

Group Law of Edwards Elliptic Curves

Rodrigo Raya

April 18, 2024

Abstract

This article gives an elementary computational proof of the group law for Edwards elliptic curves. The associative law is expressed as a polynomial identity over the integers that is directly checked by polynomial division. Unlike other proofs, no preliminaries such as intersection numbers, Bezouts theorem, projective geometry, divisors, or Riemann Roch are required. It supports the material of [1].

Contents

1	Affine Edwards curves	2
2	Extension	9
2.1	Change of variables	9
2.2	New points	9
2.3	Group transformations and inversions	10
2.4	Extended addition	15
2.4.1	Inversion and rotation invariance	16
2.4.2	Coherence and closure	19
2.4.3	Useful lemmas in the extension	21
2.5	Delta arithmetic	21
3	Projective Edwards curves	26
3.1	No fixed-point lemma and dichotomies	26
3.1.1	Meaning of dichotomy condition on deltas	31
3.2	Gluing relation and projective points	32
3.2.1	Point-class classification	33
3.3	Projective addition on points	36
3.4	Projective addition on classes	39
3.4.1	Covering	39
3.4.2	Independence of the representant	45
3.4.3	Basic properties	74

4	Group law	79
4.1	Class invariance on group operations	79
4.2	Associativities	98
4.3	Lemmas for associativity	108
4.4	Group law	164
theory	<i>Edwards-Elliptic-Curves-Group</i>	
	imports <i>HOL-Algebra.Group HOL-Library.Rewrite</i>	
	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 = x2 + c * y2 - 1 - d * x2 * y2
```

```
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
apply algebra
done
```

```
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:  $x3 = (x1*x2 - c*y1*y2)\ \text{div}\ (\text{delta-minus}\ x1\ y1\ x2\ y2)$ 
    using assms delta-minus-def by auto
  have  $y3$ -expr:  $y3 = (x1*y2 + y1*x2)\ \text{div}\ (\text{delta-plus}\ x1\ y1\ x2\ y2)$ 
    using assms delta-plus-def by auto

  have  $\exists\ r1\ r2.\ (e\ x3\ y3)*(delta\ x1\ y1\ x2\ y2)^2 - (r1 * e\ x1\ y1 + r2 * e\ x2\ y2)$ 
  = 0
    unfolding e-def x3-expr y3-expr delta-def
    apply(simp add: divide-simps assms)
    unfolding delta-plus-def delta-minus-def
    by algebra
  then show  $e\ x3\ y3 = 0$ 
    using assms
    by (simp add: delta-def)
qed

```

lemma *associativity*:

```

assumes  $z1' = (x1',y1')\ z3' = (x3',y3')$ 
assumes  $z1' = \text{add}\ (x1,y1)\ (x2,y2)\ z3' = \text{add}\ (x2,y2)\ (x3,y3)$ 
assumes  $\text{delta-minus}\ x1\ y1\ x2\ y2 \neq 0\ \text{delta-plus}\ x1\ y1\ x2\ y2 \neq 0$ 
   $\text{delta-minus}\ x2\ y2\ x3\ y3 \neq 0\ \text{delta-plus}\ x2\ y2\ x3\ y3 \neq 0$ 
   $\text{delta-minus}\ x1'\ y1'\ x3\ y3 \neq 0\ \text{delta-plus}\ x1'\ y1'\ x3\ y3 \neq 0$ 
   $\text{delta-minus}\ x1\ y1\ x3'\ y3' \neq 0\ \text{delta-plus}\ x1\ y1\ x3'\ y3' \neq 0$ 
assumes  $e\ x1\ y1 = 0\ e\ x2\ y2 = 0\ e\ x3\ y3 = 0$ 
shows  $\text{add}\ (\text{add}\ (x1,y1)\ (x2,y2))\ (x3,y3) = \text{add}\ (x1,y1)\ (\text{add}\ (x2,y2)\ (x3,y3))$ 
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  $\Delta_x =$ 
     $(\text{delta-minus}\ x1'\ y1'\ x3\ y3)*(\text{delta-minus}\ x1\ y1\ x3'\ y3')*$ 
     $(\text{delta}\ x1\ y1\ x2\ y2)*(\text{delta}\ x2\ y2\ x3\ y3)$ 
  define  $\Delta_y$  where  $\Delta_y =$ 
     $(\text{delta-plus}\ x1'\ y1'\ x3\ y3)*(\text{delta-plus}\ x1\ y1\ x3'\ y3')*$ 
     $(\text{delta}\ x1\ y1\ x2\ y2)*(\text{delta}\ x2\ y2\ x3\ y3)$ 
  define  $g_x$  where  $g_x = \text{fst}(\text{add}\ z1'\ (x3,y3)) - \text{fst}(\text{add}\ (x1,y1)\ z3')$ 
  define  $g_y$  where  $g_y = \text{snd}(\text{add}\ z1'\ (x3,y3)) - \text{snd}(\text{add}\ (x1,y1)\ z3')$ 
  define  $g_{xpoly}$  where  $g_{xpoly} = g_x * \Delta_x$ 
  define  $g_{ypoly}$  where  $g_{ypoly} = g_y * \Delta_y$ 

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

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

apply($rewrite\ x1'-expr\ y1'-expr\ x3'-expr\ y3'-expr$)
apply($rewrite\ delta-minus-def$)
apply($rewrite\ in\ - / \sqcap\ delta-minus-def[symmetric]\ delta-plus-def[symmetric]$)
unfolding $delta-def$
by($simp\ add: divide-simps\ assms(5-8)$)

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($rewrite\ x1'-expr\ y1'-expr\ x3'-expr\ y3'-expr$)
apply($rewrite\ delta-minus-def$)
apply($rewrite\ in\ - / \sqcap\ delta-minus-def[symmetric]\ delta-plus-def[symmetric]$)
unfolding $delta-def$
by($simp\ add: divide-simps\ assms(5-8)$)

have $\exists\ r1\ r2\ r3. gxpoly = r1 * e1 + r2 * e2 + r3 * e3$
unfolding $gxpoly-def\ gx-def\ Delta_x-def$
apply($simp\ add: assms(1,2)$)
apply($rewrite\ in\ - / \sqcap\ delta-minus-def[symmetric]$)
apply($simp\ add: divide-simps\ assms(9,11)$)
apply($rewrite\ left-diff-distrib$)
apply($simp\ add: simp1gx\ simp2gx$)
unfolding $delta-plus-def\ delta-minus-def\ e1-def\ e2-def\ e3-def\ e-def$
by $algebra$

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 $gx = 0$

```

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(rewrite x1'-expr y1'-expr x3'-expr y3'-expr)+
  apply(rewrite delta-plus-def)
  apply(rewrite in - / □ delta-minus-def[symmetric] delta-plus-def[symmetric])+
  unfolding 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 * 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(rewrite x1'-expr y1'-expr x3'-expr y3'-expr)+
  apply(rewrite delta-plus-def)
  apply(rewrite in - / □ delta-minus-def[symmetric] delta-plus-def[symmetric])+
  unfolding delta-def
  by(simp add: divide-simps assms(5-8))

have ∃ r1 r2 r3. gypoly = r1 * e1 + r2 * e2 + r3 * e3
  unfolding gypoly-def gy-def Deltay-def
  apply(simp add: assms(1,2))
  apply(rewrite in - / □ delta-plus-def[symmetric])+
  apply(simp add: divide-simps assms(10,12))
  apply(rewrite left-diff-distrib)
  apply(simp add: simp1gy simp2gy)
  unfolding delta-plus-def delta-minus-def
    e1-def e2-def e3-def e-def
  by algebra

then have gypoly = 0
  using e1-def assms(13-15) e2-def e3-def by auto
have Deltay ≠ 0
  using Deltay-def delta-def assms(7-12) non-unfolded-adds by auto
then have gy = 0
  using ⟨gypoly = 0⟩ gypoly-def by auto

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

```

lemma neutral: $\text{add } z (1,0) = z$ **by** (cases z,simp)

lemma inverse:

assumes $e \ a \ b = 0$ *delta-plus* $a \ b \ a \ b \neq 0$
shows $\text{add } (a,b) (a,-b) = (1,0)$
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 $\exists b. (1/d = b^2 \wedge 1/d \neq 0) \vee (1/(c*d) = b^2 \wedge 1/(c*d) \neq 0)$
proof –
define r **where** $r = (1 - c*d*y1^2*y2^2) * (1 - d*y1^2*x2^2)$
define $e1$ **where** $e1 = e \ x1 \ y1$
define $e2$ **where** $e2 = e \ x2 \ y2$
have $r = d^2 * y1^2 * y2^2 * x2^2 * e1 + (1 - d * y1^2) * \text{delta } x1 \ y1 \ x2 \ y2 - d * y1^2 * e2$
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: $(1 - c*d*y1^2*y2^2) = 0 \vee (1 - d*y1^2*x2^2) = 0$
using *r-def* **by** *auto*
have $d \neq 0$ **using** $\langle r = 0 \rangle$ *r-def* **by** *auto*
{
assume $(1 - d*y1^2*x2^2) = 0$
then have $1/d = y1^2*x2^2$ $1/d \neq 0$
apply(*auto simp add: divide-simps* $\langle d \neq 0 \rangle$)
by *algebra*
}
note *case1 = this*
{**assume** $(1 - c*d*y1^2*y2^2) = 0$ $(1 - d*y1^2*x2^2) \neq 0$
then have $c \neq 0$ **by** *auto*
then have $1/(c*d) = y1^2*y2^2$ $1/(c*d) \neq 0$
apply(*simp add: divide-simps* $\langle d \neq 0 \rangle \langle c \neq 0 \rangle$)
using $\langle (1 - c*d*y1^2*y2^2) = 0 \rangle$ **apply** *algebra*
using $\langle c \neq 0 \rangle \langle d \neq 0 \rangle$ **by** *auto*
}
note *case2 = this*

show $\exists b. (1/d = b^2 \wedge 1/d \neq 0) \vee (1/(c*d) = b^2 \wedge 1/(c*d) \neq 0)$
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  $\exists b. 1/c = b^2 \neg (\exists b. b \neq 0 \wedge 1/d = b^2)$ 
shows  $\text{delta } x1 \ y1 \ x2 \ y2 \neq 0$ 
proof(rule ccontr)
  assume  $\neg \text{delta } x1 \ y1 \ x2 \ y2 \neq 0$ 
  then have  $\text{delta } x1 \ y1 \ x2 \ y2 = 0$  by blast
  then have  $\exists b. (1/d = b^2 \wedge 1/d \neq 0) \vee (1/(c*d) = b^2 \wedge 1/(c*d) \neq 0)$ 
    using affine-closure[OF  $\langle \text{delta } x1 \ y1 \ x2 \ y2 = 0 \rangle$ 
       $\langle e \ x1 \ y1 = 0 \rangle \langle e \ x2 \ y2 = 0 \rangle$ ] by blast
  then obtain  $b$  where  $(1/(c*d) = b^2 \wedge 1/(c*d) \neq 0)$ 
    using  $\langle \neg (\exists b. b \neq 0 \wedge 1/d = b^2) \rangle$  by fastforce
  then have  $1/c \neq 0 \ c \neq 0 \ d \neq 0 \ 1/d \neq 0$  by simp+
  then have  $1/d = b^2 / (1/c)$ 
    apply(simp add: divide-simps)
  by (metis  $\langle 1 / (c * d) = b^2 \wedge 1 / (c * d) \neq 0 \rangle$  eq-divide-eq semiring-normalization-rules(18))
  then have  $\exists b. b \neq 0 \wedge 1/d = b^2$ 
    using assms(3)
  by (metis  $\langle 1 / d \neq 0 \rangle$  power-divide zero-power2)
  then show False
    using  $\langle \neg (\exists b. b \neq 0 \wedge 1/d = b^2) \rangle$  by blast
qed

```

lemma group-law:

```

assumes  $\exists b. 1/c = b^2 \neg (\exists b. b \neq 0 \wedge 1/d = b^2)$ 
shows comm-group ( $\text{carrier} = \{(x,y). e \ x \ y = 0\}$ ,  $\text{mult} = \text{add}$ ,  $\text{one} = (1,0)$ )
  (is comm-group ?g)
proof(unfold-locales)
  {fix  $x1 \ y1 \ x2 \ y2$ 
  assume  $e \ x1 \ y1 = 0 \ e \ x2 \ y2 = 0$ 
  have  $e \ (\text{fst} \ (\text{add} \ (x1,y1) \ (x2,y2))) \ (\text{snd} \ (\text{add} \ (x1,y1) \ (x2,y2))) = 0$ 
    apply(simp)
  using add-closure delta-non-zero[OF  $\langle e \ x1 \ y1 = 0 \rangle \langle e \ x2 \ y2 = 0 \rangle$  assms(1)
  assms(2)]
  delta-def  $\langle e \ x1 \ y1 = 0 \rangle \langle e \ x2 \ y2 = 0 \rangle$  by auto}
  then show
     $\bigwedge x \ y. x \in \text{carrier } ?g \implies y \in \text{carrier } ?g \implies$ 
       $x \otimes_{?g} y \in \text{carrier } ?g$  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  $\text{delta } x1 \ y1 \ x2 \ y2 \neq 0 \ \text{delta } x2 \ y2 \ x3 \ y3 \neq 0$ 
  using assms(1,2) delta-non-zero by blast+
fix  $x1' \ y1' \ x3' \ y3'$ 
assume  $(x1',y1') = \text{add} \ (x1,y1) \ (x2,y2)$ 
   $(x3',y3') = \text{add} \ (x2,y2) \ (x3,y3)$ 
then have  $e \ x1' \ y1' = 0 \ e \ x3' \ y3' = 0$ 
  using add-closure  $\langle \text{delta } x1 \ y1 \ x2 \ y2 \neq 0 \rangle \langle \text{delta } x2 \ y2 \ x3 \ y3 \neq 0 \rangle$ 
   $\langle e \ x1 \ y1 = 0 \rangle \langle e \ x2 \ y2 = 0 \rangle \langle e \ x3 \ y3 = 0 \rangle$  delta-def by fastforce+
then have  $\text{delta } x1' \ y1' \ x3 \ y3 \neq 0 \ \text{delta } x1 \ y1 \ x3' \ y3' \neq 0$ 
  using assms delta-non-zero  $\langle e \ x3 \ y3 = 0 \rangle$  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 ?g ⇒ y ∈ carrier ?g ⇒ z ∈ carrier ?g ⇒
  x ⊗ ?g y ⊗ ?g z = x ⊗ ?g (y ⊗ ?g z) by auto
next
show 1 ?g ∈ carrier ?g by (simp add: e-def)
next
show ∧ x. x ∈ carrier ?g ⇒ 1 ?g ⊗ ?g x = x
  by (simp add: commutativity neutral)
next
show ∧ x. x ∈ carrier ?g ⇒ x ⊗ ?g 1 ?g = x
  by (simp add: neutral)
next
show ∧ x y. x ∈ carrier ?g ⇒ y ∈ carrier ?g ⇒
  x ⊗ ?g y = y ⊗ ?g x
  using commutativity by auto
next
show carrier ?g ⊆ Units ?g
proof (simp, standard)
  fix z
  assume z ∈ {(x, y). local.e x y = 0}
  show z ∈ Units ?g
  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)
  end
end

```



```

    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'^2
  assumes t-ineq: t'^2 ≠ 1 t' ≠ 0
begin

```

2.1 Change of variables

```

definition t where t = t'

```

```

lemma t-nz: t ≠ 0 using t-ineq(2) t-def by auto

```

```

lemma d-nz: d ≠ 0 using t-nz t-ineq t-intro by simp

```

```

lemma t-expr: t^2 = d t^4 = d^2 using t-intro t-def by auto

```

```

lemma t-sq-n1: t^2 ≠ 1 using t-ineq(1) t-def by simp

```

```

lemma t-nm1: t ≠ -1 using t-sq-n1 by fastforce

```

```

lemma d-n1: d ≠ 1 using t-sq-n1 t-expr by blast

```

```

lemma t-n1: t ≠ 1 using t-sq-n1 by fastforce

```

```

lemma t-dneq2: 2*t ≠ -2
proof(rule ccontr)
  assume ¬ 2 * t ≠ -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^2 + y^2 - 1 - t^2 * x^2 * y^2

```

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

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

lemma $\text{circ-to-aff}: p \in e\text{-circ} \implies p \in e'\text{-aff}$
unfolding $e\text{-circ-def}$ **by** auto

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

lemma $t\text{-}2\text{-}1\text{-lines}$:
 $t^{\wedge}2 = 1 \implies e' x y = -(1 - x^{\wedge}2) * (1 - y^{\wedge}2)$
unfolding $e'\text{-def}$ **by** algebra

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

lemma $t\text{-}0\text{-circle}$:
 $t = 0 \implies e' x y = x^{\wedge}2 + y^{\wedge}2 - 1$
unfolding $e'\text{-def}$ **by** auto

2.3 Group transformations and inversions

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

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

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

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

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

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

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

lemma $\text{tau-idemp-explicit}$: $\tau(\tau(x,y)) = (x,y)$
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 × 'a ⇒ 'a × 'a where
  i (a,b) = (a,-b)

lemma i-idemp: i ∘ 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 ∈ 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 ∈ 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 ∈ rotations
shows  $\exists$  tr' ∈ rotations. (tr ∘ i) (x,y) = (i ∘ tr') (x,y) ∧ tr ∘ tr' = id
using assms rho-i-com-inverses unfolding rotations-def by fastforce

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

lemma rho-circ:
assumes p ∈ e-circ
shows  $\varrho$  p ∈ 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 $p \in e'\text{-aff}$
shows $i\ p \in e'\text{-aff}$
using *assms* **unfolding** *e'-aff-def e'-def* **by** *auto*

lemma *i-circ*:
assumes $(x,y) \in e\text{-circ}$
shows $i\ (x,y) \in e\text{-circ}$
using *assms* **unfolding** *e-circ-def e'-aff-def e'-def* **by** *auto*

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

lemma *rot-circ*:
assumes $p \in e\text{-circ}$ $tr \in \text{rotations}$
shows $tr\ p \in e\text{-circ}$
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 $\tau\text{-circ}$:
assumes $p \in e\text{-circ}$
shows $\tau\ p \in e\text{-circ}$
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*)
by(*simp add: algebra-simps*)

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

lemma *rot-tau-com*:
assumes $tr \in \text{rotations}$
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 $r \in \text{rotations}$ $r' \in \text{rotations}$

shows $r' \circ r = r \circ r'$

using *assms* **unfolding** *rotations-def* **by** *force*

lemma *rot-inv*:

assumes $r \in \text{rotations}$

shows $\exists r' \in \text{rotations}. r' \circ r = \text{id}$

using *assms* **unfolding** *rotations-def* **by** *force*

lemma *rot-aff*:

assumes $r \in \text{rotations}$ $p \in e'\text{-aff}$

shows $r p \in e'\text{-aff}$

using *assms* **unfolding** *rotations-def* *e'-aff-def* *e'-def*

by(*auto simp add: semiring-normalization-rules(16) add commute*)

lemma *rot-delta*:

assumes $r \in \text{rotations}$ $\text{delta } x1 \ y1 \ x2 \ y2 \neq 0$

shows $\text{delta } (\text{fst } (r \ (x1, y1))) \ (\text{snd } (r \ (x1, y1))) \ x2 \ y2 \neq 0$

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*: $\tau \neq \text{id}$

apply(*simp add: fun-eq-iff*)

apply(*simp add: divide-simps t-nz*)

apply(*simp add: field-simps*)

apply(*rule exI[of - 1]*)

by(*simp add: t-n1*)

lemma *sym-not-id*:

assumes $r \in \text{rotations}$

shows $\tau \circ r \neq \text{id}$

using *assms* **unfolding** *rotations-def*

apply(*subst fun-eq-iff, simp*)

apply(*safe*)

apply(*auto*)

apply(*simp-all add: divide-simps*)

apply(*rule exI[of - 1]*)

apply (*simp add: t-n1*)

apply(*rule exI[of - 1]*)

apply(*simp add: d-nz*)

apply *blast*

```

apply(rule exI[of - 1])
apply(simp add: d-nz)
using t-nm1 apply presburger
using t-ineq(2) by blast

```

lemma *sym-decomp*:

```

assumes g ∈ symmetries
shows ∃ r ∈ rotations. g = τ ∘ r
using assms unfolding symmetries-def rotations-def by auto

```

lemma *symmetries-i-inverse*:

```

assumes tr ∈ symmetries
shows ∃ tr' ∈ symmetries. (tr ∘ i) (x,y) = (i ∘ tr') (x,y) ∧ tr ∘ tr' = id
proof -

```

```

consider (1) tr = τ |
          (2) tr = τ ∘ ρ |
          (3) tr = τ ∘ ρ ∘ ρ |
          (4) tr = τ ∘ ρ ∘ ρ ∘ ρ |
using assms unfolding symmetries-def by blast

```

then show ?thesis

proof(cases)

case 1

```

define tr' where tr' = τ
have (tr ∘ i) (x, y) = (i ∘ tr') (x, y) ∧ tr ∘ tr' = id tr' ∈ symmetries
using tr'-def 1 tau-idemp symmetries-def by simp+
then show ?thesis by blast

```

next

case 2

```

define tr' where tr' = τ ∘ ρ ∘ ρ ∘ ρ
have (tr ∘ i) (x, y) = (i ∘ tr') (x, y) ∧ tr ∘ tr' = id tr' ∈ symmetries
using tr'-def 2
apply(simp)
using tau-idemp-point apply fastforce
using symmetries-def tr'-def by simp
then show ?thesis by blast

```

next

case 3

```

define tr' where tr' = τ ∘ ρ ∘ ρ
have (tr ∘ i) (x, y) = (i ∘ tr') (x, y) ∧ tr ∘ tr' = id tr' ∈ symmetries
using tr'-def 3
apply(simp)
using tau-idemp-point apply fastforce
using symmetries-def tr'-def by simp
then show ?thesis by blast

```

next

case 4

```

define tr' where tr' = τ ∘ ρ
have (tr ∘ i) (x, y) = (i ∘ tr') (x, y) ∧ tr ∘ tr' = id tr' ∈ symmetries
using tr'-def 4

```

```

    apply(simp)
    using tau-idemp-point apply fastforce
    using symmetries-def tr'-def by simp
    then show ?thesis by blast
qed
qed

```

```

lemma sym-to-rot:  $g \in \text{symmetries} \implies \tau \circ g \in \text{rotations}$ 
  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:  $(\text{delta}' x0 y0 x1 y1 = 0) = (\text{delta}' x1 y1 x0 y0 = 0)$ 
  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*:

$x \neq 0 \implies y \neq 0 \implies \text{ext-add } (x,y) (i (x,y)) = (1,0)$
by(*simp add: two-not-zero*)

lemma *ext-add-deltas*:

$\text{ext-add } (x1,y1) (x2,y2) =$
 $((\text{delta-x } x2 y1 x1 y2) \text{ div } (\text{delta-x } x1 y1 x2 y2),$
 $(\text{delta-y } x1 x2 y1 y2) \text{ div } (\text{delta-y } x1 y1 x2 y2))$
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 $\text{add } (\tau (x1,y1)) (x2,y2) = \text{add } (x1,y1) (\tau (x2,y2))$
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 $\text{ext-add } (\tau (x1,y1)) (x2,y2) = \text{ext-add } (x1,y1) (\tau (x2,y2))$
apply(*simp add: divide-simps t-nz assms*)
by *algebra*

lemma *rho-invariance-1*:

$\text{add } (\rho (x1,y1)) (x2,y2) = \rho (\text{add } (x1,y1) (x2,y2))$
apply(*simp*)
apply(*subst c-eq-1*)
apply(*simp add: divide-simps*)
by(*simp add: algebra-simps*)

lemma *rho-invariance-1-points*:

$\text{add } (\rho p1) p2 = \rho (\text{add } p1 p2)$
using *rho-invariance-1*
apply(*subst* (2 4 6 8) *surjective-pairing*)
by *blast*

lemma *rho-invariance-2*:


```

ext-add (ρ (x1,y1)) (x2,y2) =
  ρ (ext-add (x1,y1) (x2,y2))
apply(simp add: divide-simps)
by(simp add: algebra-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-1:
  assumes r ∈ 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)
  apply(simp)
  apply(simp add: divide-simps)
  apply(simp add: divide-simps)
  apply(simp add: algebra-simps)
  by (simp add: c-eq-1)

```

```

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 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)
  apply(simp add: divide-simps)
  apply(simp add: algebra-simps)
  apply (simp add: add-eq-0-iff)
  apply(simp add: divide-simps)
  apply(simp add: algebra-simps)
  using neg-eq-iff-add-eq-0 by blast

```

```

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*:
 $\text{delta } x1 \ y1 \ (\text{fst } (\varrho \ (x2, y2))) \ (\text{snd } (\varrho \ (x2, y2))) =$
 $\text{delta } x1 \ y1 \ x2 \ y2$
by(*simp add: delta-def delta-plus-def delta-minus-def, algebra*)

lemma *rotation-invariance-4*:
 $\text{delta}' \ x1 \ y1 \ (\text{fst } (\varrho \ (x2, y2))) \ (\text{snd } (\varrho \ (x2, y2))) = - \ \text{delta}' \ x1 \ y1 \ x2 \ y2$
by(*simp add: delta'-def delta-x-def delta-y-def, algebra*)

lemma *rotation-invariance-5*:
 $\text{delta}' \ (\text{fst } (\varrho \ (x1, y1))) \ (\text{snd } (\varrho \ (x1, y1))) \ x2 \ y2 = - \ \text{delta}' \ x1 \ y1 \ x2 \ y2$
by(*simp add: delta'-def delta-x-def delta-y-def, algebra*)

lemma *rotation-invariance-6*:
 $\text{delta} \ (\text{fst } (\varrho \ (x1, y1))) \ (\text{snd } (\varrho \ (x1, y1))) \ x2 \ y2 = \text{delta} \ x1 \ y1 \ x2 \ y2$
by(*simp add: delta-def delta-plus-def delta-minus-def, algebra*)

lemma *inverse-rule-1*:
 $(\tau \circ i \circ \tau) \ (x, y) = i \ (x, y)$ **by** (*simp add: t-nz*)

lemma *inverse-rule-2*:
 $(\varrho \circ i \circ \varrho) \ (x, y) = i \ (x, y)$ **by** *simp*

lemma *inverse-rule-3*:
 $i \ (\text{add} \ (x1, y1) \ (x2, y2)) = \text{add} \ (i \ (x1, y1)) \ (i \ (x2, y2))$
by(*simp add: divide-simps*)

lemma *inverse-rule-4*:
 $i \ (\text{ext-add} \ (x1, y1) \ (x2, y2)) = \text{ext-add} \ (i \ (x1, y1)) \ (i \ (x2, y2))$
apply(*simp add: divide-simps*)
by(*simp add: algebra-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** $\tau(x1, y1) = (x1', y1')$ **by** *simp*
then have $p\text{-}nz: x1' \neq 0$ $y1' \neq 0$
using *assms(1) tau-sq apply auto[1]*
using $\langle \tau(x1, y1) = (x1', y1') \rangle$ *assms(2) tau-sq by auto*
have $add(x1, y1)(x2, y2) = add(\tau(x1', y1'))(x2, y2)$
using *c-eq-1 tau-expr tau-idemp-point by auto*
also have $\dots = \tau(ext\text{-}add(x1', y1')(x2, y2))$
using *add-ext-add[OF p-nz] tau-idemp by simp*
also have $\dots = \tau(ext\text{-}add(\tau(x1, y1))(x2, y2))$
using *tau-expr tau-idemp by auto*
finally show *?thesis by blast*
qed

2.4.2 Coherence and closure

lemma *coherence-1*:
assumes $\delta\text{-}x$ $x1$ $y1$ $x2$ $y2 \neq 0$ $\delta\text{-}minus$ $x1$ $y1$ $x2$ $y2 \neq 0$
assumes e' $x1$ $y1 = 0$ e' $x2$ $y2 = 0$
shows $\delta\text{-}x$ $x1$ $y1$ $x2$ $y2 * \delta\text{-}minus$ $x1$ $y1$ $x2$ $y2 *$
 $(fst(ext\text{-}add(x1, y1)(x2, y2)) - fst(add(x1, y1)(x2, y2)))$
 $= x2 * y2 * e' x1 y1 - x1 * y1 * e' x2 y2$
apply(*simp*)
apply(*rewrite in - / \sqsupseteq delta-x-def[symmetric]*)
apply(*rewrite in - / \sqsupseteq 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(*simp add: t-expr*)
by(*simp add: power2-eq-square field-simps*)

lemma coherence-2:
assumes $\text{delta-y } x1 \ y1 \ x2 \ y2 \neq 0$ $\text{delta-plus } x1 \ y1 \ x2 \ y2 \neq 0$
assumes $e' \ x1 \ y1 = 0$ $e' \ x2 \ y2 = 0$
shows $\text{delta-y } x1 \ y1 \ x2 \ y2 * \text{delta-plus } x1 \ y1 \ x2 \ y2 *$
 $(\text{snd } (\text{ext-add } (x1,y1) (x2,y2)) - \text{snd } (\text{add } (x1,y1) (x2,y2)))$
 $= - x2 * y2 * e' \ x1 \ y1 - x1 * y1 * e' \ x2 \ y2$
apply(*simp*)
apply(*rewrite in - /* \sqsupset *delta-y-def[symmetric]*)
apply(*rewrite in - /* \sqsupset *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 $\text{delta } x1 \ y1 \ x2 \ y2 \neq 0$ $\text{delta}' \ x1 \ y1 \ x2 \ y2 \neq 0$
assumes $e' \ x1 \ y1 = 0$ $e' \ x2 \ y2 = 0$
shows $\text{ext-add } (x1,y1) (x2,y2) = \text{add } (x1,y1) (x2,y2)$
using *coherence-1 coherence-2 delta-def delta'-def assms by auto*

lemma ext-add-closure:
assumes $\text{delta}' \ x1 \ y1 \ x2 \ y2 \neq 0$
assumes $e' \ x1 \ y1 = 0$ $e' \ x2 \ y2 = 0$
assumes $(x3,y3) = \text{ext-add } (x1,y1) (x2,y2)$
shows $e' \ x3 \ y3 = 0$
proof –
have *deltas-nz: delta-x* $x1 \ y1 \ x2 \ y2 \neq 0$
 $\text{delta-y } x1 \ y1 \ x2 \ y2 \neq 0$
using *assms(1) delta'-def by auto*

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

have $\exists a \ b. t^4 * (\text{delta-x } x1 \ y1 \ x2 \ y2)^2 * (\text{delta-y } x1 \ y1 \ x2 \ y2)^2 * e' \ x3 \ y3$
 $=$
 $a * e' \ x1 \ y1 + b * e' \ x2 \ y2$
using *deltas-nz*
unfolding *e'-def v3 delta-x-def delta-y-def*
apply(*simp add: divide-simps*)
by algebra

then show $e' \ x3 \ y3 = 0$
using *assms(2,3) deltas-nz t-nz by auto*
qed

lemma ext-add-closure-points:
assumes $\text{delta}' \ x1 \ y1 \ x2 \ y2 \neq 0$
assumes $(x1,y1) \in e'\text{-aff } (x2,y2) \in e'\text{-aff}$

shows $ext\text{-}add\ (x1,y1)\ (x2,y2) \in e'\text{-}aff$
using $ext\text{-}add\text{-}closure\ assms$
unfolding $e'\text{-}aff\text{-}def$ **by** $auto$

2.4.3 Useful lemmas in the extension

lemma $inverse\text{-}generalized$:
assumes $(a,b) \in e'\text{-}aff\ delta\text{-}plus\ a\ b\ a\ b \neq 0$
shows $add\ (a,b)\ (a,-b) = (1,0)$
using $inverse\ assms$
unfolding $e'\text{-}aff\text{-}def$
using $e\text{-}e'\text{-}iff$
by $(simp)$

lemma $inverse\text{-}generalized\text{-}points$:
assumes $p \in e'\text{-}aff\ delta\text{-}plus\ (fst\ p)\ (snd\ p)\ (fst\ p)\ (snd\ p) \neq 0$
shows $add\ p\ (i\ p) = (1,0)$
using $inverse\ assms$
unfolding $e'\text{-}aff\text{-}def$
using $e\text{-}e'\text{-}iff\ e'\text{-}aff\text{-}def$ **by** $auto$

lemma $add\text{-}closure\text{-}points$:
assumes $delta\ x\ y\ x'\ y' \neq 0$
 $(x,y) \in e'\text{-}aff\ (x',y') \in e'\text{-}aff$
shows $add\ (x,y)\ (x',y') \in e'\text{-}aff$
using $add\text{-}closure\ assms\ e\text{-}e'\text{-}iff$
unfolding $delta\text{-}def\ e'\text{-}aff\text{-}def$ **by** $auto$

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

2.5 Delta arithmetic

lemma $mix\text{-}tau$:
assumes $(x1,y1) \in e'\text{-}aff\ (x2,y2) \in e'\text{-}aff\ x2 \neq 0\ y2 \neq 0$
assumes $delta'\ x1\ y1\ x2\ y2 \neq 0\ delta'\ x1\ y1\ (fst\ (\tau\ (x2,y2)))\ (snd\ (\tau\ (x2,y2)))$
 $\neq 0$
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 t-nz)
by algebra

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

lemma delta-add-delta'-1:

assumes $1: x1 \neq 0 y1 \neq 0 x2 \neq 0 y2 \neq 0$
assumes $r\text{-expr}: rx = \text{fst } (\text{add } (x1,y1) (x2,y2))$ $ry = \text{snd } (\text{add } (x1,y1) (x2,y2))$

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

lemma delta'\text{-add-delta-1:

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

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

lemma funny-field-lemma-1:

$$\begin{aligned} & ((x1 * x2 - y1 * y2) * ((x1 * x2 - y1 * y2) * (x2 * (y2 * (1 + d * x1 * y1 * x2 * y2)))) + \\ & (x1 * x2 - y1 * y2) * ((x1 * y2 + y1 * x2) * y2^2) * (1 - d * x1 * y1 * x2 * y2)) * \\ & (1 + d * x1 * y1 * x2 * y2) \neq \\ & ((x1 * y2 + y1 * x2) * ((x1 * y2 + y1 * x2) * (x2 * (y2 * (1 - d * x1 * y1 * x2 * y2)))) + \\ & (x1 * x2 - y1 * y2) * ((x1 * y2 + y1 * x2) * x2^2) * (1 + d * x1 * y1 * x2 * y2)) * \\ & (1 - d * x1 * y1 * x2 * y2) \implies \\ & (d * ((x1 * x2 - y1 * y2) * ((x1 * y2 + y1 * x2) * (x2 * y2))))^2 = \\ & ((1 - d * x1 * y1 * x2 * y2) * (1 + d * x1 * y1 * x2 * y2))^2 \implies \\ & x1^2 + y1^2 - 1 = d * x1^2 * y1^2 \implies \\ & x2^2 + y2^2 - 1 = d * x2^2 * y2^2 \implies False \end{aligned}$$

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*: $(x1,y1) \in e'\text{-aff}$ $(x2,y2) \in e'\text{-aff}$

assumes *pd*: $\delta x1 y1 x2 y2 \neq 0$

assumes *pd'*: $\delta' rx ry (fst (\tau (i (x2,y2)))) (snd (\tau (i (x2,y2)))) \neq 0$

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: algebra-simps divide-simps t-nz 1 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 power-mult-distrib[symmetric]*)

apply(*rule funny-field-lemma-1*)

by simp

lemma funny-field-lemma-2: $(x2 * y2)^2 * ((x2 * y1 - x1 * y2) * (x1 * x2 + y1 * y2))^2 \neq ((x1 * y1 - x2 * y2) * (x1 * y1 + x2 * y2))^2 \implies$
 $((x1 * y1 - x2 * y2) * ((x1 * y1 - x2 * y2) * (x2 * (y2 * (x1 * x2 + y1 * y2)))) +$
 $(x1 * y1 - x2 * y2) * ((x1 * y1 + x2 * y2) * x2^2) * (x2 * y1 - x1 * y2)) *$

$(x1 * x2 + y1 * y2) =$
 $((x1 * y1 + x2 * y2) * ((x1 * y1 + x2 * y2) * (x2 * (y2 * (x2 * y1 - x1 * y2)))) +$
 $(x1 * y1 - x2 * y2) * ((x1 * y1 + x2 * y2) * y2^2) * (x1 * x2 + y1 * y2)) *$
 $(x2 * y1 - x1 * y2) \implies$
 $x1^2 + y1^2 - 1 = d * x1^2 * y1^2 \implies$
 $x2^2 + y2^2 - 1 = d * x2^2 * y2^2 \implies \text{False}$
by algebra

lemma delta'-add-delta-2:

assumes 1: $x1 \neq 0$ $y1 \neq 0$ $x2 \neq 0$ $y2 \neq 0$
assumes *r-expr*: $rx = \text{fst}(\text{ext-add}(x1, y1)(x2, y2))$ $ry = \text{snd}(\text{ext-add}(x1, y1)(x2, y2))$
assumes *in-aff*: $(x1, y1) \in e'\text{-aff}$ $(x2, y2) \in e'\text{-aff}$
assumes *pd*: $\text{delta}' x1 y1 x2 y2 \neq 0$
assumes *pd'*: $\text{delta } rx ry (\text{fst}(\tau(i(x2, y2)))) (\text{snd}(\tau(i(x2, y2)))) \neq 0$
shows $\text{delta}' rx ry (\text{fst}(i(x2, y2))) (\text{snd}(i(x2, y2))) \neq 0$
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: algebra-simps divide-simps t-nz 1 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(*rule funny-field-lemma-2*)
by (*simp add: power-mult-distrib*)

lemma delta'-add-delta-not-add:

assumes 1: $x1 \neq 0$ $y1 \neq 0$ $x2 \neq 0$ $y2 \neq 0$
assumes *in-aff*: $(x1, y1) \in e'\text{-aff}$ $(x2, y2) \in e'\text{-aff}$
assumes *pd*: $\text{delta}' x1 y1 x2 y2 \neq 0$
assumes *add-nz*: $\text{fst}(\text{ext-add}(x1, y1)(x2, y2)) \neq 0$ $\text{snd}(\text{ext-add}(x1, y1)(x2, y2)) \neq 0$
shows *pd'*: $\text{delta}(\text{fst}(\tau(x1, y1))) (\text{snd}(\tau(x1, y1))) x2 y2 \neq 0$
unfolding *delta-def* *delta-minus-def* *delta-plus-def*
apply(*simp add: divide-simps t-nz 1*)
apply(*simp add: algebra-simps power2-eq-square[symmetric] t-expr d-nz*)
using *add-nz d-nz apply(simp)*
using *d-nz by algebra*

lemma not-add-self:

assumes *in-aff*: $(x, y) \in e'\text{-aff}$ $x \neq 0$ $y \neq 0$
shows $\text{delta } x y (\text{fst}(\tau(i(x, y)))) (\text{snd}(\tau(i(x, y)))) = 0$
 $\text{delta}' x y (\text{fst}(\tau(i(x, y)))) (\text{snd}(\tau(i(x, y)))) = 0$
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)

```

3 Projective Edwards curves

3.1 No fixed-point lemma and dichotomies

```

lemma g-no-fp:
  assumes  $g \in G$   $p \in e\text{-circ}$   $g p = p$ 
  shows  $g = id$ 
proof -
  obtain  $x y$  where  $p\text{-def}$ :  $p = (x, y)$  by fastforce
  have  $nz$ :  $x \neq 0$   $y \neq 0$  using  $assms$   $p\text{-def}$  unfolding  $e\text{-circ-def}$  by auto

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

```

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

lemma *dichotomy-1*:

assumes $p \in e'\text{-aff}$ $q \in e'\text{-aff}$

shows $(p \in e\text{-circ} \wedge (\exists g \in \text{symmetries}. q = (g \circ i) p)) \vee$
 $(p, q) \in e'\text{-aff-0} \vee (p, q) \in e'\text{-aff-1}$

proof –

obtain $x1\ y1$ where $p\text{-def}: p = (x1, y1)$ by *fastforce*

obtain $x2\ y2$ where $q\text{-def}: q = (x2, y2)$ by *fastforce*

consider (1) $(p, q) \in e'\text{-aff-0}$ |

(2) $(p, q) \in e'\text{-aff-1}$ |

(3) $(p, q) \notin e'\text{-aff-0} \wedge (p, q) \notin e'\text{-aff-1}$ 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 $\text{delta } x1\ y1\ x2\ y2 = 0$ $\text{delta}'\ x1\ y1\ x2\ y2 = 0$

unfolding $p\text{-def } q\text{-def } e'\text{-aff-0-def } e'\text{-aff-1-def}$ using *assms*
by (*simp add: assms p-def q-def*)⁺

have $x1 \neq 0\ y1 \neq 0\ x2 \neq 0\ y2 \neq 0$

using $\langle \text{delta } x1\ y1\ x2\ y2 = 0 \rangle$

unfolding $\text{delta-def } \text{delta-plus-def } \text{delta-minus-def}$ by *auto*

then have $p \in e\text{-circ } q \in e\text{-circ}$

unfolding $e\text{-circ-def}$ using *assms p-def q-def* by *blast*⁺

obtain $a0\ b0$ where $tq\text{-expr}: \tau\ q = (a0, b0)$ by *fastforce*

obtain $a1\ b1$ where $p\text{-expr}: p = (a1, b1)$ by *fastforce*

from $tq\text{-expr}$ have $q\text{-expr}: q = \tau\ (a0, b0)$ using *tau-idemp-explicit q-def* by *auto*

have $a0\text{-nz}: a0 \neq 0\ b0 \neq 0$

using $\langle \tau\ q = (a0, b0) \rangle \langle x2 \neq 0 \rangle \langle y2 \neq 0 \rangle$ *comp-apply q-def tau-sq* by *auto*

have $a1\text{-nz}: a1 \neq 0\ b1 \neq 0$

using $\langle p = (a1, b1) \rangle \langle x1 \neq 0 \rangle \langle y1 \neq 0 \rangle$ $p\text{-def}$ by *auto*

have $in\text{-aff}: (a0, b0) \in e'\text{-aff } (a1, b1) \in e'\text{-aff}$

using $\langle q \in e\text{-circ} \rangle \tau\text{-circ } \text{circ-to-aff } tq\text{-expr}$ **apply** *fastforce*

using *assms(1) p-expr* by *auto*

define $\delta' :: 'a \Rightarrow 'a \Rightarrow 'a$ where

$\delta' = (\lambda\ x0\ y0. x0 * y0 * \text{delta-minus } a1\ b1\ (1/(t*x0))\ (1/(t*y0)))$

define $p\delta' :: 'a \Rightarrow 'a \Rightarrow 'a$ 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 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 pδ'-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 δ-plus-expr: δ-plus a0 b0 = b1 * b0 - a1 * a0
unfolding δ-plus-def delta-x-def
by(simp add: divide-simps a0-nz a1-nz t-nz)
have δ-minus-expr: δ-minus a0 b0 = a1 * b0 + b1 * a0
unfolding δ-minus-def delta-y-def
by(simp add: divide-simps a0-nz a1-nz t-nz)

have cases1: δ' a0 b0 = 0 ∨ pδ' a0 b0 = 0
unfolding δ'-def pδ'-def
using ⟨delta x1 y1 x2 y2 = 0⟩ ⟨p = (a1, b1)⟩ delta-def p-def q-def q-expr by
auto
have cases2: δ-minus a0 b0 = 0 ∨ δ-plus a0 b0 = 0
using δ-minus-def δ-plus-def ⟨delta' x1 y1 x2 y2 = 0⟩ ⟨p = (a1, b1)⟩
delta'-def q-def p-def tq-expr by auto

consider
(1) δ' a0 b0 = 0 δ-minus a0 b0 = 0 |
(2) δ' a0 b0 = 0 δ-plus a0 b0 = 0 |
(3) pδ' a0 b0 = 0 δ-minus a0 b0 = 0 |
(4) pδ' a0 b0 = 0 δ-minus a0 b0 ≠ 0
using cases1 cases2 by auto
then have (a0,b0) = (b1,a1) ∨ (a0,b0) = (-b1,-a1) ∨
(a0,b0) = (a1,-b1) ∨ (a0,b0) = (-a1,b1)
proof(cases)
case 1
have zeros: a0 * b0 - a1 * b1 = 0 a1 * b0 + a0 * b1 = 0
using 1 δ-minus-expr δ'-expr
by(simp-all add: algebra-simps)
have ∃ q1 q2 q3 q4.
2*a0*b0*(b02 - a12) =
q1*(-1 + a02 + b02 - t2 * a02 * b02) +
q2*(-1 + a12 + b12 - t2 * a12 * b12) +
q3*(a0 * b0 - a1 * b1) +

```

```

      q4*(a1 * b0 + a0 * b1)
    by algebra
  then have b02 - a12 = 0 a02 - b12 = 0 a0 * b0 - a1 * b1 = 0
    using a0-nz in-aff zeros
    unfolding e'-aff-def e'-def
    apply simp-all
    apply(simp-all add: algebra-simps two-not-zero)
    by algebra
  then show ?thesis
    by algebra
next
case 2
have zeros: b1 * b0 - a1 * a0 = 0 a0 * b0 - a1 * b1 = 0
  using 2 δ-plus-expr δ'-expr by auto
have b02 - a12 = 0 a02 - b12 = 0 a0 * b0 - a1 * b1 = 0
  using in-aff zeros
  unfolding e'-aff-def e'-def
  apply simp-all
  by algebra+
then show ?thesis
  by algebra
next
case 3
have zeros: a1 * b0 + b1 * a0 = 0 a0 * b0 + a1 * b1 = 0
  using 3 δ-minus-expr pδ'-expr by auto
have a02 - a12 = 0 b02 - b12 = 0 a0 * b0 + a1 * b1 = 0
  using in-aff zeros
  unfolding e'-aff-def e'-def
  apply simp-all
  by algebra+
then show ?thesis
  by algebra
next
case 4
have zeros: a0 * b0 + a1 * b1 = 0 a1 * b0 + b1 * a0 ≠ 0
  using 4 pδ'-expr δ-minus-expr δ'-expr by auto
have a02 - b12 = 0 a12 - b02 = 0
  using in-aff zeros
  unfolding e'-aff-def e'-def
  by algebra+
then show ?thesis
  using cases2 δ-minus-expr δ-plus-expr by algebra
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 ∃ 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  $q = (\tau \circ g \circ i) p$ 
  using  $\text{tau-sq } \langle \tau \ q = (a0, b0) \rangle$   $q\text{-def}$  by auto
then have  $\exists g \in \text{symmetries}. q = (g \circ i) p$ 
  using  $\text{tau-rot-sym } \langle g \in \text{rotations} \rangle$   $\text{symmetries-def}$  by blast
then show ?thesis
  using  $\langle p \in e\text{-circ} \rangle$  by blast
qed
qed

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

  have 2:  $y1 = - y2$ 
    using  $\text{assms}(1, 2)$  unfolding  $e'\text{-aff-0-def } e'\text{-aff-def } \text{delta-def } \text{delta-plus-def}$ 
       $\text{delta-minus-def } e'\text{-def}$ 
    apply(simp)
    apply(simp add: c-eq-1 t-expr)
    by algebra

  from 1 2 show ?thesis by simp
qed

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

  have  $\text{eqs: } x1*(y1+y2) = x2*(y1+y2) \ x1 * y1 + x2 * y2 = 0$ 
    using  $\text{assms}$   $\text{in-aff}$   $\text{ds}$ 
    unfolding  $e'\text{-aff-def } e'\text{-def } \text{delta}'\text{-def } \text{delta-x-def } \text{delta-y-def}$ 
    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 $g \in \text{symmetries}$ $(x', y') = (g \circ i) (x, y)$ $(x, y) \in e\text{-circ}$
shows $\text{delta } x \ y \ x' \ y' = 0$
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 $g \in \text{symmetries}$ $(x', y') = (g \circ i) (x, y)$ $(x, y) \in e\text{-circ}$
shows $\text{delta}' \ x \ y \ x' \ y' = 0$
using *assms unfolding symmetries-def e-circ-def delta'-def delta-x-def delta-y-def*
by *auto*

lemma *meaning-of-dichotomy-1*:

assumes $(\exists g \in \text{symmetries}. (x2, y2) = (g \circ i) (x1, y1))$
shows $\text{fst} (\text{add } (x1, y1) (x2, y2)) = 0 \vee \text{snd} (\text{add } (x1, y1) (x2, y2)) = 0$
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: field-simps t-nz power2-eq-square[symmetric] t-expr*)

lemma *meaning-of-dichotomy-2*:

assumes $(\exists g \in \text{symmetries}. (x2, y2) = (g \circ i) (x1, y1))$
shows $\text{fst} (\text{ext-add } (x1, y1) (x2, y2)) = 0 \vee \text{snd} (\text{ext-add } (x1, y1) (x2, y2)) = 0$
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* :: $((('a \times 'a) \times \text{bool}) \times (('a \times 'a) \times \text{bool}))$ set where

$$\text{gluing} = \{((x0, y0), l), ((x1, y1), j) \mid$$

$$\begin{aligned} & ((x0, y0) \in e'\text{-aff} \wedge (x1, y1) \in e'\text{-aff}) \wedge \\ & ((x0 \neq 0 \wedge y0 \neq 0 \wedge (x1, y1) = \tau(x0, y0) \wedge j = \text{Not } l) \vee \\ & (x0 = x1 \wedge y0 = y1 \wedge l = j)) \} \end{aligned}$$

lemma *gluing-char*:
assumes $((x0, y0), l), ((x1, y1), j) \in \text{gluing}$
shows $((x0, y0) = (x1, y1) \wedge l = j) \vee ((x1, y1) = \tau(x0, y0) \wedge l = \text{Not } j \wedge x0 \neq 0 \wedge y0 \neq 0)$
using *assms gluing-def* **by** *force+*

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

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

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

$$e'\text{-aff-bit} = e'\text{-aff} \times \text{UNIV}$$

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

lemma *gluing-eq*: $x = y \implies \text{gluing } \{x\} = \text{gluing } \{y\}$ **by** *simp*

definition *e-proj* where $e\text{-proj} = e'\text{-aff-bit} // \text{gluing}$

3.2.1 Point-class classification

lemma *eq-class-simp*:

assumes $X \in e\text{-proj}$ $X \neq \{\}$
 shows $X // \text{gluing} = \{X\}$

proof –

have *simp-un*: $\text{gluing} \text{ “ } \{x\} = X \text{ if } x \in X \text{ for } x$
 apply (*rule quotientE*)
 using *e-proj-def* *assms(1)* apply *blast*
 using *equiv-class-eq[OF eq-rel]* that by *auto*

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

qed

lemma *gluing-class-1*:

assumes $x = 0 \vee y = 0$ $(x,y) \in e'\text{-aff}$
 shows $\text{gluing} \text{ “ } \{(x,y), l\} = \{(x,y), l\}$

proof –

have $(x,y) \notin e\text{-circ}$ 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 $x \neq 0$ $y \neq 0$ $(x,y) \in e'\text{-aff}$
 shows $\text{gluing} \text{ “ } \{(x,y), l\} = \{(x,y), l, (\tau(x,y), \text{Not } l)\}$

proof –

have $(x,y) \in e\text{-circ}$ using *assms* unfolding *e-circ-def* by *blast*
 then have $\tau(x,y) \in e'\text{-aff}$
 using $\tau\text{-circ}$ 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 $\langle \tau(x,y) \in e'\text{-aff} \rangle$ by *argo*

qed

lemma *e-proj-elim-1*:

assumes $(x,y) \in e'\text{-aff}$
 shows $\{(x,y), l\} \in e\text{-proj} \longleftrightarrow x = 0 \vee y = 0$

proof

assume *as*: $\{(x, y), l\} \in e\text{-proj}$
 have *eq*: $\text{gluing} \text{ “ } \{(x, y), l\} = \{(x,y), l\}$
 (is - = ?B)
 using *quotientI[of - ?B gluing]* *eq-class-simp* *as* by *auto*
 then show $x = 0 \vee y = 0$
 using *assms* *gluing-class-2* by *force*

next

assume $x = 0 \vee y = 0$

then have *eq: gluing* “ $\{(x, y), l\} = \{(x, y), l\}$
using *assms gluing-class-1* **by** *presburger*
show $\{(x, y), l\} \in e\text{-proj}$
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 $(x, y) \in e'\text{-aff}$
shows $\{(x, y), l, (\tau(x, y), \text{Not } l)\} \in e\text{-proj} \longleftrightarrow x \neq 0 \wedge y \neq 0$
proof
assume $x \neq 0 \wedge y \neq 0$
then have *eq: gluing* “ $\{(x, y), l\} = \{(x, y), l, (\tau(x, y), \text{Not } l)\}$
using *assms gluing-class-2* **by** *presburger*
show $\{(x, y), l, (\tau(x, y), \text{Not } l)\} \in e\text{-proj}$
apply(*subst eq[symmetric]*)
unfolding *e-proj-def* **apply**(*rule quotientI*)
unfolding *e'-aff-bit-def* **using** *assms* **by** *simp*
next
assume *as*: $\{(x, y), l, (\tau(x, y), \text{Not } l)\} \in e\text{-proj}$
have *eq: gluing* “ $\{(x, y), l\} = \{(x, y), l, (\tau(x, y), \text{Not } l)\}$
(is - = ?B)
using *quotientI[of - ?B gluing]* *eq-class-simp as* **by** *auto*
then show $x \neq 0 \wedge y \neq 0$
using *assms gluing-class-1* **by** *auto*
qed

lemma *e-proj-eq*:
assumes $p \in e\text{-proj}$
shows $\exists x y l. (p = \{(x, y), l\} \vee p = \{(x, y), l, (\tau(x, y), \text{Not } l)\}) \wedge (x, y) \in e'\text{-aff}$
proof –
obtain *g* **where** *p-expr*: $p = \text{gluing } \{g\} g \in e'\text{-aff-bit}$
using *assms* **unfolding** *e-proj-def* *quotient-def* **by** *blast+*
then obtain *x y l* **where** *g-expr*: $g = ((x, y), l) (x, y) \in e'\text{-aff}$
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 “ $\{(x, y), l\} \in e\text{-proj} \longleftrightarrow (x, y) \in e'\text{-aff}$
proof
assume *gluing* “ $\{(x, y), l\} \in e\text{-proj}$
then show $(x, y) \in e'\text{-aff}$
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) ∈ e'-aff
  show gluing “ {((x, y), l)} ∈ 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 ∈ e-proj
  obtains x0 y0 l where x = {((x0,y0),l)} ∨ x = {((x0,y0),l),(τ (x0,y0),Not l)}
  using e-proj-eq[OF assms] that by blast

lemma gluing-cases-explicit:
  assumes x ∈ e-proj x = gluing “ {((x0,y0),l)}
  shows x = {((x0,y0),l)} ∨ x = {((x0,y0),l),(τ (x0,y0),Not l)}
  proof -
    have (x0,y0) ∈ e'-aff
      using assms e-proj-aff by simp
    have gluing “ {((x0,y0),l)} = {((x0,y0),l)} ∨
      gluing “ {((x0,y0),l)} = {((x0,y0),l),(τ (x0,y0),Not l)}
      using assms gluing-class-1 gluing-class-2 ⟨(x0, y0) ∈ e'-aff⟩ by meson
    then show ?thesis using assms by fast
  qed

lemma gluing-cases-points:
  assumes x ∈ e-proj x = gluing “ {(p,l)}
  shows x = {(p,l)} ∨ x = {(p,l),(τ p,Not l)}
  using gluing-cases-explicit[OF assms(1), of fst p snd p l] assms by auto

lemma identity-equiv:
  gluing “ {((1, 0), l)} = {((1,0),l)}
  unfolding Image-def
  proof(simp,standard)
    show {y. (((1, 0), l), y) ∈ gluing} ⊆ {((1, 0), l)}
      using gluing-char-zero by(intro subrelI,fast)
    have (1,0) ∈ e'-aff
      unfolding e'-aff-def e'-def by simp
    then have ((1, 0), l) ∈ e'-aff-bit
      unfolding e'-aff-bit-def by blast
    show {((1, 0), l)} ⊆ {y. (((1, 0), l), y) ∈ gluing}
      using eq-rel ⟨((1, 0), l) ∈ e'-aff-bit⟩
      unfolding equiv-def refl-on-def by blast
  qed

lemma identity-proj:
  {((1,0),l)} ∈ e-proj

```

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

lemma *gluing-inv*:

assumes $x \neq 0\ y \neq 0\ (x,y) \in e'\text{-aff}$
shows *gluing* “ $\{(x,y),j\} = \text{gluing} \text{ “ } \{(\tau(x,y), \text{Not } j)\}$

proof –

have *taus*: $\tau(x,y) \in e'\text{-aff}$
using $e\text{-circ-def}$ *assms* $\tau\text{-circ}$ **by** *fastforce+*

have *gluing* “ $\{(x,y), j\} = \{(x, y), j, (\tau(x, y), \text{Not } j)\}$
using $gluing\text{-class-2}$ *assms* **by** *meson*

also have $\dots = \{(\tau(x, y), \text{Not } j), (\tau(\tau(x, y)), j)\}$
using $\tau\text{-idemp-explicit}$ **by** *force*

also have $\{(\tau(x, y), \text{Not } j), (\tau(\tau(x, y)), j)\} = \text{gluing} \text{ “ } \{(\tau(x,y), \text{Not } j)\}$
apply (*subst* $gluing\text{-class-2}[of\ fst\ (\tau(x,y))\ snd\ (\tau(x,y)),$
simplified prod.collapse])

using *assms* *taus* *t-nz* **by** *auto*

finally show *?thesis* **by** *blast*

qed

3.3 Projective addition on points

definition $xor :: bool \Rightarrow bool \Rightarrow bool$

where $xor\text{-def}$: $xor\ P\ Q \equiv (P \wedge \neg Q) \vee (\neg P \wedge Q)$

function (*domintros*) $proj\text{-add} :: ('a \times 'a) \times bool \Rightarrow ('a \times 'a) \times bool \Rightarrow ('a \times 'a) \times bool$

where

$proj\text{-add}\ ((x1, y1), l)\ ((x2, y2), j) = (add\ (x1, y1)\ (x2, y2), xor\ l\ j)$

if $\delta x1\ y1\ x2\ y2 \neq 0$ **and**

$(x1, y1) \in e'\text{-aff}$ **and**

$(x2, y2) \in e'\text{-aff}$

| $proj\text{-add}\ ((x1, y1), l)\ ((x2, y2), j) = (ext\text{-add}\ (x1, y1)\ (x2, y2), xor\ l\ j)$

if $\delta' x1\ y1\ x2\ y2 \neq 0$ **and**

$(x1, y1) \in e'\text{-aff}$ **and**

$(x2, y2) \in e'\text{-aff}$

| $proj\text{-add}\ ((x1, y1), l)\ ((x2, y2), j) = \text{undefined}$

if $(x1, y1) \notin e'\text{-aff} \vee (x2, y2) \notin e'\text{-aff} \vee$

$(\delta x1\ y1\ x2\ y2 = 0 \wedge \delta' x1\ y1\ x2\ y2 = 0)$

apply (*fast*)

apply (*fastforce*)

using *coherence* $e'\text{-aff-def}$ **apply** *force*

by *auto*

termination *proj-add* using *termination by blast*

lemma *proj-add-inv*:

assumes $(x0, y0) \in e'\text{-aff}$

shows *proj-add* $((x0, y0), l) (i (x0, y0), l') = ((1, 0), \text{xor } l \ l')$

proof –

have *i-in*: $i (x0, y0) \in e'\text{-aff}$

using *i-aff assms* **by** *blast*

consider (1) $x0 = 0$ | (2) $y0 = 0$ | (3) $x0 \neq 0 \ y0 \neq 0$ **by** *fast*

then show *?thesis*

proof(*cases*)

case 1

from *assms 1* **have** *y-expr*: $y0 = 1 \vee y0 = -1$

unfolding *e'-aff-def e'-def* **by**(*simp, algebra*)

then have *delta* $x0 \ y0 \ x0 \ (-y0) \neq 0$

using 1 **unfolding** *delta-def delta-minus-def delta-plus-def* **by** *simp*

then show *proj-add* $((x0, y0), l) (i (x0, y0), l') = ((1, 0), \text{xor } l \ l')$

using 1 *assms delta-plus-def i-in inverse-generalized* **by** *fastforce*

next

case 2

from *assms 2* **have** $x0 = 1 \vee x0 = -1$

unfolding *e'-aff-def e'-def* **by**(*simp, algebra*)

then have *delta* $x0 \ y0 \ x0 \ (-y0) \neq 0$

using 2 **unfolding** *delta-def delta-minus-def delta-plus-def* **by** *simp*

then show *?thesis*

using 2 *assms delta-def inverse-generalized* **by** *fastforce*

next

case 3

consider (a) *delta* $x0 \ y0 \ x0 \ (-y0) = 0$ *delta'* $x0 \ y0 \ x0 \ (-y0) = 0$ |

(b) *delta* $x0 \ y0 \ x0 \ (-y0) \neq 0$ *delta'* $x0 \ y0 \ x0 \ (-y0) = 0$ |

(c) *delta* $x0 \ y0 \ x0 \ (-y0) = 0$ *delta'* $x0 \ y0 \ x0 \ (-y0) \neq 0$ |

(d) *delta* $x0 \ y0 \ x0 \ (-y0) \neq 0$ *delta'* $x0 \ y0 \ x0 \ (-y0) \neq 0$ **by** *meson*

then show *?thesis*

proof(*cases*)

case a

then have $d * x0^2 * y0^2 = 1 \vee d * x0^2 * y0^2 = -1$

$x0^2 = y0^2$

$x0^2 + y0^2 - 1 = d * x0^2 * y0^2$

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 $2 * x0^2 = 2 \vee 2 * x0^2 = 0$

by *algebra*

then have $x0 = 1 \vee x0 = -1$

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  $((x0, y0), l) (i (x0, y0), l') =$ 
   $(add (x0, y0) (i (x0, y0)), xor l l')$ 
  using assms i-in b by simp
also have  $\dots = ((1,0),xor l l')$ 
  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  $((x0, y0), l) (i (x0, y0), l') =$ 
   $(ext-add (x0, y0) (i (x0, y0)), xor l l')$ 
  using assms i-in c by simp
also have  $\dots = ((1,0),xor l l')$ 
  apply(subst ext-add-inverse)
  using 3 by auto
finally show ?thesis
  by blast
next
case d
have proj-add  $((x0, y0), l) (i (x0, y0), l') =$ 
   $(add (x0, y0) (i (x0, y0)), xor l l')$ 
  using assms i-in d by simp
also have  $\dots = ((1,0),xor l l')$ 
  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)  $\text{delta } x0 \ y0 \ x1 \ y1 \neq 0 \wedge (x0, y0) \in e'\text{-aff} \wedge (x1, y1) \in e'\text{-aff} \mid$ 
  (2)  $\text{delta}' \ x0 \ y0 \ x1 \ y1 \neq 0 \wedge (x0, y0) \in e'\text{-aff} \wedge (x1, y1) \in e'\text{-aff} \mid$ 
  (3)  $(\text{delta } x0 \ y0 \ x1 \ y1 = 0 \wedge \text{delta}' \ x0 \ y0 \ x1 \ y1 = 0) \vee$ 
       $(x0, y0) \notin e'\text{-aff} \vee (x1, y1) \notin e'\text{-aff}$  by blast
then show ?thesis
proof(cases)
  case 1 then show ?thesis
    apply(simp add: commutativity delta-com)
    using xor-def by force
  next
    case 2 then show ?thesis
      apply(simp add: ext-add-comm delta'-com del: ext-add.simps)
      using xor-def by force
    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 ::  $((a \times a) \times \text{bool}) \text{ set} \Rightarrow$ 
       $((a \times a) \times \text{bool}) \text{ set} \Rightarrow$ 
       $((a \times a) \times \text{bool}) \text{ set} \text{ set}$ 

```

```

where
  proj-add-class c1 c2 =
    (
      {
        proj-add  $((x1, y1), i) ((x2, y2), j) \mid$ 
           $x1 \ y1 \ i \ x2 \ y2 \ j.$ 
           $((x1, y1), i) \in c1 \wedge$ 
           $((x2, y2), j) \in c2 \wedge$ 
           $((x1, y1), (x2, y2)) \in e'\text{-aff-0} \cup e'\text{-aff-1}$ 
        } // gluing
    )
  if  $c1 \in e\text{-proj}$  and  $c2 \in e\text{-proj}$ 
  | proj-add-class c1 c2 = undefined
  if  $c1 \notin e\text{-proj} \vee c2 \notin e\text{-proj}$ 
  by (meson surj-pair) auto

```

termination *proj-add-class* **using** *termination* **by** *auto*

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

3.4.1 Covering

corollary *no-fp-eq*:
assumes $p \in e\text{-circ}$

assumes $r' \in \text{rotations}$ $r \in \text{rotations}$
assumes $(r' \circ i) p = (\tau \circ r) (i p)$
shows *False*
proof –
obtain r'' **where** $r'' \circ r' = \text{id}$ $r'' \in \text{rotations}$
using *rot-inv assms* **by** *blast*
then have $i p = (r'' \circ \tau \circ r) (i p)$
using *assms* **by** *(simp,metis pointfree-idE)*
then have $i p = (\tau \circ r'' \circ r) (i p)$
using *rot-tau-com*[*OF* $\langle r'' \in \text{rotations} \rangle$] **by** *simp*
then have $\exists r''. r'' \in \text{rotations} \wedge i p = (\tau \circ r'') (i p)$
using *rot-comp*[*OF* $\langle r'' \in \text{rotations} \rangle$] *assms* **by** *fastforce*
then obtain r'' **where**
 $eq: r'' \in \text{rotations}$ $i p = (\tau \circ r'') (i p)$
by *blast*
have $\tau \circ r'' \in G$ $i p \in e\text{-circ}$
using *tau-rot-sym*[*OF* $\langle r'' \in \text{rotations} \rangle$] *G-partition* **apply** *simp*
using *i-circ-points assms(1)* **by** *simp*
then show *False*
using *g-no-fp*[*OF* $\langle \tau \circ r'' \in G \rangle \langle i p \in e\text{-circ} \rangle$]
 eq *assms(1)* *sym-not-id*[*OF* $eq(1)$] **by** *argo*
qed

lemma *covering*:
assumes $p \in e\text{-proj}$ $q \in e\text{-proj}$
shows *proj-add-class* p $q \neq \{\}$
proof –
from *e-proj-eq*[*OF* *assms(1)*] *e-proj-eq*[*OF* *assms(2)*]
obtain x y l x' y' l' **where**
 $p\text{-}q\text{-}expr: p = \{((x, y), l)\} \vee p = \{((x, y), l), (\tau (x, y), \text{Not } l)\}$
 $q = \{((x', y'), l')\} \vee q = \{((x', y'), l'), (\tau (x', y'), \text{Not } l')\}$
 $(x, y) \in e'\text{-}aff$ $(x', y') \in e'\text{-}aff$
by *blast*
then have *in-aff*: $(x, y) \in e'\text{-}aff$ $(x', y') \in e'\text{-}aff$ **by** *auto*
from *p-q-expr* **have** *gluings*: $p = (\text{gluing } \{((x, y), l)\})$
 $q = (\text{gluing } \{((x', y'), l')\})$
using *assms* *e-proj-elim-1* *e-proj-elim-2* *gluing-class-1* *gluing-class-2*
by *metis+*
then have *gluing-proj*: $(\text{gluing } \{((x, y), l)\}) \in e\text{-proj}$
 $(\text{gluing } \{((x', y'), l')\}) \in e\text{-proj}$
using *assms* **by** *blast+*

consider
 $(x, y) \in e\text{-circ} \wedge (\exists g \in \text{symmetries}. (x', y') = (g \circ i) (x, y))$
 $| ((x, y), x', y') \in e'\text{-}aff\text{-}0$
 $| ((x, y), x', y') \in e'\text{-}aff\text{-}1$
using *dichotomy-1*[*OF* $\langle (x, y) \in e'\text{-}aff \rangle \langle (x', y') \in e'\text{-}aff \rangle$] **by** *blast*
then show *?thesis*
proof(*cases*)

case 1
then obtain r **where** $r\text{-expr}: (x',y') = (\tau \circ r) (i (x,y))$ $r \in \text{rotations}$
using *sym-decomp* **by force**

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

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

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

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

have $p\text{-q-proj}$: $\{((x,y),l),(\tau (x,y),\text{Not } l)\} \in e\text{-proj}$
 $\{(\tau (x',y'),\text{Not } l'),((x',y'),l')\} \in e\text{-proj}$
using $p\text{-q-expr}'$ *assms* **by auto**

consider

$(a) \ (x, y) \in e\text{-circ} \wedge (\exists g \in \text{symmetries}. \tau (x', y') = (g \circ i) (x, y))$
 $(b) \ ((x, y), \tau (x', y')) \in e'\text{-aff-0}$
 $(c) \ ((x, y), \tau (x', y')) \in e'\text{-aff-1}$

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** $r'\text{-expr}: \tau (x',y') = (\tau \circ r') (i (x, y))$ $r' \in \text{rotations}$
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 $\dots = \tau ((\tau \circ r') (i (x, y)))$
using $r'\text{-expr}$ **by argo**

also have $\dots = r' (i (x, y))$
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 $\text{ds}: \text{delta } x \ y \ (\text{fst } (\tau (x',y'))) \ (\text{snd } (\tau (x',y'))) \neq 0$
unfolding *e'-aff-0-def* **by simp**

then have

```

    add-some: proj-add ((x,y),l) (τ (x',y'),Not l') = (add (x, y) (τ (x',y')), Not
(xor l l'))
    using proj-add.simps[of x y - - l Not l', OF - ]
      ⟨(x,y) ∈ e'-aff⟩ ⟨τ (x', y') ∈ e'-aff⟩ xor-def by auto
  show ?thesis
    apply(simp add: ex-in-conv[symmetric])
    apply(rule exI[of - gluing “{(add (x, y) (τ (x',y')), Not (xor l l'))}”])
    apply(subst proj-add-class.simps(1)[of p q, OF assms])
    apply(rule quotientI)
    apply(subst p-q-expr')+
    apply(subst add-some[symmetric])
    using b by fastforce
  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'),Not l') = (ext-add (x, y) (τ (x',y')),
Not (xor l l'))
    using proj-add.simps[of x y - - l Not l', OF - ]
      ⟨(x,y) ∈ e'-aff⟩ ⟨τ (x', y') ∈ e'-aff⟩ xor-def by force
  show ?thesis
    apply(simp add: ex-in-conv[symmetric])
    apply(rule exI[of - gluing “{(ext-add (x, y) (τ (x',y')), Not (xor l l'))}”])
    apply(subst proj-add-class.simps(1)[of p q, OF assms])
    apply(rule quotientI)
    apply(subst p-q-expr')+
    apply(subst add-some[symmetric])
    using c by fastforce
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'), xor 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 xor-def by blast
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'), xor l l')
  using proj-add.simps(2)[of x y x' y' l l', OF - ] in-aff xor-def by simp
then show ?thesis

```

```

using p-q-expr
unfolding proj-add-class.simps(1)[OF assms]
unfolding e'-aff-1-def using ds in-aff xor-def by blast
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), Not l)}
    q = {(x', y'), l'} ∨ q = {(x', y'), l'}, (τ (x', y'), Not l')}
    (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) ∈ e'-aff (x',y') ∈ e'-aff by auto

  then have gluings: p = (gluing “ {((x,y),l)})
    q = (gluing “ {(x',y'),l'})
    using p-def q-def by simp+

  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 $nz: x \neq 0 \ y \neq 0 \ x' \neq 0 \ y' \neq 0$
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 $p\text{-}q\text{-}expr'$: $p = \{((x,y),l),(\tau (x,y), \text{Not } l)\}$
 $q = \{(\tau (x',y'),\text{Not } l'),((x',y'),l')\}$
using *gluings nz gluing-class-2 taus in-aff tau-idemp-point t-nz assms* **by** *auto*

have $p\text{-}q\text{-}proj$: $\{((x,y),l),(\tau (x,y),\text{Not } l)\} \in e\text{-proj}$
 $\{(\tau (x',y'),\text{Not } l'),((x',y'),l')\} \in e\text{-proj}$
using *p-q-expr p-q-expr' assms gluing-proj gluings* **by** *auto*

consider

- (a) $(x, y) \in e\text{-circ} \wedge (\exists g \in \text{symmetries}. \tau (x', y') = (g \circ i) (x, y))$
- | (b) $((x, y), \tau (x', y')) \in e'\text{-aff-0}$
- | (c) $((x, y), \tau (x', y')) \in e'\text{-aff-1}$

using *dichotomy-1[OF $\langle(x,y) \in e'\text{-aff}\rangle \langle\tau (x', y') \in e'\text{-aff}\rangle$]* **by** *blast*

then show *?thesis*

proof(*cases*)

case *a*

then obtain r' **where** $r'\text{-expr}: \tau (x',y') = (\tau \circ r') (i (x, y))$ $r' \in \text{rotations}$
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 $\dots = \tau ((\tau \circ r') (i (x, y)))$
using $r'\text{-expr}$ **by** *argo*

also have $\dots = r' (i (x, y))$
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** $x'' = \text{fst} (\tau (x',y'))$
define y'' **where** $y'' = \text{snd} (\tau (x',y'))$
from *b* **have** $\text{delta } x \ y \ x'' \ y'' \neq 0$
unfolding *e'-aff-0-def* **using** $x''\text{-def } y''\text{-def}$ **by** *simp*

then show *?thesis*

unfolding $x''\text{-def } y''\text{-def}$ **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  $e'\text{-aff-1-def}$  using  $x''\text{-def}$   $y''\text{-def}$  by  $simp$ 
then show  $?thesis$ 
  unfolding  $x''\text{-def}$   $y''\text{-def}$  by  $blast$ 
qed
next
case  $2$ 
then have  $\delta x y x' y' \neq 0$ 
  unfolding  $e'\text{-aff-0-def}$  by  $simp$ 
then show  $?thesis$  by  $simp$ 
next
case  $3$ 
then have  $\delta' x y x' y' \neq 0$ 
  unfolding  $e'\text{-aff-1-def}$  by  $simp$ 
then show  $?thesis$  by  $simp$ 
qed
qed

```

3.4.2 Independence of the representant

lemma $proj\text{-add-class-comm}$:

assumes $c1 \in e\text{-proj}$ $c2 \in e\text{-proj}$

shows $proj\text{-add-class } c1 \ c2 = proj\text{-add-class } c2 \ c1$

proof –

have $((x1, y1), (x2, y2)) \in e'\text{-aff-0} \cup e'\text{-aff-1} \implies$

$((x2, y2), (x1, y1)) \in e'\text{-aff-0} \cup e'\text{-aff-1}$ **for** $x1 \ y1 \ x2 \ y2$

unfolding $e'\text{-aff-0-def}$ $e'\text{-aff-1-def}$

$e'\text{-aff-def}$ $e'\text{-def}$

$\delta\text{-def}$ $\delta\text{-plus-def}$ $\delta\text{-minus-def}$

$\delta'\text{-def}$ $\delta\text{-x-def}$ $\delta\text{-y-def}$

by($simp, algebra$)

then have $\{proj\text{-add } ((x1, y1), i) \ ((x2, y2), j) \mid x1 \ y1 \ i \ x2 \ y2 \ j.$

$((x1, y1), i) \in c1 \wedge ((x2, y2), j) \in c2 \wedge ((x1, y1), x2, y2) \in e'\text{-aff-0} \cup e'\text{-aff-1}\} =$

$\{proj\text{-add } ((x1, y1), i) \ ((x2, y2), j) \mid x1 \ y1 \ i \ x2 \ y2 \ j.$

$((x1, y1), i) \in c2 \wedge ((x2, y2), j) \in c1 \wedge ((x1, y1), x2, y2) \in e'\text{-aff-0} \cup e'\text{-aff-1}\}$

using $proj\text{-add-comm}$ **by** $blast$

then show $?thesis$

unfolding $proj\text{-add-class.simps}(1)[OF \ assms]$

$proj\text{-add-class.simps}(1)[OF \ assms(2) \ assms(1)]$ **by** $argo$

qed

lemma $gluing\text{-add-1}$:

assumes *gluing* “ $\{(x,y),l\} = \{(x, y), l\}$ *gluing* “ $\{(x',y'),l'\} = \{(x', y'), l'\}$
 $\neq 0$ *gluing* “ $\{(x,y),l\} \in e\text{-proj}$ *gluing* “ $\{(x',y'),l'\} \in e\text{-proj}$ $\delta x y x' y' \neq 0$
shows *proj-addition* (*gluing* “ $\{(x,y),l\}$) (*gluing* “ $\{(x',y'),l'\}$) = (*gluing* “ $\{(add(x,y)(x',y'), xor\ l\ l')\}$)
proof –
have *in-aff*: $(x,y) \in e'\text{-aff}$ $(x',y') \in e'\text{-aff}$
using *assms e-proj-eq e-proj-aff* **by** *blast+*
then have *add-in*: $add(x, y)(x', y') \in e'\text{-aff}$
using *add-closure* $\langle \delta x y x' y' \neq 0 \rangle$ *delta-def e-e'-iff e'-aff-def* **by** *auto*
from *in-aff* **have** *zeros*: $x = 0 \vee y = 0$ $x' = 0 \vee y' = 0$
using *e-proj-elim-1 assms* **by** *presburger+*
then have *add-zeros*: $fst(add(x,y)(x',y')) = 0 \vee snd(add(x,y)(x',y')) = 0$
by *auto*
then have *add-proj*: *gluing* “ $\{(add(x, y)(x', y'), xor\ l\ l')\} = \{(add(x, y)(x', y'), xor\ l\ l')\}$
using *add-in gluing-class-1* **by** *auto*
have *e-proj*: *gluing* “ $\{(x,y),l\} \in e\text{-proj}$
gluing “ $\{(x',y'),l'\} \in e\text{-proj}$
gluing “ $\{(add(x, y)(x', y'), xor\ l\ l')\} \in e\text{-proj}$
using *e-proj-aff in-aff add-in* **by** *auto*

consider
(a) $(x, y) \in e\text{-circ} \wedge (\exists g \in \text{symmetries}. (x', y') = (g \circ i)(x, y))$ |
(b) $((x, y), x', y') \in e'\text{-aff-0} \neg ((x, y) \in e\text{-circ} \wedge (\exists g \in \text{symmetries}. (x', y') = (g \circ i)(x, y)))$ |
(c) $((x, y), x', y') \in e'\text{-aff-1} \neg ((x, y) \in e\text{-circ} \wedge (\exists g \in \text{symmetries}. (x', y') = (g \circ i)(x, y)))$ $((x, y), x', y') \notin e'\text{-aff-0}$
using *dichotomy-1* [*OF* $\langle (x,y) \in e'\text{-aff} \rangle \langle (x',y') \in e'\text{-aff} \rangle$] **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* $((x, y), l)((x', y'), l') = (add(x,y)(x', y'), xor\ l\ l')$
using *proj-add.simps* $\langle \delta x y x' y' \neq 0 \rangle$ *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 $\langle \delta x y x' y' \neq 0 \rangle$ *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 $eqs: \text{delta } x \ y \ x' \ y' = 0 \ \text{delta}' \ x \ y \ x' \ y' \neq 0 \ e \ x \ y = 0 \ e \ x' \ y' = 0$
unfolding $e'\text{-aff-0-def } e'\text{-aff-1-def}$ **apply** $fast+$
using $e\text{-}e'\text{-iff in-aff}$ **unfolding** $e'\text{-aff-def}$ **by** $fast+$
then show $?thesis$ **using** $assms$ **by** $simp$
qed
qed

lemma $gluing\text{-add-2}$:

assumes $gluing \text{ “ } \{(x,y),l\} = \{(x, y), l\} \text{ gluing “ } \{(x',y'),l'\} = \{(x', y'), l'\}, (\tau \ (x', y'), \text{Not } l')$
 $gluing \text{ “ } \{(x,y),l\} \in e\text{-proj} \text{ gluing “ } \{(x',y'),l'\} \in e\text{-proj} \ \text{delta } x \ y \ x' \ y' \neq 0$

shows $proj\text{-addition} \ (gluing \text{ “ } \{(x,y),l\}) \ (gluing \text{ “ } \{(x',y'),l'\}) = (gluing \text{ “ } \{(add \ (x,y) \ (x',y'), \text{xor } l \ l')\})$

proof –

have $in\text{-aff}: (x,y) \in e'\text{-aff} \ (x',y') \in e'\text{-aff}$
using $assms \ e\text{-proj-eq } e\text{-proj-aff}$ **by** $blast+$
then have $add\text{-in}: add \ (x, y) \ (x', y') \in e'\text{-aff}$
using $add\text{-closure} \ \langle \text{delta } x \ y \ x' \ y' \neq 0 \rangle \ \text{delta-def } e\text{-}e'\text{-iff } e'\text{-aff-def}$ **by** $auto$
from $in\text{-aff}$ **have** $zeros: x = 0 \ \vee \ y = 0 \ x' \neq 0 \ y' \neq 0$
using $e\text{-proj-elim-1 } e\text{-proj-elim-2} \ assms$ **by** $presburger+$
have $e\text{-proj}: gluing \text{ “ } \{(x,y),l\} \in e\text{-proj}$
 $gluing \text{ “ } \{(x',y'),l'\} \in e\text{-proj}$
 $gluing \text{ “ } \{(add \ (x, y) \ (x', y'), \text{xor } l \ l')\} \in e\text{-proj}$
using $e\text{-proj-aff in-aff add-in}$ **by** $auto$

consider

$(a) \ (x, y) \in e\text{-circ} \wedge (\exists g \in \text{symmetries}. (x', y') = (g \circ i) \ (x, y)) \mid$
 $(b) \ ((x, y), x', y') \in e'\text{-aff-0} \ \neg \ ((x, y) \in e\text{-circ} \wedge (\exists g \in \text{symmetries}. (x', y') = (g \circ i) \ (x, y))) \mid$
 $(c) \ ((x, y), x', y') \in e'\text{-aff-1} \ \neg \ ((x, y) \in e\text{-circ} \wedge (\exists g \in \text{symmetries}. (x', y') = (g \circ i) \ (x, y))) \ ((x, y), x', y') \notin e'\text{-aff-0}$
using $dichotomy-1[OF \ \langle (x,y) \in e'\text{-aff} \rangle \ \langle (x',y') \in e'\text{-aff} \rangle]$ **by** $fast$

then show $?thesis$

proof($cases$)

case a

then have $False$

using $in\text{-aff zeros}$ **unfolding** $e\text{-circ-def}$ **by** $force$

then show $?thesis$ **by** $simp$

next

case b

then have $ld\text{-nz}: \text{delta } x \ y \ x' \ y' \neq 0$ **unfolding** $e'\text{-aff-0-def}$ **by** $auto$

have $v1: proj\text{-add} \ ((x, y), l) \ ((x', y'), l') = (add \ (x, y) \ (x', y'), \text{xor } l \ l')$

by($simp \ add: \ \langle (x,y) \in e'\text{-aff} \rangle \ \langle (x',y') \in e'\text{-aff} \rangle \ ld\text{-nz del: add.simps}$)

have $ecirc: (x',y') \in e\text{-circ} \ x' \neq 0 \ y' \neq 0$

unfolding $e\text{-circ-def}$ **using** $zeros \ \langle (x',y') \in e'\text{-aff} \rangle$ **by** $blast+$

then have $\tau (x', y') \in e\text{-circ}$
using *zeros* $\tau\text{-circ}$ **by** *blast*
then have $in\text{-aff}' : \tau (x', y') \in e'\text{-aff}$
unfolding $e\text{-circ}\text{-def}$ **by** *force*

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

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

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

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

have $v2 : proj\text{-add} ((x, y), l) (\tau (x', y'), Not\ l') = (\tau (add (x, y) (x', y')), Not\ (xor\ l\ l'))$
using $proj\text{-add}\text{-simps}$ $\langle \tau (x', y') \in e'\text{-aff} \rangle$ $in\text{-aff}$ $tau\text{-conv}$
 $\langle delta\ x\ y\ (fst\ (\tau\ (x', y')))\ (snd\ (\tau\ (x', y')))\ \neq\ 0 \rangle$ $xor\text{-def}$ **by** *auto*

have $gl\text{-class} : gluing\ \{ \{ (add (x, y) (x', y'), xor\ l\ l') \} =$
 $\{ (add (x, y) (x', y'), xor\ l\ l'), (\tau (add (x, y) (x', y')), Not\ (xor\ l\ l')) \}$
 $gluing\ \{ \{ (add (x, y) (x', y'), xor\ l\ l') \} \in e\text{-proj}$
using $gluing\text{-class}\text{-2}$ $add\text{-nz}$ $add\text{-in}$ **apply** *simp*
using $e\text{-proj}\text{-aff}$ $add\text{-in}$ **by** *auto*

show *?thesis*
proof -

have $\{ proj\text{-add} ((x1, y1), i) ((x2, y2), j) \mid x1\ y1\ i\ x2\ y2\ j.$
 $((x1, y1), i) \in \{ (x, y), l \} \wedge$
 $((x2, y2), j) \in \{ (x', y'), l' \}, (\tau (x', y'), Not\ l') \} \wedge$
 $((x1, y1), x2, y2)$
 $\in \{ ((x1, y1), x2, y2). (x1, y1) \in e'\text{-aff} \wedge (x2, y2) \in e'\text{-aff} \wedge delta\ x1\ y1\ x2$
 $y2\ \neq\ 0 \} \cup e'\text{-aff}\text{-1} \}$
 $\{ proj\text{-add} ((x, y), l) ((x', y'), l'), proj\text{-add} ((x, y), l) (\tau (x', y'), Not\ l') \}$
(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'), xor l l'), (τ (add (x, y) (x', y')), Not
(xor l l'))}
using v1 v2 by presburger
finally have eq: ?t = {(add (x, y) (x', y'), xor l l'), (τ (add (x, y) (x', y')),
Not (xor l l'))}
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}, (τ (x, y), Not l) }
gluing “ {(x', y'), l'} = {(x', y'), l'}, (τ (x', y'), Not 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'), xor 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)  $(x, y) \in e\text{-circ} \wedge (\exists g \in \text{symmetries}. (x', y') = (g \circ i)(x, y))$ 
  | (b)  $((x, y), x', y') \in e'\text{-aff-0}$ 
  | (c)  $((x, y), x', y') \in e'\text{-aff-1} \wedge ((x, y), x', y') \notin e'\text{-aff-0}$ 
  using dichotomy-1[OF in-aff] by auto
  then show ?thesis
  proof(cases)
    case a
    then obtain g where sym-expr:  $g \in \text{symmetries} \wedge (x', y') = (g \circ i)(x, y)$  by
  auto
    then have ds:  $\text{delta } x \ y \ x' \ y' = 0 \wedge \text{delta}' \ x \ y \ x' \ y' = 0$ 
    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:  $\text{delta } x \ y \ x' \ y' \neq 0$ 
    unfolding e'-aff-0-def by auto
    then have ds:  $\text{delta } (\text{fst } (\tau \ (x, y))) \ (\text{snd } (\tau \ (x, y))) \ (\text{fst } (\tau \ (x', y'))) \ (\text{snd } (\tau \ (x', y'))) \neq 0$ 
    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:  $\text{proj-add } ((x, y), l) \ ((x', y'), l') = (\text{add } (x, y) \ (x', y'), \text{xor } l \ l')$ 
    using ld-nz proj-add.simps  $\langle (x, y) \in e'\text{-aff} \rangle \langle (x', y') \in e'\text{-aff} \rangle$  by simp
    have v2:  $\text{proj-add } (\tau \ (x, y), \text{Not } l) \ (\tau \ (x', y'), \text{Not } l') = (\text{add } (x, y) \ (x', y'), \text{xor } l \ l')$ 
  xor l l'
    using ds proj-add.simps taus
    inversion-invariance-1 nz tau-idemp proj-add.simps xor-def
    by (auto simp add: c-eq-1 t-nz)

  consider (aaa)  $\text{delta } x \ y \ (\text{fst } (\tau \ (x', y'))) \ (\text{snd } (\tau \ (x', y'))) \neq 0 \mid$ 
    (bbb)  $\text{delta}' \ x \ y \ (\text{fst } (\tau \ (x', y'))) \ (\text{snd } (\tau \ (x', y'))) \neq 0$ 
     $\text{delta } x \ y \ (\text{fst } (\tau \ (x', y'))) \ (\text{snd } (\tau \ (x', y'))) = 0 \mid$ 
    (ccc)  $\text{delta}' \ x \ y \ (\text{fst } (\tau \ (x', y'))) \ (\text{snd } (\tau \ (x', y'))) = 0$ 
     $\text{delta } x \ y \ (\text{fst } (\tau \ (x', y'))) \ (\text{snd } (\tau \ (x', y'))) = 0$  by blast

  then show ?thesis
  proof(cases)
    case aaa
    have tau-conv:  $\tau \ (\text{add } (x, y) \ (\tau \ (x', y'))) = \text{add } (x, y) \ (x', y')$ 
    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*:
 $\text{proj-add } ((x, y), l) (\tau (x', y'), \text{Not } l') = (\tau (\text{add } (x, y) (x', y')), \text{Not } (\text{xor } l \ l'))$

apply(*subst proj-add.simps(1)[OF aaa $\langle(x, y) \in e'\text{-aff}\rangle$,simplified prod.collapse,OF $\langle(\tau (x', y')) \in e'\text{-aff}\rangle$]*)
apply(*subst tau-conv[symmetric]*)
apply(*subst tau-idemp-point*)
by(*auto simp add: xor-def*)

have *ds'*: $\text{delta } (\text{fst } (\tau (x, y))) (\text{snd } (\tau (x, y))) x' y' \neq 0$
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*: $\text{proj-add } (\tau (x, y), \text{Not } l) ((x', y'), l') = (\tau (\text{add } (x, y) (x', y')), \text{Not } (\text{xor } l \ l'))$

proof –
have $\text{proj-add } (\tau (x, y), \text{Not } l) ((x', y'), l') = (\text{add } (\tau (x, y)) (x', y'), \text{Not } (\text{xor } l \ l'))$
using *proj-add.simps $\langle(x, y) \in e'\text{-aff}\rangle \langle(x', y') \in e'\text{-aff}\rangle ds' xor-def$* **by** *auto*

moreover have $\text{add } (\tau (x, y)) (x', y') = \tau (\text{add } (x, y) (x', y'))$
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*: $\text{add } (x, y) (x', y') \in e'\text{-aff}$
using *in-aff add-closure ld-nz e-e'-iff* **unfolding** *delta-def e'-aff-def* **by** *auto*

have *add-nz*: $\text{fst } (\text{add } (x, y) (x', y')) \neq 0$
 $\text{snd } (\text{add } (x, y) (x', y')) \neq 0$
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*: $\text{gluing } \{(\text{add } (x, y) (x', y'), \text{xor } l \ l')\} = \{(\text{add } (x, y) (x', y'), \text{xor } l \ l'), (\tau (\text{add } (x, y) (x', y')), \text{Not } (\text{xor } l \ l'))\}$
using *add-nz add-closure gluing-class-2* **by** *auto*

```

have class-proj: gluing “  $\{(add(x, y)(x', y'), xor\ l\ l')\} \in e\text{-proj}$ 
using add-closure e-proj-aff by auto

have dom-eq:  $\{proj\text{-add}((x1, y1), i)((x2, y2), j) \mid x1\ y1\ i\ x2\ y2\ j.$ 
 $((x1, y1), i) \in \{((x, y), l), (\tau(x, y), Not\ l)\} \wedge$ 
 $((x2, y2), j) \in \{((x', y'), l'), (\tau(x', y'), Not\ l')\} \wedge ((x1, y1), x2, y2) \in$ 
 $e'\text{-aff-0} \cup e'\text{-aff-1}\} =$ 
 $\{(add(x, y)(x', y'), xor\ l\ l'), (\tau(add(x, y)(x', y'), Not(xor\ l\ l')))\}$ 
(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
 $e = proj\text{-add}((x1, y1), i)((x2, y2), j)$ 
 $((x1, y1), i) \in \{((x, y), l), (\tau(x, y), Not\ l)\}$ 
 $((x2, y2), j) \in \{((x', y'), l'), (\tau(x', y'), Not\ l')\}$ 
 $((x1, y1), x2, y2) \in e'\text{-aff-0} \cup e'\text{-aff-1}$  by blast
then have  $e = (add(x, y)(x', y'), xor\ l\ l') \vee$ 
 $e = (\tau(add(x, y)(x', y'), Not(xor\ l\ l')))$ 
using v1 v2 v3 v4 in-aff taus(1,2)
 $aaa\ ds\ ds'\ ld\text{-nz}$  by fastforce
then show  $e \in ?c$  by blast
qed
next
show ?s  $\supseteq$  ?c
proof
fix e
assume  $e \in ?c$ 
show  $e \in ?s$ 
proof(cases  $e = (add(x, y)(x', y'), xor\ l\ l')$ )
case True
have  $(add(x, y)(x', y'), xor\ l\ l') = proj\text{-add}((x, y), l)((x', y'), l')$ 
using v1 by presburger
then show ?thesis
using True b by blast
next
case False
then have  $e = (\tau(add(x, y)(x', y'), \neg xor\ l\ l')$ 
using  $\langle e \in ?c \rangle$  by fastforce
have eq:  $(\tau(add(x, y)(x', y'), \neg xor\ l\ l') = proj\text{-add}((x, y), l)(\tau(x',$ 
 $y'), \neg l')$ 
using v3 by presburger
have  $((x, y), \tau(x', y')) \in e'\text{-aff-0} \cup e'\text{-aff-1}$ 
proof(cases  $((x, y), \tau(x', y')) \in e'\text{-aff-0}$ )
case True
then show ?thesis by blast
next

```

```

      case False
      then have  $((x, y), \tau (x', y')) \in e'\text{-aff-1}$ 
        unfolding e'-aff-1-def e'-aff-0-def
        using aaa in-aff(1) taus(1) by force
      then show ?thesis
        by blast
      qed
      then show ?thesis
        using eq False  $\langle e = (\tau (\text{add } (x, y) (x', y')), \neg \text{xor } l l') \rangle$ 
        by force
      qed
    qed
  qed

show proj-addition ?p ?q = gluing “  $\{( \text{add } (x, y) (x', y'), \text{xor } l l')\}$ 
  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  $((x, y), l) (\tau (x', y'), \text{Not } l') = (\text{ext-add } (x, y) (\tau (x', y')), \text{Not}(\text{xor } l l'))$ 
  using proj-add.simps  $\langle (x, y) \in e'\text{-aff} \rangle \langle (\tau (x', y')) \in e'\text{-aff} \rangle$  xor-def by auto
  have pd: delta  $(\text{fst } (\tau (x, y))) (\text{snd } (\tau (x, y))) x' y' = 0$ 
    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'  $(\text{fst } (\tau (x, y))) (\text{snd } (\tau (x, y))) x' y' \neq 0$ 
    using bbb unfolding delta'-def delta-x-def delta-y-def
    by(simp add: t-nz nz field-simps)
  then have pd'': delta'  $x y (\text{fst } (\tau (x', y'))) (\text{snd } (\tau (x', y'))) \neq 0$ 
    unfolding delta'-def delta-x-def delta-y-def
    apply(simp add: divide-simps t-nz nz)
    by algebra
  have v4: proj-add  $(\tau (x, y), \text{Not } l) ((x', y'), l') = (\text{ext-add } (\tau (x, y)) (x', y'), \text{Not}(\text{xor } l l'))$ 
  Not(xor l l')
    using proj-add.simps in-aff taus pd pd' xor-def by auto
  have v3-eq-v4:  $(\text{ext-add } (x, y) (\tau (x', y')), \text{Not}(\text{xor } l l')) = (\text{ext-add } (\tau (x, y)) (x', y'), \text{Not}(\text{xor } l l'))$ 
    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), Not l)} ∧
  ((x2, y2), j) ∈ {(x', y'), l'}, (τ (x', y'), Not l')} ∧ ((x1, y1), x2, y2) ∈
e'-aff-0 ∪ e'-aff-1} =
  {(add (x, y) (x', y'), xor l l'), (τ (add (x, y) (x', y')), Not (xor 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
       $e = \text{proj-add } ((x1, y1), i) ((x2, y2), j)$ 
       $((x1, y1), i) \in \{((x, y), l), (\tau (x, y), \text{Not } l)\}$ 
       $((x2, y2), j) \in \{((x', y'), l'), (\tau (x', y'), \text{Not } l')\}$ 
       $((x1, y1), x2, y2) \in e'\text{-aff-0} \cup e'\text{-aff-1}$  by blast
    then have additional:
       $((x1, y1) \in e'\text{-aff} \wedge (x2, y2) \in e'\text{-aff} \wedge \text{delta } x1\ y1\ x2\ y2 \neq 0) \vee$ 
       $((x1, y1) \in e'\text{-aff} \wedge (x2, y2) \in e'\text{-aff} \wedge \text{delta}' x1\ y1\ x2\ y2 \neq 0)$ 
      unfolding  $e'\text{-aff-0-def } e'\text{-aff-1-def}$  by auto
    then have  $\text{proj-add } ((x1, y1), i) ((x2, y2), j) \in \{ (\text{add } (x1, y1) (x2, y2),$ 
xor i j),
       $(\text{ext-add } (x1, y1) (x2, y2), \text{xor } i\ j) \}$ 
    proof(cases  $\text{proj-add } ((x1, y1), i) ((x2, y2), j) = (\text{add } (x1, y1) (x2, y2),$ 
xor i j))
      case True
      then show  $?thesis$  by blast
      next
      case False
      then have  $((x1, y1) \in e'\text{-aff} \wedge (x2, y2) \in e'\text{-aff} \wedge \text{delta}' x1\ y1\ x2\ y2 \neq$ 
0)
        using additional  $\text{proj-add.simps}(1)$  by blast
        then have  $\text{proj-add } ((x1, y1), i) ((x2, y2), j) = (\text{ext-add } (x1, y1) (x2,$ 
y2), xor i j)
        using  $\text{proj-add.simps}(1)[\text{of } x1\ y1\ x2\ y2\ i\ j]$   $\text{proj-add.simps}(2)[\text{of } x1\ y1$ 
x2 y2 i j]
        by blast
        then show  $?thesis$ 
        by blast
      qed
    consider
      (1)  $((x1, y1), i) = ((x, y), l) ((x2, y2), j) = ((x', y'), l') \mid$ 
      (2)  $((x1, y1), i) = ((x, y), l) ((x2, y2), j) = (\tau (x', y'), \text{Not } l') \mid$ 
      (3)  $((x1, y1), i) = (\tau (x, y), \neg l) ((x2, y2), j) = ((x', y'), l') \mid$ 
      (4)  $((x1, y1), i) = (\tau (x, y), \neg l) ((x2, y2), j) = (\tau (x', y'), \text{Not } l')$ 
    using  $\langle ((x1, y1), i) \in \{((x, y), l), (\tau (x, y), \neg l)\} \rangle$ 
       $\langle ((x2, y2), j) \in \{((x', y'), l'), (\tau (x', y'), \text{Not } l')\} \rangle$ 
    by auto
    then have  $e \in \{ (\text{add } (x, y) (x', y'), \text{xor } l\ l'), (\tau (\text{add } (x, y) (x', y')), \text{Not}$ 
(xor l l') ) \}
    proof cases
      case 1
      then show  $?thesis$ 
      using  $\langle e = \text{proj-add } ((x1, y1), i) ((x2, y2), j) \rangle v1$  by fastforce
    next

```

```

      case 2
      then show ?thesis
        using ⟨e = proj-add ((x1, y1), i) ((x2, y2), j)⟩ trans-add(1) v3 by
auto
      next
      case 3
      then show ?thesis
        using ⟨e = proj-add ((x1, y1), i) ((x2, y2), j)⟩ trans-add(1) v3-eq-v4
v4 by auto
      next
      case 4
      then show ?thesis
        using ⟨e = proj-add ((x1, y1), i) ((x2, y2), j)⟩ v2 by auto
      qed
      then show e ∈ ?c by blast
    qed
  next
  show ?s ⊇ ?c
  proof(safe-step)
    fix e
    assume e ∈ ?c
    then have cases: e = (add (x, y) (x', y'), xor l l') ∨
      e = (τ (add (x, y) (x', y')), Not(xor l l')) by blast

    have (add (x, y) (x', y'), xor l l') ∈ ?s
    proof -
      have ((x,y),x',y') ∈ e'-aff-0 ∪ e'-aff-1
      by (simp add: b)
      then show ?thesis using v1
      apply(simp del: τ.simps add.simps ext-add.simps)
      apply safe
      apply(intro exI)
      apply auto[1]
      apply(intro exI)
      by auto
    qed

    moreover have (τ (add (x, y) (x', y')), Not(xor l l')) ∈ ?s
    proof -
      have (τ (add (x, y) (x', y')), Not(xor l l')) = proj-add ((x, y), l) (τ (x',
y'), ¬ l')
      using trans-add(1) v3 by presburger

      moreover have ((x, y), τ (x', y')) ∈ e'-aff-0 ∪ e'-aff-1
      unfolding e'-aff-0-def e'-aff-1-def
      using in-aff(1) pd'' taus(1) by auto

      ultimately show ?thesis
      apply(simp del: τ.simps add.simps ext-add.simps)

```



```

    apply safe
    apply(intro exI)
    apply auto[1]
    apply(intro exI)
    by auto
qed

ultimately show  $e \in ?s$ 
using local.cases by presburger
qed
qed

have ext-eq: gluing “  $\{(ext-add(x, y) (\tau(x', y')), Not(xor\ l\ l'))\} =$ 
 $\{(ext-add(x, y) (\tau(x', y')), Not(xor\ l\ l')), (\tau(ext-add(x, y) (\tau(x',$ 
 $y'))), xor\ l\ l')\}$ 
using add-nz add-closure gluing-class-2 by auto
have class-eq: gluing “  $\{(add(x, y) (x', y'), xor\ l\ l')\} =$ 
 $\{(add(x, y) (x', y'), xor\ l\ l'), (\tau(add(x, y) (x', y')), Not(xor\ l\ l'))\}$ 
proof –
  have gluing “  $\{(add(x, y) (x', y'), xor\ l\ l')\} =$ 
  gluing “  $\{(\tau(ext-add(x, y) (\tau(x', y'))), xor\ l\ l')\}$ 
  using trans-add by argo
  also have ... = gluing “  $\{(ext-add(x, y) (\tau(x', y')), Not(xor\ l\ l'))\}$ 
  using gluing-inv add-nz add-closure by auto
  also have ... =  $\{(ext-add(x, y) (\tau(x', y')), Not(xor\ l\ l')), (\tau(ext-add(x,$ 
 $y) (\tau(x', y'))), xor\ l\ l')\}$ 
  using ext-eq by blast
  also have ... =  $\{(add(x, y) (x', y'), xor\ l\ l'), (\tau(add(x, y) (x', y')), Not$ 
 $(xor\ l\ l'))\}$ 
  using trans-add by force
  finally show ?thesis by blast
qed

have ext-eq-proj: gluing “  $\{(ext-add(x, y) (\tau(x', y')), Not(xor\ l\ l'))\} \in e-proj$ 
using add-closure e-proj-aff by auto
then have class-proj: gluing “  $\{(add(x, y) (x', y'), xor\ l\ l')\} \in e-proj$ 
proof –
  have gluing “  $\{(add(x, y) (x', y'), xor\ l\ l')\} =$ 
  gluing “  $\{(\tau(ext-add(x, y) (\tau(x', y'))), xor\ l\ l')\}$ 
  using trans-add by argo
  also have ... = gluing “  $\{(ext-add(x, y) (\tau(x', y')), Not(xor\ l\ l'))\}$ 
  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'), Not l') = 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 field-simps power2-eq-square[symmetric] t-expr d-nz)

then have v4: proj-add (τ (x, y), Not l) ((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'), xor l l')} = {(add (x, y) (x', y'),
xor l l')}
using add-z add-closure gluing-class-1 by simp
have class-proj: gluing “{(add (x, y) (x', y'), xor 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), Not l)} ∧
  ((x2, y2), j) ∈ {(x', y'), l', (τ (x', y'), Not l')} ∧ ((x1, y1), x2, y2) ∈
e'-aff-0 ∪ e'-aff-1} =
  {(add (x, y) (x', y'), xor 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), Not l)}

```

```

      ((x2, y2), j) ∈ {(x', y'), l'}, (τ (x', y'), Not l')}
      ((x1, y1), x2, y2) ∈ e'-aff-0 ∪ e'-aff-1 by blast
    then have e = (add (x, y) (x', y'), xor 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 ∈ ?c by blast
  qed
next
show ?s ⊇ ?c
proof
  fix e
  assume e ∈ ?c
  then have e = (add (x, y) (x', y'), xor l l') by blast
  moreover have proj-add ((x, y), l) ((x', y'), l') = (add (x, y) (x', y'), xor
l l')
      using v1 by blast
      moreover have ((x,y),x',y') ∈ e'-aff-0 ∪ e'-aff-1
      by (simp add: b)
      ultimately show e ∈ ?s
      apply(simp del: τ.simps add.simps ext-add.simps)
      apply(intro exI)
      by auto
  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} ∈ e-proj gluing “ {(x2,y2),j} ∈ e-proj delta x1
y1 x2 y2 ≠ 0
  shows proj-addition (gluing “ {(x1,y1),l}) (gluing “ {(x2,y2),j}) =
  (gluing “ {(add (x1,y1) (x2,y2), xor l j)})
proof –
  have p-q-expr: (gluing “ {(x1,y1),l}) = {(x1, y1), l} ∨

```

```

      gluing “  $\{((x1,y1),l)\} = \{(x1, y1), l, (\tau (x1, y1), \text{Not } l)\}$ 
      (gluing “  $\{((x2,y2),j)\} = \{(x2, y2), j\} \vee$ 
      gluing “  $\{((x2,y2),j)\} = \{(x2, y2), j, (\tau (x2, y2), \text{Not } j)\}$ )
    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 “  $\{((x2,y2),j)\} = \{(x2, y2), j, (\tau (x2, y2), \text{Not } j)\} |$ 
    (3) gluing “  $\{((x1,y1),l)\} = \{(x1, y1), l, (\tau (x1, y1), \text{Not } l)\}$ 
        gluing “  $\{((x2,y2),j)\} = \{(x2, y2), j\} |$ 
    (4) gluing “  $\{((x1,y1),l)\} = \{(x1, y1), l, (\tau (x1, y1), \text{Not } l)\}$ 
        gluing “  $\{((x2,y2),j)\} = \{(x2, y2), j, (\tau (x2, y2), \text{Not } j)\}$  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 ≠ 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 aux: proj-addition (gluing “  $\{((x2, y2), j)\}$  (gluing “  $\{((x1, y1), l)\}$ )
=
      gluing “  $\{(add (x1, y1) (x2, y2), \text{xor } j\ l)\}$ 
      using gluing-add-2[OF 3(2) 3(1) assms(2) assms(1) pd] add-com
      by simp
  show ?thesis
  unfolding proj-addition-def
  apply(subst proj-add-class-comm[OF assms(1,2)])
  apply(subst proj-addition-def[symmetric])
  apply(subst aux)
  apply(simp add: xor-def)
  by argo
  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 “ $\{((x,y),l)\} \in e\text{-proj}$ gluing “ $\{((x',y'),l')\} \in e\text{-proj}$ delta' x y x' y'

$\neq 0$
shows *proj-addition* (*gluing* “ $\{(x,y,l)\}$ ”) (*gluing* “ $\{(x',y',l')\}$ ”) =
(*gluing* “ $\{(ext-add(x,y)(x',y'), xor\ l\ l')\}$ ”)
proof –
have *in-aff*: $(x,y) \in e'-aff\ (x',y') \in e'-aff$
using *assms e-proj-eq e-proj-aff* **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) $(x, y) \in e-circ \wedge (\exists g \in symmetries. (x', y') = (g \circ i)(x, y)) \mid$
(b) $((x, y), x', y') \in e'-aff-0$
 $\neg ((x, y) \in e-circ \wedge (\exists g \in symmetries. (x', y') = (g \circ i)(x, y))) \mid$
(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) ∈ 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*
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* “ $\{(x',y',l')\} = \{(x', y'),$
 $l'\}, (\tau(x', y'), Not\ l')\}$ ”
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* “
 $\{(ext-add(x,y)(x',y'), xor\ l\ l')\}$ ”)
proof –
have *in-aff*: $(x,y) \in e'-aff\ (x',y') \in e'-aff$
using *assms e-proj-eq e-proj-aff* **by** *blast+*
then have *add-in*: $ext-add(x, y)(x', y') \in e'-aff$
using *ext-add-closure (delta' x y x' y' ≠ 0) delta-def e-e'-iff e'-aff-def* **by** *auto*
from *in-aff* **have** *zeros*: $x = 0 \vee y = 0\ x' \neq 0\ y' \neq 0$

```

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 “  $\{(ext\text{-add } (x, y) (x', y'), xor\ l\ l')\} \in e\text{-proj}$ 
using e-proj-aff in-aff add-in by auto

consider
  (a)  $(x, y) \in e\text{-circ} \wedge (\exists g \in symmetries. (x', y') = (g \circ i) (x, y)) \mid$ 
  (b)  $((x, y), x', y') \in e'\text{-aff-0} \neg ((x, y) \in e\text{-circ} \wedge (\exists g \in symmetries. (x', y') =$ 
   $(g \circ i) (x, y)))$ 
   $((x, y), x', y') \notin e'\text{-aff-1} \mid$ 
  (c)  $((x, y), x', y') \in e'\text{-aff-1} \neg ((x, y) \in e\text{-circ} \wedge (\exists g \in symmetries. (x', y') =$ 
   $(g \circ i) (x, y)))$ 
  using dichotomy-1[OF  $\langle(x,y) \in e'\text{-aff}\rangle \langle(x',y') \in e'\text{-aff}\rangle]$  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  $\langle(x,y) \in e'\text{-aff}\rangle \langle(x',y') \in e'\text{-aff}\rangle$  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-add  $((x, y), l) ((x', y'), l') = (ext\text{-add } (x, y) (x', y'), xor\ l\ l')$ 
  by(simp add:  $\langle(x,y) \in e'\text{-aff}\rangle \langle(x',y') \in e'\text{-aff}\rangle$  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  $\langle(x',y') \in e'\text{-aff}\rangle$  by blast+
  then have  $\tau (x', y') \in e\text{-circ}$ 
  using zeros  $\tau\text{-circ}$  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') ∈ 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 (τ (x', y'))) (snd (τ (x', y'))) ≠ 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) (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'), Not l') = (τ (ext-add (x, y) (x', y')),
Not (xor l l'))
  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⟩ xor-def by auto

have gl-class: gluing “{(ext-add (x, y) (x', y'), xor l l')} =
  {(ext-add (x, y) (x', y'), xor l l'), (τ (ext-add (x, y) (x', y')), Not(xor
l l'))}
  gluing “{(ext-add (x, y) (x', y'), xor l l')} ∈ e-proj
  using gluing-class-2 add-nz add-in apply simp
  using e-proj-aff add-in by auto

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'), Not l')} ∧
  ((x1, y1), x2, y2)
  ∈ e'-aff-0 ∪ {(x1, y1), x2, y2}. (x1, y1) ∈ e'-aff ∧ (x2, y2) ∈ e'-aff ∧
delta' x1 y1 x2 y2 ≠ 0} =
  {proj-add ((x, y), l) ((x', y'), l'), proj-add ((x, y), l) (τ (x', y'), Not l')}
  (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'), xor l l'), (τ (ext-add (x, y) (x', y')),
Not(xor l l'))}
  using v1 v2 by presburger
  finally have eq: ?t = {(ext-add (x, y) (x', y'), xor l l'), (τ (ext-add (x, y)
(x', y')), Not(xor l l'))}
  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 “  $\{(x,y),l\} = \{(x, y), l, (\tau (x, y), \text{Not } l)\}$ 
gluing “  $\{(x',y'),l'\} = \{(x', y'), l', (\tau (x', y'), \text{Not } l')\}$ 
gluing “  $\{(x,y),l\} \in e\text{-proj}$  gluing “  $\{(x',y'),l'\} \in e\text{-proj}$ 
delta' x y x' y' ≠ 0
shows proj-addition (gluing “  $\{(x,y),l\}$ ) (gluing “  $\{(x',y'),l'\}$ ) = (gluing “
 $\{(ext\text{-add } (x,y) (x',y'),xor\ l\ l')\}$ )
(is proj-addition ?p ?q = -)
proof –
have in-aff:  $(x,y) \in e'\text{-aff } (x',y') \in e'\text{-aff}$ 
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:  $(x,y) \in e\text{-circ } (x',y') \in e\text{-circ}$ 
using in-aff e-circ-def nz by auto
then have taus:  $(\tau (x', y')) \in e'\text{-aff } (\tau (x, y)) \in e'\text{-aff } \tau (x', y') \in e\text{-circ}$ 
using  $\tau\text{-circ}$  circ-to-aff by auto

consider
(a)  $(x, y) \in e\text{-circ} \wedge (\exists g \in \text{symmetries}. (x', y') = (g \circ i) (x, y))$ 
| (b)  $((x, y), x', y') \in e'\text{-aff-0 } ((x, y), x', y') \notin e'\text{-aff-1}$ 
| (c)  $((x, y), x', y') \in e'\text{-aff-1}$ 
using dichotomy-1[OF in-aff] by auto
then show ?thesis
proof(cases)
case a
then obtain g where sym-expr:  $g \in \text{symmetries } (x', y') = (g \circ i) (x, y)$  by
auto
then have ds:  $\text{delta } x\ y\ x'\ y' = 0\ \text{delta}'\ x\ y\ x'\ y' = 0$ 
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' ≠ 0
  unfolding e'-aff-1-def by auto
  then have ds: delta' (fst (τ (x, y))) (snd (τ (x, y))) (fst (τ (x', y'))) (snd (τ
(x', y'))) ≠ 0
    unfolding delta'-def delta-x-def delta-y-def
    by(simp add: t-nz field-simps nz)

  have v1: proj-add ((x, y), l) ((x', y'), l') = (ext-add (x, y) (x', y'), xor l l')
    using ld-nz proj-add.simps ⟨(x,y) ∈ e'-aff⟩ ⟨(x',y') ∈ e'-aff⟩ by simp
  have v2: proj-add (τ (x, y), Not l) (τ (x', y'), Not l') = (ext-add (x, y) (x',
y'), xor 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(safe)
  apply(rule inversion-invariance-2[OF nz(1,2), of fst (τ (x',y')) snd (τ (x',y')),
simplified prod.collapse tau-idemp-point])
  using nz t-nz xor-def by auto

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 (ext-add (x, y) (τ (x', y'))) = ext-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 tauI: τ (ext-add (x, y) (x', y')) = ext-add (x, y) (τ (x', y'))
    apply(subst tau-idemp-point[of ext-add (x, y) (τ (x', y')), symmetric])
    apply(subst tau-conv)
    by blast

  have v3:
    proj-add ((x, y), l) (τ (x', y'), Not l') = (τ (ext-add (x, y) (x', y')), Not(xor
l l'))
  apply(subst tauI)
  using aaa in-aff(1) taus(1) xor-def by fastforce

```

```

have ds': delta' (fst (τ (x, y))) (snd (τ (x, y))) x' y' ≠ 0
  using aaa unfolding delta'-def delta-x-def delta-y-def
  by(simp add: field-simps t-nz nz)

have v4: proj-add (τ (x, y), Not l) ((x', y'), l') = (τ (ext-add (x, y) (x', y')),
Not(xor l l'))
  proof –
    have proj-add (τ (x, y), Not l) ((x', y'), l') = (ext-add (τ (x, y)) (x', y'),
Not (xor l l'))
      using proj-add.simps ⟨τ (x,y) ∈ e'-aff⟩ ⟨(x', y') ∈ e'-aff⟩ ds' xor-def by
auto
    moreover have ext-add (τ (x, y)) (x', y') = τ (ext-add (x, y) (x', y'))
      using inversion-invariance-2 nz tauI by presburger
    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'), xor l l')} =
  {(ext-add (x, y) (x', y'), xor l l'), (τ (ext-add (x, y) (x', y')), Not (xor l
l'))}
  using add-nz add-closure gluing-class-2 by auto
have class-proj: gluing “ {(ext-add (x, y) (x', y'), xor 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), Not l)} ∧
((x2, y2), j) ∈ {(x', y'), l'), (τ (x', y'), Not l')} ∧ ((x1, y1), x2, y2) ∈
e'-aff-0 ∪ e'-aff-1} =
  {(ext-add (x, y) (x', y'), xor l l'), (τ (ext-add (x, y) (x', y')), Not (xor 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), Not l)}
  ((x2, y2), j) ∈ {(x', y'), l'}, (τ (x', y'), Not l')}
  ((x1, y1), x2, y2) ∈ e'-aff-0 ∪ e'-aff-1 by blast
then have e = (ext-add (x, y) (x', y'), xor l l') ∨
  e = (τ (ext-add (x, y) (x', y')), Not (xor l l'))
  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 as: e ∈ ?c
  then have cases: e = proj-add ((x, y), l) (τ (x', y'), ¬ l') ∨
  e = proj-add (τ (x, y), ¬ l) (τ (x', y'), ¬ l')
  using v2 v3 by auto
  have as1: ((x, y), τ (x', y')) ∈ e'-aff-0 ∪ e'-aff-1
  unfolding e'-aff-0-def e'-aff-1-def
  using ds taus aaa in-aff(1) by auto
  have as2: (τ (x, y), τ (x', y')) ∈ e'-aff-0 ∪ e'-aff-1
  unfolding e'-aff-0-def e'-aff-1-def
  using ds taus(1) taus(2) by auto
  consider
  (1) e = proj-add ((x, y), l) (τ (x', y'), ¬ l') |
  (2) e = proj-add (τ (x, y), ¬ l) (τ (x', y'), ¬ l')
  using cases by auto
  then show e ∈ ?s
  proof(cases)
    case 1
    then show ?thesis
    using as2 as1 by force
  next
    case 2
    then show ?thesis
    using as2 as1 by force
  qed
qed
qed
show proj-addition ?p ?q = gluing “{(ext-add (x, y) (x', y'), xor 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'), Not l') = (add (x, y) (τ (x', y')), Not (xor l
l'))
    using proj-add.simps ⟨(x,y) ∈ e'-aff⟩ ⟨(τ (x', y')) ∈ e'-aff⟩ xor-def by auto
  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: field-simps t-nz nz t-expr power2-eq-square[symmetric] d-nz)
  have v4: proj-add (τ (x, y), Not l) ((x', y'), l') = (add (τ (x, y)) (x', y'),
Not(xor l l'))
    using proj-add.simps in-aff taus pd pd' xor-def by auto
  have v3-eq-v4: (add (x, y) (τ (x', y')), Not(xor l l')) = (add (τ (x, y)) (x',
y'), Not(xor l l'))
    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: field-simps t-nz nz 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) ∈ {(x, y), l}, (τ (x, y), Not l)} ∧
    ((x2, y2), j) ∈ {(x', y'), l'}, (τ (x', y'), Not l')} ∧ ((x1, y1), x2, y2) ∈
e'-aff-0 ∪ e'-aff-1} =
    {(ext-add (x, y) (x', y'), xor l l'), (τ (ext-add (x, y) (x', y')), Not (xor 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 cases:
      e = proj-add ((x1, y1), i) ((x2, y2), j)
      ((x1, y1), i) ∈ {(x, y), l}, (τ (x, y), Not l)}
      ((x2, y2), j) ∈ {(x', y'), l'}, (τ (x', y'), Not l')}
      ((x1, y1), x2, y2) ∈ e'-aff-0 ∪ e'-aff-1 by blast
    consider
      (1) ((x1, y1), i) = (x, y), l ((x2, y2), j) = (x', y'), l' |
      (2) ((x1, y1), i) = (x, y), l ((x2, y2), j) = (τ (x', y'), Not l') |
      (3) ((x1, y1), i) = (τ (x, y), Not l) ((x2, y2), j) = (x', y'), l' |
      (4) ((x1, y1), i) = (τ (x, y), Not l) ((x2, y2), j) = (τ (x', y'), Not l')
    using cases by fast
    then have e = (ext-add (x, y) (x', y'), xor l l') ∨
      e = (τ (ext-add (x, y) (x', y')), Not (xor l l'))
  proof(cases)

```

```

    case 1
    then show ?thesis
      using local.cases(1) v1 by presburger
  next
    case 2
    then show ?thesis
      using local.cases(1) trans-add(1) v3 by presburger
  next
    case 3
    then show ?thesis
      using local.cases(1) trans-add(1) v3-eq-v4 v4 by presburger
  next
    case 4
    then show ?thesis
      using local.cases(1) v2 by presburger
  qed
  then show  $e \in ?c$  by fast
qed
next
show  $?s \supseteq ?c$ 
proof
  fix e
  assume  $e \in ?c$ 
  then consider
    (1)  $e = (\text{ext-add } (x, y) (x', y'), \text{xor } l \ l') \mid$ 
    (2)  $e = (\tau (\text{ext-add } (x, y) (x', y')), \text{Not}(\text{xor } l \ l'))$  by blast
  then show  $e \in ?s$ 
  proof(cases)
    case 1
    have eq:  $(\text{ext-add } (x, y) (x', y'), \text{xor } l \ l') = \text{proj-add } ((x, y), l) ((x', y'), l')$ 
      using ds taus(1) taus(2) v1 by auto
    show ?thesis
      apply(subst 1)
      apply(clarify)
      apply(subst eq)
      using c by blast
    next
    case 2
    have eq:  $(\tau (\text{ext-add } (x, y) (x', y')), \text{Not} (\text{xor } l \ l')) = \text{proj-add } ((x, y), l)$ 
       $(\tau (x', y'), \text{Not } l')$ 
      using taus in-aff(1) pd'' trans-add(1) v3 by presburger
    have ina:  $((x, y), \tau (x', y')) \in e'\text{-aff-0} \cup e'\text{-aff-1}$ 
      by (metis UnI1 UnI2 dichotomy-1 in-aff(1) pd'' prod.collapse taus(1)
      wd-d-nz)
    show ?thesis
      apply(subst 2)
      apply(clarify)
      apply(subst eq)
      using ina by fastforce
  qed

```

```

    qed
  qed
  qed

  have ext-eq: gluing “ {(add (x, y) (τ (x', y^)), Not (xor l l'))} =
    {(add (x, y) (τ (x', y^)), Not (xor l l')), (τ (add (x, y) (τ (x', y^))), xor l
l')}
    using add-nz add-closure gluing-class-2 by auto
  have class-eq: gluing “ {(ext-add (x, y) (x', y^), xor l l')} =
    {(ext-add (x, y) (x', y^), xor l l'), (τ (ext-add (x, y) (x', y^)), Not (xor l
l'))}
  proof –
    have gluing “ {(ext-add (x, y) (x', y^), xor l l')} =
      gluing “ {(τ (add (x, y) (τ (x', y^))), xor l l')}
      using trans-add by argo
    also have ... = gluing “ {(add (x, y) (τ (x', y^)), Not (xor l l'))}
      using gluing-inv add-nz add-closure by auto
    also have ... = {(add (x, y) (τ (x', y^)), Not (xor l l')), (τ (add (x, y) (τ
(x', y^))), xor l l')}
      using ext-eq by blast
    also have ... = {(ext-add (x, y) (x', y^), xor l l'), (τ (ext-add (x, y) (x',
y^)), Not (xor l l'))}
      using trans-add by force
    finally show ?thesis by blast
  qed

  have ext-eq-proj: gluing “ {(add (x, y) (τ (x', y^)), Not (xor l l'))} ∈ e-proj
    using add-closure e-proj-aff by auto
  then have class-proj: gluing “ {(ext-add (x, y) (x', y^), xor l l')} ∈ e-proj
  proof –
    have gluing “ {(ext-add (x, y) (x', y^), xor l l')} =
      gluing “ {(τ (add (x, y) (τ (x', y^))), xor l l')}
      using trans-add by argo
    also have ... = gluing “ {(add (x, y) (τ (x', y^)), Not (xor l l'))}
      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) ( $\tau$  (x', y'), Not l') = undefined by simp
from ccc have ds': delta (fst ( $\tau$  (x, y))) (snd ( $\tau$  (x, y))) x' y' = 0
      delta' (fst ( $\tau$  (x, y))) (snd ( $\tau$  (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 field-simps power2-eq-square[symmetric] t-expr d-nz)
then have v4: proj-add ( $\tau$  (x, y), Not l) ((x', y'), l') = undefined by simp

have add-z: fst (ext-add (x, y) (x', y')) = 0  $\vee$  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')  $\in$  e'-aff
      using c(1) <(x,y)  $\in$  e'-aff> <(x',y')  $\in$  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'), xor l l')} = {(ext-add (x, y)
(x', y'), xor l l')}
      using add-z add-closure gluing-class-1 by simp
have class-proj: gluing “ {(ext-add (x, y) (x', y'), xor l l')}  $\in$  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), Not l)}  $\wedge$ 
  ((x2, y2), j)  $\in$  {( (x', y'), l'), ( $\tau$  (x', y'), Not l')}  $\wedge$  ((x1, y1), x2, y2)  $\in$ 
e'-aff-0  $\cup$  e'-aff-1} =
  {(ext-add (x, y) (x', y'), xor 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-add ((x1, y1), i) ((x2, y2), j)
    ((x1, y1), i)  $\in$  {( (x, y), l), ( $\tau$  (x, y), Not l)}
    ((x2, y2), j)  $\in$  {( (x', y'), l'), ( $\tau$  (x', y'), Not l')}
    ((x1, y1), x2, y2)  $\in$  e'-aff-0  $\cup$  e'-aff-1 by blast
  then have e = (ext-add (x, y) (x', y'), xor 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  $e \in ?c$  by blast
qed
next
show  $?s \supseteq ?c$ 
proof
  fix e
  assume  $e \in ?c$ 
  then have eq:  $e = (\text{ext-add } (x, y) (x', y'), \text{xor } l l')$  by blast
  have  $(\text{ext-add } (x, y) (x', y'), \text{xor } l l') =$ 
     $\text{proj-add } ((x, y), l) ((x', y'), l')$ 
    using v1 by presburger
  show  $e \in ?s$ 
  apply(subst eq)
  apply(subst v1[symmetric])
  apply(clarify)
  using c by blast
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 “  $\{(x1, y1), l\} \in e\text{-proj}$  gluing “  $\{(x2, y2), j\} \in e\text{-proj}$  delta'
   $x1\ y1\ x2\ y2 \neq 0$ 
  shows proj-addition (gluing “  $\{(x1, y1), l\}$ ) (gluing “  $\{(x2, y2), j\}$ ) =
    (gluing “  $\{(\text{ext-add } (x1, y1) (x2, y2), \text{xor } l j)\}$ )
proof -
  have p-q-expr: (gluing “  $\{(x1, y1), l\} = \{(x1, y1), l\} \vee$ 
    gluing “  $\{(x1, y1), l\} = \{(x1, y1), l, (\tau (x1, y1), \text{Not } l)\}$ )
    (gluing “  $\{(x2, y2), j\} = \{(x2, y2), j\} \vee$ 
    gluing “  $\{(x2, y2), j\} = \{(x2, y2), j, (\tau (x2, y2), \text{Not } j)\}$ )
  using assms(1,2) gluing-cases-explicit by auto
  then consider
    (1) gluing “  $\{(x1, y1), l\} = \{(x1, y1), l\}$ 
      gluing “  $\{(x2, y2), j\} = \{(x2, y2), j\} \mid$ 
    (2) gluing “  $\{(x1, y1), l\} = \{(x1, y1), l\}$ 
      gluing “  $\{(x2, y2), j\} = \{(x2, y2), j, (\tau (x2, y2), \text{Not } j)\} \mid$ 
    (3) gluing “  $\{(x1, y1), l\} = \{(x1, y1), l, (\tau (x1, y1), \text{Not } l)\}$ 
      gluing “  $\{(x2, y2), j\} = \{(x2, y2), j\} \mid$ 

```

(4) *gluing* “ $\{(x1,y1),l\} = \{(x1, y1), l, (\tau (x1, y1), \text{Not } l)\}$
gluing “ $\{(x2,y2),j\} = \{(x2, y2), j, (\tau (x2, y2), \text{Not } j)\}$ **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 ≠ 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*)

using *assms by auto*

also have *... = gluing “ $\{(ext-add (x2, y2) (x1, y1), xor j l)\}$*

using *gluing-ext-add-2[OF 3(2,1) assms(2,1) pd] by blast*

also have *... = gluing “ $\{(ext-add (x1, y1) (x2, y2), xor l j)\}$*

apply(*subst ext-add-comm*)

apply(*simp add: xor-def del: add.simps ext-add.simps*)

by *argo*

finally show *?thesis by fast*

qed

next

case 4 then show *?thesis using gluing-ext-add-4 assms by presburger*

qed

qed

lemma *gluing-ext-add-points:*

assumes *gluing “ $\{(p1,l)\} \in e\text{-proj}$ gluing “ $\{(p2,j)\} \in e\text{-proj}$ delta' (fst p1) (snd p1) (fst p2) (snd p2) ≠ 0*

shows *proj-addition (gluing “ $\{(p1,l)\}$) (gluing “ $\{(p2,j)\}$) =*
 (gluing “ $\{(ext-add p1 p2, xor l j)\}$)

proof –

obtain *x1 y1 x2 y2 where p1 = (x1,y1) p2 = (x2,y2)*

by *fastforce*

then show *?thesis*

using *assms(1) assms(2) assms(3) gluing-ext-add by auto*

qed

3.4.3 Basic properties

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  $p\text{-}q\text{-}expr$ :  $p = gluing\ \{\{(x,y),l\}\}$ 
            $q = gluing\ \{\{(x',y'),l'\}\}$ 
  using  $e\text{-}proj\text{-}def\ assms$ 
  apply( $simp$ )
  apply( $elim\ quotientE$ )
  by  $force$ 
then have  $in\text{-}aff$ :  $(x,y) \in e'\text{-}aff$ 
            $(x',y') \in e'\text{-}aff$ 
  using  $e\text{-}proj\text{-}aff\ assms$  by  $auto$ 

consider
  (a)  $(x, y) \in e\text{-}circ \wedge (\exists g \in symmetries. (x', y') = (g \circ i) (x, y))$ 
  | (b)  $((x, y), x', y') \in e'\text{-}aff\text{-}0$ 
            $((x, y), x', y') \notin e'\text{-}aff\text{-}1$ 
            $(x, y) \notin e\text{-}circ \vee \neg (\exists g \in symmetries. (x', y') = (g \circ i) (x, y))$ 
  | (c)  $((x, y), x', y') \in e'\text{-}aff\text{-}1$ 
  using  $dichotomy\text{-}1[OF\ in\text{-}aff]$  by  $auto$ 
then show  $?thesis$ 
proof( $cases$ )
  case  $a$ 
  then obtain  $g$  where  $sym\text{-}expr$ :  $g \in symmetries\ (x', y') = (g \circ i) (x, y)$  by
 $auto$ 
  then have  $ds$ :  $delta\ x\ y\ x'\ y' = 0\ delta'\ x\ y\ x'\ y' = 0$ 
  using  $wd\text{-}d\text{-}nz\ wd'\text{-}d'\text{-}nz\ a$  by  $auto$ 
  have  $nz$ :  $x \neq 0\ y \neq 0\ x' \neq 0\ y' \neq 0$ 
  proof –
  from  $a$  show  $x \neq 0\ y \neq 0$ 
  unfolding  $e\text{-}circ\text{-}def$  by  $auto$ 
  then show  $x' \neq 0\ y' \neq 0$ 
  using  $sym\text{-}expr\ t\text{-}nz$ 
  unfolding  $symmetries\text{-}def\ e\text{-}circ\text{-}def$ 
  by  $auto$ 
  qed
  have  $taus$ :  $\tau (x',y') \in e'\text{-}aff$ 
  using  $in\text{-}aff(2)\ e\text{-}circ\text{-}def\ nz(3,4)\ \tau\text{-}circ$  by  $force$ 
  then have  $proj$ :  $gluing\ \{\{(\tau (x', y'), Not\ l')\}\} \in e\text{-}proj$ 
            $gluing\ \{\{(x, y), l\}\} \in e\text{-}proj$ 
  using  $e\text{-}proj\text{-}aff\ in\text{-}aff$  by  $auto$ 

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

  have  $proj\text{-}addition\ p\ q = proj\text{-}addition\ (gluing\ \{\{(x, y), l\}\})\ (gluing\ \{\{(x',$ 
 $y'), l'\}\})$ 
  (is  $?lhs = proj\text{-}addition\ ?p\ ?q$ )
  unfolding  $p\text{-}q\text{-}expr$  by  $simp$ 

```

also have ... = proj-addition ?p (gluing “ {(τ (x' , y'), Not l')}
 (is - = ?rhs)
using gluing-inv nz in-aff **by** presburger
finally have eq: ?lhs = ?rhs
by auto

have eqs:
 ?d1 \neq 0 \implies ?lhs = gluing “ {(add (x , y) (τ (x' , y')), Not (xor l'))}
 ?d2 \neq 0 \implies ?lhs = gluing “ {(ext-add (x , y) (τ (x' , y')), Not (xor l'))}
subgoal
 apply(subst eq)
 apply(simp del: add.simps)
 apply(subst gluing-add)
 apply(simp-all del: τ .simps add: τ .simps[symmetric])
 apply(rule proj(2))
 apply(rule proj(1))
 apply(rule gluing-eq)
 apply(simp add: xor-def)
 by blast
subgoal
 apply(subst eq)
 apply(simp del: ext-add.simps)
 apply(subst gluing-ext-add)
 apply(rule proj(2))
 apply(simp-all del: τ .simps add: τ .simps[symmetric])
 apply(rule proj(1))
 apply(rule gluing-eq)
 apply(simp add: xor-def)
 by blast
done

have closures:
 ?d1 \neq 0 \implies add (x , y) (τ (x' , y')) \in e'-aff
 ?d2 \neq 0 \implies 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 \implies gluing “ {(add (x , y) (τ (x' , y')), Not(xor l'))} \in
 e-proj
 ?d2 \neq 0 \implies gluing “ {(ext-add (x , y) (τ (x' , y')), Not(xor l'))} \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' ≠ 0
  unfolding e'-aff-0-def by auto

have eq: proj-addition p q = gluing “ {(add (x, y) (x',y'), xor 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') ∈ 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' ≠ 0
  unfolding e'-aff-1-def by auto

have eq: proj-addition p q = gluing “ {(ext-add (x, y) (x',y'), xor 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') ∈ 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

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),
xor 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

  consider (1) delta x y x (-y) ≠ 0 | (2) delta' x y x (-y) ≠ 0
  using add-self in-aff by blast
  then show proj-addition (gluing “ {(x,y,l)}) (gluing “ {(i (x,y),l')}) = {(1,
0), xor l l'}
  proof (cases)
    case 1

```

```

have add (x,y) (i (x,y)) = (1,0)
  using 1 delta-def delta-minus-def delta-plus-def in-aff inverse-generalized by
auto
  then show ?thesis
    using 1 assms gluing-add i-proj identity-equiv by auto
next
  case 2
  have ext-add (x,y) (i (x,y)) = (1,0)
    using 2 delta'-def delta-x-def by auto
  then show ?thesis
    using 2 assms gluing-ext-add i-proj identity-equiv by auto
qed
qed

```

lemma *proj-add-class-inv-point*:

```

assumes gluing “ {(p,l)} ∈ e-proj ne = (1,0)
shows proj-addition (gluing “ {(p,l)}) (gluing “ {(i p,l')}) = {(ne, xor l l')}
  gluing “ {(i p,l')} ∈ e-proj
proof –
  obtain x y where p: p = (x,y) by fastforce
  then show proj-addition (gluing “ {(p,l)}) (gluing “ {(i p,l')}) = {(ne, xor l l')}
    using assms(1) assms(2) prod.collapse proj-add-class-inv(1) by simp
  from p show gluing “ {(i p,l')} ∈ e-proj
    using assms proj-add-class-inv(2) surj-pair by blast
qed

```

lemma *proj-add-class-identity*:

```

assumes x ∈ e-proj
shows proj-addition {(1, 0), False} 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), False} x =
  proj-addition (gluing “ {(1, 0), False}) (gluing “ {(x0,y0),l0})
  using identity-equiv[of False] 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
  apply(rule gluing-eq)
  using xor-def by presburger

```

also have $\dots = \text{gluing } \{((x0, y0), l0)\}$
using *inverse-generalized in-aff*
unfolding *e'-aff-def* **by** *simp*
also have $\dots = x$
using *x-expr* **by** *simp*
finally show *?thesis* **by** *simp*
qed

corollary *proj-addition-comm*:
assumes $c1 \in e\text{-proj } c2 \in e\text{-proj}$
shows *proj-addition* $c1 c2 = \text{proj-addition } c2 c1$
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**
 $tf\ g = \text{image } (\lambda\ p. (g\ (\text{fst } p), \text{snd } p))$

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*

lemma *tf-cong*:
 $f = f' \implies s = s' \implies tf\ f\ s = tf\ f'\ s'$
by *auto*

definition *tf'* **where**
 $tf' = \text{image } (\lambda\ p. (\text{fst } p, \text{Not } (\text{snd } p)))$

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

have *eq*: *gluing* “ $\{(x, y), l\} = \{(x, y), l\} \vee$
gluing “ $\{(x, y), l\} = \{(x, y), l, (\tau(x, y), \text{Not } l)\}$
using *assms gluing-cases-explicit* **by** *auto*
from *eq* **consider**
(1) *gluing* “ $\{(x, y), l\} = \{(x, y), l\} \mid$
(2) *gluing* “ $\{(x, y), l\} = \{(x, y), l, (\tau(x, y), \text{Not } l)\}$
by *fast*
then show *tf* ϱ (*gluing* “ $\{(x, y), l\} \in e\text{-proj}$)
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*: $x \neq 0 \wedge y \neq 0$
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 *rho-preserv-e-proj-point*:
assumes *gluing* “ $\{p\} \in e\text{-proj}$
shows *tf* ϱ (*gluing* “ $\{p\} \in e\text{-proj}$)
proof –
obtain *x y l* **where** $p = ((x, y), l)$
using *surj-pair[of p]* **by** *force*
then show *?thesis*
using *rho-preserv-e-proj assms* **by** *blast*
qed

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

have *eq*: *gluing* “ $\{(x, y), l\} = \{(x, y), l\} \vee$
gluing “ $\{(x, y), l\} = \{(x, y), l, (\tau(x, y), \text{Not } l)\}$
using *assms gluing-cases-explicit* **by** *auto*
from *eq* **consider**


```

(1) gluing “  $\{((x, y), l)\} = \{(x, y), l\} \mid$ 
(2) gluing “  $\{((x, y), l)\} = \{(x, y), l, (\tau(x, y), \text{Not } l)\}$ 
by fast
then show tf  $\varrho$  (gluing “  $\{((x, y), l)\} = \text{gluing}$  “  $\{\varrho(x, y), l\}$ )
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
  have gluing “  $\{\varrho(x, y), l\} = \{\varrho(x, y), l\}$ 
  apply(rule gluing-class-1[of fst ( $\varrho(x, y)$ ) snd ( $\varrho(x, y)$ ),
    simplified prod.collapse, OF - rho-aff])
  using zeros by auto
  then show ?thesis
  unfolding tf-def image-def 1 by simp
next
  case 2
  have zeros:  $x \neq 0 \wedge y \neq 0$ 
  using in-aff e-proj-elim-2 assms e-proj-aff 2 by auto
  then have gluing “  $\{\varrho(x, y), l\} = \{\varrho(x, y), l, (\tau(\varrho(x, y)), \text{Not } l)\}$ 
  using gluing-class-2[of fst ( $\varrho(x, y)$ ) snd ( $\varrho(x, y)$ ),
    simplified prod.collapse, OF - - rho-aff] by force
  then show ?thesis
  unfolding tf-def image-def 2 by force
qed
qed

lemma insert-rho-gluing-point:
  assumes gluing “  $\{(p, l)\} \in e\text{-proj}$ 
  shows tf  $\varrho$  (gluing “  $\{(p, l)\} = \text{gluing}$  “  $\{\varrho p, l\}$ )
proof –
  obtain x y where  $p = (x, y)$ 
  by fastforce
  then show ?thesis
  using assms insert-rho-gluing by presburger
qed

lemma rotation-preserv-e-proj:
  assumes gluing “  $\{((x, y), l)\} \in e\text{-proj}$   $r \in \text{rotations}$ 
  shows tf  $r$  (gluing “  $\{((x, y), l)\} \in e\text{-proj}$ 
  (is tf  $?r$   $?g \in -$ )
  using assms
  unfolding rotations-def
  apply(safe)
  apply(subst tf-id[of  $?g$ ], simp)
  apply(rule rho-preserv-e-proj, simp)
  apply(subst tf-comp[symmetric])
  using  $\varrho$ .simps insert-rho-gluing rho-preserv-e-proj apply presburger
  apply(subst tf-comp[symmetric])
  apply(subst tf-comp[symmetric])

```

```

using  $\varrho$ .simps insert-rho-gluing rho-preserv-e-proj by presburger

lemma rotation-preserv-e-proj-point:
  assumes gluing “  $\{p\} \in e\text{-proj } r \in \text{rotations}$ 
  shows  $tf\ r\ (\text{gluing “ } \{p\} \in e\text{-proj}$ 
proof –
  obtain  $x\ y\ l$  where  $p = ((x,y),l)$ 
  using surj-pair[of  $p$ ] by force
  then show ?thesis
  using rotation-preserv-e-proj assms by blast
qed

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

  consider (1)  $r = id$  |
    (2)  $r = \varrho$  |
    (3)  $r = \varrho \circ \varrho$  |
    (4)  $r = \varrho \circ \varrho \circ \varrho$ 
  using assms(2) unfolding rotations-def by fast
  then show ?thesis
proof(cases)
  case 1
  then show ?thesis using tf-id by auto
next
  case 2
  then show ?thesis using insert-rho-gluing assms by presburger
next
  case 3
  show ?thesis
  apply(subst 3)
  apply(subst tf-comp[symmetric])
  using 3 assms(1) in-proj(1) insert-rho-gluing by auto
next
  case 4
  then show ?thesis
  apply(subst 4)
  apply(subst tf-comp[symmetric])+
  using assms(1) insert-rho-gluing ext-curve-addition-axioms in-proj(1) in-proj(2)
by fastforce
qed
qed

```

lemma *insert-rotation-gluing-point*:
assumes *gluing* “ $\{(p, l)\} \in e\text{-proj}$ $r \in \text{rotations}$
shows $tf\ r$ (*gluing* “ $\{(p, l)\} = \text{gluing}$ “ $\{(r\ p, l)\}$)
proof –
obtain $x\ y$ **where** $p = (x, y)$ **by** *fastforce*
then show *?thesis*
using *assms(1) assms(2) insert-rotation-gluing by force*
qed

lemma *tf-tau*:
assumes *gluing* “ $\{((x, y), l)\} \in e\text{-proj}$
shows *gluing* “ $\{((x, y), \text{Not } l)\} = tf'$ (*gluing* “ $\{((x, y), l)\}$)
using *assms unfolding symmetries-def*
proof –
have *in-aff*: $(x, y) \in e'\text{-aff}$
using *e-proj-aff assms by simp*

have *gl-expr*: *gluing* “ $\{((x, y), l)\} = \{((x, y), l)\} \vee$
gluing “ $\{((x, y), l)\} = \{((x, y), l), (\tau\ (x, y), \text{Not } l)\}$
using *assms(1) gluing-cases-explicit by simp*

consider (1) *gluing* “ $\{((x, y), l)\} = \{((x, y), l)\} \mid$
(2) *gluing* “ $\{((x, y), l)\} = \{((x, y), l), (\tau\ (x, y), \text{Not } l)\}$
using *gl-expr by argo*
then show *gluing* “ $\{((x, y), \text{Not } l)\} = tf'$ (*gluing* “ $\{((x, y), l)\}$)
proof(*cases*)
case 1
then have *zeros*: $x = 0 \vee y = 0$
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*: $x \neq 0\ y \neq 0$
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* “ $\{((x, y), l)\} \in e\text{-proj}$
shows tf' (*gluing* “ $\{((x, y), l)\}$) $\in e\text{-proj}$
using *assms tf-tau[OF assms]*
e-proj-aff[of $x\ y\ l$] e-proj-aff[of $x\ y\ \text{Not } l$] by auto

lemma *tf-preserv-e-proj-point*:

```

assumes gluing “  $\{p\} \in e\text{-proj}$ 
shows tf' (gluing “  $\{p\} \in e\text{-proj}$ )
proof –
  obtain  $x\ y\ l$  where  $p = ((x,y),l)$ 
    using surj-pair[of p] by force
  then show ?thesis
    using tf-preserv-e-proj assms by blast
qed

lemma remove-rho:
assumes gluing “  $\{((x,y),l)\} \in e\text{-proj}$ 
shows gluing “  $\{\varrho(x,y),l\} = \text{tf } \varrho$  (gluing “  $\{((x,y),l)\}$ )
using assms unfolding symmetries-def
proof –
  have in-aff:  $(x,y) \in e'\text{-aff}$  using assms e-proj-aff by simp
  have rho-aff:  $\varrho(x,y) \in e'\text{-aff}$ 
    using in-aff unfolding e'-aff-def e'-def by (simp, algebra)

consider (1) gluing “  $\{((x,y),l)\} = \{((x,y),l)\} \mid$ 
  (2) gluing “  $\{((x,y),l)\} = \{((x,y),l), (\tau(x,y), \text{Not } l)\}$ 
  using assms gluing-cases-explicit by blast
then show gluing “  $\{\varrho(x,y), l\} = \text{tf } \varrho$  (gluing “  $\{((x,y), l)\}$ )
proof(cases)
  case 1
  then have zeros:  $x = 0 \vee y = 0$ 
    using assms e-proj-elim-1 in-aff by simp
  then have rho-zeros:  $\text{fst } (\varrho(x,y)) = 0 \vee \text{snd } (\varrho(x,y)) = 0$ 
    by force
  have gl-eq: gluing “  $\{\varrho(x,y), l\} = \{\varrho(x,y), l\}$ 
    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:  $x \neq 0 \wedge y \neq 0$ 
    using assms e-proj-elim-2 in-aff by auto
  then have rho-zeros:  $\text{fst } (\varrho(x,y)) \neq 0 \wedge \text{snd } (\varrho(x,y)) \neq 0$ 
    using t-nz by auto
  have gl-eqs: gluing “  $\{\varrho(x,y), l\} = \{\varrho(x,y), l, (\tau(\varrho(x,y)), \text{Not } l)\}$ 
    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* “ $\{(x,y,l)\} \in e\text{-proj } r \in \text{rotations}$
shows *gluing* “ $\{(r(x,y),l)\} = \text{tf } r (\text{gluing “ } \{(x,y,l)\})$
proof –
consider (1) $r = \text{id}$ |
(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 *assms(1) assms(2) insert-rotation-gluing by presburger*
qed
qed

lemma *remove-tau*:
assumes *gluing* “ $\{(x,y,l)\} \in e\text{-proj } \text{gluing “ } \{(\tau(x,y),l)\} \in e\text{-proj}$
shows *gluing* “ $\{(\tau(x,y),l)\} = \text{tf}' (\text{gluing “ } \{(x,y,l)\})$
(is *?gt = tf' ?g*)
proof –
have *in-aff: (x,y) ∈ e'-aff τ(x,y) ∈ e'-aff*
using *assms e-proj-aff by simp+*

consider (1) $?gt = \{(\tau(x,y),l)\}$ | (2) $?gt = \{(\tau(x,y),l), ((x,y), \text{Not } l)\}$
using *tau-idemp-point gluing-cases-points[OF assms(2), of τ(x,y) l] by presburger*
then show *?thesis*
proof(*cases*)
case 1
then have *zeros: x = 0 ∨ 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:  $x \neq 0 \ y \neq 0$ 
    using e-proj-elim-2 in-aff assms gluing-class-1 by auto
  then have gl-eq: gluing “  $\{(x,y),l\} = \{(x,y),l,(\tau(x,y), \text{Not } l)\}$  ”
    using in-aff gluing-class-2 by auto
  then show ?thesis
    by(simp add: 2 gl-eq tf'-def del:  $\tau.simps,fast$ )
qed
qed

```

lemma remove-add-rho:

```

  assumes  $p \in e\text{-proj} \ q \in e\text{-proj}$ 
  shows proj-addition (tf  $\varrho$  p)  $q = \text{tf } \varrho$  (proj-addition p q)
proof -

```

```

  obtain  $x \ y \ l \ x' \ y' \ l'$  where
    p-q-expr:  $p = \text{gluing } \{(x, y), l\}$ 
               $q = \text{gluing } \{(x', y'), l'\}$ 
    using assms
    unfolding e-proj-def
    apply(elim quotientE)
    by force

```

```

  have e-proj:
    gluing “  $\{(x, y), l\} \in e\text{-proj}$  ”
    gluing “  $\{(x', y'), l'\} \in e\text{-proj}$  ”
    using p-q-expr assms by auto
  then have rho-e-proj:
    gluing “  $\{\varrho(x, y), l\} \in e\text{-proj}$  ”
    using remove-rho rho-preserv-e-proj by auto

```

```

  have in-aff:  $(x,y) \in e'\text{-aff} \ (x',y') \in e'\text{-aff}$ 
    using assms p-q-expr e-proj-aff by auto

```

consider

- (a) $(x, y) \in e\text{-circ} \wedge (\exists g \in \text{symmetries}. (x', y') = (g \circ i)(x, y)) \mid$
- (b) $((x, y), x', y') \in e'\text{-aff-0} \wedge \neg ((x, y) \in e\text{-circ} \wedge$
 $(\exists g \in \text{symmetries}. (x', y') = (g \circ i)(x, y))) \mid$
- (c) $((x, y), x', y') \in e'\text{-aff-1} \wedge \neg ((x, y) \in e\text{-circ} \wedge$
 $(\exists g \in \text{symmetries}. (x', y') = (g \circ i)(x, y))) \mid ((x, y), x', y') \notin e'\text{-aff-0}$

```

  using dichotomy-1[OF  $\langle(x,y) \in e'\text{-aff}\rangle \langle(x',y') \in e'\text{-aff}\rangle$ ] by argo
  then show ?thesis

```

proof(cases)

```

  case a
  then have e-circ:  $(x,y) \in e\text{-circ}$  by auto
  then have zeros:  $x \neq 0 \ y \neq 0$  unfolding e-circ-def by auto

```

from a **obtain** g **where** g -*expr*:
 $g \in \text{symmetries } (x', y') = (g \circ i) (x, y)$ **by** *blast*
then obtain r **where** r -*expr*: $(x', y') = (\tau \circ r \circ i) (x, y)$ $r \in \text{rotations}$
using *sym-decomp* **by** *blast*
have ds : $\text{delta } x \ y \ x' \ y' = 0$ $\text{delta}' \ x \ y \ x' \ y' = 0$
using *wd-d-nz*[*OF g-expr e-circ*] *wd-d'-nz*[*OF g-expr e-circ*] **by** *auto*

have ren : $\tau (x', y') = (r \circ i) (x, y)$
using r -*expr* *tau-idemp-point* **by** *auto*

have ds'' : $\text{delta } x \ y \ (\text{fst } ((r \circ i) (x, y))) \ (\text{snd } ((r \circ i) (x, y))) \neq 0 \vee$
 $\text{delta}' \ x \ y \ (\text{fst } ((r \circ i) (x, y))) \ (\text{snd } ((r \circ i) (x, y))) \neq 0$
(is $?ds1 \neq 0 \vee ?ds2 \neq 0$ **)**
using ren *covering-with-deltas* ds *e-proj* **by** *fastforce*

have ds''' : $\text{delta } (\text{fst } (\varrho (x, y))) \ (\text{snd } (\varrho (x, y))) \ (\text{fst } ((r \circ i) (x, y))) \ (\text{snd } ((r \circ$
 $i) (x, y))) \neq 0 \vee$
 $\text{delta}' \ (\text{fst } (\varrho (x, y))) \ (\text{snd } (\varrho (x, y))) \ (\text{fst } ((r \circ i) (x, y))) \ (\text{snd } ((r \circ$
 $i) (x, y))) \neq 0$
(is $?ds3 \neq 0 \vee ?ds4 \neq 0$ **)**
using ds'' *rotation-invariance-5* *rotation-invariance-6* **by** *force*

have ds : $?ds3 \neq 0 \implies \text{delta } x \ y \ x \ (-y) \neq 0$
 $?ds4 \neq 0 \implies \text{delta}' \ x \ y \ x \ (-y) \neq 0$
 $?ds1 \neq 0 \implies \text{delta } x \ y \ x \ (-y) \neq 0$
 $?ds2 \neq 0 \implies \text{delta}' \ x \ y \ x \ (-y) \neq 0$
using ds''' 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*: *field-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*: *field-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*: *field-simps t-nz zeros*)
using r -*expr* **unfolding** *rotations-def*
apply(*simp* *add*: *zeros two-not-zero*)
apply(*elim* *disjE*, *safe*)
by(*simp*-*all* *add*: *field-simps t-nz zeros*)

```

have eq: gluing “  $\{((\tau \circ r \circ i) (x, y), l')\} =$ 
      gluing “  $\{((r \circ i) (x, y), \text{Not } l')\}$ 
  apply (subst gluing-inv[of fst ((r \circ i) (x, y)) snd ((r \circ i) (x, y)) Not l',
        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 “  $\{(\varrho (x, y), l)\} \in e\text{-proj}$ 
      gluing “  $\{((r \circ i) (x, y), \text{Not } l')\} \in e\text{-proj}$ 
  using e-proj(1,2) eq r-expr(1) insert-rho-gluing rho-preserv-e-proj by auto
{
  assume True: delta x y x (-y)  $\neq 0$ 
  have 1: add ( $\varrho (x, y)$ ) ((r \circ i) (x, y)) = ( $\varrho \circ r$ ) (1,0)
    (is ?lhs = ?rhs)
  proof -
    have ?lhs =  $\varrho$  (add (x, y) (r (i (x, y))))
      using rho-invariance-1-points o-apply[of r i] by presburger
    also have ... = ( $\varrho \circ r$ ) (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)  $\neq 0$ 
  have 2: ext-add ( $\varrho (x, y)$ ) ((r \circ i) (x, y)) = ( $\varrho \circ r$ ) (1,0)
    (is ?lhs = ?rhs)
  proof -
    have ?lhs =  $\varrho$  (ext-add (x, y) (r (i (x, y))))
      using rho-invariance-2-points o-apply[of r i] by presburger
    also have ... = ( $\varrho \circ r$ ) (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 “  $\{(\varrho (x, y), l)\}$ 
      (gluing “  $\{((r \circ i) (x, y), \text{Not } l')\} =$ 
      gluing “  $\{((\varrho \circ r) (1,0), \text{Not } (\text{xor } l l'))\}$ 
      (is proj-addition ?g1 ?g2 = ?g3))
proof (cases ?ds3  $\neq 0$ )

```



```

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)), Not (xor l l'))}
  apply(subst gluing-add[of fst (ρ (x, y)) snd (ρ (x, y)) l
    fst ((r ∘ i) (x, y)) snd ((r ∘ i) (x, y)) Not l',
    simplified prod.collapse, OF e-proj'] )
  apply(rule True)
  apply(subst xor-def)+
  apply(rule gluing-eq)
  by blast
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)), Not (xor l l'))}
  apply(subst gluing-ext-add[of fst (ρ (x, y)) snd (ρ (x, y)) l
    fst ((r ∘ i) (x, y)) snd ((r ∘ i) (x, y)) Not l',
    simplified prod.collapse, OF e-proj'] )
  using False ds''' apply linarith
  apply(rule gluing-eq)
  by(auto simp add: xor-def)
  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), Not l')} ∈ e-proj
  using e-proj eq r-expr(1) by auto
have simp2: tf ρ
  (proj-addition (gluing “ {((x, y), l)} (gluing “ {((r ∘ i) (x, y), Not l')})) =
  gluing “ {((ρ ∘ r) (1,0), Not (xor l l'))}
  (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 aux: delta x y x y ≠ 0
  using delta-def delta-minus-def delta-plus-def by auto
have 1: add (x, y) ((r ∘ i) (x, y)) = r (1,0)
  apply(subst commutativity)
  apply(subst o-apply[of r i])

```

```

apply(subst rotation-invariance-1-points[of r, OF r-expr(2)])
apply(rule arg-cong[of - - r])
apply(subst commutativity)
apply(subst inverse-generalized-points)
  apply (simp add: in-aff(1))
using us-ds aux unfolding delta-plus-def delta-def by auto
have proj-addition ?g1 ?g2 =
  gluing “ {(add (x, y) ((r ∘ i) (x, y)), Not (xor l l'))}
apply(subst gluing-add[of x y l
  fst ((r ∘ i) (x, y)) snd ((r ∘ i) (x, y)) Not l',
  simplified prod.collapse, OF e-proj'])
  apply(rule True)
apply(rule gluing-eq)
by(auto simp add: xor-def)
also have ... = gluing “ {(r (1, 0), Not (xor l l'))}
  using 1 by presburger
finally have eq': proj-addition ?g1 ?g2 = gluing “ {(r (1, 0), Not (xor l l'))}
  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)), Not (xor l l'))}
apply(subst gluing-ext-add-points)
apply(rule e-proj'(1))
  apply(rule e-proj'(2))
using False ds'' apply auto[1]
apply(rule gluing-eq)
by(simp add: xor-def del: ext-add.simps,force)
also have ... = gluing “ {(r (1, 0), Not (xor l l'))}
  using 2 by auto
finally have eq': proj-addition ?g1 ?g2 = gluing “ {(r (1, 0), Not (xor l l'))}
  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'), xor 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'), xor 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')) xor 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'), xor 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  $\varrho$ 
    (proj-addition (gluing “  $\{(x, y), l\}$ ” (gluing “  $\{(x', y'), l'\}$ ”)) =
    gluing “  $\{(ext-add (\varrho (x, y)) (x', y'), xor\ l\ l')\}$ ”
    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')) xor 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  $p \in e\text{-proj}$   $q \in e\text{-proj}$   $r \in rotations$ 
  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)  $r = \varrho$  | (3)  $r = \varrho \circ \varrho$  | (4)  $r = \varrho \circ \varrho \circ \varrho$ 
  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
    apply(simp)
    apply(subst tf-comp[symmetric])
    apply(subst remove-add-rho)
    using assms(1) p-q-expr(1) rho-preserv-e-proj apply force
    apply (simp add: assms(2))
    apply(subst remove-add-rho)
    by(auto simp add: assms tf-comp)
  next
    case 4

```

```

then show ?thesis
  apply(simp)
  apply(subst tf-comp[symmetric])+
  apply(subst remove-add-rho)
  using assms(1) insert-rho-gluing-point p-q-expr(1) rho-preserv-e-proj-point
apply force
  using assms(2) apply auto[1]
  apply(subst remove-add-rho)
  using assms(1) insert-rho-gluing-point p-q-expr(1) rho-preserv-e-proj-point
apply force
  using assms(2) apply auto[1]
  apply(subst remove-add-rho)
  using assms(1) insert-rho-gluing-point p-q-expr(1) rho-preserv-e-proj-point
apply force
  using assms(2) apply auto[1]
  by auto
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), Not l'} ” ∈ 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), Not l} ” ∈ e-proj
  using in-aff e-proj-aff by auto

```

consider

- (a) $(x, y) \in e\text{-circ} \wedge (\exists g \in \text{symmetries}. (x', y') = (g \circ i)(x, y)) \mid$
- (b) $((x, y), x', y') \in e'\text{-aff-0} \neg ((x, y) \in e\text{-circ} \wedge$
 $(\exists g \in \text{symmetries}. (x', y') = (g \circ i)(x, y))) \mid$
- (c) $((x, y), x', y') \in e'\text{-aff-1} \neg ((x, y) \in e\text{-circ} \wedge$

```

    (∃ 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), Not s)} for s
  apply(subst gluing-inv[of fst ((r ∘ i) (x, y)) snd ((r ∘ i) (x, y)) Not s,
    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), Not 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 ... =
    proj-addition (gluing “{((x, y), Not l)}
    (gluing “{(r (i (x, y)), Not l')}))
  using eq by auto
  also have ... =
    tf r (proj-addition (gluing “{((x, y), Not l)}
    (gluing “{(i (x, y), Not l')}))

proof -
  note lem1 = remove-rotations[of fst (i (x,y)) snd (i (x,y)) Not l',
    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), xor l l')}
  (is - = ?rhs)
  apply(subst proj-add-class-inv(1)[OF other-proj, of Not l'])
  apply(rule arg-cong)

```

```

apply(rule tf-cong)
using xor-def by auto

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)), Not l')}”)
using eq by auto
also have ... =
  tf r {(1, 0), xor l l'}
proof -
note lem1 = remove-rotations[of fst (i (x,y)) snd (i (x,y)) Not l',
  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)
apply(rule tf-cong)
using xor-def 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)
  apply(rule gluing-eq)
  using xor-def by auto
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)
  apply(rule gluing-eq)
  using xor-def by auto
qed
qed

lemma remove-add-tau':
  assumes p ∈ e-proj q ∈ e-proj
  shows proj-addition p (tf' q) = tf' (proj-addition p q)
proof -
  obtain r where gluing “ {r} = q
    using assms quotientE unfolding e-proj-def
    by blast
  then have inp: tf' q ∈ e-proj
    using assms(2) tf-preserv-e-proj-point by blast
  show ?thesis
    apply(subst proj-addition-comm)
    apply(simp add: assms(1))
    apply(simp add: inp)
    by (simp add: assms(1) assms(2) proj-addition-comm remove-add-tau)
qed

```


lemma *tf'-idemp*:
assumes $s \in e\text{-proj}$
shows $tf' (tf' s) = s$
proof –
obtain p **where** $p\text{-}q\text{-expr}$:
 $s = \text{gluing } \{p\}$
using *assms quotientE unfolding e-proj-def by blast*
obtain $c\ l$ **where** $1: p = (c,l)$
using *assms surj-pair by fastforce*
obtain $x\ y$ **where** $2: c = (x, y)$
by *fastforce*
have $s = \{(x, y), l\} \vee s = \{(x, y), l, (\tau (x, y), \text{Not } l)\}$
using *assms gluing-cases-explicit 1 2 p-q-expr by presburger*
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 $\{(x, y), l\} \in e\text{-proj}$ $\text{gluing } \{(g (x, y), l)\} \in e\text{-proj}$ $g \in \text{symmetries}$
shows $\text{gluing } \{(g (x, y), l)\} = tf'' (\tau \circ g) (\text{gluing } \{(x, y), l\})$
using *assms remove-tau remove-rotations sym-decomp*
proof –
obtain r **where** $r\text{-expr}$: $r \in \text{rotations}$ $g = \tau \circ r$
using *assms sym-decomp by blast*
then have $e\text{-proj}$: $\text{gluing } \{(r (x, y), l)\} \in e\text{-proj}$
using *rotation-preserv-e-proj insert-rotation-gluing assms by simp*
have $\text{gluing } \{(g (x, y), l)\} = \text{gluing } \{(\tau (r (x, y)), l)\}$
using $r\text{-expr}$ **by** *simp*
also have $\dots = tf' (\text{gluing } \{(r (x, y), l)\})$
using *remove-tau assms e-proj r-expr*
by (*metis calculation prod.collapse*)
also have $\dots = tf' (tf\ r\ (\text{gluing } \{(x, y), l\}))$
using *remove-rotations r-expr assms(1) by force*
also have $\dots = tf'' (\tau \circ g) (\text{gluing } \{(x, y), l\})$
using $r\text{-expr}(2)$ *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 $p \in e\text{-proj}$ $q \in e\text{-proj}$ $g \in \text{rotations}$
shows $\text{proj-addition } (tf''\ g\ p)\ q = tf''\ g\ (\text{proj-addition } p\ q)$
proof –
obtain $x\ y\ l\ x'\ y'\ l'$ **where** $p\text{-}q\text{-expr}$: $p = \text{gluing } \{(x, y), l\}$ $q = \text{gluing } \{(x', y'), l'\}$

```

{((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 fstI: x = (y, z) ⇒ y = fst x
  by simp

```

```

lemma sndI: x = (y, z) ⇒ z = snd x
  by simp

```

```

ML <
fun basic-equalities assms ctxt z1' z3' =
let
  (* Basic equalities *)

  val th1 = @{thm fstI} OF [(nth assms 0)]
  val th2 = Thm.instantiate' [SOME @{ctyp 'a}]
    [SOME @{cterm fst::'a×'a ⇒ 'a}]
    (@{thm arg-cong} OF [(nth assms 2)])
  val x1'-expr = Goal.prove ctxt [] [] (HOLogic.mk-Trueprop
    (HOLogic.mk-eq (@{term x1'::'a},HOLogic.mk-fst z1')))
    (fn - =>
      EqSubst.eqsubst-tac ctxt [1] [th1] 1
      THEN EqSubst.eqsubst-tac ctxt [1] [th2] 1
      THEN simp-tac ctxt 1)

  val th3 = @{thm sndI} OF [(nth assms 0)]
  val th4 = Thm.instantiate' [SOME @{ctyp 'a}]
    [SOME @{cterm snd::'a×'a ⇒ 'a}]
    (@{thm arg-cong} OF [(nth assms 2)])
  val y1'-expr = Goal.prove ctxt [] []
    (HOLogic.mk-Trueprop (HOLogic.mk-eq (@{term
y1'::'a},HOLogic.mk-snd z1')))
    (fn - => EqSubst.eqsubst-tac ctxt [1] [th3] 1
      THEN EqSubst.eqsubst-tac ctxt [1] [th4] 1
      THEN simp-tac ctxt 1)

  val th5 = @{thm fstI} OF [(nth assms 1)]
  val th6 = Thm.instantiate' [SOME @{ctyp 'a}]
    [SOME @{cterm fst::'a×'a ⇒ 'a}]
    (@{thm arg-cong} OF [(nth assms 3)])
  val x3'-expr = Goal.prove ctxt [] []
    (HOLogic.mk-Trueprop (HOLogic.mk-eq (@{term
x3'::'a},HOLogic.mk-fst z3')))
    (fn - => EqSubst.eqsubst-tac ctxt [1] [th5] 1
      THEN EqSubst.eqsubst-tac ctxt [1] [th6] 1
      THEN simp-tac ctxt 1)

  val th7 = @{thm sndI} OF [(nth assms 1)]
  val th8 = Thm.instantiate' [SOME @{ctyp 'a}]
    [SOME @{cterm snd::'a×'a ⇒ 'a}]
    (@{thm arg-cong} OF [(nth assms 3)])
  val y3'-expr = Goal.prove ctxt [] []
    (HOLogic.mk-Trueprop (HOLogic.mk-eq (@{term
y3'::'a},HOLogic.mk-snd z3')))
    (fn - => EqSubst.eqsubst-tac ctxt [1] [th7] 1
      THEN EqSubst.eqsubst-tac ctxt [1] [th8] 1
      THEN simp-tac ctxt 1)

in
  (x1'-expr,y1'-expr,x3'-expr,y3'-expr)

```

```

end

fun rewrite-procedures ctxt =
let
  val rewrite1 =
    let
      val pat = [Rewrite.In, Rewrite.Term
                 (@{const divide('a')} $ Var ((c, 0), typ <'a>) $ Rewrite.mk-hole 1
                 (typ <'a>), []),
                 Rewrite.At]
      val to = NONE
    in
      CCONVERSION (Rewrite.rewrite-conv ctxt (pat, to) @ {thms delta-x-def[symmetric]
delta-y-def[symmetric]
delta-minus-def[symmetric]
delta-plus-def[symmetric]}) 1
    end

  val rewrite2 =
    let
      val pat = [Rewrite.In, Rewrite.Term
                 (@{const divide('a')} $ Var ((c, 0), typ <'a>) $ Rewrite.mk-hole 1
                 (typ <'a>), []),
                 Rewrite.In]
      val to = NONE
    in
      CCONVERSION (Rewrite.rewrite-conv ctxt (pat, to) @ {thms delta-x-def[symmetric]
delta-y-def[symmetric]
delta-minus-def[symmetric]
delta-plus-def[symmetric]
}) 1
    end;

  val rewrite3 =
    let
      val pat = [Rewrite.In, Rewrite.Term (@{const divide('a')} $ Var ((c, 0),
typ <'a>) $
                 Rewrite.mk-hole 1 (typ <'a>), []), Rewrite.At]
      val to = NONE
    in
      CCONVERSION (Rewrite.rewrite-conv ctxt (pat, to) @ {thms delta-x-def[symmetric]
delta-y-def[symmetric]
delta-minus-def[symmetric]
delta-plus-def[symmetric]}) 1
    end

  val rewrite4 =
    let
      val pat = [Rewrite.In, Rewrite.Term (@{const divide('a')} $ Var ((c, 0),

```

```

typ <'a> $
                                Rewrite.mk-hole 1 (typ <'a>), [], Rewrite.In]
    val to = NONE
    in
      CCONVERSION (Rewrite.rewrite-conv ctxt (pat, to) @ { thms delta-x-def[symmetric]
delta-y-def[symmetric]
                                delta-minus-def[symmetric]
delta-plus-def[symmetric] }) 1
    end
  in
    (rewrite1, rewrite2, rewrite3, rewrite4)
  end

```

```

fun concrete-assoc first second third fourth =
let

```

```

    val ctxt0 = @ { context };
    val ctxt = ctxt0;
    val (-, ctxt) = Variable.add-fixes [z1', x1', y1',
                                        z3', x3', y3',
                                        x1, y1, x2, y2, x3, y3] ctxt

```

```

    val z1' = if first = ext then @ { term ext-add (x1, y1) (x2, y2) } else @ { term add
(x1, y1) (x2, y2) }
    val z3' = if fourth = ext then @ { term ext-add (x2, y2) (x3, y3) } else @ { term add
(x2, y2) (x3, y3) }
    val lhs = if second = ext then (fn z1' => @ { term ext-add } $ z1' $ @ { term
(x3::'a, y3::'a) })
              else (fn z1' => @ { term add } $ z1' $ @ { term (x3::'a, y3::'a) })
    val rhs = if third = ext then (fn z3' => @ { term ext-add (x1, y1) } $ z3')
              else (fn z3' => @ { term add (x1, y1) } $ z3')

```

```

    val delta1 = case z1' of @ { term ext-add } $ - $ - => [ @ { term delta' x1 y1 x2
y2 }, @ { term delta-x x1 y1 x2 y2 }, @ { term delta-y x1 y1 x2 y2 } ]
                  | @ { term add } $ - $ - => [ @ { term delta x1 y1 x2 y2 }, @ { term
delta-minus x1 y1 x2 y2 }, @ { term delta-plus x1 y1 x2 y2 } ]
                  | - => []
    val delta2 = case (lhs @ { term z1'::'a × 'a }) of
                  @ { term ext-add } $ - $ - => [ @ { term delta-x x1' y1' x3
y3 }, @ { term delta-y x1' y1' x3 y3 } ]
                  | @ { term add } $ - $ - => [ @ { term delta-minus x1' y1' x3
y3 }, @ { term delta-plus x1' y1' x3 y3 } ]
                  | - => []
    val delta3 = if third = ext then [ @ { term delta-x x1 y1 x3' y3' }, @ { term delta-y
x1 y1 x3' y3' } ]
                  else [ @ { term delta-minus x1 y1 x3' y3' }, @ { term delta-plus
x1 y1 x3' y3' } ]

```

```

    val delta4 = if fourth = ext then [@{term delta' x2 y2 x3 y3},@{term delta-x x2
y2 x3 y3},@{term delta-y x2 y2 x3 y3}]
                    else [@{term delta x2 y2 x3 y3},@{term delta-minus x2
y2 x3 y3},@{term delta-plus x2 y2 x3 y3}]

    val assms3 = Thm.ctrm-of ctxt (HOLogic.mk-Trueprop (HOLogic.mk-eq(@{term
z1'::'a × 'a},z1')))
    val assms4 = Thm.ctrm-of ctxt (HOLogic.mk-Trueprop (HOLogic.mk-eq(@{term
z3'::'a × 'a},z3')))
    val assms5 = Thm.ctrm-of ctxt (HOLogic.mk-Trueprop (HOLogic.mk-not (HOLogic.mk-eq
(nth delta1 1,@{term 0::'a}))))
    val assms6 = Thm.ctrm-of ctxt (HOLogic.mk-Trueprop (HOLogic.mk-not (HOLogic.mk-eq
(nth delta1 2,@{term 0::'a}))))
    val assms7 = Thm.ctrm-of ctxt (HOLogic.mk-Trueprop (HOLogic.mk-not (HOLogic.mk-eq
(nth delta4 1,@{term 0::'a}))))
    val assms8 = Thm.ctrm-of ctxt (HOLogic.mk-Trueprop (HOLogic.mk-not (HOLogic.mk-eq
(nth delta4 2,@{term 0::'a}))))
    val assms9 = Thm.ctrm-of ctxt (HOLogic.mk-Trueprop (HOLogic.mk-not (HOLogic.mk-eq
(nth delta2 0,@{term 0::'a}))))
    val assms10 = Thm.ctrm-of ctxt (HOLogic.mk-Trueprop (HOLogic.mk-not (HOLogic.mk-eq
(nth delta2 1,@{term 0::'a}))))
    val assms11 = Thm.ctrm-of ctxt (HOLogic.mk-Trueprop (HOLogic.mk-not (HOLogic.mk-eq
(nth delta3 0,@{term 0::'a}))))
    val assms12 = Thm.ctrm-of ctxt (HOLogic.mk-Trueprop (HOLogic.mk-not (HOLogic.mk-eq
(nth delta3 1,@{term 0::'a}))))

    val (assms,ctxt) = Assumption.add-assumes
      [@{cprop z1' = (x1'::'a,y1'::'a)}, @{cprop z3' = (x3'::'a,y3'::'a)},
       assms3,assms4,assms5,assms6,assms7, assms8,assms9,
       assms10,assms11, assms12,
       @{cprop e' x1 y1 = 0}, @{cprop e' x2 y2 = 0}, @{cprop e'
x3 y3 = 0}
      ] ctxt;

    val normalizex = List.foldl (HOLogic.mk-binop Groups.times-class.times) @{term
1::'a} [nth delta2 0, nth delta3 0, nth delta1 0, nth delta4 0]
    val normalizey = List.foldl (HOLogic.mk-binop Groups.times-class.times) @{term
1::'a} [nth delta2 1, nth delta3 1, nth delta1 0, nth delta4 0]

    val fstminus = HOLogic.mk-binop Groups.minus-class.minus
      (HOLogic.mk-fst (lhs @{term z1'::'a × 'a}), HOLogic.mk-fst (rhs
@{term z3'::'a × 'a}))
    val sndminus = HOLogic.mk-binop Groups.minus-class.minus
      (HOLogic.mk-snd (lhs @{term z1'::'a × 'a}), HOLogic.mk-snd (rhs
@{term z3'::'a × 'a}))

    val goal = HOLogic.mk-Trueprop(HOLogic.mk-eq(lhs z1',rhs z3'))

    val gxDeltax =

```

```

HOLogic.mk-Trueprop(
  HOLogic.mk-exists (r1,@{typ 'a}),
  HOLogic.mk-exists(r2,@{typ 'a}),
  HOLogic.mk-exists(r3,@{typ 'a}),
  HOLogic.mk-eq(HOLogic.mk-binop Groups.times-class.times (fstminus,normalizer),
    @{term r1 * e' x1 y1 + r2 * e' x2 y2 + r3 * e' x3 y3}))))

val gyDeltay =
  HOLogic.mk-Trueprop(
    HOLogic.mk-exists (r1,@{typ 'a}),
    HOLogic.mk-exists(r2,@{typ 'a}),
    HOLogic.mk-exists(r3,@{typ 'a}),
    HOLogic.mk-eq(HOLogic.mk-binop Groups.times-class.times (sndminus,normalize),
      @{term r1 * e' x1 y1 + r2 * e' x2 y2 + r3 * e' x3 y3}))))

val (x1'-expr,y1'-expr,x3'-expr,y3'-expr) = basic-equalities assms ctxt z1' z3'
val (rewrite1,rewrite2,rewrite3,rewrite4) = rewrite-procedures ctxt

(* First subgoal *)
val div1 = Goal.prove ctxt [] [] gxDeltax
  (fn - => asm-full-simp-tac (ctxt addsimps [nth assms 0,nth assms 1]) 1
    THEN REPEAT rewrite1
    THEN asm-full-simp-tac (ctxt
      addsimps (@{thms divide-simps} @ [nth assms 8, nth assms 10])))
1
  THEN REPEAT (EqSubst.eqsubst-tac ctxt [0]
    (@{thms left-diff-distrib delta-x-def delta-y-def delta-minus-def
delta-plus-def} @ [x1'-expr,y1'-expr,x3'-expr,y3'-expr]) 1)
  THEN simp-tac ctxt 1
  THEN REPEAT rewrite2
  THEN asm-full-simp-tac (ctxt
    addsimps (@{thms divide-simps} @ map (nth assms) [4,5,6,7] @
      [@{thm delta'-def}, @{thm delta-def}])) 1
  THEN asm-full-simp-tac (ctxt addsimps
    [@{thm c-eq-1},@{thm t-expr(1)},@{thm delta-x-def},
    @{thm delta-y-def}, @{thm delta-plus-def},
    @{thm delta-minus-def}, @{thm e'-def}]) 1
  THEN Groebner.algebra-tac [] [] ctxt 1
)

val goal1 = HOLogic.mk-Trueprop (HOLogic.mk-eq (fstminus, @{term 0::'a}))

val eq1 = Goal.prove ctxt [] [] goal1
  (fn - => Method.insert-tac ctxt [div1] 1
    THEN asm-full-simp-tac (ctxt addsimps
      (map (nth assms) [4,5,6,7,8,10,12,13,14]) @ @{thms
delta'-def delta-def}) 1 )

```

```

val div2 = Goal.prove ctxt [] [] gyDeltay
  (fn - => asm-full-simp-tac (@{context} addsimps [nth assms 0,nth assms 1]) 1
    THEN REPEAT rewrite3
      THEN asm-full-simp-tac (@{context} addsimps (@{thms divide-simps}
@ [nth assms 9,nth assms 11])) 1
      THEN REPEAT (EqSubst.eqsubst-tac ctxt [0] (@{thms left-diff-distrib
delta-x-def delta-y-def delta-minus-def delta-plus-def} @ [x1'-expr,y1'-expr,x3'-expr,y3'-expr]
1)
        THEN simp-tac @{context} 1
          THEN REPEAT rewrite4
            THEN asm-full-simp-tac (@{context} addsimps (@{thms divide-simps
delta'-def delta-def} @ (map (nth assms) [4,5,6,7]))) 1
              THEN asm-full-simp-tac (@{context} addsimps
                [@{thm c-eq-1},@{thm t-expr(1)},@{thm delta-x-def},
                  @{thm delta-y-def}, @{thm delta-plus-def},
                  @{thm delta-minus-def}, @{thm e'-def}]) 1
                THEN Groebner.algebra-tac [] [] ctxt 1
              )
        )
    )
val goal2 = HOLogic.mk-Trueprop (HOLogic.mk-eq (sndminus, @ {term 0::'a}))

val eq2 = Goal.prove ctxt [] [] goal2
  (fn - => Method.insert-tac ctxt [div2] 1
    THEN asm-full-simp-tac (ctxt addsimps
      (map (nth assms) [4,5,6,7,9,11,12,13,14]) @ @ {thms
delta'-def delta-def}) 1 );

val goal = Goal.prove ctxt [] [] goal
  (fn - => Method.insert-tac ctxt ([eq1,eq2] @ [nth assms 2,nth assms
3]) 1
    THEN asm-full-simp-tac ctxt 1 );

in
  singleton (Proof-Context.export ctxt ctxt0) goal
end

>

local-setup <
  Local-Theory.note ((@{binding ext-ext-ext-ext-assoc}, []), [concrete-assoc ext ext
ext ext]) #> snd
>

local-setup <
  Local-Theory.note ((@{binding ext-add-ext-ext-assoc}, []), [concrete-assoc add ext
ext ext]) #> snd
>

```


local-setup <

Local-Theory.note ((@{binding ext-ext-ext-add-assoc}, []), [concrete-assoc ext ext ext add]) #> snd
>

local-setup <

Local-Theory.note ((@{binding add-ext-add-ext-assoc}, []), [concrete-assoc ext add add ext]) #> snd
>

lemma *add-ext-add-ext-assoc-points*:

assumes $(x1, y1) \in e'\text{-aff } (x2, y2) \in e'\text{-aff } (x3, y3) \in e'\text{-aff}$
assumes $\text{delta}' x1 y1 x2 y2 \neq 0 \text{ delta}' x2 y2 x3 y3 \neq 0$
 $\text{delta } (\text{fst } (\text{ext-add } (x1, y1) (x2, y2))) (\text{snd } (\text{ext-add } (x1, y1) (x2, y2))) x3$
 $y3 \neq 0$
 $\text{delta } x1 y1 (\text{fst } (\text{ext-add } (x2, y2) (x3, y3))) (\text{snd } (\text{ext-add } (x2, y2) (x3, y3)))$
 $\neq 0$
shows $\text{add } (\text{ext-add } (x1, y1) (x2, y2)) (x3, y3) = \text{add } (x1, y1) (\text{ext-add } (x2, y2)$
 $(x3, y3))$
apply(rule *add-ext-add-ext-assoc*)
apply *simp+*
using *assms(4,5,6,7) delta-def delta'-def* **apply** *force+*
using *assms(1,2,3) unfolding e'-aff-def* **by** *blast+*

local-setup <

Local-Theory.note ((@{binding add-ext-ext-ext-assoc}, []), [concrete-assoc ext add ext ext]) #> snd
>

local-setup <

Local-Theory.note ((@{binding add-ext-add-add-assoc}, []), [concrete-assoc ext add add add]) #> snd
>

lemma *add-ext-add-add-assoc-points*:

assumes $(x1, y1) \in e'\text{-aff } (x2, y2) \in e'\text{-aff } (x3, y3) \in e'\text{-aff}$
assumes $\text{delta}' x1 y1 x2 y2 \neq 0 \text{ delta } x2 y2 x3 y3 \neq 0$
 $\text{delta } (\text{fst } (\text{ext-add } (x1, y1) (x2, y2))) (\text{snd } (\text{ext-add } (x1, y1) (x2, y2))) x3$
 $y3 \neq 0$
 $\text{delta } x1 y1 (\text{fst } (\text{add } (x2, y2) (x3, y3))) (\text{snd } (\text{add } (x2, y2) (x3, y3))) \neq 0$
shows $\text{add } (\text{ext-add } (x1, y1) (x2, y2)) (x3, y3) = \text{add } (x1, y1) (\text{add } (x2, y2)$
 $(x3, y3))$
apply(rule *add-ext-add-add-assoc*)
apply *simp+*
using *assms(4,5,6,7) delta-def delta'-def* **apply** *force+*
using *assms(1,2,3) unfolding e'-aff-def* **by** *blast+*

local-setup <

Local-Theory.note ((@{binding add-add-ext-add-assoc}, []), [concrete-assoc add

add ext add) #> *snd*
>

lemma *add-add-ext-add-assoc-points*:

assumes $(x1,y1) \in e'\text{-aff}$ $(x2,y2) \in e'\text{-aff}$ $(x3,y3) \in e'\text{-aff}$
assumes $\text{delta } x1 \ y1 \ x2 \ y2 \neq 0$ $\text{delta } x2 \ y2 \ x3 \ y3 \neq 0$
 $\text{delta } (\text{fst } (\text{add } (x1,y1) \ (x2,y2))) \ (\text{snd } (\text{add } (x1,y1) \ (x2,y2))) \ x3 \ y3 \neq 0$
 $\text{delta}' \ x1 \ y1 \ (\text{fst } (\text{add } (x2,y2) \ (x3,y3))) \ (\text{snd } (\text{add } (x2,y2) \ (x3,y3))) \neq 0$
shows $\text{add } (\text{add } (x1,y1) \ (x2,y2)) \ (x3,y3) = \text{ext-add } (x1,y1) \ (\text{add } (x2,y2) \ (x3,y3))$
apply(*rule add-add-ext-add-assoc*)
apply *simp+*
using *assms(4,5,6,7) delta-def delta'-def apply force+*
using *assms(1,2,3) unfolding e'-aff-def by blast+*

local-setup <

Local-Theory.note ((@{*binding add-add-ext-ext-assoc*}, []), [*concrete-assoc add add ext ext*]) #> *snd*
>

local-setup <

Local-Theory.note ((@{*binding add-add-add-ext-assoc*}, []), [*concrete-assoc add add add ext*]) #> *snd*
>

lemma *add-add-add-ext-assoc-points*:

assumes $(x1,y1) \in e'\text{-aff}$ $(x2,y2) \in e'\text{-aff}$ $(x3,y3) \in e'\text{-aff}$
assumes $\text{delta } x1 \ y1 \ x2 \ y2 \neq 0$ $\text{delta}' \ x2 \ y2 \ x3 \ y3 \neq 0$
 $\text{delta } (\text{fst } (\text{add } (x1,y1) \ (x2,y2))) \ (\text{snd } (\text{add } (x1,y1) \ (x2,y2))) \ x3 \ y3 \neq 0$
 $\text{delta } x1 \ y1 \ (\text{fst } (\text{ext-add } (x2,y2) \ (x3,y3))) \ (\text{snd } (\text{ext-add } (x2,y2) \ (x3,y3)))$
 $\neq 0$
shows $\text{add } (\text{add } (x1,y1) \ (x2,y2)) \ (x3,y3) = \text{add } (x1,y1) \ (\text{ext-add } (x2,y2) \ (x3,y3))$
apply(*rule add-add-add-ext-assoc*)
apply *simp+*
using *assms(4,5,6,7) delta-def delta'-def apply force+*
using *assms(1,2,3) unfolding e'-aff-def by blast+*

local-setup <

Local-Theory.note ((@{*binding ext-add-add-ext-assoc*}, []), [*concrete-assoc add ext add ext*]) #> *snd*
>

lemma *ext-add-add-ext-assoc-points*:

assumes $(x1,y1) \in e'\text{-aff}$ $(x2,y2) \in e'\text{-aff}$ $(x3,y3) \in e'\text{-aff}$
assumes $\text{delta } x1 \ y1 \ x2 \ y2 \neq 0$ $\text{delta}' \ x2 \ y2 \ x3 \ y3 \neq 0$
 $\text{delta}' \ (\text{fst } (\text{add } (x1,y1) \ (x2,y2))) \ (\text{snd } (\text{add } (x1,y1) \ (x2,y2))) \ x3 \ y3 \neq 0$
 $\text{delta } x1 \ y1 \ (\text{fst } (\text{ext-add } (x2,y2) \ (x3,y3))) \ (\text{snd } (\text{ext-add } (x2,y2) \ (x3,y3)))$
 $\neq 0$

shows $ext\text{-}add (add (x1,y1) (x2,y2)) (x3,y3) = add (x1,y1) (ext\text{-}add (x2,y2) (x3,y3))$
apply(rule *ext-add-add-ext-assoc*)
apply *simp+*
using *assms(4,5,6,7) delta-def delta'-def* **apply** *force+*
using *assms(1,2,3) unfolding e'-aff-def* **by** *blast+*

local-setup <

Local-Theory.note ((@{binding *ext-add-add-add-assoc*}, []), [*concrete-assoc add ext add add*]) #> *snd*
 >

lemma *ext-add-add-add-assoc-points*:

assumes $(x1,y1) \in e'\text{-}aff (x2,y2) \in e'\text{-}aff (x3,y3) \in e'\text{-}aff$
assumes $delta\ x1\ y1\ x2\ y2 \neq 0\ delta\ x2\ y2\ x3\ y3 \neq 0$
 $delta' (fst (add (x1,y1) (x2,y2))) (snd (add (x1,y1) (x2,y2)))\ x3\ y3 \neq 0$
 $delta\ x1\ y1\ (fst (add (x2,y2) (x3,y3))) (snd (add (x2,y2) (x3,y3))) \neq 0$
shows $ext\text{-}add (add (x1,y1) (x2,y2)) (x3,y3) = add (x1,y1) (add (x2,y2) (x3,y3))$
apply(rule *ext-add-add-add-assoc*)
apply *simp+*
using *assms(4,5,6,7) delta-def delta'-def* **apply** *force+*
using *assms(1,2,3) unfolding e'-aff-def* **by** *blast+*

local-setup <

Local-Theory.note ((@{binding *ext-add-ext-add-assoc*}, []), [*concrete-assoc add ext ext add*]) #> *snd*
 >

local-setup <

Local-Theory.note ((@{binding *ext-ext-add-add-assoc*}, []), [*concrete-assoc ext ext add add*]) #> *snd*
 >

lemma *ext-ext-add-add-assoc-points*:

assumes $(x1,y1) \in e'\text{-}aff (x2,y2) \in e'\text{-}aff (x3,y3) \in e'\text{-}aff$
assumes $delta'\ x1\ y1\ x2\ y2 \neq 0\ delta\ x2\ y2\ x3\ y3 \neq 0$
 $delta' (fst (ext\text{-}add (x1,y1) (x2,y2))) (snd (ext\text{-}add (x1,y1) (x2,y2)))\ x3\ y3 \neq 0$
 $delta\ x1\ y1\ (fst (add (x2,y2) (x3,y3))) (snd (add (x2,y2) (x3,y3))) \neq 0$
shows $ext\text{-}add (ext\text{-}add (x1,y1) (x2,y2)) (x3,y3) = add (x1,y1) (add (x2,y2) (x3,y3))$
apply(rule *ext-ext-add-add-assoc*)
apply *simp+*
using *assms(4,5,6,7) delta-def delta'-def* **apply** *force+*
using *assms(1,2,3) unfolding e'-aff-def* **by** *blast+*

local-setup <

Local-Theory.note ((@{binding *ext-ext-add-ext-assoc*}, []), [*concrete-assoc ext ext*

```
add ext]) #> snd
>
```

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))
apply(rule ext-ext-add-ext-assoc)
      apply simp+
using assms(4,5,6,7) delta-def delta'-def apply force+
using assms(1,2,3) unfolding e'-aff-def by blast+
```

local-setup <

```
Local-Theory.note ((@{binding add-ext-ext-add-assoc}, []), [concrete-assoc ext add
ext add]) #> snd
>
```

4.3 Lemmas for associativity

lemma *cancellation-assoc*:

```
assumes gluing “ {(x1,y1), False} ∈ e-proj
      gluing “ {(x2,y2), False} ∈ e-proj
      gluing “ {(i (x2,y2), False)} ∈ e-proj
shows proj-addition (proj-addition (gluing “ {(x1,y1), False}))
      (gluing “ {(x2,y2), False})) (gluing “ {(i (x2,y2),
False)}) =
      gluing “ {(x1,y1), False}
(is proj-addition (proj-addition ?g1 ?g2) ?g3 = ?g1)
proof –
have in-aff: (x1,y1) ∈ e'-aff (x2,y2) ∈ e'-aff i (x2,y2) ∈ e'-aff
      using assms(1,2,3) e-proj-aff by auto

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

have e-proj: gluing “ {(x1, y1), False} ∈ e-proj
      gluing “ {(x2, y2), False} ∈ e-proj
      gluing “ {(i (x1, y1), False)} ∈ e-proj
      {(1, 0), False} ∈ e-proj
      gluing “ {(i (x2, y2), False)} ∈ 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 ( $\exists g \in \text{symmetries}. (x2, y2) = (g \circ i) (x1, y1)$ )
  then obtain  $g$  where  $g\text{-expr}: g \in \text{symmetries} (x2, y2) = (g \circ i) (x1, y1)$  by
  auto
  then obtain  $g'$  where  $g\text{-expr}': g' \in \text{symmetries} i (x2,y2) = g' (x1, y1) g \circ g'$ 
  =  $id$ 
    using  $\text{symmetries-}i\text{-inverse}[OF\ g\text{-expr}(1),\ \text{of}\ x1\ y1]$ 
     $i\text{-idemp}\ \text{pointfree-idE}$  by force

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

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

  have  $g2\text{-eq}: ?g2 = \text{tf}''\ r\ (gluing\ \{\{(i\ (x1, y1),\ False)\})$ 
  (is - =  $\text{tf}'' - ?g4$ )
    apply( $\text{simp}\ \text{add}: r\text{-expr}\ \text{del}: i.\text{sims}\ o\text{-apply}$ )
    apply( $\text{subst}\ \text{remove-sym}[\text{of}\ \text{fst}\ (i\ (x1,y1))\ \text{snd}\ (i\ (x1,y1))\ False\ \tau \circ r,$ 
     $\text{simplified}\ \text{prod.collapse}]$ ,
    ( $\text{simp}\ \text{add}: e\text{-proj}\ e\text{-proj-comp}\ r\text{-expr}\ \text{del}: i.\text{sims}\ o\text{-apply}$ )+
    using  $e\text{-proj-comp}\ r\text{-expr}\ g\text{-expr}$  apply blast+
    using  $\text{tau-idemp}\ \text{comp-assoc}[\text{of}\ \tau\ \tau\ r,\ \text{symmetric}]$ 
     $id\text{-comp}[\text{of}\ r]$  by presburger
  have  $eq1: \text{proj-addition}\ (\text{proj-addition}\ ?g1\ (\text{tf}''\ r\ ?g4))\ ?g3 = ?g1$ 
    apply( $\text{subst}\ \text{proj-addition-comm}$ )
    using  $e\text{-proj}\ g2\text{-eq}[\text{symmetric}]$  apply( $\text{simp},\ \text{simp}$ )
    apply( $\text{subst}\ \text{remove-add-sym}$ )
    using  $e\text{-proj}\ r\text{-expr}$  apply( $\text{simp},\ \text{simp},\ \text{simp}$ )
    apply( $\text{subst}\ \text{proj-addition-comm}$ )
    using  $e\text{-proj}$  apply( $\text{simp},\ \text{simp}$ )
    apply( $\text{subst}\ \text{proj-add-class-inv}(1)$ )
    using  $e\text{-proj}$  apply  $\text{simp}$ 
    apply( $\text{subst}\ \text{remove-add-sym}$ )
    using  $e\text{-proj}\ r\text{-expr}\ \text{xor-def}$  apply( $\text{simp},\ \text{simp},\ \text{simp}$ )
    apply( $\text{simp}\ \text{add}: \text{xor-def}\ \text{del}: i.\text{sims}$ )
    apply( $\text{subst}\ \text{proj-add-class-identity}$ )
    using  $e\text{-proj}$  apply  $\text{simp}$ 
    apply( $\text{subst}\ \text{remove-sym}[\text{symmetric},\ \text{of}\ \text{fst}\ (i\ (x2,y2))\ \text{snd}\ (i\ (x2,y2))\ False$ 
   $\tau \circ r,$ 
     $\text{simplified}\ \text{prod.collapse}\ \text{comp-assoc}[\text{of}\ \tau\ \tau\ r,\ \text{symmetric}]$   $\text{tau-idemp}$ 
   $id-o]$ )
    using  $e\text{-proj}$  apply  $\text{simp}$ 
    using  $e\text{-proj-comp}(2)\ r\text{-expr}(3)$  apply  $\text{auto}[1]$ 

```

```

    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:  $\tau (i (x2, y2)) \in e'\text{-aff}$ 
      proof -
        have  $i (x2, y2) \in e\text{-circ}$ 
          using e-circ-def in-aff 1 by auto
        then show ?thesis
          using  $\tau\text{-circ circ-to-aff}$  by blast
      qed

    consider
      (a)  $(\exists g \in \text{symmetries. } (x2, y2) = (g \circ i) (x1, y1))$  |
      (b)  $((x1, y1), x2, y2) \in e'\text{-aff-0}$ 
           $\neg ((\exists g \in \text{symmetries. } (x2, y2) = (g \circ i) (x1, y1)))$  |
      (c)  $((x1, y1), x2, y2) \in e'\text{-aff-1}$ 
           $\neg ((\exists g \in \text{symmetries. } (x2, y2) = (g \circ i) (x1, y1))) ((x1, y1), x2, y2) \notin$ 
 $e'\text{-aff-0}$ 
      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:  $\text{delta } x1 \ y1 \ x2 \ y2 \neq 0$ 
        using b(1) unfolding e'-aff-0-def by simp

      have ds:  $\text{delta } x2 \ y2 \ x2 \ (-y2) \neq 0 \vee \text{delta}' \ x2 \ y2 \ (x2) \ (-y2) \neq 0$ 
        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)+

```

```

have eq1: proj-addition ?g1 ?g2 = gluing “ {(add (x1, y1) (x2, y2), False)}
  (is - = ?g-add)
  using gluing-add[OF assms(1,2) pd] xor-def 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), False} ∈ 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)), False)} ∈ e-proj
    gluing “ {(i (rx, ry), False)} ∈ 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
    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
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)+)
  using add-assoc e-proj(5) e-proj-r gluing-add pd' r-expr(3) xor-def by
force
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)+)
    using add-assoc e-proj(5) e-proj-r gluing-ext-add-points pd' r-expr(3)
xor-def 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),
False)}
(is - = ?g-add)
    using gluing-ext-add[OF assms(1,2) pd] xor-def by presburger
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), False} ∈ 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)), False)} ∈ e-proj
    gluing “ {(i (rx, ry), False)} ∈ 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  $(\tau \circ ri' \circ r \circ i) (rx, ry) = i (rx, ry)$ 
  by (metis comp-apply rot-tau-com)
then obtain  $g''$  where  $g''$ -expr:  $g'' \in \text{symmetries } g'' (i ((rx, ry))) = i$ 
(rx,ry)
  using  $\langle ri' \in \text{rotations} \rangle$  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  $pd'$ :  $\text{delta } rx \ ry \ (\text{fst } (\tau (i (x2, y2)))) \ (\text{snd } (\tau (i (x2, y2)))) \neq 0$ 
  unfolding e'-aff-0-def by simp
then have  $pd''$ :  $\text{delta}' \ rx \ ry \ (\text{fst } (i (x2, y2))) \ (\text{snd } (i (x2, y2))) \neq 0$ 
  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  $pd'$ :  $\text{delta}' \ rx \ ry \ (\text{fst } (\tau (i (x2, y2)))) \ (\text{snd } (\tau (i (x2, y2)))) \neq 0$ 
  unfolding e'-aff-0-def e'-aff-1-def by auto
then have  $pd''$ :  $\text{delta} \ rx \ ry \ (\text{fst } (i (x2, y2))) \ (\text{snd } (i (x2, y2))) \neq 0$ 
  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  $pd'$ :  $\text{delta} \ rx \ ry \ (\text{fst } (i (x2, y2))) \ (\text{snd } (i (x2, y2))) \neq 0$ 
  using bb unfolding e'-aff-0-def r-expr by simp
have add-assoc:  $\text{add} (\text{ext-add } (x1, y1) (x2, y2)) (i (x2, y2)) = (x1, y1)$ 
proof(cases delta x2 y2 x2 (-y2)  $\neq 0$ )
  case True
  have inv:  $\text{add} (x2, y2) (i (x2, y2)) = (1, 0)$ 
    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)+
  using add-assoc e-proj(5) e-proj-r gluing-add pd' r-expr(1) r-expr(2)
xor-def 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)+)
    using add-assoc e-proj(5) e-proj-r gluing-ext-add-points pd' r-expr(1)
r-expr(2) xor-def by auto
    qed
    qed
next
case 2

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

        have proj-addition (gluing “  $\{(x1, y1), \text{False}\}$  ”) (gluing “  $\{(x2, y2), \text{False}\}$  ”) =
        proj-addition (tf r (gluing “  $\{(1, 0), \text{False}\}$  ”)) (gluing “  $\{(x2, y2), \text{False}\}$  ”)
        using remove-rotations[OF one-in rot-expr(1)] rot-expr(2) by presburger
        also have ... = tf r (proj-addition (gluing “  $\{(1, 0), \text{False}\}$  ”) (gluing “  $\{(x2, y2), \text{False}\}$  ”))
        using remove-add-rotation assms rot-expr one-in by presburger
        also have ... = tf r (gluing “  $\{(x2, y2), \text{False}\}$  ”)
        using proj-add-class-identity
        by (simp add: e-proj(2) identity-equiv)
        finally have eq1: proj-addition (gluing “  $\{(x1, y1), \text{False}\}$  ”) (gluing “  $\{(x2, y2), \text{False}\}$  ”) =
        tf r (gluing “  $\{(x2, y2), \text{False}\}$  ”) by argo

        have proj-addition (proj-addition (gluing “  $\{(x1, y1), \text{False}\}$  ”) (gluing “  $\{(x2, y2), \text{False}\}$  ”))
        (gluing “  $\{(i (x2, y2), \text{False}\}$  ”) =
        proj-addition (tf r (gluing “  $\{(x2, y2), \text{False}\}$  ”)) (gluing “  $\{(i (x2, y2), \text{False}\}$  ”)
        using eq1 by argo
        also have ... = tf r (proj-addition (gluing “  $\{(x2, y2), \text{False}\}$  ”) (gluing “

```

```

{(i (x2, y2), False)})
  using remove-add-rotation rot-expr well-defined proj-addition-def assms
one-in by simp
  also have ... = tf r (gluing “ {(1, 0), False}”)
  using proj-addition-def proj-add-class-inv assms xor-def
  by (simp add: identity-equiv)
  finally have eq2: proj-addition (proj-addition (gluing “ {(x1, y1), False}”)
(gluing “ {(x2, y2), False}”))
      (gluing “ {(i (x2, y2), False)}”) =
      tf r (gluing “ {(1, 0), False}”) 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 ∈ rotations (x2, y2) = r (1, 0) by blast
then obtain r' where rot-expr': r' ∈ rotations i (x2, y2) = r' (i (1, 0)) r
o r' = id
  using rotations-i-inverse[OF rot-expr(1)]
  by (metis comp-def id-def rot-inv)
  have proj-addition (gluing “ {(x1, y1), False}”) (gluing “ {(x2, y2),
False}”) =
    proj-addition (gluing “ {(x1, y1), False}”) (tf r (gluing “ {(1, 0),
False}”))
  using remove-rotations[OF one-in rot-expr(1)] rot-expr(2) by presburger
  also have ... = tf r (proj-addition (gluing “ {(x1, y1), False}”) (gluing “
{(1, 0), False}”))
  using remove-add-rotation assms rot-expr one-in
  by (metis proj-addition-comm remove-rotations)
  also have ... = tf r (gluing “ {(x1, y1), False}”)
  using proj-add-class-identity assms
  identity-equiv one-in proj-addition-comm by metis
  finally have eq1: proj-addition (gluing “ {(x1, y1), False}”) (gluing “
{(x2, y2), False}”) =
    tf r (gluing “ {(x1, y1), False}”) by argo

  have proj-addition (proj-addition (gluing “ {(x1, y1), False}”) (gluing “
{(x2, y2), False}”))
      (gluing “ {(i (x2, y2), False)}”) =
    proj-addition (tf r (gluing “ {(x1, y1), False}”)) (gluing “ {(i (x2,
y2), False}”)
  using eq1 by argo
  also have ... = tf r (proj-addition (gluing “ {(x1, y1), False}”) (gluing “
{(i (x2, y2), False}”))
  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), False}”) (tf r' (gluing
“ {(i (1, 0), False}”)))
  using remove-rotations one-in rot-expr' by simp

```

also have ... = $tf\ r\ (tf\ r'\ (proj\ addition\ (gluing\ \{\{(x1, y1), False\}\}))$
 $((gluing\ \{\{(i\ (1, 0), False\}\}\}))$
using *proj-add-class-inv* *assms*
by (*metis insert-rotation-gluing-point one-in proj-addition-comm re-*
move-add-rotation rot-expr'(1) rot-expr'(2))
also have ... = $tf\ (id)\ (proj\ addition\ (gluing\ \{\{(x1, y1), False\}\}))\ ((gluing\$
 $\{\{(1, 0), False\}\}))$
using *tf-comp rot-expr'* **by force**
also have ... = $(gluing\ \{\{(x1, y1), False\}\})$
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), False)}*
 $(gluing\ \{\{(x2, y2), False\}\})\ (gluing\ \{\{(i\ (x2,$
 $y2), False\}\}) =$
 $(gluing\ \{\{(x1, y1), False\}\})$ **by blast**
show *?thesis* **by**(*subst eq2, simp*)
qed
qed
qed

lemma *e'-aff-0-invariance:*
 $((x,y),(x',y')) \in e'\text{-aff-0} \implies ((x',y'),(x,y)) \in e'\text{-aff-0}$
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:*
 $((x,y),(x',y')) \in e'\text{-aff-1} \implies ((x',y'),(x,y)) \in e'\text{-aff-1}$
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 $gluing\ \{\{(x1, y1), False\}\} \in e\text{-proj}$
 $gluing\ \{\{(x2, y2), False\}\} \in e\text{-proj}$
 $gluing\ \{\{(x3, y3), False\}\} \in e\text{-proj}$
assumes *a: g ∈ symmetries (x2, y2) = (g ∘ i) (x1, y1)*
shows
 $proj\ addition\ (gluing\ \{\{(x1, y1), False\}\})\ (gluing\ \{\{(x2, y2), False\}\}) =$
 $tf''\ (\tau \circ g)\ \{\{(1,0), False\}\}$ **(is** *proj-addition ?g1 ?g2 = -*
 $proj\ addition\ (proj\ addition\ (gluing\ \{\{(x1, y1), False\}\})\ (gluing\ \{\{(x2, y2),$
 $False\}\}))\ (gluing\ \{\{(x3, y3), False\}\}) =$
 $tf''\ (\tau \circ g)\ (gluing\ \{\{(x3, y3), False\}\})$

$\text{proj-addition } (\text{gluing } \{((x1, y1), \text{False})\}) (\text{proj-addition } (\text{gluing } \{((x2, y2), \text{False})\}) (\text{gluing } \{((x3, y3), \text{False})\})) =$
 $\text{tf'' } (\tau \circ g) (\text{gluing } \{((x3, y3), \text{False})\}) (\text{is proj-addition ?g1 } (\text{proj-addition } ?g2 ?g3) = -)$

proof –

have *in-aff*: $(x1, y1) \in e'\text{-aff } (x2, y2) \in e'\text{-aff } (x3, y3) \in e'\text{-aff}$
using *assms(1,2,3) e-proj-aff* **by** *auto*

have *one-in*: $\{((1, 0), \text{False})\} \in e\text{-proj}$
using *identity-proj* **by** *force*

have *rot*: $\tau \circ g \in \text{rotations}$ **using** *sym-to-rot assms* **by** *blast*

have *e-proj*: $\text{gluing } \{(g (i (x1, y1)), \text{False})\} \in e\text{-proj}$
 $\text{gluing } \{(i (x1, y1), \text{False})\} \in e\text{-proj}$ (**is** *?ig1* $\in -$)
 $\text{proj-addition } (\text{gluing } \{(i (x1, y1), \text{False})\}) (\text{gluing } \{((x3, y3), \text{False})\}) \in e\text{-proj}$
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*: $\text{proj-addition } ?g1 ?g2 = \text{tf'' } (\tau \circ g) \{((1,0), \text{False})\}$

proof –

have *eq1*: $?g2 = \text{tf'' } (\tau \circ g) ?ig1$
apply *(simp add: assms(5))*
apply *(subst (2 5) prod.collapse[symmetric])*
apply *(subst remove-sym)*
using *e-proj assms* **by** *auto*

have *eq2*: $\text{proj-addition } ?g1 (\text{tf'' } (\tau \circ g) ?ig1) =$
 $\text{tf'' } (\tau \circ g) (\text{proj-addition } ?g1 ?ig1)$
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*: $\text{tf'' } (\tau \circ g) (\text{proj-addition } ?g1 ?ig1) = \text{tf'' } (\tau \circ g) \{((1,0), \text{False})\}$
using *assms(1) proj-add-class-inv xor-def* **by** *auto*

show *?thesis* **using** *eq1 eq2 eq3* **by** *presburger*

qed

have *proj-addition* $(\text{proj-addition } ?g1 ?g2) ?g3 =$
 $\text{proj-addition } (\text{tf'' } (\tau \circ g) \{((1,0), \text{False})\}) ?g3$
using *1* **by** *force*

also have $\dots = \text{tf'' } (\tau \circ g) (\text{proj-addition } \{((1,0), \text{False})\}) ?g3$
by *(simp add: assms(3) one-in remove-add-sym rot)*

also have $\dots = \text{tf'' } (\tau \circ g) ?g3$

using *assms(3) identity-equiv proj-add-class-identity* **by** *simp*

finally show *2*: $\text{proj-addition } (\text{proj-addition } ?g1 ?g2) ?g3 = \text{tf'' } (\tau \circ g) ?g3$
by *blast*

```

have proj-addition ?g1 (proj-addition ?g2 ?g3) =
  proj-addition ?g1 (proj-addition (gluing “ {(g (i (x1, y1)), False)}”) ?g3)
  using assms by simp
also have ... = proj-addition ?g1 (tf'' (τ ∘ g) (proj-addition (gluing “ {(i (x1,
y1), False)}”) ?g3))
proof –
  have eq1: gluing “ {(g (i (x1, y1)), False)} = 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), False} ∈ e-proj
  gluing “ {(x2, y2), False} ∈ e-proj
  gluing “ {(x3, y3), False} ∈ e-proj
assumes a: g ∈ symmetries (x3, y3) = (g ∘ i) (x2, y2)
shows
  proj-addition (proj-addition (gluing “ {(x1, y1), False})) (gluing “ {(x2, y2),
False})) (gluing “ {(x3, y3), False})) =
  proj-addition (gluing “ {(x1, y1), False})) (proj-addition (gluing “ {(x2, y2),

```

```

False)) (gluing “ {(x3, y3), False}”))
  (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-proj-aff by auto

  have one-in: {(1, 0), False} ∈ 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)), False)} ∈ e-proj
    gluing “ {(i (x2, y2), False)} ∈ 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), False}
    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
    using xor-def 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 “  $\{(x1, y1), False\} \in e\text{-proj}$ 
    gluing “  $\{(x2, y2), False\} \in e\text{-proj}$ 
    gluing “  $\{(x3, y3), False\} \in e\text{-proj}$ 
  assumes 22:  $g \in \text{symmetries } (x1, y1) = (g \circ i) (\text{add } (x2,y2) (x3,y3)) ((x2, y2),$ 
 $x3, y3) \in e'\text{-aff-0}$ 
  shows
    proj-addition (proj-addition (gluing “  $\{(x1, y1), False\}$ )) (gluing “  $\{(x2, y2),$ 
 $False\}$ )) (gluing “  $\{(x3, y3), False\}$ ) =
    proj-addition (gluing “  $\{(x1, y1), False\}$ ) (proj-addition (gluing “  $\{(x2, y2),$ 
 $False\}$ )) (gluing “  $\{(x3, y3), False\}$ ))
    (is proj-addition (proj-addition ?g1 ?g2) ?g3 = -)
proof -
  have in-aff:  $(x1,y1) \in e'\text{-aff } (x2,y2) \in e'\text{-aff } (x3,y3) \in e'\text{-aff}$ 
    using assms(1,2,3) e-proj-aff by auto

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

  have p-delta:  $\text{delta } x2 y2 x3 y3 \neq 0$ 
     $\text{delta } (\text{fst } (i (x2,y2))) (\text{snd } (i (x2,y2)))$ 
     $(\text{fst } (i (x3,y3))) (\text{snd } (i (x3,y3))) \neq 0$ 
    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 =  $\text{add } (x2,y2) (x3,y3)$ 
  have add-in:  $\text{add-2-3} \in e'\text{-aff}$ 
    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, False)} ∈ e-proj
      gluing “ {(i add-2-3, False)} ∈ e-proj
using add-in add-2-3-def e-proj-aff apply simp
using add-in add-2-3-def e-proj-aff proj-add-class-inv by auto

from 22 have g-expr:  $g \in \text{symmetries } (x1,y1) = (g \circ i) \text{ add-2-3}$  unfolding
add-2-3-def by auto
then have rot:  $\tau \circ g \in \text{rotations}$  using sym-to-rot by blast

have e-proj-2-3-g: gluing “ {(g (i add-2-3), False)} ∈ 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, False})) (proj-addition ?g2 ?g3)
using g-expr by simp
also have ... = proj-addition (gluing “ {(g ∘ i) add-2-3, False})) (gluing “
“{(add-2-3, False)}”)
using gluing-add add-2-3-def p-delta assms(2,3) xor-def by force
also have ... = tf'' (τ ∘ g) (proj-addition (gluing “ {(i add-2-3, False)})) (gluing
“{(add-2-3, False)}”)
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), False}
apply(subst proj-addition-comm)
using add-2-3-def e-proj-2-3(1) proj-add-class-inv xor-def by auto
finally have eq1: proj-addition ?g1 (proj-addition ?g2 ?g3) =
      tf'' (τ ∘ g) {(1,0), False}

by auto

have proj-addition (proj-addition ?g1 ?g2) ?g3 =
proj-addition (proj-addition (gluing “ {(g ∘ i) add-2-3, False})) ?g2) ?g3
using g-expr by argo
also have ... = proj-addition (tf'' (τ ∘ g)
      (proj-addition (gluing “ {(i add-2-3, False)} ?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-3)
using gluing-add e-proj-0 p-delta xor-def by force
also have ... = proj-addition (tf'' (τ ∘ 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'' (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), False}
  apply(subst proj-addition-comm)
  using assms(3) proj-add-class-inv xor-def by auto
finally have eq2: proj-addition (proj-addition ?g1 ?g2) ?g3 =
  tf'' (τ ∘ g) {(1,0), False} by blast
show ?thesis using eq1 eq2 by argo
qed

lemma assoc-111-ext-add:
  assumes gluing “ {(x1, y1), False} ∈ e-proj
    gluing “ {(x2, y2), False} ∈ e-proj
    gluing “ {(x3, y3), False} ∈ 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), False})) (gluing “ {(x2, y2),
False})) (gluing “ {(x3, y3), False})) =
    proj-addition (gluing “ {(x1, y1), False})) (proj-addition (gluing “ {(x2, y2),
False})) (gluing “ {(x3, y3), False}))
    (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-proj-aff by auto

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

  have e-proj-0: gluing “ {(i (x1,y1), False)} ∈ e-proj (is ?ig1 ∈ e-proj)
    gluing “ {(i (x2,y2), False)} ∈ e-proj (is ?ig2 ∈ e-proj)
    gluing “ {(i (x3,y3), False)} ∈ 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, False)} ∈ e-proj
  gluing “ {(i add-2-3, False)} ∈ e-proj
  using add-in add-2-3-def e-proj-aff apply simp
  using add-in add-2-3-def e-proj-aff proj-add-class-inv by auto

from 22 have g-expr: g ∈ symmetries (x1,y1) = (g ∘ i) add-2-3 unfolding
add-2-3-def by auto
then have rot:  $\tau \circ g \in \text{rotations}$  using sym-to-rot by blast

have e-proj-2-3-g: gluing “ {(g (i add-2-3), False)} ∈ 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, False}) (proj-addition ?g2 ?g3)
  using g-expr by simp
also have ... = proj-addition (gluing “ {(g ∘ i) add-2-3, False}) (gluing “
{(add-2-3, False})
  using gluing-ext-add add-2-3-def p-delta assms(2,3) xor-def by force
also have ... = tf'' ( $\tau \circ g$ ) (proj-addition (gluing “ {(i add-2-3, False}) (gluing
“ {(add-2-3, False})
  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'' ( $\tau \circ g$ ) {(1,0), False}
  apply(subst proj-addition-comm)
  using add-2-3-def e-proj-2-3(1) proj-add-class-inv xor-def by auto
finally have eq1: proj-addition ?g1 (proj-addition ?g2 ?g3) =
  tf'' ( $\tau \circ g$ ) {(1,0), False}
  by auto

have proj-addition (proj-addition ?g1 ?g2) ?g3 =
  proj-addition (proj-addition (gluing “ {(g ∘ i) add-2-3, False}) ?g2) ?g3
  using g-expr by argo
also have ... = proj-addition (tf'' ( $\tau \circ g$ )
(proj-addition (gluing “ {(i add-2-3, False}) ?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 xor-def 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), False}
  using assms(3) proj-add-class-inv proj-addition-comm xor-def by auto
finally have eq2: proj-addition (proj-addition ?g1 ?g2) ?g3 =
  tf'' (τ ∘ g) {(1,0), False} by blast

show ?thesis using eq1 eq2 by argo
qed

lemma assoc-with-zeros:
  assumes gluing “ {(x1, y1), False} ∈ e-proj
    gluing “ {(x2, y2), False} ∈ e-proj
    gluing “ {(x3, y3), False} ∈ e-proj
  shows proj-addition (proj-addition (gluing “ {(x1, y1), False})) (gluing “
  {(x2, y2), False}))
    (gluing “ {(x3, y3), False}) =
    proj-addition (gluing “ {(x1, y1), False})
      (proj-addition (gluing “ {(x2, y2), False})) (gluing “ {(x3,
y3), False}))
  (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-proj-aff by auto

```



```

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

consider
  (1)  $(\exists g \in \text{symmetries}. (x2, y2) = (g \circ i)(x1, y1)) \mid$ 
  (2)  $((x1, y1), (x2, y2)) \in e'\text{-aff-0}$ 
       $\neg ((\exists g \in \text{symmetries}. (x2, y2) = (g \circ i)(x1, y1)) \mid$ 
  (3)  $((x1, y1), (x2, y2)) \in e'\text{-aff-1}$ 
       $\neg ((\exists g \in \text{symmetries}. (x2, y2) = (g \circ i)(x1, y1)) ((x1, y1), (x2, y2)) \notin$ 
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  $\neq 0$ 
      delta (fst (i (x1, y1))) (snd (i (x1, y1)))
      (fst (i (x2, y2))) (snd (i (x2, y2)))  $\neq 0$ 
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  $\in e'\text{-aff}$ 
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 “  $\{(add-1-2, False)\} \in e\text{-proj}$ 
      gluing “  $\{(i add-1-2, False)\} \in e\text{-proj}$ 
using add-in-1-2 add-1-2-def e-proj-aff proj-add-class-inv by auto

consider
  (11)  $(\exists g \in \text{symmetries}. (x3, y3) = (g \circ i)(x2, y2)) \mid$ 
  (22)  $((x2, y2), (x3, y3)) \in e'\text{-aff-0}$ 
       $\neg ((\exists g \in \text{symmetries}. (x3, y3) = (g \circ i)(x2, y2)) \mid$ 
  (33)  $((x2, y2), (x3, y3)) \in e'\text{-aff-1}$ 
       $\neg ((\exists g \in \text{symmetries}. (x3, y3) = (g \circ i)(x2, y2)) ((x2, y2), (x3, y3)) \notin$ 
e'-aff-0
using dichotomy-1 in-aff by blast

```

```

then show ?thesis
proof(cases)
  case 11
  then obtain g where g-expr:  $g \in \text{symmetries } (x3, y3) = (g \circ i) (x2, y2)$ 
by blast
  then show ?thesis using assoc-11 assms by force
next
case 22
have p-delta-2-3:  $\text{delta } x2 \ y2 \ x3 \ y3 \neq 0$ 
       $\text{delta } (\text{fst } (i \ (x2, y2))) \ (\text{snd } (i \ (x2, y2)))$ 
       $(\text{fst } (i \ (x3, y3))) \ (\text{snd } (i \ (x3, y3))) \neq 0$ 
  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 =  $\text{add } (x2, y2) \ (x3, y3)$ 
have add-in:  $\text{add-2-3} \in e'\text{-aff}$ 
  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:  $\text{gluing } \{(\text{add-2-3}, \text{False})\} \in e\text{-proj}$ 
       $\text{gluing } \{(i \ \text{add-2-3}, \text{False})\} \in e\text{-proj}$ 
  using add-in add-2-3-def e-proj-aff apply simp
  using add-in add-2-3-def e-proj-aff proj-add-class-inv by auto

consider
  (111)  $(\exists g \in \text{symmetries}. (x1, y1) = (g \circ i) \ \text{add-2-3}) \mid$ 
  (222)  $(\text{add-2-3}, (x1, y1)) \in e'\text{-aff-0} \wedge ((\exists g \in \text{symmetries}. (x1, y1) = (g \circ i)$ 
add-2-3))  $\mid$ 
  (333)  $(\text{add-2-3}, (x1, y1)) \in e'\text{-aff-1} \wedge ((\exists g \in \text{symmetries}. (x1, y1) = (g \circ i)$ 
add-2-3))
       $(\text{add-2-3}, (x1, y1)) \notin e'\text{-aff-0}$ 
  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 assms:  $((x1, y1), \text{add-2-3}) \in e'\text{-aff-0}$ 
  apply(subst (3) prod.collapse[symmetric])
  using 222 e'-aff-0-invariance by fastforce

consider

```

```

(1111) ( $\exists g \in \text{symmetries}. (x3, y3) = (g \circ i) \text{ add-1-2}$ ) |
(2222) ( $\text{add-1-2}, (x3, y3) \in e'\text{-aff-0} \wedge ((\exists g \in \text{symmetries}. (x3, y3) = (g \circ i) \text{ add-1-2}))$ ) |
(3333) ( $\text{add-1-2}, (x3, y3) \in e'\text{-aff-1} \wedge ((\exists g \in \text{symmetries}. (x3, y3) = (g \circ i) \text{ add-1-2}))$ ) ( $\text{add-1-2}, (x3, y3) \notin e'\text{-aff-0}$ )
using add-in-1-2 in-aff dichotomy-1 by blast
then show ?thesis
proof(cases)
  case 1111
  then obtain g where g-expr:  $g \in \text{symmetries}$  ( $x3, y3) = (g \circ i) \text{ add-1-2}$ 
by blast
  then have rot:  $\tau \circ g \in \text{rotations}$  using sym-to-rot assms by blast

have proj-addition (proj-addition ?g1 ?g2) ?g3 =
  proj-addition (gluing “{(add-1-2, False)}”) (gluing “{((g ∘ i) add-1-2, False)}”)
using g-expr p-delta-1-2 gluing-add assms(1,2) add-1-2-def xor-def by force

also have ... = tf'' ( $\tau \circ g$ ) ({(1, 0), False})
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'' ( $\tau \circ g$ ) ({(1, 0), False}) by blast

have proj-addition ?g1 (proj-addition ?g2 ?g3) =
  proj-addition ?g1 (proj-addition ?g2 (gluing “{((g ∘ i) add-1-2, False)}”)))
using g-expr by auto
also have ... = proj-addition ?g1
  (tf'' ( $\tau \circ g$ )
  (proj-addition (gluing “{(add (i (x1, y1)) (i (x2, y2))), ?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)), False)} =
      proj-addition ?ig1 ?ig2
    using gluing-add[symmetric,of fst (i (x1,y1)) snd (i (x1,y1)) False
      fst (i (x2,y2)) snd (i (x2,y2)) False,
      simplified prod.collapse] e-proj-0(1,2) p-delta-1-2(2)
xor-def
    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), False)})
    using assms(1) proj-add-class-comm proj-add-class-inv xor-def by simp
  finally have eq2: proj-addition ?g1 (proj-addition ?g2 ?g3) =
    tf'' (τ ∘ g) ({((1, 0), False)}) using xor-def 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), False)} ) ?g3
using gluing-add p-delta-1-2(1) e-proj-1-2 add-1-2-def assms(1,2) xor-def
by simp
also have ... = gluing “ {(add (add (x1, y1) (x2, y2)) (x3, y3), False)}
  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 xor-def by(simp,force)
also have ... = gluing “ {(add (x1, y1) (add (x2, y2) (x3, y3)), False)}
  apply(subst add-add-add-add-assoc)
  using p-delta-1-2 p-delta-2-3(1) 2222(1) assms 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),

```

```

False))}
  apply(subst (10) prod.collapse[symmetric])
  apply(subst gluing-add)
  using assms(1) e-proj-2-3(1) add-2-3-def assms xor-def
  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) xor-def 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), False)}”) ?g3
  using gluing-add p-delta-1-2(1) e-proj-1-2 add-1-2-def assms(1,2) xor-def
by simp
  also have ... = gluing “{(ext-add (add (x1, y1) (x2, y2)) (x3, y3), False)}”
  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 xor-def by(simp,force)
  also have ... = gluing “{(add (x1, y1) (add (x2, y2) (x3, y3)), False)}”
  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) assms 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), False)}”)
  apply(subst (10) prod.collapse[symmetric])
  apply(subst gluing-add)
  using assms(1) e-proj-2-3(1) add-2-3-def assms xor-def
  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) xor-def by auto
  finally show ?thesis by blast
qed
next
case 333
  have assms: ((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, False)}”) (gluing “ {(g ∘ i) add-1-2,
False}))
  using g-expr p-delta-1-2 gluing-add assms(1,2) add-1-2-def xor-def by
force

  also have ... = tf'' (τ ∘ g) {(1, 0), False}
  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])
  using e-proj-1-2(1) e-proj-1-2(2) proj-add-class-inv-point(1) proj-addition-comm
xor-def by auto
  finally have eq1: proj-addition (proj-addition ?g1 ?g2) ?g3 =
    tf'' (τ ∘ g) {(1, 0), False} by blast

  have proj-addition ?g1 (proj-addition ?g2 ?g3) =
    proj-addition ?g1 (proj-addition ?g2 (gluing “ {(g ∘ i) add-1-2,
False})))
  using g-expr by auto
  also have ... = proj-addition ?g1
    (tf'' (τ ∘ g)
      (proj-addition (gluing “ {(add (i (x1, y1)) (i (x2, y2)),
False})))
      ?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)), False)} =
      proj-addition ?ig1 ?ig2
    using gluing-add[symmetric, of fst (i (x1,y1)) snd (i (x1,y1)) False
      fst (i (x2, y2)) snd (i (x2, y2)) False,
      simplified prod.collapse] e-proj-0(1,2) p-delta-1-2(2)
xor-def
    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), False}
    using assms(1) proj-add-class-comm proj-addition-def proj-add-class-inv
xor-def by simp
  finally have eq2: proj-addition ?g1 (proj-addition ?g2 ?g3) =
    tf'' (τ ∘ g) {(1, 0), False} using xor-def 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), False)} ?g3
  using gluing-add p-delta-1-2(1) e-proj-1-2 add-1-2-def assms(1,2) xor-def
by simp
  also have ... = gluing “ {(add (add (x1, y1) (x2, y2)) (x3, y3), False)}
  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 xor-def by(simp,force)
  also have ... = gluing “ {(ext-add (x1, y1) (add (x2, y2) (x3, y3)), False)}
  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) assms 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),
False)})
    apply(subst (10) prod.collapse[symmetric])
    apply(subst gluing-ext-add)
    using assms(1) e-proj-2-3(1) add-2-3-def assms xor-def
    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) xor-def 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), False)}) ?g3
using gluing-add p-delta-1-2(1) e-proj-1-2 add-1-2-def assms(1,2) xor-def
by simp
also have ... = gluing “ {(ext-add (add (x1, y1) (x2, y2)) (x3, y3), False)}
  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 xor-def by(simp,force)
also have ... = gluing “ {(ext-add (x1, y1) (add (x2, y2) (x3, y3)), False)}
  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) assms 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),
False)})
  apply(subst (10) prod.collapse[symmetric])
  apply(subst gluing-ext-add)
  using assms(1) e-proj-2-3(1) add-2-3-def assms xor-def
  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) xor-def 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  $\in$   $e'$ -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, False)}  $\in$  e-proj
      gluing “{(i add-2-3, False)}  $\in$  e-proj
  using add-in add-2-3-def e-proj-aff proj-add-class-inv by auto

consider
  (111)  $(\exists g \in \text{symmetries}. (x1, y1) = (g \circ i) \text{ add-2-3}) \mid$ 
  (222)  $(\text{add-2-3}, (x1, y1)) \in e'\text{-aff-0}$ 
       $\neg ((\exists g \in \text{symmetries}. (x1, y1) = (g \circ i) \text{ add-2-3}) \mid$ 
  (333)  $(\text{add-2-3}, (x1, y1)) \in e'\text{-aff-1}$ 
       $\neg ((\exists g \in \text{symmetries}. (x1, y1) = (g \circ i) \text{ add-2-3})$ 
           $(\text{add-2-3}, (x1, y1)) \notin e'\text{-aff-0}$ 
  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 assms:  $((x1, y1), \text{add-2-3}) \in e'\text{-aff-0}$ 
    apply(subst (3) prod.collapse[symmetric])
    using 222  $e'$ -aff-0-invariance by fastforce
  consider
    (1111)  $(\exists g \in \text{symmetries}. (x3, y3) = (g \circ i) \text{ add-1-2}) \mid$ 
    (2222)  $(\text{add-1-2}, (x3, y3)) \in e'\text{-aff-0}$ 
         $\neg ((\exists g \in \text{symmetries}. (x3, y3) = (g \circ i) \text{ add-1-2}) \mid$ 
    (3333)  $(\text{add-1-2}, (x3, y3)) \in e'\text{-aff-1}$ 
         $\neg ((\exists g \in \text{symmetries}. (x3, y3) = (g \circ i) \text{ add-1-2})$ 
             $(\text{add-1-2}, (x3, y3))$ 
 $\notin e'\text{-aff-0}$ 
    using add-in-1-2 in-aff dichotomy-1 by blast
  then show ?thesis
proof(cases)
  case 1111
    then obtain g where g-expr:  $g \in \text{symmetries} (x3, y3) = (g \circ i) \text{ add-1-2}$ 
by blast
    then have rot:  $\tau \circ g \in \text{rotations}$  using sym-to-rot assms by blast

  have proj-addition (proj-addition ?g1 ?g2) ?g3 =

```

```

    proj-addition (gluing “ {(add-1-2, False)} (gluing “ {(g ∘ i) add-1-2,
False}))
  using g-expr p-delta-1-2 gluing-add assms(1,2) add-1-2-def xor-def by
force
  also have ... = tf'' (τ ∘ g) {((1, 0), False)}
  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) xor-def by auto
  finally have eq1: proj-addition (proj-addition ?g1 ?g2) ?g3 =
    tf'' (τ ∘ g) {((1, 0), False)} by blast

  have proj-addition ?g1 (proj-addition ?g2 ?g3) =
    proj-addition ?g1 (proj-addition ?g2 (gluing “ {(g ∘ i) add-1-2,
False}))
  using g-expr by auto
  also have ... = proj-addition ?g1
    (tf'' (τ ∘ g)
      (proj-addition (gluing “ {(add (i (x1, y1)) (i (x2, y2))),
False}))
      ?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))), False} =
    proj-addition ?ig1 ?ig2
  using gluing-add[symmetric, of fst (i (x1,y1)) snd (i (x1,y1)) False
    fst (i (x2,y2)) snd (i (x2,y2)) False,
    simplified prod.collapse] e-proj-0(1,2) p-delta-1-2(2)

```

```

xor-def
  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), False}
  using assms(1) proj-add-class-comm proj-addition-def proj-add-class-inv
xor-def by auto
  finally have eq2: proj-addition ?g1 (proj-addition ?g2 ?g3) =
    tf'' (τ ∘ g) {(1, 0), False} 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), False)}" ?g3)
  using gluing-add p-delta-1-2(1) e-proj-1-2 add-1-2-def assms(1,2) xor-def
by simp
  also have ... = gluing "{(add (add (x1, y1) (x2, y2)) (x3, y3), False)}"
  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 xor-def by(simp,force)
  also have ... = gluing "{(add (x1, y1) (ext-add (x2, y2) (x3, y3)), False)}"
  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),
False)}")
  apply(subst (10) prod.collapse[symmetric])
  apply(subst gluing-add)
  using assms(1) e-proj-2-3(1) add-2-3-def assumps xor-def
  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) xor-def 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), False)} ”) ?g3
using gluing-add p-delta-1-2(1) e-proj-1-2 add-1-2-def assms(1,2) xor-def
by simp
also have ... = gluing “ {(ext-add (add (x1, y1) (x2, y2)) (x3, y3), False)} ”
  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 xor-def unfolding e'-aff-1-def add-1-2-def by(simp,force)
also have ... = gluing “ {(add (x1, y1) (ext-add (x2, y2) (x3, y3)), False)} ”
  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),
False)} ”)
  apply(subst (10) prod.collapse[symmetric])
  apply(subst gluing-add)
  using assms(1) e-proj-2-3(1) add-2-3-def assumps xor-def
  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) xor-def 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:  $\tau \circ g \in \text{rotations}$  using sym-to-rot assms by blast

  have proj-addition (proj-addition ?g1 ?g2) ?g3 =
    proj-addition (gluing “ {(add-1-2, False)}”) (gluing “ {(g  $\circ$  i) add-1-2,
False}))
  using g-expr p-delta-1-2 gluing-add assms(1,2) add-1-2-def xor-def by
force

  also have ... = tf'' ( $\tau \circ g$ ) {(1, 0), False}
  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) xor-def by auto
  finally have eq1: proj-addition (proj-addition ?g1 ?g2) ?g3 =
    tf'' ( $\tau \circ g$ ) {(1, 0), False} by blast

  have proj-addition ?g1 (proj-addition ?g2 ?g3) =
    proj-addition ?g1 (proj-addition ?g2 (gluing “ {(g  $\circ$  i) add-1-2,
False})))
  using g-expr by auto
  also have ... = proj-addition ?g1
    (tf'' ( $\tau \circ g$ )
      (proj-addition (gluing “ {(add (i (x1, y1)) (i (x2, y2))),
False})))
      ?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))), False} =
      proj-addition ?ig1 ?ig2

```

```

      using gluing-add[symmetric, of fst (i (x1, y1)) snd (i (x1, y1)) False
        fst (i (x2, y2)) snd (i (x2, y2)) False,
        simplified prod.collapse] e-proj-0(1,2) p-delta-1-2(2)
xor-def
  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), False)}
  using assms(1) proj-add-class-comm proj-addition-def proj-add-class-inv
xor-def by auto
  finally have eq2: proj-addition (gluing “ {((x1, y1), False)}
    (proj-addition (gluing “ {((x2, y2), False)} (gluing “
{((x3, y3), False)})) =
    tf'' (τ ∘ g) {((1, 0), False)} 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), False)} ?g3
using gluing-add p-delta-1-2(1) e-proj-1-2 add-1-2-def assms(1,2) xor-def
by simp
also have ... = gluing “ {(add (add (x1, y1) (x2, y2)) (x3, y3), False)}
  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 xor-def 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)),
False)}
  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),
False)})
  apply(subst (10) prod.collapse[symmetric])
  apply(subst gluing-ext-add)
  using assms(1) e-proj-2-3(1) add-2-3-def assumps xor-def

```

```

    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) xor-def 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), False)} ”) ?g3
  using gluing-add p-delta-1-2(1) e-proj-1-2 add-1-2-def assms(1,2) xor-def
by simp
  also have ... = gluing “ {(ext-add (add (x1, y1) (x2, y2)) (x3, y3), False)} ”
  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 xor-def 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)),
False)} ”
  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) assms 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),
False)} ”)
  apply(subst (10) prod.collapse[symmetric])
  apply(subst gluing-ext-add)
  using assms(1) e-proj-2-3(1) add-2-3-def assms xor-def
  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) xor-def 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, False)} ∈ e-proj
  gluing “ {(i add-1-2, False)} ∈ e-proj
  using add-in-1-2 add-1-2-def e-proj-aff proj-add-class-inv by auto

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, False)} ∈ e-proj
  gluing “ {(i add-2-3, False)} ∈ e-proj

```



```

using add-in add-2-3-def e-proj-aff apply simp
using add-in add-2-3-def e-proj-aff proj-add-class-inv by auto

consider
  (111)  $(\exists g \in \text{symmetries}. (x1, y1) = (g \circ i) \text{ add-2-3}) \mid$ 
  (222)  $(\text{add-2-3}, (x1, y1)) \in e'\text{-aff-0}$ 
     $\neg ((\exists g \in \text{symmetries}. (x1, y1) = (g \circ i) \text{ add-2-3}) \mid$ 
  (333)  $(\text{add-2-3}, (x1, y1)) \in e'\text{-aff-1}$ 
     $\neg ((\exists g \in \text{symmetries}. (x1, y1) = (g \circ i) \text{ add-2-3}) (\text{add-2-3}, (x1, y1)) \notin$ 
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:  $((x1, y1), \text{add-2-3}) \in e'\text{-aff-0}$ 
    apply(subst (3) prod.collapse[symmetric])
    using 222 e'-aff-0-invariance by fastforce

consider
  (1111)  $(\exists g \in \text{symmetries}. (x3, y3) = (g \circ i) \text{ add-1-2}) \mid$ 
  (2222)  $(\text{add-1-2}, (x3, y3)) \in e'\text{-aff-0}$ 
     $\neg ((\exists g \in \text{symmetries}. (x3, y3) = (g \circ i) \text{ add-1-2}) \mid$ 
  (3333)  $(\text{add-1-2}, (x3, y3)) \in e'\text{-aff-1}$ 
     $\neg ((\exists g \in \text{symmetries}. (x3, y3) = (g \circ i) \text{ add-1-2}) (\text{add-1-2}, (x3, y3))$ 
 $\notin e'\text{-aff-0}$ 
  using add-in-1-2 in-aff dichotomy-1 by blast
then show ?thesis
proof(cases)
  case 1111
  then obtain g where g-expr:  $g \in \text{symmetries } (x3, y3) = (g \circ i) \text{ add-1-2}$ 
by blast
  then have rot:  $\tau \circ g \in \text{rotations}$  using sym-to-rot assms by blast

  have proj-addition (proj-addition ?g1 ?g2) ?g3 =
    proj-addition (gluing “{(add-1-2, False)}”) (gluing “{((g ∘ i) add-1-2,
False)})
  using g-expr p-delta-1-2 gluing-ext-add assms(1,2) add-1-2-def xor-def
by auto
  also have ... = tf'' ( $\tau \circ g$ ) ({(1, 0), False})
  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])
  using e-proj-1-2(1) e-proj-1-2(2) proj-add-class-inv-point(1) proj-addition-comm
xor-def by auto
  finally have eq1: proj-addition (proj-addition ?g1 ?g2) ?g3 =
    tf'' (τ ∘ g) ({((1, 0), False)}) by blast

  have proj-addition ?g1 (proj-addition ?g2 ?g3) =
    proj-addition ?g1 (proj-addition ?g2 (gluing “ {(g ∘ i) add-1-2,
False})))
  using g-expr by auto
  also have ... = proj-addition ?g1
    (tf'' (τ ∘ g)
    (proj-addition (gluing “ {(ext-add (i (x1, y1)) (i (x2, y2)),
False})))
    ?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)), False)} =
      proj-addition ?ig1 ?ig2
    using gluing-ext-add[symmetric,of fst (i (x1,y1)) snd (i (x1,y1)) False
      fst (i (x2,y2)) snd (i (x2,y2)) False,
      simplified prod.collapse] e-proj-0(1,2) p-delta-1-2(2)
xor-def
    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), False)})
    using assms(1) proj-add-class-comm proj-add-class-inv xor-def by simp
  finally have eq2: proj-addition ?g1 (proj-addition ?g2 ?g3) =

```

```

      tf'' (τ ∘ g) ({((1, 0), False)}) 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), False)} ”) ?g3
    using gluing-ext-add p-delta-1-2(1) e-proj-1-2 add-1-2-def assms(1,2)
xor-def by simp
  also have ... = gluing “ {(add (ext-add (x1, y1) (x2, y2)) (x3, y3), False)} ”
    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 xor-def by(simp,force)
  also have ... = gluing “ {(add (x1, y1) (add (x2, y2) (x3, y3)), False)} ”
    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),
False)})
    apply(subst (10) prod.collapse[symmetric])
    apply(subst gluing-add)
    using assms(1) e-proj-2-3(1) add-2-3-def assumps xor-def
    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) xor-def 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), False)} ”) ?g3
    using gluing-ext-add p-delta-1-2(1) e-proj-1-2 add-1-2-def assms(1,2)
xor-def by simp
  also have ... = gluing “ {(ext-add (ext-add (x1, y1) (x2, y2)) (x3, y3),
False)} ”
    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 xor-def by(simp,force)
  also have ... = gluing “ {(add (x1, y1) (add (x2, y2) (x3, y3)), False)} ”
    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),
False)}})
apply(subst (10) prod.collapse[symmetric])
apply(subst gluing-add)
using assms(1) e-proj-2-3(1) add-2-3-def assumps xor-def
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) xor-def 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, False)}}) (gluing “ {(g ∘ i) add-1-2,
False)}})
using g-expr p-delta-1-2 gluing-ext-add assms(1,2) add-1-2-def xor-def
by force
also have ... = tf'' (τ ∘ g) {(1, 0), False}
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 (simp add: e-proj-1-2(1) e-proj-1-2(2) proj-add-class-inv-point(1)
proj-addition-comm xor-def)
    finally have eq1: proj-addition (proj-addition ?g1 ?g2) ?g3 =
      tf'' (τ ∘ g) {(1, 0), False} by blast

    have proj-addition ?g1 (proj-addition ?g2 ?g3) =
      proj-addition ?g1 (proj-addition ?g2 (gluing “ {(g ∘ i) add-1-2,
False})))
    using g-expr by auto
    also have ... = proj-addition ?g1
      (tf'' (τ ∘ g)
      (proj-addition (gluing “ {(ext-add (i (x1, y1)) (i (x2, y2)),
False})))
      ?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)), False)} =
        proj-addition ?ig1 ?ig2
      using gluing-ext-add[symmetric, of fst (i (x1,y1)) snd (i (x1,y1)) False
        fst (i (x2, y2)) snd (i (x2, y2)) False,
        simplified prod.collapse] e-proj-0(1,2) p-delta-1-2(2)
xor-def
      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), False}
      using assms(1) proj-add-class-comm proj-addition-def proj-add-class-inv
xor-def by simp
    finally have eq2: proj-addition ?g1 (proj-addition ?g2 ?g3) =

```

```

      tf'' (τ ∘ g) {(1, 0), False} 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), False)} ”) ?g3
    using gluing-ext-add p-delta-1-2(1) e-proj-1-2 add-1-2-def assms(1,2)
xor-def by simp
  also have ... = gluing “ {(add (ext-add (x1, y1) (x2, y2)) (x3, y3), False)} ”
    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 xor-def unfolding e'-aff-0-def add-1-2-def by(simp,force)
  also have ... = gluing “ {(ext-add (x1, y1) (add (x2, y2) (x3, y3)), False)} ”
    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),
False)} ”)
    apply(subst (10) prod.collapse[symmetric])
    apply(subst gluing-ext-add)
    using assms(1) e-proj-2-3(1) add-2-3-def assumps xor-def
    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) xor-def 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), False)} ”) ?g3
    using gluing-ext-add p-delta-1-2(1) e-proj-1-2 add-1-2-def assms(1,2)
xor-def by simp
  also have ... = gluing “ {(ext-add (ext-add (x1, y1) (x2, y2)) (x3, y3),
False)} ”
    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 xor-def by(simp,force)
  also have ... = gluing “ {(ext-add (x1, y1) (add (x2, y2) (x3, y3)), False)} ”

```

```

    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) assms 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),
False)})
    apply(subst (10) prod.collapse[symmetric])
    apply(subst gluing-ext-add)
    using assms(1) e-proj-2-3(1) add-2-3-def assms xor-def
    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) xor-def 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, False)} ∈ e-proj
      gluing “ {(i add-2-3, False)} ∈ e-proj
    using add-in add-2-3-def e-proj-aff apply simp
    using add-in add-2-3-def e-proj-aff proj-add-class-inv by auto

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, False)}”) (gluing “{((g ∘ i) add-1-2,
False)})
      using g-expr p-delta-1-2 gluing-ext-add assms(1,2) add-1-2-def xor-def
    by force
      also have ... = tf'' (τ ∘ g) {((1, 0), False)}
      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) xor-def by auto
      finally have eq1: proj-addition (proj-addition ?g1 ?g2) ?g3 =
        tf'' (τ ∘ g) {((1, 0), False)} by blast

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

```



```

    using g-expr by auto
  also have ... = proj-addition ?g1
    (tf'' ( $\tau \circ g$ )
     (proj-addition (gluing “ {(ext-add (i (x1, y1)) (i (x2, y2)),
```

False)}))

```

    ?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)), False)} =
      proj-addition ?ig1 ?ig2
    using gluing-ext-add[symmetric, of fst (i (x1,y1)) snd (i (x1,y1)) False
      fst (i (x2,y2)) snd (i (x2,y2)) False,
      simplified prod.collapse] e-proj-0(1,2) p-delta-1-2(2)
xor-def
    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), False}
    using assms(1) proj-add-class-comm proj-addition-def proj-add-class-inv
xor-def by auto
  finally have eq2: proj-addition ?g1 (proj-addition ?g2 ?g3) =
    tf'' ( $\tau \circ g$ ) {(1, 0), False} 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), False)})) ?g3
    using gluing-ext-add p-delta-1-2(1) e-proj-1-2 add-1-2-def assms(1,2)
xor-def by simp
  also have ... = gluing “ {(add (ext-add (x1, y1) (x2, y2)) (x3, y3), False)}

```

```

    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 xor-def unfolding e'-aff-0-def add-1-2-def by(simp,force)
  also have ... = gluing “ {(add (x1, y1) (ext-add (x2, y2) (x3, y3)), False)}
    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),
False)})
    apply(subst (10) prod.collapse[symmetric])
    apply(subst gluing-add)
    using assms(1) e-proj-2-3(1) add-2-3-def assumps xor-def
    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) xor-def 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), False)}) ?g3
    using gluing-ext-add p-delta-1-2(1) e-proj-1-2 add-1-2-def assms(1,2)
xor-def by simp
  also have ... = gluing “ {(ext-add (ext-add (x1, y1) (x2, y2)) (x3, y3),
False)}
    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 xor-def unfolding e'-aff-1-def add-1-2-def by(simp,force)
  also have ... = gluing “ {(add (x1, y1) (ext-add (x2, y2) (x3, y3)), False)}
    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),
False)})

```

```

    apply(subst (10) prod.collapse[symmetric])
    apply(subst gluing-add)
    using assms(1) e-proj-2-3(1) add-2-3-def assms xor-def
    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) xor-def by auto
    finally show ?thesis by blast
qed
next
case 333
have assms: ((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, False)}”) (gluing “{(g ∘ i) add-1-2,
False}”)
  using g-expr p-delta-1-2 gluing-ext-add assms(1,2) add-1-2-def xor-def
by force
  also have ... = tf'' (τ ∘ g) {((1, 0), False)}
  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) xor-def by auto
  finally have eq1: proj-addition (proj-addition ?g1 ?g2) ?g3 =
    tf'' (τ ∘ g) {((1, 0), False)} using xor-def by blast

```

```

have proj-addition ?g1 (proj-addition ?g2 ?g3) =
      proj-addition ?g1 (proj-addition ?g2 (gluing “ {(g ∘ i) add-1-2,
False})))
  using g-expr by auto
also have ... = proj-addition ?g1
      (tf'' (τ ∘ g)
      (proj-addition (gluing “ {(ext-add (i (x1, y1)) (i (x2, y2)),
False})))
      ?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)), False)} =
      proj-addition ?ig1 ?ig2
  using gluing-ext-add[symmetric, of fst (i (x1, y1)) snd (i (x1, y1))
False
      fst (i (x2, y2)) snd (i (x2, y2)) False,
simplified prod.collapse] e-proj-0(1,2) p-delta-1-2(2)
xor-def
  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), False}
  using assms(1) proj-add-class-comm proj-addition-def proj-add-class-inv
xor-def by auto
  finally have eq2: proj-addition (gluing “ {(x1, y1), False})
      (proj-addition (gluing “ {(x2, y2), False})) (gluing “
{{{x3, y3}, False}})) =
      tf'' (τ ∘ g) {(1, 0), False} 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), False)} ” ?g3
    using gluing-ext-add p-delta-1-2(1) e-proj-1-2 add-1-2-def assms(1,2)
xor-def by simp
  also have ... = gluing “ {(add (ext-add (x1, y1) (x2, y2)) (x3, y3), False)} ”
    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 xor-def by(simp,force)
  also have ... = gluing “ {(ext-add (x1, y1) (ext-add (x2, y2) (x3, y3)),
False)} ”
    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),
False)} ”)
    apply(subst (10) prod.collapse[symmetric])
    apply(subst gluing-ext-add)
    using assms(1) e-proj-2-3(1) add-2-3-def assumps xor-def
    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) xor-def 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), False)} ” ?g3
    using gluing-ext-add p-delta-1-2(1) e-proj-1-2 add-1-2-def assms(1,2)
xor-def by simp
  also have ... = gluing “ {(ext-add (ext-add (x1, y1) (x2, y2)) (x3, y3),
False)} ”
    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 xor-def 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)),
False)} ”

```

```

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),
False)})
apply(subst (10) prod.collapse[symmetric])
apply(subst gluing-ext-add)
using assms(1) e-proj-2-3(1) add-2-3-def assumps xor-def
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) xor-def 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), False}) (gluing “
{(x2, y2), False})))

```

```

          (gluing “ {(x3, y3), False})))

```

```

let ?t2 = proj-addition (gluing “ {(x1, y1), False})
          (proj-addition (gluing “ {(x2, y2), False}) (gluing “ {(x3, y3),
False)})

```

```

have e-proj-0: gluing “ {(x1, y1), False} ∈ e-proj

```

```

gluing “ {(x2, y2), False} ∈ e-proj

```

```

gluing “ {(x3, y3), False} ∈ e-proj

```

```

gluing “ {(x1, y1), True} ∈ e-proj

```

```

gluing “ {(x2, y2), True} ∈ e-proj

```

```

gluing “ {(x3, y3), True} ∈ e-proj

```

```

using assms e-proj-aff by blast+

```

```

have e-proj-add-0: proj-addition (gluing “ {(x1, y1), False}) (gluing “ {(x2,
y2), False}) ∈ e-proj

```

```

      proj-addition (gluing “ {((x2, y2), False)} (gluing “ {((x3, y3),
False)}) ∈ e-proj
      proj-addition (gluing “ {((x2, y2), False)} (gluing “ {((x3, y3),
True)}) ∈ e-proj
      proj-addition (gluing “ {((x1, y1), False)} (gluing “ {((x2, y2),
True)}) ∈ e-proj
      proj-addition (gluing “ {((x2, y2), True)} (gluing “ {((x3, y3),
False)}) ∈ e-proj
      proj-addition (gluing “ {((x2, y2), True)} (gluing “ {((x3, y3),
True)}) ∈ 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 = False)
  case True
  then have l: l = False by simp
  then show ?thesis
  proof(cases m = False)
  case True
  then have m: m = False by simp
  then show ?thesis
  proof(cases n = False)
  case True
  then have n: n = False by simp
  show ?thesis
  using l m n assms assoc-with-zeros by simp
  next
  case False
  then have n: n = True by simp
  have eq1: proj-addition (proj-addition (gluing “ {((x1, y1), False)} (gluing
“ {((x2, y2), False)}))
    (gluing “ {((x3, y3), True)} = tf' (?t1)
  apply(subst tf-tau[of - - False,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), False)}
    (proj-addition (gluing “ {((x2, y2), False)} (gluing “ {((x3,
y3), True)})) =
    tf' (?t2)
  apply(subst tf-tau[of - - False,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 = True by simp
then show ?thesis
proof(cases n = False)
  case True
  then have n: n = False by simp

  have eq1: proj-addition (proj-addition (gluing “ {((x1, y1), False)}”) (gluing
“ {((x2, y2), True)}))
    (gluing “ {((x3, y3), False)}”) = tf'(?t1)
    apply(subst tf-tau[of - - False,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), False)}”)
    (proj-addition (gluing “ {((x2, y2), True)}”) (gluing “ {((x3, y3),
False)})) =
    tf'(?t2)
    apply(subst tf-tau[of - - False,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 = True by simp

  have eq1: proj-addition (proj-addition (gluing “ {((x1, y1), False)}”) (gluing
“ {((x2, y2), True)}))
    (gluing “ {((x3, y3), True)}”) = ?t1
    apply(subst tf-tau[of - - False,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 - - False,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), False)}”)
    (proj-addition (gluing “ {((x2, y2), True)}”) (gluing “ {((x3, y3), True)}))
=
    ?t2

```



```

    apply(subst tf-tau[of - - False,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 - - False,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 l: l = True by simp

  then show ?thesis
  proof(cases m = False)
    case True
    then have m: m = False by simp
    then show ?thesis
  proof(cases n = False)
    case True
    then have n: n = False by simp

      have eq1: proj-addition (proj-addition (gluing “ {((x1, y1), True)} ) (gluing
“ {((x2, y2), False)}))
        (gluing “ {((x3, y3), False)} ) = tf'(?t1)
      apply(subst tf-tau[of - - False,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), True)} )
        (proj-addition (gluing “ {((x2, y2), False)} ) (gluing “ {((x3, y3), False)}))
    =
      tf'(?t2)
      apply(subst tf-tau[of - - False,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 = True by simp
  have eq1: proj-addition (proj-addition (gluing “ {((x1, y1), True)} ) (gluing
“ {((x2, y2), False)}))
    (gluing “ {((x3, y3), True)} ) = ?t1

```

```

apply(subst tf-tau[of - - False,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 - - False,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), True}”)
  (proj-addition (gluing “ {(x2, y2), False}”) (gluing “ {(x3, y3), True}”))
=
  ?t2
apply(subst tf-tau[of - - False,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 - - False,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 = True by simp
then show ?thesis
proof(cases n = False)
  case True
  then have n: n = False by simp

  have eq1: proj-addition (proj-addition (gluing “ {(x1, y1), True}”) (gluing
“ {(x2, y2), True}”))
  (gluing “ {(x3, y3), False}”) = ?t1
  apply(subst tf-tau[of - - False,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 - - False,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), True}”)
  (proj-addition (gluing “ {(x2, y2), True}”) (gluing “ {(x3, y3), False}”))
=
  ?t2
apply(subst tf-tau[of - - False,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 - - False,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 = True by simp

have eq1: proj-addition (proj-addition (gluing “ {(x1, y1), True}”)) (gluing
“ {(x2, y2), True}”))
  (gluing “ {(x3, y3), True}”)) = tf'(?t1)
apply(subst tf-tau[of - - False,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 - - False,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 - - False,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), True}”))
(proj-addition (gluing “ {(x2, y2), True}”)) (gluing “ {(x3, y3), True}”)) =
  tf'(?t2)
apply(subst tf-tau[of - - False,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 - - False,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 - - False,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

4.4 Group law

theorem projective-group-law:
  shows comm-group (|carrier = e-proj, mult = proj-addition, one = {((1,0),False)}|)

proof(unfold-locales,simp-all)
  show one-in: {((1, 0), False)} ∈ 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), False)} x = x
    if x ∈ e-proj for x
    using proj-add-class-identity that by simp

  show id-2: proj-addition x {((1, 0), False)} = 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),
False)}|)
    unfolding Units-def
  proof(simp,standard)
    fix x
    assume x ∈ 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), False)}
      proj-addition (gluing “ {(x', y'), l'})
      (gluing “ {(i (x', y'), l')}) =
      {((1, 0), False)}
      gluing “ {(i (x', y'), l')} ∈ e-proj
    using proj-add-class-inv proj-addition-comm ⟨x ∈ e-proj⟩ xor-def by simp+
  then show x ∈ {y ∈ e-proj. ∃ x∈e-proj. proj-addition x y = {((1, 0), False)}}

```

```

^
  proj-addition y x = {((1, 0), False)}
using ⟨x = gluing “ {((x', y'), l')⟩ ⟨x ∈ e-proj⟩ by blast
qed

show proj-addition x y ∈ e-proj
if x ∈ e-proj y ∈ 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 ∈ e-proj y ∈ e-proj z ∈ e-proj for x y z
using proj-assoc that by simp
qed

end

end

```

References

- [1] T. Hales and R. Raya. Formal proof of the group law for edwards elliptic curves. In N. Peltier and V. Sofronie-Stokkermans, editors, *Automated Reasoning*, pages 254–269, Cham, 2020. Springer International Publishing.