curves-safe

raya

November 25, 2019

Contents

theory *Hales*
  imports *HOL−Algebra.Group HOL−Library.Bit HOL−Library.Rewrite*
begin

declare [[*quick-and-dirty=true*]]

1 Affine Edwards curves

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

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

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*))

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*)

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

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

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

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

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

  have simp1gx:
    (x1' ∗ x3 − c ∗ y1' ∗ y3) ∗ delta-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 - / □ 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 - / ¤ delta-minus-def[symmetric] delta-plus-def[symmetric])+
    unfolding delta-def
    by(simp add: divide-simps assms(5−8))

  have ∃ r1 r2 r3. gxpoly = r1 ∗ e1 + r2 ∗ e2 + r3 ∗ e3
    unfolding gxpoly-def g_x-def Delta_x-def
    apply(simp add: assms(1,2))
    apply(rewrite in - / ¤ 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 show ?thesis
    sorry

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

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 t where t = t'
```

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

definition *delta-x* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ **where**
delta-x x1 y1 x2 y2 = *x2∗y1* − *x1∗y2*
definition *delta-y* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ **where**
delta-y x1 y1 x2 y2 = *x1∗x2* + *y1∗y2*
definition *delta′* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ **where**
delta′ x1 y1 x2 y2 = *delta-x x1 y1 x2 y2* ∗ *delta-y x1 y1 x2 y2*

definition *e′* **where** *e′ x y* = *x^2* + *y^2* − *1* − *t^2* ∗ *x^2* ∗ *y^2*

definition *e′-aff* = {(*x*,*y*). *e′ x y* = *0*}

definition *gluing* :: ((('a × 'a) × *bit*) × (('a × 'a) × *bit*)) *set* **where**
gluing = {(((*x0*,*y0*),*l*),((*x1*,*y1*),*j*)).
((*x0*,*y0*) ∈ *e′-aff* ∧ (*x1*,*y1*) ∈ *e′-aff*) ∧
((*x0* ≠ *0* ∧ *y0* ≠ *0* ∧ (*x1*,*y1*) = τ (*x0*,*y0*) ∧ *j* = *l*+*1*) ∨
(*x0* = *x1* ∧ *y0* = *y1* ∧ *l* = *j*)))}

lemma *coherence*:
assumes *delta x1 y1 x2 y2* ≠ *0* *delta′ x1 y1 x2 y2* ≠ *0*
assumes *e′ x1 y1* = *0* *e′ x2 y2* = *0*
shows *ext-add* (*x1*,*y1*) (*x2*,*y2*) = *add* (*x1*,*y1*) (*x2*,*y2*)
sorry

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

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

definition e'*-aff-0* **where**
e'*-aff-0* = $\{((x1,y1),(x2,y2)).\ (x1,y1) \in e'\text{-}aff\ \wedge$
$(x2,y2) \in e'\text{-}aff\ \wedge$
$delta\ x1\ y1\ x2\ y2 \neq 0\ \}$

definition e'*-aff-1* **where**
e'*-aff-1* = $\{((x1,y1),(x2,y2)).\ (x1,y1) \in e'\text{-}aff\ \wedge$
$(x2,y2) \in e'\text{-}aff\ \wedge$
$delta'\ x1\ y1\ x2\ y2 \neq 0\ \}$

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

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

function (*domintros*) *proj-add-class* :: $(('a \times 'a) \times bit)$ *set* $\Rightarrow$
$(('a \times 'a) \times bit)$ *set* $\Rightarrow$
$((('a \times 'a) \times bit)$ *set*) *set*
where
proj-add-class c1 c2 =
(
{
proj-add ((*x1*, *y1*), *i*) ((*x2*, *y2*), *j*) |
x1 y1 i x2 y2 j.
$((x1, y1), i) \in c1\ \wedge$
$((x2, y2), j) \in c2\ \wedge$
$((x1, y1), (x2, y2)) \in e'\text{-}aff\text{-}0 \cup e'\text{-}aff\text{-}1$
} // *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*)

end

end