}

proof−

fix *k*::*int*

fix γ::*real*⇒*real*∗*real*

assume *ass*:(*k*, γ) ∈ *one-chain1* (*f* (*k*, γ) = {})

then have *one-chain-line-integral F basis* (*f* (*k*, γ)) = *0*

using *one-chain-line-integral-def*

by *auto*

then show *f* (*k*, γ) ∈ {*p*′. (*one-chain-line-integral F basis p*′ = *0*)}

by *auto*

qed

then have *iii*:(∑ (*k*::*int*,γ)∈*one-chain1* − {*p*. *f p* = {}}. *k*∗(*line-integral F basis* γ))

= (∑ (*k*::*int*,γ)∈*one-chain1*. *k*∗(*line-integral F basis* γ))

proof−

have ⋀ *k* γ. (*k*,γ)∈*one-chain1* − (*one-chain1* − {*p*. *f p* = {}})

⟹ *k*∗ (*line-integral F basis* γ) = *0*

proof−

fix *k* γ

assume *ass*: (*k*,γ)∈*one-chain1* − (*one-chain1* − {*p*. *f p* = {}})

then have *f*(*k*, γ) = {}

by *auto*

then have *one-chain-line-integral F basis* (*f*(*k*, γ)) = *0*

by (*auto simp add*: *one-chain-line-integral-def*)

then have *zero-line-integral*:*one-chain-line-integral F basis* (*f* (*k*, γ)) = *0*

using *one-chain-line-integral-def*

by *auto*

then show *k*∗ (*line-integral F basis* γ) = *0*

using *zero-line-integral cube-line-integ ass*

by *force*

qed
then have ∀ (*k*::*int*,*γ*)∈*one-chain1* − (*one-chain1* − {*p*. *f p* = {}}).
k∗ (*line-integral F basis γ*) = *0* **by** *auto*
then have (∑ (*k*::*int*,*γ*)∈*one-chain1* − (*one-chain1* − {*p*. *f p* = {}}). *k*∗(*line-integral F basis γ*)) = *0*
using *Groups-Big.comm-monoid-add-class.sum.neutral*
[*of one-chain1* − (*one-chain1* − {*p*. *f p* = {}}) (λ(*k*::*int*,*γ*). *k*∗ (*line-integral F basis γ*))]
by (*simp add*: *split-beta*)
then have (∑ (*k*::*int*,*γ*)∈*one-chain1*. *k*∗(*line-integral F basis γ*)) −
(∑ (*k*::*int*,*γ*)∈ (*one-chain1* − {*p*. *f p* = {}}). *k*∗(*line-integral F basis γ*)) = *0*
using *Groups-Big.sum-diff*[*OF finite-chain1*]
by (*metis* (*no-types*) *Diff-subset* ‹(∑ (*k*, *γ*)∈*one-chain1* − (*one-chain1* − {*p*. *f p* = {}}). *k* ∗ *line-integral F basis γ*) = *0*› ‹⋀*f B*. *B* ⊆ *one-chain1* ⟹ *sum f* (*one-chain1* − *B*) = *sum f one-chain1* − *sum f B*›)
then show *?thesis* **by** *auto*
qed
show *?thesis* **using** *i ii iii* **by** *auto*
qed
then show *?thesis* **using** *one-chain-line-integral-def* **by** *auto*
qed
show *?thesis* **using** *0 1* **by** *auto*
qed
show *one-chain-line-integral F basis one-chain1* = *one-chain-line-integral F basis subdiv* **using** *0 1* **by** *auto*
qed

lemma *chain-reparam-chain′-line-integral-smooth-cubes*:
assumes *chain-reparam-chain′ one-chain1 one-chain2*
∀ (*k2*,*γ2*)∈*one-chain2*. *γ2 C1-differentiable-on* {*0*..*1*}
∀ *b*∈*basis*.∀ (*k2*,*γ2*)∈*one-chain2*. *continuous-on* (*path-image γ2*) (λ*x*. *F x* · *b*)
finite basis
finite one-chain1
boundary-chain one-chain1
boundary-chain one-chain2
∀ (*k*,*γ*)∈*one-chain1*. *valid-path γ*
shows *one-chain-line-integral F basis one-chain1* = *one-chain-line-integral F basis one-chain2*
∀ (*k*, *γ*)∈*one-chain1*. *line-integral-exists F basis γ*
proof−
{**fix** *b*
assume *b* ∈ *basis*
fix *k γ*
assume (*k*, *γ*)∈*one-chain2*
have *line-integral-exists F* {*b*} *γ*
apply(*rule line-integral-exists-smooth*)
using ‹(*k*, *γ*) ∈ *one-chain2*› *assms*(*2*) **apply** *blast*
using *assms*

```
    using ‹(k, γ) ∈ one-chain2› ‹b ∈ basis› apply blast
    using ‹b ∈ basis› by blast}
  then have a: ∀ b∈basis. ∀ (k, γ)∈one-chain2. line-integral-exists F {b} γ
    by auto
  have b: ∀ (k2,γ2)∈one-chain2. valid-path γ2
    using assms(2)
     by (simp add: C1-differentiable-imp-piecewise case-prod-beta valid-path-def)

  show one-chain-line-integral F basis one-chain1 = one-chain-line-integral F basis
one-chain2
    by (rule chain-reparam-chain'-line-integral[OF assms(1) assms(6) assms(7) a
b assms(8) assms(5) assms(4) assms(3)])
  show ∀ (k, γ)∈one-chain1. line-integral-exists F basis γ
    by (rule chain-reparam-chain'-line-integral[OF assms(1) assms(6) assms(7) a
b assms(8) assms(5) assms(4) assms(3)])
qed

lemma chain-subdiv-path-pathimg-subset:
  assumes chain-subdiv-path γ subdiv
  shows ∀ (k',γ')∈subdiv. (path-image γ') ⊆ path-image γ
  using assms chain-subdiv-path.simps path-image-rec-join
  by(auto simp add: chain-subdiv-path.simps)

lemma reparam-path-image:
  assumes reparam γ1 γ2
  shows path-image γ1 = path-image γ2
  using assms
  apply (auto simp add: reparam-def path-image-def image-def bij-betw-def)
  apply (force dest!: equalityD2)
  done

lemma chain-reparam-weak-path-pathimg-subset:
  assumes chain-reparam-weak-path γ subdiv
  shows ∀ (k',γ')∈subdiv. (path-image γ') ⊆ path-image γ
  using assms
  apply(auto simp add: chain-reparam-weak-path-def)
  using path-image-rec-join reparam-path-image by blast

lemma chain-subdiv-chain-pathimg-subset':
  assumes chain-subdiv-chain one-chain subdiv
  assumes (k,γ)∈ subdiv
  shows  ∃ k' γ'. (k',γ')∈ one-chain ∧ path-image γ ⊆ path-image γ'
  using assms unfolding chain-subdiv-chain-def  pairwise-def
  apply auto
 by (metis chain-subdiv-path.cases coeff-cube-to-path.simps path-image-rec-join path-image-reversepath)

lemma chain-subdiv-chain-pathimg-subset:
  assumes chain-subdiv-chain one-chain subdiv
 shows ⋃ (path-image ‘ {γ. ∃ k. (k,γ)∈ subdiv}) ⊆ ⋃ (path-image ‘ {γ. ∃ k. (k,γ)∈
```

one-chain})
using *assms* **unfolding** *chain-subdiv-chain-def pairwise-def*
apply *auto*
by (*metis UN-iff assms chain-subdiv-chain-pathimg-subset′ subsetCE case-prodI2*)

lemma *chain-reparam-chain′-pathimg-subset′*:
assumes *chain-reparam-chain′ one-chain subdiv*
assumes $(k,\gamma)\in$ *subdiv*
shows $\exists k'\ \gamma'.\ (k',\gamma')\in$ *one-chain* $\wedge$ *path-image* $\gamma \subseteq$ *path-image* γ'
using *assms chain-reparam-weak-path-pathimg-subset*
apply(*auto simp add*: *chain-reparam-chain′-def set-eq-iff*)
using *path-image-reversepath case-prodE case-prodD old.prod.exhaust*
apply (*simp add*: *list.distinct*(*1*) *list.inject rec-join.elims*)
by (*smt case-prodD coeff-cube-to-path.simps rec-join.simps*(*2*) *reversepath-simps*(*2*) *surj-pair*)

definition *common-reparam-exists*:: (*int* × (*real* ⇒ *real* × *real*)) *set* ⇒ (*int* × (*real* ⇒ *real* × *real*)) *set* ⇒ *bool* **where**
common-reparam-exists one-chain1 one-chain2 ≡
(∃ *subdiv ps1 ps2*.
chain-reparam-chain′ (*one-chain1* − *ps1*) *subdiv* ∧
chain-reparam-chain′ (*one-chain2* − *ps2*) *subdiv* ∧
(∀ (*k*, γ)∈*subdiv*. γ *C1-differentiable-on* {*0..1*}) ∧
boundary-chain subdiv ∧
(∀ (*k*, γ)∈*ps1*. *point-path* γ) ∧
(∀ (*k*, γ)∈*ps2*. *point-path* γ))

lemma *common-reparam-exists-imp-eq-line-integral*:
assumes *finite-basis*: *finite basis* **and**
finite one-chain1
finite one-chain2
boundary-chain (*one-chain1*::(*int* × (*real* ⇒ *real* × *real*)) *set*)
boundary-chain (*one-chain2*::(*int* × (*real* ⇒ *real* × *real*)) *set*)
∀ (*k2*, γ*2*)∈*one-chain2*. ∀ *b*∈*basis*. *continuous-on* (*path-image* γ*2*) (λ*x*. *F x* · *b*)
(*common-reparam-exists one-chain1 one-chain2*)
(∀ (*k*, γ)∈*one-chain1*. *valid-path* γ)
(∀ (*k*, γ)∈*one-chain2*. *valid-path* γ)
shows *one-chain-line-integral F basis one-chain1* = *one-chain-line-integral F basis one-chain2*
∀ (*k*, γ)∈*one-chain1*. *line-integral-exists F basis* γ
proof−
obtain *subdiv ps1 ps2* **where** *props*:
chain-reparam-chain′ (*one-chain1* − *ps1*) *subdiv*
chain-reparam-chain′ (*one-chain2* − *ps2*) *subdiv*
(∀ (*k*, γ)∈*subdiv*. γ *C1-differentiable-on* {*0..1*})
boundary-chain subdiv
(∀ (*k*, γ)∈*ps1*. *point-path* γ)

(∀ (*k*, *γ*)∈*ps2*. *point-path* *γ*)
using *assms*
by (*auto simp add*: *common-reparam-exists-def*)
have *subdiv-valid*: (∀ (*k*, *γ*)∈*subdiv*. *valid-path* *γ*)
apply(*simp add*: *valid-path-def*)
using *props*(*3*)
using *C1-differentiable-imp-piecewise* **by** *blast*
have *onechain-boundary1*: *boundary-chain* (*one-chain1* − *ps1*) **using** *assms*(*4*) **by** (*auto simp add*: *boundary-chain-def*)
have *onechain-boundary2*: *boundary-chain* (*one-chain2* − *ps1*) **using** *assms*(*5*) **by** (*auto simp add*: *boundary-chain-def*)
{**fix** *k2* *γ2* *b*
assume *ass*: (*k2*, *γ2*)∈*subdiv* *b*∈*basis*
have ⋀ *k* *γ*. (*k*, *γ*) ∈ *subdiv* ⟹ ∃ *k′* *γ′*. (*k′*, *γ′*) ∈ *one-chain2* ∧ *path-image* *γ* ⊆ *path-image* *γ′*
by (*meson chain-reparam-chain′-pathimg-subset′ props Diff-subset subsetCE*)
then have *continuous-on* (*path-image* *γ2*) (*λx*. *F x* · *b*)
using *assms*(*6*) *continuous-on-subset*[**where** *?f* = (*λx*. *F x* · *b*)] *ass*
apply(*auto simp add*: *subset-iff*)
by (*metis* (*mono-tags*, *lifting*) *case-prodD*)}
then have *cont-field*: ∀ *b*∈*basis*. ∀ (*k2*, *γ2*)∈*subdiv*. *continuous-on* (*path-image* *γ2*) (*λx*. *F x* · *b*)
by *auto*
have *one-chain1-ps-valid*: (∀ (*k*, *γ*)∈*one-chain1* − *ps1*. *valid-path* *γ*) **using** *assms* **by** *auto*
have *one-chain2-ps-valid*: (∀ (*k*, *γ*)∈*one-chain2* − *ps1*. *valid-path* *γ*) **using** *assms* **by** *auto*
have *0*: *one-chain-line-integral F basis* (*one-chain1* − *ps1*) = *one-chain-line-integral F basis subdiv*
apply(*rule chain-reparam-chain′-line-integral-smooth-cubes*[*OF props*(*1*) *props*(*3*) *cont-field finite-basis*])
using *props assms*
apply *blast*
using *props assms*
using *onechain-boundary1* **apply** *blast*
using *props assms*
apply *blast*
using *one-chain1-ps-valid* **by** *blast*
have *1*:∀ (*k*, *γ*)∈(*one-chain1* − *ps1*). *line-integral-exists F basis* *γ*
apply(*rule chain-reparam-chain′-line-integral-smooth-cubes*[*OF props*(*1*) *props*(*3*) *cont-field finite-basis*])
using *props assms*
apply *blast*
using *props assms*
using *onechain-boundary1* **apply** *blast*
using *props assms*
apply *blast*
using *one-chain1-ps-valid* **by** *blast*
have *2*: *one-chain-line-integral F basis* (*one-chain2* − *ps2*) = *one-chain-line-integral*

F basis subdiv
apply(*rule chain-reparam-chain′-line-integral-smooth-cubes*[*OF props*(*2*) *props*(*3*) *cont-field finite-basis*])
using *props assms*
apply *blast*
apply (*simp add*: *assms*(*5*) *boundary-chain-diff*)
apply (*simp add*: *props*(*4*))
by (*simp add*: *assms*(*9*))
have *3*:∀ (*k*, *γ*)∈(*one-chain2* − *ps2*). *line-integral-exists F basis γ*
apply(*rule chain-reparam-chain′-line-integral-smooth-cubes*[*OF props*(*2*) *props*(*3*) *cont-field finite-basis*])
using *props assms*
apply *blast*
apply (*simp add*: *assms*(*5*) *boundary-chain-diff*)
apply (*simp add*: *props*(*4*))
by (*simp add*: *assms*(*9*))
show *line-int-ex-chain1*: ∀ (*k*, *γ*)∈*one-chain1*. *line-integral-exists F basis γ*
using *0*
using *1 finite-basis line-integral-exists-point-path props*(*5*) **by** *fastforce*
then show *one-chain-line-integral F basis one-chain1* = *one-chain-line-integral F basis one-chain2*
using *0 1 2 3*
using *assms*(*2*−*3*) *finite-basis one-chain-line-integral-point-paths props*(*5*) *props*(*6*) **by** *auto*
qed

definition *subcube* :: *real* ⇒ *real* ⇒ (*int* × (*real* ⇒ *real* × *real*)) ⇒ (*int* × (*real* ⇒ *real* × *real*)) **where**
subcube a b cube = (*fst cube*, *subpath a b* (*snd cube*))

lemma *subcube-valid-path*:
assumes *valid-path* (*snd cube*) *a* ∈ {*0..1*} *b* ∈ {*0..1*}
shows *valid-path* (*snd* (*subcube a b cube*))
using *valid-path-subpath*[*OF assms*] **by** (*auto simp add*: *subcube-def*)

end
theory *Green*
imports *Paths Derivs Integrals General-Utils*
begin

lemma *frontier-Un-subset-Un-frontier*:
frontier (*s* ∪ *t*) ⊆ (*frontier s*) ∪ (*frontier t*)
by (*simp add*: *frontier-def Un-Diff*) (*auto simp add*: *closure-def interior-def islimpt-def*)

definition *has-partial-derivative*:: ((′*a*::*euclidean-space*) ⇒ ′*b*::*euclidean-space*) ⇒ ′*a* ⇒ (′*a* ⇒ ′*b*) ⇒ (′*a*) ⇒ *bool* **where**
has-partial-derivative F base-vec F′ a
≡ ((*λx*::′*a*::*euclidean-space*. *F*((*a* − ((*a* · *base-vec*) $*_R$ *base-vec*)) + (*x* ·

base-vec) $*_R$ *base-vec*))
has-derivative F') (*at a*)

definition *has-partial-vector-derivative*:: (('*a*::*euclidean-space*) ⇒ '*b*::*euclidean-space*) ⇒ '*a* ⇒ ('*b*) ⇒ ('*a*) ⇒ *bool* **where**
has-partial-vector-derivative F base-vec F' a
≡ ((λx. *F*((*a* − ((*a* · *base-vec*) $*_R$ *base-vec*)) + *x* $*_R$ *base-vec*))
has-vector-derivative F') (*at* (*a* · *base-vec*))

definition *partially-vector-differentiable* **where**
partially-vector-differentiable F base-vec p ≡ (∃ F'. *has-partial-vector-derivative F base-vec F' p*)

definition *partial-vector-derivative*:: (('*a*::*euclidean-space*) ⇒ '*b*::*euclidean-space*) ⇒ '*a* ⇒ '*a* ⇒ '*b* **where**
partial-vector-derivative F base-vec a
≡ (*vector-derivative* (λx. *F*((*a* − ((*a* · *base-vec*) $*_R$ *base-vec*)) + *x* $*_R$ *base-vec*)) (*at* (*a* · *base-vec*)))

lemma *partial-vector-derivative-works*:
assumes *partially-vector-differentiable F base-vec a*
shows *has-partial-vector-derivative F base-vec* (*partial-vector-derivative F base-vec a*) *a*
proof −
obtain F' **where** F'*-prop*: ((λx. *F*((*a* − ((*a* · *base-vec*) $*_R$ *base-vec*)) + *x* $*_R$ *base-vec*))
has-vector-derivative F') (*at* (*a* · *base-vec*))
using *assms partially-vector-differentiable-def has-partial-vector-derivative-def*
by *blast*
show *?thesis*
using *Derivative.differentiableI-vector*[*OF* F'*-prop*]
by(*simp add*: *vector-derivative-works partial-vector-derivative-def*[*symmetric*]
has-partial-vector-derivative-def[*symmetric*])
qed

lemma *fundamental-theorem-of-calculus-partial-vector*:
fixes *a b*:: *real* **and**
F:: ('*a*::*euclidean-space* ⇒ '*b*::*euclidean-space*) **and**
i:: '*a* **and**
j:: '*b* **and**
F-j-i:: ('*a*::*euclidean-space* ⇒ *real*)
assumes *a-leq-b*: $a \leq b$ **and**
Base-vecs: *i* ∈ *Basis j* ∈ *Basis* **and**
no-i-component: *c* · *i* = *0* **and**
has-partial-deriv: ∀ *p* ∈ *D*. *has-partial-vector-derivative* (λx. (*F x*) · *j*) *i* (*F-j-i p*) *p* **and**
domain-subset-of-D: {*x* $*_R$ *i* + *c* |*x*. *a* ≤ *x* ∧ *x* ≤ *b*} ⊆ *D*
shows ((λx. *F-j-i*(*x* $*_R$ *i* + *c*)) *has-integral*
F(*b* $*_R$ *i* + *c*) · *j* − *F*(*a* $*_R$ *i* + *c*) · *j*) (*cbox a b*)

proof −
let *?domain* = {*v*. ∃ *x*. *a* ≤ *x* ∧ *x* ≤ *b* ∧ *v* = *x* $*_R$ *i* + *c*}
have ∀ *x*∈ *?domain*. *has-partial-vector-derivative* (λ*x*. (*F x*) · *j*) *i* (*F-j-i x*) *x*
using *has-partial-deriv domain-subset-of-D*
by *auto*
then have ∀ *x*∈ (*cbox a b*). ((λ*x*. *F*(*x* $*_R$ *i* + *c*) · *j*) *has-vector-derivative* (*F-j-i*(*x* $*_R$ *i* + *c*))) (*at x*)
proof(*clarsimp simp add*: *has-partial-vector-derivative-def*)
fix *x*::*real*
assume *range-of-x*: *a* ≤ *x x* ≤ *b*
assume *ass2*: ∀ *x*. (∃ *z*≥*a*. *z* ≤ *b* ∧ *x* = *z* $*_R$ *i* + *c*) ⟶
((λ*z*. *F*(*x* − (*x* · *i*) $*_R$ *i* + *z* $*_R$ *i*) · *j*) *has-vector-derivative F-j-i x*) (*at* (*x* · *i*))
have ((λ*z*. *F*((*x* $*_R$ *i* + *c*) − ((*x* $*_R$ *i* + *c*) · *i*) $*_R$ *i* + *z* $*_R$ *i*) · *j*) *has-vector-derivative F-j-i* (*x* $*_R$ *i* + *c*)) (*at* ((*x* $*_R$ *i* + *c*) · *i*))
using *range-of-x ass2* **by** *auto*
then have *0*: ((λ*x*. *F*(*c* + *x* $*_R$ *i*) · *j*) *has-vector-derivative F-j-i* (*x* $*_R$ *i* + *c*)) (*at x*)
by (*simp add*: *assms*(*2*) *inner-left-distrib no-i-component*)
have *1*: (λ*x*. *F*(*x* $*_R$ *i* + *c*) · *j*) = (λ*x*. *F*(*c* + *x* $*_R$ *i*) · *j*)
by (*simp add*: *add.commute*)
then show ((λ*x*. *F*(*x* $*_R$ *i* + *c*) · *j*) *has-vector-derivative F-j-i* (*x* $*_R$ *i* + *c*)) (*at x*)
using *0* **and** *1* **by** *auto*
qed
then have ∀ *x*∈ (*cbox a b*). ((λ*x*. *F*(*x* $*_R$ *i* + *c*) · *j*) *has-vector-derivative* (*F-j-i*(*x* $*_R$ *i* + *c*))) (*at-within x* (*cbox a b*))
using *has-vector-derivative-at-within*
by *blast*
then show ((λ*x*. *F-j-i*(*x* $*_R$ *i* + *c*)) *has-integral*
F(*b* $*_R$ *i* + *c*) · *j* − *F*(*a* $*_R$ *i* + *c*) · *j*) (*cbox a b*)
using *fundamental-theorem-of-calculus*[*of a b* (λ*x*::*real*. *F*(*x* $*_R$ *i* + *c*) · *j*) (λ*x*::*real*. *F-j-i*(*x* $*_R$ *i* + *c*))] **and**
a-leq-b
by *auto*
qed

lemma *fundamental-theorem-of-calculus-partial-vector-gen*:
fixes *k1 k2*:: *real* **and**
F:: (′*a*::*euclidean-space* ⇒ ′*b*::*euclidean-space*) **and**
i:: ′*a* **and**
F-i:: (′*a*::*euclidean-space* ⇒ ′*b*)
assumes *a-leq-b*: *k1* ≤ *k2* **and**
unit-len: *i* · *i* = *1* **and**
no-i-component: *c* · *i* = *0* **and**
has-partial-deriv: ∀ *p* ∈ *D*. *has-partial-vector-derivative F i* (*F-i p*) *p* **and**
domain-subset-of-D: {*v*. ∃ *x*. *k1* ≤ *x* ∧ *x* ≤ *k2* ∧ *v* = *x* $*_R$ *i* + *c*} ⊆ *D*
shows ((λ*x*. *F-i*(*x* $*_R$ *i* + *c*)) *has-integral*
F(*k2* $*_R$ *i* + *c*) − *F*(*k1* $*_R$ *i* + *c*)) (*cbox k1 k2*)

proof −
let *?domain* = {*v*. ∃ *x*. *k1* ≤ *x* ∧ *x* ≤ *k2* ∧ *v* = *x* $*_R$ *i* + *c*}
have ∀ *x*∈ *?domain*. *has-partial-vector-derivative F i* (*F-i x*) *x*
using *has-partial-deriv domain-subset-of-D*
by *auto*
then have ∀ *x*∈ (*cbox k1 k2*). ((λ*x*. *F*(*x* $*_R$ *i* + *c*)) *has-vector-derivative* (*F-i*(*x* $*_R$ *i* + *c*))) (*at x*)
proof (*clarsimp simp add*: *has-partial-vector-derivative-def*)
fix *x*::*real*
assume *range-of-x*: *k1* ≤ *x x* ≤ *k2*
assume *ass2*: ∀ *x*. (∃ *z*≥*k1*. *z* ≤ *k2* ∧ *x* = *z* $*_R$ *i* + *c*) ⟶
((λ*z*. *F*(*x* − (*x* · *i*) $*_R$ *i* + *z* $*_R$ *i*)) *has-vector-derivative F-i x*) (*at* (*x* · *i*))
have ((λ*z*. *F*((*x* $*_R$ *i* + *c*) − ((*x* $*_R$ *i* + *c*) · *i*) $*_R$ *i* + *z* $*_R$ *i*)) *has-vector-derivative F-i* (*x* $*_R$ *i* + *c*)) (*at* ((*x* $*_R$ *i* + *c*) · *i*))
using *range-of-x ass2* **by** *auto*
then have *0*: ((λ*x*. *F*(*c* + *x* $*_R$ *i*)) *has-vector-derivative F-i* (*x* $*_R$ *i* + *c*)) (*at x*)
by (*simp add*: *inner-commute inner-right-distrib no-i-component unit-len*)
have *1*: (λ*x*. *F*(*x* $*_R$ *i* + *c*)) = (λ*x*. *F*(*c* + *x* $*_R$ *i*))
by (*simp add*: *add.commute*)
then show ((λ*x*. *F*(*x* $*_R$ *i* + *c*)) *has-vector-derivative F-i* (*x* $*_R$ *i* + *c*)) (*at x*) **using** *0* **and** *1* **by** *auto*
qed
then have ∀ *x*∈ (*cbox k1 k2*). ((λ*x*. *F*(*x* $*_R$ *i* + *c*)) *has-vector-derivative* (*F-i*(*x* $*_R$ *i* + *c*))) (*at-within x* (*cbox k1 k2*))
using *has-vector-derivative-at-within*
by *blast*
then show ((λ*x*. *F-i*(*x* $*_R$ *i* + *c*)) *has-integral*
F(*k2* $*_R$ *i* + *c*) − *F*(*k1* $*_R$ *i* + *c*)) (*cbox k1 k2*)
using *fundamental-theorem-of-calculus*[*of k1 k2* (λ*x*::*real*. *F*(*x* $*_R$ *i* + *c*)) (λ*x*::*real*. *F-i*(*x* $*_R$ *i* + *c*))] **and**
a-leq-b
by *auto*
qed

lemma *add-scale-img*:
assumes *a* < *b* **shows** (λ*x*::*real*. *a* + (*b* − *a*) ∗ *x*) ‘ {*0* .. *1*} = {*a* .. *b*}
using *assms*
apply (*auto simp*: *algebra-simps affine-ineq image-iff*)
using *less-eq-real-def* **apply** *force*
apply (*rule-tac x*=(*x*−*a*)/(*b*−*a*) **in** *bexI*)
apply (*auto simp*: *field-simps*)
done

lemma *add-scale-img′*:
assumes *a* ≤ *b*
shows (λ*x*::*real*. *a* + (*b* − *a*) ∗ *x*) ‘ {*0* .. *1*} = {*a* .. *b*}
proof (*cases a* = *b*)

case *True*
then show *?thesis* **by** *force*
next
case *False*
then show *?thesis*
using *add-scale-img assms* **by** *auto*
qed

definition *analytically-valid*:: *'a::euclidean-space set* ⇒ (*'a* ⇒ *'b*::{*euclidean-space,times,zero-neq-one*}) ⇒ *'a* ⇒ *bool* **where**
analytically-valid s F i ≡
(∀ *a* ∈ *s*. *partially-vector-differentiable F i a*) ∧
continuous-on s F ∧ — TODO: should we replace this with saying that *F* is partially diffrerentiable on *Dy*,
— i.e. there is a partial derivative on every dimension
integrable lborel (λ*p*. (*partial-vector-derivative F i*) *p* ∗ *indicator s p*) ∧
(λ*x*. *integral UNIV* (λ*y*. (*partial-vector-derivative F i* (*y* $*_R$ *i* + *x* $*_R$ (∑ *b* ∈(*Basis* − {*i*}). *b*)))
∗ (*indicator s* (*y* $*_R$ *i* + *x* $*_R$ (∑ *b* ∈ *Basis* − {*i*}. *b*))))) ∈ *borel-measurable lborel*

lemma *analytically-valid-imp-part-deriv-integrable-on*:
assumes *analytically-valid* (*s*::(*real*∗*real*) *set*) (*f*::(*real*∗*real*)⇒ *real*) *i*
shows (*partial-vector-derivative f i*) *integrable-on s*
proof −
have *integrable lborel* (λ*p*. *partial-vector-derivative f i p* ∗ *indicator s p*)
using *assms*
by(*simp add*: *analytically-valid-def indic-ident*)
then have *integrable lborel* (λ*p*::(*real*∗*real*). *if p* ∈ *s then partial-vector-derivative f i p else 0*)
using *indic-ident*[*of partial-vector-derivative f i*]
by (*simp add*: *indic-ident*)
then have (λ*x*. *if x* ∈ *s then partial-vector-derivative f i x else 0*) *integrable-on UNIV*
using *Equivalence-Lebesgue-Henstock-Integration.integrable-on-lborel*
by *auto*
then show (*partial-vector-derivative f i*) *integrable-on s*
using *integrable-restrict-UNIV*
by *auto*
qed

definition *typeII-twoCube* :: ((*real* ∗ *real*) ⇒ (*real* ∗ *real*)) ⇒ *bool* **where**
typeII-twoCube twoC
≡ ∃ *a b g1 g2*. *a* < *b* ∧ (∀ *x* ∈ {*a*..*b*}. *g2 x* ≤ *g1 x*) ∧
twoC = (λ(*y*, *x*). ((*1* − *y*) ∗ (*g2* ((*1*−*x*)∗*a* + *x*∗*b*)) + *y* ∗ (*g1*

```
((1−x)∗a + x∗b)),
                              (1−x)∗a + x∗b)) ∧
                    g1 piecewise-C1-differentiable-on {a .. b} ∧
                    g2 piecewise-C1-differentiable-on {a .. b}

abbreviation unit-cube where unit-cube ≡ cbox (0,0) (1::real,1::real)

definition cubeImage:: two-cube ⇒ ((real∗real) set) where
  cubeImage twoC ≡ (twoC ‘ unit-cube)

lemma typeII-twoCubeImg:
  assumes typeII-twoCube twoC
  shows ∃ a b g1 g2. a < b ∧ (∀ x ∈ {a .. b}. g2 x ≤ g1 x) ∧
                    cubeImage twoC = {(y,x). x ∈ {a..b} ∧ y ∈ {g2 x .. g1 x}}
                    ∧ twoC = (λ(y, x). ((1 − y) ∗ g2 ((1 − x) ∗ a + x ∗ b) + y ∗
g1 ((1 − x) ∗ a + x ∗ b), (1 − x) ∗ a + x ∗ b))
                         ∧ g1 piecewise-C1-differentiable-on {a .. b} ∧ g2 piece-
wise-C1-differentiable-on {a .. b}
  using assms
proof (simp add: typeII-twoCube-def cubeImage-def image-def)
  assume ∃ a b. a < b ∧ (∃ g1 g2. (∀ x∈{a..b}. g2 x ≤ g1 x) ∧
                      twoC = (λ(y, x). ((1 − y) ∗ g2 ((1 − x) ∗ a + x ∗ b) + y
∗ g1 ((1 − x) ∗ a + x ∗ b), (1 − x) ∗ a + x ∗ b)) ∧
                           g1 piecewise-C1-differentiable-on {a..b} ∧ g2 piece-
wise-C1-differentiable-on {a..b})
  then obtain a b g1 g2 where
    twoCisTypeII: a < b
    (∀ x∈{a..b}. g2 x ≤ g1 x)
    twoC = (λ(y, x). ((1 − y) ∗ g2 ((1 − x) ∗ a + x ∗ b) + y ∗ g1 ((1 − x) ∗ a
+ x ∗ b), (1 − x) ∗ a + x ∗ b))
    g1 piecewise-C1-differentiable-on {a .. b}
    g2 piecewise-C1-differentiable-on {a .. b}
    by auto
  have ab1: a − x ∗ a + x ∗ b ≤ b if a1: 0 ≤ x x ≤ 1 for x
    using that apply (simp add: algebra-simps)
    by (metis affine-ineq less-eq-real-def mult.commute twoCisTypeII(1))
  have ex: ∃ z∈Green.unit-cube.
              (d, c) =
              (case z of
               (y, x) ⇒
                 (g2 (a − x ∗ a + x ∗ b) − y ∗ g2 (a − x ∗ a + x ∗ b) + y ∗ g1 (a
− x ∗ a + x ∗ b),
                  a − x ∗ a + x ∗ b))
    if c-bounds: a ≤ c c ≤ b and d-bounds: g2 c ≤ d d ≤ g1 c for c d
  proof −
    have b-minus-a-nzero: b − a ≠ 0 using twoCisTypeII(1) by auto
    have x-witness: ∃ k1. c = (1 − k1)∗a + k1 ∗ b ∧ 0 ≤ k1 ∧ k1 ≤ 1
      apply (rule-tac x=(c − a)/(b − a) in exI)
      using c-bounds ‹a < b› apply (simp add: divide-simps algebra-simps)
```

using *sum-sqs-eq* **by** *blast*
have *y-witness*: $\exists k2.\ d = (1 - k2)*(g2\ c) + k2 * (g1\ c) \wedge 0 \leq k2 \wedge k2 \leq 1$
proof (*cases* $g1\ c - g2\ c = 0$)
case *True*
with *d-bounds* **show** *?thesis* **by** (*fastforce simp add*: *algebra-simps*)
next
case *False*
let $?k2 = (d - g2\ c)/(g1\ c - g2\ c)$
have *k2-in-bounds*: $0 \leq ?k2 \wedge ?k2 \leq 1$
using *twoCisTypeII*(*2*) *c-bounds d-bounds False* **by** *simp*
have $d = (1 - ?k2) * g2\ c + ?k2 * g1\ c$
using *False sum-sqs-eq* **by** (*fastforce simp add*: *divide-simps algebra-simps*)
with *k2-in-bounds* **show** *?thesis*
by *fastforce*
qed
show $\exists x \in unit\text{-}cube.\ (d, c) = (case\ x\ of\ (y, x) \Rightarrow (g2\ (a - x * a + x * b) - y * g2\ (a - x * a + x * b) + y * g1\ (a - x * a + x * b), a - x * a + x * b))$
using *x-witness y-witness* **by** (*force simp add*: *left-diff-distrib*)
qed
have $\{y.\ \exists x \in unit\text{-}cube.\ y = twoC\ x\} = \{(y, x).\ a \leq x \wedge x \leq b \wedge g2\ x \leq y \wedge y \leq g1\ x\}$
apply (*auto simp add*: *twoCisTypeII ab1 left-diff-distrib ex*)
using *order.order-iff-strict twoCisTypeII*(*1*) **apply** *fastforce*
apply (*smt affine-ineq atLeastAtMost-iff mult-left-mono twoCisTypeII*)+
done
then show $\exists a\ b.\ a < b \wedge (\exists g1\ g2.\ (\forall x \in \{a..b\}.\ g2\ x \leq g1\ x) \wedge$
$\{y.\ \exists x \in unit\text{-}cube.\ y = twoC\ x\} = \{(y, x).\ a \leq x \wedge x \leq b \wedge g2\ x \leq y \wedge y \leq g1\ x\} \wedge$
$twoC = (\lambda(y, x).\ ((1 - y) * g2\ ((1 - x) * a + x * b) + y * g1\ ((1 - x) * a + x * b), (1 - x) * a + x * b)) \wedge$
g1 piecewise-C1-differentiable-on $\{a..b\} \wedge$ *g2 piecewise-C1-differentiable-on* $\{a..b\})$
using *twoCisTypeII* **by** *blast*
qed

definition *horizontal-boundary* :: *two-cube* ⇒ *one-chain* **where**
horizontal-boundary twoC $\equiv \{(1, (\lambda x.\ twoC(x,0))), (-1, (\lambda x.\ twoC(x,1)))\}$

definition *vertical-boundary* :: *two-cube* ⇒ *one-chain* **where**
vertical-boundary twoC $\equiv \{(-1, (\lambda y.\ twoC(0,y))), (1, (\lambda y.\ twoC(1,y)))\}$

definition *boundary* :: *two-cube* ⇒ *one-chain* **where**
boundary twoC ≡ *horizontal-boundary twoC* ∪ *vertical-boundary twoC*

definition *valid-two-cube* **where**
valid-two-cube twoC ≡ *card* (*boundary twoC*) = *4*

definition *two-chain-integral*:: *two-chain* ⇒ ((*real*∗*real*)⇒(*real*)) ⇒ *real* **where**
two-chain-integral twoChain F $\equiv \sum C \in twoChain.$ (*integral* (*cubeImage C*) *F*)

definition *valid-two-chain* **where**
valid-two-chain twoChain ≡ (∀ *twoCube* ∈ *twoChain*. *valid-two-cube twoCube*) ∧ *pairwise* (λ*c1 c2*. ((*boundary c1*) ∩ (*boundary c2*)) = {}) *twoChain* ∧ *inj-on cubeImage twoChain*

definition *two-chain-boundary*:: *two-chain* ⇒ *one-chain* **where**
two-chain-boundary twoChain == ⋃(*boundary* ‘ *twoChain*)

definition *gen-division* **where**
gen-division s S ≡ (*finite S* ∧ (⋃*S* = *s*) ∧ *pairwise* (λ*X Y*. *negligible* (*X* ∩ *Y*)) *S*)

definition *two-chain-horizontal-boundary*:: *two-chain* ⇒ *one-chain* **where**
two-chain-horizontal-boundary twoChain ≡ ⋃(*horizontal-boundary* ‘ *twoChain*)

definition *two-chain-vertical-boundary*:: *two-chain* ⇒ *one-chain* **where**
two-chain-vertical-boundary twoChain ≡ ⋃(*vertical-boundary* ‘ *twoChain*)

definition *only-horizontal-division* **where**
only-horizontal-division one-chain two-chain
≡ ∃ $\mathcal{H}$ $\mathcal{V}$. *finite* $\mathcal{H}$ ∧ *finite* $\mathcal{V}$ ∧
(∀ (*k*,γ) ∈ $\mathcal{H}$.
(∃ (*k*′, γ′) ∈ *two-chain-horizontal-boundary two-chain*.
(∃ *a* ∈ {*0..1*}. ∃ *b* ∈ {*0..1*}. *a* ≤ *b* ∧ *subpath a b* γ′ = γ))) ∧
(*common-sudiv-exists* (*two-chain-vertical-boundary two-chain*) $\mathcal{V}$
∨ *common-reparam-exists* $\mathcal{V}$ (*two-chain-vertical-boundary two-chain*))
∧
boundary-chain $\mathcal{V}$ ∧
one-chain = $\mathcal{H}$ ∪ $\mathcal{V}$ ∧ (∀ (*k*,γ)∈$\mathcal{V}$. *valid-path* γ)

lemma *sum-zero-set*:
assumes ∀ *x* ∈ *s*. *f x* = *0 finite s finite t*
shows *sum f* (*s* ∪ *t*) = *sum f t*
using *assms*
by (*simp add*: *IntE sum.union-inter-neutral sup-commute*)

abbreviation *valid-typeII-division s twoChain* ≡ ((∀ *twoCube* ∈ *twoChain*. *typeII-twoCube twoCube*) ∧
(*gen-division s* (*cubeImage* ‘ *twoChain*)) ∧
(*valid-two-chain twoChain*))

lemma *two-chain-vertical-boundary-is-boundary-chain*:
shows *boundary-chain* (*two-chain-vertical-boundary twoChain*)
by(*simp add*: *boundary-chain-def two-chain-vertical-boundary-def vertical-boundary-def*)

lemma *two-chain-horizontal-boundary-is-boundary-chain*:
shows *boundary-chain* (*two-chain-horizontal-boundary twoChain*)

by(*simp add*: *boundary-chain-def two-chain-horizontal-boundary-def horizontal-boundary-def*)

definition *typeI-twoCube* :: *two-cube* ⇒ *bool* **where**
typeI-twoCube (*twoC*::*two-cube*)
≡ ∃ *a b g1 g2*. *a* < *b* ∧ (∀ *x* ∈ {*a*..*b*}. *g2 x* ≤ *g1 x*) ∧
twoC = (λ(*x*,*y*). ((*1*−*x*)∗*a* + *x*∗*b*,
(*1* − *y*) ∗ (*g2* ((*1*−*x*)∗*a* + *x*∗*b*)) + *y* ∗ (*g1* ((*1*−*x*)∗*a* + *x*∗*b*)))) ∧
g1 piecewise-C1-differentiable-on {*a*..*b*} ∧
g2 piecewise-C1-differentiable-on {*a*..*b*}

lemma *typeI-twoCubeImg*:
assumes *typeI-twoCube twoC*
shows ∃ *a b g1 g2*. *a* < *b* ∧ (∀ *x* ∈ {*a* .. *b*}. *g2 x* ≤ *g1 x*) ∧
cubeImage twoC = {(*x*,*y*). *x* ∈ {*a*..*b*} ∧ *y* ∈ {*g2 x* .. *g1 x*}} ∧
twoC = (λ(*x*, *y*). ((*1* − *x*) ∗ *a* + *x* ∗ *b*, (*1* − *y*) ∗ *g2* ((*1* − *x*) ∗ *a* + *x* ∗ *b*) + *y* ∗ *g1* ((*1* − *x*) ∗ *a* + *x* ∗ *b*))) ∧
g1 piecewise-C1-differentiable-on {*a* .. *b*} ∧ *g2 piecewise-C1-differentiable-on* {*a* .. *b*}
proof −
have ∃ *a b*. *a* < *b* ∧
(∃ *g1 g2*. (∀ *x*∈{*a*..*b*}. *g2 x* ≤ *g1 x*) ∧
twoC = (λ(*x*, *y*). ((*1* − *x*) ∗ *a* + *x* ∗ *b*, (*1* − *y*) ∗ *g2* ((*1* − *x*) ∗ *a* + *x* ∗ *b*) + *y* ∗ *g1* ((*1* − *x*) ∗ *a* + *x* ∗ *b*)))∧
g1 piecewise-C1-differentiable-on {*a* .. *b*} ∧ *g2 piecewise-C1-differentiable-on* {*a* .. *b*})
using *assms* **by** (*simp add*: *typeI-twoCube-def*)
then obtain *a b g1 g2* **where**
twoCisTypeI: *a* < *b*
(∀ *x*∈{*a*..*b*}. *g2 x* ≤ *g1 x*)
twoC = (λ(*x*, *y*). ((*1* − *x*) ∗ *a* + *x* ∗ *b*, (*1* − *y*) ∗ *g2* ((*1* − *x*) ∗ *a* + *x* ∗ *b*) + *y* ∗ *g1* ((*1* − *x*) ∗ *a* + *x* ∗ *b*)))
g1 piecewise-C1-differentiable-on {*a* .. *b*}
g2 piecewise-C1-differentiable-on {*a* .. *b*}
by *auto*
have *ex*: ∃ *z*∈*Green.unit-cube*.
(*c*, *d*) =
(*case z of*
(*x*, *y*) ⇒
(*a* − *x* ∗ *a* + *x* ∗ *b*,
g2 (*a* − *x* ∗ *a* + *x* ∗ *b*) − *y* ∗ *g2* (*a* − *x* ∗ *a* + *x* ∗ *b*) + *y* ∗ *g1* (*a* − *x* ∗ *a* + *x* ∗ *b*)))
if *c-bounds*: *a* ≤ *c c* ≤ *b* **and** *d-bounds*: *g2 c* ≤ *d d* ≤ *g1 c* **for** *c d*
proof −
have *x-witness*: ∃ *k1*. *c* = (*1* − *k1*)∗*a* + *k1* ∗ *b* ∧ *0* ≤ *k1* ∧ *k1* ≤ *1*
proof −
let *?k1* = (*c* − *a*)/(*b* − *a*)
have *k1-in-bounds*: *0* ≤ (*c* − *a*)/(*b* − *a*) ∧ (*c* − *a*)/(*b* − *a*) ≤ *1*
using *twoCisTypeI*(*1*) *c-bounds* **by** *simp*

have *c* = (*1* − *?k1*)∗*a* + *?k1* ∗ *b*
using *twoCisTypeI*(*1*) *sum-sqs-eq*
by (*auto simp add*: *divide-simps algebra-simps*)
then show *?thesis*
using *twoCisTypeI k1-in-bounds* **by** *fastforce*
qed
have *y-witness*: ∃ *k2*. *d* = (*1* − *k2*)∗(*g2 c*) + *k2* ∗ (*g1 c*) ∧ *0* ≤ *k2* ∧ *k2* ≤ *1*
proof (*cases g1 c* − *g2 c* = *0*)
case *True*
with *d-bounds* **show** *?thesis*
by *force*
next
case *False*
let *?k2* = (*d* − *g2 c*)/(*g1 c* − *g2 c*)
have *k2-in-bounds*: *0* ≤ *?k2* ∧ *?k2* ≤ *1* **using** *twoCisTypeI*(*2*) *c-bounds d-bounds False* **by** *simp*
have *d* = (*1* − *?k2*) ∗ *g2 c* + *?k2* ∗ *g1 c*
using *False* **apply** (*simp add*: *divide-simps algebra-simps*)
using *sum-sqs-eq* **by** *fastforce*
then show *?thesis* **using** *k2-in-bounds* **by** *fastforce*
qed
show ∃ *x*∈*unit-cube*. (*c*, *d*) =
(*case x of* (*x*, *y*) ⇒ (*a* − *x* ∗ *a* + *x* ∗ *b*, *g2* (*a* − *x* ∗ *a* + *x* ∗ *b*) − *y* ∗ *g2* (*a* − *x* ∗ *a* + *x* ∗ *b*) + *y* ∗ *g1* (*a* − *x* ∗ *a* + *x* ∗ *b*)))
using *x-witness y-witness* **by** (*force simp add*: *left-diff-distrib*)
qed
have {*y*. ∃ *x*∈*unit-cube*. *y* = *twoC x*} = {(*x*, *y*). *a* ≤ *x* ∧ *x* ≤ *b* ∧ *g2 x* ≤ *y* ∧ *y* ≤ *g1 x*}
apply (*auto simp add*: *twoCisTypeI left-diff-distrib ex*)
using *less-eq-real-def twoCisTypeI*(*1*) **apply** *auto*[*1*]
apply (*smt affine-ineq twoCisTypeI*)
apply (*smt affine-ineq atLeastAtMost-iff mult-left-mono twoCisTypeI*)+
done
then show *?thesis*
unfolding *cubeImage-def image-def* **using** *twoCisTypeI* **by** *auto*
qed

lemma *typeI-cube-explicit-spec*:
assumes *typeI-twoCube twoC*
shows ∃ *a b g1 g2*. *a* < *b* ∧ (∀ *x* ∈ {*a* .. *b*}. *g2 x* ≤ *g1 x*) ∧
cubeImage twoC = {(*x*,*y*). *x* ∈ {*a*..*b*} ∧ *y* ∈ {*g2 x* .. *g1 x*}}
∧ *twoC* = (λ(*x*, *y*). ((*1* − *x*) ∗ *a* + *x* ∗ *b*, (*1* − *y*) ∗ *g2* ((*1* − *x*) ∗ *a* + *x* ∗ *b*) + *y* ∗ *g1* ((*1* − *x*) ∗ *a* + *x* ∗ *b*)))
∧ *g1 piecewise-C1-differentiable-on* {*a* .. *b*} ∧ *g2 piecewise-C1-differentiable-on* {*a* .. *b*}
∧ (λ*x*. *twoC*(*x*, *0*)) = (λ*x*. (*a* + (*b* − *a*) ∗ *x*, *g2* (*a* + (*b* − *a*) ∗ *x*)))
∧ (λ*y*. *twoC*(*1*, *y*)) = (λ*x*. (*b*, *g2 b* + *x* $*_R$ (*g1 b* − *g2 b*)))
∧ (λ*x*. *twoC*(*x*, *1*)) = (λ*x*. (*a* + (*b* − *a*) ∗ *x*, *g1* (*a* + (*b* − *a*) ∗

$x)))$

$\wedge (\lambda y.\ twoC(0,\ y)) = (\lambda x.\ (a,\ g2\ a + x *_R (g1\ a - g2\ a)))$

proof −

let *?bottom-edge* = $(\lambda x.\ twoC(x,\ 0))$
let *?right-edge* = $(\lambda y.\ twoC(1,\ y))$
let *?top-edge* = $(\lambda x.\ twoC(x,\ 1))$
let *?left-edge* = $(\lambda y.\ twoC(0,\ y))$
obtain *a b g1 g2* **where**
twoCisTypeI: $a < b$
$(\forall x \in cbox\ a\ b.\ g2\ x \le g1\ x)$
cubeImage twoC = $\{(x,y).\ x \in cbox\ a\ b \wedge y \in cbox\ (g2\ x)\ (g1\ x)\}$
twoC = $(\lambda(x,\ y).\ ((1 - x) * a + x * b,\ (1 - y) * g2\ ((1 - x) * a + x * b) + y * g1\ ((1 - x) * a + x * b)))$
g1 piecewise-C1-differentiable-on $\{a\ ..\ b\}$
g2 piecewise-C1-differentiable-on $\{a\ ..\ b\}$
using *assms* **and** *typeI-twoCubeImg*[*oftwoC*] **by** *auto*
have *bottom-edge-explicit*: *?bottom-edge* = $(\lambda x.\ (a + (b - a) * x,\ g2\ (a + (b - a) * x)))$
by (*simp add*: *twoCisTypeI*(*4*) *algebra-simps*)
have *right-edge-explicit*: *?right-edge* = $(\lambda x.\ (b,\ g2\ b + x *_R (g1\ b - g2\ b)))$
by (*simp add*: *twoCisTypeI*(*4*) *algebra-simps*)
have *top-edge-explicit*: *?top-edge* = $(\lambda x.\ (a + (b - a) * x,\ g1\ (a + (b - a) * x)))$
by (*simp add*: *twoCisTypeI*(*4*) *algebra-simps*)
have *left-edge-explicit*: *?left-edge* = $(\lambda x.\ (a,\ g2\ a + x *_R (g1\ a - g2\ a)))$
by (*simp add*: *twoCisTypeI*(*4*) *algebra-simps*)
show *?thesis*
using *bottom-edge-explicit right-edge-explicit top-edge-explicit left-edge-explicit twoCisTypeI*
by *auto*

qed

lemma *typeI-twoCube-smooth-edges*:
assumes *typeI-twoCube twoC*
$(k,\gamma) \in$ *boundary twoC*
shows γ *piecewise-C1-differentiable-on* $\{0..1\}$

proof −

let *?bottom-edge* = $(\lambda x.\ twoC(x,\ 0))$
let *?right-edge* = $(\lambda y.\ twoC(1,\ y))$
let *?top-edge* = $(\lambda x.\ twoC(x,\ 1))$
let *?left-edge* = $(\lambda y.\ twoC(0,\ y))$
obtain *a b g1 g2* **where**
twoCisTypeI: $a < b$
$(\forall x \in cbox\ a\ b.\ g2\ x \le g1\ x)$
cubeImage twoC = $\{(x,y).\ x \in cbox\ a\ b \wedge y \in cbox\ (g2\ x)\ (g1\ x)\}$
twoC = $(\lambda(x,\ y).\ ((1 - x) * a + x * b,\ (1 - y) * g2\ ((1 - x) * a + x * b) + y * g1\ ((1 - x) * a + x * b)))$
g1 piecewise-C1-differentiable-on $\{a\ ..\ b\}$
g2 piecewise-C1-differentiable-on $\{a\ ..\ b\}$

$(\lambda x.\ twoC(x,\ 0)) = (\lambda x.\ (a + (b - a) * x,\ g2\ (a + (b - a) * x)))$
$(\lambda y.\ twoC(1,\ y)) = (\lambda x.\ (b,\ g2\ b + x *_R\ (g1\ b - g2\ b)))$
$(\lambda x.\ twoC(x,\ 1)) = (\lambda x.\ (a + (b - a) * x,\ g1\ (a + (b - a) * x)))$
$(\lambda y.\ twoC(0,\ y)) = (\lambda x.\ (a,\ g2\ a + x *_R\ (g1\ a - g2\ a)))$
using *assms* **and** *typeI-cube-explicit-spec*[*oftwoC*]
by *auto*
have *bottom-edge-smooth*: $(\lambda x.\ twoC\ (x,\ 0))$ *piecewise-C1-differentiable-on* $\{0..1\}$
proof −
have $\forall x.\ (\lambda x.\ (a + (b - a) * x))-`\ \{x\} = \{(x - a)/(b - a)\}$
using *twoCisTypeI*(*1*)
by(*auto simp add*: *Set.vimage-def*)
then have *finite-vimg*: $\bigwedge x.\ finite(\{0..1\} \cap (\lambda x.\ (a + (b - a) * x))-`\ \{x\})$ **by** *auto*
have *scale-shif-smth*: $(\lambda x.\ (a + (b - a) * x))$ *C1-differentiable-on* $\{0..1\}$ **using** *scale-shift-smooth* **by** *auto*
then have *scale-shif-pw-smth*: $(\lambda x.\ (a + (b - a) * x))$ *piecewise-C1-differentiable-on* $\{0..1\}$ **using** *C1-differentiable-imp-piecewise* **by** *blast*
have *g2-smooth*: *g2 piecewise-C1-differentiable-on* $(\lambda x.\ a + (b - a) * x)\ `\ \{0..1\}$ **using** *add-scale-img*[*OF twoCisTypeI*(*1*)] *twoCisTypeI*(*6*) **by** *auto*
have $(\lambda x.\ g2\ (a + (b - a) * x))$ *piecewise-C1-differentiable-on* $\{0..1\}$
using *piecewise-C1-differentiable-compose*[*OF scale-shif-pw-smth g2-smooth finite-vimg*]
by (*auto simp add*: *o-def*)
then have $(\lambda x{::}real.\ (a + (b - a) * x,\ g2\ (a + (b - a) * x)))$ *piecewise-C1-differentiable-on* $\{0..1\}$
using *all-components-smooth-one-pw-smooth-is-pw-smooth*[**where** $f = (\lambda x{::}real.\ (a + (b - a) * x,\ g2\ (a + (b - a) * x)))$]
apply (*simp only*: *real-pair-basis*)
by *fastforce*
then show *?thesis* **using** *twoCisTypeI*(*7*) **by** *auto*
qed
have *top-edge-smooth*: *?top-edge piecewise-C1-differentiable-on* $\{0..1\}$
proof −
have $\forall x.\ (\lambda x.\ (a + (b - a) * x))-`\ \{x\} = \{(x - a)/(b - a)\}$
using *twoCisTypeI*(*1*)
by(*auto simp add*: *Set.vimage-def*)
then have *finite-vimg*: $\bigwedge x.\ finite(\{0..1\} \cap (\lambda x.\ (a + (b - a) * x))-`\ \{x\})$ **by** *auto*
have *scale-shif-smth*: $(\lambda x.\ (a + (b - a) * x))$ *C1-differentiable-on* $\{0..1\}$ **using** *scale-shift-smooth* **by** *auto*
then have *scale-shif-pw-smth*: $(\lambda x.\ (a + (b - a) * x))$ *piecewise-C1-differentiable-on* $\{0..1\}$ **using** *C1-differentiable-imp-piecewise* **by** *blast*
have *g1-smooth*: *g1 piecewise-C1-differentiable-on* $(\lambda x.\ a + (b - a) * x)\ `\ \{0..1\}$ **using** *add-scale-img*[*OF twoCisTypeI*(*1*)] *twoCisTypeI*(*5*) **by** *auto*
have $(\lambda x.\ g1\ (a + (b - a) * x))$ *piecewise-C1-differentiable-on* $\{0..1\}$
using *piecewise-C1-differentiable-compose*[*OF scale-shif-pw-smth g1-smooth finite-vimg*]
by (*auto simp add*: *o-def*)
then have $(\lambda x.\ (a + (b - a) * x,\ g1\ (a + (b - a) * x)))$ *piecewise-C1-differentiable-on*

{*0..1*}
using *all-components-smooth-one-pw-smooth-is-pw-smooth*[**where** $f = (\lambda x.$ $(a + (b - a) * x,\ g1\ (a + (b - a) * x)))$]
apply (*simp only*: *real-pair-basis*)
by *fastforce*
then show *?thesis* **using** *twoCisTypeI*(*9*) **by** *auto*
qed
have *right-edge-smooth*: *?right-edge piecewise-C1-differentiable-on* {*0..1*}
proof −
have $(\lambda x.\ (g2\ b + x *_R (g1\ b - g2\ b)))$ *C1-differentiable-on* {*0..1*}
using *scale-shift-smooth C1-differentiable-imp-piecewise* **by** *auto*
then have $(\lambda x.\ (g2\ b + x *_R (g1\ b - g2\ b)))$ *piecewise-C1-differentiable-on* {*0..1*}
using *C1-differentiable-imp-piecewise* **by** *fastforce*
then have $(\lambda x.\ (b,\ g2\ b + x *_R (g1\ b - g2\ b)))$ *piecewise-C1-differentiable-on* {*0..1*}
using *pair-prod-smooth-pw-smooth* **by** *auto*
then show *?thesis*
using *twoCisTypeI*(*8*) **by** *auto*
qed
have *left-edge-smooth*: *?left-edge piecewise-C1-differentiable-on* {*0..1*}
proof −
have $(\lambda x.\ (g2\ a + x *_R (g1\ a - g2\ a)))$ *C1-differentiable-on* {*0..1*}
using *scale-shift-smooth* **by** *auto*
then have $(\lambda x.\ (g2\ a + x *_R (g1\ a - g2\ a)))$ *piecewise-C1-differentiable-on* {*0..1*}
using *C1-differentiable-imp-piecewise* **by** *fastforce*
then have $(\lambda x.\ (a,\ g2\ a + x *_R (g1\ a - g2\ a)))$ *piecewise-C1-differentiable-on* {*0..1*}
using *pair-prod-smooth-pw-smooth* **by** *auto*
then show *?thesis*
using *twoCisTypeI*(*10*) **by** *auto*
qed
have γ = *?bottom-edge* ∨ γ = *?right-edge* ∨ γ = *?top-edge* ∨ γ = *?left-edge*
using *assms* **by** (*auto simp add*: *boundary-def horizontal-boundary-def vertical-boundary-def*)
then show *?thesis*
using *left-edge-smooth right-edge-smooth top-edge-smooth bottom-edge-smooth* **by** *auto*
qed

lemma *two-chain-integral-eq-integral-divisable*:
assumes *f-integrable*: ∀ *twoCube* ∈ *twoChain*. *F integrable-on cubeImage twoCube* **and**
gen-division: *gen-division s* (*cubeImage* ‘ *twoChain*) **and**
valid-two-chain: *valid-two-chain twoChain*
shows *integral s F* = *two-chain-integral twoChain F*
proof −
show *integral s F* = *two-chain-integral twoChain F*

proof (*simp add*: *two-chain-integral-def*)
have *partial-deriv-integrable*:
$\forall$ *twoCube* $\in$ *twoChain*. ((*F*) *has-integral* (*integral* (*cubeImage twoCube*) (*F*))) (*cubeImage twoCube*)
using *f-integrable* **by** *auto*
then have *partial-deriv-integrable*:
$\bigwedge$*twoCubeImg*. *twoCubeImg* $\in$ *cubeImage* ‘ *twoChain* $\Longrightarrow$ (*F has-integral* (*integral* (*twoCubeImg*) *F*)) (*twoCubeImg*)
using *Henstock-Kurzweil-Integration.integrable-neg* **by** *force*
have *finite-images*: *finite* (*cubeImage* ‘ *twoChain*)
using *gen-division gen-division-def* **by** *auto*
have *negligible-images*: *pairwise* ($\lambda S\ S'$. *negligible* ($S \cap S'$)) (*cubeImage* ‘ *twoChain*)
using *gen-division* **by** (*auto simp add*: *gen-division-def pairwise-def*)
have *inj*: *inj-on cubeImage twoChain*
using *valid-two-chain* **by** (*simp add*: *inj-on-def valid-two-chain-def*)
have *integral s F* = ($\sum$ *twoCubeImg*$\in$*cubeImage* ‘ *twoChain*. *integral twoCubeImg F*)
using *has-integral-Union*[*OF finite-images partial-deriv-integrable negligible-images*] *gen-division*
by (*auto simp add*: *gen-division-def*)
also have ... = ($\sum$ *C*$\in$*twoChain*. *integral* (*cubeImage C*) *F*)
using *sum.reindex inj* **by** *auto*
finally show *integral s F* = ($\sum$ *C*$\in$*twoChain*. *integral* (*cubeImage C*) *F*) **.**
qed
qed

definition *only-vertical-division* **where**
only-vertical-division one-chain two-chain $\equiv$
$\exists\,\mathcal{V}\ \mathcal{H}$. *finite* $\mathcal{H}$ $\wedge$ *finite* $\mathcal{V}$ $\wedge$
($\forall$ (k,γ) $\in$ $\mathcal{V}$.
($\exists$ (k',γ') $\in$ *two-chain-vertical-boundary two-chain*.
($\exists$ *a* $\in$ {*0..1*}. $\exists$ *b* $\in$ {*0..1*}. $a \le b$ $\wedge$ *subpath a b* $\gamma' = \gamma$))) $\wedge$
(*common-sudiv-exists* (*two-chain-horizontal-boundary two-chain*) $\mathcal{H}$
$\vee$ *common-reparam-exists* $\mathcal{H}$ (*two-chain-horizontal-boundary two-chain*)) $\wedge$
boundary-chain $\mathcal{H}$ $\wedge$ *one-chain* = $\mathcal{V} \cup \mathcal{H}$ $\wedge$
($\forall$ (k,γ)$\in\mathcal{H}$. *valid-path* γ)

abbreviation *valid-typeI-division s twoChain*
$\equiv$ ($\forall$ *twoCube* $\in$ *twoChain*. *typeI-twoCube twoCube*) $\wedge$
gen-division s (*cubeImage* ‘ *twoChain*) $\wedge$ *valid-two-chain twoChain*

lemma *field-cont-on-typeI-region-cont-on-edges*:
assumes *typeI-twoC*: *typeI-twoCube twoC*
and *field-cont*: *continuous-on* (*cubeImage twoC*) *F*
and *member-of-boundary*: (k,γ) $\in$ *boundary twoC*
shows *continuous-on* (γ ‘ {*0* .. *1*}) *F*

proof –
obtain *a b g1 g2* **where**
twoCisTypeI: $a < b$
$(\forall x \in cbox\ a\ b.\ g2\ x \leq g1\ x)$
$cubeImage\ twoC = \{(x,y).\ x \in cbox\ a\ b \wedge y \in cbox\ (g2\ x)\ (g1\ x)\}$
$twoC = (\lambda(x,\ y).\ ((1 - x) * a + x * b,\ (1 - y) * g2\ ((1 - x) * a + x * b) + y * g1\ ((1 - x) * a + x * b)))$
g1 piecewise-C1-differentiable-on $\{a\ ..\ b\}$
g2 piecewise-C1-differentiable-on $\{a\ ..\ b\}$
$(\lambda x.\ twoC(x,\ 0)) = (\lambda x.\ (a + (b - a) * x,\ g2\ (a + (b - a) * x)))$
$(\lambda y.\ twoC(1,\ y)) = (\lambda x.\ (b,\ g2\ b + x *_R (g1\ b - g2\ b)))$
$(\lambda x.\ twoC(x,\ 1)) = (\lambda x.\ (a + (b - a) * x,\ g1\ (a + (b - a) * x)))$
$(\lambda y.\ twoC(0,\ y)) = (\lambda x.\ (a,\ g2\ a + x *_R (g1\ a - g2\ a)))$
using *typeI-twoC* **and** *typeI-cube-explicit-spec*[*oftwoC*]
by *auto*
let *?bottom-edge* $= (\lambda x.\ twoC(x,\ 0))$
let *?right-edge* $= (\lambda y.\ twoC(1,\ y))$
let *?top-edge* $= (\lambda x.\ twoC(x,\ 1))$
let *?left-edge* $= (\lambda y.\ twoC(0,\ y))$
let *?Dg1* $= \{p.\ \exists x.\ x \in cbox\ a\ b \wedge p = (x,\ g1(x))\}$
have *line-is-pair-img*: *?Dg1* $= (\lambda x.\ (x,\ g1(x)))$ ‘ $(cbox\ a\ b)$
using *image-def* **by** *auto*
have *field-cont-on-top-edge-image*: *continuous-on ?Dg1 F*
by (*rule continuous-on-subset* [*OF field-cont*]) (*auto simp*: *twoCisTypeI*(*2*) *twoCisTypeI*(*3*))
have *top-edge-is-compos-of-scal-and-g1*:
$(\lambda x.\ twoC(x,\ 1)) = (\lambda x.\ (x,\ g1(x))) \circ (\lambda x.\ a + (b - a) * x)$
using *twoCisTypeI* **by** *auto*
have *Dg1-is-bot-edge-pathimg*: *path-image* $(\lambda x.\ twoC(x,\ 1))$ = *?Dg1*
using *line-is-pair-img* **and** *top-edge-is-compos-of-scal-and-g1 image-comp path-image-def add-scale-img* **and** *twoCisTypeI*(*1*)
by (*metis* (*no-types*, *lifting*) *cbox-interval*)
then have *cont-on-top*: *continuous-on* (*path-image ?top-edge*) *F*
using *field-cont-on-top-edge-image* **by** *auto*
let *?Dg2* $= \{p.\ \exists x.\ x \in cbox\ a\ b \wedge p = (x,\ g2(x))\}$
have *line-is-pair-img*: *?Dg2* $= (\lambda x.\ (x,\ g2(x)))$ ‘ $(cbox\ a\ b)$
using *image-def* **by** *auto*
have *field-cont-on-bot-edge-image*: *continuous-on ?Dg2 F*
apply (*rule continuous-on-subset* [*OF field-cont*])
using *twoCisTypeI*(*2*) *twoCisTypeI*(*3*) **by** *auto*
have *bot-edge-is-compos-of-scal-and-g2*: $(\lambda x.\ twoC(x,\ 0)) = (\lambda x.\ (x,\ g2(x))) \circ (\lambda x.\ a + (b - a) * x)$
using *twoCisTypeI* **by** *auto*
have *Dg2-is-bot-edge-pathimg*:
path-image $(\lambda x.\ twoC(x,\ 0))$ = *?Dg2*
using *line-is-pair-img* **and** *bot-edge-is-compos-of-scal-and-g2 image-comp path-image-def add-scale-img* **and** *twoCisTypeI*(*1*)
by (*metis* (*no-types*, *lifting*) *cbox-interval*)
then have *cont-on-bot*: *continuous-on* (*path-image ?bottom-edge*) *F*

using *field-cont-on-bot-edge-image* **by** *auto*
let *?D-left-edge* = {*p*. ∃ *y*. *y* ∈ *cbox* (*g2 a*) (*g1 a*) ∧ *p* = (*a*, *y*)}
have *field-cont-on-left-edge-image*: *continuous-on ?D-left-edge F*
apply (*rule continuous-on-subset* [*OF field-cont*])
using *twoCisTypeI*(*1*) *twoCisTypeI*(*3*) **by** *auto*
have *g2 a* ≤ *g1 a* **using** *twoCisTypeI*(*1*) *twoCisTypeI*(*2*) **by** *auto*
then have (λ*x*. *g2 a* + (*g1 a* − *g2 a*) ∗ *x*) ‘ {(*0*::*real*)..*1*} = {*g2 a* .. *g1 a*}
using *add-scale-img′*[*of g2 a g1 a*] **by** *blast*
then have *left-eq*: *?D-left-edge* = *?left-edge* ‘ {*0*..*1*}
unfolding *twoCisTypeI*(*10*)
by(*auto simp add*: *subset-iff image-def set-eq-iff Semiring-Normalization.comm-semiring-1-class.semiring-no*
then have *cont-on-left*: *continuous-on* (*path-image ?left-edge*) *F*
using *field-cont-on-left-edge-image path-image-def*
by (*metis left-eq field-cont-on-left-edge-image path-image-def*)
let *?D-right-edge* = {*p*. ∃ *y*. *y* ∈ *cbox* (*g2 b*) (*g1 b*) ∧ *p* = (*b*, *y*)}
have *field-cont-on-left-edge-image*: *continuous-on ?D-right-edge F*
apply (*rule continuous-on-subset* [*OF field-cont*])
using *twoCisTypeI*(*1*) *twoCisTypeI*(*3*) **by** *auto*
have *g2 b* ≤ *g1 b* **using** *twoCisTypeI*(*1*) *twoCisTypeI*(*2*) **by** *auto*
then have (λ*x*. *g2 b* + (*g1 b* − *g2 b*) ∗ *x*) ‘ {(*0*::*real*)..*1*} = {*g2 b* .. *g1 b*}
using *add-scale-img′*[*of g2 b g1 b*] **by** *blast*
then have *right-eq*: *?D-right-edge* = *?right-edge* ‘ {*0*..*1*}
unfolding *twoCisTypeI*(*8*)
by(*auto simp add*: *subset-iff image-def set-eq-iff Semiring-Normalization.comm-semiring-1-class.semiring-no*
then have *cont-on-right*:
continuous-on (*path-image ?right-edge*) *F*
using *field-cont-on-left-edge-image path-image-def*
by (*metis right-eq field-cont-on-left-edge-image path-image-def*)
have *all-edge-cases*:
(γ = *?bottom-edge* ∨ γ = *?right-edge* ∨ γ = *?top-edge* ∨ γ = *?left-edge*)
using *assms* **by** (*auto simp add*: *boundary-def horizontal-boundary-def vertical-boundary-def*)
show *?thesis*
apply (*simp add*: *path-image-def*[*symmetric*])
using *cont-on-top cont-on-bot cont-on-right cont-on-left all-edge-cases*
by *blast*
qed

lemma *typeII-cube-explicit-spec*:
assumes *typeII-twoCube twoC*
shows ∃ *a b g1 g2*. *a* < *b* ∧ (∀ *x* ∈ {*a* .. *b*}. *g2 x* ≤ *g1 x*) ∧
cubeImage twoC = {(*y*, *x*). *x* ∈ {*a*..*b*} ∧ *y* ∈ {*g2 x* .. *g1 x*}}
∧ *twoC* = (λ(*y*, *x*). ((*1* − *y*) ∗ *g2* ((*1* − *x*) ∗ *a* + *x* ∗ *b*) + *y* ∗ *g1* ((*1* − *x*) ∗ *a* + *x* ∗ *b*), (*1* − *x*) ∗ *a* + *x* ∗ *b*))
∧ *g1 piecewise-C1-differentiable-on* {*a* .. *b*} ∧ *g2 piecewise-C1-differentiable-on* {*a* .. *b*}
∧ (λ*x*. *twoC*(*0*, *x*)) = (λ*x*. (*g2* (*a* + (*b* − *a*) ∗ *x*), *a* + (*b* − *a*) ∗ *x*))
∧ (λ*y*. *twoC*(*y*, *1*)) = (λ*x*. (*g2 b* + *x* $*_R$ (*g1 b* − *g2 b*), *b*))
∧ (λ*x*. *twoC*(*1*, *x*)) = (λ*x*. (*g1* (*a* + (*b* − *a*) ∗ *x*), *a* + (*b* − *a*) ∗ *x*))

∧ (λy. twoC(y, 0)) = (λx. (g2 a + x $*_R$ (g1 a − g2 a), a))
proof −
let ?bottom-edge = (λx. twoC(0, x))
let ?right-edge = (λy. twoC(y, 1))
let ?top-edge = (λx. twoC(1, x))
let ?left-edge = (λy. twoC(y, 0))
obtain a b g1 g2 **where**
twoCisTypeII: a < b
(∀ x ∈ cbox a b. g2 x ≤ g1 x)
cubeImage twoC = {(y, x). x ∈ cbox a b ∧ y ∈ cbox (g2 x) (g1 x)}
twoC = (λ(y, x). ((1 − y) * g2 ((1 − x) * a + x * b) + y * g1 ((1 − x) * a + x * b), (1 − x) * a + x * b))
g1 piecewise-C1-differentiable-on {a .. b}
g2 piecewise-C1-differentiable-on {a .. b}
using assms **and** typeII-twoCubeImg[oftwoC] **by** auto
have bottom-edge-explicit: ?bottom-edge = (λx. (g2 (a + (b − a) * x), a + (b − a) * x))
by (simp add: twoCisTypeII(4) algebra-simps)
have right-edge-explicit: ?right-edge = (λx. (g2 b + x $*_R$ (g1 b − g2 b), b))
by (simp add: twoCisTypeII(4) algebra-simps)
have top-edge-explicit: ?top-edge = (λx. (g1 (a + (b − a) * x), a + (b − a) * x))
by (simp add: twoCisTypeII(4) algebra-simps)
have left-edge-explicit: ?left-edge = (λx. (g2 a + x $*_R$ (g1 a − g2 a), a))
by (simp add: twoCisTypeII(4) algebra-simps)
show ?thesis
using bottom-edge-explicit right-edge-explicit top-edge-explicit left-edge-explicit twoCisTypeII
by auto
qed

lemma typeII-twoCube-smooth-edges:
assumes typeII-twoCube twoC (k,γ) ∈ boundary twoC
shows γ piecewise-C1-differentiable-on {0..1}
proof −
let ?bottom-edge = (λx. twoC(0, x))
let ?right-edge = (λy. twoC(y, 1))
let ?top-edge = (λx. twoC(1, x))
let ?left-edge = (λy. twoC(y, 0))
obtain a b g1 g2 **where**
twoCisTypeII: a < b
(∀ x ∈ cbox a b. g2 x ≤ g1 x)
cubeImage twoC = {(y, x). x ∈ cbox a b ∧ y ∈ cbox (g2 x) (g1 x)}
twoC = (λ(y, x). ((1 − y) * g2 ((1 − x) * a + x * b) + y * g1 ((1 − x) * a + x * b), (1 − x) * a + x * b))
g1 piecewise-C1-differentiable-on {a .. b}
g2 piecewise-C1-differentiable-on {a .. b}
(λx. twoC(0, x)) = (λx. (g2 (a + (b − a) * x), a + (b − a) * x))
(λy. twoC(y, 1)) = (λx. (g2 b + x $*_R$ (g1 b − g2 b), b))

$(\lambda x.\ twoC(1,\ x)) = (\lambda x.\ (g1\ \ (a + (b - a) * x),\ a + (b - a) * x))$
$(\lambda y.\ twoC(y,\ 0)) = (\lambda x.\ (g2\ a + x *_R\ (g1\ a - g2\ a),\ a))$
using *assms* **and** *typeII-cube-explicit-spec*[*oftwoC*]
by *auto*
have *bottom-edge-smooth*: *?bottom-edge piecewise-C1-differentiable-on* {*0..1*}
proof −
have $\forall x.\ (\lambda x.\ (a + (b - a) * x)) -` \{x\} = \{(x - a)/(b - a)\}$
using *twoCisTypeII*(*1*) **by** *auto*
then have *finite-vimg*: $\bigwedge x.$ *finite*($\{0..1\} \cap (\lambda x.\ (a + (b - a) * x))-` \{x\}$) **by** *auto*
have *scale-shif-pw-smth*: $(\lambda x.\ (a + (b - a) * x))$ *piecewise-C1-differentiable-on* {*0..1*}
using *scale-shift-smooth C1-differentiable-imp-piecewise* **by** *blast*
have *g2-smooth*: *g2 piecewise-C1-differentiable-on* $(\lambda x.\ a + (b - a) * x)$ ` {*0..1*} **using** *add-scale-img*[*OF twoCisTypeII*(*1*)] *twoCisTypeII*(*6*) **by** *auto*
have $(\lambda x.\ g2\ \ (a + (b - a) * x))$ *piecewise-C1-differentiable-on* {*0..1*}
using *piecewise-C1-differentiable-compose*[*OF scale-shif-pw-smth g2-smooth finite-vimg*]
by (*auto simp add*: *o-def*)
then have $(\lambda x::real.\ (g2\ \ (a + (b - a) * x),\ a + (b - a) * x))$ *piecewise-C1-differentiable-on* {*0..1*}
using *all-components-smooth-one-pw-smooth-is-pw-smooth*[**where** $f = (\lambda x::real.\ (g2\ \ (a + (b - a) * x),\ a + (b - a) * x))$]
by (*fastforce simp add*: *real-pair-basis*)
then show *?thesis* **using** *twoCisTypeII*(*7*) **by** *auto*
qed
have *top-edge-smooth*: *?top-edge piecewise-C1-differentiable-on* {*0..1*}
proof −
have $\forall x.\ (\lambda x.\ (a + (b - a) * x))-` \{x\} = \{(x - a)/(b - a)\}$
using *twoCisTypeII*(*1*) **by** *auto*
then have *finite-vimg*: $\bigwedge x.$ *finite*($\{0..1\} \cap (\lambda x.\ (a + (b - a) * x))-` \{x\}$) **by** *auto*
have *scale-shif-pw-smth*: $(\lambda x.\ (a + (b - a) * x))$ *piecewise-C1-differentiable-on* {*0..1*}
using *scale-shift-smooth C1-differentiable-imp-piecewise* **by** *blast*
have *g1-smooth*: *g1 piecewise-C1-differentiable-on* $(\lambda x.\ a + (b - a) * x)$ ` {*0..1*} **using** *add-scale-img*[*OF twoCisTypeII*(*1*)] *twoCisTypeII*(*5*) **by** *auto*
have $(\lambda x.\ g1\ \ (a + (b - a) * x))$ *piecewise-C1-differentiable-on* {*0..1*}
using *piecewise-C1-differentiable-compose*[*OF scale-shif-pw-smth g1-smooth finite-vimg*]
by (*auto simp add*: *o-def*)
then have $(\lambda x::real.\ (g1\ \ (a + (b - a) * x),\ a + (b - a) * x))$ *piecewise-C1-differentiable-on* {*0..1*}
using *all-components-smooth-one-pw-smooth-is-pw-smooth*[**where** $f = (\lambda x::real.\ (g1\ \ (a + (b - a) * x),\ a + (b - a) * x))$]
by (*fastforce simp add*: *real-pair-basis*)
then show *?thesis* **using** *twoCisTypeII*(*9*) **by** *auto*
qed
have *right-edge-smooth*: *?right-edge piecewise-C1-differentiable-on* {*0..1*}

proof −
have (λ*x*. (*g2 b* + *x* $*_R$ (*g1 b* − *g2 b*))) *piecewise-C1-differentiable-on* {*0..1*}
by (*simp add*: *C1-differentiable-imp-piecewise*)
then have (λ*x*. (*g2 b* + *x* $*_R$ (*g1 b* − *g2 b*), *b*)) *piecewise-C1-differentiable-on* {*0..1*}
using *all-components-smooth-one-pw-smooth-is-pw-smooth*[*of* (*1*,*0*) (λ*x*. (*g2 b* + *x* $*_R$ (*g1 b* − *g2 b*), *b*))]
by (*auto simp add*: *real-pair-basis*)
then show *?thesis*
using *twoCisTypeII*(*8*) **by** *auto*
qed
have *left-edge-smooth*: *?left-edge piecewise-C1-differentiable-on* {*0..1*}
proof −
have *0*: (λ*x*. (*g2 a* + *x* $*_R$ (*g1 a* − *g2 a*))) *C1-differentiable-on* {*0..1*}
using *C1-differentiable-imp-piecewise* **by** *fastforce*
have (λ*x*. (*g2 a* + *x* $*_R$ (*g1 a* − *g2 a*), *a*)) *piecewise-C1-differentiable-on* {*0..1*}
using *C1-differentiable-imp-piecewise*[*OF C1-differentiable-on-pair*[*OF 0 C1-differentiable-on-const*[*of a* {*0..1*}]]]
by *force*
then show *?thesis*
using *twoCisTypeII*(*10*) **by** *auto*
qed
have γ = *?bottom-edge* ∨ γ = *?right-edge* ∨ γ = *?top-edge* ∨ γ = *?left-edge*
using *assms* **by** (*auto simp add*: *boundary-def horizontal-boundary-def vertical-boundary-def*)
then show *?thesis*
using *left-edge-smooth right-edge-smooth top-edge-smooth bottom-edge-smooth* **by** *auto*
qed

lemma *field-cont-on-typeII-region-cont-on-edges*:
assumes *typeII-twoC*:
typeII-twoCube twoC **and**
field-cont:
continuous-on (*cubeImage twoC*) *F* **and**
member-of-boundary:
(*k*,γ) ∈ *boundary twoC*
shows *continuous-on* (γ ' {*0* .. *1*}) *F*
proof −
obtain *a b g1 g2* **where**
twoCisTypeII: *a* < *b*
(∀ *x* ∈ *cbox a b*. *g2 x* ≤ *g1 x*)
cubeImage twoC = {(*y*, *x*). *x* ∈ *cbox a b* ∧ *y* ∈ *cbox* (*g2 x*) (*g1 x*)}
twoC = (λ(*y*, *x*). ((*1* − *y*) ∗ *g2* ((*1* − *x*) ∗ *a* + *x* ∗ *b*) + *y* ∗ *g1* ((*1* − *x*) ∗ *a* + *x* ∗ *b*), (*1* − *x*) ∗ *a* + *x* ∗ *b*))
g1 piecewise-C1-differentiable-on {*a* .. *b*}
g2 piecewise-C1-differentiable-on {*a* .. *b*}
(λ*x*. *twoC*(*0*, *x*)) = (λ*x*. (*g2* (*a* + (*b* − *a*) ∗ *x*), *a* + (*b* − *a*) ∗ *x*))
(λ*y*. *twoC*(*y*, *1*)) = (λ*x*. (*g2 b* + *x* $*_R$ (*g1 b* − *g2 b*), *b*))

(λx. twoC(1, x)) = (λx. (g1 (a + (b − a) ∗ x), a + (b − a) ∗ x))
(λy. twoC(y, 0)) = (λx. (g2 a + x $*_R$ (g1 a − g2 a), a))
using *typeII-twoC* **and** *typeII-cube-explicit-spec*[*oftwoC*]
by *auto*
let *?bottom-edge* = (λx. twoC(0, x))
let *?right-edge* = (λy. twoC(y, 1))
let *?top-edge* = (λx. twoC(1, x))
let *?left-edge* = (λy. twoC(y, 0))
let *?Dg1* = {p. ∃x. x ∈ cbox a b ∧ p = (g1(x), x)}
have *line-is-pair-img*: *?Dg1* = (λx. (g1(x), x)) ‘ (cbox a b)
using *image-def* **by** *auto*
have *field-cont-on-top-edge-image*: *continuous-on ?Dg1 F*
by (*rule continuous-on-subset* [*OF field-cont*]) (*auto simp*: *twoCisTypeII*(2) *twoCisTypeII*(3))
have *top-edge-is-compos-of-scal-and-g1*:
(λx. twoC(1, x)) = (λx. (g1(x), x)) ∘ (λx. a + (b − a) ∗ x)
using *twoCisTypeII* **by** *auto*
have *Dg1-is-bot-edge-pathimg*:
path-image (λx. twoC(1, x)) = *?Dg1*
using *line-is-pair-img* **and** *top-edge-is-compos-of-scal-and-g1 image-comp path-image-def add-scale-img* **and** *twoCisTypeII*(1)
by (*metis* (*no-types*, *lifting*) *cbox-interval*)
then have *cont-on-top*: *continuous-on* (*path-image ?top-edge*) *F*
using *field-cont-on-top-edge-image* **by** *auto*
let *?Dg2* = {p. ∃x. x ∈ cbox a b ∧ p = (g2(x), x)}
have *line-is-pair-img*: *?Dg2* = (λx. (g2(x), x)) ‘ (cbox a b)
using *image-def* **by** *auto*
have *field-cont-on-bot-edge-image*: *continuous-on ?Dg2 F*
by (*rule continuous-on-subset* [*OF field-cont*]) (*auto simp add*: *twoCisTypeII*(2) *twoCisTypeII*(3))
have *bot-edge-is-compos-of-scal-and-g2*:
(λx. twoC(0, x)) = (λx. (g2(x), x)) ∘ (λx. a + (b − a) ∗ x)
using *twoCisTypeII* **by** *auto*
have *Dg2-is-bot-edge-pathimg*: *path-image* (λx. twoC(0, x)) = *?Dg2*
unfolding *path-image-def*
using *line-is-pair-img* **and** *bot-edge-is-compos-of-scal-and-g2 image-comp add-scale-img* [*OF* ‹a < b›]
by (*metis* (*no-types*, *lifting*) *box-real*(2))
then have *cont-on-bot*: *continuous-on* (*path-image ?bottom-edge*) *F*
using *field-cont-on-bot-edge-image*
by *auto*
let *?D-left-edge* = {p. ∃y. y ∈ cbox (g2 a) (g1 a) ∧ p = (y, a)}
have *field-cont-on-left-edge-image*: *continuous-on ?D-left-edge F*
apply (*rule continuous-on-subset* [*OF field-cont*])
using *twoCisTypeII*(1) *twoCisTypeII*(3) **by** *auto*
have g2 a ≤ g1 a **using** *twoCisTypeII*(1) *twoCisTypeII*(2) **by** *auto*
then have (λx. g2 a + x ∗ (g1 a − g2 a)) ‘ {(0::real)..1} = {g2 a .. g1 a}
using *add-scale-img′*[*of g2 a g1 a*] **by** (*auto simp add*: *ac-simps*)
with ‹g2 a ≤ g1 a› **have** *left-eq*: *?D-left-edge* = *?left-edge* ‘ {0..1}

by (*simp only*: *twoCisTypeII*(*10*)) *auto*
then have *cont-on-left*: *continuous-on* (*path-image ?left-edge*) *F*
using *field-cont-on-left-edge-image path-image-def*
by (*metis left-eq field-cont-on-left-edge-image path-image-def*)
let *?D-right-edge* = {*p*. ∃ *y*. *y* ∈ *cbox* (*g2 b*) (*g1 b*) ∧ *p* = (*y*, *b*)}
have *field-cont-on-left-edge-image*: *continuous-on ?D-right-edge F*
apply (*rule continuous-on-subset* [*OF field-cont*])
using *twoCisTypeII*(*1*) *twoCisTypeII*(*3*) **by** *auto*
have *g2 b* ≤ *g1 b* **using** *twoCisTypeII*(*1*) *twoCisTypeII*(*2*) **by** *auto*
then have (λ*x*. *g2 b* + *x* ∗ (*g1 b* − *g2 b*)) ' {(*0*::*real*)..*1*} = {*g2 b* .. *g1 b*}
using *add-scale-img*′[*of g2 b g1 b*] **by** (*auto simp add*: *ac-simps*)
with ‹*g2 b* ≤ *g1 b*› **have** *right-eq*: *?D-right-edge* = *?right-edge* ' {*0*..*1*}
by (*simp only*: *twoCisTypeII*(*8*)) *auto*
then have *cont-on-right*:
continuous-on (*path-image ?right-edge*) *F*
using *field-cont-on-left-edge-image path-image-def*
by (*metis right-eq field-cont-on-left-edge-image path-image-def*)
have *all-edge-cases*:
(γ = *?bottom-edge* ∨ γ = *?right-edge* ∨ γ = *?top-edge* ∨ γ = *?left-edge*)
using *assms* **unfolding** *boundary-def horizontal-boundary-def vertical-boundary-def*
by *blast*
show *?thesis*
apply (*simp add*: *path-image-def*[*symmetric*])
using *cont-on-top cont-on-bot cont-on-right cont-on-left all-edge-cases*
by *blast*
qed

lemma *two-cube-boundary-is-boundary*: *boundary-chain* (*boundary C*)
by (*auto simp add*: *boundary-chain-def boundary-def horizontal-boundary-def vertical-boundary-def*)

lemma *common-boundary-subdiv-exists-refl*:
assumes ∀ (*k*,γ)∈*boundary twoC*. *valid-path* γ
shows *common-boundary-sudivision-exists* (*boundary twoC*) (*boundary twoC*)
using *assms chain-subdiv-chain-refl common-boundary-sudivision-exists-def two-cube-boundary-is-boundary*
by *blast*

lemma *common-boundary-subdiv-exists-refl*′:
assumes ∀ (*k*,γ)∈*C*. *valid-path* γ
boundary-chain (*C*::(*int* × (*real* ⇒ *real* × *real*)) *set*)
shows *common-boundary-sudivision-exists* (*C*) (*C*)
using *assms chain-subdiv-chain-refl common-boundary-sudivision-exists-def* **by** *blast*

lemma *gen-common-boundary-subdiv-exists-refl-twochain-boundary*:
assumes ∀ (*k*,γ)∈*C*. *valid-path* γ
boundary-chain (*C*::(*int* × (*real* ⇒ *real* × *real*)) *set*)
shows *common-sudiv-exists* (*C*) (*C*)
using *assms chain-subdiv-chain-refl common-boundary-sudivision-exists-def com-*

mon-subdiv-imp-gen-common-subdiv **by** *blast*

lemma *two-chain-boundary-is-boundary-chain*:
shows *boundary-chain* (*two-chain-boundary twoChain*)
by (*simp add*: *boundary-chain-def two-chain-boundary-def boundary-def horizontal-boundary-def vertical-boundary-def*)

lemma *typeI-edges-are-valid-paths*:
assumes *typeI-twoCube twoC* $(k,\gamma) \in$ *boundary twoC*
shows *valid-path* γ
using *typeI-twoCube-smooth-edges*[*OF assms*] *C1-differentiable-imp-piecewise*
by (*auto simp*: *valid-path-def*)

lemma *typeII-edges-are-valid-paths*:
assumes *typeII-twoCube twoC* $(k,\gamma) \in$ *boundary twoC*
shows *valid-path* γ
using *typeII-twoCube-smooth-edges*[*OF assms*] *C1-differentiable-imp-piecewise*
by (*auto simp*: *valid-path-def*)

lemma *finite-two-chain-vertical-boundary*:
assumes *finite two-chain*
shows *finite* (*two-chain-vertical-boundary two-chain*)
using *assms* **by** (*simp add*: *two-chain-vertical-boundary-def vertical-boundary-def*)

lemma *finite-two-chain-horizontal-boundary*:
assumes *finite two-chain*
shows *finite* (*two-chain-horizontal-boundary two-chain*)
using *assms* **by** (*simp add*: *two-chain-horizontal-boundary-def horizontal-boundary-def*)

locale *R2* =
fixes *i j*
assumes *i-is-x-axis*: $i = (1::real,0::real)$ **and**
j-is-y-axis: $j = (0::real,\ 1::real)$
begin

lemma *analytically-valid-y*:
assumes *analytically-valid s F i*
shows (λx. *integral UNIV* (λy. (*partial-vector-derivative F i*) $(y, x) * ($*indicator* $s\ (y, x)))) \in$ *borel-measurable lborel*
proof −
have $\{(1::real,\ 0::real),\ (0,\ 1)\} - \{(1,\ 0)\} = \{(0::real,1::real)\}$
by *force*
with *assms* **show** *?thesis*
using *assms* **by**(*simp add*: *real-pair-basis analytically-valid-def i-is-x-axis j-is-y-axis*)
qed

lemma *analytically-valid-x*:
assumes *analytically-valid s F j*

shows (λx. *integral UNIV* (λy. ((*partial-vector-derivative F j*) (x, y)) $*$ (*indicator s* (x, y)))) $\in$ *borel-measurable lborel*
proof –
have $\{(1::real,\ 0::real),\ (0,\ 1)\} - \{(0,\ 1)\} = \{(1::real,\ 0::real)\}$
by *force*
with *assms* **show** *?thesis*
by(*simp add*: *real-pair-basis analytically-valid-def i-is-x-axis j-is-y-axis*)
qed

lemma *Greens-thm-type-I*:
fixes F:: ((*real*$*$*real*) $\Rightarrow$ (*real* $*$ *real*)) **and**
gamma1 gamma2 gamma3 gamma4 :: (*real* $\Rightarrow$ (*real* $*$ *real*)) **and**
a:: *real* **and** *b*:: *real* **and**
g1:: (*real* $\Rightarrow$ *real*) **and** *g2*:: (*real* $\Rightarrow$ *real*)
assumes *Dy-def*: *Dy-pair* = $\{(x::real,y)\ .\ x \in cbox\ a\ b \land y \in cbox\ (g2\ x)\ (g1\ x)\}$ **and**
gamma1-def: *gamma1* = (λx. ($a + (b - a) * x$, $g2(a + (b - a) * x)$)) **and**
gamma1-smooth: *gamma1 piecewise-C1-differentiable-on* $\{0..1\}$ **and**
gamma2-def: *gamma2* = (λx. (b, $g2(b) + x *_R (g1(b) - g2(b))$)) **and**
gamma3-def: *gamma3* = (λx. ($a + (b - a) * x$, $g1(a + (b - a) * x)$)) **and**
gamma3-smooth: *gamma3 piecewise-C1-differentiable-on* $\{0..1\}$ **and**
gamma4-def: *gamma4* = (λx. (a, $g2(a) + x *_R (g1(a) - g2(a))$)) **and**
F-i-analytically-valid: *analytically-valid Dy-pair* (λp. $F(p) \cdot i$) *j* **and**
g2-leq-g1: $\forall x \in cbox\ a\ b.\ (g2\ x) \leq (g1\ x)$ **and**
a-lt-b: $a < b$
shows (*line-integral F* $\{i\}$ *gamma1*) +
(*line-integral F* $\{i\}$ *gamma2*) –
(*line-integral F* $\{i\}$ *gamma3*) –
(*line-integral F* $\{i\}$ *gamma4*)
= (*integral Dy-pair* (λa. – (*partial-vector-derivative* (λp. $F(p) \cdot i$) *j a*)))
line-integral-exists F $\{i\}$ *gamma4*
line-integral-exists F $\{i\}$ *gamma3*
line-integral-exists F $\{i\}$ *gamma2*
line-integral-exists F $\{i\}$ *gamma1*
proof –
let *?F-b'* = *partial-vector-derivative* (λa. (F a) $\cdot$ i) *j*
have *F-first-is-continuous*: *continuous-on Dy-pair* (λa. $F(a) \cdot i$)
using *F-i-analytically-valid*
by (*auto simp add*: *analytically-valid-def*)
let *?f* = (λx. *if* $x \in$ (*Dy-pair*) *then* (*partial-vector-derivative* (λa. (F a) $\cdot$ i) *j*) *x else 0*)
have *f-lesbegue-integrable*: *integrable lborel ?f*
using *F-i-analytically-valid*
by (*auto simp add*: *analytically-valid-def indic-ident*)
have *partially-vec-diff*: $\forall a \in$ *Dy-pair*. *partially-vector-differentiable* (λa. (F a) $\cdot$ i) *j a*
using *F-i-analytically-valid*
by (*auto simp add*: *analytically-valid-def indicator-def*)

have *x-axis-integral-measurable*: (λ*x*. *integral UNIV* (λ*y*. *?f*(*x*, *y*))) ∈ *borel-measurable lborel*
proof −
have (λ*p*. (*?F-b'* *p*) ∗ *indicator* (*Dy-pair*) *p*) = (λ*x*. *if* *x* ∈ (*Dy-pair*) *then* (*?F-b'*) *x* *else* *0*)
using *indic-ident*[*of ?F-b'*] **by** *auto*
then have ⋀*x y*. *?F-b'* (*x*,*y*) ∗ *indicator* (*Dy-pair*) (*x*,*y*) = (λ*x*. *if* *x* ∈ (*Dy-pair*) *then* (*?F-b'*) *x* *else* *0*) (*x*,*y*)
by *auto*
then show *?thesis*
using *analytically-valid-x*[*OF F-i-analytically-valid*]
by (*auto simp add*: *indicator-def*)
qed
have *F-partially-differentiable*: ∀ *a*∈*Dy-pair*. *has-partial-vector-derivative* (λ*x*. (*F* *x*) · *i*) *j* (*?F-b'* *a*) *a*
using *partial-vector-derivative-works partially-vec-diff*
by *fastforce*
have *g1-g2-continuous*: *continuous-on* (*cbox a b*) *g1*
continuous-on (*cbox a b*) *g2*
proof −
have *shift-scale-cont*: *continuous-on* {*a*..*b*} (λ*x*. (*x* − *a*)∗(*1*/(*b*−*a*)))
by (*intro continuous-intros*)
have *shift-scale-inv*: (λ*x*. *a* + (*b* − *a*) ∗ *x*) ∘ (λ*x*. (*x* − *a*)∗(*1*/(*b*−*a*))) = *id*
using *a-lt-b* **by** (*auto simp add*: *o-def*)
have *img-shift-scale*: (λ*x*. (*x* − *a*)∗(*1*/(*b*−*a*)))‘{*a*..*b*} = {*0*..*1*}
using *a-lt-b* **apply** (*auto simp*: *divide-simps image-iff*)
apply (*rule-tac x*=*x* ∗ (*b* − *a*) + *a* **in** *bexI*)
using *le-diff-eq* **by** *fastforce*+
have *gamma1-y-component*: (λ*x*. *g2*(*a* + (*b* − *a*) ∗ *x*)) = *g2* ∘ (λ*x*.(*a* + (*b* − *a*) ∗ *x*))
by *auto*
have *continuous-on* {*0*..*1*} (λ*x*. *g2*(*a* + (*b* − *a*) ∗ *x*))
using *continuous-on-inner*[*OF piecewise-C1-differentiable-on-imp-continuous-on*[*OF gamma1-smooth*], *of* (λ*x*. *j*), *OF continuous-on-const*]
by (*simp add*: *gamma1-def j-is-y-axis*)
then have *continuous-on* {*a*..*b*} ((λ*x*. *g2*(*a* + (*b* − *a*) ∗ *x*)) ∘ (λ*x*. (*x* − *a*)∗(*1*/(*b*−*a*))))
using *img-shift-scale continuous-on-compose shift-scale-cont*
by *force*
then have *continuous-on* {*a*..*b*} (*g2* ∘ (λ*x*.(*a* + (*b* − *a*) ∗ *x*)) ∘ (λ*x*. (*x* − *a*)∗(*1*/(*b*−*a*))))
using *gamma1-y-component* **by** *auto*
then show *continuous-on* (*cbox a b*) *g2*
using *a-lt-b* **by** (*simp add*: *shift-scale-inv*)
have *gamma3-y-component*: (λ*x*. *g1*(*a* + (*b* − *a*) ∗ *x*)) = *g1* ∘ (λ*x*.(*a* + (*b* − *a*) ∗ *x*))
by *auto*
have *continuous-on* {*0*..*1*} (λ*x*. *g1*(*a* + (*b* − *a*) ∗ *x*))
using *continuous-on-inner*[*OF piecewise-C1-differentiable-on-imp-continuous-on*[*OF*

gamma3-smooth], *of* ($\lambda x.\ j$), *OF continuous-on-const*]
by (*simp add*: *gamma3-def j-is-y-axis*)
then have *continuous-on* $\{a..b\}$ $((\lambda x.\ g1(a + (b - a) * x)) \circ (\lambda x.\ (x - a)*(1/(b-a))))$
using *img-shift-scale continuous-on-compose shift-scale-cont*
by *force*
then have *continuous-on* $\{a..b\}$ $(g1 \circ (\lambda x.(a + (b - a) * x)) \circ (\lambda x.\ (x - a)*(1/(b-a))))$
using *gamma3-y-component* **by** *auto*
then show *continuous-on* (*cbox a b*) *g1*
using *a-lt-b* **by** (*simp add*: *shift-scale-inv*)
qed
have *g2-scale-j-contin*: *continuous-on* (*cbox a b*) ($\lambda x.$ (*0*, *g2 x*))
by (*intro continuous-intros g1-g2-continuous*)
let *?Dg2* = $\{p.\ \exists x.\ x \in$ *cbox a b* $\wedge\ p = (x,\ g2(x))\}$
have *line-is-pair-img*: *?Dg2* = $(\lambda x.\ (x,\ g2(x)))$ ‘ (*cbox a b*)
using *image-def* **by** *auto*
have *g2-path-continuous*: *continuous-on* (*cbox a b*) $(\lambda x.\ (x,\ g2(x)))$
by (*intro continuous-intros g1-g2-continuous*)
have *field-cont-on-gamma1-image*: *continuous-on ?Dg2* $(\lambda a.\ F(a) \cdot i)$
apply (*rule continuous-on-subset* [*OF F-first-is-continuous*])
by (*auto simp add*: *Dy-def g2-leq-g1*)
have *gamma1-is-compos-of-scal-and-g2*:
gamma1 = $(\lambda x.\ (x,\ g2(x))) \circ (\lambda x.\ a + (b - a) * x)$
using *gamma1-def* **by** *auto*
have *add-scale-img*:
$(\lambda x.\ a + (b - a) * x)$ ‘ $\{0\ ..\ 1\} = \{a\ ..\ b\}$ **using** *add-scale-img* **and** *a-lt-b* **by** *auto*
then have *Dg2-is-gamma1-pathimg*: *path-image gamma1* = *?Dg2*
by (*metis* (*no-types*, *lifting*) *box-real*(*2*) *gamma1-is-compos-of-scal-and-g2 image-comp line-is-pair-img path-image-def*)
have *Base-vecs*: $i \in$ *Basis* $j \in$ *Basis* $i \neq j$
using *real-pair-basis* **and** *i-is-x-axis* **and** *j-is-y-axis* **by** *auto*
have *gamma1-as-euclid-space-fun*: *gamma1* = $(\lambda x.\ (a + (b - a) * x) *_R i + (0,\ g2\ (a + (b - a) * x)))$
using *i-is-x-axis gamma1-def* **by** *auto*
have *0*: *line-integral F* $\{i\}$ *gamma1* = *integral* (*cbox a b*) $(\lambda x.\ F(x,\ g2(x)) \cdot i\ )$
line-integral-exists F $\{i\}$ *gamma1*
using *line-integral-on-pair-path-strong* [*OF norm-Basis*[*OF Base-vecs*(*1*)] - *gamma1-as-euclid-space-fun*, *of F*]
gamma1-def gamma1-smooth g2-scale-j-contin a-lt-b add-scale-img
Dg2-is-gamma1-pathimg **and** *field-cont-on-gamma1-image*
by (*auto simp*: *pathstart-def pathfinish-def i-is-x-axis*)
then show (*line-integral-exists F* $\{i\}$ *gamma1*) **by** *metis*
have *gamma2-x-const*: $\forall x.$ *gamma2* $x \cdot i = b$
by (*simp add*: *i-is-x-axis gamma2-def*)
have *1*: (*line-integral F* $\{i\}$ *gamma2*) = *0* (*line-integral-exists F* $\{i\}$ *gamma2*)
using *line-integral-on-pair-straight-path*[*OF gamma2-x-const*] *straight-path-diffrentiable-x gamma2-def*

by (*auto simp add*: *mult.commute*)
then show (*line-integral-exists F* {*i*} *gamma2*) **by** *metis*
have *continuous-on* (*cbox a b*) (λ*x*. *F*(*x*, *g2*(*x*)) · *i*)
using *line-is-pair-img* **and** *g2-path-continuous* **and** *field-cont-on-gamma1-image Topological-Spaces.continuous-on-compose i-is-x-axis j-is-y-axis*
by *auto*
then have *6*: (λ*x*. *F*(*x*, *g2*(*x*)) · *i*) *integrable-on* (*cbox a b*)
using *integrable-continuous* [*of a b* (λ*x*. *F*(*x*, *g2*(*x*)) · *i*)] **by** *auto*
have *g1-scale-j-contin*: *continuous-on* (*cbox a b*) (λ*x*. (*0*, *g1 x*))
by (*intro continuous-intros g1-g2-continuous*)
let *?Dg1* = {*p*. ∃ *x*. *x* ∈ *cbox a b* ∧ *p* = (*x*, *g1*(*x*))}
have *line-is-pair-img*: *?Dg1* = (λ*x*. (*x*, *g1*(*x*))) ‘ (*cbox a b*)
using *image-def* **by** *auto*
have *g1-path-continuous*: *continuous-on* (*cbox a b*) (λ*x*. (*x*, *g1*(*x*)))
by (*intro continuous-intros g1-g2-continuous*)
have *field-cont-on-gamma3-image*: *continuous-on ?Dg1* (λ*a*. *F*(*a*) · *i*)
apply (*rule continuous-on-subset* [*OF F-first-is-continuous*])
by (*auto simp add*: *Dy-def g2-leq-g1*)
have *gamma3-is-compos-of-scal-and-g1*:
gamma3 = (λ*x*. (*x*, *g1*(*x*))) ∘ (λ*x*. *a* + (*b* − *a*) ∗ *x*)
using *gamma3-def* **by** *auto*
then have *Dg1-is-gamma3-pathimg*: *path-image gamma3* = *?Dg1*
by (*metis* (*no-types*, *lifting*) *box-real*(*2*) *image-comp line-is-pair-img local.add-scale-img path-image-def*)
have *Base-vecs*: *i* ∈ *Basis j* ∈ *Basis i* ≠ *j*
using *real-pair-basis* **and** *i-is-x-axis* **and** *j-is-y-axis* **by** *auto*
have *gamma3-as-euclid-space-fun*: *gamma3* = (λ*x*. (*a* + (*b* − *a*) ∗ *x*) $*_R$ *i* + (*0*, *g1* (*a* + (*b* − *a*) ∗ *x*)))
using *i-is-x-axis gamma3-def* **by** *auto*
have *2*: *line-integral F* {*i*} *gamma3* = *integral* (*cbox a b*) (λ*x*. *F*(*x*, *g1*(*x*)) · *i*)
line-integral-exists F {*i*} *gamma3*
using *line-integral-on-pair-path-strong* [*OF norm-Basis*[*OF Base-vecs*(*1*)] - *gamma3-as-euclid-space-fun*, *of F*]
gamma3-def **and** *gamma3-smooth* **and** *g1-scale-j-contin* **and** *a-lt-b* **and** *add-scale-img*
Dg1-is-gamma3-pathimg **and** *field-cont-on-gamma3-image*
by (*auto simp*: *pathstart-def pathfinish-def i-is-x-axis*)
then show (*line-integral-exists F* {*i*} *gamma3*) **by** *metis*
have *gamma4-x-const*: ∀ *x*. *gamma4 x* · *i* = *a*
using *gamma4-def*
by (*auto simp add*: *real-inner-class.inner-add-left inner-not-same-Basis i-is-x-axis*)
have *3*: (*line-integral F* {*i*} *gamma4*) = *0* (*line-integral-exists F* {*i*} *gamma4*)
using *line-integral-on-pair-straight-path*[*OF gamma4-x-const*] *straight-path-diffrentiable-x gamma4-def*
by (*auto simp add*: *mult.commute*)
then show (*line-integral-exists F* {*i*} *gamma4*)
by *metis*
have *continuous-on* (*cbox a b*) (λ*x*. *F*(*x*, *g1*(*x*)) · *i*)
using *line-is-pair-img* **and** *g1-path-continuous* **and** *field-cont-on-gamma3-image*

continuous-on-compose i-is-x-axis j-is-y-axis
by *auto*
then have *7*: (λ*x*. *F*(*x*, *g1*(*x*)) · *i*) *integrable-on* (*cbox a b*)
using *integrable-continuous* [*of a b* (λ*x*. *F*(*x*, *g1*(*x*)) · *i*)]
by *auto*
have *partial-deriv-one-d-integrable*:
((λ*y*. *?F-b′*(*xc*, *y*)) *has-integral*
F(*xc*,*g1*(*xc*)) · *i* − *F*(*xc*, *g2*(*xc*)) · *i*) (*cbox* (*g2 xc*) (*g1 xc*))
if *xc* ∈ *cbox a b* **for** *xc*
proof −
have {(*xc′*, *y*). *y* ∈ *cbox* (*g2 xc*) (*g1 xc*) ∧ *xc′* = *xc*} ⊆ *Dy-pair*
using *that* **by** (*auto simp add*: *Dy-def*)
then show ((λ*y*. *?F-b′* (*xc*, *y*)) *has-integral F*(*xc*, *g1 xc*) · *i* − *F*(*xc*, *g2 xc*) · *i*) (*cbox* (*g2 xc*) (*g1 xc*))
using *that* **and** *Base-vecs* **and** *F-partially-differentiable*
and *Dy-def* [*symmetric*] **and** *g2-leq-g1*
and *fundamental-theorem-of-calculus-partial-vector*
[*of g2 xc g1 xc j i xc* $*_R$ *i Dy-pair F ?F-b′*]
by (*auto simp add*: *Groups.ab-semigroup-add-class.add.commute i-is-x-axis j-is-y-axis*)
qed
have *partial-deriv-integrable*: (*?F-b′*) *integrable-on Dy-pair*
by (*simp add*: *F-i-analytically-valid analytically-valid-imp-part-deriv-integrable-on*)
have *4*: *integral Dy-pair ?F-b′*
= *integral* (*cbox a b*) (λ*x*. *integral* (*cbox* (*g2 x*) (*g1 x*)) (λ*y*. *?F-b′* (*x*, *y*)))
proof −
have *x-axis-gauge-integrable*:
⋀*x*. (λ*y*. *?f*(*x*,*y*)) *integrable-on UNIV*
proof −
fix *x*::*real*
have ∀ *x*. *x* ∉ *cbox a b* ⟶ (λ*y*. *?f* (*x*, *y*)) = (λ*y*. *0*)
by (*auto simp add*: *Dy-def*)
then have *f-integrable-x-not-in-range*:
∀ *x*. *x* ∉ *cbox a b* ⟶ (λ*y*. *?f* (*x*, *y*)) *integrable-on UNIV*
by (*simp add*: *integrable-0*)
let *?F-b′-oneD* = (λ*x*. (λ*y*. *if y* ∈ (*cbox* (*g2 x*) (*g1 x*)) *then ?F-b′* (*x*,*y*) *else 0*))
have *f-value-x-in-range*: ∀ *x* ∈ *cbox a b*. *?F-b′-oneD x* = (λ*y*. *?f*(*x*,*y*))
by (*auto simp add*: *Dy-def*)
have ∀ *x* ∈ *cbox a b*. *?F-b′-oneD x integrable-on UNIV*
using *has-integral-integrable integrable-restrict-UNIV partial-deriv-one-d-integrable* **by** *blast*
then have *f-integrable-x-in-range*:
∀ *x*. *x* ∈ *cbox a b* ⟶ (λ*y*. *?f* (*x*, *y*)) *integrable-on UNIV*
using *f-value-x-in-range* **by** *auto*
show (λ*y*. *?f* (*x*, *y*)) *integrable-on UNIV*
using *f-integrable-x-not-in-range* **and** *f-integrable-x-in-range* **by** *auto*
qed
have *arg*: (λ*a*. *if a* ∈ *Dy-pair then partial-vector-derivative* (λ*a*. *F a* · *i*) *j a else*

0) =
(λ*x*. *if x* ∈ *Dy-pair then if x* ∈ *Dy-pair then partial-vector-derivative* (λ*a*. *F a* · *i*) *j x else 0 else 0*)
by *auto*
have *arg2*: *Dy-pair* = {(*x*, *y*). (∀ *i*∈*Basis*. *a* · *i* ≤ *x* · *i* ∧ *x* · *i* ≤ *b* · *i*) ∧ (∀ *i*∈*Basis*. *g2 x* · *i* ≤ *y* · *i* ∧ *y* · *i* ≤ *g1 x* · *i*)}
using *Dy-def* **by** *auto*
have *arg3*: ⋀ *x*. *x* ∈ *Dy-pair* ⟹ (λ*x*. *if x* ∈ *Dy-pair then partial-vector-derivative* (λ*a*. *F a* · *i*) *j x else 0*) *x*
= (λ*x*. *partial-vector-derivative* (λ*a*. *F a* · *i*) *j x*) *x*
by *auto*
have *arg4*: ⋀*x*. *x* ∈ (*cbox a b*) ⟹
(λ*x*. *integral* (*cbox* (*g2 x*) (*g1 x*)) (λ*y*. *if* (*x*, *y*) ∈ *Dy-pair then partial-vector-derivative* (λ*a*. *F a* · *i*) *j* (*x*, *y*) *else 0*)) *x* =
(λ*x*. *integral* (*cbox* (*g2 x*) (*g1 x*)) (λ*y*. *partial-vector-derivative* (λ*a*. *F a* · *i*) *j* (*x*, *y*))) *x*
apply (*simp add*: *Dy-def*)
by (*smt Henstock-Kurzweil-Integration.integral-cong atLeastAtMost-iff*)
show *?thesis*
using *gauge-integral-Fubini-curve-bounded-region-x*
[*OF f-lesbegue-integrable x-axis-gauge-integrable x-axis-integral-measurable arg arg2*]
Henstock-Kurzweil-Integration.integral-cong[*OF arg3*, *of Dy-pair* (λ*x*. *x*)]
Henstock-Kurzweil-Integration.integral-cong[*OF arg4*, *of cbox a b* (λ*x*. *x*)]
by *auto*
qed
have *5*: (*integral Dy-pair* (λ*a*. (*?F-b′ a*))
= *integral* (*cbox a b*) (λ*x*. *F*(*x*, *g1*(*x*)) · *i* − *F*(*x*, *g2*(*x*)) · *i*))
using *4 Henstock-Kurzweil-Integration.integral-cong partial-deriv-one-d-integrable integrable-def*
by (*smt integral-unique*)
show (*line-integral F* {*i*} *gamma1*) + (*line-integral F* {*i*} *gamma2*) −
(*line-integral F* {*i*} *gamma3*) − (*line-integral F* {*i*} *gamma4*)
= (*integral Dy-pair* (λ*a*. − (*?F-b′ a*)))
using *0*(*1*) *1*(*1*) *2*(*1*) *3*(*1*) *5 6 7*
by (*simp add*: *Henstock-Kurzweil-Integration.integral-diff*)
qed

theorem *Greens-thm-type-II*:
fixes *F*:: ((*real*∗*real*) ⇒ (*real* ∗ *real*)) **and**
gamma4 gamma3 gamma2 gamma1 :: (*real* ⇒ (*real* ∗ *real*)) **and**
a:: *real* **and** *b*:: *real* **and**
g1:: (*real* ⇒ *real*) **and** *g2*:: (*real* ⇒ *real*)
assumes *Dx-def*: *Dx-pair* = {(*x*::*real*,*y*) . *y* ∈ *cbox a b* ∧ *x* ∈ *cbox* (*g2 y*) (*g1 y*)} **and**
gamma4-def: *gamma4* = (λ*x*. (*g2*(*a* + (*b* − *a*) ∗ *x*), *a* + (*b* − *a*) ∗ *x*)) **and**
gamma4-smooth: *gamma4 piecewise-C1-differentiable-on* {*0*..*1*} **and**
gamma3-def: *gamma3* = (λ*x*. (*g2*(*b*) + *x* $*_R$ (*g1*(*b*) − *g2*(*b*)), *b*)) **and**
gamma2-def: *gamma2* = (λ*x*. (*g1*(*a* + (*b* − *a*) ∗ *x*), *a* + (*b* − *a*) ∗ *x*)) **and**

gamma2-smooth: *gamma2 piecewise-C1-differentiable-on* $\{0..1\}$ **and**
gamma1-def: *gamma1* = (λx. (*g2*(*a*) + *x* $*_R$ (*g1*(*a*) − *g2*(*a*)), *a*)) **and**
F-j-analytically-valid: *analytically-valid Dx-pair* (λp. *F*(*p*) $\cdot$ *j*) *i* **and**
g2-leq-g1: $\forall$ *x* $\in$ *cbox a b*. (*g2 x*) $\leq$ (*g1 x*) **and**
a-lt-b: *a* < *b*
shows −(*line-integral F* {*j*} *gamma4*) −
(*line-integral F* {*j*} *gamma3*) +
(*line-integral F* {*j*} *gamma2*) +
(*line-integral F* {*j*} *gamma1*)
= (*integral Dx-pair* (λa. (*partial-vector-derivative* (λa. (*F a*) $\cdot$ *j*) *i* *a*)))
line-integral-exists F {*j*} *gamma4*
line-integral-exists F {*j*} *gamma3*
line-integral-exists F {*j*} *gamma2*
line-integral-exists F {*j*} *gamma1*
proof −
let *?F-a'* = *partial-vector-derivative* (λa. (*F a*) $\cdot$ *j*) *i*
have *F-first-is-continuous*: *continuous-on Dx-pair* (λa. *F*(*a*) $\cdot$ *j*)
using *F-j-analytically-valid*
by (*auto simp add*: *analytically-valid-def*)
let *?f* = (λx. *if x* $\in$ (*Dx-pair*) *then* (*partial-vector-derivative* (λa. (*F a*) $\cdot$ *j*) *i*) *x else 0*)
have *f-lesbegue-integrable*: *integrable lborel ?f*
using *F-j-analytically-valid*
by (*auto simp add*: *analytically-valid-def indic-ident*)
have *partially-vec-diff*: $\forall$ *a* $\in$ *Dx-pair*. *partially-vector-differentiable* (λa. (*F a*) $\cdot$ *j*) *i a*
using *F-j-analytically-valid*
by (*auto simp add*: *analytically-valid-def indicator-def*)
have $\bigwedge$*x y*. *?F-a'* (*x*,*y*) $*$ *indicator* (*Dx-pair*) (*x*,*y*) = (λx. *if x* $\in$ (*Dx-pair*) *then* *?F-a' x else 0*) (*x*,*y*)
using *indic-ident*[*of ?F-a'*] **by** *auto*
then have *y-axis-integral-measurable*: (λx. *integral UNIV* (λy. *?f*(*y*, *x*))) $\in$ *borel-measurable lborel*
using *analytically-valid-y*[*OF F-j-analytically-valid*]
by (*auto simp add*: *indicator-def*)
have *F-partially-differentiable*: $\forall$ *a*$\in$*Dx-pair*. *has-partial-vector-derivative* (λx. (*F x*) $\cdot$ *j*) *i* (*?F-a' a*) *a*
using *partial-vector-derivative-works partially-vec-diff* **by** *fastforce*
have *g1-g2-continuous*: *continuous-on* (*cbox a b*) *g1 continuous-on* (*cbox a b*) *g2*
proof −
have *shift-scale-cont*: *continuous-on* {*a*..*b*} (λx. (*x* − *a*)$*$(*1*/(*b*−*a*)))
by (*intro continuous-intros*)
have *shift-scale-inv*: (λx. *a* + (*b* − *a*) $*$ *x*) $\circ$ (λx. (*x* − *a*)$*$(*1*/(*b*−*a*))) = *id*
using *a-lt-b* **by** (*auto simp add*: *o-def*)
have *img-shift-scale*:
(λx. (*x* − *a*)$*$(*1*/(*b*−*a*)))‘{*a*..*b*} = {*0*..*1*}
apply (*auto simp*: *divide-simps image-iff*)
apply (*rule-tac x*=*x* $*$ (*b* − *a*) + *a* **in** *bexI*)

using *a-lt-b* **by** (*auto simp*: *algebra-simps mult-le-cancel-left affine-ineq*)
have *continuous-on* {*0..1*} (λ*x*. *g2*(*a* + (*b* − *a*) ∗ *x*))
using *continuous-on-inner*[*OF piecewise-C1-differentiable-on-imp-continuous-on*[*OF gamma4-smooth*], *of* (λ*x*. *i*), *OF continuous-on-const*]
by (*simp add*: *gamma4-def i-is-x-axis*)
then have *continuous-on* {*a..b*} ((λ*x*. *g2*(*a* + (*b* − *a*) ∗ *x*)) ∘ (λ*x*. (*x* − *a*)∗(*1*/(*b*−*a*))))
using *img-shift-scale continuous-on-compose shift-scale-cont* **by** *force*
then show *continuous-on* (*cbox a b*) *g2*
using *a-lt-b* **by** (*simp add*: *shift-scale-inv*)
have *continuous-on* {*0..1*} (λ*x*. *g1*(*a* + (*b* − *a*) ∗ *x*))
using *continuous-on-inner*[*OF piecewise-C1-differentiable-on-imp-continuous-on*[*OF gamma2-smooth*], *of* (λ*x*. *i*), *OF continuous-on-const*]
by (*simp add*: *gamma2-def i-is-x-axis*)
then have *continuous-on* {*a..b*} ((λ*x*. *g1*(*a* + (*b* − *a*) ∗ *x*)) ∘ (λ*x*. (*x* − *a*)∗(*1*/(*b*−*a*))))
using *img-shift-scale continuous-on-compose shift-scale-cont* **by** *force*
then show *continuous-on* (*cbox a b*) *g1*
using *a-lt-b* **by** (*simp add*: *shift-scale-inv*)
qed
have *g2-scale-i-contin*: *continuous-on* (*cbox a b*) (λ*x*. (*g2 x*, *0*))
by (*intro continuous-intros g1-g2-continuous*)
let *?Dg2* = {*p*. ∃ *x*. *x* ∈ *cbox a b* ∧ *p* = (*g2*(*x*), *x*)}
have *line-is-pair-img*: *?Dg2* = (λ*x*. (*g2*(*x*), *x*)) ‘ (*cbox a b*)
using *image-def* **by** *auto*
have *g2-path-continuous*: *continuous-on* (*cbox a b*) (λ*x*. (*g2*(*x*), *x*))
by (*intro continuous-intros g1-g2-continuous*)
have *field-cont-on-gamma4-image*: *continuous-on ?Dg2* (λ*a*. *F*(*a*) · *j*)
by (*rule continuous-on-subset* [*OF F-first-is-continuous*]) (*auto simp*: *Dx-def g2-leq-g1*)
have *gamma4-is-compos-of-scal-and-g2*: *gamma4* = (λ*x*. (*g2*(*x*), *x*)) ∘ (λ*x*. *a* + (*b* − *a*) ∗ *x*)
using *gamma4-def* **by** *auto*
have *add-scale-img*:
(λ*x*. *a* + (*b* − *a*) ∗ *x*) ‘ {*0* .. *1*} = {*a* .. *b*} **using** *add-scale-img* **and** *a-lt-b* **by** *auto*
then have *Dg2-is-gamma4-pathimg*: *path-image gamma4* = *?Dg2*
using *line-is-pair-img* **and** *gamma4-is-compos-of-scal-and-g2 image-comp path-image-def*
by (*metis* (*no-types*, *lifting*) *cbox-interval*)
have *Base-vecs*: *i* ∈ *Basis j* ∈ *Basis i* ≠ *j*
using *real-pair-basis* **and** *i-is-x-axis* **and** *j-is-y-axis* **by** *auto*
have *gamma4-as-euclid-space-fun*: *gamma4* = (λ*x*. (*a* + (*b* − *a*) ∗ *x*) $*_R$ *j* + (*g2* (*a* + (*b* − *a*) ∗ *x*), *0*))
using *j-is-y-axis gamma4-def*
by *auto*
have *0*: (*line-integral F* {*j*} *gamma4*) = *integral* (*cbox a b*) (λ*x*. *F*(*g2*(*x*), *x*) · *j*)
line-integral-exists F {*j*} *gamma4*
using *line-integral-on-pair-path-strong* [*OF norm-Basis*[*OF Base-vecs*(*2*)] - *gamma4-as-euclid-space-fun*]

gamma4-def gamma4-smooth g2-scale-i-contin a-lt-b add-scale-img
Dg2-is-gamma4-pathimg **and** *field-cont-on-gamma4-image*
by (*auto simp*: *pathstart-def pathfinish-def j-is-y-axis*)
then show *line-integral-exists F* {*j*} *gamma4* **by** *metis*
have *gamma3-y-const*: ∀ *x*. *gamma3 x* · *j* = *b*
by (*simp add*: *gamma3-def j-is-y-axis*)
have *1*: (*line-integral F* {*j*} *gamma3*) = *0* (*line-integral-exists F* {*j*} *gamma3*)
using *line-integral-on-pair-straight-path*[*OF gamma3-y-const*] *straight-path-diffrentiable-y gamma3-def*
by (*auto simp add*: *mult.commute*)
then show *line-integral-exists F* {*j*} *gamma3* **by** *auto*
have *continuous-on* (*cbox a b*) (λ*x*. *F*(*g2*(*x*), *x*) · *j*)
by (*smt Collect-mono-iff continuous-on-compose2 continuous-on-eq field-cont-on-gamma4-image g2-path-continuous line-is-pair-img*)
then have *6*: (λ*x*. *F*(*g2*(*x*), *x*) · *j*) *integrable-on* (*cbox a b*)
using *integrable-continuous* **by** *blast*
have *g1-scale-i-contin*: *continuous-on* (*cbox a b*) (λ*x*. (*g1 x*, *0*))
by (*intro continuous-intros g1-g2-continuous*)
let *?Dg1* = {*p*. ∃ *x*. *x* ∈ *cbox a b* ∧ *p* = (*g1*(*x*), *x*)}
have *line-is-pair-img*: *?Dg1* = (λ*x*. (*g1*(*x*), *x*)) ‘ (*cbox a b*)
using *image-def* **by** *auto*
have *g1-path-continuous*: *continuous-on* (*cbox a b*) (λ*x*. (*g1*(*x*), *x*))
by (*intro continuous-intros g1-g2-continuous*)
have *field-cont-on-gamma2-image*: *continuous-on ?Dg1* (λ*a*. *F*(*a*) · *j*)
by (*rule continuous-on-subset* [*OF F-first-is-continuous*]) (*auto simp*: *Dx-def g2-leq-g1*)
have *gamma2* = (λ*x*. (*g1*(*x*), *x*)) ∘ (λ*x*. *a* + (*b* − *a*) ∗ *x*)
using *gamma2-def* **by** *auto*
then have *Dg1-is-gamma2-pathimg*: *path-image gamma2* = *?Dg1*
using *line-is-pair-img image-comp path-image-def add-scale-img*
by (*metis* (*no-types*, *lifting*) *cbox-interval*)
have *Base-vecs*: *i* ∈ *Basis j* ∈ *Basis i* ≠ *j*
using *real-pair-basis* **and** *i-is-x-axis* **and** *j-is-y-axis* **by** *auto*
have *gamma2-as-euclid-space-fun*: *gamma2* = (λ*x*. (*a* + (*b* − *a*) ∗ *x*) $*_R$ *j* + (*g1* (*a* + (*b* − *a*) ∗ *x*), *0*))
using *j-is-y-axis gamma2-def* **by** *auto*
have *2*: (*line-integral F* {*j*} *gamma2*) = *integral* (*cbox a b*) (λ*x*. *F*(*g1*(*x*), *x*) · *j*)
(*line-integral-exists F* {*j*} *gamma2*)
using *line-integral-on-pair-path-strong* [*OF norm-Basis*[*OF Base-vecs*(*2*)] - *gamma2-as-euclid-space-fun*]
gamma2-def **and** *gamma2-smooth* **and** *g1-scale-i-contin* **and** *a-lt-b* **and** *add-scale-img*
Dg1-is-gamma2-pathimg **and** *field-cont-on-gamma2-image*
by (*auto simp*: *pathstart-def pathfinish-def j-is-y-axis*)
then show *line-integral-exists F* {*j*} *gamma2* **by** *metis*
have *gamma1-y-const*: ∀ *x*. *gamma1 x* · *j* = *a*
using *gamma1-def*
by (*auto simp add*: *real-inner-class.inner-add-left*
euclidean-space-class.inner-not-same-Basis j-is-y-axis)

have *3*: (*line-integral F* {*j*} *gamma1*) = *0* (*line-integral-exists F* {*j*} *gamma1*)
using *line-integral-on-pair-straight-path*[*OF gamma1-y-const*] *straight-path-diffrentiable-y gamma1-def*
by (*auto simp add*: *mult.commute*)
then show *line-integral-exists F* {*j*} *gamma1* **by** *auto*
have *continuous-on* (*cbox a b*) (λx. *F*(*g1*(*x*), *x*) $\cdot$ *j*)
by (*smt Collect-mono-iff continuous-on-compose2 continuous-on-eq field-cont-on-gamma2-image g1-path-continuous line-is-pair-img*)
then have *7*: (λx. *F*(*g1*(*x*), *x*) $\cdot$ *j*) *integrable-on* (*cbox a b*)
using *integrable-continuous* [*of a b* (λx. *F*(*g1*(*x*), *x*) $\cdot$ *j*)]
by *auto*
have *partial-deriv-one-d-integrable*:
((λy. *?F-a'*(*y*, *xc*)) *has-integral F*(*g1*(*xc*), *xc*) $\cdot$ *j* − *F*(*g2*(*xc*), *xc*) $\cdot$ *j*) (*cbox* (*g2 xc*) (*g1 xc*))
if *xc* ∈ *cbox a b* **for** *xc*::*real*
proof −
have {(*y*, *xc'*). *y* ∈ *cbox* (*g2 xc*) (*g1 xc*) ∧ *xc'* = *xc*} ⊆ *Dx-pair*
using *that* **by** (*auto simp add*: *Dx-def*)
then show *?thesis*
using *that* **and** *Base-vecs* **and** *F-partially-differentiable*
and *Dx-def* [*symmetric*] **and** *g2-leq-g1*
and *fundamental-theorem-of-calculus-partial-vector*
[*of g2 xc g1 xc i j xc* $*_R$ *j Dx-pair F ?F-a'*]
by (*auto simp add*: *Groups.ab-semigroup-add-class.add.commute i-is-x-axis j-is-y-axis*)
qed
have *?f integrable-on UNIV*
by (*simp add*: *f-lesbegue-integrable integrable-on-lborel*)
then have *partial-deriv-integrable*: *?F-a' integrable-on Dx-pair*
using *integrable-restrict-UNIV* **by** *auto*
have *4*: *integral Dx-pair ?F-a'* = *integral* (*cbox a b*) (λx. *integral* (*cbox* (*g2 x*) (*g1 x*)) (λy. *?F-a'* (*y*, *x*)))
proof −
have *y-axis-gauge-integrable*: (λy. *?f*(*y*, *x*)) *integrable-on UNIV* **for** *x*
proof −
let *?F-a'-oneD* = (λx. (λy. *if y* ∈ (*cbox* (*g2 x*) (*g1 x*)) *then ?F-a'* (*y*, *x*) *else 0*))
have ∀ *x*. *x* ∉ *cbox a b* ⟶ (λy. *?f* (*y*, *x*)) = (λy. *0*)
by (*auto simp add*: *Dx-def*)
then have *f-integrable-x-not-in-range*:
∀ *x*. *x* ∉ *cbox a b* ⟶ (λy. *?f* (*y*, *x*)) *integrable-on UNIV*
by (*simp add*: *integrable-0*)
have ∀ *x* ∈ *cbox a b*. *?F-a'-oneD x* = (λy. *?f*(*y*, *x*))
using *g2-leq-g1* **by** (*auto simp add*: *Dx-def*)
moreover have ∀ *x* ∈ *cbox a b*. *?F-a'-oneD x integrable-on UNIV*
using *has-integral-integrable integrable-restrict-UNIV partial-deriv-one-d-integrable* **by** *blast*
ultimately have ∀ *x*. *x* ∈ *cbox a b* ⟶ (λy. *?f* (*y*, *x*)) *integrable-on UNIV*
by *auto*

then show (λy. *?f* (y, x)) *integrable-on UNIV*
using *f-integrable-x-not-in-range* **by** *auto*
qed
have *arg*: (λa. *if* $a \in$ *Dx-pair then partial-vector-derivative* (λa. $F\ a \cdot j$) $i\ a$ *else* 0) =
(λx. *if* $x \in$ *Dx-pair then if* $x \in$ *Dx-pair then partial-vector-derivative* (λa. $F\ a \cdot j$) $i\ x$ *else* 0 *else* 0)
by *auto*
have *arg2*: *Dx-pair* = $\{(y, x).\ (\forall i\in Basis.\ a \cdot i \le x \cdot i \wedge x \cdot i \le b \cdot i) \wedge (\forall i\in Basis.\ g2\ x \cdot i \le y \cdot i \wedge y \cdot i \le g1\ x \cdot i)\}$
using *Dx-def* **by** *auto*
have *arg3*: $\bigwedge x.\ x \in$ *Dx-pair* $\Longrightarrow$ (λx. *if* $x \in$ *Dx-pair then partial-vector-derivative* (λa. $F\ a \cdot j$) $i\ x$ *else* 0) x
= (λx. *partial-vector-derivative* (λa. $F\ a \cdot j$) $i\ x$) x
by *auto*
have *arg4*: $\bigwedge x.\ x \in$ (*cbox* a b) $\Longrightarrow$
(λx. *integral* (*cbox* ($g2\ x$) ($g1\ x$)) (λy. *if* $(y, x) \in$ *Dx-pair then partial-vector-derivative* (λa. $F\ a \cdot j$) $i\ (y, x)$ *else* 0)) x =
(λx. *integral* (*cbox* ($g2\ x$) ($g1\ x$)) (λy. *partial-vector-derivative* (λa. $F\ a \cdot j$) $i\ (y, x)$)) x
apply (*clarsimp simp*: *Dx-def*)
by (*smt Henstock-Kurzweil-Integration.integral-cong atLeastAtMost-iff*)
show *?thesis*
using *gauge-integral-Fubini-curve-bounded-region-y*
[*OF f-lesbegue-integrable y-axis-gauge-integrable y-axis-integral-measurable arg arg2*]
Henstock-Kurzweil-Integration.integral-cong[*OF arg3*, *of Dx-pair* (λx. x)]
Henstock-Kurzweil-Integration.integral-cong[*OF arg4*, *of cbox a b* (λx. x)]
by *auto*
qed
have ((*integral Dx-pair* (λa. (*?F-a'* a)))
= *integral* (*cbox a b*) (λx. $F(g1(x), x) \cdot j - F(g2(x), x) \cdot j$))
using *4 Henstock-Kurzweil-Integration.integral-cong partial-deriv-one-d-integrable integrable-def*
by (*smt integral-unique*)
then have *integral Dx-pair* (λa. $-$ (*?F-a'* a))
= $-$ *integral* (*cbox a b*) (λx. $F(g1(x), x) \cdot j - F(g2(x), x) \cdot j$)
using *partial-deriv-integrable* **and** *integral-neg* **by** *auto*
then have *5*: *integral Dx-pair* (λa. $-$ (*?F-a'* a))
= *integral* (*cbox a b*) (λx. ($F(g2(x), x) \cdot j - F(g1(x), x) \cdot j$))
using *6 7*
and *integral-neg* [*of* - (λx. $F(g1\ x, x) \cdot j - F(g2\ x, x) \cdot j$)] **by** *auto*
have (*line-integral F* $\{j\}$ *gamma4*) + (*line-integral F* $\{j\}$ *gamma3*) $-$
(*line-integral F* $\{j\}$ *gamma2*) $-$ (*line-integral F* $\{j\}$ *gamma1*)
= (*integral Dx-pair* (λa. $-$(*?F-a'* a)))
using *0 1 2 3 5 6 7*
Henstock-Kurzweil-Integration.integral-diff[*of* (λx. $F(g2(x), x) \cdot j$)
cbox a b (λx. $F(g1(x), x) \cdot j$)] **by** (*auto*)
then show $-$(*line-integral F* $\{j\}$ *gamma4*) $-$

(*line-integral F* {*j*} *gamma3*) +
(*line-integral F* {*j*} *gamma2*) +
(*line-integral F* {*j*} *gamma1*)
= (*integral Dx-pair* (*λa.* (*?F-a′ a*)))
by (*simp add*: ‹*integral Dx-pair* (*λa.* − *?F-a′ a*) = − *integral* (*cbox a b*) (*λx.* *F*(*g1 x*, *x*) · *j* − *F*(*g2 x*, *x*) · *j*)› ‹*integral Dx-pair ?F-a′* = *integral* (*cbox a b*) (*λx.* *F*(*g1 x*, *x*) · *j* − *F*(*g2 x*, *x*) · *j*)›)
qed

end

locale *green-typeII-cube* = *R2* +
fixes *twoC F*
assumes
two-cube: *typeII-twoCube twoC* **and**
valid-two-cube: *valid-two-cube twoC* **and**
f-analytically-valid: *analytically-valid* (*cubeImage twoC*) (*λx.* (*F x*) · *j*) *i*
begin

lemma *GreenThm-typeII-twoCube*:
shows *integral* (*cubeImage twoC*) (*λa. partial-vector-derivative* (*λx.* (*F x*) · *j*) *i* *a*) = *one-chain-line-integral F* {*j*} (*boundary twoC*)
∀ (*k*,*γ*) ∈ *boundary twoC*. *line-integral-exists F* {*j*} *γ*
proof −
let *?bottom-edge* = (*λx. twoC*(*x*, *0*))
let *?right-edge* = (*λy. twoC*(*1*, *y*))
let *?top-edge* = (*λx. twoC*(*x*, *1*))
let *?left-edge* = (*λy. twoC*(*0*, *y*))
have *line-integral-around-boundary*:
one-chain-line-integral F {*j*} (*boundary twoC*) =
line-integral F {*j*} *?bottom-edge* + *line-integral F* {*j*} *?right-edge*
− *line-integral F* {*j*} *?top-edge* − *line-integral F* {*j*} *?left-edge*
proof (*simp add*: *one-chain-line-integral-def horizontal-boundary-def vertical-boundary-def* *boundary-def*)
have *finite1*: *finite* {(− *1*::*int*, *λy. twoC* (*0*, *y*)), (*1*, *λy. twoC* (*1*, *y*)), (− *1*, *λx. twoC* (*x*, *1*))} **by** *auto*
then have *sum-step1*: (∑ (*k*, *g*)∈{(− (*1*::*int*), *λy. twoC* (*0*, *y*)), (*1*, *λy. twoC* (*1*, *y*)), (*1*, *λx. twoC* (*x*, *0*)), (− *1*, *λx. twoC* (*x*, *1*))}. *k* ∗ *line-integral F* {*j*} *g*) =
line-integral F {*j*} (*λx. twoC* (*x*, *0*)) + (∑ (*k*, *g*)∈{(− (*1*::*int*), *λy. twoC* (*0*, *y*)), (*1*, *λy. twoC* (*1*, *y*)), (− *1*, *λx. twoC* (*x*, *1*))}. *k* ∗ *line-integral* *F* {*j*} *g*)
using *sum.insert-remove* [*OF finite1*]
using *valid-two-cube*
apply (*simp only*: *one-chain-line-integral-def horizontal-boundary-def vertical-boundary-def boundary-def valid-two-cube-def*)
by (*auto simp*: *card-insert-if split*: *if-split-asm*)
have *three-distinct-edges*: *card* {(− *1*::*int*, *λy. twoC* (*0*, *y*)), (*1*, *λy. twoC* (*1*, *y*)), (− *1*, *λx. twoC* (*x*, *1*))} = *3*

using *valid-two-cube*
apply (*simp add*: *one-chain-line-integral-def horizontal-boundary-def vertical-boundary-def boundary-def valid-two-cube-def*)
by (*auto simp*: *card-insert-if split*: *if-split-asm*)
have *finite2*: *finite* {(− *1*::*int*, λ*y*. *twoC* (*0*, *y*)), (*1*, λ*y*. *twoC* (*1*, *y*))} **by** *auto*
then have *sum-step2*: (∑ (*k*, *g*)∈{(− (*1*::*int*), λ*y*. *twoC* (*0*, *y*)), (*1*, λ*y*. *twoC* (*1*, *y*)), (−*1*, λ*x*. *twoC* (*x*, *1*))}. *k* ∗ *line-integral F* {*j*} *g*) =
(− *line-integral F* {*j*} (λ*x*. *twoC* (*x*, *1*))) + (∑ (*k*, *g*)∈{(− (*1*::*int*), λ*y*. *twoC* (*0*, *y*)), (*1*, λ*y*. *twoC* (*1*, *y*))}. *k* ∗ *line-integral F* {*j*} *g*)
using *sum.insert-remove* [*OF finite2*] *three-distinct-edges*
by (*auto simp*: *card-insert-if split*: *if-split-asm*)
have *two-distinct-edges*: *card* {(− *1*::*int*, λ*y*. *twoC* (*0*, *y*)), (*1*, λ*y*. *twoC* (*1*, *y*))} = *2*
using *three-distinct-edges*
by (*simp add*: *one-chain-line-integral-def horizontal-boundary-def vertical-boundary-def boundary-def*)
have *finite3*: *finite* {(− *1*::*int*, λ*y*. *twoC* (*0*, *y*))} **by** *auto*
then have *sum-step3*: (∑ (*k*, *g*)∈{(− (*1*::*int*), λ*y*. *twoC* (*0*, *y*)), (*1*, λ*y*. *twoC* (*1*, *y*))}. *k* ∗ *line-integral F* {*j*} *g*) =
(*line-integral F* {*j*} (λ*y*. *twoC* (*1*, *y*))) + (∑ (*k*, *g*)∈{(− (*1*::*real*), λ*y*. *twoC* (*0*, *y*))}. *k* ∗ *line-integral F* {*j*} *g*)
using *sum.insert-remove* [*OF finite2*] *three-distinct-edges*
by (*auto simp*: *card-insert-if split*: *if-split-asm*)
show (∑ *x*∈{(− *1*::*int*, λ*y*. *twoC* (*0*, *y*)), (*1*, λ*y*. *twoC* (*1*, *y*)), (*1*, λ*x*. *twoC* (*x*, *0*)), (− *1*, λ*x*. *twoC* (*x*, *1*))}. *case x of* (*k*, *g*) ⇒ *k* ∗ *line-integral F* {*j*} *g*) =
line-integral F {*j*} (λ*x*. *twoC* (*x*, *0*)) + *line-integral F* {*j*} (λ*y*. *twoC* (*1*, *y*)) − *line-integral F* {*j*} (λ*x*. *twoC* (*x*, *1*)) − *line-integral F* {*j*} (λ*y*. *twoC* (*0*, *y*))
using *sum-step1 sum-step2 sum-step3* **by** *auto*
qed
obtain *a b g1 g2* **where**
twoCisTypeII: *a* < *b*
(∀ *x* ∈ *cbox a b*. *g2 x* ≤ *g1 x*)
cubeImage twoC = {(*y*, *x*). *x* ∈ *cbox a b* ∧ *y* ∈ *cbox* (*g2 x*) (*g1 x*)}
twoC = (λ(*y*, *x*). ((*1* − *y*) ∗ *g2* ((*1* − *x*) ∗ *a* + *x* ∗ *b*) + *y* ∗ *g1* ((*1* − *x*) ∗ *a* + *x* ∗ *b*), (*1* − *x*) ∗ *a* + *x* ∗ *b*))
g1 piecewise-C1-differentiable-on {*a* .. *b*}
g2 piecewise-C1-differentiable-on {*a* .. *b*}
using *two-cube* **and** *typeII-twoCubeImg*[*oftwoC*]
by *auto*
have *left-edge-explicit*: *?left-edge* = (λ*x*. (*g2* (*a* + (*b* − *a*) ∗ *x*), *a* + (*b* − *a*) ∗ *x*))
by (*simp add*: *twoCisTypeII*(*4*) *algebra-simps*)
have *left-edge-smooth*: *?left-edge piecewise-C1-differentiable-on* {*0*..*1*}
proof −
have ∀ *x*. (λ*x*. (*a* + (*b* − *a*) ∗ *x*))−' {*x*} = {(*x* − *a*)/(*b* −*a*)}
using *twoCisTypeII*(*1*) **by**(*auto simp add*: *Set.vimage-def*)
then have *finite-vimg*: ⋀*x*. *finite*({*0*..*1*} ∩ (λ*x*. (*a* + (*b* − *a*) ∗ *x*))−' {*x*}) **by** *auto*

have *scale-shif-smth*: ($\lambda x.\ (a + (b - a) * x)$) *C1-differentiable-on* $\{0..1\}$ **using** *scale-shift-smooth* **by** *auto*
then have *scale-shif-pw-smth*: ($\lambda x.\ (a + (b - a) * x)$) *piecewise-C1-differentiable-on* $\{0..1\}$ **using** *C1-differentiable-imp-piecewise* **by** *blast*
have *g2-smooth*: *g2 piecewise-C1-differentiable-on* ($\lambda x.\ a + (b - a) * x$) ` $\{0..1\}$ **using** *add-scale-img*[*OF twoCisTypeII*(*1*)] *twoCisTypeII*(*6*) **by** *auto*
have ($\lambda x.\ g2\ (a + (b - a) * x)$) *piecewise-C1-differentiable-on* $\{0..1\}$
using *piecewise-C1-differentiable-compose*[*OF scale-shif-pw-smth g2-smooth finite-vimg*]
by (*auto simp add*: *o-def*)
then have (λx::*real*. ($g2\ (a + (b - a) * x),\ a + (b - a) * x$)) *piecewise-C1-differentiable-on* $\{0..1\}$
using *all-components-smooth-one-pw-smooth-is-pw-smooth*[**where** $f = (\lambda x$::*real*. ($g2\ (a + (b - a) * x),\ a + (b - a) * x$))]
by (*fastforce simp add*: *real-pair-basis*)
then show *?thesis* **using** *left-edge-explicit* **by** *auto*
qed
have *top-edge-explicit*: *?top-edge* = ($\lambda x.\ (g2\ b + x *_R (g1\ b - g2\ b),\ b)$)
and *right-edge-explicit*: *?right-edge* = ($\lambda x.\ (g1\ (a + (b - a) * x),\ a + (b - a) * x)$)
by (*simp-all add*: *twoCisTypeII*(*4*) *algebra-simps*)
have *right-edge-smooth*: *?right-edge piecewise-C1-differentiable-on* $\{0..1\}$
proof −
have $\forall x.\ (\lambda x.\ (a + (b - a) * x))-`\ \{x\} = \{(x - a)/(b - a)\}$
using *twoCisTypeII*(*1*) **by**(*auto simp add*: *Set.vimage-def*)
then have *finite-vimg*: $\bigwedge x.$ *finite*($\{0..1\} \cap (\lambda x.\ (a + (b - a) * x))-`\ \{x\}$) **by** *auto*
then have *scale-shif-pw-smth*: ($\lambda x.\ (a + (b - a) * x)$) *piecewise-C1-differentiable-on* $\{0..1\}$
using *C1-differentiable-imp-piecewise* [*OF scale-shift-smooth*] **by** *auto*
have *g1-smooth*: *g1 piecewise-C1-differentiable-on* ($\lambda x.\ a + (b - a) * x$) ` $\{0..1\}$ **using** *add-scale-img*[*OF twoCisTypeII*(*1*)] *twoCisTypeII*(*5*) **by** *auto*
have ($\lambda x.\ g1\ (a + (b - a) * x)$) *piecewise-C1-differentiable-on* $\{0..1\}$
using *piecewise-C1-differentiable-compose*[*OF scale-shif-pw-smth g1-smooth finite-vimg*]
by (*auto simp add*: *o-def*)
then have (λx::*real*. ($g1\ (a + (b - a) * x),\ a + (b - a) * x$)) *piecewise-C1-differentiable-on* $\{0..1\}$
using *all-components-smooth-one-pw-smooth-is-pw-smooth*[**where** $f = (\lambda x$::*real*. ($g1\ (a + (b - a) * x),\ a + (b - a) * x$))]
by (*fastforce simp add*: *real-pair-basis*)
then show *?thesis* **using** *right-edge-explicit* **by** *auto*
qed
have *bottom-edge-explicit*: *?bottom-edge* = ($\lambda x.\ (g2\ a + x *_R (g1\ a - g2\ a),\ a)$)
by (*simp add*: *twoCisTypeII*(*4*) *algebra-simps*)
show *integral* (*cubeImage twoC*) ($\lambda a.$ *partial-vector-derivative* ($\lambda x.\ (F\ x) \cdot j$) *i a*) = *one-chain-line-integral F* $\{j\}$ (*boundary twoC*)
using *Greens-thm-type-II*[*OF twoCisTypeII*(*3*) *left-edge-explicit left-edge-smooth top-edge-explicit right-edge-explicit right-edge-smooth*

bottom-edge-explicit f-analytically-valid
twoCisTypeII(*2*) *twoCisTypeII*(*1*)]
line-integral-around-boundary
by *auto*
have *line-integral-exists F* {*j*} *γ* **if** (*k*,*γ*) ∈ *boundary twoC* **for** *k γ*
proof −
have *line-integral-exists F* {*j*} (*λy. twoC* (*0*, *y*))
line-integral-exists F {*j*} (*λx. twoC* (*x*, *1*))
line-integral-exists F {*j*} (*λy. twoC* (*1*, *y*))
line-integral-exists F {*j*} (*λx. twoC* (*x*, *0*))
using *Greens-thm-type-II*[*OF twoCisTypeII*(*3*) *left-edge-explicit left-edge-smooth*
top-edge-explicit right-edge-explicit right-edge-smooth
bottom-edge-explicit f-analytically-valid
twoCisTypeII(*2*) *twoCisTypeII*(*1*)] *line-integral-around-boundary*
by *auto*
then show *line-integral-exists F* {*j*} *γ*
using *that* **by** (*auto simp add*: *boundary-def horizontal-boundary-def vertical-boundary-def*)
qed
then show ∀ (*k*,*γ*) ∈ *boundary twoC*. *line-integral-exists F* {*j*} *γ* **by** *auto*
qed

lemma *line-integral-exists-on-typeII-Cube-boundaries′*:
assumes (*k*,*γ*) ∈ *boundary twoC*
shows *line-integral-exists F* {*j*} *γ*
using *assms GreenThm-typeII-twoCube*(*2*) **by** *blast*

end

locale *green-typeII-chain* = *R2* +
fixes *F two-chain s*
assumes *valid-typeII-div*: *valid-typeII-division s two-chain* **and**
F-anal-valid: ∀ *twoC* ∈ *two-chain*. *analytically-valid* (*cubeImage twoC*) (*λx*. (*F x*) · *j*) *i*
begin

lemma *two-chain-valid-valid-cubes*: ∀ *two-cube* ∈ *two-chain*. *valid-two-cube two-cube*
using *valid-typeII-div*
by (*auto simp add*: *valid-two-chain-def*)

lemma *typeII-chain-line-integral-exists-boundary′*:
shows ∀ (*k*,*γ*) ∈ *two-chain-vertical-boundary two-chain*. *line-integral-exists F* {*j*} *γ*
proof −
have *integ-exis*: ∀ (*k*,*γ*) ∈ *two-chain-boundary two-chain*. *line-integral-exists F* {*j*} *γ*
using *green-typeII-cube.line-integral-exists-on-typeII-Cube-boundaries′*[*of i j*] *F-anal-valid valid-typeII-div*
apply(*auto simp add*: *two-chain-boundary-def*)

using *R2-axioms green-typeII-cube-axioms-def green-typeII-cube-def two-chain-valid-valid-cubes*
by *blast*
then show *integ-exis-vert*:
∀ (*k*,*γ*) ∈ *two-chain-vertical-boundary two-chain. line-integral-exists F* {*j*} *γ*
by (*simp add*: *two-chain-boundary-def two-chain-vertical-boundary-def boundary-def*)
qed

lemma *typeII-chain-line-integral-exists-boundary''*:
∀ (*k*,*γ*) ∈ *two-chain-horizontal-boundary two-chain. line-integral-exists F* {*j*} *γ*
proof −
have *integ-exis*: ∀ (*k*,*γ*) ∈ *two-chain-boundary two-chain. line-integral-exists F* {*j*} *γ*
using *green-typeII-cube.line-integral-exists-on-typeII-Cube-boundaries'*[*of i j*] *valid-typeII-div*
apply (*simp add*: *two-chain-boundary-def boundary-def*)
using *F-anal-valid R2-axioms green-typeII-cube-axioms-def green-typeII-cube-def two-chain-valid-valid-cubes* **by** *fastforce*
then show *integ-exis-vert*:
∀ (*k*,*γ*) ∈ *two-chain-horizontal-boundary two-chain. line-integral-exists F* {*j*} *γ*
by (*simp add*: *two-chain-boundary-def two-chain-horizontal-boundary-def boundary-def*)
qed

lemma *typeII-cube-line-integral-exists-boundary*:
∀ (*k*,*γ*) ∈ *two-chain-boundary two-chain. line-integral-exists F* {*j*} *γ*
using *valid-typeII-div typeII-chain-line-integral-exists-boundary' typeII-chain-line-integral-exists-boundary''*
apply (*auto simp add*: *two-chain-boundary-def two-chain-horizontal-boundary-def two-chain-vertical-boundary-def*)
using *boundary-def* **by** *auto*

lemma *type-II-chain-horiz-bound-valid*:
∀ (*k*,*γ*) ∈ *two-chain-horizontal-boundary two-chain. valid-path γ*
using *valid-typeII-div typeII-edges-are-valid-paths*
by (*force simp add*: *two-chain-boundary-def two-chain-horizontal-boundary-def boundary-def*)

lemma *type-II-chain-vert-bound-valid*:
∀ (*k*,*γ*) ∈ *two-chain-vertical-boundary two-chain. valid-path γ*
using *typeII-edges-are-valid-paths valid-typeII-div*
by (*force simp add*: *two-chain-boundary-def two-chain-vertical-boundary-def boundary-def*)

lemma *members-of-only-horiz-div-line-integrable'*:
assumes *only-horizontal-division one-chain two-chain*
(*k*::*int*, *γ*)∈*one-chain*
(*k*::*int*, *γ*)∈*one-chain*
finite two-chain
∀ *two-cube* ∈ *two-chain. valid-two-cube two-cube*

```
  shows line-integral-exists F {j} γ
proof −
   have integ-exis: ∀ (k,γ) ∈ two-chain-boundary two-chain. line-integral-exists F
{j} γ
    using typeII-cube-line-integral-exists-boundary by blast
  have integ-exis-vert:
    ∀ (k,γ) ∈ two-chain-vertical-boundary two-chain. line-integral-exists F {j} γ
    using typeII-chain-line-integral-exists-boundary′ assms by auto
  have integ-exis-horiz:
   (⋀k γ. (∃ (k′, γ′) ∈ two-chain-horizontal-boundary two-chain. ∃ a∈{0..1}. ∃ b∈{0..1}.
a ≤ b ∧ subpath a b γ′ = γ) ⟹
                              line-integral-exists F {j} γ)
    using integ-exis type-II-chain-horiz-bound-valid
    using line-integral-exists-subpath[of F {j}]
   by (fastforce simp add: two-chain-boundary-def two-chain-horizontal-boundary-def
        two-chain-vertical-boundary-def boundary-def)
  obtain ℋ 𝒱 where hv-props: finite ℋ
        (∀ (k,γ) ∈ ℋ. (∃ (k′, γ′) ∈ two-chain-horizontal-boundary two-chain.
            (∃ a ∈ {0 .. 1}. ∃ b ∈ {0..1}. a ≤ b ∧ subpath a b γ′ = γ)))
        one-chain = ℋ ∪ 𝒱
        ((common-sudiv-exists (two-chain-vertical-boundary two-chain) 𝒱)
          ∨ common-reparam-exists 𝒱 (two-chain-vertical-boundary two-chain))
    finite 𝒱
    boundary-chain 𝒱
    ∀ (k,γ)∈𝒱. valid-path γ
    using assms(1) unfolding only-horizontal-division-def  by blast
  have finite-j: finite {j} by auto
  show line-integral-exists F {j} γ
  proof(cases common-sudiv-exists (two-chain-vertical-boundary two-chain) 𝒱)
    case True
    show ?thesis
     using gen-common-subdivision-imp-eq-line-integral(2)[OF True two-chain-vertical-boundary-is-boundary-ch
hv-props(6) integ-exis-vert finite-two-chain-vertical-boundary[OF assms(4)] hv-props(5)
finite-j]
        integ-exis-horiz[of γ] assms(3) case-prod-conv hv-props(2) hv-props(3)
      by fastforce
  next
    case False
    have i: {j} ⊆ Basis using j-is-y-axis real-pair-basis by auto
     have ii:  ∀ (k2, γ2)∈two-chain-vertical-boundary two-chain. ∀ b∈{j}. continu-
ous-on (path-image γ2) (λx. F x · b)
      using F-anal-valid field-cont-on-typeII-region-cont-on-edges valid-typeII-div
       by (fastforce simp add: analytically-valid-def two-chain-vertical-boundary-def
boundary-def path-image-def)
    show line-integral-exists F {j} γ
     using common-reparam-exists-imp-eq-line-integral(2)[OF finite-j hv-props(5)
        finite-two-chain-vertical-boundary[OF assms(4)] hv-props(6) two-chain-vertical-boundary-is-boundary-ch
ii - hv-props(7) type-II-chain-vert-bound-valid]
        integ-exis-horiz[of γ] assms(3) hv-props False
```

by *fastforce*
qed
qed

lemma *GreenThm-typeII-twoChain*:
shows *two-chain-integral two-chain* (*partial-vector-derivative* (λ*a*. (*F a*) · *j*) *i*)
= *one-chain-line-integral F* {*j*} (*two-chain-boundary two-chain*)
proof (*simp add*: *two-chain-boundary-def one-chain-line-integral-def two-chain-integral-def*)
let *?F-a′* = *partial-vector-derivative* (λ*a*. (*F a*) · *j*) *i*
have (∑ (*k*,*g*)∈ *boundary twoCube*. *k* ∗ *line-integral F* {*j*} *g*) = *integral* (*cubeImage twoCube*) (λ*a*. *?F-a′ a*)
if *twoCube* ∈ *two-chain* **for** *twoCube*
using *green-typeII-cube.GreenThm-typeII-twoCube*(*1*) *valid-typeII-div F-anal-valid one-chain-line-integral-def valid-two-chain-def*
by (*simp add*: *R2-axioms green-typeII-cube-axioms-def green-typeII-cube-def that*)
then have *double-sum-eq-sum*:
(∑ *twoCube*∈(*two-chain*).(∑ (*k*,*g*)∈ *boundary twoCube*. *k* ∗ *line-integral F* {*j*} *g*))
= (∑ *twoCube*∈(*two-chain*). *integral* (*cubeImage twoCube*) (λ*a*. *?F-a′ a*))
using *Finite-Cartesian-Product.sum-cong-aux* **by** *auto*
have *pairwise-disjoint-boundaries*: ∀ *x*∈ (*boundary* ‘ *two-chain*). (∀ *y*∈ (*boundary* ‘ *two-chain*). (*x* ≠ *y* ⟶ (*x* ∩ *y* = {})))
using *valid-typeII-div* **by** (*fastforce simp add*: *image-def valid-two-chain-def pairwise-def*)
have *finite-boundaries*: ∀ *B* ∈ (*boundary*‘ *two-chain*). *finite B*
using *valid-typeII-div image-iff* **by** (*fastforce simp add*: *valid-two-cube-def valid-two-chain-def*)
have *boundary-inj*: *inj-on boundary two-chain*
using *valid-typeII-div* **by** (*force simp add*: *valid-two-cube-def valid-two-chain-def pairwise-def inj-on-def*)
have (∑ *x*∈(⋃ *x*∈*two-chain*. *boundary x*). *case x of* (*k*, *g*) ⇒ *k* ∗ *line-integral F* {*j*} *g*) =
(∑ *twoCube*∈(*two-chain*).(∑ (*k*,*g*)∈ *boundary twoCube*. *k* ∗ *line-integral F* {*j*} *g*))
using *sum.reindex*[*OF boundary-inj*, *of* λ*x*. (∑ (*k*,*g*)∈ *x*. *k* ∗ *line-integral F* {*j*} *g*)]
using *sum.Union-disjoint*[*OF finite-boundaries*
pairwise-disjoint-boundaries,
of λ*x*. *case x of* (*k*, *g*) ⇒ (*k*::*int*) ∗ *line-integral F* {*j*} *g*]
by *auto*
then show (∑ *C*∈*two-chain*. *integral* (*cubeImage C*) (λ*a*. *?F-a′ a*)) = (∑ *x*∈(⋃ *x*∈*two-chain*. *boundary x*). *case x of* (*k*, *g*) ⇒ *k* ∗ *line-integral F* {*j*} *g*)
using *double-sum-eq-sum* **by** *auto*
qed

lemma *GreenThm-typeII-divisible*:
assumes

gen-division: *gen-division s* (*cubeImage* ' *two-chain*)
shows *integral s* (*partial-vector-derivative* ($\lambda x.$ (*F x*) $\cdot$ *j*) *i*) = *one-chain-line-integral F* {*j*} (*two-chain-boundary two-chain*)
proof −
let *?F-a'* = (*partial-vector-derivative* ($\lambda x.$ (*F x*) $\cdot$ *j*) *i*)
have *integral s* ($\lambda x.$ *?F-a' x*) = *two-chain-integral two-chain* ($\lambda a.$ *?F-a' a*)
proof (*simp add*: *two-chain-integral-def*)
have *partial-deriv-integrable*:
(*?F-a' has-integral* (*integral* (*cubeImage twoCube*) *?F-a'*)) (*cubeImage twoCube*)
if *twoCube* $\in$ *two-chain* **for** *twoCube*
by (*simp add*: *analytically-valid-imp-part-deriv-integrable-on F-anal-valid integrable-integral that*)
then have *partial-deriv-integrable*:
$\bigwedge$*twoCubeImg*. *twoCubeImg* $\in$ *cubeImage* ' *two-chain* $\Longrightarrow$ (($\lambda x.$ *?F-a' x*) *has-integral* (*integral* (*twoCubeImg*) ($\lambda x.$ *?F-a' x*))) (*twoCubeImg*)
using *integrable-neg* **by** *force*
have *finite-images*: *finite* (*cubeImage* ' *two-chain*)
using *gen-division gen-division-def* **by** *auto*
have *negligible-images*: *pairwise* ($\lambda S\ S'.$ *negligible* ($S \cap S'$)) (*cubeImage* ' *two-chain*)
using *gen-division* **by** (*auto simp add*: *gen-division-def pairwise-def*)
have *inj-on cubeImage two-chain* **using** *valid-typeII-div valid-two-chain-def* **by** *auto*
then have ($\sum$ *twoCubeImg*$\in$*cubeImage* ' *two-chain*. *integral twoCubeImg* ($\lambda x.$ *?F-a' x*))
= ($\sum$ *C*$\in$*two-chain*. *integral* (*cubeImage C*) ($\lambda a.$ *?F-a' a*))
using *sum.reindex* **by** *auto*
then show *integral s* ($\lambda x.$ *?F-a' x*) = ($\sum$ *C*$\in$*two-chain*. *integral* (*cubeImage C*) ($\lambda a.$ *?F-a' a*))
using *has-integral-Union*[*OF finite-images partial-deriv-integrable negligible-images*] *gen-division*
by (*auto simp add*: *gen-division-def*)
qed
then show *?thesis*
using *GreenThm-typeII-twoChain F-anal-valid*
by *auto*
qed

lemma *GreenThm-typeII-divisible-region-boundary-gen*:
assumes *only-horizontal-division*: *only-horizontal-division* γ *two-chain*
shows *integral s* (*partial-vector-derivative* ($\lambda x.$ (*F x*) $\cdot$ *j*) *i*) = *one-chain-line-integral F* {*j*} γ
proof −
let *?F-a'* = (*partial-vector-derivative* ($\lambda x.$ (*F x*) $\cdot$ *j*) *i*)

have *horiz-line-integral-zero*:
one-chain-line-integral F {*j*} (*two-chain-horizontal-boundary two-chain*) = *0*
proof (*simp add*: *one-chain-line-integral-def*)
have *line-integral F* {*j*} (*snd oneCube*) = *0*

if *oneCube* ∈ *two-chain-horizontal-boundary*(*two-chain*) **for** *oneCube*
proof −
from *that* **obtain** *x y1 y2 k*
where *horiz-edge-def*: *oneCube* = (*k*, (λ*t*::*real*. ((*1* − *t*) ∗ (*y2*) + *t* ∗ *y1*, *x*::*real*)))
using *valid-typeII-div*
by (*auto simp add*: *typeII-twoCube-def two-chain-horizontal-boundary-def horizontal-boundary-def*)
let *?horiz-edge* = (*snd oneCube*)
have *horiz-edge-y-const*: ∀ *t*. (*?horiz-edge t*) · *j* = *x*
by (*auto simp add*: *horiz-edge-def real-inner-class.inner-add-left euclidean-space-class.inner-not-same-Basis j-is-y-axis*)
have *horiz-edge-is-straight-path*: *?horiz-edge* = (λ*t*. (*y2* + *t* ∗ (*y1* − *y2*), *x*))
by (*auto simp*: *horiz-edge-def algebra-simps*)
have ∀ *x*. *?horiz-edge differentiable at x*
using *horiz-edge-is-straight-path straight-path-diffrentiable-y*
by (*metis mult-commute-abs*)
then show *line-integral F* {*j*} (*snd oneCube*) = *0*
using *line-integral-on-pair-straight-path*(*1*) *j-is-y-axis real-pair-basis horiz-edge-y-const*
by *blast*
qed
then show (∑ *x*∈*two-chain-horizontal-boundary two-chain*. *case x of* (*k*, *g*) ⇒ *k* ∗ *line-integral F* {*j*} *g*) = *0*
by (*force intro*: *sum.neutral*)
qed

have *boundary-is-finite*: *finite* (*two-chain-boundary two-chain*)
unfolding *two-chain-boundary-def*
proof (*rule finite-UN-I*)
show *finite two-chain*
using *valid-typeII-div finite-image-iff gen-division-def valid-two-chain-def* **by** *auto*
show ⋀*a*. *a* ∈ *two-chain* ⟹ *finite* (*boundary a*)
by (*simp add*: *boundary-def horizontal-boundary-def vertical-boundary-def*)
qed
have *boundary-is-vert-hor*:
two-chain-boundary two-chain =
(*two-chain-vertical-boundary two-chain*) ∪
(*two-chain-horizontal-boundary two-chain*)
by(*auto simp add*: *two-chain-boundary-def two-chain-vertical-boundary-def two-chain-horizontal-boundary-de*
boundary-def)
then have *hor-vert-finite*:
finite (*two-chain-vertical-boundary two-chain*)
finite (*two-chain-horizontal-boundary two-chain*)
using *boundary-is-finite* **by** *auto*
have *horiz-verti-disjoint*:
(*two-chain-vertical-boundary two-chain*) ∩
(*two-chain-horizontal-boundary two-chain*) = {}
proof (*simp add*: *two-chain-vertical-boundary-def two-chain-horizontal-boundary-def*

horizontal-boundary-def
vertical-boundary-def)
show (⋃*x*∈*two-chain*. {(− *1*, λ*y*. *x* (*0*, *y*)), (*1*::*int*, λ*y*. *x* (*1*::*real*, *y*))}) ∩ (⋃*x*∈*two-chain*. {(*1*, λ*z*. *x* (*z*, *0*)), (− *1*, λ*z*. *x* (*z*, *1*))}) = {}
proof −
have {(− *1*, λ*y*. *twoCube* (*0*, *y*)), (*1*::*int*, λ*y*. *twoCube* (*1*, *y*))} ∩
{(*1*, λ*z*. *twoCube2* (*z*, *0*)), (− *1*, λ*z*. *twoCube2* (*z*, *1*))} = {}
if *twoCube*∈*two-chain* *twoCube2*∈*two-chain* **for** *twoCube* *twoCube2*
proof (*cases* *twoCube* = *twoCube2*)
case *True*
have *card-4*: *card* {(− *1*, λ*y*. *twoCube2* (*0*::*real*, *y*)), (*1*::*int*, λ*y*. *twoCube2* (*1*, *y*)), (*1*, λ*x*. *twoCube2* (*x*, *0*)), (− *1*, λ*x*. *twoCube2* (*x*, *1*))} = *4*
using *valid-typeII-div* *valid-two-chain-def* *that*(*2*)
by (*auto simp add*: *boundary-def* *vertical-boundary-def* *horizontal-boundary-def* *valid-two-cube-def*)
show *?thesis*
using *card-4* **by** (*auto simp*: *True* *card-insert-if* *split*: *if-split-asm*)
next
case *False*
show *?thesis*
using *valid-typeII-div* *valid-two-chain-def*
by (*simp add*: *boundary-def* *vertical-boundary-def* *horizontal-boundary-def* *pairwise-def* ‹*twoCube* ≠ *twoCube2*› *that*)
qed
then have ⋃ ((λ*twoCube*. {(− *1*, λ*y*. *twoCube* (*0*::*real*, *y*)), (*1*::*real*, λ*y*. *twoCube* (*1*::*real*, *y*))}) ‘ *two-chain*)
∩ ⋃ ((λ*twoCube*. {(*1*::*int*, λ*z*. *twoCube* (*z*, *0*::*real*)), (− *1*, λ*z*. *twoCube* (*z*, *1*::*real*))}) ‘ *two-chain*)
= {}
by (*fastforce simp add*: *Union-disjoint*)
then show *?thesis* **by** *force*
qed
qed
have *one-chain-line-integral* *F* {*j*} (*two-chain-boundary* *two-chain*)
= *one-chain-line-integral* *F* {*j*} (*two-chain-vertical-boundary* *two-chain*) +
one-chain-line-integral *F* {*j*} (*two-chain-horizontal-boundary* *two-chain*)
using *boundary-is-vert-hor* *horiz-verti-disjoint*
by (*simp add*: *hor-vert-finite* *sum.union-disjoint* *one-chain-line-integral-def*)
then have *y-axis-line-integral-is-only-vertical*:
one-chain-line-integral *F* {*j*} (*two-chain-boundary* *two-chain*)
= *one-chain-line-integral* *F* {*j*} (*two-chain-vertical-boundary* *two-chain*)
using *horiz-line-integral-zero*
by *auto*

obtain $\mathcal{H}$ $\mathcal{V}$ **where** *hv-props*: *finite* $\mathcal{H}$
(∀ (*k*,γ) ∈ $\mathcal{H}$. (∃ (*k*′, γ′) ∈ *two-chain-horizontal-boundary* *two-chain*.
(∃ *a* ∈ {*0* .. *1*}.
∃ *b* ∈ {*0*..*1*}.

$a \leq b \wedge$ *subpath a b* $\gamma' = \gamma$)))
$\gamma = \mathcal{H} \cup \mathcal{V}$
(*common-sudiv-exists* (*two-chain-vertical-boundary two-chain*) $\mathcal{V}$
$\vee$ *common-reparam-exists* $\mathcal{V}$ (*two-chain-vertical-boundary two-chain*))
finite $\mathcal{V}$
boundary-chain $\mathcal{V}$
$\forall (k,\gamma)\in\mathcal{V}$. *valid-path* γ
using *only-horizontal-division*
by(*fastforce simp add*: *only-horizontal-division-def*)
have *finite* $\{j\}$ **by** *auto*
then have *eq-integrals*: *one-chain-line-integral F* $\{j\}$ $\mathcal{V}$ = *one-chain-line-integral F* $\{j\}$ (*two-chain-vertical-boundary two-chain*)
proof(*cases common-sudiv-exists* (*two-chain-vertical-boundary two-chain*) $\mathcal{V}$)
case *True* **then show** *?thesis*
using *gen-common-subdivision-imp-eq-line-integral*(*1*)[*OF True two-chain-vertical-boundary-is-boundary-ch*
hv-props(*6*) - *hor-vert-finite*(*1*) *hv-props*(*5*)]
typeII-chain-line-integral-exists-boundary'
by *force*
next
case *False*
have *integ-exis-vert*:
$\forall (k,\gamma) \in$ *two-chain-vertical-boundary two-chain*. *line-integral-exists F* $\{j\}$ γ
using *typeII-chain-line-integral-exists-boundary' assms*
by (*fastforce simp add*: *valid-two-chain-def*)
have *integ-exis*: $\forall (k,\gamma) \in$ *two-chain-boundary two-chain*. *line-integral-exists F* $\{j\}$ γ
using *typeII-cube-line-integral-exists-boundary* **by** *blast*
have *valid-paths*: $\forall (k,\gamma) \in$ *two-chain-horizontal-boundary two-chain*. *valid-path* γ
using *typeII-edges-are-valid-paths valid-typeII-div*
by (*fastforce simp add*: *two-chain-boundary-def two-chain-horizontal-boundary-def boundary-def*)
have *integ-exis-horiz*:
$(\bigwedge k\ \gamma.\ (\exists (k', \gamma')\in$ *two-chain-horizontal-boundary two-chain*. $\exists\, a\in\{0..1\}$. $\exists\, b\in\{0..1\}$. $a \leq b \wedge$ *subpath a b* $\gamma' = \gamma) \Longrightarrow$
line-integral-exists F $\{j\}$ γ)
using *integ-exis valid-paths line-integral-exists-subpath*[*of F* $\{j\}$]
by (*fastforce simp add*: *two-chain-boundary-def two-chain-horizontal-boundary-def two-chain-vertical-boundary-def boundary-def*)
have *finite-j*: *finite* $\{j\}$ **by** *auto*
have *i*: $\{j\} \subseteq$ *Basis* **using** *j-is-y-axis real-pair-basis* **by** *auto*
have *ii*: $\forall (k2, \gamma 2)\in$*two-chain-vertical-boundary two-chain*. $\forall\, b\in\{j\}$. *continuous-on* (*path-image* $\gamma 2$) $(\lambda x.\ F\ x \cdot b)$
using *valid-typeII-div field-cont-on-typeII-region-cont-on-edges F-anal-valid*
by (*fastforce simp add*: *analytically-valid-def two-chain-vertical-boundary-def boundary-def path-image-def*)
show *one-chain-line-integral F* $\{j\}$ $\mathcal{V}$ = *one-chain-line-integral F* $\{j\}$ (*two-chain-vertical-boundary two-chain*)
using *hv-props*(*4*) *False common-reparam-exists-imp-eq-line-integral*(*1*)[*OF fi-*

```
nite-j hv-props(5) hor-vert-finite(1) hv-props(6) two-chain-vertical-boundary-is-boundary-chain
ii
          - hv-props(7) type-II-chain-vert-bound-valid]
      by fastforce
  qed

  have line-integral-on-path:
    one-chain-line-integral F {j} γ =
     one-chain-line-integral F {j} (two-chain-vertical-boundary two-chain)
  proof (simp only: one-chain-line-integral-def)
     have line-integral F {j} (snd oneCube) = 0 if oneCube: oneCube ∈ 𝓗 for
oneCube
    proof −
      obtain k γ where k-gamma: (k,γ) = oneCube
        using prod.collapse by blast
      then obtain k′ γ′ a b where kp-gammap:
        (k′::int, γ′) ∈ two-chain-horizontal-boundary two-chain
        a ∈ {0 .. 1}
        b ∈ {0..1}
        subpath a b γ′ = γ
        using hv-props oneCube
        by (smt case-prodE split-conv)
     obtain x y1 y2 where horiz-edge-def: (k′,γ′) = (k′, (λt::real. ((1 − t) ∗ (y2)
+ t ∗ y1, x::real)))
        using valid-typeII-div kp-gammap
          by (auto simp add: typeII-twoCube-def two-chain-horizontal-boundary-def
horizontal-boundary-def)
      have horiz-edge-y-const: ∀ t. γ (t) • j = x
        using horiz-edge-def kp-gammap(4)
        by (auto simp add: real-inner-class.inner-add-left
            euclidean-space-class.inner-not-same-Basis j-is-y-axis subpath-def)
      have horiz-edge-is-straight-path:
          γ = (λt::real. ((1∗y2 − a∗y2) +  a∗y1 + ((b−a)∗y1 − (b − a)∗y2)∗t,
x::real))
      proof −
        fix t::real
        have (1 − (b − a)∗t + a) ∗ (y2) + ((b−a)∗t + a) ∗ y1 = (1 − (b − a)∗t
+ a) ∗ (y2) + ((b−a)∗t + a) ∗ y1
          by auto
        then have γ = (λt::real. ((1 − (b − a)∗t − a) ∗ (y2) + ((b−a)∗t + a) ∗
y1, x::real))
       using horiz-edge-def Product-Type.snd-conv Product-Type.fst-conv kp-gammap(4)
         by (simp add: subpath-def diff-diff-eq[symmetric])
       also have . . . = (λt::real. ((1∗y2 − (b − a)∗y2∗t − a∗y2) + ((b−a)∗y1∗t
+ a∗y1), x::real))
       by(auto simp add: ring-class.ring-distribs(2) Groups.diff-diff-eq left-diff-distrib)
        also have ... = (λt::real. ((1∗y2 − a∗y2) +  a∗y1 + ((b−a)∗y1 − (b −
a)∗y2)∗t, x::real))
         by (force simp add: left-diff-distrib)
```

finally show $\gamma = (\lambda t::real.\ ((1*y2\ -\ a*y2) +\ \ a*y1\ + ((b-a)*y1\ -\ (b\ -\ a)*y2)*t,\ x::real))$ **.**
qed
show *line-integral F* $\{j\}$ (*snd oneCube*) = *0*
proof −
have $\forall\, x.\ \gamma$ *differentiable at x*
by (*simp add*: *horiz-edge-is-straight-path straight-path-diffrentiable-y*)
then have *line-integral F* $\{j\}$ γ = *0*
by (*simp add*: *horiz-edge-y-const line-integral-on-pair-straight-path*(*1*))
then show *?thesis*
using *Product-Type.snd-conv k-gamma* **by** *auto*
qed
qed
then have $\forall\, x \in \mathcal{H}$. (*case x of* (*k*, *g*) $\Rightarrow$ (*k*::*int*) $*$ *line-integral F* $\{j\}$ *g*) = *0*
by *auto*
then show ($\sum x \in \gamma$. *case x of* (*k*, *g*) $\Rightarrow$ *real-of-int k* $*$ *line-integral F* $\{j\}$ *g*) =
($\sum x \in$*two-chain-vertical-boundary two-chain*. *case x of* (*k*, *g*) $\Rightarrow$ *real-of-int k* $*$ *line-integral F* $\{j\}$ *g*)
using *hv-props*(*1*) *hv-props*(*3*) *hv-props*(*5*) *sum-zero-set hor-vert-finite*(*1*) *eq-integrals*
by (*clarsimp simp add*: *one-chain-line-integral-def sum-zero-set*)
qed
then have *one-chain-line-integral F* $\{j\}$ γ =
one-chain-line-integral F $\{j\}$ (*two-chain-vertical-boundary two-chain*)
using *line-integral-on-path* **by** *auto*
then have *one-chain-line-integral F* $\{j\}$ γ =
one-chain-line-integral F $\{j\}$ (*two-chain-boundary two-chain*)
using *y-axis-line-integral-is-only-vertical* **by** *auto*
then show *?thesis*
using *valid-typeII-div GreenThm-typeII-divisible* **by** *auto*
qed

lemma *GreenThm-typeII-divisible-region-boundary*:
assumes
two-cubes-trace-vertical-boundaries:
two-chain-vertical-boundary two-chain $\subseteq \gamma$ **and**
boundary-of-region-is-subset-of-partition-boundary:
$\gamma \subseteq$ *two-chain-boundary two-chain*
shows *integral s* (*partial-vector-derivative* ($\lambda x.$ (*F x*) $\cdot$ *j*) *i*) = *one-chain-line-integral F* $\{j\}$ γ
proof −
let *?F-a′* = (*partial-vector-derivative* ($\lambda x.$ (*F x*) $\cdot$ *j*) *i*)

have *horiz-line-integral-zero*:
one-chain-line-integral F $\{j\}$ (*two-chain-horizontal-boundary two-chain*) = *0*
proof (*simp add*: *one-chain-line-integral-def*)
have *line-integral F* $\{j\}$ (*snd oneCube*) = *0*
if *one*: *oneCube* $\in$ *two-chain-horizontal-boundary*(*two-chain*) **for** *oneCube*

proof −
obtain *x y1 y2 k* **where** *horiz-edge-def*: *oneCube* = (*k*, (λ*t*::*real*. ((*1* − *t*) ∗ (*y2*) + *t* ∗ *y1*, *x*::*real*)))
using *valid-typeII-div one*
by (*auto simp add*: *typeII-twoCube-def two-chain-horizontal-boundary-def horizontal-boundary-def*)
let *?horiz-edge* = (*snd oneCube*)
have *horiz-edge-y-const*: ∀ *t*. (*?horiz-edge t*) · *j* = *x*
using *horiz-edge-def*
by (*auto simp add*: *real-inner-class.inner-add-left*
euclidean-space-class.inner-not-same-Basis j-is-y-axis)
have *horiz-edge-is-straight-path*:
?horiz-edge = (λ*t*. (*y2* + *t* ∗ (*y1* − *y2*), *x*))
by (*simp add*: *add-diff-eq diff-add-eq mult.commute right-diff-distrib horiz-edge-def*)
show *line-integral F* {*j*} (*snd oneCube*) = *0*
by (*metis horiz-edge-is-straight-path horiz-edge-y-const line-integral-on-pair-straight-path*(*1*) *mult.commute straight-path-diffrentiable-y*)
qed
then show (∑ *x*∈*two-chain-horizontal-boundary two-chain*. *case x of* (*k*, *g*) ⇒ *k* ∗ *line-integral F* {*j*} *g*) = *0*
by (*simp add*: *prod.case-eq-if*)
qed

have *boundary-is-finite*: *finite* (*two-chain-boundary two-chain*)
unfolding *two-chain-boundary-def*
proof (*rule finite-UN-I*)
show *finite two-chain*
using *valid-typeII-div finite-image-iff gen-division-def valid-two-chain-def* **by** *auto*
show ⋀*a*. *a* ∈ *two-chain* ⟹ *finite* (*boundary a*)
by (*simp add*: *boundary-def horizontal-boundary-def vertical-boundary-def*)
qed
have *boundary-is-vert-hor*:
two-chain-boundary two-chain = (*two-chain-vertical-boundary two-chain*) ∪ (*two-chain-horizontal-boundary two-chain*)
by(*auto simp add*: *two-chain-boundary-def two-chain-vertical-boundary-def two-chain-horizontal-boundary-de boundary-def*)
then have *hor-vert-finite*:
finite (*two-chain-vertical-boundary two-chain*)
finite (*two-chain-horizontal-boundary two-chain*)
using *boundary-is-finite* **by** *auto*
have *horiz-verti-disjoint*:
(*two-chain-vertical-boundary two-chain*) ∩ (*two-chain-horizontal-boundary two-chain*) = {}
proof (*simp add*: *two-chain-vertical-boundary-def two-chain-horizontal-boundary-def horizontal-boundary-def*
vertical-boundary-def)
show (⋃*x*∈*two-chain*. {(− *1*, λ*y*. *x* (*0*, *y*)), (*1*::*int*, λ*y*. *x* (*1*::*real*, *y*))}) ∩ (⋃*x*∈*two-chain*. {(*1*, λ*z*. *x* (*z*, *0*)), (− *1*, λ*z*. *x* (*z*, *1*))}) = {}

```
    proof -
      have {(- 1, λy. twoCube (0, y)), (1::int, λy. twoCube (1, y))} ∩
            {(1, λy. twoCube2 (y, 0)), (- 1, λy. twoCube2 (y, 1))} = {}
        if  twoCube∈two-chain twoCube2∈two-chain for twoCube twoCube2
      proof (cases twoCube = twoCube2)
        case True
        have card {(- 1, λy. twoCube2 (0::real, y)), (1::int, λy. twoCube2 (1, y)),
(1, λx. twoCube2 (x, 0)), (- 1, λx. twoCube2 (x, 1))} = 4
          using valid-typeII-div valid-two-chain-def that(2)
       by (auto simp add: boundary-def vertical-boundary-def horizontal-boundary-def
valid-two-cube-def)
        then show ?thesis
          by (auto simp: True card-insert-if split: if-split-asm)
      next
        case False show ?thesis
          using valid-typeII-div valid-two-chain-def
           by (simp add: boundary-def vertical-boundary-def horizontal-boundary-def
pairwise-def ‹twoCube ≠ twoCube2› that(1) that(2))
      qed
       then have ⋃ ((λtwoCube. {(- 1, λy. twoCube (0::real, y)), (1::real, λy.
twoCube (1::real, y))}) ` two-chain)
                                          ∩ ⋃ ((λtwoCube. {(1::int, λz. twoCube (z,
0::real)), (- 1, λz. twoCube (z, 1::real))}) ` two-chain)
                                          = {}
        by (force simp add: Complete-Lattices.Union-disjoint)
      then show ?thesis by force
    qed
  qed
  have one-chain-line-integral F {j} (two-chain-boundary two-chain)
        = one-chain-line-integral F {j} (two-chain-vertical-boundary two-chain) +
          one-chain-line-integral F {j} (two-chain-horizontal-boundary two-chain)
    using boundary-is-vert-hor horiz-verti-disjoint
   by (auto simp add: one-chain-line-integral-def hor-vert-finite sum.union-disjoint)
  then have y-axis-line-integral-is-only-vertical:
    one-chain-line-integral F {j} (two-chain-boundary two-chain)
     = one-chain-line-integral F {j} (two-chain-vertical-boundary two-chain)
    using horiz-line-integral-zero by auto

  have ∃ℋ. ℋ ⊆ (two-chain-horizontal-boundary two-chain) ∧
            γ = ℋ ∪ (two-chain-vertical-boundary two-chain)
  proof
    let ?ℋ = γ - (two-chain-vertical-boundary two-chain)
   show ?ℋ ⊆ two-chain-horizontal-boundary two-chain ∧ γ = ?ℋ ∪ two-chain-vertical-boundary
two-chain
      using two-cubes-trace-vertical-boundaries
        boundary-of-region-is-subset-of-partition-boundary boundary-is-vert-hor
      by blast
  qed
  then obtain ℋ where
```

h-props: $\mathcal{H} \subseteq$ (*two-chain-horizontal-boundary two-chain*)
$\gamma = \mathcal{H} \cup$ (*two-chain-vertical-boundary two-chain*)
by *auto*
have *h-vert-disj*: $\mathcal{H} \cap$ (*two-chain-vertical-boundary two-chain*) = {}
using *horiz-verti-disjoint h-props*(*1*) **by** *auto*
have *h-finite*: *finite* $\mathcal{H}$
using *hor-vert-finite h-props*(*1*) *Finite-Set.rev-finite-subset* **by** *force*
have *line-integral-on-path*:
one-chain-line-integral F {*j*} γ =
one-chain-line-integral F {*j*} $\mathcal{H}$ + *one-chain-line-integral F* {*j*} (*two-chain-vertical-boundary two-chain*)
by (*auto simp add*: *one-chain-line-integral-def h-props sum.union-disjoint*[*OF h-finite hor-vert-finite*(*1*) *h-vert-disj*])

have *one-chain-line-integral F* {*j*} $\mathcal{H}$ = *0*
proof (*simp add*: *one-chain-line-integral-def*)
have *line-integral F* {*j*} (*snd oneCube*) = *0*
if *oneCube*: *oneCube* ∈ *two-chain-horizontal-boundary*(*two-chain*) **for** *oneCube*
proof −
obtain *x y1 y2 k* **where** *horiz-edge-def*: *oneCube* = (*k*, (λ*t*::*real*. ((*1* − *t*) ∗ (*y2*) + *t* ∗ *y1*, *x*::*real*)))
using *valid-typeII-div oneCube*
by (*auto simp add*: *typeII-twoCube-def two-chain-horizontal-boundary-def horizontal-boundary-def*)
let *?horiz-edge* = (*snd oneCube*)
have *horiz-edge-y-const*: ∀ *t*. (*?horiz-edge t*) · *j* = *x*
by (*simp add*: *j-is-y-axis horiz-edge-def*)
have *horiz-edge-is-straight-path*:
?horiz-edge = (λ*t*. (*y2* + *t* ∗ (*y1* − *y2*), *x*))
by (*simp add*: *add-diff-eq diff-add-eq mult.commute right-diff-distrib horiz-edge-def*)
show *line-integral F* {*j*} (*snd oneCube*) = *0*
by (*simp add*: *horiz-edge-is-straight-path j-is-y-axis line-integral-on-pair-straight-path*(*1*) *mult.commute straight-path-diffrentiable-y*)
qed
then have ∀ *oneCube* ∈ $\mathcal{H}$. *line-integral F* {*j*} (*snd oneCube*) = *0*
using *h-props* **by** *auto*
then have ∀ *x*∈$\mathcal{H}$. (*case x of* (*k*, *g*) ⇒ (*k*::*int*) ∗ *line-integral F* {*j*} *g*) = *0*
by *auto*
then show (∑ *x*∈$\mathcal{H}$. *case x of* (*k*, *g*) ⇒ *k* ∗ *line-integral F* {*j*} *g*) = *0*
by *simp*
qed
then have *one-chain-line-integral F* {*j*} γ =
one-chain-line-integral F {*j*} (*two-chain-vertical-boundary two-chain*)
using *line-integral-on-path* **by** *auto*
then have *one-chain-line-integral F* {*j*} γ =
one-chain-line-integral F {*j*} (*two-chain-boundary two-chain*)
using *y-axis-line-integral-is-only-vertical* **by** *auto*
then show *?thesis*

using *valid-typeII-div GreenThm-typeII-divisible* **by** *auto*
qed

end

locale *green-typeI-cube* = *R2* +
fixes *twoC F*
assumes
two-cube: *typeI-twoCube twoC* **and**
valid-two-cube: *valid-two-cube twoC* **and**
f-analytically-valid: *analytically-valid* (*cubeImage twoC*) (λ*x*. (*F x*) · *i*) *j*
begin

lemma *GreenThm-typeI-twoCube*:
shows *integral* (*cubeImage twoC*) (λ*a*. − *partial-vector-derivative* (λ*p*. *F p* · *i*) *j a*) = *one-chain-line-integral F* {*i*} (*boundary twoC*)
∀ (*k*,γ) ∈ *boundary twoC*. *line-integral-exists F* {*i*} γ
proof −
let *?bottom-edge* = (λ*x*. *twoC*(*x*, *0*))
let *?right-edge* = (λ*y*. *twoC*(*1*, *y*))
let *?top-edge* = (λ*x*. *twoC*(*x*, *1*))
let *?left-edge* = (λ*y*. *twoC*(*0*, *y*))
have *line-integral-around-boundary*:
one-chain-line-integral F {*i*} (*boundary twoC*) =
line-integral F {*i*} *?bottom-edge* + *line-integral F* {*i*} *?right-edge*
− *line-integral F* {*i*} *?top-edge* − *line-integral F* {*i*} *?left-edge*
proof (*simp add*: *one-chain-line-integral-def horizontal-boundary-def vertical-boundary-def boundary-def*)
have *finite1*: *finite* {(− *1*::*int*, λ*y*. *twoC* (*0*, *y*)), (*1*, λ*y*. *twoC* (*1*, *y*)), (− *1*, λ*x*. *twoC* (*x*, *1*))} **by** *auto*
have *sum-step1*: (∑(*k*, *g*)∈{(− (*1*::*int*), λ*y*. *twoC* (*0*, *y*)), (*1*, λ*y*. *twoC* (*1*, *y*)), (*1*, λ*x*. *twoC* (*x*, *0*)), (− *1*, λ*x*. *twoC* (*x*, *1*))}. *k* ∗ *line-integral F* {*i*} *g*) =
line-integral F {*i*} (λ*x*. *twoC* (*x*, *0*)) + (∑(*k*, *g*)∈{(− (*1*::*int*), λ*y*. *twoC* (*0*, *y*)), (*1*, λ*y*. *twoC* (*1*, *y*)), (− *1*, λ*x*. *twoC* (*x*, *1*))}. *k* ∗ *line-integral F* {*i*} *g*)
using *sum.insert-remove* [*OF finite1*] *valid-two-cube*
by (*auto simp*: *horizontal-boundary-def vertical-boundary-def boundary-def valid-two-cube-def card-insert-if split*: *if-split-asm*)
have *three-distinct-edges*: *card* {(− *1*::*int*, λ*y*. *twoC* (*0*, *y*)), (*1*, λ*y*. *twoC* (*1*, *y*)), (− *1*, λ*x*. *twoC* (*x*, *1*))} = *3*
using *valid-two-cube*
apply (*simp add*: *one-chain-line-integral-def horizontal-boundary-def vertical-boundary-def boundary-def valid-two-cube-def*)
by (*auto simp*: *card-insert-if split*: *if-split-asm*)
have *finite2*: *finite* {(− *1*::*int*, λ*y*. *twoC* (*0*, *y*)), (*1*, λ*y*. *twoC* (*1*, *y*))} **by** *auto*
have *sum-step2*: (∑(*k*, *g*)∈{(− (*1*::*int*), λ*y*. *twoC* (*0*, *y*)), (*1*, λ*y*. *twoC* (*1*, *y*)), (−*1*, λ*x*. *twoC* (*x*, *1*))}. *k* ∗ *line-integral F* {*i*} *g*) =
(− *line-integral F* {*i*} (λ*x*. *twoC* (*x*, *1*))) + (∑(*k*, *g*)∈{(− (*1*::*int*), λ*y*. *twoC* (*0*, *y*)), (*1*, λ*y*. *twoC* (*1*, *y*))}. *k* ∗ *line-integral F* {*i*} *g*)

using *sum.insert-remove* [*OF finite2*] *three-distinct-edges*
by (*auto simp*: *card-insert-if split*: *if-split-asm*)
have *two-distinct-edges*: *card* $\{(- \ 1::int, \ \lambda y. \ twoC \ (0, \ y)), \ (1, \ \lambda y. \ twoC \ (1, \ y))\} = 2$
using *three-distinct-edges*
by (*simp add*: *one-chain-line-integral-def horizontal-boundary-def vertical-boundary-def boundary-def*)
have *finite3*: *finite* $\{(- \ 1::int, \ \lambda y. \ twoC \ (0, \ y))\}$ **by** *auto*
have *sum-step3*: $(\sum (k, \ g)\in\{(- \ (1::int), \ \lambda y. \ twoC \ (0, \ y)), \ (1, \ \lambda y. \ twoC \ (1, \ y))\}. \ k * line\text{-}integral \ F \ \{i\} \ g) =$
$(line\text{-}integral \ F \ \{i\} \ (\lambda y. \ twoC \ (1, \ y))) + (\sum (k, \ g)\in\{(- \ (1::real), \ \lambda y. \ twoC \ (0, \ y))\}. \ k * line\text{-}integral \ F \ \{i\} \ g)$
using *sum.insert-remove* [*OF finite2*] *three-distinct-edges*
by (*auto simp*: *card-insert-if split*: *if-split-asm*)
show $(\sum x\in\{(- \ 1::int, \ \lambda y. \ twoC \ (0, \ y)), \ (1, \ \lambda y. \ twoC \ (1, \ y)), \ (1, \ \lambda x. \ twoC \ (x, \ 0)), \ (- \ 1, \ \lambda x. \ twoC \ (x, \ 1))\}. \ case \ x \ of \ (k, \ g) \Rightarrow k * line\text{-}integral \ F \ \{i\} \ g) =$
$line\text{-}integral \ F \ \{i\} \ (\lambda x. \ twoC \ (x, \ 0)) + line\text{-}integral \ F \ \{i\} \ (\lambda y. \ twoC \ (1, \ y)) - line\text{-}integral \ F \ \{i\} \ (\lambda x. \ twoC \ (x, \ 1)) - line\text{-}integral \ F \ \{i\} \ (\lambda y. \ twoC \ (0, \ y))$
using *sum-step1 sum-step2 sum-step3* **by** *auto*
qed
obtain *a b g1 g2* **where**
twoCisTypeI: $a < b$
$(\forall x \in cbox \ a \ b. \ g2 \ x \leq g1 \ x)$
$cubeImage \ twoC = \{(x,y). \ x \in cbox \ a \ b \wedge y \in cbox \ (g2 \ x) \ (g1 \ x)\}$
$twoC = (\lambda(x, \ y). \ ((1 - x) * a + x * b, \ (1 - y) * g2 \ ((1 - x) * a + x * b) + y * g1 \ ((1 - x) * a + x * b)))$
g1 piecewise-C1-differentiable-on $\{a \ .. \ b\}$
g2 piecewise-C1-differentiable-on $\{a \ .. \ b\}$
$(\lambda x. \ twoC(x, \ 0)) = (\lambda x. \ (a + (b - a) * x, \ g2 \ (a + (b - a) * x)))$
$(\lambda y. \ twoC(1, \ y)) = (\lambda x. \ (b, \ g2 \ b + x *_R \ (g1 \ b - g2 \ b)))$
$(\lambda x. \ twoC(x, \ 1)) = (\lambda x. \ (a + (b - a) * x, \ g1 \ (a + (b - a) * x)))$
$(\lambda y. \ twoC(0, \ y)) = (\lambda x. \ (a, \ g2 \ a + x *_R \ (g1 \ a - g2 \ a)))$
using *two-cube* **and** *typeI-cube-explicit-spec*[*oftwoC*] **by** *auto*
have *bottom-edge-smooth*: $(\lambda x. \ twoC \ (x, \ 0))$ *piecewise-C1-differentiable-on* $\{0..1\}$
using *typeI-twoCube-smooth-edges two-cube boundary-def vertical-boundary-def horizontal-boundary-def*
by *auto*
have *top-edge-smooth*: *?top-edge piecewise-C1-differentiable-on* $\{0..1\}$
using *typeI-twoCube-smooth-edges two-cube boundary-def vertical-boundary-def horizontal-boundary-def*
by *auto*
show *integral* (*cubeImage twoC*) $(\lambda a. - partial\text{-}vector\text{-}derivative \ (\lambda p. \ F \ p \cdot i) \ j \ a) = one\text{-}chain\text{-}line\text{-}integral \ F \ \{i\} \ (boundary \ twoC)$
using *Greens-thm-type-I*[*OF twoCisTypeI*(*3*) *twoCisTypeI*(*7*) *bottom-edge-smooth twoCisTypeI*(*8*) *twoCisTypeI*(*9*) *top-edge-smooth twoCisTypeI*(*10*) *f-analytically-valid twoCisTypeI*(*2*) *twoCisTypeI*(*1*)]
line-integral-around-boundary
by *auto*

have *line-integral-exists F* {*i*} (λ*y*. *twoC* (*0*, *y*))
line-integral-exists F {*i*} (λ*x*. *twoC* (*x*, *1*))
line-integral-exists F {*i*} (λ*y*. *twoC* (*1*, *y*))
line-integral-exists F {*i*} (λ*x*. *twoC* (*x*, *0*))
using *Greens-thm-type-I*[*OF twoCisTypeI*(*3*) *twoCisTypeI*(*7*) *bottom-edge-smooth twoCisTypeI*(*8*) *twoCisTypeI*(*9*) *top-edge-smooth twoCisTypeI*(*10*) *f-analytically-valid twoCisTypeI*(*2*) *twoCisTypeI*(*1*)]
line-integral-around-boundary
by *auto*
then have *line-integral-exists F* {*i*} γ **if** (*k*,γ) ∈ *boundary twoC* **for** *k* γ
using *that* **by** (*auto simp add*: *boundary-def horizontal-boundary-def vertical-boundary-def*)
then show ∀ (*k*,γ) ∈ *boundary twoC*. *line-integral-exists F* {*i*} γ **by** *auto*
qed

lemma *line-integral-exists-on-typeI-Cube-boundaries'*:
assumes (*k*,γ) ∈ *boundary twoC*
shows *line-integral-exists F* {*i*} γ
using *assms two-cube valid-two-cube f-analytically-valid GreenThm-typeI-twoCube*(*2*)
by *blast*

end

locale *green-typeI-chain* = *R2* +
fixes *F two-chain s*
assumes *valid-typeI-div*: *valid-typeI-division s two-chain* **and**
F-anal-valid: ∀ *twoC* ∈ *two-chain*. *analytically-valid* (*cubeImage twoC*) (λ*x*. (*F x*) · *i*) *j*
begin

lemma *two-chain-valid-valid-cubes*: ∀ *two-cube* ∈ *two-chain*. *valid-two-cube two-cube*
using *valid-typeI-div*
by (*auto simp add*: *valid-two-chain-def*)

lemma *typeI-cube-line-integral-exists-boundary'*:
assumes ∀ *two-cube* ∈ *two-chain*. *typeI-twoCube two-cube*
assumes ∀ *twoC* ∈ *two-chain*. *analytically-valid* (*cubeImage twoC*) (λ*x*. (*F x*) · *i*) *j*
assumes ∀ *two-cube* ∈ *two-chain*. *valid-two-cube two-cube*
shows ∀ (*k*,γ) ∈ *two-chain-vertical-boundary two-chain*. *line-integral-exists F* {*i*} γ
proof −
have *integ-exis*: ∀ (*k*,γ) ∈ *two-chain-boundary two-chain*. *line-integral-exists F* {*i*} γ
using *green-typeI-cube.line-integral-exists-on-typeI-Cube-boundaries'*[*of i j*] *assms*
using *R2-axioms green-typeI-cube-axioms-def green-typeI-cube-def two-chain-boundary-def*
by *fastforce*
then show *integ-exis-vert*:
∀ (*k*,γ) ∈ *two-chain-vertical-boundary two-chain*. *line-integral-exists F* {*i*} γ

by (*simp add*: *two-chain-boundary-def two-chain-vertical-boundary-def boundary-def*)
qed

lemma *typeI-cube-line-integral-exists-boundary''*:
∀ (*k*,γ) ∈ *two-chain-horizontal-boundary two-chain. line-integral-exists F* {*i*} γ
proof −
have *integ-exis*: ∀ (*k*,γ) ∈ *two-chain-boundary two-chain. line-integral-exists F* {*i*} γ
using *green-typeI-cube.line-integral-exists-on-typeI-Cube-boundaries'*[*of i j*] *valid-typeI-div*
apply (*simp add*: *two-chain-boundary-def boundary-def*)
using *F-anal-valid R2-axioms green-typeI-cube-axioms-def green-typeI-cube-def two-chain-valid-valid-cubes* **by** *fastforce*
then show *integ-exis-horiz*:
∀ (*k*,γ) ∈ *two-chain-horizontal-boundary two-chain. line-integral-exists F* {*i*} γ
by (*simp add*: *two-chain-boundary-def two-chain-horizontal-boundary-def boundary-def*)
qed

lemma *typeI-cube-line-integral-exists-boundary*:
∀ (*k*,γ) ∈ *two-chain-boundary two-chain. line-integral-exists F* {*i*} γ
using *typeI-cube-line-integral-exists-boundary' typeI-cube-line-integral-exists-boundary''*
apply (*auto simp add*: *two-chain-boundary-def two-chain-horizontal-boundary-def two-chain-vertical-boundary-def*)
by (*meson R2-axioms green-typeI-chain.F-anal-valid green-typeI-chain-axioms green-typeI-cube.line-integral-exists-on-typeI-Cube-boundaries' green-typeI-cube-axioms-def green-typeI-cube-def two-chain-valid-valid-cubes valid-typeI-div*)

lemma *type-I-chain-horiz-bound-valid*:
∀ (*k*,γ) ∈ *two-chain-horizontal-boundary two-chain. valid-path* γ
using *typeI-edges-are-valid-paths valid-typeI-div*
by (*force simp add*: *two-chain-boundary-def two-chain-horizontal-boundary-def boundary-def*)

lemma *type-I-chain-vert-bound-valid*:
assumes ∀ *two-cube* ∈ *two-chain. typeI-twoCube two-cube*
shows ∀ (*k*,γ) ∈ *two-chain-vertical-boundary two-chain. valid-path* γ
using *typeI-edges-are-valid-paths assms*(*1*)
by (*force simp add*: *two-chain-boundary-def two-chain-vertical-boundary-def boundary-def*)

lemma *members-of-only-vertical-div-line-integrable'*:
assumes *only-vertical-division one-chain two-chain*
(*k*::*int*, γ)∈*one-chain*
(*k*::*int*, γ)∈*one-chain*
finite two-chain
shows *line-integral-exists F* {*i*} γ
proof −
have *integ-exis*: ∀ (*k*,γ) ∈ *two-chain-boundary two-chain. line-integral-exists F*

{i} γ
  using *typeI-cube-line-integral-exists-boundary* **by** *blast*
have *integ-exis-horiz*:
  ∀ (k,γ) ∈ *two-chain-horizontal-boundary two-chain. line-integral-exists F* {i} γ
  using *typeI-cube-line-integral-exists-boundary′′ assms* **by** *auto*
have *valid-paths*: ∀ (k,γ) ∈ *two-chain-vertical-boundary two-chain. valid-path* γ
  using *type-I-chain-vert-bound-valid valid-typeI-div* **by** *linarith*
have *integ-exis-vert*:
  (⋀k γ. (∃ (k′, γ′)∈*two-chain-vertical-boundary two-chain*. ∃ a∈{0..1}. ∃ b∈{0..1}.
a ≤ b ∧ *subpath* a b γ′ = γ) ⟹
    line-integral-exists F {i} γ)
  using *integ-exis valid-paths line-integral-exists-subpath*[*of F* {i}]
  by (*fastforce simp add*: *two-chain-boundary-def two-chain-horizontal-boundary-def two-chain-vertical-boundary-def boundary-def*)
obtain ℋ 𝒱 **where** *hv-props*: *finite* 𝒱
  (∀ (k,γ) ∈ 𝒱. (∃ (k′, γ′) ∈ *two-chain-vertical-boundary two-chain*.
    (∃ a ∈ {0 .. 1}. ∃ b ∈ {0..1}. a ≤ b ∧ *subpath* a b γ′ = γ)))
  one-chain = ℋ ∪ 𝒱
  (*common-sudiv-exists* (*two-chain-horizontal-boundary two-chain*) ℋ)
  ∨ *common-reparam-exists* ℋ (*two-chain-horizontal-boundary two-chain*)
  finite ℋ
  boundary-chain ℋ
  ∀ (k,γ)∈ℋ. *valid-path* γ
  using *assms*(1) **unfolding** *only-vertical-division-def* **by** *blast*
have *finite-i*: *finite* {i} **by** *auto*
show *line-integral-exists F* {i} γ
proof(*cases common-sudiv-exists* (*two-chain-horizontal-boundary two-chain*) ℋ)
  case *True*
  show *?thesis*
  using *gen-common-subdivision-imp-eq-line-integral*(2)[*OF True two-chain-horizontal-boundary-is-boundary*
hv-props(6) *integ-exis-horiz finite-two-chain-horizontal-boundary*[*OF assms*(4)] *hv-props*(5) *finite-i*]
  integ-exis-vert[*of* γ] *assms*(3) *case-prod-conv hv-props*(2) *hv-props*(3)
  by *fastforce*
next
  case *False*
  have *i*: {i} ⊆ *Basis* **using** *i-is-x-axis real-pair-basis* **by** *auto*
  have *ii*: ∀ (k2, γ2)∈*two-chain-horizontal-boundary two-chain*. ∀ b∈{i}. *continuous-on* (*path-image* γ2) (λx. F x · b)
  using *assms field-cont-on-typeI-region-cont-on-edges F-anal-valid valid-typeI-div*
  by (*fastforce simp add*: *analytically-valid-def two-chain-horizontal-boundary-def boundary-def path-image-def*)
  show *line-integral-exists F* {i} γ
  using *common-reparam-exists-imp-eq-line-integral*(2)[*OF finite-i hv-props*(5)
finite-two-chain-horizontal-boundary[*OF assms*(4)] *hv-props*(6) *two-chain-horizontal-boundary-is-boundary-ch*
ii
  - *hv-props*(7) *type-I-chain-horiz-bound-valid*]
  integ-exis-vert[*of* γ] *False*
  assms(3) *hv-props*(2−4) **by** *fastforce*

qed
qed

lemma *GreenThm-typeI-two-chain*:
two-chain-integral two-chain ($\lambda a.$ − *partial-vector-derivative* ($\lambda x.$ (F x) · i) j a)
= *one-chain-line-integral* F {i} (*two-chain-boundary two-chain*)
proof (*simp add*: *two-chain-boundary-def one-chain-line-integral-def two-chain-integral-def*)
let *?F-b′* = *partial-vector-derivative* ($\lambda x.$ (F x) · i) j
have *all-two-cubes-have-four-distict-edges*: ∀ *twoCube* ∈ *two-chain*. *card* (*boundary twoCube*) = *4*
using *valid-typeI-div valid-two-chain-def valid-two-cube-def* **by** *auto*
have *no-shared-edges-have-similar-orientations*:
∀ *twoCube1* ∈ *two-chain*. ∀ *twoCube2* ∈ *two-chain*. *twoCube1* ≠ *twoCube2* ⟶
boundary twoCube1 ∩ *boundary twoCube2* = {}
using *valid-typeI-div valid-two-chain-def*
by (*auto simp add*: *pairwise-def*)
have ($\sum$ (*k*,*g*)∈ *boundary twoCube*. *k* ∗ *line-integral* F {i} g) = *integral* (*cubeImage twoCube*) ($\lambda a.$ − *?F-b′* a)
if *twoCube* ∈ *two-chain* **for** *twoCube*
proof −
have *analytically-val*: *analytically-valid* (*cubeImage twoCube*) ($\lambda x.$ F x · i) j
using *that F-anal-valid* **by** *auto*
show ($\sum$ (*k*, *g*)∈*boundary twoCube*. *k* ∗ *line-integral* F {i} g) = *integral* (*cubeImage twoCube*) ($\lambda a.$ − *?F-b′* a)
using *green-typeI-cube.GreenThm-typeI-twoCube*
apply (*simp add*: *one-chain-line-integral-def*)
by (*simp add*: *R2-axioms analytically-val green-typeI-cube-axioms-def green-typeI-cube-def that two-chain-valid-valid-cubes valid-typeI-div*)
qed
then have *double-sum-eq-sum*:
($\sum$ *twoCube*∈(*two-chain*).($\sum$ (*k*,*g*)∈ *boundary twoCube*. *k* ∗ *line-integral* F {i} g))
= ($\sum$ *twoCube*∈(*two-chain*). *integral* (*cubeImage twoCube*) ($\lambda a.$ − *?F-b′* a))
using *Finite-Cartesian-Product.sum-cong-aux* **by** *auto*
have *pairwise-disjoint-boundaries*: ∀ x∈ (*boundary* ‘ *two-chain*). (∀ y∈ (*boundary* ‘ *two-chain*). (x ≠ y ⟶ (x ∩ y = {})))
using *no-shared-edges-have-similar-orientations*
by (*force simp add*: *image-def disjoint-iff-not-equal*)
have *finite-boundaries*: ∀ B ∈ (*boundary*‘ *two-chain*). *finite* B
using *all-two-cubes-have-four-distict-edges*
using *image-iff* **by** *fastforce*
have *boundary-inj*: *inj-on boundary two-chain*
using *all-two-cubes-have-four-distict-edges* **and** *no-shared-edges-have-similar-orientations*
by (*force simp add*: *inj-on-def*)
have ($\sum$ x∈($\bigcup$(*boundary*‘ *two-chain*)). *case x of* (*k*, *g*) ⇒ *k* ∗ *line-integral* F {i} g)
= (*sum* ∘ *sum*) ($\lambda x.$ *case x of* (*k*, *g*) ⇒ (*k*::*int*) ∗ *line-integral* F {i} g) (*boundary* ‘ *two-chain*)

using *sum.Union-disjoint*[*OF finite-boundaries pairwise-disjoint-boundaries*]
by *simp*
also have ... = ($\sum$ *twoCube*$\in$(*two-chain*).($\sum$ (*k*,*g*)$\in$ *boundary twoCube*. *k* $*$ *line-integral F* {*i*} *g*))
using *sum.reindex*[*OF boundary-inj*, *of* λ*x*. ($\sum$ (*k*,*g*)$\in$ *x*. *k* $*$ *line-integral F* {*i*} *g*)]
by *auto*
finally show ($\sum$ *C*$\in$*two-chain*. $-$ *integral* (*cubeImage C*) (*partial-vector-derivative* (λ*x*. *F x* $\cdot$ *i*) *j*)) = ($\sum$ *x*$\in$($\bigcup$ *x*$\in$*two-chain*. *boundary x*). *case x of* (*k*, *g*) $\Rightarrow$ *real-of-int k* $*$ *line-integral F* {*i*} *g*)
using *double-sum-eq-sum* **by** *auto*
qed

lemma *GreenThm-typeI-divisible*:
assumes *gen-division*: *gen-division s* (*cubeImage* ' *two-chain*)
shows *integral s* (λ*x*. $-$ *partial-vector-derivative* (λ*a*. *F*(*a*) $\cdot$ *i*) *j x*) = *one-chain-line-integral F* {*i*} (*two-chain-boundary two-chain*)
proof $-$
let *?F-b'* = *partial-vector-derivative* (λ*a*. *F*(*a*) $\cdot$ *i*) *j*
have *integral s* (λ*x*. $-$ *?F-b' x*) = *two-chain-integral two-chain* (λ*a*. $-$ *?F-b' a*)
proof (*simp add*: *two-chain-integral-def*)
have ($\sum$ *f*$\in$*two-chain*. *integral* (*cubeImage f*) (*partial-vector-derivative* (λ*p*. *F p* $\cdot$ *i*) *j*)) = *integral s* (*partial-vector-derivative* (λ*p*. *F p* $\cdot$ *i*) *j*)
by (*metis analytically-valid-imp-part-deriv-integrable-on F-anal-valid gen-division two-chain-integral-def two-chain-integral-eq-integral-divisable valid-typeI-div*)
then show $-$ *integral s* (*partial-vector-derivative* (λ*a*. *F a* $\cdot$ *i*) *j*) = ($\sum$ *C*$\in$*two-chain*. $-$ *integral* (*cubeImage C*) (*partial-vector-derivative* (λ*a*. *F a* $\cdot$ *i*) *j*))
by (*simp add*: *sum-negf*)
qed
then show *?thesis*
using *GreenThm-typeI-two-chain assms* **by** *auto*
qed

lemma *GreenThm-typeI-divisible-region-boundary*:
assumes
gen-division: *gen-division s* (*cubeImage* ' *two-chain*) **and**
two-cubes-trace-horizontal-boundaries:
two-chain-horizontal-boundary two-chain $\subseteq$ γ **and**
boundary-of-region-is-subset-of-partition-boundary:
γ $\subseteq$ *two-chain-boundary two-chain*
shows *integral s* (λ*x*. $-$ *partial-vector-derivative* (λ*a*. *F*(*a*) $\cdot$ *i*) *j x*) = *one-chain-line-integral F* {*i*} γ
proof $-$
let *?F-b'* = *partial-vector-derivative* (λ*a*. *F*(*a*) $\cdot$ *i*)
have *all-two-cubes-have-four-distict-edges*: $\forall$ *twoCube* $\in$ *two-chain*. *card* (*boundary twoCube*) = *4*
using *valid-typeI-div valid-two-chain-def valid-two-cube-def* **by** *auto*
have *no-shared-edges-have-similar-orientations*:
$\forall$ *twoCube1* $\in$ *two-chain*. $\forall$ *twoCube2* $\in$ *two-chain*.

twoCube1 ≠ *twoCube2* ⟶ *boundary twoCube1* ∩ *boundary twoCube2* = {}
using *valid-typeI-div valid-two-chain-def* **by** (*auto simp add*: *pairwise-def*)

have *vert-line-integral-zero*:
one-chain-line-integral F {*i*} (*two-chain-vertical-boundary two-chain*) = *0*
proof (*simp add*: *one-chain-line-integral-def*)
have *line-integral F* {*i*} (*snd oneCube*) = *0*
if *oneCube*: *oneCube* ∈ *two-chain-vertical-boundary*(*two-chain*) **for** *oneCube*
proof −
obtain *x y1 y2 k* **where** *vert-edge-def*: *oneCube* = (*k*, (λ*t*::*real*. (*x*::*real*, (*1* − *t*) ∗ (*y2*) + *t* ∗ *y1*)))
using *valid-typeI-div oneCube*
by (*auto simp add*: *typeI-twoCube-def two-chain-vertical-boundary-def vertical-boundary-def*)
let *?vert-edge* = (*snd oneCube*)
have *vert-edge-x-const*: ∀ *t*. (*?vert-edge t*) · *i* = *x*
by (*simp add*: *i-is-x-axis vert-edge-def*)
have *vert-edge-is-straight-path*: *?vert-edge* = (λ*t*. (*x*, *y2* + *t* ∗ (*y1* − *y2*)))
using *vert-edge-def Product-Type.snd-conv*
by (*auto simp add*: *mult.commute right-diff-distrib′*)
show *?thesis*
by (*simp add*: *i-is-x-axis line-integral-on-pair-straight-path*(*1*) *mult.commute straight-path-diffrentiable-x vert-edge-is-straight-path*)
qed
then show (∑ *x*∈*two-chain-vertical-boundary two-chain*. *case x of* (*k*, *g*) ⇒ *k* ∗ *line-integral F* {*i*} *g*) = *0*
using *comm-monoid-add-class.sum.neutral* **by** (*simp add*: *prod.case-eq-if*)
qed

have *boundary-is-finite*: *finite* (*two-chain-boundary two-chain*)
unfolding *two-chain-boundary-def*
by (*metis all-two-cubes-have-four-distict-edges card.infinite finite-UN-I finite-imageD*

gen-division gen-division-def zero-neq-numeral valid-typeI-div valid-two-chain-def)
have *boundary-is-vert-hor*: (*two-chain-boundary two-chain*) =
(*two-chain-vertical-boundary two-chain*) ∪
(*two-chain-horizontal-boundary two-chain*)
by(*auto simp add*: *two-chain-boundary-def two-chain-vertical-boundary-def two-chain-horizontal-boundary-def boundary-def*)
then have *hor-vert-finite*:
finite (*two-chain-vertical-boundary two-chain*)
finite (*two-chain-horizontal-boundary two-chain*)
using *boundary-is-finite* **by** *auto*
have *horiz-verti-disjoint*:
(*two-chain-vertical-boundary two-chain*) ∩ (*two-chain-horizontal-boundary two-chain*) = {}
proof (*simp add*: *two-chain-vertical-boundary-def two-chain-horizontal-boundary-def horizontal-boundary-def*
vertical-boundary-def)

```
show (⋃x∈two-chain. {(− 1, λy. x (0, y)), (1::int, λy. x (1::real, y))}) ∩
(⋃x∈two-chain. {(1, λz. x (z, 0)), (− 1, λz. x (z, 1))}) = {}
  proof −
    have {(− 1, λy. twoCube (0, y)), (1::int, λy. twoCube (1, y))} ∩
        {(1, λz. twoCube2 (z, 0)), (− 1, λz. twoCube2 (z, 1))} = {}
      if twoCube∈two-chain twoCube2∈two-chain for twoCube twoCube2
    proof (cases twoCube = twoCube2)
      case True
      have card {(− 1::int, λy. twoCube2 (0::real, y)), (1::int, λy. twoCube2 (1,
y)), (1, λx. twoCube2 (x, 0)), (− 1, λx. twoCube2 (x, 1))} = 4
        using all-two-cubes-have-four-distict-edges that(2)
       by (auto simp add: boundary-def vertical-boundary-def horizontal-boundary-def)
      then show ?thesis
        by (auto simp: True card-insert-if split: if-split-asm)
    next
      case False
      then show ?thesis
        using no-shared-edges-have-similar-orientations
       by (simp add: that boundary-def vertical-boundary-def horizontal-boundary-def)
    qed
     then have ⋃ ((λtwoCube. {(−1::int, λy. twoCube (0,y)), (1, λy. twoCube
(1, y))}) ‘ two-chain)
            ∩ ⋃ ((λtwoCube. {(1, λy. twoCube (y, 0)), (−1, λz. twoCube (z, 1))})
‘ two-chain) = {}
      using Complete-Lattices.Union-disjoint by force
    then show ?thesis by force
  qed
qed
have one-chain-line-integral F {i} (two-chain-boundary two-chain)
      = one-chain-line-integral F {i} (two-chain-vertical-boundary two-chain) +
        one-chain-line-integral F {i} (two-chain-horizontal-boundary two-chain)
  using boundary-is-vert-hor horiz-verti-disjoint
 by (auto simp add: one-chain-line-integral-def hor-vert-finite sum.union-disjoint)
then have x-axis-line-integral-is-only-horizontal:
  one-chain-line-integral F {i} (two-chain-boundary two-chain)
   = one-chain-line-integral F {i} (two-chain-horizontal-boundary two-chain)
  using vert-line-integral-zero by auto

have ∃𝒱. 𝒱 ⊆ (two-chain-vertical-boundary two-chain) ∧ γ = 𝒱 ∪ (two-chain-horizontal-boundary
two-chain)
proof
  let ?𝒱 = γ − (two-chain-horizontal-boundary two-chain)
 show ?𝒱 ⊆ two-chain-vertical-boundary two-chain ∧ γ = ?𝒱 ∪ two-chain-horizontal-boundary
two-chain
    using two-cubes-trace-horizontal-boundaries
      boundary-of-region-is-subset-of-partition-boundary boundary-is-vert-hor
    by blast
qed
then obtain 𝒱 where
```

v-props: $\mathcal{V} \subseteq$ (*two-chain-vertical-boundary two-chain*) $\gamma = \mathcal{V} \cup$ (*two-chain-horizontal-boundary two-chain*)
by *auto*
have *v-horiz-disj*: $\mathcal{V} \cap$ (*two-chain-horizontal-boundary two-chain*) = {}
using *horiz-verti-disjoint v-props*(*1*) **by** *auto*
have *v-finite*: *finite* $\mathcal{V}$
using *hor-vert-finite v-props*(*1*) *Finite-Set.rev-finite-subset* **by** *force*
have *line-integral-on-path*: *one-chain-line-integral F* {*i*} γ =
one-chain-line-integral F {*i*} $\mathcal{V}$ + *one-chain-line-integral F* {*i*} (*two-chain-horizontal-boundary two-chain*)
by(*auto simp add*: *one-chain-line-integral-def v-props sum.union-disjoint*[*OF v-finite hor-vert-finite*(*2*) *v-horiz-disj*])

have *one-chain-line-integral F* {*i*} $\mathcal{V}$ = *0*
proof (*simp add*: *one-chain-line-integral-def*)
have *line-integral F* {*i*} (*snd oneCube*) = *0*
if *oneCube*: *oneCube* ∈ *two-chain-vertical-boundary*(*two-chain*) **for** *oneCube*
proof −
obtain *x y1 y2 k* **where** *vert-edge-def*: *oneCube* = (*k*, (λ*t*::*real*. (*x*::*real*, (*1* − *t*) ∗ (*y2*) + *t* ∗ *y1*)))
using *valid-typeI-div oneCube*
by (*auto simp add*: *typeI-twoCube-def two-chain-vertical-boundary-def vertical-boundary-def*)
let *?vert-edge* = (*snd oneCube*)
have *vert-edge-x-const*: ∀ *t*. (*?vert-edge t*) · *i* = *x*
by (*simp add*: *i-is-x-axis vert-edge-def*)
have *vert-edge-is-straight-path*:
?vert-edge = (λ*t*. (*x*, *y2* + *t* ∗ (*y1* − *y2*)))
by (*auto simp*: *vert-edge-def algebra-simps*)
have ∀ *x*. *?vert-edge differentiable at x*
by (*metis mult.commute vert-edge-is-straight-path straight-path-diffrentiable-x*)
then show *line-integral F* {*i*} (*snd oneCube*) = *0*
using *line-integral-on-pair-straight-path*(*1*) *vert-edge-x-const* **by** *blast*
qed
then have ∀ *oneCube* ∈ $\mathcal{V}$. *line-integral F* {*i*} (*snd oneCube*) = *0*
using *v-props* **by** *auto*
then show (∑ *x*∈$\mathcal{V}$. *case x of* (*k*, *g*) ⇒ *k* ∗ *line-integral F* {*i*} *g*) = *0*
using *comm-monoid-add-class.sum.neutral* **by** (*simp add*: *prod.case-eq-if*)
qed
then have *one-chain-line-integral F* {*i*} γ =
one-chain-line-integral F {*i*} (*two-chain-boundary two-chain*)
using *x-axis-line-integral-is-only-horizontal* **by** (*simp add*: *line-integral-on-path*)
then show *?thesis*
using *assms* **and** *GreenThm-typeI-divisible* **by** *auto*
qed

lemma *GreenThm-typeI-divisible-region-boundary-gen*:
assumes *valid-typeI-div*: *valid-typeI-division s two-chain* **and**
f-analytically-valid: ∀ *twoC* ∈ *two-chain*. *analytically-valid* (*cubeImage twoC*)

($\lambda a.\ F(a) \cdot i$) j **and**
only-vertical-division:
only-vertical-division γ *two-chain*
shows *integral s* ($\lambda x.$ − *partial-vector-derivative* ($\lambda a.\ F(a) \cdot i$) *j x*) = *one-chain-line-integral* F $\{i\}$ γ
proof −
let *?F-b′* = *partial-vector-derivative* ($\lambda a.\ F(a) \cdot i$)
have *all-two-cubes-have-four-distict-edges*: ∀ *twoCube* ∈ *two-chain*. *card* (*boundary twoCube*) = *4*
using *valid-typeI-div valid-two-chain-def valid-two-cube-def*
by *auto*
have *no-shared-edges-have-similar-orientations*:
∀ *twoCube1* ∈ *two-chain*. ∀ *twoCube2* ∈ *two-chain*.
twoCube1 ≠ *twoCube2* ⟶ *boundary twoCube1* ∩ *boundary twoCube2* = {}
using *valid-typeI-div valid-two-chain-def* **by** (*auto simp add*: *pairwise-def*)

have *vert-line-integral-zero*:
one-chain-line-integral F {*i*} (*two-chain-vertical-boundary two-chain*) = *0*
proof (*simp add*: *one-chain-line-integral-def*)
have *line-integral F* {*i*} (*snd oneCube*) = *0*
if *oneCube*: *oneCube* ∈ *two-chain-vertical-boundary*(*two-chain*) **for** *oneCube*
proof −
obtain *x y1 y2 k* **where** *vert-edge-def*: *oneCube* = (*k*, (λt::*real*. (*x*::*real*, (*1* − *t*) ∗ (*y2*) + *t* ∗ *y1*)))
using *valid-typeI-div oneCube*
by (*auto simp add*: *typeI-twoCube-def two-chain-vertical-boundary-def vertical-boundary-def*)
let *?vert-edge* = (*snd oneCube*)
have *vert-edge-x-const*: ∀ *t*. (*?vert-edge t*) · *i* = *x*
by (*simp add*: *i-is-x-axis vert-edge-def*)
have *vert-edge-is-straight-path*: *?vert-edge* = (λt. (*x*, *y2* + *t* ∗ (*y1* − *y2*)))
by (*auto simp*: *vert-edge-def algebra-simps*)
show *?thesis*
by (*simp add*: *i-is-x-axis line-integral-on-pair-straight-path*(*1*) *mult.commute straight-path-diffrentiable-x vert-edge-is-straight-path*)
qed
then show (∑ *x*∈*two-chain-vertical-boundary two-chain*. *case x of* (*k*, *g*) ⇒ *k* ∗ *line-integral F* {*i*} *g*) = *0*
using *comm-monoid-add-class.sum.neutral* **by** (*simp add*: *prod.case-eq-if*)
qed

have *boundary-is-finite*: *finite* (*two-chain-boundary two-chain*)
unfolding *two-chain-boundary-def*
proof (*rule finite-UN-I*)
show *finite two-chain*
using *assms*(*1*) *finite-imageD gen-division-def valid-two-chain-def* **by** *auto*
show ⋀*a*. *a* ∈ *two-chain* ⟹ *finite* (*boundary a*)
by (*simp add*: *boundary-def horizontal-boundary-def vertical-boundary-def*)
qed

have *boundary-is-vert-hor*: *two-chain-boundary two-chain* =
(*two-chain-vertical-boundary two-chain*) ∪ (*two-chain-horizontal-boundary two-chain*)
by (*auto simp add*: *two-chain-boundary-def two-chain-vertical-boundary-def two-chain-horizontal-boundary-def boundary-def*)
then have *hor-vert-finite*:
finite (*two-chain-vertical-boundary two-chain*)
finite (*two-chain-horizontal-boundary two-chain*)
using *boundary-is-finite* **by** *auto*
have *horiz-verti-disjoint*:
(*two-chain-vertical-boundary two-chain*) ∩ (*two-chain-horizontal-boundary two-chain*) = {}
proof (*simp add*: *two-chain-vertical-boundary-def two-chain-horizontal-boundary-def horizontal-boundary-def*
vertical-boundary-def)
show $(\bigcup x \in two\text{-}chain.\ \{(-\ 1,\ \lambda y.\ x\ (0,\ y)),\ (1{::}int,\ \lambda y.\ x\ (1{::}real,\ y))\})$
$\cap\ (\bigcup x \in two\text{-}chain.\ \{(1,\ \lambda y.\ x\ (y,\ 0)),\ (-\ 1,\ \lambda y.\ x\ (y,\ 1))\}) = \{\}$
proof −
have $\{(-\ 1,\ \lambda y.\ twoCube\ (0,\ y)),\ (1{::}int,\ \lambda y.\ twoCube\ (1,\ y))\}\ \cap$
$\{(1,\ \lambda y.\ twoCube2\ (y,\ 0)),\ (-\ 1,\ \lambda y.\ twoCube2\ (y,\ 1))\} = \{\}$
if *twoCube* ∈ *two-chain twoCube2* ∈ *two-chain* **for** *twoCube twoCube2*
proof (*cases twoCube* = *twoCube2*)
case *True*
have *card* $\{(-\ 1{::}int,\ \lambda y.\ twoCube2\ (0,\ y)),\ (1{::}int,\ \lambda y.\ twoCube2\ (1,\ y)),\ (1,\ \lambda x.\ twoCube2\ (x,\ 0)),\ (-\ 1,\ \lambda x.\ twoCube2\ (x,\ 1))\} = 4$
using *all-two-cubes-have-four-distict-edges that*(*2*)
by (*auto simp add*: *boundary-def vertical-boundary-def horizontal-boundary-def*)
then show *?thesis*
by (*auto simp*: *card-insert-if True split*: *if-split-asm*)
next
case *False*
then show *?thesis*
using *no-shared-edges-have-similar-orientations*
by (*simp add*: *that boundary-def vertical-boundary-def horizontal-boundary-def*)
qed
then have $\bigcup\ ((\lambda twoCube.\ \{(-\ 1,\ \lambda y.\ twoCube\ (0,\ y)),\ (1,\ \lambda y.\ twoCube\ (1,\ y))\})\ `\ two\text{-}chain)$
$\cap \bigcup\ ((\lambda twoCube.\ \{(1{::}int,\ \lambda y.\ twoCube\ (y,\ 0)),\ (-\ 1,\ \lambda y.\ twoCube\ (y,\ 1))\})\ `\ two\text{-}chain)$
$= \{\}$
using *Complete-Lattices.Union-disjoint* **by** *force*
then show *?thesis* **by** *force*
qed
qed
have *one-chain-line-integral F* {*i*} (*two-chain-boundary two-chain*)
= *one-chain-line-integral F* {*i*} (*two-chain-vertical-boundary two-chain*) +
one-chain-line-integral F {*i*} (*two-chain-horizontal-boundary two-chain*)
using *boundary-is-vert-hor horiz-verti-disjoint*
by (*auto simp add*: *one-chain-line-integral-def hor-vert-finite sum.union-disjoint*)

then have *x-axis-line-integral-is-only-horizontal*:
one-chain-line-integral F {*i*} (*two-chain-boundary two-chain*)
= *one-chain-line-integral F* {*i*} (*two-chain-horizontal-boundary two-chain*)
using *vert-line-integral-zero* **by** *auto*

obtain $\mathcal{H}$ $\mathcal{V}$ **where** *hv-props*: *finite* $\mathcal{V}$
(∀ (*k*,γ) ∈ $\mathcal{V}$. (∃ (*k'*, γ') ∈ *two-chain-vertical-boundary two-chain*.
(∃ *a* ∈ {*0* .. *1*}. ∃ *b* ∈ {*0*..*1*}. *a* ≤ *b* ∧ *subpath a b* γ' = γ))) γ = $\mathcal{H}$ ∪ $\mathcal{V}$
(*common-sudiv-exists* (*two-chain-horizontal-boundary two-chain*) $\mathcal{H}$)
∨ *common-reparam-exists* $\mathcal{H}$ (*two-chain-horizontal-boundary two-chain*)
finite $\mathcal{H}$
boundary-chain $\mathcal{H}$
∀ (*k*,γ)∈$\mathcal{H}$. *valid-path* γ
using *only-vertical-division* **by** (*auto simp add*: *only-vertical-division-def*)
have *finite* {*i*} **by** *auto*
then have *eq-integrals*: *one-chain-line-integral F* {*i*} $\mathcal{H}$ = *one-chain-line-integral F* {*i*} (*two-chain-horizontal-boundary two-chain*)
proof(*cases common-sudiv-exists* (*two-chain-horizontal-boundary two-chain*) $\mathcal{H}$)
case *True*
then show *?thesis*
using *gen-common-subdivision-imp-eq-line-integral*(*1*)[*OF True two-chain-horizontal-boundary-is-boundary*
hv-props(*6*) - *hor-vert-finite*(*2*) *hv-props*(*5*)]
typeI-cube-line-integral-exists-boundary''
by *force*
next
case *False*
have *integ-exis-horiz*:
∀ (*k*,γ) ∈ *two-chain-horizontal-boundary two-chain*. *line-integral-exists F* {*i*} γ
using *typeI-cube-line-integral-exists-boundary'' assms*
by (*fastforce simp add*: *valid-two-chain-def*)
have *integ-exis*: ∀ (*k*,γ) ∈ *two-chain-boundary two-chain*. *line-integral-exists F* {*i*} γ
using *typeI-cube-line-integral-exists-boundary* **by** *blast*
have *valid-paths*: ∀ (*k*,γ) ∈ *two-chain-vertical-boundary two-chain*. *valid-path* γ
using *typeI-edges-are-valid-paths assms*
by (*fastforce simp add*: *two-chain-boundary-def two-chain-vertical-boundary-def boundary-def*)
have *integ-exis-vert*:
(⋀*k* γ. (∃ (*k'*, γ') ∈ *two-chain-vertical-boundary two-chain*. ∃ *a*∈{*0*..*1*}. ∃ *b*∈{*0*..*1*}. *a* ≤ *b* ∧ *subpath a b* γ' = γ) ⟹
line-integral-exists F {*i*} γ)
using *integ-exis valid-paths line-integral-exists-subpath*[*of F* {*i*}]
by (*fastforce simp add*: *two-chain-boundary-def two-chain-vertical-boundary-def two-chain-horizontal-boundary-def boundary-def*)
have *finite-i*: *finite* {*i*} **by** *auto*
have *i*: {*i*} ⊆ *Basis* **using** *i-is-x-axis real-pair-basis* **by** *auto*
have *ii*: ∀ (*k2*, γ*2*)∈*two-chain-horizontal-boundary two-chain*. ∀ *b*∈{*i*}. *continuous-on* (*path-image* γ*2*) (λ*x*. *F x* · *b*)

using *assms*(*1*) *field-cont-on-typeI-region-cont-on-edges assms*(*2*)
by (*fastforce simp add*: *analytically-valid-def two-chain-horizontal-boundary-def boundary-def path-image-def*)
have ∗: *common-reparam-exists* $\mathcal{H}$ (*two-chain-horizontal-boundary two-chain*)
using *hv-props*(*4*) *False* **by** *auto*
show *one-chain-line-integral F* {*i*} $\mathcal{H}$ = *one-chain-line-integral F* {*i*} (*two-chain-horizontal-boundary two-chain*)
using *common-reparam-exists-imp-eq-line-integral*(*1*)[*OF finite-i hv-props*(*5*) *hor-vert-finite*(*2*) *hv-props*(*6*) *two-chain-horizontal-boundary-is-boundary-chain ii* ∗ *hv-props*(*7*) *type-I-chain-horiz-bound-valid*]
by *fastforce*
qed

have *line-integral-on-path*:
one-chain-line-integral F {*i*} γ =
one-chain-line-integral F {*i*} (*two-chain-horizontal-boundary two-chain*)
proof (*auto simp add*: *one-chain-line-integral-def*)
have *line-integral F* {*i*} (*snd oneCube*) = *0* **if** *oneCube*: *oneCube* ∈ $\mathcal{V}$ **for** *oneCube*
proof −
obtain *k* γ **where** *k-gamma*: (*k*,γ) = *oneCube*
by (*metis coeff-cube-to-path.cases*)
then obtain *k′* γ′ *a b* **where** *kp-gammap*:
(*k′*::*int*, γ′) ∈ *two-chain-vertical-boundary two-chain*
a ∈ {*0* .. *1*}
b ∈ {*0*..*1*}
subpath a b γ′ = γ
using *hv-props oneCube*
by (*smt case-prodE split-conv*)
obtain *x y1 y2* **where** *vert-edge-def*: (*k′*,γ′) = (*k′*, (λ*t*::*real*. (*x*::*real*, (*1* − *t*) ∗ (*y2*) + *t* ∗ *y1*)))
using *valid-typeI-div kp-gammap*
by (*auto simp add*: *typeI-twoCube-def two-chain-vertical-boundary-def vertical-boundary-def*)
have *vert-edge-x-const*: ∀ *t*. γ (*t*) · *i* = *x*
by (*metis* (*no-types*, *lifting*) *Pair-inject fstI i-is-x-axis inner-Pair-0*(*2*) *kp-gammap*(*4*) *real-inner-1-right subpath-def vert-edge-def*)
have γ = (λ*t*::*real*. (*x*::*real*, (*1* − (*b* − *a*)∗*t* − *a*) ∗ (*y2*) + ((*b*−*a*)∗*t* + *a*) ∗ *y1*))
using *vert-edge-def Product-Type.snd-conv Product-Type.fst-conv kp-gammap*(*4*)
by (*simp add*: *subpath-def diff-diff-eq*[*symmetric*])
also have ... = (λ*t*::*real*. (*x*::*real*, (*1*∗*y2* − *a*∗*y2*) + *a*∗*y1* + ((*b*−*a*)∗*y1* − (*b* − *a*)∗*y2*)∗*t*))
by (*simp add*: *algebra-simps*)
finally have *vert-edge-is-straight-path*:
γ = (λ*t*::*real*. (*x*::*real*, (*1*∗*y2* − *a*∗*y2*) + *a*∗*y1* + ((*b*−*a*)∗*y1* − (*b* − *a*)∗*y2*)∗*t*)) **.**
show *line-integral F* {*i*} (*snd oneCube*) = *0*
proof −

have ∀ *x*. *γ differentiable at x*
by (*simp add*: *straight-path-diffrentiable-x vert-edge-is-straight-path*)
then have *line-integral F* {*i*} *γ* = *0*
using *line-integral-on-pair-straight-path*(*1*) *vert-edge-x-const* **by** *blast*
then show *?thesis*
using *Product-Type.snd-conv k-gamma* **by** *auto*
qed
qed
then have ∀ *x*∈𝒱. (*case x of* (*k*, *g*) ⇒ (*k*::*int*) ∗ *line-integral F* {*i*} *g*) = *0*
by *auto*
then show (∑ *x*∈*γ*. *case x of* (*k*, *g*) ⇒ *real-of-int k* ∗ *line-integral F* {*i*} *g*) =
(∑ *x*∈*two-chain-horizontal-boundary two-chain*. *case x of* (*k*, *g*) ⇒ *of-int k* ∗ *line-integral F* {*i*} *g*)
using *hv-props*(*1*) *hv-props*(*3*) *hv-props*(*5*) *sum-zero-set hor-vert-finite*(*2*) *eq-integrals*
apply(*auto simp add*: *one-chain-line-integral-def*)
by (*smt Un-commute sum-zero-set*)
qed
then have *one-chain-line-integral F* {*i*} *γ* =
one-chain-line-integral F {*i*} (*two-chain-boundary two-chain*)
using *x-axis-line-integral-is-only-horizontal line-integral-on-path* **by** *auto*
then show *?thesis*
using *assms GreenThm-typeI-divisible* **by** *auto*
qed

end

locale *green-typeI-typeII-chain* = *R2*: *R2 i j* + *T1*: *green-typeI-chain i j F two-chain-typeI* + *T2*: *green-typeII-chain i j F two-chain-typeII* **for** *i j F two-chain-typeI two-chain-typeII*
begin

lemma *GreenThm-typeI-typeII-divisible-region-boundary*:
assumes
gen-divisions: *gen-division s* (*cubeImage* ‘ *two-chain-typeI*)
gen-division s (*cubeImage* ‘ *two-chain-typeII*) **and**
typeI-two-cubes-trace-horizontal-boundaries:
two-chain-horizontal-boundary two-chain-typeI ⊆ *γ* **and**
typeII-two-cubes-trace-vertical-boundaries:
two-chain-vertical-boundary two-chain-typeII ⊆ *γ* **and**
boundary-of-region-is-subset-of-partition-boundaries:
γ ⊆ *two-chain-boundary two-chain-typeI*
γ ⊆ *two-chain-boundary two-chain-typeII*
shows *integral s* (*λx*. *partial-vector-derivative* (*λa*. *F a* · *j*) *i x* − *partial-vector-derivative* (*λa*. *F a* · *i*) *j x*)
= *one-chain-line-integral F* {*i*, *j*} *γ*
proof −
let *?F-b′* = *partial-vector-derivative* (*λa*. *F a* · *i*) *j*
let *?F-a′* = *partial-vector-derivative* (*λa*. *F a* · *j*) *i*
have *typeI-regions-integral*: *integral s* (*λx*. − *partial-vector-derivative* (*λa*. *F a* ·

i) *j x*) = *one-chain-line-integral F* {*i*} *γ*
using *T1.GreenThm-typeI-divisible-region-boundary*
gen-divisions(*1*) *typeI-two-cubes-trace-horizontal-boundaries*
boundary-of-region-is-subset-of-partition-boundaries(*1*)
by *blast*
have *typeII-regions-integral*: *integral s* (*partial-vector-derivative* (*λx. F x · j*) *i*) = *one-chain-line-integral F* {*j*} *γ*
using *T2.GreenThm-typeII-divisible-region-boundary gen-divisions*(*2*)
typeII-two-cubes-trace-vertical-boundaries
boundary-of-region-is-subset-of-partition-boundaries(*2*)
by *auto*
have *integral-dis*: *integral s* (*λx. ?F-a′ x − ?F-b′ x*) = *integral s* (*λx. ?F-a′ x*) + *integral s* (*λx. − ?F-b′ x*)
proof −
have ∀ *twoCube* ∈ *two-chain-typeII*. (*?F-a′ has-integral integral* (*cubeImage twoCube*) *?F-a′*) (*cubeImage twoCube*)
by (*simp add*: *analytically-valid-imp-part-deriv-integrable-on T2.F-anal-valid has-integral-iff*)
then have ⋀*u. u* ∈ (*cubeImage ‘ two-chain-typeII*) ⟹ (*?F-a′ has-integral integral u ?F-a′*) *u*
by *auto*
then have (*?F-a′ has-integral* (∑ *img*∈*cubeImage ‘ two-chain-typeII. integral img ?F-a′*)) *s*
using *gen-divisions*(*2*) **unfolding** *gen-division-def*
by (*metis has-integral-Union*)
then have *F-a′-integrable*: (*?F-a′ integrable-on s*) **by** *auto*
have ∀ *twoCube* ∈ *two-chain-typeI*. (*?F-b′ has-integral integral* (*cubeImage twoCube*) *?F-b′*) (*cubeImage twoCube*)
using *analytically-valid-imp-part-deriv-integrable-on T1.F-anal-valid* **by** *blast*
then have ⋀*u. u* ∈ (*cubeImage ‘ two-chain-typeI*) ⟹ (*?F-b′ has-integral integral u ?F-b′*) *u*
by *auto*
then have (*?F-b′ has-integral* (∑ *img*∈*cubeImage ‘ two-chain-typeI. integral img ?F-b′*)) *s*
using *gen-divisions*(*1*) **unfolding** *gen-division-def*
by (*metis has-integral-Union*)
then show *?thesis*
by (*simp add*: *F-a′-integrable Henstock-Kurzweil-Integration.integral-diff has-integral-iff*)
qed
have *line-integral-dist*: *one-chain-line-integral F* {*i, j*} *γ* = *one-chain-line-integral F* {*i*} *γ* + *one-chain-line-integral F* {*j*} *γ*
proof (*simp add*: *one-chain-line-integral-def*)
have *k* ∗ *line-integral F* {*i, j*} *g* = *k* ∗ *line-integral F* {*i*} *g* + *k* ∗ *line-integral F* {*j*} *g*
if *kg*: (*k*,*g*) ∈ *γ* **for** *k g*
proof −
obtain *twoCube-typeI* **where** *twoCube-typeI-props*:
twoCube-typeI ∈ *two-chain-typeI*
(*k, g*) ∈ *boundary twoCube-typeI*

typeI-twoCube twoCube-typeI
continuous-on (cubeImage twoCube-typeI) (λx. F(x) · i)
using *boundary-of-region-is-subset-of-partition-boundaries(1) two-chain-boundary-def T1.valid-typeI-div*
T1.F-anal-valid kg
by (*auto simp add*: *analytically-valid-def*)
obtain *twoCube-typeII* **where** *twoCube-typeII-props*:
twoCube-typeII ∈ *two-chain-typeII*
(*k*, *g*) ∈ *boundary twoCube-typeII*
typeII-twoCube twoCube-typeII
continuous-on (*cubeImage twoCube-typeII*) (λ*x*. *F*(*x*) · *j*)
using *boundary-of-region-is-subset-of-partition-boundaries(2) two-chain-boundary-def T2.valid-typeII-div*
kg T2.F-anal-valid
by (*auto simp add*: *analytically-valid-def*)
have *line-integral F* {*i*, *j*} *g* = *line-integral F* {*i*} *g* + *line-integral F* {*j*} *g*
proof −
have *int-exists-i*: *line-integral-exists F* {*i*} *g*
using *T1.typeI-cube-line-integral-exists-boundary assms kg*
by (*auto simp add*: *valid-two-chain-def*)
have *int-exists-j*: *line-integral-exists F* {*j*} *g*
using *T2.typeII-cube-line-integral-exists-boundary assms kg*
by (*auto simp add*: *valid-two-chain-def*)
have *finite*: *finite* {*i*, *j*} **by** *auto*
show *?thesis*
using *line-integral-sum-gen*[*OF finite int-exists-i int-exists-j*] *R2.i-is-x-axis R2.j-is-y-axis*
by *auto*
qed
then show *k* ∗ *line-integral F* {*i*, *j*} *g* = *k* ∗ *line-integral F* {*i*} *g* + *k* ∗ *line-integral F* {*j*} *g*
by (*simp add*: *distrib-left*)
qed
then have *line-integral-distrib*:
(∑(*k*,*g*)∈γ. *k* ∗ *line-integral F* {*i*, *j*} *g*) =
(∑(*k*,*g*)∈γ. *k* ∗ *line-integral F* {*i*} *g* + *k* ∗ *line-integral F* {*j*} *g*)
by (*force intro*: *sum.cong split-cong*)
have (λ*x*. (*case x of* (*k*, *g*) ⇒ (*k*::*int*) ∗ *line-integral F* {*i*} *g*) + (*case x of* (*k*, *g*) ⇒ (*k*::*int*) ∗ *line-integral F* {*j*} *g*)) =
(λ*x*. (*case x of* (*k*, *g*) ⇒ (*k* ∗ *line-integral F* {*i*} *g*) + (*k*::*int*) ∗ *line-integral F* {*j*} *g*))
using *comm-monoid-add-class.sum.distrib* **by** *auto*
then show (∑(*k*, *g*)∈γ. *k* ∗ *line-integral F* {*i*, *j*} *g*) =
(∑(*k*, *g*)∈γ. (*k*::*int*) ∗ *line-integral F* {*i*} *g*) + (∑(*k*, *g*)∈γ. (*k*::*int*) ∗ *line-integral F* {*j*} *g*)
using *comm-monoid-add-class.sum.distrib*[*of* (λ(*k*, *g*). *k* ∗ *line-integral F* {*i*} *g*) (λ(*k*, *g*). *k* ∗ *line-integral F* {*j*} *g*) γ]
line-integral-distrib
by *presburger*

qed
show *?thesis*
using *integral-dis line-integral-dist typeI-regions-integral typeII-regions-integral*
by *auto*
qed

lemma *GreenThm-typeI-typeII-divisible-region′*:
assumes
only-vertical-division:
only-vertical-division one-chain-typeI two-chain-typeI
boundary-chain one-chain-typeI **and**
only-horizontal-division:
only-horizontal-division one-chain-typeII two-chain-typeII
boundary-chain one-chain-typeII **and**
typeI-and-typII-one-chains-have-gen-common-subdiv:
common-sudiv-exists one-chain-typeI one-chain-typeII
shows *integral s* (λx. *partial-vector-derivative* (λx. (F x) $\cdot$ j) i x − *partial-vector-derivative* (λx. (F x) $\cdot$ i) j x) = *one-chain-line-integral* F {i, j} *one-chain-typeI*
integral s (λx. *partial-vector-derivative* (λx. (F x) $\cdot$ j) i x − *partial-vector-derivative* (λx. (F x) $\cdot$ i) j x) = *one-chain-line-integral* F {i, j} *one-chain-typeII*
proof −
let *?F-b′* = *partial-vector-derivative* (λx. (F x) $\cdot$ i) j
let *?F-a′* = *partial-vector-derivative* (λx. (F x) $\cdot$ j) i
have *one-chain-i-integrals*:
one-chain-line-integral F {i} *one-chain-typeI* = *one-chain-line-integral* F {i} *one-chain-typeII* ∧
(∀ (k,γ)∈*one-chain-typeI*. *line-integral-exists* F {i} γ) ∧
(∀ (k,γ)∈*one-chain-typeII*. *line-integral-exists* F {i} γ)
proof (*intro conjI*)
have *finite two-chain-typeI*
using *T1.valid-typeI-div finite-image-iff*
by (*auto simp add*: *gen-division-def valid-two-chain-def*)
then show *ii*: ∀ (k, γ)∈*one-chain-typeI*. *line-integral-exists* F {i} γ
using *T1.members-of-only-vertical-div-line-integrable′ assms*
by *fastforce*
have *finite* (*two-chain-horizontal-boundary two-chain-typeI*)
by (*meson T1.valid-typeI-div finite-imageD finite-two-chain-horizontal-boundary gen-division-def valid-two-chain-def*)
then have *finite one-chain-typeI*
using *only-vertical-division*(*1*) *only-vertical-division-def* **by** *auto*
moreover have *finite one-chain-typeII*
using *only-horizontal-division*(*1*) *only-horizontal-division-def* **by** *auto*
ultimately show *one-chain-line-integral* F {i} *one-chain-typeI* = *one-chain-line-integral* F {i} *one-chain-typeII*
and ∀ (k, γ)∈*one-chain-typeII*. *line-integral-exists* F {i} γ
using *gen-common-subdivision-imp-eq-line-integral*[*OF typeI-and-typII-one-chains-have-gen-common-subdi*
only-vertical-division(*2*) *only-horizontal-division*(*2*)] *ii*
by *auto*
qed

have *one-chain-j-integrals*:
one-chain-line-integral F {*j*} *one-chain-typeII* = *one-chain-line-integral F* {*j*} *one-chain-typeI* ∧
(∀ (*k*,γ)∈*one-chain-typeII*. *line-integral-exists F* {*j*} γ) ∧
(∀ (*k*,γ)∈*one-chain-typeI*. *line-integral-exists F* {*j*} γ)
proof (*intro conjI*)
have *finite two-chain-typeII*
using *T2.valid-typeII-div finite-image-iff*
by (*auto simp add*: *gen-division-def valid-two-chain-def*)
then show *ii*: ∀ (*k*,γ)∈*one-chain-typeII*. *line-integral-exists F* {*j*} γ
using *T2.members-of-only-horiz-div-line-integrable′ assms T2.two-chain-valid-valid-cubes* **by** *blast*
have *typeII-and-typI-one-chains-have-common-subdiv*: *common-sudiv-exists one-chain-typeII one-chain-typeI*
by (*simp add*: *common-sudiv-exists-comm typeI-and-typII-one-chains-have-gen-common-subdiv*)
have *iv*: *finite one-chain-typeI*
using *only-vertical-division*(*1*) *only-vertical-division-def* **by** *auto*
moreover have *iv′*: *finite one-chain-typeII*
using *only-horizontal-division*(*1*) *only-horizontal-division-def* **by** *auto*
ultimately show *one-chain-line-integral F* {*j*} *one-chain-typeII* =
one-chain-line-integral F {*j*} *one-chain-typeI*
∀ (*k*, γ)∈*one-chain-typeI*. *line-integral-exists F* {*j*} γ
using *gen-common-subdivision-imp-eq-line-integral*[*OF typeII-and-typI-one-chains-have-common-subdiv only-horizontal-division*(*2*) *only-vertical-division*(*2*) *ii*] *ii*
by *auto*
qed
have *typeI-regions-integral*:
integral s (λ*x*. − *?F-b′ x*) = *one-chain-line-integral F* {*i*} *one-chain-typeI*
using *T1.GreenThm-typeI-divisible-region-boundary-gen T1.valid-typeI-div T1.F-anal-valid only-vertical-division*(*1*)
by *auto*
have *typeII-regions-integral*:
integral s ?F-a′ = *one-chain-line-integral F* {*j*} *one-chain-typeII*
using *T2.GreenThm-typeII-divisible-region-boundary-gen T2.valid-typeII-div T2.F-anal-valid only-horizontal-division*(*1*)
by *auto*
have *line-integral-dist*:
one-chain-line-integral F {*i*, *j*} *one-chain-typeI* = *one-chain-line-integral F* {*i*} *one-chain-typeI* + *one-chain-line-integral F* {*j*} *one-chain-typeI* ∧
one-chain-line-integral F {*i*, *j*} *one-chain-typeII* = *one-chain-line-integral F* {*i*} *one-chain-typeII* + *one-chain-line-integral F* {*j*} *one-chain-typeII*
proof (*simp add*: *one-chain-line-integral-def*)
have *line-integral-distrib*:
(∑ (*k*,*g*)∈*one-chain-typeI*. *k* ∗ *line-integral F* {*i*, *j*} *g*) =
(∑ (*k*,*g*)∈*one-chain-typeI*. *k* ∗ *line-integral F* {*i*} *g* + *k* ∗ *line-integral F* {*j*} *g*) ∧
(∑ (*k*,*g*)∈*one-chain-typeII*. *k* ∗ *line-integral F* {*i*, *j*} *g*) =
(∑ (*k*,*g*)∈*one-chain-typeII*. *k* ∗ *line-integral F* {*i*} *g* + *k* ∗ *line-integral F* {*j*} *g*)

proof −
have *0*: *k* ∗ *line-integral F* {*i*, *j*} *g* = *k* ∗ *line-integral F* {*i*} *g* + *k* ∗ *line-integral F* {*j*} *g*
if (*k*,*g*) ∈ *one-chain-typeII* **for** *k g*
proof −
have *line-integral-exists F* {*i*} *g line-integral-exists F* {*j*} *g finite* {*i*, *j*}
using *one-chain-i-integrals one-chain-j-integrals that* **by** *fastforce+*
moreover have {*i*} ∩ {*j*} = {}
by (*simp add*: *R2.i-is-x-axis R2.j-is-y-axis*)
ultimately have *line-integral F* {*i*, *j*} *g* = *line-integral F* {*i*} *g* + *line-integral F* {*j*} *g*
by (*metis insert-is-Un line-integral-sum-gen*(*1*))
then show *k* ∗ *line-integral F* {*i*, *j*} *g* = *k* ∗ *line-integral F* {*i*} *g* + *k* ∗ *line-integral F* {*j*} *g*
by (*simp add*: *distrib-left*)
qed
have *k* ∗ *line-integral F* {*i*, *j*} *g* = *k* ∗ *line-integral F* {*i*} *g* + *k* ∗ *line-integral F* {*j*} *g*
if (*k*,*g*) ∈ *one-chain-typeI* **for** *k g*
proof −
have *line-integral F* {*i*, *j*} *g* = *line-integral F* {*i*} *g* + *line-integral F* {*j*} *g*
by (*smt that disjoint-insert*(*2*) *finite.emptyI finite.insertI R2.i-is-x-axis inf-bot-right insert-absorb insert-commute insert-is-Un R2.j-is-y-axis line-integral-sum-gen*(*1*) *one-chain-i-integrals one-chain-j-integrals prod.case-eq-if singleton-inject snd-conv*)
then show *k* ∗ *line-integral F* {*i*, *j*} *g* = *k* ∗ *line-integral F* {*i*} *g* + *k* ∗ *line-integral F* {*j*} *g*
by (*simp add*: *distrib-left*)
qed
then show *?thesis*
using *0* **by** (*smt sum.cong split-cong*)
qed
show (∑(*k*::*int*, *g*)∈*one-chain-typeI*. *k* ∗ *line-integral F* {*i*, *j*} *g*) =
(∑(*k*, *g*)∈*one-chain-typeI*. *k* ∗ *line-integral F* {*i*} *g*) + (∑(*k*::*int*, *g*)∈*one-chain-typeI*. *k* ∗ *line-integral F* {*j*} *g*) ∧
(∑(*k*::*int*, *g*)∈*one-chain-typeII*. *k* ∗ *line-integral F* {*i*, *j*} *g*) =
(∑(*k*, *g*)∈*one-chain-typeII*. *k* ∗ *line-integral F* {*i*} *g*) + (∑(*k*::*int*, *g*)∈*one-chain-typeII*. *k* ∗ *line-integral F* {*j*} *g*)
proof −
have *0*: (λ*x*. (*case x of* (*k*::*int*, *g*) ⇒ *k* ∗ *line-integral F* {*i*} *g*) + (*case x of* (*k*::*int*, *g*) ⇒ *k* ∗ *line-integral F* {*j*} *g*)) =
(λ*x*. (*case x of* (*k*::*int*, *g*) ⇒ (*k* ∗ *line-integral F* {*i*} *g*) + *k* ∗ *line-integral F* {*j*} *g*))
using *comm-monoid-add-class.sum.distrib* **by** *auto*
then have *1*: (∑*x*∈*one-chain-typeI*. (*case x of* (*k*::*int*, *g*) ⇒ *k* ∗ *line-integral F* {*i*} *g*) + (*case x of* (*k*::*int*, *g*) ⇒ *k* ∗ *line-integral F* {*j*} *g*)) =
(∑*x*∈*one-chain-typeI*. (*case x of* (*k*::*int*, *g*) ⇒(*k* ∗ *line-integral F* {*i*} *g* + *k* ∗ *line-integral F* {*j*} *g*)))
by *presburger*
have (∑*x*∈*one-chain-typeII*. (*case x of* (*k*, *g*) ⇒ *k* ∗ *line-integral F* {*i*} *g*)

+ (*case x of* (*k*, *g*) ⇒ *k* ∗ *line-integral F* {*j*} *g*)) =
(∑ *x*∈*one-chain-typeII*. (*case x of* (*k*, *g*) ⇒(*k* ∗ *line-integral F* {*i*} *g* + *k* ∗ *line-integral F* {*j*} *g*)))
using *0* **by** *presburger*
then show *?thesis*
using *sum.distrib*[*of* (λ(*k*, *g*). *k* ∗ *line-integral F* {*i*} *g*) (λ(*k*, *g*). *k* ∗ *line-integral F* {*j*} *g*) *one-chain-typeI*]
sum.distrib[*of* (λ(*k*, *g*). *k* ∗ *line-integral F* {*i*} *g*) (λ(*k*, *g*). *k* ∗ *line-integral F* {*j*} *g*) *one-chain-typeII*]
line-integral-distrib 1
by *auto*
qed
qed
have *integral-dis*: *integral s* (λ*x*. *?F-a′ x* − *?F-b′ x*) = *integral s* (λ*x*. *?F-a′ x*) + *integral s* (λ*x*. − *?F-b′ x*)
proof −
have (*?F-a′ has-integral integral* (*cubeImage twoCube*) *?F-a′*) (*cubeImage twoCube*)
if *twoCube* ∈ *two-chain-typeII* **for** *twoCube*
by (*simp add*: *analytically-valid-imp-part-deriv-integrable-on T2.F-anal-valid has-integral-integrable-integral that*)
then have ⋀*u*. *u* ∈ (*cubeImage* ‘ *two-chain-typeII*) ⟹ (*?F-a′ has-integral integral u ?F-a′*) *u*
by *auto*
then have (*?F-a′ has-integral* (∑ *img*∈*cubeImage* ‘ *two-chain-typeII*. *integral img ?F-a′*)) *s*
using *T2.valid-typeII-div* **unfolding** *gen-division-def*
by (*metis has-integral-Union*)
then have *F-a′-integrable*:
(*?F-a′ integrable-on s*) **by** *auto*
have ∀ *twoCube* ∈ *two-chain-typeI*. (*?F-b′ has-integral integral* (*cubeImage twoCube*) *?F-b′*) (*cubeImage twoCube*)
using *analytically-valid-imp-part-deriv-integrable-on T1.F-anal-valid* **by** *blast*
then have ⋀*u*. *u* ∈ (*cubeImage* ‘ *two-chain-typeI*) ⟹ (*?F-b′ has-integral integral u ?F-b′*) *u*
by *auto*
then have (*?F-b′ has-integral* (∑ *img*∈*cubeImage* ‘ *two-chain-typeI*. *integral img ?F-b′*)) *s*
using *T1.valid-typeI-div* **unfolding** *gen-division-def*
by (*metis has-integral-Union*)
then show *?thesis*
by (*simp add*: *F-a′-integrable Henstock-Kurzweil-Integration.integral-diff has-integral-iff*)
qed
show *integral s* (λ*x*. *?F-a′ x* − *?F-b′ x*) = *one-chain-line-integral F* {*i*, *j*} *one-chain-typeI*
using *one-chain-j-integrals integral-dis line-integral-dist typeI-regions-integral typeII-regions-integral*
by *auto*
show *integral s* (λ*x*. *?F-a′ x* − *?F-b′ x*) = *one-chain-line-integral F* {*i*, *j*} *one-chain-typeII*

using *one-chain-i-integrals integral-dis line-integral-dist typeI-regions-integral typeII-regions-integral*
by *auto*
qed

lemma *GreenThm-typeI-typeII-divisible-region*:
assumes *only-vertical-division*:
only-vertical-division one-chain-typeI two-chain-typeI
boundary-chain one-chain-typeI **and**
only-horizontal-division:
only-horizontal-division one-chain-typeII two-chain-typeII
boundary-chain one-chain-typeII **and**
typeI-and-typII-one-chains-have-common-subdiv:
common-boundary-sudivision-exists one-chain-typeI one-chain-typeII
shows *integral s* (λx. *partial-vector-derivative* (λx. ($F\ x$) $\cdot$ j) $i\ x$ − *partial-vector-derivative* (λx. ($F\ x$) $\cdot$ i) $j\ x$) = *one-chain-line-integral* F {i, j} *one-chain-typeI*
integral s (λx. *partial-vector-derivative* (λx. ($F\ x$) $\cdot$ j) $i\ x$ − *partial-vector-derivative* (λx. ($F\ x$) $\cdot$ i) $j\ x$) = *one-chain-line-integral* F {i, j} *one-chain-typeII*
using *GreenThm-typeI-typeII-divisible-region′ only-vertical-division only-horizontal-division common-subdiv-imp-gen-common-subdiv*[*OF typeI-and-typII-one-chains-have-common-subdiv*]
by *auto*

lemma *GreenThm-typeI-typeII-divisible-region-finite-holes*:
assumes *valid-cube-boundary*: $\forall$ (k,γ)$\in$*boundary C*. *valid-path* γ **and**
only-vertical-division:
only-vertical-division (*boundary C*) *two-chain-typeI* **and**
only-horizontal-division:
only-horizontal-division (*boundary C*) *two-chain-typeII* **and**
s-is-oneCube: s = *cubeImage C*
shows *integral* (*cubeImage C*) (λx. *partial-vector-derivative* (λx. $F\ x \cdot j$) $i\ x$ − *partial-vector-derivative* (λx. $F\ x \cdot i$) $j\ x$) =
one-chain-line-integral F {i, j} (*boundary C*)
using *GreenThm-typeI-typeII-divisible-region*[*OF only-vertical-division*
two-cube-boundary-is-boundary only-horizontal-division two-cube-boundary-is-boundary
common-boundary-subdiv-exists-refl[*OF assms*(*1*)]] *s-is-oneCube*
by *auto*

lemma *GreenThm-typeI-typeII-divisible-region-equivallent-boundary*:
assumes
gen-divisions: *gen-division s* (*cubeImage* ‘ *two-chain-typeI*)
gen-division s (*cubeImage* ‘ *two-chain-typeII*) **and**
typeI-two-cubes-trace-horizontal-boundaries:
two-chain-horizontal-boundary two-chain-typeI $\subseteq$ *one-chain-typeI* **and**
typeII-two-cubes-trace-vertical-boundaries:
two-chain-vertical-boundary two-chain-typeII $\subseteq$ *one-chain-typeII* **and**
boundary-of-region-is-subset-of-partition-boundaries:
one-chain-typeI $\subseteq$ *two-chain-boundary two-chain-typeI*
one-chain-typeII $\subseteq$ *two-chain-boundary two-chain-typeII* **and**
typeI-and-typII-one-chains-have-common-subdiv:

common-boundary-sudivision-exists one-chain-typeI one-chain-typeII
shows *integral s* ($\lambda x.$ *partial-vector-derivative* ($\lambda x.$ ($F\ x$) $\cdot$ j) $i\ x$ − *partial-vector-derivative* ($\lambda x.$ ($F\ x$) $\cdot$ i) $j\ x$) = *one-chain-line-integral F* $\{i, j\}$ *one-chain-typeI*
integral s ($\lambda x.$ *partial-vector-derivative* ($\lambda x.$ ($F\ x$) $\cdot$ j) $i\ x$ − *partial-vector-derivative* ($\lambda x.$ ($F\ x$) $\cdot$ i) $j\ x$) = *one-chain-line-integral F* $\{i, j\}$ *one-chain-typeII*
proof −
let *?F-b′* = *partial-vector-derivative* ($\lambda x.$ ($F\ x$) $\cdot$ i) j
let *?F-a′* = *partial-vector-derivative* ($\lambda x.$ ($F\ x$) $\cdot$ j) i
have *one-chain-i-integrals*:
one-chain-line-integral F $\{i\}$ *one-chain-typeI* = *one-chain-line-integral F* $\{i\}$ *one-chain-typeII* ∧
(∀ (k,γ)∈*one-chain-typeI*. *line-integral-exists F* $\{i\}$ γ) ∧
(∀ (k,γ)∈*one-chain-typeII*. *line-integral-exists F* $\{i\}$ γ)
proof (*intro conjI*)
have *i*: *boundary-chain one-chain-typeI*
using *two-chain-boundary-is-boundary-chain boundary-chain-def boundary-of-region-is-subset-of-partition-boundaries*(*1*)
by *blast*
have *i′*: *boundary-chain one-chain-typeII*
using *two-chain-boundary-is-boundary-chain boundary-chain-def boundary-of-region-is-subset-of-partition-boundaries*(*2*)
by *blast*
have ⋀*k γ*. (k,γ)∈*one-chain-typeI* ⟹ *line-integral-exists F* $\{i\}$ γ
using *T1.typeI-cube-line-integral-exists-boundary assms*
by (*fastforce simp add*: *valid-two-chain-def*)
then show *ii*: ∀ (k,γ)∈*one-chain-typeI*. *line-integral-exists F* $\{i\}$ γ **by** *auto*
have *finite* (*two-chain-boundary two-chain-typeI*)
unfolding *two-chain-boundary-def*
proof (*rule finite-UN-I*)
show *finite two-chain-typeI*
using *T1.valid-typeI-div finite-imageD gen-division-def valid-two-chain-def* **by** *auto*
show ⋀*a*. *a* ∈ *two-chain-typeI* ⟹ *finite* (*boundary a*)
by (*simp add*: *boundary-def horizontal-boundary-def vertical-boundary-def*)
qed
then have *finite one-chain-typeI*
using *boundary-of-region-is-subset-of-partition-boundaries*(*1*) *finite-subset* **by** *fastforce*
moreover have *finite* (*two-chain-boundary two-chain-typeII*)
unfolding *two-chain-boundary-def*
proof (*rule finite-UN-I*)
show *finite two-chain-typeII*
using *T2.valid-typeII-div finite-imageD gen-division-def valid-two-chain-def* **by** *auto*
show ⋀*a*. *a* ∈ *two-chain-typeII* ⟹ *finite* (*boundary a*)
by (*simp add*: *boundary-def horizontal-boundary-def vertical-boundary-def*)
qed
then have *finite one-chain-typeII*
using *boundary-of-region-is-subset-of-partition-boundaries*(*2*) *finite-subset* **by**

fastforce
ultimately show *one-chain-line-integral F* $\{i\}$ *one-chain-typeI* = *one-chain-line-integral F* $\{i\}$ *one-chain-typeII*
$\forall (k, \gamma)\in$*one-chain-typeII. line-integral-exists F* $\{i\}$ γ
using *ii common-subdivision-imp-eq-line-integral*[*OF typeI-and-typII-one-chains-have-common-subdiv i i' ii*]
by *auto*
qed
have *one-chain-j-integrals*:
one-chain-line-integral F $\{j\}$ *one-chain-typeI* = *one-chain-line-integral F* $\{j\}$ *one-chain-typeII* $\wedge$
($\forall (k,\gamma)\in$*one-chain-typeI. line-integral-exists F* $\{j\}$ γ) $\wedge$
($\forall (k,\gamma)\in$*one-chain-typeII. line-integral-exists F* $\{j\}$ γ)
proof (*intro conjI*)
have *i*: *boundary-chain one-chain-typeI* **and** *i'*: *boundary-chain one-chain-typeII*
using *two-chain-boundary-is-boundary-chain boundary-of-region-is-subset-of-partition-boundaries*
unfolding *boundary-chain-def* **by** *blast+*
have *line-integral-exists F* $\{j\}$ γ **if** $(k,\gamma)\in$*one-chain-typeII* **for** *k* γ
proof −
have *F-is-continuous*: $\forall$ *twoC* $\in$ *two-chain-typeII. continuous-on* (*cubeImage twoC*) ($\lambda a.\ F(a) \cdot j$)
using *T2.F-anal-valid* **by**(*simp add*: *analytically-valid-def*)
show *line-integral-exists F* $\{j\}$ γ
using *that T2.valid-typeII-div boundary-of-region-is-subset-of-partition-boundaries*(*2*)
using *green-typeII-cube.line-integral-exists-on-typeII-Cube-boundaries' assms valid-two-chain-def*
apply (*simp add*: *two-chain-boundary-def*)
by (*metis T2.typeII-cube-line-integral-exists-boundary case-prodD subset-iff that two-chain-boundary-def*)
qed
then show *ii*: $\forall (k,\gamma)\in$*one-chain-typeII. line-integral-exists F* $\{j\}$ γ **by** *auto*
have *finite* (*two-chain-boundary two-chain-typeI*)
unfolding *two-chain-boundary-def*
proof (*rule finite-UN-I*)
show *finite two-chain-typeI*
using *T1.valid-typeI-div finite-imageD gen-division-def valid-two-chain-def* **by** *auto*
show $\bigwedge a.\ a \in$ *two-chain-typeI* $\Longrightarrow$ *finite* (*boundary a*)
by (*simp add*: *boundary-def horizontal-boundary-def vertical-boundary-def*)
qed
then have *iv*: *finite one-chain-typeI*
using *boundary-of-region-is-subset-of-partition-boundaries*(*1*) *finite-subset*
by *fastforce*
have *finite* (*two-chain-boundary two-chain-typeII*)
unfolding *two-chain-boundary-def*
proof (*rule finite-UN-I*)
show *finite two-chain-typeII*
using *T2.valid-typeII-div finite-imageD gen-division-def valid-two-chain-def*

by *auto*
show $\bigwedge a.\ a \in$ *two-chain-typeII* $\Longrightarrow$ *finite* (*boundary a*)
by (*simp add*: *boundary-def horizontal-boundary-def vertical-boundary-def*)
qed
then have *iv′*: *finite one-chain-typeII*
using *boundary-of-region-is-subset-of-partition-boundaries*(*2*) *finite-subset*
by *fastforce*
have *typeII-and-typI-one-chains-have-common-subdiv*:
common-boundary-sudivision-exists one-chain-typeII one-chain-typeI
using *typeI-and-typII-one-chains-have-common-subdiv*
common-boundary-sudivision-commutative
by *auto*
show *one-chain-line-integral F* $\{j\}$ *one-chain-typeI* = *one-chain-line-integral F* $\{j\}$ *one-chain-typeII*
$\forall (k, \gamma) \in$*one-chain-typeI*. *line-integral-exists F* $\{j\}$ γ
using *common-subdivision-imp-eq-line-integral*[*OF typeII-and-typI-one-chains-have-common-subdiv*
i′ i ii iv′ iv] *ii*
by *auto*
qed
have *typeI-regions-integral*:
integral s ($\lambda x.$ − *?F-b′ x*) = *one-chain-line-integral F* $\{i\}$ *one-chain-typeI*
using *T1.GreenThm-typeI-divisible-region-boundary gen-divisions*(*1*)
typeI-two-cubes-trace-horizontal-boundaries
boundary-of-region-is-subset-of-partition-boundaries(*1*)
by *auto*
have *typeII-regions-integral*:
integral s ?F-a′ = *one-chain-line-integral F* $\{j\}$ *one-chain-typeII*
using *T2.GreenThm-typeII-divisible-region-boundary gen-divisions*(*2*)
typeII-two-cubes-trace-vertical-boundaries
boundary-of-region-is-subset-of-partition-boundaries(*2*)
by *auto*
have *line-integral-dist*:
one-chain-line-integral F $\{i, j\}$ *one-chain-typeI* = *one-chain-line-integral F* $\{i\}$ *one-chain-typeI* + *one-chain-line-integral F* $\{j\}$ *one-chain-typeI* ∧
one-chain-line-integral F $\{i, j\}$ *one-chain-typeII* = *one-chain-line-integral F* $\{i\}$ *one-chain-typeII* + *one-chain-line-integral F* $\{j\}$ *one-chain-typeII*
proof (*simp add*: *one-chain-line-integral-def*)
have *line-integral-distrib*:
($\sum$(*k*,*g*)∈*one-chain-typeI*. *k* ∗ *line-integral F* $\{i, j\}$ *g*) =
($\sum$(*k*,*g*)∈*one-chain-typeI*. *k* ∗ *line-integral F* $\{i\}$ *g* + *k* ∗ *line-integral F* $\{j\}$ *g*) ∧
($\sum$(*k*,*g*)∈*one-chain-typeII*. *k* ∗ *line-integral F* $\{i, j\}$ *g*) =
($\sum$(*k*,*g*)∈*one-chain-typeII*. *k* ∗ *line-integral F* $\{i\}$ *g* + *k* ∗ *line-integral F* $\{j\}$ *g*)
proof −
have *0*: *k* ∗ *line-integral F* $\{i, j\}$ *g* = *k* ∗ *line-integral F* $\{i\}$ *g* + *k* ∗ *line-integral F* $\{j\}$ *g*
if (*k*,*g*) ∈ *one-chain-typeII* **for** *k g*
proof −

have *line-integral F* {*i*, *j*} *g* = *line-integral F* {*i*} *g* + *line-integral F* {*j*} *g*
proof −
have *finite*: *finite* {*i*, *j*} **by** *auto*
have *line-integral-all*: ∀ *i*∈{*i*, *j*}. *line-integral-exists F* {*i*} *g*
using *one-chain-i-integrals one-chain-j-integrals that* **by** *auto*
show *?thesis*
using *line-integral-sum-gen*[*OF finite*] *R2.i-is-x-axis R2.j-is-y-axis line-integral-all* **by** *auto*
qed
then show *k* ∗ *line-integral F* {*i*, *j*} *g* = *k* ∗ *line-integral F* {*i*} *g* + *k* ∗ *line-integral F* {*j*} *g*
by (*simp add*: *distrib-left*)
qed
have *k* ∗ *line-integral F* {*i*, *j*} *g* = *k* ∗ *line-integral F* {*i*} *g* + *k* ∗ *line-integral F* {*j*} *g*
if (*k*,*g*) ∈ *one-chain-typeI* **for** *k g*
proof −
have *finite*: *finite* {*i*, *j*} **by** *auto*
have *line-integral-all*: ∀ *i*∈{*i*, *j*}. *line-integral-exists F* {*i*} *g*
using *one-chain-i-integrals one-chain-j-integrals that* **by** *auto*
have *line-integral F* {*i*, *j*} *g* = *line-integral F* {*i*} *g* + *line-integral F* {*j*} *g*
using *line-integral-sum-gen*[*OF finite*] *R2.i-is-x-axis R2.j-is-y-axis line-integral-all* **by** *auto*
then show *k* ∗ *line-integral F* {*i*, *j*} *g* = *k* ∗ *line-integral F* {*i*} *g* + *k* ∗ *line-integral F* {*j*} *g*
by (*simp add*: *distrib-left*)
qed
then show *?thesis*
using *0* **by** (*smt sum.cong split-cong*)
qed
show (∑(*k*::*int*, *g*)∈*one-chain-typeI*. *k* ∗ *line-integral F* {*i*, *j*} *g*) =
(∑(*k*, *g*)∈*one-chain-typeI*. *k* ∗ *line-integral F* {*i*} *g*) + (∑(*k*::*int*, *g*)∈*one-chain-typeI*. *k* ∗ *line-integral F* {*j*} *g*) ∧
(∑(*k*::*int*, *g*)∈*one-chain-typeII*. *k* ∗ *line-integral F* {*i*, *j*} *g*) =
(∑(*k*, *g*)∈*one-chain-typeII*. *k* ∗ *line-integral F* {*i*} *g*) + (∑(*k*::*int*, *g*)∈*one-chain-typeII*. *k* ∗ *line-integral F* {*j*} *g*)
proof −
have *0*: (λ*x*. (*case x of* (*k*::*int*, *g*) ⇒ *k* ∗ *line-integral F* {*i*} *g*) + (*case x of* (*k*::*int*, *g*) ⇒ *k* ∗ *line-integral F* {*j*} *g*)) =
(λ*x*. (*case x of* (*k*::*int*, *g*) ⇒ (*k* ∗ *line-integral F* {*i*} *g*) + *k* ∗ *line-integral F* {*j*} *g*))
using *comm-monoid-add-class.sum.distrib* **by** *auto*
then have *1*: (∑*x*∈*one-chain-typeI*. (*case x of* (*k*::*int*, *g*) ⇒ *k* ∗ *line-integral F* {*i*} *g*) + (*case x of* (*k*::*int*, *g*) ⇒ *k* ∗ *line-integral F* {*j*} *g*)) =
(∑*x*∈*one-chain-typeI*. (*case x of* (*k*::*int*, *g*) ⇒(*k* ∗ *line-integral F* {*i*} *g* + *k* ∗ *line-integral F* {*j*} *g*)))
by *presburger*
have (∑*x*∈*one-chain-typeII*. (*case x of* (*k*, *g*) ⇒ *k* ∗ *line-integral F* {*i*} *g*) + (*case x of* (*k*, *g*) ⇒ *k* ∗ *line-integral F* {*j*} *g*)) =

($\sum$ *x*∈*one-chain-typeII*. (*case x of* (*k*, *g*) ⇒(*k* ∗ *line-integral F* {*i*} *g* + *k* ∗ *line-integral F* {*j*} *g*)))
using *0* **by** *presburger*
then show *?thesis*
using *sum.distrib*[*of* (λ(*k*, *g*). *k* ∗ *line-integral F* {*i*} *g*) (λ(*k*, *g*). *k* ∗ *line-integral F* {*j*} *g*) *one-chain-typeI*]
sum.distrib[*of* (λ(*k*, *g*). *k* ∗ *line-integral F* {*i*} *g*) (λ(*k*, *g*). *k* ∗ *line-integral F* {*j*} *g*) *one-chain-typeII*]
line-integral-distrib
1
by *auto*
qed
qed
have *integral-dis*: *integral s* (λ*x*. *?F-a′ x* − *?F-b′ x*) = *integral s* (λ*x*. *?F-a′ x*) + *integral s* (λ*x*. − *?F-b′ x*)
proof −
have (*?F-a′ has-integral* ($\sum$ *img*∈*cubeImage* ' *two-chain-typeII*. *integral img ?F-a′*)) *s*
proof −
have (*?F-a′ has-integral integral* (*cubeImage twoCube*) *?F-a′*) (*cubeImage twoCube*)
if *twoCube* ∈ *two-chain-typeII* **for** *twoCube*
by (*simp add*: *analytically-valid-imp-part-deriv-integrable-on T2.F-anal-valid has-integral-integrable-integral that*)
then have ⋀*u*. *u* ∈ (*cubeImage* ' *two-chain-typeII*) ⟹ (*?F-a′ has-integral integral u ?F-a′*) *u*
by *auto*
then show *?thesis*
using *gen-divisions*(*2*) **unfolding** *gen-division-def*
by (*metis has-integral-Union*)
qed
then have *F-a′-integrable*:
(*?F-a′ integrable-on s*) **by** *auto*
have (*?F-b′ has-integral* ($\sum$ *img*∈*cubeImage* ' *two-chain-typeI*. *integral img ?F-b′*)) *s*
proof −
have ∀ *twoCube* ∈ *two-chain-typeI*. (*?F-b′ has-integral integral* (*cubeImage twoCube*) *?F-b′*) (*cubeImage twoCube*)
by (*simp add*: *analytically-valid-imp-part-deriv-integrable-on T1.F-anal-valid has-integral-integrable-integral*)
then have ⋀*u*. *u* ∈ (*cubeImage* ' *two-chain-typeI*) ⟹ (*?F-b′ has-integral integral u ?F-b′*) *u*
by *auto*
then show *?thesis*
using *gen-divisions*(*1*) **unfolding** *gen-division-def*
by (*metis has-integral-Union*)
qed
then show *?thesis*
using *F-a′-integrable Henstock-Kurzweil-Integration.integral-diff* **by** *auto*

qed
show *integral s* (*λx*. *?F-a′ x* − *?F-b′ x*) = *one-chain-line-integral F* {*i*, *j*} *one-chain-typeI*
using *one-chain-j-integrals integral-dis line-integral-dist typeI-regions-integral typeII-regions-integral*
by *auto*
show *integral s* (*λx*. *?F-a′ x* − *?F-b′ x*) = *one-chain-line-integral F* {*i*, *j*} *one-chain-typeII*
using *one-chain-i-integrals integral-dis line-integral-dist typeI-regions-integral typeII-regions-integral*
by *auto*
qed

end
end
theory *SymmetricR2Shapes*
imports *Green*
begin

context *R2*
begin

lemma *valid-path-valid-swap*:
assumes *valid-path* (*λx*::*real*. ((*f x*)::*real*, (*g x*)::*real*))
shows *valid-path* (*prod.swap o* (*λx*. (*f x*, *g x*)))
unfolding *o-def valid-path-def piecewise-C1-differentiable-on-def swap-simp*
proof (*intro conjI*)
show *continuous-on* {*0*..*1*} (*λx*. (*g x*, *f x*))
using *assms*
using *continuous-on-Pair continuous-on-componentwise*[**where** *f* = (*λx*. (*f x*, *g x*))]
by (*auto simp add*: *real-pair-basis valid-path-def piecewise-C1-differentiable-on-def*)
show ∃ *S*. *finite S* ∧ (*λx*. (*g x*, *f x*)) *C1-differentiable-on* {*0*..*1*} − *S*
proof −
obtain *S* **where** *finite S* **and** *S*: (*λx*. (*f x*, *g x*)) *C1-differentiable-on* {*0*..*1*} − *S*
using *assms*
by (*auto simp add*: *real-pair-basis valid-path-def piecewise-C1-differentiable-on-def*)
have *0*: *f C1-differentiable-on* {*0*..*1*} − *S* **using** *S assms*
using *C1-diff-components-2*[*of* (*1*,*0*) (*λx*. (*f x*, *g x*))]
by (*auto simp add*: *real-pair-basis algebra-simps*)
have *1*: *g C1-differentiable-on* {*0*..*1*} − *S* **using** *S assms*
using *C1-diff-components-2* [*of* (*0*,*1*), *OF* - *S*] *real-pair-basis* **by** *fastforce*
have ∗: (*λx*. (*g x*, *f x*)) *C1-differentiable-on* {*0*..*1*} − *S*
using *0 1 C1-differentiable-on-components*[**where** *f* = (*λx*. (*g x*, *f x*))]
by (*auto simp add*: *real-pair-basis valid-path-def piecewise-C1-differentiable-on-def*)
then show *?thesis* **using** ‹*finite S*› **by** *auto*
qed
qed

lemma *pair-fun-components*: $C = (\lambda x.\ (C\ x \cdot i,\ C\ x \cdot j))$
  by (*simp add*: *i-is-x-axis inner-Pair-0 j-is-y-axis*)

lemma *swap-pair-fun*: $(\lambda y.\ prod.swap\ (C\ (y,\ 0))) = (\lambda x.\ (C\ (x,\ 0) \cdot j,\ C\ (x,\ 0) \cdot i))$
  by (*simp add*: *prod.swap-def i-is-x-axis inner-Pair-0 j-is-y-axis*)

lemma *swap-pair-fun'*: $(\lambda y.\ prod.swap\ (C\ (y,\ 1))) = (\lambda x.\ (C\ (x,\ 1) \cdot j,\ C\ (x,\ 1) \cdot i))$
  by (*simp add*: *prod.swap-def i-is-x-axis inner-Pair-0 j-is-y-axis*)

lemma *swap-pair-fun''*: $(\lambda y.\ prod.swap\ (C\ (0,\ y))) = (\lambda x.\ (C\ (0,x) \cdot j,\ C\ (0,x) \cdot i))$
  by (*simp add*: *prod.swap-def i-is-x-axis inner-Pair-0 j-is-y-axis*)

lemma *swap-pair-fun'''*: $(\lambda y.\ prod.swap\ (C\ (1,\ y))) = (\lambda x.\ (C\ (1,x) \cdot j,\ C\ (1,x) \cdot i))$
  by (*simp add*: *prod.swap-def i-is-x-axis inner-Pair-0 j-is-y-axis*)

lemma *swap-valid-boundaries*:
  assumes $\forall (k,\gamma) \in boundary\ C.\ valid\text{-}path\ \gamma$
  assumes $(k,\gamma) \in boundary\ (prod.swap\ o\ C\ o\ prod.swap)$
  shows *valid-path* γ
  using *assms*
    valid-path-valid-swap[*of* $\lambda x.\ (\lambda x.\ C\ (x,\ 0))\ x \cdot i\ \lambda x.\ (\lambda x.\ C\ (x,\ 0))\ x \cdot j$] *pair-fun-components*[*of* $(\lambda x.\ C\ (x,\ 0))$]
    pair-fun-components[*of* $(\lambda y.\ C\ (y,\ 0))$]
    valid-path-valid-swap[*of* $\lambda x.\ (\lambda y.\ C\ (y,\ 1))\ x \cdot i\ \lambda x.\ (\lambda y.\ C\ (y,\ 1))\ x \cdot j$] *pair-fun-components*[*of* $(\lambda y.\ C\ (y,\ 1))$]
    pair-fun-components[*of* $(\lambda x.\ C\ (x,\ 1))$]
    valid-path-valid-swap[*of* $\lambda x.\ (\lambda y.\ C\ (1,y))\ x \cdot i\ \lambda x.\ (\lambda y.\ C\ (1,y))\ x \cdot j$] *pair-fun-components*[*of* $(\lambda y.\ C\ (1,y))$]
    pair-fun-components[*of* $(\lambda x.\ C\ (1,x))$]
    valid-path-valid-swap[*of* $\lambda x.\ (\lambda y.\ C\ (0,y))\ x \cdot i\ \lambda x.\ (\lambda y.\ C\ (0,y))\ x \cdot j$] *pair-fun-components*[*of* $(\lambda y.\ C\ (0,y))$]
    pair-fun-components[*of* $(\lambda x.\ C\ (0,x))$]
  by (*auto simp add*: *boundary-def horizontal-boundary-def vertical-boundary-def o-def real-pair-basis swap-pair-fun swap-pair-fun' swap-pair-fun'' swap-pair-fun'''*)

lemma *prod-comp-eq*:
  assumes $f = prod.swap\ o\ g$
  shows $prod.swap\ o\ f = g$
  using *swap-comp-swap assms*
  by *fastforce*

lemma *swap-typeI-is-typeII*:
  assumes *typeI-twoCube C*
  shows *typeII-twoCube* (*prod.swap o C o prod.swap*)

```
proof (simp add: typeI-twoCube-def typeII-twoCube-def)
  obtain a b g1 g2 where C: a < b
    (∀x∈{a..b}. g2 x ≤ g1 x)
    cubeImage C = {(x, y). x ∈ {a..b} ∧ y ∈ {g2 x..g1 x}}
    C = (λ(x, y). ((1 − x) ∗ a + x ∗ b, (1 − y) ∗ g2 ((1 − x) ∗ a + x ∗ b) + y
∗ g1 ((1 − x) ∗ a + x ∗ b)))
    g1 piecewise-C1-differentiable-on {a..b}
    g2 piecewise-C1-differentiable-on {a..b}
    using typeI-cube-explicit-spec[OF assms]
    by blast
  show ∃a b. a < b ∧
            (∃g1 g2. (∀x∈{a..b}. g2 x ≤ g1 x) ∧
                prod.swap ∘ C ∘ prod.swap =
                (λ(y, x). ((1 − y) ∗ g2 ((1 − x) ∗ a + x ∗ b) + y ∗ g1 ((1 − x) ∗
a + x ∗ b), (1 − x) ∗ a + x ∗ b)) ∧
            g1 piecewise-C1-differentiable-on {a..b} ∧ g2 piecewise-C1-differentiable-on
{a..b})
    using C by (fastforce simp add: prod.swap-def o-def)
qed

lemma valid-cube-valid-swap:
  assumes valid-two-cube C
  shows valid-two-cube (prod.swap o C o prod.swap)
  using assms unfolding valid-two-cube-def boundary-def horizontal-boundary-def
vertical-boundary-def
  apply (auto simp: card-insert-if  split: if-split-asm)
   apply (metis swap-swap)+
  done

lemma twoChainVertDiv-of-itself:
  assumes finite C
    ∀(k, γ)∈(two-chain-boundary C). valid-path γ
  shows only-vertical-division (two-chain-boundary C) C
proof(clarsimp simp add: only-vertical-division-def)
  show ∃𝒱 ℋ. finite ℋ ∧ finite 𝒱 ∧
             (∀x∈𝒱. case x of (k, γ) ⇒ ∃x∈two-chain-vertical-boundary C. case x
of (k′, γ′) ⇒ ∃a∈{0..1}. ∃b∈{0..1}. a ≤ b ∧ subpath a b γ′ = γ) ∧
             (common-sudiv-exists (two-chain-horizontal-boundary C) ℋ ∨
              common-reparam-exists ℋ (two-chain-horizontal-boundary C)) ∧
               boundary-chain ℋ ∧ two-chain-boundary C = 𝒱 ∪ ℋ ∧ (∀(k,γ)∈ℋ.
valid-path γ)
  proof (intro exI)
    let ?ℋ = two-chain-horizontal-boundary C
    have 0: ∀(k,γ)∈?ℋ. valid-path γ using assms(2)
      by (auto simp add: two-chain-horizontal-boundary-def two-chain-boundary-def
boundary-def)
    have ⋀a b. (a, b) ∈ two-chain-vertical-boundary C ⟹
                ∃x∈two-chain-vertical-boundary C. case x of (k′, γ′) ⇒ ∃a∈{0..1}.
∃c∈{0..1}. a ≤ c ∧ subpath a c γ′ = b
```

by (*metis* (*mono-tags*, *lifting*) *atLeastAtMost-iff case-prod-conv le-numeral-extra*(*1*) *order-refl subpath-trivial*)
moreover have *common-sudiv-exists ?$\mathcal{H}$?$\mathcal{H}$*
using *gen-common-boundary-subdiv-exists-refl-twochain-boundary*[*OF 0 two-chain-horizontal-boundary-is-b*
by *auto*
moreover have *boundary-chain ?$\mathcal{H}$*
using *two-chain-horizontal-boundary-is-boundary-chain* **by** *auto*
moreover have $\bigwedge a\ b.\ (a, b) \in$ *two-chain-boundary C* $\Longrightarrow (a, b) \notin$ *?$\mathcal{H}$* $\Longrightarrow (a, b) \in$ *two-chain-vertical-boundary C*
by (*auto simp add*: *two-chain-boundary-def two-chain-horizontal-boundary-def two-chain-vertical-boundary-def boundary-def*)
moreover have $\bigwedge a\ b.\ (a, b) \in$ *two-chain-vertical-boundary C* $\Longrightarrow (a, b) \in$ *two-chain-boundary C*
$\bigwedge a\ b.\ (a, b) \in$ *?$\mathcal{H}$* $\Longrightarrow (a, b) \in$ *two-chain-boundary C*
by (*auto simp add*: *two-chain-boundary-def two-chain-horizontal-boundary-def two-chain-vertical-boundary-def boundary-def*)
moreover have $\bigwedge a\ b.\ (a, b) \in$ *?$\mathcal{H}$* $\Longrightarrow$ *valid-path b*
using *0* **by** *blast*
ultimately show *finite ?$\mathcal{H}$* $\wedge$
finite (*two-chain-vertical-boundary C*) $\wedge$
($\forall x \in$*two-chain-vertical-boundary C*.
case x of $(k, \gamma) \Rightarrow \exists x\in$*two-chain-vertical-boundary C*. *case x of* $(k', \gamma') \Rightarrow \exists a\in\{0..1\}.\ \exists b\in\{0..1\}.\ a \le b \wedge$ *subpath a b* $\gamma' = \gamma) \wedge$
(*common-sudiv-exists ?$\mathcal{H}$?$\mathcal{H}$* $\vee$
common-reparam-exists ?$\mathcal{H}$?$\mathcal{H}$) $\wedge$
boundary-chain ?$\mathcal{H}$ $\wedge$ *two-chain-boundary C* = *two-chain-vertical-boundary C* $\cup$ *?$\mathcal{H}$* $\wedge$ ($\forall (k,\gamma)\in$*?$\mathcal{H}$*. *valid-path* γ)
by (*auto simp add*: *finite-two-chain-horizontal-boundary*[*OF assms*(*1*)] *finite-two-chain-vertical-boundary*[*OF assms*(*1*)])
qed
qed

end

definition *x-coord* **where** *x-coord* $\equiv (\lambda t::real.\ t - 1/2)$

lemma *x-coord-smooth*: *x-coord C1-differentiable-on* $\{a..b\}$
by (*simp add*: *x-coord-def*)

lemma *x-coord-bounds*:
assumes $(0::real) \le x\ x \le 1$
shows $-1/2 \le$ *x-coord x* $\wedge$ *x-coord x* $\le 1/2$
using *assms* **by**(*auto simp add*: *x-coord-def*)

lemma *x-coord-img*: *x-coord* ` $\{(0::real)..1\} = \{-1/2\ ..\ 1/2\}$
by (*auto simp add*: *x-coord-def image-def algebra-simps*)

lemma *x-coord-back-img*: *finite* ($\{0..1\} \cap$ *x-coord* −` $\{x::real\}$)

by (*simp add*: *finite-vimageI inj-on-def x-coord-def*)

abbreviation *rot-x t1 t2* ≡ (*if* (*t1* − *1*/*2*) ≤ *0 then* (*2* ∗ *t2* − *1*) ∗ *t1* + *1*/*2* ::*real else 2* ∗ *t2* − *2* ∗ *t1* ∗ *t2* +*t1* −*1*/*2*::*real*)

lemma *rot-x-ivl*:
assumes *0* ≤ *x*
x ≤ *1*
0 ≤ *y*
y ≤ *1*
shows *0* ≤ *rot-x x y* ∧ *rot-x x y* ≤ *1*
proof −
have *i*: ⋀*a*::*real*. *a* ≤ *0* ⟹ *0* ≤ *y* ⟹ *y* ≤ *1* ⟹ −*1*/*2* < *a* ⟹ (*a* ∗ (*1* − *2*∗*y*) ≤ *1*/*2*)
proof −
have *0*: ⋀*a*::*real*. *a* ≤ *0* ⟹ *0* ≤ *y* ⟹ *y* ≤ *1* ⟹ −*1*/*2* < *a* ⟹ (−*a* ≤ *1*/*2*)
by (*sos* ((((*A*<*0* ∗ *A*<*1*) ∗ *R*<*1*) + (*R*<*1* ∗ (*R*<*1*/*4* ∗ [*2*∗*a* + *1*]^*2*)))))
have *1*: ⋀*a*. *a* ≤ *0* ⟹ *0* ≤ *y* ⟹ *y* ≤ *1* ⟹ −*1*/*2* < *a* ⟹ (*a* ∗ (*1* − *2*∗*y*) ≤ −*a*)
by (*sos* ((((*A*<*0* ∗ *A*<*1*) ∗ *R*<*1*) + (((*A*<=*0* ∗ (*A*<*1* ∗ *R*<*1*)) ∗ (*R*<*2*/*3* ∗ [*1*]^*2*)) + (((*A*<=*0* ∗ (*A*<=*2* ∗ *R*<*1*)) ∗ (*R*<*2*/*3* ∗ [*1*]^*2*)) + ((*A*<=*0* ∗ (*A*<=*2* ∗ (*A*<*0* ∗ *R*<*1*))) ∗ (*R*<*2*/*3* ∗ [*1*]^*2*)))))))
show ⋀*a*::*real*. *a* ≤ *0* ⟹ *0* ≤ *y* ⟹ *y* ≤ *1* ⟹ −*1*/*2* < *a* ⟹ (*a* ∗ (*1* − *2*∗*y*) ≤ *1*/*2*) **using** *0 1* **by** *force*
qed
have ∗: (*x* ∗ *2* + *y* ∗ *4* ≤ *3* + *x* ∗ (*y* ∗ *4*)) = ((*x*−*1*) ≤ *1*/*2*+ (*x*−*1*) ∗ (*y* ∗ *2*))
by (*sos* ((((*A*<*0* ∗ *R*<*1*) + ((*A*<=*0* ∗ *R*<*1*) ∗ (*R*<*2* ∗ [*1*]^*2*)))) & (((*A*<*0* ∗ *R*<*1*) + ((*A*<=*0* ∗ *R*<*1*) ∗ (*R*<*1*/*2* ∗ [*1*]^*2*))))))
show *?thesis*
using *assms*
apply(*auto simp add*: *algebra-simps divide-simps linorder-class.not-le*)
apply (*sos* (((*A*<*0* ∗ *R*<*1*) + (((*A*<=*2* ∗ (*A*<=*3* ∗ *R*<*1*)) ∗ (*R*<*1* ∗ [*1*]^*2*)) + (((*A*<=*1* ∗ *R*<*1*) ∗ (*R*<*1* ∗ [*1*]^*2*)) + ((*A*<=*0* ∗ (*A*<=*1* ∗ *R*<*1*)) ∗ (*R*<*2* ∗ [*1*]^*2*)))))))
apply (*sos* (((*A*<*0* ∗ *R*<*1*) + (((*A*<=*2* ∗ *R*<*1*) ∗ (*R*<*1* ∗ [*1*]^*2*)) + (((*A*<=*1* ∗ (*A*<=*3* ∗ *R*<*1*)) ∗ (*R*<*1* ∗ [*1*]^*2*)) + ((*A*<=*0* ∗ (*A*<=*2* ∗ *R*<*1*)) ∗ (*R*<*2* ∗ [*1*]^*2*)))))))
using *i*[*of* (*x*::*real*) − *1*] *affine-ineq*
apply (*fastforce simp*: *algebra-simps* ∗)+
done
qed

end

2 The Circle Example

theory *CircExample*

imports *Green SymmetricR2Shapes*

begin

locale *circle* = *R2* +
fixes *d*::*real*
assumes *d-gt-0*: *0* < *d*
begin

definition *circle-y* **where**
circle-y t = *sqrt* (*1/4* − *t* ∗ *t*)

definition *circle-cube* **where**
circle-cube = (λ(*x*,*y*). ((*x* − *1/2*) ∗ *d*, (*2* ∗ *y* − *1*) ∗ *d* ∗ *sqrt* (*1/4* − (*x* −*1/2*)∗(*x* −*1/2*))))

lemma *circle-cube-nice*:
shows *circle-cube* = (λ(*x*,*y*). (*d* ∗ *x-coord x*, (*2* ∗ *y* − *1*) ∗ *d* ∗ *circle-y* (*x-coord x*)))
by (*auto simp add*: *circle-cube-def circle-y-def x-coord-def*)

definition *rot-circle-cube* **where**
rot-circle-cube = *prod.swap* ∘ (*circle-cube*) ∘ *prod.swap*

abbreviation *rot-y t1 t2* ≡ ((*t1*−*1/2*)/(*2*∗ *circle-y* (*x-coord* (*rot-x t1 t2*))) +*1/2*::*real*)

definition *x-coord-inv* (*x*::*real*) = (*1/2*) + *x*

lemma *x-coord-inv-1*: *x-coord-inv* (*x-coord* (*x*::*real*)) = *x*
by (*auto simp add*: *x-coord-inv-def x-coord-def*)

lemma *x-coord-inv-2*: *x-coord* (*x-coord-inv* (*x*::*real*)) = *x*
by (*auto simp add*: *x-coord-inv-def x-coord-def*)

definition *circle-y-inv* = *circle-y*

abbreviation *rot-x''* (*x*::*real*) (*y*::*real*) ≡ (*x-coord-inv* ((*2* ∗ *y* − *1*) ∗ *circle-y* (*x-coord x*)))

lemma *circle-y-bounds*:
assumes −*1/2* ≤ (*x*::*real*) ∧ *x* ≤ *1/2*
shows *0* ≤ *circle-y x circle-y x* ≤ *1/2*
unfolding *circle-y-def real-sqrt-ge-0-iff*
proof −
show *0* ≤ *1/4* − *x* ∗ *x*
using *assms*
by (*sos* (((*A*<*0* ∗ *R*<*1*) + ((*A*<=*0* ∗ (*A*<=*1* ∗ *R*<*1*)) ∗ (*R*<*1* ∗ [*1*]^*2*)))))
show *sqrt* (*1/4* − *x* ∗ *x*) ≤ *1/2*
apply (*rule real-le-lsqrt*)

using *assms* **by**(*auto simp add*: *divide-simps algebra-simps*)
qed

lemma *circle-y-x-coord-bounds*:
assumes *0* ≤ (*x*::*real*) *x* ≤ *1*
shows *0* ≤ *circle-y* (*x-coord x*) ∧ *circle-y* (*x-coord x*) ≤ *1/2*
using *circle-y-bounds*[*OF x-coord-bounds*[*OF assms*]] **by** *auto*

lemma *rot-x-ivl*:
assumes (*0*::*real*) ≤ *x x* ≤ *1* *0* ≤ *y y* ≤ *1*
shows *0* ≤ *rot-x′′ x y* ∧ *rot-x′′ x y* ≤ *1*
proof
have ⋀*a*::*real*. *0* ≤ *a* ∧ *a* ≤ *1/2* ⟹ *0* ≤ *1/2* + (*2* ∗ *y* − *1*) ∗ *a* **using** *assms*
by (*sos* (((*A<0* ∗ *R<1*) + (((*A<=4* ∗ *R<1*) ∗ (*R<1/2* ∗ [*1*]^*2*)) + (((*A<=1* ∗ (*A<=5* ∗ *R<1*)) ∗ (*R<1* ∗ [*1*]^*2*)) + ((*A<=0* ∗ (*A<=4* ∗ *R<1*)) ∗ (*R<1* ∗ [*1*]^*2*)))))))
then show *0* ≤ *rot-x′′ x y*
using *assms circle-y-x-coord-bounds* **by**(*auto simp add*: *x-coord-inv-def*)
have ⋀*a*::*real*. *0* ≤ *a* ∧ *a* ≤ *1/2* ⟹ *1/2* + (*2* ∗ *y* − *1*) ∗ *a* ≤ *1* **using** *assms*
by (*sos* (((*A<0* ∗ *R<1*) + (((*A<=5* ∗ *R<1*) ∗ (*R<1/2* ∗ [*1*]^*2*)) + (((*A<=1* ∗ (*A<=4* ∗ *R<1*)) ∗ (*R<1* ∗ [*1*]^*2*)) + ((*A<=0* ∗ (*A<=5* ∗ *R<1*)) ∗ (*R<1* ∗ [*1*]^*2*)))))))
then show *rot-x′′ x y* ≤ *1*
using *assms circle-y-x-coord-bounds* **by** (*auto simp add*: *x-coord-inv-def*)
qed

abbreviation *rot-y′′* (*x*::*real*) (*y*::*real*) ≡ (*x-coord x*)/(*2*∗ (*circle-y* (*x-coord* (*rot-x′′ x y*)))) + *1/2*

lemma *rot-y-ivl*:
assumes (*0*::*real*) ≤ *x x* ≤ *1* *0* ≤ *y y* ≤ *1*
shows *0* ≤ *rot-y′′ x y* ∧ *rot-y′′ x y* ≤ *1*
proof
show *0* ≤ *rot-y′′ x y*
proof(*cases* (*x-coord x*) < *0*)
case *True*
have *i*: ⋀*a b*::*real*. *a* < *0* ⟹ *0* ≤ *a* + *b* ⟹ (*0* ≤ *a*/(*2*∗(*b*)) + *1/2*)
by(*auto simp add*: *algebra-simps divide-simps*)
have ∗: (*1/2* − *x*) ≤ *sqrt* (*x* ∗ *x* + (*1/4* + (*x* ∗ (*y* ∗ *4*) + *x* ∗ (*x* ∗ (*y* ∗ (*y* ∗ *4*))))) − (*x* + (*x* ∗ (*x* ∗ (*y* ∗ *4*)) + *x* ∗ (*y* ∗ (*y* ∗ *4*)))))
apply (*rule real-le-rsqrt*)
using *assms* **apply** (*simp add*: *algebra-simps power2-eq-square mult-left-le-one-le*)
by (*sos* (((*A<0* ∗ *R<1*) + ((*A<=0* ∗ (*A<=1* ∗ (*A<=2* ∗ (*A<=3* ∗ *R<1*)))) ∗ (*R<4* ∗ [*1*]^*2*)))))
have *rw*: |*x* − *x* ∗ *x*| = *x* − *x* ∗ *x* **using** *assms*
by (*sos* (() & (((((*A<0* ∗ *A<1*) ∗ *R<1*) + ((*A<=0* ∗ (*A<=1* ∗ (*A<1* ∗ *R<1*))) ∗ (*R<1* ∗ [*1*]^*2*)))) & ((((*A<0* ∗ *A<1*) ∗ *R<1*) + ((*A<=0* ∗ (*A<=1* ∗ (*A<1* ∗ *R<1*))) ∗ (*R<1* ∗ [*1*]^*2*)))))))
have *0* ≤ *x-coord x* + (*circle-y* (*x-coord* (*rot-x′′ x y*)))

using ∗ **apply** (*auto simp add*: *x-coord-inv-2*)
by (*auto simp add*: *circle-y-def algebra-simps rw x-coord-def*)
then show *?thesis*
using *True i* **by** *blast*
next
case *False*
have *i*: $\bigwedge a\ b{::}real.\ 0 \leq a \Longrightarrow 0 \leq b \ \Longrightarrow (0 \leq a/(2{*}b) + 1/2)$
by(*auto simp add*: *algebra-simps divide-simps*)
have $0 \leq$ *circle-y* (*x-coord* (*x-coord-inv* ((2 ∗ y − 1) ∗ *circle-y* (*x-coord x*))))
proof −
have *rw*: $|x - x * x| = x - x * x$ **using** *assms*
by (*sos* (() & (((((A<0 ∗ A<1) ∗ R<1) + ((A<=0 ∗ (A<=1 ∗ (A<1 ∗ R<1))) ∗ (R<1 ∗ [1]^2)))) & ((((A<0 ∗ A<1) ∗ R<1) + ((A<=0 ∗ (A<=1 ∗ (A<1 ∗ R<1))) ∗ (R<1 ∗ [1]^2)))))))
have $\bigwedge x.\ 0 \leq x \Longrightarrow x \leq 1/2 \Longrightarrow -1/2 \leq (2 * y - 1) * x$ **using** *assms*
by (*sos* (((A<0 ∗ R<1) + (((A<=4 ∗ R<1) ∗ (R<1/2 ∗ [1]^2)) + (((A<=1 ∗ (A<=5 ∗ R<1)) ∗ (R<1 ∗ [1]^2)) + ((A<=0 ∗ (A<=4 ∗ R<1)) ∗ (R<1 ∗ [1]^2)))))))
then have $-\ 1/2 \leq (2 * y - 1) *$ *circle-y* (*x-coord x*)
using *circle-y-x-coord-bounds assms*(1−2) **by** *auto*
moreover
have $\bigwedge x.\ 0 \leq x \Longrightarrow x \leq 1/2 \Longrightarrow (2 * y - 1) * x \leq 1/2$ **using** *assms*
by (*sos* (((A<0 ∗ R<1) + (((A<=5 ∗ R<1) ∗ (R<1/2 ∗ [1]^2)) + (((A<=1 ∗ (A<=4 ∗ R<1)) ∗ (R<1 ∗ [1]^2)) + ((A<=0 ∗ (A<=5 ∗ R<1)) ∗ (R<1 ∗ [1]^2)))))))
then have $(2 * y - 1) *$ *circle-y* (*x-coord x*) $\leq 1/2$
using *circle-y-x-coord-bounds assms*(1−2) **by** *auto*
ultimately show $0 \leq$ *circle-y* (*x-coord* (*x-coord-inv* ((2 ∗ y − 1) ∗ *circle-y* (*x-coord x*))))
by (*simp add*: *circle-y-bounds*(1) *x-coord-inv-2*)
qed
then show *?thesis*
using *False* **by** *auto*
qed
have *i*: $\bigwedge a\ b{::}real.\ a < 0 \Longrightarrow 0 \leq b \Longrightarrow (a/(2{*}(b)) + 1/2) \leq 1$
by(*auto simp add*: *algebra-simps divide-simps*)
show *rot-y''* $x\ y \leq 1$
proof(*cases* (*x-coord x*) < 0)
case *True*
have *i*: $\bigwedge a\ b{::}real.\ a < 0 \Longrightarrow 0 \leq b \Longrightarrow (a/(2{*}(b)) + 1/2) \leq 1$
by(*auto simp add*: *algebra-simps divide-simps*)
have $\bigwedge x.\ 0 \leq x \Longrightarrow x \leq 1/2 \Longrightarrow -1/2 \leq (2 * y - 1) * x$ **using** *assms*
by (*sos* (((A<0 ∗ R<1) + (((A<=4 ∗ R<1) ∗ (R<1/2 ∗ [1]^2)) + (((A<=1 ∗ (A<=5 ∗ R<1)) ∗ (R<1 ∗ [1]^2)) + ((A<=0 ∗ (A<=4 ∗ R<1)) ∗ (R<1 ∗ [1]^2)))))))
then have $-\ 1/2 \leq (2 * y - 1) *$ *circle-y* (*x-coord x*)
using *circle-y-x-coord-bounds assms*(1−2) **by** *auto*
moreover have $\bigwedge x.\ 0 \leq x \Longrightarrow x \leq 1/2 \Longrightarrow (2 * y - 1) * x \leq 1/2$ **using** *assms*

by (*sos* (((*A*<*0* ∗ *R*<*1*) + (((*A*<=*5* ∗ *R*<*1*) ∗ (*R*<*1*/*2* ∗ [*1*]^*2*)) + (((*A*<=*1* ∗ (*A*<=*4* ∗ *R*<*1*)) ∗ (*R*<*1* ∗ [*1*]^*2*)) + ((*A*<=*0* ∗ (*A*<=*5* ∗ *R*<*1*)) ∗ (*R*<*1* ∗ [*1*]^*2*)))))))
then have (*2* ∗ *y* − *1*) ∗ *circle-y* (*x-coord x*) ≤ *1*/*2*
using *circle-y-x-coord-bounds assms*(*1*−*2*) **by** *auto*
ultimately have *0* ≤ *circle-y* (*x-coord* (*x-coord-inv* ((*2* ∗ *y* − *1*) ∗ *circle-y* (*x-coord x*))))
by (*simp add*: *circle-y-bounds*(*1*) *x-coord-inv-2*)
then show *?thesis*
by (*simp add*: *True i*)
next
case *False*
have *i*: ⋀*a b*::*real*. *0* ≤ *a* ⟹ *a* ≤ *b* ⟹ (*a*/(*2*∗*b*) + *1*/*2*) ≤ *1*
by(*auto simp add*: *algebra-simps divide-simps*)
have (*x* − *1*/*2*) ∗ (*x* − *1*/*2*) ≤ (*x* ∗ *x* + (*1*/*4* + (*x* ∗ (*y* ∗ *4*) + *x* ∗ (*x* ∗ (*y* ∗ (*y* ∗ *4*))))) − (*x* + (*x* ∗ (*x* ∗ (*y* ∗ *4*)) + *x* ∗ (*y* ∗ (*y* ∗ *4*)))))
using *assms False*
apply(*auto simp add*: *x-coord-def*)
by (*sos* (((*A*<*0* ∗ *R*<*1*) + (((*A*<=*0* ∗ (*A*<=*1* ∗ (*A*<=*2* ∗ *R*<*1*))) ∗ (*R*<*2* ∗ [*1*]^*2*)) + ((*A*<=*0* ∗ (*A*<=*1* ∗ (*A*<=*2* ∗ (*A*<=*3* ∗ *R*<*1*)))) ∗ (*R*<*2* ∗ [*1*]^*2*))))))
then have *sqrt* ((*x* − *1*/*2*) ∗ (*x* − *1*/*2*)) ≤ *sqrt* (*x* ∗ *x* + (*1*/*4* + (*x* ∗ (*y* ∗ *4*) + *x* ∗ (*x* ∗ (*y* ∗ (*y* ∗ *4*))))) − (*x* + (*x* ∗ (*x* ∗ (*y* ∗ *4*)) + *x* ∗ (*y* ∗ (*y* ∗ *4*)))))
using *real-sqrt-le-mono* **by** *blast*
then have ∗: (*x* − *1*/*2*) ≤ *sqrt* (*x* ∗ *x* + (*1*/*4* + (*x* ∗ (*y* ∗ *4*) + *x* ∗ (*x* ∗ (*y* ∗ (*y* ∗ *4*))))) − (*x* + (*x* ∗ (*x* ∗ (*y* ∗ *4*)) + *x* ∗ (*y* ∗ (*y* ∗ *4*)))))
using *assms False* **by**(*auto simp add*: *x-coord-def*)
have *rw*: |*x* − *x* ∗ *x*| = *x* − *x* ∗ *x* **using** *assms*
by (*sos* (() & (((((*A*<*0* ∗ *A*<*1*) ∗ *R*<*1*) + ((*A*<=*0* ∗ (*A*<=*1* ∗ (*A*<*1* ∗ *R*<*1*))) ∗ (*R*<*1* ∗ [*1*]^*2*)))) & ((((*A*<*0* ∗ *A*<*1*) ∗ *R*<*1*) + ((*A*<=*0* ∗ (*A*<=*1* ∗ (*A*<*1* ∗ *R*<*1*))) ∗ (*R*<*1* ∗ [*1*]^*2*)))))))
have *x-coord x* ≤ *circle-y* (*x-coord* (*x-coord-inv* ((*2* ∗ *y* − *1*) ∗ *circle-y* (*x-coord x*))))
using ∗ **unfolding** *x-coord-inv-2*
by (*auto simp add*: *circle-y-def algebra-simps rw x-coord-def*)
then show *?thesis*
using *False i* **by** *auto*
qed
qed

lemma *circle-eq-rot-circle*:
assumes *0* ≤ *x x* ≤ *1 0* ≤ *y y* ≤ *1*
shows (*circle-cube* (*x*, *y*)) = (*rot-circle-cube* (*rot-y''* *x y*, *rot-x''* *x y*))
proof
have *rw*: |*1*/*4* − *x-coord x* ∗ *x-coord x*| = *1*/*4* − *x-coord x* ∗ *x-coord x*
apply(*rule abs-of-nonneg*)
using *assms mult-left-le* **by** (*auto simp add*: *x-coord-def divide-simps algebra-simps*)
show *fst* (*circle-cube* (*x*, *y*)) = *fst* (*rot-circle-cube* (*rot-y''* *x y*, *rot-x''* *x y*))
using *assms d-gt-0*

apply(*simp add*: *circle-cube-nice rot-circle-cube-def x-coord-inv-2 circle-y-def algebra-simps rw*)
apply(*auto simp add*: *x-coord-def algebra-simps*)
by (*sos* (((((A<0 * A<1) * ((A<0 * A<1) * R<1)) + (([~4*d^2] * A=0) + (((A<=1 * (A<=2 * (A<=3 * R<1))) * (R<8 * [d]^2)) + ((A<=1 * (A<=2 * (A<=3 * (A<1 * R<1)))) * (R<8 * [d]^2)))))) & ((((A<0 * A<1) * ((A<0 * A<1) * R<1)) + (([~4*d^2] * A=0) + (((A<=0 * (A<=2 * (A<=3 * R<1))) * (R<8 * [d]^2)) + ((A<=0 * (A<=2 * (A<=3 * (A<1 * R<1)))) * (R<8 * [d]^2))))))))
show *snd* (*circle-cube* (*x*, *y*)) = *snd* (*rot-circle-cube* (*rot-y'' x y*, *rot-x'' x y*))
using *assms*
by(*auto simp add*: *circle-cube-def rot-circle-cube-def x-coord-inv-def circle-y-def x-coord-def*)
qed

lemma *rot-circle-eq-circle*:
assumes $0 \le x$ $x \le 1$ $0 \le y$ $y \le 1$
shows (*rot-circle-cube* (*x*, *y*)) = (*circle-cube* (*rot-x'' y x*, *rot-y'' y x*))
proof
show *fst* (*rot-circle-cube* (*x*, *y*)) = *fst* (*circle-cube* (*rot-x'' y x*, *rot-y'' y x*))
using *assms*
by(*auto simp add*: *circle-cube-def rot-circle-cube-def x-coord-inv-def circle-y-def x-coord-def*)
have *rw*: |1/4 − *x-coord y* * *x-coord y*| = 1/4 − *x-coord y* * *x-coord y*
apply(*rule abs-of-nonneg*)
using *assms mult-left-le* **by** (*auto simp add*: *x-coord-def divide-simps algebra-simps*)
show *snd* (*rot-circle-cube* (*x*, *y*)) = *snd* (*circle-cube* (*rot-x'' y x*, *rot-y'' y x*))
using *assms d-gt-0*
apply(*simp add*: *circle-cube-nice rot-circle-cube-def x-coord-inv-2 circle-y-def algebra-simps rw*)
apply(*auto simp add*: *x-coord-def algebra-simps*)
by (*sos* (((((A<0 * A<1) * ((A<0 * A<1) * R<1)) + (([~4*d^2] * A=0) + (((A<=0 * (A<=1 * (A<=3 * R<1))) * (R<8 * [d]^2)) + ((A<=0 * (A<=1 * (A<=3 * (A<1 * R<1)))) * (R<8 * [d]^2)))))) & ((((A<0 * A<1) * ((A<0 * A<1) * R<1)) + (([~4*d^2] * A=0) + (((A<=0 * (A<=1 * (A<=2 * R<1))) * (R<8 * [d]^2)) + ((A<=0 * (A<=1 * (A<=2 * (A<1 * R<1)))) * (R<8 * [d]^2))))))))
qed

lemma *rot-img-eq*:
assumes $0 < d$
shows (*cubeImage* (*circle-cube*)) = (*cubeImage* (*rot-circle-cube*))
apply(*auto simp add*: *cubeImage-def image-def cbox-def real-pair-basis*)
by (*meson rot-y-ivl rot-x-ivl assms circle-eq-rot-circle rot-circle-eq-circle*)+

lemma *rot-circle-div-circle*:
assumes 0 < (*d*::*real*)
shows *gen-division* (*cubeImage circle-cube*) (*cubeImage* ‘ {*rot-circle-cube*})

using *rot-img-eq*[*OF assms*] **by**(*auto simp add*: *gen-division-def*)

lemma *circle-cube-boundary-valid*:
assumes (*k*,γ)∈*boundary circle-cube*
shows *valid-path* γ
proof −
have *f01*: *finite*{*0*,*1*}
by *simp*
show *?thesis*
using *assms*
unfolding *boundary-def horizontal-boundary-def vertical-boundary-def circle-cube-def valid-path-def piecewise-C1-differentiable-on-def*
by *safe* (*rule derivative-intros continuous-intros f01 exI ballI conjI refl* | *force simp add*: *field-simps*)+
qed

lemma *rot-circle-cube-boundary-valid*:
assumes (*k*,γ)∈*boundary rot-circle-cube*
shows *valid-path* γ
using *assms swap-valid-boundaries circle-cube-boundary-valid*
by (*fastforce simp add*: *rot-circle-cube-def*)

lemma *diff-divide-cancel*:
fixes *z*::*real* **shows** $z \neq 0 \Longrightarrow (a * z - a * (b * z)) / z = (a - a * b)$
by (*auto simp*: *field-simps*)

lemma *circle-cube-is-type-I*:
assumes $0 < d$
shows *typeI-twoCube circle-cube*
unfolding *typeI-twoCube-def*
proof (*intro exI conjI ballI*)
have *f01*: *finite*{$-d/2$,$d/2$}
by *simp*
show $-d/2 < d/2$
using *assms* **by** *simp*
show ($\lambda x.\ d * sqrt\ (1/4 - (x/d) * (x/d))$) *piecewise-C1-differentiable-on* {−d/2..d/2}
using *assms* **unfolding** *piecewise-C1-differentiable-on-def*
apply (*intro exI conjI*)
apply (*rule ballI refl f01 derivative-intros continuous-intros* | *simp*)+
apply (*auto simp*: *field-simps*)
by *sos*
show ($\lambda x.\ -d * sqrt\ (1/4 - (x/d) * (x/d))$) *piecewise-C1-differentiable-on* {−d/2..d/2}
using *assms* **unfolding** *piecewise-C1-differentiable-on-def*
apply (*intro exI conjI*)
apply (*rule ballI refl f01 derivative-intros continuous-intros* | *simp*)+
apply (*auto simp*: *field-simps*)
by *sos*

show − *d* ∗ *sqrt* (*1/4* − *x* / *d* ∗ (*x* / *d*)) ≤ *d* ∗ *sqrt* (*1/4* − *x* / *d* ∗ (*x* / *d*))
  if *x* ∈ {− *d/2..d/2*} **for** *x*
proof −
  have ∗: *x^2* ≤ (*d/2*)*^2*
    using *real-sqrt-le-iff that* **by** *fastforce*
  show *?thesis*
    apply (*rule mult-right-mono*)
    using *assms* ∗ **apply** (*simp-all add*: *divide-simps power2-eq-square*)
    done
qed
qed (*auto simp add*: *circle-cube-def divide-simps algebra-simps diff-divide-cancel*)

lemma *rot-circle-cube-is-type-II*:
  shows *typeII-twoCube rot-circle-cube*
  using *d-gt-0 swap-typeI-is-typeII circle-cube-is-type-I*
  by (*auto simp add*: *rot-circle-cube-def*)

definition *circle-bot-edge* **where**
  circle-bot-edge = (*1*::*int*, λ*t*. (*x-coord t* ∗ *d*, − *d* ∗ *circle-y* (*x-coord t*)))

definition *circle-top-edge* **where**
  circle-top-edge = (− *1*::*int*, λ*t*. (*x-coord t* ∗ *d*, *d* ∗ *circle-y* (*x-coord t*)))

definition *circle-right-edge* **where**
  circle-right-edge = (*1*::*int*, λ*y*. (*d/2*, *0*))

definition *circle-left-edge* **where**
  circle-left-edge = (− *1*::*int*, λ*y*. (− (*d/2*), *0*))

lemma *circle-cube-boundary-explicit*:
  boundary circle-cube = {*circle-left-edge*,*circle-right-edge*,*circle-bot-edge*,*circle-top-edge*}
  by (*auto simp add*: *valid-two-cube-def boundary-def horizontal-boundary-def vertical-boundary-def circle-cube-def*
    circle-top-edge-def circle-bot-edge-def circle-cube-nice x-coord-def circle-y-def
    circle-left-edge-def circle-right-edge-def)

definition *rot-circle-right-edge* **where**
  rot-circle-right-edge = (*1*::*int*, λ*t*. (*d* ∗ *circle-y* (*x-coord t*), *x-coord t* ∗ *d*))

definition *rot-circle-left-edge* **where**
  rot-circle-left-edge = (− *1*::*int*, λ*t*. (− *d* ∗ *circle-y* (*x-coord t*), *x-coord t* ∗ *d*))

definition *rot-circle-top-edge* **where**
  rot-circle-top-edge = (− *1*::*int*, λ*y*. (*0*, *d/2*))

definition *rot-circle-bot-edge* **where**
  rot-circle-bot-edge = (*1*::*int*, λ*y*. (*0*, − (*d/2*)))

lemma *rot-circle-cube-boundary-explicit*:

boundary (*rot-circle-cube*) =
{*rot-circle-top-edge*,*rot-circle-bot-edge*,*rot-circle-right-edge*,*rot-circle-left-edge*}
by (*auto simp add*: *rot-circle-cube-def valid-two-cube-def boundary-def horizontal-boundary-def vertical-boundary-def circle-cube-def*
rot-circle-right-edge-def rot-circle-left-edge-def x-coord-def circle-y-def rot-circle-top-edge-def rot-circle-bot-edge-def)

lemma *rot-circle-cube-vertical-boundary-explicit*:
vertical-boundary rot-circle-cube = {*rot-circle-right-edge*,*rot-circle-left-edge*}
by (*auto simp add*: *rot-circle-cube-def valid-two-cube-def vertical-boundary-def circle-cube-def*
rot-circle-right-edge-def rot-circle-left-edge-def x-coord-def circle-y-def)

lemma *circ-left-edge-neq-top*:
$(- 1::int, \lambda y::real.\ (- (d/2),\ 0)) \neq (- 1,\ \lambda x.\ ((x - 1/2) * d,\ d * sqrt\ (1/4 - (x - 1/2) * (x - 1/2))))$
by (*metis* (*no-types*, *lifting*) *add-diff-cancel-right′ d-gt-0 mult.commute mult-cancel-left order-less-irrefl prod.inject*)

lemma *circle-cube-valid-two-cube*: *valid-two-cube* (*circle-cube*)
proof (*auto simp add*: *valid-two-cube-def boundary-def horizontal-boundary-def vertical-boundary-def circle-cube-def*)
have *iv*: $(- 1::int, \lambda y::real.\ (- (d/2),\ 0)) \neq (- 1,\ \lambda x.\ ((x - 1/2) * d,\ d * sqrt\ (1/4 - (x - 1/2) * (x - 1/2))))$
using *d-gt-0* **apply** (*auto simp add*: *algebra-simps*)
by (*metis* (*no-types*, *opaque-lifting*) *add-diff-cancel-right′ add-uminus-conv-diff cancel-comm-monoid-add-class.diff-cancel less-eq-real-def linorder-not-le mult.left-neutral prod.simps*(*1*))
have *v*: $(1::int, \lambda y.\ (d/2,\ 0)) \neq (1,\ \lambda x.\ ((x - 1/2) * d,\ - (d * sqrt\ (1/4 - (x - 1/2) * (x - 1/2)))))$
using *d-gt-0* **apply** (*auto simp add*: *algebra-simps*)
by (*metis* (*no-types*, *opaque-lifting*) *diff-0 equal-neg-zero mult-zero-left nonzero-mult-div-cancel-left order-less-irrefl prod.sel*(*1*) *times-divide-eq-right zero-neq-numeral*)
show $card\ \{(- 1::int, \lambda y.\ (- (d/2),\ 0)),\ (1,\ \lambda y.\ (d/2,\ 0)),\ (1,\ \lambda x.\ ((x - 1/2) * d,\ - (d * sqrt\ (1/4 - (x - 1/2) * (x - 1/2))))),$
$(- 1,\ \lambda x.\ ((x - 1/2) * d,\ d * sqrt\ (1/4 - (x - 1/2) * (x - 1/2))))\} = 4$
using *iv v* **by** *auto*
qed

lemma *rot-circle-cube-valid-two-cube*:
shows *valid-two-cube rot-circle-cube*
using *valid-cube-valid-swap circle-cube-valid-two-cube d-gt-0 rot-circle-cube-def*
by *force*

definition *circle-arc-0* **where** $circle\text{-}arc\text{-}0 = (1,\ \lambda t::real.\ (0,0))$

lemma *circle-top-bot-edges-neq′* [*simp*]:
shows *circle-top-edge* ≠ *circle-bot-edge*

by (*simp add*: *circle-top-edge-def circle-bot-edge-def*)

lemma *rot-circle-top-left-edges-neq* [*simp*]: *rot-circle-top-edge* ≠ *rot-circle-left-edge*
apply (*simp add*: *rot-circle-left-edge-def rot-circle-top-edge-def x-coord-def*)
by (*metis* (*mono-tags*, *opaque-lifting*) *cancel-comm-monoid-add-class.diff-cancel d-gt-0 divide-eq-0-iff mult-zero-left order-less-irrefl prod.sel*(*2*) *zero-neq-numeral*)

lemma *rot-circle-bot-left-edges-neq* [*simp*]: *rot-circle-bot-edge* ≠ *rot-circle-left-edge*
by (*simp add*: *rot-circle-left-edge-def rot-circle-bot-edge-def x-coord-def*)

lemma *rot-circle-top-right-edges-neq* [*simp*]: *rot-circle-top-edge* ≠ *rot-circle-right-edge*
by (*simp add*: *rot-circle-right-edge-def rot-circle-top-edge-def x-coord-def*)

lemma *rot-circle-bot-right-edges-neq* [*simp*]: *rot-circle-bot-edge* ≠ *rot-circle-right-edge*
apply (*simp add*: *rot-circle-right-edge-def rot-circle-bot-edge-def x-coord-def*)
by (*metis* (*mono-tags*, *opaque-lifting*) *cancel-comm-monoid-add-class.diff-cancel d-gt-0 divide-eq-0-iff mult-zero-left neg-0-equal-iff-equal order-less-irrefl prod.sel*(*2*) *zero-neq-numeral*)

lemma *rot-circle-right-top-edges-neq'* [*simp*]: *rot-circle-right-edge* ≠ *rot-circle-left-edge*
by (*simp add*: *rot-circle-left-edge-def rot-circle-right-edge-def*)

lemma *rot-circle-left-bot-edges-neq* [*simp*]: *rot-circle-left-edge* ≠ *rot-circle-top-edge*
apply (*simp add*: *rot-circle-top-edge-def rot-circle-left-edge-def*)
by (*metis* (*no-types*, *opaque-lifting*) *cancel-comm-monoid-add-class.diff-cancel d-gt-0 mult.commute mult-zero-right nonzero-mult-div-cancel-left order-less-irrefl prod.sel*(*2*) *times-divide-eq-right x-coord-def zero-neq-numeral*)

lemma *circle-right-top-edges-neq* [*simp*]: *circle-right-edge* ≠ *circle-top-edge*
proof −
have *circle-right-edge* = (*1*, (λ*r*::*real*. (*d* / *2*, *0*::*real*)))
using *circle.circle-right-edge-def circle-axioms* **by** *blast*
then show *?thesis*
using *circle.circle-top-edge-def circle-axioms* **by** *auto*
qed

lemma *circle-left-bot-edges-neq* [*simp*]: *circle-left-edge* ≠ *circle-bot-edge*
proof −
have *circle-bot-edge* = (*1*, λ*r*. (*x-coord r* ∗ *d*, − *d* ∗ *circle-y* (*x-coord r*)))
using *circle.circle-bot-edge-def circle-axioms* **by** *blast*
then show *?thesis*
by (*simp add*: *circle-left-edge-def*)
qed

lemma *circle-left-top-edges-neq* [*simp*]: *circle-left-edge* ≠ *circle-top-edge*
proof −
have ∃ *r*. ((*r* − *1* / *2*) ∗ *d*, *d* ∗ *sqrt* (*1* / *4* − (*r* − *1* / *2*) ∗ (*r* − *1* / *2*))) ≠ (− (*d* / *2*), *0*)
by (*metis circ-left-edge-neq-top*)

then have (∃ *r*. *d* ∗ *r* ≠ − (*d* / *2*)) ∨ (∃ *r ra*. (*x-coord* (*x-coord-inv r*) ∗ *d*, *d* ∗ *circle-y* (*x-coord* (*x-coord-inv r*))) = (*x-coord ra* ∗ *d*, *d* ∗ *circle-y* (*x-coord ra*)) ∧ *d* ∗ *circle-y r* ≠ *0*)
by (*metis mult.commute*)
then have (λ*r*. (*x-coord r* ∗ *d*, *d* ∗ *circle-y* (*x-coord r*))) ≠ (λ*r*. (− (*d* / *2*), *0*))
by (*metis mult.commute prod.simps*(*1*) *x-coord-inv-2*)
then show *?thesis*
by (*simp add*: *circle-left-edge-def circle-top-edge-def*)
qed

lemma *circle-right-bot-edges-neq* [*simp*]: *circle-right-edge* ≠ *circle-bot-edge*
by (*smt Pair-inject circle-bot-edge-def d-gt-0 circle.circle-right-edge-def circle-axioms mult-le-cancel-right-pos x-coord-def*)

definition *circle-polar* **where**
circle-polar t = ((*d*/*2*) ∗ *cos* (*2* ∗ *pi* ∗ *t*), (*d*/*2*) ∗ *sin* (*2* ∗ *pi* ∗ *t*))

lemma *circle-polar-smooth*: (*circle-polar*) *C1-differentiable-on* {*0*..*1*}
proof −
have *inj* ((∗) (*2* ∗ *pi*))
by (*auto simp*: *inj-on-def*)
then have ∗: ⋀*x*. *finite* ({*0*..*1*} ∩ (∗) (*2* ∗ *pi*) −‘ {*x*})
by (*simp add*: *finite-vimageI*)
have (λ*t*. ((*d*/*2*) ∗ *cos* (*2* ∗ *pi* ∗ *t*), (*d*/*2*) ∗ *sin* (*2* ∗ *pi* ∗ *t*))) *C1-differentiable-on* {*0*..*1*}
by (*rule* ∗ *derivative-intros*)+
then show *?thesis*
apply (*rule eq-smooth-gen*)
by(*simp add*: *circle-polar-def*)
qed

abbreviation *custom-arccos* ≡ (λ*x*. (*if* −*1* ≤ *x* ∧ *x* ≤ *1 then arccos x else* (*if x* < −*1 then* −*x* + *pi else 1* −*x*)))

lemma *cont-custom-arccos*:
assumes *S* ⊆ {−*1*..*1*}
shows *continuous-on S custom-arccos*
proof −
have *continuous-on* ({−*1*..*1*} ∪ {}) *custom-arccos*
by (*auto intro*!: *continuous-on-cases continuous-intros*)
with *assms* **show** *?thesis*
using *continuous-on-subset* **by** *auto*
qed

lemma *custom-arccos-has-deriv*:
assumes − *1* < *x x* < *1*
shows *DERIV custom-arccos x* :> *inverse* (− *sqrt* (*1* − x^2))
proof −
have *x1*: $|x|^2 < 1^2$

by (*simp add*: *abs-less-iff abs-square-less-1 assms*)
show *?thesis*
apply (*rule DERIV-inverse-function* [**where** *f=cos* **and** *a=−1* **and** *b=1*])
apply (*rule x1 derivative-eq-intros* | *simp add*: *sin-arccos*)+
using *assms x1 cont-custom-arccos* [*of* {*−1<..<1*}]
apply (*auto simp*: *continuous-on-eq-continuous-at greaterThanLessThan-subseteq-atLeastAtMost-iff*)
done
qed

declare
custom-arccos-has-deriv[*THEN DERIV-chain2*, *derivative-intros*]
custom-arccos-has-deriv[*THEN DERIV-chain2*, *unfolded has-field-derivative-def*, *derivative-intros*]

lemma *circle-boundary-reparams*:
shows *rot-circ-left-edge-reparam-polar-circ-split*:
reparam (*rec-join* [(*rot-circle-left-edge*)]) (*rec-join* [(*subcube* (*1/4*) (*1/2*) (*1*, *circle-polar*)), (*subcube* (*1/2*) (*3/4*) (*1*, *circle-polar*))])
(**is** *?P1*)
and *circ-top-edge-reparam-polar-circ-split*:
reparam (*rec-join* [(*circle-top-edge*)]) (*rec-join* [(*subcube 0* (*1/4*) (*1*, *circle-polar*)), (*subcube* (*1/4*) (*1/2*) (*1*, *circle-polar*))])
(**is** *?P2*)
and *circ-bot-edge-reparam-polar-circ-split*:
reparam (*rec-join* [(*circle-bot-edge*)]) (*rec-join* [(*subcube* (*1/2*) (*3/4*) (*1*, *circle-polar*)), (*subcube* (*3/4*) *1* (*1*, *circle-polar*))])
(**is** *?P3*)
and *rot-circ-right-edge-reparam-polar-circ-split*:
reparam (*rec-join* [(*rot-circle-right-edge*)]) (*rec-join* [(*subcube* (*3/4*) *1* (*1*, *circle-polar*)), (*subcube 0* (*1/4*) (*1*, *circle-polar*))])
(**is** *?P4*)
proof −
let *?φ* = ((∗) (*1/pi*) ∘ *custom-arccos* ∘ (λ*t*. *2* ∗ *x-coord* (*1* − *t*)))
let *?LHS1* = (λ*x*. (− (*d* ∗ *sqrt* (*1/4* − *x-coord* (*1* − *x*) ∗ *x-coord* (*1* − *x*))), *x-coord* (*1* − *x*) ∗ *d*))
let *?RHS1* = ((λ*x*. *if x* ∗ *2* ≤ *1 then* (*d* ∗ *cos* (*2* ∗ *pi* ∗ (*2* ∗ *x/4* + *1/4*))/*2*, *d* ∗ *sin* (*2* ∗ *pi* ∗ (*2* ∗ *x/4* + *1/4*))/*2*)
else (*d* ∗ *cos* (*2* ∗ *pi* ∗ ((*2* ∗ *x* − *1*)/*4* + *1/2*))/*2*, *d* ∗ *sin* (*2* ∗ *pi* ∗ ((*2* ∗ *x* − *1*)/*4* + *1/2*))/*2*)) ∘ *?φ*)
let *?LHS2* = λ*x*. ((*x-coord* (*1* − *x*) ∗ *d*, *d* ∗ *sqrt* (*1/4* − *x-coord* (*1* − *x*) ∗ *x-coord* (*1* − *x*))))
let *?RHS2* = ((λ*x*. *if x* ∗ *2* ≤ *1 then* (*d* ∗ *cos* (*2* ∗ *x* ∗ *pi/2*)/*2*, *d* ∗ *sin* (*2* ∗ *x* ∗ *pi/2*)/*2*) *else* (*d* ∗ *cos* (*2* ∗ *pi* ∗ ((*2* ∗ *x* − *1*)/*4* + *1/4*))/*2*, *d* ∗ *sin* (*2* ∗ *pi* ∗ ((*2* ∗ *x* − *1*)/*4* + *1/4*))/*2*)) ∘ *?φ*)
let *?LHS3* = λ*x*. (*x-coord x* ∗ *d*, − (*d* ∗ *sqrt* (*1/4* − *x-coord x* ∗ *x-coord x*)))
let *?RHS3* = (λ*x*. *if x* ∗ *2* ≤ *1 then* (*d* ∗ *cos* (*2* ∗ *pi* ∗ (*2* ∗ *x/4* + *1/2*))/*2*, *d* ∗ *sin* (*2* ∗ *pi* ∗ (*2* ∗ *x/4* + *1/2*))/*2*)
else (*d* ∗ *cos* (*2* ∗ *pi* ∗ ((*2* ∗ *x* − *1*)/*4* + *3/4*))/*2*, *d* ∗ *sin* (*2* ∗ *pi* ∗ ((*2* ∗ *x* − *1*)/*4* + *3/4*))/*2*))

```
  let ?LHS4 = λx. (d ∗ sqrt (1/4 − x-coord x ∗ x-coord x), x-coord x ∗ d)
  let ?RHS4 = (λx. if x ∗ 2 ≤ 1 then (d ∗ cos (2 ∗ pi ∗ (2 ∗ x/4 + 3/4))/2, d
∗ sin (2 ∗ pi ∗ (2 ∗ x/4 + 3/4))/2)
              else (d ∗ cos ((2 ∗ x − 1) ∗ pi/2)/2, d ∗ sin ((2 ∗ x − 1) ∗ pi/2)/2))
  have phi-diff: ?φ piecewise-C1-differentiable-on {0..1}
    unfolding piecewise-C1-differentiable-on-def
  proof
    show continuous-on {0..1} ?φ
      unfolding x-coord-def
      by (intro continuous-on-compose cont-custom-arccos continuous-intros) auto
    have ?φ C1-differentiable-on {0..1} − {0,1}
     unfolding x-coord-def piecewise-C1-differentiable-on-def C1-differentiable-on-def
valid-path-def
      by (simp | rule has-vector-derivative-pair-within DERIV-image-chain deriva-
tive-eq-intros continuous-intros exI ballI conjI
         | force simp add: field-simps | sos)+
    then show ∃ s. finite s ∧ ?φ C1-differentiable-on {0..1} − s
      by force
  qed
  have inj: inj ?φ
    apply (intro comp-inj-on inj-on-cases inj-on-arccos)
          apply (auto simp add: inj-on-def x-coord-def)
    using pi-ge-zero apply auto[1]
     apply (smt arccos)
    by (smt arccos-lbound)
  then have fin: ⋀x. ⟦0 ≤ x; x ≤ 1⟧ ⟹ finite (?φ −‘ {x})
    by (simp add: finite-vimageI)
  have ?φ ‘ {0..1} = {0..1}
  proof
    show ?φ ‘ {0..1} ⊆ {0..1}
      by (auto simp add: x-coord-def divide-simps arccos-lbound arccos-bounded)
    have arccos (1 − 2 ∗ ((1 − cos (x ∗ pi))/2)) = x ∗ pi if 0 ≤ x x ≤ 1 for x
      using that by (simp add: field-simps arccos-cos)
    then show {0..1} ⊆ ?φ ‘ {0..1}
      apply (auto simp: x-coord-def o-def image-def)
      by (rule-tac x=(1 − cos (x ∗ pi))/2 in bexI) auto
  qed
  then have bij-phi: bij-betw ?φ {0..1} {0..1}
    unfolding bij-betw-def using inj inj-on-subset by blast
  have phi01: ?φ −‘ {0..1} ⊆ {0..1}
    by (auto simp add: subset-iff x-coord-def divide-simps)
  {
    fix x::real assume x: 0 ≤ x x ≤ 1
    then have i: − 1 ≤ (2 ∗ x − 1) (2 ∗ x − 1) ≤ 1 by auto
    have ii: cos (pi / 2 + arccos (1 − x ∗ 2)) = −sin (arccos (1 − x ∗ 2))
      using minus-sin-cos-eq[symmetric, where ?x = arccos (1 − x ∗ 2)]
      by (auto simp add: add.commute)
    have 2 ∗ sqrt (x − x ∗ x) = sqrt (4∗x − 4∗x ∗ x)
      by (metis mult.assoc real-sqrt-four real-sqrt-mult right-diff-distrib)
```

also have ... = *sqrt* (*1* − (*2* ∗ *x* − *1*) ∗ (*2* ∗ *x* − *1*))
by (*simp add*: *algebra-simps*)
finally have *iii*: *2* ∗ *sqrt* (*x* − *x* ∗ *x*) = *cos* (*arcsin* (*2* ∗ *x* − *1*)) ∧ *2* ∗ *sqrt* (*x* − *x* ∗ *x*) = *sin* (*arccos* (*1* − *2* ∗ *x*))
using *arccos-minus*[*OF i*] **unfolding** *minus-diff-eq sin-pi-minus*
by (*simp add*: *cos-arcsin i power2-eq-square sin-arccos*)
then have *1*: *?LHS1 x* = *?RHS1 x*
using *d-gt-0 i x* **apply** (*simp add*: *x-coord-def field-simps*)
apply (*auto simp add*: *diff-divide-distrib add-divide-distrib right-diff-distrib mult.commute ii*)
using *cos-sin-eq*[**where** *?x* = − *arccos* (*1* − *x* ∗ *2*)]
by (*simp add*: *cos-sin-eq*[**where** *?x* = − *arccos* (*1* − *x* ∗ *2*)] *right-diff-distrib*)
have *2*: *?LHS2 x* = *?RHS2 x*
using *x d-gt-0 iii* **by** (*auto simp add*: *x-coord-def diff-divide-distrib algebra-simps*)
have *a*: *cos* (*pi* / *2* − *arccos* (*x* ∗ *2* − *1*)) = *cos* (*pi* / *2* − *arccos* (*1* − *x* ∗ *2*))
by (*smt arccos-minus arccos-cos2 arccos-lbound cos-arccos cos-ge-minus-one cos-le-one i*(*1*) *i*(*2*) *pi-def pi-half*)
have *b*: *cos* (*arccos* (*1* − *x* ∗ *2*) + *pi* ∗ *3* / *2*) = *cos* ((*pi* / *2* − *arccos* (*x* ∗ *2* − *1*)) + *2* ∗ *pi*)
using *arccos-minus*[*OF i*] **by**(*auto simp add*: *mult.commute add.commute*)
then have *c*: ... = *cos* (*pi* / *2* − *arccos* (*x* ∗ *2* − *1*)) **using** *cos-periodic* **by** *blast*
have *cos* (− *pi* − *arccos* (*1* − *x* ∗ *2*)) = *cos* (− (*pi* + *arccos* (*1* − *x* ∗ *2*)))
by (*auto simp add*: *minus-add*[**where** *b* = *pi* **and** *a* = *arccos* (*1* − *x* ∗ *2*), *symmetric*])
moreover have ... = *cos* ((*pi* + *arccos* (*1* − *x* ∗ *2*)))
using *cos-minus* **by** *blast*
moreover have ... = *cos* (*2*∗*pi* − *arccos* (*x* ∗ *2* − *1*))
using *arccos-minus*[*OF i*] **by** (*auto simp add*: *mult.commute add.commute*)
moreover have ... = *cos* (*arccos* (*x* ∗ *2* − *1*))
using *cos-2pi-minus* **by** *auto*
ultimately have *d*: *cos* (− *pi* − *arccos* (*1* − *x* ∗ *2*)) = (*x* ∗ *2* − *1*)
using *cos-arccos*[*OF i*] *mult.commute* **by** *metis*
have *cosm*: ⋀*x*. *cos* (*x* − *pi*∗*2*) = *cos x*
by (*metis cos-periodic eq-diff-eq′ mult.commute*)
have *34*: *?LHS3 x* = (*?RHS3* ∘ *?φ*) *x* *?LHS4 x* = (*?RHS4* ∘ *?φ*) *x*
using *d-gt-0 x a b c iii cos-periodic* [*of pi* / *2* − *arccos* (*x* ∗ *2* − *1*)]
apply (*auto simp add*: *x-coord-def algebra-simps diff-divide-distrib power2-eq-square*)
apply (*auto simp add*: *sin-cos-eq cosm*)
using *d* **apply** (*auto simp add*: *right-diff-distrib*)
by (*smt cos-minus*)
note *1 2 34*
} **note** ∗ = *this*
show *?P1 ?P2 ?P3 ?P4*
apply (*auto simp add*: *subcube-def circle-bot-edge-def circle-top-edge-def circle-polar-def reversepath-def*
subpath-def joinpaths-def circle-y-def rot-circle-left-edge-def rot-circle-right-edge-def)
unfolding *reparam-def*

by (*rule ballI exI conjI impI phi-diff bij-phi phi01 fin* ∗ | *force simp add*: *x-coord-def*)+
qed

definition *circle-cube-boundary-to-polarcircle* **where**
circle-cube-boundary-to-polarcircle $\gamma \equiv$
if (γ = (*circle-top-edge*)) *then*
{*subcube 0* (*1/4*) (*1*, *circle-polar*), *subcube* (*1/4*) (*1/2*) (*1*, *circle-polar*)}
else if (γ = (*circle-bot-edge*)) *then*
{*subcube* (*1/2*) (*3/4*) (*1*, *circle-polar*), *subcube* (*3/4*) *1* (*1*, *circle-polar*)}
else {}

definition *rot-circle-cube-boundary-to-polarcircle* **where**
rot-circle-cube-boundary-to-polarcircle $\gamma \equiv$
if (γ = (*rot-circle-left-edge*)) *then*
{*subcube* (*1/4*) (*1/2*) (*1*, *circle-polar*), *subcube* (*1/2*) (*3/4*) (*1*, *circle-polar*)}
else if (γ = (*rot-circle-right-edge*)) *then*
{*subcube* (*3/4*) *1* (*1*, *circle-polar*), *subcube 0* (*1/4*) (*1*, *circle-polar*)}
else {}

lemma *circle-arcs-neq*:
assumes $0 \le k$ $k \le 1$ $0 \le n$ $n \le 1$ $n < k$ $k + n < 1$
shows *subcube k m* (*1*, *circle-polar*) ≠ *subcube n q* (*1*, *circle-polar*)
proof (*simp add*: *subcube-def subpath-def circle-polar-def*)
have *cos* (*2* ∗ *pi* ∗ *k*) ≠ *cos*(*2* ∗ *pi* ∗ *n*)
unfolding *cos-eq*
proof *safe*
show *False* **if** $2 * pi * k = 2 * pi * n + 2 * m * pi$ $m \in \mathbb{Z}$ **for** *m*
proof −
have $2 * pi * (k - n) = 2 * m * pi$
using *distrib-left that* **by** (*simp add*: *left-diff-distrib mult.commute*)
then have *a*: $m = (k - n)$ **by** *auto*
have $\lfloor k - n \rfloor = 0$
using *assms* **by** (*simp add*: *floor-eq-iff*)
then have $k - n \notin \mathbb{Z}$
using *assms* **by** (*auto simp only*: *frac-eq-0-iff*[*symmetric*] *frac-def*)
then show *False* **using** *that a* **by** *auto*
qed
show *False* **if** $2 * pi * k = -(2 * pi * n) + 2 * m * pi$ $m \in \mathbb{Z}$ **for** *m*
proof −
have $2 * pi * (k + n) = 2 * m * pi$
using *that* **by** (*auto simp add*: *distrib-left*)
then have *a*: $m = (k + n)$ **by** *auto*
have $\lfloor k + n \rfloor = 0$
using *assms* **by** (*simp add*: *floor-eq-iff*)
then have $k + n \notin \mathbb{Z}$
using *Ints-def assms* **by** *force*

then show *False* **using** *that a* **by** *auto*
qed
qed
then have $(\lambda x.\ (d * cos\ (2 * pi * ((m - k) * x + k))/2,\ d * sin\ (2 * pi * ((m - k) * x + k))/2))\ 0\ \neq (\lambda x.\ (d * cos\ (2 * pi * ((q - n) * x + n))/2,\ d * sin\ (2 * pi * ((q - n) * x + n))/2))\ 0$
using *d-gt-0* **by** *auto*
then show $(\lambda x.\ (d * cos\ (2 * pi * ((m - k) * x + k))/2,\ d * sin\ (2 * pi * ((m - k) * x + k))/2)) \neq (\lambda x.\ (d * cos\ (2 * pi * ((q - n) * x + n))/2,\ d * sin\ (2 * pi * ((q - n) * x + n))/2))$
by *metis*
qed

lemma *circle-arcs-neq-2*:
assumes $0 \leq k\ k \leq 1\ 0 \leq n\ n \leq 1\ n < k\ 0 < n$ **and** *kn12*: $1/2 < k + n$ **and** $k + n < 3/2$
shows *subcube k m* (*1*, *circle-polar*) $\neq$ *subcube n q* (*1*, *circle-polar*)
proof (*simp add*: *subcube-def subpath-def circle-polar-def*)
have $sin\ (2 * pi * k) \neq sin(2 * pi * n)$
unfolding *sin-eq*
proof *safe*
show *False* **if** $m \in \mathbb{Z}\ 2 * pi * k = 2 * pi * n + 2 * m * pi$ **for** *m*
proof −
have $2 * pi * (k - n) = 2 * m * pi$
using *that* **by** (*simp add*: *left-diff-distrib mult.commute*)
then have *a*: $m = (k - n)$ **by** *auto*
have $\lfloor k - n \rfloor = 0$
using *assms* **by** (*simp add*: *floor-eq-iff*)
then have $k - n \notin \mathbb{Z}$
using *assms* **by** (*auto simp only*: *frac-eq-0-iff*[*symmetric*] *frac-def*)
then show *False* **using** *that a* **by** *auto*
qed
show *False* **if** $2 * pi * k = - (2 * pi * n) + (2 * m + 1) * pi\ m \in \mathbb{Z}$ **for** *m*
proof −
have *i*: $\bigwedge pi.\ 0 < pi \Longrightarrow 2 * pi * (k + n\ ) = 2 * m * pi + pi \Longrightarrow m = (k + n) - 1/2$
by (*sos* (((((A<0 ∗ A<1) ∗ R<1) + ([1/2] ∗ A=0))) & ((((A<0 ∗ A<1) ∗ R<1) + ([~1/2] ∗ A=0))))))
have $2 * pi * (k + n) = 2 * m * pi + pi$
using *that* **by** (*simp add*: *algebra-simps*)
then have *a*: $m = (k + n) - 1/2$ **using** *i*[*OF pi-gt-zero*] **by** *fastforce*
have $\lfloor k + n - 1/2 \rfloor = 0$
using *assms* **by** (*auto simp add*: *floor-eq-iff*)
then have $k + n - 1/2 \notin \mathbb{Z}$
by (*metis Ints-cases add.commute add.left-neutral add-diff-cancel-left′ add-diff-eq kn12 floor-of-int of-int-0 order-less-irrefl*)
then show *False* **using** *that a* **by** *auto*
qed
qed

```
  then have (λx. (d ∗ cos (2 ∗ pi ∗ ((m − k) ∗ x + k))/2, d ∗ sin (2 ∗ pi ∗ ((m
− k) ∗ x + k))/2)) 0 ≠ (λx. (d ∗ cos (2 ∗ pi ∗ ((q − n) ∗ x + n))/2, d ∗ sin (2
∗ pi ∗ ((q − n) ∗ x + n))/2)) 0
    using d-gt-0 by auto
  then show (λx. (d ∗ cos (2 ∗ pi ∗ ((m − k) ∗ x + k))/2, d ∗ sin (2 ∗ pi ∗ ((m
− k) ∗ x + k))/2)) ≠ (λx. (d ∗ cos (2 ∗ pi ∗ ((q − n) ∗ x + n))/2, d ∗ sin (2 ∗
pi ∗ ((q − n) ∗ x + n))/2))
    by metis
qed

lemma circle-cube-is-only-horizontal-div-of-rot:
  shows only-horizontal-division (boundary (circle-cube)) {rot-circle-cube}
  unfolding only-horizontal-division-def
proof (rule exI [of - {}], simp, intro conjI ballI)
  show finite (boundary circle-cube)
    using circle.circle-cube-boundary-explicit circle-axioms by auto
  show boundary-chain (boundary circle-cube)
    by (simp add: two-cube-boundary-is-boundary)
  show ⋀x. x ∈ boundary circle-cube ⟹ case x of (k, x) ⇒ valid-path x
    using circle-cube-boundary-valid by blast
  let ?𝒱 = (boundary (circle-cube))
  let ?pi = {circle-left-edge, circle-right-edge}
  let ?pj = {rot-circle-top-edge, rot-circle-bot-edge}
  let ?f = circle-cube-boundary-to-polarcircle
  let ?one-chaini = boundary (circle-cube) − ?pi
 have c: common-reparam-exists ?𝒱 (two-chain-vertical-boundary {rot-circle-cube})
    unfolding common-reparam-exists-def
  proof (intro exI conjI)
    let ?subdiv = {(subcube 0 (1/4) (1, circle-polar)),
                   (subcube (1/4) (1/2) (1, circle-polar)),
                   (subcube (1/2) (3/4) (1, circle-polar)),
                   (subcube (3/4) 1 (1, circle-polar))}
    show (∀ (k, γ)∈?subdiv. γ C1-differentiable-on {0..1})
     using subpath-smooth[OF circle-polar-smooth] by (auto simp add: subcube-def)
    have 1: finite ?subdiv by auto
    show boundary-chain ?subdiv
      by (simp add: boundary-chain-def subcube-def)
    show chain-reparam-chain′ (boundary (circle-cube) − ?pi) ?subdiv
      unfolding chain-reparam-chain′-def
    proof (intro exI conjI impI)
      show ⋃ (?f ‘ ?one-chaini) = ?subdiv
      apply (simp add: circle-cube-boundary-to-polarcircle-def circle-cube-boundary-explicit)
        using circle-top-bot-edges-neq′ by metis
         let ?l = [subcube 0 (1/4) (1, circle-polar), subcube (1/4) (1/2) (1, cir-
cle-polar)]
       have chain-reparam-weak-path (coeff-cube-to-path (circle-top-edge)) {subcube
0 (1/4) (1, circle-polar), subcube (1/4) (1/2) (1, circle-polar)}
        unfolding chain-reparam-weak-path-def
      proof (intro exI conjI)
```

show *valid-chain-list ?l*
by (*auto simp add*: *subcube-def circle-top-edge-def x-coord-def circle-y-def pathfinish-def pathstart-def*
reversepath-def subpath-def joinpaths-def)
show *reparam* (*coeff-cube-to-path circle-top-edge*) (*rec-join ?l*)
using *circ-top-edge-reparam-polar-circ-split* **by** *auto*
show *distinct ?l*
apply *simp*
apply (*subst neq-commute*)
apply (*simp add*: *circle-arcs-neq*)
done
qed *auto*
moreover have *chain-reparam-weak-path* (*coeff-cube-to-path* (*circle-bot-edge*)) {*subcube* (*1/2*) (*3/4*) (*1*, *circle-polar*), *subcube* (*3/4*) *1* (*1*, *circle-polar*)}
unfolding *chain-reparam-weak-path-def*
proof
let *?l* = [*subcube* (*1/2*) (*3/4*) (*1*, *circle-polar*), *subcube* (*3/4*) *1* (*1*, *circle-polar*)]
have *a*: *valid-chain-list ?l*
by (*auto simp add*: *subcube-def circle-top-edge-def x-coord-def circle-y-def pathfinish-def pathstart-def*
reversepath-def subpath-def joinpaths-def)
have *b*: *reparam* (*rec-join* [*circle-bot-edge*]) (*rec-join ?l*)
using *circ-bot-edge-reparam-polar-circ-split* **by** *auto*
have *c*: *subcube* (*3/4*) *1* (*1*, *circle-polar*) ≠ *subcube* (*1/2*) (*3/4*) (*1*, *circle-polar*)
apply(*rule circle-arcs-neq-2*) **using** *d-gt-0*(*1*) **by** *auto*
show *set ?l* = {*subcube* (*1/2*) (*3/4*) (*1*, *circle-polar*), *subcube* (*3/4*) *1* (*1*, *circle-polar*)} ∧
distinct ?l ∧ *reparam* (*coeff-cube-to-path* (*circle-bot-edge*)) (*rec-join ?l*) ∧ *valid-chain-list ?l* ∧ *?l* ≠ [] **using** *a b c* **by** *auto*
qed
ultimately
show (∀ *cube*∈*?one-chaini*. *chain-reparam-weak-path* (*rec-join* [*cube*]) (*?f cube*))
by (*auto simp add*: *circle-cube-boundary-to-polarcircle-def UNION-eq circle-cube-boundary-explicit*)
show (∀ *p*∈*?one-chaini*. ∀ *p′*∈*?one-chaini*. *p* ≠ *p′* ⟶ *?f p* ∩ *?f p′* = {})
using *circle-arcs-neq circle-arcs-neq-2*
apply (*auto simp add*: *circle-cube-boundary-to-polarcircle-def UNION-eq circle-cube-boundary-explicit neq-commute d-gt-0*)
using *circle-top-bot-edges-neq′ d-gt-0* **apply** *auto*[*1*]
apply (*smt atLeastAtMost-iff divide-less-eq-1-pos zero-less-divide-1-iff*)
apply (*smt atLeastAtMost-iff divide-less-eq-1-pos zero-less-divide-iff*)
apply (*smt atLeastAtMost-iff divide-cancel-left divide-less-eq-1-pos field-sum-of-halves zero-less-divide-1-iff*)
done
show (∀ *x*∈*?one-chaini*. *finite* (*?f x*))
by (*auto simp add*: *circle-cube-boundary-to-polarcircle-def circle-cube-boundary-explicit*)
qed

show (∀ (*k*, *γ*)∈*?pi*. *point-path* *γ*)
using *d-gt-0* **by** (*auto simp add*: *point-path-def circle-left-edge-def circle-right-edge-def*)
let *?f* = *rot-circle-cube-boundary-to-polarcircle*
let *?one-chain2.0* = *two-chain-vertical-boundary* {*rot-circle-cube*} − *?pj*
show *chain-reparam-chain′* (*two-chain-vertical-boundary* {*rot-circle-cube*} − *?pj*) *?subdiv*
unfolding *chain-reparam-chain′-def*
proof (*intro exI conjI*)
have *rw*: *?one-chain2.0* = {*rot-circle-left-edge*, *rot-circle-right-edge*}
by(*auto simp add*: *rot-circle-cube-vertical-boundary-explicit two-chain-vertical-boundary-def*)
show ⋃ (*?f* ‘ *?one-chain2.0*) = *?subdiv*
using *rot-circle-right-top-edges-neq′*
by (*auto simp add*: *rot-circle-cube-boundary-to-polarcircle-def rw*)
show (∀ *cube*∈*?one-chain2.0*. *chain-reparam-weak-path* (*rec-join* [*cube*]) (*?f* *cube*))
proof (*clarsimp simp add*: *rot-circle-cube-boundary-to-polarcircle-def rw*, *intro conjI*)
let *?l* = [*subcube* (*1/4*) (*1/2*) (*1*, *circle-polar*), *subcube* (*1/2*) (*3/4*) (*1*, *circle-polar*)]
show *chain-reparam-weak-path* (*coeff-cube-to-path* (*rot-circle-left-edge*)) {*subcube* (*1/4*) (*1/2*) (*1*, *circle-polar*), *subcube* (*1/2*) (*3/4*) (*1*, *circle-polar*)}
unfolding *chain-reparam-weak-path-def*
proof (*intro exI conjI*)
show *valid-chain-list ?l*
by (*auto simp add*: *subcube-def pathfinish-def pathstart-def reversepath-def subpath-def joinpaths-def*)
show *reparam* (*coeff-cube-to-path rot-circle-left-edge*) (*rec-join ?l*)
using *rot-circ-left-edge-reparam-polar-circ-split* **by** *auto*
show *distinct ?l*
apply *simp*
apply (*subst neq-commute*)
apply (*simp add*: *circle-arcs-neq*)
done
qed *auto*
show *chain-reparam-weak-path* (*coeff-cube-to-path* (*rot-circle-right-edge*)) {*subcube* (*3/4*) *1* (*1*, *circle-polar*), *subcube* *0* (*1/4*) (*1*, *circle-polar*)}
unfolding *chain-reparam-weak-path-def*
proof (*intro exI conjI*)
let *?l* = [*subcube* (*3/4*) *1* (*1*, *circle-polar*), *subcube* *0* (*1/4*) (*1*, *circle-polar*)]
show *valid-chain-list ?l*
by (*auto simp add*: *circle-polar-def subcube-def pathfinish-def pathstart-def reversepath-def subpath-def joinpaths-def*)
show *reparam* (*coeff-cube-to-path rot-circle-right-edge*) (*rec-join ?l*)
using *rot-circ-right-edge-reparam-polar-circ-split* **by** *auto*
show *distinct ?l*
by (*simp add*: *circle-arcs-neq*)
qed *auto*
qed
show (∀ *p*∈*?one-chain2.0*. ∀ *p′*∈*?one-chain2.0*. *p* ≠ *p′* ⟶ *?f p* ∩ *?f p′* = {})

using *circle-arcs-neq circle-arcs-neq-2*
apply (*auto simp add*: *rot-circle-cube-boundary-to-polarcircle-def neq-commute*)
apply (*metis add.right-neutral divide-less-eq-1-pos dual-order.order-iff-strict num.distinct*(*1*) *one-less-numeral-iff prod.sel*(*1*) *prod.sel*(*2*) *semiring-norm*(*68*) *sub-cube-def zero-less-divide-1-iff zero-less-numeral*)
apply (*smt field-sum-of-halves*)
done
show (∀ *x*∈*?one-chain2.0*. *finite* (*?f x*))
by (*auto simp add*: *rot-circle-cube-boundary-to-polarcircle-def UNION-eq rw*)
qed
show (∀ (*k*, *γ*)∈*?pj*. *point-path γ*)
using *d-gt-0* **by** (*auto simp add*: *point-path-def rot-circle-top-edge-def rot-circle-bot-edge-def*)
qed
then show *common-sudiv-exists* (*two-chain-vertical-boundary* {*rot-circle-cube*}) (*boundary circle-cube*) ∨
common-reparam-exists (*boundary circle-cube*) (*two-chain-vertical-boundary* {*rot-circle-cube*})
by *blast*
qed

lemma *GreenThm-cirlce*:
assumes ∀ *twoC*∈{*circle-cube*}. *analytically-valid* (*cubeImage twoC*) (*λx. F x · i*) *j*
∀ *twoC*∈{*rot-circle-cube*}. *analytically-valid* (*cubeImage twoC*) (*λx. F x · j*) *i*
shows *integral* (*cubeImage* (*circle-cube*)) (*λx. partial-vector-derivative* (*λx. F x · j*) *i x* − *partial-vector-derivative* (*λx. F x · i*) *j x*) =
one-chain-line-integral F {*i*, *j*} (*boundary* (*circle-cube*))
proof(*rule green-typeI-typeII-chain.GreenThm-typeI-typeII-divisible-region-finite-holes*[*of* (*cubeImage* (*circle-cube*)) *i j F* {*circle-cube*} {*rot-circle-cube*},
OF - - - *circle-cube-is-only-horizontal-div-of-rot* -], *auto*)
show ⋀ *a b*. (*a*, *b*) ∈ *boundary circle-cube* ⟹ *valid-path b* **using** *circle-cube-boundary-valid* **by** *auto*
show *green-typeI-typeII-chain* (*cubeImage circle-cube*) *i j F* {*circle-cube*} {*rot-circle-cube*}
using *assms*
proof(*auto simp add*: *green-typeI-typeII-chain-def green-typeI-chain-def green-typeII-chain-def green-typeI-chain-axioms-def green-typeII-chain-axioms-def*
intro!: *circle-cube-is-type-I rot-circle-cube-is-type-II d-gt-0 R2-axioms*)
show *gen-division* (*cubeImage circle-cube*) {*cubeImage circle-cube*} **by** (*simp add*: *gen-division-def*)
show *gen-division* (*cubeImage* (*circle-cube*)) ({*cubeImage rot-circle-cube*})
using *rot-circle-div-circle d-gt-0* **by** *auto*
show *valid-two-chain* {*rot-circle-cube*} *valid-two-chain* {*circle-cube*}
apply (*auto simp add*: *valid-two-chain-def*)
using *rot-circle-cube-valid-two-cube circle-cube-valid-two-cube assms*(*1*) **by** *auto*
qed
show *only-vertical-division* (*boundary* (*circle-cube*)) {*circle-cube*}
using *twoChainVertDiv-of-itself*[*of* {*circle-cube*}]
apply(*simp add*: *two-chain-boundary-def*)

using *circle-cube-boundary-valid*
by *auto*
qed
end
end

3 The Diamond Example

theory *DiamExample*
imports *Green SymmetricR2Shapes*
begin

lemma *abs-if′*:
fixes *a* :: *′a* :: {*abs-if*,*ordered-ab-group-add*}
shows $|a| = (if\ a \leq 0\ then\ -\ a\ else\ a)$
by (*simp add*: *abs-if dual-order.order-iff-strict*)

locale *diamond* = *R2* +
fixes *d*::*real*
assumes *d-gt-0*: $0 < d$
begin

definition *diamond-y-gen* :: *real* ⇒ *real* **where**
diamond-y-gen ≡$\lambda t.\ 1/2 - |t|$

definition *diamond-cube-gen*:: ((*real* ∗ *real*) ⇒ (*real* ∗ *real*)) **where**
diamond-cube-gen ≡ (λ(*x*,*y*). (*d* ∗ *x-coord x*, (*2* ∗ *y* − *1*) ∗ (*d* ∗ *diamond-y-gen* (*x-coord x*))))

lemma *diamond-y-gen-valid*:
assumes $a \leq 0\ 0 \leq b$
shows *diamond-y-gen piecewise-C1-differentiable-on* {*a*..*b*}
unfolding *piecewise-C1-differentiable-on-def diamond-y-gen-def*
proof (*intro conjI exI*)
show *continuous-on* {*a*..*b*} (λ*t*. *1* / *2* − |*t*|)
by (*intro continuous-intros*)
show *finite*{*0*}
by *simp*
show (λ*t*. *1* / *2* − |*t*|) *C1-differentiable-on* {*a*..*b*} − {*0*}
by (*intro derivative-intros*) *auto*
qed

lemma *diamond-cube-gen-boundary-valid*:
assumes (*k*,γ)∈*boundary* (*diamond-cube-gen*)
shows *valid-path* γ
using *assms*
proof (*auto simp add*: *valid-path-def boundary-def horizontal-boundary-def vertical-boundary-def diamond-cube-gen-def*)
have *rw2*: (λ*x*. *diamond-y-gen* (*x-coord x*)) = *diamond-y-gen o x-coord* **by** *auto*

note [*derivative-intros*] = *C1-differentiable-on-pair pair-prod-smooth-pw-smooth scale-piecewise-C1-differentiable-on piecewise-C1-differentiable-neg piecewise-C1-differentiable-compose diamond-y-gen-valid*
show (λx. ($d * $ *x-coord x*, $-$ ($d *$ *diamond-y-gen* (*x-coord x*)))) *piecewise-C1-differentiable-on* {*0..1*}
apply(*auto intro*!: *derivative-intros*)+
apply (*auto simp add*: *x-coord-smooth rw2*)
by(*auto intro*!: *derivative-intros simp add*: *x-coord-img x-coord-back-img C1-differentiable-imp-piecewise*[*OF x-coord-smooth*])+
show (λx. ($d * $ *x-coord x*, $d *$ *diamond-y-gen* (*x-coord x*))) *piecewise-C1-differentiable-on* {*0..1*}
apply(*auto intro*!: *derivative-intros*)+
apply (*auto simp add*: *x-coord-smooth rw2*)
by(*auto intro*!: *derivative-intros simp add*: *x-coord-img x-coord-back-img C1-differentiable-imp-piecewise*[*OF x-coord-smooth*])+
qed

definition *diamond-x* **where**
diamond-x ≡ λt. (t − *1/2*) ∗ d

definition *diamond-y* **where**
diamond-y ≡ λt. d/*2* − |t|

definition *diamond-cube* **where**
diamond-cube = (λ(*x*,*y*). (*diamond-x x*, (*2* ∗ *y* − *1*) ∗ (*diamond-y* (*diamond-x x*))))

definition *rot-diamond-cube* **where**
rot-diamond-cube = *prod.swap o* (*diamond-cube*) *o prod.swap*

lemma *diamond-eq-characterisations*:
shows *diamond-cube* (*x*,*y*)= *diamond-cube-gen* (*x*,*y*)
by(*auto simp add*: *diamond-cube-def diamond-cube-gen-def diamond-x-def x-coord-def diamond-y-def diamond-y-gen-def d-gt-0 field-simps mult-le-0-iff abs-if split*: *if-split-asm*)

lemma *diamond-eq-characterisations-fun*: *diamond-cube* = *diamond-cube-gen*
using *diamond-eq-characterisations* **by** *auto*

lemma *diamond-y-valid*:
shows *diamond-y piecewise-C1-differentiable-on* {−*d*/*2*..*d*/*2*} (**is** *?P*)
(λx. *diamond-y x*) *piecewise-C1-differentiable-on* {−*d*/*2*..*d*/*2*} (**is** *?Q*)
proof −
have *f0*: *finite* {*0*}
by *simp*
show *?P ?Q*
unfolding *piecewise-C1-differentiable-on-def diamond-y-def*
by (*fastforce intro*!: *f0 continuous-intros derivative-intros*)+
qed

lemma *diamond-cube-boundary-valid*:
  assumes $(k,\gamma) \in$ *boundary* (*diamond-cube*)
  shows *valid-path* γ
  using *diamond-cube-gen-boundary-valid assms d-gt-0*
  by(*simp add*: *diamond-eq-characterisations-fun*)

lemma *diamond-cube-is-type-I*:
  shows *typeI-twoCube* (*diamond-cube*)
  unfolding *typeI-twoCube-def*
proof (*intro exI conjI ballI*)
  show $-d/2 < d/2$
    using *d-gt-0* **by** *auto*
  show $\bigwedge x.\ x \in \{- d\ /\ 2..d\ /\ 2\} \Longrightarrow -$ *diamond-y* $x \leq$ *diamond-y* x
    using *diamond-y-def* **by** *auto*
  have *f0*: *finite* $\{0\}$
    by *simp*
  show *diamond-y piecewise-C1-differentiable-on* $\{- d\ /\ 2..d\ /\ 2\}$
      ($\lambda x. -$ *diamond-y* x) *piecewise-C1-differentiable-on* $\{- d\ /\ 2..d\ /\ 2\}$
    unfolding *diamond-y-def piecewise-C1-differentiable-on-def*
    by (*rule conjI exI f0 continuous-intros derivative-intros* | *force*)+
  show *diamond-cube* =
    $(\lambda(x, y).\ ((1 - x) * (- d\ /\ 2) + x * (d\ /\ 2),$
      $(1 - y) * -$ *diamond-y* $((1 - x) * (- d\ /\ 2) + x * (d\ /\ 2)) +$
      $y *$ *diamond-y* $((1 - x) * (- d\ /\ 2) + x * (d\ /\ 2))))$
     by (*auto simp*: *diamond-cube-def diamond-x-def diamond-y-def divide-simps*
algebra-simps)
qed

lemma *diamond-cube-valid-two-cube*:
  shows *valid-two-cube* (*diamond-cube*)
   apply (*auto simp add*: *valid-two-cube-def boundary-def horizontal-boundary-def*
vertical-boundary-def diamond-cube-def
      diamond-x-def card-insert-if)
   apply (*metis* (*no-types*, *opaque-lifting*) *cancel-comm-monoid-add-class.diff-cancel*
d-gt-0 mult.commute mult-2 mult-zero-right order-less-irrefl prod.inject field-sum-of-halves)
  apply (*metis* (*no-types*, *opaque-lifting*) *add-diff-cancel-right′ d-gt-0 mult-cancel-left*
mult-zero-right order-less-irrefl prod.inject)
  done

lemma *rot-diamond-cube-boundary-valid*:
  assumes $(k,\gamma) \in$ *boundary* (*rot-diamond-cube*)
  shows *valid-path* γ
  using *d-gt-0 swap-valid-boundaries diamond-cube-boundary-valid*
  using *assms diamond-cube-is-type-I rot-diamond-cube-def* **by** *fastforce*

lemma *rot-diamond-cube-is-type-II*:
  shows *typeII-twoCube* (*rot-diamond-cube*)
  using *d-gt-0 swap-typeI-is-typeII diamond-cube-is-type-I*
  by (*auto simp add*: *rot-diamond-cube-def*)

lemma *rot-diamond-cube-valid-two-cube*: *valid-two-cube* (*rot-diamond-cube*)
using *valid-cube-valid-swap diamond-cube-valid-two-cube d-gt-0 rot-diamond-cube-def*
by *force*

definition *diamond-top-edges* **where**
diamond-top-edges = (− *1*::*int*, λ*x*. (*diamond-x x*, *diamond-y* (*diamond-x x*)))

definition *diamond-bot-edges* **where**
diamond-bot-edges = (*1*::*int*, λ*x*. (*diamond-x x*, − *diamond-y* (*diamond-x x*)))

lemma *diamond-cube-boundary-explicit*:
boundary (*diamond-cube*) =
{*diamond-top-edges*,
diamond-bot-edges,
(− *1*::*int*, λ*y*. (*diamond-x 0*, (*2* ∗ *y* − *1*) ∗ *diamond-y* (*diamond-x 0*))),
(*1*::*int*, λ*y*. (*diamond-x 1*, (*2* ∗ *y* − *1*) ∗ *diamond-y* (*diamond-x 1*)))}
by (*auto simp only*: *diamond-top-edges-def diamond-bot-edges-def valid-two-cube-def boundary-def horizontal-boundary-def vertical-boundary-def diamond-cube-def Un-iff algebra-simps*)

definition *diamond-top-left-edge* **where**
diamond-top-left-edge = (− *1*::*int*, (λ*x*. (*diamond-x* (*1/2* ∗ *x*), (*diamond-x* (*1/2* ∗ *x*)) + *d/2*)))

definition *diamond-top-right-edge* **where**
diamond-top-right-edge = (− *1*::*int*, (λ*x*. (*diamond-x* (*1/2* ∗ *x* + *1/2*), (−(*diamond-x* (*1/2* ∗ *x* + *1/2*)) + *d/2*))))

definition *diamond-bot-left-edge* **where**
diamond-bot-left-edge = (*1*::*int*, (λ*x*. (*diamond-x* (*1/2* ∗ *x*), − (*diamond-x* (*1/2* ∗ *x*) + *d/2*))))

definition *diamond-bot-right-edge* **where**
diamond-bot-right-edge = (*1*::*int*, (λ*x*. (*diamond-x* (*1/2* ∗ *x* + *1/2*), − (−(*diamond-x* (*1/2* ∗ *x* + *1/2*)) + *d/2*))))

lemma *diamond-edges-are-valid*:
valid-path (*snd* (*diamond-top-left-edge*))
valid-path (*snd* (*diamond-top-right-edge*))
valid-path (*snd* (*diamond-bot-left-edge*))
valid-path (*snd* (*diamond-bot-right-edge*))
by (*auto simp add*: *valid-path-def diamond-top-left-edge-def diamond-bot-right-edge-def diamond-bot-left-edge-def diamond-top-right-edge-def diamond-x-def*)

definition *diamond-cube-boundary-to-subdiv* **where**
diamond-cube-boundary-to-subdiv (*gamma*::(*int* × (*real* ⇒ *real* × *real*))) ≡
if (*gamma* = *diamond-top-edges*) *then*
{*diamond-top-left-edge*, *diamond-top-right-edge*}

else if (*gamma* = *diamond-bot-edges*) *then*
{*diamond-bot-left-edge*, *diamond-bot-right-edge*}
else {}

lemma *rot-diam-edge-1*:
(*1*::*int*, λ*x*::*real*. ((*x*::*real*) ∗ (*2* ∗ *diamond-y* (*diamond-x 0*)) − *1* ∗ *diamond-y* (*diamond-x 0*), *diamond-x 0*)) =
(*1*, λ*x*. (*x* ∗ (*2* ∗ *diamond-y* (*diamond-x 0*)) − (*diamond-y* (*diamond-x 0*)), *diamond-x 0*))
by (*auto simp add*: *algebra-simps*)

definition *diamond-left-edges* **where**
diamond-left-edges = (− *1*, λ*y*. (− *diamond-y* (*diamond-x y*), *diamond-x y*))

definition *diamond-right-edges* **where**
diamond-right-edges = (*1*, λ*y*. (*diamond-y* (*diamond-x y*), *diamond-x y*))

lemma *rot-diamond-cube-boundary-explicit*:
boundary (*rot-diamond-cube*) = {(*1*::*int*, λ*x*::*real*. ((*2* ∗ *x* − *1*) ∗ *diamond-y* (*diamond-x 0*), *diamond-x 0*)),
(− *1*, λ*x*. ((*2* ∗ *x* − *1*) ∗ *diamond-y* (*diamond-x 1*), *diamond-x 1*)),
diamond-left-edges, *diamond-right-edges*}
proof −
have *boundary* (*rot-diamond-cube*) = { (*1*::*int*, λ*x*::*real*. ((*2* ∗ *x* − *1*) ∗ *diamond-y* (*diamond-x 0*), *diamond-x 0*)),
(− *1*, λ*x*. ((*2* ∗ *x* − *1*) ∗ *diamond-y* (*diamond-x 1*), *diamond-x 1*)),
(− *1*, λ*y*. ((*2* ∗ *0* − *1*) ∗ *diamond-y* (*diamond-x y*), *diamond-x y*)),
(*1*, λ*y*. ((*2* ∗ *1* − *1*) ∗ *diamond-y* (*diamond-x y*), *diamond-x y*))}
by (*auto simp only*: *valid-two-cube-def boundary-def horizontal-boundary-def vertical-boundary-def rot-diamond-cube-def diamond-cube-def o-def swap-simp Un-iff*)
then show *?thesis*
by (*auto simp add*: *diamond-left-edges-def diamond-right-edges-def*)
qed

lemma *rot-diamond-cube-vertical-boundary-explicit*:
vertical-boundary (*rot-diamond-cube*) = {*diamond-left-edges*, *diamond-right-edges*}
proof −
have *vertical-boundary* (*rot-diamond-cube*) = {(− *1*::*int*, λ*y*. ((*2* ∗ *0* − *1*) ∗ *diamond-y* (*diamond-x y*), *diamond-x y*)),
(*1*, λ*y*. ((*2* ∗ *1* − *1*) ∗ *diamond-y* (*diamond-x y*), *diamond-x y*))}
by (*auto simp only*: *valid-two-cube-def boundary-def horizontal-boundary-def vertical-boundary-def rot-diamond-cube-def diamond-cube-def o-def swap-simp Un-iff*)
then show *?thesis*
by (*auto simp add*: *diamond-left-edges-def diamond-right-edges-def*)
qed

definition *rot-diamond-cube-boundary-to-subdiv* **where**

rot-diamond-cube-boundary-to-subdiv (*gamma*::(*int* × (*real* ⇒ *real* × *real*))) ≡
if (*gamma* = *diamond-left-edges*) *then* {*diamond-bot-left-edge* , *diamond-top-left-edge*}
else if (*gamma* = *diamond-right-edges*) *then* {*diamond-bot-right-edge*, *diamond-top-right-edge*}
else {}

definition *diamond-boundaries-reparam-map* **where**
diamond-boundaries-reparam-map ≡ *id*

lemma *diamond-boundaries-reparam-map-bij*:
bij (*diamond-boundaries-reparam-map*)
by(*auto simp add*: *bij-def full-SetCompr-eq*[*symmetric*] *diamond-boundaries-reparam-map-def*)

lemma *diamond-bot-edges-neq-diamond-top-edges*:
diamond-bot-edges ≠ *diamond-top-edges*
by(*simp add*: *diamond-bot-edges-def diamond-top-edges-def*)

lemma *diamond-top-left-edge-neq-diamond-top-right-edge*:
diamond-top-left-edge ≠ *diamond-top-right-edge*
apply (*simp add*: *diamond-top-left-edge-def diamond-top-right-edge-def diamond-x-def diamond-y-def*)
using *d-gt-0*
apply (*auto simp add*: *algebra-simps divide-simps*)
by (*metis* (*no-types*, *opaque-lifting*) *diff-zero div-0 divide-divide-eq-right order-less-irrefl prod.inject field-sum-of-halves*)

lemma *neqs1*:
shows (λ*x*. (*diamond-x x*, *diamond-y* (*diamond-x x*))) ≠ (λ*x*. (*diamond-x x*, − *diamond-y* (*diamond-x x*)))
and (λ*y*. (− *diamond-y* (*diamond-x y*), *diamond-x y*)) ≠ (λ*y*. (*diamond-y* (*diamond-x y*), *diamond-x y*))
and (λ*x*. (*diamond-x*(*x*/*2* + *1*/*2*), *diamond-x*(*x*/*2* + *1*/*2*) − *d*/*2*)) ≠ (λ*x*. (*diamond-x*(*x*/*2*), − *diamond-x*(*x*/*2*) − *d*/*2*))
and (λ*x*. (*diamond-x*(*x*/*2* + *1*/*2*), *d*/*2* − *diamond-x*(*x*/*2* + *1*/*2*))) ≠ (λ*x*. (*diamond-x*(*x*/*2*), *diamond-x*(*x*/*2*) + *d*/*2*))
and (λ*x*. (*diamond-x*(*x*/*2*), − *diamond-x*(*x*/*2*) − *d*/*2*)) ≠ (λ*x*. (*diamond-x*(*x*/*2* + *1*/*2*), *diamond-x*(*x*/*2* + *1*/*2*) − *d*/*2*))
and (λ*x*. (*diamond-x*(*x*/*2*), *diamond-x*(*x*/*2*) + *d*/*2*)) ≠ (λ*x*. (*diamond-x*(*x*/*2* + *1*/*2*), *d*/*2* − *diamond-x*(*x*/*2* + *1*/*2*)))
using *d-gt-0* **by** (*auto simp*: *diamond-x-def diamond-y-def dest*: *fun-cong* [**where** *x* = *1*/*2*])

lemma *neqs2*:
shows (λ*x*. (*diamond-x x*, *diamond-y* (*diamond-x x*))) ≠ (λ*x*. ((*2* ∗ *x* − *1*) ∗ *diamond-y* (*diamond-x 1*), *diamond-x 1*))
and (λ*x*. (*diamond-x x*, − *diamond-y* (*diamond-x x*))) ≠ (λ*x*. ((*2* ∗ *x* − *1*) ∗ *diamond-y* (*diamond-x 0*), *diamond-x 0*))
using *d-gt-0* **by** (*auto simp*: *diamond-x-def diamond-y-def dest*: *fun-cong* [**where** *x* = *1*])

lemma *diamond-cube-is-only-horizontal-div-of-rot*:
shows *only-horizontal-division* (*boundary* (*diamond-cube*)) {*rot-diamond-cube*}
unfolding *only-horizontal-division-def*
proof (*rule exI* [*of* - {}], *simp*, *intro conjI ballI*)
show *finite* (*boundary diamond-cube*)
by (*simp add*: *boundary-def horizontal-boundary-def vertical-boundary-def*)
show *boundary-chain* (*boundary diamond-cube*)
by (*simp add*: *two-cube-boundary-is-boundary*)
show $\bigwedge x.\ x \in$ *boundary diamond-cube* $\Longrightarrow$ *case x of* (*k*, *x*) $\Rightarrow$ *valid-path x*
using *diamond-cube-boundary-valid* **by** *blast*
let *?𝒱* = (*boundary* (*diamond-cube*))
have *0*: *finite ?𝒱*
by (*auto simp add*: *boundary-def horizontal-boundary-def vertical-boundary-def*)
have *1*: *boundary-chain ?𝒱* **using** *two-cube-boundary-is-boundary* **by** *auto*
let *?subdiv* = {*diamond-top-left-edge*, *diamond-top-right-edge*, *diamond-bot-left-edge*, *diamond-bot-right-edge*}
let *?pi* = {(*1*::*int*, λ*x*. ((*2* ∗ *x* − *1*) ∗ *diamond-y* (*diamond-x 0*), *diamond-x 0*)),
(− *1*, λ*x*. ((*2* ∗ *x* − *1*) ∗ *diamond-y* (*diamond-x 1*), *diamond-x 1*))}
let *?pj* = {(−*1*::*int*, λ*y*. (*diamond-x 0*, (*2* ∗ *y* − *1*) ∗ *diamond-y* (*diamond-x 0*))),
(*1*, λ*y*. (*diamond-x 1*, (*2* ∗ *y* − *1*) ∗ *diamond-y* (*diamond-x 1*)))}
let *?f* = *diamond-cube-boundary-to-subdiv*
have *2*: *common-sudiv-exists* (*two-chain-vertical-boundary* {*rot-diamond-cube*}) *?𝒱*
unfolding *common-sudiv-exists-def*
proof (*intro exI conjI*)
show *chain-subdiv-chain* (*boundary* (*diamond-cube*) − *?pj*) *?subdiv*
unfolding *chain-subdiv-chain-character*
proof (*intro exI conjI*)
have *1*: (*boundary* (*diamond-cube*)) − *?pj* = {*diamond-top-edges*, *diamond-bot-edges*}
apply (*auto simp add*: *diamond-cube-boundary-explicit diamond-x-def diamond-top-edges-def diamond-bot-edges-def*)
apply (*metis* (*no-types*, *opaque-lifting*) *abs-zero cancel-comm-monoid-add-class.diff-cancel diamond-x-def diamond-y-def diff-0 minus-diff-eq mult.commute mult-zero-right prod.inject neqs2*)
by (*metis* (*no-types*, *opaque-lifting*) *cancel-comm-monoid-add-class.diff-cancel d-gt-0 divide-eq-0-iff linorder-not-le mult.commute mult-zero-right order-refl prod.sel*(*1*) *zero-neq-numeral*)
then show $\bigcup$ (*?f* ‘ (*boundary* (*diamond-cube*) − *?pj*)) = *?subdiv*
by (*auto simp add*: *diamond-top-edges-def diamond-bot-edges-def diamond-cube-boundary-to-subdiv-def*)
have *chain-subdiv-path* (*reversepath* (λ*x*. (*diamond-x x*, *diamond-y* (*diamond-x x*))))
{(− *1*::*int*, λ*x*. (*diamond-x*(*x*/*2*), *diamond-x*(*x*/*2*) + *d*/*2*)),
(− *1*::*int*, λ*x*. (*diamond-x*(*x*/*2* + *1*/*2*), *d*/*2* − *diamond-x*(*x*/*2* + *1*/*2*)))}
proof −
let *?F* = λ*x*. (*diamond-x*(*x*/*2* + *1*/*2*), *d*/*2* − *diamond-x*(*x*/*2* + *1*/*2*))

let *?G* = *λx*. (*diamond-x*(*x*/*2*), *diamond-x*(*x*/*2*) + *d*/*2*)
have *dist*: *distinct* [(−*1*, *?F*), (−*1*, *?G*)]
using *d-gt-0* **by** (*auto simp*: *diamond-x-def diamond-y-def dest*: *fun-cong*)
have *rj*: *rec-join* [(−*1*, *?F*), (−*1*, *?G*)] = *reversepath* (*λx*. (*diamond-x x*, *diamond-y* (*diamond-x x*)))
using *d-gt-0* **by** (*auto simp add*: *subpath-def diamond-x-def diamond-y-def joinpaths-def reversepath-def algebra-simps divide-simps*)
have *pathstart ?F* = *pathfinish ?G*
using *d-gt-0* **by**(*auto simp add*: *subpath-def diamond-x-def diamond-y-def pathfinish-def pathstart-def*)
then have *val*: *valid-chain-list* [(−*1*, *?F*), (−*1*, *?G*)]
by (*auto simp add*: *join-subpaths-middle*)
show *?thesis*
using *chain-subdiv-path.I* [*OF dist rj val*]
by (*simp add*: *insert-commute*)
qed
moreover have *chain-subdiv-path* (*λx*. (*diamond-x x*, − *diamond-y* (*diamond-x x*)))
{(*1*::*int*, *λx*. (*diamond-x*(*x*/*2*), − *diamond-x*(*x*/*2*) − *d*/*2*)),
(*1*::*int*, *λx*. (*diamond-x*(*x*/*2* + *1*/*2*), *diamond-x*(*x*/*2* + *1*/*2*) − *d*/*2*))}
proof −
let *?F* = *λx*. (*diamond-x*(*x*/*2*), − *diamond-x*(*x*/*2*) − *d*/*2*)
let *?G* = *λx*. (*diamond-x*(*x*/*2* + *1*/*2*), *diamond-x*(*x*/*2* + *1*/*2*) − *d*/*2*)
have *dist*: *distinct* [(*1*, *?F*), (*1*, *?G*)]
using *d-gt-0* **by** (*auto simp*: *diamond-x-def diamond-y-def dest*: *fun-cong*)
have *rj*: *rec-join* [(*1*, *?F*), (*1*, *?G*)] = (*λx*. (*diamond-x x*, − *diamond-y* (*diamond-x x*)))
using *d-gt-0* **by** (*auto simp add*: *subpath-def diamond-x-def diamond-y-def joinpaths-def algebra-simps divide-simps*)
have *pathfinish ?F* = *pathstart ?G*
using *d-gt-0* **by**(*auto simp add*: *subpath-def diamond-x-def diamond-y-def pathfinish-def pathstart-def*)
then have *val*: *valid-chain-list* [(*1*, *?F*), (*1*, *?G*)]
by (*auto simp add*: *join-subpaths-middle*)
show *?thesis*
using *chain-subdiv-path.I* [*OF dist rj val*] **by** *simp*
qed
ultimately show ∗∗:
∀ (*k*::*int*, *γ*)∈*boundary* (*diamond-cube*) − *?pj*.
if k = (*1*::*int*) *then chain-subdiv-path γ* (*?f* (*k*::*int*, *γ*))
else chain-subdiv-path (*reversepath γ*) (*?f* (*k*::*int*, *γ*))
∀ *p*∈*boundary* (*diamond-cube*) − *?pj*. ∀ *p'*∈*boundary* (*diamond-cube*) − *?pj*.
p ≠ *p'* ⟶ *?f p* ∩ *?f p'* = {}
∀ *x*∈*boundary* (*diamond-cube*) − *?pj*. *finite* (*?f x*)
by(*simp-all add*: *diamond-cube-boundary-to-subdiv-def UNION-eq 1 diamond-top-edges-def diamond-bot-edges-def diamond-bot-left-edge-def diamond-bot-right-edge-def diamond-top-left-edge-def diamond-top-right-edge-def neqs1*)
qed

have ∗: $\bigwedge f.\ \bigcup\ (f\ `\ \{$*rot-diamond-cube*$\}) = f$ (*rot-diamond-cube*) **by** *auto*
show *chain-subdiv-chain* (*two-chain-vertical-boundary* {*rot-diamond-cube*} − *?pi*) *?subdiv*
unfolding *chain-subdiv-chain-character two-chain-vertical-boundary-def* ∗
proof (*intro exI conjI*)
let *?f* = *rot-diamond-cube-boundary-to-subdiv*
have *1*: (*vertical-boundary* (*rot-diamond-cube*) − *?pi*) = {*diamond-left-edges*, *diamond-right-edges*}
apply (*auto simp add*: *rot-diamond-cube-vertical-boundary-explicit diamond-left-edges-def diamond-right-edges-def*)
apply (*metis* (*no-types*, *opaque-lifting*) *add.inverse-inverse add-diff-cancel-right′ diff-numeral-special*(*11*) *mult.left-neutral mult.right-neutral prod.inject neqs1*(*2*) *uminus-add-conv-diff*)
by (*metis* (*no-types*, *opaque-lifting*) *diff-0 mult.left-neutral mult-minus-left mult-zero-right prod.inject neqs1*(*2*))
show $\bigcup$ (*?f* ‘ (*vertical-boundary* (*rot-diamond-cube*) − *?pi*)) = *?subdiv*
apply (*simp add*: *rot-diamond-cube-boundary-to-subdiv-def 1 UNION-eq subpath-def*)
apply (*auto simp add*: *set-eq-iff diamond-right-edges-def diamond-left-edges-def*)
done
have *chain-subdiv-path* (*reversepath* (λ*y*. (− *diamond-y* (*diamond-x y*), *diamond-x y*)))
{(*1*, λ*x*. (*diamond-x*(*x*/*2*), − *diamond-x*(*x*/*2*) − *d*/*2*)),
(−*1*, λ*x*. (*diamond-x*(*x*/*2*), *diamond-x*(*x*/*2*) + *d*/*2*))}
proof −
let *?F* = λ*x*. (*diamond-x*(*x*/*2*), − *diamond-x*(*x*/*2*) − *d*/*2*)
let *?G* = λ*x*. (*diamond-x*(*x*/*2*), *diamond-x*(*x*/*2*) + *d*/*2*)
have *dist*: *distinct* [(−*1*, *?G*), (*1*::*int*, *?F*)]
using *d-gt-0* **by** *simp*
have *rj*: *rec-join* [(−*1*, *?G*), (*1*::*int*, *?F*)] = *reversepath* (λ*y*. (− *diamond-y* (*diamond-x y*), *diamond-x y*))
using *d-gt-0* **by** (*auto simp add*: *subpath-def diamond-x-def diamond-y-def joinpaths-def reversepath-def algebra-simps divide-simps*)
have *pathstart ?G* = *pathstart ?F*
using *d-gt-0* **by**(*auto simp add*: *subpath-def diamond-x-def diamond-y-def pathstart-def*)
then have *val*: *valid-chain-list* [(−*1*, *?G*), (*1*::*int*, *?F*)]
by (*auto simp add*: *join-subpaths-middle*)
show *?thesis*
using *chain-subdiv-path.I* [*OF dist rj val*] **by** (*simp add*: *insert-commute*)
qed
moreover have *chain-subdiv-path* (λ*y*. (*diamond-y* (*diamond-x y*), *diamond-x y*))
{(*1*, λ*x*. (*diamond-x*(*x*/*2* + *1*/*2*), *diamond-x*(*x*/*2* + *1*/*2*) − *d*/*2*)),
(−*1*, λ*x*. (*diamond-x*(*x*/*2* + *1*/*2*), *d*/*2* − *diamond-x*(*x*/*2* + *1*/*2*)))}
proof −
let *?F* = λ*x*. (*diamond-x*(*x*/*2* + *1*/*2*), *d*/*2* − *diamond-x*(*x*/*2* + *1*/*2*))

let *?G* = λ*x*. (*diamond-x*(*x*/*2* + *1*/*2*), *diamond-x*(*x*/*2* + *1*/*2*) − *d*/*2*)
have *dist*: *distinct* [(*1*::*int*, *?G*), (−*1*, *?F*)]
by *simp*
have *rj*: *rec-join* [(*1*::*int*, *?G*), (−*1*, *?F*)] = (λ*y*. (*diamond-y* (*diamond-x* *y*), *diamond-x* *y*))
using *d-gt-0* **by** (*auto simp add*: *subpath-def diamond-x-def diamond-y-def joinpaths-def reversepath-def algebra-simps divide-simps*)
have *pathfinish ?F* = *pathfinish ?G*
using *d-gt-0* **by**(*auto simp add*: *subpath-def diamond-x-def diamond-y-def pathfinish-def pathstart-def*)
then have *val*: *valid-chain-list* [(*1*::*int*, *?G*), (−*1*, *?F*)]
by (*auto simp add*: *join-subpaths-middle*)
show *?thesis*
using *chain-subdiv-path.I* [*OF dist rj val*] **by** *simp*
qed
ultimately show ∀ (*k*, γ)∈*vertical-boundary* (*rot-diamond-cube*) − *?pi*.
if k = *1 then chain-subdiv-path* γ (*?f* (*k*, γ))
else chain-subdiv-path (*reversepath* γ) (*?f* (*k*, γ))
∀ *p*∈*vertical-boundary* (*rot-diamond-cube*) − *?pi*.
∀ *p*′∈*vertical-boundary* (*rot-diamond-cube*) − *?pi*.
p ≠ *p*′ ⟶ *?f p* ∩ *?f p*′ = {}
∀ *x*∈*vertical-boundary* (*rot-diamond-cube*) − *?pi*. *finite* (*?f x*)
by(*auto simp add*: *rot-diamond-cube-boundary-to-subdiv-def 1 diamond-left-edges-def*

diamond-right-edges-def diamond-bot-left-edge-def diamond-bot-right-edge-def

diamond-top-left-edge-def diamond-top-right-edge-def neqs1)
qed
show (∀ (*k*::*int*, γ)∈*?subdiv*. *valid-path* γ)
by (*simp add*: *diamond-edges-are-valid prod.case-eq-if set-eq-iff*)
show *boundary-chain ?subdiv*
by (*auto simp add*: *boundary-chain-def diamond-top-left-edge-def diamond-top-right-edge-def diamond-bot-left-edge-def diamond-bot-right-edge-def*)
show (∀ (*k*, γ)∈*?pi*. *point-path* γ)
using *d-gt-0* **by** (*auto simp add*: *point-path-def diamond-x-def diamond-y-def*)
show (∀ (*k*, γ)∈*?pj*. *point-path* γ)
using *d-gt-0* **by** (*auto simp add*: *point-path-def diamond-x-def diamond-y-def*)
qed
show *common-sudiv-exists* (*two-chain-vertical-boundary* {*rot-diamond-cube*}) (*boundary diamond-cube*) ∨
common-reparam-exists (*boundary diamond-cube*) (*two-chain-vertical-boundary* {*rot-diamond-cube*})
using *0 1 2 diamond-cube-boundary-valid* **by** *auto*
qed

abbreviation *rot-y t1 t2* ≡ (*t1* − *1*/*2*) / (*2*∗ *diamond-y-gen* (*x-coord* (*rot-x t1 t2*))) + *1*/*2*

lemma *rot-y-ivl*:

assumes (*0::real*) ≤ *x x* ≤ *1 0* ≤ *y y* ≤ *1*
shows *0* ≤ *rot-y x y* ∧ *rot-y x y* ≤ *1*
using *assms*
apply(*auto simp add*: *x-coord-def diamond-y-gen-def algebra-simps divide-simps linorder-class.not-le abs-if*)
using *mult-nonneg-nonneg* **apply** *fastforce*
using *dual-order.order-iff-strict* **apply** *fastforce*
apply (*sos* (((((*A<0* ∗ *A<1*) ∗ *R<1*) + (((*A<=4* ∗ (*A<1* ∗ *R<1*)) ∗ (*R<1/2* ∗ [*1*]^*2*)) + (((*A<=3* ∗ (*A<0* ∗ *R<1*)) ∗ (*R<1/2* ∗ [*1*]^*2*)) + ((*A<=0* ∗ (*A<=2* ∗ (*A<=3* ∗ *R<1*))) ∗ (*R<1* ∗ [*1*]^*2*)))))) & ((((*A<0* ∗ *A<1*) ∗ *R<1*) + (((*A<=4* ∗ (*A<1* ∗ *R<1*)) ∗ (*R<1/3* ∗ [*1*]^*2*)) + (((*A<=4* ∗ (*A<0* ∗ *R<1*)) ∗ (*R<1/3* ∗ [*1*]^*2*)) + ((*A<=3* ∗ (*A<=4* ∗ *R<1*)) ∗ (*R<1/3* ∗ [*1*]^*2*))))))))
using *assms*(*1*) *mult-left-le-one-le* **apply** *blast*
using *affine-ineq* **apply** *fastforce+*
done

lemma *diamond-gen-eq-rot-diamond*:
assumes*0* ≤ *x x* ≤ *1 0* ≤ *y y* ≤ *1*
shows (*diamond-cube-gen* (*x*, *y*)) = (*rot-diamond-cube* (*rot-y x y*, *rot-x x y*))
proof
show *snd* (*diamond-cube-gen* (*x*, *y*)) = *snd* (*rot-diamond-cube* (*rot-y x y*, *rot-x x y*))
apply(*simp only*: *rot-diamond-cube-def diamond-eq-characterisations-fun*)
by(*auto simp add*: *diamond-cube-gen-def x-coord-def diamond-y-gen-def algebra-simps divide-simps*)
show *fst* (*diamond-cube-gen* (*x*, *y*)) = *fst* (*rot-diamond-cube* (*rot-y x y*, *rot-x x y*))
using *assms*
apply(*auto simp add*: *diamond-cube-gen-def rot-diamond-cube-def diamond-eq-characterisations-fun*)
apply(*auto simp add*: *x-coord-def diamond-y-gen-def algebra-simps divide-simps abs-if split*: *if-split-asm*)
apply *sos+*
done
qed

lemma *rot-diamond-eq-diamond-gen*:
assumes *0* ≤ *x x* ≤ *1 0* ≤ *y y* ≤ *1*
shows *rot-diamond-cube* (*x*, *y*) = *diamond-cube-gen* (*rot-x y x*, *rot-y y x*)
proof
show *fst* (*rot-diamond-cube* (*x*, *y*)) = *fst* (*diamond-cube-gen* (*rot-x y x*, *rot-y y x*))
apply(*simp only*: *rot-diamond-cube-def diamond-eq-characterisations-fun*)
by(*auto simp add*: *diamond-cube-gen-def x-coord-def diamond-y-gen-def algebra-simps divide-simps*)
show *snd* (*rot-diamond-cube* (*x*, *y*)) = *snd* (*diamond-cube-gen* (*rot-x y x*, *rot-y y x*))
using *assms*
apply(*auto simp add*: *diamond-cube-gen-def rot-diamond-cube-def diamond-eq-characterisations-fun*)
apply(*auto simp add*: *x-coord-def diamond-y-gen-def algebra-simps divide-simps*

abs-if split: *if-split-asm*)
  apply *sos+*
  done
qed

lemma *rot-img-eq*: *cubeImage* (*diamond-cube-gen*) = *cubeImage* (*rot-diamond-cube*)
proof(*auto simp add*: *cubeImage-def image-def cbox-def real-pair-basis*)
 show ∃ *a*≥*0*. *a* ≤ *1* ∧ (∃ *b*≥*0*. *b* ≤ *1* ∧ *diamond-cube-gen* (*x*, *y*) = *rot-diamond-cube* (*a*, *b*))
  if *0* ≤ *x x* ≤ *1 0* ≤ *y y* ≤ *1* (*a*, *b*) = *diamond-cube-gen* (*x*, *y*)
  for *a b x y*
 proof (*intro exI conjI*)
  let *?a* = *rot-y x y*
  let *?b* = *rot-x x y*
  show *0* ≤ *?a ?a* ≤ *1*
   using *rot-y-ivl that* **by** *blast+*
  show *0* ≤ *?b ?b* ≤ *1*
   using *rot-x-ivl that* **by** *blast+*
  show *diamond-cube-gen* (*x*, *y*) = *rot-diamond-cube* (*?a*, *?b*)
   using *that d-gt-0 diamond-gen-eq-rot-diamond* **by** *auto*
 qed
 show ∃ *a*≥*0*. *a* ≤ *1* ∧ (∃ *b*≥*0*. *b* ≤ *1* ∧ *rot-diamond-cube* (*x*, *y*) = *diamond-cube-gen* (*a*, *b*))
  if *0* ≤ *x x* ≤ *1 0* ≤ *y y* ≤ *1* (*a*, *b*) = *rot-diamond-cube* (*x*, *y*) **for** *a b x y*
 proof (*intro exI conjI*)
  let *?a* = *rot-x y x*
  let *?b* = *rot-y y x*
  show *0* ≤ *?a ?a* ≤ *1*
   using *rot-x-ivl that* **by** *blast+*
  show *0* ≤ *?b ?b* ≤ *1*
   using *rot-y-ivl that* **by** *blast+*
  show *rot-diamond-cube* (*x*, *y*) = *diamond-cube-gen* (*?a*, *?b*)
   using *that d-gt-0 rot-diamond-eq-diamond-gen* **by** *auto*
 qed
qed

lemma *rot-diamond-gen-div-diamond-gen*:
 shows *gen-division* (*cubeImage* (*diamond-cube-gen*)) (*cubeImage* ‘ {*rot-diamond-cube*})
 using *rot-img-eq* **by**(*auto simp add*: *gen-division-def*)

lemma *rot-diamond-gen-div-diamond*:
 shows *gen-division* (*cubeImage* (*diamond-cube*)) (*cubeImage* ‘ {*rot-diamond-cube*})
 using *rot-diamond-gen-div-diamond-gen*
 by(*simp only*: *diamond-eq-characterisations-fun*)

lemma *GreenThm-diamond*:
 assumes *analytically-valid* (*cubeImage* (*diamond-cube*)) (λ*x*. *F x* · *i*) *j*
  analytically-valid (*cubeImage* (*diamond-cube*)) (λ*x*. *F x* · *j*) *i*
 shows *integral* (*cubeImage* (*diamond-cube*)) (λ*x*. *partial-vector-derivative* (λ*x*. *F*

$x \cdot j$) i x − *partial-vector-derivative* ($\lambda x.$ F $x \cdot i$) j x) =
one-chain-line-integral F $\{i, j\}$ (*boundary* (*diamond-cube*))
proof −
have *0*: ∀ (k, γ)∈*boundary* (*diamond-cube*). *valid-path* γ
using *diamond-cube-boundary-valid d-gt-0* **by** *auto*
have ⋀*twoCube*. *twoCube* ∈ {*diamond-cube*} ⟹ *typeI-twoCube twoCube*
using *assms diamond-cube-is-type-I* **by** *auto*
moreover have *valid-two-chain* {*diamond-cube*}
using *assms*(*1*) *diamond-cube-valid-two-cube valid-two-chain-def* **by** *auto*
moreover have *gen-division* (*cubeImage* (*diamond-cube*)) (*cubeImage* ‘ {*diamond-cube*})
by (*simp add*: *gen-division-def*)
moreover have (∀ *twoCube*∈({*rot-diamond-cube*}). *typeII-twoCube twoCube*)
using *assms rot-diamond-cube-is-type-II* **by** *auto*
moreover have *valid-two-chain* {*rot-diamond-cube*}
using *assms*(*1*) *rot-diamond-cube-valid-two-cube valid-two-chain-def* **by** *auto*
moreover have *gen-division* (*cubeImage* (*diamond-cube*)) (*cubeImage* ‘ {*rot-diamond-cube*})
using *rot-diamond-gen-div-diamond* **by** *auto*
moreover have *3*: *only-vertical-division* (*boundary* (*diamond-cube*)) {*diamond-cube*}
using *twoChainVertDiv-of-itself*[*of* {*diamond-cube*}] *diamond-cube-boundary-valid assms*
by(*auto simp add*: *two-chain-boundary-def*)
moreover have *4*: ∀ *twoC*∈{*diamond-cube*}. *analytically-valid* (*cubeImage twoC*) ($\lambda x.$ F $x \cdot i$) j
using *assms*
by *fastforce*
moreover have *5*: ∀ *twoC*∈{*rot-diamond-cube*}. *analytically-valid* (*cubeImage twoC*) ($\lambda x.$ F $x \cdot j$) i
using *assms diamond-eq-characterisations-fun rot-img-eq* **by** *auto*
ultimately show *?thesis*
using *green-typeI-typeII-chain.GreenThm-typeI-typeII-divisible-region-finite-holes*[*of* (*cubeImage* (*diamond-cube*)) *i j F* {*diamond-cube*} {*rot-diamond-cube*}, *OF - 0 3 diamond-cube-is-only-horizontal-div-of-rot* -]
R2-axioms
by(*auto simp add*: *green-typeI-typeII-chain-def green-typeI-chain-def green-typeII-chain-def green-typeI-chain-axioms-def green-typeII-chain-axioms-def*)
qed
end
end