

# The Detour Calculus

Manuel Eberl

June 25, 2026

## Abstract

This entry provides the *detour calculus*, a relational framework to construct deformed versions of integration contours by adding small indentations around problematic points (typically poles). It consists of:

- the definition of a relation that relates the original contour  $\gamma$  to a family of deformed contours  $(\gamma_\epsilon)_{\epsilon>0}$
- theorems that allow transferring geometric properties from  $\gamma$  to  $\gamma_\epsilon$  for sufficiently small  $\epsilon$
- a library of “basic avoidance patterns” to add indentations to lines, circular arcs, or the corners between them, and prove that they are in relation with the original contour
- a calculus of inference rules which, in particular, allows dealing with composite contours by proving the relation for its individual parts separately

The motivating example for this calculus was the proof of the valence formula for modular forms (not included here), which requires such a complicated set of indentations that I deemed it beyond the reach of formalisation without a systematic approach such as the present one.

# Contents

<b>1</b>	<b>Auxiliary material</b>	<b>3</b>
<b>2</b>	<b>Neighbourhoods of a path</b>	<b>3</b>
2.1	Nearby paths . . . . .	3
2.2	Piecewise smooth paths in the neighbourhood . . . . .	5
2.3	Lipschitz-continuity and paths . . . . .	6
<b>3</b>	<b>Prerequisites for the Detour Calculus</b>	<b>7</b>
3.1	Miscellaneous . . . . .	7
3.2	Lipschitz continuity . . . . .	9
3.3	Homotopy . . . . .	11
3.4	Winding numbers . . . . .	11
3.5	Continuous transformations that preserve the winding number in some way . . . . .	12
3.6	Paths . . . . .	13
3.7	Topology . . . . .	15
<b>4</b>	<b>The Detour Calculus</b>	<b>16</b>
4.1	Local deformations of a path . . . . .	17
4.2	The left/right detour relation . . . . .	20
4.3	Inference rules . . . . .	24
4.4	Basic avoidance patterns . . . . .	26
4.4.1	Generic helper lemmas . . . . .	26
4.4.2	Straight line . . . . .	29
4.4.3	Circular arc . . . . .	30
4.4.4	Line–line corner . . . . .	34
4.4.5	Line–arc corner . . . . .	35
4.5	Consequences of the detour relation . . . . .	36

# 1 Auxiliary material

## 2 Neighbourhoods of a path

```
theory Path_Nhds
  imports "HOL-Complex_Analysis.Complex_Analysis"
begin

lemma le_filterD_frequently [trans]: "F ≤ G ⇒ frequently P F ⇒
frequently P G"
  ⟨proof⟩

lemma le_filterD_frequently' [trans]: "frequently P F ⇒ F ≤ G ⇒
frequently P G"
  ⟨proof⟩

lemma frequently_filtermap: "frequently P (filtermap f F) ↔ frequently
(λx. P (f x)) F"
  ⟨proof⟩

lemma eventually_uniformly_on_iff:
  "eventually P (uniformly_on A f) ↔ (∃ e>0. ∀ g. (∀ y∈A. dist (g y)
(f y) < e) → P g)"
  (is "?lhs = ?rhs")
  ⟨proof⟩

lemma eventually_uniformly_onI [intro?]:
  "e > 0 ⇒ (∧ g. (∧ y. y ∈ A ⇒ dist (g y) (f y) < e) ⇒ P g) ⇒
eventually P (uniformly_on A f)"
  ⟨proof⟩

abbreviation same_ends :: "(real ⇒ 'a :: topological_space) ⇒ (real
⇒ 'a) ⇒ bool"
  where "same_ends p q ≡ pathstart p = pathstart q ∧ pathfinish p =
pathfinish q"
```

### 2.1 Nearby paths

```
definition path_nhds :: "(real ⇒ 'a :: real_normed_vector) ⇒ (real ⇒
'a) filter" where
  "path_nhds γ = inf (uniformly_on {0..1} γ) (principal {p. path p ∧
same_ends p γ})"
```

```
lemma eventually_path_nhds_iff:
  "eventually P (path_nhds γ) ↔
(∃ e>0. ∀ p. path p → same_ends p γ → (∀ y∈{0..1}. dist (p y)
(γ y) < e) → P p)"
  ⟨proof⟩
```

```

lemma frequently_path_nhds_iff:
  "frequently P (path_nhds  $\gamma$ )  $\longleftrightarrow$ 
    ( $\forall e > 0. \exists p. \text{path } p \wedge \text{same\_ends } p \ \gamma \wedge (\forall y \in \{0..1\}. \text{dist } (p \ y) (\gamma \ y) < e) \wedge P \ p$ )"
  <proof>

lemma eventually_path_nhdsI [intro?]:
  " $e > 0 \implies (\bigwedge p. \text{path } p \implies \text{same\_ends } p \ \gamma \implies (\bigwedge y. y \in \{0..1\} \implies \text{dist } (p \ y) (\gamma \ y) < e) \implies P \ p) \implies \text{eventually } P \ (\text{path\_nhds } \gamma)$ "
  <proof>

lemma eventually_path_path_nhds: "eventually ( $\lambda p. \text{path } p$ ) (path_nhds  $\gamma$ )"
  <proof>

lemma path_nhds_neq_bot [simp]: "path  $\gamma \implies \text{path\_nhds } \gamma \neq \text{bot}$ "
  <proof>

lemma eventually_dist_less_path_nhds:
  assumes " $e > 0$ "
  shows "eventually ( $\lambda p. \forall t \in \{0..1\}. \text{dist } (p \ t) (\gamma \ t) < e$ ) (path_nhds  $\gamma$ )"
  <proof>

lemma eventually_winding_number_eq_path_nhds:
  assumes "path  $\gamma$ " "z  $\notin$  path_image  $\gamma$ "
  shows "eventually ( $\lambda p. \text{winding\_number } p \ z = \text{winding\_number } \gamma \ z$ ) (path_nhds  $\gamma$ )"
  <proof>

lemma eventually_path_image_subset_path_nhds:
  assumes "path  $\gamma$ " "open A" "path_image  $\gamma \subseteq A$ "
  shows "eventually ( $\lambda p. \text{path\_image } p \subseteq A$ ) (path_nhds  $\gamma$ )"
  <proof>

lemma eventually_path_nhds_avoid:
  assumes "path  $\gamma$ " "closed A" " $A \cap \text{path\_image } \gamma = \{\}$ "
  shows "eventually ( $\lambda p. \text{path\_image } p \cap A = \{\}$ ) (path_nhds  $\gamma$ )"
  <proof>

```

If we have a path  $p$  and transform it with a function that is continuous in some open neighbourhood of  $p$ , then all the paths that are close to  $p$  are also transformed to paths close to the image of  $p$ .

```

lemma continuous_path_image:
  fixes p :: "real  $\Rightarrow$  'a :: euclidean_space"
  assumes "path p" "continuous_on A f" "open A" "path_image p  $\subseteq$  A"
  shows "filterlim ( $\lambda p. f \circ p$ ) (path_nhds (f  $\circ$  p)) (path_nhds p)"

```

*<proof>*

## 2.2 Piecewise smooth paths in the neighbourhood

**definition** `valid_path_nhds` :: "(real  $\Rightarrow$  'a :: real\_normed\_vector)  $\Rightarrow$  (real  $\Rightarrow$  'a) filter" **where**  
"valid\_path\_nhds  $\gamma$  = inf (uniformly\_on {0..1}  $\gamma$ ) (principal {p. valid\_path p  $\wedge$  same\_ends p  $\gamma$ })"

**lemma** `eventually_valid_path_nhds_iff`:  
"eventually P (valid\_path\_nhds  $\gamma$ )  $\iff$   
( $\exists e > 0$ .  $\forall p$ . valid\_path p  $\longrightarrow$  same\_ends p  $\gamma \longrightarrow (\forall y \in \{0..1\}$ . dist (p y) ( $\gamma$  y) < e)  $\longrightarrow$  P p)"  
*<proof>*

**lemma** `frequently_valid_path_nhds_iff`:  
"frequently P (valid\_path\_nhds  $\gamma$ )  $\iff$   
( $\forall e > 0$ .  $\exists p$ . valid\_path p  $\wedge$  same\_ends p  $\gamma \wedge (\forall y \in \{0..1\}$ . dist (p y) ( $\gamma$  y) < e)  $\wedge$  P p)"  
*<proof>*

**lemma** `eventually_valid_path_nhdsI [intro?]`:  
" $e > 0 \implies (\bigwedge p$ . valid\_path p  $\implies$  same\_ends p  $\gamma \implies (\bigwedge y$ .  $y \in \{0..1\} \implies$  dist (p y) ( $\gamma$  y) < e)  $\implies$  P p)"  
 $\implies$  eventually P (valid\_path\_nhds  $\gamma$ )"  
*<proof>*

**lemma** `eventually_valid_path_valid_path_nhds`: "eventually ( $\lambda p$ . valid\_path p) (valid\_path\_nhds  $\gamma$ )"  
*<proof>*

**lemma** `path_nhds_le_valid_path_nhds`: "valid\_path\_nhds  $\gamma \leq$  path\_nhds  $\gamma$ "  
*<proof>*

**lemma** `valid_path_nhds_neq_bot [simp]`: "valid\_path  $\gamma \implies$  valid\_path\_nhds  $\gamma \neq$  bot"  
*<proof>*

**lemma** `valid_path_nhds_eq_bot' [simp]`:  
assumes "path ( $\gamma$  :: real  $\Rightarrow$  'a :: euclidean\_space)"  
shows "valid\_path\_nhds  $\gamma \neq$  bot"  
*<proof>*

**lemma** `eventually_dist_less_valid_path_nhds`:  
assumes " $e > 0$ "  
shows "eventually ( $\lambda p$ .  $\forall t \in \{0..1\}$ . dist (p t) ( $\gamma$  t) < e) (valid\_path\_nhds  $\gamma$ )"  
*<proof>*

```

lemma eventually_same_ends_path_nhds:
  "eventually ( $\lambda p$ . same_ends p  $\gamma$ ) (path_nhds  $\gamma$ )"
  and eventually_same_ends_valid_path_nhds:
    "eventually ( $\lambda p$ . same_ends p  $\gamma$ ) (valid_path_nhds  $\gamma$ )"
  <proof>

lemma eventually_valid_path_nhds_avoid:
  assumes "path  $\gamma$ " "closed A" "A  $\cap$  path_image  $\gamma$  = {}"
  shows "eventually ( $\lambda p$ . path_image p  $\cap$  A = {}) (valid_path_nhds  $\gamma$ )"
  <proof>

lemma winding_number_unique':
  assumes "frequently ( $\lambda p$ . winding_number p z = n) (valid_path_nhds  $\gamma$ )"
  assumes "path  $\gamma$ " "z  $\notin$  path_image  $\gamma$ "
  shows "winding_number  $\gamma$  z = n"
  <proof>

lemma eventually_path_image_subset_valid_path_nhds:
  assumes "path  $\gamma$ " "open A" "path_image  $\gamma \subseteq A$ "
  shows "eventually ( $\lambda p$ . path_image p  $\subseteq A$ ) (valid_path_nhds  $\gamma$ )"
  <proof>

```

A set is defined to be path-connected if any two points in it are connected by a continuous path. The following shows that for open sets, one can also take the paths to be piecewise C1.

```

lemma path_connected_open_has_valid_path:
  fixes A :: "'a :: euclidean_space set"
  assumes "path_connected A" "open A" "x  $\in$  A" "y  $\in$  A"
  obtains p where "valid_path p" "path_image p  $\subseteq$  A" "pathstart p = x"
  "pathfinish p = y"
  <proof>

```

A path  $p$  always has arbitrarily close smooth paths in its vicinity. (i.e. it can be approximated by smooth paths to arbitrary precision)

```

lemma frequently_valid_path:
  assumes "path (p :: real  $\Rightarrow$  'a :: euclidean_space)"
  shows "frequently ( $\lambda p'$ . valid_path p') (path_nhds p)"
  <proof>

```

## 2.3 Lipschitz-continuity and paths

```

lemma path_nhds_compose:
  assumes "uniformly_continuous_on A f" "path  $\gamma$ " "path_image  $\gamma \subseteq A$ "
  "open A"
  shows "filterlim (( $\circ$ ) f) (path_nhds (f  $\circ$   $\gamma$ )) (path_nhds  $\gamma$ )"
  <proof>

```

```

lemma valid_path_nhds_compose:

```

```

  assumes "f analytic_on A" "uniformly_continuous_on A f" "path  $\gamma$ " "path_image
 $\gamma \subseteq A$ " "open A"
  shows "filterlim (( $\circ$ ) f) (valid_path_nhds (f  $\circ$   $\gamma$ )) (valid_path_nhds
 $\gamma$ )"
<proof>

```

```

lemma winding_number_comp':
  assumes f: "f holomorphic_on A" "uniformly_continuous_on A f" "inj_on
f A" "open A"
  assumes  $\gamma$ : "path  $\gamma$ " "path_image  $\gamma \subseteq A$ "
  assumes z: "z  $\in A$ " "z  $\notin$  path_image  $\gamma$ "
  assumes int: " $\exists_F p$  in valid_path_nhds  $\gamma$ .
contour_integral p ( $\lambda w$ . deriv f w / (f w - f z)) = 2
* pi * i * c"
  shows "winding_number (f  $\circ$   $\gamma$ ) (f z) = c"
<proof>

```

end

### 3 Prerequisites for the Detour Calculus

```

theory Detour_Prerequisites
  imports
    "HOL-Complex_Analysis.Residue_Theorem"
    "Winding_Number_Eval.Missing_Analysis"
    "Winding_Number_Eval.Missing_Transcendental"
    "Path_Automation.Path_Automation"
    Path_Nhds
    "HOL-Library.Real_Mod"
begin

```

#### 3.1 Miscellaneous

```

lemma inverse_conv_cnj: "norm z = 1  $\implies$  inverse z = cnj z"
<proof>

```

```

lemma dist_cnj: "dist (cnj x) (cnj y) = dist x y"
<proof>

```

```

lemma closed_segment_cnj: "closed_segment (cnj w) (cnj z) = cnj ` closed_segment
w z"
<proof>

```

```

lemma closed_segment_mult:
  "(*) (c :: 'a :: real_algebra_1) ` closed_segment a b = closed_segment
(c * a) (c * b)"
<proof>

```

```

lemma arcsin_pos: "x  $\in$  {0<..1}  $\implies$  arcsin x > 0"

```

$\langle proof \rangle$

**lemma** `sin_lt_zero'`: " $x \in \{-\pi < \dots < 0\} \implies \sin x < 0$ "  
 $\langle proof \rangle$

**lemma** `tan_arcsin`: " $x \in \{-1..1\} \implies \tan (\arcsin x) = x / \text{sqrt } (1 - x^2)$ "  
 $\langle proof \rangle$

**lemma** `Arg_neg_real [simp]`: " $\text{Im } x = 0 \implies \text{Re } x < 0 \implies \text{Arg } x = \pi$ "  
 $\langle proof \rangle$

**lemma** `cis_eq_1_iff'`:  
 assumes " $|x| < 2 * \pi$ "  
 shows " $\text{cis } x = 1 \iff x = 0$ "  
 $\langle proof \rangle$

**lemma** `DeMoivre_int`: " $\text{cis } x \text{ powi } n = \text{cis } (\text{of\_int } n * x)$ "  
 $\langle proof \rangle$

**lemma** `cis_of_int_times_pi_half`: " $\text{cis } (\text{of\_int } n * \pi / 2) = i \text{ powi } n$ "  
 $\langle proof \rangle$

**lemma** `cis_of_nat_times_pi_half`: " $\text{cis } (\text{real } n * \pi / 2) = i ^ n$ "  
 $\langle proof \rangle$

**lemma** `cis_of_int_times_pi`: " $\text{cis } (\text{of\_int } n * \pi) = (-1) \text{ powi } n$ "  
 $\langle proof \rangle$

**lemma** `cis_of_nat_times_pi`: " $\text{cis } (\text{real } n * \pi) = (-1) ^ n$ "  
 $\langle proof \rangle$

**lemma** `power3_i [simp]`: " $i ^ 3 = -i$ "  
**and** `power4_i [simp]`: " $i ^ 4 = 1$ "  
 $\langle proof \rangle$

**lemma** `i_power_mod4`: " $i ^ n = i ^ (n \bmod 4)$ "  
 $\langle proof \rangle$

**lemma** `cis_numeral_times_pi_half [simp]`:  
 " $\text{cis } (\text{numeral num } * \pi / 2) = i ^ (\text{numeral num } \bmod 4)$ "  
 $\langle proof \rangle$

**lemma** `cis_numeral_pi_times_numeral_half [simp]`:  
 " $\text{cis } (\pi * \text{numeral num } / 2) = i ^ (\text{numeral num } \bmod 4)$ "  
 $\langle proof \rangle$

**lemma** `cis_numeral_times_pi [simp]`:  
 " $\text{cis } (\text{numeral num } * \pi) = (-1) ^ (\text{numeral num } \bmod 2)$ "

*<proof>*

**lemma** *cis\_pi\_times\_numeral* [simp]:  
"cis (pi \* numeral num) = (-1) ^ (numeral num mod 2)"  
*<proof>*

**lemma** *cis\_minus\_numeral\_times\_pi\_half* [simp]:  
"cis (-(numeral num \* pi / 2)) = (-i) ^ (numeral num mod 4)"  
*<proof>*

**lemma** *cis\_minus\_numeral\_times\_pi* [simp]:  
"cis (-(numeral num \* pi)) = (-1) ^ (numeral num mod 2)"  
*<proof>*

**lemma** *cis\_minus\_pi\_times\_numeral\_half* [simp]:  
"cis (-(pi \* numeral num / 2)) = (-i) ^ (numeral num mod 4)"  
*<proof>*

**lemma** *cis\_minus\_pi\_times\_numeral\_pi* [simp]:  
"cis (-(pi \* numeral num)) = (-1) ^ (numeral num mod 2)"  
*<proof>*

**lemma** *cis\_minus\_pi* [simp]: "cis (-pi) = -1"  
*<proof>*

**lemma** *inner\_mult\_both\_complex*: "(z \* x :: complex) \* (z \* y) = norm z  
^ 2 \* (x \* y)"  
*<proof>*

**lemma** *orthogonal\_transformation\_mult\_complex* [intro]:  
"norm z = 1  $\implies$  orthogonal\_transformation ((\* (z :: complex))"  
*<proof>*

**lemma** *contour\_integral\_affine*:  
assumes "valid\_path  $\gamma$ " "c  $\neq$  0"  
shows "contour\_integral (( $\lambda x. c * x + b$ )  $\circ \gamma$ ) f = contour\_integral  
 $\gamma (\lambda w. c * f (c * w + b))"$   
*<proof>*

**lemma** *finite\_imp\_eventually\_sparse\_at\_0*:  
assumes "finite X"  
shows "eventually ( $\lambda \varepsilon. \text{sparse } \varepsilon X$ ) (at\_right 0)"  
*<proof>*

## 3.2 Lipschitz continuity

**lemma** *complex\_derivative\_on\_convex\_imp\_lipschitz*:  
fixes f' :: "complex  $\Rightarrow$  complex"

```

    assumes deriv: " $\bigwedge z. z \in A \implies (f \text{ has\_field\_derivative } f' z)$  (at z
within A)"
    assumes A: "convex A" and C: " $\bigwedge x. x \in A \implies \text{norm } (f' x) \leq C$ " "C  $\geq$ 
0"
    shows "C-lipschitz_on A f"
<proof>

```

```

lemma analytic_on_compact_convex_imp_lipschitz:
  assumes "f analytic_on A" "convex A" "compact A"
  obtains C where "C-lipschitz_on A f"
<proof>

```

```

lemma lipschitz_on_complex_inverse:
  assumes "C > 0"
  shows "(1/C^2)-lipschitz_on {z. Im z  $\geq$  C} ( $\lambda z. \text{inverse } z :: \text{complex}$ )"
<proof>

```

```

lemma lipschitz_on_cnj [lipschitz_intros]:
  fixes f::"'a::metric_space  $\Rightarrow$  complex"
  assumes "C-lipschitz_on U f"
  shows "C-lipschitz_on U ( $\lambda x. \text{cnj } (f x)$ )"
<proof>

```

```

lemma lipschitz_on_cmult_complex [lipschitz_intros]:
  fixes f::"'a::metric_space  $\Rightarrow$  complex"
  assumes "C-lipschitz_on U f"
  shows "(norm c * C)-lipschitz_on U ( $\lambda x. c * f x$ )"
<proof>

```

```

lemma lipschitz_on_cmult_complex' [lipschitz_intros]:
  fixes f::"'a::metric_space  $\Rightarrow$  complex"
  assumes "C-lipschitz_on U f" "C'  $\geq$  norm c * C"
  shows "C'-lipschitz_on U ( $\lambda x. c * f x$ )"
<proof>

```

```

lemma lipschitz_on_cadd_left [lipschitz_intros]:
  fixes f :: "_  $\Rightarrow$  'b :: real_normed_vector"
  assumes "C-lipschitz_on A f"
  shows "C-lipschitz_on A ( $\lambda x. c + f x$ )"
<proof>

```

```

lemma lipschitz_on_cadd_right [lipschitz_intros]:
  fixes f :: "_  $\Rightarrow$  'b :: real_normed_vector"
  assumes "C-lipschitz_on A f"
  shows "C-lipschitz_on A ( $\lambda x. f x + c$ )"
<proof>

```

### 3.3 Homotopy

```
lemma simply_connected_imp_homotopic_paths:
  fixes S :: "'a :: real_normed_vector set"
  assumes "simply_connected S" "path p" "path_image p  $\subseteq$  S" "path q" "path_image
q  $\subseteq$  S"
  assumes "pathstart q = pathstart p  $\wedge$  pathfinish q = pathfinish p"
  shows "homotopic_paths S p q"
  <proof>
```

```
lemma homotopic_loops_part_circlepath_circlepath:
  assumes "b = a + 2 * pi" "sphere x r  $\subseteq$  A" "r  $\geq$  0"
  shows "homotopic_loops A (part_circlepath x r a b) (circlepath x r)"
  <proof>
```

```
lemma homotopic_loops_reversepath_D:
  "homotopic_loops A p q  $\implies$  homotopic_loops A (reversepath p) (reversepath
q)"
  <proof>
```

```
lemma homotopic_loops_reversepath:
  "homotopic_loops A (reversepath p) (reversepath q)  $\longleftrightarrow$  homotopic_loops
A p q"
  <proof>
```

```
lemmas [trans] = homotopic_loops_trans
```

```
lemma homotopic_paths_split:
  assumes p: "path p" and A: "path_image p  $\subseteq$  A"
  assumes a: "a  $\in$  {0..1}"
  assumes eq1: " $\bigwedge x. x \in \{0..1\} \implies p1\ x = \text{subpath } 0\ a\ p\ x$ "
  assumes eq2: " $\bigwedge x. x \in \{0..1\} \implies p2\ x = \text{subpath } a\ 1\ p\ x$ "
  shows "homotopic_paths A p (p1 +++ p2)"
  <proof>
```

### 3.4 Winding numbers

```
lemma winding_number_comp_plus:
  assumes "path  $\gamma$ " "z  $\notin$  path_image  $\gamma$ "
  shows "winding_number ((+) c  $\circ$   $\gamma$ ) (z + c) = winding_number  $\gamma$  z"
  <proof>
```

```
lemma winding_number_comp_times:
  assumes "path  $\gamma$ "
  and "z  $\notin$  path_image  $\gamma$ "
  and "c  $\neq$  0"
  shows "winding_number ((* ) c  $\circ$   $\gamma$ ) (z * c) = winding_number  $\gamma$  z"
  <proof>
```

```

lemma winding_number_part_circlepath_full:
  assumes "y ∈ ball x r" "α + 2 * pi = β"
  shows "winding_number (part_circlepath x r α β) y = 1"
⟨proof⟩

lemma winding_number_part_circlepath_full':
  assumes "y ∈ ball x r" "α - 2 * pi = β"
  shows "winding_number (part_circlepath x r α β) y = -1"
⟨proof⟩

lemma winding_number_inverse_valid_path:
  assumes "valid_path γ" "0 ∉ path_image γ" "z ∉ path_image γ" "z ≠ 0"
  shows "winding_number (inverse ∘ γ) (inverse z) = winding_number γ z - winding_number γ 0"
⟨proof⟩

lemma winding_number_inverse:
  assumes "path γ" "path_image γ ⊆ {z. Im z > 0}" "z ∉ path_image γ"
  "Im z > 0"
  shows "winding_number (inverse ∘ γ) (inverse z) = winding_number γ z - winding_number γ 0"
⟨proof⟩

lemma winding_number_inverse_valid_path_0:
  assumes "valid_path γ" "pathstart γ = pathfinish γ" "0 ∉ path_image γ"
  shows "winding_number (inverse ∘ γ) 0 = -winding_number γ 0"
⟨proof⟩

```

The following allows us to compute the winding number of a non-closed circular arc with respect to its centre.

```

lemma winding_number_part_circlepath_centre:
  assumes "r > 0"
  shows "winding_number (part_circlepath z r a b) z = (b - a) / (2 * pi)"
⟨proof⟩

```

### 3.5 Continuous transformations that preserve the winding number in some way

```

locale winding_preserving =
  fixes A :: "complex set" and f :: "complex ⇒ complex" and g :: "complex ⇒ complex"
  assumes inj: "inj_on f A"
  assumes cont: "continuous_on A f"
  assumes winding_number_eq:
    "∧p x. path p ⇒ path_image p ⊆ A ⇒ pathstart p = pathfinish

```

$p \implies x \in A - \text{path\_image } p \implies$   
 $\text{winding\_number } (f \circ p) (f x) = g (\text{winding\_number } p x)$ "

**lemma** *winding\_preserving\_comp*:  
 assumes "winding\_preserving B f2 g2"  
 assumes "winding\_preserving A f1 g1"  
 assumes subset: "f1 ' A  $\subseteq$  B"  
 shows "winding\_preserving A (f2  $\circ$  f1) (g2  $\circ$  g1)"  
 <proof>

lemmas *winding\_preserving\_comp'* = *winding\_preserving\_comp* [unfolded *o\_def*]

**lemma** *winding\_preserving\_subset*:  
 assumes "winding\_preserving A f g" "B  $\subseteq$  A"  
 shows "winding\_preserving B f g"  
 <proof>

**lemma** *winding\_preserving\_translate*: "winding\_preserving A ( $\lambda x. c + x$ ) ( $\lambda x. x$ )"  
 <proof>

**lemma** *winding\_preserving\_mult*: " $c \neq 0 \implies$  winding\_preserving A ( $\lambda x. c * x$ ) ( $\lambda x. x$ )"  
 <proof>

**lemma** *winding\_preserving\_cnj*: "winding\_preserving A *cnj* ( $\lambda x. -cnj x$ )"  
 <proof>

**lemma** *winding\_preserving\_uminus*: "winding\_preserving A ( $\lambda x. -x$ ) ( $\lambda x. x$ )"  
 <proof>

### 3.6 Paths

**lemma** *simple\_path\_cnj* [*simp*]: "simple\_path (*cnj*  $\circ$  *p*)  $\longleftrightarrow$  simple\_path *p*"  
 <proof>

**lemma** *part\_circlepath\_cnj'*: "*cnj*  $\circ$  part\_circlepath *c r a b* = part\_circlepath (*cnj c*) *r* (-*a*) (-*b*)"  
 <proof>

**lemma** *linepath\_minus*: "linepath (-*a*) (-*b*) *x* = -linepath *a b x*"  
 <proof>

The following lemma is very difficult to bypass some nasty geometric reasoning: If a path only touches the frontier of a set at its beginning or end, then it is either fully inside the set or fully outside the set.

**lemma** *path\_fully\_inside\_or\_fully\_outside*:

```

fixes p :: "real  $\Rightarrow$  'a :: euclidean_space"
assumes "path p" " $\wedge x. x \in \{0 < \dots < 1\} \implies p\ x \notin \text{frontier } A$ "
shows "path_image p  $\subseteq$  closure A  $\vee$  path_image p  $\cap$  interior A = {}"
<proof>

lemma simple_path_joinE':
  assumes "simple_path (g1 +++ g2)" and "pathfinish g1 = pathstart g2"
  shows "path_image g1  $\cap$  path_image g2  $\subseteq$ 
        (insert (pathstart g2) (if pathstart g1 = pathfinish g2 then
{pathstart g1} else {}))"
  <proof>

lemma simple_path_joinE'':
  assumes "simple_path (g1 +++ g2)" and "pathfinish g1 = pathstart g2"
        "x  $\in$  path_image g1" "x  $\in$  path_image g2"
  shows "x = pathstart g2  $\vee$  x = pathfinish g2  $\wedge$  pathstart g1 = pathfinish
g2"
  <proof>

lemma arc_continuous_image:
  assumes "arc p" "inj_on f (path_image p)" "continuous_on (path_image
p) f"
  shows "arc (f  $\circ$  p)"
  <proof>

lemma eventually_path_image_cball_subset:
  fixes p :: "real  $\Rightarrow$  'a :: real_normed_vector"
  assumes "path p" "path_image p  $\subseteq$  interior A"
  shows "eventually ( $\lambda \varepsilon. (\bigcup_{x \in \text{path\_image } p}. \text{cball } x \ \varepsilon) \subseteq A$ ) (at_right
0)"
  <proof>

lemma eventually_path_image_cball_subset':
  fixes p :: "real  $\Rightarrow$  'a :: real_normed_vector"
  assumes "path p" "path_image p  $\subseteq$  interior A" "X  $\subseteq$  path_image p"
  shows "eventually ( $\lambda \varepsilon. \text{path\_image } p \cup (\bigcup_{x \in X}. \text{cball } x \ \varepsilon) \subseteq A$ ) (at_right
0)"
  <proof>

We say that a path does not cross a set A if it enters A at most at its
beginning and end, and never inbetween.

definition does_not_cross :: "(real  $\Rightarrow$  'a :: real_vector)  $\Rightarrow$  'a set  $\Rightarrow$ 
bool" where
  "does_not_cross p A  $\iff (\forall x \in \{0 < \dots < 1\}. p\ x \notin A)$ "

lemma does_not_cross_simple_path:
  assumes "simple_path p"
  shows "does_not_cross p A  $\iff$  path_image p  $\cap$  A  $\subseteq$  {pathstart p, pathfinish
p}"

```

*<proof>*

**lemma path\_fully\_inside\_or\_fully\_outside':**  
fixes p :: "real  $\Rightarrow$  'a :: euclidean\_space"  
assumes "path p" "does\_not\_cross p (frontier A)"  
shows "path\_image p  $\subseteq$  closure A  $\vee$  path\_image p  $\cap$  interior A = {}"  
*<proof>*

**lemma in\_path\_image\_part\_circlepathI:**  
assumes "y = x + rcis r u" "u  $\in$  closed\_segment a b"  
shows "y  $\in$  path\_image (part\_circlepath x r a b)"  
*<proof>*

**lemma in\_path\_image\_part\_circlepathI':**  
assumes "Arg (y - x)  $\in$  closed\_segment a b" "dist y x = r"  
shows "y  $\in$  path\_image (part\_circlepath x r a b)"  
*<proof>*

**lemma path\_image\_part\_circlepath':**  
"path\_image (part\_circlepath x r a b) = ( $\lambda t. x + rcis r t$ ) ' closed\_segment a b"  
*<proof>*

**lemma path\_image\_part\_circlepath:**  
assumes "a  $\in$  {-pi<..pi}" "b  $\in$  {-pi<..pi}" "r > 0"  
shows "path\_image (part\_circlepath x r a b) = {y  $\in$  sphere x r. Arg (y - x)  $\in$  closed\_segment a b}"  
*<proof>*

**lemma path\_image\_part\_circlepath\_mono:**  
assumes "min a' b'  $\leq$  min a b" "max a' b'  $\geq$  max a b"  
shows "path\_image (part\_circlepath x r a b)  $\subseteq$  path\_image (part\_circlepath x r a' b')"  
*<proof>*

### 3.7 Topology

**lemma eventually\_at\_within\_in\_open:**  
assumes "open X" "x  $\in$  X"  
shows "eventually ( $\lambda z. z \in X \cap A - \{x\}$ ) (at x within A)"  
*<proof>*

**lemma filterlim\_at\_withinD:**  
assumes "filterlim f (at L within A) F" "open X" "L  $\in$  X"  
shows "eventually ( $\lambda x. f x \in X \cap A - \{L\}$ ) F"  
*<proof>*

**lemma filterlim\_at\_withinD':**  
assumes "filterlim f (at L within A) F" "open X" "L  $\in$  X"

```

shows "eventually ( $\lambda x. f x \in X \cap A$ ) F"
⟨proof⟩

lemma filterlim_at_rightD:
  assumes "filterlim f (at_right L) F" "a > L"
  shows "eventually ( $\lambda x. f x \in \{L <..<a\}$ ) F"
  ⟨proof⟩

lemma filterlim_at_leftD:
  assumes "filterlim f (at_left L) F" "a < L"
  shows "eventually ( $\lambda x. f x \in \{a <..<L\}$ ) F"
  ⟨proof⟩

lemma eventually_Ball_at_right_0_real:
  assumes "eventually P (at_right (0 :: real))"
  shows "eventually ( $\lambda x. \forall y \in \{0 <..x\}. P y$ ) (at_right 0)"
  ⟨proof⟩

lemma open_segment_same_Re:
  assumes "Re a = Re b"
  shows "open_segment a b = {z. Re z = Re a  $\wedge$  Im z  $\in$  open_segment (Im a) (Im b)}"
  ⟨proof⟩

lemma open_segment_same_Im:
  assumes "Im a = Im b"
  shows "open_segment a b = {z. Im z = Im a  $\wedge$  Re z  $\in$  open_segment (Re a) (Re b)}"
  ⟨proof⟩

lemma interior_halfspace_Re_ge [simp]: "interior {z. Re z  $\geq$  x} = {z. Re z > x}"
and interior_halfspace_Re_le [simp]: "interior {z. Re z  $\leq$  x} = {z. Re z < x}"
and interior_halfspace_Im_ge [simp]: "interior {z. Im z  $\geq$  x} = {z. Im z > x}"
and interior_halfspace_Im_le [simp]: "interior {z. Im z  $\leq$  x} = {z. Im z < x}"
  ⟨proof⟩

thm arc_def
thm simple_path_def

end

```

## 4 The Detour Calculus

```

theory Detour_Calculus
  imports "HOL-Complex_Analysis.Complex_Analysis" "Path_Automation.Path_Automation"

```

```

Detour_Prerequisites
begin

```

```

lemma shiftpath_reversepath_loop:
  assumes "x ∈ {0..1}" "pathstart p = pathfinish p"
  shows "shiftpath c (reversepath p) x = reversepath (shiftpath (1-c)
p) x"
  ⟨proof⟩

```

```

lemma eqloops_reversepath_cong:
  assumes "p ≡○ q"
  shows "reversepath p ≡○ reversepath q"
  ⟨proof⟩

```

#### 4.1 Local deformations of a path

```

locale detour_rel_aux_locale =
  fixes ε :: real and X :: "complex set" and p :: "real ⇒ complex" and
p' :: "real ⇒ complex"
  assumes ε_pos: "ε > 0"
  assumes p'_simple [simp, intro]: "simple_path p'"
  assumes p'_valid [simp, intro]: "valid_path p'"
  assumes X_subset: "X ⊆ path_image p"
  assumes X_disjoint: "X ∩ path_image p' = {}"
  assumes homotopic: "homotopic_paths (path_image p ∪ (⋃x∈X. cball
x ε)) p p'"
begin

```

```

lemma X_disjoint' [simp]: "x ∈ X ⇒ x ∉ path_image p'"
and X_subset' [simp]: "x ∈ X ⇒ x ∈ path_image p"
  ⟨proof⟩

```

```

lemma p'_path [simp, intro]: "path p'"
  ⟨proof⟩

```

```

lemma path_image_p': "path_image p' ⊆ path_image p ∪ (⋃x∈X. cball
x ε)"
  ⟨proof⟩

```

```

lemma same_ends [simp]: "pathstart p' = pathstart p" "pathfinish p' =
pathfinish p"
  ⟨proof⟩

```

```

lemma ends_not_in_X: "pathstart p ∉ X" "pathfinish p ∉ X"
  ⟨proof⟩

```

```

lemma translate:

```

```

"detour_rel_aux_locale  $\varepsilon$  ((+) a ' X) ((+) a  $\circ$  p) ((+) a  $\circ$  p')"
<proof>

lemma orthogonal_transformation:
  assumes f [intro]: "orthogonal_transformation f"
  shows "detour_rel_aux_locale  $\varepsilon$  (f ' X) (f  $\circ$  p) (f  $\circ$  p')"
<proof>

lemma rotate:
  assumes "norm z = 1"
  shows "detour_rel_aux_locale  $\varepsilon$  ((* z ' X) ((* z  $\circ$  p) ((* z  $\circ$  p'))"
<proof>

lemma analytic_image:
  assumes inj: "inj_on f (path_image p  $\cup$  ( $\bigcup_{x \in X}$ . cball x  $\varepsilon$ ))"
  assumes ana: "f analytic_on (path_image p  $\cup$  ( $\bigcup_{x \in X}$ . cball x  $\varepsilon$ ))"
  assumes cball_image: " $\bigwedge x. x \in X \implies f$  ' cball x  $\varepsilon \subseteq$  cball (f x)  $\varepsilon$ '"
  assumes " $\varepsilon' > 0$ "
  shows "detour_rel_aux_locale  $\varepsilon'$  (f ' X) (f  $\circ$  p) (f  $\circ$  p')"
<proof>

lemma reverse [intro!]:
  "detour_rel_aux_locale  $\varepsilon$  X (reversepath p) (reversepath p')"
<proof>

theorem winding_number_unchanged:
  assumes "z  $\notin$  path_image p  $\cup$  ( $\bigcup_{x \in X}$ . cball x  $\varepsilon$ )"
  shows "winding_number p' z = winding_number p z"
<proof>

lemma congI:
  assumes "p  $\equiv_p$  p2" "p'  $\equiv_p$  p'2" "valid_path p'2"
  shows "detour_rel_aux_locale  $\varepsilon$  X p2 p'2"
<proof>

lemma mono:
  assumes " $\varepsilon \leq \varepsilon'$ "
  shows "detour_rel_aux_locale  $\varepsilon'$  X p p'"
<proof>

lemma cnj: "detour_rel_aux_locale  $\varepsilon$  (cnj ' X) (cnj  $\circ$  p) (cnj  $\circ$  p')"
<proof>

end

lemma detour_rel_aux_locale_refl [intro!]:
  " $\varepsilon > 0 \implies$  simple_path p  $\implies$  valid_path p  $\implies$  detour_rel_aux_locale
 $\varepsilon$  { } p p"
<proof>

```

```

lemma eq_paths_imp_detour_rel_aux_locale:
  assumes "ε > 0" "simple_path p" "valid_path q" "eq_paths p q"
  shows "detour_rel_aux_locale ε {} p q"
  ⟨proof⟩

lemma detour_rel_aux_join_aux1:
  assumes setdist: "setdist_gt ε X1 (path_image p2)"
  shows "(⋃x∈X1. cball x ε) ∩ path_image p2 = {}"
  ⟨proof⟩

lemma detour_rel_aux_join_aux2:
  assumes "detour_rel_aux_locale ε X2 p2 p2'"
  assumes setdist: "setdist_gt ε X1 (path_image p2)"
  shows "X1 ∩ path_image p2' = {}"
  ⟨proof⟩

locale detour_rel_aux_locale_join =
  p1: detour_rel_aux_locale ε X1 p1 p1' +
  p2: detour_rel_aux_locale ε X2 p2 p2' for ε X1 p1 p1' X2 p2 p2' +
  assumes p12_simple: "simple_path (p1 +++ p2)"
  assumes pathfinish_p1 [simp]: "pathfinish p1 = pathstart p2"
  assumes setdist: "setdist_gt (2*ε) X1 (path_image p2)" "setdist_gt
(2*ε) X2 (path_image p1)"
begin

lemma cball_X1_inter_p2: "(⋃x∈X1. cball x ε) ∩ path_image p2 = {}"
  and cball_X2_inter_p1: "(⋃x∈X2. cball x ε) ∩ path_image p1 = {}"
  ⟨proof⟩

lemma X1_inter_p2: "X1 ∩ path_image p2' = {}"
  and X2_inter_p1: "X2 ∩ path_image p1' = {}"
  ⟨proof⟩

lemma X1_inter_p2' [simp]: "x ∈ X1 ⇒ x ∉ path_image p2'"
  and X2_inter_p1' [simp]: "x ∈ X2 ⇒ x ∉ path_image p1'"
  ⟨proof⟩

lemma X1_X2_disjoint: "X1 ∩ X2 = {}"
  ⟨proof⟩

lemma X1_X2_disjoint' [simp]: "x ∈ X1 ⇒ x ∉ X2"
  ⟨proof⟩

lemma cball_X1_inter_cball_X2:
  assumes "x ∈ X1" "y ∈ X2"
  shows "cball x ε ∩ cball y ε = {}"
  ⟨proof⟩

```

```

lemma cball_X1_inter_cball_X2':
  shows "( $\bigcup_{y \in X1}. \text{cball } y \ \varepsilon) \cap (\bigcup_{y \in X2}. \text{cball } y \ \varepsilon) = \{\}$ "
  <proof>

sublocale p12: detour_rel_aux_locale  $\varepsilon$  "X1  $\cup$  X2" "p1 +++ p2" "p1' +++
p2'"
<proof>

end

locale detour_rel_aux_loop = detour_rel_aux_locale +
  assumes simple_loop [simp, intro]: "simple_loop p"
begin

lemma loop [simp]: "pathfinish p = pathstart p"
  and p_simple [simp, intro]: "simple_path p"
  and p_path [simp, intro]: "path p"
  and p'_simple_loop [simp, intro]: "simple_loop p'"
  <proof>

theorem same_orientation:
  assumes "winding_number p z  $\neq$  0" "z  $\notin$  path_image p  $\cup$  ( $\bigcup_{x \in X}. \text{cball }
x \ \varepsilon)$ "
  shows "simple_loop_orientation p' = simple_loop_orientation p"
  <proof>

end

```

## 4.2 The left/right detour relation

```

locale detour_rel_locale =
  pl: detour_rel_aux_locale  $\varepsilon$  "L  $\cup$  R" p pl +
  pr: detour_rel_aux_locale  $\varepsilon$  "L  $\cup$  R" p pr
  for  $\varepsilon$  L R p pl pr +
  assumes L_closed [intro]: "closed L" and R_closed [intro]: "closed
R"
  and winding_number_L: " $x \in L \implies \text{winding\_number } pl \ x - \text{winding\_number }
pr \ x = -1$ "
  and winding_number_R: " $x \in R \implies \text{winding\_number } pl \ x - \text{winding\_number }
pr \ x = 1$ "
begin

lemma L_R_disjoint: "L  $\cap$  R =  $\{\}$ "
  <proof>

lemma ends_not_in_L: "pathstart p  $\notin$  L" "pathfinish p  $\notin$  L"

```

```

and ends_not_in_R: "pathstart p  $\notin$  R" "pathfinish p  $\notin$  R"
  <proof>

lemma  $\varepsilon$ _pos: " $\varepsilon > 0$ "
  <proof>

lemma swap: "detour_rel_locale  $\varepsilon$  R L p pr pl"
  <proof>

lemma reverse [intro!]:
  "detour_rel_locale  $\varepsilon$  R L (reversepath p) (reversepath pl) (reversepath
pr)"
  <proof>

lemma winding_preserving:
  assumes ana: "f analytic_on (path_image p  $\cup$  ( $\bigcup_{x \in L \cup R}$ . cball x  $\varepsilon$ ))"
  assumes "winding_preserving (path_image p  $\cup$  ( $\bigcup_{x \in L \cup R}$ . cball x  $\varepsilon$ ))"
  f ( $\lambda x$ . x)
  assumes cball_image: " $\bigwedge x$ .  $x \in L \cup R \implies f \text{ ` cball } x \ \varepsilon \subseteq \text{ cball } (f$ 
x)  $\varepsilon'$ "
  assumes "path p"
  assumes " $\varepsilon' > 0$ "
  shows "detour_rel_locale  $\varepsilon'$  (f ` L) (f ` R) (f  $\circ$  p) (f  $\circ$  pl) (f  $\circ$  pr)"
  <proof>

lemma winding_preserving_flip:
  assumes ana: "f analytic_on (path_image p  $\cup$  ( $\bigcup_{x \in L \cup R}$ . cball x  $\varepsilon$ ))"
  assumes "winding_preserving (path_image p  $\cup$  ( $\bigcup_{x \in L \cup R}$ . cball x  $\varepsilon$ ))"
  f ( $\lambda x$ . -cnj x)
  assumes cball_image: " $\bigwedge x$ .  $x \in L \cup R \implies f \text{ ` cball } x \ \varepsilon \subseteq \text{ cball } (f$ 
x)  $\varepsilon'$ "
  assumes "path p"
  assumes " $\varepsilon' > 0$ "
  shows "detour_rel_locale  $\varepsilon'$  (f ` R) (f ` L) (f  $\circ$  p) (f  $\circ$  pl) (f  $\circ$  pr)"
  <proof>

lemma congI:
  assumes "p  $\equiv_p$  p'" "pl  $\equiv_p$  pl'" "pr  $\equiv_p$  pr'" "valid_path pl'" "valid_path
pr'"
  shows "detour_rel_locale  $\varepsilon$  L R p' pl' pr'"
  <proof>

lemma mono:
  assumes " $\varepsilon \leq \varepsilon'$ "
  shows "detour_rel_locale  $\varepsilon'$  L R p pl pr"
  <proof>

lemma cnj: "detour_rel_locale  $\varepsilon$  (cnj ` R) (cnj ` L) (cnj  $\circ$  p) (cnj  $\circ$ 

```

```

p1) (cnj ∘ pr)"
⟨proof⟩

end

locale detour_rel_locale_join =
  p1: detour_rel_locale ε L1 R1 p1 p11 pr1 +
  p2: detour_rel_locale ε L2 R2 p2 p12 pr2 for ε L1 R1 p1 p11 pr1 L2 R2
p2 p12 pr2 +
  assumes p12_simple: "simple_path (p1 +++ p2)"
  assumes pathfinish_p1 [simp]: "pathfinish p1 = pathstart p2"
  assumes setdist: "setdist_gt (2*ε) (L1 ∪ R1) (path_image p2)"
    "setdist_gt (2*ε) (L2 ∪ R2) (path_image p1)"

begin

sublocale p1: detour_rel_aux_locale_join ε "L1 ∪ R1" p1 p11 "L2 ∪ R2"
p2 p12
⟨proof⟩

sublocale pr: detour_rel_aux_locale_join ε "L1 ∪ R1" p1 pr1 "L2 ∪ R2"
p2 pr2
⟨proof⟩

sublocale p12: detour_rel_locale ε "L1 ∪ L2" "R1 ∪ R2" "p1 +++ p2" "p11
+++ p12" "pr1 +++ pr2"
⟨proof⟩

end

locale detour_rel_loop = detour_rel_locale +
  assumes simple_loop [simp, intro]: "simple_loop p"
  assumes nontrivial: "∃z. winding_number p z ≠ 0 ∧ z ∉ path_image
p ∪ (⋃x∈L∪R. cball x ε)"
begin

sublocale p1: detour_rel_aux_loop ε "L ∪ R" p p1
⟨proof⟩

sublocale pr: detour_rel_aux_loop ε "L ∪ R" p pr
⟨proof⟩

lemma same_orientation:
  "simple_loop_orientation p1 = simple_loop_orientation p"
  "simple_loop_orientation pr = simple_loop_orientation p"
⟨proof⟩

lemma reverse_loop [intro!]:

```

"detour\_rel\_loop  $\varepsilon$  R L (reversepath p) (reversepath pl) (reversepath pr)"  
 <proof>

**theorem**

assumes x: "x  $\in$  L"  
 shows winding\_number\_L\_left: "winding\_number pl x = (if simple\_loop\_ccw p then -1 else 0)"  
 and winding\_number\_L\_right: "winding\_number pr x = (if simple\_loop\_ccw p then 1 else 0)"  
 and inside\_pl\_L\_iff: "x  $\in$  inside (path\_image pl)  $\longleftrightarrow$  simple\_loop\_ccw p"  
 and inside\_pr\_L\_iff: "x  $\in$  inside (path\_image pr)  $\longleftrightarrow$  simple\_loop\_ccw p"  
 <proof>

**corollary**

assumes x: "x  $\in$  R"  
 shows winding\_number\_R\_left: "winding\_number pl x = (if simple\_loop\_ccw p then 1 else 0)"  
 and winding\_number\_R\_right: "winding\_number pr x = (if simple\_loop\_ccw p then -1 else 0)"  
 and inside\_pl\_R\_iff: "x  $\in$  inside (path\_image pl)  $\longleftrightarrow$  simple\_loop\_ccw p"  
 and inside\_pr\_R\_iff: "x  $\in$  inside (path\_image pr)  $\longleftrightarrow$  simple\_loop\_ccw p"  
 <proof>

**end**

**lemma detour\_rel\_locale\_swap:** "detour\_rel\_locale  $\varepsilon$  L R p pl pr  $\longleftrightarrow$  detour\_rel\_locale  $\varepsilon$  R L p pr pl"  
 <proof>

**lemma eq\_paths\_imp\_detour\_rel\_locale:**

assumes " $\varepsilon > 0$ " "simple\_path p" "eq\_paths p pl" "eq\_paths p pr" "valid\_path pl" "valid\_path pr"  
 shows "detour\_rel\_locale  $\varepsilon$  {} {} p pl pr"  
 <proof>

**lemma detour\_rel\_localeI [intro?]:**

assumes "detour\_rel\_aux\_locale  $\varepsilon$  (L  $\cup$  R) p pl" "detour\_rel\_aux\_locale  $\varepsilon$  (L  $\cup$  R) p pr"  
 "closed L" "closed R"  
 " $\bigwedge x. x \in L \implies$  winding\_number pl x - winding\_number pr x = -1"  
 " $\bigwedge x. x \in R \implies$  winding\_number pl x - winding\_number pr x = 1"

shows "detour\_rel\_locale  $\varepsilon$  L R p pl pr"  
 <proof>

**definition detour\_rel where**

"detour\_rel L R p pl pr  $\longleftrightarrow$   
 (simple\_path p  $\wedge$  valid\_path p  $\longrightarrow$   
 eventually ( $\lambda\varepsilon$ . detour\_rel\_locale  $\varepsilon$  L R p (pl  $\varepsilon$ ) (pr  $\varepsilon$ )) (at\_right  
 0))"

**lemma detour\_rel\_congI:**

assumes "detour\_rel L R p pl pr" "p  $\equiv_p$  p'" "valid\_path p" "L = L'"  
 "R = R'"  
 "eventually ( $\lambda\varepsilon$ . pl  $\varepsilon \equiv_p$  pl'  $\varepsilon$ ) (at\_right 0)"  
 "eventually ( $\lambda\varepsilon$ . pr  $\varepsilon \equiv_p$  pr'  $\varepsilon$ ) (at\_right 0)"  
 "eventually ( $\lambda\varepsilon$ . valid\_path (pl'  $\varepsilon$ )) (at\_right 0)"  
 "eventually ( $\lambda\varepsilon$ . valid\_path (pr'  $\varepsilon$ )) (at\_right 0)"  
 shows "detour\_rel L' R' p' pl' pr'"  
 <proof>

**lemma detour\_rel\_eq\_paths\_trans [trans]:**

assumes "p  $\equiv_p$  p'" "detour\_rel L R p' pl pr" "valid\_path p'"  
 shows "detour\_rel L R p pl pr"  
 <proof>

**lemma detour\_rel\_eq\_paths\_trans' [trans]:**

assumes "detour\_rel L R p pl pr" "p  $\equiv_p$  p'" "valid\_path p"  
 shows "detour\_rel L R p' pl pr"  
 <proof>

**lemma detour\_rel\_imp\_valid\_simple\_path:**

assumes "detour\_rel L R p pl pr" "simple\_path p" "valid\_path p"  
 shows "eventually ( $\lambda\varepsilon$ . simple\_path (pl  $\varepsilon$ )  $\wedge$  simple\_path (pr  $\varepsilon$ )  $\wedge$

valid\_path (pl  $\varepsilon$ )  $\wedge$  valid\_path (pr  $\varepsilon$ )) (at\_right 0)"  
 <proof>

### 4.3 Inference rules

**lemma detour\_rel\_image:**

assumes ind: "winding\_preserving A f ( $\lambda x$ . x)"  
 assumes ana: "f analytic\_on A"  
 assumes lipschitz: "M-lipschitz\_on A f" "M > 0"  
 assumes rel: "detour\_rel L R p pl pr"  
 assumes p: "valid\_path (f  $\circ$  p)  $\implies$  valid\_path p" "path\_image p  $\subseteq$  interior  
 A" and "closed A"  
 shows "detour\_rel (f ' L) (f ' R) (f  $\circ$  p) ( $\lambda\varepsilon$ . f  $\circ$  pl ( $\varepsilon$  / M)) ( $\lambda\varepsilon$ .  
 f  $\circ$  pr ( $\varepsilon$  / M))"

*<proof>*

**lemma** *detour\_rel\_mult:*

**assumes** "detour\_rel L R p pl pr" "c ≠ 0"

**shows** "detour\_rel ((\* c ' L) ((\* c ' R) ((\* c o p)  
(λε. (\* c o pl (ε / norm c)) (λε. (\* c o pr (ε / norm c)))"

*<proof>*

**lemma** *detour\_rel\_uminus:*

**assumes** "detour\_rel L R p pl pr"

**shows** "detour\_rel ((λx. -x) ' L) ((λx. -x) ' R) ((λx. -x) o p)  
(λε. (λx. -x) o pl ε) (λε. (λx. -x) o pr ε)"

*<proof>*

**lemma** *detour\_rel\_cnj:*

**assumes** "detour\_rel L R p pl pr"

**shows** "detour\_rel (cnj ' R) (cnj ' L) (cnj o p) (λε. cnj o pl ε)  
(λε. cnj o pr ε)"

*<proof>*

**lemma** *detour\_rel\_translate:*

**assumes** "detour\_rel L R p pl pr"

**shows** "detour\_rel ((+) c ' L) ((+) c ' R) ((+) c o p) (λε. (+) c  
o pl ε) (λε. (+) c o pr ε)"

*<proof>*

**lemma** *detour\_relI:*

**assumes** "eps > 0" "closed L" "closed R"

**assumes** "∧ε. ε > 0 ⇒ ε < eps ⇒ simple\_path p ⇒ valid\_path p  
⇒ detour\_rel\_aux\_locale ε (L ∪ R) p (pl ε)"

**assumes** "∧ε. ε > 0 ⇒ ε < eps ⇒ simple\_path p ⇒ valid\_path p ⇒  
detour\_rel\_aux\_locale ε (L ∪ R) p (pr ε)"

**assumes** "∧ε x. ε > 0 ⇒ ε < eps ⇒ x ∈ L ⇒ simple\_path p ⇒  
valid\_path p ⇒

winding\_number (pl ε) x - winding\_number (pr ε) x = -1"

**assumes** "∧ε x. ε > 0 ⇒ ε < eps ⇒ x ∈ R ⇒ simple\_path p ⇒  
valid\_path p ⇒

winding\_number (pl ε) x - winding\_number (pr ε) x = 1"

**shows** "detour\_rel L R p pl pr"

*<proof>*

**lemma** *detour\_rel\_swap:* "detour\_rel L R p pl pr ⇒ detour\_rel R L p  
pr pl"

*<proof>*

**lemma** *detour\_rel\_swap\_iff:* "detour\_rel L R p pl pr ⇔ detour\_rel R  
L p pr pl"

*<proof>*

named\_theorems detour\_rel\_intros

```
lemma detour_rel_refl [detour_rel_intros, simp, intro!]: "detour_rel
{} {} p (λ_. p) (λ_. p)"
  <proof>
```

```
lemma detour_rel_rescale:
  assumes "detour_rel L R p pl pr" "c ≥ 1"
  shows "detour_rel L R p (λε. pl (ε / c)) (λε. pr (ε / c))"
  <proof>
```

```
lemma eq_paths_imp_detour_rel:
  assumes "eq_paths p pl" "eq_paths p pr" "valid_path pl" "valid_path
pr"
  shows "detour_rel {} {} p (λ_. pl) (λ_. pr)"
  <proof>
```

```
lemma detour_rel_reverse [detour_rel_intros, intro!]:
  assumes "detour_rel L R p pl pr"
  shows "detour_rel R L (reversepath p) (λε. reversepath (pl ε)) (λε.
reversepath (pr ε))"
  <proof>
```

```
lemma detour_rel_join [detour_rel_intros, intro?]:
  assumes "detour_rel L1 R1 p1 p1 pr1"
  assumes "detour_rel L2 R2 p2 p2 pr2"
  assumes [simp]: "pathfinish p1 = pathstart p2"
  shows "detour_rel (L1 ∪ L2) (R1 ∪ R2) (p1 +++ p2) (λε. p1 ε +++
p2 ε) (λε. pr1 ε +++ pr2 ε)"
  <proof>
```

## 4.4 Basic avoidance patterns

### 4.4.1 Generic helper lemmas

```
lemma detour_rel_avoid_basic:
  fixes p :: "real ⇒ complex" and L R :: "complex set"
  defines "S ≡ (λε. path_image p ∪ (∪y∈L∪R. cball y ε))"
  assumes simple: "∧ε. ε > 0 ⇒ ε < eps ⇒ simple_path p ⇒ simple_path
(p1 ε +++ p1 ε +++ p2 ε)"
  "∧ε. ε > 0 ⇒ ε < eps ⇒ simple_path p ⇒ simple_path
(p1 ε +++ pr ε +++ p2 ε)"
  assumes LR_subset: "simple_path p ⇒ L ∪ R ⊆ path_image p"
  assumes homo: "∧ε. ε > 0 ⇒ ε < eps ⇒ simple_path p ⇒ homotopic_paths
(S ε) p (p1 ε +++ p1 ε +++ p2 ε)"
  "∧ε. ε > 0 ⇒ ε < eps ⇒ simple_path p ⇒ homotopic_paths
(S ε) p (p1 ε +++ pr ε +++ p2 ε)"
  assumes ind: "∧ε x. ε > 0 ⇒ ε < eps ⇒ simple_path p ⇒ x ∈ L
⇒ winding_number (p1 ε +++ reversepath (pr ε)) x = -1"
```

```

      " $\bigwedge \varepsilon x. \varepsilon > 0 \implies \varepsilon < \text{eps} \implies \text{simple\_path } p \implies x \in R$ "
 $\implies \text{winding\_number } (p1 \ \varepsilon \ \text{+++} \ \text{reversepath } (pr \ \varepsilon)) \ x = 1$ "
    assumes ends: " $\bigwedge \varepsilon. \varepsilon > 0 \implies \varepsilon < \text{eps} \implies \text{simple\_path } p \implies \text{pathfinish}$ 
       $(p1 \ \varepsilon) = \text{pathstart } (p2 \ \varepsilon)$ "
      " $\bigwedge \varepsilon. \varepsilon > 0 \implies \varepsilon < \text{eps} \implies \text{simple\_path } p \implies \text{pathfinish}$ 
       $(pr \ \varepsilon) = \text{pathstart } (p2 \ \varepsilon)$ "
      " $\bigwedge \varepsilon. \varepsilon > 0 \implies \varepsilon < \text{eps} \implies \text{simple\_path } p \implies \text{pathstart}$ 
       $(p1 \ \varepsilon) = \text{pathfinish } (p1 \ \varepsilon)$ "
      " $\bigwedge \varepsilon. \varepsilon > 0 \implies \varepsilon < \text{eps} \implies \text{simple\_path } p \implies \text{pathstart}$ 
       $(pr \ \varepsilon) = \text{pathfinish } (p1 \ \varepsilon)$ "
    assumes LR: " $\bigwedge \varepsilon. \varepsilon > 0 \implies \varepsilon < \text{eps} \implies \text{simple\_path } p \implies \text{path\_image}$ 
       $(p1 \ \varepsilon) \cap (L \cup R) = \{\}$ "
      " $\bigwedge \varepsilon. \varepsilon > 0 \implies \varepsilon < \text{eps} \implies \text{simple\_path } p \implies \text{path\_image}$ 
       $(pr \ \varepsilon) \cap (L \cup R) = \{\}$ "
      " $\bigwedge \varepsilon. \varepsilon > 0 \implies \varepsilon < \text{eps} \implies \text{simple\_path } p \implies \text{path\_image}$ 
       $(p1 \ \varepsilon) \cap (L \cup R) = \{\}$ "
      " $\bigwedge \varepsilon. \varepsilon > 0 \implies \varepsilon < \text{eps} \implies \text{simple\_path } p \implies \text{path\_image}$ 
       $(p2 \ \varepsilon) \cap (L \cup R) = \{\}$ "
    assumes valid: " $\bigwedge \varepsilon. \varepsilon > 0 \implies \varepsilon < \text{eps} \implies \text{valid\_path } p \implies \text{valid\_path}$ 
       $(p1 \ \varepsilon)$ "
      " $\bigwedge \varepsilon. \varepsilon > 0 \implies \varepsilon < \text{eps} \implies \text{valid\_path } p \implies \text{valid\_path}$ 
       $(pr \ \varepsilon)$ "
      " $\bigwedge \varepsilon. \varepsilon > 0 \implies \varepsilon < \text{eps} \implies \text{valid\_path } p \implies \text{valid\_path}$ 
       $(p1 \ \varepsilon)$ "
      " $\bigwedge \varepsilon. \varepsilon > 0 \implies \varepsilon < \text{eps} \implies \text{valid\_path } p \implies \text{valid\_path}$ 
       $(p2 \ \varepsilon)$ "
    assumes "eps > 0" and closed: "closed L" "closed R"
    shows "detour_rel L R p ( $\lambda \varepsilon. p1 \ \varepsilon \ \text{+++} \ p1 \ \varepsilon \ \text{+++} \ p2 \ \varepsilon$ ) ( $\lambda \varepsilon. p1 \ \varepsilon \ \text{+++}$ 
       $pr \ \varepsilon \ \text{+++} \ p2 \ \varepsilon$ )"
    <proof>

```

This is the main rule for proving the common avoidance pattern where we avoid a single bad point *bad* on a curve by replacing some segment of it with a circular arc with radius  $\varepsilon$  around *bad*. We only need to show that the original path *p* splits into segments  $p_1$ ,  $p_2$ , and  $p_3$  such that  $p_1$  and  $p_2$  do not overlap and none of the segments crosses the  $\varepsilon$ -sphere around *bad*.

**lemma** *detour\_rel\_avoid\_basic\_part\_circlepath\_left*:

```

  fixes p :: "real  $\Rightarrow$  complex" and bad :: complex and a b a' b' :: "real
 $\Rightarrow$  real"
  defines "S  $\equiv$  ( $\lambda \varepsilon. \text{path\_image } p \cup \text{cball } \text{bad } \varepsilon)$ "
  defines "c1  $\equiv$  ( $\lambda \varepsilon. \text{part\_circlepath } \text{bad } \varepsilon (a' \ \varepsilon) (b' \ \varepsilon)$ )"
  defines "c2  $\equiv$  ( $\lambda \varepsilon. \text{part\_circlepath } \text{bad } \varepsilon (a \ \varepsilon) (b \ \varepsilon)$ )"
  assumes "bad  $\in$  path_image p - {pathstart p, pathfinish p}"
  assumes eq_paths: " $\bigwedge \varepsilon. \varepsilon > 0 \implies \varepsilon < \text{eps} \implies \text{simple\_path } p \implies \text{eq\_paths}$ 
    p (p1  $\ \varepsilon \ \text{+++} \ pm \ \varepsilon \ \text{+++} \ p2 \ \varepsilon)$ "
  assumes p12_disjoint:
    " $\bigwedge \varepsilon. \varepsilon > 0 \implies \varepsilon < \text{eps} \implies \text{simple\_path } p \implies$ 
      path_image (p1  $\ \varepsilon$ )  $\cap$  path_image (p2  $\ \varepsilon$ )  $\subseteq$  {pathstart p}  $\cap$ 
      {pathfinish p}"

```

```

    assumes p1_dnc: " $\bigwedge \varepsilon. \varepsilon > 0 \implies \varepsilon < \text{eps} \implies \text{simple\_path } p \implies \text{does\_not\_cross}$ 
(p1  $\varepsilon$ ) (sphere bad  $\varepsilon$ )"
    assumes p2_dnc: " $\bigwedge \varepsilon. \varepsilon > 0 \implies \varepsilon < \text{eps} \implies \text{simple\_path } p \implies \text{does\_not\_cross}$ 
(p2  $\varepsilon$ ) (sphere bad  $\varepsilon$ )"
    assumes pm_dnc: " $\bigwedge \varepsilon. \varepsilon > 0 \implies \varepsilon < \text{eps} \implies \text{simple\_path } p \implies \text{does\_not\_cross}$ 
(pm  $\varepsilon$ ) (sphere bad  $\varepsilon$ )"
    assumes finish_p1: " $\bigwedge \varepsilon. \varepsilon > 0 \implies \varepsilon < \text{eps} \implies \text{simple\_path } p \implies \text{pathfinish}$ 
(p1  $\varepsilon$ ) = bad + rcis  $\varepsilon$  (a  $\varepsilon$ )"
    assumes start_p2: " $\bigwedge \varepsilon. \varepsilon > 0 \implies \varepsilon < \text{eps} \implies \text{simple\_path } p \implies \text{pathstart}$ 
(p2  $\varepsilon$ ) = bad + rcis  $\varepsilon$  (b  $\varepsilon$ )"
    assumes valid: " $\bigwedge \varepsilon. \varepsilon > 0 \implies \varepsilon < \text{eps} \implies \text{valid\_path } p \implies \text{valid\_path}$ 
(p1  $\varepsilon$ )"
    " $\bigwedge \varepsilon. \varepsilon > 0 \implies \varepsilon < \text{eps} \implies \text{valid\_path } p \implies \text{valid\_path}$ 
(p2  $\varepsilon$ )"
    assumes pm_ends: " $\bigwedge \varepsilon. \varepsilon > 0 \implies \varepsilon < \text{eps} \implies \text{simple\_path } p \implies \text{pathstart}$ 
(pm  $\varepsilon$ ) = pathfinish (p1  $\varepsilon$ )"
    " $\bigwedge \varepsilon. \varepsilon > 0 \implies \varepsilon < \text{eps} \implies \text{simple\_path } p \implies \text{pathfinish}$ 
(pm  $\varepsilon$ ) = pathstart (p2  $\varepsilon$ )"
    assumes ab: " $\bigwedge \varepsilon. \varepsilon > 0 \implies \varepsilon < \text{eps} \implies \text{simple\_path } p \implies a \ \varepsilon \neq b$ 
 $\varepsilon \wedge |b \ \varepsilon - a \ \varepsilon| < 2 * \text{pi}$ "
    assumes ab': " $\bigwedge \varepsilon. \varepsilon > 0 \implies \varepsilon < \text{eps} \implies \text{simple\_path } p \implies a' \ \varepsilon \neq b'$ 
 $\varepsilon \wedge |b' \ \varepsilon - a' \ \varepsilon| < 2 * \text{pi} \wedge$ 
    cis (a'  $\varepsilon$ ) = cis (a  $\varepsilon$ )  $\wedge$  cis (b'  $\varepsilon$ ) = cis (b  $\varepsilon$ )  $\wedge$ 
    b  $\varepsilon - a \ \varepsilon + a' \ \varepsilon - b' \ \varepsilon = 2 * \text{pi}$ "
    assumes "eps > 0"
    shows "detour_rel {bad} {} p ( $\lambda \varepsilon. p1 \ \varepsilon \ +++ \ c1 \ \varepsilon \ +++ \ p2 \ \varepsilon$ ) ( $\lambda \varepsilon. p1$ 
 $\varepsilon \ +++ \ cr \ \varepsilon \ +++ \ p2 \ \varepsilon$ )"
    <proof>

```

lemma detour\_rel\_avoid\_basic\_part\_circlepath\_right:

```

    fixes p :: "real  $\implies$  complex" and bad :: complex and a b a' b' :: "real
 $\implies$  real"
    defines "S  $\equiv$  ( $\lambda \varepsilon. \text{path\_image } p \cup \text{cball bad } \varepsilon$ )"
    defines "c1  $\equiv$  ( $\lambda \varepsilon. \text{part\_circlepath bad } \varepsilon$  (a'  $\varepsilon$ ) (b'  $\varepsilon$ ))"
    defines "cr  $\equiv$  ( $\lambda \varepsilon. \text{part\_circlepath bad } \varepsilon$  (a  $\varepsilon$ ) (b  $\varepsilon$ ))"
    assumes "bad  $\in$  path_image p - {pathstart p, pathfinish p}"
    assumes eq_paths: " $\bigwedge \varepsilon. \varepsilon > 0 \implies \varepsilon < \text{eps} \implies \text{simple\_path } p \implies$ 
eq_paths p (p1  $\varepsilon \ +++ \ pm \ \varepsilon \ +++ \ p2 \ \varepsilon$ )"
    assumes p12_disjoint: " $\bigwedge \varepsilon. \varepsilon > 0 \implies \varepsilon < \text{eps} \implies \text{simple\_path } p \implies$ 
path_image (p1  $\varepsilon$ )  $\cap$  path_image (p2  $\varepsilon$ )  $\subseteq$  {pathstart p}  $\cap$  {pathfinish p}"
    assumes p1_dnc: " $\bigwedge \varepsilon. \varepsilon > 0 \implies \varepsilon < \text{eps} \implies \text{simple\_path } p \implies \text{does\_not\_cross}$ 
(p1  $\varepsilon$ ) (sphere bad  $\varepsilon$ )"
    assumes p2_dnc: " $\bigwedge \varepsilon. \varepsilon > 0 \implies \varepsilon < \text{eps} \implies \text{simple\_path } p \implies \text{does\_not\_cross}$ 
(p2  $\varepsilon$ ) (sphere bad  $\varepsilon$ )"
    assumes pm_dnc: " $\bigwedge \varepsilon. \varepsilon > 0 \implies \varepsilon < \text{eps} \implies \text{simple\_path } p \implies \text{does\_not\_cross}$ 
(pm  $\varepsilon$ ) (sphere bad  $\varepsilon$ )"
    assumes finish_p1: " $\bigwedge \varepsilon. \varepsilon > 0 \implies \varepsilon < \text{eps} \implies \text{simple\_path } p \implies \text{pathfinish}$ 
(p1  $\varepsilon$ ) = bad + rcis  $\varepsilon$  (a  $\varepsilon$ )"
    assumes start_p2: " $\bigwedge \varepsilon. \varepsilon > 0 \implies \varepsilon < \text{eps} \implies \text{simple\_path } p \implies \text{pathstart}$ 

```

```

(p2 ε) = bad + rcis ε (b ε)"
  assumes pm_ends: "∧ε. ε > 0 ⇒ ε < eps ⇒ simple_path p ⇒ pathstart
(pm ε) = pathfinish (p1 ε)"
                    "∧ε. ε > 0 ⇒ ε < eps ⇒ simple_path p ⇒ pathfinish
(pm ε) = pathstart (p2 ε)"
  assumes valid: "∧ε. ε > 0 ⇒ ε < eps ⇒ valid_path p ⇒ valid_path
(p1 ε)"
                    "∧ε. ε > 0 ⇒ ε < eps ⇒ valid_path p ⇒ valid_path
(p2 ε)"
  assumes ab: "∧ε. ε > 0 ⇒ ε < eps ⇒ simple_path p ⇒ a ε ≠ b
ε ∧ |b ε - a ε| < 2 * pi"
  assumes ab': "∧ε. ε > 0 ⇒ ε < eps ⇒ simple_path p ⇒ a' ε ≠ b'
ε ∧ |b' ε - a' ε| < 2 * pi ∧
                    cis (a' ε) = cis (a ε) ∧ cis (b' ε) = cis (b ε) ∧
                    b ε - a ε + a' ε - b' ε = 2 * pi"
  assumes "eps > 0"
  shows "detour_rel {} {bad} p (λε. p1 ε +++ cr ε +++ p2 ε) (λε. p1
ε +++ cl ε +++ p2 ε)"
    ⟨proof⟩

```

#### 4.4.2 Straight line

**definition** `avoid_linepath` where

```

"avoid_linepath left a b bad ε =
  linepath a (bad - (ε / dist a b) *R (b - a)) +++
  part_circlepath bad ε (Arg (b - a) + (if left then pi else -pi))
(Arg (b - a)) +++
  linepath (bad + (ε / dist a b) *R (b - a)) b"

```

**lemma** `valid_path_avoid_linepath`:

```

  assumes "a ≠ b"
  shows "valid_path (avoid_linepath left a b bad ε)"
    ⟨proof⟩

```

**lemma** `avoid_linepath_translate`:

```

" (+) c ◦ avoid_linepath left a b bad ε =
  avoid_linepath left (c + a) (c + b) (c + bad) ε"
    ⟨proof⟩

```

**lemma** `avoid_linepath_mult`:

```

  assumes "a ≠ b"
  shows "(*) c ◦ avoid_linepath left a b 0 ε =
  avoid_linepath left (c * a) (c * b) 0 (norm c * ε)"
    ⟨proof⟩

```

**lemma** `detour_rel_linepath_semicircle_left_aux1`:

```

  assumes "a < 0" "b > 0"
  shows "detour_rel {} {} (linepath (complex_of_real a) (complex_of_real
b))"

```

```

      (avoid_linepath True (complex_of_real a) (complex_of_real
b) 0)
      (avoid_linepath False (complex_of_real a) (complex_of_real
b) 0)"
⟨proof⟩

```

**lemma** `detour_rel_linepath_semicircle_left_aux2`:

```

  assumes "0 ∈ open_segment a b"
  shows "detour_rel {0} {} (linepath a b)
        (avoid_linepath True a b 0)
        (avoid_linepath False a b 0)"
⟨proof⟩

```

**lemma** `detour_rel_linepath_semicircle_left [detour_rel_intros]`:

```

  assumes "bad ∈ open_segment a b"
  shows "detour_rel {bad} {} (linepath a b)
        (avoid_linepath True a b bad)
        (avoid_linepath False a b bad)"
⟨proof⟩

```

**lemma** `detour_rel_linepath_semicircle_right [detour_rel_intros]`:

```

  assumes "bad ∈ open_segment a b"
  shows "detour_rel {} {bad} (linepath a b)
        (avoid_linepath False a b bad)
        (avoid_linepath True a b bad)"
⟨proof⟩

```

### 4.4.3 Circular arc

**definition** `avoid_part_circlepath` ::

```

  "bool ⇒ complex ⇒ real ⇒ real ⇒ real ⇒ real ⇒ real ⇒ real
⇒ complex" where
  "avoid_part_circlepath left x r a b φ ε = (
    let s = sgn (b - a);
        bad = x + rcis r φ;
        α = 2 * arcsin (ε / (2 * r));
        β = arcsin (ε / (2 * r)) + pi / 2
    in part_circlepath x r a (φ - s * α) +++
       part_circlepath bad ε (φ - s * β + (if left = (a > b) then 0
else 2 * s * pi)) (φ + s * β) +++
       part_circlepath x r (φ + s * α) b)"

```

**lemma** `avoid_part_circlepath_translate`:

```

  "(+) c ◦ avoid_part_circlepath left x r a b φ ε =
  avoid_part_circlepath left (c + x) r a b φ ε"
⟨proof⟩

```

**lemma** `avoid_part_circlepath_mult_scaleR_0`:

```

  assumes [simp]: "bad ≠ 0" and "c > 0"

```

```

  shows "(*_R) c ◦ avoid_part_circlepath left 0 r a b φ ε =
         avoid_part_circlepath left 0 (norm c * r) a b φ (norm c *
ε)"
⟨proof⟩

```

```

lemma avoid_part_circlepath_mult:
  assumes [simp]: "c ≠ 0"
  shows "(*) c ◦ avoid_part_circlepath left 0 r a b φ ε =
         avoid_part_circlepath left 0 (norm c * r)
         (a + Arg c) (b + Arg c) (φ + Arg c) (norm c * ε)"
⟨proof⟩

```

```

lemma avoid_part_circlepath_cong:
  assumes "cis a = cis a'" "b' = b + a' - a" "φ' = φ + a' - a"
  shows "avoid_part_circlepath left x r a b φ =
         avoid_part_circlepath left x r a' b' φ'"
⟨proof⟩

```

```

lemma avoid_part_circlepath_wf_aux1:
  assumes "r ≥ 0" "r ≤ 2"
  shows "(1 + of_real r * cis (arcsin (r / 2) + pi / 2)) = cis (2 *
arcsin (r / 2))"
⟨proof⟩

```

```

locale avoid_part_circlepath_locale =
  fixes x :: complex and r a b φ ε :: real
  assumes ε: "ε > 0" "ε < 2 * r" and ab: "a ≠ b"
begin

```

```

definition s where "s = sgn (b - a)"
definition bad where "bad = x + rcis r φ"
definition "α = 2 * arcsin (ε / (2 * r))"
definition "β = arcsin (ε / (2 * r)) + pi / 2"

```

```

lemma ends:
  "x + rcis r (φ - s * α) = bad +
   rcis ε (φ - s * β + (if left = (a > b) then 0 else 2 * s * pi))"
(is ?th1)
  "x + rcis r (φ + s * α) = bad + rcis ε (φ + s * β)" (is ?th2)
⟨proof⟩

```

```

lemma valid_path: "valid_path (avoid_part_circlepath left x r a b φ
ε)"
⟨proof⟩

```

```

end

```

```

lemma valid_path_avoid_part_circlepath:

```

```

    assumes " $\varepsilon \in \{0 < \dots < 2 * r\}$ " "a  $\neq$  b"
    shows "valid_path (avoid_part_circlepath left x r a b  $\varphi$   $\varepsilon$ )"
  <proof>

lemma valid_path_avoid_part_circlepath':
  assumes "a  $\neq$  b" "r > 0"
  shows "eventually ( $\lambda \varepsilon$ . valid_path (avoid_part_circlepath left x r
a b  $\varphi$   $\varepsilon$ )) (at_right 0)"
  <proof>

lemma sphere_inter_sphere_pos_real_line_aux1:
  assumes r: "r  $\geq$  0" "r  $\leq$  1"
  shows "dist 1 (cis (2 * arcsin (r / 2))) = r"
  <proof>

lemma sphere_inter_sphere_pos_real_line_aux2':
  assumes r: "r  $\geq$  0" "r  $\leq$  1"
  assumes dist: "dist 1 (cis x) = r"
  shows "[x = 2 * arcsin (r / 2)] (rmod 2 * pi)  $\vee$  [x = -2 * arcsin (r
/ 2)] (rmod 2 * pi)"
  <proof>

lemma sphere_inter_sphere_pos_real_line_aux2:
  assumes r: "r  $\geq$  0" "r  $\leq$  1" and x: "x  $\in$  {-pi..pi}"
  assumes dist: "dist 1 (cis x) = r"
  shows "|x| = 2 * arcsin (r / 2)"
  <proof>

lemma sphere_inter_sphere_aux1:
  assumes r: "r  $\geq$  0" "r  $\leq$  1"
  shows "dist (cis  $\varphi$ ) (cis ( $\varphi$  + 2 * arcsin (r / 2))) = r"
  <proof>

lemma sphere_inter_sphere_aux2:
  assumes r: "r  $\geq$  0" "r  $\leq$  1"
  assumes dist: "dist (cis  $\varphi$ ) (cis x) = r"
  shows "[x =  $\varphi$  + 2 * arcsin (r / 2)] (rmod 2 * pi)  $\vee$  [x =  $\varphi$  - 2 * arcsin
(r / 2)] (rmod 2 * pi)"
  <proof>

lemma detour_rel_part_circlepath_semicircle_left_aux1:
  assumes ab: "a < 0" "0 < b" "a  $\geq$  -pi" "b  $\leq$  pi"
  shows "detour_rel {1} {} (part_circlepath 0 1 a b)
(avoid_part_circlepath True 0 1 a b 0)
(avoid_part_circlepath False 0 1 a b 0)"
  <proof>

lemma avoid_part_circlepath_extend_right:
  assumes "a < b  $\wedge$  b < c  $\vee$  c < b  $\wedge$  b < a" " $\varphi \in$  open_segment a b" "r

```

```

> 0"
  assumes  $\varepsilon$ : " $\varepsilon > 0$ " " $\varepsilon < r * (2 * \sin (|b - \varphi| / 2))$ "
  shows "avoid_part_circlepath left x r a b  $\varphi$   $\varepsilon$  +++ part_circlepath x
r b c  $\equiv_p$ 
      avoid_part_circlepath left x r a c  $\varphi$   $\varepsilon$ "
<proof>

```

```

lemma avoid_part_circlepath_extend_left:
  assumes " $c < a \wedge a < b \vee c > a \wedge a > b$ " " $\varphi \in \text{open\_segment } a \ b$ " " $r > 0$ "
  assumes  $\varepsilon$ : " $\varepsilon > 0$ " " $\varepsilon < r * (2 * \sin (|a - \varphi| / 2))$ "
  shows "part_circlepath x r c a +++ avoid_part_circlepath left x r
a b  $\varphi$   $\varepsilon \equiv_p$ 
      avoid_part_circlepath left x r c b  $\varphi$   $\varepsilon$ "
<proof>

```

```

lemma avoid_part_circlepath_reverse:
  assumes " $a \neq b$ " " $\varepsilon > 0$ " " $\varepsilon < 2 * r$ "
  shows "reversepath (avoid_part_circlepath left x r a b  $\varphi$   $\varepsilon$ )  $\equiv_p$ 
      avoid_part_circlepath ( $\neg$ left) x r b a  $\varphi$   $\varepsilon$ "
<proof>

```

```

lemma detour_rel_part_circlepath_semicircle_left_aux2:
  assumes ab: " $a < 0$ " " $0 < b$ "
  shows "detour_rel {1} {} (part_circlepath 0 1 a b)
      (avoid_part_circlepath True 0 1 a b 0)
      (avoid_part_circlepath False 0 1 a b 0)"
<proof>

```

```

lemma detour_rel_part_circlepath_semicircle_left_aux3:
  assumes ab: " $0 \in \text{open\_segment } a \ b$ "
  shows "detour_rel {1} {} (part_circlepath 0 1 a b)
      (avoid_part_circlepath True 0 1 a b 0)
      (avoid_part_circlepath False 0 1 a b 0)"
<proof>

```

```

lemma detour_rel_part_circlepath_semicircle_left_aux4:
  assumes " $\varphi \in \text{open\_segment } a \ b$ " " $\text{bad} = \text{rcis } r \ \varphi$ " " $r > 0$ "
  shows "detour_rel {bad} {} (part_circlepath 0 r a b)
      (avoid_part_circlepath True 0 r a b  $\varphi$ )
      (avoid_part_circlepath False 0 r a b  $\varphi$ )"
<proof>

```

```

lemma detour_rel_part_circlepath_semicircle_left [detour_rel_intros]:

```

```

assumes " $\varphi \in \text{open\_segment } a \ b$ " " $\text{bad} = x + \text{rcis } r \ \varphi$ " " $r > 0$ "
shows " $\text{detour\_rel } \{\text{bad}\} \ \{\}$  ( $\text{part\_circlepath } x \ r \ a \ b$ )
        ( $\text{avoid\_part\_circlepath } \text{True } \ x \ r \ a \ b \ \varphi$ )
        ( $\text{avoid\_part\_circlepath } \text{False } \ x \ r \ a \ b \ \varphi$ )"
<proof>

```

```

lemma  $\text{detour\_rel\_part\_circlepath\_semicircle\_right}$  [ $\text{detour\_rel\_intros}$ ]:
assumes " $\varphi \in \text{open\_segment } a \ b$ " " $\text{bad} = x + \text{rcis } r \ \varphi$ " " $r > 0$ "
shows " $\text{detour\_rel } \ \{\} \ \{\text{bad}\}$  ( $\text{part\_circlepath } x \ r \ a \ b$ )
        ( $\text{avoid\_part\_circlepath } \text{False } \ x \ r \ a \ b \ \varphi$ )
        ( $\text{avoid\_part\_circlepath } \text{True } \ x \ r \ a \ b \ \varphi$ )"
<proof>

```

#### 4.4.4 Line–line corner

```

definition  $\text{avoid\_linepath\_linepath}$  where
  " $\text{avoid\_linepath\_linepath } \text{left } a \ b \ c \ \varepsilon = ($ 
     $\text{let } s = \text{sgn } (\text{Arg } (c - b) - \text{Arg } (a - b));$ 
     $\delta = (\text{if } \text{left} \longleftrightarrow \text{Arg } (a - b) \geq \text{Arg } (c - b) \text{ then } 0 \text{ else } 2 * s$ 
   $* \text{pi})$ 
  in  $\text{linepath } a \ (b + \varepsilon *_{\mathbb{R}} \text{sgn } (a - b)) \ \text{+++}$ 
   $\text{part\_circlepath } b \ \varepsilon \ (\text{Arg } (a - b) + \delta) \ (\text{Arg } (c - b)) \ \text{+++}$ 
   $\text{linepath } (b + \varepsilon *_{\mathbb{R}} \text{sgn } (c - b)) \ c$ "

```

```

lemma  $\text{avoid\_linepath\_linepath\_translate}$ :
  " $(+) \ d \circ \text{avoid\_linepath\_linepath } \text{left } a \ b \ c \ \varepsilon =$ 
   $\text{avoid\_linepath\_linepath } \text{left } (d + a) \ (d + b) \ (d + c) \ \varepsilon$ "
<proof>

```

```

lemma  $\text{avoid\_linepath\_linepath\_mult}$ :
assumes " $a \neq 0$ " " $b \neq 0$ "
shows " $(*) \ c \circ \text{avoid\_linepath\_linepath } \text{left } a \ 0 \ b \ \varepsilon =$ 
   $\text{avoid\_linepath\_linepath } \text{left } (c * a) \ 0 \ (c * b) \ (\text{norm } c * \varepsilon)$ "
<proof>

```

```

lemma  $\text{norm\_linepath}_0$ : " $\text{norm } (\text{linepath } 0 \ a \ x) = |x| * \text{norm } a$ "
<proof>

```

```

lemma  $\text{norm\_linepath}_0'$ : " $\text{norm } (\text{linepath } a \ 0 \ x) = |1 - x| * \text{norm } a$ "
<proof>

```

```

lemma  $\text{detour\_rel\_avoid\_linepath\_linepath\_aux1}$ :
assumes " $1 \notin \text{closed\_segment } 0 \ c$ " " $c \notin \text{closed\_segment } 0 \ 1$ " " $\text{Arg } c >$ 
   $0$ "
shows " $\text{detour\_rel } \{0\} \ \{\}$  ( $\text{linepath } 1 \ 0 \ \text{+++} \ \text{linepath } 0 \ c$ )
        ( $\text{avoid\_linepath\_linepath } \text{True } 1 \ 0 \ c$ ) ( $\text{avoid\_linepath\_linepath}$ 
   $\text{False } 1 \ 0 \ c$ )"
<proof>

```

```

lemma detour_rel_avoid_linepath_linepath_aux2:
  assumes "1  $\notin$  closed_segment 0 c" "c  $\notin$  closed_segment 0 1"
  shows "detour_rel {0} {} (linepath 1 0 +++ linepath 0 c)
        (avoid_linepath_linepath True 1 0 c) (avoid_linepath_linepath
False 1 0 c)"
  <proof>

```

```

lemma detour_rel_avoid_linepath_linepath_aux3:
  assumes "a  $\notin$  closed_segment 0 c" "c  $\notin$  closed_segment a 0"
  shows "detour_rel {0} {} (linepath a 0 +++ linepath 0 c)
        (avoid_linepath_linepath True a 0 c) (avoid_linepath_linepath
False a 0 c)"
  <proof>

```

```

lemma detour_rel_avoid_linepath_linepath_left [detour_rel_intros]:
  assumes "a  $\notin$  closed_segment b c" "c  $\notin$  closed_segment a b"
  shows "detour_rel {b} {} (linepath a b +++ linepath b c)
        (avoid_linepath_linepath True a b c) (avoid_linepath_linepath
False a b c)"
  <proof>

```

```

lemma detour_rel_avoid_linepath_linepath_right [detour_rel_intros]:
  assumes "a  $\notin$  closed_segment b c" "c  $\notin$  closed_segment a b"
  shows "detour_rel {} {b} (linepath a b +++ linepath b c)
        (avoid_linepath_linepath False a b c) (avoid_linepath_linepath
True a b c)"
  <proof>

```

#### 4.4.5 Line–arc corner

Avoidance pattern for a bad point lying on the junction between a straight vertical line coming from above to the point  $e^{2i\pi/3}$  and a circular arc with radius 1 around the origin that continues from there to the right.

The bad point is avoided by cutting out a circle of radius  $\varepsilon$  around it from the path and replacing the removed section with a circular arc of radius  $\varepsilon$  around the bad point.

```

definition avoid_linepath_circlepath where
  "avoid_linepath_circlepath left y a  $\varepsilon$  =
    (let  $\alpha$  = 2 * arcsin ( $\varepsilon$  / 2);
        bad = cis (2*pi/3);
         $\beta$  = pi / 3 + arcsin ( $\varepsilon$  / 2)
    in linepath (-1/2 + y *R i) (bad +  $\varepsilon$  *R i) +++
      part_circlepath bad  $\varepsilon$  (pi/2) (pi/2 -  $\beta$  + (if left then 0 else
2 * pi)) +++
      part_circlepath 0 1 (2*pi/3 -  $\alpha$ ) a)"

```

```

lemma cis_double: "cis (2 * x) = cis x ^ 2"
  ⟨proof⟩

lemma valid_path_avoid_linepath_circlepath:
  assumes "ε ≥ 0" "ε ≤ 2"
  shows "valid_path (avoid_linepath_circlepath left y a ε)"
  ⟨proof⟩

locale avoid_linepath_circlepath_locale =
  fixes y a ε :: real
  assumes ε: "ε ∈ {0<..<2}"
begin

definition α where "α = 2 * arcsin (ε / 2)"
definition bad where "bad = cis (2 * pi / 3)"
definition β where "β = pi / 3 + arcsin (ε / 2)"

lemma ends: "bad + ε *R i = bad + rcis ε (pi / 2)"
  "bad + rcis ε (pi/2 - β + (if left then 2 * pi else 0)) =
cis (2*pi/3 - α)"
  ⟨proof⟩

end

lemma detour_rel_avoid_linepath_circlepath_left:
  fixes a b c :: real
  assumes "y > sqrt 3 / 2" "a > pi / 2" "a < 2 * pi / 3"
  defines "bad ≡ cis (2 * pi / 3)"
  shows "detour_rel {bad} {} (linepath (-1/2 + y *R i) bad +++ part_circlepath
0 1 (2*pi/3) a)
  (avoid_linepath_circlepath True y a) (avoid_linepath_circlepath
False y a)"
  ⟨proof⟩

lemmas detour_rel_avoid_linepath_circlepath_right =
  detour_rel_swap[OF detour_rel_avoid_linepath_circlepath_left]

```

## 4.5 Consequences of the detour relation

```

locale detour =
  aux: detour_rel_aux_locale ε "I ∪ X" p p'
  for ε :: real and I X P :: "complex set" and p_orig p p' :: "real ⇒
complex" +
  assumes eq_loops: "eq_loops p_orig p"
  assumes simple_loop_original: "simple_loop p_orig"
  assumes same_orientation': "simple_loop_orientation p' = simple_loop_orientation
p"
  assumes I_inside: "I ⊆ inside (path_image p)"

```

```

    assumes X_outside: "X ⊆ outside (path_image p')"
    assumes disjoint: "P ∩ (path_image p_orig ∪ (⋃x∈I∪X. cball x ε))
= {}"
begin

```

```

lemma same_orientation: "simple_loop_orientation p' = simple_loop_orientation
p_orig"
  ⟨proof⟩

```

$p'$  is a simple loop that can be obtained from  $p$  through local deformations in  $\varepsilon$ -neighbourhoods of  $I \cup X$ .

```

lemma same_ends: "pathstart p' = pathstart p" "pathfinish p' = pathfinish
p"
  ⟨proof⟩

```

```

lemmas valid = aux.p'_valid

```

```

lemma simple_loop: "simple_loop p'"
  ⟨proof⟩

```

```

lemma path_image_eq [simp]: "path_image p = path_image p_orig"
  ⟨proof⟩

```

```

lemma homotopic': "homotopic_paths (path_image p_orig ∪ (⋃x∈I ∪ X.
cball x ε)) p p'"
  ⟨proof⟩

```

```

lemma homotopic'': "homotopic_loops (path_image p_orig ∪ (⋃x∈I ∪ X.
cball x ε)) p p'"
  ⟨proof⟩

```

```

lemma homotopic: "homotopic_loops (path_image p_orig ∪ (⋃x∈I ∪ X.
cball x ε)) p_orig p'"
  ⟨proof⟩

```

```

lemma path_image_subset: "path_image p' ⊆ path_image p_orig ∪ (⋃x∈I∪X.
cball x ε)"
  ⟨proof⟩

```

All the points in  $I$  are strictly inside  $p'$ , all the points in  $X$  are strictly outside, and all the points in  $P$  are unchanged (i.e. if they were outside before they still are and if they were inside before, they still are).

```

lemma I_not_on_path: "I ∩ path_image p' = {}"
  ⟨proof⟩

```

```

lemma X_not_on_path: "X ∩ path_image p' = {}"
  ⟨proof⟩

lemma P_not_on_path: "P ∩ path_image p' = {}"
  ⟨proof⟩

lemma I_X_disjoint: "I ∩ X = {}"
  ⟨proof⟩

lemma I_P_disjoint: "I ∩ P = {}"
  ⟨proof⟩

lemma X_P_disjoint: "X ∩ P = {}"
  ⟨proof⟩

lemma winding_number_unchanged:
  assumes "z ∈ P"
  shows "winding_number p' z = winding_number p_orig z"
  ⟨proof⟩

lemma P_inside_iff:
  assumes "z ∈ P"
  shows "z ∈ inside (path_image p') ↔ z ∈ inside (path_image p_orig)"
  ⟨proof⟩

lemma P_outside_iff:
  assumes "z ∈ P"
  shows "z ∈ outside (path_image p') ↔ z ∈ outside (path_image p_orig)"
  ⟨proof⟩

lemma winding_number_I:
  assumes "z ∈ I"
  shows "winding_number p' z = simple_loop_orientation p_orig"
  ⟨proof⟩

lemma winding_number_X:
  assumes "z ∈ X"
  shows "winding_number p' z = 0"
  ⟨proof⟩

end

```

Our final result: If  $p$  is a simple closed curve that is in detour relation with a family of deformed versions  $p'$  of itself, then for any closed set  $P$  of points of interest not on  $p$  there is an  $\varepsilon > 0$  such that all curves  $p'(\varepsilon)$  for  $\varepsilon' < \varepsilon$  have basically the same properties as  $p$ ; namely:

- $p'(\varepsilon)$  is  $pr(\varepsilon)$  also a closed curve with the same orientation and the same start/end as  $p$ .

- $p'(\varepsilon)$  is homotopic to  $p$  with a homotopy transformation that only passes through the original path image of  $p$  plus  $\varepsilon$ -balls around  $I \cup X$ . In other words,  $p$  is identical to  $p'(\varepsilon)$  except for small deformations of size  $\varepsilon$  around  $I \cup X$ .
- All points in  $I$  (“include”) are inside  $p'(\varepsilon)$  and all points in  $X$  (“exclude”) are outside.
- Each point in  $P$  (“preserve”) is in  $p'(\varepsilon)$  if and only if it was already in  $p$ .

**theorem** *detour\_generic*:

```

assumes rel: "detour_rel I X p pl pr"
assumes eq: "eq_loops p_orig p"
assumes p_orig: "simple_loop p_orig" and valid: "valid_path p"
assumes P: "closed P" "P ∩ path_image p_orig = {}"
defines "p' ≡ (if simple_loop_ccw p_orig then pr else pl)"
shows   "eventually (λε. detour ε I X P p_orig p (p' ε)) (at_right 0)"
⟨proof⟩

```

**corollary** *detour\_ccw*:

```

assumes "detour_rel I X p pl pr" "p_orig ≡○ p"
assumes "simple_loop_ccw p_orig" "valid_path p"
assumes "closed P" "P ∩ path_image p_orig = {}"
shows   "eventually (λε. detour ε I X P p_orig p (pr ε)) (at_right 0)"
⟨proof⟩

```

**corollary** *detour\_cw*:

```

assumes "detour_rel I X p pl pr" "p_orig ≡○ p"
assumes "simple_loop_cw p_orig" "valid_path p" "closed P" "P ∩ path_image
p_orig = {}"
shows   "eventually (λε. detour ε I X P p_orig p (pl ε)) (at_right 0)"
⟨proof⟩

```

**end**