curves-safe

raya

December 20, 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*: $2 \neq 0$

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

fun *add* :: $'a \times 'a \Rightarrow 'a \times 'a \Rightarrow 'a \times 'a$ **where**
 $add\ (x_1,y_1)\ (x_2,y_2) =$
   $((x_1*x_2\ -\ c*y_1*y_2)\ div\ (1-d*x_1*y_1*x_2*y_2),$
   $(x_1*y_2+y_1*x_2)\ div\ (1+d*x_1*y_1*x_2*y_2))$

definition *delta-plus* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ **where**
 delta-plus x1 y1 x2 y2 = *1* + *d* ∗ *x1* ∗ *y1* ∗ *x2* ∗ *y2*

definition *delta-minus* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ **where**
 delta-minus x1 y1 x2 y2 = *1* − *d* ∗ *x1* ∗ *y1* ∗ *x2* ∗ *y2*

definition *delta* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ **where**
 delta x1 y1 x2 y2 = (*delta-plus x1 y1 x2 y2*) ∗
        (*delta-minus x1 y1 x2 y2*)

definition *e* :: $'a \Rightarrow 'a \Rightarrow 'a$ **where**
 $e\ x\ y = x\hat{}2 + c * y\hat{}2 - 1 - d * x\hat{}2 * y\hat{}2$

```
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 $\tau :: 'a \times 'a \Rightarrow 'a \times 'a$ **where**
$\quad \tau\ (x,y) = (1/(t*x),1/(t*y))$

definition $\mathit{delta\text{-}x} :: 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ **where**
$\quad \mathit{delta\text{-}x}\ x1\ y1\ x2\ y2 = x2*y1 - x1*y2$
definition $\mathit{delta\text{-}y} :: 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ **where**
$\quad \mathit{delta\text{-}y}\ x1\ y1\ x2\ y2 = x1*x2 + y1*y2$
definition $\mathit{delta}' :: 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ **where**
$\quad \mathit{delta}'\ x1\ y1\ x2\ y2 = \mathit{delta\text{-}x}\ x1\ y1\ x2\ y2 * \mathit{delta\text{-}y}\ x1\ y1\ x2\ y2$

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

definition $e'\text{-}\mathit{aff} = \{(x,y).\ e'\ x\ y = 0\}$

definition $\mathit{gluing} :: ((('a \times 'a) \times \mathit{bit}) \times (('a \times 'a) \times \mathit{bit}))\ \mathit{set}$ **where**
$\quad \mathit{gluing} = \{(((x0,y0),l),((x1,y1),j)).$
$\qquad ((x0,y0) \in e'\text{-}\mathit{aff} \wedge (x1,y1) \in e'\text{-}\mathit{aff}) \wedge$
$\qquad ((x0 \neq 0 \wedge y0 \neq 0 \wedge (x1,y1) = \tau\ (x0,y0) \wedge j = l+1) \vee$
$\qquad\ (x0 = x1 \wedge y0 = y1 \wedge l = j))\}$

lemma *coherence*:
assumes $\mathit{delta}\ x1\ y1\ x2\ y2 \neq 0\ \mathit{delta}'\ x1\ y1\ x2\ y2 \neq 0$
assumes $e'\ x1\ y1 = 0\ e'\ x2\ y2 = 0$
shows $\mathit{ext\text{-}add}\ (x1,y1)\ (x2,y2) = \mathit{add}\ (x1,y1)\ (x2,y2)$
sorry

type-synonym $('b)\ \mathit{ppoint} = \langle(('b \times 'b) \times \mathit{bit})\rangle$

function $\mathit{proj\text{-}add} :: 'a\ \mathit{ppoint} \times 'a\ \mathit{ppoint} \Rightarrow 'a\ \mathit{ppoint}$ **where**
$\quad \mathit{proj\text{-}add}\ (((x_1,\ y_1),\ l),((x_2,\ y_2),\ j)) = (\mathit{add}\ (x_1,\ y_1)\ (x_2,\ y_2),\ l+j)$
if $\mathit{delta}\ x_1\ y_1\ x_2\ y_2 \neq 0 \wedge (x_1,\ y_1) \in e'\text{-}\mathit{aff} \wedge (x_2,\ y_2) \in e'\text{-}\mathit{aff}$
$|\ \mathit{proj\text{-}add}\ (((x_1,\ y_1),\ l),((x_2,\ y_2),\ j)) = (\mathit{ext\text{-}add}\ (x_1,\ y_1)\ (x_2,\ y_2),\ l+j)$
if $\mathit{delta}'\ x_1\ y_1\ x_2\ y_2 \neq 0 \wedge (x_1,\ y_1) \in e'\text{-}\mathit{aff} \wedge (x_2,\ y_2) \in e'\text{-}\mathit{aff}$
sorry

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

definition $e'\text{-}\mathit{aff}\text{-}0$ **where**
$\quad e'\text{-}\mathit{aff}\text{-}0 = \{((x_1,y_1),(x_2,y_2)).\ (x_1,y_1) \in e'\text{-}\mathit{aff} \wedge$
$\qquad (x_2,y_2) \in e'\text{-}\mathit{aff} \wedge$
$\qquad \mathit{delta}\ x_1\ y_1\ x_2\ y_2 \neq 0\ \}$

definition $e'\text{-}\mathit{aff}\text{-}1$ **where**
$\quad e'\text{-}\mathit{aff}\text{-}1 = \{((x_1,y_1),(x_2,y_2)).\ (x_1,y_1) \in e'\text{-}\mathit{aff} \wedge$

$(x_2,y_2) \in e'\text{-}aff \wedge$
$delta'\ x_1\ y_1\ x_2\ y_2 \neq 0\ \}$

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

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

term $proj\text{-}add\ `\ \{(((x_1,\ y_1),\ i),((x_2,\ y_2),\ j)).$
$(((x_1,\ y_1),\ i),((x_2,\ y_2),\ j)) \in c_1 \times c_2 \wedge$
$((x_1,\ y_1),\ (x_2,\ y_2)) \in e'\text{-}aff\text{-}0 \cup e'\text{-}aff\text{-}1\}$

term $\{(((x_1,\ y_1),\ i),((x_2,\ y_2),\ j)).$
$(((x_1,\ y_1),\ i),((x_2,\ y_2),\ j)) \in c_1 \times c_2 \wedge$
$((x_1,\ y_1),\ (x_2,\ y_2)) \in e'\text{-}aff\text{-}0 \cup e'\text{-}aff\text{-}1\}$

type-synonym $('b)\ pclass = \langle('b)\ ppoint\ set\rangle$

function *proj-add-class* :: $('a)\ pclass \Rightarrow ('a)\ pclass \Rightarrow ('a)\ pclass\ set$
where
$proj\text{-}add\text{-}class\ c_1\ c_2 =$
$(proj\text{-}add\ `\ \{(((x_1,\ y_1),\ i),((x_2,\ y_2),\ j)).$
$(((x_1,\ y_1),\ i),((x_2,\ y_2),\ j)) \in c_1 \times c_2 \wedge$
$((x_1,\ y_1),\ (x_2,\ y_2)) \in e'\text{-}aff\text{-}0 \cup e'\text{-}aff\text{-}1\})\ //\ gluing$

if $c_1 \in e\text{-}proj$ **and** $c_2 \in e\text{-}proj$
sorry

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

definition *proj-addition* **where**
$proj\text{-}addition\ c_1\ c_2 = the\text{-}elem\ (proj\text{-}add\text{-}class\ c_1\ c_2)$

end

end