

Schönhage-Strassen Multiplication on Integers

Jakob Schulz

February 6, 2026

Abstract

We give a verified implementation of the Schönhage-Strassen Multiplication on Integers based on the original paper by Schönhage and Strassen [3] and verify its asymptotic complexity of $\mathcal{O}(n \log n \log \log n)$ bit operations.

Integers are represented as LSBF (least significant bit first) boolean lists. The running time is verified using the Time Monad defined in [2]. For verifying correctness, we adapt the formalization of Number Theoretic Transforms (NTTs) by Ammer and Kreuzer [1] to the context of rings that need not be fields.

Contents

1	Preliminaries	2
1.1	Some Running Time Formalizations	4
1.2	Auxiliary Lemmas for Landau Notation	6
1.3	Multiplicative Subgroups	8
1.4	Additive Subgroups	10
2	Number Theoretic Transforms in Rings	10
2.1	Roots of Unity	11
2.2	Primitive Roots	13
2.3	Number Theoretic Transforms	13
2.3.1	Inversion Rule	14
2.3.2	Convolution Theorem	15
2.4	Fast Number Theoretic Transforms in Rings	16
3	The Schoenhage-Strassen Algorithm	20
3.1	Representing \mathbb{Z}_{2^n}	20
3.2	Representing \mathbb{Z}_{F_n}	23
3.3	Implementing FNTT in \mathbb{Z}_{F_n}	29
3.4	Final Preparations	40
3.4.1	A special residue problem	46
3.4.2	Subroutine for combining the final result	47

3.5	Schoenhage-Strassen Multiplication in \mathbb{Z}_{F_m}	48
3.6	Schoenhage-Strassen Multiplication in \mathbb{N}	52
4	Running Time Formalization	53
4.1	Multiplication in \mathbb{N}	60

1 Preliminaries

theory *Schoenhage-Strassen-Preliminaries*

imports

Main

HOL-Library.FuncSet

Karatsuba.Karatsuba-Preliminaries

Karatsuba.Nat-LSBF

begin

lemmas *cong-def = unique-euclidean-semiring-class.cong-def*

lemma *two-pow-pos: (2 :: nat) ^ x > 0*
<proof>

lemma *length-take-cobounded1: length (take n xs) ≤ n*
<proof>

lemma *const-diff-mod-idem:*
assumes $m \geq (n :: nat)$
 $f = (\lambda i. (m - i) \bmod n)$
shows $(\bigwedge i. i \in \{0..<n\} \implies f (f i) = i)$
<proof>

lemma *const-diff-mod-bij: $m \geq (n :: nat) \implies \text{bij-betw } (\lambda i. (m - i) \bmod n) \{0..<n\} \{0..<n\}$*
<proof>

lemma *const-add-mod-bij: $\text{bij-betw } (\lambda i. ((m :: nat) + i) \bmod n) \{0..<n\} \{0..<n\}$*
<proof>

lemma *mod-diff-add-eq: $(a - b + c) \bmod (m :: int) = (a \bmod m - b \bmod m + c \bmod m) \bmod m$*
<proof>

lemma *set-map-subseteqI:*
assumes $\bigwedge x. x \in A \implies f x \in B$
assumes $\text{set } xs \subseteq A$
shows $\text{set } (\text{map } f xs) \subseteq B$
<proof>

lemma *in-set-conv-nth-map2:*

assumes $z \in \text{set } (\text{map2 } f \text{ } xs \text{ } ys)$
shows $\exists i. i < \min (\text{length } xs) (\text{length } ys) \wedge z = f (xs ! i) (ys ! i)$
 $\langle \text{proof} \rangle$

lemma *set-map2*:
assumes $z \in \text{set } (\text{map2 } f \text{ } xs \text{ } ys)$
shows $\exists x y. x \in \text{set } xs \wedge y \in \text{set } ys \wedge z = f x y$
 $\langle \text{proof} \rangle$

lemma *set-map2-subseteqI*:
assumes $\bigwedge x y. x \in A \implies y \in B \implies f x y \in C$
assumes $\text{set } xs \subseteq A \text{ set } ys \subseteq B$
shows $\text{set } (\text{map2 } f \text{ } xs \text{ } ys) \subseteq C$
 $\langle \text{proof} \rangle$

lemma *in-set-conv-nth-map3*:
assumes $w \in \text{set } (\text{map3 } f \text{ } xs \text{ } ys \text{ } zs)$
shows $\exists i. i < \min (\min (\text{length } xs) (\text{length } ys)) (\text{length } zs) \wedge w = f (xs ! i) (ys ! i) (zs ! i)$
 $\langle \text{proof} \rangle$

lemma *set-map3*:
assumes $w \in \text{set } (\text{map3 } f \text{ } xs \text{ } ys \text{ } zs)$
shows $\exists x y z. x \in \text{set } xs \wedge y \in \text{set } ys \wedge z \in \text{set } zs \wedge w = f x y z$
 $\langle \text{proof} \rangle$

lemma *set-map3-subseteqI*:
assumes $\bigwedge x y z. x \in A \implies y \in B \implies z \in C \implies f x y z \in D$
assumes $\text{set } xs \subseteq A \text{ set } ys \subseteq B \text{ set } zs \subseteq C$
shows $\text{set } (\text{map3 } f \text{ } xs \text{ } ys \text{ } zs) \subseteq D$
 $\langle \text{proof} \rangle$

lemma *map3-compose3*: $\text{map3 } (\lambda x y z. f x y (g z)) \text{ } xs \text{ } ys \text{ } zs = \text{map3 } f \text{ } xs \text{ } ys (\text{map } g \text{ } zs)$
 $\langle \text{proof} \rangle$

definition *rotate-left* :: $\text{nat} \Rightarrow 'a \text{ list} \Rightarrow 'a \text{ list}$ **where**
 $\text{rotate-left } k \text{ } xs = (\text{let } (xs1, xs2) = \text{split-at } (k \text{ mod } \text{length } xs) \text{ } xs \text{ in } xs2 @ xs1)$

lemma *rotate-left-rotate[simp]*: $\text{rotate-left } k \text{ } xs = \text{rotate } k \text{ } xs$
 $\langle \text{proof} \rangle$

definition *rotate-right* **where**
 $\text{rotate-right } k \text{ } xs = \text{rotate-left } (\text{length } xs - (k \text{ mod } \text{length } xs)) \text{ } xs$

lemma *length-rotate-right[simp]*: $\text{length } (\text{rotate-right } k \text{ } xs) = \text{length } xs$
 $\langle \text{proof} \rangle$

lemma *rotate-right-rotate*[simp]: $\text{rotate-right } k (\text{rotate } k \text{ } xs) = xs$
 ⟨proof⟩

lemma *rotate-rotate-right*[simp]: $\text{rotate } k (\text{rotate-right } k \text{ } xs) = xs$
 ⟨proof⟩

value *rotate* 5 [1::nat..<8]

value *rotate-right* 3 [True, False, False]

lemma *rotate-right-append*: $\text{rotate-right } (\text{length } q) (l @ q) = q @ l$
 ⟨proof⟩

lemma *rotate-right-conv-mod*: $\text{rotate-right } n \text{ } xs = \text{rotate-right } (n \bmod \text{length } xs) \text{ } xs$
 ⟨proof⟩

lemma *mod-diff-right-eq-nat*:

assumes $(a::\text{nat}) \geq b$

shows $(a - b) \bmod m = (a - (b \bmod m)) \bmod m$

⟨proof⟩

lemma *rotate-right k (rotate-right l xs) = rotate-right (k + l) xs*
 ⟨proof⟩

lemma *nth-rotate-right*: $n < \text{length } xs \implies m < \text{length } xs \implies \text{rotate-right } m \text{ } xs !$
 $n = xs ! ((n + \text{length } xs - m) \bmod \text{length } xs)$
 ⟨proof⟩

end

1.1 Some Running Time Formalizations

theory *Schoenhage-Strassen-Runtime-Preliminaries*

imports

Main

Karatsuba.Time-Monad-Extended

Karatsuba.Main-TM

Karatsuba.Karatsuba-Preliminaries

Karatsuba.Nat-LSBF

Karatsuba.Nat-LSBF-TM

Karatsuba.Estimation-Method

Schoenhage-Strassen-Preliminaries

Akra-Bazzi.Akra-Bazzi

HOL-Library.Landau-Symbols

begin

fun *zip-tm* :: 'a list \Rightarrow 'b list \Rightarrow ('a \times 'b) list tm **where**

zip-tm xs [] =1 return []

| *zip-tm* [] ys =1 return []

| *zip-tm* (x # xs) (y # ys) =1 do { rs \leftarrow *zip-tm* xs ys; return ((x, y) # rs) }

lemma *val- zip-tm* [simp, val-simp]: $\text{val } (\text{zip-tm } xs \ ys) = \text{zip } xs \ ys$
 ⟨proof⟩

lemma *time- zip-tm* [simp]: $\text{time } (\text{zip-tm } xs \ ys) = \min (\text{length } xs) (\text{length } ys) + 1$
 ⟨proof⟩

fun *map3-tm* :: ('a \Rightarrow 'b \Rightarrow 'c \Rightarrow 'd tm) \Rightarrow 'a list \Rightarrow 'b list \Rightarrow 'c list \Rightarrow 'd list tm
where
map3-tm f (x # xs) (y # ys) (z # zs) = 1 do {
 r \leftarrow f x y z;
 rs \leftarrow *map3-tm* f xs ys zs;
 return (r # rs)
 }
 | *map3-tm* f - - - = 1 return []

lemma *val- map3-tm* [simp, val-simp]: $\text{val } (\text{map3-tm } f \ xs \ ys \ zs) = \text{map3 } (\lambda x \ y \ z. \text{val } (f \ x \ y \ z)) \ xs \ ys \ zs$
 ⟨proof⟩

lemma *time- map3-tm -bounded*:
assumes $\bigwedge x \ y \ z. x \in \text{set } xs \Longrightarrow y \in \text{set } ys \Longrightarrow z \in \text{set } zs \Longrightarrow \text{time } (f \ x \ y \ z) \leq c$
shows $\text{time } (\text{map3-tm } f \ xs \ ys \ zs) \leq (c + 1) * \min (\min (\text{length } xs) (\text{length } ys)) (\text{length } zs) + 1$
 ⟨proof⟩

fun *map4-tm* :: ('a \Rightarrow 'b \Rightarrow 'c \Rightarrow 'd \Rightarrow 'e tm) \Rightarrow 'a list \Rightarrow 'b list \Rightarrow 'c list \Rightarrow 'd list \Rightarrow 'e list tm **where**
map4-tm f (x # xs) (y # ys) (z # zs) (w # ws) = 1 do {
 r \leftarrow f x y z w;
 rs \leftarrow *map4-tm* f xs ys zs ws;
 return (r # rs)
 }
 | *map4-tm* f - - - - = 1 return []

lemma *val- map4-tm* [simp, val-simp]: $\text{val } (\text{map4-tm } f \ xs \ ys \ zs \ ws) = \text{map4 } (\lambda x \ y \ z \ w. \text{val } (f \ x \ y \ z \ w)) \ xs \ ys \ zs \ ws$
 ⟨proof⟩

lemma *time- map4-tm -bounded*:
assumes $\bigwedge x \ y \ z \ w. x \in \text{set } xs \Longrightarrow y \in \text{set } ys \Longrightarrow z \in \text{set } zs \Longrightarrow w \in \text{set } ws \Longrightarrow \text{time } (f \ x \ y \ z \ w) \leq c$
shows $\text{time } (\text{map4-tm } f \ xs \ ys \ zs \ ws) \leq (c + 1) * \min (\min (\min (\text{length } xs) (\text{length } ys)) (\text{length } zs)) (\text{length } ws) + 1$
 ⟨proof⟩

definition *map2-tm* **where**
map2-tm f xs ys = 1 do {
 xys \leftarrow *zip-tm* xs ys;

```

  map-tm ( $\lambda(x,y). f x y$ ) xs
}

```

lemma *val-map2-tm[simp, val-simp]*: $val (map2\text{-tm } f \text{ } xs \text{ } ys) = map2 (\lambda x y. val (f x y)) xs ys$
 ⟨proof⟩

lemma *time-map2-tm-bounded*:
 assumes $length \text{ } xs = length \text{ } ys$
 assumes $\bigwedge x y. x \in set \text{ } xs \implies y \in set \text{ } ys \implies time (f x y) \leq c$
 shows $time (map2\text{-tm } f \text{ } xs \text{ } ys) \leq (c + 2) * length \text{ } xs + 3$
 ⟨proof⟩

definition *rotate-left-tm* :: $nat \Rightarrow 'a \text{ list} \Rightarrow 'a \text{ list tm}$ **where**
rotate-left-tm $k \text{ } xs = 1$ do {
 lenxs \leftarrow length-tm xs;
 kmod \leftarrow $k \text{ mod}_t \text{ lenxs}$;
 (xs1, xs2) \leftarrow split-at-tm kmod xs;
 xs2 @_t xs1
}

lemma *val-rotate-left-tm[simp, val-simp]*: $val (rotate\text{-left}\text{-tm } k \text{ } xs) = rotate\text{-left } k \text{ } xs$
 ⟨proof⟩

lemma *time-rotate-left-tm-le*: $time (rotate\text{-left}\text{-tm } k \text{ } xs) \leq 13 + 14 * max \text{ } k (length \text{ } xs)$
 ⟨proof⟩

definition *rotate-right-tm* :: $nat \Rightarrow 'a \text{ list} \Rightarrow 'a \text{ list tm}$ **where**
rotate-right-tm $k \text{ } xs = 1$ do {
 lenxs \leftarrow length-tm xs;
 kmod \leftarrow $k \text{ mod}_t \text{ lenxs}$;
 rk \leftarrow $lenxs -_t \text{ kmod}$;
 rotate-left-tm rk xs
}

lemma *val-rotate-right-tm[simp, val-simp]*: $val (rotate\text{-right}\text{-tm } k \text{ } xs) = rotate\text{-right } k \text{ } xs$
 ⟨proof⟩

lemma *time-rotate-right-tm-le*: $time (rotate\text{-right}\text{-tm } k \text{ } xs) \leq 23 + 26 * max \text{ } k (length \text{ } xs)$
 ⟨proof⟩

1.2 Auxiliary Lemmas for Landau Notation

lemma *eventually-early-nat*:
 fixes $f g :: nat \Rightarrow nat$

assumes $f \in O(g)$
assumes $\bigwedge x. x \geq n0 \implies g\ x > 0$
shows $\exists c. (\forall x. x \geq n0 \implies f\ x \leq c * g\ x)$
 $\langle proof \rangle$

lemma eventually-early-real:
fixes $f\ g :: nat \Rightarrow real$
assumes $f \in O(g)$
assumes $\bigwedge x. x \geq n0 \implies f\ x \geq 0 \wedge g\ x \geq 1$
shows $\exists c. (\forall x \geq n0. f\ x \leq c * g\ x)$
 $\langle proof \rangle$

lemma floor-in-nat-iff: $floor\ x \in \mathbf{N} \longleftrightarrow x \geq 0$
 $\langle proof \rangle$

lemma bigo-floor:
fixes $f :: nat \Rightarrow nat$
fixes $g :: nat \Rightarrow real$
assumes $(\lambda x. real\ (f\ x)) \in O(g)$
assumes *eventually* $(\lambda x. g\ x \geq 1)$ *at-top*
shows $(\lambda x. real\ (f\ x)) \in O(\lambda x. real\ (nat\ (floor\ (g\ x))))$
 $\langle proof \rangle$

end

theory Schoenhage-Strassen-Ring-Lemmas

imports *HOL-Algebra.Ring* *HOL-Algebra.Multiplicative-Group*

begin

context *cring*

begin

lemma diff-diff:
assumes $a \in carrier\ R\ b \in carrier\ R\ c \in carrier\ R$
shows $a \ominus (b \ominus c) = a \ominus b \oplus c$
 $\langle proof \rangle$

lemma minus-eq-mult-one:

assumes $a \in carrier\ R$

shows $\ominus a = (\ominus \mathbf{1}) \otimes a$

$\langle proof \rangle$

lemma diff-eq-add-mult-one:

assumes $a \in carrier\ R\ b \in carrier\ R$

shows $a \ominus b = a \oplus (\ominus \mathbf{1}) \otimes b$

$\langle proof \rangle$

lemma minus-cancel:

assumes $a \in carrier\ R\ b \in carrier\ R$

shows $a \ominus b \oplus b = a$

$\langle proof \rangle$

lemma assoc4:

assumes $a \in carrier\ R\ b \in carrier\ R\ c \in carrier\ R\ d \in carrier\ R$

```

  shows  $a \otimes (b \otimes (c \otimes d)) = a \otimes b \otimes c \otimes d$ 
  <proof>
lemma diff-sum:
  assumes  $a \in \text{carrier } R$   $b \in \text{carrier } R$   $c \in \text{carrier } R$   $d \in \text{carrier } R$ 
  shows  $(a \ominus c) \oplus (b \ominus d) = (a \oplus b) \ominus (c \oplus d)$ 
  <proof>

end

lemma (in ring) inv-cancel-left:
  assumes  $x \in \text{carrier } R$ 
  assumes  $y \in \text{carrier } R$ 
  assumes  $z \in \text{Units } R$ 
  assumes  $x = z \otimes y$ 
  shows  $\text{inv } z \otimes x = y$ 
  <proof>

lemma (in ring) r-distr-diff:
  assumes  $x \in \text{carrier } R$ 
  assumes  $y \in \text{carrier } R$ 
  assumes  $z \in \text{carrier } R$ 
  shows  $x \otimes (y \ominus z) = x \otimes y \ominus x \otimes z$ 
  <proof>

lemma (in group)
  assumes  $x \in \text{carrier } G$ 
  shows  $\bigwedge i. i \in \{1..<\text{ord } x\} \implies x [i] \neq \mathbf{1}$ 
  <proof>

1.3 Multiplicative Subgroups

locale multiplicative-subgroup = cring +
  fixes X
  fixes M
  assumes Units-subset:  $X \subseteq \text{Units } R$ 
  assumes M-def:  $M = (\text{carrier} = X, \text{monoid.mult} = (\otimes), \text{one} = \mathbf{1})$ 
  assumes M-group: group M
begin

lemma carrier-M[simp]: carrier M = X <proof>

lemma one-eq:  $\mathbf{1}_M = \mathbf{1}$  <proof>

lemma mult-eq:  $a \otimes_M b = a \otimes b$  <proof>

lemma inv-eq:
  assumes  $x \in X$ 
  shows  $\text{inv}_M x = \text{inv } x$ 
  <proof>

```

lemma *nat-pow-eq*: $x \text{ [}\wedge\text{]}_M (m :: \text{nat}) = x \text{ [}\wedge\text{]} m$
<proof>

lemma *int-pow-eq*:
 assumes $x \in X$
 shows $x \text{ [}\wedge\text{]}_M (i :: \text{int}) = x \text{ [}\wedge\text{]} i$
<proof>

end

context *cring*
begin

interpretation *units-group*: *group units-of R*
<proof>

lemma *units-subgroup*: *multiplicative-subgroup R (Units R) (units-of R)*
<proof>

interpretation *units-subgroup*: *multiplicative-subgroup R Units R units-of R*
<proof>

lemma *inv-nat-pow*:
 assumes $a \in \text{Units } R$
 shows $\text{inv } (a \text{ [}\wedge\text{]} (b :: \text{nat})) = \text{inv } a \text{ [}\wedge\text{]} b$
<proof>

lemma *int-pow-mult*:
 fixes $m1\ m2 :: \text{int}$
 assumes $x \in \text{Units } R$
 shows $x \text{ [}\wedge\text{]} m1 \otimes x \text{ [}\wedge\text{]} m2 = x \text{ [}\wedge\text{]} (m1 + m2)$
<proof>

lemma *int-pow-pow*:
 fixes $m1\ m2 :: \text{int}$
 assumes $x \in \text{Units } R$
 shows $(x \text{ [}\wedge\text{]} m1) \text{ [}\wedge\text{]} m2 = x \text{ [}\wedge\text{]} (m1 * m2)$
<proof>

lemma *int-pow-one*:
 1 $\text{ [}\wedge\text{]} (i :: \text{int}) = \mathbf{1}$
<proof>

lemma *int-pow-closed*:
 assumes $x \in \text{Units } R$
 shows $x \text{ [}\wedge\text{]} (i :: \text{int}) \in \text{Units } R$
<proof>

lemma *units-of-int-pow*: $\mu \in \text{Units } R \implies \mu \text{ [}\wedge\text{]}_{(\text{units-of } R)} i = \mu \text{ [}\wedge\text{]} (i :: \text{int})$
<proof>

lemma *units-int-pow-neg*: $\mu \in \text{Units } R \implies (\text{inv } \mu) \text{ [}\wedge\text{]} (n :: \text{int}) = \mu \text{ [}\wedge\text{]} (- n)$

<proof>

lemma *units-inv-int-pow*: $\mu \in \text{Units } R \implies \text{inv } \mu = \mu [\uparrow] (- (1 :: \text{int}))$
<proof>

lemma *inv-prod*: $\mu \in \text{Units } R \implies \nu \in \text{Units } R \implies \text{inv } (\mu \otimes \nu) = \text{inv } \nu \otimes \text{inv } \mu$
<proof>

lemma *powers-of-negative*:
 fixes $r :: \text{nat}$
 assumes $x \in \text{carrier } R$
 shows $\text{even } r \implies (\ominus x) [\uparrow] r = x [\uparrow] r$ $\text{odd } r \implies (\ominus x) [\uparrow] r = \ominus (x [\uparrow] r)$
 <proof>

end

1.4 Additive Subgroups

locale *additive-subgroup* = *cring* +
 fixes X
 fixes M
 assumes *Units-subset*: $X \subseteq \text{carrier } R$
 assumes *M-def*: $M = (\text{carrier} = X, \text{monoid.mult} = (\oplus), \text{one} = \mathbf{0})$
 assumes *M-group*: *group* M
begin

lemma *carrier-M[simp]*: $\text{carrier } M = X$
<proof>

lemma *one-eq*: $\mathbf{1}_M = \mathbf{0}$ *<proof>*

lemma *mult-eq*: $a \otimes_M b = a \oplus b$
<proof>

lemma *inv-eq*:
 assumes $a \in X$
 shows $\text{inv}_M a = \ominus a$
 <proof>

end

end

2 Number Theoretic Transforms in Rings

theory *NTT-Rings*
imports
 Number-Theoretic-Transform.NTT
 Karatsuba.Monoid-Sums

Karatsuba.Karatsuba-Preliminaries
../Preliminaries/Schoenhage-Strassen-Preliminaries
../Preliminaries/Schoenhage-Strassen-Ring-Lemmas
begin

lemma *max-dividing-power-factorization*:

fixes $a :: \text{nat}$
assumes $a \neq 0$
assumes $k = \text{Max } \{s. p \wedge s \text{ dvd } a\}$
assumes $r = a \text{ div } (p \wedge k)$
assumes *prime* p
shows $a = r * p \wedge k$ *coprime* p r
 $\langle \text{proof} \rangle$

context *cring*
begin

interpretation *units-group*: *group units-of* R
 $\langle \text{proof} \rangle$

interpretation *units-subgroup*: *multiplicative-subgroup* R *Units* R *units-of* R
 $\langle \text{proof} \rangle$

2.1 Roots of Unity

definition *root-of-unity* $:: \text{nat} \Rightarrow 'a \Rightarrow \text{bool}$ **where**
root-of-unity n $\mu \equiv \mu \in \text{carrier } R \wedge \mu [\wedge] n = \mathbf{1}$

lemma *root-of-unityI*[*intro*]: $\mu \in \text{carrier } R \implies \mu [\wedge] n = \mathbf{1} \implies \text{root-of-unity } n$ μ
 $\langle \text{proof} \rangle$

lemma *root-of-unityD*[*simp*]: *root-of-unity* n $\mu \implies \mu [\wedge] n = \mathbf{1}$
 $\langle \text{proof} \rangle$

lemma *root-of-unity-closed*[*simp*]: *root-of-unity* n $\mu \implies \mu \in \text{carrier } R$
 $\langle \text{proof} \rangle$

context
fixes $n :: \text{nat}$
assumes $n > 0$
begin

lemma *roots-Units*[*simp*]:
assumes *root-of-unity* n μ
shows $\mu \in \text{Units } R$
 $\langle \text{proof} \rangle$

definition *roots-of-unity-group* **where**

roots-of-unity-group \equiv (\mid carrier = $\{\mu. \text{root-of-unity } n \ \mu\}$, monoid.mult = (\otimes) , one = $\mathbf{1}$ \mid)

lemma *roots-of-unity-group-is-group*:
shows group roots-of-unity-group
 \langle proof \rangle

interpretation *root-group* : group roots-of-unity-group
 \langle proof \rangle

interpretation *root-subgroup* : multiplicative-subgroup $R \ \{\mu. \text{root-of-unity } n \ \mu\}$
roots-of-unity-group
 \langle proof \rangle

lemma *root-of-unity-inv*:
assumes root-of-unity $n \ \mu$
shows root-of-unity $n \ (\text{inv } \mu)$
 \langle proof \rangle

lemma *inv-root-of-unity*:
assumes root-of-unity $n \ \mu$
shows $\text{inv } \mu = \mu \ [\wedge] \ (n - 1)$
 \langle proof \rangle

lemma *inv-pow-root-of-unity*:
assumes root-of-unity $n \ \mu$
assumes $i \in \{1..<n\}$
shows $(\text{inv } \mu) \ [\wedge] \ i = \mu \ [\wedge] \ (n - i) \ n - i \in \{1..<n\}$
 \langle proof \rangle

lemma *root-of-unity-nat-pow-closed*:
assumes root-of-unity $n \ \mu$
shows root-of-unity $n \ (\mu \ [\wedge] \ (m :: \text{nat}))$
 \langle proof \rangle

lemma *root-of-unity-powers*:
assumes root-of-unity $n \ \mu$
shows $\mu \ [\wedge] \ i = \mu \ [\wedge] \ (i \bmod n)$
 \langle proof \rangle

lemma *root-of-unity-powers-modint*:
assumes root-of-unity $n \ \mu$
shows $\mu \ [\wedge] \ (i :: \text{int}) = \mu \ [\wedge] \ (i \bmod \text{int } n)$
 \langle proof \rangle

lemma *root-of-unity-powers-nat*:
assumes root-of-unity $n \ \mu$
assumes $i \bmod n = j \bmod n$
shows $\mu \ [\wedge] \ i = \mu \ [\wedge] \ j$

<proof>

lemma *root-of-unity-powers-int*:
 assumes *root-of-unity* $n \ \mu$
 assumes $i \bmod \text{int } n = j \bmod \text{int } n$
 shows $\mu [\wedge] i = \mu [\wedge] j$
 <proof>

end

2.2 Primitive Roots

definition *primitive-root* :: $\text{nat} \Rightarrow 'a \Rightarrow \text{bool}$ **where**
 $\text{primitive-root } n \ \mu \equiv \text{root-of-unity } n \ \mu \wedge (\forall i \in \{1..<n\}. \ \mu [\wedge] i \neq \mathbf{1})$

lemma *primitive-rootI*[intro]:
 assumes $\mu \in \text{carrier } R$
 assumes $\mu [\wedge] n = \mathbf{1}$
 assumes $\bigwedge i. i > 0 \implies i < n \implies \mu [\wedge] i \neq \mathbf{1}$
 shows *primitive-root* $n \ \mu$
 <proof>

lemma *primitive-root-is-root-of-unity*[simp]: *primitive-root* $n \ \mu \implies \text{root-of-unity } n \ \mu$
<proof>

lemma *primitive-root-recursion*:
 assumes *even* n
 assumes *primitive-root* $n \ \mu$
 shows *primitive-root* $(n \text{ div } 2) (\mu [\wedge] (2 :: \text{nat}))$
 <proof>

lemma *primitive-root-inv*:
 assumes $n > 0$
 assumes *primitive-root* $n \ \mu$
 shows *primitive-root* $n (\text{inv } \mu)$
 <proof>

2.3 Number Theoretic Transforms

definition *NTT* :: $'a \Rightarrow 'a \text{ list} \Rightarrow 'a \text{ list}$ **where**
 $\text{NTT } \mu \ a \equiv \text{let } n = \text{length } a \text{ in } [\bigoplus j \leftarrow [0..<n]. (a ! j) \otimes (\mu [\wedge] i) [\wedge] j. i \leftarrow [0..<n]]$

lemma *NTT-length*[simp]: $\text{length } (\text{NTT } \mu \ a) = \text{length } a$
<proof>

lemma *NTT-nth*:
 assumes $\text{length } a = n$
 assumes $i < n$

shows $NTT \mu a ! i = (\bigoplus j \leftarrow [0..<n]. (a ! j) \otimes (\mu [\uparrow] i) [\uparrow] j)$
 ⟨proof⟩

lemma *NTT-nth-2*:

assumes $length\ a = n$

assumes $i < n$

assumes $\mu \in carrier\ R$

shows $NTT \mu a ! i = (\bigoplus j \leftarrow [0..<n]. (a ! j) \otimes (\mu [\uparrow] (i * j)))$
 ⟨proof⟩

lemma *NTT-nth-closed*:

assumes $set\ a \subseteq carrier\ R$

assumes $\mu \in carrier\ R$

assumes $length\ a = n$

assumes $i < n$

shows $NTT \mu a ! i \in carrier\ R$

⟨proof⟩

lemma *NTT-closed*:

assumes $set\ a \subseteq carrier\ R$

assumes $\mu \in carrier\ R$

shows $set\ (NTT \mu a) \subseteq carrier\ R$

⟨proof⟩

lemma *primitive-root 1 1*

⟨proof⟩

lemma $(\ominus \mathbf{1}) [\uparrow] (2::nat) = \mathbf{1}$

⟨proof⟩

lemma $\mathbf{1} \oplus \mathbf{1} \neq \mathbf{0} \implies primitive_root\ 2\ (\ominus \mathbf{1})$

⟨proof⟩

2.3.1 Inversion Rule

theorem *inversion-rule*:

fixes $\mu :: 'a$

fixes $n :: nat$

assumes $n > 0$

assumes *primitive-root* $n\ \mu$

assumes *good*: $\bigwedge i. i \in \{1..<n\} \implies (\bigoplus j \leftarrow [0..<n]. (\mu [\uparrow] i) [\uparrow] j) = \mathbf{0}$

assumes_[simp]: $length\ a = n$

assumes_[simp]: $set\ a \subseteq carrier\ R$

shows $NTT\ (inv\ \mu)\ (NTT\ \mu\ a) = map\ (\lambda x. nat_embedding\ n\ \otimes\ x)\ a$

⟨proof⟩

lemma *inv-good*:

assumes $n > 0$

assumes *primitive-root* $n\ \mu$

assumes *good*: $\bigwedge i. i \in \{1..<n\} \implies (\bigoplus j \leftarrow [0..<n]. (\mu [\uparrow] i) [\uparrow] j) = \mathbf{0}$

shows *primitive-root* n (*inv* μ)
 $\bigwedge i. i \in \{1..<n\} \implies (\bigoplus j \leftarrow [0..<n]. ((\text{inv } \mu) [\uparrow] i) [\uparrow] j) = \mathbf{0}$
 $\langle \text{proof} \rangle$

lemma *inv-halfway-property*:
assumes $\mu \in \text{Units } R$
assumes $\mu [\uparrow] (i::\text{nat}) = \ominus \mathbf{1}$
shows $(\text{inv } \mu) [\uparrow] i = \ominus \mathbf{1}$
 $\langle \text{proof} \rangle$

lemma *sufficiently-good-aux*:
assumes *primitive-root* m η
assumes $m = 2 \wedge j$
assumes $\eta [\uparrow] (m \text{ div } 2) = \ominus \mathbf{1}$
assumes *odd* r
assumes $r * 2 \wedge k < m$
shows $(\bigoplus l \leftarrow [0..<m]. (\eta [\uparrow] (r * 2 \wedge k)) [\uparrow] l) = \mathbf{0}$
 $\langle \text{proof} \rangle$

lemma *sufficiently-good*:
assumes *primitive-root* n μ
assumes $\text{domain } R \vee (n = 2 \wedge k \wedge \mu [\uparrow] (n \text{ div } 2) = \ominus \mathbf{1})$
shows *good*: $\bigwedge i. i \in \{1..<n\} \implies (\bigoplus j \leftarrow [0..<n]. (\mu [\uparrow] i) [\uparrow] j) = \mathbf{0}$
 $\langle \text{proof} \rangle$

corollary *inversion-rule-inv*:
fixes $\mu :: 'a$
fixes $n :: \text{nat}$
assumes $n > 0$
assumes *primitive-root* n μ
assumes *good*: $\bigwedge i. i \in \{1..<n\} \implies (\bigoplus j \leftarrow [0..<n]. (\mu [\uparrow] i) [\uparrow] j) = \mathbf{0}$
assumes[*simp*]: *length* $a = n$
assumes[*simp*]: *set* $a \subseteq \text{carrier } R$
shows $\text{NTT } \mu (\text{NTT } (\text{inv } \mu) a) = \text{map } (\lambda x. \text{nat-embedding } n \otimes x) a$
 $\langle \text{proof} \rangle$

2.3.2 Convolution Theorem

lemma *root-of-unity-power-sum-product*:
assumes *root-of-unity* n x
assumes[*simp*]: $\bigwedge i. i < n \implies f i \in \text{carrier } R$
assumes[*simp*]: $\bigwedge i. i < n \implies g i \in \text{carrier } R$
shows $(\bigoplus i \leftarrow [0..<n]. f i \otimes x [\uparrow] i) \otimes (\bigoplus i \leftarrow [0..<n]. g i \otimes x [\uparrow] i) =$
 $(\bigoplus k \leftarrow [0..<n]. (\bigoplus i \leftarrow [0..<n]. f i \otimes g ((n + k - i) \bmod n)) \otimes x [\uparrow] k)$
 $\langle \text{proof} \rangle$

context
fixes $n :: \text{nat}$

begin

definition *cyclic-convolution* :: 'a list \Rightarrow 'a list \Rightarrow 'a list (**infixl** \star 70) **where**
cyclic-convolution a b \equiv $[(\bigoplus \sigma \leftarrow [0..<n]. (a ! \sigma \otimes b ! ((n + i - \sigma) \bmod n)))] . i$
 $\leftarrow [0..<n]$

lemma *cyclic-convolution-length[simp]*:
 $length (a \star b) = n$ \langle proof \rangle

lemma *cyclic-convolution-nth*:
 $i < n \implies (a \star b) ! i = (\bigoplus \sigma \leftarrow [0..<n]. (a ! \sigma \otimes b ! ((n + i - \sigma) \bmod n)))$
 \langle proof \rangle

lemma *cyclic-convolution-closed*:
assumes $length\ a = n$ $length\ b = n$
assumes $set\ a \subseteq carrier\ R$ $set\ b \subseteq carrier\ R$
shows $set\ (a \star b) \subseteq carrier\ R$
 \langle proof \rangle

theorem *convolution-rule*:
assumes $length\ a = n$
assumes $length\ b = n$
assumes $set\ a \subseteq carrier\ R$
assumes $set\ b \subseteq carrier\ R$
assumes *root-of-unity* n μ
assumes $i < n$
shows $NTT\ \mu\ a ! i \otimes NTT\ \mu\ b ! i = NTT\ \mu\ (a \star b) ! i$
 \langle proof \rangle

end

end

end

2.4 Fast Number Theoretic Transforms in Rings

theory *FNNT-Rings*
imports *NTT-Rings Number-Theoretic-Transform.Butterfly*
begin

context *cring* **begin**

The following lemma is the essence of Fast Number Theoretic Transforms (FNNTs).

lemma *NTT-recursion*:
assumes *even* n
assumes *primitive-root* n μ
assumes[*simp*]: $length\ a = n$

assumes_[simp]: $j < n$
assumes_[simp]: $set\ a \subseteq carrier\ R$
defines $j' \equiv (if\ j < n\ div\ 2\ then\ j\ else\ j - n\ div\ 2)$
shows $j' < n\ div\ 2\ j = (if\ j < n\ div\ 2\ then\ j' else\ j' + n\ div\ 2)$
and $(NTT\ \mu\ a) ! j = (NTT\ (\mu\ [\uparrow]\ (2::nat)) [a ! i. i \leftarrow filter\ even\ [0..<n]]) ! j'$
 $\oplus \mu\ [\uparrow]\ j \otimes (NTT\ (\mu\ [\uparrow]\ (2::nat)) [a ! i. i \leftarrow filter\ odd\ [0..<n]]) ! j'$
 $\langle proof \rangle$

lemma *NTT-recursion-1*:

assumes *even* n
assumes *primitive-root* $n\ \mu$
assumes_[simp]: $length\ a = n$
assumes_[simp]: $j < n\ div\ 2$
assumes_[simp]: $set\ a \subseteq carrier\ R$
shows $(NTT\ \mu\ a) ! j =$
 $(NTT\ (\mu\ [\uparrow]\ (2::nat)) [a ! i. i \leftarrow filter\ even\ [0..<n]]) ! j$
 $\oplus \mu\ [\uparrow]\ j \otimes (NTT\ (\mu\ [\uparrow]\ (2::nat)) [a ! i. i \leftarrow filter\ odd\ [0..<n]]) ! j$
 $\langle proof \rangle$

lemma *NTT-recursion-2*:

assumes *even* n
assumes *primitive-root* $n\ \mu$
assumes_[simp]: $length\ a = n$
assumes_[simp]: $j < n\ div\ 2$
assumes_[simp]: $set\ a \subseteq carrier\ R$
assumes *halfway-property*: $\mu\ [\uparrow]\ (n\ div\ 2) = \ominus\ \mathbf{1}$
shows $(NTT\ \mu\ a) ! (n\ div\ 2 + j) =$
 $(NTT\ (\mu\ [\uparrow]\ (2::nat)) [a ! i. i \leftarrow filter\ even\ [0..<n]]) ! j$
 $\oplus \mu\ [\uparrow]\ j \otimes (NTT\ (\mu\ [\uparrow]\ (2::nat)) [a ! i. i \leftarrow filter\ odd\ [0..<n]]) ! j$
 $\langle proof \rangle$

lemma *NTT-diffs*:

assumes *even* n
assumes *primitive-root* $n\ \mu$
assumes $length\ a = n$
assumes $j < n\ div\ 2$
assumes $set\ a \subseteq carrier\ R$
assumes $\mu\ [\uparrow]\ (n\ div\ 2) = \ominus\ \mathbf{1}$
shows $NTT\ \mu\ a ! j \ominus NTT\ \mu\ a ! (n\ div\ 2 + j) = nat\text{-embedding}\ 2 \otimes (\mu\ [\uparrow]\ j$
 $\otimes NTT\ (\mu\ [\uparrow]\ (2::nat)) (map\ (!)\ a)\ (filter\ odd\ [0..<n])) ! j$
 $\langle proof \rangle$

The following algorithm is adapted from *Number-Theoretic-Transform.Butterfly*

lemma *FNTT-term-aux*_[simp]: $length\ (filter\ P\ [0..<l]) < Suc\ l$

$\langle proof \rangle$

fun *FNTT* :: 'a \Rightarrow 'a list \Rightarrow 'a list **where**

FNTT $\mu\ [] = []$

| *FNTT* $\mu\ [x] = [x]$

| *FNTT* $\mu\ [x, y] = [x \oplus y, x \ominus y]$

```

| FNTT  $\mu$  a = (let n = length a;
  nums1 = [a!i. i  $\leftarrow$  filter even [0.. $n$ ]];
  nums2 = [a!i. i  $\leftarrow$  filter odd [0.. $n$ ]];
  b = FNTT ( $\mu$  [ $\hat{\ }]$  (2::nat)) nums1;
  c = FNTT ( $\mu$  [ $\hat{\ }]$  (2::nat)) nums2;
  g = [b!i  $\oplus$  ( $\mu$  [ $\hat{\ }]$  i)  $\otimes$  c!i. i  $\leftarrow$  [0.. $(n \text{ div } 2)$ ]];
  h = [b!i  $\ominus$  ( $\mu$  [ $\hat{\ }]$  i)  $\otimes$  c!i. i  $\leftarrow$  [0.. $(n \text{ div } 2)$ ]]
  in g@h)
lemmas [simp del] = FNTT-term-aux

```

```

declare FNTT.simps[simp del]

```

```

lemma length-FNTT[simp]:
  assumes length a = 2  $\wedge$  k
  shows length (FNTT  $\mu$  a) = length a
  <proof>

```

```

theorem FNTT-NTT:
  assumes[simp]:  $\mu \in$  carrier R
  assumes n = 2  $\wedge$  k
  assumes primitive-root n  $\mu$ 
  assumes half-way-property:  $\mu$  [ $\hat{\ }]$  (n div 2) =  $\ominus$  1
  assumes[simp]: length a = n
  assumes set a  $\subseteq$  carrier R
  shows FNTT  $\mu$  a = NTT  $\mu$  a
  <proof>

```

end

The following is copied from *Number-Theoretic-Transform.Butterfly* and moved outside of the *butterfly* locale.

```

fun evens-odds where
  evens-odds - [] = []
| evens-odds True (x#xs) = (x # evens-odds False xs)
| evens-odds False (x#xs) = evens-odds True xs

```

```

lemma map-filter-shift: map f (filter even [0.. $\text{Suc } g$ ]) =
  f 0 # map ( $\lambda$  x. f (x+1)) (filter odd [0.. $g$ ])
  <proof>

```

```

lemma map-filter-shift': map f (filter odd [0.. $\text{Suc } g$ ]) =
  map ( $\lambda$  x. f (x+1)) (filter even [0.. $g$ ])
  <proof>

```

```

lemma filter-comprehension-evens-odds:
  [xs ! i. i  $\leftarrow$  filter even [0.. $\text{length } xs$ ]] = evens-odds True xs  $\wedge$ 
  [xs ! i. i  $\leftarrow$  filter odd [0.. $\text{length } xs$ ]] = evens-odds False xs
  <proof>

```

lemma *FNTT'-termination-aux[simp]*: $\text{length} (\text{evens-odds True } xs) < \text{Suc} (\text{length } xs)$

$\text{length} (\text{evens-odds False } xs) < \text{Suc} (\text{length } xs)$
 ⟨proof⟩

(End of copy)

lemma *map-evens-odds*: $\text{map } f (\text{evens-odds } x \ a) = \text{evens-odds } x (\text{map } f \ a)$
 ⟨proof⟩

lemma *length-evens-odds*:

$\text{length} (\text{evens-odds True } a) = (\text{if even } (\text{length } a) \text{ then } \text{length } a \ \text{div } 2 \ \text{else } \text{length } a \ \text{div } 2 + 1)$

$\text{length} (\text{evens-odds False } a) = \text{length } a \ \text{div } 2$
 ⟨proof⟩

lemma *set-evens-odds*:

$\text{set} (\text{evens-odds } x \ a) \subseteq \text{set } a$
 ⟨proof⟩

context *cring* **begin**

Similar to *Number-Theoretic-Transform.Butterfly*, we give an abstract algorithm that can be refined more easily to a verifiably efficient FNTT algorithm.

fun *FNTT'* :: 'a ⇒ 'a list ⇒ 'a list **where**

FNTT' μ [] = []
 | *FNTT'* μ [x] = [x]
 | *FNTT'* μ [x, y] = [x ⊕ y, x ⊖ y]
 | *FNTT'* μ a = (let n = length a;
 nums1 = evens-odds True a;
 nums2 = evens-odds False a;
 b = *FNTT'* (μ [∧] (2::nat)) nums1;
 c = *FNTT'* (μ [∧] (2::nat)) nums2;
 g = [b!i ⊕ (μ [∧] i) ⊗ c!i. i ← [0..<(n div 2)]];
 h = [b!i ⊖ (μ [∧] i) ⊗ c!i. i ← [0..<(n div 2)]]
 in g@h)

lemma *FNTT'-FNTT*: $\text{FNTT}' \ \mu \ xs = \text{FNTT} \ \mu \ xs$

⟨proof⟩

fun *FNTT''* :: 'a ⇒ 'a list ⇒ 'a list **where**

FNTT'' μ [] = []
 | *FNTT''* μ [x] = [x]
 | *FNTT''* μ [x, y] = [x ⊕ y, x ⊖ y]
 | *FNTT''* μ a = (let n = length a;
 nums1 = evens-odds True a;
 nums2 = evens-odds False a;
 b = *FNTT''* (μ [∧] (2::nat)) nums1;
 c = *FNTT''* (μ [∧] (2::nat)) nums2;

$$g = \text{map2 } (\oplus) b (\text{map2 } (\otimes) [\mu \ [\lceil] i. i \leftarrow [0..<(n \text{ div } 2)]] c);$$

$$h = \text{map2 } (\lambda x y. x \ominus y) b (\text{map2 } (\otimes) [\mu \ [\lceil] i. i \leftarrow [0..<(n \text{ div } 2)]]$$

c)

in $g@h$)

lemma *FNTT''-FNTT'*:

assumes $\text{length } a = 2 \wedge k$

shows $FNTT'' \ \mu \ a = FNTT' \ \mu \ a$

<proof>

end

end

3 The Schoenhage-Strassen Algorithm

3.1 Representing \mathbb{Z}_{2^n}

theory *Z-mod-power-of-2*

imports

Karatsuba.Nat-LSBF-TM

Finite-Fields.Ring-Characteristic

Karatsuba.Abstract-Representations-2

HOL-Number-Theory.Number-Theory

begin

context *cring* **begin**

lemma *pow-one-imp-unit*:

$(n::\text{nat}) > 0 \implies a \in \text{carrier } R \implies a \ [\lceil] n = \mathbf{1} \implies a \in \text{Units } R$

<proof>

end

definition *ensure-length* **where** $\text{ensure-length } k \ xs = \text{take } k \ (\text{fill } k \ xs)$

lemma *ensure-length-correct[simp]*: $\text{length } (\text{ensure-length } k \ xs) = k$ *<proof>*

lemma *to-nat-ensure-length*: $\text{Nat-LSBF.to-nat } xs < 2 \wedge n \implies \text{Nat-LSBF.to-nat}$

$(\text{ensure-length } n \ xs) = \text{Nat-LSBF.to-nat } xs$

<proof>

locale *int-lsbf-mod* =

fixes $k :: \text{nat}$

assumes *k-positive*: $k > 0$

begin

abbreviation n **where** $n \equiv (2::\text{nat}) \wedge k$

definition Zn **where** $Zn \equiv \text{residue-ring } (\text{int } n)$

lemma *n-positive[simp]*: $n > 0$

<proof>

sublocale *residues* n Z_n

<proof>

definition *to-residue-ring* :: *nat-lsbf* \Rightarrow *int* **where**

to-residue-ring $xs = \text{int } (\text{Nat-LSBF.to-nat } xs) \bmod \text{int } n$

lemma *to-residue-ring-in-carrier*: *to-residue-ring* $xs \in \text{carrier } Z_n$

<proof>

definition *from-residue-ring* :: *int* \Rightarrow *nat-lsbf* **where**

from-residue-ring $x = \text{fill } k (\text{Nat-LSBF.from-nat } (\text{nat } x))$

definition *reduce* **where**

reduce $xs = \text{ensure-length } k \ xs$

lemma *length-reduce*: *length* (*reduce* xs) = k

<proof>

lemma *to-nat-reduce*: *Nat-LSBF.to-nat* (*reduce* xs) = *Nat-LSBF.to-nat* $xs \bmod n$

<proof>

definition *add-mod* **where**

add-mod $x \ y = \text{reduce } (\text{add-nat } x \ y)$

definition *subtract-mod* **where**

subtract-mod $xs \ ys =$

(if *compare-nat* $xs \ ys$ *then*

reduce (*subtract-nat* (*fill* $k \ xs$) @ [*True*]) ys)

else

subtract-nat $xs \ ys$)

lemma *to-nat-add-mod*: *Nat-LSBF.to-nat* (*add-mod* $x \ y$) = (*Nat-LSBF.to-nat* x

+ *Nat-LSBF.to-nat* y) $\bmod n$

<proof>

lemma *to-nat-subtract-mod*: *length* $xs \leq k \Longrightarrow \text{length } ys \leq k \Longrightarrow \text{int } (\text{Nat-LSBF.to-nat}$

(*subtract-mod* $xs \ ys$) = (*int* (*Nat-LSBF.to-nat* xs) - *int* (*Nat-LSBF.to-nat* ys))

mod n

<proof>

lemma *length-subtract-mod*: *length* $xs \leq k \Longrightarrow \text{length } ys \leq k \Longrightarrow \text{length } (\text{subtract-mod}$

$xs \ ys) \leq k$

<proof>

lemma *add-mod-correct*: *to-residue-ring* (*add-mod* $x \ y$) = *to-residue-ring* $x \oplus_{Z_n}$

to-residue-ring y

<proof>

lemma *subtract-mod-correct*:

assumes *length x ≤ k*

assumes *length y ≤ k*

assumes *n > 1*

shows *to-residue-ring (subtract-mod x y) = to-residue-ring x ⊖_{Z_n} to-residue-ring y*

<proof>

lemma *length-from-residue-ring*: *x < 2^k ⇒ length (from-residue-ring x) = k*

<proof>

interpretation *int-lsbf-mod*: *abstract-representation-2 from-residue-ring to-residue-ring {0..<int n}*

rewrites *int-lsbf-mod.reduce = reduce*

and *int-lsbf-mod.representations = {x :: bool list. length x = k}*

<proof>

lemma *add-mod-closed*: *length (add-mod x y) = k*

<proof>

end

end

theory *Z-mod-power-of-2-TM*

imports *Z-mod-power-of-2 Karatsuba.Nat-LSBF-TM*

begin

definition *ensure-length-tm* :: *nat ⇒ nat-lsbf ⇒ nat-lsbf tm* **where**

ensure-length-tm k xs = 1 fill-tm k xs ≫≡ take-tm k

lemma *val-ensure-length-tm[simp, val-simp]*: *val (ensure-length-tm k xs) = ensure-length k xs*

<proof>

lemma *time-ensure-length-tm[simp]*: *time (ensure-length-tm k xs) = 7 + 2 * length xs + 2 * k*

<proof>

context *int-lsbf-mod*

begin

definition *reduce-tm* :: *nat-lsbf ⇒ nat-lsbf tm* **where**

reduce-tm xs = 1 ensure-length-tm k xs

lemma *val-reduce-tm[simp, val-simp]*: *val (reduce-tm xs) = reduce xs*

<proof>

lemma *time-reduce-tm[simp]*: $\text{time}(\text{reduce-tm } xs) = 8 + 2 * \text{length } xs + 2 * k$
 ⟨proof⟩

definition *add-mod-tm* :: $\text{nat-lsbf} \Rightarrow \text{nat-lsbf} \Rightarrow \text{nat-lsbf } tm$ **where**
add-mod-tm $xs\ ys = 1\ xs +_{nt}\ ys \ggg \text{reduce-tm}$

lemma *val-add-mod-tm[simp, val-simp]*: $\text{val}(\text{add-mod-tm } xs\ ys) = \text{add-mod } xs\ ys$
 ⟨proof⟩

lemma *time-add-mod-tm-le*: $\text{time}(\text{add-mod-tm } xs\ ys) \leq 14 + 4 * \max(\text{length } xs)$
 $(\text{length } ys) + 2 * k$
 ⟨proof⟩

definition *subtract-mod-tm* :: $\text{nat-lsbf} \Rightarrow \text{nat-lsbf} \Rightarrow \text{nat-lsbf } tm$ **where**
subtract-mod-tm $xs\ ys = 1$ **do** {
 $b \leftarrow xs \leq_{nt} ys$;
 if b **then** **do** {
 $\text{fillx} \leftarrow \text{fill-tm } k\ xs$;
 $\text{fillx1} \leftarrow \text{fillx} @_t [True]$;
 $\text{fillx1} -_{nt} ys \ggg \text{reduce-tm}$
 } **else** $xs -_{nt} ys$
 }

lemma *val-subtract-mod-tm[simp, val-simp]*: $\text{val}(\text{subtract-mod-tm } xs\ ys) = \text{subtract-mod } xs\ ys$
 ⟨proof⟩

lemma *time-subtract-mod-tm-le*: $\text{time}(\text{subtract-mod-tm } xs\ ys) \leq 118 + 51 * \max$
 $k(\max(\text{length } xs)(\text{length } ys))$
 ⟨proof⟩

end

end

3.2 Representing \mathbb{Z}_{F_n}

theory *Z-mod-Fermat*

imports

Z-mod-power-of-2

../NTT-Rings/FNTT-Rings

../Preliminaries/Schoenhage-Strassen-Preliminaries

Karatsuba.Estimation-Method

begin

lemma *to-nat-replicate-True2*:

assumes *Nat-LSBF.to-nat* $xs = 2^{\wedge}(\text{length } xs) - 1$

shows $xs = \text{replicate } (\text{length } xs)\ True$

⟨proof⟩

lemma *residue-ring-pow*: $n > 1 \implies a [\wedge]_{\text{residue-ring } n} b = (a \wedge b) \text{ mod } n$
 ⟨proof⟩

lemma (in *residues*) *pow-nat-eq*:
 $a [\wedge]_R (n :: \text{nat}) = a \wedge n \text{ mod } m$
 ⟨proof⟩

locale *int-lsbfermat* =
 fixes $k :: \text{nat}$
 begin

abbreviation *n where* $n \equiv (2 :: \text{nat}) \wedge (2 \wedge k) + 1$

lemma *n-positive[simp]*: $n > 0$ ⟨proof⟩

lemma *n-gt-1[simp]*: $n > 1$ ⟨proof⟩

lemma *n-gt-2[simp]*: $n > 2$
 ⟨proof⟩

definition *Fn where* $Fn \equiv \text{residue-ring } (\text{int } n)$

sublocale *residues n Fn*
 ⟨proof⟩

definition *fermat-non-unique-carrier where*
 $\text{fermat-non-unique-carrier} \equiv \{xs :: \text{nat-lsbf. length } xs = 2 \wedge (k + 1)\}$

lemma *fermat-non-unique-carrierI[intro]*:
 $\text{length } xs = 2 \wedge (k + 1) \implies xs \in \text{fermat-non-unique-carrier}$
 ⟨proof⟩

lemma *fermat-non-unique-carrierE[elim]*:
 $xs \in \text{fermat-non-unique-carrier} \implies (\text{length } xs = 2 \wedge (k + 1) \implies P) \implies P$
 ⟨proof⟩

lemma *two-pow-half-carrier-length[simp]*: $(\text{int } 2 \wedge (2 \wedge k)) \text{ mod } n = -1 \text{ mod } n$
 ⟨proof⟩

lemma *two-pow-half-carrier-length-neq-1*: $2 \wedge (2 \wedge k) \text{ mod } n \neq 1$
 ⟨proof⟩

lemma *two-pow-carrier-length[simp]*: $(2 :: \text{nat}) \wedge (2 \wedge (k + 1)) \text{ mod } n = 1$
 ⟨proof⟩

lemma *two-pow-half-carrier-length-residue-ring[simp]*:
 $(2 :: \text{int}) [\wedge]_{Fn} (2 :: \text{nat}) \wedge k = \ominus_{Fn} \mathbf{1}_{Fn}$
 ⟨proof⟩

lemma *two-pow-carrier-length-residue-ring[simp]*:

$(2::int) [\wedge]_{Fn} (2::nat) \wedge (k + 1) = \mathbf{1}_{Fn}$
 $\langle proof \rangle$

corollary *two-is-unit*: $2 \in Units Fn$
 $\langle proof \rangle$

corollary *two-in-carrier*: $2 \in carrier Fn$
 $\langle proof \rangle$

lemma *nat-mod-eqE*: $(a::nat) \bmod m = b \bmod m \implies \exists i j. a + i * m = b + j * m$
 $\langle proof \rangle$

corollary *pow-mod-carrier-length*:
assumes $(a::nat) \bmod 2 \wedge (k + 1) = b \bmod 2 \wedge (k + 1)$
shows $2 [\wedge]_{Fn} a = 2 [\wedge]_{Fn} b$
 $\langle proof \rangle$

lemma *two-powers-trivial*:
assumes $s \leq 2 \wedge k$
shows $2 [\wedge]_{Fn} s = 2 \wedge s$
 $\langle proof \rangle$

lemma *two-powers-Units*:
assumes $s \leq 2 \wedge k$
shows $2 \wedge s \in Units Fn$
 $\langle proof \rangle$

corollary *two-powers-in-carrier*:
assumes $s \leq 2 \wedge k$
shows $2 \wedge s \in carrier Fn$
 $\langle proof \rangle$

lemma *two-powers-half-carrier-length-residue-ring[simp]*:
assumes $i + s = k$
shows $(2 \wedge 2 \wedge i) [\wedge]_{Fn} (2::nat) \wedge s = \ominus_{Fn} \mathbf{1}_{Fn}$
 $\langle proof \rangle$

interpretation *z-mod-fermat-unit-group*: *group units-of Fn*
 $\langle proof \rangle$

lemma *inv-of-2[simp]*:
 $inv_{Fn} 2 = 2 [\wedge]_{Fn} ((2::nat) \wedge (k + 1) - 1)$
 $\langle proof \rangle$

lemma *inv-of-2-powers*:
assumes $s \leq 2 \wedge k$
shows $inv_{Fn} (2 \wedge s) = 2 [\wedge]_{Fn} (2 \wedge (k + 1) - s)$
 $\langle proof \rangle$

lemma *inv-pow-mod-carrier-length*:

assumes $(a::nat) \bmod 2^{k+1} = b \bmod 2^{k+1}$

shows $(\text{inv}_{Fn} 2) [\uparrow]_{Fn} a = (\text{inv}_{Fn} 2) [\uparrow]_{Fn} b$
<proof>

lemma

assumes $m > 0$

shows $\exists i j. (a::nat) = j + i * m \wedge j < m$
<proof>

corollary *two-powers*: $(2::nat)^a \bmod n = (2::nat)^{a \bmod (2^{k+1})} \bmod n$

<proof>

lemma *fermat-carrier-length[simp]*: $xs \in \text{fermat-non-unique-carrier} \implies \text{length } xs = 2^{k+1}$

<proof>

fun *to-residue-ring* :: $\text{nat-lsbf} \Rightarrow \text{int}$ **where**

to-residue-ring $xs = \text{int } (\text{Nat-LSBF.to-nat } xs) \bmod \text{int } n$

fun *from-residue-ring* :: $\text{int} \Rightarrow \text{nat-lsbf}$ **where**

from-residue-ring $x = \text{fill } (2^{k+1}) (\text{Nat-LSBF.from-nat } (\text{nat } x))$

lemma *to-residue-ring-in-carrier[simp]*: $\text{to-residue-ring } xs \in \text{carrier } Fn$

<proof>

lemma *to-residue-ring-eq-to-nat*: $\text{Nat-LSBF.to-nat } xs < n \implies \text{to-residue-ring } xs = \text{int } (\text{Nat-LSBF.to-nat } xs)$

<proof>

definition *multiply-with-power-of-2* :: $\text{nat-lsbf} \Rightarrow \text{nat} \Rightarrow \text{nat-lsbf}$ **where**

multiply-with-power-of-2 $xs m = \text{rotate-right } m xs$

definition *divide-by-power-of-2* :: $\text{nat-lsbf} \Rightarrow \text{nat} \Rightarrow \text{nat-lsbf}$ **where**

divide-by-power-of-2 $xs m = \text{rotate-left } m xs$

lemma *length-multiply-with-power-of-2[simp]*: $\text{length } (\text{multiply-with-power-of-2 } xs m) = \text{length } xs$

<proof>

lemma *length-divide-by-power-of-2[simp]*: $\text{length } (\text{divide-by-power-of-2 } xs m) = \text{length } xs$

<proof>

lemma (**in** *euclidean-semiring-cancel*) *sum-list-mod*: $(\sum i \leftarrow xs. (f i \bmod m)) \bmod m = (\sum i \leftarrow xs. f i) \bmod m$

<proof>

lemma (in *euclidean-semiring-cancel*) *sum-list-mod'*:
assumes $\bigwedge i. i \in \text{set } xs \implies f\ i \bmod m = g\ i \bmod m$
shows $(\sum i \leftarrow xs. f\ i) \bmod m = (\sum i \leftarrow xs. g\ i) \bmod m$
<proof>

lemma *multiply-with-power-of-2-correct'*: $xs \in \text{fermat-non-unique-carrier} \implies \text{Nat-LSBF.to-nat}$
 $(\text{multiply-with-power-of-2 } xs\ m) \bmod n = \text{Nat-LSBF.to-nat } xs * 2^m \bmod n \wedge$
 $\text{multiply-with-power-of-2 } xs\ m \in \text{fermat-non-unique-carrier}$
<proof>

corollary *multiply-with-power-of-2-closed*:
assumes $xs \in \text{fermat-non-unique-carrier}$
shows $\text{multiply-with-power-of-2 } xs\ m \in \text{fermat-non-unique-carrier}$
<proof>

corollary *multiply-with-power-of-2-correct*:
assumes $xs \in \text{fermat-non-unique-carrier}$
shows $\text{to-residue-ring } (\text{multiply-with-power-of-2 } xs\ m) = \text{to-residue-ring } xs \otimes_{F_n} 2^m \uparrow_{F_n} m$
<proof>

lemma
assumes $xs \in \text{fermat-non-unique-carrier}$
shows *divide-by-power-of-2-correct*: $\text{to-residue-ring } (\text{divide-by-power-of-2 } xs\ m)$
 $= \text{to-residue-ring } xs \otimes_{F_n} (\text{inv}_{F_n} 2) \uparrow_{F_n} m$
and *divide-by-power-of-2-closed*: $\text{divide-by-power-of-2 } xs\ m \in \text{fermat-non-unique-carrier}$
<proof>

definition *add-fermat where*
 $\text{add-fermat } xs\ ys = (\text{let } zs = \text{add-nat } xs\ ys \text{ in if length } zs = 2^{k+1} + 1 \text{ then}$
 $\text{inc-nat } (\text{butlast } zs) \text{ else } zs)$

lemma *add-fermat-correct'*:
assumes $xs \in \text{fermat-non-unique-carrier}$
assumes $ys \in \text{fermat-non-unique-carrier}$
shows $\text{add-fermat } xs\ ys \in \text{fermat-non-unique-carrier} \wedge \text{Nat-LSBF.to-nat } (\text{add-fermat } xs\ ys) \bmod n = (\text{Nat-LSBF.to-nat } xs + \text{Nat-LSBF.to-nat } ys) \bmod n$
<proof>

corollary *add-fermat-closed*:
assumes $xs \in \text{fermat-non-unique-carrier}$
assumes $ys \in \text{fermat-non-unique-carrier}$
shows $\text{add-fermat } xs\ ys \in \text{fermat-non-unique-carrier}$
<proof>

corollary *add-fermat-correct*:
assumes $xs \in \text{fermat-non-unique-carrier}$
assumes $ys \in \text{fermat-non-unique-carrier}$
shows $\text{to-residue-ring } (\text{add-fermat } xs\ ys) = \text{to-residue-ring } xs \oplus_{F_n} \text{to-residue-ring } ys$

ys
{proof}

definition *subtract-fermat* **where**

subtract-fermat xs ys = add-fermat xs (multiply-with-power-of-2 ys (2 ^ k))

lemma *subtract-fermat-correct'*:

assumes *xs ∈ fermat-non-unique-carrier*

assumes *ys ∈ fermat-non-unique-carrier*

shows *subtract-fermat xs ys ∈ fermat-non-unique-carrier ∧ int (Nat-LSBF.to-nat (subtract-fermat xs ys)) mod n = (int (Nat-LSBF.to-nat xs) - int (Nat-LSBF.to-nat ys)) mod n*

{proof}

corollary *subtract-fermat-closed*:

assumes *xs ∈ fermat-non-unique-carrier*

assumes *ys ∈ fermat-non-unique-carrier*

shows *subtract-fermat xs ys ∈ fermat-non-unique-carrier*

{proof}

corollary *subtract-fermat-correct*:

assumes *xs ∈ fermat-non-unique-carrier*

assumes *ys ∈ fermat-non-unique-carrier*

shows *to-residue-ring (subtract-fermat xs ys) = to-residue-ring xs ⊖_{F_n} to-residue-ring ys*

{proof}

end

context *int-lsbf-fermat* **begin**

definition *reduce* :: *nat-lsbf ⇒ nat-lsbf* **where**

reduce xs = (let (ys, zs) = split xs in

if compare-nat zs ys then

subtract-nat ys zs

else

subtract-nat (add-nat (True # replicate (2 ^ k - 1) False @ [True]) ys) zs)

lemma *reduce-correct'*:

assumes *xs ∈ fermat-non-unique-carrier*

shows *Nat-LSBF.to-nat (reduce xs) < n ∧ Nat-LSBF.to-nat (reduce xs) mod n = Nat-LSBF.to-nat xs mod n* **and** *length (reduce xs) ≤ 2 ^ k + 2*

{proof}

lemma *reduce-correct*:

assumes *xs ∈ fermat-non-unique-carrier*

shows *Nat-LSBF.to-nat xs mod n = Nat-LSBF.to-nat (reduce xs)*

{proof}

lemma *add-take-drop-carry-aux*:
assumes $xs' = \text{add-nat } (\text{take } e \text{ } xs) (\text{drop } e \text{ } xs)$
assumes $\text{length } xs = e + 1$
assumes $e \geq 1$
shows $\text{length } xs' \leq e \vee (xs' = \text{replicate } e \text{ } \text{False} @ [\text{True}] \wedge xs = \text{replicate } e \text{ } \text{True} @ [\text{True}])$
 $\langle \text{proof} \rangle$

function *from-nat-lsbf* :: $\text{nat-lsbf} \Rightarrow \text{nat-lsbf}$ **where**
from-nat-lsbf $xs = (\text{if } \text{length } xs \leq 2^{k+1} \text{ then fill } (2^{k+1}) \text{ } xs$
else from-nat-lsbf ($\text{add-nat } (\text{take } (2^{k+1}) \text{ } xs) (\text{drop } (2^{k+1}) \text{ } xs)$)
 $\langle \text{proof} \rangle$

lemma *from-nat-lsbf-dom-termination*: *All from-nat-lsbf-dom*
 $\langle \text{proof} \rangle$

termination $\langle \text{proof} \rangle$

declare *from-nat-lsbf.simps*[*simp del*]

lemma *from-nat-lsbf-correct*:
shows $\text{from-nat-lsbf } xs \in \text{fermat-non-unique-carrier}$
 $\text{to-residue-ring } (\text{from-nat-lsbf } xs) = \text{to-residue-ring } xs$
 $\langle \text{proof} \rangle$

lemma *length-from-nat-lsbf*: $\text{length } (\text{from-nat-lsbf } xs) = 2^{k+1}$
 $\langle \text{proof} \rangle$

3.3 Implementing FNTT in \mathbb{Z}_{F_n}

lemma *n-odd*: *odd n*
 $\langle \text{proof} \rangle$

lemma *ord-2*: $\text{ord } n \ 2 = 2^{k+1}$
 $\langle \text{proof} \rangle$

corollary *ord-2-int*: $\text{ord } (\text{int } n) \ 2 = 2^{k+1}$
 $\langle \text{proof} \rangle$

lemma *two-is-primitive-root*: $\text{primitive-root } (2^{k+1}) \ 2$
 $\langle \text{proof} \rangle$

lemma *two-inv-is-primitive-root*: $\text{primitive-root } (2^{k+1}) \ (\text{inv}_{F_n} \ 2)$
 $\langle \text{proof} \rangle$

lemma *two-powers-primitive-root*:
assumes $i + s = k + 1$
assumes $i \leq k$
shows $\text{primitive-root } (2^s) \ (2 \ [\overset{\wedge}{F_n}] (2::\text{nat})^i)$
 $\langle \text{proof} \rangle$

fun *fft-combine-b-c-aux* :: $(\text{nat-lsbf} \Rightarrow \text{nat-lsbf} \Rightarrow \text{nat-lsbf}) \Rightarrow (\text{nat-lsbf} \Rightarrow \text{nat} \Rightarrow$

$\text{nat-lsbf}) \Rightarrow \text{nat} \Rightarrow \text{nat-lsbf list} \times \text{nat} \Rightarrow \text{nat-lsbf list} \Rightarrow \text{nat-lsbf list} \Rightarrow \text{nat-lsbf list}$
where

$\text{fft-combine-b-c-aux } f g l (\text{revs}, e) [] [] = \text{rev revs}$
 $| \text{fft-combine-b-c-aux } f g l (\text{revs}, e) (b \# bs) (c \# cs) =$
 $\quad \text{fft-combine-b-c-aux } f g l ((f b (g c e)) \# \text{revs}, (e + l) \bmod 2^{k+1}) bs cs$
 $| \text{fft-combine-b-c-aux } f g l - - - = \text{undefined}$

fun $\text{fft-iff-combine-b-c-add}$ **where**

$\text{fft-iff-combine-b-c-add True } l bs cs = \text{fft-combine-b-c-aux add-fermat divide-by-power-of-2}$
 $l ([] , 0) bs cs$
 $| \text{fft-iff-combine-b-c-add False } l bs cs = \text{fft-combine-b-c-aux add-fermat multiply-with-power-of-2}$
 $l ([] , 0) bs cs$

fun $\text{fft-iff-combine-b-c-subtract}$ **where**

$\text{fft-iff-combine-b-c-subtract True } l bs cs = \text{fft-combine-b-c-aux subtract-fermat di-}$
 $\text{vide-by-power-of-2 } l ([] , 0) bs cs$
 $| \text{fft-iff-combine-b-c-subtract False } l bs cs = \text{fft-combine-b-c-aux subtract-fermat}$
 $\text{multiply-with-power-of-2 } l ([] , 0) bs cs$

lemma $\text{fft-combine-b-c-aux-correct}$:

assumes $\text{length } bs = \text{len-bc}$ $\text{length } cs = \text{len-bc}$
assumes $e < 2^{k+1}$
shows $\text{fft-combine-b-c-aux } f g l (\text{revs}, e) bs cs = \text{rev revs} @ \text{map3 } (\lambda x y i. f x (g$
 $y ((e + l * i) \bmod 2^{k+1}))) bs cs [0..<\text{len-bc}]$
 $\langle \text{proof} \rangle$

lemma $\text{fft-iff-combine-b-c-add-correct}$:

assumes $\text{length } bs = \text{len-bc}$ $\text{length } cs = \text{len-bc}$
shows $\text{fft-iff-combine-b-c-add it } l bs cs = \text{map3 } (\lambda x y i. \text{add-fermat } x ((\text{if it then}$
 $\text{divide-by-power-of-2 else multiply-with-power-of-2}) y ((l * i) \bmod 2^{k+1})))$
 $bs cs [0..<\text{len-bc}]$
 $\langle \text{proof} \rangle$

lemma $\text{fft-iff-combine-b-c-subtract-correct}$:

assumes $\text{length } bs = \text{len-bc}$ $\text{length } cs = \text{len-bc}$
shows $\text{fft-iff-combine-b-c-subtract it } l bs cs = \text{map3 } (\lambda x y i. \text{subtract-fermat } x$
 $((\text{if it then divide-by-power-of-2 else multiply-with-power-of-2}) y ((l * i) \bmod 2^{k+1})))$
 $bs cs [0..<\text{len-bc}]$
 $\langle \text{proof} \rangle$

lemma $\text{fft-iff-combine-b-c-add-carrier}$:

assumes $\text{length } bs = \text{len-bc}$ $\text{length } cs = \text{len-bc}$
assumes $\text{set } bs \subseteq \text{fermat-non-unique-carrier}$
assumes $\text{set } cs \subseteq \text{fermat-non-unique-carrier}$
shows $\text{set } (\text{fft-iff-combine-b-c-add it } l bs cs) \subseteq \text{fermat-non-unique-carrier}$
 $\langle \text{proof} \rangle$

lemma $\text{fft-iff-combine-b-c-subtract-carrier}$:

assumes $\text{length } bs = \text{len-bc}$ $\text{length } cs = \text{len-bc}$

assumes set bs \subseteq *fermat-non-unique-carrier*
assumes set cs \subseteq *fermat-non-unique-carrier*
shows set (*fft-iff-combine-b-c-subtract* it l bs cs) \subseteq *fermat-non-unique-carrier*
 <proof>

fun *fft-iff* :: *bool* \Rightarrow *nat* \Rightarrow *nat-lsb* list \Rightarrow *nat-lsb* list **where**
fft-iff it l [] = []
 | *fft-iff* it l [x] = [x]
 | *fft-iff* it l [x, y] = [*add-fermat* x y, *subtract-fermat* x y]
 | *fft-iff* it l a = (let *nums1* = *evens-odds* True a;
 nums2 = *evens-odds* False a;
 b = *fft-iff* it (2 * l) *nums1*;
 c = *fft-iff* it (2 * l) *nums2*;
 g = *fft-iff-combine-b-c-add* it l b c;
 h = *fft-iff-combine-b-c-subtract* it l b c
 in g@h)

fun *fft* **where** *fft* l xs = *fft-iff* False l xs
fun *iff* **where** *iff* l xs = *fft-iff* True l xs

end

locale *fft-context* = *int-lsb-fermat* +
 fixes it :: *bool*
 fixes l e :: *nat*
 fixes a1 a2 a3 :: *nat-lsb*
 fixes as :: *nat-lsb* list
 assumes *length-a'*: *length* (a1 # a2 # a3 # as) = 2 ^e
begin

definition a **where** a = a1 # a2 # a3 # as
definition *nums1* **where** *nums1* = *evens-odds* True a
definition *nums2* **where** *nums2* = *evens-odds* False a
definition b **where** b = *fft-iff* it (2 * l) *nums1*
definition c **where** c = *fft-iff* it (2 * l) *nums2*
definition g **where** g = *fft-iff-combine-b-c-add* it l b c
definition h **where** h = *fft-iff-combine-b-c-subtract* it l b c
lemmas *defs* = a-def *nums1-def* *nums2-def* b-def c-def g-def h-def

lemma *length-a*: *length* a = 2 ^e <proof>

lemma *e-ge-2*: e \geq 2

<proof>

lemma *e-pos*: e > 0 <proof>

lemma *two-pow-e-div-2*: (2::nat) ^e div 2 = 2 ^(e - 1)

<proof>

lemma *two-pow-e-as-sum*: (2::nat) ^e = 2 ^(e - 1) + 2 ^(e - 1)

<proof>

lemma
shows *length-nums1*: $\text{length } \text{nums1} = 2^{e-1}$
and *length-nums2*: $\text{length } \text{nums2} = 2^{e-1}$
<proof>

lemma *result-eq*: $\text{fft-iff } l \ a = g \ @ \ h$
<proof>

lemma
assumes *set a* \subseteq *fermat-non-unique-carrier*
shows *nums1-carrier*: $\text{set } \text{nums1} \subseteq$ *fermat-non-unique-carrier*
and *nums2-carrier*: $\text{set } \text{nums2} \subseteq$ *fermat-non-unique-carrier*
<proof>

end

context *int-lsb-fermat*
begin

lemma *length-fft-iff*:
assumes $\text{length } a = 2^e$
shows $\text{length } (\text{fft-iff } l \ a) = 2^e$
<proof>

lemma *length-fft*:
assumes $\text{length } a = 2^e$
shows $\text{length } (\text{fft } l \ a) = 2^e$
<proof>

lemma *length-iff*:
assumes $\text{length } a = 2^e$
shows $\text{length } (\text{iff } l \ a) = 2^e$
<proof>

end

context *fft-context* **begin**

lemma *length-b*: $\text{length } b = 2^{e-1}$
<proof>

lemma *length-c*: $\text{length } c = 2^{e-1}$
<proof>

lemma *length-g*: $\text{length } g = 2^{e-1}$
<proof>

lemma *length-h*: $\text{length } h = 2^{e-1}$
<proof>

end

context *int-lsbj-fermat*
begin

lemma *fft-iffit-carrier*:
assumes $\text{length } a = 2 \wedge l$
assumes $\text{set } a \subseteq \text{fermat-non-unique-carrier}$
shows $\text{set } (\text{fft-iffit } it \ s \ a) \subseteq \text{fermat-non-unique-carrier}$
 $\langle \text{proof} \rangle$

lemma *fft-carrier*:
assumes $\text{length } a = 2 \wedge l$
assumes $\text{set } a \subseteq \text{fermat-non-unique-carrier}$
shows $\text{set } (\text{fft } s \ a) \subseteq \text{fermat-non-unique-carrier}$
 $\langle \text{proof} \rangle$

lemma *iffit-carrier*:
assumes $\text{length } a = 2 \wedge l$
assumes $\text{set } a \subseteq \text{fermat-non-unique-carrier}$
shows $\text{set } (\text{iffit } s \ a) \subseteq \text{fermat-non-unique-carrier}$
 $\langle \text{proof} \rangle$

lemma *fft-iffit-correct'*:
assumes $\text{length } a = 2 \wedge l$
assumes $l \leq k + 1$
assumes $\text{set } a \subseteq \text{fermat-non-unique-carrier}$
shows $\text{map to-residue-ring } (\text{fft-iffit } it \ s \ a) = \text{FNNTT'' } ((\text{if } it \ \text{then } \text{inv}_{F_n} \ 2 \ \text{else } 2) \ [\wedge]_{F_n} \ s) \ (\text{map to-residue-ring } a)$
 $\langle \text{proof} \rangle$

lemma *fft-iffit-correct*:
assumes $\text{length } a = 2 \wedge l$
assumes $s = 2 \wedge i$
assumes $i + l = k + 1$
assumes $l > 0$
assumes $\text{set } a \subseteq \text{fermat-non-unique-carrier}$
shows $\text{map to-residue-ring } (\text{fft-iffit } it \ s \ a) = \text{NTT } ((\text{if } it \ \text{then } \text{inv}_{F_n} \ 2 \ \text{else } 2) \ [\wedge]_{F_n} \ s) \ (\text{map to-residue-ring } a)$
 $\langle \text{proof} \rangle$

lemma *fft-correct*:
assumes $\text{length } a = 2 \wedge l$
assumes $s = 2 \wedge i$
assumes $i + l = k + 1$
assumes $l > 0$
assumes $\text{set } a \subseteq \text{fermat-non-unique-carrier}$
shows $\text{map to-residue-ring } (\text{fft } s \ a) = \text{NTT } (2 \ [\wedge]_{F_n} \ s) \ (\text{map to-residue-ring } a)$
 $\langle \text{proof} \rangle$

lemma *iffit-correct*:

```

assumes length a = 2 ^ l
assumes s = 2 ^ i
assumes i + l = k + 1
assumes l > 0
assumes set a ⊆ fermat-non-unique-carrier
shows map to-residue-ring (ifft s a) = NTT ((invFn 2) [∧]Fn s) (map to-residue-ring
a)
  ⟨proof⟩

```

end

end

theory Z-mod-Fermat-TM

imports

Z-mod-Fermat

Z-mod-power-of-2-TM

../Preliminaries/Schoenhage-Strassen-Runtime-Preliminaries

begin

fun evens-odds-tm :: bool ⇒ 'a list ⇒ 'a list tm **where**

evens-odds-tm b [] = 1 return []

| evens-odds-tm True (x # xs) = 1 do {

rs ← evens-odds-tm False xs;

return (x # rs)

}

| evens-odds-tm False (x # xs) = 1 evens-odds-tm True xs

lemma val-evens-odds-tm[simp, val-simp]: val (evens-odds-tm b xs) = evens-odds
b xs

⟨proof⟩

lemma time-evens-odds-tm-le: time (evens-odds-tm b xs) ≤ length xs + 1

⟨proof⟩

context int-lsbf-fermat

begin

definition multiply-with-power-of-2-tm :: nat-lsbf ⇒ nat ⇒ nat-lsbf tm **where**

multiply-with-power-of-2-tm xs m = 1 rotate-right-tm m xs

lemma val-multiply-with-power-of-2-tm[simp, val-simp]:

val (multiply-with-power-of-2-tm xs m) = multiply-with-power-of-2 xs m

⟨proof⟩

lemma time-multiply-with-power-of-2-tm-le:

time (multiply-with-power-of-2-tm xs m) ≤ 24 + 26 * max m (length xs)

⟨proof⟩

definition divide-by-power-of-2-tm :: nat-lsbf ⇒ nat ⇒ nat-lsbf tm **where**

divide-by-power-of-2-tm xs m =1 rotate-left-tm m xs

lemma *val-divide-by-power-of-2-tm[simp, val-simp]:*

val (divide-by-power-of-2-tm xs m) = divide-by-power-of-2 xs m
 ⟨proof⟩

lemma *time-divide-by-power-of-2-tm-le:*

*time (divide-by-power-of-2-tm xs m) ≤ 24 + 26 * max m (length xs)*
 ⟨proof⟩

definition *add-fermat-tm :: nat-lsbf ⇒ nat-lsbf ⇒ nat-lsbf tm where*

add-fermat-tm xs ys =1 do {

zs ← xs +_{nt} ys;
lenzs ← length-tm zs;
k1 ← k +_t 1;
powk ← 2 ^t k1;
powk1 ← powk +_t 1;
b ← lenzs =_t powk1;
if b then do {
zsr ← butlast-tm zs;
inc-nat-tm zsr
} else return zs
}

lemma *val-add-fermat-tm[simp, val-simp]: val (add-fermat-tm xs ys) = add-fermat*

xs ys
 ⟨proof⟩

lemma *time-add-fermat-tm-le: time (add-fermat-tm xs ys) ≤ 13 + 7 * max (length*

*xs) (length ys) + 28 * 2 ^k*
 ⟨proof⟩

definition *subtract-fermat-tm :: nat-lsbf ⇒ nat-lsbf ⇒ nat-lsbf tm where*

subtract-fermat-tm xs ys =1 do {

powk ← 2 ^t k;
minusy ← multiply-with-power-of-2-tm ys powk;
add-fermat-tm xs minusy
}

lemma *val-subtract-fermat-tm[simp, val-simp]: val (subtract-fermat-tm xs ys) =*

subtract-fermat xs ys
 ⟨proof⟩

lemma *time-subtract-fermat-tm-le: time (subtract-fermat-tm xs ys) ≤*

*38 + 66 * 2 ^k + 26 * length ys + 7 * max (length xs) (length ys)*
 ⟨proof⟩

definition *reduce-tm :: nat-lsbf ⇒ nat-lsbf tm where*

reduce-tm xs =1 do {

```

(ys, zs) ← split-tm xs;
b ← zs ≤nt ys;
if b then ys −nt zs
else do {
  kpow ← 2 t k;
  kpow1 ← kpow −t 1;
  zeros ← replicate-tm kpow1 False;
  a1 ← zeros @t [True];
  s ← (True # a1) +nt ys;
  s −nt zs
}
}

```

lemma *val-reduce-tm*[*simp*, *val-simp*]: *val (reduce-tm xs) = reduce xs*
 ⟨*proof*⟩

lemma *time-reduce-tm-le*: *time (reduce-tm xs) ≤ 155 + 85 * length xs + 46 * 2*
^{*k*}
 ⟨*proof*⟩

function (*domintros*) *from-nat-lsbf-tm* :: *nat-lsbf* ⇒ *nat-lsbf tm* **where**
from-nat-lsbf-tm xs = 1 do {
k1 ← *k* +_{*t*} 1;
powk ← 2 ^{*t*} *k1*;
lenxs ← *length-tm xs*;
b ← *lenxs* ≤_{*t*} *powk*;
 if *b* then *fill-tm powk xs* else do {
xs1 ← *take-tm powk xs*;
xs2 ← *drop-tm powk xs*;
a ← *xs1* +_{*nt*} *xs2*;
from-nat-lsbf-tm a
 }
}
 ⟨*proof*⟩

termination
 ⟨*proof*⟩

declare *from-nat-lsbf-tm.simps*[*simp del*]

lemma *val-from-nat-lsbf-tm*[*simp*, *val-simp*]: *val (from-nat-lsbf-tm xs) = from-nat-lsbf xs*
 ⟨*proof*⟩

abbreviation *e* :: *nat* **where** *e* ≡ 2 ^{*k*} (*k* + 1)

lemma *e-ge-1*: *e* ≥ 1 ⟨*proof*⟩

lemma *e-ge-2*: *e* ≥ 2 ⟨*proof*⟩

lemma *e-ge-4*: *k* > 0 ⇒ *e* ≥ 4 ⟨*proof*⟩

lemma *time-from-nat-lsbf-tm-le-aux*:

assumes $xs' = \text{add-nat } (\text{take } e \text{ } xs) \text{ } (\text{drop } e \text{ } xs)$
shows $\text{time } (\text{from-nat-lsbf-tm } xs) \leq 18 * e + 4 * \text{length } xs + 9 +$
 $(\text{if } \text{length } xs \leq e \text{ then } 0 \text{ else } \text{time } (\text{from-nat-lsbf-tm } xs'))$
 $\langle \text{proof} \rangle$

lemma *time-from-nat-lsbf-tm-le-aux'*:

assumes $xs' = \text{add-nat } (\text{take } e \text{ } xs) \text{ } (\text{drop } e \text{ } xs)$
shows $\text{time } (\text{from-nat-lsbf-tm } xs) \leq 66 * e + 4 * \text{length } xs + 35 +$
 $(\text{if } \text{length } xs \leq e + 1 \text{ then } 0 \text{ else } \text{time } (\text{from-nat-lsbf-tm } xs'))$
 $\langle \text{proof} \rangle$

function *time-from-nat-lsbf-tm-bound* **where**

time-from-nat-lsbf-tm-bound $l = 88 * e + 4 * l + 48 +$
 $(\text{if } l \leq 2 * e \text{ then } 0 \text{ else } \text{time-from-nat-lsbf-tm-bound } (l - (e - 1)))$
 $\langle \text{proof} \rangle$

termination

$\langle \text{proof} \rangle$

declare *time-from-nat-lsbf-tm-bound.simps*[*simp del*]

lemma *time-from-nat-lsbf-tm-le-bound*:

assumes $\text{length } xs \leq l$
shows $\text{time } (\text{from-nat-lsbf-tm } xs) \leq \text{time-from-nat-lsbf-tm-bound } l$
 $\langle \text{proof} \rangle$

lemma *time-from-nat-lsbf-tm-bound-closed*:

assumes $x \leq 2 * e$
assumes $x \geq e + 2$
shows $\text{time-from-nat-lsbf-tm-bound } (x + l * (e - 1)) =$
 $(l + 1) * (88 * e + 4 * x + 48) + 4 * (\sum \{0..l\}) * (e - 1)$
 $\langle \text{proof} \rangle$

lemma *time-from-nat-lsbf-tm-le*:

assumes $e \geq 4$
assumes $\text{length } xs \leq c * e$
shows $\text{time } (\text{from-nat-lsbf-tm } xs) \leq (288 * c + 144) + (96 + 192 * c + 8 * c$
 $* c) * e$
 $\langle \text{proof} \rangle$

fun *fft-combine-b-c-aux-tm* **where**

fft-combine-b-c-aux-tm $f \ g \ l \ (\text{revs}, s) \ [] \ [] = 1 \ \text{rev-tm } \text{revs}$
 $| \ \text{fft-combine-b-c-aux-tm } f \ g \ l \ (\text{revs}, s) \ (b \ \# \ bs) \ (c \ \# \ cs) = 1 \ \text{do } \{$
 $\quad c\text{-shifted} \leftarrow g \ c \ s;$
 $\quad r \leftarrow f \ b \ c\text{-shifted};$
 $\quad s\text{-new} \leftarrow s +_t \ l;$
 $\quad k1 \leftarrow k +_t \ 1;$
 $\quad \text{powk1} \leftarrow 2 \hat{=} _t \ k1;$
 $\quad s\text{-new-mod} \leftarrow s\text{-new} \ \text{mod}_t \ \text{powk1};$
 $\quad \text{fft-combine-b-c-aux-tm } f \ g \ l \ (r \ \# \ \text{revs}, s\text{-new-mod}) \ bs \ cs$
 $\quad \}$

| *fft-combine-b-c-aux-tm* - - - - - = *undefined*

lemma *val-fft-combine-b-c-aux-tm*[*simp*, *val-simp*]:

assumes *length bs = length cs*

shows *val (fft-combine-b-c-aux-tm f g l (revs, s) bs cs) =*

fft-combine-b-c-aux (λx y. val (f x y)) (λx y. val (g x y)) l (revs, s) bs cs

<proof>

lemma *time-fft-combine-b-c-aux-tm-le*:

assumes *length bs = length cs*

assumes $\bigwedge b. b \in \text{set } bs \implies \text{length } b = e$

assumes $\bigwedge c. c \in \text{set } cs \implies \text{length } c = e$

assumes $\bigwedge xs \ ys. \text{time } (f \ xs \ ys) \leq 38 + 66 * 2^k + 26 * \text{length } ys + 7 * \max(\text{length } xs) (\text{length } ys)$

assumes $s < e$

assumes $g = \text{multiply-with-power-of-2-tm} \vee g = \text{divide-by-power-of-2-tm}$

shows $\text{time } (\text{fft-combine-b-c-aux-tm } f \ g \ l \ (\text{revs}, s) \ bs \ cs) \leq \text{length } revs + 3 + \text{length } bs * (72 + 116 * e + 8 * l)$

<proof>

fun *fft-iff-fft-combine-b-c-add-tm* :: *bool* \implies *nat* \implies *nat-lsbf list* \implies *nat-lsbf list* \implies *nat-lsbf list tm* **where**

fft-iff-fft-combine-b-c-add-tm True l bs cs = 1 fft-combine-b-c-aux-tm add-fermat-tm divide-by-power-of-2-tm l ([], 0) bs cs

| *fft-iff-fft-combine-b-c-add-tm False l bs cs = 1 fft-combine-b-c-aux-tm add-fermat-tm multiply-with-power-of-2-tm l ([], 0) bs cs*

fun *fft-iff-fft-combine-b-c-subtract-tm* :: *bool* \implies *nat* \implies *nat-lsbf list* \implies *nat-lsbf list* \implies *nat-lsbf list tm* **where**

fft-iff-fft-combine-b-c-subtract-tm True l bs cs = 1 fft-combine-b-c-aux-tm subtract-fermat-tm divide-by-power-of-2-tm l ([], 0) bs cs

| *fft-iff-fft-combine-b-c-subtract-tm False l bs cs = 1 fft-combine-b-c-aux-tm subtract-fermat-tm multiply-with-power-of-2-tm l ([], 0) bs cs*

lemma *val-fft-iff-fft-combine-b-c-add-tm*[*simp*, *val-simp*]:

assumes *length bs = length cs*

shows *val (fft-iff-fft-combine-b-c-add-tm it l bs cs) = fft-iff-fft-combine-b-c-add it l bs cs*

<proof>

lemma *val-fft-iff-fft-combine-b-c-subtract-tm*[*simp*, *val-simp*]:

assumes *length bs = length cs*

shows *val (fft-iff-fft-combine-b-c-subtract-tm it l bs cs) = fft-iff-fft-combine-b-c-subtract it l bs cs*

<proof>

lemma *time-fft-combine-b-c-add-tm-le*:

assumes *length bs = length cs*

assumes $\bigwedge b. b \in \text{set } bs \implies \text{length } b = e$

assumes $\bigwedge c. c \in \text{set } cs \implies \text{length } c = e$
shows $\text{time } (\text{fft-iffit-combine-b-c-add-tm it } l \text{ } bs \text{ } cs) \leq 4 + \text{length } bs * (72 + 116 * e + 8 * l)$
 <proof>

lemma *time-fft-combine-b-c-subtract-tm-le:*

assumes $\text{length } bs = \text{length } cs$
assumes $\bigwedge b. b \in \text{set } bs \implies \text{length } b = e$
assumes $\bigwedge c. c \in \text{set } cs \implies \text{length } c = e$
shows $\text{time } (\text{fft-iffit-combine-b-c-subtract-tm it } l \text{ } bs \text{ } cs) \leq 4 + \text{length } bs * (72 + 116 * e + 8 * l)$
 <proof>

fun *fft-iffit-tm* **where**

```

fft-iffit-tm it l [] =1 return []
| fft-iffit-tm it l [x] =1 return [x]
| fft-iffit-tm it l [x, y] =1 do {
  r1 ← add-fermat-tm x y;
  r2 ← subtract-fermat-tm x y;
  return [r1, r2]
}
| fft-iffit-tm it l a =1 do {
  nums1 ← evens-odds-tm True a;
  nums2 ← evens-odds-tm False a;
  b ← fft-iffit-tm it (2 * l) nums1;
  c ← fft-iffit-tm it (2 * l) nums2;
  g ← fft-iffit-combine-b-c-add-tm it l b c;
  h ← fft-iffit-combine-b-c-subtract-tm it l b c;
  g @t h
}

```

lemma *val-fft-iffit-tm[simp, val-simp]:* $\text{length } a = 2^m \implies \text{val } (\text{fft-iffit-tm it } l \text{ } a) = \text{fft-iffit it } l \text{ } a$
 <proof>

lemma *time-fft-iffit-tm-le-aux:*

assumes $\bigwedge x. x \in \text{set } a \implies \text{length } x = e$
assumes $\text{length } a = 2^m$
shows $\text{time } (\text{fft-iffit-tm it } l \text{ } a) \leq 2^{m-1} * (52 + 87 * e) + (m - 1) * 2^m * (76 + 116 * e) + (\sum i \leftarrow [0..<m-1]. 2^i) * (8 * 2^m * l + 13)$
 <proof>

lemma *time-fft-iffit-tm-le:*

assumes $\bigwedge x. x \in \text{set } a \implies \text{length } x = e$
assumes $\text{length } a = 2^m$
shows $\text{time } (\text{fft-iffit-tm it } l \text{ } a) \leq 2^m * (65 + 87 * e) + m * 2^m * (76 + 116 * e) + (8 * l) * 2^{(2 * m)}$
 <proof>

```

fun fft-tm where
fft-tm l a =1 fft-iff-tm False l a
fun iff-tm where
iff-tm l a =1 fft-iff-tm True l a

```

```

lemma val-fft-tm[simp, val-simp]: length a = 2 ^ m  $\implies$  val (fft-tm l a) = fft l a
  <proof>

```

```

lemma val-iff-tm[simp, val-simp]: length a = 2 ^ m  $\implies$  val (iff-tm l a) = iff l a
  <proof>

```

```

lemma time-fft-tm-le:

```

```

  assumes  $\bigwedge x. x \in \text{set } a \implies \text{length } x = e$ 
  assumes length a = 2 ^ m
  shows time (fft-tm l a)  $\leq 2 ^ m * (66 + 87 * e) + m * 2 ^ m * (76 + 116 * e) + (8 * l) * 2 ^ (2 * m)$ 
  <proof>

```

```

lemma time-iff-tm-le:

```

```

  assumes  $\bigwedge x. x \in \text{set } a \implies \text{length } x = e$ 
  assumes length a = 2 ^ m
  shows time (iff-tm l a)  $\leq 2 ^ m * (66 + 87 * e) + m * 2 ^ m * (76 + 116 * e) + (8 * l) * 2 ^ (2 * m)$ 
  <proof>

```

```

end

```

```

end

```

3.4 Final Preparations

```

theory Schoenhage-Strassen

```

```

imports

```

```

  Main
  HOL-Algebra.IntRing
  HOL-Algebra.QuotRing
  HOL-Algebra.Chinese-Remainder
  HOL-Algebra.Ring
  HOL-Algebra.Polynomials
  Word-Lib.Bit-Comprehension
  Z-mod-power-of-2
  Z-mod-Fermat
  Karatsuba.Nat-LSBF
  Karatsuba.Karatsuba-Sum-Lemmas
  Karatsuba.Karatsuba
  ../Preliminaries/Schoenhage-Strassen-Ring-Lemmas

```

```

begin

```

```

lemma aux-ineq-1: n > 1  $\implies 2 ^ (2 * n - 1) > n + 1 + 2 ^ n$ 

```

<proof>

lemma *aux-ineq-2*: $n > 2 \implies 2^{(2 * n - 2)} > n + 2^n$

<proof>

lemma *aux-ineq-3*: $n > 1 \implies 2^n \geq n + 2$

<proof>

lemma (*in residues*) *nat-embedding-eq*: $\text{ring.nat-embedding } R \ x = \text{int } x \text{ mod } m$

<proof>

lemma (*in residues*) *carrier-mod-eq*: $x \in \text{carrier } R \implies x \text{ mod } m = x$

<proof>

The Schoenhage-Strassen Multiplication in \mathbb{Z}_{F_m} works recursively. In the following, we will state some lemmas that will be useful in the recursion case ($m \geq 3$).

locale *m-lemmas* =

fixes $m :: \text{nat}$

assumes *m-ge-3*: $\neg m < 3$

begin

Lemmas in *nat* resp. *int*.

lemma *m-gt-0*: $m > 0$ *<proof>*

definition $n :: \text{nat}$ **where**

$n \equiv (\text{if odd } m \text{ then } (m + 1) \text{ div } 2 \text{ else } (m + 2) \text{ div } 2)$

definition $oe-n :: \text{nat}$ **where**

$oe-n \equiv (\text{if odd } m \text{ then } n + 1 \text{ else } n)$

lemma *n-gt-1*: $n > 1$ *<proof>*

lemma *n-ge-2*: $n \geq 2$ *<proof>*

lemma *n-gt-0*: $n > 0$ *<proof>*

lemma *even-m-imp-n-ge-3*: $\text{even } m \implies n \geq 3$ *<proof>*

lemma *n-lt-m*: $n < m$ *<proof>*

lemma *oe-n-gt-1*: $oe-n > 1$ *<proof>*

lemma *oe-n-gt-0*: $oe-n > 0$ *<proof>*

lemma *oe-n-le-n*: $oe-n \leq n + 1$ *<proof>*

lemma *oe-n-minus-1-le-n*: $oe-n - 1 \leq n$ *<proof>*

lemma *two-pow-oe-n-div-2*: $(2::\text{nat})^{oe-n \text{ div } 2} = 2^{(oe-n - 1)}$

<proof>

lemma *two-pow-oe-n-as-halves*: $(2::\text{nat})^{oe-n} = 2^{(oe-n - 1)} + 2^{(oe-n - 1)}$

<proof>

<proof>

lemma *two-pow-Suc-oe-n-as-prod*: $(2::\text{nat})^{(oe-n + 1)} = 4 * 2^{(oe-n - 1)}$

$\langle \text{proof} \rangle$

lemma *index-intros*:

fixes $i :: \text{nat}$

assumes $i < 2^{\wedge}(oe-n - 1)$

shows $i < 2^{\wedge}oe-n \ 2^{\wedge}(oe-n - 1) + i < 2^{\wedge}oe-n$

$\langle \text{proof} \rangle$

lemma *index-decomp*:

assumes $j < (2::\text{nat})^{\wedge}(oe-n + 1)$

shows

$j \text{ div } 2^{\wedge}(oe-n - 1) < 4$

$j \text{ mod } 2^{\wedge}(oe-n - 1) < 2^{\wedge}(oe-n - 1)$

$j = (j \text{ div } 2^{\wedge}(oe-n - 1)) * 2^{\wedge}(oe-n - 1) + (j \text{ mod } 2^{\wedge}(oe-n - 1))$

$\langle \text{proof} \rangle$

lemma *index-comp*:

fixes $i \ j :: \text{nat}$

assumes $i < 4 \ j < 2^{\wedge}(oe-n - 1)$

shows

$i * 2^{\wedge}(oe-n - 1) + j < 2^{\wedge}(oe-n + 1)$

$(i * 2^{\wedge}(oe-n - 1) + j) \text{ div } 2^{\wedge}(oe-n - 1) = i$

$(i * 2^{\wedge}(oe-n - 1) + j) \text{ mod } 2^{\wedge}(oe-n - 1) = j$

$\langle \text{proof} \rangle$

lemma *mn*:

odd $m \implies m = 2 * n - 1$

even $m \implies m = 2 * n - 2$

$\langle \text{proof} \rangle$

lemma *m0*: $m = (n - 1) + (oe-n - 1)$

$\langle \text{proof} \rangle$

lemma *m1*: $m + 1 = (n - 1) + oe-n$

$\langle \text{proof} \rangle$

lemma *two-pow-m1-as-prod*: $(2::\text{nat})^{\wedge}(m + 1) = 2^{\wedge}(n - 1) * 2^{\wedge}oe-n$

$\langle \text{proof} \rangle$

lemma *two-pow-m0-as-prod*: $(2::\text{nat})^{\wedge}m = 2^{\wedge}(n - 1) * 2^{\wedge}(oe-n - 1)$

$\langle \text{proof} \rangle$

lemma *two-pow-two-n-le*: $(2::\text{nat})^{\wedge}(2 * n) \leq 2 * 2^{\wedge}(m + 1)$

$\langle \text{proof} \rangle$

lemma *oe-n-m-bound-0*: $oe-n + 2^{\wedge}n < 2^{\wedge}m$

$\langle \text{proof} \rangle$

lemma *oe-n-m-bound-1*: $oe-n + 1 + 2^{\wedge}n \leq 2^{\wedge}m$

$\langle \text{proof} \rangle$

lemma *two-pow-oe-n-m-bound-1*: $(2::'a::\text{linordered-semidom})^{\wedge}(oe-n + 1 + 2^{\wedge}n) \leq 2^{\wedge}2^{\wedge}m$

$\langle \text{proof} \rangle$

lemma *two-pow-oe-n-m-bound-0-int*: $2^{\wedge}(\text{oe-n} + 2^{\wedge}n) < \text{int-lsb-f-fermat.n m}$
 ⟨proof⟩

lemma *two-pow-oe-n-m-bound-1-int*: $2^{\wedge}(\text{oe-n} + 1 + 2^{\wedge}n) < \text{int-lsb-f-fermat.n m}$
 ⟨proof⟩

lemma *oe-n-n-bound-1*: $\text{oe-n} + 1 + 2^{\wedge}n \leq 2^{\wedge}(n + 1)$
 ⟨proof⟩

definition *pad-length* **where** *pad-length* = $3 * n + 5$

Lemmas using residue rings.

definition *Zn* **where** *Zn* = *residue-ring (int-lsb-f-mod.n (n + 2))*

definition *Fn* **where** *Fn* = *residue-ring (int-lsb-f-fermat.n n)*

definition *Fm* **where** *Fm* = *residue-ring (int-lsb-f-fermat.n m)*

Lemmas in $\mathbb{Z}_{2^{n+2}}$

sublocale *Znr* : *int-lsb-f-mod n + 2*
rewrites *Znr.Zn* \equiv *Zn*
 ⟨proof⟩

Lemmas in \mathbb{Z}_{F_m} resp. \mathbb{Z}_{F_n} .

sublocale *Fnr* : *int-lsb-f-fermat n*
rewrites *Fnr.Fn* \equiv *Fn*
 ⟨proof⟩

sublocale *Fnr-M* : *multiplicative-subgroup Fn Units Fn units-of Fn*
 ⟨proof⟩

sublocale *Fmr* : *int-lsb-f-fermat m*
rewrites *Fmr.Fn* \equiv *Fm*
 ⟨proof⟩

sublocale *Fmr-M* : *multiplicative-subgroup Fm Units Fm units-of Fm*
 ⟨proof⟩

lemma *two-pow-oe-n-primitive-root-Fm*:
Fmr.primitive-root $(2^{\wedge}\text{oe-n}) (2 [\wedge]_{Fm} (2::\text{nat})^{\wedge}(n - 1))$
 ⟨proof⟩

lemma *two-pow-oe-n-root-of-unity-Fm*:
Fmr.root-of-unity $(2^{\wedge}\text{oe-n}) (2 [\wedge]_{Fm} (2::\text{nat})^{\wedge}(n - 1))$
 ⟨proof⟩

lemma *four-prim-root-Fn*: *Fnr.primitive-root* $(2^{\wedge}n) (2 [\wedge]_{Fn} (2::\text{nat}))$
 ⟨proof⟩

lemma *two-oe-n*: $2 [\wedge]_{Fn} \text{oe-n} = 2^{\wedge}\text{oe-n}$
 ⟨proof⟩

lemma *two-oe-n-Units-Fn*: $2^{\wedge} oe-n \in Units Fn$
 ⟨proof⟩

lemma *two-oe-n-carrier-Fn*: $2^{\wedge} oe-n \in carrier Fn$
 ⟨proof⟩

definition *prim-root-exponent* :: nat **where** *prim-root-exponent* = (if odd m then 1 else 2)

definition μ **where** $\mu = 2 [\wedge]_{Fn} prim-root-exponent$

lemma *μ -Units-Fn*: $\mu \in Units Fn$
 ⟨proof⟩

lemma *μ -carrier-Fn*: $\mu \in carrier Fn$
 ⟨proof⟩

lemma *μ -prim-root*: *Fnr.primitive-root* ($2^{\wedge} oe-n$) μ
 ⟨proof⟩

lemma *μ -root-of-unity*: *Fnr.root-of-unity* ($2^{\wedge} oe-n$) μ
 ⟨proof⟩

lemma *μ -halfway-property*: $\mu [\wedge]_{Fn} ((2::nat)^{\wedge} oe-n \text{ div } 2) = \ominus_{Fn} \mathbf{1}_{Fn}$
 ⟨proof⟩

end

Lemmas only depending on one of the input arguments (and m).

locale *carrier-input = m-lemmas* +
fixes *num* :: nat-lsbf
assumes *num-carrier*: *num* $\in int-lsbf-fermat.fermat-non-unique-carrier m$
begin

definition *num-blocks* **where** *num-blocks* = *subdivide* ($2^{\wedge} (n - 1)$) *num*

definition *num-blocks-carrier* **where** *num-blocks-carrier* = *map* (*fill* ($2^{\wedge} (n + 1)$)) *num-blocks*

definition *num-Zn* **where** *num-Zn* = *map* *Znr.reduce* *num-blocks*

definition *num-Zn-pad* **where** *num-Zn-pad* = *concat* (*map* (*fill pad-length*) *num-Zn*)

definition *num-dft* **where** *num-dft* = *Fnr.fft* *prim-root-exponent* *num-blocks-carrier*

definition *num-dft-odds* **where** *num-dft-odds* = *evens-odds* *False* *num-dft*

lemmas *defs* = *num-blocks-def* *num-blocks-carrier-def* *num-Zn-def* *num-Zn-pad-def* *num-dft-def* *num-dft-odds-def*

lemma *length-num[simp]*: *length num* = $2^{\wedge} (m + 1)$
 ⟨proof⟩

lemma *length-num-blocks[simp]*: *length num-blocks* = $2^{\wedge} oe-n$
 ⟨proof⟩

lemma *length-nth-num-blocks[simp]*:
fixes *i* :: nat
assumes *i* < $2^{\wedge} oe-n$

<proof>

lemma *length-num-dft-odds[simp]*: $\text{length num-dft-odds} = 2^{\text{oe-n} - 1}$

<proof>

lemma *num-dft-odds-carrier[simp]*: $\text{set num-dft-odds} \subseteq \text{Fnr.fermat-non-unique-carrier}$

<proof>

end

3.4.1 A special residue problem

definition *solve-special-residue-problem* **where**

solve-special-residue-problem $n \xi \eta =$

$(\text{let } \delta = \text{int-lsbf-mod.subtract-mod } (n + 2) \eta \text{ (take } (n + 2) \xi) \text{ in}$
 $\text{add-nat } \xi \text{ (add-nat } (\delta \gg_n (2^n)) \delta))$

lemma *two-pow-n-geq-n-plus-2*: $n \geq 2 \implies 2^n \geq n + 2$

<proof>

lemma *fermat-mod-pow-2-aux*: $n \geq 2 \implies (2::\text{nat})^{(2^n)} \bmod 2^{(n+2)} = 0$

<proof>

definition *solves-special-residue-problem* **where**

solves-special-residue-problem $z n \xi \eta \equiv$

$z < 2^{(n+2)} * \text{int-lsbf-fermat.n } n$
 $\wedge z \bmod \text{int-lsbf-fermat.n } n = \xi$
 $\wedge z \bmod (2^{(n+2)}) = \eta$

lemma *solve-special-residue-problem-correct*:

fixes $n :: \text{nat}$

fixes $\xi \eta :: \text{nat-lsbf}$

assumes $n \geq 2$

assumes $\text{length } \eta \leq n + 2$

assumes $\text{Nat-LSBF.to-nat } \xi < \text{int-lsbf-fermat.n } n$

assumes $z = \text{solve-special-residue-problem } n \xi \eta$

shows *solves-special-residue-problem* $(\text{Nat-LSBF.to-nat } z) n (\text{Nat-LSBF.to-nat } \xi) (\text{Nat-LSBF.to-nat } \eta)$

<proof>

lemma *fn-zn-coprime*: $\text{coprime } (\text{int-lsbf-fermat.n } n) (2^{(n+2)})$

<proof>

lemma *int-ideal-add*: $\text{Idl}_{\mathbb{Z}} \{m\} \langle + \rangle_{\mathbb{Z}} \text{Idl}_{\mathbb{Z}} \{n\} = \text{Idl}_{\mathbb{Z}} \{\text{gcd } m \ n\}$

<proof>

lemma *int-ideal-inter*: $\text{Idl}_{\mathbb{Z}} \{m\} \cap \text{Idl}_{\mathbb{Z}} \{n\} = \text{Idl}_{\mathbb{Z}} \{\text{lcm } m \ n\}$

<proof>

corollary *coprime* $m\ n \implies \text{Idl}_{\mathcal{Z}}\ \{m\} \langle + \rangle_{\mathcal{Z}} \text{Idl}_{\mathcal{Z}}\ \{n\} = \text{carrier } \mathcal{Z}$
 ⟨proof⟩

lemma *genideal-uminus*: $\text{Idl}_{\mathcal{Z}}\ \{-x\} = \text{Idl}_{\mathcal{Z}}\ \{x\}$
 ⟨proof⟩

lemma *genideal-normalize*: $\text{Idl}_{\mathcal{Z}}\ \{x\} = \text{Idl}_{\mathcal{Z}}\ \{\text{normalize } x\}$
 ⟨proof⟩

corollary *coprime* $m\ n \implies \text{Idl}_{\mathcal{Z}}\ \{m\} \cap \text{Idl}_{\mathcal{Z}}\ \{n\} = \text{Idl}_{\mathcal{Z}}\ \{m * n\}$
 ⟨proof⟩

lemma *int-ideal-inter-a-r-coset-distrib*: $(\text{Idl}_{\mathcal{Z}}\ \{m\} \cap \text{Idl}_{\mathcal{Z}}\ \{n\}) \langle + \rangle_{\mathcal{Z}} x = (\text{Idl}_{\mathcal{Z}}\ \{m\} \langle + \rangle_{\mathcal{Z}} x) \cap (\text{Idl}_{\mathcal{Z}}\ \{n\} \langle + \rangle_{\mathcal{Z}} x)$
 ⟨proof⟩

lemma *chinese-remainder-very-simple-int*:
 fixes $x\ y\ m\ n :: \text{int}$
 assumes $x \bmod m = y \bmod m$
 assumes $x \bmod n = y \bmod n$
 shows $x \bmod (\text{lcm } m\ n) = y \bmod (\text{lcm } m\ n)$
 ⟨proof⟩

lemma *chinese-remainder-very-simple-nat*:
 fixes $x\ y\ m\ n :: \text{nat}$
 assumes $x \bmod m = y \bmod m$
 assumes $x \bmod n = y \bmod n$
 shows $x \bmod (\text{lcm } m\ n) = y \bmod (\text{lcm } m\ n)$
 ⟨proof⟩

lemma *special-residue-problem-unique-solution*:
 fixes $n :: \text{nat}$
 fixes $\xi\ \eta :: \text{nat}$
 assumes *solves-special-residue-problem* $z1\ n\ \xi\ \eta$
 assumes *solves-special-residue-problem* $z2\ n\ \xi\ \eta$
 shows $z1 = z2$
 ⟨proof⟩

3.4.2 Subroutine for combining the final result

fun *combine-z-aux* **where**
combine-z-aux $l\ acc\ [] = \text{concat } (\text{rev } acc)$
 | *combine-z-aux* $l\ acc\ [z] = \text{combine-z-aux } l\ (z \# acc)\ []$
 | *combine-z-aux* $l\ acc\ (z1 \# z2 \# zs) = (\text{let}$
 $(z1h, z1t) = \text{split-at } l\ z1\ \text{in}$
 combine-z-aux $l\ (z1h \# acc)\ ((\text{add-nat } z1t\ z2) \# zs)$
)

definition *combine-z* $:: \text{nat} \Rightarrow \text{nat-lsbf list} \Rightarrow \text{nat-lsbf}$ **where**

$combine\text{-}z\ l\ zs = combine\text{-}z\text{-}aux\ l\ []\ zs$

lemma *combine-z-aux-correct*:

assumes $l > 0$

assumes $\bigwedge z. z \in set\ zs \implies length\ z \geq l$

shows $Nat\text{-}LSBF.to\text{-}nat\ (combine\text{-}z\text{-}aux\ l\ acc\ zs) = Nat\text{-}LSBF.to\text{-}nat\ (concat\ (rev\ acc)) +$

$2^{\wedge}(length\ (concat\ acc)) * (\sum i \leftarrow [0..<length\ zs]. Nat\text{-}LSBF.to\text{-}nat\ (zs\ !\ i) * 2^{\wedge}(i * l))$

<proof>

lemma *combine-z-correct*:

assumes $l > 0$

assumes $\bigwedge z. z \in set\ zs \implies length\ z \geq l$

shows $Nat\text{-}LSBF.to\text{-}nat\ (combine\text{-}z\ l\ zs) = (\sum i \leftarrow [0..<length\ zs]. Nat\text{-}LSBF.to\text{-}nat\ (zs\ !\ i) * 2^{\wedge}(i * l))$

<proof>

lemma *length-combine-z-aux-le*:

assumes $\bigwedge z. z \in set\ zs \implies length\ z \leq lz$

assumes $length\ z \leq lz + 1$

assumes $l > 0$

shows $length\ (combine\text{-}z\text{-}aux\ l\ acc\ (z\ \#\ zs)) \leq (lz + 1) * (length\ zs + 1) + length\ (concat\ acc)$

<proof>

lemma *length-combine-z-le*:

assumes $\bigwedge z. z \in set\ zs \implies length\ z \leq lz$

assumes $l > 0$

shows $length\ (combine\text{-}z\ l\ zs) \leq (lz + 1) * length\ zs$

<proof>

3.5 Schoenhage-Strassen Multiplication in \mathbb{Z}_{F_m}

function *schoenhage-strassen* :: $nat \Rightarrow nat\text{-}lsbf \Rightarrow nat\text{-}lsbf \Rightarrow nat\text{-}lsbf$ **where**
schoenhage-strassen $m\ a\ b =$

(if $m < 3$ *then* *int-lsbf-fermat.from-nat-lsbf* $m\ (grid\text{-}mul\text{-}nat\ a\ b)$ *else*
let

$n = (if\ odd\ m\ then\ (m + 1)\ div\ 2\ else\ (m + 2)\ div\ 2);$

$oe\text{-}n = (if\ odd\ m\ then\ n + 1\ else\ n);$

$a' = subdivide\ (2^{\wedge}(n - 1))\ a;$

$b' = subdivide\ (2^{\wedge}(n - 1))\ b;$

— residue mod 2^{n+2}

$\alpha = map\ (int\text{-}lsbf\text{-}mod.reduce\ (n + 2))\ a';$

$u = concat\ (map\ (fill\ (3 * n + 5))\ \alpha);$

$\beta = map\ (int\text{-}lsbf\text{-}mod.reduce\ (n + 2))\ b';$

$v = concat\ (map\ (fill\ (3 * n + 5))\ \beta);$

$uw = ensure\text{-}length\ ((3 * n + 5) * 2^{\wedge}(oe\text{-}n + 1))\ (karatsuba\text{-}mul\text{-}nat\ u\ v);$

```

γ = subdivide (2 ^ (oe-n - 1)) (subdivide (3*n + 5) uv);
η = map4 (λx y z w.
  int-lsbf-mod.add-mod (n + 2)
  (int-lsbf-mod.subtract-mod (n + 2) (take (n + 2) x) (take (n + 2) y))
  (int-lsbf-mod.subtract-mod (n + 2) (take (n + 2) z) (take (n + 2) w))
)
(γ ! 0) (γ ! 1) (γ ! 2) (γ ! 3);

```

— residue mod F_n

```

prim-root-exponent = (if odd m then 1 else 2);
a'-carrier = map (fill (2 ^ (n + 1))) a';
b'-carrier = map (fill (2 ^ (n + 1))) b';
a-dft = int-lsbf-fermat.fft n prim-root-exponent a'-carrier;
b-dft = int-lsbf-fermat.fft n prim-root-exponent b'-carrier;
a-dft-odds = evens-odds False a-dft;
b-dft-odds = evens-odds False b-dft;
c-dft-odds = map2 (schoenhage-strassen n) a-dft-odds b-dft-odds;
c-diffs = int-lsbf-fermat.iffn n (prim-root-exponent * 2) c-dft-odds;
ξ' = map2 (λc j. int-lsbf-fermat.add-fermat n
  (int-lsbf-fermat.divide-by-power-of-2 c j (oe-n + prim-root-exponent * j - 1))
  (int-lsbf-fermat.from-nat-lsbf n (replicate (oe-n + 2 ^ n) False @ [True])))
c-diffs [0..<2 ^ (oe-n - 1)];
ξ = map (int-lsbf-fermat.reduce n) ξ';

```

— calculate z_j for $j < 2^n$

```

z = map2 (solve-special-residue-problem n) ξ η;
z-filled = map (fill (2 ^ (n - 1))) z;
z-consts = replicate (2 ^ (oe-n - 1)) (replicate (oe-n + 2 ^ n) False @ [True]);
z-sum = combine-z (2 ^ (n - 1)) (z-filled @ z-consts);
result = int-lsbf-fermat.from-nat-lsbf m z-sum

```

— return the resulting sum

```
in result)
```

```
<proof>
```

termination

```
<proof>
```

declare *schoenhage-strassen.simps*[*simp del*]

locale *schoenhage-strassen-context* =

```
fixes m :: nat
```

```
fixes a :: nat-lsbf
```

```
fixes b :: nat-lsbf
```

```
assumes m-ge-3: ¬ m < 3
```

```
assumes a-carrier: a ∈ int-lsbf-fermat.fermat-non-unique-carrier m
```

```
assumes b-carrier: b ∈ int-lsbf-fermat.fermat-non-unique-carrier m
```

begin

sublocale *m-lemmas*

<proof>

sublocale *A: carrier-input m a*

<proof>

sublocale *B: carrier-input m b*

<proof>

definition *wv-length* **where** *wv-length* = *pad-length* * $2^{\wedge}(\text{oe-n} + 1)$

definition *wv-unpadded* **where** *wv-unpadded* = *karatsuba-mul-nat* *A.num-Zn-pad* *B.num-Zn-pad*

definition *wv* **where** *wv* = *ensure-length* *wv-length* *wv-unpadded*

definition *γs* **where** *γs* = *subdivide* *pad-length* *wv*

definition *γ* **where** *γ* = *subdivide* ($2^{\wedge}(\text{oe-n} - 1)$) *γs*

definition *η* **where** *η* = *map4* ($\lambda x y z w.$ *int-lsb-f-mod.add-mod* (*n* + 2)
(*int-lsb-f-mod.subtract-mod* (*n* + 2) (*take* (*n* + 2) *x*) (*take* (*n* + 2) *y*))
(*int-lsb-f-mod.subtract-mod* (*n* + 2) (*take* (*n* + 2) *z*) (*take* (*n* + 2) *w*))
) (*γ* ! 0) (*γ* ! 1) (*γ* ! 2) (*γ* ! 3)

definition *c-dft-odds* **where** *c-dft-odds* = *map2* (*schoenhage-strassen* *n*) *A.num-dft-odds* *B.num-dft-odds*

definition *c-diffs* **where** *c-diffs* = *int-lsb-f-fermat.iff* *n* (*prim-root-exponent* * 2)
c-dft-odds

definition *ξ'* **where** *ξ'* = *map2* ($\lambda c j.$ *int-lsb-f-fermat.add-fermat* *n*
(*int-lsb-f-fermat.divide-by-power-of-2* *cj* (*oe-n* + *prim-root-exponent* * *j* - 1))
(*int-lsb-f-fermat.from-nat-lsb* *n* (*replicate* (*oe-n* + $2^{\wedge}n$) *False* @ [*True*])))
c-diffs [*0*.. $2^{\wedge}(\text{oe-n} - 1)$]

definition *ξ* **where** *ξ* = *map* (*int-lsb-f-fermat.reduce* *n*) *ξ'*

definition *z* **where** *z* = *map2* (*solve-special-residue-problem* *n*) *ξ* *η*

definition *z-filled* **where** *z-filled* = *map* (*fill* ($2^{\wedge}(n - 1)$)) *z*

definition *z-consts* **where** *z-consts* = *replicate* ($2^{\wedge}(\text{oe-n} - 1)$) (*replicate* (*oe-n* + $2^{\wedge}n$) *False* @ [*True*])

definition *z-sum* **where** *z-sum* = *combine-z* ($2^{\wedge}(n - 1)$) (*z-filled* @ *z-consts*)

definition *result* **where** *result* = *int-lsb-f-fermat.from-nat-lsb* *m* *z-sum*

lemmas *defs = n-def oe-n-def A.defs B.defs pad-length-def wv-length-def wv-unpadded-def wv-def*

γs-def γ-def η-def c-dft-odds-def c-diffs-def ξ'-def ξ-def z-def z-filled-def z-consts-def z-sum-def result-def prim-root-exponent-def μ-def

lemma *result-eq: schoenhage-strassen m a b = result*

<proof>

lemma *length-wv: length wv = wv-length*

<proof>

lemma *pad-length-gt-0: pad-length > 0* *<proof>*

lemma *scuv*:

$\text{length } (\text{subdivide pad-length } uv) = 2^{\wedge}(oe-n + 1)$
 $x \in \text{set } (\text{subdivide pad-length } uv) \implies \text{length } x = \text{pad-length}$
(proof)

lemma *length-c-dft-odds*: $\text{length } c\text{-dft-odds} = 2^{\wedge}(oe-n - 1)$
(proof)

lemma *length-c-diffs*: $\text{length } c\text{-diffs} = 2^{\wedge}(oe-n - 1)$
(proof)

lemma *length-ξ'*: $\text{length } \xi' = 2^{\wedge}(oe-n - 1)$
(proof)

lemma *length-ξ*: $\text{length } \xi = 2^{\wedge}(oe-n - 1)$
(proof)

lemma *γ-nth*: $i < 4 \implies j < 2^{\wedge}(oe-n - 1) \implies \gamma ! i ! j = (\text{subdivide pad-length } uv) ! (i * 2^{\wedge}(oe-n - 1) + j)$ **for** $i j$
(proof)

lemma *γ-nth'*: $\bigwedge j. j < 2^{\wedge}(oe-n + 1) \implies \gamma ! (j \text{ div } 2^{\wedge}(oe-n - 1)) ! (j \text{ mod } 2^{\wedge}(oe-n - 1)) = \text{subdivide pad-length } uv ! j$
(proof)

lemma *scγ*: $\text{length } \gamma = 4 \bigwedge i. i < 4 \implies \text{length } (\gamma ! i) = 2^{\wedge}(oe-n - 1)$
(proof)

lemmas *length-γ* = $sc\gamma(1)$

lemmas *length-γ-i* = $sc\gamma(2)$

lemma *length-γ-nth*: $i < 4 \implies j < 2^{\wedge}(oe-n - 1) \implies \text{length } (\gamma ! i ! j) = \text{pad-length}$
(proof)

lemma *length-η*: $\text{length } \eta = 2^{\wedge}(oe-n - 1)$ (proof)

lemma *length-z*: $\text{length } z = 2^{\wedge}(oe-n - 1)$
(proof)

lemma *nth-z*: $z ! j = \text{solve-special-residue-problem } n (\xi ! j) (\eta ! j)$ **if** $j < 2^{\wedge}(oe-n - 1)$ **for** j
(proof)

lemma *length-z-filled*: $\text{length } z\text{-filled} = 2^{\wedge}(oe-n - 1)$
(proof)

lemma *length-z-consts*: $\text{length } z\text{-consts} = 2^{\wedge}(oe-n - 1)$
(proof)

end

theorem *schoenhage-strassen-correct'*:

assumes $a \in \text{int-lsbf-fermat.fermat-non-unique-carrier } m$

assumes $b \in \text{int-lsbf-fermat.fermat-non-unique-carrier } m$

shows $\text{int-lsbf-fermat.to-residue-ring } m (\text{schoenhage-strassen } m a b)$

$= \text{int-lsbf-fermat.to-residue-ring } m a \otimes_{\text{int-lsbf-fermat.Fn } m} \text{int-lsbf-fermat.to-residue-ring } m b \wedge \text{schoenhage-strassen } m a b \in \text{int-lsbf-fermat.fermat-non-unique-carrier } m$
(proof)

3.6 Schoenhage-Strassen Multiplication in \mathbb{N}

In order to multiply a and b (given in LSBF representation), find an m s.t. $a \cdot b < F_m$.

It suffices to just pick $m = \max(\text{bitsize}(\text{length } a)) (\text{bitsize}(\text{length } b)) + 1$.

definition *schoenhage-strassen-mul* **where**

schoenhage-strassen-mul a $b = (\text{let } m = \max(\text{bitsize}(\text{length } a)) (\text{bitsize}(\text{length } b)) + 1 \text{ in}$
 $\text{int-lsbf-fermat.reduce } m (\text{schoenhage-strassen } m (\text{fill } (2 \wedge (m + 1)) a) (\text{fill } (2 \wedge (m + 1)) b))$
 $)$

locale *schoenhage-strassen-mul-context* =

fixes a $b :: \text{nat-lsbf}$

begin

definition *bits-a* **where** $\text{bits-a} = \text{bitsize}(\text{length } a)$

definition *bits-b* **where** $\text{bits-b} = \text{bitsize}(\text{length } b)$

definition m' **where** $m' = \max \text{bits-a } \text{bits-b}$

definition m **where** $m = m' + 1$

definition *car-len* **where** $\text{car-len} = (2::\text{nat}) \wedge (m + 1)$

definition *fill-a* **where** $\text{fill-a} = \text{fill } \text{car-len } a$

definition *fill-b* **where** $\text{fill-b} = \text{fill } \text{car-len } b$

definition *fm-result* **where** $\text{fm-result} = \text{schoenhage-strassen } m \text{ fill-a } \text{fill-b}$

lemmas *defs* = *bits-a-def bits-b-def m'-def m-def car-len-def fill-a-def fill-b-def*

lemma

shows *length-a*: $\text{length } a < 2 \wedge (m - 1)$

and *length-b*: $\text{length } b < 2 \wedge (m - 1)$

$\langle \text{proof} \rangle$

lemma

shows *length-a'*: $\text{length } a \leq 2 \wedge (m + 1)$

and *length-b'*: $\text{length } b \leq 2 \wedge (m + 1)$

$\langle \text{proof} \rangle$

lemma *length-fill-a*: $\text{length } \text{fill-a} = 2 \wedge (m + 1)$

$\langle \text{proof} \rangle$

lemma *length-fill-b*: $\text{length } \text{fill-b} = 2 \wedge (m + 1)$

$\langle \text{proof} \rangle$

sublocale *fm*: *int-lsbf-fermat* m $\langle \text{proof} \rangle$

definition Fm **where** $Fm = \text{residue-ring } (\text{int-lsbf-fermat.n } m)$

sublocale *Fmr*: *residues* $\text{fm.n } Fm$

rewrites *fm-Fm*: $\text{fm.Fn} \equiv Fm$

<proof>

lemma *fill-a-carrier*[simp, intro]: *fill-a* ∈ *fm.fermat-non-unique-carrier*
<proof>

lemma *fill-b-carrier*[simp, intro]: *fill-b* ∈ *fm.fermat-non-unique-carrier*
<proof>

lemma *fm-result-carrier*[simp, intro]: *fm-result* ∈ *fm.fermat-non-unique-carrier*
<proof>

lemma *ssc'*: *fm.to-residue-ring fm-result* = *fm.to-residue-ring fill-a* ⊗_{*Fm*} *fm.to-residue-ring fill-b*

and *fm-result* ∈ *int-lsbf-fermat.fermat-non-unique-carrier m*
<proof>

end

theorem *schoenhage-strassen-mul-correct*: *Nat-LSBF.to-nat (schoenhage-strassen-mul a b)* = *Nat-LSBF.to-nat a* * *Nat-LSBF.to-nat b*
<proof>

end

4 Running Time Formalization

theory *Schoenhage-Strassen-TM*

imports

Schoenhage-Strassen

../Preliminaries/Schoenhage-Strassen-Preliminaries

Z-mod-Fermat-TM

Karatsuba.Karatsuba-TM

Landau-Symbols.Landau-More

begin

definition *solve-special-residue-problem-tm* **where**

solve-special-residue-problem-tm n ξ η = 1 do {

n2 ← *n +_t 2*;

ξmod ← *take-tm n2 ξ*;

δ ← *int-lsbf-mod.subtract-mod-tm n2 η ξmod*;

pown ← *2[^]_t n*;

δ-shifted ← *δ >>_{nt} pown*;

δ1 ← *δ-shifted +_{nt} δ*;

ξ +_{nt} δ1

}

lemma *val-solve-special-residue-problem-tm*[simp, val-simp]:

val (solve-special-residue-problem-tm n ξ η) = *solve-special-residue-problem n ξ η*

<proof>

lemma *time-solve-special-residue-problem-tm-le*:

$time (solve-special-residue-problem-tm\ n\ \xi\ \eta) \leq 245 + 74 * 2^{\wedge} n + 55 * length\ \eta + 2 * length\ \xi$
 ⟨proof⟩

fun *combine-z-aux-tm* **where**

combine-z-aux-tm $l\ acc\ [] = 1\ rev-tm\ acc\ \gg\ concat-tm$
 | *combine-z-aux-tm* $l\ acc\ [z] = 1\ combine-z-aux-tm\ l\ (z\ \# \ acc)\ []$
 | *combine-z-aux-tm* $l\ acc\ (z1\ \# \ z2\ \# \ zs) = 1\ do\ \{$
 $(z1h,\ z1t) \leftarrow split-at-tm\ l\ z1;$
 $r \leftarrow z1t\ +_{nt}\ z2;$
 combine-z-aux-tm $l\ (z1h\ \# \ acc)\ (r\ \# \ zs)$
 }
 }

lemma *val-combine-z-aux-tm[simp, val-simp]*: $val (combine-z-aux-tm\ l\ acc\ zs) = combine-z-aux\ l\ acc\ zs$

⟨proof⟩

lemma *time-combine-z-aux-tm-le*:

assumes $\bigwedge z. z \in set\ zs \implies length\ z \leq lz$
assumes $length\ z \leq lz + 1$
assumes $l > 0$
shows $time (combine-z-aux-tm\ l\ acc\ (z\ \# \ zs)) \leq (2 * l + 2 * lz + 7) * length\ zs + 3 * (length\ acc + length\ zs) + length (concat\ acc) + length\ zs * l + lz + 9$
 ⟨proof⟩

definition *combine-z-tm* **where** *combine-z-tm* $l\ zs = 1\ combine-z-aux-tm\ l\ []\ zs$

lemma *val-combine-z-tm[simp, val-simp]*: $val (combine-z-tm\ l\ zs) = combine-z\ l\ zs$

⟨proof⟩

lemma *time-combine-z-tm-le*:

assumes $\bigwedge z. z \in set\ zs \implies length\ z \leq lz$
assumes $l > 0$
shows $time (combine-z-tm\ l\ zs) \leq 10 + (3 * l + 2 * lz + 10) * length\ zs$
 ⟨proof⟩

lemma *schoenhage-strassen-tm-termination-aux*: $\neg m < 3 \implies Suc\ (m\ div\ 2) < m$

⟨proof⟩

function *schoenhage-strassen-tm* :: $nat \Rightarrow nat-lsbf \Rightarrow nat-lsbf \Rightarrow nat-lsbf\ tm$ **where**

schoenhage-strassen-tm $m\ a\ b = 1\ do\ \{$
 $m-le-3 \leftarrow m <_t\ 3;$
 if $m-le-3$ then do {
 $ab \leftarrow a *_{nt}\ b;$
 int-lsbf-fermat.from-nat-lsbf-tm $m\ ab$
 } else do {
 $odd-m \leftarrow odd-tm\ m;$

```

n ← (if odd-m then do {
  m1 ← m +t 1;
  m1 divt 2
} else do {
  m2 ← m +t 2;
  m2 divt 2
});
n-plus-1 ← n +t 1;
n-minus-1 ← n -t 1;
n-plus-2 ← n +t 2;
oe-n ← (if odd-m then return n-plus-1 else return n);
segment-lens ← 2 ^t n-minus-1;
a' ← subdivide-tm segment-lens a;
b' ← subdivide-tm segment-lens b;
α ← map-tm (int-lsbf-mod.reduce-tm n-plus-2) a';
three-n ← 3 *t n;
pad-length ← three-n +t 5;
α-padded ← map-tm (fill-tm pad-length) α;
u ← concat-tm α-padded;
β ← map-tm (int-lsbf-mod.reduce-tm n-plus-2) b';
β-padded ← map-tm (fill-tm pad-length) β;
v ← concat-tm β-padded;
oe-n-plus-1 ← oe-n +t 1;
two-pow-oe-n-plus-1 ← 2 ^t oe-n-plus-1;
uv-length ← pad-length *t two-pow-oe-n-plus-1;
uv-unpadded ← karatsuba-mul-nat-tm u v;
uv ← ensure-length-tm uv-length uv-unpadded;
oe-n-minus-1 ← oe-n -t 1;
two-pow-oe-n-minus-1 ← 2 ^t oe-n-minus-1;
γs ← subdivide-tm pad-length uv;
γ ← subdivide-tm two-pow-oe-n-minus-1 γs;
γ0 ← nth-tm γ 0;
γ1 ← nth-tm γ 1;
γ2 ← nth-tm γ 2;
γ3 ← nth-tm γ 3;
η ← map4-tm
  (λx y z w. do {
    xmod ← take-tm n-plus-2 x;
    ymod ← take-tm n-plus-2 y;
    zmod ← take-tm n-plus-2 z;
    wmod ← take-tm n-plus-2 w;
    xy ← int-lsbf-mod.subtract-mod-tm n-plus-2 xmod ymod;
    zw ← int-lsbf-mod.subtract-mod-tm n-plus-2 zmod wmod;
    int-lsbf-mod.add-mod-tm n-plus-2 xy zw
  })
  γ0 γ1 γ2 γ3;
prim-root-exponent ← if odd-m then return 1 else return 2;
fn-carrier-len ← 2 ^t n-plus-1;
a'-carrier ← map-tm (fill-tm fn-carrier-len) a';

```


definition *oe-n-plus-two-pow-n-zeros* **where** *oe-n-plus-two-pow-n-zeros* = replicate
 (*oe-n* + 2 ^ *n*) *False*

definition *oe-n-plus-two-pow-n-one* **where** *oe-n-plus-two-pow-n-one* = append *oe-n-plus-two-pow-n-zeros*
 [*True*]

definition *z-complete* **where** *z-complete* = *z-filled* @ *z-consts*

lemmas *defs'* =

segment-lens-def *fn-carrier-len-def*
c-diffs-def *interval1-def* *interval2-def*
oe-n-plus-two-pow-n-zeros-def *oe-n-plus-two-pow-n-one-def*
z-complete-def

lemma *z-filled-def'*: *z-filled* = map (*fill segment-lens*) *z*
 ⟨*proof*⟩

lemma *z-sum-def'*: *z-sum* = combine-*z segment-lens z-complete*
 ⟨*proof*⟩

lemmas *defs''* = *defs'* *z-filled-def'* *z-sum-def'*

lemma *segment-lens-pos*: *segment-lens* > 0 ⟨*proof*⟩

lemma *length-γs*: *length γs* = 2 ^ (*oe-n* + 1)
 ⟨*proof*⟩

lemma *length-γs'*: *length γs* = 2 ^ (*oe-n* - 1) * 4
 ⟨*proof*⟩

lemma *val-nth-γ[simp, val-simp]*:

val (nth-tm γ 0) = γ ! 0
val (nth-tm γ 1) = γ ! 1
val (nth-tm γ 2) = γ ! 2
val (nth-tm γ 3) = γ ! 3

⟨*proof*⟩

lemma *val-fft1[simp, val-simp]*: *val (int-lsbfermat.fft-tm n prim-root-exponent*
A.num-blocks-carrier) =

int-lsbfermat.fft n prim-root-exponent A.num-blocks-carrier

⟨*proof*⟩

lemma *val-fft2[simp, val-simp]*: *val (int-lsbfermat.fft-tm n prim-root-exponent*
B.num-blocks-carrier) =

int-lsbfermat.fft n prim-root-exponent B.num-blocks-carrier

⟨*proof*⟩

lemma *val-iff[simp, val-simp]*: *val (int-lsbfermat.iff-tm n (prim-root-exponent * 2)*
c-dft-odds) =

*int-lsbfermat.iff n (prim-root-exponent * 2) c-dft-odds*

⟨*proof*⟩

end

lemma *val-schoenhage-strassen-tm*[*simp*, *val-simp*]:

assumes $a \in \text{int-lsbj-fermat.fermat-non-unique-carrier } m$

assumes $b \in \text{int-lsbj-fermat.fermat-non-unique-carrier } m$

shows $\text{val } (\text{schoenhage-strassen-tm } m \ a \ b) = \text{schoenhage-strassen } m \ a \ b$

<proof>

fun *schoenhage-strassen-Fm-bound* **where**

schoenhage-strassen-Fm-bound $m = (\text{if } m < 3 \text{ then } 5336 \text{ else}$

$\text{let } n = (\text{if odd } m \text{ then } (m + 1) \text{ div } 2 \text{ else } (m + 2) \text{ div } 2);$

$\text{oe-}n = (\text{if odd } m \text{ then } n + 1 \text{ else } n) \text{ in}$

$23525 * 2^m + 8093 * (n * 2^{(2 * n)}) + 8410 +$

$\text{time-karatsuba-mul-nat-bound } ((3 * n + 5) * 2^{\text{oe-}n}) +$

$4 * \text{karatsuba-lower-bound} +$

$\text{schoenhage-strassen-Fm-bound } n * 2^{(\text{oe-}n - 1)}$)

declare *schoenhage-strassen-Fm-bound.simps*[*simp del*]

lemma *time-schoenhage-strassen-tm-le*:

assumes $a \in \text{int-lsbj-fermat.fermat-non-unique-carrier } m$

assumes $b \in \text{int-lsbj-fermat.fermat-non-unique-carrier } m$

shows $\text{time } (\text{schoenhage-strassen-tm } m \ a \ b) \leq \text{schoenhage-strassen-Fm-bound } m$

<proof>

definition *karatsuba-const* **where**

karatsuba-const = (*SOME* $c. (\forall x. x > 0 \longrightarrow \text{time-karatsuba-mul-nat-bound } x \leq c * \text{nat } (\text{floor } (\text{real } x \text{ powr } \log 2 \ 3))))$)

lemma *real-divide-mult-eq*:

assumes $(c :: \text{real}) \neq 0$

shows $a / c * c = a$

<proof>

lemma *powr-unbounded*:

assumes $(c :: \text{real}) > 0$

shows *eventually* $(\lambda x. d \leq x \text{ powr } c)$ *at-top*

<proof>

lemma *time-kar-le-kar-const*:

assumes $x > 0$

shows $\text{time-karatsuba-mul-nat-bound } x \leq \text{karatsuba-const} * \text{nat } (\text{floor } (\text{real } x \text{ powr } \log 2 \ 3))$

<proof>

lemma *poly-smallo-exp*:

assumes $c > 1$

shows $(\lambda n. (\text{real } n) \text{ powr } d) \in o(\lambda n. c \text{ powr } (\text{real } n))$

<proof>

lemma *kar-aux-lem*: $(\lambda n. \text{real } (n * 2 \wedge n) \text{ powr log } 2 \ 3) \in O(\lambda n. \text{real } (2 \wedge (2 * n)))$
 ⟨proof⟩

definition *kar-aux-const* **where** *kar-aux-const* = (SOME *c*. $\forall n \geq 1. \text{real } (n * 2 \wedge n) \text{ powr log } 2 \ 3 \leq c * \text{real } (2 \wedge (2 * n))$)

lemma *kar-aux-lem-le*:

assumes $n > 0$

shows $\text{real } (n * 2 \wedge n) \text{ powr log } 2 \ 3 \leq \text{kar-aux-const} * \text{real } (2 \wedge (2 * n))$

⟨proof⟩

lemma *kar-aux-const-gt-0*: $\text{kar-aux-const} > 0$

⟨proof⟩

definition *kar-aux-const-nat* **where** *kar-aux-const-nat* = $\text{karatsuba-const} * \text{nat } [16 \text{ powr log } 2 \ 3] * \text{nat } [\text{kar-aux-const}]$

definition *s-const1* **where** *s-const1* = $55897 + 4 * \text{kar-aux-const-nat}$

definition *s-const2* **where** *s-const2* = $8410 + 4 * \text{karatsuba-lower-bound}$

function *schoenhage-strassen-Fm-bound'* :: $\text{nat} \Rightarrow \text{nat}$ **where**

$m < 3 \implies \text{schoenhage-strassen-Fm-bound}' \ m = 5336$

$| \ m \geq 3 \implies \text{schoenhage-strassen-Fm-bound}' \ m = \text{s-const1} * (m * 2 \wedge m) + \text{s-const2} + \text{schoenhage-strassen-Fm-bound}' \ ((m + 2) \text{ div } 2) * 2 \wedge ((m + 1) \text{ div } 2)$

⟨proof⟩

termination

⟨proof⟩

declare *schoenhage-strassen-Fm-bound'.simps*[simp del]

lemma *schoenhage-strassen-Fm-bound-le-schoenhage-strassen-Fm-bound'*:

shows $\text{schoenhage-strassen-Fm-bound} \ m \leq \text{schoenhage-strassen-Fm-bound}' \ m$

⟨proof⟩

definition $\gamma\text{-}0$ **where** $\gamma\text{-}0 = 2 * \text{s-const1} + \text{s-const2}$

lemma *schoenhage-strassen-Fm-bound'-oe-rec*:

assumes $n \geq 3$

shows $\text{schoenhage-strassen-Fm-bound}' \ (2 * n - 2) \leq \gamma\text{-}0 * n * 2 \wedge (2 * n - 2) + \text{schoenhage-strassen-Fm-bound}' \ n * 2 \wedge (n - 1)$

and $\text{schoenhage-strassen-Fm-bound}' \ (2 * n - 1) \leq \gamma\text{-}0 * n * 2 \wedge (2 * n - 1) + \text{schoenhage-strassen-Fm-bound}' \ n * 2 \wedge n$

⟨proof⟩

definition γ **where** $\gamma = \text{Max } \{\gamma\text{-}0, \text{schoenhage-strassen-Fm-bound}' \ 0, \text{schoenhage-strassen-Fm-bound}' \ 1, \text{schoenhage-strassen-Fm-bound}' \ 2, \text{schoenhage-strassen-Fm-bound}' \ 3\}$

lemma *schoenhage-strassen-Fm-bound'-le-aux1*:
assumes $m \leq 2^{\wedge} \text{Suc } k + 1$
shows *schoenhage-strassen-Fm-bound'* $m \leq \gamma * \text{Suc } k * 2^{\wedge} (\text{Suc } k + m)$
<proof>

lemma *schoenhage-strassen-Fm-bound'-le-aux2*:
assumes $k \geq 1$
assumes $m \leq 2^{\wedge} k + 1$
shows *schoenhage-strassen-Fm-bound'* $m \leq \gamma * k * 2^{\wedge} (k + m)$
<proof>

4.1 Multiplication in \mathbb{N}

definition *schoenhage-strassen-mul-tm* **where**
schoenhage-strassen-mul-tm $a \ b = 1$ **do** {
 $\text{bits-a} \leftarrow \text{length-tm } a \ggg \text{bitsize-tm};$
 $\text{bits-b} \leftarrow \text{length-tm } b \ggg \text{bitsize-tm};$
 $m' \leftarrow \text{max-nat-tm } \text{bits-a } \text{bits-b};$
 $m \leftarrow m' +_t 1;$
 $m\text{-plus-1} \leftarrow m +_t 1;$
 $\text{car-len} \leftarrow 2^{\wedge}_t m\text{-plus-1};$
 $\text{fill-a} \leftarrow \text{fill-tm } \text{car-len } a;$
 $\text{fill-b} \leftarrow \text{fill-tm } \text{car-len } b;$
 $\text{fm-result} \leftarrow \text{schoenhage-strassen-tm } m \ \text{fill-a } \text{fill-b};$
 $\text{int-lsbf-fermat.reduce-tm } m \ \text{fm-result}$
}

lemma *val-schoenhage-strassen-mul-tm[simp, val-simp]*:
val (*schoenhage-strassen-mul-tm* $a \ b$) = *schoenhage-strassen-mul* $a \ b$
<proof>

lemma *real-power*: $a > 0 \implies \text{real } ((a :: \text{nat})^{\wedge} x) = \text{real } a \ \text{powr } \text{real } x$
<proof>

definition *schoenhage-strassen-bound* **where**
schoenhage-strassen-bound $n = 146 * n + 218 + 4 * (\text{bitsize } n + 1) + 126 * 2^{\wedge} (\text{bitsize } n + 2) +$
 $\gamma * \text{bitsize } (\text{bitsize } n + 1) * 2^{\wedge} (\text{bitsize } (\text{bitsize } n + 1) + (\text{bitsize } n + 1))$

theorem *time-schoenhage-strassen-mul-tm-le*:
assumes $\text{length } a \leq n \ \text{length } b \leq n$
shows $\text{time } (\text{schoenhage-strassen-mul-tm } a \ b) \leq \text{schoenhage-strassen-bound } n$
<proof>

lemma *real-diff*: $a \leq b \implies \text{real } (b - a) = \text{real } b - \text{real } a$
<proof>

lemma *bitsize-le-log*: $n > 0 \implies \text{real } (\text{bitsize } n) \leq \log 2 (\text{real } n) + 1$

<proof>

lemma *powr-mono-base2*: $a \leq b \implies 2 \text{ powr } (a :: \text{real}) \leq 2 \text{ powr } b$
<proof>

lemma *log-mono-base2*: $a > 0 \implies b > 0 \implies a \leq b \implies \log 2 a \leq \log 2 b$
<proof>

lemma *log-nonneg-base2*: $x \geq 1 \implies \log 2 x \geq 0$
<proof>

lemma *powr-log-cancel-base2*: $x > 0 \implies 2 \text{ powr } (\log 2 x) = x$
<proof>

lemma *const-bigo-log*: $1 \in O(\log 2)$
<proof>

lemma *const-bigo-log-log*: $1 \in O(\lambda x. \log 2 (\log 2 x))$
<proof>

theorem *schoenhage-strassen-bound-bigo*: $\text{schoenhage-strassen-bound} \in O(\lambda n. n * \log 2 n * \log 2 (\log 2 n))$
<proof>
end

References

- [1] T. Ammer and K. Kreuzer. Number theoretic transform. *Archive of Formal Proofs*, August 2022. https://isa-afp.org/entries/Number_Theoretic_Transform.html, Formal proof development.
- [2] T. Nipkow. Verified root-balanced trees. In B.-Y. E. Chang, editor, *Asian Symposium on Programming Languages and Systems, APLAS 2017*, volume 10695 of *LNCS*, pages 255–272. Springer, 2017. <https://www21.in.tum.de/~nipkow/pubs/aplas17.pdf>.
- [3] A. Schönhage and V. Strassen. Schnelle multiplikation großer zahlen. *Computing*, 7:281–292, 1971.