safe-curves

raya

November 23, 2019

Contents

theory *Hales*
 imports *Complex-Main HOL−Algebra.Group HOL−Algebra.Bij*
  HOL−Library.Bit HOL−Library.Rewrite
begin

1 Affine Edwards curves

```
class ell-field = field +
  assumes two-not-zero: 2 ≠ 0

locale curve-addition =
  fixes c d :: 'a::ell-field
begin

definition e :: 'a ⇒ 'a ⇒ 'a where
 e x y = x^2 + c * y^2 − 1 − d * x^2 * y^2

definition delta-plus :: 'a ⇒ 'a ⇒ 'a ⇒ 'a ⇒ 'a where
 delta-plus x1 y1 x2 y2 = 1 + d * x1 * y1 * x2 * y2

definition delta-minus :: 'a ⇒ 'a ⇒ 'a ⇒ 'a ⇒ 'a where
 delta-minus x1 y1 x2 y2 = 1 − d * x1 * y1 * x2 * y2

definition delta :: 'a ⇒ 'a ⇒ 'a ⇒ 'a ⇒ 'a where
 delta x1 y1 x2 y2 = (delta-plus x1 y1 x2 y2) *
                     (delta-minus x1 y1 x2 y2)

lemma delta-com:
  (delta x0 y0 x1 y1 = 0) = (delta x1 y1 x0 y0 = 0)
  unfolding delta-def delta-plus-def delta-minus-def
  by algebra

fun add :: 'a × 'a ⇒ 'a × 'a ⇒ 'a × 'a where
 add (x1,y1) (x2,y2) =
    ((x1*x2 − c*y1*y2) div (1−d*x1*y1*x2*y2),
     (x1*y2+y1*x2) div (1+d*x1*y1*x2*y2))

lemma commutativity: add z1 z2 = add z2 z1
  by(cases z1,cases z2,simp add: algebra-simps)

lemma add-closure:
  assumes z3 = (x3,y3) z3 = add (x1,y1) (x2,y2)
  assumes delta-minus x1 y1 x2 y2 ≠ 0 delta-plus x1 y1 x2 y2 ≠ 0
  assumes e x1 y1 = 0 e x2 y2 = 0
  shows e x3 y3 = 0
proof −
  have x3-expr: x3 = (x1*x2 − c*y1*y2) div (delta-minus x1 y1 x2 y2)
    using assms delta-minus-def by auto
  have y3-expr: y3 = (x1*y2+y1*x2) div (delta-plus x1 y1 x2 y2)
    using assms delta-plus-def by auto
  define prod where prod =
    −1 + x1^2 * x2^2 + c * x2^2 * y1^2 − d * x1^2 * x2^4 * y1^2 +
     c * x1^2 * y2^2 − d * x1^4 * x2^2 * y2^2 + c^2 * y1^2 * y2^2 −
     4 * c * d * x1^2 * x2^2 * y1^2 * y2^2 +
```

$2 * d^2 * x1^2 * x2^2 * y1^2 * y2^2 + d^2 * x1^4 * x2^4 * y1^2 * y2^2 -$
$c^2 * d * x2^2 * y1^4 * y2^2 + c * d^2 * x1^2 * x2^4 * y1^4 * y2^2 -$
$c^2 * d * x1^2 * y1^2 * y2^4 + c * d^2 * x1^4 * x2^2 * y1^2 * y2^4 +$
$c^2 * d^2 * x1^2 * x2^2 * y1^4 * y2^4 -$
$d^4 * x1^4 * x2^4 * y1^4 * y2^4$

define *e1* **where** *e1 = e x1 y1*
define *e2* **where** *e2 = e x2 y2*

have *prod-eq-1*: ∃ *r1 r2. prod − (r1 ∗ e1 + r2 ∗ e2) = 0*
unfolding *prod-def e1-def e2-def e-def*
by *algebra*

define *a* **where** *a = x1∗x2 − c∗y1∗y2*
define *b* **where** *b = x1∗y2+y1∗x2*

have *(e x3 y3)∗(delta x1 y1 x2 y2)*^{2} =
e (a div (delta-minus x1 y1 x2 y2))
(b div (delta-plus x1 y1 x2 y2)) ∗ (delta x1 y1 x2 y2)^2
unfolding *a-def b-def*
by *(simp add: mult.commute mult.left-commute x3-expr y3-expr)*
also have ... =
((a div delta-minus x1 y1 x2 y2)^2 +
c ∗ (b div delta-plus x1 y1 x2 y2)^2 −
1 −
d ∗ (a div delta-minus x1 y1 x2 y2)^2 ∗
(b div delta-plus x1 y1 x2 y2)^2*) ∗ (delta x1 y1 x2 y2)*^{2}
unfolding *delta-plus-def delta-minus-def delta-def e-def* **by** *simp*
also have ... =
((a div delta-minus x1 y1 x2 y2)^2 *∗ (delta x1 y1 x2 y2)*^{2} +
c ∗ (b div delta-plus x1 y1 x2 y2)^2 *∗ (delta x1 y1 x2 y2)*^{2} −
1 ∗ (delta x1 y1 x2 y2)^2 −
d ∗ (a div delta-minus x1 y1 x2 y2)^2 ∗
(b div delta-plus x1 y1 x2 y2)^2 *∗ (delta x1 y1 x2 y2)*^{2}*)*
by*(simp add: algebra-simps)*
also have ... =
((a ∗ delta-plus x1 y1 x2 y2)^2 + *c ∗ (b ∗ delta-minus x1 y1 x2 y2)*^{2} −
(delta x1 y1 x2 y2)^2 − *d ∗ a*^{2} *∗ b*^{2}*)*
unfolding *delta-def* **by***(simp add: field-simps assms(3,4))+*
also have ... − *prod = 0*
unfolding *prod-def delta-plus-def delta-minus-def delta-def a-def b-def* **by** *algebra*
finally have *(e x3 y3)∗(delta x1 y1 x2 y2)*^{2} = *prod* **by** *simp*
then have *prod-eq-2*: *(e x3 y3) = prod div (delta x1 y1 x2 y2)*^{2}
using *assms(3,4) delta-def* **by** *auto*

have *e1 = 0* **unfolding** *e1-def* **using** *assms(5)* **by** *simp*
moreover have *e2 = 0* **unfolding** *e2-def* **using** *assms(6)* **by** *simp*
ultimately have *prod = 0* **using** *prod-eq-1* **by** *simp*

then show *e x3 y3 = 0* **using** *prod-eq-2* **by** *simp*
qed

lemma *associativity*:
assumes *z1' = (x1',y1') z3' = (x3',y3')*
assumes *z1' = add (x1,y1) (x2,y2) z3' = add (x2,y2) (x3,y3)*
assumes *delta-minus x1 y1 x2 y2 ≠ 0 delta-plus x1 y1 x2 y2 ≠ 0*
delta-minus x2 y2 x3 y3 ≠ 0 delta-plus x2 y2 x3 y3 ≠ 0
delta-minus x1' y1' x3 y3 ≠ 0 delta-plus x1' y1' x3 y3 ≠ 0
delta-minus x1 y1 x3' y3' ≠ 0 delta-plus x1 y1 x3' y3' ≠ 0
assumes *e x1 y1 = 0 e x2 y2 = 0 e x3 y3 = 0*
shows *add (add (x1,y1) (x2,y2)) (x3,y3) = add (x1,y1) (add (x2,y2) (x3,y3))*

proof −
define *e1* **where** *e1 = e x1 y1*
define *e2* **where** *e2 = e x2 y2*
define *e3* **where** *e3 = e x3 y3*
define $Delta_x$ **where** $Delta_x$ =
(delta-minus x1' y1' x3 y3)∗(delta-minus x1 y1 x3' y3')∗
(delta x1 y1 x2 y2)∗(delta x2 y2 x3 y3)
define $Delta_y$ **where** $Delta_y$ =
(delta-plus x1' y1' x3 y3)∗(delta-plus x1 y1 x3' y3')∗
(delta x1 y1 x2 y2)∗(delta x2 y2 x3 y3)
define g_x **where** g_x = *fst(add z1' (x3,y3)) − fst(add (x1,y1) z3')*
define g_y **where** g_y = *snd(add z1' (x3,y3)) − snd(add (x1,y1) z3')*
define *gxpoly* **where** *gxpoly* = g_x ∗ $Delta_x$
define *gypoly* **where** *gypoly* = g_y ∗ $Delta_y$

define *gxpoly-expr* **where** *gxpoly-expr =*
d∗x2∗ y2∗ (x1^2∗ x2∗ x3∗ y1−x1^2∗ x2∗ x3^3∗ y1−c∗ x1∗ x3∗ y1^2∗ y2+d∗ x1^3∗ x2^2∗ x3∗ y1^2∗ y2
+c∗ x1∗ x3^3∗ y1^2∗ y2−d∗ x1^3∗ x2^2∗ x3^3∗ y1^2∗ y2−c∗ d∗ x1^2∗ x2∗ x3∗ y1^3∗ y2^2+c∗ d∗ x1^2∗ x2∗ x3^3∗ y1^3∗ y2^2
−x1∗ x2∗ x3^2∗ y3+x1^3∗ x2∗ x3^2∗ y3+c∗ x1∗ x2∗ y1^2∗ y3−d∗ x1^3∗ x2^3∗ x3^2∗ y1^2∗ y3+c∗ x1^2∗ y1∗ y2∗ y3
−c∗ x3^2∗ y1∗ y2∗ y3−c∗ d∗ x1^2∗ x2^2∗ y1^3∗ y2∗ y3+c^2∗ x3^2∗ y1^3∗ y2∗ y3−c∗ d∗ x1^3∗ x2∗ y1^2∗ y2^2∗ y3
+d^2∗ x1^3∗ x2^3∗ x3^2∗ y1^2∗ y2^2∗ y3−c^2∗ d∗ x1^2∗ x3^2∗ y1^3∗ y2^3∗ y3+c∗ d^2∗ x1^2∗ x2^2∗ x3^2∗ y1^3∗ y2^3∗ y3
−c∗ x2∗ x3∗ y1∗ y3^2+d∗ x1^2∗ x2^3∗ x3^3∗ y1∗ y3^2+c^2∗ x2∗ x3∗ y1^3∗ y3^2−c∗ d∗ x1^2∗ x2^3∗ x3∗ y1^3∗ y3^2
+c∗ x1∗ x3∗ y2∗ y3^2−c∗ x1^3∗ x3∗ y2∗ y3^2−d∗ x1∗ x2^2∗ x3^3∗ y2∗ y3^2+d∗ x1^3∗ x2^2∗ x3^3∗ y2∗ y3^2
+c∗ d∗ x2∗ x3^3∗ y1∗ y2^2∗ y3^2−d^2∗ x1^2∗ x2^3∗ x3^3∗ y1∗ y2^2∗ y3^2+c∗ d^2∗ x1^2∗ x2^3∗ x3∗ y1^3∗ y2^2∗ y3^2
−c^2∗ d ∗x2∗ x3^3∗ y1^3∗ y2^2∗ y3^2+c^2∗ d∗ x1^3∗ x3∗ y1^2∗ y2^3∗ y3^2−c∗ d^2∗ x1^3 ∗x2^2∗ x3∗ y1^2∗ y2^3∗ y3^2
−c^2∗ d∗ x1∗ x3^3∗ y1^2∗ y2^3∗ y3^2+c∗ d^2∗ x1∗ x2^2∗ x3^3∗ y1^2∗ y2^3∗ y3^2−c^2∗ x1∗ x2∗ y1^2∗ y3^3

+c* d* x1* x2^3* x3^2* y1^2* y3^3−c^2* x1^2* y1* y2* y3^3+c* d* x2^2* x3^2* y1* y2* y3^3+c^2* d* x1^2* x2^2* y1^3* y2* y3^3
−c^2* d* x2^2* x3^2* y1^3* y2* y3^3+c* d* x1* x2* x3^2* y2^2* y3^3−c* d *x1^3* x2* x3^2* y2^2* y3^3
+c^2* d* x1^3* x2* y1^2* y2^2* y3^3−c* d^2* x1* x2^3* x3^2* y1^2* y2^2* y3^3+c^2* d* x1^2* x3^2* y1* y2^3* y3^3
−c* d^2* x1^2* x2^2* x3^2* y1* y2^3* y3^3)

define *gypoly-expr* **where** *gypoly-expr* =
−d* x2* y2* (x1* x2* x3* y1^2−x1* x2* x3^3* y1^2+x1^2* x3* y1* y2−x1^2* x3^3* y1* y2−d* x1^2* x2^2* x3* y1^3* y2
+d* x1^2* x2^2* x3^3* y1^3* y2−d* x1^3* x2* x3* y1^2* y2^2+d* x1^3* x2* x3^3* y1^2 *y2^2−x1^2* x2* y1* y3
+x2* x3^2* y1* y3−c* x2* x3^2* y1^3* y3+d* x1^2* x2^3* x3^2* y1^3* y3−x1* x3^2* y2* y3+x1^3* x3^2* y2* y3
+c* x1* y1^2* y2* y3−d* x1^3* x2^2* y1^2* y2* y3+c* d* x1^2* x2* y1^3* y2^2* y3−d^2* x1^2* x2^3* x3^2* y1^3* y2^2* y3
−c* d* x1^3* x3^2* y1^2* y2^3* y3+d^2* x1^3* x2^2* x3^2* y1^2* y2^3* y3−x1* x2* x3* y3^2+x1^3* x2* x3* y3^2
−d* x1^3* x2^3* x3* y1^2* y3^2+d* x1* x2^3* x3^3* y1^2* y3^2−c* x3* y1* y2* y3^2+d *x2^2* x3^3* y1* y2* y3^2
+c^2* x3* y1^3* y2* y3^2−c* d* x2^2* x3^3* y1^3* y2 *y3^2+d* x1* x2* x3^3* y2^2* y3^2−d* x1^3* x2* x3^3* y2^2* y3^2
+d^2* x1^3* x2^3* x3 *y1^2* y2^2* y3^2−d^2* x1* x2^3* x3^3* y1^2* y2^2* y3^2+c* d* x1^2* x3^3* y1* y2^3* y3^2
−d^2* x1^2* x2^2* x3^3* y1* y2^3* y3^2−c^2* d* x1^2* x3* y1^3* y2^3* y3^2+c* d^2* x1^2* x2^2* x3* y1^3* y2^3* y3^2
+c* x1^2* x2* y1* y3^3−d* x1^2* x2^3* x3^2* y1* y3^3+d* x1* x2^2* x3^2* y2* y3^3−d* x1^3* x2^2* x3^2* y2* y3^3
−c^2* x1* y1^2 *y2* y3^3+c* d *x1^3* x2^2* y1^2* y2* y3^3−c* d* x2* x3^2* y1* y2^2* y3^3+d^2* x1^2* x2^3* x3^2* y1* y2^2* y3^3
−c^2* d* x1^2* x2* y1^3* y2^2* y3^3+c^2* d* x2* x3^2* y1^3* y2^2* y3^3+c^2* d* x1* x3^2* y1^2* y2^3* y3^3
−c* d^2* x1* x2^2* x3^2* y1^2* y2^3* y3^3)

have *x1'-expr*: x1' = (x1 * x2 − c * y1 * y2) / (1 − d * x1 * y1 * x2 * y2)
using *assms*(1,3) **by** *auto*
have *y1'-expr*: y1' = (x1 * y2 + y1 * x2) / (1 + d * x1 * y1 * x2 * y2)
using *assms*(1,3) **by** *auto*
have *x3'-expr*: x3' = (x2 * x3 − c * y2 * y3) / (1 − d * x2 * y2 * x3 * y3)
using *assms*(2,4) **by** *auto*
have *y3'-expr*: y3' = (x2 * y3 + y2 * x3) / (1 + d * x2 * y2 * x3 * y3)
using *assms*(2,4) **by** *auto*

have *non-unfolded-adds*:
delta x1 y1 x2 y2 ≠ 0 **using** *delta-def assms*(5,6) **by** *auto*

have *gx-div*: ∃ r1 r2 r3. gxpoly-expr = r1 * e1 + r2 * e2 + r3 * e3
unfolding *gxpoly-expr-def e1-def e2-def e3-def e-def*

by *algebra*

have *gy-div*: $\exists$ *r1 r2 r3. gypoly-expr* = *r1* $*$ *e1* + *r2* $*$ *e2* + *r3* $*$ *e3*
unfolding *gypoly-expr-def e1-def e2-def e3-def e-def*
by *algebra*

have *simp1gx*:
(*x1'* $*$ *x3* − *c* $*$ *y1'* $*$ *y3*) $*$ *delta-minus x1 y1 x3' y3'* $*$
(*delta x1 y1 x2 y2* $*$ *delta x2 y2 x3 y3*) =
((*x1* $*$ *x2* − *c* $*$ *y1* $*$ *y2*) $*$ *x3* $*$ *delta-plus x1 y1 x2 y2* −
c $*$ (*x1* $*$ *y2* + *y1* $*$ *x2*) $*$ *y3* $*$ *delta-minus x1 y1 x2 y2*) $*$
(*delta-minus x2 y2 x3 y3* $*$ *delta-plus x2 y2 x3 y3* −
d $*$ *x1* $*$ *y1* $*$ (*x2* $*$ *x3* − *c* $*$ *y2* $*$ *y3*) $*$
(*x2* $*$ *y3* + *y2* $*$ *x3*))

apply((*subst x1'-expr*)+, (*subst y1'-expr*)+,(*subst x3'-expr*)+,(*subst y3'-expr*)+)
apply((*subst delta-minus-def*[*symmetric*])+,(*subst delta-plus-def*[*symmetric*])+)
apply(*subst* (*3*) *delta-minus-def*)
unfolding *delta-def*
by(*simp add*: *field-simps assms*(*5*−*8*))

have *simp2gx*:
(*x1* $*$ *x3'* − *c* $*$ *y1* $*$ *y3'*) $*$ *delta-minus x1' y1' x3 y3* $*$
(*delta x1 y1 x2 y2* $*$ *delta x2 y2 x3 y3*) =
(*x1* $*$ (*x2* $*$ *x3* − *c* $*$ *y2* $*$ *y3*) $*$ *delta-plus x2 y2 x3 y3* −
c $*$ *y1* $*$ (*x2* $*$ *y3* + *y2* $*$ *x3*) $*$ *delta-minus x2 y2 x3 y3*) $*$
(*delta-minus x1 y1 x2 y2* $*$ *delta-plus x1 y1 x2 y2* −
d $*$ (*x1* $*$ *x2* − *c* $*$ *y1* $*$ *y2*) $*$ (*x1* $*$ *y2* + *y1* $*$ *x2*) $*$ *x3* $*$ *y3*)
apply((*subst x1'-expr*)+, (*subst y1'-expr*)+,(*subst x3'-expr*)+,(*subst y3'-expr*)+)
apply((*subst delta-minus-def*[*symmetric*])+,(*subst delta-plus-def*[*symmetric*])+)
apply(*subst* (*3*) *delta-minus-def*)
unfolding *delta-def*
by(*simp add*: *field-simps assms*(*5*−*8*))

have *gxpoly* = *gxpoly-expr*
unfolding *gxpoly-def* g_x*-def* $Delta_x$*-def*
apply(*simp add*: *assms*(*1*,*2*))
apply(*subst delta-minus-def*[*symmetric*])+
apply(*simp add*: *divide-simps assms*(*9*,*11*))
apply(*subst* (*3*) *left-diff-distrib*)
apply(*simp add*: *simp1gx simp2gx*)
unfolding *delta-minus-def delta-plus-def*
unfolding *gxpoly-expr-def*
by *algebra*

obtain *r1x r2x r3x* **where** *gxpoly* = *r1x* $*$ *e1* + *r2x* $*$ *e2* + *r3x* $*$ *e3*
using ‹*gxpoly* = *gxpoly-expr*› *gx-div* **by** *auto*
then have *gxpoly* = *0*
using *e1-def assms*(*13*−*15*) *e2-def e3-def* **by** *auto*

have $Delta_x \neq 0$
  using $Delta_x$*-def delta-def assms*(*7*−*11*) *non-unfolded-adds* **by** *auto*
then have $g_x = 0$
  using ⟨*gxpoly* = *0*⟩ *gxpoly-def* **by** *auto*

have *simp1gy*: (*x1* ′ ∗ *y3* + *y1* ′ ∗ *x3*) ∗ *delta-plus x1 y1 x3* ′ *y3* ′ ∗
  (*delta x1 y1 x2 y2* ∗ *delta x2 y2 x3 y3*) =
  ((*x1* ∗ *x2* − *c* ∗ *y1* ∗ *y2*) ∗ *y3* ∗ *delta-plus x1 y1 x2 y2* +
   (*x1* ∗ *y2* + *y1* ∗ *x2*) ∗ *x3* ∗ *delta-minus x1 y1 x2 y2*) ∗
  (*delta-minus x2 y2 x3 y3* ∗ *delta-plus x2 y2 x3 y3* +
   d ∗ *x1* ∗ *y1* ∗ (*x2* ∗ *x3* − *c* ∗ *y2* ∗ *y3*) ∗ (*x2* ∗ *y3* + *y2* ∗ *x3*))
 apply((*subst x1* ′*-expr*)+, (*subst y1* ′*-expr*)+,(*subst x3* ′*-expr*)+,(*subst y3* ′*-expr*)+)
 apply((*subst delta-minus-def* [*symmetric*])+,(*subst delta-plus-def* [*symmetric*])+)
  apply(*subst* (*2*) *delta-plus-def*)
  unfolding *delta-def*
  by(*simp add*: *field-simps assms*(*5*−*8*))

have *simp2gy*: (*x1* ∗ *y3* ′ + *y1* ∗ *x3* ′) ∗ *delta-plus x1* ′ *y1* ′ *x3 y3* ∗
  (*delta x1 y1 x2 y2* ∗ *delta x2 y2 x3 y3*) =
   (*x1* ∗ (*x2* ∗ *y3* + *y2* ∗ *x3*) ∗ *delta-minus x2 y2 x3 y3* +
   y1 ∗ (*x2* ∗ *x3* − *c* ∗ *y2* ∗ *y3*) ∗ *delta-plus x2 y2 x3 y3*) ∗
  (*delta-minus x1 y1 x2 y2* ∗ *delta-plus x1 y1 x2 y2* +
   d ∗ (*x1* ∗ *x2* − *c* ∗ *y1* ∗ *y2*) ∗ (*x1* ∗ *y2* + *y1* ∗ *x2*) ∗ *x3* ∗ *y3*)
 apply((*subst x1* ′*-expr*)+, (*subst y1* ′*-expr*)+,(*subst x3* ′*-expr*)+,(*subst y3* ′*-expr*)+)
 apply((*subst delta-minus-def* [*symmetric*])+,(*subst delta-plus-def* [*symmetric*])+)
  apply(*subst* (*3*) *delta-plus-def*)
  unfolding *delta-def*
  by(*simp add*: *field-simps assms*(*5*−*8*))

have *gypoly* = *gypoly-expr*
  unfolding *gypoly-def* g_y*-def* $Delta_y$*-def*
  apply(*simp add*: *assms*(*1*,*2*))
  apply(*subst delta-plus-def* [*symmetric*])+
  apply(*simp add*: *divide-simps assms*(*10*,*12*))
  apply(*subst left-diff-distrib*)
  apply(*simp add*: *simp1gy simp2gy*)
  unfolding *delta-minus-def delta-plus-def*
  unfolding *gypoly-expr-def*
  by *algebra*

obtain *r1y r2y r3y* **where** *gypoly* = *r1y* ∗ *e1* + *r2y* ∗ *e2* + *r3y* ∗ *e3*
  using ⟨*gypoly* = *gypoly-expr*⟩ *gy-div* **by** *auto*
then have *gypoly* = *0*
  using *e1-def assms*(*13*−*15*) *e2-def e3-def* **by** *auto*
have $Delta_y \neq 0$
  using $Delta_y$*-def delta-def assms*(*7*−*12*) *non-unfolded-adds* **by** *auto*
then have $g_y = 0$
  using ⟨*gypoly* = *0*⟩ *gypoly-def* **by** *auto*

```
  show ?thesis
    using ‹g_y = 0› ‹g_x = 0›
    unfolding g_x-def g_y-def assms(3,4)
    by (simp add: prod-eq-iff)
qed

lemma neutral: add z (1,0) = z by(cases z,simp)

lemma inverse:
  assumes e a b = 0 delta-plus a b a b ≠ 0
  shows add (a,b) (a,−b) = (1,0)
  using assms
  apply(simp add: delta-plus-def e-def)
  by algebra

lemma affine-closure:
  assumes delta x1 y1 x2 y2 = 0 e x1 y1 = 0 e x2 y2 = 0
  shows ∃ b. (1/d = bˆ2 ∧ 1/d ≠ 0) ∨ (1/(c∗d) = bˆ2 ∧ 1/(c∗d) ≠ 0)
proof −
  define r where r = (1 − c∗d∗y1ˆ2∗y2ˆ2) ∗ (1 − d∗y1ˆ2∗x2ˆ2)
  define e1 where e1 = e x1 y1
  define e2 where e2 = e x2 y2
  have r = dˆ2 ∗ y1ˆ2 ∗ y2ˆ2 ∗ x2ˆ2 ∗ e1 + (1 − d ∗ y1ˆ2) ∗ delta x1 y1 x2 y2
− d ∗ y1ˆ2 ∗ e2
    unfolding r-def e1-def e2-def delta-def delta-plus-def delta-minus-def e-def
    by algebra
  then have r = 0
    using assms e1-def e2-def by simp
  then have cases: (1 − c∗d∗y1ˆ2∗y2ˆ2) = 0 ∨ (1 − d∗y1ˆ2∗x2ˆ2) = 0
    using r-def by auto
  have d ≠ 0 using ‹r = 0› r-def by auto
  {
    assume (1 − d∗y1ˆ2∗x2ˆ2) = 0
    then have 1/d = y1ˆ2∗x2ˆ2 1/d ≠ 0
      apply(auto simp add: divide-simps ‹d ≠ 0›)
      by algebra
  }
  note case1 = this
  {assume (1 − c∗d∗y1ˆ2∗y2ˆ2) = 0 (1 − d∗y1ˆ2∗x2ˆ2) ≠ 0
    then have c ≠ 0 by auto
    then have 1/(c∗d) = y1ˆ2∗y2ˆ2 1/(c∗d) ≠ 0
      apply(simp add: divide-simps ‹d ≠ 0› ‹c ≠ 0›)
      using ‹(1 − c∗d∗y1ˆ2∗y2ˆ2) = 0› apply algebra
      using ‹c ≠ 0› ‹d ≠ 0› by auto
  }
  note case2 = this

  show ∃ b. (1/d = bˆ2 ∧ 1/d ≠ 0) ∨ (1/(c∗d) = bˆ2 ∧ 1/(c∗d) ≠ 0)
    using cases case1 case2 by (metis power-mult-distrib)
```

qed

lemma *delta-non-zero*:
fixes *x1 y1 x2 y2*
assumes *e x1 y1 = 0 e x2 y2 = 0*
assumes ∃ *b*. *1/c = b^2* ¬ (∃ *b*. *b* ≠ *0* ∧ *1/d = b^2*)
shows *delta x1 y1 x2 y2* ≠ *0*
proof(*rule ccontr*)
assume ¬ *delta x1 y1 x2 y2* ≠ *0*
then have *delta x1 y1 x2 y2 = 0* **by** *blast*
then have ∃ *b*. (*1/d = b^2* ∧ *1/d* ≠ *0*) ∨ (*1/(c∗d) = b^2* ∧ *1/(c∗d)* ≠ *0*)
using *affine-closure*[*OF* ‹*delta x1 y1 x2 y2 = 0*›
‹*e x1 y1 = 0*› ‹*e x2 y2 = 0*›] **by** *blast*
then obtain *b* **where** (*1/(c∗d) = b^2* ∧ *1/(c∗d)* ≠ *0*)
using ‹¬ (∃ *b*. *b* ≠ *0* ∧ *1/d = b^2*)› **by** *fastforce*
then have *1/c* ≠ *0 c* ≠ *0 d* ≠ *0 1/d* ≠ *0* **by** *simp+*
then have *1/d = b^2 / (1/c)*
apply(*simp add*: *divide-simps*)
by (*metis* ‹*1 / (c ∗ d) = b²* ∧ *1 / (c ∗ d)* ≠ *0*› *eq-divide-eq semiring-normalization-rules*(*18*))
then have ∃ *b*. *b* ≠ *0* ∧ *1/d = b^2*
using *assms*(*3*)
by (*metis* ‹*1 / d* ≠ *0*› *power-divide zero-power2*)
then show *False*
using ‹¬ (∃ *b*. *b* ≠ *0* ∧ *1/d = b^2*)› **by** *blast*
qed

lemma *group-law*:
assumes ∃ *b*. *1/c = b^2* ¬ (∃ *b*. *b* ≠ *0* ∧ *1/d = b^2*)
shows *comm-group* (|*carrier* = {(*x*,*y*). *e x y = 0*}, *mult = add*, *one* = (*1*,*0*)|)
proof(*unfold-locales*)
{**fix** *x1 y1 x2 y2*
assume *e x1 y1 = 0 e x2 y2 = 0*
have *e* (*fst* (*add* (*x1*,*y1*) (*x2*,*y2*))) (*snd* (*add* (*x1*,*y1*) (*x2*,*y2*))) = *0*
apply(*simp*)
using *add-closure delta-non-zero*[*OF* ‹*e x1 y1 = 0*› ‹*e x2 y2 = 0*› *assms*(*1*)
assms(*2*)]
delta-def ‹*e x1 y1 = 0*› ‹*e x2 y2 = 0*› **by** *auto*}
then show
⋀*x y*. *x* ∈ *carrier* (|*carrier* = {(*x*, *y*). *local.e x y = 0*}, *mult = local.add*, *one*
= (*1*, *0*)|) ⟹
y ∈ *carrier* (|*carrier* = {(*x*, *y*). *local.e x y = 0*}, *mult = local.add*, *one*
= (*1*, *0*)|) ⟹
x ⊗(|*carrier* = {(*x*, *y*). *local.e x y = 0*}, *mult = local.add*, *one* = (*1*, *0*)|) *y*
∈ *carrier* (|*carrier* = {(*x*, *y*). *local.e x y = 0*}, *mult = local.add*, *one* =
(*1*, *0*)|) **by** *auto*
next
{**fix** *x1 y1 x2 y2 x3 y3*
assume *e x1 y1 = 0 e x2 y2 = 0 e x3 y3 = 0*
then have *delta x1 y1 x2 y2* ≠ *0 delta x2 y2 x3 y3* ≠ *0*

using *assms(1,2) delta-non-zero* **by** *blast+*
fix *x1′ y1′ x3′ y3′*
assume *(x1′,y1′) = add (x1,y1) (x2,y2)*
(x3′,y3′) = add (x2,y2) (x3,y3)
then have *e x1′ y1′ = 0 e x3′ y3′ = 0*
using *add-closure ‹delta x1 y1 x2 y2 ≠ 0› ‹delta x2 y2 x3 y3 ≠ 0›*
‹e x1 y1 = 0› ‹e x2 y2 = 0› ‹e x3 y3 = 0› delta-def **by** *fastforce+*
then have *delta x1′ y1′ x3 y3 ≠ 0 delta x1 y1 x3′ y3′ ≠ 0*
using *assms delta-non-zero ‹e x3 y3 = 0›* **apply** *blast*
by *(simp add: ‹e x1 y1 = 0› ‹e x3′ y3′ = 0› assms delta-non-zero)*

have *add (add (x1,y1) (x2,y2)) (x3,y3) =*
add (x1,y1) (local.add (x2,y2) (x3,y3))
using *associativity*
by *(metis ‹(x1′, y1′) = add (x1, y1) (x2, y2)› ‹(x3′, y3′) = add (x2, y2) (x3, y3)› ‹delta x1 y1 x2 y2 ≠ 0›*
‹delta x1 y1 x3′ y3′ ≠ 0› ‹delta x1′ y1′ x3 y3 ≠ 0› ‹delta x2 y2 x3 y3 ≠ 0› ‹e x1 y1 = 0›
*‹e x2 y2 = 0› ‹e x3 y3 = 0› delta-def mult-eq-0-iff)***}**

then show
⋀x y z.
x ∈ carrier (|carrier = {(x, y). local.e x y = 0}, mult = local.add, one = (1, 0)|) ⟹
y ∈ carrier (|carrier = {(x, y). local.e x y = 0}, mult = local.add, one = (1, 0)|) ⟹
z ∈ carrier (|carrier = {(x, y). local.e x y = 0}, mult = local.add, one = (1, 0)|) ⟹
x ⊗(|carrier = {(x, y). local.e x y = 0}, mult = local.add, one = (1, 0)|)
y ⊗(|carrier = {(x, y). local.e x y = 0}, mult = local.add, one = (1, 0)|)
z =
x ⊗(|carrier = {(x, y). local.e x y = 0}, mult = local.add, one = (1, 0)|)
(y ⊗(|carrier = {(x, y). local.e x y = 0}, mult = local.add, one = (1, 0)|)
z) **by** *auto*
next
show
1(|carrier = {(x, y). e x y = 0}, mult = local.add, one = (1, 0)|)
∈ carrier (|carrier = {(x, y). e x y = 0}, mult = local.add, one = (1, 0)|)
by *(simp add: e-def)*
next
show
⋀x. x ∈ carrier (|carrier = {(x, y). local.e x y = 0}, mult = local.add, one = (1, 0)|) ⟹
1(|carrier = {(x, y). local.e x y = 0}, mult = local.add, one = (1, 0)|) ⊗(|carrier = {(x, y). local.e x y = 0}, n
x = x
by *(simp add: commutativity neutral)*
next
show *⋀x. x ∈ carrier (|carrier = {(x, y). local.e x y = 0}, mult = local.add, one = (1, 0)|) ⟹*

```
          x ⊗(|carrier = {(x, y). local.e x y = 0}, mult = local.add, one = (1, 0)|)
        1(|carrier = {(x, y). local.e x y = 0}, mult = local.add, one = (1, 0)|) = x
    by (simp add: neutral)
next
  show ⋀x y. x ∈ carrier (|carrier = {(x, y). local.e x y = 0}, mult = local.add,
one = (1, 0)|) ⟹
           y ∈ carrier (|carrier = {(x, y). local.e x y = 0}, mult = local.add, one
= (1, 0)|) ⟹
         x ⊗(|carrier = {(x, y). local.e x y = 0}, mult = local.add, one = (1, 0)|) y =
         y ⊗(|carrier = {(x, y). local.e x y = 0}, mult = local.add, one = (1, 0)|) x
    using commutativity by auto
next
  show
   carrier (|carrier = {(x, y). local.e x y = 0}, mult = local.add, one = (1, 0)|)
   ⊆ Units (|carrier = {(x, y). local.e x y = 0}, mult = local.add, one = (1, 0)|)
  proof(simp,standard)
    fix z
    assume z ∈ {(x, y). local.e x y = 0}
    show z ∈ Units
        (|carrier = {(x, y). local.e x y = 0}, mult = local.add,
          one = (1, 0)|)
      unfolding Units-def
    proof(simp, cases z, rule conjI)
      fix x y
      assume z = (x,y)
      from this ‹z ∈ {(x, y). local.e x y = 0}›
      show case z of (x, y) ⇒ local.e x y = 0 by blast
      then obtain x y where z = (x,y) e x y = 0 by blast
      have e x (−y) = 0
        using ‹e x y = 0› unfolding e-def by simp
      have add (x,y) (x,−y) = (1,0)
        using inverse[OF ‹e x y = 0› ] delta-non-zero[OF ‹e x y = 0› ‹e x y = 0›
assms] delta-def by fastforce
      then have add (x,−y) (x,y) = (1,0) by simp
      show ∃ a b. e a b = 0 ∧
                  add (a, b) z = (1, 0) ∧
                  add z (a, b) = (1, 0)
        using ‹add (x, y) (x, − y) = (1, 0)›
           ‹e x (− y) = 0› ‹z = (x, y)› by fastforce
    qed
  qed
qed



end
```

2 Extension

locale *ext-curve-addition* = *curve-addition* +
  fixes t' :: *'a::ell-field*
  assumes *c-eq-1*: $c = 1$
  assumes *t-intro*: $d = t'\hat{}2$
  assumes *t-ineq*: $t'\hat{}2 \neq 1$ $t' \neq 0$
begin

2.1 Change of variables

definition t **where** $t = t'$

lemma *t-nz*: $t \neq 0$ **using** *t-ineq(2) t-def* **by** *auto*

lemma *d-nz*: $d \neq 0$ **using** *t-nz t-ineq t-intro* **by** *simp*

lemma *t-expr*: $t\hat{}2 = d$ $t\hat{}4 = d\hat{}2$ **using** *t-intro t-def* **by** *auto*

lemma *t-sq-n1*: $t\hat{}2 \neq 1$ **using** *t-ineq(1) t-def* **by** *simp*

lemma *t-nm1*: $t \neq -1$ **using** *t-sq-n1* **by** *fastforce*

lemma *d-n1*: $d \neq 1$ **using** *t-sq-n1 t-expr* **by** *blast*

lemma *t-n1*: $t \neq 1$ **using** *t-sq-n1* **by** *fastforce*

lemma *t-dneq2*: $2*t \neq -2$
proof(*rule ccontr*)
  assume $\neg\ 2 * t \neq -\ 2$
  then have $2*t = -2$ **by** *auto*
  then have $t = -1$
    using *two-not-zero mult-cancel-left* **by** *fastforce*
  then show *False*
    using *t-nm1 t-def* **by** *argo*
qed

2.2 New points

definition e' **where** $e'\ x\ y = x\hat{}2 + y\hat{}2 - 1 - t\hat{}2 * x\hat{}2 * y\hat{}2$

definition *e'-aff* = $\{(x,y).\ e'\ x\ y = 0\}$
  definition *e-circ* = $\{(x,y).\ x \neq 0 \wedge y \neq 0 \wedge (x,y) \in$ *e'-aff*$\}$

lemma *e-e'-iff*: $e\ x\ y = 0 \longleftrightarrow e'\ x\ y = 0$
  unfolding *e-def e'-def* **using** *c-eq-1 t-expr(1) t-def* **by** *simp*

lemma *circ-to-aff*: $p \in$ *e-circ* $\Longrightarrow p \in$ *e'-aff*
  unfolding *e-circ-def* **by** *auto*

The case $t\hat{\ }2 = 1$ corresponds to a product of intersecting lines which cannot be a group

lemma *t-2-1-lines*:
$t\hat{\ }2 = 1 \Longrightarrow e'\ x\ y = -\ (1 - x\hat{\ }2) * (1 - y\hat{\ }2)$
unfolding *e′-def* **by** *algebra*

The case $t = 0$ corresponds to a circle which has been treated before

lemma *t-0-circle*:
$t = 0 \Longrightarrow e'\ x\ y = x\hat{\ }2 + y\hat{\ }2 - 1$
unfolding *e′-def* **by** *auto*

2.3 Group transformations and inversions

fun ϱ :: $'a \times 'a \Rightarrow 'a \times 'a$ **where**
$\varrho\ (x,y) = (-y,x)$
fun τ :: $'a \times 'a \Rightarrow 'a \times 'a$ **where**
$\tau\ (x,y) = (1/(t*x),1/(t*y))$

definition G **where**
$G \equiv \{id,\varrho,\varrho \circ \varrho,\varrho \circ \varrho \circ \varrho,\tau,\tau \circ \varrho,\tau \circ \varrho \circ \varrho,\tau \circ \varrho \circ \varrho \circ \varrho\}$

definition *symmetries* **where**
symmetries $= \{\tau,\tau \circ \varrho,\tau \circ \varrho \circ \varrho,\tau \circ \varrho \circ \varrho \circ \varrho\}$

definition *rotations* **where**
rotations $= \{id,\varrho,\varrho \circ \varrho,\varrho \circ \varrho \circ \varrho\}$

lemma *G-partition*: $G =$ *rotations* $\cup$ *symmetries*
unfolding *G-def rotations-def symmetries-def* **by** *fastforce*

lemma *tau-sq*: $(\tau \circ \tau)\ (x,y) = (x,y)$ **by**(*simp add*: *t-nz*)

lemma *tau-idemp*: $\tau \circ \tau = id$
using *t-nz comp-def* **by** *auto*

lemma *tau-idemp-explicit*: $\tau(\tau(x,y)) = (x,y)$
using *tau-idemp pointfree-idE* **by** *fast*

lemma *tau-idemp-point*: $\tau(\tau\ p) = p$
using *o-apply*[*symmetric*, *of* $\tau\ \tau\ p$] *tau-idemp* **by** *simp*

fun i :: $'a \times 'a \Rightarrow 'a \times 'a$ **where**
$i\ (a,b) = (a,-b)$

lemma *i-idemp*: $i \circ i = id$
using *comp-def* **by** *auto*

lemma *i-idemp-explicit*: $i(i(x,y)) = (x,y)$

using *i-idemp pointfree-idE* **by** *fast*

lemma *tau-rot-sym*:
assumes $r \in$ *rotations*
shows $\tau \circ r \in$ *symmetries*
using *assms* **unfolding** *rotations-def symmetries-def* **by** *auto*

lemma *tau-rho-com*:
$\tau \circ \varrho = \varrho \circ \tau$ **by** *auto*

lemma *tau-rot-com*:
assumes $r \in$ *rotations*
shows $\tau \circ r = r \circ \tau$
using *assms* **unfolding** *rotations-def* **by** *fastforce*

lemma *rho-order-4*:
$\varrho \circ \varrho \circ \varrho \circ \varrho = id$ **by** *auto*

lemma *rho-i-com-inverses*:
$i\ (id\ (x,y)) = id\ (i\ (x,y))$
$i\ (\varrho\ (x,y)) = (\varrho \circ \varrho \circ \varrho)\ (i\ (x,y))$
$i\ ((\varrho \circ \varrho)\ (x,y)) = (\varrho \circ \varrho)\ (i\ (x,y))$
$i\ ((\varrho \circ \varrho \circ \varrho)\ (x,y)) = \varrho\ (i\ (x,y))$
by(*simp*)+

lemma *rotations-i-inverse*:
assumes $tr \in$ *rotations*
shows $\exists\ tr' \in rotations.\ (tr \circ i)\ (x,y) = (i \circ tr')\ (x,y) \land tr \circ tr' = id$
using *assms rho-i-com-inverses* **unfolding** *rotations-def* **by** *fastforce*

lemma *tau-i-com-inverses*:
$(i \circ \tau)\ (x,y) = (\tau \circ i)\ (x,y)$
$(i \circ \tau \circ \varrho)\ (x,y) = (\tau \circ \varrho \circ \varrho \circ \varrho \circ i)\ (x,y)$
$(i \circ \tau \circ \varrho \circ \varrho)\ (x,y) = (\tau \circ \varrho \circ \varrho \circ i)\ (x,y)$
$(i \circ \tau \circ \varrho \circ \varrho \circ \varrho)\ (x,y) = (\tau \circ \varrho \circ i)\ (x,y)$
by(*simp*)+

lemma *rho-circ*:
assumes $p \in$ *e-circ*
shows $\varrho\ p \in$ *e-circ*
using *assms* **unfolding** *e-circ-def e′-aff-def e′-def*
by(*simp split*: *prod.splits add*: *add.commute*)

lemma *i-aff*:
assumes $p \in$ *e′-aff*
shows $i\ p \in$ *e′-aff*
using *assms* **unfolding** *e′-aff-def e′-def* **by** *auto*

lemma *i-circ*:

assumes $(x,y) \in$ *e-circ*
shows $i\ (x,y) \in$ *e-circ*
using *assms* **unfolding** *e-circ-def e′-aff-def e′-def* **by** *auto*

lemma *i-circ-points*:
assumes $p \in$ *e-circ*
shows $i\ p \in$ *e-circ*
using *assms* **unfolding** *e-circ-def e′-aff-def e′-def* **by** *auto*

lemma *rot-circ*:
assumes $p \in$ *e-circ* $tr \in$ *rotations*
shows $tr\ p \in$ *e-circ*
proof −
consider (*1*) $tr = id$ | (*2*) $tr = \varrho$ | (*3*) $tr = \varrho \circ \varrho$ | (*4*) $tr = \varrho \circ \varrho \circ \varrho$
using *assms*(*2*) **unfolding** *rotations-def* **by** *blast*
then show *?thesis* **by**(*cases*,*auto simp add*: *assms*(*1*) *rho-circ*)
qed

lemma τ*-circ*:
assumes $p \in$ *e-circ*
shows $\tau\ p \in$ *e-circ*
using *assms* **unfolding** *e-circ-def*
apply(*simp split*: *prod.splits*)
apply(*simp add*: *divide-simps t-nz*)
unfolding *e′-aff-def e′-def*
apply(*simp split*: *prod.splits*)
apply(*simp add*: *divide-simps t-nz*)
apply(*subst power-mult-distrib*)+
apply(*subst ring-distribs*(*1*)[*symmetric*])+
apply(*subst* (*1*) *mult.assoc*)
apply(*subst right-diff-distrib*[*symmetric*])
apply(*simp add*: *t-nz*)
by(*simp add*: *algebra-simps*)

lemma *rot-comp*:
assumes *t1* $\in$ *rotations t2* $\in$ *rotations*
shows *t1* $\circ$ *t2* $\in$ *rotations*
using *assms* **unfolding** *rotations-def* **by** *auto*

lemma *rot-tau-com*:
assumes $tr \in$ *rotations*
shows $tr \circ \tau = \tau \circ tr$
using *assms* **unfolding** *rotations-def* **by**(*auto*)

lemma *tau-i-com*:
$\tau \circ i = i \circ \tau$ **by** *auto*

lemma *rot-com*:
  assumes $r \in$ *rotations* $r' \in$ *rotations*
  shows $r' \circ r = r \circ r'$
  using *assms* **unfolding** *rotations-def* **by** *force*

lemma *rot-inv*:
  assumes $r \in$ *rotations*
  shows $\exists\ r' \in$ *rotations*. $r' \circ r = id$
  using *assms* **unfolding** *rotations-def* **by** *force*

lemma *rot-aff*:
  assumes $r \in$ *rotations* $p \in e'$-*aff*
  shows $r\ p \in e'$-*aff*
  using *assms* **unfolding** *rotations-def* e'-*aff-def* e'-*def*
  by(*auto simp add*: *semiring-normalization-rules*(*16*) *add.commute*)

lemma *rot-delta*:
  assumes $r \in$ *rotations* *delta x1 y1 x2 y2* $\neq$ *0*
  shows *delta* (*fst* (*r* (*x1*,*y1*))) (*snd* (*r* (*x1*,*y1*))) *x2 y2* $\neq$ *0*
  using *assms* **unfolding** *rotations-def delta-def delta-plus-def delta-minus-def*
  apply(*safe*)
  apply *simp*
  apply (*simp add*: *semiring-normalization-rules*(*16*))
  apply(*simp*)
  by(*simp add*: *add-eq-0-iff equation-minus-iff semiring-normalization-rules*(*16*))

lemma *tau-not-id*: $\tau \neq id$
  apply(*simp add*: *fun-eq-iff*)
  apply(*simp add*: *divide-simps t-nz*)
  apply(*simp add*: *field-simps*)
  by (*metis mult.left-neutral t-n1 zero-neq-one*)

lemma *sym-not-id*:
  assumes $r \in$ *rotations*
  shows $\tau \circ r \neq id$
  using *assms* **unfolding** *rotations-def*
  apply(*subst fun-eq-iff*,*simp*)
  apply(*auto*)
  using *tau-not-id* **apply** *auto*[*1*]
    apply (*metis d-nz*)
   apply(*simp add*: *divide-simps t-nz*)
  apply(*simp add*: *field-simps*)
   apply (*metis c-eq-1 mult-numeral-1 numeral-One one-neq-zero*
              power2-minus power-one t-sq-n1)
  by (*metis d-nz*)

lemma *sym-decomp*:
  assumes $g \in$ *symmetries*

shows $\exists\ r \in rotations.\ g = \tau \circ r$
using *assms* **unfolding** *symmetries-def rotations-def* **by** *auto*

lemma *symmetries-i-inverse*:
assumes $tr \in symmetries$
shows $\exists\ tr' \in symmetries.\ (tr \circ i)\ (x,y) = (i \circ tr')\ (x,y) \wedge tr \circ tr' = id$
proof −
consider $(1)\ tr = \tau\ |$
$(2)\ tr = \tau \circ \varrho\ |$
$(3)\ tr = \tau \circ \varrho \circ \varrho\ |$
$(4)\ tr = \tau \circ \varrho \circ \varrho \circ \varrho$
using *assms* **unfolding** *symmetries-def* **by** *blast*
then show *?thesis*
proof(*cases*)
case *1*
define tr' **where** $tr' = \tau$
have $(tr \circ i)\ (x,\ y) = (i \circ tr')\ (x,\ y) \wedge tr \circ tr' = id\ tr' \in symmetries$
using *tr'-def 1 tau-idemp symmetries-def* **by** *simp+*
then show *?thesis* **by** *blast*
next
case *2*
define tr' **where** $tr' = \tau \circ \varrho \circ \varrho \circ \varrho$
have $(tr \circ i)\ (x,\ y) = (i \circ tr')\ (x,\ y) \wedge tr \circ tr' = id\ tr' \in symmetries$
using *tr'-def 2*
apply(*simp*)
apply(*metis* (*no-types*, *hide-lams*) *comp-id fun.map-comp rho-order-4 tau-idemp tau-rho-com*)
using *symmetries-def tr'-def* **by** *simp*
then show *?thesis* **by** *blast*
next
case *3*
define tr' **where** $tr' = \tau \circ \varrho \circ \varrho$
have $(tr \circ i)\ (x,\ y) = (i \circ tr')\ (x,\ y) \wedge tr \circ tr' = id\ tr' \in symmetries$
using *tr'-def 3*
apply(*simp*)
apply(*metis* (*no-types*, *hide-lams*) *comp-id fun.map-comp rho-order-4 tau-idemp tau-rho-com*)
using *symmetries-def tr'-def* **by** *simp*
then show *?thesis* **by** *blast*
next
case *4*
define tr' **where** $tr' = \tau \circ \varrho$
have $(tr \circ i)\ (x,\ y) = (i \circ tr')\ (x,\ y) \wedge tr \circ tr' = id\ tr' \in symmetries$
using *tr'-def 4*
apply(*simp*)
apply(*metis* (*no-types*, *hide-lams*) *comp-id fun.map-comp rho-order-4 tau-idemp tau-rho-com*)
using *symmetries-def tr'-def* **by** *simp*

```
    then show ?thesis by blast
  qed
qed

lemma sym-to-rot: g ∈ symmetries ⟹ τ ∘ g ∈ rotations
  using tau-idemp unfolding symmetries-def rotations-def
  apply(simp)
  apply(elim disjE)
  apply fast
  by(simp add: fun.map-comp)+
```

2.4 Extended addition

```
fun ext-add :: 'a × 'a ⇒ 'a × 'a ⇒ 'a × 'a where
 ext-add (x1,y1) (x2,y2) =
    ((x1*y1−x2*y2) div (x2*y1−x1*y2),
     (x1*y1+x2*y2) div (x1*x2+y1*y2))

definition delta-x :: 'a ⇒ 'a ⇒ 'a ⇒ 'a ⇒ 'a where
  delta-x x1 y1 x2 y2 = x2*y1 − x1*y2
definition delta-y :: 'a ⇒ 'a ⇒ 'a ⇒ 'a ⇒ 'a where
  delta-y x1 y1 x2 y2 = x1*x2 + y1*y2
definition delta' :: 'a ⇒ 'a ⇒ 'a ⇒ 'a ⇒ 'a where
  delta' x1 y1 x2 y2 = delta-x x1 y1 x2 y2 * delta-y x1 y1 x2 y2

lemma delta'-com: (delta' x0 y0 x1 y1 = 0) = (delta' x1 y1 x0 y0 = 0)
  unfolding delta'-def delta-x-def delta-y-def
  by algebra

definition e'-aff-0 where
  e'-aff-0 = {((x1,y1),(x2,y2)). (x1,y1) ∈ e'-aff ∧
                                 (x2,y2) ∈ e'-aff ∧
                                 delta x1 y1 x2 y2 ≠ 0 }

definition e'-aff-1 where
  e'-aff-1 = {((x1,y1),(x2,y2)). (x1,y1) ∈ e'-aff ∧
                                 (x2,y2) ∈ e'-aff ∧
                                 delta' x1 y1 x2 y2 ≠ 0 }

lemma ext-add-comm:
  ext-add (x1,y1) (x2,y2) = ext-add (x2,y2) (x1,y1)
  by(simp add: divide-simps,algebra)

lemma ext-add-comm-points:
  ext-add z1 z2 = ext-add z2 z1
  using ext-add-comm
  apply(subst (1 3 4 6) surjective-pairing)
  by presburger
```

lemma *ext-add-inverse*:
  $x \neq 0 \Longrightarrow y \neq 0 \Longrightarrow$ *ext-add* (x,y) $(i$ $(x,y)) = (1,0)$
  by(*simp add*: *two-not-zero*)

lemma *ext-add-deltas*:
  ext-add $(x1,y1)$ $(x2,y2) =$
    $((delta\text{-}x\ x2\ y1\ x1\ y2)\ div\ (delta\text{-}x\ x1\ y1\ x2\ y2),$
     $(delta\text{-}y\ x1\ x2\ y1\ y2)\ div\ (delta\text{-}y\ x1\ y1\ x2\ y2))$
  unfolding *delta-x-def delta-y-def* **by** *simp*

2.4.1 Inversion and rotation invariance

lemma *inversion-invariance-1*:
  assumes $x1 \neq 0$ $y1 \neq 0$ $x2 \neq 0$ $y2 \neq 0$
  shows *add* $(\tau$ $(x1,y1))$ $(x2,y2) =$ *add* $(x1,y1)$ $(\tau$ $(x2,y2))$
  apply(*simp*)
  apply(*subst c-eq-1*)+
  apply(*simp add*: *algebra-simps*)
  apply(*subst power2-eq-square*[*symmetric*])+
  apply(*subst t-expr*)+
  apply(*rule conjI*)
  apply(*simp-all add*: *divide-simps assms t-nz d-nz*)
  by(*simp-all add*: *algebra-simps*)

lemma *inversion-invariance-2*:
  assumes $x1 \neq 0$ $y1 \neq 0$ $x2 \neq 0$ $y2 \neq 0$
  shows *ext-add* $(\tau$ $(x1,y1))$ $(x2,y2) =$ *ext-add* $(x1,y1)$ $(\tau$ $(x2,y2))$
  apply(*simp add*: *divide-simps t-nz assms*)
  by *algebra*

lemma *rho-invariance-1*:
  add $(\varrho$ $(x1,y1))$ $(x2,y2) = \varrho$ $($*add* $(x1,y1)$ $(x2,y2))$
  apply(*simp*)
  apply(*subst c-eq-1*)+
  by(*simp add*: *algebra-simps divide-simps*)

lemma *rho-invariance-1-points*:
  add $(\varrho$ *p1*$)$ *p2* $= \varrho$ $($*add p1 p2*$)$
  using *rho-invariance-1*
  apply(*subst* (*2 4 6 8*) *surjective-pairing*)
  by *blast*

lemma *rotation-invariance-1*:
  assumes $r \in$ *rotations*
  shows *add* $(r$ $(x1,y1))$ $(x2,y2) = r$ $($*add* $(x1,y1)$ $(x2,y2))$
  using *rho-invariance-1-points assms* **unfolding** *rotations-def*

```
  apply(safe)
  apply(simp,simp)
  by(metis comp-apply prod.exhaust-sel)+

lemma rotation-invariance-1-points:
  assumes r ∈ rotations
  shows add (r p1) p2 = r (add p1 p2)
  using rotation-invariance-1 assms
  unfolding rotations-def
  apply(safe)
  apply(simp,simp)
  using rho-invariance-1-points by auto

lemma rho-invariance-2:
  ext-add (ϱ (x1,y1)) (x2,y2) =
   ϱ (ext-add (x1,y1) (x2,y2))
  by(simp add: algebra-simps divide-simps)

lemma rho-invariance-2-points:
  ext-add (ϱ p1) p2 = ϱ (ext-add p1 p2)
  using rho-invariance-2
  apply(subst (2 4 6 8) surjective-pairing)
  by blast

lemma rotation-invariance-2:
  assumes r ∈ rotations
  shows ext-add (r (x1,y1)) (x2,y2) = r (ext-add (x1,y1) (x2,y2))
  using rho-invariance-2-points assms unfolding rotations-def
  apply(safe)
  apply(simp,simp)
  by(metis comp-apply prod.exhaust-sel)+

lemma rotation-invariance-2-points:
  assumes r ∈ rotations
  shows ext-add (r p1) p2 = r (ext-add p1 p2)
  using rotation-invariance-2 assms
  unfolding rotations-def
  apply(safe)
  apply(simp,simp)
  using rho-invariance-2-points by auto

lemma rotation-invariance-3:
  delta x1 y1 (fst (ϱ (x2,y2))) (snd (ϱ (x2,y2))) =
   delta x1 y1 x2 y2
  by(simp add: delta-def delta-plus-def delta-minus-def,algebra)

lemma rotation-invariance-4:
  delta′ x1 y1 (fst (ϱ (x2,y2))) (snd (ϱ (x2,y2))) = − delta′ x1 y1 x2 y2
  by(simp add: delta′-def delta-x-def delta-y-def,algebra)
```

lemma *inverse-rule-1*:
$(\tau \circ i \circ \tau)$ $(x,y) = i$ (x,y) **by** $(simp\ add: t\text{-}nz)$
lemma *inverse-rule-2*:
$(\varrho \circ i \circ \varrho)$ $(x,y) = i$ (x,y) **by** *simp*
lemma *inverse-rule-3*:
i $(add\ (x1,y1)\ (x2,y2)) = add\ (i\ (x1,y1))\ (i\ (x2,y2))$
by(*simp add*: *divide-simps*)
lemma *inverse-rule-4*:
i $(ext\text{-}add\ (x1,y1)\ (x2,y2)) = ext\text{-}add\ (i\ (x1,y1))\ (i\ (x2,y2))$
by(*simp add*: *algebra-simps divide-simps*)

lemma *e′-aff-x0*:
assumes $x = 0$ $(x,y) \in e'\text{-}aff$
shows $y = 1 \vee y = -1$
using *assms* **unfolding** *e′-aff-def e′-def*
by(*simp*,*algebra*)

lemma *e′-aff-y0*:
assumes $y = 0$ $(x,y) \in e'\text{-}aff$
shows $x = 1 \vee x = -1$
using *assms* **unfolding** *e′-aff-def e′-def*
by(*simp*,*algebra*)

lemma *add-ext-add*:
assumes $x1 \neq 0$ $y1 \neq 0$
shows $ext\text{-}add\ (x1,y1)\ (x2,y2) = \tau\ (add\ (\tau\ (x1,y1))\ (x2,y2))$
apply(*simp*)
apply(*rule conjI*)
apply(*simp add*: *c-eq-1*)
apply(*simp add*: *divide-simps t-nz power2-eq-square*[*symmetric*] *assms t-expr*(*1*) *d-nz*)
apply(*simp add*: *algebra-simps power2-eq-square*[*symmetric*] *t-expr*(*1*))
apply (*simp add*: *semiring-normalization-rules*(*18*) *semiring-normalization-rules*(*29*) *t-intro*)
apply(*simp add*: *divide-simps t-nz power2-eq-square*[*symmetric*] *assms t-expr*(*1*) *d-nz*)
apply(*simp add*: *algebra-simps power2-eq-square*[*symmetric*] *t-expr*(*1*))
by (*simp add*: *power2-eq-square t-intro*)

corollary *add-ext-add-2*:
assumes $x1 \neq 0$ $y1 \neq 0$
shows $add\ (x1,y1)\ (x2,y2) = \tau\ (ext\text{-}add\ (\tau\ (x1,y1))\ (x2,y2))$
proof −
obtain $x1'$ $y1'$ **where** *tau-expr*: $\tau\ (x1,y1) = (x1',y1')$ **by** *simp*
then have *p-nz*: $x1' \neq 0$ $y1' \neq 0$

```
    using assms(1) tau-sq apply auto[1]
    using ‹τ (x1, y1) = (x1', y1')› assms(2) tau-sq by auto
  have add (x1,y1) (x2,y2) = add (τ (x1', y1')) (x2, y2)
    using tau-expr tau-idemp
    by (metis comp-apply id-apply)
  also have ... = τ (ext-add (x1', y1') (x2, y2))
    using add-ext-add[OF p-nz] tau-idemp by simp
  also have ... = τ (ext-add (τ (x1, y1)) (x2, y2))
    using tau-expr tau-idemp by auto
  finally show ?thesis by blast
qed
```

2.4.2 Coherence and closure

```
lemma coherence-1:
  assumes delta-x x1 y1 x2 y2 ≠ 0 delta-minus x1 y1 x2 y2 ≠ 0
  assumes e' x1 y1 = 0 e' x2 y2 = 0
  shows delta-x x1 y1 x2 y2 ∗ delta-minus x1 y1 x2 y2 ∗
        (fst (ext-add (x1,y1) (x2,y2)) − fst (add (x1,y1) (x2,y2)))
        = x2 ∗ y2 ∗ e' x1 y1 − x1 ∗ y1 ∗ e' x2 y2
  apply(simp)
  apply(subst (2) delta-x-def[symmetric])
  apply(subst delta-minus-def[symmetric])
  apply(simp add: c-eq-1 assms(1,2) divide-simps)
  unfolding delta-minus-def delta-x-def e'-def
  apply(subst t-expr)+
  by(simp add: power2-eq-square field-simps)

lemma coherence-2:
  assumes delta-y x1 y1 x2 y2 ≠ 0 delta-plus x1 y1 x2 y2 ≠ 0
  assumes e' x1 y1 = 0 e' x2 y2 = 0
  shows delta-y x1 y1 x2 y2 ∗ delta-plus x1 y1 x2 y2 ∗
        (snd (ext-add (x1,y1) (x2,y2)) − snd (add (x1,y1) (x2,y2)))
        = − x2 ∗ y2 ∗ e' x1 y1 − x1 ∗ y1 ∗ e' x2 y2
  apply(simp)
  apply(subst (2) delta-y-def[symmetric])
  apply(subst delta-plus-def[symmetric])
  apply(simp add: c-eq-1 assms(1,2) divide-simps)
  unfolding delta-plus-def delta-y-def e'-def
  apply(subst t-expr)+
  by(simp add: power2-eq-square  field-simps)

lemma coherence:
  assumes delta x1 y1 x2 y2 ≠ 0 delta' x1 y1 x2 y2 ≠ 0
  assumes e' x1 y1 = 0 e' x2 y2 = 0
  shows ext-add (x1,y1) (x2,y2) = add (x1,y1) (x2,y2)
  using coherence-1 coherence-2 delta-def delta'-def assms by auto

lemma ext-add-closure:
```

```
  assumes delta' x1 y1 x2 y2 ≠ 0
  assumes e' x1 y1 = 0 e' x2 y2 = 0
  assumes (x3,y3) = ext-add (x1,y1) (x2,y2)
  shows e' x3 y3 = 0
proof -
  have deltas-nz: delta-x x1 y1 x2 y2 ≠ 0
                  delta-y x1 y1 x2 y2 ≠ 0
    using assms(1) delta'-def by auto

  define closure1 where closure1 =
    2 - t^2 + t^2 * x1^2 - 2 * x2^2 - t^2 * x1^2 * x2^2 +
    t^2 * x2^4 + t^2 * y1^2 + t^4 * x1^2 * y1^2 -
    t^2 * x2^2 * y1^2 - 2 * y2^2 - t^2 * x1^2 * y2^2 +
    (4 * t^2 - 2 * t^4) * x2^2 * y2^2 - t^2 * y1^2 * y2^2 +
    t^2 * y2^4

  define closure2 where closure2 =
    -2 + t^2 + (2 - 2 * t^2) * x1^2 + t^2 * x1^4 + t^2 * x2^2 -
    t^2 * x1^2 * x2^2 + (2 - 2 * t^2) * y1^2 - t^2 * x2^2 * y1^2 +
    t^2 * y1^4 + t^2 * y2^2 - t^2 * x1^2 * y2^2 + t^4 * x2^2 * y2^2 -
    t^2 * y1^2 * y2^2

  define p where p =
    -1 * t^4 * (x1^2 * x2^4 * y1^2 -x1^4 * x2^2 * y1^2 +
    t^2 * x1^4 * y1^4 - x1^2 * x2^2 * y1^4 + x1^4 * x2^2 * y2^2 -
    x1^2 * x2^4 * y2^2 - x1^4 * y1^2 * y2^2 + 4 * x1^2 * x2^2 * y1^2 * y2^2
-
    2 * t^2 * x1^2 * x2^2 * y1^2 * y2^2 - x2^4 * y1^2 * y2^2 - x1^2 * y1^4
* y2^2 +
    x2^2 * y1^4 * y2^2 - x1^2 * x2^2 * y2^4 + t^2 * x2^4 * y2^4 + x1^2 *
y1^2 * y2^4 -
    x2^2 * y1^2 * y2^4)

  have v3: x3 = fst (ext-add (x1,y1) (x2,y2))
           y3 = snd (ext-add (x1,y1) (x2,y2))
    using assms(4) by simp+

  have t^4 * (delta-x x1 y1 x2 y2)^2 * (delta-y x1 y1 x2 y2)^2 * e' x3 y3 = p
    unfolding e'-def v3
    apply(simp)
    apply(subst (2) delta-x-def[symmetric])+
    apply(subst (2) delta-y-def[symmetric])+
    apply(subst power-divide)+
    apply(simp add: divide-simps deltas-nz)
    unfolding p-def delta-x-def delta-y-def
    by algebra
  also have ... = closure1 * e' x1 y1 +  closure2 * e' x2 y2
    unfolding p-def e'-def closure1-def closure2-def by algebra
  finally have t^4 * (delta-x x1 y1 x2 y2)^2 * (delta-y x1 y1 x2 y2)^2 * e' x3 y3
```

=
closure1 ∗ *e′ x1 y1* + *closure2* ∗ *e′ x2 y2*
by *blast*

then show *e′ x3 y3* = *0*
using *assms*(*2*,*3*) *deltas-nz t-nz* **by** *auto*
qed

lemma *ext-add-closure-points*:
assumes *delta′ x1 y1 x2 y2* ≠ *0*
assumes (*x1*,*y1*) ∈ *e′-aff* (*x2*,*y2*) ∈ *e′-aff*
shows *ext-add* (*x1*,*y1*) (*x2*,*y2*) ∈ *e′-aff*
using *ext-add-closure assms*
unfolding *e′-aff-def* **by** *auto*

2.4.3 Useful lemmas in the extension

lemma *inverse-generalized*:
assumes (*a*,*b*) ∈ *e′-aff delta-plus a b a b* ≠ *0*
shows *add* (*a*,*b*) (*a*,−*b*) = (*1*,*0*)
using *inverse assms*
unfolding *e′-aff-def*
using *e-e′-iff*
by(*simp*)

lemma *add-closure-points*:
assumes *delta x y x′ y′* ≠ *0*
(*x*,*y*) ∈ *e′-aff* (*x′*,*y′*) ∈ *e′-aff*
shows *add* (*x*,*y*) (*x′*,*y′*) ∈ *e′-aff*
using *add-closure assms e-e′-iff*
unfolding *delta-def e′-aff-def* **by** *auto*

end

3 Projective Edwards curves

locale *projective-curve* =
ext-curve-addition
begin

3.1 No fixed-point lemma and dichotomies

lemma *g-no-fp*:
assumes *g* ∈ *G p* ∈ *e-circ g p* = *p*
shows *g* = *id*
proof −
obtain *x y* **where** *p-def*: *p* = (*x*,*y*) **by** *fastforce*
have *nz*: *x* ≠ *0 y* ≠ *0* **using** *assms p-def* **unfolding** *e-circ-def* **by** *auto*

consider (*id*) $g = id$ | (*rot*) $g \in rotations$ $g \neq id$ | (*sym*) $g \in symmetries$ $g \neq id$
  using *G-partition assms* **by** *blast*
then show *?thesis*
proof(*cases*)
  case *id* **then show** *?thesis* **by** *simp*
next
  case *rot*
  then have $x = 0$
    using *assms*(*3*) *two-not-zero*
    unfolding *rotations-def p-def*
    by *auto*
  then have *False*
    using *nz* **by** *blast*
  then show *?thesis* **by** *blast*
next
  case *sym*
  then have $t*x*y = 0 \vee (t*x\hat{}2 \in \{-1,1\} \wedge t*y\hat{}2 \in \{-1,1\} \wedge t*x\hat{}2 = t*y\hat{}2)$
    using *assms*(*3*) *two-not-zero*
    unfolding *symmetries-def p-def power2-eq-square*
    apply(*safe*)
    apply(*auto simp add*: *algebra-simps divide-simps two-not-zero*)
    using *two-not-zero* **by** *metis+*
  then have $e'\ x\ y = 2 * (1 - t) / t \vee e'\ x\ y = 2 * (-1 - t) / t$
    using *nz t-nz* **unfolding** *e′-def*
    by(*simp add*: *algebra-simps divide-simps*,*algebra*)
  then have $e'\ x\ y \neq 0$
    using *t-dneq2 t-n1*
    by(*auto simp add*: *algebra-simps divide-simps t-nz*)
  then have *False*
    using *assms nz p-def* **unfolding** *e-circ-def e′-aff-def* **by** *fastforce*
  then show *?thesis* **by** *simp*
qed
qed

lemma *dichotomy-1*:
  assumes $p \in e'$*-aff* $q \in e'$*-aff*
  shows ($p \in$ *e-circ* $\wedge$ ($\exists$ $g \in symmetries.$ $q = (g \circ i)$ p)) $\vee$
    $(p,q) \in e'$*-aff-0* $\vee$ $(p,q) \in e'$*-aff-1*
proof −
  obtain *x1 y1* **where** *p-def*: $p = (x1,y1)$ **by** *fastforce*
  obtain *x2 y2* **where** *q-def*: $q = (x2,y2)$ **by** *fastforce*

  consider (*1*) $(p,q) \in e'$*-aff-0* |
      (*2*) $(p,q) \in e'$*-aff-1* |
      (*3*) $(p,q) \notin e'$*-aff-0* $\wedge$ $(p,q) \notin e'$*-aff-1* **by** *blast*
  then show *?thesis*
  proof(*cases*)
    case *1* **then show** *?thesis* **by** *blast*

```
next
  case 2 then show ?thesis by simp
next
  case 3
  then have delta x1 y1 x2 y2 = 0 delta' x1 y1 x2 y2 = 0
    unfolding p-def q-def e'-aff-0-def e'-aff-1-def using assms
    by (simp add: assms p-def q-def)+
  have x1 ≠ 0 y1 ≠ 0 x2 ≠ 0 y2 ≠ 0
    using ‹delta x1 y1 x2 y2 = 0›
    unfolding delta-def delta-plus-def delta-minus-def by auto
  then have p ∈ e-circ q ∈ e-circ
    unfolding e-circ-def using assms p-def q-def by blast+

  obtain a0 b0 where tq-expr: τ q = (a0,b0) by fastforce
  then have q-expr: q = τ (a0,b0) using tau-idemp-explicit q-def by auto
  obtain a1 b1 where p-expr: p = (a1,b1) by fastforce
  have a0-nz: a0 ≠ 0 b0 ≠ 0
    using ‹τ q = (a0, b0)› ‹x2 ≠ 0› ‹y2 ≠ 0› comp-apply q-def tau-sq by auto

  have a1-nz: a1 ≠ 0 b1 ≠ 0
    using ‹p = (a1, b1)› ‹x1 ≠ 0› ‹y1 ≠ 0› p-def by auto

  have in-aff: (a0,b0) ∈ e'-aff (a1,b1) ∈ e'-aff
    using ‹q ∈ e-circ› τ-circ circ-to-aff tq-expr apply fastforce
    using assms(1) p-expr by auto

  define δ' :: 'a ⇒ 'a ⇒ 'a where
    δ'= (λ x0 y0. x0 * y0 * delta-minus a1 b1 (1/(t*x0)) (1/(t*y0)))
  define pδ' :: 'a ⇒ 'a ⇒ 'a where
    pδ'= (λ x0 y0. x0 * y0 * delta-plus a1 b1 (1/(t*x0)) (1/(t*y0)))
  define δ-plus :: 'a ⇒ 'a ⇒ 'a where
    δ-plus = (λ x0 y0. t * x0 * y0 * delta-x a1 b1 (1/(t*x0)) (1/(t*y0)))
  define δ-minus :: 'a ⇒ 'a ⇒ 'a where
    δ-minus = (λ x0 y0. t * x0 * y0 * delta-y a1 b1 (1/(t*x0)) (1/(t*y0)))

  have (∃ g ∈ symmetries. q = (g ∘ i) p)
  proof(cases delta-minus a1 b1 (fst q) (snd q) = 0)
    case True
    then have t1: delta-minus a1 b1 (fst q) (snd q) = 0
      using ‹delta x1 y1 x2 y2 = 0› ‹p = (a1, b1)› delta-def p-def q-def by auto
    then show ?thesis
    proof(cases δ-plus a0 b0 = 0)
      case True
      then have cas1: delta-minus a1 b1 (fst q) (snd q) = 0
                      δ-plus a0 b0 = 0
        using t1 by auto
      have δ'-expr: δ' a0 b0 = a0*b0 − a1*b1
       unfolding δ'-def delta-minus-def
      by(simp add: algebra-simps a0-nz a1-nz power2-eq-square[symmetric] t-expr
```

```
d-nz)

        have eq1': a0*b0 − a1*b1 = 0
          using δ'-expr q-def tau-sq tq-expr cas1(1) unfolding δ'-def by fastforce
        then have eq1: a0 = a1 * (b1 / b0)
          using a0-nz(2) by(simp add: divide-simps)

        have eq2: b0^2 − a1^2 = 0
          using cas1(2) unfolding δ-plus-def delta-x-def
       by(simp add: divide-simps a0-nz a1-nz t-nz eq1 power2-eq-square[symmetric])

        have eq3: a0^2 − b1^2 = 0
          using eq1 eq2
       by(simp add: divide-simps a0-nz a1-nz eq1 eq2 power2-eq-square right-diff-distrib')

        have (a0,b0) = (b1,a1) ∨ (a0,b0) = (−b1,−a1)
          using eq2 eq3 eq1' by algebra
        then have (a0,b0) ∈ {(b1,a1),(−b1,−a1)} by simp
        moreover have {(b1,a1),(−b1,−a1)} ⊆ {i p, (ϱ ∘ i) p, (ϱ ∘ ϱ ∘ i) p, (ϱ ∘
ϱ ∘ ϱ ∘ i) p}
          using ‹p = (a1, b1)› by auto
        ultimately have ∃ g ∈ rotations. τ q = (g ∘ i) p
          unfolding rotations-def by (auto simp add: ‹τ q = (a0, b0)›)

        then obtain g where g ∈ rotations τ q = (g ∘ i) p by blast
        then have q = (τ ∘ g ∘ i) p
          using tau-sq ‹τ q = (a0, b0)› q-def by auto
        then show ?thesis
          using tau-rot-sym ‹g ∈ rotations› symmetries-def by blast
    next
      case False

        then have cas2: delta-minus a1 b1 (fst q) (snd q) = 0
                        δ-minus a0 b0 = 0
          using t1 δ-minus-def δ-plus-def ‹delta' x1 y1 x2 y2 = 0› ‹p = (a1, b1)›
                delta'-def 3 q-def p-def tq-expr by auto

        have δ'-expr: δ' a0 b0 = a0*b0 − a1*b1
          unfolding δ'-def delta-minus-def
         by(simp add: algebra-simps a0-nz a1-nz power2-eq-square[symmetric] t-expr
d-nz)

        have eq1: a1 * b0 + b1 * a0 = 0
          using cas2(2) unfolding δ-minus-def delta-y-def
          by(simp add: divide-simps a0-nz a1-nz t-nz power2-eq-square[symmetric])

        have eq2: a0*b0 − a1*b1 = 0
          using δ'-expr q-def tau-sq tq-expr cas2(1) unfolding δ'-def by fastforce
```

```
      define c1 where c1 = a0^2*b0^2
      define c2 where c2 = a1^2*b0^2
      have c-eqs: c1 = a0^2*b0^2 c2 = a1^2*b0^2
                c1 = a1^2*b1^2 c2 = a0^2*b1^2
        using c1-def c2-def eq1 eq2 by algebra+

      have c1 * (a0^2 + b0^2 − 1) = c1 * (a1^2 + b1^2 − 1)
        using in-aff c-eqs
        unfolding e'-aff-def e'-def
        by(simp add: a1-nz a0-nz)
      then have eq3: (c1−c2) * (a0^2−b1^2) = 0
                  (c1−c2) * (a1^2−b0^2) = 0
        apply(simp-all add: algebra-simps)
        unfolding c1-def c2-def
        using c-eqs by algebra+
      then consider
        (1) c1 = c2 |
        (2) a0^2 − b1^2 = 0 a1^2−b0^2 = 0 by force
      then have (a0,b0) ∈ {(b1,a1),(−b1,−a1)}
      proof(cases)
        case 1
        then have b0^2 = b1^2 a0^2 = a1^2
          using c-eqs a0-nz a1-nz by auto
        then have b0 = b1 ∨ b0 = −b1 a0 = a1 ∨ a0 = −a1
          by algebra+
        then show ?thesis
          using eq2 eq1 a0-nz(1) a1-nz(2) nonzero-mult-div-cancel-left
                two-not-zero by force
      next
        case 2
        then show ?thesis
          using eq2 by algebra
      qed
      then  have (a0,b0) ∈ {i p, (ρ ∘ i) p, (ρ ∘ ρ ∘ i) p, (ρ ∘ ρ ∘ ρ ∘ i) p}
        using p-expr by auto
      then have (∃ g ∈ rotations. τ q = (g ∘ i) p)
        unfolding rotations-def by (simp add: ‹τ q = (a0, b0)›)
      then obtain g where g ∈ rotations ∧ τ q = (g ∘ i) p
        by blast
      then have q = (τ ∘ g ∘ i) p
        using tau-sq ‹τ q = (a0, b0)› q-def by auto
      then show ?thesis
        unfolding symmetries-def rotations-def
         using tau-rot-sym ‹g ∈ rotations ∧ τ q = (g ∘ i) p› symmetries-def by
blast
    qed
    next
      case False
      then have t1: delta-plus a1 b1 (fst q) (snd q) = 0
```

```
      using ‹delta x1 y1 x2 y2 = 0› ‹p = (a1, b1)› delta-def p-def q-def by auto
    then show ?thesis
    proof(cases δ-minus a0 b0 = 0)
      case True
      then have cas1: delta-plus a1 b1 (fst q) (snd q) = 0
                δ-minus a0 b0 = 0 using t1 by auto
      have δ'-expr: pδ' a0 b0 = a0 ∗ b0 + a1 ∗ b1
        unfolding pδ'-def delta-plus-def
        by(simp add: algebra-simps a0-nz a1-nz power2-eq-square[symmetric] t-expr d-nz)

      have eq1': a0 ∗ b0 + a1 ∗ b1 = 0
        using δ'-expr cas1(1) pδ'-def q-def tau-sq tq-expr by auto
      then have eq1: a0 = − (a1 ∗ b1) / b0
        using a0-nz(2)
        by(simp add: divide-simps,algebra)
      have eq2: b0^2 − b1^2 = 0
        using cas1(2) unfolding δ-minus-def delta-y-def
       by(simp add: divide-simps t-nz a0-nz a1-nz eq1 power2-eq-square[symmetric])
      have eq3: a0^2 − a1^2 = 0
        using eq2 eq1'
        by(simp add: algebra-simps divide-simps a0-nz a1-nz eq1 power2-eq-square)

      from eq2 have pos1: b0 = b1 ∨ b0 = −b1 by algebra
      from eq3 have pos2: a0 = a1 ∨ a0 = −a1 by algebra
      have (a0 = a1 ∧ b0 = −b1) ∨ (a0 = −a1 ∧ b0 = b1)
        using pos1 pos2 eq2 eq3 eq1' by fastforce
      then have (a0,b0) = (a1,−b1) ∨ (a0,b0) = (−a1,b1) by auto
      then have (a0,b0) ∈ {(a1,−b1),(−a1,b1)} by simp
      moreover have {(a1,−b1),(−a1,b1)} ⊆ {i p, (ϱ ∘ i) p, (ϱ ∘ ϱ ∘ i) p, (ϱ ∘ ϱ ∘ ϱ ∘ i) p}
        using ‹p = (a1, b1)› p-def by auto
      ultimately have (a0,b0) ∈ {i p, (ϱ ∘ i) p, (ϱ ∘ ϱ ∘ i) p, (ϱ ∘ ϱ ∘ ϱ ∘ i) p}
        by blast

      then have (∃ g ∈ rotations. τ q = (g ∘ i) p)
        unfolding rotations-def by (simp add: ‹τ q = (a0, b0)›)
      then obtain g where g ∈ rotations ∧ τ q = (g ∘ i) p
        by blast
      then have q = (τ ∘ g ∘ i) p
        using tau-sq ‹τ q = (a0, b0)› q-def by auto
      then show (∃ g ∈ symmetries. q = (g ∘ i) p)
        unfolding symmetries-def rotations-def
         using tau-rot-sym ‹g ∈ rotations ∧ τ q = (g ∘ i) p› symmetries-def by blast
    next
    case False
      then have cas2: delta-plus a1 b1 (fst q) (snd q) = 0
                 δ-plus a0 b0 = 0
```

```
      using t1 False δ-minus-def δ-plus-def ‹delta' x1 y1 x2 y2 = 0› ‹p = (a1,
b1)›
          delta'-def p-def q-def tq-expr by auto
    have δ'-expr: pδ' a0 b0 = a0*b0 + a1*b1
      unfolding pδ'-def delta-plus-def
      by(simp add: algebra-simps a0-nz a1-nz power2-eq-square[symmetric] t-expr
d-nz)
    then have eq1: a0*b0 + a1*b1 = 0
      using pδ'-def δ'-expr tq-expr q-def tau-sq cas2(1) by force

    have eq2: b0 * b1 − a0 * a1 = 0
      using cas2 unfolding δ-plus-def delta-x-def
      by(simp add: algebra-simps t-nz a0-nz)

    define c1 where c1 = a0^2*b0^2
    define c2 where c2 = a0^2*a1^2
    have c-eqs: c1 = a0^2*b0^2 c2 = a0^2*a1^2
              c1 = a1^2*b1^2 c2 = b0^2*b1^2
      using c1-def c2-def eq1 eq2 by algebra+

    have c1 * (a0^2 + b0^2 − 1) = c1 * (a1^2 + b1^2 − 1)
      using in-aff c-eqs
      unfolding e'-aff-def e'-def
      by(simp add: a1-nz a0-nz)
    then have eq3: (c1−c2) * (a0^2−a1^2) = 0
                 (c1−c2) * (b0^2−b1^2) = 0
      apply(simp-all add: algebra-simps)
      unfolding c1-def c2-def
      using c-eqs by algebra+

    then consider
      (1) c1 = c2 |
      (2) a0^2−a1^2 = 0 b0^2−b1^2 = 0 by force
    then have (a0,b0) ∈ {(b1,a1),(−b1,−a1)}
    proof(cases)
      case 1
      then have b0^2 = a1^2 a0^2 = b1^2
        using c-eqs a0-nz a1-nz by auto
      then show ?thesis
        using eq2 by algebra
    next
      case 2
      then have b0 = b1 ∨ b0 = −b1 a0 = a1 ∨ a0 = −a1
        by algebra+
      then show ?thesis
        using eq1 eq2 False δ-minus-def a1-nz(1) a1-nz(2) delta-y-def by auto
    qed
    then have (a0,b0) ∈ {i p, (ϱ ∘ i) p, (ϱ ∘ ϱ ∘ i) p, (ϱ ∘ ϱ ∘ ϱ ∘ i) p}
      unfolding p-expr by auto
```

then have $(\exists\ g \in rotations.\ \tau\ q = (g \circ i)\ p)$
unfolding *rotations-def* **by** $(simp\ add: \langle\tau\ q = (a0,\ b0)\rangle)$
then obtain g **where** $g \in rotations \wedge \tau\ q = (g \circ i)\ p$
by *blast*
then have $q = (\tau \circ g \circ i)\ p$
using *tau-sq* $\langle\tau\ q = (a0,\ b0)\rangle$ *q-def* **by** *auto*
then show $(\exists\ g \in symmetries.\ q = (g \circ i)\ p)$
unfolding *symmetries-def rotations-def*
using *tau-rot-sym* $\langle g \in rotations \wedge \tau\ q = (g \circ i)\ p\rangle$ *symmetries-def* **by**
blast
qed
qed
then show *?thesis*
using $\langle p \in e\text{-}circ\rangle$ **by** *blast*
qed
qed

lemma *dichotomy-2*:
assumes $add\ (x1,y1)\ (x2,y2) = (1,0)$
$((x1,y1),(x2,y2)) \in e'\text{-}aff\text{-}0$
shows $(x2,y2) = i\ (x1,y1)$
proof −
have 1: $x1 = x2$
using $assms(1,2)$ **unfolding** *e′-aff-0-def e′-aff-def delta-def delta-plus-def delta-minus-def e′-def*
apply(*simp*)
apply(*simp add*: *c-eq-1 t-expr*)
by *algebra*

have 2: $y1 = -\ y2$
using $assms(1,2)$ **unfolding** *e′-aff-0-def e′-aff-def delta-def delta-plus-def delta-minus-def e′-def*
apply(*simp*)
apply(*simp add*: *c-eq-1 t-expr*)
by *algebra*

from *1 2* **show** *?thesis* **by** *simp*
qed

lemma *dichotomy-3*:
assumes $ext\text{-}add\ (x1,y1)\ (x2,y2) = (1,0)$
$((x1,y1),(x2,y2)) \in e'\text{-}aff\text{-}1$
shows $(x2,y2) = i\ (x1,y1)$
proof −
have *nz*: $x1 \neq 0\ y1 \neq 0\ x2 \neq 0\ y2 \neq 0$
using *assms* **by**(*simp*,*force*)+
have *in-aff*: $(x1,y1) \in e'\text{-}aff\ \ (x2,y2) \in e'\text{-}aff$

using *assms* **unfolding** *e′-aff-1-def* **by** *auto*
have *ds*: *delta′ x1 y1 x2 y2* $\neq$ *0*
using *assms* **unfolding** *e′-aff-1-def* **by** *auto*

have *eqs*: *x1*∗(*y1*+*y2*) = *x2*∗(*y1*+*y2*) *x1* ∗ *y1* + *x2* ∗ *y2* = *0*
using *assms in-aff ds*
unfolding *e′-aff-def e′-def delta′-def delta-x-def delta-y-def*
apply *simp-all*
by *algebra*

then consider (*1*) *y1* + *y2* = *0* | (*2*) *x1* = *x2* **by** *auto*
then have *1*: *x1* = *x2*
proof(*cases*)
case *1*
then show *?thesis*
using *eqs nz* **by** *algebra*
next
case *2*
then show *?thesis* **by** *auto*
qed

have *2*: *y1* = − *y2*
using *eqs 1 nz*
by *algebra*

from *1 2* **show** *?thesis* **by** *simp*
qed

3.1.1 Meaning of dichotomy condition on deltas

lemma *wd-d-nz*:
assumes *g* ∈ *symmetries* (*x′*, *y′*) = (*g* ∘ *i*) (*x*, *y*) (*x*,*y*) ∈ *e-circ*
shows *delta x y x′ y′* = *0*
using *assms* **unfolding** *symmetries-def e-circ-def delta-def delta-minus-def delta-plus-def*
by(*auto*,*auto simp add*: *divide-simps t-nz t-expr*(*1*) *power2-eq-square*[*symmetric*]
d-nz)

lemma *wd-d′-nz*:
assumes *g* ∈ *symmetries* (*x′*, *y′*) = (*g* ∘ *i*) (*x*, *y*) (*x*,*y*) ∈ *e-circ*
shows *delta′ x y x′ y′* = *0*
using *assms* **unfolding** *symmetries-def e-circ-def delta′-def delta-x-def delta-y-def*
by *auto*

lemma *meaning-of-dichotomy-1*:
assumes (∃ *g*∈*symmetries*. (*x2*, *y2*) = (*g* ∘ *i*) (*x1*, *y1*))
shows *fst* (*add* (*x1*,*y1*) (*x2*,*y2*)) = *0* ∨ *snd* (*add* (*x1*,*y1*) (*x2*,*y2*)) = *0*
using *assms*
apply(*simp*)
apply(*simp add*: *c-eq-1*)

unfolding *symmetries-def*
apply(*safe*)
apply(*simp-all*)
apply(*simp-all split*: *if-splits add*: *t-nz divide-simps*)
by(*simp-all add*: *algebra-simps t-nz divide-simps power2-eq-square*[*symmetric*] *t-expr*)

lemma *meaning-of-dichotomy-2*:
assumes ($\exists\, g \in symmetries.\ (x2,\ y2) = (g \circ i)\ (x1,\ y1)$)
shows $fst\ (ext\text{-}add\ (x1,y1)\ (x2,y2)) = 0 \lor snd\ (ext\text{-}add\ (x1,y1)\ (x2,y2)) = 0$
using *assms*
apply(*simp*)
unfolding *symmetries-def*
apply(*safe*)
apply(*simp-all*)
by(*simp-all split*: *if-splits add*: *t-nz divide-simps*)

3.2 Gluing relation and projective points

definition *gluing* :: $((('a \times 'a) \times bit) \times (('a \times 'a) \times bit))$ *set* **where**
$gluing = \{(((x0,y0),l),((x1,y1),j)).$
$((x0,y0) \in e'\text{-}aff \land (x1,y1) \in e'\text{-}aff) \land$
$(((x0,y0) \in e\text{-}circ \land (x1,y1) = \tau\ (x0,y0) \land j = l+1) \lor$
$((x0,y0) \in e'\text{-}aff \land x0 = x1 \land y0 = y1 \land l = j))\}$

lemma *gluing-char*:
assumes $(((x0,y0),l),((x1,y1),j)) \in gluing$
shows $((x0,y0) = (x1,y1) \land l = j) \lor ((x1,y1) = \tau\ (x0,y0) \land l = j+1)$
using *assms gluing-def* **by** *force+*

lemma *gluing-char-zero*:
assumes $(((x0,y0),l),((x1,y1),j)) \in gluing\ x0 = 0 \lor y0 = 0$
shows $(x0,y0) = (x1,y1) \land l = j$
using *assms* **unfolding** *gluing-def e-circ-def* **by** *force*

lemma *gluing-aff*:
assumes $(((x0,y0),l),((x1,y1),j)) \in gluing$
shows $(x0,y0) \in e'\text{-}aff\ (x1,y1) \in e'\text{-}aff$
using *assms* **unfolding** *gluing-def* **by** *force+*

definition *e'-aff-bit* :: $(('a \times 'a) \times bit)$ *set* **where**
$e'\text{-}aff\text{-}bit = e'\text{-}aff \times UNIV$

lemma *eq-rel*: *equiv e'-aff-bit gluing*
unfolding *equiv-def*
proof(*safe*)
show *refl-on e'-aff-bit gluing*

unfolding *refl-on-def e'-aff-bit-def gluing-def* **by** *auto*
show *sym gluing*
unfolding *sym-def gluing-def* **by**(*auto simp add*: *e-circ-def t-nz*)
show *trans gluing*
unfolding *trans-def gluing-def* **by**(*auto simp add*: *e-circ-def t-nz*)
qed

definition *e-proj* **where** *e-proj* = *e'-aff-bit* // *gluing*

3.2.1 Point-class classification

lemma *eq-class-simp*:
assumes $X \in$ *e-proj* $X \neq \{\}$
shows X // *gluing* = $\{X\}$
proof −
have *simp-un*: *gluing* `` $\{x\} = X$ **if** $x \in X$ **for** x
apply(*rule quotientE*)
using *e-proj-def assms*(*1*) **apply** *blast*
using *equiv-class-eq*[*OF eq-rel*] *that* **by** *auto*

show X // *gluing* = $\{X\}$
unfolding *quotient-def* **by**(*simp add*: *simp-un assms*)
qed

lemma *gluing-class-1*:
assumes $x = 0 \vee y = 0$ $(x,y) \in$ *e'-aff*
shows *gluing* `` $\{((x,y), l)\} = \{((x,y), l)\}$
proof −
have $(x,y) \notin$ *e-circ* **using** *assms* **unfolding** *e-circ-def* **by** *blast*
then show *?thesis*
using *assms* **unfolding** *gluing-def Image-def*
by(*simp split*: *prod.splits del*: τ*.simps add*: *assms*,*safe*)
qed

lemma *gluing-class-2*:
assumes $x \neq 0$ $y \neq 0$ $(x,y) \in$ *e'-aff*
shows *gluing* `` $\{((x,y), l)\} = \{((x,y), l), (\tau\ (x,y), l + 1)\}$
proof −
have $(x,y) \in$ *e-circ* **using** *assms* **unfolding** *e-circ-def* **by** *blast*
then have $\tau\ (x,y) \in$ *e'-aff*
using τ*-circ* **using** *e-circ-def* **by** *force*
show *?thesis*
using *assms* **unfolding** *gluing-def Image-def*
apply(*simp add*: *e-circ-def assms del*: τ*.simps*,*safe*)
using ‹$\tau\ (x,y) \in$ *e'-aff*› **by** *argo*
qed

lemma *e-proj-elim-1*:
assumes $(x,y) \in$ *e'-aff*

```
  shows {((x,y),l)} ∈ e-proj ⟷ x = 0 ∨ y = 0
proof
  assume as: {((x, y), l)} ∈ e-proj
  have eq: gluing `` {((x, y), l)} = {((x,y),l)}
    (is - = ?B)
   using quotientI[of - ?B gluing] eq-class-simp as by auto
  then show x = 0 ∨ y = 0
    using assms gluing-class-2 by force
next
  assume x = 0 ∨ y = 0
  then have eq: gluing `` {((x, y), l)} = {((x,y),l)}
    using assms gluing-class-1 by presburger
  show {((x,y),l)} ∈ e-proj
    apply(subst eq[symmetric])
    unfolding e-proj-def apply(rule quotientI)
    unfolding e'-aff-bit-def using assms by simp
qed

lemma e-proj-elim-2:
  assumes (x,y) ∈ e'-aff
  shows {((x,y),l),(τ (x,y),l+1)} ∈ e-proj ⟷ x ≠ 0 ∧ y ≠ 0
proof
  assume x ≠ 0 ∧ y ≠ 0
  then have eq: gluing `` {((x, y), l)} = {((x,y),l),(τ (x,y),l+1)}
    using assms gluing-class-2 by presburger
  show {((x,y),l),(τ (x,y),l+1)} ∈ e-proj
    apply(subst eq[symmetric])
    unfolding e-proj-def apply(rule quotientI)
    unfolding e'-aff-bit-def using assms by simp
next
  assume as: {((x, y), l), (τ (x, y), l + 1)} ∈ e-proj
  have eq: gluing `` {((x, y), l)} = {((x,y),l),(τ (x,y),l+1)}
    (is - = ?B)
   using quotientI[of - ?B gluing] eq-class-simp as by auto
  then show x ≠ 0 ∧ y ≠ 0
    using assms gluing-class-1 by auto
qed

lemma e-proj-eq:
  assumes p ∈ e-proj
  shows ∃ x y l. (p = {((x,y),l)} ∨ p = {((x,y),l),(τ (x,y),l+1)}) ∧ (x,y) ∈ e'-aff

proof -
  obtain g where p-expr: p = gluing `` {g} g ∈ e'-aff-bit
    using assms unfolding e-proj-def quotient-def by blast+
  then obtain x y l where g-expr: g = ((x,y),l) (x,y) ∈ e'-aff
    using e'-aff-bit-def by auto
  show ?thesis
    using e-proj-elim-1 e-proj-elim-2 gluing-class-1 gluing-class-2 g-expr p-expr by
```

meson
qed

lemma *e-proj-aff*:
gluing `` $\{((x,y),l)\} \in$ *e-proj* $\longleftrightarrow (x,y) \in$ *e′-aff*
proof
assume *gluing* `` $\{((x, y), l)\} \in$ *e-proj*
then show $(x,y) \in$ *e′-aff*
unfolding *e-proj-def e′-aff-bit-def*
apply(*rule quotientE*)
using *eq-equiv-class gluing-aff*
e′-aff-bit-def eq-rel **by** *fastforce*
next
assume *as*: $(x, y) \in$ *e′-aff*
show *gluing* `` $\{((x, y), l)\} \in$ *e-proj*
using *gluing-class-1*[*OF - as*] *gluing-class-2*[*OF - - as*]
e-proj-elim-1[*OF as*] *e-proj-elim-2*[*OF as*] **by** *fastforce*
qed

lemma *gluing-cases*:
assumes $x \in$ *e-proj*
obtains *x0 y0 l* **where** $x = \{((x0,y0),l)\} \lor x = \{((x0,y0),l),(\tau\ (x0,y0),l+1)\}$
using *e-proj-eq*[*OF assms*] *that* **by** *blast*

lemma *gluing-cases-explicit*:
assumes $x \in$ *e-proj* $x =$ *gluing* `` $\{((x0,y0),l)\}$
shows $x = \{((x0,y0),l)\} \lor x = \{((x0,y0),l),(\tau\ (x0,y0),l+1)\}$
proof −
have $(x0,y0) \in$ *e′-aff*
using *assms e-proj-aff* **by** *simp*
have *gluing* `` $\{((x0,y0),l)\} = \{((x0,y0),l)\} \lor$
gluing `` $\{((x0,y0),l)\} = \{((x0,y0),l),(\tau\ (x0,y0),l+1)\}$
using *assms gluing-class-1 gluing-class-2* ‹$(x0, y0) \in$ *e′-aff*› **by** *meson*
then show *?thesis* **using** *assms* **by** *fast*
qed

lemma *gluing-cases-points*:
assumes $x \in$ *e-proj* $x =$ *gluing* `` $\{(p,l)\}$
shows $x = \{(p,l)\} \lor x = \{(p,l),(\tau\ p,l+1)\}$
using *gluing-cases-explicit*[*OF assms*(*1*), *of fst p snd p l*] *assms* **by** *auto*

lemma *e-points*:
assumes $(x,y) \in$ *e′-aff*
shows *gluing* `` $\{((x,y),l)\} \in$ *e-proj*
using *assms e-proj-aff* **by** *simp*
lemma *e-class*:
assumes *gluing* `` $\{(p,l)\} \in$ *e-proj*

shows $p \in e'$-*aff*
using *assms e-proj-aff*
apply(*subst* (*asm*) *prod.collapse*[*symmetric*])
apply(*subst prod.collapse*[*symmetric*])
by *blast*

lemma *identity-equiv*:
gluing `` $\{((1,\ 0),\ l)\} = \{((1,0),l)\}$
unfolding *Image-def*
proof(*simp*,*standard*)
show $\{y.\ (((1,\ 0),\ l),\ y) \in gluing\} \subseteq \{((1,\ 0),\ l)\}$
using *gluing-char-zero* **by**(*intro subrelI*,*fast*)
have $(1,0) \in e'$-*aff*
unfolding e'-*aff-def* e'-*def* **by** *simp*
then have $((1,\ 0),\ l) \in e'$-*aff-bit*
using *zero-bit-def* **unfolding** e'-*aff-bit-def* **by** *blast*
show $\{((1,\ 0),\ l)\} \subseteq \{y.\ (((1,\ 0),\ l),\ y) \in gluing\}$
using *eq-rel* ‹$((1,\ 0),\ l) \in e'$-*aff-bit*›
unfolding *equiv-def refl-on-def* **by** *blast*
qed

lemma *identity-proj*:
$\{((1,0),l)\} \in e\text{-}proj$
proof −
have $(1,0) \in e'$-*aff*
unfolding e'-*aff-def* e'-*def* **by** *auto*
then show *?thesis*
using *e-proj-aff*[*of 1 0 l*] *identity-equiv* **by** *auto*
qed

lemma *gluing-inv*:
assumes $x \neq 0$ $y \neq 0$ $(x,y) \in e'$-*aff*
shows *gluing* `` $\{((x,y),j)\}$ = *gluing* `` $\{(\tau\ (x,y),j+1)\}$
proof −
have *taus*: $\tau\ (x,y) \in e'$-*aff*
using *e-circ-def assms* τ-*circ* **by** *fastforce*+

have *gluing* `` $\{((x,y),\ j)\} = \{((x,\ y),\ j),\ (\tau\ (x,\ y),\ j\ +\ 1)\}$
using *gluing-class-2 assms* **by** *meson*
also have ... $= \{(\tau\ (x,\ y),\ j{+}1),\ (\tau\ (\tau\ (x,\ y)),\ j)\}$
using *tau-idemp-explicit* **by** *force*
also have $\{(\tau\ (x,\ y),\ j{+}1),\ (\tau\ (\tau\ (x,\ y)),\ j)\}$ = *gluing* `` $\{(\tau\ (x,y),\ j\ +\ 1)\}$
apply(*subst gluing-class-2*[*of fst* $(\tau\ (x,y))$ *snd* $(\tau\ (x,y))$,
simplified prod.collapse])
using *assms taus t-nz* **by** *auto*
finally show *?thesis* **by** *blast*
qed

3.3 Projective addition on points

```
function (domintros) proj-add :: ('a × 'a) × bit ⇒ ('a × 'a) × bit ⇒ ('a × 'a)
× bit
  where
    proj-add ((x1, y1), l) ((x2, y2), j) = (add (x1, y1) (x2, y2), l+j)
      if delta x1 y1 x2 y2 ≠ 0 and (x1, y1) ∈ e'-aff and (x2, y2) ∈ e'-aff
  | proj-add ((x1, y1), l) ((x2, y2), j) = (ext-add (x1, y1) (x2, y2), l+j)
      if delta' x1 y1 x2 y2 ≠ 0 and (x1, y1) ∈ e'-aff and (x2, y2) ∈ e'-aff
  | proj-add ((x1, y1), l) ((x2, y2), j) = undefined
      if (x1, y1) ∉ e'-aff ∨ (x2, y2) ∉ e'-aff ∨
        (delta x1 y1 x2 y2 = 0 ∧ delta' x1 y1 x2 y2 = 0)
  apply(fast)
  apply(fastforce)
  using coherence e'-aff-def apply force
  by auto

termination proj-add using termination by blast

lemma proj-add-def:
    (proj-add ((x1, y1), l) ((x2, y2), j)) =
      (
        if ((x1, y1) ∈ e'-aff ∧ (x2, y2) ∈ e'-aff ∧ delta x1 y1 x2 y2 ≠ 0)
        then (add (x1, y1) (x2, y2), l + j)
        else
          (
            if ((x1, y1) ∈ e'-aff ∧ (x2, y2) ∈ e'-aff ∧ delta' x1 y1 x2 y2 ≠ 0)
            then (ext-add (x1, y1) (x2, y2), l + j)
            else undefined
          )
      )
    (is ?lhs = ?rhs)
proof(cases ‹delta x1 y1 x2 y2 ≠ 0 ∧ (x1, y1) ∈ e'-aff ∧ (x2, y2) ∈ e'-aff›)
  case True
  then have True-exp: delta x1 y1 x2 y2 ≠ 0 (x1, y1) ∈ e'-aff (x2, y2) ∈ e'-aff
    by auto
  then have rhs: ?rhs = (add (x1, y1) (x2, y2), l + j) by simp
  show ?thesis unfolding proj-add.simps(1)[OF True-exp, of l j] rhs ..
next
  case n0: False show ?thesis
  proof(cases ‹delta' x1 y1 x2 y2 ≠ 0 ∧ (x1, y1) ∈ e'-aff ∧ (x2, y2) ∈ e'-aff›)
    case True show ?thesis
    proof−
      from True n0 have False-exp:
        delta' x1 y1 x2 y2 ≠ 0 (x1, y1) ∈ e'-aff (x2, y2) ∈ e'-aff
        by auto
      with n0 have rhs: ?rhs = (ext-add (x1, y1) (x2, y2), l + j) by auto
      show ?thesis using proj-add.simps(2)[OF False-exp, of l j] rhs ..
    qed
  next
```

```
    case False then show ?thesis using n0 proj-add.simps(3) by auto
  qed
qed

lemma proj-add-inv:
  assumes (x0,y0) ∈ e'-aff
  shows proj-add ((x0,y0),l) (i (x0,y0),l') = ((1,0),l+l')
proof −
  have i-in: i (x0,y0) ∈ e'-aff
    using i-aff assms by blast

  consider (1) x0 = 0 | (2) y0 = 0 | (3) x0 ≠ 0 y0 ≠ 0 by fast
  then show ?thesis
  proof(cases)
    case 1
    from assms 1 have y-expr: y0 = 1 ∨ y0 = −1
      unfolding e'-aff-def e'-def by(simp,algebra)
    then show proj-add ((x0,y0),l) (i (x0,y0),l') = ((1,0),l+l')
      using 1
      apply(simp add: proj-add-def)
      unfolding delta-def delta-minus-def delta-plus-def
      apply(simp add: c-eq-1)
      unfolding e'-aff-def e'-def by auto
  next
    case 2
    from assms 2 have x0 = 1 ∨ x0 = −1
      unfolding e'-aff-def e'-def by(simp,algebra)
    then show ?thesis
      using 2
      apply(simp add: proj-add-def)
      unfolding delta-def delta-minus-def delta-plus-def
      apply(simp add: c-eq-1)
      unfolding e'-aff-def e'-def by force
  next
    case 3

    consider (a) delta x0 y0 x0 (−y0) = 0 delta' x0 y0 x0 (−y0) = 0 |
             (b) delta x0 y0 x0 (−y0) ≠ 0 delta' x0 y0 x0 (−y0) = 0 |
             (c) delta x0 y0 x0 (−y0) = 0 delta' x0 y0 x0 (−y0) ≠ 0 |
             (d) delta x0 y0 x0 (−y0) ≠ 0 delta' x0 y0 x0 (−y0) ≠ 0 by meson
    then show ?thesis
    proof(cases)
      case a
      then have d ∗ x0^2 ∗ y0^2 = 1 ∨ d ∗ x0^2 ∗ y0^2 = −1
                x0^2 = y0^2
                x0^2 + y0^2 − 1 = d ∗ x0^2 ∗ y0^2
        unfolding power2-eq-square
        using a unfolding delta-def delta-plus-def delta-minus-def apply algebra
        using 3 two-not-zero a unfolding delta'-def delta-x-def delta-y-def apply
```

```
force
      using assms t-expr unfolding e'-aff-def e'-def power2-eq-square by force
    then have 2*x0^2 = 2 ∨ 2*x0^2 = 0
      by algebra
    then have x0 = 1 ∨ x0 = -1
      using 3
      apply(simp add: two-not-zero)
      by algebra
    then have y0 = 0
      using assms t-n1 t-nm1
      unfolding e'-aff-def e'-def
      apply simp
      by algebra
    then have False
      using 3 by auto
    then show ?thesis by auto
  next
    case b
    have proj-add ((x0, y0), l) (i (x0, y0), l') =
          (add (x0, y0) (i (x0, y0)), l+l')
      using assms i-in b
      by(simp add: proj-add-def)
    also have ... = ((1,0),l+l')
      using inverse-generalized[OF assms] b
      unfolding delta-def delta-plus-def delta-minus-def
      by auto
    finally show ?thesis
      by blast
  next
    case c
    have proj-add ((x0, y0), l) (i (x0, y0), l') =
          (ext-add (x0, y0) (i (x0, y0)), l+l')
      using assms i-in c
      by(simp add: proj-add-def)
    also have ... = ((1,0),l+l')
      apply(subst ext-add-inverse)
      using 3 by auto
    finally show ?thesis
      by blast
  next
    case d
    have proj-add ((x0, y0), l) (i (x0, y0), l') =
          (add (x0, y0) (i (x0, y0)), l+l')
      using assms i-in d
      by(simp add: proj-add-def)
    also have ... = ((1,0),l+l')
      using inverse-generalized[OF assms] d
      unfolding delta-def delta-plus-def delta-minus-def
      by auto
```

```
    finally show ?thesis
      by blast
    qed
  qed
qed

lemma proj-add-comm:
  proj-add ((x0,y0),l) ((x1,y1),j) = proj-add ((x1,y1),j) ((x0,y0),l)
proof -
  consider
   (1) delta x0 y0 x1 y1 ≠ 0 ∧ (x0,y0) ∈ e'-aff ∧ (x1,y1) ∈ e'-aff |
   (2) delta' x0 y0 x1 y1 ≠ 0 ∧ (x0,y0) ∈ e'-aff ∧ (x1,y1) ∈ e'-aff |
   (3) (delta x0 y0 x1 y1 = 0 ∧ delta' x0 y0 x1 y1 = 0) ∨
         (x0,y0) ∉ e'-aff ∨ (x1,y1) ∉ e'-aff by blast
  then show ?thesis
  proof(cases)
    case 1 then show ?thesis by(simp add: commutativity delta-com)
  next
   case 2 then show ?thesis by(simp add: ext-add-comm delta'-com del: ext-add.simps)

  next
    case 3 then show ?thesis by(auto simp add: delta-com delta'-com)
  qed
qed
```

3.4 Projective addition on classes

```
function (domintros) proj-add-class :: (('a × 'a) × bit ) set ⇒ (('a × 'a) × bit )
set ⇒ ((('a × 'a) × bit ) set) set
  where
    proj-add-class c1 c2 =
        (
          {
            proj-add ((x1, y1), i) ((x2, y2), j) | x1 y1 i x2 y2 j.
              ((x1, y1), i) ∈ c1 ∧ ((x2, y2), j) ∈ c2 ∧
              ((x1, y1), (x2, y2)) ∈ e'-aff-0 ∪ e'-aff-1
          } // gluing
        )
      if c1 ∈ e-proj and c2 ∈ e-proj
      | proj-add-class c1 c2 = undefined
      if c1 ∉ e-proj ∨ c2 ∉ e-proj
  by (meson surj-pair) auto

termination proj-add-class using termination by auto

definition proj-add-class' where
  proj-add-class' c1 c2 =
      (
        (case-prod (proj-add) `
```

({(x, y). x ∈ c1 ∧ y ∈ c2 ∧ (fst x, fst y) ∈ e′-aff-0 ∪ e′-aff-1})) // gluing
)

definition *proj-addition* **where**
proj-addition c1 c2 = the-elem (proj-add-class c1 c2)

lemma *proj-add-class-eq*:
assumes *c1 ∈ e-proj* **and** *c2 ∈ e-proj*
shows *proj-add-class′ c1 c2 = proj-add-class c1 c2*
proof−
have
(λ(x, y). proj-add x y) ‘
{(x, y). x ∈ c1 ∧ y ∈ c2 ∧ (fst x, fst y) ∈ e′-aff-0 ∪ e′-aff-1} =
{
proj-add ((x1, y1), i) ((x2, y2), j) | x1 y1 i x2 y2 j.
((x1, y1), i) ∈ c1 ∧ ((x2, y2), j) ∈ c2 ∧
((x1, y1), (x2, y2)) ∈ e′-aff-0 ∪ e′-aff-1
}
apply (*standard*; *standard*)
subgoal unfolding *image-def* **by** *clarsimp blast*
subgoal unfolding *image-def* **by** *clarsimp blast*
done
then show *?thesis*
unfolding *proj-add-class′-def proj-add-class.simps(1)[OF assms]*
by *auto*
qed

3.4.1 Covering

corollary *no-fp-eq*:
assumes *p ∈ e-circ*
assumes *r′ ∈ rotations r ∈ rotations*
assumes *(r′ ∘ i) p = (τ ∘ r) (i p)*
shows *False*
proof −
obtain *r′′* **where** *r′′ ∘ r′ = id r′′ ∈ rotations*
using *rot-inv assms* **by** *blast*
then have *i p = (r′′ ∘ τ ∘ r) (i p)*
using *assms* **by** (*simp*,*metis pointfree-idE*)
then have *i p = (τ ∘ r′′ ∘ r) (i p)*
using *rot-tau-com[OF ‹r′′ ∈ rotations›]* **by** *simp*
then have *∃ r′′. r′′ ∈ rotations ∧ i p = (τ ∘ r′′) (i p)*
using *rot-comp[OF ‹r′′ ∈ rotations›] assms* **by** *fastforce*
then obtain *r′′* **where**
eq: *r′′ ∈ rotations i p = (τ ∘ r′′) (i p)*
by *blast*
have *τ ∘ r′′ ∈ G i p ∈ e-circ*
using *tau-rot-sym[OF ‹r′′ ∈ rotations›] G-partition* **apply** *simp*
using *i-circ-points assms(1)* **by** *simp*

```
  then show False
    using g-no-fp[OF ‹τ ∘ r'' ∈ G› ‹i p ∈ e-circ›]
          eq assms(1) sym-not-id[OF eq(1)] by argo
qed

lemma covering:
  assumes p ∈ e-proj q ∈ e-proj
  shows proj-add-class p q ≠ {}
proof −
  from e-proj-eq[OF assms(1)] e-proj-eq[OF assms(2)]
  obtain x y l x' y' l' where
    p-q-expr: p = {((x, y), l)} ∨ p = {((x, y), l), (τ (x, y), l + 1)}
              q = {((x', y'), l')} ∨ q = {((x', y'), l'), (τ (x', y'), l' + 1)}
              (x,y) ∈ e'-aff (x',y') ∈ e'-aff
    by blast
  then have in-aff: (x,y) ∈ e'-aff (x',y') ∈ e'-aff by auto
  from p-q-expr have gluings: p = (gluing `` {((x,y),l)})
                               q = (gluing `` {((x',y'),l')})
    using assms e-proj-elim-1 e-proj-elim-2 gluing-class-1 gluing-class-2
    by metis+
  then have gluing-proj: (gluing `` {((x,y),l)}) ∈ e-proj
                         (gluing `` {((x',y'),l')}) ∈ e-proj
    using assms by blast+

  consider
      (x, y) ∈ e-circ ∧ (∃ g∈symmetries. (x', y') = (g ∘ i) (x, y))
    | ((x, y), x', y') ∈ e'-aff-0
    | ((x, y), x', y') ∈ e'-aff-1
    using dichotomy-1[OF ‹(x,y) ∈ e'-aff› ‹(x',y') ∈ e'-aff›] by blast
  then show ?thesis
  proof(cases)
    case 1
    then obtain r where r-expr: (x',y') = (τ ∘ r) (i (x,y)) r ∈ rotations
      using sym-decomp by force

    then have nz: x ≠ 0 y ≠ 0 x' ≠ 0 y' ≠ 0
      using 1 t-nz unfolding e-circ-def rotations-def by force+

    have taus: τ (x',y') ∈ e'-aff
      using nz i-aff p-q-expr(3) r-expr rot-aff tau-idemp-point by auto

    have circ: (x,y) ∈ e-circ
      using nz in-aff e-circ-def by blast

    have p-q-expr': p = {((x,y),l),(τ (x,y),l+1)}
                    q = {(τ (x',y'),l'+1),((x',y'),l')}
     using gluings nz gluing-class-2 taus in-aff tau-idemp-point t-nz assms by auto

    have p-q-proj: {((x,y),l),(τ (x,y),l+1)} ∈ e-proj
```

```
          {(τ (x',y'),l'+1),((x',y'),l')} ∈ e-proj
    using p-q-expr' assms by auto

  consider
   (a)  (x, y) ∈ e-circ ∧ (∃ g∈symmetries. τ (x', y') = (g ∘ i) (x, y))
  |(b)  ((x, y), τ (x', y')) ∈ e'-aff-0
  |(c)  ((x, y), τ (x', y')) ∈ e'-aff-1
    using dichotomy-1[OF ‹(x,y) ∈ e'-aff› ‹τ (x', y') ∈ e'-aff›] by blast
  then show ?thesis
  proof(cases)
    case a
    then obtain r' where r'-expr: τ (x',y') = (τ ∘ r') (i (x, y)) r' ∈ rotations
      using sym-decomp by force
    have (x',y') = r' (i (x, y))
    proof−
      have (x',y') = τ (τ (x',y'))
        using tau-idemp-point by presburger
      also have ... = τ ((τ ∘ r') (i (x, y)))
        using r'-expr by argo
      also have ... = r' (i (x, y))
        using tau-idemp-point by simp
      finally show ?thesis by simp
    qed
    then have False
      using no-fp-eq[OF circ r'-expr(2) r-expr(2)] r-expr by simp
    then show ?thesis by blast
  next
    case b
    then have ds: delta x y (fst (τ (x',y'))) (snd (τ (x',y'))) ≠ 0
      unfolding e'-aff-0-def by simp
    then have
        add-some: proj-add ((x,y),l) (τ (x',y'),l'+1) = (add (x, y) (τ (x',y')),
l+l'+1)
      using proj-add.simps[of x y - - l l'+1, OF - ]
            ‹(x,y) ∈ e'-aff› ‹τ (x', y') ∈ e'-aff› by force
    then show ?thesis
      unfolding p-q-expr' proj-add-class.simps(1)[OF p-q-proj]
      unfolding e'-aff-0-def using ds in-aff taus by force
  next
    case c
    then have ds: delta' x y (fst (τ (x',y'))) (snd (τ (x',y'))) ≠ 0
      unfolding e'-aff-1-def by simp
    then have
       add-some: proj-add ((x,y),l) (τ (x',y'),l'+1) = (ext-add (x, y) (τ (x',y')),
l+l'+1)
      using proj-add.simps[of x y - - l l'+1, OF - ]
            ‹(x,y) ∈ e'-aff› ‹τ (x', y') ∈ e'-aff› by force
    then show ?thesis
      unfolding p-q-expr' proj-add-class.simps(1)[OF p-q-proj]
```

```
      unfolding e'-aff-1-def using ds in-aff taus by force
  qed
  next
    case 2
    then have ds: delta x y x' y' ≠ 0
      unfolding e'-aff-0-def by simp
    then have
      add-some: proj-add ((x,y),l) ((x',y'),l') = (add (x, y) (x',y'), l+l')
      using proj-add.simps(1)[of x y x' y' l l', OF - ] in-aff by blast
    then show ?thesis
      using p-q-expr
      unfolding proj-add-class.simps(1)[OF assms]
      unfolding e'-aff-0-def using ds in-aff by fast
  next
    case 3
    then have ds: delta' x y x' y' ≠ 0
      unfolding e'-aff-1-def by simp
    then have
      add-some: proj-add ((x,y),l) ((x',y'),l') = (ext-add (x, y) (x',y'), l+l')
      using proj-add.simps(2)[of x y x' y' l l', OF - ] in-aff by blast
    then show ?thesis
      using p-q-expr
      unfolding proj-add-class.simps(1)[OF assms]
      unfolding e'-aff-1-def using ds in-aff by fast
  qed
qed

lemma covering-with-deltas:
  assumes (gluing `` {((x,y),l)}) ∈ e-proj (gluing `` {((x',y'),l')}) ∈ e-proj
  shows delta x y x' y' ≠ 0 ∨ delta' x y x' y' ≠ 0 ∨
        delta x y (fst (τ (x',y'))) (snd (τ (x',y'))) ≠ 0 ∨
        delta' x y (fst (τ (x',y'))) (snd (τ (x',y'))) ≠ 0
proof –
  define p where p = (gluing `` {((x,y),l)})
  define q where q = (gluing `` {((x',y'),l')})
  have p ∈ e'-aff-bit // gluing
    using assms(1) p-def unfolding e-proj-def by blast
  from e-proj-eq[OF assms(1)] e-proj-eq[OF assms(2)]
  have
    p-q-expr: p = {((x, y), l)} ∨ p = {((x, y), l), (τ (x, y), l + 1)}
    q = {((x', y'), l')} ∨ q = {((x', y'), l'), (τ (x', y'), l' + 1)}
    (x,y) ∈ e'-aff (x',y') ∈ e'-aff
    using p-def q-def
    using assms(1) gluing-cases-explicit apply auto[1]
    using assms(2) gluing-cases-explicit q-def apply auto[1]
   using assms(1) e'-aff-bit-def e-proj-def eq-rel gluing-cases-explicit in-quotient-imp-subset
apply fastforce
   using assms(2) e'-aff-bit-def e-proj-def eq-rel gluing-cases-explicit in-quotient-imp-subset
by fastforce
```

then have *in-aff*: $(x,y) \in e'\text{-}aff$ $(x',y') \in e'\text{-}aff$ **by** *auto*

then have *gluings*: $p = (gluing \text{ `` } \{((x,y),l)\})$
$q = (gluing \text{ `` } \{((x',y'),l')\})$
using *p-def q-def* **by** *simp+*

then have *gluing-proj*: $(gluing \text{ `` } \{((x,y),l)\}) \in e\text{-}proj$
$(gluing \text{ `` } \{((x',y'),l')\}) \in e\text{-}proj$
using *assms* **by** *blast+*

consider
$(x, y) \in e\text{-}circ \wedge (\exists g{\in}symmetries.\ (x', y') = (g \circ i)\ (x, y))$
| $((x, y), x', y') \in e'\text{-}aff\text{-}0$
| $((x, y), x', y') \in e'\text{-}aff\text{-}1$
using *dichotomy-1*[*OF* ‹$(x,y) \in e'\text{-}aff$› ‹$(x',y') \in e'\text{-}aff$›] **by** *blast*
then show *?thesis*
proof(*cases*)
case *1*
then obtain *r* **where** *r-expr*: $(x',y') = (\tau \circ r)\ (i\ (x,y))$ $r \in rotations$
using *sym-decomp* **by** *force*

then have *nz*: $x \neq 0$ $y \neq 0$ $x' \neq 0$ $y' \neq 0$
using *1 t-nz* **unfolding** *e-circ-def rotations-def* **by** *force+*

have *taus*: $\tau\ (x',y') \in e'\text{-}aff$
using *nz i-aff p-q-expr(3) r-expr rot-aff tau-idemp-point* **by** *auto*

have *circ*: $(x,y) \in e\text{-}circ$
using *nz in-aff e-circ-def* **by** *blast*

have *p-q-expr'*: $p = \{((x,y),l),(\tau\ (x,y),l{+}1)\}$
$q = \{(\tau\ (x',y'),l'{+}1),((x',y'),l')\}$
using *gluings nz gluing-class-2 taus in-aff tau-idemp-point t-nz assms* **by** *auto*

have *p-q-proj*: $\{((x,y),l),(\tau\ (x,y),l{+}1)\} \in e\text{-}proj$
$\{(\tau\ (x',y'),l'{+}1),((x',y'),l')\} \in e\text{-}proj$
using *p-q-expr p-q-expr' assms gluing-proj gluings* **by** *auto*

consider
(*a*) $(x, y) \in e\text{-}circ \wedge (\exists g{\in}symmetries.\ \tau\ (x', y') = (g \circ i)\ (x, y))$
| (*b*) $((x, y), \tau\ (x', y')) \in e'\text{-}aff\text{-}0$
| (*c*) $((x, y), \tau\ (x', y')) \in e'\text{-}aff\text{-}1$
using *dichotomy-1*[*OF* ‹$(x,y) \in e'\text{-}aff$› ‹$\tau\ (x', y') \in e'\text{-}aff$›] **by** *blast*
then show *?thesis*
proof(*cases*)
case *a*
then obtain *r'* **where** *r'-expr*: $\tau\ (x',y') = (\tau \circ r')\ (i\ (x, y))$ $r' \in rotations$
using *sym-decomp* **by** *force*
have $(x',y') = r'\ (i\ (x, y))$

proof−
have $(x',y') = \tau\ (\tau\ (x',y'))$
using *tau-idemp-point* **by** *presburger*
also have ... $= \tau\ ((\tau \circ r')\ (i\ (x,\ y)))$
using *r′-expr* **by** *argo*
also have ... $= r'\ (i\ (x,\ y))$
using *tau-idemp-point* **by** *simp*
finally show *?thesis* **by** *simp*
qed
then have *False*
using *no-fp-eq*[*OF circ r′-expr*(*2*) *r-expr*(*2*)] *r-expr* **by** *simp*
then show *?thesis* **by** *blast*
next
case *b*
define x'' **where** $x'' = fst\ (\tau\ (x',y'))$
define y'' **where** $y'' = snd\ (\tau\ (x',y'))$
from *b* **have** $delta\ x\ y\ x''\ y'' \neq 0$
unfolding *e′-aff-0-def* **using** *x′′-def y′′-def* **by** *simp*
then show *?thesis*
unfolding *x′′-def y′′-def* **by** *blast*
next
case *c*
define x'' **where** $x'' = fst\ (\tau\ (x',y'))$
define y'' **where** $y'' = snd\ (\tau\ (x',y'))$
from *c* **have** $delta'\ x\ y\ x''\ y'' \neq 0$
unfolding *e′-aff-1-def* **using** *x′′-def y′′-def* **by** *simp*
then show *?thesis*
unfolding *x′′-def y′′-def* **by** *blast*
qed
next
case *2*
then have $delta\ x\ y\ x'\ y' \neq 0$
unfolding *e′-aff-0-def* **by** *simp*
then show *?thesis* **by** *simp*
next
case *3*
then have $delta'\ x\ y\ x'\ y' \neq 0$
unfolding *e′-aff-1-def* **by** *simp*
then show *?thesis* **by** *simp*
qed
qed

3.4.2 Independence of the representant

lemma *proj-add-class-comm*:
assumes $c1 \in e\text{-}proj\ c2 \in e\text{-}proj$
shows $proj\text{-}add\text{-}class\ c1\ c2 = proj\text{-}add\text{-}class\ c2\ c1$
proof −
have $((x1,\ y1),\ x2,\ y2) \in e'\text{-}aff\text{-}0 \cup e'\text{-}aff\text{-}1 \Longrightarrow$

$((x2, y2), x1, y1) \in e'\text{-}aff\text{-}0 \cup e'\text{-}aff\text{-}1$ **for** *x1 y1 x2 y2*
unfolding *e′-aff-0-def e′-aff-1-def*
e′-aff-def e′-def
delta-def delta-plus-def delta-minus-def
delta′-def delta-x-def delta-y-def
by(*simp*,*algebra*)
then have {*proj-add* ((*x1*, *y1*), *i*) ((*x2*, *y2*), *j*) |*x1 y1 i x2 y2 j*.
((*x1*, *y1*), *i*) ∈ *c1* ∧ ((*x2*, *y2*), *j*) ∈ *c2* ∧ ((*x1*, *y1*), *x2*, *y2*) ∈ *e′-aff-0* ∪ *e′-aff-1*} =
{*proj-add* ((*x1*, *y1*), *i*) ((*x2*, *y2*), *j*) |*x1 y1 i x2 y2 j*.
((*x1*, *y1*), *i*) ∈ *c2* ∧ ((*x2*, *y2*), *j*) ∈ *c1* ∧ ((*x1*, *y1*), *x2*, *y2*) ∈ *e′-aff-0* ∪ *e′-aff-1*}
using *proj-add-comm* **by** *blast*
then show *?thesis*
unfolding *proj-add-class.simps*(*1*)[*OF assms*]
proj-add-class.simps(*1*)[*OF assms*(*2*) *assms*(*1*)] **by** *argo*
qed

lemma *gluing-add-1*:
assumes *gluing* ‘‘ {((*x*,*y*),*l*)} = {((*x*, *y*), *l*)} *gluing* ‘‘ {((*x′*,*y′*),*l′*)} = {((*x′*, *y′*), *l′*)}
gluing ‘‘ {((*x*,*y*),*l*)} ∈ *e-proj gluing* ‘‘ {((*x′*,*y′*),*l′*)} ∈ *e-proj delta x y x′ y′* ≠ *0*
shows *proj-addition* (*gluing* ‘‘ {((*x*,*y*),*l*)}) (*gluing* ‘‘ {((*x′*,*y′*),*l′*)}) = (*gluing* ‘‘ {(*add* (*x*,*y*) (*x′*,*y′*),*l*+*l′*)})
proof −
have *in-aff*: (*x*,*y*) ∈ *e′-aff* (*x′*,*y′*) ∈ *e′-aff*
using *assms e-proj-eq e-class* **by** *blast*+
then have *add-in*: *add* (*x*, *y*) (*x′*, *y′*) ∈ *e′-aff*
using *add-closure* ‹*delta x y x′ y′* ≠ *0*› *delta-def e-e′-iff e′-aff-def* **by** *auto*
from *in-aff* **have** *zeros*: *x* = *0* ∨ *y* = *0* *x′* = *0* ∨ *y′* = *0*
using *e-proj-elim-1 assms* **by** *presburger*+
then have *add-zeros*: *fst* (*add* (*x*,*y*) (*x′*,*y′*)) = *0* ∨ *snd* (*add* (*x*,*y*) (*x′*,*y′*)) = *0*
by *auto*
then have *add-proj*: *gluing* ‘‘ {(*add* (*x*, *y*) (*x′*, *y′*), *l* + *l′*)} = {(*add* (*x*, *y*) (*x′*, *y′*), *l* + *l′*)}
using *add-in gluing-class-1* **by** *auto*
have *e-proj*: *gluing* ‘‘ {((*x*,*y*),*l*)} ∈ *e-proj*
gluing ‘‘ {((*x′*,*y′*),*l′*)} ∈ *e-proj*
gluing ‘‘ {(*add* (*x*, *y*) (*x′*, *y′*), *l* + *l′*)} ∈ *e-proj*
using *e-proj-aff in-aff add-in* **by** *auto*

consider
(*a*) (*x*, *y*) ∈ *e-circ* ∧ (∃ *g*∈*symmetries*. (*x′*, *y′*) = (*g* ∘ *i*) (*x*, *y*)) |
(*b*) ((*x*, *y*), *x′*, *y′*) ∈ *e′-aff-0* ¬ ((*x*, *y*) ∈ *e-circ* ∧ (∃ *g*∈*symmetries*. (*x′*, *y′*) = (*g* ∘ *i*) (*x*, *y*))) |
(*c*) ((*x*, *y*), *x′*, *y′*) ∈ *e′-aff-1* ¬ ((*x*, *y*) ∈ *e-circ* ∧ (∃ *g*∈*symmetries*. (*x′*, *y′*) = (*g* ∘ *i*) (*x*, *y*))) ((*x*, *y*), *x′*, *y′*) ∉ *e′-aff-0*
using *dichotomy-1*[*OF* ‹(*x*,*y*) ∈ *e′-aff*› ‹(*x′*,*y′*) ∈ *e′-aff*›] **by** *argo*

then show *?thesis*
proof(*cases*)
  case *a*
  then have *False*
    using *in-aff zeros* **unfolding** *e-circ-def* **by** *force*
  then show *?thesis* **by** *simp*
next
  case *b*
  have *add-eq*: *proj-add* $((x,\ y),\ l)\ ((x',\ y'),\ l') = (add\ (x,y)\ (x',\ y'),\ l{+}l')$
    using *proj-add.simps* ‹*delta x y x′ y′* ≠ *0*› *in-aff* **by** *simp*
  show *?thesis*
    unfolding *proj-addition-def*
    unfolding *proj-add-class.simps*(*1*)[*OF e-proj*(*1*,*2*)] *add-proj*
    unfolding *assms*(*1*,*2*) *e′-aff-0-def*
    using ‹*delta x y x′ y′* ≠ *0*› *in-aff*
    apply(*simp add*: *add-eq del*: *add.simps*)
    apply(*subst eq-class-simp*)
    using *add-proj e-proj* **by** *auto*
next
  case *c*
  then have *eqs*: *delta x y x′ y′* = *0 delta′ x y x′ y′* ≠ *0 e x y* = *0 e x′ y′* = *0*
    unfolding *e′-aff-0-def e′-aff-1-def* **apply** *fast+*
    using *e-e′-iff in-aff* **unfolding** *e′-aff-def* **by** *fast+*
  then show *?thesis* **using** *assms* **by** *simp*
qed
qed

lemma *gluing-add-2*:
  assumes *gluing* `` {((*x*,*y*),*l*)} = {((*x*, *y*), *l*)} *gluing* `` {((*x′*,*y′*),*l′*)} = {((*x′*, *y′*), *l′*), (*τ* (*x′*, *y′*), *l′* + *1*)}
    gluing `` {((*x*,*y*),*l*)} ∈ *e-proj gluing* `` {((*x′*,*y′*),*l′*)} ∈ *e-proj delta x y x′ y′* ≠ *0*
  shows *proj-addition* (*gluing* `` {((*x*,*y*),*l*)}) (*gluing* `` {((*x′*,*y′*),*l′*)}) = (*gluing* `` {(*add* (*x*,*y*) (*x′*,*y′*),*l*+*l′*)})
proof −
  have *in-aff*: (*x*,*y*) ∈ *e′-aff* (*x′*,*y′*) ∈ *e′-aff*
    using *assms e-proj-eq e-class* **by** *blast+*
  then have *add-in*: *add* (*x*, *y*) (*x′*, *y′*) ∈ *e′-aff*
    using *add-closure* ‹*delta x y x′ y′* ≠ *0*› *delta-def e-e′-iff e′-aff-def* **by** *auto*
  from *in-aff* **have** *zeros*: $x = 0 \lor y = 0\ x' \neq 0\ \ y' \neq 0$
    using *e-proj-elim-1 e-proj-elim-2 assms* **by** *presburger+*
  have *e-proj*: *gluing* `` {((*x*,*y*),*l*)} ∈ *e-proj*
      gluing `` {((*x′*,*y′*),*l′*)} ∈ *e-proj*
      gluing `` {(*add* (*x*, *y*) (*x′*, *y′*), *l* + *l′*)} ∈ *e-proj*
    using *e-proj-aff in-aff add-in* **by** *auto*

  consider
    (*a*) (*x*, *y*) ∈ *e-circ* ∧ (∃ *g*∈*symmetries*. (*x′*, *y′*) = (*g* ∘ *i*) (*x*, *y*)) |
    (*b*) ((*x*, *y*), *x′*, *y′*) ∈ *e′-aff-0* ¬ ((*x*, *y*) ∈ *e-circ* ∧ (∃ *g*∈*symmetries*. (*x′*, *y′*)

$= (g \circ i)\ (x,\ y)))$ |
  (c) $((x,\ y),\ x',\ y') \in$ *e′-aff-1* $\neg$ $((x,\ y) \in$ *e-circ* $\wedge$ $(\exists\, g{\in}$*symmetries*. $(x',\ y')$ $= (g \circ i)\ (x,\ y)))$ $((x,\ y),\ x',\ y') \notin$ *e′-aff-0*
    using *dichotomy-1*[*OF* ‹$(x,y) \in$ *e′-aff*› ‹$(x',y') \in$ *e′-aff*›] **by** *fast*
 then show *?thesis*
 proof(*cases*)
  case *a*
  then have *False*
   using *in-aff zeros* **unfolding** *e-circ-def* **by** *force*
  then show *?thesis* **by** *simp*
 next
  case *b*
  then have *ld-nz*: *delta x y x′ y′* $\neq$ *0* **unfolding** *e′-aff-0-def* **by** *auto*

  have *v1*: *proj-add* $((x,\ y),\ l)$ $((x',\ y'),\ l') = ($*add* $(x,\ y)$ $(x',\ y'),\ l + l')$
   by(*simp add*: ‹$(x,y) \in$ *e′-aff*› ‹$(x',y') \in$ *e′-aff*› *ld-nz del*: *add.simps*)

  have *ecirc*: $(x',y') \in$ *e-circ* $x' \neq 0$ $y' \neq 0$
   unfolding *e-circ-def* **using** *zeros* ‹$(x',y') \in$ *e′-aff*› **by** *blast+*
  then have τ $(x',\ y') \in$ *e-circ*
   using *zeros* τ*-circ* **by** *blast*
  then have *in-aff′*: τ $(x',\ y') \in$ *e′-aff*
   unfolding *e-circ-def* **by** *force*

  have *add-nz*: *fst* (*add* $(x,\ y)$ $(x',\ y')$) $\neq$ *0*
      snd (*add* $(x,\ y)$ $(x',\ y')$) $\neq$ *0*
   using *zeros ld-nz in-aff*
   unfolding *delta-def delta-plus-def delta-minus-def e′-aff-def e′-def*
   apply(*simp-all*)
   apply(*simp-all add*: *c-eq-1*)
   by *auto*

  have *add-in*: *add* $(x,\ y)$ $(x',\ y') \in$ *e′-aff*
   using *add-closure in-aff e-e′-iff ld-nz* **unfolding** *e′-aff-def delta-def* **by** *simp*

  have *ld-nz′*: *delta x y* (*fst* (τ (x',y'))) (*snd* (τ (x',y'))) $\neq$ *0*
   unfolding *delta-def delta-plus-def delta-minus-def*
   using *zeros* **by** *fastforce*

  have *tau-conv*: τ (*add* $(x,\ y)$ $(x',\ y')$) = *add* $(x,\ y)$ (τ $(x',\ y')$)
   using *zeros e′-aff-x0*[*OF - in-aff*(*1*)] *e′-aff-y0*[*OF - in-aff*(*1*)]
   apply(*simp-all*)
   apply(*simp-all add*: *c-eq-1 divide-simps d-nz t-nz*)
   apply(*elim disjE*)
   apply(*simp-all add*: *t-nz zeros*)
   by *auto*

  have *v2*: *proj-add* $((x,\ y),\ l)$ (τ $(x',\ y'),\ l' + 1$) = (τ (*add* $(x,\ y)$ $(x',\ y')$), $l{+}l'{+}1$)

```
      using proj-add.simps ‹τ (x', y') ∈ e'-aff› in-aff tau-conv
            ‹delta x y (fst (τ (x', y'))) (snd (τ (x', y'))) ≠ 0› by auto

    have gl-class: gluing `` {(add (x, y) (x', y'), l + l')} =
              {(add (x, y) (x', y'), l + l'), (τ (add (x, y) (x', y')), l + l' + 1)}
          gluing `` {(add (x, y) (x', y'), l + l')} ∈ e-proj
      using gluing-class-2 e-points add-nz add-in apply simp
      using e-points add-nz add-in by force

    show ?thesis
    proof −
      have {proj-add ((x1, y1), i) ((x2, y2), j) |x1 y1 i x2 y2 j.
       ((x1, y1), i) ∈ {((x, y), l)} ∧
       ((x2, y2), j) ∈ {((x', y'), l'), (τ (x', y'), l' + 1)} ∧
       ((x1, y1), x2, y2)
       ∈ {((x1, y1), x2, y2). (x1, y1) ∈ e'-aff ∧ (x2, y2) ∈ e'-aff ∧ delta x1 y1 x2
y2 ≠ 0} ∪ e'-aff-1} =
      {proj-add ((x, y), l) ((x', y'), l'), proj-add ((x, y), l) (τ (x', y'), l' + 1)}
        (is ?t = -)
        using ld-nz ld-nz' in-aff in-aff'
        apply(simp del: τ.simps add.simps)
        by force
      also have ... = {(add (x, y) (x', y'), l + l'), (τ (add (x, y) (x', y')), l + l'
+ 1)}
        using v1 v2 by presburger
      finally have eq: ?t = {(add (x, y) (x', y'), l + l'), (τ (add (x, y) (x', y')), l
+ l' + 1)}
        by blast

      show ?thesis
       unfolding proj-addition-def
       unfolding proj-add-class.simps(1)[OF e-proj(1,2)]
       unfolding assms(1,2) gl-class e'-aff-0-def
       apply(subst eq)
       apply(subst eq-class-simp)
       using gl-class by auto
   qed
  next
   case c
    have ld-nz: delta x y x' y' = 0
     using ‹(x,y) ∈ e'-aff› ‹(x',y') ∈ e'-aff› c
     unfolding e'-aff-0-def by force
    then have False
      using assms e-proj-elim-1 in-aff
      unfolding delta-def delta-minus-def delta-plus-def by blast
    then show ?thesis by blast
  qed
qed
```

lemma *gluing-add-4*:
assumes *gluing* `` $\{((x, y), l)\} = \{((x, y), l), (\tau\ (x, y), l + 1)\}$
gluing `` $\{((x', y'), l')\} = \{((x', y'), l'), (\tau\ (x', y'), l' + 1)\}$
gluing `` $\{((x, y), l)\} \in$ *e-proj gluing* `` $\{((x', y'), l')\} \in$ *e-proj delta x y x' y'* $\neq 0$
shows *proj-addition* (*gluing* `` $\{((x, y), l)\}$) (*gluing* `` $\{((x', y'), l')\}$) =
gluing `` $\{($*add* $(x, y)\ (x',y'), l+l')\}$
(**is** *proj-addition ?p ?q* = -)
proof −
have *in-aff*: $(x,y) \in$ *e'-aff* $(x',y') \in$ *e'-aff*
using *e-proj-aff assms* **by** *meson+*
then have *nz*: $x \neq 0$ $y \neq 0$ $x' \neq 0$ $y' \neq 0$
using *assms e-proj-elim-2* **by** *auto*
then have *circ*: $(x,y) \in$ *e-circ* $(x',y') \in$ *e-circ*
using *in-aff e-circ-def nz* **by** *auto*
then have *taus*: $(\tau\ (x', y')) \in$ *e'-aff* $(\tau\ (x, y)) \in$ *e'-aff* $\tau\ (x', y') \in$ *e-circ*
using *τ-circ circ-to-aff* **by** *auto*

consider
(*a*) $(x, y) \in$ *e-circ* $\wedge$ ($\exists g \in$*symmetries*. $(x', y') = (g \circ i)\ (x, y)$)
| (*b*) $((x, y), x', y') \in$ *e'-aff-0*
| (*c*) $((x, y), x', y') \in$ *e'-aff-1* $((x, y), x', y') \notin$ *e'-aff-0*
using *dichotomy-1*[*OF in-aff*] **by** *auto*
then show *?thesis*
proof(*cases*)
case *a*
then obtain *g* **where** *sym-expr*: $g \in$ *symmetries* $(x', y') = (g \circ i)\ (x, y)$ **by** *auto*
then have *ds*: *delta x y x' y'* = *0* *delta' x y x' y'* = *0*
using *wd-d-nz wd-d'-nz a* **by** *auto*
then have *False*
using *assms* **by** *auto*
then show *?thesis* **by** *blast*
next
case *b*
then have *ld-nz*: *delta x y x' y'* $\neq 0$
unfolding *e'-aff-0-def* **by** *auto*
then have *ds*: *delta* (*fst* $(\tau\ (x, y))$) (*snd* $(\tau\ (x, y))$) (*fst* $(\tau\ (x', y'))$) (*snd* $(\tau\ (x', y'))$) $\neq 0$
unfolding *delta-def delta-plus-def delta-minus-def*
apply(*simp add*: *algebra-simps power2-eq-square*[*symmetric*])
unfolding *t-expr*[*symmetric*]
by(*simp add*: *field-simps*)
have *v1*: *proj-add* $((x, y), l)\ ((x', y'), l') = ($*add* $(x, y)\ (x', y'), l + l')$
using *ld-nz proj-add.simps* ‹$(x,y) \in$ *e'-aff*› ‹$(x',y') \in$ *e'-aff*› **by** *simp*
have *v2*: *proj-add* $(\tau\ (x, y), l+1)\ (\tau\ (x', y'), l'+1) = ($*add* $(x, y)\ (x', y'), l + l')$
using *ds proj-add.simps taus*
inversion-invariance-1 nz tau-idemp proj-add.simps

```
    by (simp add: c-eq-1 t-nz)

  consider (aaa) delta x y (fst (τ (x', y'))) (snd (τ (x', y'))) ≠ 0 |
          (bbb) delta' x y (fst (τ (x', y'))) (snd (τ (x', y'))) ≠ 0
                delta x y (fst (τ (x', y'))) (snd (τ (x', y'))) = 0 |
          (ccc) delta' x y (fst (τ (x', y'))) (snd (τ (x', y'))) = 0
                delta x y (fst (τ (x', y'))) (snd (τ (x', y'))) = 0 by blast
  then show ?thesis
  proof(cases)
    case aaa
    have tau-conv: τ (add (x, y) (τ (x', y'))) = add (x,y) (x',y')
      apply(simp)
      apply(simp add: c-eq-1)
      using aaa in-aff ld-nz
      unfolding e'-aff-def e'-def delta-def delta-minus-def delta-plus-def
      apply(safe)
      apply(simp-all add: divide-simps t-nz nz)
     apply(simp-all add: algebra-simps power2-eq-square[symmetric] t-expr d-nz)
      unfolding t-expr[symmetric]
      by algebra+

    have v3:
      proj-add ((x, y), l) (τ (x', y'), l' + 1) = (τ (add (x, y) (x', y')), l+l'+1)
      using proj-add.simps ‹(τ (x', y')) ∈ e'-aff›
      apply(simp del: add.simps τ.simps)
      using tau-conv tau-idemp-explicit
            proj-add.simps(1)[OF aaa ‹(x,y) ∈ e'-aff›,simplified prod.collapse,OF
‹(τ (x', y')) ∈ e'-aff›]
      by (metis (no-types, lifting) add.assoc prod.collapse)

    have ds': delta (fst (τ (x, y))) (snd (τ (x, y))) x' y' ≠ 0
      using aaa unfolding delta-def delta-plus-def delta-minus-def
       apply(simp add: t-nz nz algebra-simps power2-eq-square[symmetric] t-expr
d-nz)
      by(simp add: divide-simps nz t-nz)

     have v4: proj-add (τ (x, y), l+1) ((x', y'), l') = (τ (add (x, y) (x', y')),
l+l'+1)
    proof -
     have proj-add (τ (x, y), l+1) ((x', y'), l') = (add (τ (x, y)) (x', y'), l+l'+1)

        using proj-add.simps ‹τ (x,y) ∈ e'-aff› ‹(x', y') ∈ e'-aff› ds' by auto
      moreover have add (τ (x, y)) (x', y') = τ (add (x, y) (x', y'))
          by (metis inversion-invariance-1 nz(1) nz(2) nz(3) nz(4) tau-conv
tau-idemp-point)
      ultimately show ?thesis by argo
    qed

    have add-closure: add (x,y) (x',y') ∈ e'-aff
```

using *in-aff add-closure ld-nz e-e′-iff* **unfolding** *delta-def e′-aff-def* **by** *auto*

have *add-nz*: *fst* (*add* (*x*,*y*) (*x′*,*y′*)) ≠ *0*
snd (*add* (*x*,*y*) (*x′*,*y′*)) ≠ *0*
using *ld-nz* **unfolding** *delta-def delta-minus-def*
apply(*simp-all*)
apply(*simp-all add*: *c-eq-1*)
using *aaa in-aff ld-nz* **unfolding** *e′-aff-def e′-def delta-def delta-minus-def delta-plus-def*
apply(*simp-all add*: *t-expr nz t-nz divide-simps*)
apply(*simp-all add*: *algebra-simps power2-eq-square*[*symmetric*] *t-expr d-nz*)

unfolding *t-expr*[*symmetric*]
by *algebra+*

have *class-eq*: *gluing* ‘‘ {(*add* (*x*, *y*) (*x′*, *y′*), *l* + *l′*)} =
{(*add* (*x*, *y*) (*x′*, *y′*), *l* + *l′*), (τ (*add* (*x*, *y*) (*x′*, *y′*)), *l* + *l′* + *1*)}
using *add-nz add-closure gluing-class-2* **by** *auto*
have *class-proj*: *gluing* ‘‘ {(*add* (*x*, *y*) (*x′*, *y′*), *l* + *l′*)} ∈ *e-proj*
using *add-closure e-proj-aff* **by** *auto*

have *dom-eq*: {*proj-add* ((*x1*, *y1*), *i*) ((*x2*, *y2*), *j*) |*x1 y1 i x2 y2 j*.
((*x1*, *y1*), *i*) ∈ {((*x*, *y*), *l*), (τ (*x*, *y*), *l* + *1*)} ∧
((*x2*, *y2*), *j*) ∈ {((*x′*, *y′*), *l′*), (τ (*x′*, *y′*), *l′* + *1*)} ∧ ((*x1*, *y1*), *x2*, *y2*) ∈ *e′-aff-0* ∪ *e′-aff-1*} =
{(*add* (*x*, *y*) (*x′*, *y′*), *l* + *l′*), (τ (*add* (*x*, *y*) (*x′*, *y′*)), *l* + *l′* + *1*)}
(**is** *?s* = *?c*)
proof(*standard*)
show *?s* ⊆ *?c*
proof
fix *e*
assume *e* ∈ *?s*
then obtain *x1 y1 x2 y2 i j* **where**
e = *proj-add* ((*x1*, *y1*), *i*) ((*x2*, *y2*), *j*)
((*x1*, *y1*), *i*) ∈ {((*x*, *y*), *l*), (τ (*x*, *y*), *l* + *1*)}
((*x2*, *y2*), *j*) ∈ {((*x′*, *y′*), *l′*), (τ (*x′*, *y′*), *l′* + *1*)}
((*x1*, *y1*), *x2*, *y2*) ∈ *e′-aff-0* ∪ *e′-aff-1* **by** *blast*
then have *e* = (*add* (*x*, *y*) (*x′*, *y′*), *l* + *l′*) ∨
e = (τ (*add* (*x*, *y*) (*x′*, *y′*)), *l* + *l′* + *1*)
using *v1 v2 v3 v4 in-aff taus*(*1*,*2*)
aaa ds ds′ ld-nz **by** *fastforce*
then show *e* ∈ *?c* **by** *blast*
qed
next
show *?s* ⊇ *?c*
proof
fix *e*
assume *e* ∈ *?c*
then show *e* ∈ *?s*

using *v1 v3 in-aff taus*(*1*,*2*)
aaa ld-nz **unfolding** *e′-aff-0-def* **by** *force*
qed
qed

show *proj-addition ?p ?q* = *gluing* ‘‘ {(*add* (*x*, *y*) (*x′*, *y′*), *l* + *l′*)}
unfolding *proj-addition-def*
unfolding *proj-add-class.simps*(*1*)[*OF assms*(*3*,*4*)]
unfolding *assms*
using *v1 v2 v3 v4 in-aff taus*(*1*,*2*)
aaa ds ds′ ld-nz
apply(*subst dom-eq*)
apply(*subst class-eq*[*symmetric*])
apply(*subst eq-class-simp*)
using *class-proj class-eq* **by** *auto*
next
case *bbb*
from *bbb* **have** *v3*:
proj-add ((*x*, *y*), *l*) (*τ* (*x′*, *y′*), *l′* + *1*) = (*ext-add* (*x*, *y*) (*τ* (*x′*, *y′*)), *l*+*l′*+*1*)

using *proj-add.simps* ‹(*x*,*y*) ∈ *e′-aff*› ‹(*τ* (*x′*, *y′*)) ∈ *e′-aff*› **by** *simp*
have *pd*: *delta* (*fst* (*τ* (*x*, *y*))) (*snd* (*τ* (*x*, *y*))) *x′ y′* = *0*
using *bbb* **unfolding** *delta-def delta-plus-def delta-minus-def*
delta′-def delta-x-def delta-y-def
apply(*simp add*: *t-nz nz algebra-simps power2-eq-square*[*symmetric*] *t-expr d-nz*)
by(*simp add*: *divide-simps t-nz nz*)
have *pd′*: *delta′* (*fst* (*τ* (*x*, *y*))) (*snd* (*τ* (*x*, *y*))) *x′ y′* ≠ *0*
using *bbb* **unfolding** *delta′-def delta-x-def delta-y-def*
by(*simp add*: *t-nz nz divide-simps algebra-simps power2-eq-square*[*symmetric*] *t-expr d-nz*)
then have *pd″*: *delta′ x y* (*fst* (*τ* (*x′*, *y′*))) (*snd* (*τ* (*x′*, *y′*))) ≠ *0*
unfolding *delta′-def delta-x-def delta-y-def*
apply(*simp add*: *divide-simps t-nz nz*)
by *algebra*
have *v4*: *proj-add* (*τ* (*x*, *y*), *l*+*1*) ((*x′*, *y′*), *l′*) = (*ext-add* (*τ* (*x*, *y*)) (*x′*, *y′*), *l*+*l′*+*1*)
using *proj-add.simps in-aff taus pd pd′* **by** *simp*
have *v3-eq-v4*: (*ext-add* (*x*, *y*) (*τ* (*x′*, *y′*)), *l*+*l′*+*1*) = (*ext-add* (*τ* (*x*, *y*)) (*x′*, *y′*), *l*+*l′*+*1*)
using *inversion-invariance-2 nz* **by** *auto*

have *add-closure*: *ext-add* (*x*, *y*) (*τ* (*x′*, *y′*)) ∈ *e′-aff*
proof −
obtain *x1 y1* **where** *z2-d*: *τ* (*x′*, *y′*) = (*x1*,*y1*) **by** *fastforce*
define *z3* **where** *z3* = *ext-add* (*x*,*y*) (*x1*,*y1*)
obtain *x2 y2* **where** *z3-d*: *z3* = (*x2*,*y2*) **by** *fastforce*
have *d′*: *delta′ x y x1 y1* ≠ *0*
using *bbb z2-d* **by** *auto*

```
    have (x1,y1) ∈ e'-aff
      unfolding z2-d[symmetric]
      using ‹τ (x', y') ∈ e'-aff› by auto
    have e-eq: e' x y = 0 e' x1 y1 = 0
      using ‹(x,y) ∈ e'-aff› ‹(x1,y1) ∈ e'-aff› unfolding e'-aff-def by(auto)

    have e' x2 y2 = 0
      using z3-d z3-def ext-add-closure[OF d' e-eq, of x2 y2] by blast
    then show ?thesis
      unfolding e'-aff-def using e-e'-iff z3-d z3-def z2-d by simp
  qed

  have eq: x * y' + y * x' ≠ 0  y * y' ≠ x * x'
    using bbb unfolding delta'-def delta-x-def delta-y-def
    by(simp add: t-nz nz divide-simps)+

  have add-nz: fst(ext-add (x, y) (τ (x', y'))) ≠ 0
               snd(ext-add (x, y) (τ (x', y'))) ≠ 0
    apply(simp-all add: algebra-simps power2-eq-square[symmetric] t-expr)
    apply(simp-all add: divide-simps d-nz t-nz nz)
    apply(safe)
    using ld-nz eq unfolding delta-def delta-minus-def delta-plus-def
    unfolding t-expr[symmetric]
    by algebra+

    have trans-add: τ (add (x, y) (x', y')) = (ext-add (x, y) (τ (x', y')))
                    add (x, y) (x', y') = τ (ext-add (x, y) (τ (x', y')))
    proof -
      show τ (add (x, y) (x', y')) = (ext-add (x, y) (τ (x', y')))
       using add-ext-add-2 inversion-invariance-2 assms e-proj-elim-2 in-aff by
auto
      then show add (x, y) (x', y') = τ (ext-add (x, y) (τ (x', y')))
        using tau-idemp-point[of add (x, y) (x', y')] by argo
    qed

  have dom-eq: {proj-add ((x1, y1), i) ((x2, y2), j) |x1 y1 i x2 y2 j.
   ((x1, y1), i) ∈ {((x, y), l), (τ (x, y), l + 1)} ∧
    ((x2, y2), j) ∈ {((x', y'), l'), (τ (x', y'), l' + 1)} ∧ ((x1, y1), x2, y2) ∈
e'-aff-0 ∪ e'-aff-1} =
    {(add (x, y) (x', y'), l + l'), (τ (add (x, y) (x', y')), l + l' + 1)}
  (is ?s = ?c)
  proof(standard)
    show ?s ⊆ ?c
    proof
      fix e
      assume e ∈ ?s
      then obtain x1 y1 x2 y2 i j where
        e = proj-add ((x1, y1), i) ((x2, y2), j)
        ((x1, y1), i) ∈ {((x, y), l), (τ (x, y), l + 1)}
```

$((x2,\ y2),\ j) \in \{((x',\ y'),\ l'),\ (\tau\ (x',\ y'),\ l' + 1)\}$
$((x1,\ y1),\ x2,\ y2) \in e'\text{-}aff\text{-}0 \cup e'\text{-}aff\text{-}1$ **by** *blast*
then have $e = (add\ (x,\ y)\ (x',\ y'),\ l + l') \vee$
$e = (\tau\ (add\ (x,\ y)\ (x',\ y')),\ l + l' + 1)$
using *v1 v2 v3 v4 in-aff taus*(*1*,*2*)
bbb ds ld-nz
by (*metis empty-iff insert-iff trans-add*(*1*) *v3-eq-v4*)
then show $e \in ?c$ **by** *blast*
qed
next
show $?s \supseteq ?c$
proof
fix *e*
assume $e \in ?c$
then have $e = (add\ (x,\ y)\ (x',\ y'),\ l + l') \vee$
$e = (\tau\ (add\ (x,\ y)\ (x',\ y')),\ l + l' + 1)$ **by** *blast*
then show $e \in ?s$
apply(*elim disjE*)
using *v1 ld-nz in-aff* **unfolding** *e′-aff-0-def* **apply** *force*
thm *trans-add*
apply(*subst* (*asm*) *trans-add*)
using *v3 bbb in-aff taus* **unfolding** *e′-aff-1-def* **by** *force*
qed
qed

have *ext-eq*: *gluing* ‘‘ $\{(ext\text{-}add\ (x,\ y)\ (\tau\ (x',\ y')),\ l + l'+1)\} =$
$\{(ext\text{-}add\ (x,\ y)\ (\tau\ (x',\ y')),\ l + l'+1),\ (\tau\ (ext\text{-}add\ (x,\ y)\ (\tau\ (x',\ y'))),\ l + l')\}$
using *add-nz add-closure gluing-class-2* **by** *auto*
have *class-eq*: *gluing* ‘‘ $\{(add\ (x,\ y)\ (x',\ y'),\ l + l')\} =$
$\{(add\ (x,\ y)\ (x',\ y'),\ l + l'),\ (\tau\ (add\ (x,\ y)\ (x',\ y')),\ l + l' + 1)\}$
proof −
have *gluing* ‘‘ $\{(add\ (x,\ y)\ (x',\ y'),\ l + l')\} =$
gluing ‘‘ $\{(\tau\ (ext\text{-}add\ (x,\ y)\ (\tau\ (x',\ y'))),\ l + l')\}$
using *trans-add* **by** *argo*
also have ... = *gluing* ‘‘ $\{(ext\text{-}add\ (x,\ y)\ (\tau\ (x',\ y')),\ l + l'+1)\}$
using *gluing-inv add-nz add-closure* **by** *auto*
also have ... = $\{(ext\text{-}add\ (x,\ y)\ (\tau\ (x',\ y')),\ l + l'+1),\ (\tau\ (ext\text{-}add\ (x,\ y)\ (\tau\ (x',\ y'))),\ l + l')\}$
using *ext-eq* **by** *blast*
also have ... = $\{(add\ (x,\ y)\ (x',\ y'),\ l + l'),\ (\tau\ (add\ (x,\ y)\ (x',\ y')),\ l + l' + 1)\}$
using *trans-add* **by** *force*
finally show *?thesis* **by** *blast*
qed

have *ext-eq-proj*: *gluing* ‘‘ $\{(ext\text{-}add\ (x,\ y)\ (\tau\ (x',\ y')),\ l + l'+1)\} \in e\text{-}proj$
using *add-closure e-proj-aff* **by** *auto*
then have *class-proj*: *gluing* ‘‘ $\{(add\ (x,\ y)\ (x',\ y'),\ l + l')\} \in e\text{-}proj$

```
proof -
  have gluing `` {(add (x, y) (x', y'), l + l')} =
        gluing `` {(τ (ext-add (x, y) (τ (x', y'))), l + l')}
    using trans-add by argo
  also have ... = gluing `` {(ext-add (x, y) (τ (x', y')), l + l'+1)}
    using gluing-inv add-nz add-closure by auto
  finally show ?thesis using ext-eq-proj by argo
qed

show ?thesis
  unfolding proj-addition-def
  unfolding proj-add-class.simps(1)[OF assms(3,4)]
  unfolding assms
  using v1 v2 v3 v4 in-aff taus(1,2)
        bbb ds  ld-nz
  apply(subst dom-eq)
  apply(subst class-eq[symmetric])
  apply(subst eq-class-simp)
  using class-proj class-eq by auto
next
  case ccc
  then have v3: proj-add ((x, y), l) (τ (x', y'), l' + 1) = undefined by simp
  from ccc have ds': delta (fst (τ (x, y))) (snd (τ (x, y))) x' y' = 0
                delta' (fst (τ (x, y))) (snd (τ (x, y))) x' y' = 0
    unfolding delta-def delta-plus-def delta-minus-def
              delta'-def delta-x-def delta-y-def
  by(simp-all add: t-nz nz divide-simps algebra-simps power2-eq-square[symmetric]
t-expr d-nz)
  then have v4: proj-add (τ (x, y), l+1) ((x', y'), l') = undefined by simp

  have add-z: fst (add (x, y) (x', y')) = 0 ∨ snd (add (x, y) (x', y')) = 0
    using b ccc unfolding e'-aff-0-def
                          delta-def delta'-def delta-plus-def delta-minus-def
                          delta-x-def delta-y-def e'-aff-def e'-def
    apply(simp add: t-nz nz field-simps)
    apply(simp add: c-eq-1)
    by algebra

  have add-closure: add (x, y) (x', y') ∈ e'-aff
    using b(1) ‹(x,y) ∈ e'-aff› ‹(x',y') ∈ e'-aff› add-closure e-e'-iff
    unfolding e'-aff-0-def delta-def e'-aff-def by(simp del: add.simps,blast)
  have class-eq: gluing `` {(add (x, y) (x', y'), l + l')} = {(add (x, y) (x', y'),
l + l')}
    using add-z add-closure gluing-class-1 by simp
  have class-proj: gluing `` {(add (x, y) (x', y'), l + l')} ∈ e-proj
    using add-closure e-proj-aff by simp

  have dom-eq:
    {proj-add ((x1, y1), i) ((x2, y2), j) |x1 y1 i x2 y2 j.
```

$((x1, y1), i) \in \{((x, y), l), (\tau\ (x, y), l + 1)\} \wedge$
$((x2, y2), j) \in \{((x', y'), l'), (\tau\ (x', y'), l' + 1)\} \wedge ((x1, y1), x2, y2) \in$
$e'\text{-}aff\text{-}0 \cup e'\text{-}aff\text{-}1\} =$
$\{(add\ (x, y)\ (x', y'), l + l')\}$
(**is** $?s = ?c$)
proof(*standard*)
show $?s \subseteq ?c$
proof
fix e
assume $e \in ?s$
then obtain *x1 y1 x2 y2 i j* **where**
$e = proj\text{-}add\ ((x1, y1), i)\ ((x2, y2), j)$
$((x1, y1), i) \in \{((x, y), l), (\tau\ (x, y), l + 1)\}$
$((x2, y2), j) \in \{((x', y'), l'), (\tau\ (x', y'), l' + 1)\}$
$((x1, y1), x2, y2) \in e'\text{-}aff\text{-}0 \cup e'\text{-}aff\text{-}1$ **by** *blast*
then have $e = (add\ (x, y)\ (x', y'), l + l')$
using *v1 v2 v3 v4 in-aff taus*(*1*,*2*)
ld-nz ds ds′ ccc
unfolding *e′-aff-0-def e′-aff-1-def* **by** *auto*
then show $e \in ?c$ **by** *blast*
qed
next
show $?s \supseteq ?c$
proof
fix e
assume $e \in ?c$
then have $e = (add\ (x, y)\ (x', y'), l + l')$ **by** *blast*
then show $e \in ?s$
using *v1 ld-nz in-aff* **unfolding** *e′-aff-0-def* **by** *force*
qed
qed
show *?thesis*
unfolding *proj-addition-def*
unfolding *proj-add-class.simps*(*1*)[*OF assms*(*3*,*4*)]
unfolding *assms*
apply(*subst dom-eq*)
apply(*subst class-eq*[*symmetric*])
apply(*subst eq-class-simp*)
using *class-proj class-eq* **by** *auto*
qed
next
case *c*
have *False*
using *c assms* **unfolding** *e′-aff-1-def e′-aff-0-def* **by** *simp*
then show *?thesis* **by** *simp*
qed
qed

lemma *gluing-add*:

assumes *gluing* ‘‘ $\{((x1,y1),l)\} \in$ *e-proj gluing* ‘‘ $\{((x2,y2),j)\} \in$ *e-proj delta x1 y1 x2 y2* $\neq 0$
shows *proj-addition* (*gluing* ‘‘ $\{((x1,y1),l)\}$) (*gluing* ‘‘ $\{((x2,y2),j)\}$) =
(*gluing* ‘‘ $\{(add\ (x1,y1)\ (x2,y2),l+j)\}$)
proof −
have *p-q-expr*: (*gluing* ‘‘ $\{((x1,y1),l)\} = \{((x1,\ y1),\ l)\} \vee$ *gluing* ‘‘ $\{((x1,y1),l)\} = \{((x1,\ y1),\ l), (\tau\ (x1,\ y1),\ l + 1)\}$)
(*gluing* ‘‘ $\{((x2,y2),j)\} = \{((x2,\ y2),\ j)\} \vee$ *gluing* ‘‘ $\{((x2,y2),j)\} = \{((x2,\ y2),\ j), (\tau\ (x2,\ y2),\ j + 1)\}$)
using *assms*(*1*,*2*) *gluing-cases-explicit* **by** *auto*
then consider
(*1*) *gluing* ‘‘ $\{((x1,y1),l)\} = \{((x1,\ y1),\ l)\}$ *gluing* ‘‘ $\{((x2,y2),j)\} = \{((x2,\ y2),\ j)\}$ |
(*2*) *gluing* ‘‘ $\{((x1,y1),l)\} = \{((x1,\ y1),\ l)\}$ *gluing* ‘‘ $\{((x2,y2),j)\} = \{((x2,\ y2),\ j), (\tau\ (x2,\ y2),\ j + 1)\}$ |
(*3*) *gluing* ‘‘ $\{((x1,y1),l)\} = \{((x1,\ y1),\ l), (\tau\ (x1,\ y1),\ l + 1)\}$ *gluing* ‘‘ $\{((x2,y2),j)\} = \{((x2,\ y2),\ j)\}$ |
(*4*) *gluing* ‘‘ $\{((x1,y1),l)\} = \{((x1,\ y1),\ l), (\tau\ (x1,\ y1),\ l + 1)\}$ *gluing* ‘‘ $\{((x2,y2),j)\} = \{((x2,\ y2),\ j), (\tau\ (x2,\ y2),\ j + 1)\}$ **by** *argo*
then show *?thesis*
proof(*cases*)
case *1*
then show *?thesis* **using** *gluing-add-1 assms* **by** *presburger*
next
case *2* **then show** *?thesis* **using** *gluing-add-2 assms* **by** *presburger*
next
case *3* **then show** *?thesis*
proof −
have *pd*: *delta x2 y2 x1 y1* $\neq 0$
using *assms*(*3*) **unfolding** *delta-def delta-plus-def delta-minus-def*
by(*simp*,*algebra*)
have *add-com*: *add* (*x2*, *y2*) (*x1*, *y1*) = *add* (*x1*, *y1*) (*x2*, *y2*)
using *commutativity* **by** *simp*
have *proj-addition* (*gluing* ‘‘ $\{((x2,\ y2),\ j)\}$) (*gluing* ‘‘ $\{((x1,\ y1),\ l)\}$) =
gluing ‘‘ $\{(add\ (x1,\ y1)\ (x2,\ y2),\ j + l)\}$
using *gluing-add-2*[*OF 3*(*2*) *3*(*1*) *assms*(*2*) *assms*(*1*) *pd*] *add-com*
by *simp*
then show *?thesis*
using *proj-add-class-comm add.commute assms*
unfolding *proj-addition-def* **by** *metis*
qed
next
case *4* **then show** *?thesis* **using** *gluing-add-4 assms* **by** *presburger*
qed
qed

lemma *gluing-ext-add-1*:
assumes *gluing* ‘‘ $\{((x,y),l)\} = \{((x,\ y),\ l)\}$ *gluing* ‘‘ $\{((x',y'),l')\} = \{((x',\ y'),\ l')\}$

gluing " $\{((x,y),l)\} \in$ *e-proj* *gluing* " $\{((x',y'),l')\} \in$ *e-proj* *delta'* x y x' $y' \neq 0$
shows *proj-addition* (*gluing* " $\{((x,y),l)\}$) (*gluing* " $\{((x',y'),l')\}$) = (*gluing* " $\{($*ext-add* (x,y) (x',y'),$l+l')\}$)
proof −
have *in-aff*: $(x,y) \in e'$-*aff* $(x',y') \in e'$-*aff*
using *assms e-proj-eq e-class* **by** *blast+*
then have *zeros*: $x = 0 \vee y = 0$ $x' = 0 \vee y' = 0$
using *e-proj-elim-1 assms* **by** *presburger+*

have *ds*: *delta'* x y x' $y' = 0$ *delta'* x y x' $y' \neq 0$
using *delta'-def delta-x-def delta-y-def zeros*(1) *zeros*(2) **apply** *fastforce*
using *assms*(5) **by** *simp*
consider
(a) $(x, y) \in$ *e-circ* $\wedge$ ($\exists g \in$*symmetries*. $(x', y') = (g \circ i)$ (x, y)) |
(b) $((x, y), x', y') \in e'$-*aff-0* $\neg$ ($(x, y) \in$ *e-circ* $\wedge$ ($\exists g \in$*symmetries*. $(x', y') = (g \circ i)$ (x, y))) |
(c) $((x, y), x', y') \in e'$-*aff-1* $\neg$ ($(x, y) \in$ *e-circ* $\wedge$ ($\exists g \in$*symmetries*. $(x', y') = (g \circ i)$ (x, y))) $((x, y), x', y') \notin e'$-*aff-0*
using *dichotomy-1*[*OF* ‹$(x,y) \in e'$-*aff*› ‹$(x',y') \in e'$-*aff*›] **by** *argo*
then show *?thesis*
proof(*cases*)
case *a*
then have *False*
using *in-aff zeros* **unfolding** *e-circ-def* **by** *force*
then show *?thesis* **by** *simp*
next
case *b*
from *ds* **show** *?thesis* **by** *simp*
next
case *c*
from *ds* **show** *?thesis* **by** *simp*
qed
qed

lemma *gluing-ext-add-2*:
assumes *gluing* " $\{((x,y),l)\} = \{((x, y), l)\}$ *gluing* " $\{((x',y'),l')\} = \{((x', y'), l'), (\tau\ (x', y'), l' + 1)\}$
gluing " $\{((x,y),l)\} \in$ *e-proj* *gluing* " $\{((x',y'),l')\} \in$ *e-proj* *delta'* x y x' $y' \neq 0$
shows *proj-addition* (*gluing* " $\{((x,y),l)\}$) (*gluing* " $\{((x',y'),l')\}$) = (*gluing* " $\{($*ext-add* (x,y) (x',y'),$l+l')\}$)
proof −
have *in-aff*: $(x,y) \in e'$-*aff* $(x',y') \in e'$-*aff*
using *assms e-proj-eq e-class* **by** *blast+*
then have *add-in*: *ext-add* (x, y) $(x', y') \in e'$-*aff*
using *ext-add-closure* ‹*delta'* x y x' $y' \neq 0$› *delta-def e-e'-iff e'-aff-def* **by** *auto*
from *in-aff* **have** *zeros*: $x = 0 \vee y = 0$ $x' \neq 0$ $y' \neq 0$

using *e-proj-elim-1 e-proj-elim-2 assms* **by** *presburger+*
have *e-proj*: $gluing\ `` \{((x,y),l)\} \in e\text{-}proj$
$gluing\ `` \{((x',y'),l')\} \in e\text{-}proj$
$gluing\ `` \{(ext\text{-}add\ (x,\ y)\ (x',\ y'),\ l + l')\} \in e\text{-}proj$
using *e-proj-aff in-aff add-in* **by** *auto*

consider
$(a)\ (x,\ y) \in e\text{-}circ \wedge (\exists g\in symmetries.\ (x',\ y') = (g \circ i)\ (x,\ y))\ |$
$(b)\ ((x,\ y),\ x',\ y') \in e'\text{-}aff\text{-}0\ \neg\ ((x,\ y) \in e\text{-}circ \wedge (\exists g\in symmetries.\ (x',\ y') = (g \circ i)\ (x,\ y)))\ ((x,\ y),\ x',\ y') \notin e'\text{-}aff\text{-}1\ |$
$(c)\ ((x,\ y),\ x',\ y') \in e'\text{-}aff\text{-}1\ \neg\ ((x,\ y) \in e\text{-}circ \wedge (\exists g\in symmetries.\ (x',\ y') = (g \circ i)\ (x,\ y)))$
using *dichotomy-1*[*OF* ‹$(x,y) \in e'\text{-}aff$› ‹$(x',y') \in e'\text{-}aff$›] **by** *fast*
then show *?thesis*
proof(*cases*)
case *a*
then have *False*
using *in-aff zeros* **unfolding** *e-circ-def* **by** *force*
then show *?thesis* **by** *simp*
next
case *b*
have *ld-nz*: $delta'\ x\ y\ x'\ y' = 0$
using ‹$(x,y) \in e'\text{-}aff$› ‹$(x',y') \in e'\text{-}aff$› *b*
unfolding *e′-aff-1-def* **by** *force*
then have *False*
using *assms e-proj-elim-1 in-aff*
unfolding *delta-def delta-minus-def delta-plus-def* **by** *blast*
then show *?thesis* **by** *blast*
next
case *c*
then have *ld-nz*: $delta'\ x\ y\ x'\ y' \neq 0$ **unfolding** *e′-aff-1-def* **by** *auto*

have *v1*: $proj\text{-}add\ ((x,\ y),\ l)\ ((x',\ y'),\ l') = (ext\text{-}add\ (x,\ y)\ (x',\ y'),\ l + l')$
by(*simp add*: ‹$(x,y) \in e'\text{-}aff$› ‹$(x',y') \in e'\text{-}aff$› *ld-nz del*: *add.simps*)

have *ecirc*: $(x',y') \in e\text{-}circ\ x' \neq 0\ y' \neq 0$
unfolding *e-circ-def* **using** *zeros* ‹$(x',y') \in e'\text{-}aff$› **by** *blast+*
then have $\tau\ (x',\ y') \in e\text{-}circ$
using *zeros* τ*-circ* **by** *blast*
then have *in-aff′*: $\tau\ (x',\ y') \in e'\text{-}aff$
unfolding *e-circ-def* **by** *force*

have *add-nz*: $fst\ (ext\text{-}add\ (x,\ y)\ (x',\ y')) \neq 0$
$snd\ (ext\text{-}add\ (x,\ y)\ (x',\ y')) \neq 0$
using *zeros ld-nz in-aff*
unfolding *delta-def delta-plus-def delta-minus-def e′-aff-def e′-def*
apply(*simp-all*)
by *auto*

have *add-in*: *ext-add* (x, y) $(x', y') \in e'$-*aff*
using *ext-add-closure in-aff e-e'-iff ld-nz* **unfolding** *e'-aff-def delta-def* **by** *simp*

have *ld-nz'*: *delta'* x y (*fst* $(\tau$ $(x',y')))$ (*snd* $(\tau$ $(x',y'))) \neq 0$
using *ld-nz*
unfolding *delta'-def delta-x-def delta-y-def*
using *zeros* **by**(*auto simp add*: *divide-simps t-nz*)

have *tau-conv*: τ (*ext-add* (x, y) $(x', y')) =$ *ext-add* (x, y) $(\tau$ $(x', y'))$
using *zeros e'-aff-x0*[*OF - in-aff*(*1*)] *e'-aff-y0*[*OF - in-aff*(*1*)]
apply(*simp-all*)
apply(*simp-all add*: *c-eq-1 divide-simps d-nz t-nz*)
apply(*elim disjE*)
apply(*simp-all add*: *t-nz zeros*)
by *auto*

have *v2*: *proj-add* $((x, y), l)$ $(\tau$ $(x', y'), l' + 1) = (\tau$ (*ext-add* (x, y) $(x', y')), l+l'+1)$
using *proj-add.simps* ‹τ $(x', y') \in e'$-*aff*› *in-aff tau-conv*
‹*delta'* x y (*fst* $(\tau$ $(x', y')))$ (*snd* $(\tau$ $(x', y'))) \neq 0$› **by** *auto*

have *gl-class*: *gluing* `` $\{($*ext-add* (x, y) $(x', y'), l + l')\} =$
$\{($*ext-add* (x, y) $(x', y'), l + l'), (\tau$ (*ext-add* (x, y) $(x', y')), l + l' + 1)\}$
gluing `` $\{($*ext-add* (x, y) $(x', y'), l + l')\} \in$ *e-proj*
using *gluing-class-2 e-points add-nz add-in* **apply** *simp*
using *e-points add-nz add-in* **by** *force*

show *?thesis*
proof −
have $\{$*proj-add* $((x1, y1), i)$ $((x2, y2), j)$ $|x1$ $y1$ i $x2$ $y2$ $j.$
$((x1, y1), i) \in \{((x, y), l)\} \wedge$
$((x2, y2), j) \in \{((x', y'), l'), (\tau$ $(x', y'), l' + 1)\} \wedge$
$((x1, y1), x2, y2)$
$\in e'$-*aff-0* $\cup$ $\{((x1, y1), x2, y2).$ $(x1, y1) \in e'$-*aff* $\wedge$ $(x2, y2) \in e'$-*aff* $\wedge$ *delta'* $x1$ $y1$ $x2$ $y2 \neq 0\}\} =$
$\{$*proj-add* $((x, y), l)$ $((x', y'), l')$, *proj-add* $((x, y), l)$ $(\tau$ $(x', y'), l' + 1)\}$
(**is** *?t* = -)
using *ld-nz ld-nz' in-aff in-aff'*
apply(*simp del*: *τ.simps add.simps*)
by *force*
also have ... = $\{($*ext-add* (x, y) $(x', y'), l + l'), (\tau$ (*ext-add* (x, y) $(x', y')), l + l' + 1)\}$
using *v1 v2* **by** *presburger*
finally have *eq*: *?t* = $\{($*ext-add* (x, y) $(x', y'), l + l'), (\tau$ (*ext-add* (x, y) $(x', y')), l + l' + 1)\}$
by *blast*

show *?thesis*
unfolding *proj-addition-def*
unfolding *proj-add-class.simps(1)[OF e-proj(1,2)]*
unfolding *assms(1,2) gl-class e′-aff-1-def*
apply(*subst eq*)
apply(*subst eq-class-simp*)
using *gl-class* **by** *auto*
qed
qed
qed

lemma *gluing-ext-add-4*:
assumes *gluing* ‘‘ {((*x*,*y*),*l*)} = {((*x*, *y*), *l*), (*τ* (*x*, *y*), *l* + *1*)} *gluing* ‘‘ {((*x′*,*y′*),*l′*)} = {((*x′*, *y′*), *l′*), (*τ* (*x′*, *y′*), *l′* + *1*)}
gluing ‘‘ {((*x*,*y*),*l*)} ∈ *e-proj gluing* ‘‘ {((*x′*,*y′*),*l′*)} ∈ *e-proj delta′ x y x′ y′* ≠ *0*
shows *proj-addition* (*gluing* ‘‘ {((*x*,*y*),*l*)}) (*gluing* ‘‘ {((*x′*,*y′*),*l′*)}) = (*gluing* ‘‘ {(*ext-add* (*x*,*y*) (*x′*,*y′*),*l*+*l′*)})
(**is** *proj-addition ?p ?q* = -)
proof −
have *in-aff*: (*x*,*y*) ∈ *e′-aff* (*x′*,*y′*) ∈ *e′-aff*
using *e-proj-aff assms* **by** *meson+*
then have *nz*: *x* ≠ *0 y* ≠ *0 x′* ≠ *0 y′* ≠ *0*
using *assms e-proj-elim-2* **by** *auto*
then have *circ*: (*x*,*y*) ∈ *e-circ* (*x′*,*y′*) ∈ *e-circ*
using *in-aff e-circ-def nz* **by** *auto*
then have *taus*: (*τ* (*x′*, *y′*)) ∈ *e′-aff* (*τ* (*x*, *y*)) ∈ *e′-aff τ* (*x′*, *y′*) ∈ *e-circ*
using *τ-circ circ-to-aff* **by** *auto*

consider
(*a*) (*x*, *y*) ∈ *e-circ* ∧ (∃ *g*∈*symmetries*. (*x′*, *y′*) = (*g* ∘ *i*) (*x*, *y*))
| (*b*) ((*x*, *y*), *x′*, *y′*) ∈ *e′-aff-0* ((*x*, *y*), *x′*, *y′*) ∉ *e′-aff-1*
| (*c*) ((*x*, *y*), *x′*, *y′*) ∈ *e′-aff-1*
using *dichotomy-1[OF in-aff]* **by** *auto*
then show *?thesis*
proof(*cases*)
case *a*
then obtain *g* **where** *sym-expr*: *g* ∈ *symmetries* (*x′*, *y′*) = (*g* ∘ *i*) (*x*, *y*) **by** *auto*
then have *ds*: *delta x y x′ y′* = *0 delta′ x y x′ y′* = *0*
using *wd-d-nz wd-d′-nz a* **by** *auto*
then have *False*
using *assms* **by** *auto*
then show *?thesis* **by** *blast*
next
case *b*
have *False*
using *b assms* **unfolding** *e′-aff-1-def e′-aff-0-def* **by** *simp*

then show *?thesis* by *simp*
next
case *c*
then have *ld-nz*: $delta'\ x\ y\ x'\ y' \neq 0$
unfolding *e'-aff-1-def* by *auto*
then have *ds*: $delta'\ (fst\ (\tau\ (x,\ y)))\ (snd\ (\tau\ (x,\ y)))\ (fst\ (\tau\ (x',\ y')))\ (snd\ (\tau\ (x',\ y'))) \neq 0$
unfolding *delta'-def delta-x-def delta-y-def*

by(*simp add*: *t-nz field-simps nz*)

have *v1*: $proj\text{-}add\ ((x,\ y),\ l)\ ((x',\ y'),\ l') = (ext\text{-}add\ (x,\ y)\ (x',\ y'),\ l + l')$
using *ld-nz proj-add.simps* ‹$(x,y) \in e'\text{-}aff$› ‹$(x',y') \in e'\text{-}aff$› by *simp*
have *v2*: $proj\text{-}add\ (\tau\ (x,\ y),\ l{+}1)\ (\tau\ (x',\ y'),\ l'{+}1) = (ext\text{-}add\ (x,\ y)\ (x',\ y'),\ l + l')$
apply(*subst proj-add.simps*(*2*)[*OF ds*,*simplified prod.collapse taus*(*2*) *taus*(*1*)])
apply *simp*
apply(*simp del*: *ext-add.simps* τ*.simps*)
apply(*rule inversion-invariance-2*[*OF nz*(*1*,*2*), *of fst* $(\tau\ (x',y'))$ *snd* $(\tau\ (x',y'))$,
simplified prod.collapse tau-idemp-point])
using *nz t-nz* by *auto*

consider (*aaa*) $delta'\ x\ y\ (fst\ (\tau\ (x',\ y')))\ (snd\ (\tau\ (x',\ y'))) \neq 0$ |
(*bbb*) $delta\ x\ y\ (fst\ (\tau\ (x',\ y')))\ (snd\ (\tau\ (x',\ y'))) \neq 0$
$delta'\ x\ y\ (fst\ (\tau\ (x',\ y')))\ (snd\ (\tau\ (x',\ y'))) = 0$ |
(*ccc*) $delta'\ x\ y\ (fst\ (\tau\ (x',\ y')))\ (snd\ (\tau\ (x',\ y'))) = 0$
$delta\ x\ y\ (fst\ (\tau\ (x',\ y')))\ (snd\ (\tau\ (x',\ y'))) = 0$ by *blast*
then show *?thesis*
proof(*cases*)
case *aaa*
have *tau-conv*: $\tau\ (ext\text{-}add\ (x,\ y)\ (\tau\ (x',\ y'))) = ext\text{-}add\ (x,y)\ (x',y')$
apply(*simp*)
using *aaa in-aff ld-nz*
unfolding *e'-aff-def e'-def delta'-def delta-x-def delta-y-def*
apply(*safe*)
apply(*simp-all add*: *divide-simps t-nz nz*)
by *algebra+*

have *v3*:
$proj\text{-}add\ ((x,\ y),\ l)\ (\tau\ (x',\ y'),\ l' + 1) = (\tau\ (ext\text{-}add\ (x,\ y)\ (x',\ y')),\ l{+}l'{+}1)$

using *proj-add.simps* ‹$(\tau\ (x',\ y')) \in e'\text{-}aff$›
apply(*simp del*: *ext-add.simps* τ*.simps*)
using *tau-conv tau-idemp-explicit*
proj-add.simps(*2*)[*OF aaa* ‹$(x,y) \in e'\text{-}aff$›,*simplified prod.collapse*,*OF* ‹$(\tau\ (x',\ y')) \in e'\text{-}aff$›]
by (*metis* (*no-types*, *lifting*) *add.assoc prod.collapse*)

have *ds'*: $delta'\ (fst\ (\tau\ (x,\ y)))\ (snd\ (\tau\ (x,\ y)))\ x'\ y' \neq 0$

using *aaa* **unfolding** *delta′-def delta-x-def delta-y-def*
by(*simp add*: *divide-simps t-nz nz algebra-simps power2-eq-square*[*symmetric*] *t-expr d-nz*)

have *v4*: *proj-add* (τ (*x*, *y*), *l*+*1*) ((*x′*, *y′*), *l′*) = (τ (*ext-add* (*x*, *y*) (*x′*, *y′*)), *l*+*l′*+*1*)
proof −
have *proj-add* (τ (*x*, *y*), *l*+*1*) ((*x′*, *y′*), *l′*) = (*ext-add* (τ (*x*, *y*)) (*x′*, *y′*), *l*+*l′*+*1*)
using *proj-add.simps* ‹τ (*x*,*y*) ∈ *e′-aff*› ‹(*x′*, *y′*) ∈ *e′-aff*› *ds′* **by** *auto*
moreover have *ext-add* (τ (*x*, *y*)) (*x′*, *y′*) = τ (*ext-add* (*x*, *y*) (*x′*, *y′*))
by (*metis inversion-invariance-2 nz tau-conv tau-idemp-point*)
ultimately show *?thesis* **by** *argo*
qed

have *add-closure*: *ext-add* (*x*,*y*) (*x′*,*y′*) ∈ *e′-aff*
using *in-aff ext-add-closure ld-nz e-e′-iff* **unfolding** *delta′-def e′-aff-def* **by** *auto*

have *add-nz*: *fst* (*ext-add* (*x*,*y*) (*x′*,*y′*)) ≠ *0*
snd (*ext-add* (*x*,*y*) (*x′*,*y′*)) ≠ *0*
using *ld-nz* **unfolding** *delta-def delta-minus-def*
apply(*simp-all*)
using *aaa in-aff ld-nz* **unfolding** *e′-aff-def e′-def delta′-def delta-x-def delta-y-def*
apply(*simp-all add*: *t-expr nz t-nz divide-simps*)
apply(*simp-all add*: *algebra-simps power2-eq-square*[*symmetric*] *t-expr d-nz*)

by *algebra*+

have *class-eq*: *gluing* `` {(*ext-add* (*x*, *y*) (*x′*, *y′*), *l* + *l′*)} =
{(*ext-add* (*x*, *y*) (*x′*, *y′*), *l* + *l′*), (τ (*ext-add* (*x*, *y*) (*x′*, *y′*)), *l* + *l′* + *1*)}
using *add-nz add-closure gluing-class-2* **by** *auto*
have *class-proj*: *gluing* `` {(*ext-add* (*x*, *y*) (*x′*, *y′*), *l* + *l′*)} ∈ *e-proj*
using *add-closure e-proj-aff* **by** *auto*

have *dom-eq*: {*proj-add* ((*x1*, *y1*), *i*) ((*x2*, *y2*), *j*) |*x1 y1 i x2 y2 j*.
((*x1*, *y1*), *i*) ∈ {((*x*, *y*), *l*), (τ (*x*, *y*), *l* + *1*)} ∧
((*x2*, *y2*), *j*) ∈ {((*x′*, *y′*), *l′*), (τ (*x′*, *y′*), *l′* + *1*)} ∧ ((*x1*, *y1*), *x2*, *y2*) ∈ *e′-aff-0* ∪ *e′-aff-1*} =
{(*ext-add* (*x*, *y*) (*x′*, *y′*), *l* + *l′*), (τ (*ext-add* (*x*, *y*) (*x′*, *y′*)), *l* + *l′* + *1*)}

(**is** *?s* = *?c*)
proof(*standard*)
show *?s* ⊆ *?c*
proof
fix *e*
assume *e* ∈ *?s*

```
        then obtain x1 y1 x2 y2 i j where
          e = proj-add ((x1, y1), i) ((x2, y2), j)
          ((x1, y1), i) ∈ {((x, y), l), (τ (x, y), l + 1)}
          ((x2, y2), j) ∈ {((x', y'), l'), (τ (x', y'), l' + 1)}
          ((x1, y1), x2, y2) ∈ e'-aff-0 ∪ e'-aff-1 by blast
        then have e = (ext-add (x, y) (x', y'), l + l') ∨
                  e = (τ (ext-add (x, y) (x', y')), l + l' + 1)
          using v1 v2 v3 v4 in-aff taus(1,2)
              aaa ds ds' ld-nz by fastforce
        then show e ∈ ?c by blast
      qed
    next
      show ?s ⊇ ?c
      proof
        fix e
        assume e ∈ ?c
        then show e ∈ ?s
          using v1 v3 in-aff taus(1,2)
              aaa ld-nz unfolding e'-aff-1-def by force
      qed
    qed

    show proj-addition ?p ?q = gluing `` {(ext-add (x, y) (x', y'), l + l')}
      unfolding proj-addition-def
      unfolding proj-add-class.simps(1)[OF assms(3,4)]
      unfolding assms
      using v1 v2 v3 v4 in-aff taus(1,2)
            aaa ds ds' ld-nz
      apply(subst dom-eq)
      apply(subst class-eq[symmetric])
      apply(subst eq-class-simp)
      using class-proj class-eq by auto
  next
    case bbb
    from bbb have v3:
      proj-add ((x, y), l) (τ (x', y'), l' + 1) = (add (x, y) (τ (x', y')), l+l'+1)
      using proj-add.simps ‹(x,y) ∈ e'-aff› ‹(τ (x', y')) ∈ e'-aff› by simp
    have pd: delta' (fst (τ (x, y))) (snd (τ (x, y))) x' y' = 0
      using bbb unfolding delta-def delta-plus-def delta-minus-def
                      delta'-def delta-x-def delta-y-def
      apply(simp add: divide-simps t-nz nz)
       apply(simp add: t-nz nz algebra-simps power2-eq-square[symmetric] t-expr
d-nz)
      by presburger
    have pd': delta (fst (τ (x, y))) (snd (τ (x, y))) x' y' ≠ 0
      using bbb unfolding delta'-def delta-x-def delta-y-def
                      delta-def delta-plus-def delta-minus-def
      by(simp add: t-nz nz field-simps power2-eq-square[symmetric] t-expr d-nz)
    then have pd'': delta x y (fst (τ (x', y'))) (snd (τ (x', y'))) ≠ 0
```

```
      unfolding delta-def delta-plus-def delta-minus-def
    by(simp add: divide-simps t-nz nz algebra-simps t-expr power2-eq-square[symmetric]
d-nz)
     have v4: proj-add (τ (x, y), l+1) ((x', y'), l') = (add (τ (x, y)) (x', y'),
l+l'+1)
      using proj-add.simps in-aff taus pd pd' by auto
     have v3-eq-v4: (add (x, y) (τ (x', y')), l+l'+1) = (add (τ (x, y)) (x', y'),
l+l'+1)
      using inversion-invariance-1 nz by auto

    have add-closure: add (x, y) (τ (x', y')) ∈ e'-aff
    proof -
      obtain x1 y1 where z2-d: τ (x', y') = (x1,y1) by fastforce
      define z3 where z3 = add (x,y) (x1,y1)
      obtain x2 y2 where z3-d: z3 = (x2,y2) by fastforce
      have d': delta x y x1 y1 ≠ 0
        using bbb z2-d by auto
      have (x1,y1) ∈ e'-aff
        unfolding z2-d[symmetric]
        using ‹τ (x', y') ∈ e'-aff› by auto
      have e-eq: e' x y = 0 e' x1 y1 = 0
        using ‹(x,y) ∈ e'-aff› ‹(x1,y1) ∈ e'-aff› unfolding e'-aff-def by(auto)

      have e' x2 y2 = 0
        using d' add-closure[OF z3-d z3-def] e-e'-iff e-eq unfolding delta-def by
auto
      then show ?thesis
        unfolding e'-aff-def using e-e'-iff z3-d z3-def z2-d by simp
    qed

    have add-nz: fst(add (x, y) (τ (x', y'))) ≠ 0
                 snd(add (x, y) (τ (x', y'))) ≠ 0
      apply(simp-all add: algebra-simps power2-eq-square[symmetric] t-expr)
      apply(simp-all add: divide-simps d-nz t-nz nz c-eq-1)
       apply(safe)
      using bbb ld-nz unfolding delta'-def delta-x-def delta-y-def
                          delta-def delta-plus-def delta-minus-def
      by(simp-all add: divide-simps t-nz nz algebra-simps
                          power2-eq-square[symmetric] t-expr d-nz)


      have trans-add: τ (ext-add (x, y) (x', y')) = (add (x, y) (τ (x', y')))
                      ext-add (x, y) (x', y') = τ (add (x, y) (τ (x', y')))
      proof -
        show τ (ext-add (x, y) (x', y')) = (add (x, y) (τ (x', y')))
           using inversion-invariance-1 assms add-ext-add nz tau-idemp-point by
presburger
        then show ext-add (x, y) (x', y') = τ (add (x, y) (τ (x', y')))
          using tau-idemp-point[of ext-add (x, y) (x', y')] by argo
```

qed

have *dom-eq*: $\{$*proj-add* $((x1, y1), i)$ $((x2, y2), j)$ $|x1\ y1\ i\ x2\ y2\ j.$
$((x1, y1), i) \in \{((x, y), l), (\tau\ (x, y), l + 1)\} \wedge$
$((x2, y2), j) \in \{((x', y'), l'), (\tau\ (x', y'), l' + 1)\} \wedge ((x1, y1), x2, y2) \in$ *e′-aff-0* $\cup$ *e′-aff-1*$\} =$
$\{$(*ext-add* (x, y) (x', y'), $l + l'$), (τ (*ext-add* (x, y) (x', y')), $l + l' + 1$)$\}$
(**is** $?s = ?c$)
proof(*standard*)
show $?s \subseteq ?c$
proof
fix *e*
assume $e \in ?s$
then obtain *x1 y1 x2 y2 i j* **where**
$e =$ *proj-add* $((x1, y1), i)$ $((x2, y2), j)$
$((x1, y1), i) \in \{((x, y), l), (\tau\ (x, y), l + 1)\}$
$((x2, y2), j) \in \{((x', y'), l'), (\tau\ (x', y'), l' + 1)\}$
$((x1, y1), x2, y2) \in$ *e′-aff-0* $\cup$ *e′-aff-1* **by** *blast*
then have $e =$ (*ext-add* (x, y) (x', y'), $l + l'$) $\vee$
$e =$ (τ (*ext-add* (x, y) (x', y')), $l + l' + 1$)
using *v1 v2 v3 v4 in-aff taus*(*1*,*2*)
bbb ds ld-nz
by (*metis empty-iff insert-iff trans-add*(*1*) *v3-eq-v4*)
then show $e \in ?c$ **by** *blast*
qed
next
show $?s \supseteq ?c$
proof
fix *e*
assume $e \in ?c$
then have $e =$ (*ext-add* (x, y) (x', y'), $l + l'$) $\vee$
$e =$ (τ (*ext-add* (x, y) (x', y')), $l + l' + 1$) **by** *blast*
then show $e \in ?s$
apply(*elim disjE*)
using *v1 ld-nz in-aff* **unfolding** *e′-aff-1-def* **apply** *force*
apply(*subst* (*asm*) *trans-add*)
using *v3 bbb in-aff taus* **unfolding** *e′-aff-0-def* **by** *force*
qed
qed

have *ext-eq*: *gluing* ‘‘ $\{$(*add* (x, y) (τ (x', y')), $l + l'+1$)$\} =$
$\{$(*add* (x, y) (τ (x', y')), $l + l'+1$), (τ (*add* (x, y) (τ (x', y'))), $l + l'$)$\}$
using *add-nz add-closure gluing-class-2* **by** *auto*
have *class-eq*: *gluing* ‘‘ $\{$(*ext-add* (x, y) (x', y'), $l + l'$)$\} =$
$\{$(*ext-add* (x, y) (x', y'), $l + l'$), (τ (*ext-add* (x, y) (x', y')), $l + l' + 1$)$\}$
proof −
have *gluing* ‘‘ $\{$(*ext-add* (x, y) (x', y'), $l + l'$)$\} =$
gluing ‘‘ $\{$(τ (*add* (x, y) (τ (x', y'))), $l + l'$)$\}$

using *trans-add* **by** *argo*
also have ... = *gluing* ‘‘ {(*add* (*x*, *y*) (τ (*x*′, *y*′)), *l* + *l*′+*1*)}
using *gluing-inv add-nz add-closure* **by** *auto*
also have ... = {(*add* (*x*, *y*) (τ (*x*′, *y*′)), *l* + *l*′+*1*), (τ (*add* (*x*, *y*) (τ (*x*′, *y*′))), *l* + *l*′)}
using *ext-eq* **by** *blast*
also have ... = {(*ext-add* (*x*, *y*) (*x*′, *y*′), *l* + *l*′), (τ (*ext-add* (*x*, *y*) (*x*′, *y*′)), *l* + *l*′ + *1*)}
using *trans-add* **by** *force*
finally show *?thesis* **by** *blast*
qed

have *ext-eq-proj*: *gluing* ‘‘ {(*add* (*x*, *y*) (τ (*x*′, *y*′)), *l* + *l*′+*1*)} ∈ *e-proj*
using *add-closure e-proj-aff* **by** *auto*
then have *class-proj*: *gluing* ‘‘ {(*ext-add* (*x*, *y*) (*x*′, *y*′), *l* + *l*′)} ∈ *e-proj*
proof −
have *gluing* ‘‘ {(*ext-add* (*x*, *y*) (*x*′, *y*′), *l* + *l*′)} =
gluing ‘‘ {(τ (*add* (*x*, *y*) (τ (*x*′, *y*′))), *l* + *l*′)}
using *trans-add* **by** *argo*
also have ... = *gluing* ‘‘ {(*add* (*x*, *y*) (τ (*x*′, *y*′)), *l* + *l*′+*1*)}
using *gluing-inv add-nz add-closure* **by** *auto*
finally show *?thesis* **using** *ext-eq-proj* **by** *argo*
qed

show *?thesis*
unfolding *proj-addition-def*
unfolding *proj-add-class.simps*(*1*)[*OF assms*(*3*,*4*)]
unfolding *assms*
using *v1 v2 v3 v4 in-aff taus*(*1*,*2*)
bbb ds ld-nz
apply(*subst dom-eq*)
apply(*subst class-eq*[*symmetric*])
apply(*subst eq-class-simp*)
using *class-proj class-eq* **by** *auto*
next
case *ccc*
then have *v3*: *proj-add* ((*x*, *y*), *l*) (τ (*x*′, *y*′), *l*′ + *1*) = *undefined* **by** *simp*
from *ccc* **have** *ds*′: *delta* (*fst* (τ (*x*, *y*))) (*snd* (τ (*x*, *y*))) *x*′ *y*′ = *0*
delta′ (*fst* (τ (*x*, *y*))) (*snd* (τ (*x*, *y*))) *x*′ *y*′ = *0*
unfolding *delta-def delta-plus-def delta-minus-def*
delta′*-def delta-x-def delta-y-def*
by(*simp-all add*: *t-nz nz divide-simps algebra-simps power2-eq-square*[*symmetric*] *t-expr d-nz*)
then have *v4*: *proj-add* (τ (*x*, *y*), *l*+*1*) ((*x*′, *y*′), *l*′) = *undefined* **by** *simp*

have *add-z*: *fst* (*ext-add* (*x*, *y*) (*x*′, *y*′)) = *0* ∨ *snd* (*ext-add* (*x*, *y*) (*x*′, *y*′)) = *0*
using *c ccc ld-nz* **unfolding** *e*′*-aff-0-def*
delta-def delta′*-def delta-plus-def delta-minus-def*

```
                       delta-x-def delta-y-def e'-aff-def e'-def
    apply(simp-all add: field-simps t-nz nz)
    unfolding t-expr[symmetric] power2-eq-square
    apply(simp-all add: divide-simps d-nz t-nz)
    by algebra

  have add-closure: ext-add (x, y) (x', y') ∈ e'-aff
    using c(1) ‹(x,y) ∈ e'-aff› ‹(x',y') ∈ e'-aff› ext-add-closure e-e'-iff
    unfolding e'-aff-1-def delta-def e'-aff-def by simp
  have class-eq: gluing `` {(ext-add (x, y) (x', y'), l + l')} = {(ext-add (x, y)
(x', y'), l + l')}
    using add-z add-closure gluing-class-1 by simp
  have class-proj: gluing `` {(ext-add (x, y) (x', y'), l + l')} ∈ e-proj
    using add-closure e-proj-aff by simp

  have dom-eq:
    {proj-add ((x1, y1), i) ((x2, y2), j) |x1 y1 i x2 y2 j.
   ((x1, y1), i) ∈ {((x, y), l), (τ (x, y), l + 1)} ∧
    ((x2, y2), j) ∈ {((x', y'), l'), (τ (x', y'), l' + 1)} ∧ ((x1, y1), x2, y2) ∈
e'-aff-0 ∪ e'-aff-1} =
     {(ext-add (x, y) (x', y'), l + l')}
    (is ?s = ?c)
  proof(standard)
    show ?s ⊆ ?c
    proof
      fix e
      assume e ∈ ?s
      then obtain x1 y1 x2 y2 i j where
        e = proj-add ((x1, y1), i) ((x2, y2), j)
        ((x1, y1), i) ∈ {((x, y), l), (τ (x, y), l + 1)}
        ((x2, y2), j) ∈ {((x', y'), l'), (τ (x', y'), l' + 1)}
        ((x1, y1), x2, y2) ∈ e'-aff-0 ∪ e'-aff-1 by blast
      then have e = (ext-add (x, y) (x', y'), l + l')
        using v1 v2 v3 v4 in-aff taus(1,2)
              ld-nz ds ds' ccc
        unfolding e'-aff-0-def e'-aff-1-def
        by fastforce
      then show e ∈ ?c by blast
    qed
  next
    show ?s ⊇ ?c
    proof
      fix e
      assume e ∈ ?c
      then have e = (ext-add (x, y) (x', y'), l + l') by blast
      then show e ∈ ?s
        using v1 ld-nz in-aff unfolding e'-aff-1-def by force
    qed
  qed
```

show *?thesis*
  unfolding *proj-addition-def*
  unfolding *proj-add-class.simps(1)[OF assms(3,4)]*
  unfolding *assms*
  apply(*subst dom-eq*)
  apply(*subst class-eq[symmetric]*)
  apply(*subst eq-class-simp*)
  using *class-proj class-eq* **by** *auto*
qed
qed
qed

lemma *gluing-ext-add*:
assumes *gluing* ‘‘ $\{((x1,y1),l)\} \in$ *e-proj gluing* ‘‘ $\{((x2,y2),j)\} \in$ *e-proj delta′ x1 y1 x2 y2* $\neq 0$
shows *proj-addition* (*gluing* ‘‘ $\{((x1,y1),l)\}$) (*gluing* ‘‘ $\{((x2,y2),j)\}$) =
(*gluing* ‘‘ $\{($*ext-add* $(x1,y1)$ $(x2,y2),l+j)\}$)
proof −
have *p-q-expr*: (*gluing* ‘‘ $\{((x1,y1),l)\} = \{((x1, y1), l)\} \vee$ *gluing* ‘‘ $\{((x1,y1),l)\} = \{((x1, y1), l), (\tau\ (x1, y1), l + 1)\}$)
(*gluing* ‘‘ $\{((x2,y2),j)\} = \{((x2, y2), j)\} \vee$ *gluing* ‘‘ $\{((x2,y2),j)\} = \{((x2, y2), j), (\tau\ (x2, y2), j + 1)\}$)
using *assms(1,2) gluing-cases-explicit* **by** *auto*
then consider
(1) *gluing* ‘‘ $\{((x1,y1),l)\} = \{((x1, y1), l)\}$ *gluing* ‘‘ $\{((x2,y2),j)\} = \{((x2, y2), j)\}$ |
(2) *gluing* ‘‘ $\{((x1,y1),l)\} = \{((x1, y1), l)\}$ *gluing* ‘‘ $\{((x2,y2),j)\} = \{((x2, y2), j), (\tau\ (x2, y2), j + 1)\}$ |
(3) *gluing* ‘‘ $\{((x1,y1),l)\} = \{((x1, y1), l), (\tau\ (x1, y1), l + 1)\}$ *gluing* ‘‘ $\{((x2,y2),j)\} = \{((x2, y2), j)\}$ |
(4) *gluing* ‘‘ $\{((x1,y1),l)\} = \{((x1, y1), l), (\tau\ (x1, y1), l + 1)\}$ *gluing* ‘‘ $\{((x2,y2),j)\} = \{((x2, y2), j), (\tau\ (x2, y2), j + 1)\}$ **by** *argo*
then show *?thesis*
proof(*cases*)
case *1*
then show *?thesis* **using** *gluing-ext-add-1 assms* **by** *presburger*
next
case *2* **then show** *?thesis* **using** *gluing-ext-add-2 assms* **by** *presburger*
next
case *3* **then show** *?thesis*
proof −
have *pd*: *delta′ x2 y2 x1 y1* $\neq 0$
using *assms(3)* **unfolding** *delta′-def delta-x-def delta-y-def* **by** *algebra*
have *proj-addition* (*gluing* ‘‘ $\{((x1, y1), l)\}$) (*gluing* ‘‘ $\{((x2, y2), j)\}$) =
proj-addition (*gluing* ‘‘ $\{((x2, y2), j)\}$) (*gluing* ‘‘ $\{((x1, y1), l)\}$)
unfolding *proj-addition-def*
apply(*subst proj-add-class-comm[OF]*)
using *assms* **by** *auto*
also have ... = *gluing* ‘‘ $\{($*ext-add* $(x2, y2)$ $(x1, y1), j+l)\}$

        using *gluing-ext-add-2*[*OF 3*(*2*,*1*) *assms*(*2*,*1*) *pd*] **by** *blast*
      also have ... = *gluing* ‘‘ {(*ext-add* (*x1*, *y1*) (*x2*, *y2*), *l*+*j*)}
        by (*metis add.commute ext-add-comm*)
      finally show *?thesis* **by** *fast*
    qed
  next
    case *4* **then show** *?thesis* **using** *gluing-ext-add-4 assms* **by** *presburger*
  qed
qed

3.4.3 Basic properties

lemma *move-tau-in-delta*:
  assumes *delta* (*fst* (*τ* (*x1*,*y1*))) (*snd* (*τ* (*x1*,*y1*))) *x2 y2* ≠ *0*
  shows *delta x1 y1* (*fst* (*τ* (*x2*,*y2*))) (*snd* (*τ* (*x2*,*y2*))) ≠ *0*
  using *assms*
  unfolding *delta-def delta-plus-def delta-minus-def*
  apply(*simp add*: *t-nz power2-eq-square*[*symmetric*] *algebra-simps t-expr d-nz*)
  apply(*simp split*: *if-splits add*: *divide-simps*)
  by *fastforce*

lemma *move-tau-in-delta-points*:
  assumes *delta* (*fst* (*τ p*)) (*snd* (*τ p*)) (*fst q*) (*snd q*) ≠ *0*
  shows *delta* (*fst p*) (*snd p*) (*fst* (*τ q*)) (*snd* (*τ q*)) ≠ *0*
  using *move-tau-in-delta*
  by (*metis assms prod.collapse*)

lemma *move-tau-in-delta′*:
  assumes *delta′* (*fst* (*τ* (*x1*,*y1*))) (*snd* (*τ* (*x1*,*y1*))) *x2 y2* ≠ *0*
  shows *delta′ x1 y1* (*fst* (*τ* (*x2*,*y2*))) (*snd* (*τ* (*x2*,*y2*))) ≠ *0*
  using *assms*
  unfolding *delta′-def delta-x-def delta-y-def*
  apply(*simp add*: *t-nz power2-eq-square*[*symmetric*] *algebra-simps t-expr d-nz*)
  apply(*simp split*: *if-splits add*: *divide-simps t-nz d-nz*)
  apply(*safe*)
  apply *simp*
  apply(*simp add*: *algebra-simps power2-eq-square*)
  apply(*simp add*: *t-expr power2-eq-square*[*symmetric*] *algebra-simps*)
  by *algebra*

lemma *move-tau-in-delta′-points*:
  assumes *delta′* (*fst* (*τ p*)) (*snd* (*τ p*)) (*fst q*) (*snd q*) ≠ *0*
  shows *delta′* (*fst p*) (*snd p*) (*fst* (*τ q*)) (*snd* (*τ q*)) ≠ *0*
  using *move-tau-in-delta′*
  by (*metis assms prod.collapse*)

lemma *proj-add-class-inv*:

assumes *gluing* ‘‘ {((*x*,*y*),*l*)} ∈ *e-proj*
shows *proj-addition* (*gluing* ‘‘ {((*x*,*y*),*l*)}) (*gluing* ‘‘ {(*i* (*x*,*y*),*l*′)}) = {((*1*, *0*), *l*+*l*′)}
gluing ‘‘ {(*i* (*x*,*y*),*l*′)} ∈ *e-proj*
proof −
have *in-aff*: (*x*,*y*) ∈ *e*′*-aff*
using *assms e-proj-aff* **by** *blast*
then have *i-aff*: *i* (*x*, *y*) ∈ *e*′*-aff*
using *i-aff* **by** *blast*
show *i-proj*: *gluing* ‘‘ {(*i* (*x*,*y*),*l*′)} ∈ *e-proj*
using *e-proj-aff i-aff* **by** *simp*

have *gl-form*: *gluing* ‘‘ {((*x*,*y*),*l*)} = {((*x*,*y*),*l*)} ∨
gluing ‘‘ {((*x*,*y*),*l*)} = {((*x*,*y*),*l*),(τ (*x*,*y*),*l*+*1*)}
using *assms gluing-cases-explicit* **by** *simp*

then consider (*1*) *gluing* ‘‘ {((*x*,*y*),*l*)} = {((*x*,*y*),*l*)} |
(*2*) *gluing* ‘‘ {((*x*,*y*),*l*)} = {((*x*,*y*),*l*),(τ (*x*,*y*),*l*+*1*)} **by** *fast*
then show *proj-addition* (*gluing* ‘‘ {((*x*,*y*),*l*)}) (*gluing* ‘‘ {(*i* (*x*,*y*),*l*′)}) =
{((*1*, *0*), *l*+*l*′)}
proof(*cases*)
case *1*
then have *zeros*: *x* = *0* ∨ *y* = *0*
using *e-proj-elim-1 in-aff assms* **by** *auto*
have *gl-eqs*: *gluing* ‘‘ {((*x*,*y*),*l*)} = {((*x*, *y*), *l*)}
gluing ‘‘ {(*i* (*x*,*y*),*l*′)} = {(*i* (*x*, *y*), *l*′)}
using *zeros in-aff i-aff gluing-class-1* **by** *auto*
have *e-proj*: {((*x*, *y*), *l*)} ∈ *e-proj*
{(*i* (*x*, *y*), *l*′)} ∈ *e-proj*
using *assms e-proj-elim-1 i-aff in-aff zeros* **by** *auto*

have *i-delta*: *delta x y* (*fst* (*i* (*x*,*y*))) (*snd* (*i* (*x*,*y*))) ≠ *0*
using *i-aff in-aff zeros*
unfolding *e*′*-aff-def e*′*-def*
unfolding *delta-def delta-plus-def delta-minus-def*
delta′*-def delta-x-def delta-y-def*
apply(*simp add*: *t-expr*)
by *algebra*
have *add-eq*: *proj-add* ((*x*, *y*), *l*) (*i* (*x*, *y*), *l*′) = ((*1*,*0*),*l*+*l*′)
using *proj-add-inv*[*OF* ‹(*x*,*y*) ∈ *e*′*-aff*›] **by** *simp*
have *dom-eq*: {*proj-add* ((*x1*, *y1*), *ia*) ((*x2*, *y2*), *j*) |*x1 y1 ia x2 y2 j*.
((*x1*, *y1*), *ia*) ∈ {((*x*, *y*), *l*)} ∧ ((*x2*, *y2*), *j*) ∈ {(*i* (*x*, *y*), *l*′)} ∧ ((*x1*, *y1*), *x2*, *y2*) ∈ *e*′*-aff-0* ∪ *e*′*-aff-1*} =
{((*1*,*0*),*l*+*l*′)}
(**is** *?s* = *?t*)
proof
show *?s* ⊆ *?t*
proof
fix *e*

```
      assume e ∈ ?s
      then have e = proj-add ((x, y), l) (i (x, y), l') by force
      then have e = ((1,0),l+l') using add-eq by auto
      then show e ∈ ?t by blast
    qed
  next
    show ?t ⊆ ?s
    proof
      fix e
      assume e ∈ ?t
      then have e = ((1,0),l+l') by force
      then have e = proj-add ((x, y), l) (i (x, y), l')
        using add-eq by auto
      then show e ∈ ?s
        unfolding e'-aff-0-def
        using in-aff i-aff i-delta by force
    qed
  qed

  show proj-addition (gluing `` {((x,y),l)}) (gluing `` {(i (x,y),l')}) =
      {((1, 0), l+l')}
    unfolding proj-addition-def gl-eqs
    apply(subst proj-add-class.simps(1)[OF e-proj])
    apply(subst dom-eq)
    by (simp add: identity-equiv singleton-quotient)
next
  case 2
  from e-proj-elim-2[OF ‹(x,y) ∈ e'-aff›]
  have nz: x ≠ 0 y ≠ 0
    using 2 assms by force+
  have taus: τ (x,y) ∈ e'-aff τ (i (x,y)) ∈ e'-aff
    using e-proj-aff gluing-inv[OF nz in-aff, of l] assms apply simp
    using e-proj-aff gluing-inv nz i-aff i-proj by force

  have gl-eqs:
     gluing `` {((x, y), l)} = {((x, y), l),(τ (x, y), l+1)}
     gluing `` {(i (x, y), l')} = {(i (x, y), l'),(τ (i (x, y)), l'+1)}
    using ‹x ≠ 0› ‹y ≠ 0› gluing-class-2 i-aff in-aff by auto
  have gl-proj:
     gluing `` {((x, y), l)} ∈ e-proj
     gluing `` {(i (x, y), l')} ∈ e-proj
    using in-aff i-aff e-proj-aff by auto

  have deltas:
      delta (fst (τ (x,y))) (snd (τ (x,y)))
            (fst (i (x,y))) (snd (i (x,y))) = 0
      delta' (fst (τ (x,y))) (snd (τ (x,y)))
             (fst (i (x,y))) (snd (i (x,y))) = 0
      delta x y (fst (τ (i (x,y)))) (snd (τ (i (x,y)))) = 0
```

```
    delta' x y (fst (τ (i (x,y)))) (snd (τ (i (x,y)))) = 0
  using nz in-aff taus
  unfolding delta-def delta-plus-def delta-minus-def
            delta'-def delta-x-def delta-y-def
            e'-aff-def e'-def
  apply(simp-all add: divide-simps t-nz)
  apply(simp-all add: algebra-simps power2-eq-square)
  by(simp-all add: algebra-simps power2-eq-square[symmetric] t-expr)

have v1: proj-add ((x,y),l) (i (x, y), l') = ((1, 0), l+l')
  using ‹(x, y) ∈ e'-aff› proj-add-inv by auto
have v2: proj-add (τ (x,y),l+1) (τ (i (x, y)), l'+1) = ((1, 0), l+l')
  using taus proj-add-inv by force
have v3: proj-add (τ (x,y),l+1) (i (x, y), l') = undefined
  using proj-add.simps deltas by auto
have v4: proj-add ((x, y), l) (τ (i (x, y)), l'+1) = undefined
  using proj-add.simps deltas by auto

have good-deltas:
    delta x y (fst (i (x,y))) (snd (i (x,y))) ≠ 0 ∨
     delta' x y (fst (i (x,y))) (snd (i (x,y))) ≠ 0 (is ?one)
    delta (fst (τ (x,y))) (snd (τ (x,y)))
          (fst (τ (i (x,y)))) (snd (τ (i (x,y)))) ≠ 0 ∨
     delta' (fst (τ (x,y))) (snd (τ (x,y)))
          (fst (τ (i (x,y)))) (snd (τ (i (x,y)))) ≠ 0 (is ?two)
proof –

  show ?one
    unfolding delta-def delta-plus-def delta-minus-def
              delta'-def delta-x-def delta-y-def
  proof(simp-all add: two-not-zero nz,cases x^2 = y^2)
    case True
    have simp: 2 ∗ y² ≠ 2 2 ∗ y^2 ≠ 0
      using in-aff t-n1 t-nm1 two-not-zero unfolding e'-aff-def e'-def
      apply simp-all
      using True t-def t-ineq(1) by auto
    then show d ∗ x ∗ y ∗ x ∗ y ≠ 1 ∧ 1 + d ∗ x ∗ y ∗ x ∗ y ≠ 0 ∨ x ∗ x ≠
y ∗ y
      using in-aff unfolding e'-aff-def e'-def
      apply(simp add: t-expr)
      by algebra+
  next
    case False
    then show d ∗ x ∗ y ∗ x ∗ y ≠ 1 ∧ 1 + d ∗ x ∗ y ∗ x ∗ y ≠ 0 ∨ x ∗ x ≠
y ∗ y
      by algebra
  qed

  then show ?two
```

proof(*cases delta x y* (*fst* (*i* (*x*, *y*))) (*snd* (*i* (*x*, *y*))) $\neq$ *0*)
case *True*
then have *delta* (*fst* (τ (τ (*x*,*y*)))) (*snd* (τ (τ (*x*,*y*)))) (*fst* (*i* (*x*, *y*))) (*snd* (*i* (*x*, *y*))) $\neq$ *0*
using *tau-idemp-point* **by** *fastforce*
then have *delta* (*fst* (τ (*x*, *y*))) (*snd* (τ (*x*, *y*))) (*fst* (τ (*i* (*x*, *y*)))) (*snd* (τ (*i* (*x*, *y*)))) $\neq$ *0*
using *move-tau-in-delta-points* **by** *blast*
then show *?thesis* **by** *auto*
next
case *False*
then have *delta*$'$ *x y* (*fst* (*i* (*x*, *y*))) (*snd* (*i* (*x*, *y*))) $\neq$ *0*
using ‹*?one*› **by** *blast*
then have *delta*$'$ (*fst* (τ (τ (*x*,*y*)))) (*snd* (τ (τ (*x*,*y*)))) (*fst* (*i* (*x*, *y*))) (*snd* (*i* (*x*, *y*))) $\neq$ *0*
using *tau-idemp-point* **by** *fastforce*
then have *delta*$'$ (*fst* (τ (*x*, *y*))) (*snd* (τ (*x*, *y*))) (*fst* (τ (*i* (*x*, *y*)))) (*snd* (τ (*i* (*x*, *y*)))) $\neq$ *0*
using *move-tau-in-delta*$'$*-points* **by** *blast*
then show *?thesis* **by** *auto*
qed
qed

have *dom-eq*: {*proj-add* ((*x1*, *y1*), *ia*) ((*x2*, *y2*), *j*) |*x1 y1 ia x2 y2 j*.
((*x1*, *y1*), *ia*) $\in$ {((*x*, *y*), *l*), (τ (*x*, *y*), *l* + *1*)} $\wedge$
((*x2*, *y2*), *j*) $\in$ {(*i* (*x*, *y*), *l*$'$), (τ (*i* (*x*, *y*)), *l*$'$ + *1*)} $\wedge$
((*x1*, *y1*), *x2*, *y2*) $\in$ *e*$'$*-aff-0* $\cup$ *e*$'$*-aff-1*} =
{((*1*, *0*), *l*+*l*$'$)}
(**is** *?s* = *?t*)
proof
show *?s* $\subseteq$ *?t*
proof
fix *e*
assume *e* $\in$ *?s*
then have *e* = *proj-add* ((*x*, *y*), *l*) (*i* (*x*, *y*), *l*$'$) $\vee$
e = *proj-add* (τ (*x*,*y*),*l*+*1*) (τ (*i* (*x*, *y*)), *l*$'$+*1*)
using *v1 v2 v3 v4 deltas*
unfolding *e*$'$*-aff-0-def e*$'$*-aff-1-def*
by *force*
then have *e* = ((*1*,*0*),*l*+*l*$'$) **using** *v1 v2* **by** *argo*
then show *e* $\in$ *?t* **by** *blast*
qed
next
show *?t* $\subseteq$ *?s*
proof
fix *e*
assume *e* $\in$ *?t*
then have *e* = ((*1*,*0*),*l*+*l*$'$) **by** *force*
then have *e* = *proj-add* ((*x*, *y*), *l*) (*i* (*x*, *y*), *l*$'$) $\vee$

```
            e = proj-add (τ (x,y),l+1) (τ (i (x, y)), l'+1)
        using v1 v2 v3 v4 deltas
        unfolding e'-aff-0-def e'-aff-1-def
        by force
      then show e ∈ ?s
        apply(elim disjE)
        subgoal
          using v1 good-deltas(1) in-aff i-aff
          unfolding e'-aff-0-def e'-aff-1-def
          apply(simp del: τ.simps)
          by metis
        subgoal
          using v2 good-deltas(2) taus
          unfolding e'-aff-0-def e'-aff-1-def
          apply(simp del: )
          by metis
        done
    qed
  qed

  show ?thesis
    unfolding proj-addition-def
    unfolding proj-add-class.simps(1)[OF gl-proj]
    unfolding gl-eqs
    apply(subst dom-eq)
    by (simp add: identity-equiv singleton-quotient)
 qed
qed


lemma proj-add-class-identity:
 assumes x ∈ e-proj
 shows proj-addition {((1, 0), 0)} x = x
proof −
 obtain x0 y0 l0 where
   x-expr: x = gluing `` {((x0,y0),l0)}
   using assms e-proj-def
   apply(simp)
   apply(elim quotientE)
   by force
 then have in-aff: (x0,y0) ∈ e'-aff
   using e-proj-aff assms by blast

 have proj-addition {((1, 0), 0)} x =
      proj-addition (gluing `` {((1, 0), 0)}) (gluing `` {((x0,y0),l0)})
   using identity-equiv[of 0] x-expr by argo
 also have ... = gluing `` {(add (1,0) (x0,y0),l0)}
   apply(subst gluing-add)
```

```
    using identity-equiv identity-proj apply simp
    using x-expr assms apply simp
    unfolding delta-def delta-plus-def delta-minus-def apply simp
    by simp
  also have ... = gluing `` {((x0,y0),l0)}
    using inverse-generalized in-aff
    unfolding e'-aff-def by simp
  also have ... = x
    using x-expr by simp
  finally show ?thesis by simp
qed

theorem well-defined:
  assumes p ∈ e-proj q ∈ e-proj
  shows proj-addition p q ∈ e-proj
proof −
  obtain x y l x' y' l'
    where p-q-expr: p = gluing `` {((x,y),l)}
                    q = gluing `` {((x',y'),l')}
    using e-proj-def assms
    apply(simp)
    apply(elim quotientE)
    by force
  then have in-aff: (x,y) ∈ e'-aff
                    (x',y') ∈ e'-aff
    using e-proj-aff assms by auto

  consider
   (a) (x, y) ∈ e-circ ∧ (∃ g∈symmetries. (x', y') = (g ∘ i) (x, y))
  | (b) ((x, y), x', y') ∈ e'-aff-0
        ((x, y), x', y') ∉ e'-aff-1
        (x, y) ∉ e-circ ∨ ¬ (∃ g∈symmetries. (x', y') = (g ∘ i) (x, y))
  | (c) ((x, y), x', y') ∈ e'-aff-1
    using dichotomy-1[OF in-aff] by auto
  then show ?thesis
  proof(cases)
    case a
    then obtain g where sym-expr: g ∈ symmetries (x', y') = (g ∘ i) (x, y) by
auto
    then have ds: delta x y x' y' = 0 delta' x y x' y' = 0
      using wd-d-nz wd-d'-nz a by auto
    have nz: x ≠ 0 y ≠ 0 x' ≠ 0 y' ≠ 0
    proof −
      from a show x ≠ 0 y ≠ 0
        unfolding e-circ-def by auto
      then show x' ≠ 0 y' ≠ 0
        using sym-expr t-nz
        unfolding symmetries-def e-circ-def
        by auto
```

qed
have *taus*: τ (x',y') $\in$ e'-*aff*
using *in-aff(2) e-circ-def nz(3,4)* τ-*circ* **by** *force*
then have *proj*: *gluing* ‘‘ $\{(\tau$ $(x', y'), l'+1)\}$ $\in$ *e-proj*
gluing ‘‘ $\{((x, y), l)\}$ $\in$ *e-proj*
using *e-proj-aff in-aff* **by** *auto*

have *alt-ds*: *delta x y* (*fst* (τ (x',y'))) (*snd* (τ (x',y'))) $\neq 0$ $\vee$
delta' *x y* (*fst* (τ (x',y'))) (*snd* (τ (x',y'))) $\neq 0$
(**is** *?d1* $\neq 0$ $\vee$ *?d2* $\neq 0$)
using *covering-with-deltas ds assms p-q-expr* **by** *blast*

have *proj-addition p q* = *proj-addition* (*gluing* ‘‘ $\{((x, y), l)\}$) (*gluing* ‘‘ $\{((x', y'), l')\}$)
(**is** *?lhs* = *proj-addition ?p ?q*)
unfolding *p-q-expr* **by** *simp*
also have ... = *proj-addition ?p* (*gluing* ‘‘ $\{(\tau$ $(x', y'), l'+1)\}$)
(**is** - = *?rhs*)
using *gluing-inv nz in-aff* **by** *presburger*
finally have *?lhs* = *?rhs*
by *auto*
then have *eqs*:
?d1 $\neq 0 \Longrightarrow$ *?lhs* = *gluing* ‘‘ $\{(add\ (x, y)\ (\tau\ (x', y')), l+l'+1)\}$
?d2 $\neq 0 \Longrightarrow$ *?lhs* = *gluing* ‘‘ $\{(ext\text{-}add\ (x, y)\ (\tau\ (x', y')), l+l'+1)\}$
using *gluing-add gluing-ext-add proj alt-ds*
by (*metis* (*no-types*, *lifting*) *add.assoc prod.collapse*)+
have *closures*:
?d1 $\neq 0 \Longrightarrow$ *add* (x, y) (τ (x', y')) $\in$ e'-*aff*
?d2 $\neq 0 \Longrightarrow$ *ext-add* (x, y) (τ (x', y')) $\in$ e'-*aff*
using *e-proj-aff add-closure in-aff taus delta-def* e'-*aff-def* *e*-e'-*iff*
apply *fastforce*
using *e-proj-aff ext-add-closure in-aff taus delta-def* e'-*aff-def* *e*-e'-*iff*
by *fastforce*

have *f-proj*: *?d1* $\neq 0 \Longrightarrow$ *gluing* ‘‘ $\{(add\ (x, y)\ (\tau\ (x', y')), l+l'+1)\}$ $\in$ *e-proj*
?d2 $\neq 0 \Longrightarrow$ *gluing* ‘‘ $\{(ext\text{-}add\ (x, y)\ (\tau\ (x', y')), l+l'+1)\}$ $\in$ *e-proj*
using *e-proj-aff closures* **by** *force*+

then show *?thesis*
using *eqs alt-ds* **by** *auto*
next
case *b*
then have *ds*: *delta x y x' y'* $\neq 0$
unfolding e'-*aff-0-def* **by** *auto*

have *eq*: *proj-addition p q* = *gluing* ‘‘ $\{(add\ (x, y)\ (x',y'), l+l')\}$
(**is** *?lhs* = *?rhs*)
unfolding *p-q-expr*
using *gluing-add assms p-q-expr ds* **by** *meson*

have *add-in*: *add* $(x,\ y)$ (x',y') $\in$ *e'-aff*
using *add-closure in-aff ds e-e'-iff*
unfolding *delta-def e'-aff-def* **by** *auto*
then show *?thesis*
using *eq e-proj-aff* **by** *auto*
next
case *c*
then have *ds*: *delta'* $x\ y\ x'\ y' \neq 0$
unfolding *e'-aff-1-def* **by** *auto*

have *eq*: *proj-addition* $p\ q$ = *gluing* `` {(*ext-add* $(x,\ y)$ (x',y'), $l+l'$)}
(**is** *?lhs* = *?rhs*)
unfolding *p-q-expr*
using *gluing-ext-add assms p-q-expr ds* **by** *meson*
have *add-in*: *ext-add* $(x,\ y)$ (x',y') $\in$ *e'-aff*
using *ext-add-closure in-aff ds e-e'-iff*
unfolding *delta-def e'-aff-def* **by** *auto*
then show *?thesis*
using *eq e-proj-aff* **by** *auto*
qed
qed

corollary *proj-addition-comm*:
assumes *c1* $\in$ *e-proj c2* $\in$ *e-proj*
shows *proj-addition c1 c2* = *proj-addition c2 c1*
using *proj-add-class-comm*[*OF assms*]
unfolding *proj-addition-def* **by** *auto*

4 Group law

4.1 Class invariance on group operations

definition *tf* **where**
tf g = *image* (λ *p*. (*g* (*fst p*), *snd p*))

lemma *tf-comp*:
tf g (*tf f s*) = *tf* ($g \circ f$) *s*
unfolding *tf-def* **by** *force*

lemma *tf-id*:
tf id s = *s*
unfolding *tf-def* **by** *fastforce*

definition *tf'* **where**
tf' = *image* (λ *p*. (*fst p*, (*snd p*)+*1*))

lemma *tf-tf'-commute*:
tf r (*tf' p*) = *tf'* (*tf r p*)
unfolding *tf'-def tf-def image-def*

by *auto*

lemma *rho-preserv-e-proj*:
assumes *gluing* ‘‘ $\{((x, y), l)\} \in$ *e-proj*
shows *tf* ϱ (*gluing* ‘‘ $\{((x, y), l)\}$) $\in$ *e-proj*
proof –
have *in-aff*: $(x,y) \in e'$*-aff*
using *assms e-proj-aff* **by** *blast*
have *rho-aff*: ϱ $(x,y) \in e'$*-aff*
using *rot-aff*[*of* ϱ,*OF* - *in-aff*] *rotations-def* **by** *blast*

have *eq*: *gluing* ‘‘ $\{((x, y), l)\} = \{((x, y), l)\} \vee$
gluing ‘‘ $\{((x, y), l)\} = \{((x, y), l), (\tau\ (x, y), l+1)\}$
using *assms gluing-cases-explicit* **by** *auto*
from *eq* **consider**
(*1*) *gluing* ‘‘ $\{((x, y), l)\} = \{((x, y), l)\}$ |
(*2*) *gluing* ‘‘ $\{((x, y), l)\} = \{((x, y), l), (\tau\ (x, y), l+1)\}$
by *fast*
then show *tf* ϱ (*gluing* ‘‘ $\{((x, y), l)\}$) $\in$ *e-proj*
proof(*cases*)
case *1*
have *zeros*: $x = 0 \vee y = 0$
using *in-aff e-proj-elim-1 assms e-proj-aff 1* **by** *auto*
show *?thesis*
unfolding *tf-def*
using *rho-aff zeros e-proj-elim-1 1* **by** *auto*
next
case *2*
have *zeros*: $x \neq 0$ $y \neq 0$
using *in-aff e-proj-elim-2 assms e-proj-aff 2* **by** *auto*
show *?thesis*
unfolding *tf-def*
using *rho-aff zeros e-proj-elim-2 2* **by** *fastforce*
qed
qed

lemma *insert-rho-gluing*:
assumes *gluing* ‘‘ $\{((x, y), l)\} \in$ *e-proj*
shows *tf* ϱ (*gluing* ‘‘ $\{((x, y), l)\}$) = *gluing* ‘‘ $\{(\varrho\ (x, y), l)\}$
proof –
have *in-aff*: $(x,y) \in e'$*-aff*
using *assms e-proj-aff* **by** *blast*
have *rho-aff*: ϱ $(x,y) \in e'$*-aff*
using *rot-aff*[*of* ϱ,*OF* - *in-aff*] *rotations-def* **by** *blast*

have *eq*: *gluing* ‘‘ $\{((x, y), l)\} = \{((x, y), l)\} \vee$
gluing ‘‘ $\{((x, y), l)\} = \{((x, y), l), (\tau\ (x, y), l+1)\}$
using *assms gluing-cases-explicit* **by** *auto*
from *eq* **consider**

(*1*) *gluing* `` $\{((x, y), l)\} = \{((x, y), l)\}$ |
(*2*) *gluing* `` $\{((x, y), l)\} = \{((x, y), l), (\tau\ (x, y), l+1)\}$
by *fast*
then show *tf* ϱ (*gluing* `` $\{((x, y), l)\}$) = *gluing* `` $\{(\varrho\ (x, y), l)\}$
proof(*cases*)
case *1*
have *zeros*: $x = 0 \lor y = 0$
using *in-aff e-proj-elim-1 assms e-proj-aff 1* **by** *auto*
then have *gluing* `` $\{(\varrho\ (x, y), l)\} = \{(\varrho\ (x, y), l)\}$
using *gluing-class-1*[*of fst* (ϱ (x, y)) *snd* (ϱ (x, y)),
simplified prod.collapse,
OF - rho-aff] **by** *fastforce*
then show *?thesis*
unfolding *tf-def image-def 1* **by** *simp*
next
case *2*
have *zeros*: $x \neq 0\ y \neq 0$
using *in-aff e-proj-elim-2 assms e-proj-aff 2* **by** *auto*
then have *gluing* `` $\{(\varrho\ (x, y), l)\} = \{(\varrho\ (x, y), l), (\tau\ (\varrho\ (x, y)), l+1)\}$
using *gluing-class-2*[*of fst* (ϱ (x, y)) *snd* (ϱ (x, y)),
simplified prod.collapse, *OF - - rho-aff*] **by** *force*
then show *?thesis*
unfolding *tf-def image-def 2* **by** *force*
qed
qed

lemma *rotation-preserv-e-proj*:
assumes *gluing* `` $\{((x, y), l)\} \in$ *e-proj* $r \in$ *rotations*
shows *tf r* (*gluing* `` $\{((x, y), l)\}$) $\in$ *e-proj*
(**is** *tf ?r ?g* $\in$ -)
using *assms*
unfolding *rotations-def*
apply(*safe*)
using *tf-id*[*of ?g*] **apply** *simp*
using *rho-preserv-e-proj* **apply** *simp*
using *tf-comp rho-preserv-e-proj insert-rho-gluing*
by(*metis* (*no-types*, *hide-lams*) *prod.collapse*)+

lemma *insert-rotation-gluing*:
assumes *gluing* `` $\{((x, y), l)\} \in$ *e-proj* $r \in$ *rotations*
shows *tf r* (*gluing* `` $\{((x, y), l)\}$) = *gluing* `` $\{(r\ (x, y), l)\}$
proof −
have *in-proj*: *gluing* `` $\{(\varrho\ (x, y), l)\} \in$ *e-proj gluing* `` $\{((\varrho \circ \varrho)\ (x, y), l)\} \in$ *e-proj*
using *rho-preserv-e-proj assms insert-rho-gluing* **by** *auto+*

consider (*1*) $r = id$ |
(*2*) $r = \varrho$ |
(*3*) $r = \varrho \circ \varrho$ |

```
      (4) r = ϱ ∘ ϱ ∘ ϱ
    using assms(2) unfolding rotations-def by fast
  then show ?thesis
  proof(cases)
    case 1
    then show ?thesis using tf-id by auto
  next
    case 2
    then show ?thesis using insert-rho-gluing assms by presburger
  next
    case 3
    then show ?thesis
      using insert-rho-gluing assms tf-comp in-proj(1)
      by (metis (no-types, lifting) ϱ.simps comp-apply)
  next
    case 4
    then show ?thesis
      using insert-rho-gluing assms tf-comp in-proj
      by (metis (no-types, lifting) ϱ.simps comp-apply)
  qed
qed


lemma tf-tau:
  assumes gluing `` {((x,y),l)} ∈ e-proj
  shows gluing `` {((x,y),l+1)} = tf' (gluing `` {((x,y),l)})
  using assms unfolding symmetries-def
proof -
  have in-aff: (x,y) ∈ e'-aff
    using e-proj-aff assms by simp

  have gl-expr: gluing `` {((x,y),l)} = {((x,y),l)} ∨
                gluing `` {((x,y),l)} = {((x,y),l),(τ (x,y),l+1)}
    using assms(1) gluing-cases-explicit by simp

  consider (1) gluing `` {((x,y),l)} = {((x,y),l)} |
           (2) gluing `` {((x,y),l)} = {((x,y),l),(τ (x,y),l+1)}
    using gl-expr by argo
  then show gluing `` {((x,y), l+1)} = tf' (gluing `` {((x,y), l)})
  proof(cases)
    case 1
    then have zeros: x = 0 ∨ y = 0
      using e-proj-elim-1 in-aff assms by auto
    show ?thesis
      apply(simp add: 1 tf'-def del: τ.simps)
      using gluing-class-1 zeros in-aff by auto
  next
    case 2
    then have zeros: x ≠ 0 y ≠ 0
      using assms e-proj-elim-2 in-aff by auto
```

show *?thesis*
apply(*simp add*: *2 tf′-def del*: *τ.simps*)
using *gluing-class-2 zeros in-aff* **by** *auto*
qed
qed

lemma *tf-preserv-e-proj*:
assumes *gluing ‘‘* {((*x*,*y*),*l*)} ∈ *e-proj*
shows *tf′* (*gluing ‘‘* {((*x*,*y*),*l*)}) ∈ *e-proj*
using *assms tf-tau*[*OF assms*]
e-proj-aff[*of x y l*] *e-proj-aff*[*of x y l+1*] **by** *auto*

lemma *remove-rho*:
assumes *gluing ‘‘* {((*x*,*y*),*l*)} ∈ *e-proj*
shows *gluing ‘‘* {(*ϱ* (*x*,*y*),*l*)} = *tf ϱ* (*gluing ‘‘* {((*x*,*y*),*l*)})
using *assms* **unfolding** *symmetries-def*
proof −
have *in-aff*: (*x*,*y*) ∈ *e′-aff* **using** *assms e-proj-aff* **by** *simp*
have *rho-aff*: *ϱ* (*x*,*y*) ∈ *e′-aff*
using *in-aff* **unfolding** *e′-aff-def e′-def* **by**(*simp*,*algebra*)

consider (*1*) *gluing ‘‘* {((*x*,*y*),*l*)} = {((*x*,*y*),*l*)} |
(*2*) *gluing ‘‘* {((*x*,*y*),*l*)} = {((*x*,*y*),*l*),(*τ* (*x*,*y*),*l+1*)}
using *assms gluing-cases-explicit* **by** *blast*
then show *gluing ‘‘* {(*ϱ* (*x*,*y*), *l*)} = *tf ϱ* (*gluing ‘‘* {((*x*,*y*), *l*)})
proof(*cases*)
case *1*
then have *zeros*: *x* = *0* ∨ *y* = *0*
using *assms e-proj-elim-1 in-aff* **by** *simp*
then have *rho-zeros*: *fst* (*ϱ* (*x*,*y*)) = *0* ∨ *snd* (*ϱ* (*x*,*y*)) = *0*
by *force*
have *gl-eq*: *gluing ‘‘* {(*ϱ* (*x*, *y*), *l*)} = {(*ϱ* (*x*, *y*), *l*)}
using *gluing-class-1 rho-zeros rho-aff* **by** *force*
show *?thesis*
unfolding *gl-eq 1*
unfolding *tf-def image-def*
by *simp*
next
case *2*
then have *zeros*: *x* ≠ *0 y* ≠ *0*
using *assms e-proj-elim-2 in-aff* **by** *auto*
then have *rho-zeros*: *fst* (*ϱ* (*x*,*y*)) ≠ *0 snd* (*ϱ* (*x*,*y*)) ≠ *0*
using *t-nz* **by** *auto*
have *gl-eqs*: *gluing ‘‘* {(*ϱ* (*x*, *y*), *l*)} = {(*ϱ* (*x*, *y*), *l*), (*τ* (*ϱ* (*x*, *y*)), *l+1*)}
using *gluing-class-2 rho-zeros rho-aff* **by** *force*
show *?thesis*
unfolding *gl-eqs 2*
unfolding *tf-def image-def*
by *force*

qed
qed

lemma *remove-rotations*:
  assumes *gluing* ‘‘ $\{((x,y),l)\} \in$ *e-proj* $r \in$ *rotations*
  shows *gluing* ‘‘ $\{(r\ (x,y),l)\} =$ *tf* r (*gluing* ‘‘ $\{((x,y),l)\}$)
proof −
  consider (*1*) $r = id$ |
           (*2*) $r = \varrho$ |
           (*3*) $r = \varrho \circ \varrho$ |
           (*4*) $r = \varrho \circ \varrho \circ \varrho$
    using *assms*(*2*) **unfolding** *rotations-def* **by** *fast*
  then show *?thesis*
  proof(*cases*)
    case *1*
    then show *?thesis* **using** *tf-id* **by** *fastforce*
  next
    case *2*
    then show *?thesis* **using** *remove-rho*[*OF assms*(*1*)] **by** *fast*
  next
    case *3*
    then show *?thesis*
      using *remove-rho rho-preserv-e-proj assms*(*1*)
      by (*simp add*: *tf-comp*)
  next
    case *4*
    then show *?thesis*
      using *remove-rho rho-preserv-e-proj assms*(*1*)
      by (*metis* (*no-types*, *lifting*) ϱ*.simps comp-apply tf-comp*)
  qed
qed

lemma *remove-tau*:
  assumes *gluing* ‘‘ $\{((x,y),l)\} \in$ *e-proj gluing* ‘‘ $\{(\tau\ (x,y),l)\} \in$ *e-proj*
  shows *gluing* ‘‘ $\{(\tau\ (x,y),l)\} =$ *tf′* (*gluing* ‘‘ $\{((x,y),l)\}$)
  (**is** *?gt* = *tf′ ?g*)
proof −
  have *in-aff*: $(x,y) \in$ *e′-aff* $\tau\ (x,y) \in$ *e′-aff*
    using *assms e-class* **by** *simp*+

  consider (*1*) *?gt* = $\{(\tau\ (x,y),l)\}$ | (*2*) *?gt* = $\{(\tau\ (x,y),l),((x,y),l+1)\}$
    using *tau-idemp-point gluing-cases-points*[*OF assms*(*2*), *of* $\tau\ (x,y)\ l$] **by** *pres-burger*
  then show *?thesis*
  proof(*cases*)
    case *1*
    then have *zeros*: $x = 0 \vee y = 0$
      using *e-proj-elim-1 in-aff assms* **by**(*simp add*: *t-nz*)
    have *False*

```
      using zeros in-aff t-n1 d-n1
      unfolding e'-aff-def e'-def
      apply(simp)
      apply(safe)
      apply(simp-all add: power2-eq-square algebra-simps)
      apply(simp-all add: power2-eq-square[symmetric] t-expr)
      by algebra+
    then show ?thesis by simp
  next
    case 2
    then have zeros: x ≠ 0 y ≠ 0
      using e-proj-elim-2 in-aff assms gluing-class-1 by auto
    then have gl-eq: gluing `` {((x,y),l)} = {((x,y),l),(τ (x,y),l+1)}
      using in-aff gluing-class-2 by auto
    then show ?thesis
      by(simp add: 2 gl-eq tf'-def del: τ.simps,fast)
  qed
qed

lemma remove-add-rho:
  assumes p ∈ e-proj q ∈ e-proj
  shows proj-addition (tf ϱ p) q = tf ϱ (proj-addition p q)
proof −
  obtain x y l x' y' l' where
    p-q-expr: p = gluing `` {((x, y), l)}
              q = gluing `` {((x', y'), l')}
    using assms
    unfolding e-proj-def
    apply(elim quotientE)
    by force
  have e-proj:
    gluing `` {((x, y), l)} ∈ e-proj
    gluing `` {((x', y'), l')} ∈ e-proj
    using p-q-expr assms by auto
  then have rho-e-proj:
    gluing `` {(ϱ (x, y), l)} ∈ e-proj
    using remove-rho rho-preserv-e-proj by auto

  have in-aff: (x,y) ∈ e'-aff (x',y') ∈ e'-aff
    using assms p-q-expr e-proj-aff by auto

  consider
    (a) (x, y) ∈ e-circ ∧ (∃ g∈symmetries. (x', y') = (g ∘ i) (x, y)) |
    (b) ((x, y), x', y') ∈ e'-aff-0 ¬ ((x, y) ∈ e-circ ∧ (∃ g∈symmetries. (x', y') =
(g ∘ i) (x, y))) |
    (c) ((x, y), x', y') ∈ e'-aff-1 ¬ ((x, y) ∈ e-circ ∧ (∃ g∈symmetries. (x', y') =
(g ∘ i) (x, y))) ((x, y), x', y') ∉ e'-aff-0
    using dichotomy-1[OF ‹(x,y) ∈ e'-aff› ‹(x',y') ∈ e'-aff›] by argo
  then show ?thesis
```

```
proof(cases)
  case a
  then have e-circ: (x,y) ∈ e-circ by auto
  then have zeros: x ≠ 0 y ≠ 0 unfolding e-circ-def by auto
  from a obtain g where g-expr:
    g ∈ symmetries (x', y') = (g ∘ i) (x, y) by blast
  then obtain r where r-expr: (x', y') = (τ ∘ r ∘ i) (x, y) r ∈ rotations
    using sym-decomp by blast
  have ds: delta x y x' y' = 0 delta' x y x' y' = 0
    using wd-d-nz[OF g-expr e-circ] wd-d'-nz[OF g-expr e-circ] by auto

  have ds'': delta x y (fst ((r ∘ i) (x, y))) (snd ((r ∘ i) (x, y))) ≠ 0 ∨
             delta' x y (fst ((r ∘ i) (x, y))) (snd ((r ∘ i) (x, y))) ≠ 0
    (is ?ds1 ≠ 0 ∨ ?ds2 ≠ 0)
    using r-expr covering-with-deltas tau-idemp-point ds
    by (metis comp-apply e-proj(1) e-proj(2))

  have ds''': delta (fst (ϱ (x,y))) (snd (ϱ (x,y))) (fst ((r ∘ i) (x, y))) (snd ((r ∘
i) (x, y))) ≠ 0 ∨
              delta' (fst (ϱ (x,y))) (snd (ϱ (x,y))) (fst ((r ∘ i) (x, y))) (snd ((r ∘
i) (x, y))) ≠ 0
    (is ?ds3 ≠ 0 ∨ ?ds4 ≠ 0)
    using r-expr(2) rotation-invariance-3 rotation-invariance-4 delta-com ds''
   by (metis (no-types, hide-lams) add.inverse-inverse delta'-com diff-0 minus-diff-eq)
  have ds:?ds3 ≠ 0 ⟹ delta x y x (−y) ≠ 0
        ?ds4 ≠ 0 ⟹ delta' x y x (−y) ≠ 0
        ?ds1 ≠ 0 ⟹ delta x y x (−y) ≠ 0
        ?ds2 ≠ 0 ⟹ delta' x y x (−y) ≠ 0
    using ds'''

    using r-expr
    unfolding delta-def delta-plus-def delta-minus-def
              delta'-def delta-x-def delta-y-def rotations-def
    apply(simp add: zeros two-not-zero)
    apply(elim disjE,safe)
    apply(simp-all add: algebra-simps divide-simps t-nz zeros)
    using eq-neg-iff-add-eq-0 apply force
    using eq-neg-iff-add-eq-0 apply force
    using r-expr unfolding rotations-def
    apply(simp add: zeros two-not-zero)
    apply(elim disjE,safe)
    apply(simp-all add: algebra-simps divide-simps t-nz zeros)
    using r-expr unfolding rotations-def
    apply(simp add: zeros two-not-zero)
    apply(elim disjE,safe)
    apply(simp-all add: algebra-simps divide-simps t-nz zeros)
    apply(simp add: zeros two-not-zero)
    using r-expr unfolding rotations-def
    apply(simp add: zeros two-not-zero)
```

```
  apply(elim disjE,safe)
  by(simp-all add: algebra-simps divide-simps t-nz zeros)

have eq: gluing `` {((τ ∘ r ∘ i) (x, y), l')} =
           gluing `` {((r ∘ i) (x, y), l'+1)}
    apply(subst gluing-inv[of fst ((r ∘ i) (x, y)) snd ((r ∘ i) (x, y)) l'+1,
                  simplified prod.collapse])
    using zeros r-expr unfolding rotations-def apply fastforce+
    using i-aff[of (x,y),OF in-aff(1)] rot-aff[OF r-expr(2)] apply fastforce
    by force
have e-proj': gluing `` {(ϱ (x, y), l)} ∈ e-proj
             gluing `` {((r ∘ i) (x, y), l' + 1)} ∈ e-proj
    using e-proj(1) insert-rho-gluing rho-preserv-e-proj apply auto[1]
    using e-proj(2) eq r-expr(1) by auto
{
  assume True: delta x y x (−y) ≠ 0
  have 1: add (ϱ (x, y)) ((r ∘ i) (x, y)) = (ϱ ∘ r) (1,0)
    (is ?lhs = ?rhs)
  proof −
    have ?lhs = ϱ (add (x, y) (r (i (x, y))))
      using rho-invariance-1-points o-apply[of r i] by presburger
    also have ... = (ϱ ∘ r) (add (x, y) (i (x, y)))
      using rotation-invariance-1-points[OF
             r-expr(2),simplified commutativity] by fastforce
    also have ... = ?rhs
      using inverse-generalized[OF in-aff(1)] True in-aff
      unfolding delta-def delta-plus-def delta-minus-def by simp
    finally show ?thesis by auto
  qed
}
note add-case = this
{
  assume us-ds: delta' x y x (−y) ≠ 0
  have 2: ext-add (ϱ (x, y)) ((r ∘ i) (x, y)) = (ϱ ∘ r) (1,0)
    (is ?lhs = ?rhs)
  proof −
    have ?lhs = ϱ (ext-add (x, y) (r (i (x, y))))
      using rho-invariance-2-points o-apply[of r i] by presburger
    also have ... = (ϱ ∘ r) (ext-add (x, y) (i (x, y)))
      using rotation-invariance-2-points[OF
             r-expr(2),simplified ext-add-comm-points] by force
    also have ... = ?rhs
      using ext-add-inverse[OF zeros] by argo
    finally show ?thesis by auto
  qed
}
note ext-add-case = this

have simp1: proj-addition (gluing `` {(ϱ (x, y), l)})
```

```
                    (gluing `` {((r ∘ i) (x, y), l' + 1)})=
         gluing `` {((ϱ ∘ r) (1,0),l+l'+1)}
      (is proj-addition ?g1 ?g2 = ?g3)
    proof(cases ?ds3 ≠ 0)
      case True
      then have delta x y x (−y) ≠ 0 using ds by blast
      then have 1: add (ϱ (x, y)) ((r ∘ i) (x, y)) = (ϱ ∘ r) (1,0)
        using add-case by auto
      have proj-addition ?g1 ?g2 =
                gluing `` {(add (ϱ (x, y)) ((r ∘ i) (x, y)), l+l'+1)}
          using gluing-add[of fst (ϱ (x, y)) snd (ϱ (x, y)) l
                            fst ((r ∘ i) (x, y)) snd ((r ∘ i) (x, y)) l'+1,
                          simplified prod.collapse, OF e-proj' ] True
          by (simp add: add.assoc)
        also have ... = ?g3
          using 1 by auto
        finally show ?thesis by auto
    next
      case False
      then have delta' x y x (−y) ≠ 0 using ds ds''' by fast
      then have 2: ext-add (ϱ (x, y)) ((r ∘ i) (x, y)) = (ϱ ∘ r) (1,0)
        using ext-add-case by auto
      then have proj-addition ?g1 ?g2 =
                gluing `` {(ext-add (ϱ (x, y)) ((r ∘ i) (x, y)), l+l'+1)}
          using gluing-ext-add[of fst (ϱ (x, y)) snd (ϱ (x, y)) l
                            fst ((r ∘ i) (x, y)) snd ((r ∘ i) (x, y)) l'+1,
                          simplified prod.collapse, OF e-proj'] False
          by (metis (no-types, lifting) add.assoc ds''')
        also have ... = ?g3
          using 2 by auto
        finally show ?thesis by auto
    qed

    have e-proj': gluing `` {((x, y), l)} ∈ e-proj
                gluing `` {((r ∘ i) (x, y), l' + 1)} ∈ e-proj
      using e-proj apply auto[1]
      using e-proj(2) eq r-expr(1) by auto
    have simp2: tf ϱ
     (proj-addition (gluing `` {((x, y), l)})
       (gluing `` {((r ∘ i) (x, y), l'+1)})) =
      gluing `` {((ϱ ∘ r) (1,0),l+l'+1)}
      (is tf - (proj-addition ?g1 ?g2) = ?g3)
    proof(cases ?ds1 ≠ 0)
      case True
      then have us-ds: delta x y x (−y) ≠ 0 using ds by blast
      then have 1: add (x, y) ((r ∘ i) (x, y)) = r (1,0)
         using add-case rho-invariance-1[of x y fst ((r ∘ i) (x, y)) snd ((r ∘ i) (x,
y)),
                                    simplified prod.collapse]
```

```
        by (metis comp-apply i-idemp-explicit inverse-rule-2 prod.exhaust-sel)
      have proj-addition ?g1 ?g2 =
              gluing ‘‘ {(add (x, y) ((r ∘ i) (x, y)), l+l'+1)}
        using gluing-add[of x y l
                        fst ((r ∘ i) (x, y)) snd ((r ∘ i) (x, y)) l'+1,
                     simplified prod.collapse, OF e-proj'] True
        by (metis (no-types, lifting) is-num-normalize(1))
      also have ... = gluing ‘‘ {(r (1, 0), l + l' + 1)}
        using 1 by presburger
      finally have eq': proj-addition ?g1 ?g2 = gluing ‘‘ {(r (1, 0), l + l' + 1)}
        by auto
      show ?thesis
        apply(subst eq')
        apply(subst remove-rho[symmetric, of fst (r (1,0)) snd (r (1,0)),
                    simplified prod.collapse])
        using e-proj' eq' well-defined by force+
    next
      case False
      then have us-ds: delta' x y x (−y) ≠ 0 using ds ds'' by argo
      then have 2: ext-add (x, y) ((r ∘ i) (x, y)) = r (1,0)
      using ext-add-comm-points ext-add-inverse r-expr(2) rotation-invariance-2-points
zeros by auto
      have proj-addition ?g1 ?g2 =
              gluing ‘‘ {(ext-add (x, y) ((r ∘ i) (x, y)), l+l'+1)}
        using gluing-ext-add e-proj' False
          by (metis (no-types, lifting) add.assoc ds'' prod.collapse)
      also have ... = gluing ‘‘ {(r (1, 0), l + l' + 1)}
        using 2 by auto
      finally have eq': proj-addition ?g1 ?g2 = gluing ‘‘ {(r (1, 0), l + l' + 1)}
        by auto
      then show ?thesis
        apply(subst eq')
        apply(subst remove-rho[symmetric, of fst (r (1,0)) snd (r (1,0)),
                    simplified prod.collapse])
        using e-proj' eq' well-defined by force+
    qed
    show ?thesis
      unfolding p-q-expr
      unfolding remove-rho[OF e-proj(1),symmetric] r-expr eq
      unfolding simp1 simp2 by blast
next
  case b
    then have ds: delta x y x' y' ≠ 0
      unfolding e'-aff-0-def by auto
    have eq1: proj-addition (tf ϱ (gluing ‘‘ {((x, y), l)}))
                     (gluing ‘‘ {((x', y'), l')}) =
          gluing ‘‘ {(add (ϱ (x,y)) (x', y'), l+l')}
      apply(subst insert-rho-gluing)
      using e-proj apply simp
```

```
    apply(subst gluing-add[of fst (ϱ (x,y)) snd (ϱ (x,y)) l
                  x′ y′ l′,simplified prod.collapse])
    using rho-e-proj apply simp
    using e-proj apply simp
    using ds unfolding delta-def delta-plus-def delta-minus-def
    apply(simp add: algebra-simps)
    by auto

  have eq2: tf ϱ
   (proj-addition (gluing ‘‘ {((x, y), l)}) (gluing ‘‘ {((x′, y′), l′)})) =
   gluing ‘‘ {(add (ϱ (x,y)) (x′, y′), l+l′)}
    apply(subst gluing-add)
    using e-proj ds apply blast+
    apply(subst rho-invariance-1-points)
    apply(subst insert-rho-gluing[of fst (add (x, y) (x′, y′))
                          snd (add (x, y) (x′, y′)) l+l′,
                       simplified prod.collapse])
    using add-closure-points in-aff ds e-proj-aff apply force
    by auto

  then show ?thesis
    unfolding p-q-expr
    using eq1 eq2 by auto
next
  case c
  then have ds: delta′ x y x′ y′ ≠ 0
    unfolding e′-aff-1-def by auto
  have eq1: proj-addition (tf ϱ (gluing ‘‘ {((x, y), l)}))
                  (gluing ‘‘ {((x′, y′), l′)}) =
        gluing ‘‘ {(ext-add (ϱ (x,y)) (x′, y′), l+l′)}
    apply(subst insert-rho-gluing)
    using e-proj apply simp
    apply(subst gluing-ext-add[of fst (ϱ (x,y)) snd (ϱ (x,y)) l
                  x′ y′ l′,simplified prod.collapse])
    using rho-e-proj apply simp
    using e-proj apply simp
    using ds unfolding delta′-def delta-x-def delta-y-def
    apply(simp add: algebra-simps)
    by auto

  have eq2: tf ϱ
   (proj-addition (gluing ‘‘ {((x, y), l)}) (gluing ‘‘ {((x′, y′), l′)})) =
   gluing ‘‘ {(ext-add (ϱ (x,y)) (x′, y′), l+l′)}
    apply(subst gluing-ext-add)
    using e-proj ds apply blast+
    apply(subst rho-invariance-2-points)
    apply(subst insert-rho-gluing[of fst (ext-add (x, y) (x′, y′))
                          snd (ext-add (x, y) (x′, y′)) l+l′,
                       simplified prod.collapse])
```

```
      using ext-add-closure in-aff ds e-proj-aff
      unfolding e'-aff-def
      by auto

    then show ?thesis
      unfolding p-q-expr
      using eq1 eq2 by auto
  qed
qed

lemma remove-add-rotation:
  assumes p ∈ e-proj q ∈ e-proj r ∈ rotations
  shows proj-addition (tf r p) q = tf r (proj-addition p q)
proof −
  obtain x y l x' y' l' where p-q-expr: p = gluing `` {((x, y), l)} p = gluing ``
{((x', y'), l')}
    by (metis assms(1) e-proj-def prod.collapse quotientE)
  consider (1) r = id | (2) r = ϱ | (3) r = ϱ ∘ ϱ | (4) r = ϱ ∘ ϱ ∘ ϱ
    using assms(3) unfolding rotations-def by fast
  then show ?thesis
  proof(cases)
    case 1
    then show ?thesis using tf-id by metis
  next
    case 2
    then show ?thesis using remove-add-rho assms(1,2) by auto
  next
    case 3
    then show ?thesis
      unfolding p-q-expr
      using remove-add-rho assms(1,2)  rho-preserv-e-proj insert-rho-gluing
      by (metis (no-types, lifting) p-q-expr(1) tf-comp)
  next
    case 4
    then show ?thesis
      unfolding p-q-expr
      using remove-add-rho assms(1,2) rho-preserv-e-proj insert-rho-gluing
      by (smt ϱ.simps p-q-expr(1) p-q-expr(2) tf-comp)
  qed
qed

lemma remove-add-tau:
  assumes p ∈ e-proj q ∈ e-proj
  shows proj-addition (tf' p) q = tf' (proj-addition p q)
proof −
  obtain x y l x' y' l' where
    p-q-expr: p = gluing `` {((x, y), l)}
              q = gluing `` {((x', y'), l')}
    using assms
```

```
    unfolding e-proj-def
    apply(elim quotientE)
    by force
  have e-proj:
    gluing `` {((x, y), s)} ∈ e-proj
    gluing `` {((x', y'), s')} ∈ e-proj for s s'
    using p-q-expr assms e-proj-aff by auto
  then have i-proj:
    gluing `` {(i (x, y), l'+1)} ∈ e-proj
    using proj-add-class-inv(2) by auto

  have in-aff: (x,y) ∈ e'-aff (x',y') ∈ e'-aff
    using assms p-q-expr e-proj-aff by auto

  have other-proj:
    gluing `` {((x, y), l+1)} ∈ e-proj
    using in-aff e-proj-aff by auto

  consider
    (a) (x, y) ∈ e-circ ∧ (∃ g∈symmetries. (x', y') = (g ∘ i) (x, y)) |
    (b) ((x, y), x', y') ∈ e'-aff-0 ¬ ((x, y) ∈ e-circ ∧ (∃ g∈symmetries. (x', y') =
(g ∘ i) (x, y))) |
    (c) ((x, y), x', y') ∈ e'-aff-1 ¬ ((x, y) ∈ e-circ ∧ (∃ g∈symmetries. (x', y') =
(g ∘ i) (x, y))) ((x, y), x', y') ∉ e'-aff-0
    using dichotomy-1[OF ‹(x,y) ∈ e'-aff› ‹(x',y') ∈ e'-aff›] by argo
  then show ?thesis
  proof(cases)
    case a
    then have e-circ: (x,y) ∈ e-circ by auto
    then have zeros: x ≠ 0 y ≠ 0 unfolding e-circ-def by auto
    from a obtain g where g-expr:
      g ∈ symmetries (x', y') = (g ∘ i) (x, y) by blast
    then obtain r where r-expr: (x', y') = (τ ∘ r ∘ i) (x, y) r ∈ rotations
      using sym-decomp by blast
    have eq: gluing `` {((τ ∘ r ∘ i) (x, y),s)} =
              gluing `` {((r ∘ i) (x, y), s+1)} for s
        apply(subst gluing-inv[of fst ((r ∘ i) (x, y)) snd ((r ∘ i) (x, y)) s+1,
                    simplified prod.collapse])
        using zeros r-expr unfolding rotations-def apply fastforce+
        using i-aff[of (x,y),OF in-aff(1)] rot-aff[OF r-expr(2)] apply fastforce
        by force

    have proj-addition (tf' (gluing `` {((x, y), l)}))
                    (gluing `` {((x', y'), l')}) =
          proj-addition (gluing `` {((x, y), l+1)})
                    (gluing `` {((τ ∘ r ∘ i) (x, y), l')})
      (is ?lhs = -)
      using assms(1) p-q-expr(1) tf-tau r-expr by auto
    also have ... =
```

proj-addition (*gluing* ‘‘ {((*x*, *y*), *l*+*1*)})
(*gluing* ‘‘ {(*r* (*i* (*x*, *y*)), *l*′+*1*)})
using *eq* **by** *auto*
also have ... =
tf r (*proj-addition* (*gluing* ‘‘ {((*x*, *y*), *l*+*1*)})
(*gluing* ‘‘ {(*i* (*x*, *y*), *l*′+*1*)}))
proof −
note *lem1* = *remove-rotations*[*of fst* (*i* (*x*,*y*)) *snd* (*i* (*x*,*y*)) *l*′+*1*,
OF - *r-expr*(*2*), *simplified prod.collapse*, *OF i-proj*]
show *?thesis*
apply(*subst lem1*)
apply(*subst proj-addition-comm*)
using *other-proj* **apply** *simp*
using *lem1 assms*(*2*) *eq p-q-expr*(*2*) *r-expr*(*1*) **apply** *auto*[*1*]
apply(*subst remove-add-rotation*[*OF* - - *r-expr*(*2*)])
using *i-proj other-proj* **apply**(*simp*,*simp*)
apply(*subst proj-addition-comm*)
using *i-proj other-proj* **by** *auto*
qed
also have ... = *tf r* {((*1*,*0*),*l*+*l*′)}
(**is** - = *?rhs*)
using *proj-add-class-inv*(*1*)[*OF other-proj*, *of l*′+*1*] **by** *force*
finally have *simp1*: *?lhs* = *?rhs*
by *auto*

have *tf*′ (*proj-addition* (*gluing* ‘‘ {((*x*, *y*), *l*)})
(*gluing* ‘‘ {((*x*′, *y*′), *l*′)})) =
tf′ (*proj-addition* (*gluing* ‘‘ {((*x*, *y*), *l*)})
(*gluing* ‘‘ {((*τ* ∘ *r* ∘ *i*) (*x*, *y*), *l*′)}))
(**is** *?lhs* = -)
using *assms*(*1*) *p-q-expr*(*1*) *tf-tau r-expr* **by** *auto*
also have ... =
tf′ (*proj-addition* (*gluing* ‘‘ {((*x*, *y*), *l*)})
(*gluing* ‘‘ {(*r* (*i* (*x*, *y*)), *l*′+*1*)}))
using *eq* **by** *auto*
also have ... =
tf r {((*1*, *0*), *l* + *l*′)}
proof −
note *lem1* = *remove-rotations*[*of fst* (*i* (*x*,*y*)) *snd* (*i* (*x*,*y*)) *l*′+*1*,
OF - *r-expr*(*2*), *simplified prod.collapse*, *OF i-proj*]
show *?thesis*
apply(*subst lem1*)
apply(*subst proj-addition-comm*)
using *i-proj e-proj* **apply**(*simp*,*simp*)
apply (*simp add*: *r-expr*(*2*) *rotation-preserv-e-proj*)
apply(*subst remove-add-rotation*[*OF* - - *r-expr*(*2*)])
using *i-proj e-proj* **apply**(*simp*,*simp*)
apply(*subst proj-addition-comm*)
using *i-proj e-proj* **apply**(*simp*,*simp*)

```
      apply(subst proj-add-class-inv(1))
      using e-proj apply simp
      apply(subst tf-tf'-commute[symmetric])
      apply(subst identity-equiv[symmetric])
      apply(subst tf-tau[symmetric])
      apply (simp add: identity-equiv identity-proj)
      apply(subst identity-equiv)
      by auto
    qed
    finally have simp2: ?lhs = ?rhs
      by auto

    show ?thesis
      unfolding p-q-expr
      unfolding remove-rho[OF e-proj(1),symmetric]
      unfolding simp1 simp2 by auto
  next
    case b
    then have ds: delta x y x' y' ≠ 0
      unfolding e'-aff-0-def by auto
    have add-proj: gluing `` {(add (x, y) (x', y'), s)} ∈ e-proj for s
      using e-proj add-closure-points ds e-proj-aff by auto
    show ?thesis
      unfolding p-q-expr
      apply(subst tf-tau[symmetric],simp add: e-proj)
      apply(subst (1 2) gluing-add,
          (simp add: e-proj ds other-proj add-proj del: add.simps)+)
      apply(subst tf-tau[of fst (add (x, y) (x', y'))
                   snd (add (x, y) (x', y')),simplified prod.collapse,symmetric],
           simp add: add-proj del: add.simps)
      by(simp add: algebra-simps)
  next
    case c
    then have ds: delta' x y x' y' ≠ 0
      unfolding e'-aff-1-def by auto
    have add-proj: gluing `` {(ext-add (x, y) (x', y'), s)} ∈ e-proj for s
      using e-proj ext-add-closure-points ds e-proj-aff by auto
    show ?thesis
      unfolding p-q-expr
      apply(subst tf-tau[symmetric],simp add: e-proj)
      apply(subst (1 2) gluing-ext-add,
          (simp add: e-proj ds other-proj add-proj del: ext-add.simps)+)
      apply(subst tf-tau[of fst (ext-add (x, y) (x', y'))
                   snd (ext-add (x, y) (x', y')),simplified prod.collapse,symmetric],
           simp add: add-proj del: ext-add.simps)
      by(simp add: algebra-simps)
  qed
qed
```

lemma *remove-add-tau′*:
assumes $p \in$ *e-proj* $q \in$ *e-proj*
shows *proj-addition* p (*tf′* q) $=$ *tf′* (*proj-addition* p q)
using *assms proj-addition-comm remove-add-tau*
by (*metis proj-add-class.simps*(*2*) *proj-addition-def*)

lemma *tf′-idemp*:
assumes $s \in$ *e-proj*
shows *tf′* (*tf′* s) $= s$
proof −
obtain x y l **where** *p-q-expr*:
$s =$ *gluing* ‘‘ $\{((x, y), l)\}$
by (*metis assms e-proj-def prod.collapse quotientE*)
then have $s = \{((x, y), l)\} \vee s = \{((x, y), l), (\tau\ (x, y), l+1)\}$
using *assms gluing-cases-explicit* **by** *auto*
then show *?thesis*
apply(*elim disjE*)
by(*simp add*: *tf′-def*)+
qed

definition *tf′′* **where**
tf′′ g s $=$ *tf′* (*tf* g s)

lemma *remove-sym*:
assumes *gluing* ‘‘ $\{((x, y), l)\} \in$ *e-proj* *gluing* ‘‘ $\{(g\ (x, y), l)\} \in$ *e-proj* $g \in$ *symmetries*
shows *gluing* ‘‘ $\{(g\ (x, y), l)\}$ $=$ *tf′′* $(\tau \circ g)$ (*gluing* ‘‘ $\{((x, y), l)\}$)
using *assms remove-tau remove-rotations sym-decomp*
proof −
obtain r **where** *r-expr*: $r \in$ *rotations* $g = \tau \circ r$
using *assms sym-decomp* **by** *blast*
then have *e-proj*: *gluing* ‘‘ $\{(r\ (x, y), l)\} \in$ *e-proj*
using *rotation-preserv-e-proj insert-rotation-gluing assms* **by** *simp*
have *gluing* ‘‘ $\{(g\ (x, y), l)\}$ $=$ *gluing* ‘‘ $\{(\tau\ (r\ (x, y)), l)\}$
using *r-expr* **by** *simp*
also have ... $=$ *tf′* (*gluing* ‘‘ $\{(r\ (x, y), l)\}$)
using *remove-tau assms e-proj r-expr*
by (*metis calculation prod.collapse*)
also have ... $=$ *tf′* (*tf* r (*gluing* ‘‘ $\{((x, y), l)\}$))
using *remove-rotations r-expr assms*(*1*) **by** *force*
also have ... $=$ *tf′′* $(\tau \circ g)$ (*gluing* ‘‘ $\{((x, y), l)\}$)
using *r-expr*(*2*) *tf′′-def tau-idemp-explicit*
by (*metis* (*no-types*, *lifting*) *comp-assoc id-comp tau-idemp*)
finally show *?thesis* **by** *simp*
qed

lemma *remove-add-sym*:
assumes $p \in$ *e-proj* $q \in$ *e-proj* $g \in$ *rotations*
shows *proj-addition* (*tf′′* g p) q $=$ *tf′′* g (*proj-addition* p q)

proof −
  obtain *x y l x′ y′ l′* **where** *p-q-expr*: *p* = *gluing* ‘‘ {((*x*, *y*), *l*)} *q* = *gluing* ‘‘ {((*x′*, *y′*), *l′*)}
    by (*metis assms*(*1*,*2*) *e-proj-def prod.collapse quotientE*)+
  then have *e-proj*: (*tf g p*) ∈ *e-proj*
    using *rotation-preserv-e-proj assms* **by** *fast*
  have *proj-addition* (*tf′′ g p*) *q* = *proj-addition* (*tf′* (*tf g p*)) *q*
    unfolding *tf′′-def* **by** *simp*
  also have ... = *tf′* (*proj-addition* (*tf g p*) *q*)
    using *remove-add-tau assms e-proj* **by** *blast*
  also have ... = *tf′* (*tf g* (*proj-addition p q*))
    using *remove-add-rotation assms* **by** *presburger*
  also have ... = *tf′′ g* (*proj-addition p q*)
    using *tf′′-def* **by** *auto*
  finally show *?thesis* **by** *simp*
qed

lemma *tf′′-preserv-e-proj*:
  assumes *gluing* ‘‘ {((*x*,*y*),*l*)} ∈ *e-proj r* ∈ *rotations*
  shows *tf′′ r* (*gluing* ‘‘ {((*x*,*y*),*l*)}) ∈ *e-proj*
  unfolding *tf′′-def*
  apply(*subst insert-rotation-gluing*[*OF assms*])
  using *rotation-preserv-e-proj*[*OF assms*] *tf-preserv-e-proj insert-rotation-gluing*[*OF assms*]
  by (*metis i.cases*)

lemma *tf′-injective*:
  assumes *c1* ∈ *e-proj c2* ∈ *e-proj*
  assumes *tf′* (*c1*) = *tf′* (*c2*)
  shows *c1* = *c2*
  using *assms* **by** (*metis tf′-idemp*)

4.2 Associativities

lemma *add-add-add-add-assoc*:
  assumes (*x1*,*y1*) ∈ *e′-aff* (*x2*,*y2*) ∈ *e′-aff* (*x3*,*y3*) ∈ *e′-aff*
  assumes *delta x1 y1 x2 y2* ≠ *0 delta x2 y2 x3 y3* ≠ *0*
    delta (*fst* (*add* (*x1*,*y1*) (*x2*,*y2*))) (*snd* (*add* (*x1*,*y1*) (*x2*,*y2*))) *x3 y3* ≠ *0*
    delta x1 y1 (*fst* (*add* (*x2*,*y2*) (*x3*,*y3*))) (*snd* (*add* (*x2*,*y2*) (*x3*,*y3*))) ≠ *0*
    shows *add* (*add* (*x1*,*y1*) (*x2*,*y2*)) (*x3*,*y3*) = *add* (*x1*,*y1*) (*add* (*x2*,*y2*) (*x3*,*y3*))
  using *assms* **unfolding** *e′-aff-def delta-def* **apply**(*simp*)
  using *associativity e-e′-iff* **by** *fastforce*

lemma *ext-add-hard-1*:
$x2 \neq 0 \Longrightarrow$
$y2 = 0 \Longrightarrow$
$x3 \neq 0 \Longrightarrow$
$y3 \neq 0 \Longrightarrow$
$y1 \neq 0 \Longrightarrow$
$x1 \neq 0 \Longrightarrow$
$x1 * (x1 * (x2 * (x3 * y1))) + x1 * (x2 * (y1 * (y1 * y3))) \neq 0 \Longrightarrow$
$- (x1 * (x2 * (x3 * (x3 * y3)))) \neq x2 * (x3 * (y1 * (y3 * y3))) \Longrightarrow$
$x1 * x1 + y1 * y1 = 1 + d * (x1 * (x1 * (y1 * y1))) \Longrightarrow$
$x2 * x2 = 1 \Longrightarrow$
$x3 * x3 + y3 * y3 = 1 + d * (x3 * (x3 * (y3 * y3))) \Longrightarrow$
$x3 * y1 \neq x1 * y3 \wedge x1 * x3 + y1 * y3 \neq 0 \Longrightarrow$
$x1 * (x1 * (x2 * (x3 * (x3 * (x3 * (y1 * (y3 * y3))))))) +$
$(x1 * (x2 * (x3 * (x3 * (y1 * (y1 * (y3 * (y3 * y3))))))) +$
$(x1 * (x1 * (x1 * (x2 * (x2 * (x2 * (x3 * (x3 * (y1 * (y1 * y3))))))))) +$
$x1 * (x1 * (x2 * (x2 * (x2 * (x3 * (y1 * (y1 * (y1 * (y3 * y3)))))))))) =$
$x1 * (x1 * (x1 * (x2 * (x3 * (x3 * (y1 * (y1 * y3))))))) +$
$(x1 * (x1 * (x2 * (x3 * (y1 * (y1 * (y1 * (y3 * y3))))))) +$
$(x1 * (x1 * (x2 * (x2 * (x2 * (x3 * (x3 * (x3 * (y1 * (y3 * y3))))))))) +$
$x1 * (x2 * (x2 * (x2 * (x3 * (x3 * (y1 * (y1 * (y3 * (y3 * y3))))))))))$
proof −
assume *a1*: $x2 * x2 = 1$
have *f2*: $\forall r\ ra.\ (ra::real) * r = r * ra$
by *auto*
have $\forall r.\ x2 * (r * x2) = r$
using *a1* **by** *auto*
then have $x1 * (x1 * (y1 * (x3 * (x3 * (x3 * (y3 * (x2 * y3))))))) + (x1 * (y1 * (y1 * (x3 * (x3 * (y3 * (y3 * (x2 * y3))))))) + (x1 * (x1 * (x1 * (y1 * (y1 * (x3 * (x3 * (x2 * (x2 * (x2 * y3))))))))) + x1 * (x1 * (y1 * (y1 * (y1 * (x3 * (y3 * (x2 * (x2 * (x2 * y3)))))))))) = x1 * (x1 * (x1 * (y1 * (y1 * (x3 * (x3 * (x2 * y3))))))) + (x1 * (x1 * (y1 * (y1 * (y1 * (x3 * (y3 * (x2 * y3))))))) + (x1 * (x1 * (y1 * (x3 * (x3 * (x3 * (y3 * (x2 * (x2 * (x2 * y3))))))))) + x1 * (y1 * (y1 * (x3 * (x3 * (y3 * (y3 * (x2 * (x2 * (x2 * y3))))))))))$
using *f2*
apply(*simp add*: *algebra-simps*)
by (*simp add*: *a1 semiring-normalization-rules*(*18*))
then show $x1 * (x1 * (x2 * (x3 * (x3 * (x3 * (y1 * (y3 * y3))))))) + (x1 * (x2 * (x3 * (x3 * (y1 * (y1 * (y3 * (y3 * y3))))))) + (x1 * (x1 * (x1 * (x2 * (x2 * (x2 * (x3 * (x3 * (y1 * (y1 * y3))))))))) + x1 * (x1 * (x2 * (x2 * (x2 * (x3 * (y1 * (y1 * (y1 * (y3 * y3)))))))))) = x1 * (x1 * (x1 * (x2 * (x3 * (x3 * (y1 * (y1 * y3))))))) + (x1 * (x1 * (x2 * (x3 * (y1 * (y1 * (y1 * (y3 * y3))))))) + (x1 * (x1 * (x2 * (x2 * (x2 * (x3 * (x3 * (x3 * (y1 * (y3 * y3))))))))) + x1 * (x2 * (x2 * (x2 * (x3 * (x3 * (y1 * (y1 * (y3 * (y3 * y3))))))))))$
by (*simp add*: *mult.left-commute*)
qed

lemma *ext-ext-ext-ext-assoc*:
assumes $z1' = (x1',y1')$ $z3' = (x3',y3')$

assumes $z1' = ext\text{-}add\ (x1,y1)\ (x2,y2)\ z3' = ext\text{-}add\ (x2,y2)\ (x3,y3)$
assumes $delta\text{-}x\ x1\ y1\ x2\ y2 \neq 0\ delta\text{-}y\ x1\ y1\ x2\ y2 \neq 0$
$delta\text{-}x\ x2\ y2\ x3\ y3 \neq 0\ delta\text{-}y\ x2\ y2\ x3\ y3 \neq 0$
$delta\text{-}x\ x1'\ y1'\ x3\ y3 \neq 0\ delta\text{-}y\ x1'\ y1'\ x3\ y3 \neq 0$
$delta\text{-}x\ x1\ y1\ x3'\ y3' \neq 0\ delta\text{-}y\ x1\ y1\ x3'\ y3' \neq 0$
assumes $e'\ x1\ y1 = 0\ e'\ x2\ y2 = 0\ e'\ x3\ y3 = 0$
shows $ext\text{-}add\ (ext\text{-}add\ (x1,y1)\ (x2,y2))\ (x3,y3) = ext\text{-}add\ (x1,y1)\ (ext\text{-}add\ (x2,y2)\ (x3,y3))$
proof −
define $e1$ **where** $e1 = e'\ x1\ y1$
define $e2$ **where** $e2 = e'\ x2\ y2$
define $e3$ **where** $e3 = e'\ x3\ y3$
define $Delta_x$ **where** $Delta_x =$
$(delta\text{-}x\ x1'\ y1'\ x3\ y3)*(delta\text{-}x\ x1\ y1\ x3'\ y3')*$
$(delta'\ x1\ y1\ x2\ y2)*(delta'\ x2\ y2\ x3\ y3)$
define $Delta_y$ **where** $Delta_y =$
$(delta\text{-}y\ x1'\ y1'\ x3\ y3)*(delta\text{-}y\ x1\ y1\ x3'\ y3')*$
$(delta'\ x1\ y1\ x2\ y2)*(delta'\ x2\ y2\ x3\ y3)$
define g_x **where** $g_x = fst(ext\text{-}add\ z1'\ (x3,y3)) - fst(ext\text{-}add\ (x1,y1)\ z3')$
define g_y **where** $g_y = snd(ext\text{-}add\ z1'\ (x3,y3)) - snd(ext\text{-}add\ (x1,y1)\ z3')$
define $gxpoly$ **where** $gxpoly = g_x * Delta_x$
define $gypoly$ **where** $gypoly = g_y * Delta_y$

define $gxpoly\text{-}expr$ **where** $gxpoly\text{-}expr =$
$((x1 * y1 - x2 * y2) * (x1 * y1 + x2 * y2) -$
$x3 * y3 * ((x2 * y1 - x1 * y2) * (x1 * x2 + y1 * y2))) *$
$((x2 * y2 - x3 * y3) * y1 * (x2 * x3 + y2 * y3) -$
$x1 * (x2 * y2 + x3 * y3) * (x3 * y2 - x2 * y3)) -$
$(x1 * y1 * ((x3 * y2 - x2 * y3) * (x2 * x3 + y2 * y3)) -$
$(x2 * y2 - x3 * y3) * (x2 * y2 + x3 * y3)) *$
$(x3 * (x1 * y1 + x2 * y2) * (x2 * y1 - x1 * y2) -$
$(x1 * y1 - x2 * y2) * y3 * (x1 * x2 + y1 * y2))$
define $gypoly\text{-}expr$ **where** $gypoly\text{-}expr =$
$((x1 * y1 - x2 * y2) * (x1 * y1 + x2 * y2) +$
$x3 * y3 * ((x2 * y1 - x1 * y2) * (x1 * x2 + y1 * y2))) *$
$(x1 * (x2 * y2 - x3 * y3) * (x2 * x3 + y2 * y3) +$
$y1 * (x2 * y2 + x3 * y3) * (x3 * y2 - x2 * y3)) -$
$(x1 * y1 * ((x3 * y2 - x2 * y3) * (x2 * x3 + y2 * y3)) +$
$(x2 * y2 - x3 * y3) * (x2 * y2 + x3 * y3)) *$
$((x1 * y1 - x2 * y2) * x3 * (x1 * x2 + y1 * y2) +$
$(x1 * y1 + x2 * y2) * y3 * (x2 * y1 - x1 * y2))$

have $x1'\text{-}expr$: $x1' = (x1 * y1 - x2 * y2) / (x2 * y1 - x1 * y2)$
using $assms(1,3)$ **by** $simp$
have $y1'\text{-}expr$: $y1' = (x1 * y1 + x2 * y2) / (x1 * x2 + y1 * y2)$
using $assms(1,3)$ **by** $simp$
have $x3'\text{-}expr$: $x3' = (x2 * y2 - x3 * y3) / (x3 * y2 - x2 * y3)$
using $assms(2,4)$ **by** $simp$
have $y3'\text{-}expr$: $y3' = (x2 * y2 + x3 * y3) / (x2 * x3 + y2 * y3)$

```
    using assms(2,4) by simp

  have non-unfolded-adds:
      delta' x1 y1 x2 y2 ≠ 0 using delta'-def assms(5,6) by auto

  have gx-div: ∃ r1 r2 r3. gxpoly-expr = r1 * e1 + r2 * e2 + r3 * e3
    unfolding gxpoly-expr-def e1-def e2-def e3-def e'-def by algebra

  have gy-div: ∃ r1 r2 r3. gypoly-expr = r1 * e1 + r2 * e2 + r3 * e3
    unfolding gypoly-expr-def e1-def e2-def e3-def e'-def
    by algebra

  have simp1gx:
    (x1' * y1' − x3 * y3) * delta-x x1 y1 x3' y3' * (delta' x1 y1 x2 y2 * delta' x2
y2 x3 y3) =
     ((x1 * y1 − x2 * y2) * (x1 * y1 + x2 * y2) −
     x3 * y3 * (delta-x x1 y1 x2 y2 * delta-y x1 y1 x2 y2)) *
    ((x2 * y2 − x3 * y3) * y1 * delta-y x2 y2 x3 y3 −
     x1 * (x2 * y2 + x3 * y3) * delta-x x2 y2 x3 y3)

   apply((subst x1'-expr)+, (subst y1'-expr)+,(subst x3'-expr)+,(subst y3'-expr)+)
    apply(subst (2 3 5) delta-x-def[symmetric])
    apply(subst (2 4) delta-y-def[symmetric])
    apply(subst (2 4) delta-x-def)
    unfolding delta'-def
    by(simp add: divide-simps assms(5−8))

  have simp2gx:
    (x1 * y1 − x3' * y3') * delta-x x1' y1' x3 y3 * (delta' x1 y1 x2 y2 * delta' x2
y2 x3 y3) =
     (x1 * y1 * (delta-x x2 y2 x3 y3 * delta-y x2 y2 x3 y3) −
     (x2 * y2 − x3 * y3) * (x2 * y2 + x3 * y3)) *
    (x3 * (x1 * y1 + x2 * y2) * delta-x x1 y1 x2 y2 −
     (x1 * y1 − x2 * y2) * y3 * delta-y x1 y1 x2 y2)
   apply((subst x1'-expr)+, (subst y1'-expr)+,(subst x3'-expr)+,(subst y3'-expr)+)
    apply(subst (3 5) delta-x-def[symmetric])
    apply(subst (2 4) delta-y-def[symmetric])
    apply(subst (3) delta-x-def)
    unfolding delta'-def
    by(simp add: divide-simps assms(5−8))

  have gxpoly = gxpoly-expr
    unfolding gxpoly-def g_x-def Delta_x-def
    apply(simp add: assms(1,2))
    apply(subst (2 4) delta-x-def[symmetric])+
    apply(simp add: divide-simps assms(9,11))
    apply(subst (3) left-diff-distrib)
    apply(simp add: simp1gx simp2gx)
    unfolding delta-x-def delta-y-def delta'-def
```

unfolding *gxpoly-expr-def* **by** *blast*

obtain *r1x r2x r3x* **where** *gxpoly* = *r1x* ∗ *e1* + *r2x* ∗ *e2* + *r3x* ∗ *e3*
using ‹*gxpoly* = *gxpoly-expr*› *gx-div* **by** *auto*
then have *gxpoly* = *0*
using *e1-def assms*(*13*−*15*) *e2-def e3-def* **by** *auto*
have $Delta_x \neq 0$
using $Delta_x$*-def delta′-def assms*(*7*−*11*) *non-unfolded-adds* **by** *auto*
then have $g_x = 0$
using ‹*gxpoly* = *0*› *gxpoly-def* **by** *auto*

have *simp1gy*: *delta-y x1′ x3 y1′ y3* ∗ *delta-y x1 y1 x3′ y3′* ∗ (*delta′ x1 y1 x2 y2* ∗ *delta′ x2 y2 x3 y3*) =
((*x1* ∗ *y1* − *x2* ∗ *y2*) ∗ (*x1* ∗ *y1* + *x2* ∗ *y2*) +
x3 ∗ *y3* ∗ (*delta-x x1 y1 x2 y2* ∗ *delta-y x1 y1 x2 y2*)) ∗
(*x1* ∗ (*x2* ∗ *y2* − *x3* ∗ *y3*) ∗ *delta-y x2 y2 x3 y3* +
y1 ∗ (*x2* ∗ *y2* + *x3* ∗ *y3*) ∗ *delta-x x2 y2 x3 y3*)
apply((*subst x1′-expr*)+, (*subst y1′-expr*)+,(*subst x3′-expr*)+,(*subst y3′-expr*)+)
apply(*subst* (*2 4*) *delta-x-def*[*symmetric*])
apply(*subst* (*2 4*) *delta-y-def*[*symmetric*])
apply(*subst* (*2 3*) *delta-y-def*)
unfolding *delta′-def*
by(*simp add*: *divide-simps assms*(*5*−*8*))

have *simp2gy*: *delta-y x1 x3′ y1 y3′* ∗ *delta-y x1′ y1′ x3 y3* ∗ (*delta′ x1 y1 x2 y2* ∗ *delta′ x2 y2 x3 y3*) =
(*x1* ∗ *y1* ∗ (*delta-x x2 y2 x3 y3* ∗ *delta-y x2 y2 x3 y3*) +
(*x2* ∗ *y2* − *x3* ∗ *y3*) ∗ (*x2* ∗ *y2* + *x3* ∗ *y3*)) ∗
((*x1* ∗ *y1* − *x2* ∗ *y2*) ∗ *x3* ∗ *delta-y x1 y1 x2 y2* +
(*x1* ∗ *y1* + *x2* ∗ *y2*) ∗ *y3* ∗ *delta-x x1 y1 x2 y2*)
apply((*subst x1′-expr*)+, (*subst y1′-expr*)+,(*subst x3′-expr*)+,(*subst y3′-expr*)+)
apply(*subst* (*2 4*) *delta-x-def*[*symmetric*])
apply(*subst* (*2 4*) *delta-y-def*[*symmetric*])
apply(*subst* (*1 4*) *delta-y-def*)
unfolding *delta′-def*
by(*simp add*: *divide-simps assms*(*5*−*8*))

have *gypoly* = *gypoly-expr*
unfolding *gypoly-def* g_y*-def* $Delta_y$*-def*
apply(*simp add*: *assms*(*1*,*2*))
apply(*subst delta-y-def*[*symmetric*])+
apply(*simp add*: *divide-simps assms*(*10*,*12*))
apply(*subst left-diff-distrib*)
apply(*simp add*: *simp1gy simp2gy*)
unfolding *delta-x-def delta-y-def*
unfolding *gypoly-expr-def* **by** *blast*

obtain *r1y r2y r3y* **where** *gypoly* = *r1y* ∗ *e1* + *r2y* ∗ *e2* + *r3y* ∗ *e3*
using ‹*gypoly* = *gypoly-expr*› *gy-div* **by** *auto*

then have *gypoly* = *0*
  using *e1-def assms*(*13*−*15*) *e2-def e3-def* **by** *auto*
have $Delta_y \neq 0$
  using $Delta_y$*-def delta′-def assms*(*7*−*12*) *non-unfolded-adds* **by** *auto*
then have $g_y = 0$
  using ⟨*gypoly* = *0*⟩ *gypoly-def* **by** *auto*

show *?thesis*
  using ⟨$g_y = 0$⟩ ⟨$g_x = 0$⟩
  unfolding g_x*-def* g_y*-def assms*(*3*,*4*)
  by (*simp add*: *prod-eq-iff*)
qed

lemma *ext-ext-ext-add-assoc*:
  assumes *z1′* = (*x1′*,*y1′*) *z3′* = (*x3′*,*y3′*)
  assumes *z1′* = *ext-add* (*x1*,*y1*) (*x2*,*y2*) *z3′* = *add* (*x2*,*y2*) (*x3*,*y3*)
  assumes *delta-x x1 y1 x2 y2* ≠ *0 delta-y x1 y1 x2 y2* ≠ *0*
    delta-minus x2 y2 x3 y3 ≠ *0 delta-plus x2 y2 x3 y3* ≠ *0*
    delta-x x1′ y1′ x3 y3 ≠ *0 delta-y x1′ y1′ x3 y3* ≠ *0*
    delta-x x1 y1 x3′ y3′ ≠ *0 delta-y x1 y1 x3′ y3′* ≠ *0*
  assumes *e′ x1 y1* = *0 e′ x2 y2* = *0 e′ x3 y3* = *0*
  shows *ext-add* (*ext-add* (*x1*,*y1*) (*x2*,*y2*)) (*x3*,*y3*) = *ext-add* (*x1*,*y1*) (*add* (*x2*,*y2*) (*x3*,*y3*))
proof −
  define *e1* **where** *e1* = *e′ x1 y1*
  define *e2* **where** *e2* = *e′ x2 y2*
  define *e3* **where** *e3* = *e′ x3 y3*
  define $Delta_x$ **where** $Delta_x$ =
  (*delta-x x1′ y1′ x3 y3*)∗(*delta-x x1 y1 x3′ y3′*)∗
  (*delta′ x1 y1 x2 y2*)∗(*delta x2 y2 x3 y3*)
  define $Delta_y$ **where** $Delta_y$ =
  (*delta-y x1′ y1′ x3 y3*)∗(*delta-y x1 y1 x3′ y3′*)∗
  (*delta′ x1 y1 x2 y2*)∗(*delta x2 y2 x3 y3*)
  define g_x **where** g_x = *fst*(*ext-add z1′* (*x3*,*y3*)) − *fst*(*ext-add* (*x1*,*y1*) *z3′*)
  define g_y **where** g_y = *snd*(*ext-add z1′* (*x3*,*y3*)) − *snd*(*ext-add* (*x1*,*y1*) *z3′*)
  define *gxpoly* **where** *gxpoly* = $g_x * Delta_x$
  define *gypoly* **where** *gypoly* = $g_y * Delta_y$

  have *x1′-expr*: *x1′* = (*x1* ∗ *y1* − *x2* ∗ *y2*) / (*x2* ∗ *y1* − *x1* ∗ *y2*)
    using *assms*(*1*,*3*) **by** *simp*
  have *y1′-expr*: *y1′* = (*x1* ∗ *y1* + *x2* ∗ *y2*) / (*x1* ∗ *x2* + *y1* ∗ *y2*)
    using *assms*(*1*,*3*) **by** *simp*
  have *x3′-expr*: *x3′* = (*x2* ∗ *x3* − *c* ∗ *y2* ∗ *y3*) / (*1* − *d* ∗ *x2* ∗ *y2* ∗ *x3* ∗ *y3*)
    using *assms*(*2*,*4*) **by** *simp*
  have *y3′-expr*: *y3′* = (*x2* ∗ *y3* + *y2* ∗ *x3*) / (*1* + *d* ∗ *x2* ∗ *y2* ∗ *x3* ∗ *y3*)
    using *assms*(*2*,*4*) **by** *simp*

  have *non-unfolded-adds*:

delta' x1 y1 x2 y2 ≠ *0* **using** *delta'-def assms*(*5*,*6*) **by** *auto*

have *simp1gx*:
(x1' ∗ y1' − x3 ∗ y3) ∗ delta-x x1 y1 x3' y3' ∗
(delta' x1 y1 x2 y2 ∗ delta x2 y2 x3 y3) =
((x1 ∗ y1 − x2 ∗ y2) ∗ (x1 ∗ y1 + x2 ∗ y2) −
x3 ∗ y3 ∗ (delta-x x1 y1 x2 y2 ∗ delta-y x1 y1 x2 y2)) ∗
((x2 ∗ x3 − c ∗ y2 ∗ y3) ∗ y1 ∗ delta-plus x2 y2 x3 y3 −
x1 ∗ (x2 ∗ y3 + y2 ∗ x3) ∗ delta-minus x2 y2 x3 y3)

apply((*subst x1'-expr*)+, (*subst y1'-expr*)+,(*subst x3'-expr*)+,(*subst y3'-expr*)+)
apply(*subst delta-x-def*)
apply(*subst* (*2*) *delta-x-def*[*symmetric*])
apply(*subst* (*2*) *delta-y-def*[*symmetric*])
apply(*subst* (*1*) *delta-minus-def*[*symmetric*])
apply(*subst* (*1*) *delta-plus-def*[*symmetric*])
unfolding *delta'-def delta-def*
by(*simp add*: *divide-simps assms*(*5*−*8*))

have *simp2gx*:
(x1 ∗ y1 − x3' ∗ y3') ∗ delta-x x1' y1' x3 y3 ∗
(delta' x1 y1 x2 y2 ∗ delta x2 y2 x3 y3) =
(x1 ∗ y1 ∗ (delta-minus x2 y2 x3 y3 ∗ delta-plus x2 y2 x3 y3) −
(x2 ∗ x3 − c ∗ y2 ∗ y3) ∗ (x2 ∗ y3 + y2 ∗ x3)) ∗
(x3 ∗ (x1 ∗ y1 + x2 ∗ y2) ∗ delta-x x1 y1 x2 y2 −
(x1 ∗ y1 − x2 ∗ y2) ∗ y3 ∗ delta-y x1 y1 x2 y2)
apply((*subst x1'-expr*)+, (*subst y1'-expr*)+,(*subst x3'-expr*)+,(*subst y3'-expr*)+)
apply(*subst delta-x-def*)
apply(*subst* (*5*) *delta-x-def*[*symmetric*])
apply(*subst* (*3*) *delta-y-def*[*symmetric*])
apply(*subst* (*1*) *delta-minus-def*[*symmetric*])
apply(*subst* (*1*) *delta-plus-def*[*symmetric*])
unfolding *delta'-def delta-def*
by(*simp add*: *divide-simps assms*(*5*−*8*))

have ∃ *r1 r2 r3*. *gxpoly* = *r1* ∗ *e1* + *r2* ∗ *e2* + *r3* ∗ *e3*
unfolding *gxpoly-def* g_x*-def* $Delta_x$*-def*
apply(*simp add*: *assms*(*1*,*2*))
apply(*subst* (*2 4*) *delta-x-def*[*symmetric*])+
apply(*simp add*: *divide-simps assms*(*9*,*11*))
apply(*subst* (*3*) *left-diff-distrib*)
apply(*simp add*: *simp1gx simp2gx*)
unfolding *delta-x-def delta-y-def delta-plus-def delta-minus-def*
e1-def e2-def e3-def e'-def
by(*simp add*: *c-eq-1 t-expr*,*algebra*)

then have *gxpoly* = *0*
using *e1-def assms*(*13*−*15*) *e2-def e3-def* **by** *auto*
have $Delta_x$ ≠ *0*

```
  using Delta_x-def delta'-def delta-def assms(7−11) non-unfolded-adds by auto
then have g_x = 0
  using ‹gxpoly = 0› gxpoly-def by auto

have simp1gy: (x1' ∗ y1' + x3 ∗ y3) ∗ delta-y x1 y1 x3' y3' ∗
     (delta' x1 y1 x2 y2 ∗ delta x2 y2 x3 y3) =
   ((x1 ∗ y1 − x2 ∗ y2) ∗ (x1 ∗ y1 + x2 ∗ y2) +
   x3 ∗ y3 ∗ (delta-x x1 y1 x2 y2 ∗ delta-y x1 y1 x2 y2)) ∗
  (x1 ∗ (x2 ∗ x3 − c ∗ y2 ∗ y3) ∗ delta-plus x2 y2 x3 y3 +
   y1 ∗ (x2 ∗ y3 + y2 ∗ x3) ∗ delta-minus x2 y2 x3 y3)
 apply((subst x1'-expr)+, (subst y1'-expr)+,(subst x3'-expr)+,(subst y3'-expr)+)
  apply(subst delta-y-def)
  thm assms(5−8)
  apply(rewrite at x2 ∗ y1 − x1 ∗ y2
                 delta-x-def[symmetric])

  apply(subst (2) delta-y-def[symmetric])
  apply(subst (1) delta-minus-def[symmetric])
  apply(subst (1) delta-plus-def[symmetric])
  unfolding delta'-def delta-def
  by(simp add: divide-simps assms(5−8))

have simp2gy: (x1 ∗ y1 + x3' ∗ y3') ∗ delta-y x1' y1' x3 y3 ∗
     (delta' x1 y1 x2 y2 ∗ delta x2 y2 x3 y3) =
   (x1 ∗ y1 ∗ (delta-minus x2 y2 x3 y3 ∗ delta-plus x2 y2 x3 y3) +
   (x2 ∗ x3 − c ∗ y2 ∗ y3) ∗ (x2 ∗ y3 + y2 ∗ x3)) ∗
  ((x1 ∗ y1 − x2 ∗ y2) ∗ x3 ∗ delta-y x1 y1 x2 y2 +
   (x1 ∗ y1 + x2 ∗ y2) ∗ y3 ∗ delta-x x1 y1 x2 y2)
 apply((subst x1'-expr)+, (subst y1'-expr)+,(subst x3'-expr)+,(subst y3'-expr)+)
  apply(subst delta-y-def)
  apply(subst (3) delta-x-def[symmetric])
  apply(subst (5) delta-y-def[symmetric])
  apply(subst (1) delta-minus-def[symmetric])
  apply(subst (1) delta-plus-def[symmetric])
  unfolding delta'-def delta-def
  by(simp add: divide-simps assms(5−8))

have ∃ r1 r2 r3. gypoly = r1 ∗ e1 + r2 ∗ e2 + r3 ∗ e3
  unfolding gypoly-def g_y-def Delta_y-def
  apply(simp add: assms(1,2))
  apply(subst (2 4) delta-y-def[symmetric])
  apply(simp add: divide-simps assms(10,12))
  apply(subst left-diff-distrib)
  apply(simp add: simp1gy simp2gy)
  unfolding delta-x-def delta-y-def delta-plus-def delta-minus-def
           e1-def e2-def e3-def e'-def
  by(simp add: c-eq-1 t-expr,algebra)

then have gypoly = 0
```

using *e1-def assms*(*13−15*) *e2-def e3-def* **by** *auto*
have $Delta_y \neq 0$
using $Delta_y$*-def delta′-def delta-def assms*(*7−12*) *non-unfolded-adds* **by** *auto*
then have $g_y = 0$
using ‹*gypoly* = *0*› *gypoly-def* **by** *auto*

show *?thesis*
using ‹$g_y = 0$› ‹$g_x = 0$›
unfolding g_x*-def* g_y*-def assms*(*3*,*4*)
by (*simp add*: *prod-eq-iff*)
qed

lemma *ext-add-ext-ext-assoc*:
assumes *z1′* = (*x1′*,*y1′*) *z3′* = (*x3′*,*y3′*)
assumes *z1′* = *add* (*x1*,*y1*) (*x2*,*y2*) *z3′* = *ext-add* (*x2*,*y2*) (*x3*,*y3*)
assumes *delta-minus x1 y1 x2 y2* ≠ *0 delta-plus x1 y1 x2 y2* ≠ *0*
delta-x x2 y2 x3 y3 ≠ *0 delta-y x2 y2 x3 y3* ≠ *0*
delta-x x1′ y1′ x3 y3 ≠ *0 delta-y x1′ y1′ x3 y3* ≠ *0*
delta-x x1 y1 x3′ y3′ ≠ *0 delta-y x1 y1 x3′ y3′* ≠ *0*
assumes *e′ x1 y1* = *0 e′ x2 y2* = *0 e′ x3 y3* = *0*
shows *ext-add* (*add* (*x1*,*y1*) (*x2*,*y2*)) (*x3*,*y3*) = *ext-add* (*x1*,*y1*) (*ext-add* (*x2*,*y2*) (*x3*,*y3*))
proof −
define *e1* **where** *e1* = *e′ x1 y1*
define *e2* **where** *e2* = *e′ x2 y2*
define *e3* **where** *e3* = *e′ x3 y3*
define $Delta_x$ **where** $Delta_x$ =
(*delta-x x1′ y1′ x3 y3*)∗(*delta-x x1 y1 x3′ y3′*)∗
(*delta x1 y1 x2 y2*)∗(*delta′ x2 y2 x3 y3*)
define $Delta_y$ **where** $Delta_y$ =
(*delta-y x1′ y1′ x3 y3*)∗(*delta-y x1 y1 x3′ y3′*)∗
(*delta x1 y1 x2 y2*)∗(*delta′ x2 y2 x3 y3*)
define g_x **where** g_x = *fst*(*ext-add z1′* (*x3*,*y3*)) − *fst*(*ext-add* (*x1*,*y1*) *z3′*)
define g_y **where** g_y = *snd*(*ext-add z1′* (*x3*,*y3*)) − *snd*(*ext-add* (*x1*,*y1*) *z3′*)

have *x1′-expr*: *x1′* = (*x1* ∗ *x2* − *c* ∗ *y1* ∗ *y2*) / (*1* − *d* ∗ *x1* ∗ *y1* ∗ *x2* ∗ *y2*) **using** *assms*(*1*,*3*) **by** *simp*
have *y1′-expr*: *y1′* = (*x1* ∗ *y2* + *y1* ∗ *x2*) / (*1* + *d* ∗ *x1* ∗ *y1* ∗ *x2* ∗ *y2*) **using** *assms*(*1*,*3*) **by** *simp*
have *x3′-expr*: *x3′* = (*x2* ∗ *y2* − *x3* ∗ *y3*) / (*x3* ∗ *y2* − *x2* ∗ *y3*) **using** *assms*(*2*,*4*) **by** *simp*
have *y3′-expr*: *y3′* = (*x2* ∗ *y2* + *x3* ∗ *y3*) / (*x2* ∗ *x3* + *y2* ∗ *y3*) **using** *assms*(*2*,*4*) **by** *simp*

have *non-unfolded-adds*:
delta x1 y1 x2 y2 ≠ *0* **using** *delta-def assms*(*5*,*6*) **by** *auto*

have *simp1gx*:
(*x1′* ∗ *y1′* − *x3* ∗ *y3*) ∗ *delta-x x1 y1 x3′ y3′* ∗ (*delta x1 y1 x2 y2* ∗ *delta′ x2*

y2 x3 y3) =
((x1 * x2 − c * y1 * y2) * (x1 * y2 + y1 * x2) −
x3 * y3 * (delta-minus x1 y1 x2 y2 * delta-plus x1 y1 x2 y2)) *
((x2 * y2 − x3 * y3) * y1 * delta-y x2 y2 x3 y3 −
x1 * (x2 * y2 + x3 * y3) * delta-x x2 y2 x3 y3)

apply((subst x1'-expr)+, (subst y1'-expr)+,(subst x3'-expr)+,(subst y3'-expr)+)
apply(subst delta-plus-def[symmetric])
apply(subst delta-minus-def[symmetric])
apply(subst (4) delta-x-def[symmetric])
apply(subst (3) delta-y-def[symmetric])
apply(subst (2) delta-x-def)
unfolding delta'-def delta-def
by(simp add: divide-simps assms(5−8))

have simp2gx:
(x1 * y1 − x3' * y3') * delta-x x1' y1' x3 y3 * (delta x1 y1 x2 y2 * delta' x2 y2 x3 y3) =
(x1 * y1 * (delta-x x2 y2 x3 y3 * delta-y x2 y2 x3 y3) −
(x2 * y2 − x3 * y3) * (x2 * y2 + x3 * y3)) *
(x3 * (x1 * y2 + y1 * x2) * delta-minus x1 y1 x2 y2 −
(x1 * x2 − c * y1 * y2) * y3 * delta-plus x1 y1 x2 y2)
apply((subst x1'-expr)+, (subst y1'-expr)+,(subst x3'-expr)+,(subst y3'-expr)+)
apply(subst delta-plus-def[symmetric])
apply(subst delta-minus-def[symmetric])
apply(subst (3) delta-x-def[symmetric])
apply(subst (2) delta-y-def[symmetric])
apply(subst (2) delta-x-def)
unfolding delta'-def delta-def
by(simp add: divide-simps assms(5−8))

have ∃ r1 r2 r3. g_x * $Delta_x$ = r1 * e1 + r2 * e2 + r3 * e3
unfolding g_x-def $Delta_x$-def
apply(simp add: assms(1,2))
apply(subst (2 4) delta-x-def[symmetric])
apply(simp add: divide-simps assms(9,11))
apply(subst (3) left-diff-distrib)
apply(simp add: simp1gx simp2gx)
unfolding delta-x-def delta-y-def delta'-def delta-plus-def delta-minus-def delta-def e1-def e2-def e3-def e'-def
by(simp add: t-expr c-eq-1,algebra)
then have g_x * $Delta_x$ = 0 $Delta_x$ ≠ 0
using e1-def assms(13−15) e2-def e3-def **apply** auto
using $Delta_x$-def delta'-def assms(7−11) non-unfolded-adds **by** auto
then have g_x = 0 **by** auto

have simp1gy: delta-y x1' x3 y1' y3 * delta-y x1 y1 x3' y3' * (delta x1 y1 x2 y2 * delta' x2 y2 x3 y3) =
((x1 * x2 − c * y1 * y2) * (x1 * y2 + y1 * x2) +

```
    x3 * y3 * (delta-minus x1 y1 x2 y2 * delta-plus x1 y1 x2 y2)) *
   (x1 * (x2 * y2 - x3 * y3) * delta-y x2 y2 x3 y3 +
    y1 * (x2 * y2 + x3 * y3) * delta-x x2 y2 x3 y3)
  apply((subst x1'-expr)+, (subst y1'-expr)+,(subst x3'-expr)+,(subst y3'-expr)+)
   apply(subst delta-plus-def[symmetric])
   apply(subst delta-minus-def[symmetric])
   apply(subst (3) delta-x-def[symmetric])
   apply(subst (3) delta-y-def[symmetric])
   apply(subst (1 2) delta-y-def)
   unfolding delta'-def delta-def
   by(simp add: divide-simps assms(5-8))

  have simp2gy: delta-y x1 x3' y1 y3' * delta-y x1' y1' x3 y3 * (delta x1 y1 x2 y2
* delta' x2 y2 x3 y3) =
   (x1 * y1 * (delta-x x2 y2 x3 y3 * delta-y x2 y2 x3 y3) +
    (x2 * y2 - x3 * y3) * (x2 * y2 + x3 * y3)) *
   ((x1 * x2 - c * y1 * y2) * x3 * delta-plus x1 y1 x2 y2 +
    (x1 * y2 + y1 * x2) * y3 * delta-minus x1 y1 x2 y2)
  apply((subst x1'-expr)+, (subst y1'-expr)+,(subst x3'-expr)+,(subst y3'-expr)+)
   apply(subst delta-minus-def[symmetric])
   apply(subst delta-plus-def[symmetric])
   apply(subst (2) delta-x-def[symmetric])
   apply(subst (2) delta-y-def[symmetric])
   apply(subst (1 3) delta-y-def)
   unfolding delta'-def delta-def
   by(simp add: divide-simps assms(5-8))

  have ∃ r1 r2 r3. g_y * Delta_y = r1 * e1 + r2 * e2 + r3 * e3
   unfolding g_y-def Delta_y-def
   apply(simp add: assms(1,2))
   apply(subst delta-y-def[symmetric])+
   apply(simp add: divide-simps assms(10,12))
   apply(subst left-diff-distrib)
   apply(simp add: simp1gy simp2gy)
   unfolding delta-x-def delta-y-def delta-minus-def delta-plus-def
             e1-def e2-def e3-def e'-def
   by(simp add: c-eq-1 t-expr,algebra)

  then have g_y * Delta_y = 0 Delta_y ≠ 0
   using e1-def assms(13-15) e2-def e3-def apply auto
   using Delta_y-def delta'-def assms(7-12) non-unfolded-adds by auto
  then have g_y = 0 by auto

  show ?thesis
    using ‹g_y = 0› ‹g_x = 0› unfolding g_x-def g_y-def assms(3,4) by (simp add:
prod-eq-iff)
qed
```

lemma *add-ext-add-ext-assoc*:
assumes $z1' = (x1',y1')$ $z3' = (x3',y3')$
assumes $z1' =$ *ext-add* $(x1,y1)$ $(x2,y2)$ $z3' =$ *ext-add* $(x2,y2)$ $(x3,y3)$
assumes *delta-x* $x1\ y1\ x2\ y2 \neq 0$ *delta-y* $x1\ y1\ x2\ y2 \neq 0$
delta-x $x2\ y2\ x3\ y3 \neq 0$ *delta-y* $x2\ y2\ x3\ y3 \neq 0$
delta-plus $x1'\ y1'\ x3\ y3 \neq 0$ *delta-minus* $x1'\ y1'\ x3\ y3 \neq 0$
delta-plus $x1\ y1\ x3'\ y3' \neq 0$ *delta-minus* $x1\ y1\ x3'\ y3' \neq 0$
assumes $e'\ x1\ y1 = 0$ $e'\ x2\ y2 = 0$ $e'\ x3\ y3 = 0$
shows *add* (*ext-add* $(x1,y1)$ $(x2,y2)$) $(x3,y3)$ = *add* $(x1,y1)$ (*ext-add* $(x2,y2)$ $(x3,y3)$)
proof −
define $e1$ **where** $e1 = e'\ x1\ y1$
define $e2$ **where** $e2 = e'\ x2\ y2$
define $e3$ **where** $e3 = e'\ x3\ y3$
define $Delta_x$ **where** $Delta_x =$
(*delta-minus* $x1'\ y1'\ x3\ y3$)∗(*delta-minus* $x1\ y1\ x3'\ y3'$)∗
(*delta'* $x1\ y1\ x2\ y2$)∗(*delta'* $x2\ y2\ x3\ y3$)
define $Delta_y$ **where** $Delta_y =$
(*delta-plus* $x1'\ y1'\ x3\ y3$)∗(*delta-plus* $x1\ y1\ x3'\ y3'$)∗
(*delta'* $x1\ y1\ x2\ y2$)∗(*delta'* $x2\ y2\ x3\ y3$)
define g_x **where** $g_x =$ *fst*(*add* $z1'$ $(x3,y3)$) − *fst*(*add* $(x1,y1)$ $z3'$)
define g_y **where** $g_y =$ *snd*(*add* $z1'$ $(x3,y3)$) − *snd*(*add* $(x1,y1)$ $z3'$)

have *x1'-expr*: $x1' = (x1 * y1 - x2 * y2) / (x2 * y1 - x1 * y2)$ **using** *assms*(1,3) **by** *simp*
have *y1'-expr*: $y1' = (x1 * y1 + x2 * y2) / (x1 * x2 + y1 * y2)$ **using** *assms*(1,3) **by** *simp*
have *x3'-expr*: $x3' = (x2 * y2 - x3 * y3) / (x3 * y2 - x2 * y3)$ **using** *assms*(2,4) **by** *simp*
have *y3'-expr*: $y3' = (x2 * y2 + x3 * y3) / (x2 * x3 + y2 * y3)$ **using** *assms*(2,4) **by** *simp*

have *non-unfolded-adds*:
delta' $x1\ y1\ x2\ y2 \neq 0$ **using** *delta'-def* *assms*(5,6) **by** *auto*

have *simp1gx*:
$(x1' * x3 - c * y1' * y3) *$ *delta-minus* $x1\ y1\ x3'\ y3' *$ (*delta'* $x1\ y1\ x2\ y2 *$ *delta'* $x2\ y2\ x3\ y3$) =
$((x1 * y1 - x2 * y2) * x3 *$ *delta-y* $x1\ y1\ x2\ y2 - (x1 * y1 + x2 * y2) * y3 *$ *delta-x* $x1\ y1\ x2\ y2) *$
(*delta-x* $x2\ y2\ x3\ y3 *$ *delta-y* $x2\ y2\ x3\ y3 - d * x1 * y1 * (x2 * y2 - x3 * y3) * (x2 * y2 + x3 * y3)$)

apply((*subst* *x1'-expr*)+, (*subst* *y1'-expr*)+,(*subst* *x3'-expr*)+,(*subst* *y3'-expr*)+)
apply(*subst* (2 5) *delta-x-def*[*symmetric*])
apply(*subst* (2 4) *delta-y-def*[*symmetric*])
apply(*subst* *delta-minus-def*)
unfolding *delta'-def* *delta-def*
by(*simp add*: *divide-simps* *assms*(5−8) *c-eq-1*)

have *simp2gx*:
$(x1 * x3' - c * y1 * y3') *$ *delta-minus* $x1'\ y1'\ x3\ y3 * ($*delta'* $x1\ y1\ x2\ y2 *$ *delta'* $x2\ y2\ x3\ y3) =$
$(x1 * (x2 * y2 - x3 * y3) *$ *delta-y* $x2\ y2\ x3\ y3 - c * y1 * (x2 * y2 + x3 * y3) *$ *delta-x* $x2\ y2\ x3\ y3) *$
$($*delta-x* $x1\ y1\ x2\ y2 *$ *delta-y* $x1\ y1\ x2\ y2 - d * (x1 * y1 - x2 * y2) * (x1 * y1 + x2 * y2) * x3 * y3)$
apply((*subst x1'-expr*)+, (*subst y1'-expr*)+,(*subst x3'-expr*)+,(*subst y3'-expr*)+)
apply(*subst* (*2 5*) *delta-x-def*[*symmetric*])
apply(*subst* (*2 4*) *delta-y-def*[*symmetric*])
apply(*subst delta-minus-def*)
unfolding *delta'-def delta-def*
by(*simp add*: *divide-simps assms*(*5−8*))

have $\exists\ r1\ r2\ r3.\ g_x * Delta_x = r1 * e1 + r2 * e2 + r3 * e3$
unfolding g_x*-def* $Delta_x$*-def*
apply(*simp add*: *assms*(*1*,*2*))
apply(*subst* (*1 2*) *delta-minus-def*[*symmetric*])
apply(*simp add*: *divide-simps assms*(*10*,*12*))
apply(*subst* (*3*) *left-diff-distrib*)
apply(*simp add*: *simp1gx simp2gx*)
unfolding *delta-x-def delta-y-def delta'-def delta-plus-def delta-minus-def delta-def e1-def e2-def e3-def e'-def*
by(*simp add*: *t-expr c-eq-1*,*algebra*)
then have $g_x * Delta_x = 0$ $Delta_x \neq 0$
apply(*safe*)
using *e1-def e2-def e3-def assms*(*13−15*) **apply** *auto*
using $Delta_x$*-def delta'-def assms non-unfolded-adds* **by** *auto*
then have $g_x = 0$ **by** *auto*

have *simp1gy*: $(x1' * y3 + y1' * x3) *$ *delta-plus* $x1\ y1\ x3'\ y3' * ($*delta'* $x1\ y1\ x2\ y2 *$ *delta'* $x2\ y2\ x3\ y3) =$
$((x1 * y1 - x2 * y2) * y3 *$ *delta-y* $x1\ y1\ x2\ y2 + (x1 * y1 + x2 * y2) * x3 *$ *delta-x* $x1\ y1\ x2\ y2) *$
$($*delta-x* $x2\ y2\ x3\ y3 *$ *delta-y* $x2\ y2\ x3\ y3 + d * x1 * y1 * (x2 * y2 - x3 * y3) * (x2 * y2 + x3 * y3))$
apply((*subst x1'-expr*)+, (*subst y1'-expr*)+,(*subst x3'-expr*)+,(*subst y3'-expr*)+)
apply(*subst* (*2 4*) *delta-x-def*[*symmetric*])
apply(*subst* (*3 5*) *delta-y-def*[*symmetric*])
apply(*subst delta-plus-def*)
unfolding *delta'-def delta-def*
by(*simp add*: *divide-simps assms*(*5−8*))

have *simp2gy*: $(x1 * y3' + y1 * x3') *$ *delta-plus* $x1'\ y1'\ x3\ y3 * ($*delta'* $x1\ y1\ x2\ y2 *$ *delta'* $x2\ y2\ x3\ y3) =$
$(x1 * (x2 * y2 + x3 * y3) *$ *delta-x* $x2\ y2\ x3\ y3 + y1 * (x2 * y2 - x3 * y3) *$ *delta-y* $x2\ y2\ x3\ y3) *$
$($*delta-x* $x1\ y1\ x2\ y2 *$ *delta-y* $x1\ y1\ x2\ y2 + d * (x1 * y1 - x2 * y2) * (x1 *$

$y1 + x2 * y2) * x3 * y3$)
  apply((*subst x1'-expr*)+, (*subst y1'-expr*)+,(*subst x3'-expr*)+,(*subst y3'-expr*)+)
   apply(*subst* (*2 4*) *delta-x-def*[*symmetric*])
   apply(*subst* (*2 5*) *delta-y-def*[*symmetric*])
   apply(*subst delta-plus-def*)
   unfolding *delta'-def delta-def*
   by(*simp add*: *divide-simps assms*(*5*−*8*))
  have ∃ *r1 r2 r3*. g_y ∗ $Delta_y$ = *r1* ∗ *e1* + *r2* ∗ *e2* + *r3* ∗ *e3*
   unfolding g_y*-def* $Delta_y$*-def*
   apply(*simp add*: *assms*(*1*,*2*))
   apply(*subst delta-plus-def*[*symmetric*])+
   apply(*simp add*: *divide-simps assms*)
   apply(*subst left-diff-distrib*)
   apply(*simp add*: *simp1gy simp2gy*)
   unfolding *delta-x-def delta-y-def delta-minus-def delta-plus-def*
     e1-def e2-def e3-def e'-def
   by(*simp add*: *c-eq-1 t-expr*,*algebra*)

  then have g_y ∗ $Delta_y$ = *0* $Delta_y$ ≠ *0*
   using *e1-def assms*(*13*−*15*) *e2-def e3-def* **apply** *auto*
   using $Delta_y$*-def delta'-def assms*(*7*−*12*) *non-unfolded-adds* **by** *auto*
  then have g_y = *0* **by** *auto*

  show *?thesis*
   using ‹g_y = *0*› ‹g_x = *0*› **unfolding** g_x*-def* g_y*-def assms*(*3*,*4*) **by** (*simp add*: *prod-eq-iff*)
qed

lemma *add-ext-add-ext-assoc-points*:
  assumes (*x1*,*y1*) ∈ *e'-aff* (*x2*,*y2*) ∈ *e'-aff* (*x3*,*y3*) ∈ *e'-aff*
  assumes *delta' x1 y1 x2 y2* ≠ *0 delta' x2 y2 x3 y3* ≠ *0*
    delta (*fst* (*ext-add* (*x1*,*y1*) (*x2*,*y2*))) (*snd* (*ext-add* (*x1*,*y1*) (*x2*,*y2*))) *x3 y3* ≠ *0*
    delta x1 y1 (*fst* (*ext-add* (*x2*,*y2*) (*x3*,*y3*))) (*snd* (*ext-add* (*x2*,*y2*) (*x3*,*y3*))) ≠ *0*
    shows *add* (*ext-add* (*x1*,*y1*) (*x2*,*y2*)) (*x3*,*y3*) = *add* (*x1*,*y1*) (*ext-add* (*x2*,*y2*) (*x3*,*y3*))
  using *assms*
  unfolding *e'-aff-def delta-def delta'-def*
  apply(*simp del*: *add.simps*)
  using *add-ext-add-ext-assoc*
  apply(*safe*)
  using *ext-add.simps* **by** *metis*

lemma *add-ext-ext-ext-assoc*:
  assumes *z1'* = (*x1'*,*y1'*) *z3'* = (*x3'*,*y3'*)
  assumes *z1'* = *ext-add* (*x1*,*y1*) (*x2*,*y2*) *z3'* = *ext-add* (*x2*,*y2*) (*x3*,*y3*)
  assumes *delta-x x1 y1 x2 y2* ≠ *0 delta-y x1 y1 x2 y2* ≠ *0*
    delta-x x2 y2 x3 y3 ≠ *0 delta-y x2 y2 x3 y3* ≠ *0*

delta-plus x1' y1' x3 y3 $\neq$ *0 delta-minus x1' y1' x3 y3* $\neq$ *0*
delta-x x1 y1 x3' y3' $\neq$ *0 delta-y x1 y1 x3' y3'* $\neq$ *0*
assumes *e' x1 y1* = *0 e' x2 y2* = *0 e' x3 y3* = *0*
shows *add* (*ext-add* (*x1*,*y1*) (*x2*,*y2*)) (*x3*,*y3*) = *ext-add* (*x1*,*y1*) (*ext-add* (*x2*,*y2*) (*x3*,*y3*))
proof −
define *e1* **where** *e1* = *e' x1 y1*
define *e2* **where** *e2* = *e' x2 y2*
define *e3* **where** *e3* = *e' x3 y3*
define $Delta_x$ **where** $Delta_x$ =
(*delta-minus x1' y1' x3 y3*)∗(*delta-x x1 y1 x3' y3'*)∗
(*delta' x1 y1 x2 y2*)∗(*delta' x2 y2 x3 y3*)
define $Delta_y$ **where** $Delta_y$ =
(*delta-plus x1' y1' x3 y3*)∗(*delta-y x1 y1 x3' y3'*)∗
(*delta' x1 y1 x2 y2*)∗(*delta' x2 y2 x3 y3*)
define g_x **where** g_x = *fst*(*add z1'* (*x3*,*y3*)) − *fst*(*ext-add* (*x1*,*y1*) *z3'*)
define g_y **where** g_y = *snd*(*add z1'* (*x3*,*y3*)) − *snd*(*ext-add* (*x1*,*y1*) *z3'*)

have *x1'-expr*: *x1'* = (*x1* ∗ *y1* − *x2* ∗ *y2*) / (*x2* ∗ *y1* − *x1* ∗ *y2*) **using** *assms*(*1*,*3*) **by** *simp*
have *y1'-expr*: *y1'* = (*x1* ∗ *y1* + *x2* ∗ *y2*) / (*x1* ∗ *x2* + *y1* ∗ *y2*) **using** *assms*(*1*,*3*) **by** *simp*
have *x3'-expr*: *x3'* = (*x2* ∗ *y2* − *x3* ∗ *y3*) / (*x3* ∗ *y2* − *x2* ∗ *y3*) **using** *assms*(*2*,*4*) **by** *simp*
have *y3'-expr*: *y3'* = (*x2* ∗ *y2* + *x3* ∗ *y3*) / (*x2* ∗ *x3* + *y2* ∗ *y3*) **using** *assms*(*2*,*4*) **by** *simp*

have *non-unfolded-adds*:
delta' x1 y1 x2 y2 $\neq$ *0* **using** *delta'-def assms*(*5*,*6*) **by** *auto*

have *simp1gx*:
(*x1'* ∗ *x3* − *c* ∗ *y1'* ∗ *y3*) ∗ *delta-x x1 y1 x3' y3'* ∗ (*delta' x1 y1 x2 y2* ∗ *delta' x2 y2 x3 y3*) =
((*x1* ∗ *y1* − *x2* ∗ *y2*) ∗ *x3* ∗ *delta-y x1 y1 x2 y2* − (*x1* ∗ *y1* + *x2* ∗ *y2*) ∗ *y3* ∗ *delta-x x1 y1 x2 y2*) ∗
((*x2* ∗ *y2* − *x3* ∗ *y3*) ∗ *y1* ∗ *delta-y x2 y2 x3 y3* − *x1* ∗ (*x2* ∗ *y2* + *x3* ∗ *y3*) ∗ *delta-x x2 y2 x3 y3*)

apply((*subst x1'-expr*)+, (*subst y1'-expr*)+,(*subst x3'-expr*)+,(*subst y3'-expr*)+)
apply(*subst delta-x-def*)
apply(*subst* (*2 5*) *delta-x-def*[*symmetric*])
apply(*subst* (*2 4*) *delta-y-def*[*symmetric*])
unfolding *delta'-def delta-def*
by(*simp add*: *divide-simps assms*(*5*−*8*) *c-eq-1*)

have *simp2gx*:
(*x1* ∗ *y1* − *x3'* ∗ *y3'*) ∗ *delta-minus x1' y1' x3 y3* ∗ (*delta' x1 y1 x2 y2* ∗ *delta' x2 y2 x3 y3*) =
(*x1* ∗ *y1* ∗ (*delta-x x2 y2 x3 y3* ∗ *delta-y x2 y2 x3 y3*) −

$(x2 * y2 - x3 * y3) * (x2 * y2 + x3 * y3)) *$
$(delta\text{-}x\ x1\ y1\ x2\ y2 * delta\text{-}y\ x1\ y1\ x2\ y2 -$
$d * (x1 * y1 - x2 * y2) * (x1 * y1 + x2 * y2) * x3 * y3)$
apply((*subst x1'-expr*)+, (*subst y1'-expr*)+,(*subst x3'-expr*)+,(*subst y3'-expr*)+)
apply(*subst delta-minus-def*)
apply(*subst* (*3 5*) *delta-x-def*[*symmetric*])
apply(*subst* (*2 4*) *delta-y-def*[*symmetric*])
unfolding *delta'-def delta-def*
by(*simp add*: *divide-simps assms*(*5−8*))

have $\exists$ *r1 r2 r3*. $g_x * Delta_x = r1 * e1 + r2 * e2 + r3 * e3$
unfolding g_x*-def* $Delta_x$*-def*
apply(*simp add*: *assms*(*1*,*2*))
apply(*subst* (*1*) *delta-minus-def*[*symmetric*])
apply(*subst* (*3*) *delta-x-def*[*symmetric*])
apply(*simp add*: *divide-simps assms*)
apply(*subst* (*3*) *left-diff-distrib*)
apply(*simp add*: *simp1gx simp2gx*)
unfolding *delta-x-def delta-y-def delta'-def delta-plus-def delta-minus-def delta-def e1-def e2-def e3-def e'-def*
by(*simp add*: *t-expr c-eq-1*,*algebra*)
then have $g_x * Delta_x = 0$ $Delta_x \neq 0$
apply(*safe*)
using *e1-def e2-def e3-def assms*(*13−15*) **apply** *auto*
using $Delta_x$*-def delta'-def assms non-unfolded-adds* **by** *auto*
then have $g_x = 0$ **by** *auto*

have *simp1gy*:
$(x1' * y3 + y1' * x3) * delta\text{-}y\ x1\ y1\ x3'\ y3' * (delta'\ x1\ y1\ x2\ y2 * delta'\ x2\ y2\ x3\ y3) =$
$((x1 * y1 - x2 * y2) * y3 * delta\text{-}y\ x1\ y1\ x2\ y2 + (x1 * y1 + x2 * y2) * x3 * delta\text{-}x\ x1\ y1\ x2\ y2) *$
$(x1 * (x2 * y2 - x3 * y3) * delta\text{-}y\ x2\ y2\ x3\ y3 + y1 * (x2 * y2 + x3 * y3) * delta\text{-}x\ x2\ y2\ x3\ y3)$
apply((*subst x1'-expr*)+, (*subst y1'-expr*)+,(*subst x3'-expr*)+,(*subst y3'-expr*)+)
apply(*subst delta-y-def*)
apply(*subst* (*2 4*) *delta-x-def*[*symmetric*])
apply(*subst* (*3 6*) *delta-y-def*[*symmetric*])
unfolding *delta'-def delta-def*
by(*simp add*: *divide-simps assms*(*5−8*))

have *simp2gy*:
$(x1 * y1 + x3' * y3') * delta\text{-}plus\ x1'\ y1'\ x3\ y3 * (delta'\ x1\ y1\ x2\ y2 * delta'\ x2\ y2\ x3\ y3) =$
$(x1 * y1 * (delta\text{-}x\ x2\ y2\ x3\ y3 * delta\text{-}y\ x2\ y2\ x3\ y3) +$
$(x2 * y2 - x3 * y3) * (x2 * y2 + x3 * y3)) *$
$(delta\text{-}x\ x1\ y1\ x2\ y2 * delta\text{-}y\ x1\ y1\ x2\ y2 +$
$d * (x1 * y1 - x2 * y2) * (x1 * y1 + x2 * y2) * x3 * y3)$
apply((*subst x1'-expr*)+, (*subst y1'-expr*)+,(*subst x3'-expr*)+,(*subst y3'-expr*)+)

apply(*subst delta-plus-def*)
apply(*subst* (*2 4*) *delta-x-def*[*symmetric*])
apply(*subst* (*3 5*) *delta-y-def*[*symmetric*])
unfolding *delta′-def delta-def*
by(*simp add*: *divide-simps assms*(*5*−*8*))
have ∃ *r1 r2 r3*. g_y ∗ $Delta_y$ = *r1* ∗ *e1* + *r2* ∗ *e2* + *r3* ∗ *e3*
unfolding g_y*-def* $Delta_y$*-def*
apply(*simp add*: *assms*(*1*,*2*))
apply(*subst delta-plus-def*[*symmetric*])
apply(*subst* (*3*) *delta-y-def*[*symmetric*])
apply(*simp add*: *divide-simps assms*)
apply(*subst left-diff-distrib*)
apply(*simp add*: *simp1gy simp2gy*)
unfolding *delta-x-def delta-y-def delta-minus-def delta-plus-def*
e1-def e2-def e3-def e′-def
by(*simp add*: *c-eq-1 t-expr*,*algebra*)

then have g_y ∗ $Delta_y$ = *0* $Delta_y$ ≠ *0*
using *e1-def assms*(*13*−*15*) *e2-def e3-def* **apply** *auto*
using $Delta_y$*-def delta′-def assms*(*7*−*12*) *non-unfolded-adds* **by** *auto*
then have g_y = *0* **by** *auto*

show *?thesis*
using ‹g_y = *0*› ‹g_x = *0*› **unfolding** g_x*-def* g_y*-def assms*(*3*,*4*) **by** (*simp add*: *prod-eq-iff*)
qed

lemma *add-ext-add-add-assoc*:
assumes *z1′* = (*x1′*,*y1′*) *z3′* = (*x3′*,*y3′*)
assumes *z1′* = *ext-add* (*x1*,*y1*) (*x2*,*y2*) *z3′* = *add* (*x2*,*y2*) (*x3*,*y3*)
assumes *delta-x x1 y1 x2 y2* ≠ *0 delta-y x1 y1 x2 y2* ≠ *0*
delta-plus x2 y2 x3 y3 ≠ *0 delta-minus x2 y2 x3 y3* ≠ *0*
delta-plus x1′ y1′ x3 y3 ≠ *0 delta-minus x1′ y1′ x3 y3* ≠ *0*
delta-plus x1 y1 x3′ y3′ ≠ *0 delta-minus x1 y1 x3′ y3′* ≠ *0*
assumes *e′ x1 y1* = *0 e′ x2 y2* = *0 e′ x3 y3* = *0*
shows *add* (*ext-add* (*x1*,*y1*) (*x2*,*y2*)) (*x3*,*y3*) = *add* (*x1*,*y1*) (*add* (*x2*,*y2*) (*x3*,*y3*))
proof −
define *e1* **where** *e1* = *e′ x1 y1*
define *e2* **where** *e2* = *e′ x2 y2*
define *e3* **where** *e3* = *e′ x3 y3*
define $Delta_x$ **where** $Delta_x$ =
(*delta-minus x1′ y1′ x3 y3*)∗(*delta-minus x1 y1 x3′ y3′*)∗
(*delta′ x1 y1 x2 y2*)∗(*delta x2 y2 x3 y3*)
define $Delta_y$ **where** $Delta_y$ =
(*delta-plus x1′ y1′ x3 y3*)∗(*delta-plus x1 y1 x3′ y3′*)∗
(*delta′ x1 y1 x2 y2*)∗(*delta x2 y2 x3 y3*)
define g_x **where** g_x = *fst*(*add z1′* (*x3*,*y3*)) − *fst*(*add* (*x1*,*y1*) *z3′*)
define g_y **where** g_y = *snd*(*add z1′* (*x3*,*y3*)) − *snd*(*add* (*x1*,*y1*) *z3′*)

have *x1'-expr*: $x1' = (x1 * y1 - x2 * y2) / (x2 * y1 - x1 * y2)$ **using** *assms*(*1*,*3*) **by** *simp*
have *y1'-expr*: $y1' = (x1 * y1 + x2 * y2) / (x1 * x2 + y1 * y2)$ **using** *assms*(*1*,*3*) **by** *simp*
have *x3'-expr*: $x3' = (x2 * x3 - c * y2 * y3) / (1 - d * x2 * y2 * x3 * y3)$ **using** *assms*(*2*,*4*) **by** *simp*
have *y3'-expr*: $y3' = (x2 * y3 + y2 * x3) / (1 + d * x2 * y2 * x3 * y3)$ **using** *assms*(*2*,*4*) **by** *simp*

have *non-unfolded-adds*:
delta' x1 y1 x2 y2 $\neq$ *0* **using** *delta'-def assms*(*5*,*6*) **by** *auto*

have *simp1gx*:
(*x1' * x3 − c * y1' * y3*) * *delta-minus x1 y1 x3' y3'* *
(*delta' x1 y1 x2 y2* * *delta x2 y2 x3 y3*) =
((*x1 * y1 − x2 * y2*) * *x3* * *delta-y x1 y1 x2 y2* −
(*x1 * y1 + x2 * y2*) * *y3* * *delta-x x1 y1 x2 y2*) *
(*delta-minus x2 y2 x3 y3* * *delta-plus x2 y2 x3 y3* −
*d * x1 * y1* * (*x2 * x3 − y2 * y3*) * (*x2 * y3 + y2 * x3*))

apply((*subst x1'-expr*)+, (*subst y1'-expr*)+,(*subst x3'-expr*)+,(*subst y3'-expr*)+)
apply(*subst delta-minus-def*)
apply(*subst* (*2*) *delta-x-def*[*symmetric*])
apply(*subst* (*2*) *delta-y-def*[*symmetric*])
apply(*subst* (*2*) *delta-minus-def*[*symmetric*])
apply(*subst* (*1*) *delta-plus-def*[*symmetric*])
unfolding *delta'-def delta-def*
by(*simp add*: *divide-simps assms*(*5*−*8*) *c-eq-1*)

have *simp2gx*:
(*x1 * x3' − c * y1 * y3'*) * *delta-minus x1' y1' x3 y3* *
(*delta' x1 y1 x2 y2* * *delta x2 y2 x3 y3*) =
(*x1* * (*x2 * x3 − c * y2 * y3*) * *delta-plus x2 y2 x3 y3* −
*c * y1* * (*x2 * y3 + y2 * x3*) * *delta-minus x2 y2 x3 y3*) *
(*delta-x x1 y1 x2 y2* * *delta-y x1 y1 x2 y2* −
d * (*x1 * y1 − x2 * y2*) * (*x1 * y1 + x2 * y2*) * *x3 * y3*)
apply((*subst x1'-expr*)+, (*subst y1'-expr*)+,(*subst x3'-expr*)+,(*subst y3'-expr*)+)
apply(*subst delta-minus-def*)
apply(*subst* (*4*) *delta-x-def*[*symmetric*])
apply(*subst* (*3*) *delta-y-def*[*symmetric*])
apply(*subst* (*1*) *delta-minus-def*[*symmetric*])
apply(*subst* (*1*) *delta-plus-def*[*symmetric*])
unfolding *delta'-def delta-def*
by(*simp add*: *divide-simps assms*(*5*−*8*))

have $\exists$ *r1 r2 r3*. $g_x * Delta_x = r1 * e1 + r2 * e2 + r3 * e3$
unfolding g_x*-def* $Delta_x$*-def*
apply(*simp add*: *assms*(*1*,*2*))

apply(*subst* (*1 2*) *delta-minus-def* [*symmetric*])
apply(*simp add*: *divide-simps assms*(*10*,*12*))
apply(*subst* (*3*) *left-diff-distrib*)
apply(*simp add*: *simp1gx simp2gx*)
unfolding *delta-x-def delta-y-def delta′-def delta-plus-def delta-minus-def delta-def e1-def e2-def e3-def e′-def*
by(*simp add*: *t-expr c-eq-1*,*algebra*)
then have $g_x * Delta_x = 0$ $Delta_x \neq 0$
apply(*safe*)
using *e1-def e2-def e3-def assms*(*13*−*15*) **apply** *force*
using $Delta_x$*-def delta′-def delta-def assms non-unfolded-adds* **by** *force*
then have $g_x = 0$ **by** *auto*

have *simp1gy*: $(x1' * y3 + y1' * x3) * delta\text{-}plus\ x1\ y1\ x3'\ y3' *$
$(delta'\ x1\ y1\ x2\ y2 * delta\ x2\ y2\ x3\ y3) =$
$((x1 * y1 - x2 * y2) * y3 * delta\text{-}y\ x1\ y1\ x2\ y2 +$
$(x1 * y1 + x2 * y2) * x3 * delta\text{-}x\ x1\ y1\ x2\ y2) *$
$(delta\text{-}minus\ x2\ y2\ x3\ y3 * delta\text{-}plus\ x2\ y2\ x3\ y3 +$
$d * x1 * y1 * (x2 * x3 - c * y2 * y3) * (x2 * y3 + y2 * x3))$
apply((*subst x1′-expr*)+, (*subst y1′-expr*)+,(*subst x3′-expr*)+,(*subst y3′-expr*)+)
apply(*subst delta-plus-def*)
apply(*subst* (*2*) *delta-x-def* [*symmetric*])
apply(*subst* (*3*) *delta-y-def* [*symmetric*])
apply(*subst* (*2*) *delta-plus-def* [*symmetric*])
apply(*subst* (*1*) *delta-minus-def* [*symmetric*])
unfolding *delta′-def delta-def*
by(*simp add*: *divide-simps assms*(*5*−*8*))

have *simp2gy*: $(x1 * y3' + y1 * x3') * delta\text{-}plus\ x1'\ y1'\ x3\ y3 *$
$(delta'\ x1\ y1\ x2\ y2 * delta\ x2\ y2\ x3\ y3) =$
$(x1 * (x2 * y3 + y2 * x3) * delta\text{-}minus\ x2\ y2\ x3\ y3 +$
$y1 * (x2 * x3 - c * y2 * y3) * delta\text{-}plus\ x2\ y2\ x3\ y3) *$
$(delta\text{-}x\ x1\ y1\ x2\ y2 * delta\text{-}y\ x1\ y1\ x2\ y2 +$
$d * (x1 * y1 - x2 * y2) * (x1 * y1 + x2 * y2) * x3 * y3)$
apply((*subst x1′-expr*)+, (*subst y1′-expr*)+,(*subst x3′-expr*)+,(*subst y3′-expr*)+)
apply(*subst delta-plus-def*)
apply(*subst* (*3*) *delta-x-def* [*symmetric*])
apply(*subst* (*4*) *delta-y-def* [*symmetric*])
apply(*subst* (*1*) *delta-plus-def* [*symmetric*])
apply(*subst* (*1*) *delta-minus-def* [*symmetric*])
unfolding *delta′-def delta-def*
by(*simp add*: *divide-simps assms*(*5*−*8*))
have $\exists\ r1\ r2\ r3.\ g_y * Delta_y = r1 * e1 + r2 * e2 + r3 * e3$
unfolding g_y*-def* $Delta_y$*-def*
apply(*simp add*: *assms*(*1*,*2*))
apply(*subst delta-plus-def* [*symmetric*])+
apply(*simp add*: *divide-simps assms*)
apply(*subst left-diff-distrib*)
apply(*simp add*: *simp1gy simp2gy*)

unfolding *delta-x-def delta-y-def delta-minus-def delta-plus-def*
e1-def e2-def e3-def e′-def
by(*simp add*: *c-eq-1 t-expr*,*algebra*)

then have $g_y * Delta_y = 0$ $Delta_y \neq 0$
using *e1-def assms*(*13*−*15*) *e2-def e3-def* **apply** *force*
using $Delta_y$*-def delta′-def delta-def assms*(*7*−*12*) *non-unfolded-adds* **by** *auto*
then have $g_y = 0$ **by** *auto*

show *?thesis*
using ‹$g_y = 0$› ‹$g_x = 0$› **unfolding** g_x*-def* g_y*-def assms*(*3*,*4*) **by** (*simp add*: *prod-eq-iff*)
qed

lemma *add-ext-add-add-assoc-points*:
assumes (*x1*,*y1*) ∈ *e′-aff* (*x2*,*y2*) ∈ *e′-aff* (*x3*,*y3*) ∈ *e′-aff*
assumes *delta′ x1 y1 x2 y2* ≠ *0 delta x2 y2 x3 y3* ≠ *0*
delta (*fst* (*ext-add* (*x1*,*y1*) (*x2*,*y2*))) (*snd* (*ext-add* (*x1*,*y1*) (*x2*,*y2*))) *x3 y3* ≠ *0*
delta x1 y1 (*fst* (*add* (*x2*,*y2*) (*x3*,*y3*))) (*snd* (*add* (*x2*,*y2*) (*x3*,*y3*))) ≠ *0*
shows *add* (*ext-add* (*x1*,*y1*) (*x2*,*y2*)) (*x3*,*y3*) = *add* (*x1*,*y1*) (*add* (*x2*,*y2*) (*x3*,*y3*))
using *assms*
unfolding *e′-aff-def delta-def delta′-def*
apply(*simp del*: *add.simps*)
using *add-ext-add-add-assoc*
apply(*safe*)
using *prod.collapse ext-add.simps* **by** *metis*

lemma *add-add-ext-add-assoc*:
assumes *z1′* = (*x1′*,*y1′*) *z3′* = (*x3′*,*y3′*)
assumes *z1′* = *add* (*x1*,*y1*) (*x2*,*y2*) *z3′* = *add* (*x2*,*y2*) (*x3*,*y3*)
assumes *delta-minus x1 y1 x2 y2* ≠ *0 delta-plus x1 y1 x2 y2* ≠ *0*
delta-minus x2 y2 x3 y3 ≠ *0 delta-plus x2 y2 x3 y3* ≠ *0*
delta-minus x1′ y1′ x3 y3 ≠ *0 delta-plus x1′ y1′ x3 y3* ≠ *0*
delta-x x1 y1 x3′ y3′ ≠ *0 delta-y x1 y1 x3′ y3′* ≠ *0*
assumes *e′ x1 y1* = *0 e′ x2 y2* = *0 e′ x3 y3* = *0*
shows *add* (*add* (*x1*,*y1*) (*x2*,*y2*)) (*x3*,*y3*) = *ext-add* (*x1*,*y1*) (*add* (*x2*,*y2*) (*x3*,*y3*))
proof −
define *e1* **where** *e1* = *e′ x1 y1*
define *e2* **where** *e2* = *e′ x2 y2*
define *e3* **where** *e3* = *e′ x3 y3*
define $Delta_x$ **where** $Delta_x$ =
(*delta-minus x1′ y1′ x3 y3*)∗(*delta-x x1 y1 x3′ y3′*)∗
(*delta x1 y1 x2 y2*)∗(*delta x2 y2 x3 y3*)
define $Delta_y$ **where** $Delta_y$ =
(*delta-plus x1′ y1′ x3 y3*)∗(*delta-y x1 y1 x3′ y3′*)∗

(*delta x1 y1 x2 y2*)∗(*delta x2 y2 x3 y3*)
define g_x **where** g_x = *fst*(*add z1′* (*x3*,*y3*)) − *fst*(*ext-add* (*x1*,*y1*) *z3′*)
define g_y **where** g_y = *snd*(*add z1′* (*x3*,*y3*)) − *snd*(*ext-add* (*x1*,*y1*) *z3′*)

have *x1′-expr*: *x1′* = (*x1* ∗ *x2* − *c* ∗ *y1* ∗ *y2*) / (*1* − *d* ∗ *x1* ∗ *y1* ∗ *x2* ∗ *y2*) **using** *assms*(*1*,*3*) **by** *simp*
have *y1′-expr*: *y1′* = (*x1* ∗ *y2* + *y1* ∗ *x2*) / (*1* + *d* ∗ *x1* ∗ *y1* ∗ *x2* ∗ *y2*) **using** *assms*(*1*,*3*) **by** *simp*
have *x3′-expr*: *x3′* = (*x2* ∗ *x3* − *c* ∗ *y2* ∗ *y3*) / (*1* − *d* ∗ *x2* ∗ *y2* ∗ *x3* ∗ *y3*) **using** *assms*(*2*,*4*) **by** *simp*
have *y3′-expr*: *y3′* = (*x2* ∗ *y3* + *y2* ∗ *x3*) / (*1* + *d* ∗ *x2* ∗ *y2* ∗ *x3* ∗ *y3*) **using** *assms*(*2*,*4*) **by** *simp*

have *non-unfolded-adds*:
delta x1 y1 x2 y2 ≠ *0* **using** *delta-def assms*(*5*,*6*) **by** *auto*

have *simp1gx*:
(*x1′* ∗ *x3* − *c* ∗ *y1′* ∗ *y3*) ∗ *delta-x x1 y1 x3′ y3′* ∗ (*delta x1 y1 x2 y2* ∗ *delta x2 y2 x3 y3*) =
((*x1* ∗ *x2* − *y1* ∗ *y2*) ∗ *x3* ∗ *delta-plus x1 y1 x2 y2* −
(*x1* ∗ *y2* + *y1* ∗ *x2*) ∗ *y3* ∗ *delta-minus x1 y1 x2 y2*) ∗
((*x2* ∗ *x3* − *y2* ∗ *y3*) ∗ *y1* ∗ *delta-plus x2 y2 x3 y3* −
x1 ∗ (*x2* ∗ *y3* + *y2* ∗ *x3*) ∗ *delta-minus x2 y2 x3 y3*)

apply((*subst x1′-expr*)+, (*subst y1′-expr*)+,(*subst x3′-expr*)+,(*subst y3′-expr*)+)
apply(*subst delta-x-def*)
apply(*subst* (*1 2*) *delta-minus-def*[*symmetric*])
apply(*subst* (*1 2*) *delta-plus-def*[*symmetric*])
unfolding *delta′-def delta-def*
by(*simp add*: *divide-simps assms*(*5*−*8*) *c-eq-1*)

have *simp2gx*:
(*x1* ∗ *y1* − *x3′* ∗ *y3′*) ∗ *delta-minus x1′ y1′ x3 y3* ∗ (*delta x1 y1 x2 y2* ∗ *delta x2 y2 x3 y3*) =
(*x1* ∗ *y1* ∗ (*delta-minus x2 y2 x3 y3* ∗ *delta-plus x2 y2 x3 y3*) −
(*x2* ∗ *x3* − *c* ∗ *y2* ∗ *y3*) ∗ (*x2* ∗ *y3* + *y2* ∗ *x3*)) ∗
(*delta-minus x1 y1 x2 y2* ∗ *delta-plus x1 y1 x2 y2* −
d ∗ (*x1* ∗ *x2* − *c* ∗ *y1* ∗ *y2*) ∗ (*x1* ∗ *y2* + *y1* ∗ *x2*) ∗ *x3* ∗ *y3*)
apply((*subst x1′-expr*)+, (*subst y1′-expr*)+,(*subst x3′-expr*)+,(*subst y3′-expr*)+)
apply(*subst delta-minus-def*)
apply(*subst* (*1 3*) *delta-minus-def*[*symmetric*])
apply(*subst* (*1 2*) *delta-plus-def*[*symmetric*])
unfolding *delta′-def delta-def*
by(*simp add*: *divide-simps assms*(*5*−*8*))

have ∃ *r1 r2 r3*. g_x ∗ $Delta_x$ = *r1* ∗ *e1* + *r2* ∗ *e2* + *r3* ∗ *e3*
unfolding g_x*-def* $Delta_x$*-def*
apply(*simp add*: *assms*(*1*,*2*))
apply(*subst delta-minus-def*[*symmetric*])

apply(*subst* (*3*) *delta-x-def*[*symmetric*])
apply(*simp add*: *divide-simps assms*(*9*,*11*))
apply(*subst* (*3*) *left-diff-distrib*)
apply(*simp add*: *simp1gx simp2gx*)
unfolding *delta-x-def delta-y-def delta′-def delta-plus-def delta-minus-def delta-def e1-def e2-def e3-def e′-def*
by(*simp add*: *t-expr c-eq-1*,*algebra*)
then have $g_x * Delta_x = 0$ $Delta_x \neq 0$
apply(*safe*)
using *e1-def e2-def e3-def assms*(*13*−*15*) **apply** *simp*
using $Delta_x$*-def delta-def delta′-def assms non-unfolded-adds* **by** *simp*
then have $g_x = 0$ **by** *auto*

have *simp1gy*: (*x1′* ∗ *y3* + *y1′* ∗ *x3*) ∗ *delta-y x1 y1 x3′ y3′* ∗ (*delta x1 y1 x2 y2* ∗ *delta x2 y2 x3 y3*) =
((*x1* ∗ *x2* − *c* ∗ *y1* ∗ *y2*) ∗ *y3* ∗ *delta-plus x1 y1 x2 y2* +
(*x1* ∗ *y2* + *y1* ∗ *x2*) ∗ *x3* ∗ *delta-minus x1 y1 x2 y2*) ∗
(*x1* ∗ (*x2* ∗ *x3* − *c* ∗ *y2* ∗ *y3*) ∗ *delta-plus x2 y2 x3 y3* +
y1 ∗ (*x2* ∗ *y3* + *y2* ∗ *x3*) ∗ *delta-minus x2 y2 x3 y3*)
apply((*subst x1′-expr*)+, (*subst y1′-expr*)+,(*subst x3′-expr*)+,(*subst y3′-expr*)+)
apply(*subst delta-y-def*)
apply(*subst* (*1 2*) *delta-plus-def*[*symmetric*])
apply(*subst* (*1 2*) *delta-minus-def*[*symmetric*])
unfolding *delta′-def delta-def*
by(*simp add*: *divide-simps assms*(*5*−*8*))

have *simp2gy*: (*x1* ∗ *y1* + *x3′* ∗ *y3′*) ∗ *delta-plus x1′ y1′ x3 y3* ∗ (*delta x1 y1 x2 y2* ∗ *delta x2 y2 x3 y3*) =
(*x1* ∗ *y1* ∗ (*delta-minus x2 y2 x3 y3* ∗ *delta-plus x2 y2 x3 y3*) +
(*x2* ∗ *x3* − *c* ∗ *y2* ∗ *y3*) ∗ (*x2* ∗ *y3* + *y2* ∗ *x3*)) ∗
(*delta-minus x1 y1 x2 y2* ∗ *delta-plus x1 y1 x2 y2* +
d ∗ (*x1* ∗ *x2* − *c* ∗ *y1* ∗ *y2*) ∗ (*x1* ∗ *y2* + *y1* ∗ *x2*) ∗ *x3* ∗ *y3*)
apply((*subst x1′-expr*)+, (*subst y1′-expr*)+,(*subst x3′-expr*)+,(*subst y3′-expr*)+)
apply(*subst delta-plus-def*)
apply(*subst* (*1 3*) *delta-plus-def*[*symmetric*])
apply(*subst* (*1 2*) *delta-minus-def*[*symmetric*])
unfolding *delta′-def delta-def*
by(*simp add*: *divide-simps assms*(*5*−*8*))

have ∃ *r1 r2 r3*. $g_y * Delta_y = r1 * e1 + r2 * e2 + r3 * e3$
unfolding g_y*-def* $Delta_y$*-def*
apply(*simp add*: *assms*(*1*,*2*))
apply(*subst delta-plus-def*[*symmetric*])
apply(*subst* (*3*) *delta-y-def*[*symmetric*])
apply(*simp add*: *divide-simps assms*)
apply(*subst left-diff-distrib*)
apply(*simp add*: *simp1gy simp2gy*)
unfolding *delta-x-def delta-y-def delta-minus-def delta-plus-def e1-def e2-def e3-def e′-def*

by(*simp add*: *c-eq-1 t-expr*,*algebra*)

then have $g_y * Delta_y = 0$ $Delta_y \neq 0$
using *e1-def assms*(*13−15*) *e2-def e3-def* **apply** *simp*
using *Delta*$_y$*-def delta-def delta'-def assms non-unfolded-adds* **by** *simp*
then have $g_y = 0$ **by** *auto*

show *?thesis*
using ‹$g_y = 0$› ‹$g_x = 0$› **unfolding** g_x*-def* g_y*-def assms*(*3*,*4*) **by** (*simp add*: *prod-eq-iff*)
qed

lemma *add-add-ext-add-assoc-points*:
assumes (*x1*,*y1*) ∈ *e'-aff* (*x2*,*y2*) ∈ *e'-aff* (*x3*,*y3*) ∈ *e'-aff*
assumes *delta x1 y1 x2 y2* ≠ *0 delta x2 y2 x3 y3* ≠ *0*
delta (*fst* (*add* (*x1*,*y1*) (*x2*,*y2*))) (*snd* (*add* (*x1*,*y1*) (*x2*,*y2*))) *x3 y3* ≠ *0*
delta' x1 y1 (*fst* (*add* (*x2*,*y2*) (*x3*,*y3*))) (*snd* (*add* (*x2*,*y2*) (*x3*,*y3*))) ≠ *0*
shows *add* (*add* (*x1*,*y1*) (*x2*,*y2*)) (*x3*,*y3*) = *ext-add* (*x1*,*y1*) (*add* (*x2*,*y2*) (*x3*,*y3*))
using *assms*
unfolding *e'-aff-def delta-def delta'-def*
apply(*simp del*: *add.simps*)
using *add-add-ext-add-assoc*
apply(*safe*)
using *prod.collapse* **by** *blast*

lemma *add-add-ext-ext-assoc*:
assumes *z1'* = (*x1'*,*y1'*) *z3'* = (*x3'*,*y3'*)
assumes *z1'* = *add* (*x1*,*y1*) (*x2*,*y2*) *z3'* = *ext-add* (*x2*,*y2*) (*x3*,*y3*)
assumes *delta-minus x1 y1 x2 y2* ≠ *0 delta-plus x1 y1 x2 y2* ≠ *0*
delta-x x2 y2 x3 y3 ≠ *0 delta-y x2 y2 x3 y3* ≠ *0*
delta-minus x1' y1' x3 y3 ≠ *0 delta-plus x1' y1' x3 y3* ≠ *0*
delta-x x1 y1 x3' y3' ≠ *0 delta-y x1 y1 x3' y3'* ≠ *0*
assumes *e' x1 y1* = *0 e' x2 y2* = *0 e' x3 y3* = *0*
shows *add* (*add* (*x1*,*y1*) (*x2*,*y2*)) (*x3*,*y3*) = *ext-add* (*x1*,*y1*) (*ext-add* (*x2*,*y2*) (*x3*,*y3*))
proof −
define *e1* **where** *e1* = *e' x1 y1*
define *e2* **where** *e2* = *e' x2 y2*
define *e3* **where** *e3* = *e' x3 y3*
define $Delta_x$ **where** $Delta_x$ =
(*delta-minus x1' y1' x3 y3*)∗(*delta-x x1 y1 x3' y3'*)∗
(*delta x1 y1 x2 y2*)∗(*delta' x2 y2 x3 y3*)
define $Delta_y$ **where** $Delta_y$ =
(*delta-plus x1' y1' x3 y3*)∗(*delta-y x1 y1 x3' y3'*)∗
(*delta x1 y1 x2 y2*)∗(*delta' x2 y2 x3 y3*)
define g_x **where** g_x = *fst*(*add z1'* (*x3*,*y3*)) − *fst*(*ext-add* (*x1*,*y1*) *z3'*)
define g_y **where** g_y = *snd*(*add z1'* (*x3*,*y3*)) − *snd*(*ext-add* (*x1*,*y1*) *z3'*)

have *x1′-expr*: *x1′* = (*x1* ∗ *x2* − *c* ∗ *y1* ∗ *y2*) / (*1* − *d* ∗ *x1* ∗ *y1* ∗ *x2* ∗ *y2*) **using** *assms*(*1*,*3*) **by** *simp*
have *y1′-expr*: *y1′* = (*x1* ∗ *y2* + *y1* ∗ *x2*) / (*1* + *d* ∗ *x1* ∗ *y1* ∗ *x2* ∗ *y2*) **using** *assms*(*1*,*3*) **by** *simp*
have *x3′-expr*: *x3′* = (*x2* ∗ *y2* − *x3* ∗ *y3*) / (*x3* ∗ *y2* − *x2* ∗ *y3*) **using** *assms*(*2*,*4*) **by** *simp*
have *y3′-expr*: *y3′* = (*x2* ∗ *y2* + *x3* ∗ *y3*) / (*x2* ∗ *x3* + *y2* ∗ *y3*) **using** *assms*(*2*,*4*) **by** *simp*

have *non-unfolded-adds*:
delta x1 y1 x2 y2 ≠ *0* **using** *delta-def assms*(*5*,*6*) **by** *auto*

have *simp1gx*:
(*x1′* ∗ *x3* − *c* ∗ *y1′* ∗ *y3*) ∗ *delta-x x1 y1 x3′ y3′* ∗ (*delta x1 y1 x2 y2* ∗ *delta′ x2 y2 x3 y3*) =
((*x1* ∗ *x2* − *y1* ∗ *y2*) ∗ *x3* ∗ *delta-plus x1 y1 x2 y2* −
(*x1* ∗ *y2* + *y1* ∗ *x2*) ∗ *y3* ∗ *delta-minus x1 y1 x2 y2*) ∗
((*x2* ∗ *y2* − *x3* ∗ *y3*) ∗ *y1* ∗ *delta-y x2 y2 x3 y3* − *x1* ∗ (*x2* ∗ *y2* + *x3* ∗ *y3*) ∗ *delta-x x2 y2 x3 y3*)

apply((*subst x1′-expr*)+, (*subst y1′-expr*)+,(*subst x3′-expr*)+,(*subst y3′-expr*)+)
apply(*subst delta-x-def*)
apply(*subst* (*1*) *delta-minus-def*[*symmetric*])
apply(*subst* (*1*) *delta-plus-def*[*symmetric*])
apply(*subst* (*4*) *delta-x-def*[*symmetric*])
apply(*subst* (*3*) *delta-y-def*[*symmetric*])
unfolding *delta′-def delta-def*
by(*simp add*: *divide-simps assms*(*5*−*8*) *c-eq-1*)

have *simp2gx*:
(*x1* ∗ *y1* − *x3′* ∗ *y3′*) ∗ *delta-minus x1′ y1′ x3 y3* ∗ (*delta x1 y1 x2 y2* ∗ *delta′ x2 y2 x3 y3*) =
(*x1* ∗ *y1* ∗ (*delta-x x2 y2 x3 y3* ∗ *delta-y x2 y2 x3 y3*) −
(*x2* ∗ *y2* − *x3* ∗ *y3*) ∗ (*x2* ∗ *y2* + *x3* ∗ *y3*)) ∗
(*delta-minus x1 y1 x2 y2* ∗ *delta-plus x1 y1 x2 y2* −
d ∗ (*x1* ∗ *x2* − *c* ∗ *y1* ∗ *y2*) ∗ (*x1* ∗ *y2* + *y1* ∗ *x2*) ∗ *x3* ∗ *y3*)
apply((*subst x1′-expr*)+, (*subst y1′-expr*)+,(*subst x3′-expr*)+,(*subst y3′-expr*)+)
apply(*subst delta-minus-def*)
apply(*subst* (*2*) *delta-minus-def*[*symmetric*])
apply(*subst* (*1*) *delta-plus-def*[*symmetric*])
apply(*subst* (*3*) *delta-x-def*[*symmetric*])
apply(*subst* (*2*) *delta-y-def*[*symmetric*])
unfolding *delta′-def delta-def*
by(*simp add*: *divide-simps assms*(*5*−*8*))

have ∃ *r1 r2 r3*. g_x ∗ $Delta_x$ = *r1* ∗ *e1* + *r2* ∗ *e2* + *r3* ∗ *e3*
unfolding g_x*-def* $Delta_x$*-def*
apply(*simp add*: *assms*(*1*,*2*))
apply(*subst delta-minus-def*[*symmetric*])

apply(*subst* (*3*) *delta-x-def*[*symmetric*])
apply(*simp add*: *divide-simps assms*(*9*,*11*))
apply(*subst* (*3*) *left-diff-distrib*)
apply(*simp add*: *simp1gx simp2gx*)
unfolding *delta-x-def delta-y-def delta′-def delta-plus-def delta-minus-def delta-def e1-def e2-def e3-def e′-def*
by(*simp add*: *t-expr c-eq-1*,*algebra*)
then have $g_x * Delta_x = 0$ $Delta_x \neq 0$
apply(*safe*)
using *e1-def e2-def e3-def assms*(*13*−*15*) **apply** *simp*
using $Delta_x$*-def delta-def delta′-def assms non-unfolded-adds* **by** *simp*
then have $g_x = 0$ **by** *auto*

have *simp1gy*:
(*x1′* ∗ *y3* + *y1′* ∗ *x3*) ∗ *delta-y x1 y1 x3′ y3′* ∗ (*delta x1 y1 x2 y2* ∗ *delta′ x2 y2 x3 y3*) =
((*x1* ∗ *x2* − *c* ∗ *y1* ∗ *y2*) ∗ *y3* ∗ *delta-plus x1 y1 x2 y2* +
(*x1* ∗ *y2* + *y1* ∗ *x2*) ∗ *x3* ∗ *delta-minus x1 y1 x2 y2*) ∗
(*x1* ∗ (*x2* ∗ *y2* − *x3* ∗ *y3*) ∗ *delta-y x2 y2 x3 y3* + *y1* ∗ (*x2* ∗ *y2* + *x3* ∗ *y3*) ∗ *delta-x x2 y2 x3 y3*)
apply((*subst x1′-expr*)+, (*subst y1′-expr*)+,(*subst x3′-expr*)+,(*subst y3′-expr*)+)
apply(*subst delta-y-def*)
apply(*subst* (*1*) *delta-minus-def*[*symmetric*])
apply(*subst* (*1*) *delta-plus-def*[*symmetric*])
apply(*subst* (*3*) *delta-x-def*[*symmetric*])
apply(*subst* (*5*) *delta-y-def*[*symmetric*])
unfolding *delta′-def delta-def*
by(*simp add*: *divide-simps assms*(*5*−*8*))

have *simp2gy*:
(*x1* ∗ *y1* + *x3′* ∗ *y3′*) ∗ *delta-plus x1′ y1′ x3 y3* ∗ (*delta x1 y1 x2 y2* ∗ *delta′ x2 y2 x3 y3*) =
(*x1* ∗ *y1* ∗ (*delta-x x2 y2 x3 y3* ∗ *delta-y x2 y2 x3 y3*) +
(*x2* ∗ *y2* − *x3* ∗ *y3*) ∗ (*x2* ∗ *y2* + *x3* ∗ *y3*)) ∗
(*delta-minus x1 y1 x2 y2* ∗ *delta-plus x1 y1 x2 y2* +
d ∗ (*x1* ∗ *x2* − *c* ∗ *y1* ∗ *y2*) ∗ (*x1* ∗ *y2* + *y1* ∗ *x2*) ∗ *x3* ∗ *y3*)
apply((*subst x1′-expr*)+, (*subst y1′-expr*)+,(*subst x3′-expr*)+,(*subst y3′-expr*)+)
apply(*subst delta-plus-def*)
apply(*subst* (*1*) *delta-minus-def*[*symmetric*])
apply(*subst* (*2*) *delta-plus-def*[*symmetric*])
apply(*subst* (*2*) *delta-x-def*[*symmetric*])
apply(*subst* (*3*) *delta-y-def*[*symmetric*])
unfolding *delta′-def delta-def*
by(*simp add*: *divide-simps assms*(*5*−*8*))

have ∃ *r1 r2 r3*. $g_y * Delta_y = r1 * e1 + r2 * e2 + r3 * e3$
unfolding g_y*-def* $Delta_y$*-def*
apply(*simp add*: *assms*(*1*,*2*))
apply(*subst delta-plus-def*[*symmetric*])

apply(*subst* (*3*) *delta-y-def* [*symmetric*])
apply(*simp add*: *divide-simps assms*)
apply(*subst left-diff-distrib*)
apply(*simp add*: *simp1gy simp2gy*)
unfolding *delta-x-def delta-y-def delta-minus-def delta-plus-def*
e1-def e2-def e3-def e'-def
by(*simp add*: *c-eq-1 t-expr*,*algebra*)

then have $g_y * Delta_y = 0$ $Delta_y \neq 0$
using *e1-def assms*(*13*−*15*) *e2-def e3-def* **apply** *simp*
using $Delta_y$*-def delta-def delta'-def assms non-unfolded-adds* **by** *simp*
then have $g_y = 0$ **by** *auto*

show *?thesis*
using ‹$g_y = 0$› ‹$g_x = 0$› **unfolding** g_x*-def* g_y*-def assms*(*3*,*4*) **by** (*simp add*: *prod-eq-iff*)
qed

lemma *add-add-add-ext-assoc*:
assumes *z1'* = (*x1'*,*y1'*) *z3'* = (*x3'*,*y3'*)
assumes *z1'* = *add* (*x1*,*y1*) (*x2*,*y2*) *z3'* = *ext-add* (*x2*,*y2*) (*x3*,*y3*)
assumes *delta-minus x1 y1 x2 y2* ≠ *0 delta-plus x1 y1 x2 y2* ≠ *0*
delta-x x2 y2 x3 y3 ≠ *0 delta-y x2 y2 x3 y3* ≠ *0*
delta-minus x1' y1' x3 y3 ≠ *0 delta-plus x1' y1' x3 y3* ≠ *0*
delta-minus x1 y1 x3' y3' ≠ *0 delta-plus x1 y1 x3' y3'* ≠ *0*
assumes *e' x1 y1* = *0 e' x2 y2* = *0 e' x3 y3* = *0*
shows *add* (*add* (*x1*,*y1*) (*x2*,*y2*)) (*x3*,*y3*) = *add* (*x1*,*y1*) (*ext-add* (*x2*,*y2*) (*x3*,*y3*))
proof −
define *e1* **where** *e1* = *e' x1 y1*
define *e2* **where** *e2* = *e' x2 y2*
define *e3* **where** *e3* = *e' x3 y3*
define $Delta_x$ **where** $Delta_x$ =
(*delta-minus x1' y1' x3 y3*)∗(*delta-minus x1 y1 x3' y3'*)∗
(*delta x1 y1 x2 y2*)∗(*delta' x2 y2 x3 y3*)
define $Delta_y$ **where** $Delta_y$ =
(*delta-plus x1' y1' x3 y3*)∗(*delta-plus x1 y1 x3' y3'*)∗
(*delta x1 y1 x2 y2*)∗(*delta' x2 y2 x3 y3*)
define g_x **where** g_x = *fst*(*add z1'* (*x3*,*y3*)) − *fst*(*add* (*x1*,*y1*) *z3'*)
define g_y **where** g_y = *snd*(*add z1'* (*x3*,*y3*)) − *snd* (*add* (*x1*,*y1*) *z3'*)

have *x1'-expr*: *x1'* = (*x1* ∗ *x2* − *c* ∗ *y1* ∗ *y2*) / (*1* − *d* ∗ *x1* ∗ *y1* ∗ *x2* ∗ *y2*) **using** *assms*(*1*,*3*) **by** *simp*
have *y1'-expr*: *y1'* = (*x1* ∗ *y2* + *y1* ∗ *x2*) / (*1* + *d* ∗ *x1* ∗ *y1* ∗ *x2* ∗ *y2*) **using** *assms*(*1*,*3*) **by** *simp*
have *x3'-expr*: *x3'* = (*x2* ∗ *y2* − *x3* ∗ *y3*) / (*x3* ∗ *y2* − *x2* ∗ *y3*) **using** *assms*(*2*,*4*) **by** *simp*
have *y3'-expr*: *y3'* = (*x2* ∗ *y2* + *x3* ∗ *y3*) / (*x2* ∗ *x3* + *y2* ∗ *y3*) **using**

assms(*2*,*4*) **by** *simp*

have *non-unfolded-adds*:
delta x1 y1 x2 y2 $\neq$ *0* **using** *delta-def assms*(*5*,*6*) **by** *auto*

have *simp1gx*:
(*x1*′ ∗ *x3* − *c* ∗ *y1*′ ∗ *y3*) ∗ *delta-minus x1 y1 x3*′ *y3*′ ∗
(*delta x1 y1 x2 y2* ∗ *delta*′ *x2 y2 x3 y3*) =
((*x1* ∗ *x2* − *y1* ∗ *y2*) ∗ *x3* ∗ *delta-plus x1 y1 x2 y2* −
(*x1* ∗ *y2* + *y1* ∗ *x2*) ∗ *y3* ∗ *delta-minus x1 y1 x2 y2*) ∗
(*delta-x x2 y2 x3 y3* ∗ *delta-y x2 y2 x3 y3* −
d ∗ *x1* ∗ *y1* ∗ (*x2* ∗ *y2* − *x3* ∗ *y3*) ∗ (*x2* ∗ *y2* + *x3* ∗ *y3*))

apply((*subst x1*′*-expr*)+, (*subst y1*′*-expr*)+,(*subst x3*′*-expr*)+,(*subst y3*′*-expr*)+)
apply(*subst delta-minus-def*)
apply(*subst delta-minus-def*[*symmetric*])
apply(*subst delta-plus-def*[*symmetric*])
apply(*subst* (*4*) *delta-x-def*[*symmetric*])
apply(*subst* (*3*) *delta-y-def*[*symmetric*])
unfolding *delta*′*-def delta-def*
by(*simp add*: *divide-simps assms*(*5*−*8*) *c-eq-1*)

have *simp2gx*:
(*x1* ∗ *x3*′ − *c* ∗ *y1* ∗ *y3*′) ∗ *delta-minus x1*′ *y1*′ *x3 y3* ∗
(*delta x1 y1 x2 y2* ∗ *delta*′ *x2 y2 x3 y3*) =
(*x1* ∗ (*x2* ∗ *y2* − *x3* ∗ *y3*) ∗ *delta-y x2 y2 x3 y3* −
c ∗ *y1* ∗ (*x2* ∗ *y2* + *x3* ∗ *y3*) ∗ *delta-x x2 y2 x3 y3*) ∗
(*delta-minus x1 y1 x2 y2* ∗ *delta-plus x1 y1 x2 y2* −
d ∗ (*x1* ∗ *x2* − *c* ∗ *y1* ∗ *y2*) ∗ (*x1* ∗ *y2* + *y1* ∗ *x2*) ∗ *x3* ∗ *y3*)
apply((*subst x1*′*-expr*)+, (*subst y1*′*-expr*)+,(*subst x3*′*-expr*)+,(*subst y3*′*-expr*)+)
apply(*subst delta-minus-def*)
apply(*subst* (*2*) *delta-minus-def*[*symmetric*])
apply(*subst* (*1*) *delta-plus-def*[*symmetric*])
apply(*subst* (*2*) *delta-x-def*[*symmetric*])
apply(*subst* (*2*) *delta-y-def*[*symmetric*])
unfolding *delta*′*-def delta-def*
by(*simp add*: *divide-simps assms*(*5*−*8*))

have ∃ *r1 r2 r3*. g_x ∗ $Delta_x$ = *r1* ∗ *e1* + *r2* ∗ *e2* + *r3* ∗ *e3*
unfolding g_x*-def* $Delta_x$*-def*
apply(*simp add*: *assms*(*1*,*2*))
apply(*subst delta-minus-def*[*symmetric*])+
apply(*simp add*: *divide-simps assms*(*9*,*11*))
apply(*subst* (*3*) *left-diff-distrib*)
apply(*simp add*: *simp1gx simp2gx*)
unfolding *delta-x-def delta-y-def delta*′*-def delta-plus-def delta-minus-def delta-def*
e1-def e2-def e3-def e′*-def*
by(*simp add*: *t-expr c-eq-1*,*algebra*)
then have g_x ∗ $Delta_x$ = *0* $Delta_x$ $\neq$ *0*

apply(*safe*)
using *e1-def e2-def e3-def assms*(*13*−*15*) **apply** *simp*
using $Delta_x$-*def delta-def delta′-def assms non-unfolded-adds* **by** *simp*
then have $g_x = 0$ **by** *auto*

have *simp1gy*: (*x1′* ∗ *y3* + *y1′* ∗ *x3*) ∗ *delta-plus x1 y1 x3′ y3′* ∗ (*delta x1 y1 x2 y2* ∗ *delta′ x2 y2 x3 y3*) =
((*x1* ∗ *x2* − *c* ∗ *y1* ∗ *y2*) ∗ *y3* ∗ *delta-plus x1 y1 x2 y2* +
(*x1* ∗ *y2* + *y1* ∗ *x2*) ∗ *x3* ∗ *delta-minus x1 y1 x2 y2*) ∗
(*delta-x x2 y2 x3 y3* ∗ *delta-y x2 y2 x3 y3* +
d ∗ *x1* ∗ *y1* ∗ (*x2* ∗ *y2* − *x3* ∗ *y3*) ∗ (*x2* ∗ *y2* + *x3* ∗ *y3*))
apply((*subst x1′-expr*)+, (*subst y1′-expr*)+,(*subst x3′-expr*)+,(*subst y3′-expr*)+)
apply(*subst delta-plus-def*)
apply(*subst* (*1*) *delta-minus-def*[*symmetric*])
apply(*subst* (*1*) *delta-plus-def*[*symmetric*])
apply(*subst* (*3*) *delta-x-def*[*symmetric*])
apply(*subst* (*4*) *delta-y-def*[*symmetric*])
unfolding *delta′-def delta-def*
by(*simp add*: *divide-simps assms*(*5*−*8*))

have *simp2gy*: (*x1* ∗ *y3′* + *y1* ∗ *x3′*) ∗ *delta-plus x1′ y1′ x3 y3* ∗ (*delta x1 y1 x2 y2* ∗ *delta′ x2 y2 x3 y3*) =
(*x1* ∗ (*x2* ∗ *y2* + *x3* ∗ *y3*) ∗ *delta-x x2 y2 x3 y3* +
y1 ∗ (*x2* ∗ *y2* − *x3* ∗ *y3*) ∗ *delta-y x2 y2 x3 y3*) ∗
(*delta-minus x1 y1 x2 y2* ∗ *delta-plus x1 y1 x2 y2* +
d ∗ (*x1* ∗ *x2* − *c* ∗ *y1* ∗ *y2*) ∗ (*x1* ∗ *y2* + *y1* ∗ *x2*) ∗ *x3* ∗ *y3*)
apply((*subst x1′-expr*)+, (*subst y1′-expr*)+,(*subst x3′-expr*)+,(*subst y3′-expr*)+)
apply(*subst delta-plus-def*)
apply(*subst* (*1*) *delta-minus-def*[*symmetric*])
apply(*subst* (*2*) *delta-plus-def*[*symmetric*])
apply(*subst* (*2*) *delta-x-def*[*symmetric*])
apply(*subst* (*2*) *delta-y-def*[*symmetric*])
unfolding *delta′-def delta-def*
by(*simp add*: *divide-simps assms*(*5*−*8*))

have ∃ *r1 r2 r3*. $g_y * Delta_y$ = *r1* ∗ *e1* + *r2* ∗ *e2* + *r3* ∗ *e3*
unfolding g_y-*def* $Delta_y$-*def*
apply(*simp add*: *assms*(*1*,*2*))
apply(*subst* (*1 2*) *delta-plus-def*[*symmetric*])
apply(*simp add*: *divide-simps assms*)
apply(*subst left-diff-distrib*)
apply(*simp add*: *simp1gy simp2gy*)
unfolding *delta-x-def delta-y-def*
delta-def delta′-def
delta-minus-def delta-plus-def
e1-def e2-def e3-def e′-def
by(*simp add*: *c-eq-1 t-expr*,*algebra*)

then have $g_y * Delta_y = 0$ $Delta_y \neq 0$

using *e1-def assms*(*13−15*) *e2-def e3-def* **apply** *simp*
using *Delta*$_y$*-def delta-def delta′-def assms non-unfolded-adds* **by** *simp*
then have $g_y = 0$ **by** *auto*

show *?thesis*
using ‹$g_y = 0$› ‹$g_x = 0$› **unfolding** g_x*-def* g_y*-def assms*(*3*,*4*) **by** (*simp add*: *prod-eq-iff*)
qed

lemma *add-add-add-ext-assoc-points*:
assumes (*x1*,*y1*) ∈ *e′-aff* (*x2*,*y2*) ∈ *e′-aff* (*x3*,*y3*) ∈ *e′-aff*
assumes *delta x1 y1 x2 y2* ≠ *0 delta′ x2 y2 x3 y3* ≠ *0*
delta (*fst* (*add* (*x1*,*y1*) (*x2*,*y2*))) (*snd* (*add* (*x1*,*y1*) (*x2*,*y2*))) *x3 y3* ≠ *0*
delta x1 y1 (*fst* (*ext-add* (*x2*,*y2*) (*x3*,*y3*))) (*snd* (*ext-add* (*x2*,*y2*) (*x3*,*y3*))) ≠ *0*
shows *add* (*add* (*x1*,*y1*) (*x2*,*y2*)) (*x3*,*y3*) = *add* (*x1*,*y1*) (*ext-add* (*x2*,*y2*) (*x3*,*y3*))
using *assms*
unfolding *e′-aff-def delta-def delta′-def*
apply(*simp del*: *add.simps*)
using *add-add-add-ext-assoc*
apply(*safe*)
by (*metis ext-add.simps prod.collapse*)

lemma *ext-add-add-ext-assoc*:
assumes *z1′* = (*x1′*,*y1′*) *z3′* = (*x3′*,*y3′*)
assumes *z1′* = *add* (*x1*,*y1*) (*x2*,*y2*) *z3′* = *ext-add* (*x2*,*y2*) (*x3*,*y3*)
assumes *delta-minus x1 y1 x2 y2* ≠ *0 delta-plus x1 y1 x2 y2* ≠ *0*
delta-x x2 y2 x3 y3 ≠ *0 delta-y x2 y2 x3 y3* ≠ *0*
delta-x x1′ y1′ x3 y3 ≠ *0 delta-y x1′ y1′ x3 y3* ≠ *0*
delta-minus x1 y1 x3′ y3′ ≠ *0 delta-plus x1 y1 x3′ y3′* ≠ *0*
assumes *e′ x1 y1* = *0 e′ x2 y2* = *0 e′ x3 y3* = *0*
shows *ext-add* (*add* (*x1*,*y1*) (*x2*,*y2*)) (*x3*,*y3*) = *add* (*x1*,*y1*) (*ext-add* (*x2*,*y2*) (*x3*,*y3*))
proof −
define *e1* **where** *e1* = *e′ x1 y1*
define *e2* **where** *e2* = *e′ x2 y2*
define *e3* **where** *e3* = *e′ x3 y3*
define *Delta*$_x$ **where** *Delta*$_x$ =
(*delta-x x1′ y1′ x3 y3*)∗(*delta-minus x1 y1 x3′ y3′*)∗
(*delta x1 y1 x2 y2*)∗(*delta′ x2 y2 x3 y3*)
define *Delta*$_y$ **where** *Delta*$_y$ =
(*delta-y x1′ y1′ x3 y3*)∗(*delta-plus x1 y1 x3′ y3′*)∗
(*delta x1 y1 x2 y2*)∗(*delta′ x2 y2 x3 y3*)
define g_x **where** g_x = *fst*(*ext-add z1′* (*x3*,*y3*)) − *fst*(*add* (*x1*,*y1*) *z3′*)
define g_y **where** g_y = *snd*(*ext-add z1′* (*x3*,*y3*)) − *snd* (*add* (*x1*,*y1*) *z3′*)

have *x1′-expr*: *x1′* = (*x1* ∗ *x2* − *c* ∗ *y1* ∗ *y2*) / (*1* − *d* ∗ *x1* ∗ *y1* ∗ *x2* ∗ *y2*) **using** *assms*(*1*,*3*) **by** *simp*

have *y1'-expr*: $y1' = (x1 * y2 + y1 * x2) \;/\; (1 + d * x1 * y1 * x2 * y2)$ **using** *assms*(*1*,*3*) **by** *simp*
have *x3'-expr*: $x3' = (x2 * y2 - x3 * y3) \;/\; (x3 * y2 - x2 * y3)$ **using** *assms*(*2*,*4*) **by** *simp*
have *y3'-expr*: $y3' = (x2 * y2 + x3 * y3) \;/\; (x2 * x3 + y2 * y3)$ **using** *assms*(*2*,*4*) **by** *simp*

have *non-unfolded-adds*:
delta x1 y1 x2 y2 $\neq$ *0* **using** *delta-def assms*(*5*,*6*) **by** *auto*

have *simp1gx*:
(*x1'* $*$ *y1'* $-$ *x3* $*$ *y3*) $*$ *delta-minus x1 y1 x3' y3'* $*$ (*delta x1 y1 x2 y2* $*$ *delta' x2 y2 x3 y3*) =
((*x1* $*$ *x2* $-$ *y1* $*$ *y2*) $*$ (*x1* $*$ *y2* $+$ *y1* $*$ *x2*) $-$
x3 $*$ *y3* $*$ (*delta-minus x1 y1 x2 y2* $*$ *delta-plus x1 y1 x2 y2*)) $*$
(*delta-x x2 y2 x3 y3* $*$ *delta-y x2 y2 x3 y3* $-$
d $*$ *x1* $*$ *y1* $*$ (*x2* $*$ *y2* $-$ *x3* $*$ *y3*) $*$ (*x2* $*$ *y2* $+$ *x3* $*$ *y3*))

apply((*subst x1'-expr*)+, (*subst y1'-expr*)+,(*subst x3'-expr*)+,(*subst y3'-expr*)+)
apply(*subst delta-minus-def*)
apply(*subst delta-minus-def*[*symmetric*])
apply(*subst delta-plus-def*[*symmetric*])
apply(*subst* (*4*) *delta-x-def*[*symmetric*])
apply(*subst* (*3*) *delta-y-def*[*symmetric*])
unfolding *delta'-def delta-def*
by(*simp add*: *divide-simps assms*(*5*$-$*8*) *c-eq-1*)

have *simp2gx*:
(*x1* $*$ *x3'* $-$ *c* $*$ *y1* $*$ *y3'*) $*$ *delta-x x1' y1' x3 y3* $*$ (*delta x1 y1 x2 y2* $*$ *delta' x2 y2 x3 y3*) =
(*x1* $*$ (*x2* $*$ *y2* $-$ *x3* $*$ *y3*) $*$ *delta-y x2 y2 x3 y3* $-$
c $*$ *y1* $*$ (*x2* $*$ *y2* $+$ *x3* $*$ *y3*) $*$ *delta-x x2 y2 x3 y3*) $*$
(*x3* $*$ (*x1* $*$ *y2* $+$ *y1* $*$ *x2*) $*$ *delta-minus x1 y1 x2 y2* $-$
(*x1* $*$ *x2* $-$ *c* $*$ *y1* $*$ *y2*) $*$ *y3* $*$ *delta-plus x1 y1 x2 y2*)
apply((*subst x1'-expr*)+, (*subst y1'-expr*)+,(*subst x3'-expr*)+,(*subst y3'-expr*)+)
apply(*subst delta-x-def*)
apply(*subst* (*1*) *delta-minus-def*[*symmetric*])
apply(*subst* (*1*) *delta-plus-def*[*symmetric*])
apply(*subst* (*2*) *delta-x-def*[*symmetric*])
apply(*subst* (*2*) *delta-y-def*[*symmetric*])
unfolding *delta'-def delta-def*
by(*simp add*: *divide-simps assms*(*5*$-$*8*))

have $\exists$ *r1 r2 r3*. $g_x * Delta_x = r1 * e1 + r2 * e2 + r3 * e3$
unfolding g_x*-def* $Delta_x$*-def*
apply(*simp add*: *assms*(*1*,*2*))
apply(*subst delta-minus-def*[*symmetric*])
apply(*subst* (*2*) *delta-x-def*[*symmetric*])
apply(*simp add*: *divide-simps assms*(*9*,*11*))

apply(*subst* (*3*) *left-diff-distrib*)
apply(*simp add*: *simp1gx simp2gx*)
unfolding *delta-x-def delta-y-def delta′-def delta-plus-def delta-minus-def delta-def e1-def e2-def e3-def e′-def*
by(*simp add*: *t-expr c-eq-1*,*algebra*)
then have $g_x * Delta_x = 0$ $Delta_x \neq 0$
apply(*safe*)
using *e1-def e2-def e3-def assms*(*13*−*15*) **apply** *simp*
using $Delta_x$*-def delta-def delta′-def assms non-unfolded-adds* **by** *simp*
then have $g_x = 0$ **by** *auto*

have *simp1gy*:
$(x1' * y1' + x3 * y3) * delta\text{-}plus\ x1\ y1\ x3'\ y3' * (delta\ x1\ y1\ x2\ y2 * delta'\ x2\ y2\ x3\ y3) =$
$((x1 * x2 - c * y1 * y2) * (x1 * y2 + y1 * x2) +$
$x3 * y3 * (delta\text{-}minus\ x1\ y1\ x2\ y2 * delta\text{-}plus\ x1\ y1\ x2\ y2)) *$
$(delta\text{-}x\ x2\ y2\ x3\ y3 * delta\text{-}y\ x2\ y2\ x3\ y3 +$
$d * x1 * y1 * (x2 * y2 - x3 * y3) * (x2 * y2 + x3 * y3))$
apply((*subst x1′-expr*)+, (*subst y1′-expr*)+,(*subst x3′-expr*)+,(*subst y3′-expr*)+)
apply(*subst delta-plus-def*)
apply(*subst* (*1*) *delta-minus-def*[*symmetric*])
apply(*subst* (*1*) *delta-plus-def*[*symmetric*])
apply(*subst* (*3*) *delta-x-def*[*symmetric*])
apply(*subst* (*4*) *delta-y-def*[*symmetric*])
unfolding *delta′-def delta-def*
by(*simp add*: *divide-simps assms*(*5*−*8*))

have *simp2gy*:
$(x1 * y3' + y1 * x3') * delta\text{-}y\ x1'\ y1'\ x3\ y3 * (delta\ x1\ y1\ x2\ y2 * delta'\ x2\ y2\ x3\ y3) =$
$(x1 * (x2 * y2 + x3 * y3) * delta\text{-}x\ x2\ y2\ x3\ y3 + y1 * (x2 * y2 - x3 * y3) * delta\text{-}y\ x2\ y2\ x3\ y3) *$
$((x1 * x2 - c * y1 * y2) * x3 * delta\text{-}plus\ x1\ y1\ x2\ y2 +$
$(x1 * y2 + y1 * x2) * y3 * delta\text{-}minus\ x1\ y1\ x2\ y2)$
apply((*subst x1′-expr*)+, (*subst y1′-expr*)+,(*subst x3′-expr*)+,(*subst y3′-expr*)+)
apply(*subst delta-y-def*)
apply(*subst* (*1*) *delta-minus-def*[*symmetric*])
apply(*subst* (*1*) *delta-plus-def*[*symmetric*])
apply(*subst* (*2*) *delta-x-def*[*symmetric*])
apply(*subst* (*2*) *delta-y-def*[*symmetric*])
unfolding *delta′-def delta-def*
by(*simp add*: *divide-simps assms*(*5*−*8*))

have $\exists\ r1\ r2\ r3.\ g_y * Delta_y = r1 * e1 + r2 * e2 + r3 * e3$
unfolding g_y*-def* $Delta_y$*-def*
apply(*simp add*: *assms*(*1*,*2*))
apply(*subst* (*1*) *delta-plus-def*[*symmetric*])
apply(*subst* (*2*) *delta-y-def*[*symmetric*])
apply(*simp add*: *divide-simps assms*)

apply(*subst left-diff-distrib*)
apply(*simp add*: *simp1gy simp2gy*)
unfolding *delta-x-def delta-y-def*
delta-def delta′-def
delta-minus-def delta-plus-def
e1-def e2-def e3-def e′-def
by(*simp add*: *c-eq-1 t-expr*,*algebra*)

then have $g_y * Delta_y = 0$ $Delta_y \neq 0$
using *e1-def assms*(*13*−*15*) *e2-def e3-def* **apply** *simp*
using $Delta_y$*-def delta-def delta′-def assms non-unfolded-adds* **by** *simp*
then have $g_y = 0$ **by** *auto*

show *?thesis*
using ‹$g_y = 0$› ‹$g_x = 0$› **unfolding** g_x*-def* g_y*-def assms*(*3*,*4*) **by** (*simp add*: *prod-eq-iff*)
qed

lemma *ext-add-add-ext-assoc-points*:
assumes (*x1*,*y1*) ∈ *e′-aff* (*x2*,*y2*) ∈ *e′-aff* (*x3*,*y3*) ∈ *e′-aff*
assumes *delta x1 y1 x2 y2* ≠ *0 delta′ x2 y2 x3 y3* ≠ *0*
delta′ (*fst* (*add* (*x1*,*y1*) (*x2*,*y2*))) (*snd* (*add* (*x1*,*y1*) (*x2*,*y2*))) *x3 y3* ≠ *0*
delta x1 y1 (*fst* (*ext-add* (*x2*,*y2*) (*x3*,*y3*))) (*snd* (*ext-add* (*x2*,*y2*) (*x3*,*y3*))) ≠ *0*
shows *ext-add* (*add* (*x1*,*y1*) (*x2*,*y2*)) (*x3*,*y3*) = *add* (*x1*,*y1*) (*ext-add* (*x2*,*y2*) (*x3*,*y3*))
using *assms*
unfolding *e′-aff-def delta-def delta′-def*
apply(*simp del*: *add.simps*)
using *ext-add-add-ext-assoc*
apply(*safe*)
using *prod.collapse ext-add.simps* **by** *metis*

lemma *ext-add-add-add-assoc*:
assumes *z1′* = (*x1′*,*y1′*) *z3′* = (*x3′*,*y3′*)
assumes *z1′* = *add* (*x1*,*y1*) (*x2*,*y2*) *z3′* = *add* (*x2*,*y2*) (*x3*,*y3*)
assumes *delta-minus x1 y1 x2 y2* ≠ *0 delta-plus x1 y1 x2 y2* ≠ *0*
delta-x x1′ y1′ x3 y3 ≠ *0 delta-y x1′ y1′ x3 y3* ≠ *0*
delta-minus x1 y1 x3′ y3′ ≠ *0 delta-plus x1 y1 x3′ y3′* ≠ *0*
delta-minus x2 y2 x3 y3 ≠ *0 delta-plus x2 y2 x3 y3* ≠ *0*
assumes *e′ x1 y1* = *0 e′ x2 y2* = *0 e′ x3 y3* = *0*
shows *ext-add* (*add* (*x1*,*y1*) (*x2*,*y2*)) (*x3*,*y3*) = *add* (*x1*,*y1*) (*add* (*x2*,*y2*) (*x3*,*y3*))
proof −
define *e1* **where** *e1* = *e′ x1 y1*
define *e2* **where** *e2* = *e′ x2 y2*
define *e3* **where** *e3* = *e′ x3 y3*
define $Delta_x$ **where** $Delta_x$ =

(*delta-x x1' y1' x3 y3*)∗(*delta-minus x1 y1 x3' y3'*)∗
(*delta x1 y1 x2 y2*)∗(*delta x2 y2 x3 y3*)
define $Delta_y$ **where** $Delta_y$ =
(*delta-y x1' y1' x3 y3*)∗(*delta-plus x1 y1 x3' y3'*)∗
(*delta x1 y1 x2 y2*)∗(*delta x2 y2 x3 y3*)
define g_x **where** g_x = *fst*(*ext-add z1'* (*x3*,*y3*)) − *fst*(*add* (*x1*,*y1*) *z3'*)
define g_y **where** g_y = *snd*(*ext-add z1'* (*x3*,*y3*)) − *snd* (*add* (*x1*,*y1*) *z3'*)

have *x1'-expr*: *x1'* = (*x1* ∗ *x2* − *c* ∗ *y1* ∗ *y2*) / (*1* − *d* ∗ *x1* ∗ *y1* ∗ *x2* ∗ *y2*) **using** *assms*(*1*,*3*) **by** *simp*
have *y1'-expr*: *y1'* = (*x1* ∗ *y2* + *y1* ∗ *x2*) / (*1* + *d* ∗ *x1* ∗ *y1* ∗ *x2* ∗ *y2*) **using** *assms*(*1*,*3*) **by** *simp*
have *x3'-expr*: *x3'* = (*x2* ∗ *x3* − *c* ∗ *y2* ∗ *y3*) / (*1* − *d* ∗ *x2* ∗ *y2* ∗ *x3* ∗ *y3*) **using** *assms*(*2*,*4*) **by** *simp*
have *y3'-expr*: *y3'* = (*x2* ∗ *y3* + *y2* ∗ *x3*) / (*1* + *d* ∗ *x2* ∗ *y2* ∗ *x3* ∗ *y3*) **using** *assms*(*2*,*4*) **by** *simp*

have *non-unfolded-adds*:
delta x1 y1 x2 y2 ≠ *0* **using** *delta-def assms*(*5*,*6*) **by** *auto*

have *simp1gx*:
(*x1'* ∗ *y1'* − *x3* ∗ *y3*) ∗ *delta-minus x1 y1 x3' y3'* ∗ (*delta x1 y1 x2 y2* ∗ *delta x2 y2 x3 y3*) =
((*x1* ∗ *x2* − *y1* ∗ *y2*) ∗ (*x1* ∗ *y2* + *y1* ∗ *x2*) −
x3 ∗ *y3* ∗ (*delta-minus x1 y1 x2 y2* ∗ *delta-plus x1 y1 x2 y2*)) ∗
(*delta-minus x2 y2 x3 y3* ∗ *delta-plus x2 y2 x3 y3* −
d ∗ *x1* ∗ *y1* ∗ (*x2* ∗ *x3* − *y2* ∗ *y3*) ∗ (*x2* ∗ *y3* + *y2* ∗ *x3*))

apply((*subst x1'-expr*)+, (*subst y1'-expr*)+,(*subst x3'-expr*)+,(*subst y3'-expr*)+)
apply(*subst delta-minus-def*)
apply(*subst* (*1 3*) *delta-minus-def*[*symmetric*])
apply(*subst* (*1 2*) *delta-plus-def*[*symmetric*])
unfolding *delta'-def delta-def*
by(*simp add*: *divide-simps assms c-eq-1*)

have *simp2gx*:
(*x1* ∗ *x3'* − *c* ∗ *y1* ∗ *y3'*) ∗ *delta-x x1' y1' x3 y3* ∗ (*delta x1 y1 x2 y2* ∗ *delta x2 y2 x3 y3*) =
(*x1* ∗ (*x2* ∗ *x3* − *c* ∗ *y2* ∗ *y3*) ∗ *delta-plus x2 y2 x3 y3* −
c ∗ *y1* ∗ (*x2* ∗ *y3* + *y2* ∗ *x3*) ∗ *delta-minus x2 y2 x3 y3*) ∗
(*x3* ∗ (*x1* ∗ *y2* + *y1* ∗ *x2*) ∗ *delta-minus x1 y1 x2 y2* −
(*x1* ∗ *x2* − *c* ∗ *y1* ∗ *y2*) ∗ *y3* ∗ *delta-plus x1 y1 x2 y2*)
apply((*subst x1'-expr*)+, (*subst y1'-expr*)+,(*subst x3'-expr*)+,(*subst y3'-expr*)+)
apply(*subst delta-x-def*)
apply(*subst* (*1 2*) *delta-minus-def*[*symmetric*])
apply(*subst* (*1 2*) *delta-plus-def*[*symmetric*])
unfolding *delta'-def delta-def*
by(*simp add*: *divide-simps assms*)

have ∃ *r1 r2 r3*. g_x ∗ $Delta_x$ = *r1* ∗ *e1* + *r2* ∗ *e2* + *r3* ∗ *e3*
unfolding g_x*-def* $Delta_x$*-def*
apply(*simp add*: *assms*(*1*,*2*))
apply(*subst delta-minus-def*[*symmetric*])
apply(*subst* (*2*) *delta-x-def*[*symmetric*])
apply(*simp add*: *divide-simps assms*)
apply(*subst* (*3*) *left-diff-distrib*)
apply(*simp add*: *simp1gx simp2gx*)
unfolding *delta-x-def delta-y-def delta′-def delta-plus-def delta-minus-def delta-def*
e1-def e2-def e3-def e′-def
by(*simp add*: *t-expr c-eq-1*,*algebra*)
then have g_x ∗ $Delta_x$ = *0* $Delta_x$ ≠ *0*
apply(*safe*)
using *e1-def e2-def e3-def assms*(*13*−*15*) **apply** *simp*
using $Delta_x$*-def delta-def delta′-def assms non-unfolded-adds* **by** *simp*
then have g_x = *0* **by** *auto*

have *simp1gy*:
(*x1′* ∗ *y1′* + *x3* ∗ *y3*) ∗ *delta-plus x1 y1 x3′ y3′* ∗ (*delta x1 y1 x2 y2* ∗ *delta x2 y2 x3 y3*) =
((*x1* ∗ *x2* − *c* ∗ *y1* ∗ *y2*) ∗ (*x1* ∗ *y2* + *y1* ∗ *x2*) +
x3 ∗ *y3* ∗ (*delta-minus x1 y1 x2 y2* ∗ *delta-plus x1 y1 x2 y2*)) ∗
(*delta-minus x2 y2 x3 y3* ∗ *delta-plus x2 y2 x3 y3* +
d ∗ *x1* ∗ *y1* ∗ (*x2* ∗ *x3* − *c* ∗ *y2* ∗ *y3*) ∗ (*x2* ∗ *y3* + *y2* ∗ *x3*))
apply((*subst x1′-expr*)+, (*subst y1′-expr*)+,(*subst x3′-expr*)+,(*subst y3′-expr*)+)
apply(*subst delta-plus-def*)
apply(*subst* (*1 2*) *delta-minus-def*[*symmetric*])
apply(*subst* (*1 3*) *delta-plus-def*[*symmetric*])
unfolding *delta′-def delta-def*
by(*simp add*: *divide-simps assms*)

have *simp2gy*:
(*x1* ∗ *y3′* + *y1* ∗ *x3′*) ∗ *delta-y x1′ y1′ x3 y3* ∗ (*delta x1 y1 x2 y2* ∗ *delta x2 y2 x3 y3*) =
(*x1* ∗ (*x2* ∗ *y3* + *y2* ∗ *x3*) ∗ *delta-minus x2 y2 x3 y3* +
y1 ∗ (*x2* ∗ *x3* − *c* ∗ *y2* ∗ *y3*) ∗ *delta-plus x2 y2 x3 y3*) ∗
((*x1* ∗ *x2* − *c* ∗ *y1* ∗ *y2*) ∗ *x3* ∗ *delta-plus x1 y1 x2 y2* +
(*x1* ∗ *y2* + *y1* ∗ *x2*) ∗ *y3* ∗ *delta-minus x1 y1 x2 y2*)
apply((*subst x1′-expr*)+, (*subst y1′-expr*)+,(*subst x3′-expr*)+,(*subst y3′-expr*)+)
apply(*subst delta-y-def*)
apply(*subst* (*1 2*) *delta-minus-def*[*symmetric*])
apply(*subst* (*1 2*) *delta-plus-def*[*symmetric*])
unfolding *delta′-def delta-def*
by(*simp add*: *divide-simps assms*)

have ∃ *r1 r2 r3*. g_y ∗ $Delta_y$ = *r1* ∗ *e1* + *r2* ∗ *e2* + *r3* ∗ *e3*
unfolding g_y*-def* $Delta_y$*-def*
apply(*simp add*: *assms*(*1*,*2*))
apply(*subst* (*2*) *delta-y-def*[*symmetric*])

apply(*subst* (*1*) *delta-plus-def*[*symmetric*])
apply(*simp add*: *divide-simps assms*)
apply(*subst left-diff-distrib*)
apply(*simp add*: *simp1gy simp2gy*)
unfolding *delta-x-def delta-y-def*
delta-def delta'-def
delta-minus-def delta-plus-def
e1-def e2-def e3-def e'-def
by(*simp add*: *c-eq-1 t-expr*,*algebra*)

then have $g_y * Delta_y = 0$ $Delta_y \neq 0$
using *e1-def assms*(*13*−*15*) *e2-def e3-def* **apply** *simp*
using $Delta_y$*-def delta-def delta'-def assms non-unfolded-adds* **by** *simp*
then have $g_y = 0$ **by** *auto*

show *?thesis*
using ‹$g_y = 0$› ‹$g_x = 0$› **unfolding** g_x*-def* g_y*-def assms*(*3*,*4*) **by** (*simp add*: *prod-eq-iff*)
qed

lemma *ext-add-add-add-assoc-points*:
assumes (*x1*,*y1*) ∈ *e'-aff* (*x2*,*y2*) ∈ *e'-aff* (*x3*,*y3*) ∈ *e'-aff*
assumes *delta x1 y1 x2 y2* ≠ *0 delta x2 y2 x3 y3* ≠ *0*
delta' (*fst* (*add* (*x1*,*y1*) (*x2*,*y2*))) (*snd* (*add* (*x1*,*y1*) (*x2*,*y2*))) *x3 y3* ≠ *0*
delta x1 y1 (*fst* (*add* (*x2*,*y2*) (*x3*,*y3*))) (*snd* (*add* (*x2*,*y2*) (*x3*,*y3*))) ≠ *0*
shows *ext-add* (*add* (*x1*,*y1*) (*x2*,*y2*)) (*x3*,*y3*) = *add* (*x1*,*y1*) (*add* (*x2*,*y2*) (*x3*,*y3*))
using *assms*
unfolding *e'-aff-def delta-def delta'-def*
apply(*simp del*: *add.simps*)
using *ext-add-add-add-assoc*
apply(*safe*)
using *prod.collapse* **by** *blast*

lemma *ext-add-ext-add-assoc*:
assumes *z1'* = (*x1'*,*y1'*) *z3'* = (*x3'*,*y3'*)
assumes *z1'* = *add* (*x1*,*y1*) (*x2*,*y2*) *z3'* = *add* (*x2*,*y2*) (*x3*,*y3*)
assumes *delta-minus x1 y1 x2 y2* ≠ *0 delta-plus x1 y1 x2 y2* ≠ *0*
delta-x x1' y1' x3 y3 ≠ *0 delta-y x1' y1' x3 y3* ≠ *0*
delta-x x1 y1 x3' y3' ≠ *0 delta-y x1 y1 x3' y3'* ≠ *0*
delta-minus x2 y2 x3 y3 ≠ *0 delta-plus x2 y2 x3 y3* ≠ *0*
assumes *e' x1 y1* = *0 e' x2 y2* = *0 e' x3 y3* = *0*
shows *ext-add* (*add* (*x1*,*y1*) (*x2*,*y2*)) (*x3*,*y3*) = *ext-add* (*x1*,*y1*) (*add* (*x2*,*y2*) (*x3*,*y3*))
proof −
define *e1* **where** *e1* = *e' x1 y1*
define *e2* **where** *e2* = *e' x2 y2*
define *e3* **where** *e3* = *e' x3 y3*
define $Delta_x$ **where** $Delta_x$ =

(*delta-x x1' y1' x3 y3*)∗(*delta-x x1 y1 x3' y3'*)∗
(*delta x1 y1 x2 y2*)∗(*delta x2 y2 x3 y3*)
define $Delta_y$ **where** $Delta_y$ =
(*delta-y x1' y1' x3 y3*)∗(*delta-y x1 y1 x3' y3'*)∗
(*delta x1 y1 x2 y2*)∗(*delta x2 y2 x3 y3*)
define g_x **where** g_x = *fst*(*ext-add z1'* (*x3*,*y3*)) − *fst*(*ext-add* (*x1*,*y1*) *z3'*)
define g_y **where** g_y = *snd*(*ext-add z1'* (*x3*,*y3*)) − *snd* (*ext-add* (*x1*,*y1*) *z3'*)

have *x1'-expr*: *x1'* = (*x1* ∗ *x2* − *c* ∗ *y1* ∗ *y2*) / (*1* − *d* ∗ *x1* ∗ *y1* ∗ *x2* ∗ *y2*) **using** *assms*(*1*,*3*) **by** *simp*
have *y1'-expr*: *y1'* = (*x1* ∗ *y2* + *y1* ∗ *x2*) / (*1* + *d* ∗ *x1* ∗ *y1* ∗ *x2* ∗ *y2*) **using** *assms*(*1*,*3*) **by** *simp*
have *x3'-expr*: *x3'* = (*x2* ∗ *x3* − *c* ∗ *y2* ∗ *y3*) / (*1* − *d* ∗ *x2* ∗ *y2* ∗ *x3* ∗ *y3*) **using** *assms*(*2*,*4*) **by** *simp*
have *y3'-expr*: *y3'* = (*x2* ∗ *y3* + *y2* ∗ *x3*) / (*1* + *d* ∗ *x2* ∗ *y2* ∗ *x3* ∗ *y3*) **using** *assms*(*2*,*4*) **by** *simp*

have *non-unfolded-adds*:
delta x1 y1 x2 y2 ≠ *0* **using** *delta-def assms*(*5*,*6*) **by** *auto*

have *simp1gx*:
(*x1'* ∗ *y1'* − *x3* ∗ *y3*) ∗ *delta-x x1 y1 x3' y3'* ∗ (*delta x1 y1 x2 y2* ∗ *delta x2 y2 x3 y3*) =
((*x1* ∗ *x2* − *y1* ∗ *y2*) ∗ (*x1* ∗ *y2* + *y1* ∗ *x2*) −
x3 ∗ *y3* ∗ (*delta-minus x1 y1 x2 y2* ∗ *delta-plus x1 y1 x2 y2*)) ∗
((*x2* ∗ *x3* − *y2* ∗ *y3*) ∗ *y1* ∗ *delta-plus x2 y2 x3 y3* −
x1 ∗ (*x2* ∗ *y3* + *y2* ∗ *x3*) ∗ *delta-minus x2 y2 x3 y3*)

apply((*subst x1'-expr*)+, (*subst y1'-expr*)+,(*subst x3'-expr*)+,(*subst y3'-expr*)+)
apply(*subst delta-x-def*)
apply(*subst* (*1 2*) *delta-minus-def*[*symmetric*])
apply(*subst* (*1 2*) *delta-plus-def*[*symmetric*])
unfolding *delta'-def delta-def*
by(*simp add*: *divide-simps assms c-eq-1*)

have *simp2gx*:
(*x1* ∗ *y1* − *x3'* ∗ *y3'*) ∗ *delta-x x1' y1' x3 y3* ∗ (*delta x1 y1 x2 y2* ∗ *delta x2 y2 x3 y3*) =
(*x1* ∗ *y1* ∗ (*delta-minus x2 y2 x3 y3* ∗ *delta-plus x2 y2 x3 y3*) −
(*x2* ∗ *x3* − *c* ∗ *y2* ∗ *y3*) ∗ (*x2* ∗ *y3* + *y2* ∗ *x3*)) ∗
(*x3* ∗ (*x1* ∗ *y2* + *y1* ∗ *x2*) ∗ *delta-minus x1 y1 x2 y2* −
(*x1* ∗ *x2* − *c* ∗ *y1* ∗ *y2*) ∗ *y3* ∗ *delta-plus x1 y1 x2 y2*)
apply((*subst x1'-expr*)+, (*subst y1'-expr*)+,(*subst x3'-expr*)+,(*subst y3'-expr*)+)
apply(*subst delta-x-def*)
apply(*subst* (*1 2*) *delta-minus-def*[*symmetric*])
apply(*subst* (*1 2*) *delta-plus-def*[*symmetric*])
unfolding *delta'-def delta-def*
by(*simp add*: *divide-simps assms*)

have ∃ *r1 r2 r3*. g_x ∗ $Delta_x$ = *r1* ∗ *e1* + *r2* ∗ *e2* + *r3* ∗ *e3*
unfolding g_x-*def* $Delta_x$-*def*
apply(*simp add*: *assms*(*1*,*2*))
apply(*subst* (*2 4*) *delta-x-def*[*symmetric*])
apply(*simp add*: *divide-simps assms*)
apply(*subst* (*3*) *left-diff-distrib*)
apply(*simp add*: *simp1gx simp2gx*)
unfolding *delta-x-def delta-y-def delta′-def delta-plus-def delta-minus-def delta-def e1-def e2-def e3-def e′-def*
by(*simp add*: *t-expr c-eq-1*,*algebra*)
then have g_x ∗ $Delta_x$ = *0* $Delta_x$ ≠ *0*
apply(*safe*)
using *e1-def e2-def e3-def assms*(*13*−*15*) **apply** *simp*
using $Delta_x$-*def delta-def delta′-def assms non-unfolded-adds* **by** *simp*
then have g_x = *0* **by** *auto*

have *simp1gy*:
(*x1′* ∗ *y1′* + *x3* ∗ *y3*) ∗ *delta-y x1 y1 x3′ y3′* ∗ (*delta x1 y1 x2 y2* ∗ *delta x2 y2 x3 y3*) =
((*x1* ∗ *x2* − *c* ∗ *y1* ∗ *y2*) ∗ (*x1* ∗ *y2* + *y1* ∗ *x2*) +
x3 ∗ *y3* ∗ (*delta-minus x1 y1 x2 y2* ∗ *delta-plus x1 y1 x2 y2*)) ∗
(*x1* ∗ (*x2* ∗ *x3* − *c* ∗ *y2* ∗ *y3*) ∗ *delta-plus x2 y2 x3 y3* +
y1 ∗ (*x2* ∗ *y3* + *y2* ∗ *x3*) ∗ *delta-minus x2 y2 x3 y3*)
apply((*subst x1′-expr*)+, (*subst y1′-expr*)+,(*subst x3′-expr*)+,(*subst y3′-expr*)+)
apply(*subst delta-y-def*)
apply(*subst* (*1 2*) *delta-minus-def*[*symmetric*])
apply(*subst* (*1 2*) *delta-plus-def*[*symmetric*])
unfolding *delta′-def delta-def*
by(*simp add*: *divide-simps assms*)

have *simp2gy*:
(*x1* ∗ *y1* + *x3′* ∗ *y3′*) ∗ *delta-y x1′ y1′ x3 y3* ∗ (*delta x1 y1 x2 y2* ∗ *delta x2 y2 x3 y3*) =
(*x1* ∗ *y1* ∗ (*delta-minus x2 y2 x3 y3* ∗ *delta-plus x2 y2 x3 y3*) +
(*x2* ∗ *x3* − *c* ∗ *y2* ∗ *y3*) ∗ (*x2* ∗ *y3* + *y2* ∗ *x3*)) ∗
((*x1* ∗ *x2* − *c* ∗ *y1* ∗ *y2*) ∗ *x3* ∗ *delta-plus x1 y1 x2 y2* +
(*x1* ∗ *y2* + *y1* ∗ *x2*) ∗ *y3* ∗ *delta-minus x1 y1 x2 y2*)
apply((*subst x1′-expr*)+, (*subst y1′-expr*)+,(*subst x3′-expr*)+,(*subst y3′-expr*)+)
apply(*subst delta-y-def*)
apply(*subst* (*1 2*) *delta-minus-def*[*symmetric*])
apply(*subst* (*1 2*) *delta-plus-def*[*symmetric*])
unfolding *delta′-def delta-def*
by(*simp add*: *divide-simps assms*)

have ∃ *r1 r2 r3*. g_y ∗ $Delta_y$ = *r1* ∗ *e1* + *r2* ∗ *e2* + *r3* ∗ *e3*
unfolding g_y-*def* $Delta_y$-*def*
apply(*simp add*: *assms*(*1*,*2*))
apply(*subst* (*2 4*) *delta-y-def*[*symmetric*])
apply(*simp add*: *divide-simps assms*)

  apply(*subst left-diff-distrib*)
  apply(*simp add*: *simp1gy simp2gy*)
  unfolding *delta-x-def delta-y-def*
      delta-def delta'-def
      delta-minus-def delta-plus-def
      e1-def e2-def e3-def e'-def
  by(*simp add*: *c-eq-1 t-expr*,*algebra*)

 then have $g_y * Delta_y = 0$ $Delta_y \neq 0$
  using *e1-def assms*(*13*−*15*) *e2-def e3-def* **apply** *simp*
  using $Delta_y$*-def delta-def delta'-def assms non-unfolded-adds* **by** *simp*
 then have $g_y = 0$ **by** *auto*

 show *?thesis*
  using ‹$g_y = 0$› ‹$g_x = 0$› **unfolding** g_x*-def* g_y*-def assms*(*3*,*4*) **by** (*simp add*: *prod-eq-iff*)
qed

lemma *ext-ext-add-add-assoc*:
 assumes *z1' = (x1',y1') z3' = (x3',y3')*
 assumes *z1' = ext-add (x1,y1) (x2,y2) z3' = add (x2,y2) (x3,y3)*
 assumes *delta-x x1 y1 x2 y2* ≠ *0 delta-y x1 y1 x2 y2* ≠ *0*
    delta-x x1' y1' x3 y3 ≠ *0 delta-y x1' y1' x3 y3* ≠ *0*
    delta-minus x1 y1 x3' y3' ≠ *0 delta-plus x1 y1 x3' y3'* ≠ *0*
    delta-minus x2 y2 x3 y3 ≠ *0 delta-plus x2 y2 x3 y3* ≠ *0*
 assumes *e' x1 y1 = 0 e' x2 y2 = 0 e' x3 y3 = 0*
 shows *ext-add (ext-add (x1,y1) (x2,y2)) (x3,y3) = add (x1,y1) (add (x2,y2) (x3,y3))*
proof −
 define *e1* **where** *e1 = e' x1 y1*
 define *e2* **where** *e2 = e' x2 y2*
 define *e3* **where** *e3 = e' x3 y3*
 define $Delta_x$ **where** $Delta_x$ =
 (*delta-x x1' y1' x3 y3*)∗(*delta-minus x1 y1 x3' y3'*)∗
 (*delta' x1 y1 x2 y2*)∗(*delta x2 y2 x3 y3*)
 define $Delta_y$ **where** $Delta_y$ =
 (*delta-y x1' y1' x3 y3*)∗(*delta-plus x1 y1 x3' y3'*)∗
 (*delta' x1 y1 x2 y2*)∗(*delta x2 y2 x3 y3*)
 define g_x **where** g_x = *fst*(*ext-add z1' (x3,y3)*) − *fst*(*add (x1,y1) z3'*)
 define g_y **where** g_y = *snd*(*ext-add z1' (x3,y3)*) − *snd* (*add (x1,y1) z3'*)

 have *x1'-expr*: *x1' = (x1 ∗ y1 − x2 ∗ y2) / (x2 ∗ y1 − x1 ∗ y2)* **using** *assms*(*1*,*3*) **by** *simp*
 have *y1'-expr*: *y1' = (x1 ∗ y1 + x2 ∗ y2) / (x1 ∗ x2 + y1 ∗ y2)* **using** *assms*(*1*,*3*) **by** *simp*
 have *x3'-expr*: *x3' = (x2 ∗ x3 − c ∗ y2 ∗ y3) / (1 − d ∗ x2 ∗ y2 ∗ x3 ∗ y3)* **using** *assms*(*2*,*4*) **by** *simp*
 have *y3'-expr*: *y3' = (x2 ∗ y3 + y2 ∗ x3) / (1 + d ∗ x2 ∗ y2 ∗ x3 ∗ y3)* **using** *assms*(*2*,*4*) **by** *simp*

have *non-unfolded-adds*:
delta′ x1 y1 x2 y2 ≠ *0* **using** *delta′-def assms*(*5*,*6*) **by** *auto*

have *simp1gx*:
(*x1* ∗ *x3′* − *c* ∗ *y1* ∗ *y3′*) ∗ *delta-x x1′ y1′ x3 y3* ∗ (*delta′ x1 y1 x2 y2* ∗ *delta x2 y2 x3 y3*) =
(*x1* ∗ (*x2* ∗ *x3* − *y2* ∗ *y3*) ∗ *delta-plus x2 y2 x3 y3* −
y1 ∗ (*x2* ∗ *y3* + *y2* ∗ *x3*) ∗ *delta-minus x2 y2 x3 y3*) ∗
(*x3* ∗ (*x1* ∗ *y1* + *x2* ∗ *y2*) ∗ *delta-x x1 y1 x2 y2* − (*x1* ∗ *y1* − *x2* ∗ *y2*) ∗ *y3* ∗ *delta-y x1 y1 x2 y2*)

apply((*subst x1′-expr*)+, (*subst y1′-expr*)+,(*subst x3′-expr*)+,(*subst y3′-expr*)+)
apply(*subst delta-x-def*)
apply(*subst* (*1*) *delta-minus-def*[*symmetric*])
apply(*subst* (*1*) *delta-plus-def*[*symmetric*])
apply(*subst* (*5*) *delta-x-def*[*symmetric*])
apply(*subst* (*3*) *delta-y-def*[*symmetric*])
unfolding *delta′-def delta-def*
by(*simp add*: *divide-simps assms c-eq-1*)

have *simp2gx*:
(*x1′* ∗ *y1′* − *x3* ∗ *y3*) ∗ *delta-minus x1 y1 x3′ y3′* ∗ (*delta′ x1 y1 x2 y2* ∗ *delta x2 y2 x3 y3*) =
((*x1* ∗ *y1* − *x2* ∗ *y2*) ∗ (*x1* ∗ *y1* + *x2* ∗ *y2*) −
x3 ∗ *y3* ∗ (*delta-x x1 y1 x2 y2* ∗ *delta-y x1 y1 x2 y2*)) ∗
(*delta-minus x2 y2 x3 y3* ∗ *delta-plus x2 y2 x3 y3* −
d ∗ *x1* ∗ *y1* ∗ (*x2* ∗ *x3* − *c* ∗ *y2* ∗ *y3*) ∗ (*x2* ∗ *y3* + *y2* ∗ *x3*))
apply((*subst x1′-expr*)+, (*subst y1′-expr*)+,(*subst x3′-expr*)+,(*subst y3′-expr*)+)
apply(*subst delta-minus-def*)
apply(*subst* (*2*) *delta-minus-def*[*symmetric*])
apply(*subst* (*1*) *delta-plus-def*[*symmetric*])
apply(*subst* (*2*) *delta-x-def*[*symmetric*])
apply(*subst* (*2*) *delta-y-def*[*symmetric*])
unfolding *delta′-def delta-def*
by(*simp add*: *divide-simps assms*)

have ∃ *r1 r2 r3*. g_x ∗ $Delta_x$ = *r1* ∗ *e1* + *r2* ∗ *e2* + *r3* ∗ *e3*
unfolding g_x-*def* $Delta_x$-*def*
apply(*simp add*: *assms*(*1*,*2*))
apply(*subst delta-minus-def*[*symmetric*])
apply(*subst* (*2*) *delta-x-def*[*symmetric*])
apply(*simp add*: *divide-simps assms*)
apply(*subst* (*3*) *left-diff-distrib*)
apply(*simp add*: *simp1gx simp2gx*)
unfolding *delta-x-def delta-y-def delta′-def delta-plus-def delta-minus-def delta-def e1-def e2-def e3-def e′-def*
by(*simp add*: *t-expr c-eq-1*,*algebra*)
then have g_x ∗ $Delta_x$ = *0* $Delta_x$ ≠ *0*

apply(*safe*)
using *e1-def e2-def e3-def assms*(*13*−*15*) **apply** *simp*
using $Delta_x$-*def delta-def delta'-def assms non-unfolded-adds* **by** *simp*
then have $g_x = 0$ **by** *auto*

have *simp1gy*:
(*x1' ∗ y1' + x3 ∗ y3*) ∗ *delta-plus x1 y1 x3' y3'* ∗ (*delta' x1 y1 x2 y2* ∗ *delta x2 y2 x3 y3*) =
((*x1 ∗ y1 − x2 ∗ y2*) ∗ (*x1 ∗ y1 + x2 ∗ y2*) +
x3 ∗ y3 ∗ (*delta-x x1 y1 x2 y2* ∗ *delta-y x1 y1 x2 y2*)) ∗
(*delta-minus x2 y2 x3 y3* ∗ *delta-plus x2 y2 x3 y3* +
d ∗ x1 ∗ y1 ∗ (*x2 ∗ x3 − c ∗ y2 ∗ y3*) ∗ (*x2 ∗ y3 + y2 ∗ x3*))
apply((*subst x1'-expr*)+, (*subst y1'-expr*)+,(*subst x3'-expr*)+,(*subst y3'-expr*)+)
apply(*subst delta-plus-def*)
apply(*subst* (*1*) *delta-minus-def*[*symmetric*])
apply(*subst* (*2*) *delta-plus-def*[*symmetric*])
apply(*subst* (*2*) *delta-x-def*[*symmetric*])
apply(*subst* (*2*) *delta-y-def*[*symmetric*])
unfolding *delta'-def delta-def*
by(*simp add*: *divide-simps assms*)

have *simp2gy*:
(*x1 ∗ y3' + y1 ∗ x3'*) ∗ *delta-y x1' y1' x3 y3* ∗ (*delta' x1 y1 x2 y2* ∗ *delta x2 y2 x3 y3*) =
(*x1* ∗ (*x2 ∗ y3 + y2 ∗ x3*) ∗ *delta-minus x2 y2 x3 y3* +
y1 ∗ (*x2 ∗ x3 − c ∗ y2 ∗ y3*) ∗ *delta-plus x2 y2 x3 y3*) ∗
((*x1 ∗ y1 − x2 ∗ y2*) ∗ *x3* ∗ *delta-y x1 y1 x2 y2* + (*x1 ∗ y1 + x2 ∗ y2*) ∗ *y3* ∗ *delta-x x1 y1 x2 y2*)
apply((*subst x1'-expr*)+, (*subst y1'-expr*)+,(*subst x3'-expr*)+,(*subst y3'-expr*)+)
apply(*subst delta-y-def*)
apply(*subst* (*1*) *delta-minus-def*[*symmetric*])
apply(*subst* (*1*) *delta-plus-def*[*symmetric*])
apply(*subst* (*3*) *delta-x-def*[*symmetric*])
apply(*subst* (*5*) *delta-y-def*[*symmetric*])
unfolding *delta'-def delta-def*
by(*simp add*: *divide-simps assms*)

have ∃ *r1 r2 r3*. $g_y * Delta_y = r1 * e1 + r2 * e2 + r3 * e3$
unfolding g_y-*def* $Delta_y$-*def*
apply(*simp add*: *assms*(*1*,*2*))
apply(*subst* (*2*) *delta-y-def*[*symmetric*])
apply(*subst* (*1*) *delta-plus-def*[*symmetric*])
apply(*simp add*: *divide-simps assms*)
apply(*subst left-diff-distrib*)
apply(*simp add*: *simp1gy simp2gy*)
unfolding *delta-x-def delta-y-def*
delta-def delta'-def
delta-minus-def delta-plus-def
e1-def e2-def e3-def e'-def

by(*simp add*: *c-eq-1 t-expr*,*algebra*)

then have g_y ∗ $Delta_y$ = *0* $Delta_y$ ≠ *0*
using *e1-def assms*(*13*−*15*) *e2-def e3-def* **apply** *simp*
using $Delta_y$*-def delta-def delta′-def assms non-unfolded-adds* **by** *simp*
then have g_y = *0* **by** *auto*

show *?thesis*
using ‹g_y = *0*› ‹g_x = *0*› **unfolding** g_x*-def* g_y*-def assms*(*3*,*4*) **by** (*simp add*: *prod-eq-iff*)
qed

lemma *ext-ext-add-add-assoc-points*:
assumes (*x1*,*y1*) ∈ *e′-aff* (*x2*,*y2*) ∈ *e′-aff* (*x3*,*y3*) ∈ *e′-aff*
assumes *delta′ x1 y1 x2 y2* ≠ *0 delta x2 y2 x3 y3* ≠ *0*
delta′ (*fst* (*ext-add* (*x1*,*y1*) (*x2*,*y2*))) (*snd* (*ext-add* (*x1*,*y1*) (*x2*,*y2*))) *x3 y3* ≠ *0*
delta x1 y1 (*fst* (*add* (*x2*,*y2*) (*x3*,*y3*))) (*snd* (*add* (*x2*,*y2*) (*x3*,*y3*))) ≠ *0*
shows *ext-add* (*ext-add* (*x1*,*y1*) (*x2*,*y2*)) (*x3*,*y3*) = *add* (*x1*,*y1*) (*add* (*x2*,*y2*) (*x3*,*y3*))
using *assms*
unfolding *e′-aff-def delta-def delta′-def*
apply(*simp del*: *ext-add.simps add.simps*)
using *ext-ext-add-add-assoc*
apply(*safe*)
using *prod.collapse* **by** *blast*

lemma *ext-ext-add-ext-assoc*:
assumes *z1′* = (*x1′*,*y1′*) *z3′* = (*x3′*,*y3′*)
assumes *z1′* = *ext-add* (*x1*,*y1*) (*x2*,*y2*) *z3′* = *ext-add* (*x2*,*y2*) (*x3*,*y3*)
assumes *delta-x x1 y1 x2 y2* ≠ *0 delta-y x1 y1 x2 y2* ≠ *0*
delta-x x1′ y1′ x3 y3 ≠ *0 delta-y x1′ y1′ x3 y3* ≠ *0*
delta-minus x1 y1 x3′ y3′ ≠ *0 delta-plus x1 y1 x3′ y3′* ≠ *0*
delta-x x2 y2 x3 y3 ≠ *0 delta-y x2 y2 x3 y3* ≠ *0*
assumes *e′ x1 y1* = *0 e′ x2 y2* = *0 e′ x3 y3* = *0*
shows *ext-add* (*ext-add* (*x1*,*y1*) (*x2*,*y2*)) (*x3*,*y3*) = *add* (*x1*,*y1*) (*ext-add* (*x2*,*y2*) (*x3*,*y3*))
proof −
define *e1* **where** *e1* = *e′ x1 y1*
define *e2* **where** *e2* = *e′ x2 y2*
define *e3* **where** *e3* = *e′ x3 y3*
define $Delta_x$ **where** $Delta_x$ =
(*delta-x x1′ y1′ x3 y3*)∗(*delta-minus x1 y1 x3′ y3′*)∗
(*delta′ x1 y1 x2 y2*)∗(*delta′ x2 y2 x3 y3*)
define $Delta_y$ **where** $Delta_y$ =
(*delta-y x1′ y1′ x3 y3*)∗(*delta-plus x1 y1 x3′ y3′*)∗
(*delta′ x1 y1 x2 y2*)∗(*delta′ x2 y2 x3 y3*)
define g_x **where** g_x = *fst*(*ext-add z1′* (*x3*,*y3*)) − *fst*(*add* (*x1*,*y1*) *z3′*)
define g_y **where** g_y = *snd*(*ext-add z1′* (*x3*,*y3*)) − *snd* (*add* (*x1*,*y1*) *z3′*)

have *x1'-expr*: $x1' = (x1 * y1 - x2 * y2) / (x2 * y1 - x1 * y2)$ **using** *assms*(*1*,*3*) **by** *simp*
have *y1'-expr*: $y1' = (x1 * y1 + x2 * y2) / (x1 * x2 + y1 * y2)$ **using** *assms*(*1*,*3*) **by** *simp*
have *x3'-expr*: $x3' = (x2 * y2 - x3 * y3) / (x3 * y2 - x2 * y3)$ **using** *assms*(*2*,*4*) **by** *simp*
have *y3'-expr*: $y3' = (x2 * y2 + x3 * y3) / (x2 * x3 + y2 * y3)$ **using** *assms*(*2*,*4*) **by** *simp*

have *non-unfolded-adds*:
delta' x1 y1 x2 y2 $\neq$ *0* **using** *delta'-def assms*(*5*,*6*) **by** *auto*

have *simp1gx*:
(*x1'* $*$ *y1'* $-$ *x3* $*$ *y3*) $*$ *delta-minus x1 y1 x3' y3'* $*$ (*delta' x1 y1 x2 y2* $*$ *delta' x2 y2 x3 y3*) =
((*x1* $*$ *y1* $-$ *x2* $*$ *y2*) $*$ (*x1* $*$ *y1* + *x2* $*$ *y2*) $-$
x3 $*$ *y3* $*$ (*delta-x x1 y1 x2 y2* $*$ *delta-y x1 y1 x2 y2*)) $*$
(*delta-x x2 y2 x3 y3* $*$ *delta-y x2 y2 x3 y3* $-$
d $*$ *x1* $*$ *y1* $*$ (*x2* $*$ *y2* $-$ *x3* $*$ *y3*) $*$ (*x2* $*$ *y2* + *x3* $*$ *y3*))

apply((*subst x1'-expr*)+, (*subst y1'-expr*)+,(*subst x3'-expr*)+,(*subst y3'-expr*)+)
apply(*subst delta-minus-def*)
apply(*subst* (*2 5*) *delta-x-def*[*symmetric*])
apply(*subst* (*2 4*) *delta-y-def*[*symmetric*])
unfolding *delta'-def delta-def*
by(*simp add*: *divide-simps assms c-eq-1*)

have *simp2gx*:
(*x1* $*$ *x3'* $-$ *c* $*$ *y1* $*$ *y3'*) $*$ *delta-x x1' y1' x3 y3* $*$ (*delta' x1 y1 x2 y2* $*$ *delta' x2 y2 x3 y3*) =
(*x1* $*$ (*x2* $*$ *y2* $-$ *x3* $*$ *y3*) $*$ *delta-y x2 y2 x3 y3* $-$
c $*$ *y1* $*$ (*x2* $*$ *y2* + *x3* $*$ *y3*) $*$ *delta-x x2 y2 x3 y3*) $*$
(*x3* $*$ (*x1* $*$ *y1* + *x2* $*$ *y2*) $*$ *delta-x x1 y1 x2 y2* $-$ (*x1* $*$ *y1* $-$ *x2* $*$ *y2*) $*$ *y3* $*$ *delta-y x1 y1 x2 y2*)
apply((*subst x1'-expr*)+, (*subst y1'-expr*)+,(*subst x3'-expr*)+,(*subst y3'-expr*)+)
apply(*subst delta-x-def*)
apply(*subst* (*2 6*) *delta-x-def*[*symmetric*])
apply(*subst* (*2 4*) *delta-y-def*[*symmetric*])
unfolding *delta'-def delta-def*
by(*simp add*: *divide-simps assms*)

have $\exists$ *r1 r2 r3*. $g_x * Delta_x = r1 * e1 + r2 * e2 + r3 * e3$
unfolding g_x*-def* $Delta_x$*-def*
apply(*simp add*: *assms*(*1*,*2*))
apply(*subst delta-minus-def*[*symmetric*])
apply(*subst* (*2*) *delta-x-def*[*symmetric*])
apply(*simp add*: *divide-simps assms*)
apply(*subst* (*3*) *left-diff-distrib*)

apply(*simp add*: *simp1gx simp2gx*)
unfolding *delta-x-def delta-y-def delta'-def delta-plus-def delta-minus-def delta-def e1-def e2-def e3-def e'-def*
by(*simp add*: *t-expr c-eq-1*,*algebra*)
then have $g_x * Delta_x = 0$ $Delta_x \neq 0$
apply(*safe*)
using *e1-def e2-def e3-def assms*(*13−15*) **apply** *simp*
using $Delta_x$*-def delta-def delta'-def assms non-unfolded-adds* **by** *simp*
then have $g_x = 0$ **by** *auto*

have *simp1gy*:
$(x1' * y1' + x3 * y3) * delta\text{-}plus\ x1\ y1\ x3'\ y3' * (delta'\ x1\ y1\ x2\ y2 * delta'\ x2\ y2\ x3\ y3) =$
$((x1 * y1 - x2 * y2) * (x1 * y1 + x2 * y2) +$
$x3 * y3 * (delta\text{-}x\ x1\ y1\ x2\ y2 * delta\text{-}y\ x1\ y1\ x2\ y2)) *$
$(delta\text{-}x\ x2\ y2\ x3\ y3 * delta\text{-}y\ x2\ y2\ x3\ y3 +$
$d * x1 * y1 * (x2 * y2 - x3 * y3) * (x2 * y2 + x3 * y3))$
apply((*subst x1'-expr*)+, (*subst y1'-expr*)+,(*subst x3'-expr*)+,(*subst y3'-expr*)+)
apply(*subst delta-plus-def*)
apply(*subst* (*2 4*) *delta-x-def*[*symmetric*])
apply(*subst* (*2 5*) *delta-y-def*[*symmetric*])
unfolding *delta'-def delta-def*
by(*simp add*: *divide-simps assms*)

have *simp2gy*:
$(x1 * y3' + y1 * x3') * delta\text{-}y\ x1'\ y1'\ x3\ y3 * (delta'\ x1\ y1\ x2\ y2 * delta'\ x2\ y2\ x3\ y3) =$
$(x1 * (x2 * y2 + x3 * y3) * delta\text{-}x\ x2\ y2\ x3\ y3 + y1 * (x2 * y2 - x3 * y3) * delta\text{-}y\ x2\ y2\ x3\ y3) *$
$((x1 * y1 - x2 * y2) * x3 * delta\text{-}y\ x1\ y1\ x2\ y2 + (x1 * y1 + x2 * y2) * y3 * delta\text{-}x\ x1\ y1\ x2\ y2)$
apply((*subst x1'-expr*)+, (*subst y1'-expr*)+,(*subst x3'-expr*)+,(*subst y3'-expr*)+)
apply(*subst delta-y-def*)
apply(*subst* (*2 4*) *delta-x-def*[*symmetric*])
apply(*subst* (*2 6*) *delta-y-def*[*symmetric*])
unfolding *delta'-def delta-def*
by(*simp add*: *divide-simps assms*)

have $\exists\ r1\ r2\ r3.\ g_y * Delta_y = r1 * e1 + r2 * e2 + r3 * e3$
unfolding g_y*-def* $Delta_y$*-def*
apply(*simp add*: *assms*(*1*,*2*))
apply(*subst* (*2*) *delta-y-def*[*symmetric*])
apply(*subst* (*1*) *delta-plus-def*[*symmetric*])
apply(*simp add*: *divide-simps assms*)
apply(*subst left-diff-distrib*)
apply(*simp add*: *simp1gy simp2gy*)
unfolding *delta-x-def delta-y-def*
delta-def delta'-def
delta-minus-def delta-plus-def

e1-def e2-def e3-def e′-def
by(*simp add*: *c-eq-1 t-expr*,*algebra*)

then have $g_y * Delta_y = 0$ $Delta_y \neq 0$
using *e1-def assms*(*13*−*15*) *e2-def e3-def* **apply** *simp*
using $Delta_y$*-def delta-def delta′-def assms non-unfolded-adds* **by** *simp*
then have $g_y = 0$ **by** *auto*

show *?thesis*
using ‹$g_y = 0$› ‹$g_x = 0$› **unfolding** g_x*-def* g_y*-def assms*(*3*,*4*) **by** (*simp add*: *prod-eq-iff*)
qed

lemma *ext-ext-add-ext-assoc-points*:
assumes (*x1*,*y1*) ∈ *e′-aff* (*x2*,*y2*) ∈ *e′-aff* (*x3*,*y3*) ∈ *e′-aff*
assumes *delta′ x1 y1 x2 y2* ≠ *0* *delta′ x2 y2 x3 y3* ≠ *0*
delta′ (*fst* (*ext-add* (*x1*,*y1*) (*x2*,*y2*))) (*snd* (*ext-add* (*x1*,*y1*) (*x2*,*y2*))) *x3 y3* ≠ *0*
delta x1 y1 (*fst* (*ext-add* (*x2*,*y2*) (*x3*,*y3*))) (*snd* (*ext-add* (*x2*,*y2*) (*x3*,*y3*))) ≠ *0*
shows *ext-add* (*ext-add* (*x1*,*y1*) (*x2*,*y2*)) (*x3*,*y3*) = *add* (*x1*,*y1*) (*ext-add* (*x2*,*y2*) (*x3*,*y3*))
using *assms*
unfolding *e′-aff-def delta-def delta′-def*
apply(*simp del*: *ext-add.simps add.simps*)
using *ext-ext-add-ext-assoc*
apply(*safe*)
using *prod.collapse* **by** *blast*

lemma *add-ext-ext-add-assoc*:
assumes *z1′* = (*x1′*,*y1′*) *z3′* = (*x3′*,*y3′*)
assumes *z1′* = *ext-add* (*x1*,*y1*) (*x2*,*y2*) *z3′* = *add* (*x2*,*y2*) (*x3*,*y3*)
assumes *delta-x x1 y1 x2 y2* ≠ *0* *delta-y x1 y1 x2 y2* ≠ *0*
delta-plus x2 y2 x3 y3 ≠ *0* *delta-minus x2 y2 x3 y3* ≠ *0*
delta-plus x1′ y1′ x3 y3 ≠ *0* *delta-minus x1′ y1′ x3 y3* ≠ *0*
delta-x x1 y1 x3′ y3′ ≠ *0* *delta-y x1 y1 x3′ y3′* ≠ *0*
assumes *e′ x1 y1* = *0* *e′ x2 y2* = *0* *e′ x3 y3* = *0*
shows *add* (*ext-add* (*x1*,*y1*) (*x2*,*y2*)) (*x3*,*y3*) = *ext-add* (*x1*,*y1*) (*add* (*x2*,*y2*) (*x3*,*y3*))
proof −
define *e1* **where** *e1* = *e′ x1 y1*
define *e2* **where** *e2* = *e′ x2 y2*
define *e3* **where** *e3* = *e′ x3 y3*
define $Delta_x$ **where** $Delta_x$ =
(*delta-minus x1′ y1′ x3 y3*)∗(*delta-x x1 y1 x3′ y3′*)∗
(*delta′ x1 y1 x2 y2*)∗(*delta x2 y2 x3 y3*)
define $Delta_y$ **where** $Delta_y$ =
(*delta-plus x1′ y1′ x3 y3*)∗(*delta-y x1 y1 x3′ y3′*)∗
(*delta′ x1 y1 x2 y2*)∗(*delta x2 y2 x3 y3*)

define g_x **where** $g_x = fst(add\ z1'\ (x3,y3)) - fst(ext\text{-}add\ (x1,y1)\ z3')$
define g_y **where** $g_y = snd(add\ z1'\ (x3,y3)) - snd(ext\text{-}add\ (x1,y1)\ z3')$

have *x1'-expr*: $x1' = (x1 * y1 - x2 * y2) / (x2 * y1 - x1 * y2)$ **using** *assms*(*1*,*3*) **by** *simp*
have *y1'-expr*: $y1' = (x1 * y1 + x2 * y2) / (x1 * x2 + y1 * y2)$ **using** *assms*(*1*,*3*) **by** *simp*
have *x3'-expr*: $x3' = (x2 * x3 - c * y2 * y3) / (1 - d * x2 * y2 * x3 * y3)$ **using** *assms*(*2*,*4*) **by** *simp*
have *y3'-expr*: $y3' = (x2 * y3 + y2 * x3) / (1 + d * x2 * y2 * x3 * y3)$ **using** *assms*(*2*,*4*) **by** *simp*

have *non-unfolded-adds*:
delta' x1 y1 x2 y2 $\neq$ *0* **using** *delta'-def assms*(*5*,*6*) **by** *auto*

have *simp1gx*:
($x1' * x3 - c * y1' * y3$) $*$ *delta-x x1 y1 x3' y3'* $*$ (*delta' x1 y1 x2 y2* $*$ *delta x2 y2 x3 y3*) $=$
(($x1 * y1 - x2 * y2$) $* x3 *$ *delta-y x1 y1 x2 y2* $-$ ($x1 * y1 + x2 * y2$) $* y3 *$ *delta-x x1 y1 x2 y2*) $*$
(($x2 * x3 - y2 * y3$) $* y1 *$ *delta-plus x2 y2 x3 y3* $-$
$x1 *$ ($x2 * y3 + y2 * x3$) $*$ *delta-minus x2 y2 x3 y3*)

apply((*subst x1'-expr*)+, (*subst y1'-expr*)+,(*subst x3'-expr*)+,(*subst y3'-expr*)+)
apply(*subst delta-x-def*)
apply(*subst* (*2*) *delta-x-def*[*symmetric*])
apply(*subst* (*2*) *delta-y-def*[*symmetric*])
apply(*subst* (*1*) *delta-minus-def*[*symmetric*])
apply(*subst* (*1*) *delta-plus-def*[*symmetric*])
unfolding *delta'-def delta-def*
by(*simp add*: *divide-simps assms*(*5*−*8*) *c-eq-1*)

have *simp2gx*:
($x1 * y1 - x3' * y3'$) $*$ *delta-minus x1' y1' x3 y3* $*$ (*delta' x1 y1 x2 y2* $*$ *delta x2 y2 x3 y3*) $=$
($x1 * y1 *$ (*delta-minus x2 y2 x3 y3* $*$ *delta-plus x2 y2 x3 y3*) $-$
($x2 * x3 - c * y2 * y3$) $*$ ($x2 * y3 + y2 * x3$)) $*$
(*delta-x x1 y1 x2 y2* $*$ *delta-y x1 y1 x2 y2* $-$
$d *$ ($x1 * y1 - x2 * y2$) $*$ ($x1 * y1 + x2 * y2$) $* x3 * y3$)
apply((*subst x1'-expr*)+, (*subst y1'-expr*)+,(*subst x3'-expr*)+,(*subst y3'-expr*)+)
apply(*subst delta-minus-def*)
apply(*subst* (*4*) *delta-x-def*[*symmetric*])
apply(*subst* (*3*) *delta-y-def*[*symmetric*])
apply(*subst* (*1*) *delta-minus-def*[*symmetric*])
apply(*subst* (*1*) *delta-plus-def*[*symmetric*])
unfolding *delta'-def delta-def*
by(*simp add*: *divide-simps assms*(*5*−*8*))

have $\exists$ *r1 r2 r3*. $g_x * Delta_x = r1 * e1 + r2 * e2 + r3 * e3$

unfolding g_x-def $Delta_x$-def
apply(simp add: assms(1,2))
apply(subst (1) delta-minus-def[symmetric])
apply(subst (3) delta-x-def[symmetric])
apply(simp add: divide-simps assms)
apply(subst (3) left-diff-distrib)
apply(simp add: simp1gx simp2gx)
unfolding delta-x-def delta-y-def delta'-def delta-plus-def delta-minus-def delta-def e1-def e2-def e3-def e'-def
by(simp add: t-expr c-eq-1,algebra)
then have $g_x * Delta_x = 0$ $Delta_x \neq 0$
apply(safe)
using e1-def e2-def e3-def assms(13−15) **apply** force
using $Delta_x$-def delta'-def delta-def assms non-unfolded-adds **by** force
then have $g_x = 0$ **by** auto

have simp1gy:
(x1' * y3 + y1' * x3) * delta-y x1 y1 x3' y3' * (delta' x1 y1 x2 y2 * delta x2 y2 x3 y3) =
((x1 * y1 − x2 * y2) * y3 * delta-y x1 y1 x2 y2 + (x1 * y1 + x2 * y2) * x3 * delta-x x1 y1 x2 y2) *
(x1 * (x2 * x3 − c * y2 * y3) * delta-plus x2 y2 x3 y3 +
y1 * (x2 * y3 + y2 * x3) * delta-minus x2 y2 x3 y3)
apply((subst x1'-expr)+, (subst y1'-expr)+,(subst x3'-expr)+,(subst y3'-expr)+)
apply(subst delta-y-def)
apply(subst (2) delta-x-def[symmetric])
apply(subst (3) delta-y-def[symmetric])
apply(subst (1) delta-plus-def[symmetric])
apply(subst (1) delta-minus-def[symmetric])
unfolding delta'-def delta-def
by(simp add: divide-simps assms(5−8))

have simp2gy:
(x1 * y1 + x3' * y3') * delta-plus x1' y1' x3 y3 * (delta' x1 y1 x2 y2 * delta x2 y2 x3 y3) =
(x1 * y1 * (delta-minus x2 y2 x3 y3 * delta-plus x2 y2 x3 y3) +
(x2 * x3 − c * y2 * y3) * (x2 * y3 + y2 * x3)) *
(delta-x x1 y1 x2 y2 * delta-y x1 y1 x2 y2 +
d * (x1 * y1 − x2 * y2) * (x1 * y1 + x2 * y2) * x3 * y3)
apply((subst x1'-expr)+, (subst y1'-expr)+,(subst x3'-expr)+,(subst y3'-expr)+)
apply(subst delta-plus-def)
apply(subst (3) delta-x-def[symmetric])
apply(subst (4) delta-y-def[symmetric])
apply(subst (1) delta-plus-def[symmetric])
apply(subst (1) delta-minus-def[symmetric])
unfolding delta'-def delta-def
by(simp add: divide-simps assms(5−8))
have $\exists$ r1 r2 r3. $g_y * Delta_y = r1 * e1 + r2 * e2 + r3 * e3$
unfolding g_y-def $Delta_y$-def

apply(*simp add*: *assms*(*1*,*2*))
apply(*subst delta-plus-def*[*symmetric*])
apply(*subst* (*3*) *delta-y-def*[*symmetric*])
apply(*simp add*: *divide-simps assms*)
apply(*subst left-diff-distrib*)
apply(*simp add*: *simp1gy simp2gy*)
unfolding *delta-x-def delta-y-def delta-minus-def delta-plus-def*
e1-def e2-def e3-def e′-def
by(*simp add*: *c-eq-1 t-expr*,*algebra*)

then have $g_y * Delta_y = 0$ $Delta_y \neq 0$
using *e1-def assms*(*13*−*15*) *e2-def e3-def* **apply** *force*
using $Delta_y$*-def delta′-def delta-def assms*(*7*−*12*) *non-unfolded-adds* **by** *auto*
then have $g_y = 0$ **by** *auto*

show *?thesis*
using ‹$g_y = 0$› ‹$g_x = 0$› **unfolding** g_x*-def* g_y*-def assms*(*3*,*4*) **by** (*simp add*: *prod-eq-iff*)
qed

4.3 Some relations between deltas

lemma *mix-tau*:
assumes (*x1*,*y1*) ∈ *e′-aff* (*x2*,*y2*) ∈ *e′-aff* *x2* ≠ *0* *y2* ≠ *0*
assumes *delta′ x1 y1 x2 y2* ≠ *0* *delta′ x1 y1* (*fst* (τ (*x2*,*y2*))) (*snd* (τ (*x2*,*y2*))) ≠ *0*
shows *delta x1 y1 x2 y2* ≠ *0*
using *assms*
unfolding *e′-aff-def e′-def delta-def delta-plus-def delta-minus-def delta′-def delta-y-def delta-x-def*
apply(*simp*)
apply(*simp add*: *t-nz algebra-simps*)
apply(*simp add*: *power2-eq-square*[*symmetric*] *t-expr d-nz*)
apply(*simp add*: *divide-simps t-nz*)
by *algebra*

lemma *mix-tau-0*:
assumes (*x1*,*y1*) ∈ *e′-aff* (*x2*,*y2*) ∈ *e′-aff* *x2* ≠ *0* *y2* ≠ *0*
assumes *delta x1 y1 x2 y2* = *0*
shows *delta′ x1 y1 x2 y2* = *0* ∨ *delta′ x1 y1* (*fst* (τ (*x2*,*y2*))) (*snd* (τ (*x2*,*y2*))) = *0*
using *assms*
unfolding *e′-aff-def e′-def delta-def delta-plus-def delta-minus-def delta′-def delta-y-def delta-x-def*
apply(*simp*)
apply(*simp add*: *t-nz algebra-simps*)
apply(*simp add*: *power2-eq-square*[*symmetric*] *t-expr d-nz*)
apply(*simp add*: *divide-simps t-nz*)
by *algebra*

lemma *mix-tau-prime*:
  assumes $(x1,y1) \in e'$-*aff* $(x2,y2) \in e'$-*aff* $x2 \neq 0$ $y2 \neq 0$
  assumes *delta x1 y1 x2 y2* $\neq 0$ *delta x1 y1* (*fst* (τ (*x2*,*y2*))) (*snd* (τ (*x2*,*y2*)))
$\neq 0$
  shows *delta' x1 y1 x2 y2* $\neq 0$
  using *assms*
 unfolding *e'-aff-def e'-def delta-def delta-plus-def delta-minus-def delta'-def delta-y-def delta-x-def*
  apply(*simp*)
  apply(*simp add*: *t-nz algebra-simps*)
  apply(*simp add*: *power2-eq-square*[*symmetric*] *t-expr d-nz*)
  apply(*simp add*: *divide-simps*)
  by *algebra*

lemma *tau-tau-d*:
  assumes $(x1,y1) \in e'$-*aff* $(x2,y2) \in e'$-*aff* $x2 \neq 0$ $y2 \neq 0$
   assumes *delta* (*fst* (τ (*x1*,*y1*))) (*snd* (τ (*x1*,*y1*))) (*fst* (τ (*x2*,*y2*))) (*snd* (τ (*x2*,*y2*))) $\neq 0$
  shows *delta x1 y1 x2 y2* $\neq 0$
  using *assms*
 unfolding *e'-aff-def e'-def delta-def delta-plus-def delta-minus-def delta'-def delta-y-def delta-x-def*
  apply(*simp*)
  apply(*simp add*: *t-expr*)
  apply(*simp split*: *if-splits add*: *divide-simps t-nz*)
  apply(*simp-all add*: *t-nz algebra-simps power2-eq-square*[*symmetric*] *t-expr d-nz*)
  apply *algebra*
  by *algebra*

lemma *tau-tau-d'*:
  assumes $(x1,y1) \in e'$-*aff* $(x2,y2) \in e'$-*aff* $x2 \neq 0$ $y2 \neq 0$
   assumes *delta'* (*fst* (τ (*x1*,*y1*))) (*snd* (τ (*x1*,*y1*))) (*fst* (τ (*x2*,*y2*))) (*snd* (τ (*x2*,*y2*))) $\neq 0$
  shows *delta' x1 y1 x2 y2* $\neq 0$
  using *assms*
 unfolding *e'-aff-def e'-def delta-def delta-plus-def delta-minus-def delta'-def delta-y-def delta-x-def*
  apply(*simp*)
  apply(*simp add*: *t-expr*)
  apply(*simp split*: *if-splits add*: *divide-simps t-nz*)
  by *algebra*

lemma *zero-coord-expr*:
  assumes $(x,y) \in e'$-*aff* $x = 0 \vee y = 0$
  shows $\exists$ $r \in$ *rotations*. $(x,y) = r\ (1,0)$
proof –

consider (*1*) $x = 0$ | (*2*) $y = 0$ **using** *assms* **by** *blast*
then show *?thesis*
proof(*cases*)
case *1*
then have *y-expr*: $y = 1 \vee y = -1$
using *assms* **unfolding** *e'-aff-def e'-def* **by**(*simp*,*algebra*)
then show *?thesis*
using *1* **unfolding** *rotations-def* **by** *auto*
next
case *2*
then have *x-expr*: $x = 1 \vee x = -1$
using *assms* **unfolding** *e'-aff-def e'-def* **by**(*simp*,*algebra*)
then show *?thesis*
using *2* **unfolding** *rotations-def* **by** *auto*
qed
qed

lemma *delta-add-delta'-1*:
assumes *1*: $x1 \neq 0$ $y1 \neq 0$ $x2 \neq 0$ $y2 \neq 0$
assumes *r-expr*: *rx = fst (add (x1,y1) (x2,y2)) ry = snd (add (x1,y1) (x2,y2))*

assumes *in-aff*: *(x1,y1) ∈ e'-aff (x2,y2) ∈ e'-aff*
assumes *pd*: *delta x1 y1 x2 y2* $\neq 0$
assumes *pd'*: *delta rx ry (fst (τ (i (x2,y2)))) (snd (τ (i (x2,y2))))* $\neq 0$
shows *delta' rx ry (fst (i (x2,y2))) (snd (i (x2,y2)))* $\neq 0$
using *pd'* **unfolding** *delta-def delta-minus-def delta-plus-def*
delta'-def delta-x-def delta-y-def
apply(*simp split*: *if-splits add*: *divide-simps t-nz 1 algebra-simps power2-eq-square*[*symmetric*] *t-expr d-nz*)
using *pd in-aff* **unfolding** *r-expr delta-def delta-minus-def delta-plus-def*
e'-aff-def e'-def
apply(*simp add*: *divide-simps t-expr*)
apply(*simp add*: *c-eq-1 algebra-simps*)
by *algebra*

lemma *delta'-add-delta-1*:
assumes *1*: $x1 \neq 0$ $y1 \neq 0$ $x2 \neq 0$ $y2 \neq 0$
assumes *r-expr*: *rx = fst (ext-add (x1,y1) (x2,y2)) ry = snd (ext-add (x1,y1) (x2,y2))*
assumes *in-aff*: *(x1,y1) ∈ e'-aff (x2,y2) ∈ e'-aff*
assumes *pd'*: *delta' rx ry (fst (τ (i (x2,y2)))) (snd (τ (i (x2,y2))))* $\neq 0$
shows *delta rx ry (fst (i (x2,y2))) (snd (i (x2,y2)))* $\neq 0$
using *pd'* **unfolding** *delta-def delta-minus-def delta-plus-def*
delta'-def delta-x-def delta-y-def
apply(*simp split*: *if-splits add*: *divide-simps t-nz 1 algebra-simps power2-eq-square*[*symmetric*] *t-expr d-nz*)
using *in-aff* **unfolding** *r-expr delta-def delta-minus-def delta-plus-def*
e'-aff-def e'-def
apply(*simp split*: *if-splits add*: *divide-simps t-expr*)

apply(*simp add*: *c-eq-1 algebra-simps*)
by *algebra*

lemma *add-self*:
assumes *in-aff*: $(x2,y2) \in e'$*-aff*
shows $delta\ x2\ y2\ x2\ (-y2) \neq 0 \lor delta'\ x2\ y2\ x2\ (-y2) \neq 0$
using *in-aff d-n1*
unfolding *delta-def delta-plus-def delta-minus-def*
delta′-def delta-x-def delta-y-def
e′-aff-def e′-def
apply(*simp add*: *t-expr two-not-zero*)
apply(*safe*)
apply(*simp-all add*: *algebra-simps*)
by(*simp add*: *semiring-normalization-rules*(*18*) *semiring-normalization-rules*(*29*) *two-not-zero*)+

lemma *not-add-self*:
assumes *in-aff*: $(x2,y2) \in e'$*-aff* $x2 \neq 0$ $y2 \neq 0$
shows $delta\ x2\ y2\ (fst\ (\tau\ (i\ (x2,y2))))\ (snd\ (\tau\ (i\ (x2,y2)))) = 0$
$delta'\ x2\ y2\ (fst\ (\tau\ (i\ (x2,y2))))\ (snd\ (\tau\ (i\ (x2,y2)))) = 0$
using *in-aff d-n1*
unfolding *delta-def delta-plus-def delta-minus-def*
delta′-def delta-x-def delta-y-def
e′-aff-def e′-def
apply(*simp add*: *t-expr two-not-zero*)
apply(*safe*)
by(*simp-all add*: *algebra-simps t-nz power2-eq-square*[*symmetric*] *t-expr*)

lemma *funny-field-lemma-1*:
$((x1 * x2 - y1 * y2) * ((x1 * x2 - y1 * y2) * (x2 * (y2 * (1 + d * x1 * y1 * x2 * y2)))) +$
$(x1 * x2 - y1 * y2) * ((x1 * y2 + y1 * x2) * y2^2) * (1 - d * x1 * y1 * x2 * y2)) *$
$(1 + d * x1 * y1 * x2 * y2) \neq$
$((x1 * y2 + y1 * x2) * ((x1 * y2 + y1 * x2) * (x2 * (y2 * (1 - d * x1 * y1 * x2 * y2)))) +$
$(x1 * x2 - y1 * y2) * ((x1 * y2 + y1 * x2) * x2^2) * (1 + d * x1 * y1 * x2 * y2)) *$
$(1 - d * x1 * y1 * x2 * y2) \Longrightarrow$
$(d * ((x1 * x2 - y1 * y2) * ((x1 * y2 + y1 * x2) * (x2 * y2))))^2 =$
$((1 - d * x1 * y1 * x2 * y2) * (1 + d * x1 * y1 * x2 * y2))^2 \Longrightarrow$
$x1^2 + y1^2 - 1 = d * x1^2 * y1^2 \Longrightarrow$
$x2^2 + y2^2 - 1 = d * x2^2 * y2^2 \Longrightarrow False$
by *algebra*

lemma *delta-add-delta′-2*:
assumes *1*: $x1 \neq 0$ $y1 \neq 0$ $x2 \neq 0$ $y2 \neq 0$
assumes *r-expr*: $rx = fst\ (add\ (x1,y1)\ (x2,y2))$ $ry = snd\ (add\ (x1,y1)\ (x2,y2))$

assumes *in-aff*: $(x1,y1) \in e'$-*aff* $(x2,y2) \in e'$-*aff*
assumes *pd*: *delta x1 y1 x2 y2* $\neq 0$
assumes *pd'*: *delta' rx ry* (*fst* (τ (*i* (*x2*,*y2*)))) (*snd* (τ (*i* (*x2*,*y2*)))) $\neq 0$
shows *delta rx ry* (*fst* (*i* (*x2*,*y2*))) (*snd* (*i* (*x2*,*y2*))) $\neq 0$
using *pd'* **unfolding** *delta-def delta-minus-def delta-plus-def*
delta'-def delta-x-def delta-y-def
apply(*simp split*: *if-splits add*: *divide-simps t-nz 1 algebra-simps power2-eq-square*[*symmetric*] *t-expr d-nz*)
apply *safe*
using *pd* **unfolding** *r-expr delta-def delta-minus-def delta-plus-def*
apply(*simp*)
apply(*simp add*: *c-eq-1 divide-simps*)
using *in-aff* **unfolding** *e'-aff-def e'-def*
apply(*simp add*: *t-expr*)
apply *safe*
using *funny-field-lemma-1* **by** *blast*

lemma *funny-field-lemma-2*: $(x2 * y2)^2 * ((x2 * y1 - x1 * y2) * (x1 * x2 + y1 * y2))^2 \neq ((x1 * y1 - x2 * y2) * (x1 * y1 + x2 * y2))^2 \Longrightarrow$
$((x1 * y1 - x2 * y2) * ((x1 * y1 - x2 * y2) * (x2 * (y2 * (x1 * x2 + y1 * y2)))) +$
$(x1 * y1 - x2 * y2) * ((x1 * y1 + x2 * y2) * x2^2) * (x2 * y1 - x1 * y2)) *$
$(x1 * x2 + y1 * y2) =$
$((x1 * y1 + x2 * y2) * ((x1 * y1 + x2 * y2) * (x2 * (y2 * (x2 * y1 - x1 * y2)))) +$
$(x1 * y1 - x2 * y2) * ((x1 * y1 + x2 * y2) * y2^2) * (x1 * x2 + y1 * y2)) *$
$(x2 * y1 - x1 * y2) \Longrightarrow$
$x1^2 + y1^2 - 1 = d * x1^2 * y1^2 \Longrightarrow$
$x2^2 + y2^2 - 1 = d * x2^2 * y2^2 \Longrightarrow$ *False*
by *algebra*

lemma *delta'-add-delta-2*:
assumes *1*: $x1 \neq 0$ $y1 \neq 0$ $x2 \neq 0$ $y2 \neq 0$
assumes *r-expr*: *rx* = *fst* (*ext-add* (*x1*,*y1*) (*x2*,*y2*)) *ry* = *snd* (*ext-add* (*x1*,*y1*) (*x2*,*y2*))
assumes *in-aff*: $(x1,y1) \in e'$-*aff* $(x2,y2) \in e'$-*aff*
assumes *pd*: *delta' x1 y1 x2 y2* $\neq 0$
assumes *pd'*: *delta rx ry* (*fst* (τ (*i* (*x2*,*y2*)))) (*snd* (τ (*i* (*x2*,*y2*)))) $\neq 0$
shows *delta' rx ry* (*fst* (*i* (*x2*,*y2*))) (*snd* (*i* (*x2*,*y2*))) $\neq 0$
using *pd'* **unfolding** *delta-def delta-minus-def delta-plus-def*
delta'-def delta-x-def delta-y-def
apply(*simp split*: *if-splits add*: *divide-simps t-nz 1 algebra-simps power2-eq-square*[*symmetric*] *t-expr d-nz*)
apply *safe*
using *pd* **unfolding** *r-expr delta'-def delta-x-def delta-y-def*
apply(*simp*)
apply(*simp split*: *if-splits add*: *c-eq-1 divide-simps*)
using *in-aff* **unfolding** *e'-aff-def e'-def*

apply(*simp add*: *t-expr*)
apply *safe*
using *funny-field-lemma-2* **by** *fast*

lemma *delta′-add-delta-not-add*:
assumes *1*: $x1 \neq 0$ $y1 \neq 0$ $x2 \neq 0$ $y2 \neq 0$
assumes *in-aff*: $(x1,y1) \in e'$-*aff* $(x2,y2) \in e'$-*aff*
assumes *pd*: *delta′ x1 y1 x2 y2* $\neq 0$
assumes *add-nz*: *fst* (*ext-add* $(x1,y1)$ $(x2,y2)$) $\neq 0$ *snd* (*ext-add* $(x1,y1)$ $(x2,y2)$) $\neq 0$
shows *pd′*: *delta* (*fst* (τ $(x1,y1)$)) (*snd* (τ $(x1,y1)$)) *x2 y2* $\neq 0$
using *add-ext-add*[*OF*] *1 in-aff*
using *pd 1* **unfolding** *delta-def delta-minus-def delta-plus-def*
delta′-def delta-x-def delta-y-def
e′-aff-def e′-def
apply(*simp add*: *divide-simps t-nz*)
apply(*simp-all add*: *c-eq-1*)
apply(*simp-all split*: *if-splits add*: *divide-simps t-nz 1 algebra-simps power2-eq-square*[*symmetric*] *t-expr d-nz*)
using *add-nz*
apply(*simp add*: *d-nz*)
using *d-nz*
by (*metis distrib-left mult-eq-0-iff*)

4.4 Lemmas for associativity

lemma *cancellation-assoc*:
assumes *gluing* ‘‘ $\{((x1,y1),0)\} \in$ *e-proj gluing* ‘‘ $\{((x2,y2),0)\} \in$ *e-proj gluing* ‘‘ $\{(i\ (x2,y2),0)\} \in$ *e-proj*
shows *proj-addition* (*proj-addition* (*gluing* ‘‘ $\{((x1,y1),0)\}$) (*gluing* ‘‘ $\{((x2,y2),0)\}$)) (*gluing* ‘‘ $\{(i\ (x2,y2),0)\}$) =
gluing ‘‘ $\{((x1,y1),0)\}$
(**is** *proj-addition* (*proj-addition ?g1 ?g2*) *?g3* = *?g1*)
proof −
have *in-aff*: $(x1,y1) \in e'$-*aff* $(x2,y2) \in e'$-*aff* $i\ (x2,y2) \in e'$-*aff*
using *assms*(*1*,*2*,*3*) *e-class* **by** *auto*

have *one-in*: *gluing* ‘‘ $\{((1,\ 0),\ 0)\} \in$ *e-proj*
using *identity-proj identity-equiv* **by** *auto*

have *e-proj*: *gluing* ‘‘ $\{((x1,\ y1),\ 0)\} \in$ *e-proj*
gluing ‘‘ $\{((x2,\ y2),\ 0)\} \in$ *e-proj*
gluing ‘‘ $\{(i\ (x1,\ y1),\ 0)\} \in$ *e-proj*
$\{((1,\ 0),\ 0)\} \in$ *e-proj*
gluing ‘‘ $\{(i\ (x2,\ y2),\ 0)\} \in$ *e-proj*
using *e-proj-aff in-aff* **apply**(*simp*,*simp*)
using *assms proj-add-class-inv* **apply** *blast*
using *identity-equiv one-in* **apply** *auto*[*1*]
using *assms*(*2*) *proj-add-class-inv* **by** *blast*

{
  assume $(\exists g \in symmetries.\ (x2,\ y2) = (g \circ i)\ (x1,\ y1))$
  then obtain g **where** *g-expr*: $g \in symmetries$ $(x2,\ y2) = (g \circ i)\ (x1,\ y1)$ **by** *auto*
  then obtain g' **where** *g-expr'*: $g' \in symmetries$ $i\ (x2,y2) = g'\ (x1,\ y1)$ $g \circ g' = id$
    using *symmetries-i-inverse*[*OF g-expr*(*1*), *of x1 y1*]
      i-idemp pointfree-idE **by** *force*

  obtain r **where** *r-expr*: $r \in rotations$ $(x2,\ y2) = (\tau \circ r)\ (i\ (x1,\ y1))$ $g = \tau \circ r$
    using *g-expr sym-decomp* **by** *force*

  have *e-proj-comp*:
    gluing `` $\{(g\ (i\ (x1,\ y1)),\ 0)\} \in$ *e-proj*
    gluing `` $\{(g\ (i\ (x2,\ y2)),\ 0)\} \in$ *e-proj*
    using *assms g-expr* **apply** *force*
    using *assms g-expr' g-expr' pointfree-idE* **by** *fastforce*

  have *g2-eq*: *?g2* = *tf''* r (*gluing* `` $\{(i\ (x1,\ y1),\ 0)\}$)
    (**is** - = *tf''* - *?g4*)
    apply(*simp add*: *r-expr del*: *i.simps o-apply*)
    apply(*subst remove-sym*[*of fst* $(i\ (x1,y1))$ *snd* $(i\ (x1,y1))$ *0* $\tau \circ r$,
        simplified prod.collapse],
      (*simp add*: *e-proj e-proj-comp r-expr del*: *i.simps o-apply*)+)
    using *e-proj-comp r-expr g-expr* **apply** *blast+*
    using *tau-idemp comp-assoc*[*of* τ τ *r*,*symmetric*]
      id-comp[*of r*] **by** *presburger*

  have *eq1*: *proj-addition* (*proj-addition* *?g1* (*tf''* r *?g4*)) *?g3* = *?g1*
    apply(*subst proj-addition-comm*)
    using *e-proj g2-eq*[*symmetric*] **apply**(*simp*,*simp*)
    apply(*subst remove-add-sym*)
    using *e-proj r-expr* **apply**(*simp*,*simp*,*simp*)
    apply(*subst proj-addition-comm*)
    using *e-proj* **apply**(*simp*,*simp*)
    apply(*subst proj-add-class-inv*(*1*))
    using *e-proj* **apply** *simp*
    apply(*subst remove-add-sym*)
    using *e-proj r-expr* **apply**(*simp*,*simp*,*simp*)
    apply(*simp del*: *i.simps*)
    apply(*subst proj-add-class-identity*)
    using *e-proj* **apply** *simp*
    apply(*subst remove-sym*[*symmetric*, *of fst* $(i\ (x2,y2))$ *snd* $(i\ (x2,y2))$ *0* $\tau \circ r$,
        simplified prod.collapse comp-assoc[*of* τ τ *r*,*symmetric*]
          tau-idemp id-o])
    using *e-proj* **apply** *simp*

using *e-proj-comp*(*2*) *r-expr*(*3*) **apply** *auto*[*1*]
using *g-expr*(*1*) *r-expr*(*3*) **apply** *auto*[*1*]
using *g-expr*′(*2*) *g-expr*′(*3*) *pointfree-idE* *r-expr*(*3*) **by** *fastforce*
have *?thesis*
unfolding *g2-eq eq1* **by** *auto*
}
note *dichotomy-case* = *this*

consider (*1*) $x1 \neq 0$ $y1 \neq 0$ $x2 \neq 0$ $y2 \neq 0$ | (*2*) $x1 = 0 \vee y1 = 0 \vee x2 = 0 \vee y2 = 0$ **by** *fastforce*
then show *?thesis*
proof(*cases*)
case *1*
have *taus*: τ (*i* (*x2*, *y2*)) ∈ *e*′*-aff*
proof −
have *i* (*x2*,*y2*) ∈ *e-circ*
using *e-circ-def in-aff 1* **by** *auto*
then show *?thesis*
using τ*-circ circ-to-aff* **by** *blast*
qed

consider
(*a*) (∃ *g*∈*symmetries*. (*x2*, *y2*) = (*g* ∘ *i*) (*x1*, *y1*)) |
(*b*) ((*x1*, *y1*), *x2*, *y2*) ∈ *e*′*-aff-0* ¬ ((∃ *g*∈*symmetries*. (*x2*, *y2*) = (*g* ∘ *i*) (*x1*, *y1*))) |
(*c*) ((*x1*, *y1*), *x2*, *y2*) ∈ *e*′*-aff-1* ¬ ((∃ *g*∈*symmetries*. (*x2*, *y2*) = (*g* ∘ *i*) (*x1*, *y1*))) ((*x1*, *y1*), *x2*, *y2*) ∉ *e*′*-aff-0*
using *dichotomy-1 in-aff* **by** *blast*
then show *?thesis*
proof(*cases*)
case *a*
then show *?thesis*
using *dichotomy-case* **by** *auto*
next
case *b*
have *pd*: *delta x1 y1 x2 y2* ≠ *0*
using *b*(*1*) **unfolding** *e*′*-aff-0-def* **by** *simp*

have *ds*: *delta x2 y2 x2* (−*y2*) ≠ *0* ∨ *delta*′ *x2 y2* (*x2*) (−*y2*) ≠ *0*
using *in-aff d-n1*
unfolding *delta-def delta-plus-def delta-minus-def*
delta′*-def delta-x-def delta-y-def*
e′*-aff-def e*′*-def*
apply(*simp add*: *t-expr two-not-zero*)
apply(*safe*)
apply(*simp-all add*: *algebra-simps*)
by(*simp add*: *semiring-normalization-rules*(*18*) *semiring-normalization-rules*(*29*) *two-not-zero*)+

```
    have eq1: proj-addition ?g1 ?g2 = gluing `` {(add (x1, y1) (x2, y2), 0)}
      (is - = ?g-add)
      using gluing-add[OF assms(1,2) pd] by force
    then obtain rx ry where r-expr:
      rx = fst (add (x1, y1) (x2, y2))
      ry = snd (add (x1, y1) (x2, y2))
      (rx,ry) = add (x1,y1) (x2,y2)
      by simp
    have in-aff-r: (rx,ry) ∈ e'-aff
      using in-aff add-closure-points pd r-expr by auto
    have e-proj-r: gluing `` {((rx,ry), 0)} ∈ e-proj
      using e-proj-aff in-aff-r by auto

    consider
      (aa) (rx, ry) ∈ e-circ ∧ (∃ g∈symmetries. i (x2, y2) = (g ∘ i) (rx, ry)) |
     (bb) ((rx, ry), i (x2, y2)) ∈ e'-aff-0 ¬ ((rx, ry) ∈ e-circ ∧ (∃ g∈symmetries.
i (x2, y2) = (g ∘ i) (rx, ry))) |
     (cc) ((rx, ry), i (x2, y2)) ∈ e'-aff-1 ¬ ((rx, ry) ∈ e-circ ∧ (∃ g∈symmetries.
i (x2, y2) = (g ∘ i) (rx, ry))) ((rx, ry), i (x2, y2)) ∉ e'-aff-0
      using dichotomy-1[OF in-aff-r in-aff(3)] by fast
    then show ?thesis
    proof(cases)
      case aa
      then obtain g where g-expr:
        g ∈ symmetries (i (x2, y2)) = (g ∘ i) (rx, ry) by blast
      then obtain r where rot-expr:
        r ∈ rotations (i (x2, y2)) = (τ ∘ r ∘ i) (rx, ry) τ ∘ g = r
        using sym-decomp pointfree-idE sym-to-rot tau-idemp by fastforce

      have e-proj-sym: gluing `` {(g (i (rx, ry)), 0)} ∈ e-proj
                 gluing `` {(i (rx, ry), 0)} ∈ e-proj
        using assms(3) g-expr(2) apply force
        using e-proj-r proj-add-class-inv(2) by blast

      from aa have pd': delta rx ry (fst (i (x2,y2))) (snd (i (x2,y2))) = 0
        using wd-d-nz by auto
      consider
        (aaa) (rx, ry) ∈ e-circ ∧ (∃ g∈symmetries. τ (i (x2, y2)) = (g ∘ i) (rx,
ry)) |
          (bbb) ((rx, ry), τ (i (x2, y2))) ∈ e'-aff-0 ¬ ((rx, ry) ∈ e-circ ∧
(∃ g∈symmetries. τ (i (x2, y2)) = (g ∘ i) (rx, ry))) |
          (ccc) ((rx, ry), τ (i (x2, y2))) ∈ e'-aff-1 ¬ ((rx, ry) ∈ e-circ ∧
(∃ g∈symmetries. τ (i (x2, y2)) = (g ∘ i) (rx, ry))) ((rx, ry), τ (i (x2, y2)))
∉ e'-aff-0
        using dichotomy-1[OF in-aff-r taus] by fast
      then show ?thesis
      proof(cases)
        case aaa
        have pd'': delta rx ry (fst (τ (i (x2, y2)))) (snd (τ (i (x2, y2)))) = 0
```

```
          using wd-d-nz aaa by auto
        from aaa obtain g′ where g′-expr:
          g′ ∈ symmetries τ (i (x2, y2)) = (g′ ∘ i) (rx, ry)
          by blast
        then obtain r′ where r′-expr:
          r′ ∈ rotations τ (i (x2, y2)) = (τ ∘ r′ ∘ i) (rx, ry)
          using sym-decomp by blast
        from r′-expr have
          i (x2, y2) = (r′ ∘ i) (rx, ry)
          using tau-idemp-point by (metis comp-apply)
        from this rot-expr have (τ ∘ r ∘ i) (rx, ry) = (r′ ∘ i) (rx, ry)
          by argo
        then obtain ri′ where ri′ ∈ rotations ri′( (τ ∘ r ∘ i) (rx, ry)) = i (rx,
ry)
           by (metis comp-def rho-i-com-inverses(1) r′-expr(1) rot-inv tau-idemp
tau-sq)
        then have (τ ∘ ri′ ∘ r ∘ i) (rx, ry) = i (rx, ry)
          by (metis comp-apply rot-tau-com)
         then obtain g′′ where g′′-expr: g′′ ∈ symmetries g′′ (i ((rx, ry))) = i
(rx,ry)
          using ‹ri′ ∈ rotations› rot-expr(1) rot-comp tau-rot-sym by force
        then show ?thesis
        proof −
          have in-g: g′′ ∈ G
            using g′′-expr(1) unfolding G-def symmetries-def by blast
          have in-circ: i (rx, ry) ∈ e-circ
            using aa i-circ by blast
          then have g′′ = id
            using g-no-fp in-g in-circ g′′-expr(2) by blast
          then have False
            using sym-not-id sym-decomp  g′′-expr(1) by fastforce
          then show ?thesis by simp
        qed
      next
        case bbb
        then have pd′: delta rx ry (fst (τ (i (x2,y2)))) (snd (τ (i (x2,y2)))) ≠ 0
          unfolding e′-aff-0-def by simp
        then have pd′′: delta′ rx ry (fst (i (x2,y2))) (snd (i (x2,y2))) ≠ 0
          using 1 delta-add-delta′-1 in-aff pd r-expr by auto
        have False
          using aa pd′′ wd-d′-nz by auto
        then show ?thesis by auto
      next
        case ccc
        then have pd′: delta′ rx ry (fst (τ (i (x2,y2)))) (snd (τ (i (x2,y2)))) ≠ 0
          unfolding e′-aff-0-def e′-aff-1-def by auto
        then have pd′′: delta rx ry (fst (i (x2,y2))) (snd (i (x2,y2))) ≠ 0
          using 1 delta-add-delta′-2 in-aff pd r-expr by auto
        have False
```

```
      using aa pd'' wd-d-nz by auto
    then show ?thesis by auto
  qed
next
  case bb
  then have pd': delta rx ry (fst (i (x2,y2))) (snd (i (x2,y2))) ≠ 0
    using bb unfolding e'-aff-0-def r-expr by simp
  have add-assoc: add (add (x1, y1) (x2, y2)) (i (x2, y2)) = (x1,y1)
  proof(cases delta x2 y2 x2 (−y2) ≠ 0)
    case True
    have inv: add (x2, y2) (i (x2, y2)) = (1,0)
      using inverse-generalized[OF in-aff(2)] True
      unfolding delta-def delta-minus-def delta-plus-def by auto
    show ?thesis
      apply(subst add-add-add-add-assoc[OF in-aff(1,2),
            of fst (i (x2,y2)) snd (i (x2,y2)),
            simplified prod.collapse])
      using in-aff(3) pd True pd' r-expr apply force+
     using inv unfolding delta-def delta-plus-def delta-minus-def apply simp
      using inv neutral by presburger
  next
    case False
    then have ds': delta' x2 y2 x2 (− y2) ≠ 0
      using ds by auto
    have inv: ext-add (x2, y2) (i (x2, y2)) = (1,0)
      using ext-add-inverse 1 by simp
    show ?thesis
      apply(subst add-add-add-ext-assoc-points[of x1 y1 x2 y2
             fst (i (x2,y2)) snd (i (x2,y2)), simplified prod.collapse])
      using in-aff pd ds' pd' r-expr apply force+
     using inv unfolding delta-def delta-plus-def delta-minus-def apply simp
      using inv neutral by presburger
  qed

  show ?thesis
    apply(subst gluing-add,(simp add: e-proj pd del: add.simps i.simps)+)
    apply(subst gluing-add[of rx ry 0 fst (i (x2,y2)) snd (i (x2,y2)),
                simplified r-expr prod.collapse])
    using e-proj-r r-expr e-proj pd' apply(simp,simp,simp)
    apply(subst add-assoc)
    by auto
next
  case cc
  then have pd': delta' rx ry (fst (i (x2,y2))) (snd (i (x2,y2))) ≠ 0
    using cc unfolding e'-aff-1-def r-expr by simp
  have add-assoc: ext-add (add (x1, y1) (x2, y2)) (i (x2, y2)) = (x1,y1)
  proof(cases delta x2 y2 x2 (−y2) ≠ 0)
    case True
    have inv: add (x2, y2) (i (x2, y2)) = (1,0)
```

```
      using inverse-generalized[OF in-aff(2)] True
      unfolding delta-def delta-minus-def delta-plus-def by auto
    show ?thesis
      apply(subst ext-add-add-add-assoc-points[OF in-aff(1,2),
           of fst (i (x2,y2)) snd (i (x2,y2)),
           simplified prod.collapse])
      using in-aff(3) pd True pd' r-expr apply force+
     using inv unfolding delta-def delta-plus-def delta-minus-def apply simp
      using inv neutral by presburger
  next
    case False
    then have ds': delta' x2 y2 x2 (− y2) ≠ 0
      using ds by auto
    have inv: ext-add (x2, y2) (i (x2, y2)) = (1,0)
      using ext-add-inverse 1 by simp
    show ?thesis
      apply(subst ext-add-add-ext-assoc-points[of x1 y1 x2 y2
             fst (i (x2,y2)) snd (i (x2,y2)), simplified prod.collapse])
      using in-aff pd ds' pd' r-expr apply force+
     using inv unfolding delta-def delta-plus-def delta-minus-def apply simp
      using inv neutral by presburger
  qed

  show ?thesis
    apply(subst gluing-add,(simp add: e-proj pd del: add.simps i.simps)+)
    apply(subst gluing-ext-add[of rx ry 0 fst (i (x2,y2)) snd (i (x2,y2)),
                 simplified r-expr prod.collapse])
    using e-proj-r r-expr e-proj pd' apply(simp,simp,simp)
    apply(subst add-assoc)
    by auto
 qed
next
 case c
 have pd: delta' x1 y1 x2 y2 ≠ 0
   using c unfolding e'-aff-1-def by simp

 have ds: delta x2 y2 x2 (−y2) ≠ 0 ∨
         delta' x2 y2 (x2) (−y2) ≠ 0
   using in-aff d-n1 add-self by blast

 have eq1: proj-addition ?g1 ?g2 = gluing ‘‘ {(ext-add (x1, y1) (x2, y2), 0)}
   (is - = ?g-add)
   using gluing-ext-add[OF assms(1,2) pd] by force
 then obtain rx ry where r-expr:
   rx = fst (ext-add (x1, y1) (x2, y2))
   ry = snd (ext-add (x1, y1) (x2, y2))
   (rx,ry) = ext-add (x1,y1) (x2,y2)
   by simp
 have in-aff-r: (rx,ry) ∈ e'-aff
```

using *in-aff ext-add-closure-points pd r-expr* **by** *auto*
have *e-proj-r*: *gluing* ‘‘ $\{((rx,ry),\ 0)\} \in$ *e-proj*
using *e-proj-aff in-aff-r* **by** *auto*

consider
(aa) $(rx,\ ry) \in$ *e-circc* $\wedge$ $(\exists\, g{\in}symmetries.\ i\ (x2,\ y2) = (g \circ i)\ (rx,\ ry))$ |
(bb) $((rx,\ ry),\ i\ (x2,\ y2)) \in e'\text{-}aff\text{-}0$ $\neg$ $((rx,\ ry) \in e\text{-}circ \wedge (\exists\, g{\in}symmetries.\ i\ (x2,\ y2) = (g \circ i)\ (rx,\ ry)))$ |
(cc) $((rx,\ ry),\ i\ (x2,\ y2)) \in e'\text{-}aff\text{-}1$ $\neg$ $((rx,\ ry) \in e\text{-}circ \wedge (\exists\, g{\in}symmetries.\ i\ (x2,\ y2) = (g \circ i)\ (rx,\ ry)))$ $((rx,\ ry),\ i\ (x2,\ y2)) \notin e'\text{-}aff\text{-}0$
using *dichotomy-1*[*OF in-aff-r in-aff*(*3*)] **by** *fast*
then show *?thesis*
proof(*cases*)
case *aa*
then obtain *g* **where** *g-expr*:
$g \in symmetries$ $(i\ (x2,\ y2)) = (g \circ i)\ (rx,\ ry)$ **by** *blast*
then obtain *r* **where** *rot-expr*:
$r \in rotations$ $(i\ (x2,\ y2)) = (\tau \circ r \circ i)\ (rx,\ ry)$ $\tau \circ g = r$
using *sym-decomp pointfree-idE sym-to-rot tau-idemp* **by** *fastforce*

have *e-proj-sym*: *gluing* ‘‘ $\{(g\ (i\ (rx,\ ry)),\ 0)\} \in$ *e-proj*
gluing ‘‘ $\{(i\ (rx,\ ry),\ 0)\} \in$ *e-proj*
using *assms*(*3*) *g-expr*(*2*) **apply** *force*
using *e-proj-r proj-add-class-inv*(*2*) **by** *blast*

from *aa* **have** *pd'*: $delta\ rx\ ry\ (fst\ (i\ (x2,y2)))\ (snd\ (i\ (x2,y2))) = 0$
using *wd-d-nz* **by** *auto*
consider
(aaa) $(rx,\ ry) \in e\text{-}circ \wedge (\exists\, g{\in}symmetries.\ \tau\ (i\ (x2,\ y2)) = (g \circ i)\ (rx,\ ry))$ |
(bbb) $((rx,\ ry),\ \tau\ (i\ (x2,\ y2))) \in e'\text{-}aff\text{-}0$ $\neg$ $((rx,\ ry) \in e\text{-}circ \wedge (\exists\, g{\in}symmetries.\ \tau\ (i\ (x2,\ y2)) = (g \circ i)\ (rx,\ ry)))$ |
(ccc) $((rx,\ ry),\ \tau\ (i\ (x2,\ y2))) \in e'\text{-}aff\text{-}1$ $\neg$ $((rx,\ ry) \in e\text{-}circ \wedge (\exists\, g{\in}symmetries.\ \tau\ (i\ (x2,\ y2)) = (g \circ i)\ (rx,\ ry)))$ $((rx,\ ry),\ \tau\ (i\ (x2,\ y2))) \notin e'\text{-}aff\text{-}0$
using *dichotomy-1*[*OF in-aff-r taus*] **by** *fast*
then show *?thesis*
proof(*cases*)
case *aaa*
have *pd''*: $delta\ rx\ ry\ (fst\ (\tau\ (i\ (x2,\ y2))))\ (snd\ (\tau\ (i\ (x2,\ y2)))) = 0$
using *wd-d-nz aaa* **by** *auto*
from *aaa* **obtain** *g'* **where** *g'-expr*:
$g' \in symmetries$ $\tau\ (i\ (x2,\ y2)) = (g' \circ i)\ (rx,\ ry)$
by *blast*
then obtain *r'* **where** *r'-expr*:
$r' \in rotations$ $\tau\ (i\ (x2,\ y2)) = (\tau \circ r' \circ i)\ (rx,\ ry)$
using *sym-decomp* **by** *blast*
from *r'-expr* **have**
$i\ (x2,\ y2) = (r' \circ i)\ (rx,\ ry)$

using *tau-idemp-point* **by** (*metis comp-apply*)
from *this rot-expr* **have** $(\tau \circ r \circ i)$ $(rx,\ ry) = (r' \circ i)$ $(rx,\ ry)$
by *argo*
then obtain ri' **where** $ri' \in$ *rotations* $ri'(\ (\tau \circ r \circ i)\ (rx,\ ry)) = i\ (rx,\ ry)$
by (*metis comp-def rho-i-com-inverses*(*1*) r'*-expr*(*1*) *rot-inv tau-idemp tau-sq*)
then have $(\tau \circ ri' \circ r \circ i)$ $(rx,\ ry) = i\ (rx,\ ry)$
by (*metis comp-apply rot-tau-com*)
then obtain g'' **where** g''*-expr*: $g'' \in$ *symmetries* $g''\ (i\ ((rx,\ ry))) = i\ (rx,ry)$
using ‹$ri' \in$ *rotations*› *rot-expr*(*1*) *rot-comp tau-rot-sym* **by** *force*
then show *?thesis*
proof −
have *in-g*: $g'' \in G$
using g''*-expr*(*1*) **unfolding** *G-def symmetries-def* **by** *blast*
have *in-circ*: $i\ (rx,\ ry) \in$ *e-circ*
using *aa i-circ* **by** *blast*
then have $g'' = id$
using *g-no-fp in-g in-circ* g''*-expr*(*2*) **by** *blast*
then have *False*
using *sym-not-id sym-decomp* g''*-expr*(*1*) **by** *fastforce*
then show *?thesis* **by** *simp*
qed
next
case *bbb*
then have pd': *delta rx ry* (*fst* $(\tau\ (i\ (x2,y2))))$ (*snd* $(\tau\ (i\ (x2,y2)))) \neq 0$
unfolding e'*-aff-0-def* **by** *simp*
then have pd'': $delta'$ *rx ry* (*fst* $(i\ (x2,y2)))$ (*snd* $(i\ (x2,y2))) \neq 0$
using *1* $delta'$*-add-delta-2 in-aff pd r-expr* **by** *meson*
have *False*
using *aa* pd'' *wd-d*$'$*-nz* **by** *auto*
then show *?thesis* **by** *auto*
next
case *ccc*
then have pd': $delta'$ *rx ry* (*fst* $(\tau\ (i\ (x2,y2))))$ (*snd* $(\tau\ (i\ (x2,y2)))) \neq 0$
unfolding e'*-aff-0-def* e'*-aff-1-def* **by** *auto*
then have pd'': *delta rx ry* (*fst* $(i\ (x2,y2)))$ (*snd* $(i\ (x2,y2))) \neq 0$
using *1* $delta'$*-add-delta-1 in-aff pd r-expr* **by** *auto*
have *False*
using *aa* pd'' *wd-d-nz* **by** *auto*
then show *?thesis* **by** *auto*
qed
next
case *bb*
then have pd': *delta rx ry* (*fst* $(i\ (x2,y2)))$ (*snd* $(i\ (x2,y2))) \neq 0$
using *bb* **unfolding** e'*-aff-0-def r-expr* **by** *simp*
have *add-assoc*: *add* (*ext-add* $(x1,\ y1)$ $(x2,\ y2)$) $(i\ (x2,\ y2)) = (x1,y1)$
proof(*cases delta x2 y2 x2* $(-y2) \neq 0$)

```
      case True
      have inv: add (x2, y2) (i (x2, y2)) = (1,0)
        using inverse-generalized[OF in-aff(2)] True
        unfolding delta-def delta-minus-def delta-plus-def by auto
      show ?thesis
        apply(subst add-ext-add-add-assoc-points[OF in-aff(1,2),
              of fst (i (x2,y2)) snd (i (x2,y2)),
              simplified prod.collapse])
        using in-aff(3) pd True pd' r-expr apply force+
       using inv unfolding delta-def delta-plus-def delta-minus-def apply simp
        using inv neutral by presburger
    next
      case False
      then have ds': delta' x2 y2 x2 (− y2) ≠ 0
        using ds by auto
      have inv: ext-add (x2, y2) (i (x2, y2)) = (1,0)
        using ext-add-inverse 1 by simp
      show ?thesis
        apply(subst add-ext-add-ext-assoc-points[of x1 y1 x2 y2
               fst (i (x2,y2)) snd (i (x2,y2)), simplified prod.collapse])
        using in-aff pd ds' pd' r-expr apply force+
       using inv unfolding delta-def delta-plus-def delta-minus-def apply simp
        using inv neutral by presburger
    qed

    show ?thesis
         apply(subst gluing-ext-add,(simp add: e-proj pd del: ext-add.simps
i.simps)+)
      apply(subst gluing-add[of rx ry 0 fst (i (x2,y2)) snd (i (x2,y2)),
               simplified r-expr prod.collapse])
      using e-proj-r r-expr e-proj pd' apply(simp,simp,simp)
      apply(subst add-assoc)
      by auto
  next
    case cc
    then have pd': delta' rx ry (fst (i (x2,y2))) (snd (i (x2,y2))) ≠ 0
      using cc unfolding e'-aff-1-def r-expr by simp
    have add-assoc: ext-add (ext-add (x1, y1) (x2, y2)) (i (x2, y2)) = (x1,y1)
    proof(cases delta x2 y2 x2 (−y2) ≠ 0)
      case True
      have inv: add (x2, y2) (i (x2, y2)) = (1,0)
        using inverse-generalized[OF in-aff(2)] True
        unfolding delta-def delta-minus-def delta-plus-def by auto
      show ?thesis
        apply(subst ext-ext-add-add-assoc-points[OF in-aff(1,2),
              of fst (i (x2,y2)) snd (i (x2,y2)),
              simplified prod.collapse])
        using in-aff(3) pd True pd' r-expr apply force+
       using inv unfolding delta-def delta-plus-def delta-minus-def apply simp
```

```
          using inv neutral by presburger
        next
          case False
          then have ds': delta' x2 y2 x2 (- y2) ≠ 0
            using ds by auto
          have inv: ext-add (x2, y2) (i (x2, y2)) = (1,0)
            using ext-add-inverse 1 by simp
          show ?thesis
            apply(subst ext-ext-add-ext-assoc-points[of x1 y1 x2 y2
                  fst (i (x2,y2)) snd (i (x2,y2)), simplified prod.collapse])
            using in-aff pd ds' pd' r-expr apply force+
           using inv unfolding delta-def delta-plus-def delta-minus-def apply simp
            using inv neutral by presburger
        qed

        show ?thesis
              apply(subst gluing-ext-add,(simp add: e-proj pd del: ext-add.simps
i.simps)+)
          apply(subst gluing-ext-add[of rx ry 0 fst (i (x2,y2)) snd (i (x2,y2)),
                       simplified r-expr prod.collapse])
          using e-proj-r r-expr e-proj pd' apply(simp,simp,simp)
          apply(subst add-assoc)
          by auto
      qed
    qed
  next
    case 2

    then have (∃ r ∈ rotations. (x1,y1) = r (1,0)) ∨ (∃ r ∈ rotations. (x2,y2)
= r (1,0))
      using in-aff(1,2) unfolding e'-aff-def e'-def
      apply(safe)
      unfolding rotations-def
      by(simp,algebra)+
    then consider
      (a) (∃ r ∈ rotations. (x1,y1) = r (1,0)) |
      (b) (∃ r ∈ rotations. (x2,y2) = r (1,0)) by argo
    then show ?thesis
      proof(cases)
        case a
        then obtain r where rot-expr: r ∈ rotations (x1, y1) = r (1, 0) by blast

        have proj-addition (gluing `` {((x1, y1), 0)}) (gluing `` {((x2, y2), 0)}) =
              proj-addition (tf r (gluing `` {((1, 0), 0)})) (gluing `` {((x2, y2), 0)})
          using remove-rotations[OF one-in rot-expr(1)] rot-expr(2) by presburger
        also have ... = tf r (proj-addition (gluing `` {((1, 0), 0)}) (gluing `` {((x2,
y2), 0)}))
          using remove-add-rotation assms rot-expr one-in by presburger
        also have ... = tf r (gluing `` {((x2, y2), 0)})
```

using *proj-add-class-identity*
by (*simp add*: *e-proj*(*2*) *identity-equiv*)
finally have *eq1*: *proj-addition* (*gluing* ‘‘ {((*x1*, *y1*), *0*)}) (*gluing* ‘‘ {((*x2*, *y2*), *0*)}) =
tf r (*gluing* ‘‘ {((*x2*, *y2*), *0*)}) **by** *argo*

have *proj-addition* (*proj-addition* (*gluing* ‘‘ {((*x1*, *y1*), *0*)}) (*gluing* ‘‘ {((*x2*, *y2*), *0*)})) (*gluing* ‘‘ {(*i* (*x2*, *y2*), *0*)}) =
proj-addition (*tf r* (*gluing* ‘‘ {((*x2*, *y2*), *0*)})) (*gluing* ‘‘ {(*i* (*x2*, *y2*), *0*)})
using *eq1* **by** *argo*
also have ... = *tf r* (*proj-addition* (*gluing* ‘‘ {((*x2*, *y2*), *0*)}) (*gluing* ‘‘ {(*i* (*x2*, *y2*), *0*)}))
using *remove-add-rotation rot-expr well-defined proj-addition-def assms one-in* **by** *simp*
also have ... = *tf r* (*gluing* ‘‘ {((*1*, *0*), *0*)})
using *proj-addition-def proj-add-class-inv assms*
by (*simp add*: *identity-equiv*)
finally have *eq2*: *proj-addition* (*proj-addition* (*gluing* ‘‘ {((*x1*, *y1*), *0*)}) (*gluing* ‘‘ {((*x2*, *y2*), *0*)})) (*gluing* ‘‘ {(*i* (*x2*, *y2*), *0*)}) =
tf r (*gluing* ‘‘ {((*1*, *0*), *0*)}) **by** *blast*
show *?thesis*
apply(*subst eq2*)
using *remove-rotations*[*OF one-in rot-expr*(*1*)] *rot-expr*(*2*) **by** *presburger*
next
case *b*
then obtain *r* **where** *rot-expr*: $r \in$ *rotations* (*x2*, *y2*) = *r* (*1*, *0*) **by** *blast*
then obtain r' **where** *rot-expr'*: $r' \in$ *rotations i* (*x2*, *y2*) = r' (*i* (*1*, *0*)) $r \circ r' = id$
using *rotations-i-inverse*[*OF rot-expr*(*1*)]
by (*metis* (*no-types*, *hide-lams*) *comp-apply comp-assoc comp-id diff-0 diff-zero i.simps id-apply id-comp rot-inv*)
have *proj-addition* (*gluing* ‘‘ {((*x1*, *y1*), *0*)}) (*gluing* ‘‘ {((*x2*, *y2*), *0*)}) =
proj-addition (*gluing* ‘‘ {((*x1*, *y1*), *0*)}) (*tf r* (*gluing* ‘‘ {((*1*, *0*), *0*)}))
using *remove-rotations*[*OF one-in rot-expr*(*1*)] *rot-expr*(*2*) **by** *presburger*
also have ... = *tf r* (*proj-addition* (*gluing* ‘‘ {((*x1*, *y1*), *0*)}) (*gluing* ‘‘ {((*1*, *0*), *0*)}))
using *remove-add-rotation assms rot-expr one-in*
by (*metis proj-addition-comm remove-rotations*)
also have ... = *tf r* (*gluing* ‘‘ {((*x1*, *y1*), *0*)})
using *proj-add-class-identity assms*
identity-equiv one-in proj-addition-comm **by** *metis*
finally have *eq1*: *proj-addition* (*gluing* ‘‘ {((*x1*, *y1*), *0*)}) (*gluing* ‘‘ {((*x2*, *y2*), *0*)}) =
tf r (*gluing* ‘‘ {((*x1*, *y1*), *0*)}) **by** *argo*

have *proj-addition* (*proj-addition* (*gluing* ‘‘ {((*x1*, *y1*), *0*)}) (*gluing* ‘‘ {((*x2*, *y2*), *0*)})) (*gluing* ‘‘ {(*i* (*x2*, *y2*), *0*)}) =
proj-addition (*tf r* (*gluing* ‘‘ {((*x1*, *y1*), *0*)})) (*gluing* ‘‘ {(*i* (*x2*, *y2*),

0)})
    using *eq1* **by** *argo*
   also have ... = *tf r* (*proj-addition* (*gluing* `` {((*x1*, *y1*), *0*)}) (*gluing* `` {(*i* (*x2*, *y2*), *0*)}))
    using *remove-add-rotation rot-expr well-defined proj-addition-def assms one-in* **by** *simp*
   also have ... = *tf r* (*proj-addition* (*gluing* `` {((*x1*, *y1*), *0*)}) (*tf r′* (*gluing* `` {(*i* (*1*, *0*), *0*)})))
    using *remove-rotations one-in rot-expr′* **by** *simp*
   also have ... = *tf r* (*tf r′* (*proj-addition* (*gluing* `` {((*x1*, *y1*), *0*)}) ((*gluing* `` {(*i* (*1*, *0*), *0*)}))))
    using *proj-add-class-inv assms*
   by (*metis i.simps one-in proj-addition-comm projective-curve.remove-add-rotation projective-curve-axioms rot-expr′*(*1*) *rotation-preserv-e-proj*)
   also have ... = *tf* (*id*) (*proj-addition* (*gluing* `` {((*x1*, *y1*), *0*)}) ((*gluing* `` {((*1*, *0*), *0*)})))
    using *tf-comp rot-expr′* **by** *force*
   also have ... = (*gluing* `` {((*x1*, *y1*), *0*)})
    apply(*subst tf-id*)
    by (*simp add*: *e-proj*(*1*) *identity-equiv identity-proj proj-addition-comm proj-add-class-identity*)
   finally have *eq2*: *proj-addition* (*proj-addition* (*gluing* `` {((*x1*, *y1*), *0*)}) (*gluing* `` {((*x2*, *y2*), *0*)})) (*gluing* `` {(*i* (*x2*, *y2*), *0*)}) =
     (*gluing* `` {((*x1*, *y1*), *0*)}) **by** *blast*
   show *?thesis* **by**(*subst eq2*,*simp*)
  qed
 qed
qed

lemma *e′-aff-0-invariance*:
 ((*x*,*y*),(*x′*,*y′*)) ∈ *e′-aff-0* ⟹ ((*x′*,*y′*),(*x*,*y*)) ∈ *e′-aff-0*
 unfolding *e′-aff-0-def*
 apply(*subst* (*1*) *prod.collapse*[*symmetric*])
 apply(*simp*)
 unfolding *delta-def delta-plus-def delta-minus-def*
 by *algebra*

lemma *e′-aff-1-invariance*:
 ((*x*,*y*),(*x′*,*y′*)) ∈ *e′-aff-1* ⟹ ((*x′*,*y′*),(*x*,*y*)) ∈ *e′-aff-1*
 unfolding *e′-aff-1-def*
 apply(*subst* (*1*) *prod.collapse*[*symmetric*])
 apply(*simp*)
 unfolding *delta′-def delta-x-def delta-y-def*
 by *algebra*

lemma *assoc-1*:
 assumes *gluing* `` {((*x1*, *y1*), *0*)} ∈ *e-proj*

gluing `` {((*x2*, *y2*), *0*)} ∈ *e-proj*
gluing `` {((*x3*, *y3*), *0*)} ∈ *e-proj*
assumes *a*: *g* ∈ *symmetries* (*x2*, *y2*) = (*g* ∘ *i*) (*x1*, *y1*)
shows
proj-addition (*gluing* `` {((*x1*, *y1*), *0*)}) (*gluing* `` {((*x2*, *y2*), *0*)}) =
tf'' (τ ∘ *g*) {((*1*,*0*),*0*)} (**is** *proj-addition ?g1 ?g2* = -)
proj-addition (*proj-addition* (*gluing* `` {((*x1*, *y1*), *0*)}) (*gluing* `` {((*x2*, *y2*), *0*)})) (*gluing* `` {((*x3*, *y3*), *0*)}) =
tf'' (τ ∘ *g*) (*gluing* `` {((*x3*, *y3*), *0*)})
proj-addition (*gluing* `` {((*x1*, *y1*), *0*)}) (*proj-addition* (*gluing* `` {((*x2*, *y2*), *0*)}) (*gluing* `` {((*x3*, *y3*), *0*)})) =
tf'' (τ ∘ *g*) (*gluing* `` {((*x3*, *y3*), *0*)}) (**is** *proj-addition ?g1* (*proj-addition ?g2 ?g3*) = -)
proof −
have *in-aff*: (*x1*,*y1*) ∈ *e'-aff* (*x2*,*y2*) ∈ *e'-aff* (*x3*,*y3*) ∈ *e'-aff*
using *assms*(*1*,*2*,*3*) *e-class* **by** *auto*

have *one-in*: {((*1*, *0*), *0*)} ∈ *e-proj*
using *identity-proj* **by** *force*

have *rot*: τ ∘ *g* ∈ *rotations* **using** *sym-to-rot assms* **by** *blast*

have *e-proj*: *gluing* `` {(*g* (*i* (*x1*, *y1*)), *0*)} ∈ *e-proj*
gluing `` {(*i* (*x1*, *y1*), *0*)} ∈ *e-proj* (**is** *?ig1* ∈ -)
proj-addition (*gluing* `` {(*i* (*x1*, *y1*), *0*)}) (*gluing* `` {((*x3*, *y3*), *0*)}) ∈ *e-proj*
using *assms*(*2*,*5*) **apply** *auto*[*1*]
using *assms*(*1*) *proj-add-class-inv*(*2*) **apply** *auto*[*1*]
using *assms*(*1*,*3*) *proj-add-class-inv*(*2*) *well-defined* **by** *blast*

show *1*: *proj-addition ?g1 ?g2* = *tf''* (τ ∘ *g*) {((*1*,*0*),*0*)}
proof −
have *eq1*: *?g2* = *tf''* (τ ∘ *g*) *?ig1*
apply(*simp add*: *assms*(*5*))
apply(*subst* (*2 5*) *prod.collapse*[*symmetric*])
apply(*subst remove-sym*)
using *e-proj assms* **by** *auto*
have *eq2*: *proj-addition ?g1* (*tf''* (τ ∘ *g*) *?ig1*) =
tf'' (τ ∘ *g*) (*proj-addition ?g1 ?ig1*)
apply(*subst* (*1 2*) *proj-addition-comm*)
using *assms e-proj* **apply**(*simp*,*simp*)
using *assms*(*2*) *eq1* **apply** *auto*[*1*]
apply(*subst remove-add-sym*)
using *assms*(*1*) *e-proj*(*2*) *rot* **by** *auto*
have *eq3*: *tf''* (τ ∘ *g*) (*proj-addition ?g1 ?ig1*) = *tf''* (τ ∘ *g*) {((*1*,*0*),*0*)}
using *assms*(*1*) *proj-add-class-inv* **by** *auto*
show *?thesis* **using** *eq1 eq2 eq3* **by** *presburger*
qed

```
  have proj-addition (proj-addition ?g1 ?g2) ?g3 =
        proj-addition (tf'' (τ ∘ g) {((1,0),0)}) ?g3
    using 1 by force
  also have ... = tf'' (τ ∘ g) (proj-addition ({((1,0),0)}) ?g3)
    by (simp add: assms(3) one-in remove-add-sym rot)
  also have ... = tf'' (τ ∘ g) ?g3
    using assms(3) identity-equiv proj-add-class-identity by simp
  finally show 2: proj-addition (proj-addition ?g1 ?g2) ?g3 = tf'' (τ ∘ g) ?g3
    by blast

  have proj-addition ?g1 (proj-addition ?g2 ?g3) =
    proj-addition ?g1 (proj-addition (gluing `` {(g (i (x1, y1)), 0)}) ?g3)
      using assms by simp
  also have ... = proj-addition ?g1 (tf'' (τ ∘ g) (proj-addition (gluing `` {(i (x1,
y1), 0)}) ?g3))
  proof −
    have eq1: gluing `` {(g (i (x1, y1)), 0)} = tf'' (τ ∘ g) ?ig1
      apply(subst (2 5) prod.collapse[symmetric])
      apply(subst remove-sym)
      using e-proj assms by auto
    have eq2: proj-addition (tf'' (τ ∘ g) ?ig1) ?g3 =
             tf'' (τ ∘ g) (proj-addition ?ig1 ?g3)
      apply(subst remove-add-sym)
      using assms(3) e-proj(2) rot by auto

    show ?thesis using eq1 eq2 by presburger
  qed
  also have ... = tf'' (τ ∘ g)  (proj-addition ?g1 (proj-addition ?ig1 ?g3))
    apply(subst (1 3) proj-addition-comm)
    using assms apply simp
    using e-proj(3) apply auto[1]
     apply (metis assms(3) e-proj(2) i.simps remove-add-sym rot
              tf''-preserv-e-proj well-defined)
    apply(subst remove-add-sym)
    using e-proj(3) assms(1) rot by auto
  also have ... = tf'' (τ ∘ g) ?g3
  proof −
    have proj-addition ?g1 (proj-addition ?ig1 ?g3) = ?g3
      apply(subst (1 2) proj-addition-comm)
      using e-proj assms apply (simp,simp,simp)
      using assms(3) e-proj(2) well-defined apply auto[1]
      using cancellation-assoc i-idemp-explicit
      by (metis assms(1) assms(3) e-proj(2) i.simps)
    then show ?thesis by argo
  qed
  finally show 3: proj-addition ?g1 (proj-addition ?g2 ?g3) =
                  tf'' (τ ∘ g) ?g3 by blast
qed
```

```
lemma assoc-11:
  assumes gluing ‘‘ {((x1, y1), 0)}  ∈ e-proj gluing ‘‘ {((x2, y2), 0)} ∈ e-proj
gluing ‘‘ {((x3, y3), 0)} ∈ e-proj
  assumes a: g ∈ symmetries (x3, y3) = (g ∘ i) (x2, y2)
  shows
     proj-addition (proj-addition (gluing ‘‘ {((x1, y1), 0)}) (gluing ‘‘ {((x2, y2),
0)})) (gluing ‘‘ {((x3, y3), 0)}) =
      proj-addition (gluing ‘‘ {((x1, y1), 0)}) (proj-addition (gluing ‘‘ {((x2, y2),
0)}) (gluing ‘‘ {((x3, y3), 0)}))
    (is proj-addition (proj-addition ?g1 ?g2) ?g3 = -)
proof –
  have in-aff: (x1,y1) ∈ e′-aff (x2,y2) ∈ e′-aff (x3,y3) ∈ e′-aff
    using assms(1,2,3) e-class by auto

  have one-in: {((1, 0), 0)} ∈ e-proj
    using identity-equiv identity-proj by auto

  have rot: τ ∘ g ∈ rotations using sym-to-rot assms by blast

  have e-proj: gluing ‘‘ {(g (i (x2, y2)), 0)}  ∈ e-proj
             gluing ‘‘ {(i (x2, y2), 0)}  ∈ e-proj (is ?ig2 ∈ -)
             proj-addition ?g1 ?g2 ∈ e-proj
    using assms(3,5) apply simp
    using proj-add-class-inv assms(2) apply fast
    using assms(1,2) well-defined by simp

  have eq1: ?g3 = tf′′ (τ ∘ g) ?ig2
    apply(subst a)
    apply(subst comp-apply)
    apply(subst (2) prod.collapse[symmetric])
    apply(subst remove-sym[OF - - assms(4)])
    using e-proj apply(simp,simp)
    by(subst prod.collapse,simp)
  have eq2: proj-addition (proj-addition ?g1 ?g2) (tf′′ (τ ∘ g) ?ig2) =
           tf′′ (τ ∘ g) ?g1
    apply(subst (2) proj-addition-comm)
    using e-proj eq1 assms(3) apply(simp,simp)
    apply(subst remove-add-sym)
    using e-proj rot apply(simp,simp,simp)
    apply(subst proj-addition-comm)
    using e-proj apply(simp,simp)
    apply(subst cancellation-assoc)
    using assms(1,2) e-proj by(simp,simp,simp,simp)
  have eq3: proj-addition ?g2 (tf′′ (τ ∘ g) ?ig2) =
           tf′′ (τ ∘ g) {((1, 0), 0)}
    apply(subst proj-addition-comm)
    using e-proj eq1 assms(2,3) apply(simp,simp)
    apply(subst remove-add-sym)
    using e-proj rot assms(2) apply(simp,simp,simp)
```

apply(*subst proj-addition-comm*)
using *e-proj eq1 assms*(*2*,*3*) **apply**(*simp*,*simp*)
apply(*subst proj-add-class-inv*(*1*))
using *assms*(*2*) **apply** *blast*
by *simp*

show *?thesis*
apply(*subst eq1*)
apply(*subst eq2*)
apply(*subst eq1*)
apply(*subst eq3*)
apply(*subst proj-addition-comm*)
using *assms*(*1*) **apply**(*simp*)
using *tf''-preserv-e-proj*[*OF - rot*] *one-in identity-equiv* **apply** *metis*
apply(*subst remove-add-sym*)
using *identity-equiv one-in assms*(*1*) *rot* **apply**(*argo*,*simp*,*simp*)
apply(*subst proj-add-class-identity*)
using *assms*(*1*) **apply**(*simp*)
by *blast*
qed

lemma *assoc-111-add*:
assumes *gluing* `` $\{((x1, y1), 0)\}$ $\in$ *e-proj gluing* `` $\{((x2, y2), 0)\}$ $\in$ *e-proj gluing* `` $\{((x3, y3), 0)\}$ $\in$ *e-proj*
assumes *22*: *g*$\in$*symmetries* $(x1, y1) = (g \circ i)$ (*add* $(x2,y2)$ $(x3,y3)$) $((x2, y2), x3, y3) \in$ *e'-aff-0*
shows
proj-addition (*proj-addition* (*gluing* `` $\{((x1, y1), 0)\}$) (*gluing* `` $\{((x2, y2), 0)\}$)) (*gluing* `` $\{((x3, y3), 0)\}$) =
proj-addition (*gluing* `` $\{((x1, y1), 0)\}$) (*proj-addition* (*gluing* `` $\{((x2, y2), 0)\}$) (*gluing* `` $\{((x3, y3), 0)\}$))
(**is** *proj-addition* (*proj-addition ?g1 ?g2*) *?g3* = -)
proof −
have *in-aff*: $(x1,y1) \in$ *e'-aff* $(x2,y2) \in$ *e'-aff* $(x3,y3) \in$ *e'-aff*
using *assms*(*1*,*2*,*3*) *e-class* **by** *auto*

have *e-proj-0*: *gluing* `` $\{(i\ (x1,y1), 0)\}$ $\in$ *e-proj* (**is** *?ig1* $\in$ -)
gluing `` $\{(i\ (x2,y2), 0)\}$ $\in$ *e-proj* (**is** *?ig2* $\in$ -)
gluing `` $\{(i\ (x3,y3), 0)\}$ $\in$ *e-proj* (**is** *?ig3* $\in$ -)
using *assms proj-add-class-inv* **by** *blast+*

have *p-delta*: *delta x2 y2 x3 y3* $\neq 0$
delta (*fst* (*i* $(x2,y2)$)) (*snd* (*i* $(x2,y2)$))
(*fst* (*i* $(x3,y3)$)) (*snd* (*i* $(x3,y3)$)) $\neq 0$
using *22* **unfolding** *e'-aff-0-def* **apply** *simp*
using *22* **unfolding** *e'-aff-0-def delta-def delta-plus-def delta-minus-def* **by** *simp*

define *add-2-3* **where** *add-2-3* = *add* $(x2,y2)$ $(x3,y3)$

have *add-in*: *add-2-3* ∈ *e′-aff*
unfolding *e′-aff-def add-2-3-def*
apply(*simp del*: *add.simps*)
apply(*subst* (*2*) *prod.collapse*[*symmetric*])
apply(*standard*)
apply(*simp del*: *add.simps add*: *e-e′-iff* [*symmetric*])
apply(*subst add-closure*)
using *in-aff e-e′-iff 22* **unfolding** *e′-aff-def e′-aff-0-def delta-def* **by**(*fastforce*)+
have *e-proj-2-3*: *gluing* ‘‘ {(*add-2-3*, *0*)} ∈ *e-proj*
gluing ‘‘ {(*i add-2-3*, *0*)} ∈ *e-proj*
using *add-in add-2-3-def e-points* **apply** *simp*
using *add-in add-2-3-def e-points proj-add-class-inv* **by** *force*

from *22* **have** *g-expr*: *g* ∈ *symmetries* (*x1*,*y1*) = (*g* ∘ *i*) *add-2-3* **unfolding** *add-2-3-def* **by** *auto*
then have *rot*: *τ* ∘ *g* ∈ *rotations* **using** *sym-to-rot* **by** *blast*

have *e-proj-2-3-g*: *gluing* ‘‘ {(*g* (*i add-2-3*), *0*)} ∈ *e-proj*
using *e-proj-2-3 g-expr assms*(*1*) **by** *auto*

have *proj-addition ?g1* (*proj-addition ?g2 ?g3*) =
proj-addition (*gluing* ‘‘ {((*g* ∘ *i*) *add-2-3*, *0*)}) (*proj-addition ?g2 ?g3*)
using *g-expr* **by** *simp*
also have ... = *proj-addition* (*gluing* ‘‘ {((*g* ∘ *i*) *add-2-3*, *0*)}) (*gluing* ‘‘ {(*add-2-3*, *0*)})
using *gluing-add add-2-3-def p-delta assms*(*2*,*3*) **by** *force*
also have ... = *tf ′′* (*τ* ∘ *g*) (*proj-addition* (*gluing* ‘‘ {(*i add-2-3*, *0*)}) (*gluing* ‘‘ {(*add-2-3*, *0*)}))
apply(*subst comp-apply*,*subst* (*2*) *prod.collapse*[*symmetric*])
apply(*subst remove-sym*)
using *g-expr e-proj-2-3 e-proj-2-3-g* **apply**(*simp*,*simp*,*simp*)
apply(*subst remove-add-sym*)
using *e-proj-2-3 e-proj-2-3-g rot* **by** *auto*
also have ... = *tf ′′* (*τ* ∘ *g*) {((*1*,*0*), *0*)}
apply(*subst proj-addition-comm*)
using *add-2-3-def e-proj-2-3*(*1*) *proj-add-class-inv* **by** *auto*
finally have *eq1*: *proj-addition ?g1* (*proj-addition ?g2 ?g3*) =
tf ′′ (*τ* ∘ *g*) {((*1*,*0*), *0*)}
by *auto*

have *proj-addition* (*proj-addition ?g1 ?g2*) *?g3* =
proj-addition (*proj-addition* (*gluing* ‘‘ {((*g* ∘ *i*) *add-2-3*, *0*)}) *?g2*) *?g3*
using *g-expr* **by** *argo*
also have ... = *proj-addition* (*tf ′′* (*τ* ∘ *g*)
(*proj-addition* (*gluing* ‘‘ {(*i add-2-3*, *0*)}) *?g2*)) *?g3*
apply(*subst comp-apply*,*subst* (*2*) *prod.collapse*[*symmetric*])
apply(*subst remove-sym*)
using *g-expr e-proj-2-3 e-proj-2-3-g* **apply**(*simp*,*simp*,*simp*)
apply(*subst remove-add-sym*)

using *e-proj-2-3 e-proj-2-3-g assms*(*2*) *rot* **by** *auto*
also have ... = *proj-addition* (*tf''* ($\tau \circ g$)
(*proj-addition* (*proj-addition ?ig2 ?ig3*) *?g2*)) *?g3*
unfolding *add-2-3-def*
apply(*subst inverse-rule-3*)
using *gluing-add e-proj-0 p-delta* **by** *force*
also have ... = *proj-addition* (*tf''* ($\tau \circ g$) *?ig3*) *?g3*
using *cancellation-assoc*
proof −
have *proj-addition ?g2* (*proj-addition ?ig3 ?ig2*) = *?ig3*
by (*metis* (*no-types*, *lifting*) *assms*(*2*) *cancellation-assoc e-proj-0*(*2*) *e-proj-0*(*3*) *i.simps i-idemp-explicit proj-addition-comm well-defined*)
then show *?thesis*
using *assms*(*2*) *e-proj-0*(*2*) *e-proj-0*(*3*) *proj-addition-comm well-defined* **by** *auto*
qed
also have ... = *tf''* ($\tau \circ g$) (*proj-addition ?ig3 ?g3*)
apply(*subst remove-add-sym*)
using *assms*(*3*) *rot e-proj-0*(*3*) **by** *auto*
also have ... = *tf''* ($\tau \circ g$) {((*1*,*0*), *0*)}
apply(*subst proj-addition-comm*)
using *assms*(*3*) *proj-add-class-inv* **by** *auto*
finally have *eq2*: *proj-addition* (*proj-addition ?g1 ?g2*) *?g3* =
tf'' ($\tau \circ g$) {((*1*,*0*), *0*)} **by** *blast*
show *?thesis* **using** *eq1 eq2* **by** *argo*
qed

lemma *assoc-111-ext-add*:
assumes *gluing* `` {((*x1*, *y1*), *0*)} ∈ *e-proj gluing* `` {((*x2*, *y2*), *0*)} ∈ *e-proj gluing* `` {((*x3*, *y3*), *0*)} ∈ *e-proj*
assumes *22*: *g*∈*symmetries* (*x1*, *y1*) = (*g* ∘ *i*) (*ext-add* (*x2*,*y2*) (*x3*,*y3*)) ((*x2*, *y2*), *x3*, *y3*) ∈ *e'-aff-1*
shows
proj-addition (*proj-addition* (*gluing* `` {((*x1*, *y1*), *0*)}) (*gluing* `` {((*x2*, *y2*), *0*)})) (*gluing* `` {((*x3*, *y3*), *0*)}) =
proj-addition (*gluing* `` {((*x1*, *y1*), *0*)}) (*proj-addition* (*gluing* `` {((*x2*, *y2*), *0*)}) (*gluing* `` {((*x3*, *y3*), *0*)}))
(**is** *proj-addition* (*proj-addition ?g1 ?g2*) *?g3* = -)
proof −
have *in-aff*: (*x1*,*y1*) ∈ *e'-aff* (*x2*,*y2*) ∈ *e'-aff* (*x3*,*y3*) ∈ *e'-aff*
using *assms*(*1*,*2*,*3*) *e-class* **by** *auto*

have *one-in*: *gluing* `` {((*1*, *0*), *0*)} ∈ *e-proj*
using *identity-equiv identity-proj* **by** *force*

have *e-proj-0*: *gluing* `` {(*i* (*x1*,*y1*), *0*)} ∈ *e-proj* (**is** *?ig1* ∈ *e-proj*)
gluing `` {(*i* (*x2*,*y2*), *0*)} ∈ *e-proj* (**is** *?ig2* ∈ *e-proj*)
gluing `` {(*i* (*x3*,*y3*), *0*)} ∈ *e-proj* (**is** *?ig3* ∈ *e-proj*)
using *assms proj-add-class-inv* **by** *blast+*

have *p-delta*: *delta′ x2 y2 x3 y3* ≠ *0*
delta′ (*fst* (*i* (*x2*,*y2*))) (*snd* (*i* (*x2*,*y2*)))
(*fst* (*i* (*x3*,*y3*))) (*snd* (*i* (*x3*,*y3*))) ≠ *0*
using *22* **unfolding** *e′-aff-1-def* **apply** *simp*
using *22* **unfolding** *e′-aff-1-def delta′-def delta-x-def delta-y-def* **by** *force*

define *add-2-3* **where** *add-2-3* = *ext-add* (*x2*,*y2*) (*x3*,*y3*)
have *add-in*: *add-2-3* ∈ *e′-aff*
unfolding *e′-aff-def add-2-3-def*
apply(*simp del*: *ext-add.simps*)
apply(*subst* (*2*) *prod.collapse*[*symmetric*])
apply(*standard*)
apply(*subst ext-add-closure*)
using *in-aff 22* **unfolding** *e′-aff-def e′-aff-1-def* **by**(*fastforce*)+

have *e-proj-2-3*: *gluing* ‘‘ {(*add-2-3*, *0*)} ∈ *e-proj*
gluing ‘‘ {(*i add-2-3*, *0*)} ∈ *e-proj*
using *add-in add-2-3-def e-points* **apply** *simp*
using *add-in add-2-3-def e-points proj-add-class-inv* **by** *force*

from *22* **have** *g-expr*: *g* ∈ *symmetries* (*x1*,*y1*) = (*g* ∘ *i*) *add-2-3* **unfolding** *add-2-3-def* **by** *auto*
then have *rot*: *τ* ∘ *g* ∈ *rotations* **using** *sym-to-rot* **by** *blast*

have *e-proj-2-3-g*: *gluing* ‘‘ {(*g* (*i add-2-3*), *0*)} ∈ *e-proj*
using *e-proj-2-3 g-expr assms*(*1*) **by** *auto*

have *proj-addition ?g1* (*proj-addition ?g2 ?g3*) =
proj-addition (*gluing* ‘‘ {((*g* ∘ *i*) *add-2-3*, *0*)}) (*proj-addition ?g2 ?g3*)
using *g-expr* **by** *simp*
also have ... = *proj-addition* (*gluing* ‘‘ {((*g* ∘ *i*) *add-2-3*, *0*)}) (*gluing* ‘‘ {(*add-2-3*, *0*)})
using *gluing-ext-add add-2-3-def p-delta assms*(*2*,*3*) **by** *force*
also have ... = *tf′′* (*τ* ∘ *g*) (*proj-addition* (*gluing* ‘‘ {(*i add-2-3*, *0*)}) (*gluing* ‘‘ {(*add-2-3*, *0*)}))
apply(*subst comp-apply*,*subst* (*2*) *prod.collapse*[*symmetric*])
apply(*subst remove-sym*)
using *g-expr e-proj-2-3 e-proj-2-3-g* **apply**(*simp*,*simp*,*simp*)
apply(*subst remove-add-sym*)
using *e-proj-2-3 e-proj-2-3-g rot* **by** *auto*
also have ... = *tf′′* (*τ* ∘ *g*) {((*1*,*0*), *0*)}
apply(*subst proj-addition-comm*)
using *add-2-3-def e-proj-2-3*(*1*) *proj-add-class-inv* **by** *auto*
finally have *eq1*: *proj-addition ?g1* (*proj-addition ?g2 ?g3*) =
tf′′ (*τ* ∘ *g*) {((*1*,*0*), *0*)}
by *auto*

have *proj-addition* (*proj-addition* *?g1* *?g2*) *?g3* =
proj-addition (*proj-addition* (*gluing* ‘‘ {((*g* ∘ *i*) *add-2-3*, *0*)}) *?g2*) *?g3*
using *g-expr* **by** *argo*
also have ... = *proj-addition* (*tf* ′′ (τ ∘ *g*)
(*proj-addition* (*gluing* ‘‘ {(*i* *add-2-3*, *0*)}) *?g2*)) *?g3*
apply(*subst* *comp-apply*,*subst* (*2*) *prod.collapse*[*symmetric*])
apply(*subst* *remove-sym*)
using *g-expr* *e-proj-2-3* *e-proj-2-3-g* **apply**(*simp*,*simp*,*simp*)
apply(*subst* *remove-add-sym*)
using *e-proj-2-3* *e-proj-2-3-g* *assms*(*2*) *rot* **by** *auto*
also have ... = *proj-addition* (*tf* ′′ (τ ∘ *g*)
(*proj-addition* (*proj-addition* *?ig2* *?ig3*) *?g2*)) *?g3*
unfolding *add-2-3-def*
apply(*subst* *inverse-rule-4*)
using *gluing-ext-add* *e-proj-0* *p-delta* **by** *force*
also have ... = *proj-addition* (*tf* ′′ (τ ∘ *g*) *?ig3*) *?g3*
proof −
have *proj-addition* *?g2* (*proj-addition* *?ig3* *?ig2*) = *?ig3*
apply(*subst* *proj-addition-comm*)
using *assms* *e-proj-0* *well-defined* **apply**(*simp*,*simp*)
apply(*subst* *cancellation-assoc*[*of* *fst* (*i* (*x3*,*y3*)) *snd* (*i* (*x3*,*y3*))
fst (*i* (*x2*,*y2*)) *snd* (*i* (*x2*,*y2*)),
simplified *prod.collapse* *i-idemp-explicit*])
using *assms* *e-proj-0* **by** *auto*
then show *?thesis*
using *assms*(*2*) *e-proj-0*(*2*) *e-proj-0*(*3*) *proj-addition-comm* *well-defined* **by** *auto*
qed
also have ... = *tf* ′′ (τ ∘ *g*) (*proj-addition* *?ig3* *?g3*)
apply(*subst* *remove-add-sym*)
using *assms*(*3*) *rot* *e-proj-0*(*3*) **by** *auto*
also have ... = *tf* ′′ (τ ∘ *g*) {((*1*,*0*), *0*)}
using *assms*(*3*) *proj-add-class-inv* *proj-addition-comm* **by** *auto*
finally have *eq2*: *proj-addition* (*proj-addition* *?g1* *?g2*) *?g3* =
tf ′′ (τ ∘ *g*) {((*1*,*0*), *0*)} **by** *blast*

show *?thesis* **using** *eq1* *eq2* **by** *argo*
qed

lemma *assoc-with-zeros*:
assumes *gluing* ‘‘ {((*x1*, *y1*), *0*)} ∈ *e-proj*
gluing ‘‘ {((*x2*, *y2*), *0*)} ∈ *e-proj*
gluing ‘‘ {((*x3*, *y3*), *0*)} ∈ *e-proj*
shows *proj-addition* (*proj-addition* (*gluing* ‘‘ {((*x1*, *y1*), *0*)}) (*gluing* ‘‘ {((*x2*, *y2*), *0*)})) (*gluing* ‘‘ {((*x3*, *y3*), *0*)}) =
proj-addition (*gluing* ‘‘ {((*x1*, *y1*), *0*)}) (*proj-addition* (*gluing* ‘‘ {((*x2*, *y2*), *0*)}) (*gluing* ‘‘ {((*x3*, *y3*), *0*)}))
(**is** *proj-addition* (*proj-addition* *?g1* *?g2*) *?g3* =
proj-addition *?g1* (*proj-addition* *?g2* *?g3*))

```
proof -
  have in-aff: (x1,y1) ∈ e'-aff (x2,y2) ∈ e'-aff (x3,y3) ∈ e'-aff
    using assms(1,2,3) e-class by auto

  have e-proj-0: gluing `` {(i (x1,y1), 0)} ∈ e-proj (is ?ig1 ∈ e-proj)
                 gluing `` {(i (x2,y2), 0)} ∈ e-proj (is ?ig2 ∈ e-proj)
                 gluing `` {(i (x3,y3), 0)} ∈ e-proj (is ?ig3 ∈ e-proj)
    using assms proj-add-class-inv by auto

  consider
    (1) (∃ g∈symmetries. (x2, y2) = (g ∘ i) (x1, y1)) |
    (2) ((x1, y1), x2, y2) ∈ e'-aff-0 ¬ ((∃ g∈symmetries. (x2, y2) = (g ∘ i) (x1,
y1))) |
    (3) ((x1, y1), x2, y2) ∈ e'-aff-1 ¬ ((∃ g∈symmetries. (x2, y2) = (g ∘ i) (x1,
y1))) ((x1, y1), x2, y2) ∉ e'-aff-0
    using dichotomy-1 in-aff by blast
  then show ?thesis
  proof(cases)
    case 1 then show ?thesis using assoc-1(2,3) assms by force
  next
    case 2
    have p-delta-1-2: delta x1 y1 x2 y2 ≠ 0
                      delta (fst (i (x1, y1))) (snd (i (x1, y1)))
                            (fst (i (x2, y2))) (snd (i (x2, y2))) ≠ 0
        using 2 unfolding e'-aff-0-def apply simp
       using 2 in-aff unfolding e'-aff-0-def delta-def delta-minus-def delta-plus-def

        by auto

    define add-1-2 where add-1-2 = add (x1, y1) (x2, y2)
    have add-in-1-2: add-1-2 ∈ e'-aff
      unfolding e'-aff-def add-1-2-def
      apply(simp del: add.simps)
      apply(subst (2) prod.collapse[symmetric])
      apply(standard)
      apply(simp add: e-e'-iff[symmetric] del: add.simps)
      apply(subst add-closure)
      using in-aff p-delta-1-2(1) e-e'-iff
      unfolding delta-def e'-aff-def by(blast,(simp)+)

    have e-proj-1-2: gluing `` {(add-1-2, 0)} ∈ e-proj
                     gluing `` {(i add-1-2, 0)} ∈ e-proj
      using add-in-1-2 add-1-2-def e-points apply simp
      using add-in-1-2 add-1-2-def e-points proj-add-class-inv by force

    consider
      (11) (∃ g∈symmetries. (x3, y3) = (g ∘ i) (x2, y2)) |
      (22) ((x2, y2), (x3, y3)) ∈ e'-aff-0 ¬ ((∃ g∈symmetries. (x3, y3) = (g ∘ i)
(x2, y2))) |
```

(33) $((x2,\ y2),\ (x3,\ y3)) \in e'$-*aff-1* $\neg$ $((\exists\, g{\in}symmetries.\ (x3,\ y3) = (g \circ i)\ (x2,\ y2)))$ $((x2,\ y2),\ (x3,\ y3)) \notin e'$-*aff-0*
using *dichotomy-1 in-aff* **by** *blast*
then show *?thesis*
proof(*cases*)
case *11*
then obtain *g* **where** *g-expr*: $g \in symmetries$ $(x3,\ y3) = (g \circ i)\ (x2,\ y2)$ **by** *blast*
then show *?thesis* **using** *assoc-11 assms* **by** *force*
next
case *22*
have *p-delta-2-3*: *delta x2 y2 x3 y3* $\neq 0$
delta (*fst* (*i* (*x2*,*y2*))) (*snd* (*i* (*x2*,*y2*)))
(*fst* (*i* (*x3*,*y3*))) (*snd* (*i* (*x3*,*y3*))) $\neq 0$
using *22* **unfolding** *e'-aff-0-def* **apply** *simp*
using *22* **unfolding** *e'-aff-0-def delta-def delta-plus-def delta-minus-def* **by** *simp*

define *add-2-3* **where** *add-2-3* = *add* (*x2*,*y2*) (*x3*,*y3*)
have *add-in*: *add-2-3* $\in$ *e'-aff*
unfolding *e'-aff-def add-2-3-def*
apply(*simp del*: *add.simps*)
apply(*subst* (*2*) *prod.collapse*[*symmetric*])
apply(*standard*)
apply(*simp del*: *add.simps add*: *e-e'-iff*[*symmetric*])
apply(*subst add-closure*)
using *in-aff e-e'-iff 22* **unfolding** *e'-aff-def e'-aff-0-def delta-def* **by**(*fastforce*)+
have *e-proj-2-3*: *gluing* ‘‘ {(*add-2-3*, *0*)} $\in$ *e-proj*
gluing ‘‘ {(*i add-2-3*, *0*)} $\in$ *e-proj*
using *add-in add-2-3-def e-points* **apply** *simp*
using *add-in add-2-3-def e-points proj-add-class-inv* **by** *force*

consider
(*111*) $(\exists\, g{\in}symmetries.\ (x1,y1) = (g \circ i)$ *add-2-3*) |
(*222*) (*add-2-3*, (*x1*,*y1*)) $\in$ *e'-aff-0* $\neg$ $((\exists\, g{\in}symmetries.\ (x1,y1) = (g \circ i)$ *add-2-3*)) |
(*333*) (*add-2-3*, (*x1*,*y1*)) $\in$ *e'-aff-1* $\neg$ $((\exists\, g{\in}symmetries.\ (x1,y1) = (g \circ i)$ *add-2-3*)) (*add-2-3*, (*x1*,*y1*)) $\notin$ *e'-aff-0*
using *add-in in-aff dichotomy-1* **by** *blast*
then show *?thesis*
proof(*cases*)
case *111*
then show *?thesis* **using** *assoc-111-add* **using** *22*(*1*) *add-2-3-def assms*(*1*) *assms*(*2*) *assms*(*3*) **by** *blast*
next
case *222*
have *assumps*: ((*x1*, *y1*),*add-2-3*) $\in$ *e'-aff-0*
apply(*subst* (*3*) *prod.collapse*[*symmetric*])
using *222 e'-aff-0-invariance* **by** *fastforce*

```
    consider
      (1111) (∃ g∈symmetries. (x3,y3) = (g ∘ i) add-1-2) |
      (2222) (add-1-2, (x3,y3)) ∈ e′-aff-0 ¬ ((∃ g∈symmetries. (x3,y3) = (g ∘
i) add-1-2)) |
      (3333) (add-1-2, (x3,y3)) ∈ e′-aff-1 ¬ ((∃ g∈symmetries. (x3,y3) = (g ∘
i) add-1-2)) (add-1-2, (x3,y3)) ∉ e′-aff-0
      using add-in-1-2 in-aff dichotomy-1 by blast
    then show ?thesis
    proof(cases)
      case 1111
     then obtain g where g-expr: g ∈ symmetries (x3, y3) = (g ∘ i) add-1-2
by blast
      then have rot: τ ∘ g ∈ rotations using sym-to-rot assms by blast

      have proj-addition (proj-addition ?g1 ?g2) ?g3 =
            proj-addition (gluing ‘‘ {(add-1-2, 0)}) (gluing ‘‘ {((g ∘ i) add-1-2,
0)})
        using g-expr p-delta-1-2 gluing-add assms(1,2) add-1-2-def by force
      also have ... = tf′′ (τ ∘ g) ({((1, 0), 0)})
        apply(subst proj-addition-comm)
        using e-proj-1-2(1) g-expr(2) assms(3) apply(simp,simp)
        apply(subst comp-apply,subst (2) prod.collapse[symmetric])
        apply(subst remove-sym)
        using e-proj-1-2(2) g-expr assms(3) apply(simp,simp,simp)
        apply(subst remove-add-sym)
        using e-proj-1-2 rot apply(simp,simp,simp)
        apply(subst prod.collapse, subst (2 4) prod.collapse[symmetric])
        by (metis cancellation-assoc e-proj-1-2(1) e-proj-1-2(2) identity-equiv
          identity-proj prod.collapse proj-add-class-identity proj-addition-comm)
      finally have eq1: proj-addition (proj-addition ?g1 ?g2) ?g3 =
                     tf′′ (τ ∘ g) ({((1, 0), 0)}) by blast

      have proj-addition ?g1 (proj-addition ?g2 ?g3) =
          proj-addition ?g1 (proj-addition ?g2 (gluing ‘‘ {((g ∘ i) add-1-2, 0)}))

        using g-expr by auto
      also have ... = proj-addition ?g1
                     (tf′′ (τ ∘ g)
                        (proj-addition (gluing ‘‘ {(add (i (x1, y1)) (i (x2, y2)),
0)})
                      ?g2))
        apply(subst comp-apply,subst (6) prod.collapse[symmetric])
        apply(subst (3) remove-sym)
        using e-proj-1-2(2) g-expr assms(3) apply(simp,simp,simp)
        apply(subst prod.collapse)
        apply(subst (2) proj-addition-comm)
        using assms(2) apply simp
        using tf′′-preserv-e-proj rot e-proj-1-2(2) apply (metis prod.collapse)
```

apply(*subst remove-add-sym*)
using *assms*(*2*) *e-proj-1-2*(*2*) *rot* **apply**(*simp*,*simp*,*simp*)
unfolding *add-1-2-def*
by(*subst inverse-rule-3*,*blast*)
also have ... = *proj-addition ?g1* (*tf''* ($\tau \circ g$)
(*proj-addition* (*proj-addition ?ig1 ?ig2*)
?g2))
proof −
have *gluing* ‘‘ {(*add* (*i* (*x1*, *y1*)) (*i* (*x2*, *y2*)), *0*)} =
proj-addition ?ig1 ?ig2
using *gluing-add*[*symmetric*,*of fst* (*i* (*x1*,*y1*)) *snd* (*i* (*x1*,*y1*)) *0*
fst (*i* (*x2*,*y2*)) *snd* (*i* (*x2*,*y2*)) *0*,
simplified prod.collapse] *e-proj-0*(*1*,*2*) *p-delta-1-2*(*2*)
by *simp*
then show *?thesis* **by** *presburger*
qed
also have ... = *proj-addition ?g1* (*tf''* ($\tau \circ g$) *?ig1*)
using *cancellation-assoc*
by (*metis assms*(*2*) *e-proj-0*(*1*) *e-proj-0*(*2*) *i.simps i-idemp-explicit*)
also have ... = *tf''* ($\tau \circ g$) (*proj-addition ?g1 ?ig1*)
using *assms*(*1*) *e-proj-0*(*1*) *proj-addition-comm remove-add-sym rot*
tf''-preserv-e-proj **by** *fastforce*
also have ... = *tf''* ($\tau \circ g$) ({((*1*, *0*), *0*)})
using *assms*(*1*) *proj-add-class-comm proj-add-class-inv* **by** *simp*
finally have *eq2*: *proj-addition ?g1* (*proj-addition ?g2 ?g3*) =
tf'' ($\tau \circ g$) ({((*1*, *0*), *0*)}) **by** *auto*
then show *?thesis*
using *eq1 eq2* **by** *blast*
next
case *2222*
have *proj-addition* (*proj-addition ?g1 ?g2*) *?g3* =
proj-addition (*gluing* ‘‘ {(*add* (*x1*, *y1*) (*x2*, *y2*), *0*)}) *?g3*
using *gluing-add p-delta-1-2*(*1*) *e-proj-1-2 add-1-2-def assms*(*1*,*2*) **by**
simp
also have ... = *gluing* ‘‘ {(*add* (*add* (*x1*, *y1*) (*x2*, *y2*)) (*x3*, *y3*), *0*)}
apply(*subst* (*2*) *prod.collapse*[*symmetric*])
apply(*subst gluing-add*)
apply(*subst prod.collapse*)
using *gluing-ext-add p-delta-1-2*(*1*) *e-proj-1-2 add-1-2-def assms*(*1*,*2*,*3*)
apply(*simp*,*simp*)
using *2222* **unfolding** *e'-aff-0-def add-1-2-def* **by**(*simp*,*force*)
also have ... = *gluing* ‘‘ {(*add* (*x1*, *y1*) (*add* (*x2*, *y2*) (*x3*, *y3*)), *0*)}
apply(*subst add-add-add-add-assoc*)
using *p-delta-1-2 p-delta-2-3*(*1*) *2222*(*1*) *assumps in-aff*
unfolding *e'-aff-0-def e'-aff-1-def delta-def delta'-def*
add-1-2-def add-2-3-def e'-aff-def
by *auto*
also have ... = *proj-addition ?g1* (*gluing* ‘‘ {(*add* (*x2*, *y2*) (*x3*, *y3*), *0*)})
apply(*subst* (*10*) *prod.collapse*[*symmetric*])

apply(*subst gluing-add*)
using *assms*(*1*) *e-proj-2-3*(*1*) *add-2-3-def assumps*
unfolding *e′-aff-0-def* **by**(*simp*,*simp*,*force*,*simp*)
also have ... = *proj-addition ?g1* (*proj-addition ?g2 ?g3*)
apply(*subst gluing-add*)
using *assms*(*2*,*3*) *p-delta-2-3*(*1*) **by** *auto*
finally show *?thesis* **by** *blast*
next
case *3333*

have *proj-addition* (*proj-addition ?g1 ?g2*) *?g3* =
proj-addition (*gluing* ‘‘ {(*add* (*x1*, *y1*) (*x2*, *y2*), *0*)}) *?g3*
using *gluing-add p-delta-1-2*(*1*) *e-proj-1-2 add-1-2-def assms*(*1*,*2*) **by** *simp*
also have ... = *gluing* ‘‘ {(*ext-add* (*add* (*x1*, *y1*) (*x2*, *y2*)) (*x3*, *y3*), *0*)}
apply(*subst* (*2*) *prod.collapse*[*symmetric*])
apply(*subst gluing-ext-add*)
apply(*subst prod.collapse*)
using *gluing-ext-add p-delta-1-2*(*1*) *e-proj-1-2 add-1-2-def assms*(*1*,*2*,*3*) **apply**(*simp*,*simp*)
using *3333* **unfolding** *e′-aff-1-def add-1-2-def* **by**(*simp*,*force*)
also have ... = *gluing* ‘‘ {(*add* (*x1*, *y1*) (*add* (*x2*, *y2*) (*x3*, *y3*)), *0*)}
apply(*subst ext-add-add-add-assoc*)
apply(*simp*,*simp*)
apply(*subst prod.collapse*[*symmetric*],*subst prod.inject*,*fast*)+
using *p-delta-1-2 p-delta-2-3*(*1*) *3333*(*1*) *assumps in-aff*
unfolding *e′-aff-0-def e′-aff-1-def delta-def delta′-def*
add-1-2-def add-2-3-def e′-aff-def
by *auto*
also have ... = *proj-addition ?g1*
(*gluing* ‘‘ {(*add* (*x2*, *y2*) (*x3*, *y3*), *0*)})
apply(*subst* (*10*) *prod.collapse*[*symmetric*])
apply(*subst gluing-add*)
using *assms*(*1*) *e-proj-2-3*(*1*) *add-2-3-def assumps*
unfolding *e′-aff-0-def* **by**(*simp*,*simp*,*force*,*simp*)
also have ... = *proj-addition ?g1* (*proj-addition ?g2 ?g3*)
apply(*subst gluing-add*)
using *assms*(*2*,*3*) *p-delta-2-3*(*1*) **by** *auto*
finally show *?thesis* **by** *blast*
qed
next
case *333*
have *assumps*: ((*x1*, *y1*),*add-2-3*) ∈ *e′-aff-1*
using *333*(*1*) *e′-aff-1-invariance add-2-3-def* **by** *auto*

consider
(*1111*) (∃ *g*∈*symmetries*. (*x3*,*y3*) = (*g* ∘ *i*) *add-1-2*) |
(*2222*) (*add-1-2*, (*x3*,*y3*)) ∈ *e′-aff-0* ¬ ((∃ *g*∈*symmetries*. (*x3*,*y3*) = (*g* ∘ *i*) *add-1-2*)) |

(3333) (*add-1-2*, (*x3*,*y3*)) ∈ *e′-aff-1* ¬ ((∃ *g*∈*symmetries*. (*x3*,*y3*) = (*g* ∘ *i*) *add-1-2*)) (*add-1-2*, (*x3*,*y3*)) ∉ *e′-aff-0*
using *add-in-1-2 in-aff dichotomy-1* **by** *blast*
then show *?thesis*
proof(*cases*)
case *1111*
then obtain *g* **where** *g-expr*: *g* ∈ *symmetries* (*x3*, *y3*) = (*g* ∘ *i*) *add-1-2* **by** *blast*
then have *rot*: τ ∘ *g* ∈ *rotations* **using** *sym-to-rot assms* **by** *blast*

have *proj-addition* (*proj-addition ?g1 ?g2*) *?g3* =
proj-addition (*gluing* ‘‘ {(*add-1-2*, *0*)}) (*gluing* ‘‘ {((*g* ∘ *i*) *add-1-2*, *0*)})
using *g-expr p-delta-1-2 gluing-add assms*(*1*,*2*) *add-1-2-def* **by** *force*
also have ... = *tf′′* (τ ∘ *g*) {((*1*, *0*), *0*)}
apply(*subst proj-addition-comm*)
using *e-proj-1-2*(*1*) *g-expr*(*2*) *assms*(*3*) **apply**(*simp*,*simp*)
apply(*subst comp-apply*,*subst* (*2*) *prod.collapse*[*symmetric*])
apply(*subst remove-sym*)
using *e-proj-1-2*(*2*) *g-expr assms*(*3*) **apply**(*simp*,*simp*,*simp*)
apply(*subst remove-add-sym*)
using *e-proj-1-2 rot* **apply**(*simp*,*simp*,*simp*)
apply(*subst prod.collapse*, *subst* (*2 4*) *prod.collapse*[*symmetric*])
by (*metis* (*no-types*, *lifting*) *cancellation-assoc e-proj-1-2*(*1*) *e-proj-1-2*(*2*) *identity-equiv*
identity-proj prod.collapse proj-add-class-identity proj-addition-comm)
finally have *eq1*: *proj-addition* (*proj-addition ?g1 ?g2*) *?g3* =
tf′′ (τ ∘ *g*) {((*1*, *0*), *0*)} **by** *blast*

have *proj-addition ?g1* (*proj-addition ?g2 ?g3*) =
proj-addition ?g1 (*proj-addition ?g2* (*gluing* ‘‘ {((*g* ∘ *i*) *add-1-2*, *0*)}))

using *g-expr* **by** *auto*
also have ... = *proj-addition ?g1*
(*tf′′* (τ ∘ *g*)
(*proj-addition* (*gluing* ‘‘ {(*add* (*i* (*x1*, *y1*)) (*i* (*x2*, *y2*)), *0*)})
?g2))
apply(*subst comp-apply*,*subst* (*6*) *prod.collapse*[*symmetric*])
apply(*subst* (*3*) *remove-sym*)
using *e-proj-1-2*(*2*) *g-expr assms*(*3*) **apply**(*simp*,*simp*,*simp*)
apply(*subst prod.collapse*)
apply(*subst* (*2*) *proj-addition-comm*)
using *assms*(*2*) **apply** *simp*
using *tf′′-preserv-e-proj rot e-proj-1-2*(*2*)
apply (*metis prod.collapse*)
apply(*subst remove-add-sym*)
using *assms*(*2*) *e-proj-1-2*(*2*) *rot* **apply**(*simp*,*simp*,*simp*)
unfolding *add-1-2-def*

by(*subst inverse-rule-3*,*blast*)
also have ... = *proj-addition ?g1* (*tf''* (τ ∘ *g*)
(*proj-addition* (*proj-addition ?ig1 ?ig2*) *?g2*))
proof −
have *gluing* `` {(*add* (*i* (*x1*, *y1*)) (*i* (*x2*, *y2*)), *0*)} =
proj-addition ?ig1 ?ig2
using *gluing-add*[*symmetric*, *of fst* (*i* (*x1*,*y1*)) *snd* (*i* (*x1*,*y1*)) *0*
fst (*i* (*x2*, *y2*)) *snd* (*i* (*x2*, *y2*)) *0*,
simplified prod.collapse] *e-proj-0*(*1*,*2*) *p-delta-1-2*(*2*)
by *simp*
then show *?thesis* **by** *presburger*
qed
also have ... = *proj-addition ?g1* (*tf''* (τ ∘ *g*) *?ig1*)
using *cancellation-assoc*
by (*metis assms*(*2*) *e-proj-0*(*1*) *e-proj-0*(*2*) *i.simps i-idemp-explicit*)
also have ... = *tf''* (τ ∘ *g*) (*proj-addition ?g1 ?ig1*)
using *assms*(*1*) *e-proj-0*(*1*) *proj-addition-comm remove-add-sym rot*
tf''-preserv-e-proj **by** *fastforce*
also have ... = *tf''* (τ ∘ *g*) {((*1*, *0*), *0*)}
using *assms*(*1*) *proj-add-class-comm proj-addition-def proj-add-class-inv*
by *simp*
finally have *eq2*: *proj-addition ?g1* (*proj-addition ?g2 ?g3*) =
tf'' (τ ∘ *g*) {((*1*, *0*), *0*)} **by** *auto*
then show *?thesis* **using** *eq1 eq2* **by** *blast*
next
case *2222*

have *proj-addition* (*proj-addition ?g1 ?g2*) *?g3* =
proj-addition (*gluing* `` {(*add* (*x1*, *y1*) (*x2*, *y2*), *0*)}) *?g3*
using *gluing-add p-delta-1-2*(*1*) *e-proj-1-2 add-1-2-def assms*(*1*,*2*) **by**
simp
also have ... = *gluing* `` {(*add* (*add* (*x1*, *y1*) (*x2*, *y2*)) (*x3*, *y3*), *0*)}
apply(*subst* (*2*) *prod.collapse*[*symmetric*])
apply(*subst gluing-add*)
apply(*subst prod.collapse*)
using *gluing-add p-delta-1-2*(*1*) *e-proj-1-2 add-1-2-def assms*(*1*,*2*,*3*)
apply(*simp*,*simp*)
using *2222* **unfolding** *e'-aff-0-def add-1-2-def* **by**(*simp*,*force*)
also have ... = *gluing* `` {(*ext-add* (*x1*, *y1*) (*add* (*x2*, *y2*) (*x3*, *y3*)), *0*)}
apply(*subst add-add-ext-add-assoc*)
apply(*simp*,*simp*)
apply(*subst prod.collapse*[*symmetric*],*subst prod.inject*,*fast*)+
using *p-delta-1-2 p-delta-2-3*(*1*) *2222*(*1*) *assumps in-aff*
unfolding *e'-aff-0-def e'-aff-1-def delta-def delta'-def*
add-1-2-def add-2-3-def e'-aff-def
by *force*+
also have ... = *proj-addition ?g1* (*gluing* `` {(*add* (*x2*, *y2*) (*x3*, *y3*), *0*)})
apply(*subst* (*10*) *prod.collapse*[*symmetric*])
apply(*subst gluing-ext-add*)

using *assms*(*1*) *e-proj-2-3*(*1*) *add-2-3-def assumps*
unfolding *e'-aff-1-def* **by**(*blast*,*auto*)
also have ... = *proj-addition ?g1* (*proj-addition ?g2 ?g3*)
apply(*subst gluing-add*)
using *assms*(*2*,*3*) *p-delta-2-3*(*1*) **by** *auto*
finally show *?thesis* **by** *blast*
next
case *3333*
have *proj-addition* (*proj-addition ?g1 ?g2*) *?g3* =
proj-addition (*gluing* `` {(*add* (*x1*, *y1*) (*x2*, *y2*), *0*)}) *?g3*
using *gluing-add p-delta-1-2*(*1*) *e-proj-1-2 add-1-2-def assms*(*1*,*2*) **by** *simp*
also have ... = *gluing* `` {(*ext-add* (*add* (*x1*, *y1*) (*x2*, *y2*)) (*x3*, *y3*), *0*)}
apply(*subst* (*2*) *prod.collapse*[*symmetric*])
apply(*subst gluing-ext-add*)
apply(*subst prod.collapse*)
using *gluing-add p-delta-1-2*(*1*) *e-proj-1-2 add-1-2-def assms*(*1*,*2*,*3*) **apply**(*simp*,*simp*)
using *3333* **unfolding** *e'-aff-1-def add-1-2-def* **by**(*simp*,*force*)
also have ... = *gluing* `` {(*ext-add* (*x1*, *y1*) (*add* (*x2*, *y2*) (*x3*, *y3*)), *0*)}
apply(*subst ext-add-ext-add-assoc*)
apply(*simp*,*simp*)
apply(*subst prod.collapse*[*symmetric*],*subst prod.inject*,*fast*)+
using *p-delta-1-2 p-delta-2-3*(*1*) *3333*(*1*) *assumps in-aff*
unfolding *e'-aff-0-def e'-aff-1-def delta-def delta'-def*
add-1-2-def add-2-3-def e'-aff-def
by(*force*)+
also have ... = *proj-addition ?g1* (*gluing* `` {(*add* (*x2*, *y2*) (*x3*, *y3*), *0*)})
apply(*subst* (*10*) *prod.collapse*[*symmetric*])
apply(*subst gluing-ext-add*)
using *assms*(*1*) *e-proj-2-3*(*1*) *add-2-3-def assumps*
unfolding *e'-aff-1-def* **by**(*simp*,*simp*,*force*,*simp*)
also have ... = *proj-addition ?g1* (*proj-addition ?g2 ?g3*)
apply(*subst gluing-add*)
using *assms*(*2*,*3*) *p-delta-2-3*(*1*) **by** *auto*
finally show *?thesis* **by** *blast*
qed
qed
next
case *33*
have *p-delta-2-3*: *delta' x2 y2 x3 y3* ≠ *0*
delta' (*fst* (*i* (*x2*,*y2*))) (*snd* (*i* (*x2*,*y2*)))
(*fst* (*i* (*x3*,*y3*))) (*snd* (*i* (*x3*,*y3*))) ≠ *0*
using *33* **unfolding** *e'-aff-1-def* **apply** *simp*
using *33* **unfolding** *e'-aff-1-def delta'-def delta-x-def delta-y-def* **by** *force*

define *add-2-3* **where** *add-2-3* = *ext-add* (*x2*,*y2*) (*x3*,*y3*)
have *add-in*: *add-2-3* ∈ *e'-aff*
unfolding *e'-aff-def add-2-3-def*

```
      apply(simp del: ext-add.simps)
      apply(subst (2) prod.collapse[symmetric])
      apply(standard)
      apply(subst ext-add-closure)
    using in-aff e-e′-iff 33 unfolding e′-aff-def e′-aff-1-def delta′-def by(fastforce)+
    have e-proj-2-3: gluing ‘‘ {(add-2-3, 0)} ∈ e-proj
                 gluing ‘‘ {(i add-2-3, 0)} ∈ e-proj
      using add-in add-2-3-def e-points apply simp
      using add-in add-2-3-def e-points proj-add-class-inv by force

    consider
      (111) (∃ g∈symmetries. (x1,y1) = (g ∘ i) add-2-3) |
      (222) (add-2-3, (x1,y1)) ∈ e′-aff-0 ¬ ((∃ g∈symmetries. (x1,y1) = (g ∘ i)
add-2-3)) |
      (333) (add-2-3, (x1,y1)) ∈ e′-aff-1 ¬ ((∃ g∈symmetries. (x1,y1) = (g ∘ i)
add-2-3)) (add-2-3, (x1,y1)) ∉ e′-aff-0
      using add-in in-aff dichotomy-1 by blast
    then show ?thesis
    proof(cases)
      case 111
        then show ?thesis using assoc-111-ext-add using 33(1) add-2-3-def
assms(1) assms(2) assms(3) by blast
    next
      case 222
      have assumps: ((x1, y1),add-2-3) ∈ e′-aff-0
        apply(subst (3) prod.collapse[symmetric])
        using 222 e′-aff-0-invariance by fastforce
      consider
        (1111) (∃ g∈symmetries. (x3,y3) = (g ∘ i) add-1-2) |
        (2222) (add-1-2, (x3,y3)) ∈ e′-aff-0 ¬ ((∃ g∈symmetries. (x3,y3) = (g ∘
i) add-1-2)) |
        (3333) (add-1-2, (x3,y3)) ∈ e′-aff-1 ¬ ((∃ g∈symmetries. (x3,y3) = (g ∘
i) add-1-2)) (add-1-2, (x3,y3)) ∉ e′-aff-0
        using add-in-1-2 in-aff dichotomy-1 by blast
      then show ?thesis
      proof(cases)
        case 1111
        then obtain g where g-expr: g ∈ symmetries (x3, y3) = (g ∘ i) add-1-2
by blast
        then have rot: τ ∘ g ∈ rotations using sym-to-rot assms by blast

        have proj-addition (proj-addition ?g1 ?g2) ?g3 =
              proj-addition (gluing ‘‘ {(add-1-2, 0)}) (gluing ‘‘ {((g ∘ i) add-1-2,
0)})
          using g-expr p-delta-1-2 gluing-add assms(1,2) add-1-2-def by force
        also have ... = tf′′ (τ ∘ g) {((1, 0), 0)}
          apply(subst proj-addition-comm)
          using e-proj-1-2(1) g-expr(2) assms(3) apply(simp,simp)
          apply(subst comp-apply,subst (2) prod.collapse[symmetric])
```

```
  apply(subst remove-sym)
  using e-proj-1-2(2) g-expr assms(3) apply(simp,simp,simp)
  apply(subst remove-add-sym)
  using e-proj-1-2 rot apply(simp,simp,simp)
  apply(subst prod.collapse, subst (2 4) prod.collapse[symmetric])
  apply(subst proj-addition-comm)
  using e-proj-1-2 apply(simp,simp)
  apply(subst proj-add-class-inv(1))
  using e-proj-1-2 apply simp
  using e-proj-1-2(1) by auto
finally have eq1: proj-addition (proj-addition ?g1 ?g2) ?g3 =
                 tf'' (τ ∘ g) {((1, 0), 0)} by blast

have proj-addition ?g1 (proj-addition ?g2 ?g3) =
    proj-addition ?g1 (proj-addition ?g2 (gluing `` {((g ∘ i) add-1-2, 0)}))

  using g-expr by auto
also have ... = proj-addition ?g1
                (tf'' (τ ∘ g)
                   (proj-addition (gluing `` {(add (i (x1, y1)) (i (x2, y2)),
0)})
                  ?g2))
  apply(subst comp-apply,subst (6) prod.collapse[symmetric])
  apply(subst (3) remove-sym)
  using e-proj-1-2(2) g-expr assms(3) apply(simp,simp,simp)
  apply(subst prod.collapse)
  apply(subst (2) proj-addition-comm)
  using assms(2) apply simp
  using tf''-preserv-e-proj rot e-proj-1-2(2) apply (metis prod.collapse)
  apply(subst remove-add-sym)
  using assms(2) e-proj-1-2(2) rot apply(simp,simp,simp)
  unfolding add-1-2-def
  by(subst inverse-rule-3,blast)
also have ... = proj-addition ?g1 (tf'' (τ ∘ g)
                  (proj-addition (proj-addition ?ig1 ?ig2) ?g2))
proof −
  have gluing `` {(add (i (x1, y1)) (i (x2, y2)), 0)} =
        proj-addition ?ig1 ?ig2
    using gluing-add[symmetric, of fst (i (x1,y1)) snd (i (x1,y1)) 0
                                fst (i (x2,y2)) snd (i (x2,y2)) 0,
                   simplified prod.collapse] e-proj-0(1,2) p-delta-1-2(2)
    by simp
  then show ?thesis by presburger
qed
also have ... = proj-addition ?g1 (tf'' (τ ∘ g) ?ig1)
  using cancellation-assoc
  by (metis assms(2) e-proj-0(1) e-proj-0(2) i.simps i-idemp-explicit)
also have ... = tf'' (τ ∘ g) (proj-addition ?g1 ?ig1)
     using assms(1) e-proj-0(1) proj-addition-comm remove-add-sym rot
```

```
tf''-preserv-e-proj by fastforce
        also have ... = tf'' (τ ∘ g) {((1, 0), 0)}
          using assms(1) proj-add-class-comm proj-addition-def proj-add-class-inv
by auto
        finally have eq2: proj-addition ?g1 (proj-addition ?g2 ?g3) =
                       tf'' (τ ∘ g) {((1, 0), 0)} by blast
        then show ?thesis using eq1 eq2 by blast
      next
        case 2222

        have proj-addition (proj-addition ?g1 ?g2) ?g3 =
          proj-addition (gluing `` {(add (x1, y1) (x2, y2), 0)}) ?g3
            using gluing-add p-delta-1-2(1) e-proj-1-2 add-1-2-def assms(1,2) by
simp
        also have ... = gluing `` {(add (add (x1, y1) (x2, y2)) (x3, y3), 0)}
          apply(subst (2) prod.collapse[symmetric])
          apply(subst gluing-add)
          apply(subst prod.collapse)
          using gluing-ext-add p-delta-1-2(1) e-proj-1-2 add-1-2-def assms(1,2,3)
apply(simp,simp)
          using 2222 unfolding e'-aff-0-def add-1-2-def by(simp,force)
        also have ... = gluing `` {(add (x1, y1) (ext-add (x2, y2) (x3, y3)), 0)}
          apply(subst add-add-add-ext-assoc)
          apply(simp,simp)
          apply(subst prod.collapse[symmetric],subst prod.inject,fast)+
          using p-delta-1-2 p-delta-2-3(1) 2222(1) assumps in-aff
          unfolding e'-aff-0-def e'-aff-1-def delta-def delta'-def
                    add-1-2-def add-2-3-def e'-aff-def
          by auto
        also have ... = proj-addition ?g1 (gluing `` {(ext-add (x2, y2) (x3, y3),
0)})
          apply(subst (10) prod.collapse[symmetric])
          apply(subst gluing-add)
          using assms(1) e-proj-2-3(1) add-2-3-def assumps
          unfolding e'-aff-0-def by auto
        also have ... = proj-addition ?g1 (proj-addition ?g2 ?g3)
          apply(subst gluing-ext-add)
          using assms(2,3) p-delta-2-3(1) by auto
        finally show ?thesis by blast
      next
        case 3333

        have proj-addition (proj-addition ?g1 ?g2) ?g3 =
          proj-addition (gluing `` {(add (x1, y1) (x2, y2), 0)}) ?g3
            using gluing-add p-delta-1-2(1) e-proj-1-2 add-1-2-def assms(1,2) by
simp
        also have ... = gluing `` {(ext-add (add (x1, y1) (x2, y2)) (x3, y3), 0)}
          apply(subst (2) prod.collapse[symmetric])
          apply(subst gluing-ext-add)
```

```
        apply(subst prod.collapse)
        using gluing-ext-add p-delta-1-2(1) e-proj-1-2 add-1-2-def assms(1,2,3)
apply(simp,simp)
        using 3333 unfolding e'-aff-1-def add-1-2-def by(simp,force)
      also have ... = gluing `` {(add (x1, y1) (ext-add (x2, y2) (x3, y3)), 0)}
        apply(subst ext-add-add-ext-assoc)
        apply(simp,simp)
        apply(subst prod.collapse[symmetric],subst prod.inject,fast)+
        using p-delta-1-2 p-delta-2-3(1) 3333(1) assumps in-aff
        unfolding e'-aff-0-def e'-aff-1-def delta-def delta'-def
              add-1-2-def add-2-3-def e'-aff-def
        by auto
      also have ... = proj-addition ?g1 (gluing `` {(ext-add (x2, y2) (x3, y3),
0)})
        apply(subst (10) prod.collapse[symmetric])
        apply(subst gluing-add)
        using assms(1) e-proj-2-3(1) add-2-3-def assumps
        unfolding e'-aff-0-def by(simp,simp,force,simp)
      also have ... = proj-addition ?g1 (proj-addition ?g2 ?g3)
        apply(subst gluing-ext-add)
        using assms(2,3) p-delta-2-3(1) by auto
      finally show ?thesis by blast
    qed
  next
    case 333
    have assumps: ((x1, y1),add-2-3) ∈ e'-aff-1
      using 333(1) e'-aff-1-invariance  add-2-3-def by auto

    consider
      (1111) (∃ g∈symmetries. (x3,y3) = (g ∘ i) add-1-2) |
      (2222) (add-1-2, (x3,y3)) ∈ e'-aff-0 ¬ ((∃ g∈symmetries. (x3,y3) = (g ∘
i) add-1-2)) |
      (3333) (add-1-2, (x3,y3)) ∈ e'-aff-1 ¬ ((∃ g∈symmetries. (x3,y3) = (g ∘
i) add-1-2)) (add-1-2, (x3,y3)) ∉ e'-aff-0
      using add-in-1-2 in-aff dichotomy-1 by blast
    then show ?thesis
    proof(cases)
      case 1111
      then obtain g where g-expr: g ∈ symmetries (x3, y3) = (g ∘ i) add-1-2
by blast
      then have rot: τ ∘ g ∈ rotations using sym-to-rot assms by blast

      have proj-addition (proj-addition ?g1 ?g2) ?g3 =
            proj-addition (gluing `` {(add-1-2, 0)}) (gluing `` {((g ∘ i) add-1-2,
0)})
        using g-expr p-delta-1-2 gluing-add assms(1,2) add-1-2-def by force
      also have ... = tf'' (τ ∘ g) {((1, 0), 0)}
        apply(subst proj-addition-comm)
        using e-proj-1-2(1) g-expr(2) assms(3) apply(simp,simp)
```

```
  apply(subst comp-apply,subst (2) prod.collapse[symmetric])
  apply(subst remove-sym)
  using e-proj-1-2(2) g-expr assms(3) apply(simp,simp,simp)
  apply(subst remove-add-sym)
  using e-proj-1-2 rot apply(simp,simp,simp)
  apply(subst prod.collapse, subst (2 4) prod.collapse[symmetric])
  apply(subst proj-addition-comm)
  using e-proj-1-2 rot apply(simp,simp)
  apply(subst proj-add-class-inv(1))
  using e-proj-1-2(1) by auto
finally have eq1: proj-addition (proj-addition ?g1 ?g2) ?g3 =
                tf'' (τ ∘ g) {((1, 0), 0)} by blast

have proj-addition ?g1 (proj-addition ?g2 ?g3) =
    proj-addition ?g1 (proj-addition ?g2 (gluing `` {((g ∘ i) add-1-2, 0)}))

  using g-expr by auto
also have ... = proj-addition ?g1
               (tf'' (τ ∘ g)
                  (proj-addition (gluing `` {(add (i (x1, y1)) (i (x2, y2)),
0)})
                  ?g2))
  apply(subst comp-apply,subst (6) prod.collapse[symmetric])
  apply(subst (3) remove-sym)
  using e-proj-1-2(2) g-expr assms(3) apply(simp,simp,simp)
  apply(subst prod.collapse)
  apply(subst (2) proj-addition-comm)
  using assms(2) apply simp
  using tf''-preserv-e-proj rot e-proj-1-2(2) apply (metis prod.collapse)
  apply(subst remove-add-sym)
  using assms(2) e-proj-1-2(2) rot apply(simp,simp,simp)
  unfolding add-1-2-def
  by(subst inverse-rule-3,blast)
also have ... = proj-addition ?g1 (tf'' (τ ∘ g)
                (proj-addition (proj-addition ?ig1 ?ig2) ?g2))
proof -
  have gluing `` {(add (i (x1, y1)) (i (x2, y2)), 0)} =
        proj-addition ?ig1 ?ig2
    using gluing-add[symmetric, of fst (i (x1, y1)) snd (i (x1, y1)) 0
                          fst (i (x2, y2)) snd (i (x2, y2)) 0,
                simplified prod.collapse] e-proj-0(1,2) p-delta-1-2(2)
    by simp
  then show ?thesis by presburger
qed
also have ... = proj-addition ?g1 (tf'' (τ ∘ g) ?ig1)
  using cancellation-assoc
  by (metis assms(2) e-proj-0(1) e-proj-0(2) i.simps i-idemp-explicit)
also have ... = tf'' (τ ∘ g) (proj-addition ?g1 ?ig1)
     using assms(1) e-proj-0(1) proj-addition-comm remove-add-sym rot
```

tf''-preserv-e-proj **by** *fastforce*
also have ... = *tf''* (τ ∘ *g*) {((*1*, *0*), *0*)}
using *assms*(*1*) *proj-add-class-comm proj-addition-def proj-add-class-inv*
by *auto*
finally have *eq2*: *proj-addition* (*gluing* ‘‘ {((*x1*, *y1*), *0*)})
(*proj-addition* (*gluing* ‘‘ {((*x2*, *y2*), *0*)}) (*gluing* ‘‘ {((*x3*,
y3), *0*)})) =
tf'' (τ ∘ *g*) {((*1*, *0*), *0*)} **by** *blast*
then show *?thesis* **using** *eq1 eq2* **by** *blast*
next
case *2222*

have *proj-addition* (*proj-addition ?g1 ?g2*) *?g3* =
proj-addition (*gluing* ‘‘ {(*add* (*x1*, *y1*) (*x2*, *y2*), *0*)}) *?g3*
using *gluing-add p-delta-1-2*(*1*) *e-proj-1-2 add-1-2-def assms*(*1*,*2*) **by**
simp
also have ... = *gluing* ‘‘ {(*add* (*add* (*x1*, *y1*) (*x2*, *y2*)) (*x3*, *y3*), *0*)}
apply(*subst* (*2*) *prod.collapse*[*symmetric*])
apply(*subst gluing-add*)
apply(*subst prod.collapse*)
using *gluing-ext-add p-delta-1-2*(*1*) *e-proj-1-2 add-1-2-def assms*(*1*,*2*,*3*)
apply(*simp*,*simp*)
using *2222* **unfolding** *e'-aff-0-def add-1-2-def* **by**(*simp*,*force*)
also have ... = *gluing* ‘‘ {(*ext-add* (*x1*, *y1*) (*ext-add* (*x2*, *y2*) (*x3*, *y3*)),
0)}
apply(*subst add-add-ext-ext-assoc*)
apply(*simp*,*simp*)
apply(*subst prod.collapse*[*symmetric*],*subst prod.inject*,*fast*)+
using *p-delta-1-2 p-delta-2-3*(*1*) *2222*(*1*) *assumps in-aff*
unfolding *e'-aff-0-def e'-aff-1-def delta-def delta'-def*
add-1-2-def add-2-3-def e'-aff-def
by *force*+
also have ... = *proj-addition ?g1* (*gluing* ‘‘ {(*ext-add* (*x2*, *y2*) (*x3*, *y3*),
0)})
apply(*subst* (*10*) *prod.collapse*[*symmetric*])
apply(*subst gluing-ext-add*)
using *assms*(*1*) *e-proj-2-3*(*1*) *add-2-3-def assumps*
unfolding *e'-aff-1-def* **by**(*blast*,*auto*)
also have ... = *proj-addition ?g1* (*proj-addition ?g2 ?g3*)
apply(*subst gluing-ext-add*)
using *assms*(*2*,*3*) *p-delta-2-3*(*1*) **by** *auto*
finally show *?thesis* **by** *blast*
next
case *3333*

have *proj-addition* (*proj-addition ?g1 ?g2*) *?g3* =
proj-addition (*gluing* ‘‘ {(*add* (*x1*, *y1*) (*x2*, *y2*), *0*)}) *?g3*
using *gluing-add p-delta-1-2*(*1*) *e-proj-1-2 add-1-2-def assms*(*1*,*2*) **by**
simp

```
        also have ... = gluing `` {(ext-add (add (x1, y1) (x2, y2)) (x3, y3), 0)}
          apply(subst (2) prod.collapse[symmetric])
          apply(subst gluing-ext-add)
          apply(subst prod.collapse)
          using gluing-ext-add p-delta-1-2(1) e-proj-1-2 add-1-2-def assms(1,2,3)
apply(simp,simp)
          using 3333 unfolding e'-aff-1-def add-1-2-def by(simp,force)
        also have ... = gluing `` {(ext-add (x1, y1) (ext-add (x2, y2) (x3, y3)),
0)}
          apply(subst ext-add-ext-ext-assoc)
          apply(simp,simp)
          apply(subst prod.collapse[symmetric],subst prod.inject,fast)+
          using p-delta-1-2 p-delta-2-3(1) 3333(1) assumps in-aff
          unfolding e'-aff-0-def e'-aff-1-def delta-def delta'-def
                    add-1-2-def add-2-3-def e'-aff-def
          by(force)+
        also have ... = proj-addition ?g1 (gluing `` {(ext-add (x2, y2) (x3, y3),
0)})
          apply(subst (10) prod.collapse[symmetric])
          apply(subst gluing-ext-add)
          using assms(1) e-proj-2-3(1) add-2-3-def assumps
          unfolding e'-aff-1-def by(simp,simp,force,simp)
        also have ... = proj-addition ?g1 (proj-addition ?g2 ?g3)
          apply(subst gluing-ext-add)
          using assms(2,3) p-delta-2-3(1) by auto
        finally show ?thesis by blast
      qed
    qed
  qed
 next
  case 3
  have p-delta-1-2: delta' x1 y1 x2 y2 ≠ 0
                    delta' (fst (i (x1, y1))) (snd (i (x1, y1)))
                           (fst (i (x2, y2))) (snd (i (x2, y2))) ≠ 0
    using 3 unfolding e'-aff-1-def apply simp
    using 3 in-aff unfolding e'-aff-1-def delta'-def delta-x-def delta-y-def
    by auto

  define add-1-2 where add-1-2 = ext-add (x1, y1) (x2, y2)
  have add-in-1-2: add-1-2 ∈ e'-aff
    unfolding e'-aff-def add-1-2-def
    apply(simp del: ext-add.simps)
    apply(subst (2) prod.collapse[symmetric])
    apply(standard)
    apply(subst ext-add-closure)
    using in-aff p-delta-1-2(1) e-e'-iff
    unfolding delta'-def e'-aff-def by(blast,(simp)+)

  have e-proj-1-2: gluing `` {(add-1-2, 0)} ∈ e-proj
```

```
              gluing `` {(i add-1-2, 0)} ∈ e-proj
      using add-in-1-2 add-1-2-def e-points apply simp
      using add-in-1-2 add-1-2-def e-points proj-add-class-inv by force

    consider
      (11) (∃ g∈symmetries. (x3, y3) = (g ∘ i) (x2, y2)) |
      (22) ((x2, y2), (x3, y3)) ∈ e′-aff-0 ¬ ((∃ g∈symmetries. (x3, y3) = (g ∘ i)
(x2, y2))) |
      (33) ((x2, y2), (x3, y3)) ∈ e′-aff-1 ¬ ((∃ g∈symmetries. (x3, y3) = (g ∘ i)
(x2, y2))) ((x2, y2), (x3, y3)) ∉ e′-aff-0
      using dichotomy-1 in-aff by blast
    then show ?thesis
    proof(cases)
      case 11
       then obtain g where g-expr: g ∈ symmetries (x3, y3) = (g ∘ i) (x2, y2)
by blast
      then show ?thesis  using assoc-11 assms by force
    next
      case 22
      have p-delta-2-3: delta x2 y2 x3 y3 ≠ 0
                delta (fst (i (x2,y2))) (snd (i (x2,y2)))
                      (fst (i (x3,y3))) (snd (i (x3,y3))) ≠ 0
        using 22 unfolding e′-aff-0-def apply simp
        using 22 unfolding e′-aff-0-def delta-def delta-plus-def delta-minus-def by
simp

      define add-2-3 where add-2-3 = add (x2,y2) (x3,y3)
      have add-in: add-2-3 ∈ e′-aff
        unfolding e′-aff-def add-2-3-def
        apply(simp del: add.simps)
        apply(subst (2) prod.collapse[symmetric])
        apply(standard)
        apply(simp del: add.simps add: e-e′-iff[symmetric])
        apply(subst add-closure)
      using in-aff e-e′-iff 22 unfolding e′-aff-def e′-aff-0-def delta-def by(fastforce)+
      have e-proj-2-3: gluing `` {(add-2-3, 0)} ∈ e-proj
                gluing `` {(i add-2-3, 0)} ∈ e-proj
        using add-in add-2-3-def e-points apply simp
        using add-in add-2-3-def e-points proj-add-class-inv by force

      consider
        (111) (∃ g∈symmetries. (x1,y1) = (g ∘ i) add-2-3) |
        (222) (add-2-3, (x1,y1)) ∈ e′-aff-0 ¬ ((∃ g∈symmetries. (x1,y1) = (g ∘ i)
add-2-3)) |
        (333) (add-2-3, (x1,y1)) ∈ e′-aff-1 ¬ ((∃ g∈symmetries. (x1,y1) = (g ∘ i)
add-2-3)) (add-2-3, (x1,y1)) ∉ e′-aff-0
        using add-in in-aff dichotomy-1 by blast
      then show ?thesis
      proof(cases)
```

case *111*
then show *?thesis* **using** *assoc-111-add* **using** *22*(*1*) *add-2-3-def assms*(*1*) *assms*(*2*) *assms*(*3*) **by** *blast*
next
case *222*
have *assumps*: ((*x1*, *y1*),*add-2-3*) ∈ *e′-aff-0*
apply(*subst* (*3*) *prod.collapse*[*symmetric*])
using *222 e′-aff-0-invariance* **by** *fastforce*

consider
(*1111*) (∃ *g*∈*symmetries*. (*x3*,*y3*) = (*g* ∘ *i*) *add-1-2*) |
(*2222*) (*add-1-2*, (*x3*,*y3*)) ∈ *e′-aff-0* ¬ ((∃ *g*∈*symmetries*. (*x3*,*y3*) = (*g* ∘ *i*) *add-1-2*)) |
(*3333*) (*add-1-2*, (*x3*,*y3*)) ∈ *e′-aff-1* ¬ ((∃ *g*∈*symmetries*. (*x3*,*y3*) = (*g* ∘ *i*) *add-1-2*)) (*add-1-2*, (*x3*,*y3*)) ∉ *e′-aff-0*
using *add-in-1-2 in-aff dichotomy-1* **by** *blast*
then show *?thesis*
proof(*cases*)
case *1111*
then obtain *g* **where** *g-expr*: *g* ∈ *symmetries* (*x3*, *y3*) = (*g* ∘ *i*) *add-1-2* **by** *blast*
then have *rot*: *τ* ∘ *g* ∈ *rotations* **using** *sym-to-rot assms* **by** *blast*

have *proj-addition* (*proj-addition ?g1 ?g2*) *?g3* =
proj-addition (*gluing* ‘‘ {(*add-1-2*, *0*)}) (*gluing* ‘‘ {((*g* ∘ *i*) *add-1-2*, *0*)})
using *g-expr p-delta-1-2 gluing-ext-add assms*(*1*,*2*) *add-1-2-def* **by** *auto*
also have ... = *tf′′* (*τ* ∘ *g*) ({((*1*, *0*), *0*)})
apply(*subst proj-addition-comm*)
using *e-proj-1-2*(*1*) *g-expr*(*2*) *assms*(*3*) **apply**(*simp*,*simp*)
apply(*subst comp-apply*,*subst* (*2*) *prod.collapse*[*symmetric*])
apply(*subst remove-sym*)
using *e-proj-1-2*(*2*) *g-expr assms*(*3*) **apply**(*simp*,*simp*,*simp*)
apply(*subst remove-add-sym*)
using *e-proj-1-2 rot* **apply**(*simp*,*simp*,*simp*)
apply(*subst prod.collapse*, *subst* (*2 4*) *prod.collapse*[*symmetric*])
by (*metis cancellation-assoc e-proj-1-2*(*1*) *e-proj-1-2*(*2*) *identity-equiv identity-proj*
prod.collapse proj-add-class-identity proj-addition-comm)
finally have *eq1*: *proj-addition* (*proj-addition ?g1 ?g2*) *?g3* =
tf′′ (*τ* ∘ *g*) ({((*1*, *0*), *0*)}) **by** *blast*

have *proj-addition ?g1* (*proj-addition ?g2 ?g3*) =
proj-addition ?g1 (*proj-addition ?g2* (*gluing* ‘‘ {((*g* ∘ *i*) *add-1-2*, *0*)}))

using *g-expr* **by** *auto*
also have ... = *proj-addition ?g1*
(*tf′′* (*τ* ∘ *g*)
(*proj-addition* (*gluing* ‘‘ {(*ext-add* (*i* (*x1*, *y1*)) (*i* (*x2*,

```
y2)), 0)})
                  ?g2))
      apply(subst comp-apply,subst (6) prod.collapse[symmetric])
      apply(subst (3) remove-sym)
      using e-proj-1-2(2) g-expr assms(3) apply(simp,simp,simp)
      apply(subst prod.collapse)
      apply(subst (2) proj-addition-comm)
      using assms(2) apply simp
      using tf''-preserv-e-proj rot e-proj-1-2(2) apply (metis prod.collapse)
      apply(subst remove-add-sym)
      using assms(2) e-proj-1-2(2) rot apply(simp,simp,simp)
      unfolding add-1-2-def
      by(subst inverse-rule-4,blast)
    also have ... = proj-addition ?g1 (tf'' (τ ∘ g)
                  (proj-addition (proj-addition ?ig1 ?ig2)
                  ?g2))
    proof −
      have gluing `` {(ext-add (i (x1, y1)) (i (x2, y2)), 0)} =
           proj-addition ?ig1 ?ig2
        using gluing-ext-add[symmetric,of fst (i (x1,y1)) snd (i (x1,y1)) 0
                                    fst (i (x2,y2)) snd (i (x2,y2)) 0,
                        simplified prod.collapse] e-proj-0(1,2) p-delta-1-2(2)
        by simp
      then show ?thesis by presburger
    qed
    also have ... = proj-addition ?g1 (tf'' (τ ∘ g) ?ig1)
      using cancellation-assoc
      by (metis assms(2) e-proj-0(1) e-proj-0(2) i.simps i-idemp-explicit)
    also have ... = tf'' (τ ∘ g) (proj-addition ?g1 ?ig1)
        using assms(1) e-proj-0(1) proj-addition-comm remove-add-sym rot
tf''-preserv-e-proj by fastforce
    also have ... = tf'' (τ ∘ g) ({((1, 0), 0)})
      using assms(1) proj-add-class-comm proj-add-class-inv by simp
    finally have eq2: proj-addition ?g1 (proj-addition ?g2 ?g3) =
                 tf'' (τ ∘ g) ({((1, 0), 0)}) by auto
    then show ?thesis
      using eq1 eq2 by blast
  next
    case 2222
    have proj-addition (proj-addition ?g1 ?g2) ?g3 =
      proj-addition (gluing `` {(ext-add (x1, y1) (x2, y2), 0)}) ?g3
       using gluing-ext-add p-delta-1-2(1) e-proj-1-2 add-1-2-def assms(1,2)
by simp
    also have ... = gluing `` {(add (ext-add (x1, y1) (x2, y2)) (x3, y3), 0)}
      apply(subst (2) prod.collapse[symmetric])
      apply(subst gluing-add)
      apply(subst prod.collapse)
      using gluing-ext-add p-delta-1-2(1) e-proj-1-2 add-1-2-def assms(1,2,3)
apply(simp,simp)
```

```
          using 2222 unfolding e′-aff-0-def add-1-2-def by(simp,force)
        also have ... = gluing ‘‘ {(add (x1, y1) (add (x2, y2) (x3, y3)), 0)}
          apply(subst add-ext-add-add-assoc-points)
          using p-delta-1-2 p-delta-2-3 2222  assumps in-aff
          unfolding add-1-2-def add-2-3-def e′-aff-0-def
          by auto
        also have ... = proj-addition ?g1 (gluing ‘‘ {(add (x2, y2) (x3, y3), 0)})
          apply(subst (10) prod.collapse[symmetric])
          apply(subst gluing-add)
          using assms(1) e-proj-2-3(1) add-2-3-def assumps
          unfolding e′-aff-0-def by(simp,simp,force,simp)
        also have ... = proj-addition ?g1 (proj-addition ?g2 ?g3)
          apply(subst gluing-add)
          using assms(2,3) p-delta-2-3(1) by auto
        finally show ?thesis by blast
      next
        case 3333

        have proj-addition (proj-addition ?g1 ?g2) ?g3 =
          proj-addition (gluing ‘‘ {(ext-add (x1, y1) (x2, y2), 0)}) ?g3
            using gluing-ext-add p-delta-1-2(1) e-proj-1-2 add-1-2-def assms(1,2)
by simp
         also have ... = gluing ‘‘ {(ext-add (ext-add (x1, y1) (x2, y2)) (x3, y3),
0)}
          apply(subst (2) prod.collapse[symmetric])
          apply(subst gluing-ext-add)
          apply(subst prod.collapse)
          using gluing-ext-add p-delta-1-2(1) e-proj-1-2 add-1-2-def assms(1,2,3)
apply(simp,simp)
          using 3333 unfolding e′-aff-1-def add-1-2-def by(simp,force)
        also have ... = gluing ‘‘ {(add (x1, y1) (add (x2, y2) (x3, y3)), 0)}
          apply(subst ext-ext-add-add-assoc)
          apply(simp,simp)
          apply(subst prod.collapse[symmetric],subst prod.inject,fast)+
          using p-delta-1-2 p-delta-2-3(1) 3333(1) assumps in-aff
          unfolding e′-aff-0-def e′-aff-1-def delta-def delta′-def
                  add-1-2-def add-2-3-def e′-aff-def
          by auto
        also have ... = proj-addition ?g1 (gluing ‘‘ {(add (x2, y2) (x3, y3), 0)})
          apply(subst (10) prod.collapse[symmetric])
          apply(subst gluing-add)
          using assms(1) e-proj-2-3(1) add-2-3-def assumps
          unfolding e′-aff-0-def by(simp,simp,force,simp)
        also have ... = proj-addition ?g1 (proj-addition ?g2 ?g3)
          apply(subst gluing-add)
          using assms(2,3) p-delta-2-3(1) by auto
        finally show ?thesis by blast
      qed
    next
```

case *333*
have *assumps*: $((x1,\ y1),add\text{-}2\text{-}3) \in e'\text{-}aff\text{-}1$
using *333*(*1*) *e′-aff-1-invariance add-2-3-def* **by** *auto*

consider
(*1111*) $(\exists g{\in}symmetries.\ (x3,y3) = (g \circ i)\ add\text{-}1\text{-}2)$ |
(*2222*) $(add\text{-}1\text{-}2,\ (x3,y3)) \in e'\text{-}aff\text{-}0\ \neg\ ((\exists g{\in}symmetries.\ (x3,y3) = (g \circ i)\ add\text{-}1\text{-}2))$ |
(*3333*) $(add\text{-}1\text{-}2,\ (x3,y3)) \in e'\text{-}aff\text{-}1\ \neg\ ((\exists g{\in}symmetries.\ (x3,y3) = (g \circ i)\ add\text{-}1\text{-}2))\ (add\text{-}1\text{-}2,\ (x3,y3)) \notin e'\text{-}aff\text{-}0$
using *add-in-1-2 in-aff dichotomy-1* **by** *blast*
then show *?thesis*
proof(*cases*)
case *1111*
then obtain *g* **where** *g-expr*: $g \in symmetries\ (x3,\ y3) = (g \circ i)\ add\text{-}1\text{-}2$ **by** *blast*
then have *rot*: $\tau \circ g \in rotations$ **using** *sym-to-rot assms* **by** *blast*

have $proj\text{-}addition\ (proj\text{-}addition\ ?g1\ ?g2)\ ?g3 =$
$proj\text{-}addition\ (gluing\ ``\ \{(add\text{-}1\text{-}2,\ 0)\})\ (gluing\ ``\ \{((g \circ i)\ add\text{-}1\text{-}2,\ 0)\})$
using *g-expr p-delta-1-2 gluing-ext-add assms*(*1*,*2*) *add-1-2-def* **by** *force*
also have ... $= tf''\ (\tau \circ g)\ \{((1,\ 0),\ 0)\}$
apply(*subst proj-addition-comm*)
using *e-proj-1-2*(*1*) *g-expr*(*2*) *assms*(*3*) **apply**(*simp*,*simp*)
apply(*subst comp-apply*,*subst* (*2*) *prod.collapse*[*symmetric*])
apply(*subst remove-sym*)
using *e-proj-1-2*(*2*) *g-expr assms*(*3*) **apply**(*simp*,*simp*,*simp*)
apply(*subst remove-add-sym*)
using *e-proj-1-2 rot* **apply**(*simp*,*simp*,*simp*)
apply(*subst prod.collapse*, *subst* (*2 4*) *prod.collapse*[*symmetric*])
by (*metis* (*no-types*, *lifting*) *cancellation-assoc e-proj-1-2*(*1*) *e-proj-1-2*(*2*) *identity-equiv*
identity-proj prod.collapse proj-add-class-identity proj-addition-comm)
finally have *eq1*: $proj\text{-}addition\ (proj\text{-}addition\ ?g1\ ?g2)\ ?g3 =$
$tf''\ (\tau \circ g)\ \{((1,\ 0),\ 0)\}$ **by** *blast*

have $proj\text{-}addition\ ?g1\ (proj\text{-}addition\ ?g2\ ?g3) =$
$proj\text{-}addition\ ?g1\ (proj\text{-}addition\ ?g2\ (gluing\ ``\ \{((g \circ i)\ add\text{-}1\text{-}2,\ 0)\}))$

using *g-expr* **by** *auto*
also have ... $=\ proj\text{-}addition\ ?g1$
$(tf''\ (\tau \circ g)$
$(proj\text{-}addition\ (gluing\ ``\ \{(ext\text{-}add\ (i\ (x1,\ y1))\ (i\ (x2,\ y2)),\ 0)\})$
$?g2))$
apply(*subst comp-apply*,*subst* (*6*) *prod.collapse*[*symmetric*])
apply(*subst* (*3*) *remove-sym*)
using *e-proj-1-2*(*2*) *g-expr assms*(*3*) **apply**(*simp*,*simp*,*simp*)

apply(*subst prod.collapse*)
apply(*subst* (*2*) *proj-addition-comm*)
using *assms*(*2*) **apply** *simp*
using *tf''-preserv-e-proj rot e-proj-1-2*(*2*)
apply (*metis prod.collapse*)
apply(*subst remove-add-sym*)
using *assms*(*2*) *e-proj-1-2*(*2*) *rot* **apply**(*simp*,*simp*,*simp*)
unfolding *add-1-2-def*
by(*subst inverse-rule-4*,*blast*)
also have ... = *proj-addition ?g1* (*tf''* ($\tau \circ g$)
(*proj-addition* (*proj-addition ?ig1 ?ig2*) *?g2*))
proof −
have *gluing* ‘‘ {(*ext-add* (*i* (*x1*, *y1*)) (*i* (*x2*, *y2*)), *0*)} =
proj-addition ?ig1 ?ig2
using *gluing-ext-add*[*symmetric*, *of fst* (*i* (*x1*,*y1*)) *snd* (*i* (*x1*,*y1*)) *0*
fst (*i* (*x2*, *y2*)) *snd* (*i* (*x2*, *y2*)) *0*,
simplified prod.collapse] *e-proj-0*(*1*,*2*) *p-delta-1-2*(*2*)
by *simp*
then show *?thesis* **by** *presburger*
qed
also have ... = *proj-addition ?g1* (*tf''* ($\tau \circ g$) *?ig1*)
using *cancellation-assoc*
by (*metis assms*(*2*) *e-proj-0*(*1*) *e-proj-0*(*2*) *i.simps i-idemp-explicit*)
also have ... = *tf''* ($\tau \circ g$) (*proj-addition ?g1 ?ig1*)
using *assms*(*1*) *e-proj-0*(*1*) *proj-addition-comm remove-add-sym rot tf''-preserv-e-proj* **by** *fastforce*
also have ... = *tf''* ($\tau \circ g$) {((*1*, *0*), *0*)}
using *assms*(*1*) *proj-add-class-comm proj-addition-def proj-add-class-inv* **by** *simp*
finally have *eq2*: *proj-addition ?g1* (*proj-addition ?g2 ?g3*) =
tf'' ($\tau \circ g$) {((*1*, *0*), *0*)} **by** *auto*
then show *?thesis* **using** *eq1 eq2* **by** *blast*
next
case *2222*

have *proj-addition* (*proj-addition ?g1 ?g2*) *?g3* =
proj-addition (*gluing* ‘‘ {(*ext-add* (*x1*, *y1*) (*x2*, *y2*), *0*)}) *?g3*
using *gluing-ext-add p-delta-1-2*(*1*) *e-proj-1-2 add-1-2-def assms*(*1*,*2*) **by** *simp*
also have ... = *gluing* ‘‘ {(*add* (*ext-add* (*x1*, *y1*) (*x2*, *y2*)) (*x3*, *y3*), *0*)}
apply(*subst* (*2*) *prod.collapse*[*symmetric*])
apply(*subst gluing-add*)
apply(*subst prod.collapse*)
using *gluing-add p-delta-1-2*(*1*) *e-proj-1-2 add-1-2-def assms*(*1*,*2*,*3*) **apply**(*simp*,*simp*)
using *2222* **unfolding** *e'-aff-0-def add-1-2-def* **by**(*simp*,*force*)
also have ... = *gluing* ‘‘ {(*ext-add* (*x1*, *y1*) (*add* (*x2*, *y2*) (*x3*, *y3*)), *0*)}
apply(*subst add-ext-ext-add-assoc*)
apply(*simp*,*simp*)

apply(*subst prod.collapse*[*symmetric*],*subst prod.inject*,*fast*)+
using *p-delta-1-2 p-delta-2-3*(*1*) *2222*(*1*) *assumps in-aff*
unfolding *e'-aff-0-def e'-aff-1-def delta-def delta'-def*
add-1-2-def add-2-3-def e'-aff-def
by *force*+
also have ... = *proj-addition ?g1* (*gluing* ‘‘ {(*add* (*x2*, *y2*) (*x3*, *y3*), *0*)})
apply(*subst* (*10*) *prod.collapse*[*symmetric*])
apply(*subst gluing-ext-add*)
using *assms*(*1*) *e-proj-2-3*(*1*) *add-2-3-def assumps*
unfolding *e'-aff-1-def* **by**(*blast*,*auto*)
also have ... = *proj-addition ?g1* (*proj-addition ?g2 ?g3*)
apply(*subst gluing-add*)
using *assms*(*2*,*3*) *p-delta-2-3*(*1*) **by** *auto*
finally show *?thesis* **by** *blast*
next
case *3333*
have *proj-addition* (*proj-addition ?g1 ?g2*) *?g3* =
proj-addition (*gluing* ‘‘ {(*ext-add* (*x1*, *y1*) (*x2*, *y2*), *0*)}) *?g3*
using *gluing-ext-add p-delta-1-2*(*1*) *e-proj-1-2 add-1-2-def assms*(*1*,*2*)
by *simp*
also have ... = *gluing* ‘‘ {(*ext-add* (*ext-add* (*x1*, *y1*) (*x2*, *y2*)) (*x3*, *y3*),
0)}
apply(*subst* (*2*) *prod.collapse*[*symmetric*])
apply(*subst gluing-ext-add*)
apply(*subst prod.collapse*)
using *gluing-add p-delta-1-2*(*1*) *e-proj-1-2 add-1-2-def assms*(*1*,*2*,*3*)
apply(*simp*,*simp*)
using *3333* **unfolding** *e'-aff-1-def add-1-2-def* **by**(*simp*,*force*)
also have ... = *gluing* ‘‘ {(*ext-add* (*x1*, *y1*) (*add* (*x2*, *y2*) (*x3*, *y3*)), *0*)}
apply(*subst ext-ext-ext-add-assoc*)
apply(*simp*,*simp*)
apply(*subst prod.collapse*[*symmetric*],*subst prod.inject*,*fast*)+
using *p-delta-1-2 p-delta-2-3*(*1*) *3333*(*1*) *assumps in-aff*
unfolding *e'-aff-0-def e'-aff-1-def delta-def delta'-def*
add-1-2-def add-2-3-def e'-aff-def
by(*force*)+
also have ... = *proj-addition ?g1* (*gluing* ‘‘ {(*add* (*x2*, *y2*) (*x3*, *y3*), *0*)})
apply(*subst* (*10*) *prod.collapse*[*symmetric*])
apply(*subst gluing-ext-add*)
using *assms*(*1*) *e-proj-2-3*(*1*) *add-2-3-def assumps*
unfolding *e'-aff-1-def* **by**(*simp*,*simp*,*force*,*simp*)
also have ... = *proj-addition ?g1* (*proj-addition ?g2 ?g3*)
apply(*subst gluing-add*)
using *assms*(*2*,*3*) *p-delta-2-3*(*1*) **by** *auto*
finally show *?thesis* **by** *blast*
qed
qed
next
case *33*

```
have p-delta-2-3: delta' x2 y2 x3 y3 ≠ 0
                  delta' (fst (i (x2,y2))) (snd (i (x2,y2)))
                         (fst (i (x3,y3))) (snd (i (x3,y3))) ≠ 0
  using 33 unfolding e'-aff-1-def apply simp
 using 33 unfolding e'-aff-1-def delta'-def delta-x-def delta-y-def by fastforce

define add-2-3 where add-2-3 = ext-add (x2,y2) (x3,y3)
have add-in: add-2-3 ∈ e'-aff
  unfolding e'-aff-def add-2-3-def
  apply(simp del: ext-add.simps)
  apply(subst (2) prod.collapse[symmetric])
  apply(standard)
  apply(subst ext-add-closure)
 using in-aff e-e'-iff 33 unfolding e'-aff-def e'-aff-1-def delta'-def by(fastforce)+
have e-proj-2-3: gluing `` {(add-2-3, 0)} ∈ e-proj
                 gluing `` {(i add-2-3, 0)} ∈ e-proj
  using add-in add-2-3-def e-points apply simp
  using add-in add-2-3-def e-points proj-add-class-inv by force

consider
  (111) (∃ g∈symmetries. (x1,y1) = (g ∘ i) add-2-3) |
  (222) (add-2-3, (x1,y1)) ∈ e'-aff-0 ¬ ((∃ g∈symmetries. (x1,y1) = (g ∘ i)
add-2-3)) |
  (333) (add-2-3, (x1,y1)) ∈ e'-aff-1 ¬ ((∃ g∈symmetries. (x1,y1) = (g ∘ i)
add-2-3)) (add-2-3, (x1,y1)) ∉ e'-aff-0
  using add-in in-aff dichotomy-1 by blast
then show ?thesis
proof(cases)
  case 111
    then show ?thesis using assoc-111-ext-add using 33(1) add-2-3-def
assms(1) assms(2) assms(3) by blast
next
  case 222
  have assumps: ((x1, y1),add-2-3) ∈ e'-aff-0
    apply(subst (3) prod.collapse[symmetric])
    using 222 e'-aff-0-invariance by fastforce
  consider
    (1111) (∃ g∈symmetries. (x3,y3) = (g ∘ i) add-1-2) |
    (2222) (add-1-2, (x3,y3)) ∈ e'-aff-0 ¬ ((∃ g∈symmetries. (x3,y3) = (g ∘
i) add-1-2)) |
    (3333) (add-1-2, (x3,y3)) ∈ e'-aff-1 ¬ ((∃ g∈symmetries. (x3,y3) = (g ∘
i) add-1-2)) (add-1-2, (x3,y3)) ∉ e'-aff-0
    using add-in-1-2 in-aff dichotomy-1 by blast
  then show ?thesis
  proof(cases)
    case 1111
   then obtain g where g-expr: g ∈ symmetries (x3, y3) = (g ∘ i) add-1-2
by blast
    then have rot: τ ∘ g ∈ rotations using sym-to-rot assms by blast
```

```
have proj-addition (proj-addition ?g1 ?g2) ?g3 =
      proj-addition (gluing `` {(add-1-2, 0)}) (gluing `` {((g ∘ i) add-1-2,
0)})
  using g-expr p-delta-1-2 gluing-ext-add assms(1,2) add-1-2-def by force
also have ... = tf'' (τ ∘ g) {((1, 0), 0)}
  apply(subst proj-addition-comm)
  using e-proj-1-2(1) g-expr(2) assms(3) apply(simp,simp)
  apply(subst comp-apply,subst (2) prod.collapse[symmetric])
  apply(subst remove-sym)
  using e-proj-1-2(2) g-expr assms(3) apply(simp,simp,simp)
  apply(subst remove-add-sym)
  using e-proj-1-2 rot apply(simp,simp,simp)
  apply(subst prod.collapse, subst (2 4) prod.collapse[symmetric])
  apply(subst proj-addition-comm)
  using e-proj-1-2 apply(simp,simp)
  apply(subst proj-add-class-inv(1))
  using e-proj-1-2 apply simp
  using e-proj-1-2(1) by auto
finally have eq1: proj-addition (proj-addition ?g1 ?g2) ?g3 =
                tf'' (τ ∘ g) {((1, 0), 0)} by blast

have proj-addition ?g1 (proj-addition ?g2 ?g3) =
    proj-addition ?g1 (proj-addition ?g2 (gluing `` {((g ∘ i) add-1-2, 0)}))

  using g-expr by auto
also have ... = proj-addition ?g1
                (tf'' (τ ∘ g)
                     (proj-addition (gluing `` {(ext-add (i (x1, y1)) (i (x2,
y2)), 0)})
                  ?g2))
  apply(subst comp-apply,subst (6) prod.collapse[symmetric])
  apply(subst (3) remove-sym)
  using e-proj-1-2(2) g-expr assms(3) apply(simp,simp,simp)
  apply(subst prod.collapse)
  apply(subst (2) proj-addition-comm)
  using assms(2) apply simp
  using tf''-preserv-e-proj rot e-proj-1-2(2) apply (metis prod.collapse)
  apply(subst remove-add-sym)
  using assms(2) e-proj-1-2(2) rot apply(simp,simp,simp)
  unfolding add-1-2-def
  by(subst inverse-rule-4,blast)
also have ... = proj-addition ?g1 (tf'' (τ ∘ g)
                  (proj-addition (proj-addition ?ig1 ?ig2) ?g2))
proof -
  have gluing `` {(ext-add (i (x1, y1)) (i (x2, y2)), 0)} =
        proj-addition ?ig1 ?ig2
    using gluing-ext-add[symmetric, of fst (i (x1,y1)) snd (i (x1,y1)) 0
                              fst (i (x2,y2)) snd (i (x2,y2)) 0,
```

simplified prod.collapse] *e-proj-0*(*1*,*2*) *p-delta-1-2*(*2*)
by *simp*
then show *?thesis* **by** *presburger*
qed
also have ... = *proj-addition ?g1* (*tf''* ($\tau \circ g$) *?ig1*)
using *cancellation-assoc*
by (*metis assms*(*2*) *e-proj-0*(*1*) *e-proj-0*(*2*) *i.simps i-idemp-explicit*)
also have ... = *tf''* ($\tau \circ g$) (*proj-addition ?g1 ?ig1*)
using *assms*(*1*) *e-proj-0*(*1*) *proj-addition-comm remove-add-sym rot tf''-preserv-e-proj* **by** *fastforce*
also have ... = *tf''* ($\tau \circ g$) {((*1*, *0*), *0*)}
using *assms*(*1*) *proj-add-class-comm proj-addition-def proj-add-class-inv* **by** *auto*
finally have *eq2*: *proj-addition ?g1* (*proj-addition ?g2 ?g3*) =
tf'' ($\tau \circ g$) {((*1*, *0*), *0*)} **by** *blast*
then show *?thesis* **using** *eq1 eq2* **by** *blast*
next
case *2222*

have *proj-addition* (*proj-addition ?g1 ?g2*) *?g3* =
proj-addition (*gluing* ‘‘ {(*ext-add* (*x1*, *y1*) (*x2*, *y2*), *0*)}) *?g3*
using *gluing-ext-add p-delta-1-2*(*1*) *e-proj-1-2 add-1-2-def assms*(*1*,*2*) **by** *simp*
also have ... = *gluing* ‘‘ {(*add* (*ext-add* (*x1*, *y1*) (*x2*, *y2*)) (*x3*, *y3*), *0*)}
apply(*subst* (*2*) *prod.collapse*[*symmetric*])
apply(*subst gluing-add*)
apply(*subst prod.collapse*)
using *gluing-ext-add p-delta-1-2*(*1*) *e-proj-1-2 add-1-2-def assms*(*1*,*2*,*3*) **apply**(*simp*,*simp*)
using *2222* **unfolding** *e'-aff-0-def add-1-2-def* **by**(*simp*,*force*)
also have ... = *gluing* ‘‘ {(*add* (*x1*, *y1*) (*ext-add* (*x2*, *y2*) (*x3*, *y3*)), *0*)}
apply(*subst add-ext-add-ext-assoc*)
apply(*simp*,*simp*)
apply(*subst prod.collapse*[*symmetric*],*subst prod.inject*,*fast*)+
using *p-delta-1-2 p-delta-2-3*(*1*) *2222*(*1*) *assumps in-aff*
unfolding *e'-aff-0-def e'-aff-1-def delta-def delta'-def*
add-1-2-def add-2-3-def e'-aff-def
by *auto*
also have ... = *proj-addition ?g1* (*gluing* ‘‘ {(*ext-add* (*x2*, *y2*) (*x3*, *y3*), *0*)})
apply(*subst* (*10*) *prod.collapse*[*symmetric*])
apply(*subst gluing-add*)
using *assms*(*1*) *e-proj-2-3*(*1*) *add-2-3-def assumps*
unfolding *e'-aff-0-def* **by** *auto*
also have ... = *proj-addition ?g1* (*proj-addition ?g2 ?g3*)
apply(*subst gluing-ext-add*)
using *assms*(*2*,*3*) *p-delta-2-3*(*1*) **by** *auto*
finally show *?thesis* **by** *blast*
next

```
        case 3333

        have proj-addition (proj-addition ?g1 ?g2) ?g3 =
          proj-addition (gluing ‘‘ {(ext-add (x1, y1) (x2, y2), 0)}) ?g3
            using gluing-ext-add p-delta-1-2(1) e-proj-1-2 add-1-2-def assms(1,2)
by simp
        also have ... = gluing ‘‘ {(ext-add (ext-add (x1, y1) (x2, y2)) (x3, y3),
0)}
          apply(subst (2) prod.collapse[symmetric])
          apply(subst gluing-ext-add)
          apply(subst prod.collapse)
          using gluing-ext-add p-delta-1-2(1) e-proj-1-2 add-1-2-def assms(1,2,3)
apply(simp,simp)
          using 3333 unfolding e′-aff-1-def add-1-2-def by(simp,force)
        also have ... = gluing ‘‘ {(add (x1, y1) (ext-add (x2, y2) (x3, y3)), 0)}
          apply(subst ext-ext-add-ext-assoc)
          apply(simp,simp)
          apply(subst prod.collapse[symmetric],subst prod.inject,fast)+
          using p-delta-1-2 p-delta-2-3(1) 3333(1) assumps in-aff
          unfolding e′-aff-0-def e′-aff-1-def delta-def delta′-def
                add-1-2-def add-2-3-def e′-aff-def
          by auto
        also have ... = proj-addition ?g1 (gluing ‘‘ {(ext-add (x2, y2) (x3, y3),
0)})
          apply(subst (10) prod.collapse[symmetric])
          apply(subst gluing-add)
          using assms(1) e-proj-2-3(1) add-2-3-def assumps
          unfolding e′-aff-0-def by(simp,simp,force,simp)
        also have ... = proj-addition ?g1 (proj-addition ?g2 ?g3)
          apply(subst gluing-ext-add)
          using assms(2,3) p-delta-2-3(1) by auto
        finally show ?thesis by blast
      qed
    next
      case 333
      have assumps: ((x1, y1),add-2-3) ∈ e′-aff-1
        using 333(1) e′-aff-1-invariance  add-2-3-def by auto

      consider
        (1111) (∃ g∈symmetries. (x3,y3) = (g ∘ i) add-1-2) |
        (2222) (add-1-2, (x3,y3)) ∈ e′-aff-0 ¬ ((∃ g∈symmetries. (x3,y3) = (g ∘
i) add-1-2)) |
        (3333) (add-1-2, (x3,y3)) ∈ e′-aff-1 ¬ ((∃ g∈symmetries. (x3,y3) = (g ∘
i) add-1-2)) (add-1-2, (x3,y3)) ∉ e′-aff-0
        using add-in-1-2 in-aff dichotomy-1 by blast
      then show ?thesis
      proof(cases)
        case 1111
        then obtain g where g-expr: g ∈ symmetries (x3, y3) = (g ∘ i) add-1-2
```

```
by blast
        then have rot: τ ∘ g ∈ rotations using sym-to-rot assms by blast

        have proj-addition (proj-addition ?g1 ?g2) ?g3 =
             proj-addition (gluing `` {(add-1-2, 0)}) (gluing `` {((g ∘ i) add-1-2,
0)})
          using g-expr p-delta-1-2 gluing-ext-add assms(1,2) add-1-2-def by force
        also have ... = tf'' (τ ∘ g) {((1, 0), 0)}
          apply(subst proj-addition-comm)
          using e-proj-1-2(1) g-expr(2) assms(3) apply(simp,simp)
          apply(subst comp-apply,subst (2) prod.collapse[symmetric])
          apply(subst remove-sym)
          using e-proj-1-2(2) g-expr assms(3) apply(simp,simp,simp)
          apply(subst remove-add-sym)
          using e-proj-1-2 rot apply(simp,simp,simp)
          apply(subst prod.collapse, subst (2 4) prod.collapse[symmetric])
          apply(subst proj-addition-comm)
          using e-proj-1-2 rot apply(simp,simp)
          apply(subst proj-add-class-inv(1))
          using e-proj-1-2(1) by auto
        finally have eq1: proj-addition (proj-addition ?g1 ?g2) ?g3 =
                        tf'' (τ ∘ g) {((1, 0), 0)} by blast

        have proj-addition ?g1 (proj-addition ?g2 ?g3) =
            proj-addition ?g1 (proj-addition ?g2 (gluing `` {((g ∘ i) add-1-2, 0)}))

          using g-expr by auto
        also have ... =  proj-addition ?g1
                        (tf'' (τ ∘ g)
                            (proj-addition (gluing `` {(ext-add (i (x1, y1)) (i (x2,
y2)), 0)})
                          ?g2))
          apply(subst comp-apply,subst (6) prod.collapse[symmetric])
          apply(subst (3) remove-sym)
          using e-proj-1-2(2) g-expr assms(3) apply(simp,simp,simp)
          apply(subst prod.collapse)
          apply(subst (2) proj-addition-comm)
          using assms(2) apply simp
          using tf''-preserv-e-proj rot e-proj-1-2(2) apply (metis prod.collapse)
          apply(subst remove-add-sym)
          using assms(2) e-proj-1-2(2) rot apply(simp,simp,simp)
          unfolding add-1-2-def
          by(subst inverse-rule-4,blast)
        also have ... = proj-addition ?g1 (tf'' (τ ∘ g)
                         (proj-addition (proj-addition ?ig1 ?ig2) ?g2))
        proof -
          have gluing `` {(ext-add (i (x1, y1)) (i (x2, y2)), 0)} =
                proj-addition ?ig1 ?ig2
            using gluing-ext-add[symmetric, of fst (i (x1, y1)) snd (i (x1, y1)) 0
```

fst (i (x2, y2)) snd (i (x2, y2)) 0,
simplified prod.collapse] e-proj-0(1,2) p-delta-1-2(2)
by simp
then show ?thesis **by** presburger
qed
also have ... = proj-addition ?g1 (tf'' ($\tau \circ g$) ?ig1)
using cancellation-assoc
by (metis assms(2) e-proj-0(1) e-proj-0(2) i.simps i-idemp-explicit)
also have ... = tf'' ($\tau \circ g$) (proj-addition ?g1 ?ig1)
using assms(1) e-proj-0(1) proj-addition-comm remove-add-sym rot tf''-preserv-e-proj **by** fastforce
also have ... = tf'' ($\tau \circ g$) {((1, 0), 0)}
using assms(1) proj-add-class-comm proj-addition-def proj-add-class-inv **by** auto
finally have eq2: proj-addition (gluing `` {((x1, y1), 0)})
(proj-addition (gluing `` {((x2, y2), 0)}) (gluing `` {((x3, y3), 0)})) =
tf'' ($\tau \circ g$) {((1, 0), 0)} **by** blast
then show ?thesis **using** eq1 eq2 **by** blast
next
case 2222

have proj-addition (proj-addition ?g1 ?g2) ?g3 =
proj-addition (gluing `` {(ext-add (x1, y1) (x2, y2), 0)}) ?g3
using gluing-ext-add p-delta-1-2(1) e-proj-1-2 add-1-2-def assms(1,2) **by** simp
also have ... = gluing `` {(add (ext-add (x1, y1) (x2, y2)) (x3, y3), 0)}
apply(subst (2) prod.collapse[symmetric])
apply(subst gluing-add)
apply(subst prod.collapse)
using gluing-ext-add p-delta-1-2(1) e-proj-1-2 add-1-2-def assms(1,2,3) **apply**(simp,simp)
using 2222 **unfolding** e'-aff-0-def add-1-2-def **by**(simp,force)
also have ... = gluing `` {(ext-add (x1, y1) (ext-add (x2, y2) (x3, y3)), 0)}
apply(subst add-ext-ext-ext-assoc)
apply(simp,simp)
apply(subst prod.collapse[symmetric],subst prod.inject,fast)+
using p-delta-1-2 p-delta-2-3(1) 2222(1) assumps in-aff
unfolding e'-aff-0-def e'-aff-1-def delta-def delta'-def
add-1-2-def add-2-3-def e'-aff-def
by force+
also have ... = proj-addition ?g1 (gluing `` {(ext-add (x2, y2) (x3, y3), 0)})
apply(subst (10) prod.collapse[symmetric])
apply(subst gluing-ext-add)
using assms(1) e-proj-2-3(1) add-2-3-def assumps
unfolding e'-aff-1-def **by**(blast,auto)
also have ... = proj-addition ?g1 (proj-addition ?g2 ?g3)

apply(*subst gluing-ext-add*)
using *assms*(*2*,*3*) *p-delta-2-3*(*1*) **by** *auto*
finally show *?thesis* **by** *blast*
next
case *3333*

have *proj-addition* (*proj-addition ?g1 ?g2*) *?g3* =
proj-addition (*gluing* ‘‘ {(*ext-add* (*x1*, *y1*) (*x2*, *y2*), *0*)}) *?g3*
using *gluing-ext-add p-delta-1-2*(*1*) *e-proj-1-2 add-1-2-def assms*(*1*,*2*)
by *simp*
also have ... = *gluing* ‘‘ {(*ext-add* (*ext-add* (*x1*, *y1*) (*x2*, *y2*)) (*x3*, *y3*),
0)}
apply(*subst* (*2*) *prod.collapse*[*symmetric*])
apply(*subst gluing-ext-add*)
apply(*subst prod.collapse*)
using *gluing-ext-add p-delta-1-2*(*1*) *e-proj-1-2 add-1-2-def assms*(*1*,*2*,*3*)
apply(*simp*,*simp*)
using *3333* **unfolding** *e′-aff-1-def add-1-2-def* **by**(*simp*,*force*)
also have ... = *gluing* ‘‘ {(*ext-add* (*x1*, *y1*) (*ext-add* (*x2*, *y2*) (*x3*, *y3*)),
0)}
apply(*subst ext-ext-ext-ext-assoc*)
apply(*simp*,*simp*)
apply(*subst prod.collapse*[*symmetric*],*subst prod.inject*,*fast*)+
using *p-delta-1-2 p-delta-2-3*(*1*) *3333*(*1*) *assumps in-aff*
unfolding *e′-aff-0-def e′-aff-1-def delta-def delta′-def*
add-1-2-def add-2-3-def e′-aff-def
by(*force*)+
also have ... = *proj-addition ?g1* (*gluing* ‘‘ {(*ext-add* (*x2*, *y2*) (*x3*, *y3*),
0)})
apply(*subst* (*10*) *prod.collapse*[*symmetric*])
apply(*subst gluing-ext-add*)
using *assms*(*1*) *e-proj-2-3*(*1*) *add-2-3-def assumps*
unfolding *e′-aff-1-def* **by**(*simp*,*simp*,*force*,*simp*)
also have ... = *proj-addition ?g1* (*proj-addition ?g2 ?g3*)
apply(*subst gluing-ext-add*)
using *assms*(*2*,*3*) *p-delta-2-3*(*1*) **by** *auto*
finally show *?thesis* **by** *blast*
qed
qed
qed
qed
qed

lemma *general-assoc*:
assumes *gluing* ‘‘ {((*x1*, *y1*), *l*)} ∈ *e-proj gluing* ‘‘ {((*x2*, *y2*), *m*)} ∈ *e-proj*
gluing ‘‘ {((*x3*, *y3*), *n*)} ∈ *e-proj*
shows *proj-addition* (*proj-addition* (*gluing* ‘‘ {((*x1*, *y1*), *l*)}) (*gluing* ‘‘ {((*x2*,

y2), *m*)}))
(*gluing* ‘‘ {((*x3*, *y3*), *n*)}) =
proj-addition (*gluing* ‘‘ {((*x1*, *y1*), *l*)})
(*proj-addition* (*gluing* ‘‘ {((*x2*, *y2*), *m*)}) (*gluing* ‘‘ {((*x3*, *y3*), *n*)}))
proof −
let *?t1* = (*proj-addition* (*proj-addition* (*gluing* ‘‘ {((*x1*, *y1*), *0*)}) (*gluing* ‘‘ {((*x2*, *y2*), *0*)}))
(*gluing* ‘‘ {((*x3*, *y3*), *0*)}))
let *?t2* = *proj-addition* (*gluing* ‘‘ {((*x1*, *y1*), *0*)})
(*proj-addition* (*gluing* ‘‘ {((*x2*, *y2*), *0*)}) (*gluing* ‘‘ {((*x3*, *y3*), *0*)}))

have *e-proj-0*: *gluing* ‘‘ {((*x1*, *y1*), *0*)} ∈ *e-proj*
gluing ‘‘ {((*x2*, *y2*), *0*)} ∈ *e-proj*
gluing ‘‘ {((*x3*, *y3*), *0*)} ∈ *e-proj*
gluing ‘‘ {((*x1*, *y1*), *1*)} ∈ *e-proj*
gluing ‘‘ {((*x2*, *y2*), *1*)} ∈ *e-proj*
gluing ‘‘ {((*x3*, *y3*), *1*)} ∈ *e-proj*
using *assms e-class e-points* **by** *blast+*
have *e-proj-add-0*: *proj-addition* (*gluing* ‘‘ {((*x1*, *y1*), *0*)}) (*gluing* ‘‘ {((*x2*, *y2*), *0*)}) ∈ *e-proj*
proj-addition (*gluing* ‘‘ {((*x2*, *y2*), *0*)}) (*gluing* ‘‘ {((*x3*, *y3*), *0*)}) ∈ *e-proj*
proj-addition (*gluing* ‘‘ {((*x2*, *y2*), *0*)}) (*gluing* ‘‘ {((*x3*, *y3*), *1*)}) ∈ *e-proj*
proj-addition (*gluing* ‘‘ {((*x1*, *y1*), *0*)}) (*gluing* ‘‘ {((*x2*, *y2*), *1*)}) ∈ *e-proj*
proj-addition (*gluing* ‘‘ {((*x2*, *y2*), *1*)}) (*gluing* ‘‘ {((*x3*, *y3*), *0*)}) ∈ *e-proj*
proj-addition (*gluing* ‘‘ {((*x2*, *y2*), *1*)}) (*gluing* ‘‘ {((*x3*, *y3*), *1*)}) ∈ *e-proj*
using *e-proj-0 well-defined proj-addition-def* **by** *blast+*

have *complex-e-proj*: *?t1* ∈ *e-proj*
?t2 ∈ *e-proj*
using *e-proj-0 e-proj-add-0 well-defined proj-addition-def* **by** *blast+*

have *eq3*: *?t1* = *?t2*
by(*subst assoc-with-zeros*,(*simp add*: *e-proj-0*)+)

show *?thesis*
proof(*cases l* = *0*)
case *True*
then have *l*: *l* = *0* **by** *simp*
then show *?thesis*
proof(*cases m* = *0*)
case *True*
then have *m*: *m* = *0* **by** *simp*

then show *?thesis*
proof(*cases n = 0*)
case *True*
then have *n*: *n = 0* **by** *simp*
show *?thesis*
using *l m n assms assoc-with-zeros* **by** *simp*
next
case *False*
then have *n*: *n = 1* **by** *simp*
have *eq1*: *proj-addition* (*proj-addition* (*gluing* ‘‘ {((*x1*, *y1*), *0*)}) (*gluing* ‘‘ {((*x2*, *y2*), *0*)}))
(*gluing* ‘‘ {((*x3*, *y3*), *1*)}) = *tf′* (*?t1*)

apply(*subst tf-tau*[*of - - 0*,*simplified*],*simp add*: *e-proj-0*)
by(*subst remove-add-tau′*,*auto simp add*: *e-proj-0 e-proj-add-0*)

have *eq2*: *proj-addition* (*gluing* ‘‘ {((*x1*, *y1*), *0*)})
(*proj-addition* (*gluing* ‘‘ {((*x2*, *y2*), *0*)}) (*gluing* ‘‘ {((*x3*, *y3*), *1*)})) =
tf′(*?t2*)
apply(*subst tf-tau*[*of - - 0*,*simplified*],*simp add*: *e-proj-0*)
apply(*subst remove-add-tau′*,(*simp add*: *e-proj-0*)+)
by(*subst remove-add-tau′*,(*simp add*: *e-proj-0 e-proj-add-0*)+)

show *?thesis*
apply(*simp add*: *l m n*)
using *eq1 eq2 eq3* **by** *argo*
qed
next
case *False*
then have *m*: *m = 1* **by** *simp*
then show *?thesis*
proof(*cases n = 0*)
case *True*
then have *n*: *n = 0* **by** *simp*

have *eq1*: *proj-addition* (*proj-addition* (*gluing* ‘‘ {((*x1*, *y1*), *0*)}) (*gluing* ‘‘ {((*x2*, *y2*), *1*)}))
(*gluing* ‘‘ {((*x3*, *y3*), *0*)}) = *tf′*(*?t1*)
apply(*subst tf-tau*[*of - - 0*,*simplified*],*simp add*: *e-proj-0*)
apply(*subst remove-add-tau′*,(*simp add*: *e-proj-0*)+)
by(*subst remove-add-tau*,(*simp add*: *e-proj-0 e-proj-add-0*)+)
have *eq2*: *proj-addition* (*gluing* ‘‘ {((*x1*, *y1*), *0*)})
(*proj-addition* (*gluing* ‘‘ {((*x2*, *y2*), *1*)}) (*gluing* ‘‘ {((*x3*, *y3*), *0*)})) =
tf′(*?t2*)
apply(*subst tf-tau*[*of - - 0*,*simplified*],*simp add*: *e-proj-0*)
apply(*subst remove-add-tau*,(*simp add*: *e-proj-0*)+)
by(*subst remove-add-tau′*,(*simp add*: *e-proj-0 e-proj-add-0*)+)

```
        then show ?thesis
          apply(simp add: l m n)
          using eq1 eq2 eq3 by argo
      next
        case False
        then have n: n = 1 by simp

        have eq1: proj-addition (proj-addition (gluing `` {((x1, y1), 0)}) (gluing ``
{((x2, y2), 1)}))
                  (gluing `` {((x3, y3), 1)}) = ?t1
          apply(subst tf-tau[of - - 0,simplified],simp add: e-proj-0)
          apply(subst remove-add-tau',(simp add: e-proj-0)+)
          apply(subst remove-add-tau,(simp add: e-proj-0 e-proj-add-0)+)
          apply(subst tf-tau[of - - 0,simplified],simp add: e-proj-0)
          apply(subst remove-add-tau',(simp add: e-proj-0 e-proj-add-0)+)
          by(subst tf'-idemp,auto simp add: complex-e-proj)

        have eq2: proj-addition (gluing `` {((x1, y1), 0)})
            (proj-addition (gluing `` {((x2, y2), 1)}) (gluing `` {((x3, y3), 1)})) =
                ?t2
          apply(subst tf-tau[of - - 0,simplified],simp add: e-proj-0)
          apply(subst remove-add-tau,(simp add: e-proj-0)+)
          apply(subst remove-add-tau',(simp add: e-proj-0 e-proj-add-0)+)
          apply(subst tf-tau[of - - 0,simplified],simp add: e-proj-0)
          apply(subst remove-add-tau',(simp add: e-proj-0 e-proj-add-0)+)
          apply(subst remove-add-tau',(simp add: e-proj-0 e-proj-add-0)+)
          by(subst tf'-idemp,auto simp add: complex-e-proj)

        then show ?thesis
          apply(simp add: l m n)
          using eq1 eq2 eq3 by argo
      qed
    qed
  next
    case False
    then have l: l = 1 by simp

    then show ?thesis
    proof(cases m = 0)
      case True
      then have m: m = 0 by simp
      then show ?thesis
      proof(cases n = 0)
        case True
        then have n: n = 0 by simp

        have eq1: proj-addition (proj-addition (gluing `` {((x1, y1), 1)}) (gluing ``
{((x2, y2), 0)}))
```

(*gluing* `` {((*x3*, *y3*), *0*)}) = *tf* '(*?t1*)
apply(*subst tf-tau*[*of - - 0*,*simplified*],*simp add*: *e-proj-0*)
apply(*subst remove-add-tau*,(*simp add*: *e-proj-0*)+)
by(*subst remove-add-tau*,(*simp add*: *e-proj-0 e-proj-add-0*)+)

have *eq2*: *proj-addition* (*gluing* `` {((*x1*, *y1*), *1*)})
(*proj-addition* (*gluing* `` {((*x2*, *y2*), *0*)}) (*gluing* `` {((*x3*, *y3*), *0*)})) =
tf '(*?t2*)
apply(*subst tf-tau*[*of - - 0*,*simplified*],*simp add*: *e-proj-0*)
by(*subst remove-add-tau*,(*simp add*: *e-proj-0 e-proj-add-0*)+)

then show *?thesis*
apply(*simp add*: *l m n*)
using *eq1 eq2 eq3* **by** *argo*
next
case *False*
then have *n*: *n* = *1* **by** *simp*
have *eq1*: *proj-addition* (*proj-addition* (*gluing* `` {((*x1*, *y1*), *1*)}) (*gluing* ``
{((*x2*, *y2*), *0*)}))
(*gluing* `` {((*x3*, *y3*), *1*)}) = *?t1*
apply(*subst tf-tau*[*of - - 0*,*simplified*],*simp add*: *e-proj-0*)
apply(*subst remove-add-tau*,(*simp add*: *e-proj-0*)+)
apply(*subst remove-add-tau*,(*simp add*: *e-proj-0 e-proj-add-0*)+)
apply(*subst tf-tau*[*of - - 0*,*simplified*],*simp add*: *e-proj-0*)
apply(*subst remove-add-tau'*,(*simp add*: *e-proj-0 e-proj-add-0*)+)
by(*subst tf '-idemp*,*auto simp add*: *complex-e-proj*)

have *eq2*: *proj-addition* (*gluing* `` {((*x1*, *y1*), *1*)})
(*proj-addition* (*gluing* `` {((*x2*, *y2*), *0*)}) (*gluing* `` {((*x3*, *y3*), *1*)})) =
?t2
apply(*subst tf-tau*[*of - - 0*,*simplified*],*simp add*: *e-proj-0*)
apply(*subst remove-add-tau*,(*simp add*: *e-proj-0 e-proj-add-0*)+)
apply(*subst tf-tau*[*of - - 0*,*simplified*],*simp add*: *e-proj-0*)
apply(*subst remove-add-tau'*,(*simp add*: *e-proj-0 e-proj-add-0*)+)
apply(*subst remove-add-tau'*,(*simp add*: *e-proj-0 e-proj-add-0*)+)
by(*subst tf '-idemp*,*auto simp add*: *complex-e-proj*)

then show *?thesis*
apply(*simp add*: *l m n*)
using *eq1 eq2 eq3* **by** *argo*
qed
next
case *False*
then have *m*: *m* = *1* **by** *simp*
then show *?thesis*
proof(*cases n* = *0*)
case *True*
then have *n*: *n* = *0* **by** *simp*

```
    have eq1: proj-addition (proj-addition (gluing ‘‘ {((x1, y1), 1)}) (gluing ‘‘
{((x2, y2), 1)}))
              (gluing ‘‘ {((x3, y3), 0)}) = ?t1
      apply(subst tf-tau[of - - 0,simplified],simp add: e-proj-0)
      apply(subst remove-add-tau,(simp add: e-proj-0 e-proj-add-0)+)
      apply(subst remove-add-tau,(simp add: e-proj-0 e-proj-add-0)+)
      apply(subst tf-tau[of - - 0,simplified],simp add: e-proj-0)
      apply(subst remove-add-tau',(simp add: e-proj-0 e-proj-add-0)+)
      apply(subst remove-add-tau,(simp add: e-proj-0 e-proj-add-0)+)
      by(subst tf'-idemp,auto simp add: complex-e-proj)

    have eq2: proj-addition (gluing ‘‘ {((x1, y1), 1)})
       (proj-addition (gluing ‘‘ {((x2, y2), 1)}) (gluing ‘‘ {((x3, y3), 0)})) =
          ?t2
      apply(subst tf-tau[of - - 0,simplified],simp add: e-proj-0)
      apply(subst remove-add-tau,(simp add: e-proj-0 e-proj-add-0)+)
      apply(subst tf-tau[of - - 0,simplified],simp add: e-proj-0)
      apply(subst remove-add-tau,(simp add: e-proj-0 e-proj-add-0)+)
      apply(subst remove-add-tau',(simp add: e-proj-0 e-proj-add-0)+)
      by(subst tf'-idemp,auto simp add: complex-e-proj)

    then show ?thesis
      apply(simp add: l m n)
      using eq1 eq2 eq3 by argo
  next
    case False
    then have n: n = 1 by simp

    have eq1: proj-addition (proj-addition (gluing ‘‘ {((x1, y1), 1)}) (gluing ‘‘
{((x2, y2), 1)}))
             (gluing ‘‘ {((x3, y3), 1)}) = tf'(?t1)
      apply(subst tf-tau[of - - 0,simplified],simp add: e-proj-0)
      apply(subst remove-add-tau,(simp add: e-proj-0 e-proj-add-0)+)
      apply(subst remove-add-tau,(simp add: e-proj-0 e-proj-add-0)+)
      apply(subst tf-tau[of - - 0,simplified],simp add: e-proj-0)
      apply(subst remove-add-tau',(simp add: e-proj-0 e-proj-add-0)+)
      apply(subst remove-add-tau,(simp add: e-proj-0 e-proj-add-0)+)
      apply(subst tf-tau[of - - 0,simplified],simp add: e-proj-0)
      apply(subst remove-add-tau',(simp add: e-proj-0 e-proj-add-0)+)
      by(subst tf'-idemp,auto simp add: complex-e-proj)

    have eq2: proj-addition (gluing ‘‘ {((x1, y1), 1)})
   (proj-addition (gluing ‘‘ {((x2, y2), 1)}) (gluing ‘‘ {((x3, y3), 1)})) =
             tf'(?t2)
      apply(subst tf-tau[of - - 0,simplified],simp add: e-proj-0)
      apply(subst remove-add-tau,(simp add: e-proj-0 e-proj-add-0)+)
      apply(subst tf-tau[of - - 0,simplified],simp add: e-proj-0)
      apply(subst remove-add-tau,(simp add: e-proj-0 e-proj-add-0)+)
      apply(subst remove-add-tau',(simp add: e-proj-0 e-proj-add-0)+)
```

```
      apply(subst tf-tau[of - - 0,simplified],simp add: e-proj-0)
      apply(subst remove-add-tau',(simp add: e-proj-0 e-proj-add-0)+)
      apply(subst remove-add-tau',(simp add: e-proj-0 e-proj-add-0)+)
      by(subst tf'-idemp,auto simp add: complex-e-proj)

     then show ?thesis
      apply(simp add: l m n)
      using eq1 eq2 eq3 by argo
   qed
  qed
 qed
qed

lemma proj-assoc:
 assumes x ∈ e-proj y ∈ e-proj z ∈ e-proj
 shows proj-addition (proj-addition x y) z = proj-addition x (proj-addition y z)
proof −
 obtain x1 y1 l x2 y2 m x3 y3 n where
  x = gluing `` {((x1, y1), l)}
  y = gluing `` {((x2, y2), m)}
  z = gluing `` {((x3, y3), n)}
  by (metis assms e-proj-def prod.collapse quotientE)

 then show ?thesis
  using assms general-assoc by force
qed
```

4.5 Group law

```
lemma projective-group-law:
 shows comm-group (|carrier = e-proj, mult = proj-addition, one = {((1,0),0)}|)

proof(unfold-locales,simp-all)
 show one-in: {((1, 0), 0)} ∈ e-proj
  using identity-proj by auto

 show comm: proj-addition x y = proj-addition y x
        if x ∈ e-proj y ∈ e-proj for x y
  using proj-addition-comm that by simp

 show id-1: proj-addition {((1, 0), 0)} x = x
        if x ∈ e-proj for x
  using proj-add-class-identity that by simp

 show id-2: proj-addition x {((1, 0), 0)} = x
        if x ∈ e-proj for x
   using comm id-1 one-in that by simp

 show e-proj ⊆ Units (|carrier = e-proj, mult = proj-addition, one = {((1, 0),
```

$0)\}\rangle$
unfolding *Units-def*
proof(*simp*,*standard*)
fix x
assume $x \in$ *e-proj*
then obtain x' y' l' **where** $x = gluing$ `` $\{((x', y'), l')\}$
unfolding *e-proj-def*
apply(*elim quotientE*)
by *auto*
then have *proj-addition* (*gluing* `` $\{(i\ (x', y'), l')\}$)
(*gluing* `` $\{((x', y'), l')\}$) =
$\{((1, 0), 0)\}$
proj-addition (*gluing* `` $\{((x', y'), l')\}$)
(*gluing* `` $\{(i\ (x', y'), l')\}$) =
$\{((1, 0), 0)\}$
gluing `` $\{(i\ (x', y'), l')\} \in$ *e-proj*
using *proj-add-class-inv proj-addition-comm* ‹$x \in$ *e-proj*› **by** *simp*+
then show $x \in \{y \in$ *e-proj*. $\exists x \in$*e-proj*. *proj-addition* x y = $\{((1, 0), 0)\} \wedge$
proj-addition y x = $\{((1, 0), 0)\}\}$
using ‹$x = gluing$ `` $\{((x', y'), l')\}$› ‹$x \in$ *e-proj*› **by** *blast*
qed

show *proj-addition* x $y \in$ *e-proj*
if $x \in$ *e-proj* $y \in$ *e-proj* **for** x y
using *well-defined that* **by** *blast*

show *proj-addition* (*proj-addition* x y) z = *proj-addition* x (*proj-addition* y z)
if $x \in$ *e-proj* $y \in$ *e-proj* $z \in$ *e-proj* **for** x y z
using *proj-assoc that* **by** *simp*
qed

end

end