

curves-safe

raya

December 21, 2019

Contents

1 Affine Edwards curves	1
2 Extension	3

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 $c\ d :: 'a :: \text{ell-field}$
begin

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

definition *delta-plus* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ **where**
delta-plus $x_1\ y_1\ x_2\ y_2 = 1 + d * x_1 * y_1 * x_2 * y_2$

definition *delta-minus* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ **where**
delta-minus $x_1\ y_1\ x_2\ y_2 = 1 - d * x_1 * y_1 * x_2 * y_2$

definition *delta* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ **where**
delta $x_1\ y_1\ x_2\ y_2 = (\text{delta-plus}\ x_1\ y_1\ x_2\ y_2) *$
 $(\text{delta-minus}\ x_1\ y_1\ x_2\ y_2)$

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

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

have $\text{non-unfolded-adds}:$

$\text{delta } x1 \ y1 \ x2 \ y2 \neq 0$ **using** $\text{delta-def assms}(5,6)$ **by** auto

have $\text{simp1gx}:$

$(x1' * x3 - c * y1' * y3) * \text{delta-minus } x1 \ y1 \ x3' \ y3' *$
 $(\text{delta } x1 \ y1 \ x2 \ y2 * \text{delta } x2 \ y2 \ x3 \ y3) =$
 $((x1 * x2 - c * y1 * y2) * x3 * \text{delta-plus } x1 \ y1 \ x2 \ y2 -$
 $c * (x1 * y2 + y1 * x2) * y3 * \text{delta-minus } x1 \ y1 \ x2 \ y2) *$
 $(\text{delta-minus } x2 \ y2 \ x3 \ y3 * \text{delta-plus } x2 \ y2 \ x3 \ y3 -$
 $d * x1 * y1 * (x2 * x3 - c * y2 * y3) * (x2 * y3 + y2 * x3))$

apply($\text{rewrite } x1'\text{-expr } y1'\text{-expr } x3'\text{-expr } y3'\text{-expr}$)**+**

apply($\text{rewrite delta-minus-def}$)

apply($\text{rewrite in } - / \sqcap \text{delta-minus-def}[\text{symmetric}] \text{delta-plus-def}[\text{symmetric}]$)**+**

```

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 Deltax-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 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  $\neq$  1 t'  $\neq$  0
begin

fun ext-add :: 'a  $\times$  'a  $\Rightarrow$  'a  $\times$  'a  $\Rightarrow$  'a  $\times$  '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**

$\tau (x,y) = (1/(t*x), 1/(t*y))$

definition $\text{delta-x} :: 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ **where**

$\text{delta-x } x1 \ y1 \ x2 \ y2 = x2*y1 - x1*y2$

definition $\text{delta-y} :: 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ **where**

$\text{delta-y } x1 \ y1 \ x2 \ y2 = x1*x2 + y1*y2$

definition delta' $:: 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ **where**

$\text{delta' } x1 \ y1 \ x2 \ y2 = \text{delta-x } x1 \ y1 \ x2 \ y2 * \text{delta-y } x1 \ y1 \ x2 \ y2$

definition e' **where** $e' \ x \ y = x^2 + y^2 - 1 - t^2 * x^2 * y^2$

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

definition $\text{gluing} :: ((('a \times 'a) \times \text{bit}) \times ((('a \times 'a) \times \text{bit})))$ **set where**

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

$((x0,y0) \in e'\text{-aff} \wedge (x1,y1) \in e'\text{-aff}) \wedge$

$((x0 \neq 0 \wedge y0 \neq 0 \wedge (x1,y1) = \tau (x0,y0) \wedge j = l+1) \vee$

$(x0 = x1 \wedge y0 = y1 \wedge l = j))\}$

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 $\text{ext-add } (x1,y1) \ (x2,y2) = \text{add } (x1,y1) \ (x2,y2)$

sorry

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

function $\text{proj-add} :: 'a \text{ ppoint} \times 'a \text{ ppoint} \Rightarrow 'a \text{ ppoint}$ **where**

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

if $\text{delta } x1 \ y1 \ x2 \ y2 \neq 0 \wedge (x1, y1) \in e'\text{-aff} \wedge (x2, y2) \in e'\text{-aff}$

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

if $\text{delta' } x1 \ y1 \ x2 \ y2 \neq 0 \wedge (x1, y1) \in e'\text{-aff} \wedge (x2, y2) \in e'\text{-aff}$

sorry

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

definition $e'\text{-aff-0}$ **where**

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

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

$\text{delta } x1 \ y1 \ x2 \ y2 \neq 0 \}$

definition $e'\text{-aff-1}$ **where**

$e'\text{-aff-1} = \{((x1,y1),(x2,y2)). (x1,y1) \in e'\text{-aff} \wedge$

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

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

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

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

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

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

function $\text{proj-add-class} :: ('a) \text{ pclass} \Rightarrow ('a) \text{ pclass} \Rightarrow ('a) \text{ pclass set}$
where

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

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**

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

end

end