

# 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

<b>1</b>	<b>Preliminaries</b>	<b>2</b>
1.1	Some Running Time Formalizations . . . . .	8
1.2	Auxiliary Lemmas for Landau Notation . . . . .	11
1.3	Multiplicative Subgroups . . . . .	16
1.4	Additive Subgroups . . . . .	19
<b>2</b>	<b>Number Theoretic Transforms in Rings</b>	<b>19</b>
2.1	Roots of Unity . . . . .	20
2.2	Primitive Roots . . . . .	24
2.3	Number Theoretic Transforms . . . . .	25
2.3.1	Inversion Rule . . . . .	27
2.3.2	Convolution Theorem . . . . .	35
2.4	Fast Number Theoretic Transforms in Rings . . . . .	38
<b>3</b>	<b>The Schoenhage-Strassen Algorithm</b>	<b>51</b>
3.1	Representing $\mathbb{Z}_{2^n}$ . . . . .	51
3.2	Representing $\mathbb{Z}_{F_n}$ . . . . .	60
3.3	Implementing FNTT in $\mathbb{Z}_{F_n}$ . . . . .	79
3.4	Final Preparations . . . . .	118
3.4.1	A special residue problem . . . . .	128
3.4.2	Subroutine for combining the final result . . . . .	134

3.5	Schoenhage-Strassen Multiplication in $\mathbb{Z}_{F_m}$ . . . . .	137
3.6	Schoenhage-Strassen Multiplication in $\mathbb{N}$ . . . . .	172
<b>4</b>	<b>Running Time Formalization</b> . . . . .	<b>174</b>
4.1	Multiplication in $\mathbb{N}$ . . . . .	213

## 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*

**by** *simp*

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

**by** *simp*

**lemma** *const-diff-mod-idem:*

**assumes** *m ≥ (n :: nat)*

*f = (λi. (m - i) mod n)*

**shows** *(∧i. i ∈ {0.. $n$ } ⇒ f (f i) = i)*

**proof** -

**fix** *i*

**assume** *i ∈ {0.. $n$ }*

**then have** *i < n* **by** *simp*

**then have** *i ≤ m* **using** *⟨n ≤ m⟩* **by** *simp*

**have** *n > 0* **using** *⟨i < n⟩* **by** *simp*

**have** *int (f (f i)) = int ((m - (m - i) mod n) mod n)*

**using** *assms* **by** *simp*

**also have** *... = (int m - int (m - i) mod int n) mod int n*

**unfolding** *zmod-int*

**using** *⟨n ≤ m⟩ int-ops(6)[of m (m - i) mod n] pos-mod-bound[of n] ⟨n > 0⟩*

**by** *(intro arg-cong2[where f = (mod)] refl)*

*(metis diff-diff-cancel less-imp-diff-less mod-le-divisor mod-less mod-self nat-int-comparison(2)*

*of-nat-less-0-iff of-nat-mod)*

**also have** *... = int i mod int n*

**using** *assms(1) ⟨i < n⟩* **by** *(simp add: mod-diff-right-eq)*

**also have** *... = int i* **using** *⟨i < n⟩* **by** *simp*

**finally show** *f (f i) = i* **by** *simp*

**qed**

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

**proof** (*intro bij-betwI*)

**show**  $(\lambda i. (m - i) \text{ mod } n) \in \{0..<n\} \rightarrow \{0..<n\}$  **by** *simp*

**show**  $(\lambda i. (m - i) \text{ mod } n) \in \{0..<n\} \rightarrow \{0..<n\}$  **by** *simp*

**show**  $\bigwedge x. n \leq m \implies x \in \{0..<n\} \implies (m - (m - x) \text{ mod } n) \text{ mod } n = x$

**using** *const-diff-mod-idem[of n]* **by** *blast*

**show**  $\bigwedge x. n \leq m \implies x \in \{0..<n\} \implies (m - (m - x) \text{ mod } n) \text{ mod } n = x$

**using** *const-diff-mod-idem[of n]* **by** *blast*

**qed**

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

**proof** (*intro bij-betwI*)

**show**  $(\lambda i. (m + i) \text{ mod } n) \in \{0..<n\} \rightarrow \{0..<n\}$  **by** *simp*

**show**  $(\lambda i. (n - (m \text{ mod } n) + i) \text{ mod } n) \in \{0..<n\} \rightarrow \{0..<n\}$  **by** *simp*

**show**  $\bigwedge x. x \in \{0..<n\} \implies (n - m \text{ mod } n + (m + x) \text{ mod } n) \text{ mod } n = x$

**proof** -

**fix**  $x$

**assume**  $x \in \{0..<n\}$

**then have**  $x < n$  **by** *simp*

**have**  $\text{int } ((n - m \text{ mod } n + (m + x) \text{ mod } n) \text{ mod } n) = (\text{int } n - \text{int } m \text{ mod } \text{int } n + (\text{int } m + \text{int } x) \text{ mod } \text{int } n) \text{ mod } \text{int } n$

**using**  $\langle x < n \rangle$  *zmod-int*

**by** (*metis less-nat-zero-code mod-le-divisor not-gr-zero of-nat-add of-nat-diff*)

**also have**  $\dots = (\text{int } n + (\text{int } x) \text{ mod } \text{int } n) \text{ mod } \text{int } n$

**by** (*metis (no-types, opaque-lifting) add commute add-diff-eq diff-diff-eq2 diff-self minus-int-code(1) mod-diff-left-eq*)

**also have**  $\dots = \text{int } x$  **using**  $\langle x < n \rangle$  *mod-add-self1* **by** *simp*

**finally show**  $(n - m \text{ mod } n + (m + x) \text{ mod } n) \text{ mod } n = x$  **by** *linarith*

**qed**

**show**  $\bigwedge y. y \in \{0..<n\} \implies (m + (n - m \text{ mod } n + y) \text{ mod } n) \text{ mod } n = y$

**proof** -

**fix**  $y$

**assume**  $y \in \{0..<n\}$

**then have**  $y < n$  **by** *simp*

**then show**  $(m + (n - m \text{ mod } n + y) \text{ mod } n) \text{ mod } n = y$

**by** (*metis (no-types, opaque-lifting) add commute add-diff-eq diff-diff-eq2 diff-self minus-int-code(1) mod-diff-left-eq*)

**qed**

**qed**

**lemma** *mod-diff-add-eq*:  $(a - b + c) \text{ mod } (m :: \text{int}) = (a \text{ mod } m - b \text{ mod } m + c \text{ mod } m) \text{ mod } m$

**proof** -

**have**  $(a - b + c) \text{ mod } m = ((a - b) \text{ mod } m + c \text{ mod } m) \text{ mod } m$

**by** (*intro mod-add-eq[symmetric]*)

**also have**  $\dots = ((a \text{ mod } m - b \text{ mod } m) \text{ mod } m + c \text{ mod } m) \text{ mod } m$

**by** (*simp only: mod-diff-eq*)

**also have**  $\dots = (a \text{ mod } m - b \text{ mod } m + c \text{ mod } m) \text{ mod } m$

by (*simp only: mod-add-left-eq*)  
**finally show**  $(a - b + c) \bmod m = (a \bmod m - b \bmod m + c \bmod m) \bmod m$  .  
**qed**

**lemma** *set-map-subseteqI*:  
**assumes**  $\bigwedge x. x \in A \implies f x \in B$   
**assumes**  $\text{set } xs \subseteq A$   
**shows**  $\text{set } (\text{map } f \text{ } xs) \subseteq B$   
**using** *assms* **by** *auto*

**lemma** *in-set-conv-nth-map2*:  
**assumes**  $z \in \text{set } (\text{map2 } f \text{ } xs \text{ } ys)$   
**shows**  $\exists i. i < \min (\text{length } xs) (\text{length } ys) \wedge z = f (xs ! i) (ys ! i)$   
**proof** –  
**from** *assms* **obtain** *i* **where**  $i < \text{length } (\text{map2 } f \text{ } xs \text{ } ys) \wedge z = \text{map2 } f \text{ } xs \text{ } ys ! i$   
**by** (*metis in-set-conv-nth*)  
**have**  $i < \min (\text{length } xs) (\text{length } ys)$   
**using**  $\langle i < \text{length } (\text{map2 } f \text{ } xs \text{ } ys) \rangle$  **by** *simp*  
**moreover** **have**  $z = f (xs ! i) (ys ! i)$   
**using**  $\langle z = \text{map2 } f \text{ } xs \text{ } ys ! i \rangle \langle i < \min (\text{length } xs) (\text{length } ys) \rangle$  **by** *simp*  
**ultimately show** *?thesis* **by** *blast*  
**qed**

**lemma** *set-map2*:  
**assumes**  $z \in \text{set } (\text{map2 } f \text{ } xs \text{ } ys)$   
**shows**  $\exists x y. x \in \text{set } xs \wedge y \in \text{set } ys \wedge z = f x y$   
**using** *in-set-conv-nth-map2[OF assms]* **by** *force*

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

**proof**  
**fix** *z*  
**assume**  $z \in \text{set } (\text{map2 } f \text{ } xs \text{ } ys)$   
**then** **obtain** *x y* **where**  $z = f x y \wedge x \in \text{set } xs \wedge y \in \text{set } ys$   
**using** *set-map2* **by** *meson*  
**then** **show**  $z \in C$  **using** *assms* **by** *auto*  
**qed**

**lemma** *in-set-conv-nth-map3*:  
**assumes**  $w \in \text{set } (\text{map3 } f \text{ } xs \text{ } ys \text{ } zs)$   
**shows**  $\exists i. i < \min (\min (\text{length } xs) (\text{length } ys)) (\text{length } zs) \wedge w = f (xs ! i) (ys ! i) (zs ! i)$   
**proof** –  
**from** *assms* **obtain** *i* **where**  $i < \text{length } (\text{map3 } f \text{ } xs \text{ } ys \text{ } zs) \wedge w = \text{map3 } f \text{ } xs \text{ } ys \text{ } zs ! i$   
**by** (*metis in-set-conv-nth*)  
**have**  $i < \min (\min (\text{length } xs) (\text{length } ys)) (\text{length } zs)$

**using**  $\langle i < \text{length } (\text{map3 } f \text{ } xs \text{ } ys \text{ } zs) \rangle$   
**unfolding** *map3-as-map* **by** *simp*  
**moreover have**  $w = f (xs ! i) (ys ! i) (zs ! i)$   
**using**  $\langle w = \text{map3 } f \text{ } xs \text{ } ys \text{ } zs ! i \rangle \langle i < \min (\min (\text{length } xs) (\text{length } ys)) (\text{length } zs) \rangle$   
**unfolding** *map3-as-map* **by** *simp*  
**ultimately show** *?thesis* **by** *blast*  
**qed**

**lemma** *set-map3*:

**assumes**  $w \in \text{set } (\text{map3 } f \text{ } xs \text{ } ys \text{ } zs)$   
**shows**  $\exists x \ y \ z. x \in \text{set } xs \wedge y \in \text{set } ys \wedge z \in \text{set } zs \wedge w = f \ x \ y \ z$   
**using** *in-set-conv-nth-map3[OF assms]* **by** *force*

**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**  $\text{set } (\text{map3 } f \text{ } xs \text{ } ys \text{ } zs) \subseteq D$

**proof**

**fix**  $w$   
**assume**  $w \in \text{set } (\text{map3 } f \text{ } xs \text{ } ys \text{ } zs)$   
**then obtain**  $x \ y \ z$  **where**  $w = f \ x \ y \ z \ x \in \text{set } xs \ y \in \text{set } ys \ z \in \text{set } zs$   
**using** *set-map3* **by** *meson*  
**then show**  $w \in D$  **using** *assms* **by** *fastforce*

**qed**

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

**apply** (*induction zs arbitrary: xs ys*)  
**subgoal by** *simp*  
**subgoal for**  $z \ zs \ xs \ ys$  **by** (*cases xs; cases ys; simp*)  
**done**

**definition** *rotate-left* ::  $\text{nat} \Rightarrow 'a \ \text{list} \Rightarrow 'a \ \text{list}$  **where**

$\text{rotate-left } k \ xs = (\text{let } (xs1, xs2) = \text{split-at } (k \ \text{mod } \text{length } xs) \ xs \ \text{in } xs2 \ @ \ xs1)$

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

**unfolding** *rotate-left-def* **by** (*simp add: rotate-drop-take*)

**definition** *rotate-right* **where**

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

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

**unfolding** *rotate-right-def* **by** *simp*

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

**proof** (*cases xs = []*)

**case** *True*

```

then show ?thesis unfolding rotate-right-def by simp
next
  case False
  then have length xs > 0 by simp
  have rotate-right k (rotate k xs) = rotate (length xs - k mod length xs + k) xs
    by (simp add: rotate-rotate rotate-right-def)
  also have ... = rotate (length xs + (k - k mod length xs)) xs
    using mod-le-divisor[of length xs k] ⟨length xs > 0⟩ by simp
  also have ... = rotate ((length xs + (k - k mod length xs)) mod length xs) xs
    using rotate-conv-mod by simp
  also have ... = rotate ((k - k mod length xs) mod length xs) xs
    by (metis mod-add-self1)
  also have ... = rotate 0 xs
    by simp
  also have ... = xs by simp
  finally show ?thesis .
qed
lemma rotate-rotate-right[simp]: rotate k (rotate-right k xs) = xs
proof -
  have rotate k (rotate-right k xs) = rotate (k + (length xs - k mod length xs)) xs
    by (simp add: rotate-rotate rotate-right-def)
  also have ... = rotate-right k (rotate k xs)
    by (simp add: rotate-rotate add.commute rotate-right-def)
  finally show ?thesis using rotate-right-rotate by metis
qed

value rotate 5 [1::nat..<8]
value rotate-right 3 [True, False, False]

lemma rotate-right-append: rotate-right (length q) (l @ q) = q @ l
  unfolding rotate-right-def rotate-left-rotate
  using rotate-append[of l q]
  by (metis length-rev rev-append rev-rev-ident rotate-append rotate-rev)

lemma rotate-right-conv-mod: rotate-right n xs = rotate-right (n mod length xs)
xs
  unfolding rotate-right-def by simp

lemma mod-diff-right-eq-nat:
  assumes (a::nat) ≥ b
  shows (a - b) mod m = (a - (b mod m)) mod m
proof -
  have int ((a - b) mod m) = (int (a - b)) mod int m
    using zmod-int by presburger
  also have ... = (int a - int b) mod int m
    using assms by (simp add: of-nat-diff)
  also have ... = (int a - (int b mod int m)) mod int m
    using mod-diff-right-eq by metis
  also have ... = (int a - int (b mod m)) mod int m

```

**using** *zmod-int* **by** *presburger*  
**also have** ... = (int (a - (b mod m))) mod int m  
**by** (*metis calculation diff-diff-cancel diff-is-0-eq' less-imp-diff-less less-le-not-le mod-less-eq-dividend of-nat-diff verit-comp-simplify1* (3) *zmod-int*)  
**also have** ... = int ((a - (b mod m)) mod m)  
**using** *zmod-int* **by** *presburger*  
**finally show** ?*thesis* **by** *simp*  
**qed**

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

**proof** (*cases* xs = [])

**case** *True*

**then show** ?*thesis* **unfolding** *rotate-right-def* **by** *simp*

**next**

**case** *False*

**then have** rotate-right k (rotate-right l xs) = rotate (length xs - k mod length xs + (length xs - l mod length xs)) xs

**unfolding** *rotate-right-def* **by** (*simp add: rotate-rotate*)

**also have** ... = rotate ((length xs + length xs) - (k mod length xs + l mod length xs)) xs

**using** *False* **by** *simp*

**also have** ... = rotate (((length xs + length xs) - (k mod length xs + l mod length xs)) mod length xs) xs

**using** *rotate-conv-mod* **by** *blast*

**also have** ... = rotate (((length xs + length xs) - (k mod length xs + l mod length xs) mod length xs) mod length xs) xs

**using** *mod-diff-right-eq-nat* *False*

**by** (*metis add-le-mono length-greater-0-conv mod-le-divisor*)

**also have** ... = rotate (((length xs + length xs) - ((k + l) mod length xs) mod length xs) mod length xs) xs

**by** (*simp add: mod-add-eq*)

**also have** ... = rotate ((length xs + (length xs - ((k + l) mod length xs))) mod length xs) xs

**using** *False* **by** *simp*

**also have** ... = rotate ((length xs - ((k + l) mod length xs)) mod length xs) xs

**by** *simp*

**also have** ... = rotate (length xs - ((k + l) mod length xs)) xs

**using** *rotate-conv-mod* **by** *metis*

**also have** ... = rotate-right (k + l) xs **unfolding** *rotate-right-def* **by** *simp*

**finally show** ?*thesis* .

**qed**

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

**by** (*simp add: nth-rotate add.commute rotate-right-def*)

**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]*: val (*zip-tm* xs ys) = *zip* xs ys

**by** (*induction* xs ys rule: *zip-tm.induct*; *simp*)

**lemma** *time-zip-tm[simp]*: time (*zip-tm* xs ys) = min (length xs) (length ys) + 1

**by** (*induction* xs ys rule: *zip-tm.induct*; *simp*)

**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]*: val (*map3-tm* f xs ys zs) = *map3* ( $\lambda$ x y z.

val (f x y z)) xs ys zs

**by** (*induction* f xs ys zs rule: *map3-tm.induct*; *simp*)

**lemma** *time-map3-tm-bounded*:

**assumes**  $\bigwedge$ x y z. x  $\in$  set xs  $\implies$  y  $\in$  set ys  $\implies$  z  $\in$  set zs  $\implies$  time (f x y z)  $\leq$  c

**shows** time (*map3-tm* f xs ys zs)  $\leq$  (c + 1) \* min (min (length xs) (length ys)) (length zs) + 1

**using** *assms* **proof** (*induction* f xs ys zs rule: *map3.induct*)

**case** (1 f x xs y ys z zs)

**then have** *ih*: time (*map3-tm* f xs ys zs)  $\leq$  (c + 1) \* min (min (length xs) (length ys)) (length zs) + 1

**by** *simp*

**from** 1.premis **have** *fxyz*: time (f x y z)  $\leq$  c **by** *simp*

```

show ?case
  unfolding map3-tm.simps tm-time-simps
  apply (estimation estimate: ih)
  apply (estimation estimate: fxyz)
  by simp
qed simp-all

```

```

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

```

```

lemma val-map4-tm[simp, val-simp]: val (map4-tm f xs ys zs ws) = map4 (λx y z
w. val (f x y z w)) xs ys zs ws
by (induction f xs ys zs ws rule: map4-tm.induct; simp)

```

**lemma** time-map4-tm-bounded:

```

assumes ∧x y z w. x ∈ set xs ⇒ y ∈ set ys ⇒ z ∈ set zs ⇒ w ∈ set ws ⇒
time (f x y z w) ≤ c
shows time (map4-tm f xs ys zs ws) ≤ (c + 1) * min (min (min (length xs)
(length ys)) (length zs)) (length ws) + 1
using assms proof (induction f xs ys zs ws rule: map4.induct)
case (1 f x xs y ys z zs w ws)
then have ih: time (map4-tm f xs ys zs ws) ≤ (c + 1) * min (min (min (length
xs) (length ys)) (length zs)) (length ws) + 1
by simp
from 1.prem1 have fxyzw: time (f x y z w) ≤ c by simp
show ?case
  unfolding map4-tm.simps tm-time-simps
  apply (estimation estimate: ih)
  apply (estimation estimate: fxyzw)
  by simp
qed simp-all

```

**definition** map2-tm **where**

```

map2-tm f xs ys =1 do {
  xys ← zip-tm xs ys;
  map-tm (λ(x,y). f x y) xys
}

```

```

lemma val-map2-tm[simp, val-simp]: val (map2-tm f xs ys) = map2 (λx y. val (f
x y)) xs ys
unfolding map2-tm-def by (simp split: prod.splits)

```

**lemma** time-map2-tm-bounded:

**assumes**  $\text{length } xs = \text{length } ys$   
**assumes**  $\bigwedge x y. x \in \text{set } xs \implies y \in \text{set } ys \implies \text{time } (f x y) \leq c$   
**shows**  $\text{time } (\text{map2-tm } f xs ys) \leq (c + 2) * \text{length } xs + 3$   
**proof** –  
**have**  $\text{time } (\text{map2-tm } f xs ys) = \text{length } xs + 2 + \text{time } (\text{map-tm } (\lambda(x, y). f x y) (\text{zip } xs ys))$   
**unfolding**  $\text{map2-tm-def}$  **by** ( $\text{simp add: assms}$ )  
**also have**  $\dots \leq \text{length } xs + 2 + ((c + 1) * \text{length } (\text{zip } xs ys) + 1)$   
**apply** ( $\text{intro add-mono order.refl time-map-tm-bounded}$ )  
**using**  $\text{assms}$  **by** ( $\text{auto split: prod.splits elim: in-set-zipE}$ )  
**also have**  $\dots = (c + 2) * \text{length } xs + 3$   
**using**  $\text{assms}$  **by**  $\text{simp}$   
**finally show**  $?thesis$  .  
**qed**

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

**lemma**  $\text{val-rotate-left-tm}[\text{simp}, \text{val-simp}]$ :  $\text{val } (\text{rotate-left-tm } k xs) = \text{rotate-left } k xs$   
**unfolding**  $\text{rotate-left-tm-def}$   $\text{rotate-left-def}$  **by** ( $\text{simp add: Let-def}$ )

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

**proof** –  
**obtain**  $xs1 xs2$  **where**  $1: (xs1, xs2) = \text{split-at } (k \text{ mod } \text{length } xs) xs$   
**by**  $\text{simp}$   
**then have**  $2: \text{length } xs2 \leq \text{length } xs$  **by**  $\text{simp}$   
**have**  $\text{time } (\text{rotate-left-tm } k xs) =$   
 $\text{time } (\text{length-tm } xs) +$   
 $\text{time } (k \text{ mod}_t (\text{length } xs)) +$   
 $\text{time } (\text{split-at-tm } (k \text{ mod } \text{length } xs) xs) + \text{time } (xs2 @_t xs1) + 1$   
**unfolding**  $\text{rotate-left-tm-def}$   $\text{tm-time-simps}$   $\text{val-length-tm}$   $\text{val-mod-nat-tm}$   $\text{val-split-at-tm}$   
 $\text{Product-Type.prod.case } 1[\text{symmetric}]$  **by**  $\text{simp}$   
**also have**  $\dots \leq (\text{length } xs + 1) + (8 * k + 2 * \text{length } xs + 7) + (2 * \text{length } xs$   
 $+ 3) + (\text{length } xs + 1) + 1$   
**apply** ( $\text{intro add-mono order.refl}$ )  
**subgoal by**  $\text{simp}$   
**subgoal by** ( $\text{estimation estimate: time-mod-nat-tm-le}$ ) ( $\text{rule order.refl}$ )  
**subgoal by** ( $\text{simp add: time-split-at-tm}$ )  
**subgoal by** ( $\text{simp add: 2}$ )  
**done**  
**also have**  $\dots = 13 + 6 * \text{length } xs + 8 * k$  **by**  $\text{simp}$   
**finally show**  $?thesis$  **by**  $\text{simp}$

qed

**definition** *rotate-right-tm* :: nat  $\Rightarrow$  'a list  $\Rightarrow$  'a list tm **where**

```
rotate-right-tm k xs =1 do {  
  lenxs  $\leftarrow$  length-tm xs;  
  kmod  $\leftarrow$  k modt lenxs;  
  rk  $\leftarrow$  lenxs -t kmod;  
  rotate-left-tm rk xs  
}
```

**lemma** *val-rotate-right-tm*[*simp*, *val-simp*]: val (rotate-right-tm k xs) = rotate-right k xs

**unfolding** *rotate-right-tm-def* *rotate-right-def* **by** (*simp add: Let-def*)

**lemma** *time-rotate-right-tm-le*: time (rotate-right-tm k xs)  $\leq$  23 + 26 \* max k (length xs)

**proof** -

```
have time (rotate-right-tm k xs) =  
  time (length-tm xs) +  
  time (k modt length xs) +  
  time (length xs -t (k mod length xs)) +  
  time (rotate-left-tm (length xs - k mod length xs) xs) + 1
```

**unfolding** *rotate-right-tm-def* *tm-time-simps* *val-length-tm* *val-mod-nat-tm* *val-minus-nat-tm* **by** *simp*

```
also have ...  $\leq$  (length xs + 1) +  
  (8 * k + 2 * length xs + 7) +  
  (length xs + 1) +  
  (14 * length xs + 13) + 1
```

**apply** (*intro add-mono order.refl*)

**subgoal by** *simp*

**subgoal by** (*estimation estimate: time-mod-nat-tm-le*) (*rule order.refl*)

**subgoal by** *simp*

**subgoal by** (*estimation estimate: time-rotate-left-tm-le*) *simp*

**done**

```
also have ... = 23 + 18 * length xs + 8 * k by simp
```

```
finally show ?thesis by simp
```

qed

## 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**  $\exists c. (\forall x. x \geq n0 \implies f\ x \leq c * g\ x)$

**proof** -

```
from landau-o.bigE[OF  $\langle f \in O(g) \rangle$ ]
```

**obtain** *c-real* **where** *eventually* ( $\lambda x. norm (f\ x) \leq c\text{-real} * norm (g\ x)$ ) *sequentially*

```

    by auto
  then have eventually ( $\lambda x. f x \leq c\text{-real} * g x$ ) at-top by simp
  then obtain n1 where f-le-g-real:  $f x \leq c\text{-real} * g x$  if  $x \geq n1$  for x
    using eventually-at-top-linorder by meson
  define c where  $c = \text{nat} (\text{ceiling } c\text{-real})$ 
  then have f-le-g:  $f x \leq c * g x$  if  $x \geq n1$  for x
  proof -
    have  $\text{real} (f x) \leq c\text{-real} * \text{real} (g x)$  using f-le-g-real[OF that] .
    also have  $\dots \leq \text{real } c * \text{real} (g x)$  unfolding c-def
      by (simp add: mult-mono real-nat-ceiling-ge)
    also have  $\dots = \text{real} (c * g x)$  by simp
    finally show ?thesis by linarith
  qed
  consider  $n1 \leq n0 \mid n1 > n0$  by linarith
  then show ?thesis
  proof cases
    case 1
      then show ?thesis
        apply (intro exI[of - c]) using f-le-g by simp
    next
      case 2
        define M where  $M = \text{Max} (f \text{ ` } \{n0..<n1\})$ 
        define C where  $C = (\text{max } M 1) * (\text{max } c 1)$ 
        have  $f x \leq C * g x$  if  $x \geq n0$  for x
        proof (cases  $x < n1$ )
          case True
            then have  $f x \leq M$ 
              unfolding M-def using 2
              by (intro Max.coboundedI; simp add: that)
            also have  $\dots \leq C$  unfolding C-def
              using nat-mult-max-right by auto
            also have  $\dots \leq C * g x$ 
              using assms(2)[OF that] by simp
            finally show ?thesis .
          next
            case False
              then have  $f x \leq c * g x$  using f-le-g by simp
              also have  $\dots \leq C * g x$  unfolding C-def using nat-mult-max-left
                by simp
              finally show ?thesis .
        qed
      then show ?thesis by blast
    qed
  qed
  lemma eventually-early-real:
    fixes  $f g :: \text{nat} \Rightarrow \text{real}$ 
    assumes  $f \in O(g)$ 
    assumes  $\bigwedge x. x \geq n0 \implies f x \geq 0 \wedge g x \geq 1$ 

```

```

shows  $\exists c. (\forall x \geq n0. f x \leq c * g x)$ 
proof -
  from landau-o.bigE[OF  $\langle f \in O(g) \rangle$ ]
  obtain c where eventually  $(\lambda x. norm (f x) \leq c * norm (g x))$  at-top
  by auto
  then obtain n1 where f-le-g:  $norm (f x) \leq c * norm (g x)$  if  $x \geq n1$  for x
  using eventually-at-top-linorder by meson
  consider  $n1 \leq n0 \mid n1 > n0$  by linarith
  then show ?thesis
  proof cases
    case 1
      then show ?thesis
      apply (intro exI[of - c] allI impI)
      subgoal for x using f-le-g[of x] assms(2)[of x] by simp
      done
    next
      case 2
        define M where  $M = Max (f \{n0..<n1\})$ 
        define C where  $C = (max M 1) * (max c 1)$ 
        then have  $C \geq 1$  using mult-mono[OF max.cobounded2[of 1 M] max.cobounded2[of
1 c]] by argo
        have  $C \geq c$  unfolding C-def using mult-mono[OF max.cobounded2[of 1 M]
max.cobounded1[of c 1]]
        by linarith
        have  $f x \leq C * g x$  if  $x \geq n0$  for x
        proof (cases x < n1)
          case True
            then have  $f x \leq M$ 
            unfolding M-def using 2
            by (intro Max.coboundedI; simp add: that)
            also have  $\dots \leq C$  unfolding C-def
            using mult-mono[OF max.cobounded1[of M 1] max.cobounded2[of 1 c]] by
simp
            also have  $\dots \leq C * g x$ 
            using assms(2)[OF that] mult-left-mono[of 1 g x C]  $\langle C \geq 1 \rangle$  by argo
            finally show ?thesis .
          next
            case False
              then have  $f x \leq c * g x$  using f-le-g[of x] assms(2)[OF that] by simp
              also have  $\dots \leq C * g x$  apply (intro mult-mono[OF  $\langle C \geq c \rangle$ ])
              subgoal by (rule order.refl)
              subgoal using  $\langle C \geq 1 \rangle$  by simp
              subgoal using assms(2)[OF that] by simp
              done
              finally show ?thesis .
        qed
      then show ?thesis by blast
    qed
  qed

```

**lemma** *floor-in-nat-iff*:  $\text{floor } x \in \mathbb{N} \longleftrightarrow x \geq 0$   
**proof**  
  **assume**  $\text{floor } x \in \mathbb{N}$   
  **then obtain**  $n$  **where**  $\text{floor } x = \text{of-nat } n$  **unfolding** *Nats-def* **by** *auto*  
  **then have**  $\text{floor } x \geq 0$  **using** *of-nat-0-le-iff* **by** *simp*  
  **then show**  $x \geq 0$  **by** *simp*  
**next**  
  **assume**  $0 \leq x$   
  **then have**  $\text{floor } x \geq 0$  **by** *simp*  
  **then obtain**  $n$  **where**  $\text{floor } x = \text{of-nat } n$  **using** *nat-0-le* **by** *metis*  
  **then show**  $\text{floor } x \in \mathbb{N}$  **unfolding** *Nats-def* **by** *simp*  
**qed**

**lemma** *bigO-floor*:  
  **fixes**  $f :: \text{nat} \Rightarrow \text{real}$   
  **fixes**  $g :: \text{nat} \Rightarrow \text{real}$   
  **assumes**  $(\lambda x. \text{real } (f x)) \in O(g)$   
  **assumes** *eventually*  $(\lambda x. g x \geq 1)$  *at-top*  
  **shows**  $(\lambda x. \text{real } (f x)) \in O(\lambda x. \text{real } (\text{nat } (\text{floor } (g x))))$   
**proof** –  
  **have** *ineq*:  $x \leq 2 * \text{real-of-int } (\text{floor } x)$  **if**  $x \geq 1$  **for**  $x :: \text{real}$   
  **proof** –  
    **have**  $x \leq \text{real-of-int } (\text{floor } x) + 1$   
    **by** (*rule real-of-int-floor-add-one-ge*)  
    **also have**  $\dots \leq 2 * \text{real-of-int } (\text{floor } x)$   
    **using** *that* **by** *simp*  
    **finally show** *?thesis* .  
  **qed**  
  **obtain**  $c$  **where**  $c > 0$  **and** *f-le-g*: *eventually*  $(\lambda x. \text{real } (f x) \leq c * \text{norm } (g x))$   
*at-top*  
  **using** *landau-o.bigE[OF assms(1)]* **by** *auto*  
  **have** *eventually*  $(\lambda x. g x \leq 2 * \text{of-int } (\text{floor } (g x)))$  *at-top*  
  **using** *eventually-rev-mp[OF assms(2), of  $\lambda x. g x \leq 2 * \text{of-int } (\text{floor } (g x))$ ]*  
  **using** *assms(2)* *ineq* **by** *simp*  
  **then have** *1*: *eventually*  $(\lambda x. c * g x \leq (2 * c) * \text{of-int } (\text{floor } (g x)))$  *at-top*  
  **using** *eventually-mp[of  $\lambda x. g x \leq 2 * \text{of-int } (\text{floor } (g x))$   $\lambda x. c * g x \leq (2 * c) * \text{of-int } (\text{floor } (g x))$ ]*  
  **using**  $\langle c > 0 \rangle$  **by** *simp*  
  **have** *2*: *eventually*  $(\lambda x. c * \text{norm } (g x) = c * g x)$  *at-top*  
  **using** *eventually-rev-mp[OF assms(2)]* **by** *simp*  
  **have** *3*: *eventually*  $(\lambda x. c * \text{norm } (g x) \leq (2 * c) * \text{of-int } (\text{floor } (g x)))$  *at-top*  
  **apply** (*intro eventually-rev-mp[OF eventually-conj[OF 1 2], of  $\lambda x. c * \text{norm } (g x) \leq (2 * c) * \text{of-int } (\text{floor } (g x))$ ]*)  
  **apply** (*intro always-eventually allI impI*)  
  **by** *argo*  
  **have** *4*: *eventually*  $(\lambda x. \text{real } (f x) \leq (2 * c) * \text{of-int } (\text{floor } (g x)))$  *at-top*  
  **apply** (*intro eventually-rev-mp[OF eventually-conj[OF f-le-g 3], where  $Q = \lambda x. \text{real } (f x) \leq (2 * c) * \text{of-int } (\text{floor } (g x))$ ]*)

```

    by simp
  show ?thesis
    apply (intro landau-o.bigI[where c = 2 * c])
    subgoal using ⟨c > 0⟩ by argo
    subgoal apply (intro eventually-rew-mp[OF eventually-conj[OF 4 assms(2)]],
where Q = λx. norm (real (f x)) ≤ (2 * c) * norm (real (nat [g x])))
    by simp
  done
qed

```

```

end
theory Schoenhage-Strassen-Ring-Lemmas
  imports HOL-Algebra.Ring HOL-Algebra.Multiplicative-Group
begin

```

```

context cring
begin

```

```

lemma diff-diff:

```

```

  assumes a ∈ carrier R b ∈ carrier R c ∈ carrier R
  shows a ⊖ (b ⊖ c) = a ⊖ b ⊕ c
  using assms by algebra

```

```

lemma minus-eq-mult-one:

```

```

  assumes a ∈ carrier R
  shows ⊖ a = (⊖ 1) ⊗ a
  using assms by algebra

```

```

lemma diff-eq-add-mult-one:

```

```

  assumes a ∈ carrier R b ∈ carrier R
  shows a ⊖ b = a ⊕ (⊖ 1) ⊗ b
  using assms by algebra

```

```

lemma minus-cancel:

```

```

  assumes a ∈ carrier R b ∈ carrier R
  shows a ⊖ b ⊕ b = a
  using assms by algebra

```

```

lemma assoc4:

```

```

  assumes a ∈ carrier R b ∈ carrier R c ∈ carrier R d ∈ carrier R
  shows a ⊗ (b ⊗ (c ⊗ d)) = a ⊗ b ⊗ c ⊗ d
  using assms by algebra

```

```

lemma diff-sum:

```

```

  assumes a ∈ carrier R b ∈ carrier R c ∈ carrier R d ∈ carrier R
  shows (a ⊖ c) ⊕ (b ⊖ d) = (a ⊕ b) ⊖ (c ⊕ d)
  using assms by algebra

```

```

end

```

```

lemma (in ring) inv-cancel-left:

```

```

  assumes x ∈ carrier R
  assumes y ∈ carrier R
  assumes z ∈ Units R

```

**assumes**  $x = z \otimes y$   
**shows**  $\text{inv } z \otimes x = y$   
**using** *assms*  
**by** (*metis Units-closed Units-inv-closed Units-l-inv l-one m-assoc*)

**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**  $x \otimes (y \ominus z) = x \otimes y \ominus x \otimes z$   
**using** *assms* **by** *algebra*

**lemma** (**in** *group*)  
**assumes**  $x \in \text{carrier } G$   
**shows**  $\bigwedge i. i \in \{1..<\text{ord } x\} \implies x [\wedge] i \neq \mathbf{1}$   
**using** *assms* **using** *pow-eq-id* **by** *auto*

### 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]*:  $\text{carrier } M = X$  **using** *M-def* **by** *auto*

**lemma** *one-eq*:  $\mathbf{1}_M = \mathbf{1}$  **using** *M-def* **by** *simp*

**lemma** *mult-eq*:  $a \otimes_M b = a \otimes b$  **using** *M-def* **by** *simp*

**lemma** *inv-eq*:  
**assumes**  $x \in X$   
**shows**  $\text{inv}_M x = \text{inv } x$   
**proof** (*intro comm-inv-char[symmetric]*)  
**show**  $x \in \text{carrier } R$  **using** *assms Units-subset* **by** *blast*  
**from** *assms* **have**  $\text{inv}_M x \in X$  **using** *group.inv-closed[OF M-group]* **by** *simp*  
**then** **show**  $\text{inv}_M x \in \text{carrier } R$  **using** *Units-subset* **by** *blast*  
**have**  $x \otimes_M \text{inv}_M x = \mathbf{1}_M$   
**using** *group.Units-eq[OF M-group]* *monoid.Units-r-inv[OF group.is-monoid[OF M-group]]*  
**using** *assms* **by** *simp*  
**then** **show**  $x \otimes \text{inv}_M x = \mathbf{1}$  **using** *M-def* **by** *simp*  
**qed**

**lemma** *nat-pow-eq*:  $x [\wedge]_M (m :: \text{nat}) = x [\wedge] m$   
**by** (*induction m*) (*simp-all add: M-def*)

```

lemma int-pow-eq:
  assumes  $x \in X$ 
  shows  $x [\wedge]_M (i :: \text{int}) = x [\wedge] i$ 
proof (cases  $i \geq 0$ )
  case True
  then have  $x [\wedge]_M i = x [\wedge]_M (\text{nat } i)$ 
    by simp
  also have  $\dots = x [\wedge] (\text{nat } i)$ 
    using nat-pow-eq by simp
  also have  $\dots = x [\wedge] i$ 
    using True by simp
  finally show ?thesis .
next
  case False
  then have  $x [\wedge]_M i = \text{inv}_M (x [\wedge]_M (\text{nat } (- i)))$ 
    using int-pow-def2[of M] by presburger
  also have  $\dots = \text{inv} (x [\wedge]_M (\text{nat } (- i)))$ 
    apply (intro inv-eq)
    using monoid.nat-pow-closed[OF group.is-monoid[OF M-group]] assms by simp
  also have  $\dots = \text{inv} (x [\wedge] (\text{nat } (- i)))$ 
    by (simp add: nat-pow-eq)
  also have  $\dots = x [\wedge] i$ 
    using int-pow-def2 False by (metis leI)
  finally show ?thesis .
qed

end

context cring
begin

interpretation units-group: group units-of R
  by (rule units-group)

lemma units-subgroup: multiplicative-subgroup R (Units R) (units-of R)
  apply unfold-locales unfolding units-of-def by simp-all

interpretation units-subgroup: multiplicative-subgroup R Units R units-of R
  by (rule units-subgroup)

lemma inv-nat-pow:
  assumes  $a \in \text{Units } R$ 
  shows  $\text{inv} (a [\wedge] (b :: \text{nat})) = \text{inv } a [\wedge] b$ 
proof –
  have  $\text{inv} (a [\wedge] b) = \text{inv}_{\text{units-of } R} (a [\wedge]_{\text{units-of } R} b)$ 
    using assms units-subgroup.nat-pow-eq units-subgroup.inv-eq Units-pow-closed
by simp
  also have  $\dots = \text{inv}_{\text{units-of } R} a [\wedge]_{\text{units-of } R} b$ 

```

```

    apply (intro group.nat-pow-inv[OF units-group, symmetric])
    using assms units-subgroup.carrier-M by argo
  also have ... = inv a [^] b
    using assms units-subgroup.nat-pow-eq units-subgroup.inv-eq by simp
  finally show ?thesis .
qed
lemma int-pow-mult:
  fixes m1 m2 :: int
  assumes x ∈ Units R
  shows x [^] m1 ⊗ x [^] m2 = x [^] (m1 + m2)
  using units-group.int-pow-mult[of x]
  unfolding units-subgroup.carrier-M
  using assms units-subgroup.int-pow-eq[OF assms]
  by (simp add: units-subgroup.mult-eq)
lemma int-pow-pow:
  fixes m1 m2 :: int
  assumes x ∈ Units R
  shows (x [^] m1) [^] m2 = x [^] (m1 * m2)
  using units-group.int-pow-pow[of x] assms
  unfolding units-subgroup.carrier-M
  using units-group.int-pow-closed units-subgroup.int-pow-eq by auto
lemma int-pow-one:
  1 [^] (i :: int) = 1
  using units-group.int-pow-one[of i]
  using units-subgroup.int-pow-eq[OF Units-one-closed] units-subgroup.one-eq by
simp
lemma int-pow-closed:
  assumes x ∈ Units R
  shows x [^] (i :: int) ∈ Units R
  using units-group.int-pow-closed units-subgroup.carrier-M assms units-subgroup.int-pow-eq
  by simp
lemma units-of-int-pow: μ ∈ Units R ⇒ μ [^]_{(units-of R)} i = μ [^] (i :: int)
  using units-of-pow[of μ]
  apply (simp add: int-pow-def)
  by (metis Units-pow-closed nat-pow-def units-of-inv)
lemma units-int-pow-neg: μ ∈ Units R ⇒ (inv μ) [^] (n :: int) = μ [^] (- n)
  by (metis Units-inv-Units units-of-int-pow units-group.int-pow-inv units-group.int-pow-neg
  units-of-carrier units-of-inv)
lemma units-inv-int-pow: μ ∈ Units R ⇒ inv μ = μ [^] (- (1 :: int))
  using units-int-pow-neg[of μ 1 :: int]
  by (simp add: int-pow-def2)
lemma inv-prod: μ ∈ Units R ⇒ ν ∈ Units R ⇒ inv (μ ⊗ ν) = inv ν ⊗ inv μ
  by (metis Units-m-closed group.inv-mult-group units-group units-of-carrier units-of-inv
  units-of-mult)

```

```

lemma powers-of-negative:
  fixes  $r :: \text{nat}$ 
  assumes  $x \in \text{carrier } R$ 
  shows  $\text{even } r \implies (\ominus x) [\uparrow] r = x [\uparrow] r$   $\text{odd } r \implies (\ominus x) [\uparrow] r = \ominus (x [\uparrow] r)$ 
  using assms by (induction r) (simp-all add: l-minus r-minus)

end

```

## 1.4 Additive Subgroups

```

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

```

```

lemma carrier-M[simp]: carrier  $M = X$ 
  unfolding M-def by simp

```

```

lemma one-eq:  $\mathbf{1}_M = \mathbf{0}$  unfolding M-def by simp

```

```

lemma mult-eq:  $a \otimes_M b = a \oplus b$ 
  unfolding M-def by simp

```

```

lemma inv-eq:
  assumes  $a \in X$ 
  shows  $\text{inv}_M a = \ominus a$ 
  apply (intro sum-zero-eq-neg set-mp[OF Units-subset] assms)
  subgoal using group.inv-closed[OF M-group] assms unfolding carrier-M by
simp
  subgoal
    unfolding mult-eq[symmetric] one-eq[symmetric]
    apply (intro group.l-inv M-group)
    unfolding carrier-M using assms .
  done

```

**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 :: nat
  assumes a ≠ 0
  assumes k = Max {s. p ^ s dvd a}
  assumes r = a div (p ^ k)
  assumes prime p
  shows a = r * p ^ k coprime p r
  subgoal
  proof -
    have p ^ 0 dvd a by simp
    then have {s. p ^ s dvd a} ≠ {} by blast
    with assms have p ^ k dvd a
      by (metis Max-in finite-divisor-powers mem-Collect-eq not-prime-unit)
    with assms show ?thesis by simp
  qed
  subgoal
  proof (rule ccontr)
    assume ¬ coprime p r
    then have p dvd r using prime-imp-coprime-nat ⟨prime p⟩ by blast
    then have p ^ (k + 1) dvd a using ⟨a = r * p ^ k⟩ by simp
    then have k ≥ k + 1
      using assms Max-ge[of {s. p ^ s dvd a} k] Max-in[of {s. p ^ s dvd a}]
      by (metis Max.coboundedI finite-divisor-powers mem-Collect-eq not-prime-unit)
    then show False by simp
  qed
done

context cring
begin

interpretation units-group: group units-of R
  by (rule units-group)

interpretation units-subgroup: multiplicative-subgroup R Units R units-of R
  by (rule units-subgroup)

```

## 2.1 Roots of Unity

**definition** *root-of-unity* :: nat ⇒ 'a ⇒ bool **where**  
*root-of-unity* n μ ≡ μ ∈ carrier R ∧ μ [^] n = 1

**lemma** *root-of-unityI*[intro]: μ ∈ carrier R ⇒ μ [^] n = 1 ⇒ *root-of-unity* n μ  
**unfolding** *root-of-unity-def* **by** *simp*

**lemma** *root-of-unityD*[simp]: *root-of-unity* n μ ⇒ μ [^] n = 1

```

unfolding root-of-unity-def by simp

lemma root-of-unity-closed[simp]: root-of-unity n  $\mu \implies \mu \in \text{carrier } R$ 
unfolding root-of-unity-def by simp

context
  fixes n :: nat
  assumes n > 0
begin

lemma roots-Units[simp]:
  assumes root-of-unity n  $\mu$ 
  shows  $\mu \in \text{Units } R$ 
proof -
  from  $\langle n > 0 \rangle$  obtain n' where n = Suc n'
    using gr0-implies-Suc by auto
  then have 1 =  $\mu \otimes (\mu [\wedge] n')$ 
    using assms nat-pow-Suc2 unfolding root-of-unity-def by auto
  then show  $\mu \in \text{Units } R$  using assms m-comm[of  $\mu \mu [\wedge] n'$ ] nat-pow-closed[of
 $\mu n'$ ]
    unfolding Units-def root-of-unity-def by auto
qed

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

lemma roots-of-unity-group-is-group:
  shows group roots-of-unity-group
  apply (intro groupI)
  unfolding roots-of-unity-group-def root-of-unity-def
  apply (simp-all add: nat-pow-distrib m-assoc)
  subgoal for x
    using  $\langle n > 0 \rangle$ 
    by (metis Group.nat-pow-Suc Nat.lessE mult.commute nat-pow-closed nat-pow-one
  nat-pow-pow)
  done

interpretation root-group : group roots-of-unity-group
  by (rule roots-of-unity-group-is-group)

interpretation root-subgroup : multiplicative-subgroup R  $\{\mu. \text{root-of-unity } n \mu\}$ 
  roots-of-unity-group
  apply unfold-locales
  subgoal using roots-Units  $\langle n > 0 \rangle$  by blast
  subgoal unfolding roots-of-unity-group-def by simp
  done

```

**lemma** *root-of-unity-inv*:  
**assumes** *root-of-unity*  $n \ \mu$   
**shows** *root-of-unity*  $n \ (\text{inv } \mu)$   
**using** *assms* *root-group.inv-closed*[of  $\mu$ ] *root-subgroup.carrier-M* *root-subgroup.inv-eq*[of  $\mu$ ]  
**by** *simp*

**lemma** *inv-root-of-unity*:  
**assumes** *root-of-unity*  $n \ \mu$   
**shows**  $\text{inv } \mu = \mu \text{ } [^{\wedge}] \ (n - 1)$   
**proof** –  
**have**  $\mu \in \text{Units } R$  **using** *assms*  
**using** *roots-Units* **by** *blast*  
**then have**  $\text{inv } \mu = \mu \text{ } [^{\wedge}] \ (-1 :: \text{int})$   
**using** *units-group.int-pow-neg* *units-subgroup.inv-eq* *units-subgroup.int-pow-eq*  
**using** *units-group.int-pow-1* **by** *force*  
**also have**  $\dots = \mathbf{1} \otimes \mu \text{ } [^{\wedge}] \ (-1 :: \text{int})$   
**apply** (*intro l-one[symmetric]*)  
**using**  $\langle \mu \in \text{Units } R \rangle$  **by** (*metis Units-inv-closed calculation*)  
**also have**  $\dots = \mu \text{ } [^{\wedge}] \ n \otimes \mu \text{ } [^{\wedge}] \ (-1 :: \text{int})$   
**using** *assms* **by** *simp*  
**also have**  $\dots = \mu \text{ } [^{\wedge}] \ (\text{int } n) \otimes \mu \text{ } [^{\wedge}] \ (-1 :: \text{int})$   
**using** *Units-closed*[OF  $\langle \mu \in \text{Units } R \rangle$ ]  
**by** (*simp add: int-pow-int*)  
**also have**  $\dots = \mu \text{ } [^{\wedge}] \ (\text{int } n - 1)$   
**using** *units-group.int-pow-mult*[of  $\mu$ ]  $\langle \mu \in \text{Units } R \rangle$  *units-subgroup.int-pow-eq*[of  $\mu$ ]  
**using** *units-of-mult* *units-subgroup.carrier-M*  
**by** (*metis add commute uminus-add-conv-diff*)  
**also have**  $\dots = \mu \text{ } [^{\wedge}] \ (n - 1)$   
**using**  $\langle n > 0 \rangle$  *Units-closed*[OF  $\langle \mu \in \text{Units } R \rangle$ ]  
**by** (*metis Suc-diff-1 add-diff-cancel-left' int-pow-int mult-Suc-right nat-mult-1 of-nat-1 of-nat-add*)  
**finally show** *?thesis* .  
**qed**

**lemma** *inv-pow-root-of-unity*:  
**assumes** *root-of-unity*  $n \ \mu$   
**assumes**  $i \in \{1..<n\}$   
**shows**  $(\text{inv } \mu) \text{ } [^{\wedge}] \ i = \mu \text{ } [^{\wedge}] \ (n - i) \ n - i \in \{1..<n\}$   
**proof** –  
**have**  $(\text{inv } \mu) \text{ } [^{\wedge}] \ i = (\mu \text{ } [^{\wedge}] \ (n - (1::\text{nat}))) \text{ } [^{\wedge}] \ i$   
**using** *assms* *inv-root-of-unity* **by** *algebra*  
**also have**  $\dots = \mu \text{ } [^{\wedge}] \ ((n - 1) * i)$   
**apply** (*intro nat-pow-pow*) **using** *assms* *roots-Units* *Units-closed* **by** *blast*  
**also have**  $\dots = \mu \text{ } [^{\wedge}] \ n \otimes \mu \text{ } [^{\wedge}] \ ((n - 1) * i)$   
**using** *assms* *root-of-unity-def*[of  $n \ \mu$ ] **by** *fastforce*  
**also have**  $\dots = \mu \text{ } [^{\wedge}] \ (n + (n - 1) * i)$   
**apply** (*intro nat-pow-mult*) **using** *assms* *roots-Units* *Units-closed* **by** *blast*  
**also have**  $\dots = \mu \text{ } [^{\wedge}] \ (n * i + (n - i))$

```

proof (intro arg-cong[where  $f = ([\wedge]) \mu$ ])
  have  $\text{int } (n + (n - 1) * i) = \text{int } (n * i + (n - i))$ 
  proof -
    have  $\text{int } (n + (n - 1) * i) = \text{int } n + \text{int } (n - 1) * \text{int } i$ 
      by simp
    also have  $\dots = \text{int } n + (\text{int } n - \text{int } 1) * \text{int } i$ 
      using  $\langle n > 0 \rangle$  by fastforce
    also have  $\dots = \text{int } n + \text{int } n * \text{int } i - \text{int } i$ 
      by (simp add: left-diff-distrib)
    also have  $\dots = \text{int } n * \text{int } i + (\text{int } n - \text{int } i)$ 
      by simp
    also have  $\dots = \text{int } (n * i) + \text{int } (n - i)$ 
      using assms(2) by fastforce
    finally show ?thesis by presburger
  qed
  then show  $n + (n - 1) * i = n * i + (n - i)$  by presburger
qed
also have  $\dots = (\mu [\wedge] n) [\wedge] i \otimes \mu [\wedge] (n - i)$ 
  using nat-pow-mult nat-pow-pow
  using assms roots-Units Units-closed by algebra
also have  $\dots = \mu [\wedge] (n - i)$ 
  using assms unfolding root-of-unity-def by simp
finally show (inv  $\mu$ )  $[\wedge] i = \mu [\wedge] (n - i)$  by blast
show  $n - i \in \{1..<n\}$  using assms by auto
qed

lemma root-of-unity-nat-pow-closed:
  assumes root-of-unity  $n \mu$ 
  shows root-of-unity  $n (\mu [\wedge] (m :: nat))$ 
  using assms root-group.nat-pow-closed root-subgroup.nat-pow-eq by simp

lemma root-of-unity-powers:
  assumes root-of-unity  $n \mu$ 
  shows  $\mu [\wedge] i = \mu [\wedge] (i \bmod n)$ 
proof -
  have[simp]:  $\mu \in \text{carrier } R$  using assms by simp
  define  $s t$  where  $s = i \text{ div } n$   $t = i \bmod n$ 
  then have  $i = s * n + t$   $t < n$  using  $\langle n > 0 \rangle$  by simp-all
  then have  $\mu [\wedge] i = \mu [\wedge] (s * n) \otimes \mu [\wedge] t$  by (simp add: nat-pow-mult)
  also have  $\mu [\wedge] (s * n) = (\mu [\wedge] n) [\wedge] s$  by (simp add: nat-pow-pow mult.commute)
  also have  $\dots = 1$  using assms by simp
  finally show ?thesis using  $\langle t = i \bmod n \rangle$  by simp
qed

lemma root-of-unity-powers-modint:
  assumes root-of-unity  $n \mu$ 
  shows  $\mu [\wedge] (i :: \text{int}) = \mu [\wedge] (i \bmod \text{int } n)$ 
proof -
  have  $\mu \in \text{Units } R$   $\mu [\wedge] n = 1$  using assms by simp-all

```

**define**  $s\ t$  **where**  $s = i \text{ div } \text{int } n\ t = i \text{ mod } \text{int } n$   
**then have**  $i = s * \text{int } n + t\ t \geq 0\ t < \text{int } n$  **using**  $\langle n > 0 \rangle$  **by** *simp-all*  
**then have**  $\mu [\wedge] i = \mu [\wedge] (s * \text{int } n) \otimes \mu [\wedge] t$   
**using** *int-pow-mult*[*OF*  $\langle \mu \in \text{Units } R \rangle$ ] **by** *simp*  
**also have**  $\dots = (\mu [\wedge] \text{int } n) [\wedge] s \otimes \mu [\wedge] t$   
**by** (*intro-cong* [*cong-tag-2* ( $\otimes$ )] *more: refl*) (*simp add: int-pow-pow*  $\langle \mu \in \text{Units } R \rangle$  *mult.commute*)  
**also have**  $\dots = (\mu [\wedge] n) [\wedge] s \otimes \mu [\wedge] t$   
**apply** (*intro-cong* [*cong-tag-2* ( $\otimes$ ), *cong-tag-1* ( $\lambda i. i [\wedge] s$ )] *more: refl*)  
**using**  $\langle n > 0 \rangle$  **by** (*simp add: int-pow-int*)  
**also have**  $\dots = \mu [\wedge] t$   
**using** *int-pow-closed*[*OF*  $\langle \mu \in \text{Units } R \rangle$ ] *Units-closed l-one*  
**by** (*simp add:*  $\langle \mu [\wedge] n = \mathbf{1} \rangle$  *int-pow-one int-pow-closed*)  
**finally show** *?thesis* **unfolding** *s-t-def* .  
**qed**

**lemma** *root-of-unity-powers-nat*:  
**assumes** *root-of-unity*  $n\ \mu$   
**assumes**  $i \text{ mod } n = j \text{ mod } n$   
**shows**  $\mu [\wedge] i = \mu [\wedge] j$   
**using** *assms root-of-unity-powers* **by** *metis*

**lemma** *root-of-unity-powers-int*:  
**assumes** *root-of-unity*  $n\ \mu$   
**assumes**  $i \text{ mod } \text{int } n = j \text{ mod } \text{int } n$   
**shows**  $\mu [\wedge] i = \mu [\wedge] j$   
**using** *assms root-of-unity-powers-modint* **by** *metis*

**end**

## 2.2 Primitive Roots

**definition** *primitive-root*  $:: \text{nat} \Rightarrow 'a \Rightarrow \text{bool}$  **where**  
*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*  $n\ \mu$   
**unfolding** *primitive-root-def root-of-unity-def* **using** *assms* **by** *simp*

**lemma** *primitive-root-is-root-of-unity*[*simp*]: *primitive-root*  $n\ \mu \implies \text{root-of-unity } n\ \mu$   
**unfolding** *primitive-root-def* **by** *simp*

**lemma** *primitive-root-recursion*:  
**assumes** *even*  $n$   
**assumes** *primitive-root*  $n\ \mu$

```

shows primitive-root (n div 2) (μ [⌈] (2 :: nat))
unfolding primitive-root-def root-of-unity-def
apply (intro conjI)
subgoal
  using assms(2) unfolding primitive-root-def root-of-unity-def by blast
subgoal
  using nat-pow-pow[of μ 2::nat n div 2] assms apply simp
  unfolding primitive-root-def root-of-unity-def apply simp
  done
subgoal
proof
  fix i
  assume i ∈ {1..<n div 2}
  then have 2 * i ∈ {1..<n} using ‹even n› by auto
  have (μ [⌈] (2::nat)) [⌈] i = μ [⌈] (2 * i)
    using assms unfolding primitive-root-def root-of-unity-def by (simp add:
nat-pow-pow)
  also have ... ≠ 1
    using assms unfolding primitive-root-def using ‹2 * i ∈ {1..<n}› by blast
  finally show (μ [⌈] (2::nat)) [⌈] i ≠ 1 .
qed
done

```

```

lemma primitive-root-inv:
  assumes n > 0
  assumes primitive-root n μ
  shows primitive-root n (inv μ)
  unfolding primitive-root-def
proof (intro conjI)
  show root-of-unity n (inv μ) using assms unfolding primitive-root-def
    by (simp add: root-of-unity-inv)
  show ∀ i ∈ {1..<n}. inv μ [⌈] i ≠ 1 using assms unfolding primitive-root-def
    by (metis Group.nat-pow-0 Units-inv-inv bot-nat-0.extremum-strict nat-neq-iff
root-of-unity-def root-of-unity-inv roots-Units)
qed

```

## 2.3 Number Theoretic Transforms

```

definition NTT :: 'a ⇒ 'a list ⇒ 'a list where
NTT μ a ≡ let n = length a in [⊕ j ← [0..<n]. (a ! j) ⊗ (μ [⌈] i) [⌈] j. i ←
[0..<n]]

```

```

lemma NTT-length[simp]: length (NTT μ a) = length a
  unfolding NTT-def by (metis length-map map-nth)

```

```

lemma NTT-nth:
  assumes length a = n
  assumes i < n
  shows NTT μ a ! i = (⊕ j ← [0..<n]. (a ! j) ⊗ (μ [⌈] i) [⌈] j)

```

**unfolding** *NTT-def* **using** *assms* **by** *auto*

**lemma** *NTT-nth-2*:

**assumes** *length a = n*

**assumes** *i < n*

**assumes**  $\mu \in \text{carrier } R$

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

**unfolding** *NTT-nth*[*OF assms(1) assms(2)*]

**by** (*intro monoid-sum-list-cong arg-cong*[**where**  $f = (\otimes) -$ ] *nat-pow-pow assms(3)*)

**lemma** *NTT-nth-closed*:

**assumes**  $set a \subseteq \text{carrier } R$

**assumes**  $\mu \in \text{carrier } R$

**assumes** *length a = n*

**assumes** *i < n*

**shows**  $NTT \mu a ! i \in \text{carrier } R$

**proof** –

**have**  $NTT \mu a ! i = (\bigoplus j \leftarrow [0..<\text{length } a]. (a ! j) \otimes (\mu [\uparrow] i) [\uparrow] j)$

**using** *NTT-nth assms* **by** *blast*

**also have**  $\dots \in \text{carrier } R$

**by** (*intro monoid-sum-list-closed m-closed nat-pow-closed assms(2) set-subseteqD*[*OF assms(1)*]) *simp*

**finally show** *?thesis* .

**qed**

**lemma** *NTT-closed*:

**assumes**  $set a \subseteq \text{carrier } R$

**assumes**  $\mu \in \text{carrier } R$

**shows**  $set (NTT \mu a) \subseteq \text{carrier } R$

**using** *assms NTT-nth-closed*[*of a*  $\mu$ ]

**by** (*intro subsetI*)(*metis NTT-length in-set-conv-nth*)

**lemma** *primitive-root 1 1*

**unfolding** *primitive-root-def root-of-unity-def*

**by** *simp*

**lemma**  $(\ominus 1) [\uparrow] (2::nat) = 1$

**by** (*simp add: numeral-2-eq-2*) *algebra*

**lemma**  $1 \oplus 1 \neq 0 \implies \text{primitive-root } 2 (\ominus 1)$

**unfolding** *primitive-root-def root-of-unity-def*

**apply** (*intro conjI*)

**subgoal by** *simp*

**subgoal by** (*simp add: numeral-2-eq-2, algebra*)

**subgoal**

**proof** (*standard, rule ccontr*)

**fix** *i*

**assume**  $1 \oplus 1 \neq 0$   $i \in \{1::nat..<2\}$

**then have**  $i = 1$  **by** *simp*

**assume**  $\neg \ominus 1 [\uparrow] i \neq 1$

```

then have  $\ominus 1 = 1$  using  $\langle i = 1 \rangle$  by simp
then have  $1 \oplus 1 = 0$  using l-neg by fastforce
thus False using  $\langle 1 \oplus 1 \neq 0 \rangle$  by simp
qed
done

```

### 2.3.1 Inversion Rule

**theorem** *inversion-rule*:

```

fixes  $\mu :: 'a$ 
fixes  $n :: nat$ 
assumes  $n > 0$ 
assumes primitive-root  $n \ \mu$ 
assumes good:  $\bigwedge i. i \in \{1..<n\} \implies (\bigoplus j \leftarrow [0..<n]. (\mu [\ ] i) [\ ] j) = 0$ 
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 (intro nth-equalityI)
  have  $\mu \in Units \ R$  using assms unfolding primitive-root-def using roots-Units
  by blast
  then have[simp]:  $\mu \in carrier \ R$  by blast
  show  $length \ (NTT \ (inv \ \mu) \ (NTT \ \mu \ a)) = length \ (map \ ((\otimes) \ (nat-embedding \ n)) \ a)$ 
  using NTT-length
  by simp
  fix  $i$ 
  assume  $i < length \ (NTT \ (inv \ \mu) \ (NTT \ \mu \ a))$ 
  then have  $i < n$  by simp

  have[simp]:  $inv \ \mu \in carrier \ R$  using assms roots-Units unfolding primitive-root-def
  by blast
  then have[simp]:  $\bigwedge i :: nat. (inv \ \mu) [\ ] i \in carrier \ R$  by simp

  have  $0: NTT \ (inv \ \mu) \ (NTT \ \mu \ a) ! i = (\bigoplus j \leftarrow [0..<n]. (NTT \ \mu \ a ! j) \otimes ((inv \ \mu) [\ ] i) [\ ] j)$ 
  using NTT-nth
  using assms NTT-length  $\langle i < n \rangle$  by blast
  also have  $\dots = (\bigoplus j \leftarrow [0..<n]. (\bigoplus k \leftarrow [0..<n]. a ! k \otimes \mu [\ ] ((int \ k - int \ i) * int \ j)))$ 
  proof (intro monoid-sum-list-cong)
    fix  $j$ 
    assume  $j \in set \ [0..<n]$ 
    then have[simp]:  $j < n$  by simp
    have  $nj: (NTT \ \mu \ a ! j) = (\bigoplus k \leftarrow [0..<n]. a ! k \otimes (\mu [\ ] j) [\ ] k)$ 
    using NTT-nth by simp
    have  $\dots \otimes ((inv \ \mu) [\ ] i) [\ ] j = (\bigoplus k \leftarrow [0..<n]. a ! k \otimes ((\mu [\ ] j) [\ ] k) \otimes ((inv \ \mu) [\ ] i) [\ ] j)$ 
    apply (intro monoid-sum-list-in-right[symmetric] nat-pow-closed m-closed)
    using set-subseteqD[OF assms(5)] by simp-all
    also have  $\dots = (\bigoplus k \leftarrow [0..<n]. a ! k \otimes \mu [\ ] ((int \ k - int \ i) * int \ j))$ 

```

```

proof (intro monoid-sum-list-cong)
  fix k
  assume k ∈ set [0..<n]
  have a ! k ⊗ (μ [↑] j) [↑] k ⊗ (inv μ [↑] i) [↑] j = a ! k ⊗ ((μ [↑] j) [↑] k ⊗
(inv μ [↑] i) [↑] j)
    apply (intro m-assoc nat-pow-closed)
    using set-subseteqD[OF assms(5)] ⟨k ∈ set [0..<n]⟩ by simp-all
  also have inv μ [↑] i = μ [↑] (- int i)
    by (metis ⟨μ ∈ Units R⟩ cring.units-int-pow-neg int-pow-int is-cring)
  also have ((μ [↑] j) [↑] k ⊗ (μ [↑] (- int i)) [↑] j) = μ [↑] (int j * int k -
int i * int j)
    using ⟨μ ∈ Units R⟩
    by (simp add: int-pow-int[symmetric] int-pow-pow int-pow-mult)
  also have ... = μ [↑] ((int k - int i) * int j)
    apply (intro arg-cong[where f = ([↑] -)])
    by (simp add: mult.commute right-diff-distrib^)
  finally show a ! k ⊗ (μ [↑] j) [↑] k ⊗ (inv μ [↑] i) [↑] j = a ! k ⊗ μ [↑] ((int
k - int i) * int j)
    using ⟨inv μ [↑] i = μ [↑] (- int i)⟩ by argo
  qed
  finally show NTT μ a ! j ⊗ (inv μ [↑] i) [↑] j = monoid-sum-list (λk. a ! k ⊗
μ [↑] ((int k - int i) * int j)) [0..<n]
    by (simp add: nj)
  qed
  also have ... = (⊕ k ← [0..<n]. (⊕ j ← [0..<n]. a ! k ⊗ μ [↑] ((int k - int i)
* int j)))
    apply (intro monoid-sum-list-swap m-closed)
    subgoal for j k
      using assms by (metis atLeastLessThan-iff atLeastLessThan-upt nth-mem
subset-eq)
    subgoal for j k
      using ⟨μ ∈ Units R⟩
      using units-of-int-pow[OF ⟨μ ∈ Units R⟩]
      using group.int-pow-closed[OF units-group, of μ]
      by (metis Units-closed units-of-carrier)
    done
  also have ... = (⊕ k ← [0..<n]. a ! k ⊗ (⊕ j ← [0..<n]. μ [↑] ((int k - int i)
* int j)))
    apply (intro monoid-sum-list-cong monoid-sum-list-in-left)
    subgoal using set-subseteqD[OF assms(5)] by simp
    subgoal for j
      by (simp add: Units-closed int-pow-closed ⟨μ ∈ Units R⟩)
    done
  also have ... = (⊕ k ← [0..<n]. a ! k ⊗ (if i = k then nat-embedding n else 0))
proof (intro monoid-sum-list-cong arg-cong[where f = (⊗) -])
  fix k
  assume k ∈ set [0..<n]
  then have[simp]: k < n by simp
  consider i < k | i = k | i > k by fastforce

```

**then show**  $(\bigoplus j \leftarrow [0..<n]. \mu [\uparrow] ((int\ k - int\ i) * int\ j)) = (if\ i = k\ then\ nat\text{-}embedding\ n\ else\ \mathbf{0})$   
**proof** (*cases*)  
**case 1**  
**then have**  $\bigwedge j. j < n \implies \mu [\uparrow] ((int\ k - int\ i) * int\ j) = (\mu [\uparrow] (k - i)) [\uparrow] j$   
**proof** –  
**fix**  $j$   
**assume**  $j < n$   
**have**  $(int\ k - int\ i) * int\ j = int\ ((k - i) * j)$  **using 1 by auto**  
**then have**  $\mu [\uparrow] ((int\ k - int\ i) * int\ j) = \mu [\uparrow] int\ ((k - i) * j)$   
**by** *argo*  
**also have**  $\dots = \mu [\uparrow] ((k - i) * j)$   
**by** (*intro int-pow-int*)  
**also have**  $\dots = (\mu [\uparrow] (k - i)) [\uparrow] j$   
**by** (*intro nat-pow-pow[symmetric] <\mu \in carrier R>*)  
**finally show**  $\mu [\uparrow] ((int\ k - int\ i) * int\ j) = (\mu [\uparrow] (k - i)) [\uparrow] j$ .  
**qed**  
**then have**  $(\bigoplus j \leftarrow [0..<n]. \mu [\uparrow] ((int\ k - int\ i) * int\ j)) = (\bigoplus j \leftarrow [0..<n]. (\mu [\uparrow] (k - i)) [\uparrow] j)$   
**by** (*intro monoid-sum-list-cong, simp*)  
**also have**  $\dots = \mathbf{0}$   
**using** *good[of k - i]*  
**proof**  
**show**  $k - i \in \{1..<n\}$  **using 1 <k < n> by (simp add: less-imp-diff-less)**  
**qed simp**  
**finally show** *?thesis using 1 by simp*  
**next**  
**case 2**  
**then have**  $\bigwedge j. j < n \implies \mu [\uparrow] ((int\ k - int\ i) * int\ j) = \mathbf{1}$  **by simp**  
**then have**  $(\bigoplus j \leftarrow [0..<n]. \mu [\uparrow] ((int\ k - int\ i) * int\ j)) = nat\text{-}embedding\ n$   
**using** *monoid-sum-list-const[of 1 [0..<n]]*  
**using** *monoid-sum-list-cong[of [0..<n] \lambda j. \mu [\uparrow] ((int\ k - int\ i) \* int\ j) \lambda j.*  
**1]**  
**by simp**  
**then show** *?thesis using 2 by simp*  
**next**  
**case 3**  
**then have**  $\bigwedge j. j < n \implies \mu [\uparrow] ((int\ k - int\ i) * int\ j) = (\mu [\uparrow] (n + k - i)) [\uparrow] j$   
**proof** –  
**fix**  $j$   
**assume**  $j < n$   
**have**  $\mu [\uparrow] ((int\ k - int\ i) * int\ j) = (\mu [\uparrow] (int\ k - int\ i)) [\uparrow] j$   
**using** *int-pow-pow by (metis <\mu \in Units R> int-pow-int)*  
**also have**  $\dots = (\mu [\uparrow] n \otimes \mu [\uparrow] (int\ k - int\ i)) [\uparrow] j$   
**proof** –  
**have**  $\mu [\uparrow] (int\ k - int\ i) \in carrier\ R$   
**using** *<\mu \in Units R> int-pow-closed Units-closed by simp*  
**then have**  $\mu [\uparrow] (int\ k - int\ i) = \mu [\uparrow] n \otimes \mu [\uparrow] (int\ k - int\ i)$

```

    using l-one assms(2) unfolding primitive-root-def root-of-unity-def
    by presburger
  then show ?thesis by simp
qed
also have ... = (μ [⌈] (int n) ⊗ μ [⌈] (int k - int i)) [⌈] j
  by (simp add: int-pow-int)
also have ... = (μ [⌈] (int n + int k - int i)) [⌈] j
  using ⟨μ ∈ Units R⟩ by (simp add: int-pow-mult add-diff-eq)
finally show μ [⌈] ((int k - int i) * int j) = (μ [⌈] (n + k - i)) [⌈] j using
3
  by (metis (no-types, opaque-lifting) ⟨i < n⟩ diff-cancel2 diff-diff-cancel
diff-le-self int-plus int-pow-int less-or-eq-imp-le of-nat-diff)
qed
then have (⊕ j ← [0..<n]. μ [⌈] ((int k - int i) * int j)) = (⊕ j ← [0..<n].
(μ [⌈] (n + k - i)) [⌈] j)
  by (intro monoid-sum-list-cong, simp)
also have ... = 0
  using good[of n + k - i]
proof
  show n + k - i ∈ {1..<n} using 3 ⟨k < n⟩ ⟨i < n⟩ by fastforce
qed simp
finally show ?thesis using 3 by simp
qed
qed
also have ... = (⊕ k ← [0..<n]. a ! k ⊗ (nat-embedding n ⊗ delta k i))
  apply (intro monoid-sum-list-cong)
  unfolding delta-def
  by simp
also have ... = (⊕ k ← [0..<n]. nat-embedding n ⊗ (delta k i ⊗ a ! k))
  apply (intro monoid-sum-list-cong)
  using m-assoc m-comm delta-closed set-subseteqD[OF assms(5)] nat-embedding-closed
by simp
also have ... = nat-embedding n ⊗ (⊕ k ← [0..<n]. delta k i ⊗ a ! k)
  using set-subseteqD[OF assms(5)]
  by (intro monoid-sum-list-in-left) auto
also have ... = nat-embedding n ⊗ a ! i
  using monoid-sum-list-delta[of n λk. a ! k i] ⟨i < n⟩ assms
  by (metis (no-types, lifting) nth-mem subsetD)
finally show NTT (inv μ) (NTT μ a) ! i = map ((⊗) (nat-embedding n)) a ! i
  using nth-map ⟨i < n⟩ ⟨length a = n⟩ NTT-length 0
  by simp
qed

```

**lemma** *inv-good*:

**assumes**  $n > 0$

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

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

**shows** *primitive-root*  $n \ (\text{inv } \mu)$

$\bigwedge i. i \in \{1..<n\} \implies (\bigoplus j \leftarrow [0..<n]. ((\text{inv } \mu) [⌈] i) [⌈] j) = 0$

**subgoal using** *assms* **by** (*simp add: primitive-root-inv*)  
**subgoal for**  $i$   
**proof** –  
 assume  $i \in \{1..<n\}$   
 then have  $n - i \in \{1..<n\}$  **by** *auto*  
 then have  $(\bigoplus j \leftarrow [0..<n]. (\mu [\uparrow] (n - i)) [\uparrow] j) = \mathbf{0}$   
 using *assms* **by** *blast*  
 moreover have  $\mu [\uparrow] (n - i) = \text{inv } \mu [\uparrow] i$   
 using *assms inv-pow-root-of-unity*[of  $n \mu i$ ]  $\langle i \in \{1..<n\} \rangle$   
 by *auto*  
 ultimately show  $(\bigoplus j \leftarrow [0..<n]. ((\text{inv } \mu) [\uparrow] i) [\uparrow] j) = \mathbf{0}$  **by** *simp*  
**qed**  
**done**

**lemma** *inv-halfway-property*:

assumes  $\mu \in \text{Units } R$   
 assumes  $\mu [\uparrow] (i::\text{nat}) = \ominus \mathbf{1}$   
 shows  $(\text{inv } \mu) [\uparrow] i = \ominus \mathbf{1}$   
**proof** –  
 have  $(\text{inv } \mu) [\uparrow] i = (\text{inv}_{\text{units-of } R} \mu) [\uparrow] i$   
 by (*intro arg-cong*[**where**  $f = \lambda j. j [\uparrow] i$ ] *units-of-inv*[*symmetric*] *assms*(1))  
 also have  $\dots = (\text{inv}_{\text{units-of } R} \mu) [\uparrow]_{\text{units-of } R} i$   
 apply (*intro units-of-pow*[*symmetric*])  
 using *units-group.Units-inv-Units* *assms*(1) **by** *simp*  
 also have  $\dots = \text{inv}_{\text{units-of } R} (\mu [\uparrow]_{\text{units-of } R} i)$   
 apply (*intro units-group.nat-pow-inv*)  
 using *assms*(1) **by** (*simp add: units-of-def*)  
 also have  $\dots = \text{inv } (\mu [\uparrow]_{\text{units-of } R} i)$   
 apply (*intro units-of-inv*)  
 using *assms*(1) *units-group.nat-pow-closed* **by** (*simp add: units-of-def*)  
 also have  $\dots = \text{inv } (\mu [\uparrow] i)$   
 using *units-of-pow* *assms*(1) **by** *simp*  
 finally have  $(\text{inv } \mu) [\uparrow] i = \text{inv } (\mu [\uparrow] i)$  .  
 also have  $\dots = \text{inv } (\ominus \mathbf{1})$  **using** *assms*(2) **by** *simp*  
 also have  $\dots = \ominus \mathbf{1}$  **by** *simp*  
 finally show *?thesis* .  
**qed**

**lemma** *sufficiently-good-aux*:

assumes *primitive-root*  $m \eta$   
 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  $(\bigoplus l \leftarrow [0..<m]. (\eta [\uparrow] (r * 2^{\wedge} k)) [\uparrow] l) = \mathbf{0}$   
 using *assms*  
**proof** (*induction*  $k$  *arbitrary: \eta m j*)  
 case 0  
 then have *root-of-unity*  $m \eta$  **by** *simp*

```

then have  $\eta \in \text{carrier } R$  by simp
have  $j > 0$ 
proof (rule ccontr)
  assume  $\neg j > 0$ 
  then have  $j = 0$  by simp
  then have  $m = 1$  using 0 by simp
  then have  $r * 2^k = 0$  using 0 by simp
  then have  $r = 0$  by simp
  then show False using ‹odd r› by simp
qed
then have even m using 0 by simp
then have  $m = m \text{ div } 2 + m \text{ div } 2$  by auto
then have  $(\bigoplus l \leftarrow [0..<m]. (\eta [\uparrow] (r * 2^k))) [\uparrow] l = (\bigoplus l \leftarrow [0..<m \text{ div } 2 + m \text{ div } 2]. (\eta [\uparrow] r) [\uparrow] l)$ 
  by simp
  also have ... =  $(\bigoplus l \leftarrow [0..<m \text{ div } 2]. (\eta [\uparrow] r) [\uparrow] l) \oplus (\bigoplus l \leftarrow [m \text{ div } 2..<m \text{ div } 2 + m \text{ div } 2]. (\eta [\uparrow] r) [\uparrow] l)$ 
    by (intro monoid-sum-list-split[symmetric] nat-pow-closed, rule ‹ $\eta \in \text{carrier } R$ ›)
  also have ... =  $(\bigoplus l \leftarrow [0..<m \text{ div } 2]. (\eta [\uparrow] r) [\uparrow] l) \oplus (\bigoplus l \leftarrow [0..<m \text{ div } 2]. (\eta [\uparrow] r) [\uparrow] (m \text{ div } 2 + l))$ 
    by (intro arg-cong[where f = ( $\oplus$ ) -] monoid-sum-list-index-shift-0)
  also have ... =  $(\bigoplus l \leftarrow [0..<m \text{ div } 2]. (\eta [\uparrow] r) [\uparrow] l \oplus (\eta [\uparrow] r) [\uparrow] (m \text{ div } 2 + l))$ 
    by (intro monoid-sum-list-add-in nat-pow-closed; rule ‹ $\eta \in \text{carrier } R$ ›)
  also have ... =  $(\bigoplus l \leftarrow [0..<m \text{ div } 2]. (\eta [\uparrow] r) [\uparrow] l \ominus (\eta [\uparrow] r) [\uparrow] l)$ 
    proof (intro monoid-sum-list-cong)
      fix l
      have  $(\eta [\uparrow] r) [\uparrow] (m \text{ div } 2 + l) = (\eta [\uparrow] r) [\uparrow] (m \text{ div } 2) \otimes (\eta [\uparrow] r) [\uparrow] l$ 
        by (intro nat-pow-mult[symmetric] nat-pow-closed, rule ‹ $\eta \in \text{carrier } R$ ›)
      also have  $(\eta [\uparrow] r) [\uparrow] (m \text{ div } 2) = (\ominus \mathbf{1}) [\uparrow] r$ 
        unfolding nat-pow-pow[OF ‹ $\eta \in \text{carrier } R$ ›] mult.commute[of r -]
        by (simp only: nat-pow-pow[symmetric] ‹ $\eta \in \text{carrier } R$ › ‹ $\eta [\uparrow] (m \text{ div } 2) = \ominus \mathbf{1}$ ›)
      also have ... =  $\ominus \mathbf{1}$  using ‹odd r›
        by (simp add: powers-of-negative)
      finally have  $(\eta [\uparrow] r) [\uparrow] (m \text{ div } 2 + l) = \ominus ((\eta [\uparrow] r) [\uparrow] l)$ 
        using ‹ $\eta \in \text{carrier } R$ › nat-pow-closed by algebra
      then show  $(\eta [\uparrow] r) [\uparrow] l \oplus (\eta [\uparrow] r) [\uparrow] (m \text{ div } 2 + l) = (\eta [\uparrow] r) [\uparrow] l \ominus (\eta [\uparrow] r) [\uparrow] l$ 
        unfolding minus-eq
        by (intro arg-cong[where f = ( $\oplus$ ) -])
    qed
  also have ... =  $(\bigoplus l \leftarrow [0..<m \text{ div } 2]. \mathbf{0})$ 
    by (intro monoid-sum-list-cong) (simp add: ‹ $\eta \in \text{carrier } R$ ›)
  also have ... =  $\mathbf{0}$  by simp
  finally show ?case .
next
case (Suc k)
have  $j > 0$ 

```

**proof** (*rule ccontr*)  
**assume**  $\neg j > 0$   
**then have**  $j = 0$  **by** *simp*  
**then have**  $m = 1$  **using** *Suc* **by** *simp*  
**then have**  $r * 2^k = 0$  **using** *Suc* **by** *simp*  
**then have**  $r = 0$  **by** *simp*  
**then show** *False* **using**  $\langle \text{odd } r \rangle$  **by** *simp*  
**qed**  
**then have** *even m* **using** *Suc* **by** *simp*  
**then have**  $m = m \text{ div } 2 + m \text{ div } 2$  **by** *auto*  
**have** *root-of-unity m η* **using**  $\langle \text{primitive-root } m \eta \rangle$  **by** *simp*  
**then have**  $\eta \in \text{carrier } R$  **by** *simp*  
**from**  $\langle j > 0 \rangle$  **obtain**  $j'$  **where**  $j = \text{Suc } j'$   
**using** *gr0-implies-Suc* **by** *blast*  
**then have**  $m \text{ div } 2 = 2^{j'}$  **using**  $\langle m = 2^j \rangle$  **by** *simp*  
**have**  $j' > 0$   
**proof** (*rule ccontr*)  
**assume**  $\neg j' > 0$   
**then have**  $j' = 0$  **by** *simp*  
**then have**  $m = 2$  **using**  $\langle m = 2^j \rangle$   $\langle j = \text{Suc } j' \rangle$  **by** *simp*  
**then have**  $r * 2^{\text{Suc } k} < 2$  **using** *Suc* **by** *simp*  
**then show** *False* **using**  $\langle \text{odd } r \rangle$  **by** *simp*  
**qed**  
**then have** *even (m div 2)* **using**  $\langle m \text{ div } 2 = 2^{j'} \rangle$  **by** *simp*  
**have** *IH'*:  $(\bigoplus l \leftarrow [0..<m \text{ div } 2]). ((\eta [\ ] (2::\text{nat})) [\ ] (r * 2^k)) [\ ] l) = \mathbf{0}$   
**apply** (*intro Suc.IH[of m div 2 η [ ] (2::nat) j']*)  
**subgoal using** *primitive-root-recursion[OF ⟨even m⟩, OF ⟨primitive-root m η⟩]*  
**subgoal using**  $\langle m = 2^j \rangle$   $\langle j = \text{Suc } j' \rangle$  **by** *simp*  
**subgoal**  
**by** (*simp add: ⟨η ∈ carrier R⟩ nat-pow-pow ⟨even (m div 2)⟩ ⟨η [ ] (m div 2) = ⊕ 1⟩*)  
**subgoal using**  $\langle \text{odd } r \rangle$  .  
**subgoal using**  $\langle r * 2^{(\text{Suc } k)} < m \rangle$   $\langle \text{even } m \rangle$  **by** *auto*  
**done**  
**have**  $(\bigoplus l \leftarrow [0..<m]). (\eta [\ ] (r * 2^{(\text{Suc } k)})) [\ ] l) = (\bigoplus l \leftarrow [0..<m]). ((\eta [\ ] (2::\text{nat})) [\ ] (r * 2^k)) [\ ] l)$   
**unfolding** *nat-pow-pow[OF ⟨η ∈ carrier R⟩]*  
**apply** (*intro monoid-sum-list-cong arg-cong[where f = λi. i [ ] -]*)  
**apply** (*intro arg-cong[where f = ([ ] -]*)  
**by** *simp*  
**also have**  $\dots = (\bigoplus l \leftarrow [0..<m \text{ div } 2 + m \text{ div } 2]). ((\eta [\ ] (2::\text{nat})) [\ ] (r * 2^k)) [\ ] l)$   
**using**  $\langle m = m \text{ div } 2 + m \text{ div } 2 \rangle$  **by** *argo*  
**also have**  $\dots = (\bigoplus l \leftarrow [0..<m \text{ div } 2]). ((\eta [\ ] (2::\text{nat})) [\ ] (r * 2^k)) [\ ] l) \oplus (\bigoplus l \leftarrow [m \text{ div } 2..<m \text{ div } 2 + m \text{ div } 2]). ((\eta [\ ] (2::\text{nat})) [\ ] (r * 2^k)) [\ ] l)$   
**by** (*intro monoid-sum-list-split[symmetric] nat-pow-closed, rule ⟨η ∈ carrier R⟩*)  
**also have**  $\dots = \mathbf{0} \oplus (\bigoplus l \leftarrow [m \text{ div } 2..<m \text{ div } 2 + m \text{ div } 2]). ((\eta [\ ] (2::\text{nat})) [\ ] (r * 2^k)) [\ ] l)$

**using**  $IH'$  **by** *argo*  
**also have**  $\dots = (\bigoplus l \leftarrow [m \text{ div } 2..<m \text{ div } 2 + m \text{ div } 2]. ((\eta [\uparrow] (2::\text{nat})) [\uparrow] (r * 2 \wedge k)) [\uparrow] l)$   
**by** (*intro l-zero monoid-sum-list-closed nat-pow-closed, rule  $\langle \eta \in \text{carrier } R \rangle$* )  
**also have**  $\dots = (\bigoplus l \leftarrow [0..<m \text{ div } 2]. ((\eta [\uparrow] (2::\text{nat})) [\uparrow] (r * 2 \wedge k)) [\uparrow] (m \text{ div } 2 + l))$   
**by** (*intro monoid-sum-list-index-shift-0*)  
**also have**  $\dots = (\bigoplus l \leftarrow [0..<m \text{ div } 2]. ((\eta [\uparrow] (2::\text{nat})) [\uparrow] (r * 2 \wedge k)) [\uparrow] (m \text{ div } 2) \otimes ((\eta [\uparrow] (2::\text{nat})) [\uparrow] (r * 2 \wedge k)) [\uparrow] l)$   
**by** (*intro monoid-sum-list-cong nat-pow-mult[symmetric] nat-pow-closed, rule  $\langle \eta \in \text{carrier } R \rangle$* )  
**also have**  $\dots = ((\eta [\uparrow] (2::\text{nat})) [\uparrow] (r * 2 \wedge k)) [\uparrow] (m \text{ div } 2) \otimes (\bigoplus l \leftarrow [0..<m \text{ div } 2]. ((\eta [\uparrow] (2::\text{nat})) [\uparrow] (r * 2 \wedge k)) [\uparrow] l)$   
**by** (*intro monoid-sum-list-in-left nat-pow-closed; rule  $\langle \eta \in \text{carrier } R \rangle$* )  
**also have**  $\dots = ((\eta [\uparrow] (2::\text{nat})) [\uparrow] (r * 2 \wedge k)) [\uparrow] (m \text{ div } 2) \otimes \mathbf{0}$   
**using**  $IH'$  **by** *argo*  
**also have**  $\dots = \mathbf{0}$   
**by** (*intro r-null nat-pow-closed, rule  $\langle \eta \in \text{carrier } R \rangle$* )  
**finally show** *?case* .  
**qed**

**lemma** *sufficiently-good:*

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

**case** *True*  
**fix**  $i$   
**assume**  $i \in \{1..<n\}$

**have** *root-of-unity*  $n \mu$  **using** *assms(1)* **by** *simp*  
**then have**  $\mu \in \text{carrier } R \mu [\uparrow] n = \mathbf{1}$  **by** *simp-all*

**have**  $\mu [\uparrow] i \neq \mathbf{1}$  **using** *assms(1)*  $\langle i \in \{1..<n\} \rangle$  **unfolding** *primitive-root-def*  
**by** *simp*  
**then have**  $\mathbf{1} \ominus \mu [\uparrow] i \neq \mathbf{0}$  **using**  $\langle \mu \in \text{carrier } R \rangle$  **by** *simp*

**have**  $(\mu [\uparrow] i) [\uparrow] n = \mathbf{1}$   
**unfolding** *nat-pow-pow[OF  $\langle \mu \in \text{carrier } R \rangle$*   
**using** *root-of-unity-powers[OF -  $\langle \text{root-of-unity } n \mu \rangle$ , of  $i * n$ ]*  
**by** (*cases  $n > 0$ ; simp*)  
**then have**  $\mathbf{0} = \mathbf{1} \ominus (\mu [\uparrow] i) [\uparrow] n$  **by** *algebra*  
**also have**  $\dots = (\mathbf{1} \ominus \mu [\uparrow] i) \otimes (\bigoplus j \leftarrow [0..<n]. (\mu [\uparrow] i) [\uparrow] j)$   
**by** (*intro geo-monoid-list-sum[symmetric] nat-pow-closed  $\langle \mu \in \text{carrier } R \rangle$* )  
**finally show**  $(\bigoplus j \leftarrow [0..<n]. (\mu [\uparrow] i) [\uparrow] j) = \mathbf{0}$   
**using**  $\langle \mathbf{1} \ominus \mu [\uparrow] i \neq \mathbf{0} \rangle$  *True  $\langle \mu \in \text{carrier } R \rangle$*   
**by** (*metis domain.integral minus-closed monoid-sum-list-closed nat-pow-closed one-closed*)

```

next
case False
then have  $n = 2^k \mu \lceil (n \text{ div } 2) = \ominus \mathbf{1}$  using assms(2) by auto

have root-of-unity  $n \mu$  using  $\langle \text{primitive-root } n \mu \rangle$  by simp
then have  $\mu \in \text{carrier } R \mu \lceil n = \mathbf{1}$  by simp-all

fix  $i$ 
assume  $i \in \{1..<n\}$ 
define  $l$  where  $l = \text{Max } \{s. 2^s \text{ dvd } i\}$ 
define  $r$  where  $r = i \text{ div } 2^l$ 
from  $\langle i \in \{1..<n\} \rangle$  have  $i \neq 0$  by simp
then have  $i = r * 2^l \text{ odd } r$  using max-dividing-power-factorization[of i l 2 r]
using l-def r-def coprime-left-2-iff-odd[of r] by simp-all

show  $(\bigoplus_{j \leftarrow [0..<n]}. (\mu \lceil i) \lceil j) = \mathbf{0}$ 
  apply (simp only:  $\langle i = r * 2^l \rangle$ )
  apply (intro sufficiently-good-aux[of n  $\mu$  k r l, OF  $\langle \text{primitive-root } n \mu \rangle \langle n = 2^k \rangle \langle \mu \lceil (n \text{ div } 2) = \ominus \mathbf{1} \rangle \langle \text{odd } r \rangle$ )
  using  $\langle i = r * 2^l \rangle \langle i \in \{1..<n\} \rangle$  by simp
qed

```

**corollary** *inversion-rule-inv*:

```

fixes  $\mu :: 'a$ 
fixes  $n :: \text{nat}$ 
assumes  $n > 0$ 
assumes primitive-root  $n \mu$ 
assumes good:  $\bigwedge i. i \in \{1..<n\} \implies (\bigoplus_{j \leftarrow [0..<n]}. (\mu \lceil i) \lceil j) = \mathbf{0}$ 
assumes[simp]: length  $a = n$ 
assumes[simp]: set  $a \subseteq \text{carrier } R$ 
shows NTT  $\mu$  (NTT (inv  $\mu$ )  $a$ ) = map  $(\lambda x. \text{nat-embedding } n \otimes x) a$ 
using assms inv-good[of n  $\mu$  inversion-rule[of n inv  $\mu$  a]
using Units-inv-inv[of  $\mu$ ]
using roots-Units[of n  $\mu$ ]
unfolding primitive-root-def
by algebra

```

### 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  $(\bigoplus_{i \leftarrow [0..<n]}. f i \otimes x \lceil i) \otimes (\bigoplus_{i \leftarrow [0..<n]}. g i \otimes x \lceil i) =$ 
   $(\bigoplus_{k \leftarrow [0..<n]}. (\bigoplus_{i \leftarrow [0..<n]}. f i \otimes g ((n + k - i) \text{ mod } n)) \otimes x \lceil k)$ 
proof (cases  $n > 0$ )
case False
then have  $n = 0$  by simp
then show ?thesis by simp

```

```

next
case True
have[simp]: x ∈ carrier R using ⟨root-of-unity n x⟩ by simp

have (⊕ k ← [0..<n]. (⊕ i ← [0..<n]. f i ⊗ g ((n + k - i) mod n)) ⊗ x [∧] k)
=
  (⊕ k ← [0..<n]. (⊕ i ← [0..<n]. f i ⊗ g ((n + k - i) mod n) ⊗ x [∧] k))
  by (intro monoid-sum-list-cong monoid-sum-list-in-right[symmetric] nat-pow-closed
m-closed)
    simp-all
  also have ... = (⊕ k ← [0..<n]. (⊕ i ← [0..<n]. f i ⊗ g ((n + k - i) mod n)
⊗ x [∧] ((n + k - i) mod n + i)))
    apply (intro monoid-sum-list-cong arg-cong[where f = (⊗) -])
    apply (intro root-of-unity-powers-nat[OF ⟨n > 0⟩ ⟨root-of-unity n x⟩])
    by (simp add: add commute mod-add-right-eq)
  also have ... = (⊕ k ← [0..<n]. (⊕ i ← [0..<n]. f i ⊗ g ((n + k - i) mod n)
⊗ (x [∧] ((n + k - i) mod n) ⊗ x [∧] i)))
    by (intro monoid-sum-list-cong arg-cong[where f = (⊗) -] nat-pow-mult[symmetric])
    simp
  also have ... = (⊕ k ← [0..<n]. (⊕ i ← [0..<n]. f i ⊗ x [∧] i ⊗ (g ((n + k -
i) mod n) ⊗ x [∧] ((n + k - i) mod n))))
    proof -
      have reorder: ∧ a b c d. [ a ∈ carrier R; b ∈ carrier R; c ∈ carrier R; d ∈
carrier R ] ⇒
        a ⊗ b ⊗ (c ⊗ d) = a ⊗ d ⊗ (b ⊗ c)
        using m-comm m-assoc by algebra
      show ?thesis
        by (intro monoid-sum-list-cong reorder nat-pow-closed) simp-all
    qed
  also have ... = (⊕ i ← [0..<n]. (⊕ k ← [0..<n]. f i ⊗ x [∧] i ⊗ (g ((n + k -
i) mod n) ⊗ x [∧] ((n + k - i) mod n))))
    by (intro monoid-sum-list-swap m-closed nat-pow-closed) simp-all
  also have ... = (⊕ i ← [0..<n]. f i ⊗ x [∧] i ⊗ (⊕ k ← [0..<n]. (g ((n + k -
i) mod n) ⊗ x [∧] ((n + k - i) mod n))))
    by (intro monoid-sum-list-cong monoid-sum-list-in-left m-closed nat-pow-closed)
    simp-all
  also have ... = (⊕ i ← [0..<n]. f i ⊗ x [∧] i ⊗ (⊕ j ← [0..<n]. (g j ⊗ x [∧] j)))
    (is (⊕ i ← -. - ⊗ ?lhs i) = (⊕ i ← -. - ⊗ ?rhs i))
    proof -
      have ∧ i. i ∈ set [0..<n] ⇒ ?lhs i = ?rhs i
      proof (intro monoid-sum-list-index-permutation[symmetric] m-closed nat-pow-closed)
        fix i
        assume i ∈ set [0..<n]
        have bij-betw (λia. (n - i + ia) mod n) {0..<n} {0..<n}
          by (intro const-add-mod-bij)
        also have bij-betw (λia. (n - i + ia) mod n) {0..<n} {0..<n} =
          bij-betw (λia. (n + ia - i) mod n) {0..<n} {0..<n}
          apply (intro bij-betw-cong)
          using ⟨i ∈ set [0..<n]⟩ by simp
      qed
    end

```

**finally show** *bij-betw* ( $\lambda ia. (n + ia - i) \text{ mod } n$ ) (*set*  $[0..<n]$ ) (*set*  $[0..<n]$ )  
**by** *simp*  
**qed** *simp-all*  
**then show** *?thesis*  
**by** (*intro monoid-sum-list-cong*) (*intro arg-cong*[**where**  $f = (\otimes) \text{ -}$ ])  
**qed**  
**also have**  $\dots = (\bigoplus i \leftarrow [0..<n]. f\ i \otimes x\ [\hat{\ } i]) \otimes (\bigoplus j \leftarrow [0..<n]. (g\ j \otimes x\ [\hat{\ } j]))$   
**by** (*intro monoid-sum-list-in-right monoid-sum-list-closed*) *simp-all*  
**finally show** *?thesis* **by** *argo*  
**qed**

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

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

**lemma** *cyclic-convolution-length*[*simp*]:  
 $\text{length } (a \star b) = n$  **unfolding** *cyclic-convolution-def* **by** *simp*

**lemma** *cyclic-convolution-nth*:  
 $i < n \implies (a \star b)\ !\ i = (\bigoplus \sigma \leftarrow [0..<n]. (a\ !\ \sigma \otimes b\ !\ ((n + i - \sigma) \text{ mod } n)))$   
**unfolding** *cyclic-convolution-def* **by** *simp*

**lemma** *cyclic-convolution-closed*:  
**assumes**  $\text{length } a = n$   $\text{length } b = n$   
**assumes**  $\text{set } a \subseteq \text{carrier } R$   $\text{set } b \subseteq \text{carrier } R$   
**shows**  $\text{set } (a \star b) \subseteq \text{carrier } R$   
**proof** (*intro set-subseteqI*)  
**fix**  $i$   
**assume**  $i < \text{length } (a \star b)$   
**then have**  $i < n$  **using** *assms(1)* *assms(2)* **by** *simp*  
**then have**  $(a \star b)\ !\ i = (\bigoplus \sigma \leftarrow [0..<n]. (a\ !\ \sigma \otimes b\ !\ ((n + i - \sigma) \text{ mod } n)))$   
**using** *cyclic-convolution-nth* **by** *presburger*  
**also have**  $\dots \in \text{carrier } R$   
**apply** (*intro monoid-sum-list-closed m-closed*)  
**subgoal for**  $\sigma$  **using** *set-subseteqD[OF assms(3)]*  $\langle \text{length } a = n \rangle$  **by** *simp*  
**subgoal for**  $\sigma$  **using** *set-subseteqD[OF assms(4)]*  $\langle \text{length } b = n \rangle$  **by** *simp*  
**done**  
**finally show**  $(a \star b)\ !\ i \in \text{carrier } R$  .  
**qed**

**theorem** *convolution-rule*:  
**assumes**  $\text{length } a = n$   
**assumes**  $\text{length } b = n$   
**assumes**  $\text{set } a \subseteq \text{carrier } R$   
**assumes**  $\text{set } b \subseteq \text{carrier } R$

```

assumes root-of-unity  $n \mu$ 
assumes  $i < n$ 
shows  $NTT \mu a ! i \otimes NTT \mu b ! i = NTT \mu (a \star b) ! i$ 
proof (cases  $n > 0$ )
  case False
  then show ?thesis using  $\langle i < n \rangle$  by simp
next
  case True

  then interpret root-group : group roots-of-unity-group  $n$ 
    by (rule roots-of-unity-group-is-group)

  interpret root-subgroup : multiplicative-subgroup  $R \{\mu. \text{root-of-unity } n \mu\}$  roots-of-unity-group
   $n$ 
  apply unfold-locales
  subgoal using roots-Units  $\langle n > 0 \rangle$  by blast
  subgoal unfolding roots-of-unity-group-def[OF  $\langle n > 0 \rangle$ ] by simp
  done

  have  $\mu \in \text{carrier } R$  using assms(5) by simp
  have  $NTT \mu a ! i \otimes NTT \mu b ! i =$ 
     $(\bigoplus_{j \leftarrow [0..<n]}. a ! j \otimes (\mu [\uparrow] i) [\uparrow] j) \otimes (\bigoplus_{j \leftarrow [0..<n]}. b ! j \otimes (\mu [\uparrow] i) [\uparrow]$ 
   $j)$ 
  unfolding NTT-nth[OF assms(1)  $\langle i < n \rangle$ ] NTT-nth[OF assms(2)  $\langle i < n \rangle$ ] by
  argo
  also have  $\dots = (\bigoplus_{j \leftarrow [0..<n]}. (\bigoplus_{k \leftarrow [0..<n]}. (a ! k) \otimes (b ! ((n + j - k)$ 
   $\text{mod } n))) \otimes (\mu [\uparrow] i) [\uparrow] j)$ 
  apply (intro root-of-unity-power-sum-product root-of-unity-nat-pow-closed)
  using True  $\langle \text{root-of-unity } n \mu \rangle$  set-subseteqD[OF assms(3)] set-subseteqD[OF
  assms(4)] assms(1) assms(2)
  by simp-all
  also have  $\dots = (\bigoplus_{j \leftarrow [0..<n]}. (a \star b) ! j \otimes (\mu [\uparrow] i) [\uparrow] j)$ 
  apply (intro monoid-sum-list-cong arg-cong[where  $f = \lambda j. j \otimes -$ ] cyclic-convolution-nth[symmetric])
  by simp
  also have  $\dots = NTT \mu (a \star b) ! i$ 
  apply (intro NTT-nth[symmetric]) using  $\langle i < n \rangle$  by simp-all
  finally show ?thesis .
qed

end

end

end

```

## 2.4 Fast Number Theoretic Transforms in Rings

```

theory FNFT-Rings
  imports NTT-Rings Number-Theoretic-Transform.Butterfly

```

**begin**

**context** *cring* **begin**

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

**lemma** *NTT-recursion*:

**assumes** *even n*

**assumes** *primitive-root n μ*

**assumes**<sub>[simp]</sub>: *length a = n*

**assumes**<sub>[simp]</sub>: *j < n*

**assumes**<sub>[simp]</sub>: *set a ⊆ 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**  $(NTT\ \mu\ a) ! j = (NTT\ (\mu\ [\wedge]\ (2::nat)) [a ! i. i \leftarrow filter\ even\ [0..<n]]) ! j'$   
 $\oplus \mu\ [\wedge]\ j \otimes (NTT\ (\mu\ [\wedge]\ (2::nat)) [a ! i. i \leftarrow filter\ odd\ [0..<n]]) ! j'$

**proof** –

**from** *assms* **have**  $n > 0$  **by** *linarith*

**have**<sub>[simp]</sub>:  $\mu \in carrier\ R$  **using**  $\langle primitive-root\ n\ \mu \rangle$  **unfolding** *primitive-root-def*  
*root-of-unity-def* **by** *blast*

**then** **have**  $\mu\text{-pow-carrier}$ <sub>[simp]</sub>:  $\mu\ [\wedge]\ i \in carrier\ R$  **for**  $i :: nat$  **by** *simp*

**show**  $j' < n\ div\ 2$  **unfolding** *j'-def* **using**  $\langle j < n \rangle$   $\langle even\ n \rangle$  **by** *fastforce*

**show** *j'-alt*:  $j = (if\ j < n\ div\ 2\ then\ j' else\ j' + n\ div\ 2)$

**unfolding** *j'-def* **by** *simp*

**have** *a-even-carrier*<sub>[simp]</sub>:  $a ! (2 * i) \in carrier\ R$  **if**  $i < n\ div\ 2$  **for**  $i$

**using** *set-subseteqD[OF <set a ⊆ carrier R>]* *assms* **that** **by** *simp*

**have** *a-odd-carrier*<sub>[simp]</sub>:  $a ! (2 * i + 1) \in carrier\ R$  **if**  $i < n\ div\ 2$  **for**  $i$

**using** *set-subseteqD[OF <set a ⊆ carrier R>]* *assms* **that** **by** *simp*

**have**  $\mu\text{-pow}$ :  $\mu\ [\wedge]\ (j * (2 * i)) = (\mu\ [\wedge]\ (2::nat))\ [\wedge]\ (j' * i)$  **for**  $i$

**proof** –

**have**  $\mu\ [\wedge]\ (j * (2 * i)) = (\mu\ [\wedge]\ (j * 2))\ [\wedge]\ i$

**using** *mult.assoc nat-pow-pow[symmetric]* **by** *simp*

**also** **have**  $\mu\ [\wedge]\ (j * 2) = \mu\ [\wedge]\ (j' * 2)$

**proof** (*cases j < n div 2*)

**case** *True*

**then** **show** *?thesis* **unfolding** *j'-def* **by** *simp*

**next**

**case** *False*

**then** **have**  $\mu\ [\wedge]\ (j * 2) = \mu\ [\wedge]\ (j' * 2 + n)$

**using** *j'-alt* **by** (*simp add: <even n>*)

**also** **have**  $\dots = \mu\ [\wedge]\ (j' * 2)$

**using**  $\langle n > 0 \rangle$   $\langle primitive-root\ n\ \mu \rangle$

**by** (*intro root-of-unity-powers-nat[of n]*) *auto*

**finally** **show** *?thesis* .

**qed**

**finally** **show** *?thesis* **unfolding** *nat-pow-pow[OF <μ ∈ carrier R>]*

**by** (*simp add: mult.assoc mult.commute*)

qed

**have**  $(NTT \mu a) ! j = (\bigoplus i \leftarrow [0..<n]. a ! i \otimes (\mu [\uparrow] (j * i)))$   
**using**  $NTT\text{-}nth\text{-}2[of\ a\ n\ j\ \mu]$  **by**  $simp$   
**also have**  $\dots = (\bigoplus i \leftarrow [0..<n\ div\ 2]. a ! (2 * i) \otimes (\mu [\uparrow] (j * (2 * i))))$   
 $\oplus (\bigoplus i \leftarrow [0..<n\ div\ 2]. a ! (2 * i + 1) \otimes (\mu [\uparrow] (j * (2 * i + 1))))$   
**using**  $\langle even\ n \rangle$   
**by**  $(intro\ monoid\text{-}sum\text{-}list\text{-}even\text{-}odd\text{-}split\ m\text{-}closed\ nat\text{-}pow\text{-}closed\ set\text{-}subsetqD)$   
 $simp\text{-}all$   
**also have**  $(\bigoplus i \leftarrow [0..<n\ div\ 2]. a ! (2 * i + 1) \otimes (\mu [\uparrow] (j * (2 * i + 1))))$   
 $= (\bigoplus i \leftarrow [0..<n\ div\ 2]. \mu [\uparrow] j \otimes (a ! (2 * i + 1) \otimes (\mu [\uparrow] (j * (2 * i * i))))))$   
**proof**  $(intro\ monoid\text{-}sum\text{-}list\text{-}cong)$   
**fix**  $i$   
**assume**  $i \in set\ [0..<n\ div\ 2]$   
**then have** $[simp]: i < n\ div\ 2$  **by**  $simp$   
**have**  $a ! (2 * i + 1) \otimes (\mu [\uparrow] (j * (2 * i + 1))) =$   
 $a ! (2 * i + 1) \otimes (\mu [\uparrow] (j * (2 * i))) \otimes \mu [\uparrow] j$   
**unfolding**  $distrib\text{-}left\ mult\text{-}1\text{-}right$   
**unfolding**  $nat\text{-}pow\text{-}mult[symmetric, OF\ \langle \mu \in carrier\ R \rangle]$   
**by**  $(rule\ refl)$   
**also have**  $\dots = (a ! (2 * i + 1) \otimes \mu [\uparrow] (j * (2 * i))) \otimes \mu [\uparrow] j$   
**using**  $a\text{-}odd\text{-}carrier[OF\ \langle i < n\ div\ 2 \rangle]$   
**by**  $(intro\ m\text{-}assoc[symmetric]; simp)$   
**also have**  $\dots = \mu [\uparrow] j \otimes (a ! (2 * i + 1) \otimes \mu [\uparrow] (j * (2 * i)))$   
**using**  $a\text{-}odd\text{-}carrier[OF\ \langle i < n\ div\ 2 \rangle]$   
**by**  $(intro\ m\text{-}comm; simp)$   
**finally show**  $a ! (2 * i + 1) \otimes \mu [\uparrow] (j * (2 * i + 1)) = \dots$  .  
**qed**  
**also have**  $\dots = \mu [\uparrow] j \otimes (\bigoplus i \leftarrow [0..<n\ div\ 2]. a ! (2 * i + 1) \otimes (\mu [\uparrow] (j * (2 * i * i))))$   
**using**  $a\text{-}odd\text{-}carrier$  **by**  $(intro\ monoid\text{-}sum\text{-}list\text{-}in\text{-}left; simp)$   
**finally have**  $(NTT \mu a) ! j = (\bigoplus i \leftarrow [0..<n\ div\ 2]. a ! (2 * i) \otimes (\mu [\uparrow] (2::nat))$   
 $[\uparrow] (j' * i))$   
 $\oplus \mu [\uparrow] j \otimes (\bigoplus i \leftarrow [0..<n\ div\ 2]. a ! (2 * i + 1) \otimes (\mu [\uparrow] (2::nat)) [\uparrow] (j' * i)))$   
**unfolding**  $\mu\text{-}pow$  .  
**also have**  $\dots = (\bigoplus i \leftarrow [0..<n\ div\ 2]. [a ! i. i \leftarrow filter\ even\ [0..<n]] ! i \otimes (\mu [\uparrow]$   
 $(2::nat)) [\uparrow] (j' * i))$   
 $\oplus \mu [\uparrow] j \otimes (\bigoplus i \leftarrow [0..<n\ div\ 2]. [a ! i. i \leftarrow filter\ odd\ [0..<n]] ! i \otimes (\mu [\uparrow]$   
 $(2::nat)) [\uparrow] (j' * i))$   
**by**  $(intro\text{-}cong\ [cong\text{-}tag\text{-}2\ (\oplus),\ cong\text{-}tag\text{-}2\ (\otimes)]\ more: monoid\text{-}sum\text{-}list\text{-}cong)$   
 $(simp\text{-}all\ add: filter\text{-}even\text{-}nth\ length\text{-}filter\text{-}even\ length\text{-}filter\text{-}odd\ filter\text{-}odd\text{-}nth)$   
**also have**  $\dots = (NTT (\mu [\uparrow] (2::nat)) [a ! i. i \leftarrow filter\ even\ [0..<n]]) ! j'$   
 $\oplus \mu [\uparrow] j \otimes (NTT (\mu [\uparrow] (2::nat)) [a ! i. i \leftarrow filter\ odd\ [0..<n]]) ! j'$   
**by**  $(intro\text{-}cong\ [cong\text{-}tag\text{-}2\ (\oplus),\ cong\text{-}tag\text{-}2\ (\otimes)]\ more: NTT\text{-}nth\text{-}2[symmetric])$   
 $(simp\text{-}all\ add: length\text{-}filter\text{-}even\ length\text{-}filter\text{-}odd\ \langle even\ n \rangle\ \langle j' < n\ div\ 2 \rangle)$   
**finally show**  $(NTT \mu a) ! j = \dots$  .  
**qed**

**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**  $(NTT\ \mu\ a)\ !\ j =$   
 $(NTT\ (\mu\ [\ ]\ (2::nat))\ [a\ !\ i.\ i \leftarrow filter\ even\ [0..<n]])\ !\ j$   
 $\oplus\ \mu\ [\ ]\ j \otimes (NTT\ (\mu\ [\ ]\ (2::nat))\ [a\ !\ i.\ i \leftarrow filter\ odd\ [0..<n]])\ !\ j$   
**proof** –  
**have**  $j < n$  **using**  $\langle j < n\ div\ 2 \rangle$  **by** *linarith*  
**show** *?thesis*  
**using** *NTT-recursion* $[OF\ \langle even\ n \rangle\ \langle primitive-root\ n\ \mu \rangle\ \langle length\ a = n \rangle\ \langle j < n \rangle$   
 $\langle set\ a \subseteq carrier\ R \rangle]$   
**using**  $\langle j < n\ div\ 2 \rangle$  **by** *presburger*  
**qed**

**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*:  $\mu\ [\ ]\ (n\ div\ 2) = \ominus\ \mathbf{1}$   
**shows**  $(NTT\ \mu\ a)\ !\ (n\ div\ 2 + j) =$   
 $(NTT\ (\mu\ [\ ]\ (2::nat))\ [a\ !\ i.\ i \leftarrow filter\ even\ [0..<n]])\ !\ j$   
 $\oplus\ \mu\ [\ ]\ j \otimes (NTT\ (\mu\ [\ ]\ (2::nat))\ [a\ !\ i.\ i \leftarrow filter\ odd\ [0..<n]])\ !\ j$   
**proof** –  
**from** *assms* **have**  $\mu \in carrier\ R$  **unfolding** *primitive-root-def* *root-of-unity-def*  
**by** *simp*  
**then** **have** *carrier-1*:  $\mu\ [\ ]\ j \in carrier\ R$   
**by** *simp*  
**have** *carrier-2*:  $NTT\ (\mu\ [\ ]\ (2::nat))\ (map\ (!)\ a\ (filter\ odd\ [0..<n]))\ !\ j \in$   
 $carrier\ R$   
**apply** (*intro* *NTT-nth-closed* $[where\ n = n\ div\ 2]$ )  
**subgoal** **using**  $\langle set\ a \subseteq carrier\ R \rangle\ \langle length\ a = n \rangle$  **by** *fastforce*  
**subgoal** **using**  $\langle \mu \in carrier\ R \rangle$  **by** *simp*  
**subgoal** **by** (*simp* *add: length-filter-odd*)  
**subgoal** **using**  $\langle j < n\ div\ 2 \rangle$  .  
**done**  
**have**  $n\ div\ 2 + j < n$  **using**  $\langle j < n\ div\ 2 \rangle\ \langle even\ n \rangle$  **by** *linarith*  
**then** **have**  $(NTT\ \mu\ a)\ !\ (n\ div\ 2 + j) =$   
 $(NTT\ (\mu\ [\ ]\ (2::nat))\ [a\ !\ i.\ i \leftarrow filter\ even\ [0..<n]])\ !\ j$   
 $\oplus\ \mu\ [\ ]\ (n\ div\ 2 + j) \otimes (NTT\ (\mu\ [\ ]\ (2::nat))\ [a\ !\ i.\ i \leftarrow filter\ odd\ [0..<n]])$   
 $\ !\ j$   
**using** *NTT-recursion* $[OF\ \langle even\ n \rangle\ \langle primitive-root\ n\ \mu \rangle\ \langle length\ a = n \rangle\ \langle n\ div\ 2$   
 $+ j < n \rangle\ \langle set\ a \subseteq carrier\ R \rangle]$   
**by** *simp*

**also have**  $\mu [\wedge] (n \text{ div } 2 + j) = \ominus (\mu [\wedge] j)$   
**unfolding** *nat-pow-mult[symmetric, OF  $\langle \mu \in \text{carrier } R \rangle$  halfway-property*  
**by** (*intro minus-eq-mult-one[symmetric]; simp add:  $\langle \mu \in \text{carrier } R \rangle$* )  
**finally show** *?thesis unfolding minus-eq l-minus[OF carrier-1 carrier-2]* .  
**qed**

**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**  $\mu [\wedge] (n \text{ div } 2) = \ominus \mathbf{1}$   
**shows** *NTT  $\mu$  a ! j  $\ominus$  NTT  $\mu$  a ! (n div 2 + j) = nat-embedding 2  $\otimes$  ( $\mu [\wedge] j$*   
 $\otimes$  *NTT ( $\mu [\wedge] (2::\text{nat}))$*  (*map (!) a*) (*filter odd [0..<n]*)) ! j)  
**proof** –  
**have**[*simp*]:  $\mu \in \text{carrier } R$  **using**  $\langle \text{primitive-root } n \ \mu \rangle$  **unfolding** *primitive-root-def*  
*root-of-unity-def* **by** *blast*  
**define** *ntt1* **where** *ntt1 = NTT ( $\mu [\wedge] (2::\text{nat}))$*  (*map (!) a*) (*filter even [0..<n]*))  
! j  
**have** *ntt1  $\in$  carrier R* **unfolding** *ntt1-def*  
**apply** (*intro set-subseteqD[OF NTT-closed] set-subseteqI*)  
**subgoal for** *i*  
**using** *set-subseteqD[OF  $\langle \text{set } a \subseteq \text{carrier } R \rangle$*   
**by** (*simp add: filter-even-nth  $\langle \text{length } a = n \rangle$   $\langle \text{even } n \rangle$  length-filter-even*)  
**subgoal by** *simp*  
**subgoal using** *assms* **by** (*simp add: length-filter-even  $\langle \text{even } n \rangle$* )  
**done**  
**define** *ntt2* **where** *ntt2 = NTT ( $\mu [\wedge] (2::\text{nat}))$*  (*map (!) a*) (*filter odd [0..<n]*))  
! j  
**have** *ntt2  $\in$  carrier R* **unfolding** *ntt2-def*  
**apply** (*intro set-subseteqD[OF NTT-closed] set-subseteqI*)  
**subgoal for** *i*  
**using** *set-subseteqD[OF  $\langle \text{set } a \subseteq \text{carrier } R \rangle$*   
**by** (*simp add: filter-odd-nth  $\langle \text{length } a = n \rangle$   $\langle \text{even } n \rangle$  length-filter-odd*)  
**subgoal by** *simp*  
**subgoal using** *assms* **by** (*simp add: length-filter-odd  $\langle \text{even } n \rangle$* )  
**done**  
**have** *NTT  $\mu$  a ! j  $\ominus$  NTT  $\mu$  a ! (n div 2 + j) =*  
*(ntt1  $\oplus$   $\mu [\wedge] j \otimes$  ntt2)  $\ominus$  (ntt1  $\ominus$   $\mu [\wedge] j \otimes$  ntt2)*  
**apply** (*intro arg-cong2[where f =  $\lambda i j. i \ominus j$ ]*)  
**unfolding** *ntt1-def ntt2-def*  
**subgoal by** (*intro NTT-recursion-1 assms*)  
**subgoal by** (*intro NTT-recursion-2 assms*)  
**done**  
**also have** *... =  $\mu [\wedge] j \otimes$  (ntt2  $\oplus$  ntt2)*  
**using**  $\langle \text{ntt1} \in \text{carrier } R \rangle$   $\langle \text{ntt2} \in \text{carrier } R \rangle$  *nat-pow-closed[OF  $\langle \mu \in \text{carrier}$*   
*R  $\rangle$*   
**by** *algebra*

```

also have ... =  $\mu [\wedge] j \otimes ((\mathbf{1} \oplus \mathbf{1}) \otimes ntt2)$ 
  using  $\langle ntt2 \in \text{carrier } R \rangle$  one-closed by algebra
also have ... =  $\mu [\wedge] j \otimes (\text{nat-embedding } 2 \otimes ntt2)$ 
  by (simp add: numeral-2-eq-2)
also have ... =  $\text{nat-embedding } 2 \otimes (\mu [\wedge] j \otimes ntt2)$ 
  using nat-pow-closed[OF  $\langle \mu \in \text{carrier } R \rangle \langle ntt2 \in \text{carrier } R \rangle$  nat-embedding-closed
  by algebra
finally show ?thesis unfolding ntt2-def .
qed

```

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

```

lemma FNTT-term-aux[simp]: length (filter P [0..<l]) < Suc l
  by (metis diff-zero le-imp-less-Suc length-filter-le length-upt)
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 ( $\mu$  [ $\wedge$ ] (2::nat)) nums1;
    c = FNTT ( $\mu$  [ $\wedge$ ] (2::nat)) nums2;
    g = [b!i  $\oplus$  ( $\mu$  [ $\wedge$ ] i)  $\otimes$  c!i. i  $\leftarrow$  [0..<(n div 2)]];
    h = [b!i  $\ominus$  ( $\mu$  [ $\wedge$ ] i)  $\otimes$  c!i. i  $\leftarrow$  [0..<(n div 2)]]
    in g@h)
lemmas [simp del] = FNTT-term-aux

```

```

declare FNTT.simps[simp del]

```

```

lemma length-FNTT[simp]:
  assumes length a = 2 ^ k
  shows length (FNTT  $\mu$  a) = length a
  using assms
proof (induction rule: FNTT.induct)
  case (1  $\mu$ )
  then show ?case by simp
next
  case (2  $\mu$  x)
  then show ?case by (simp add: FNTT.simps)
next
  case (3  $\mu$  x y)
  then show ?case by (simp add: FNTT.simps)
next
  case (4  $\mu$  a1 a2 a3 as)
  define a where a = a1 # a2 # a3 # as
  define n where n = length a
  with a-def have even n using 4(3)
  by (cases k = 0) simp-all
  define nums1 where nums1 = [a!i. i  $\leftarrow$  filter even [0..<n]]

```

```

define nums2 where nums2 = [a!i. i ← filter odd [0..n]]
define b where b = FNTT (μ [∧] (2::nat)) nums1
define c where c = FNTT (μ [∧] (2::nat)) nums2
define g where g = [b!i ⊕ (μ [∧] i) ⊗ c!i. i ← [0..(n div 2)]]
define h where h = [b!i ⊖ (μ [∧] i) ⊗ c!i. i ← [0..(n div 2)]]

note defs = a-def n-def nums1-def nums2-def b-def c-def g-def h-def

have length (FNTT μ a) = length g + length h
  using defs by (simp add: Let-def FNTT.simps)
also have ... = (n div 2) + (n div 2) unfolding g-def h-def by simp
also have ... = n using ⟨even n⟩ by fastforce
finally show ?case by (simp only: a-def n-def)
qed

theorem FNTT-NTT:
  assumes[simp]: μ ∈ carrier R
  assumes n = 2 ^ k
  assumes primitive-root n μ
  assumes halfway-property: μ [∧] (n div 2) = ⊖ 1
  assumes[simp]: length a = n
  assumes set a ⊆ carrier R
  shows FNTT μ a = NTT μ a
  using assms
proof (induction μ a arbitrary: n k rule: FNTT.induct)
  case (1 μ)
    then show ?case unfolding NTT-def by simp
  next
    case (2 μ x)
      then have n = 1 by simp
      then have k = 0 using ⟨n = 2 ^ k⟩ by simp
      moreover have x ∈ carrier R using 2 by simp
      ultimately show ?case unfolding NTT-def by (simp add: Let-def FNTT.simps)
  next
    case (3 μ x y)
      then have[simp]: x ∈ carrier R y ∈ carrier R by simp-all
      from 3 have n = 2 by simp
      with ⟨μ [∧] (n div 2) = ⊖ 1⟩ have μ [∧] (1 :: nat) = ⊖ 1 by simp
      then have μ = ⊖ 1 by (simp add: ⟨μ ∈ carrier R⟩)
      have NTT μ [x, y] = [x ⊕ y, x ⊖ y]
        unfolding NTT-def
        apply (simp add: Let-def 3 ⟨μ = ⊖ 1⟩)
        using ⟨x ∈ carrier R⟩ ⟨y ∈ carrier R⟩ by algebra
      then show ?case by (simp add: FNTT.simps)
  next
    case (4 μ a1 a2 a3 as)
      define a where a = a1 # a2 # a3 # as
      then have[simp]: length a = n using 4(7) by simp
      define nums1 where nums1 = [a!i. i ← filter even [0..n]]

```

```

define nums2 where nums2 = [a!i. i ← filter odd [0..n]]
define b where b = FNTT ( $\mu$  [ $\uparrow$ ] (2::nat)) nums1
define c where c = FNTT ( $\mu$  [ $\uparrow$ ] (2::nat)) nums2
define g where g = [b!i  $\oplus$  ( $\mu$  [ $\uparrow$ ] i)  $\otimes$  c!i. i ← [0..(n div 2)]]
then have length g = n div 2 by simp
define h where h = [b!i  $\ominus$  ( $\mu$  [ $\uparrow$ ] i)  $\otimes$  c!i. i ← [0..(n div 2)]]
then have length h = n div 2 by simp

```

```

note defs = a-def nums1-def nums2-def b-def c-def g-def h-def

```

```

have k > 0
  using  $\langle \text{length } (a1 \# a2 \# a3 \# as) = n \rangle \langle n = 2^k \rangle$ 
  by (cases k = 0) simp-all
then have even n n div 2 =  $2^{k-1}$ 
  using  $\langle n = 2^k \rangle$  by (simp-all add: power-diff)

```

```

have FNTT  $\mu$  (a1 # a2 # a3 # as) = g @ h
  unfolding FNTT.simps
  using  $\langle \text{length } (a1 \# a2 \# a3 \# as) = n \rangle$  by (simp only: Let-def defs)
then have FNTT  $\mu$  a = g @ h using a-def by simp

```

```

have recursive-halfway: ( $\mu$  [ $\uparrow$ ] (2::nat)) [ $\uparrow$ ] (n div 2 div 2) =  $\ominus$  1
proof –
  have n ≥ 3
    using  $\langle \text{length } (a1 \# a2 \# a3 \# as) = n \rangle$  by simp
    then have k ≥ 2 using  $\langle n = 2^k \rangle$  by (cases k ∈ {0, 1}) auto
    then have even (n div 2) using  $\langle n \text{ div } 2 = 2^{k-1} \rangle$  by fastforce
    then show ?thesis
      by (simp add: nat-pow-pow  $\langle \mu \in \text{carrier } R \rangle \langle \mu$  [ $\uparrow$ ] (n div 2) =  $\ominus$  1)
qed

```

```

have b = NTT ( $\mu$  [ $\uparrow$ ] (2::nat)) nums1
  unfolding b-def
  apply (intro 4(1)[of n nums1 nums2 n div 2 k - 1])
  subgoal using  $\langle \text{length } (a1 \# a2 \# a3 \# as) = n \rangle$  by simp
  subgoal using nums1-def a-def by simp
  subgoal using nums2-def a-def by simp
  subgoal using  $\langle \mu \in \text{carrier } R \rangle$  by simp
  subgoal using  $\langle n \text{ div } 2 = 2^{k-1} \rangle$  .
  subgoal using primitive-root-recursion  $\langle \text{even } n \rangle \langle \text{primitive-root } n \mu \rangle$  by blast
  subgoal using recursive-halfway .
  subgoal using nums1-def length-filter-even  $\langle \text{even } n \rangle$  by simp
  subgoal
    unfolding nums1-def
    apply (intro set-subseteqI)
    using set-subseteqD[OF]  $\langle \text{set } (a1 \# a2 \# a3 \# as) \subseteq \text{carrier } R \rangle$ 
    by (simp add: a-def[symmetric] filter-even-nth length-filter-even  $\langle \text{even } n \rangle$ )
  done

```

```

have c = NTT (μ [⌈] (2::nat)) nums2
  unfolding c-def
  apply (intro 4(2)[of n nums1 nums2 b n div 2 k - 1])
  subgoal using ⟨length (a1 # a2 # a3 # as) = n⟩ by simp
  subgoal unfolding nums1-def a-def by simp
  subgoal unfolding nums2-def a-def by simp
  subgoal using b-def .
  subgoal using ⟨μ ∈ carrier R⟩ by simp
  subgoal using ⟨n div 2 = 2 ^ (k - 1)⟩ .
  subgoal using primitive-root-recursion ⟨even n⟩ ⟨primitive-root n μ⟩ by blast
  subgoal using recursive-halfway .
  subgoal unfolding nums2-def using length-filter-odd by simp
  subgoal
    unfolding nums2-def
    apply (intro set-subseteqI)
    using set-subseteqD[OF ⟨set (a1 # a2 # a3 # as) ⊆ carrier R⟩]
    by (simp add: a-def[symmetric] filter-odd-nth length-filter-odd)
  done

show ?case
proof (intro nth-equalityI)
  have[simp]: length (FNNT μ (a1 # a2 # a3 # as)) = n
    using ⟨length (a1 # a2 # a3 # as) = n⟩ ⟨n = 2 ^ k⟩ length-FNNT[of a1 #
a2 # a3 # as]
    by blast
  then show length (FNNT μ (a1 # a2 # a3 # as)) = length (NTT μ (a1 #
a2 # a3 # as))
    using NTT-length[of μ a1 # a2 # a3 # as] ⟨length (a1 # a2 # a3 # as)
= n⟩ by argo
  fix i
  assume i < length (FNNT μ (a1 # a2 # a3 # as))
  then have i < n by simp

  have FNNT μ a ! i = NTT μ a ! i
  proof (cases i < n div 2)
  case True
  then have NTT μ a ! i =
    (NTT (μ [⌈] (2::nat)) [a ! i. i ← filter even [0..<n]]) ! i
  ⊕ μ [⌈] i ⊗ (NTT (μ [⌈] (2::nat)) [a ! i. i ← filter odd [0..<n]]) ! i
  apply (intro NTT-recursion-1)
  using True ⟨even n⟩ ⟨primitive-root n μ⟩ ⟨set (a1 # a2 # a3 # as) ⊆
carrier R⟩ a-def
  using ⟨μ ∈ carrier R⟩ ⟨length (a1 # a2 # a3 # as) = n⟩
  by simp-all

  also have ... = (NTT (μ [⌈] (2::nat)) nums1) ! i
  ⊕ μ [⌈] i ⊗ (NTT (μ [⌈] (2::nat)) nums2) ! i
  unfolding nums1-def nums2-def by blast
  also have ... = b ! i ⊕ μ [⌈] i ⊗ c ! i

```

```

    using ⟨b = NTT (μ [⌈] 2) nums1⟩ ⟨c = NTT (μ [⌈] 2) nums2⟩ by blast
  also have ... = g ! i
    unfolding g-def using True by simp
  also have ... = FNTT μ a ! i
    using ⟨FNTT μ a = g @ h⟩ ⟨length g = n div 2⟩ True
    by (simp add: nth-append)

  finally show ?thesis by simp
next
case False
then obtain j where j-def: i = n div 2 + j j < n div 2
  using ⟨i < n⟩ ⟨even n⟩
  by (metis add-diff-inverse-nat add-self-div-2 div-plus-div-distrib-dvd-right
nat-add-left-cancel-less)
  have NTT μ a ! (n div 2 + j) =
    (NTT (μ [⌈] (2::nat)) [a ! i. i ← filter even [0..<n]]) ! j
  ⊖ μ [⌈] j ⊗ (NTT (μ [⌈] (2::nat)) [a ! i. i ← filter odd [0..<n]]) ! j
  apply (intro NTT-recursion-2)
  subgoal using ⟨even n⟩ .
  subgoal using ⟨primitive-root n μ⟩ .
  subgoal using ⟨length (a1 # a2 # a3 # as) = n⟩ a-def by simp
  subgoal using j-def by simp
  subgoal using ⟨set (a1 # a2 # a3 # as) ⊆ carrier R⟩ a-def by simp
  subgoal using ⟨μ [⌈] (n div 2) = ⊖ 1⟩ .
  done

  also have ... = (NTT (μ [⌈] (2::nat)) nums1) ! j
  ⊖ μ [⌈] j ⊗ (NTT (μ [⌈] (2::nat)) nums2) ! j
    unfolding nums1-def nums2-def by blast
  also have ... = b ! j ⊖ μ [⌈] j ⊗ c ! j
    using ⟨b = NTT (μ [⌈] 2) nums1⟩ ⟨c = NTT (μ [⌈] 2) nums2⟩ by blast
  also have ... = h ! j
    unfolding g-def h-def using j-def by simp
  also have ... = FNTT μ a ! i
    using ⟨FNTT μ a = g @ h⟩ ⟨length g = n div 2⟩ j-def
    by (simp add: nth-append)

  finally show ?thesis using j-def by simp
qed
then show FNTT μ (a1 # a2 # a3 # as) ! i = NTT μ (a1 # a2 # a3 #
as) ! i
  using a-def by simp
qed
qed
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..<Suc g]) =
  f 0 # map (λ x. f (x+1)) (filter odd [0..<g])
by (induction g) auto

lemma map-filter-shift': map f (filter odd [0..<Suc g]) =
  map (λ x. f (x+1)) (filter even [0..<g])
by (induction g) auto

lemma filter-comprehension-evens-odds:
  [xs ! i. i ← filter even [0..<length xs]] = evens-odds True xs ∧
  [xs ! i. i ← filter odd [0..<length xs]] = evens-odds False xs
apply(induction xs)
apply simp
subgoal for x xs
apply rule
subgoal
apply(subst evens-odds.simps)
apply(rule trans[of - map (!) (x # xs)] (filter even [0..<Suc (length xs)]))
subgoal by simp
apply(rule trans[OF map-filter-shift[of (!) (x # xs) length xs]])
apply simp
done

apply(subst evens-odds.simps)
apply(rule trans[of - map (!) (x # xs)] (filter odd [0..<Suc (length xs)]))
subgoal by simp
apply(rule trans[OF map-filter-shift'[of (!) (x # xs) length xs]])
apply simp
done
done

lemma FNTT'-termination-aux[simp]: length (evens-odds True xs) < Suc (length
xs)
  length (evens-odds False xs) < Suc (length xs)
by (metis filter-comprehension-evens-odds le-imp-less-Suc length-filter-le length-map
map-nth)+

(End of copy)

lemma map-evens-odds: map f (evens-odds x a) = evens-odds x (map f a)
by (induction x a rule: evens-odds.induct) simp-all

lemma length-evens-odds:
  length (evens-odds True a) = (if even (length a) then length a div 2 else length a
div 2 + 1)

```

```

length (evens-odds False a) = length a div 2
using filter-comprehension-evens-odds[of a] length-filter-even[of length a] length-filter-odd[of
length a]
using length-map by (metis, metis)

```

**lemma** *set-evens-odds*:

```

set (evens-odds x a)  $\subseteq$  set a
by (induction x a rule: evens-odds.induct) fastforce+

```

**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  $\Rightarrow$  '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 = evens-odds True a;
                 nums2 = evens-odds False a;
                 b = FNTT' ( $\mu$  [^] (2::nat)) nums1;
                 c = FNTT' ( $\mu$  [^] (2::nat)) nums2;
                 g = [b!i  $\oplus$  ( $\mu$  [^] i)  $\otimes$  c!i. i  $\leftarrow$  [0.. $(n \text{ div } 2)$ ]];
                 h = [b!i  $\ominus$  ( $\mu$  [^] i)  $\otimes$  c!i. i  $\leftarrow$  [0.. $(n \text{ div } 2)$ ]]
                 in g@h)

```

**lemma** *FNTT'-FNTT*: FNTT'  $\mu$  xs = FNTT  $\mu$  xs

```

apply (induction  $\mu$  xs rule: FNTT'.induct)
subgoal by (simp add: FNTT.simps)
subgoal by (simp add: FNTT.simps)
subgoal by (simp add: FNTT.simps)
subgoal for  $\mu$  a1 a2 a3 as
  unfolding FNTT.simps FNTT'.simps Let-def
  using filter-comprehension-evens-odds[of a1 # a2 # a3 # as] by presburger
done

```

**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 = evens-odds True a;
                 nums2 = evens-odds False a;
                 b = FNTT'' ( $\mu$  [^] (2::nat)) nums1;
                 c = FNTT'' ( $\mu$  [^] (2::nat)) nums2;
                 g = map2 ( $\oplus$ ) b (map2 ( $\otimes$ ) [ $\mu$  [^] i. i  $\leftarrow$  [0.. $(n \text{ div } 2)$ ]] c);
                 h = map2 ( $\lambda x y. x \ominus y$ ) b (map2 ( $\otimes$ ) [ $\mu$  [^] i. i  $\leftarrow$  [0.. $(n \text{ div } 2)$ ]])

```

c)

in  $g@h$ )

**lemma**  $FNTT''$ - $FNTT'$ :

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

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

**using**  $\text{assms}$

**proof** ( $\text{induction } \mu a \text{ arbitrary: } k \text{ rule: } FNTT''.\text{induct}$ )

**case** ( $\_4 \mu a1 a2 a3 as$ )

**define**  $a$  **where**  $a = a1 \# a2 \# a3 \# as$

**define**  $n$  **where**  $n = \text{length } a$

**then have**  $n = 2^{\wedge} k$  **using**  $\_4 a\text{-def}$  **by**  $\text{simp}$

**then have**  $k \geq 2$  **using**  $n\text{-def } a\text{-def}$  **by** ( $\text{cases } k = 0; \text{ cases } k = 1$ )  $\text{simp-all}$

**then have**  $\text{even } n$  **using**  $\langle n = 2^{\wedge} k \rangle$  **by**  $\text{simp}$

**have**  $n \text{ div } 2 = 2^{\wedge} (k - 1)$  **using**  $\langle n = 2^{\wedge} k \rangle \langle k \geq 2 \rangle$  **by** ( $\text{simp add: power-diff}$ )

**then have**  $\text{even } (n \text{ div } 2)$  **using**  $\langle k \geq 2 \rangle$  **by**  $\text{simp}$

**define**  $\text{nums1}$  **where**  $\text{nums1} = \text{evens-odds True } a$

**then have**  $\text{length } \text{nums1} = n \text{ div } 2$

**using**  $\text{length-filter-even[of } n] \text{ filter-comprehension-evens-odds[of } a] n\text{-def } \langle \text{even}$

$n \rangle$

**by** ( $\text{metis length-map}$ )

**define**  $\text{nums2}$  **where**  $\text{nums2} = \text{evens-odds False } a$

**then have**  $\text{length } \text{nums2} = n \text{ div } 2$

**using**  $\text{length-filter-odd[of } n] \text{ filter-comprehension-evens-odds[of } a] n\text{-def}$

**by** ( $\text{metis length-map}$ )

**define**  $b$  **where**  $b = FNTT' (\mu [\_ ] (2::\text{nat})) \text{nums1}$

**then have**  $\text{length } b = n \text{ div } 2$

**by** ( $\text{simp add: } FNTT'\text{-}FNTT \langle \text{length } \text{nums1} = n \text{ div } 2 \rangle \langle n \text{ div } 2 = 2^{\wedge} (k - 1) \rangle$ )

**define**  $c$  **where**  $c = FNTT' (\mu [\_ ] (2::\text{nat})) \text{nums2}$

**then have**  $\text{length } c = n \text{ div } 2$

**by** ( $\text{simp add: } FNTT'\text{-}FNTT \langle \text{length } \text{nums2} = n \text{ div } 2 \rangle \langle n \text{ div } 2 = 2^{\wedge} (k - 1) \rangle$ )

**define**  $g1$  **where**  $g1 = [b!i \oplus (\mu [\_ ] i) \otimes c!i. i \leftarrow [0..<(n \text{ div } 2)]]$

**then have**  $\text{length } g1 = n \text{ div } 2$  **by**  $\text{simp}$

**define**  $h1$  **where**  $h1 = [b!i \ominus (\mu [\_ ] i) \otimes c!i. i \leftarrow [0..<(n \text{ div } 2)]]$

**then have**  $\text{length } h1 = n \text{ div } 2$  **by**  $\text{simp}$

**define**  $g2$  **where**  $g2 = \text{map2 } (\oplus) b (\text{map2 } (\otimes) [\mu [\_ ] i. i \leftarrow [0..<(n \text{ div } 2)]] c)$

**then have**  $\text{length } g2 = n \text{ div } 2$

**by** ( $\text{simp add: } \langle \text{length } b = n \text{ div } 2 \rangle \langle \text{length } c = n \text{ div } 2 \rangle$ )

**have**  $g1 = g2$

**apply** ( $\text{intro nth-equalityI}$ )

**subgoal** **by** ( $\text{simp only: } \langle \text{length } g1 = n \text{ div } 2 \rangle \langle \text{length } g2 = n \text{ div } 2 \rangle$ )

**subgoal** **for**  $i$

**by** ( $\text{simp add: } g1\text{-def } g2\text{-def } \langle \text{length } b = n \text{ div } 2 \rangle \langle \text{length } c = n \text{ div } 2 \rangle$ )

**done**

**define**  $h2$  **where**  $h2 = \text{map2 } (\lambda x y. x \ominus y) b (\text{map2 } (\otimes) [\mu [\_ ] i. i \leftarrow [0..<(n \text{ div } 2)]] c)$

```

then have length h2 = n div 2
  by (simp add: <length b = n div 2> <length c = n div 2>)

have h1 = h2
  apply (intro nth-equalityI)
  subgoal by (simp only: <length h1 = n div 2> <length h2 = n div 2>)
  subgoal for i
    by (simp add: h1-def h2-def <length b = n div 2> <length c = n div 2>)
  done

have 1: FNTT'' (μ [∧] (2::nat)) nums1 = FNTT' (μ [∧] (2::nat)) nums1
  apply (intro 4(1))
  using a-def n-def <length (a1 # a2 # a3 # as) = 2 ^ k> <length nums1 = n
div 2> <n div 2 = 2 ^ (k - 1)>
  by (simp-all add: nums1-def)
have 2: FNTT'' (μ [∧] (2::nat)) nums2 = FNTT' (μ [∧] (2::nat)) nums2
  apply (intro 4(2))
  using a-def n-def <length (a1 # a2 # a3 # as) = 2 ^ k> <length nums2 = n
div 2> <n div 2 = 2 ^ (k - 1)>
  by (simp-all add: nums2-def)

show ?case
  apply (simp only: FNTT'.simps FNTT''.simps)
  apply (simp only: Let-def 1 2 a-def[symmetric] nums1-def[symmetric] nums2-def[symmetric]
    b-def[symmetric] c-def[symmetric])
  using <h1 = h2> <g1 = g2> n-def g1-def h1-def g2-def h2-def
  by argo
qed simp-all

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::nat) > 0  $\implies a \in \text{carrier } R \implies a [\wedge] n = \mathbf{1} \implies a \in \text{Units } R$ 
    using gr0-implies-Suc[of n] nat-pow-Suc2[of a]

```

**by** (*metis Units-one-closed nat-pow-closed unit-factor*)  
**end**

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

**lemma** *ensure-length-correct*[*simp*]:  $length\ (ensure-length\ k\ xs) = k$  **using** *fill-def ensure-length-def* **by** *simp*

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

**by** (*simp add: to-nat-take ensure-length-def*)

**locale** *int-lsbf-mod* =

**fixes**  $k :: nat$

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

**begin**

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

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

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

**by** *simp*

**sublocale** *residues*  $n\ Zn$

**apply** *unfold-locales*

**subgoal** **using** *k-positive* **by** *simp*

**subgoal** **by** (*rule Zn-def*)

**done**

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

*to-residue-ring*  $x = int\ (Nat-LSBF.to-nat\ xs)\ mod\ int\ n$

**lemma** *to-residue-ring-in-carrier*: *to-residue-ring*  $xs \in carrier\ Zn$

**unfolding** *to-residue-ring-def res-carrier-eq* **by** *simp*

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

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

**definition** *reduce* **where**

*reduce*  $xs = ensure-length\ k\ xs$

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

**unfolding** *reduce-def* **using** *fill-def ensure-length-def* **by** *simp*

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

**proof** (*cases length xs ≤ k*)

**case** *True*

**then** **have** *reduce xs = fill k xs* **unfolding** *reduce-def* **using** *fill-def ensure-length-def* **by** *simp*

**also** **have**  $\dots = xs\ @\ (replicate\ (k - length\ xs)\ False)$  **using** *fill-def* **by** *simp*

**finally have**  $\text{Nat-LSBF.to-nat (reduce } xs) = \text{Nat-LSBF.to-nat } xs$  **by** *simp*  
**moreover have**  $\text{Nat-LSBF.to-nat } xs \leq 2^k - 1$  **using** *to-nat-length-upper-bound*[of  
*xs*] *True*  
**by** (*meson diff-le-mono le-trans one-le-numeral power-increasing*)  
**hence**  $\text{Nat-LSBF.to-nat } xs < 2^k$   
**using** *Nat.le-diff-conv2* **by** *auto*  
**ultimately show** *?thesis* **by** *simp*  
**next**  
**case** *False*  
**then have**  $\text{length (take } k \text{ } xs) = k$  **fill**  $k \text{ } xs = xs$   $xs = (\text{take } k \text{ } xs) @ (\text{drop } k \text{ } xs)$   
**using** *fill-def* **by** *simp-all*  
**then have**  $\text{Nat-LSBF.to-nat } xs = \text{Nat-LSBF.to-nat (take } k \text{ } xs) + n * \text{Nat-LSBF.to-nat (drop } k \text{ } xs)$   
**using** *to-nat-app*[of *take k xs drop k xs*] **by** *simp*  
**moreover have**  $\text{Nat-LSBF.to-nat (take } k \text{ } xs) \leq 2^k - 1$   
**using** *to-nat-length-upper-bound*[of *take k xs*]  $\langle \text{length (take } k \text{ } xs) = k \rangle$  **by** *simp*  
**hence**  $\text{Nat-LSBF.to-nat (take } k \text{ } xs) < 2^k$   
**using** *Nat.le-diff-conv2* **by** *auto*  
**ultimately show** *?thesis* **unfolding** *reduce-def* **using** *fill-def ensure-length-def*  
**by** *simp*  
**qed**

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

**definition** *subtract-mod* **where**  
 $\text{subtract-mod } xs \ ys =$   
*(if compare-nat xs ys then*  
 $\text{reduce (subtract-nat ((fill } k \text{ } xs) @ [True]) } ys)$   
*else*  
 $\text{subtract-nat } xs \ ys)$

**lemma** *to-nat-add-mod*:  $\text{Nat-LSBF.to-nat (add-mod } x \ y) = (\text{Nat-LSBF.to-nat } x + \text{Nat-LSBF.to-nat } y) \bmod n$   
**by** (*simp only: to-nat-reduce add-nat-correct add-mod-def*)

**lemma** *to-nat-subtract-mod*:  $\text{length } xs \leq k \implies \text{length } ys \leq k \implies \text{int (Nat-LSBF.to-nat (subtract-mod } xs \ ys)) = (\text{int (Nat-LSBF.to-nat } xs) - \text{int (Nat-LSBF.to-nat } ys)) \bmod n$

**proof** (*cases Nat-LSBF.to-nat xs ≤ Nat-LSBF.to-nat ys*)  
**case** *True*  
**assume**  $\text{length } xs \leq k$   
**assume**  $\text{length } ys \leq k$   
**then have**  $\text{Nat-LSBF.to-nat } ys \leq n - 1$   
**using** *to-nat-length-upper-bound*[of *ys*]  
**by** (*meson diff-le-mono le-trans one-le-numeral power-increasing*)  
**then have**  $\text{Nat-LSBF.to-nat } ys \leq \text{Nat-LSBF.to-nat } xs + n$  **by** *simp*

```

have int (Nat-LSBF.to-nat (subtract-nat (fill k xs @ [True]) ys) mod n)
  = int ((Nat-LSBF.to-nat xs + n - Nat-LSBF.to-nat ys) mod n)
  by (simp add: subtract-nat-correct to-nat-app length-fill ⟨length xs ≤ k⟩)
also have ... = (int (Nat-LSBF.to-nat xs + n - Nat-LSBF.to-nat ys)) mod n
  using zmod-int by simp
also have ... = (int (Nat-LSBF.to-nat xs) + int n - int (Nat-LSBF.to-nat ys))
mod n
  using ⟨Nat-LSBF.to-nat ys ≤ Nat-LSBF.to-nat xs + n⟩ by (simp add: of-nat-diff)
also have ... = (int (Nat-LSBF.to-nat xs) - int (Nat-LSBF.to-nat ys)) mod n
  by (metis diff-add-eq int-ops(3) mod-add-self2 of-nat-power)
finally have int (Nat-LSBF.to-nat (subtract-nat (fill k xs @ [True]) ys) mod n)
= (int (Nat-LSBF.to-nat xs) - int (Nat-LSBF.to-nat ys)) mod n .
  then show ?thesis
    by (simp add: subtract-mod-def compare-nat-correct to-nat-reduce True split:
if-splits)
next
  case False
    assume length xs ≤ k
    then have Nat-LSBF.to-nat xs ≤ n - 1 using to-nat-length-upper-bound[of xs]
      by (meson diff-le-mono le-trans one-le-numeral power-increasing)
    assume length ys ≤ k
    from False have int (Nat-LSBF.to-nat (subtract-nat xs ys)) = int (Nat-LSBF.to-nat
xs) - int (Nat-LSBF.to-nat ys)
      by (simp add: subtract-nat-correct)
    moreover have ... ∈ {0..<n}
    proof -
      have int (Nat-LSBF.to-nat xs) - int (Nat-LSBF.to-nat ys) ≤ int (Nat-LSBF.to-nat
xs) by simp
      also have ... ≤ n - 1 using ⟨Nat-LSBF.to-nat xs ≤ n - 1⟩ n-positive by
linarith
      also have ... < n by simp
      finally have int (Nat-LSBF.to-nat xs) - int (Nat-LSBF.to-nat ys) < n by
simp
    moreover have int (Nat-LSBF.to-nat xs) - int (Nat-LSBF.to-nat ys) ≥ 0
using ⟨¬ Nat-LSBF.to-nat xs ≤ Nat-LSBF.to-nat ys⟩ by simp
    ultimately show ?thesis by simp
  qed
  ultimately have int (Nat-LSBF.to-nat (subtract-nat xs ys)) = (int (Nat-LSBF.to-nat
xs) - int (Nat-LSBF.to-nat ys)) mod n
    by simp
  then show ?thesis by (simp add: subtract-mod-def compare-nat-correct to-nat-reduce
False split: if-splits)
qed

lemma length-subtract-mod: length xs ≤ k ⇒ length ys ≤ k ⇒ length (subtract-mod
xs ys) ≤ k
  unfolding subtract-mod-def
  apply (cases compare-nat xs ys)

```

**using** *subtract-nat-aux*[of *xs ys*]  
**by** (*auto simp: Let-def reduce-def ensure-length-def*)

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

**proof** –

**have**  $\text{to-residue-ring } (\text{add-mod } x \ y) = \text{to-residue-ring } (\text{reduce } (\text{add-nat } x \ y))$   
**unfolding** *add-mod-def* **by** *simp*  
**also have**  $\dots = (\text{Nat-LSBF.to-nat } x + \text{Nat-LSBF.to-nat } y) \text{ mod } n$   
**using** *to-nat-reduce add-nat-correct to-residue-ring-def* **by** *simp*  
**also have**  $\dots = (\text{int } (\text{Nat-LSBF.to-nat } x) \text{ mod } n + (\text{int } (\text{Nat-LSBF.to-nat } y) \text{ mod } n)) \text{ mod } n$   
**by** (*simp add: zmod-int mod-add-eq*)  
**finally show** *?thesis*  
**by** (*simp only: res-add-eq to-residue-ring-def*)

**qed**

**lemma** *subtract-mod-correct*:

**assumes**  $\text{length } x \leq k$

**assumes**  $\text{length } y \leq k$

**assumes**  $n > 1$

**shows**  $\text{to-residue-ring } (\text{subtract-mod } x \ y) = \text{to-residue-ring } x \ominus_{Z_n} \text{to-residue-ring } y$

**proof** –

**have**  $\text{to-residue-ring } (\text{subtract-mod } x \ y) = \text{int } (\text{Nat-LSBF.to-nat } (\text{subtract-mod } x \ y)) \text{ mod int } n$

**unfolding** *to-residue-ring-def* **by** *argo*

**also have**  $\dots = (\text{int } (\text{Nat-LSBF.to-nat } x) - (\text{int } (\text{Nat-LSBF.to-nat } y))) \text{ mod int } n$

**by** (*simp add: to-nat-subtract-mod assms*)

**also have**  $\dots = (\text{to-residue-ring } x + (- \text{to-residue-ring } y \text{ mod } n)) \text{ mod } n$

**unfolding** *diff-conv-add-uminus to-residue-ring-def*

**by** (*simp add: mod-add-eq mod-diff-right-eq*)

**also have**  $\dots = (\text{to-residue-ring } x + (\ominus_{\text{residue-ring } n} (\text{to-residue-ring } y \text{ mod } n))) \text{ mod } n$

**apply** (*intro-cong [cong-tag-2 (mod), cong-tag-2 (+)] more: refl*)

**using** *residues.neg-cong[symmetric, of n]* **unfolding** *residues-def* **using**  $\langle n > 1 \rangle$

**by** (*metis int-ops(2) nat-int-comparison(2)*)

**also have**  $\dots = \text{to-residue-ring } x \ominus_{\text{residue-ring } n} (\text{to-residue-ring } y \text{ mod } n)$

**unfolding** *a-minus-def*

**by** (*simp add: residue-ring-def*)

**also have**  $\text{to-residue-ring } y \text{ mod } n = \text{to-residue-ring } y$

**using** *to-residue-ring-def* **by** *simp*

**finally show** *?thesis* **unfolding** *Zn-def* .

**qed**

**lemma** *length-from-residue-ring*:  $x < 2^{\wedge} k \implies \text{length } (\text{from-residue-ring } x) = k$

**proof** –

```

assume  $x < 2^k$ 
have truncated (Nat-LSBF.from-nat (nat x))
  using truncate-from-nat by simp
moreover have Nat-LSBF.to-nat (Nat-LSBF.from-nat (nat x)) = nat x
  using nat-lsbf.to-from by simp
ultimately have length (Nat-LSBF.from-nat (nat x))  $\leq k$  using  $\langle x < 2^k \rangle$ 
to-nat-length-bound-truncated
  by simp
then show length (from-residue-ring x) = k
  unfolding from-residue-ring-def using length-fill by simp
qed

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 =  $\{x :: \text{bool list. length } x = k\}$ 
proof –
  show abstract-representation-2 from-residue-ring to-residue-ring {0..<int n}
    apply unfold-locales
    unfolding to-residue-ring-def from-residue-ring-def by simp-all
  then interpret int-lsbf-mod: abstract-representation-2 from-residue-ring to-residue-ring
{0..<int n} .
  show int-lsbf-mod.reduce = reduce
    unfolding int-lsbf-mod.reduce-def reduce-def
    apply (intro ext)
    apply (intro nat-lsbf-eqI)
    subgoal for x
      unfolding from-residue-ring-def to-nat-fill to-nat-from-nat
    proof –
      have nat (to-residue-ring x) = nat (int (Nat-LSBF.to-nat x) mod int n)
        by (simp add: from-residue-ring-def to-residue-ring-def ensure-length-def
to-nat-take)
      also have ... = Nat-LSBF.to-nat x mod n
        unfolding zmod-int[symmetric] nat-int by (rule refl)
      also have ... = Nat-LSBF.to-nat (ensure-length k x)
        unfolding ensure-length-def by (simp add: to-nat-take)
      finally show nat (to-residue-ring x) = ... .
    qed
    subgoal for x
      proof –
        have length (from-residue-ring (to-residue-ring x)) = k
          apply (intro length-from-residue-ring)
          unfolding to-residue-ring-def
          using mod-less-divisor[OF n-positive] by simp
        then show ?thesis by simp
      qed
    done
  show int-lsbf-mod.representations =  $\{x :: \text{bool list. length } x = k\}$ 
  proof (intro equalityI subsetI)

```

```

fix x
assume x ∈ int-lsbf-mod.representations
then obtain y where y < 2 ^ k x = from-residue-ring y
  unfolding int-lsbf-mod.representations-def by auto
then have length x = k by (simp add: length-from-residue-ring)
then show x ∈ {x. length x = k} by simp
next
  fix x :: bool list
  assume x ∈ {x. length x = k}
  then have length x = k by simp
  have from-residue-ring (to-residue-ring x) = int-lsbf-mod.reduce x
    using int-lsbf-mod.reduce-def by simp
  also have ... = reduce x using ⟨int-lsbf-mod.reduce = reduce⟩ by simp
  also have ... = x using ⟨length x = k⟩ unfolding reduce-def ensure-length-def
  fill-def by simp
  finally show x ∈ int-lsbf-mod.representations
    unfolding int-lsbf-mod.representations-def
    using int-lsbf-mod.to-type-in-represented-set
    by (metis imageI)
  qed
qed

lemma add-mod-closed: length (add-mod x y) = k
  using int-lsbf-mod.range-reduce add-mod-def by blast

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
  unfolding ensure-length-tm-def ensure-length-def by simp

lemma time-ensure-length-tm[simp]: time (ensure-length-tm k xs) = 7 + 2 * length xs + 2 * k
  unfolding ensure-length-tm-def tm-time-simps val-fill-tm time-fill-tm time-take-tm length-fill'
  using add-min-max[of k length xs] by simp

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*  
**unfolding** *reduce-tm-def reduce-def by simp*

**lemma** *time-reduce-tm[simp]: time (reduce-tm xs) = 8 + 2 \* length xs + 2 \* k*  
**unfolding** *reduce-tm-def by simp*

**definition** *add-mod-tm :: nat-lsbf  $\Rightarrow$  nat-lsbf  $\Rightarrow$  nat-lsbf tm where*  
*add-mod-tm xs ys =1 xs +<sub>nt</sub> ys  $\gg$  reduce-tm*

**lemma** *val-add-mod-tm[simp, val-simp]: val (add-mod-tm xs ys) = add-mod xs ys*  
**unfolding** *add-mod-tm-def add-mod-def by simp*

**lemma** *time-add-mod-tm-le: time (add-mod-tm xs ys)  $\leq$  14 + 4 \* max (length xs)*  
*(length ys) + 2 \* k*  
**unfolding** *add-mod-tm-def tm-time-simps val-add-nat-tm time-reduce-tm*  
**apply** *(estimation estimate: time-add-nat-tm-le)*  
**apply** *(estimation estimate: length-add-nat-upper)*  
**by** *simp*

**definition** *subtract-mod-tm :: nat-lsbf  $\Rightarrow$  nat-lsbf  $\Rightarrow$  nat-lsbf tm where*  
*subtract-mod-tm xs ys =1 do {*  
*b  $\leftarrow$  xs  $\leq_{nt}$  ys;*  
*if b then do {*  
*fillx  $\leftarrow$  fill-tm k xs;*  
*fillx1  $\leftarrow$  fillx @<sub>t</sub> [True];*  
*fillx1 -<sub>nt</sub> ys  $\gg$  reduce-tm*  
*} else xs -<sub>nt</sub> ys*  
*}*

**lemma** *val-subtract-mod-tm[simp, val-simp]: val (subtract-mod-tm xs ys) = subtract-mod xs ys*  
**unfolding** *subtract-mod-tm-def subtract-mod-def by simp*

**lemma** *time-subtract-mod-tm-le: time (subtract-mod-tm xs ys)  $\leq$  118 + 51 \* max*  
*k (max (length xs) (length ys))*

**proof** –

**define** *m where m = max k (max (length xs) (length ys))*  
**have** *1: max (length (fill k xs @ [True])) (length ys)  $\leq$  m + 1*  
**unfolding** *length-append length-fill' m-def by (auto simp add: max.assoc)*  
**have** *time (subtract-mod-tm xs ys) = time (xs  $\leq_{nt}$  ys) +*  
*(if xs  $\leq_n$  ys*  
*then time (fill-tm k xs) +*  
*time ((fill k xs) @<sub>t</sub> [True]) +*  
*time ((fill k xs @ [True]) -<sub>nt</sub> ys) +*  
*time (reduce-tm ((fill k xs @ [True]) -<sub>n</sub> ys))*  
*else time (xs -<sub>nt</sub> ys)) + 1*  
**(is ?t = - + (if ?b then ?c else ?d) + 1)**

```

  unfolding subtract-mod-tm-def tm-time-simps val-compare-nat-tm
  val-fill-tm val-append-tm val-subtract-nat-tm by simp
  moreover have ?c ≤ (2 * length xs + k + 5) +
    (max k (length xs) + 1) +
    (30 * m + 78) +
    (10 + 2 * m + 2 * k)
  apply (intro add-mono)
  subgoal unfolding time-fill-tm by simp
  subgoal unfolding time-append-tm length-fill' by simp
  subgoal
    apply (estimation estimate: time-subtract-nat-tm-le)
    apply (itrans 30 * (m + 1) + 48)
    subgoal by (intro add-mono mult-le-mono2 order.refl 1)
    subgoal by simp
  done
  subgoal
    unfolding time-reduce-tm
    apply (estimation estimate: conjunct2[OF subtract-nat-aux])
    apply (estimation estimate: 1)
    by simp
  done
  moreover have ?d ≤ 30 * m + 78
    apply (estimation estimate: time-subtract-nat-tm-le)
    unfolding m-def by simp
  ultimately have ?t ≤ time (xs ≤nt ys) +
    ((2 * length xs + k + 5) +
    (max k (length xs) + 1) +
    (30 * m + 78) +
    (10 + 2 * m + 2 * k)) + 1
    by simp
  also have ... ≤ (13 * m + 23) + ((2 * m + m + 5) + (m + 1) + (30 * m +
  78) + (10 + 2 * m + 2 * m)) + 1
    apply (intro add-mono order.refl)
  subgoal
    apply (estimation estimate: time-compare-nat-tm-le)
    apply (intro add-mono mult-le-mono2 order.refl)
    unfolding m-def by simp
  subgoal unfolding m-def by simp
  subgoal unfolding m-def by simp
  subgoal unfolding m-def by simp
  subgoal unfolding m-def by simp
  done
  also have ... = 118 + 51 * m by simp
  finally show ?thesis unfolding m-def .
qed

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 ^ (length xs) - 1*

**shows** *xs = replicate (length xs) True*

**proof** (*intro iffD2[OF list-is-replicate-iff]*, *rule ccontr*)

**assume**  $\neg (\forall i \in \{0..<length\ xs\}. xs ! i = True)$

**then obtain** *i where i < length xs xs ! i = False* **by** *auto*

**then obtain** *xs1 xs2 where xs = xs1 @ False # xs2*

**by** (*metis(full-types) id-take-nth-drop*)

**then have** *Nat-LSBF.to-nat xs < Nat-LSBF.to-nat (xs1 @ True # xs2)*

**using** *change-bit-ineq[of xs1 xs1 xs2]* **by** *argo*

**also have**  $\dots \leq 2 ^ (length (xs1 @ True # xs2)) - 1$

**by** (*intro to-nat-length-upper-bound*)

**also have**  $\dots = 2 ^ (length xs) - 1$

**using**  $\langle xs = xs1 @ False # xs2 \rangle$  **by** *simp*

**finally show** *False* **using** *assms* **by** *simp*

**qed**

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

**by** (*induction b*) (*simp-all add: residue-ring-def mod-mult-right-eq mult.commute*)

**lemma** (**in** *residues*) *pow-nat-eq*:

$a [\overset{\wedge}{\int}]_R (n :: nat) = a \wedge n \text{ mod } m$

**using** *R-m-def m-gt-one residue-ring-pow* **by** *blast*

**locale** *int-lsbf-fermat* =

**fixes** *k :: nat*

**begin**

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

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

**lemma** *n-gt-1[simp]*:  $n > 1$  **by** *simp*

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

**by** (*metis add-less-mono1 nat-1-add-1 one-less-numeral-iff one-less-power pos2 semiring-norm(76) zero-less-power*)

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

**sublocale** *residues n Fn*

**apply** *unfold-locales*

**subgoal by** *simp*  
**subgoal by** (rule *Fn-def*)  
**done**

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

**lemma** *fermat-non-unique-carrierI*[*intro*]:  
 $\text{length } xs = 2^{\wedge}(k + 1) \implies xs \in \text{fermat-non-unique-carrier}$   
**unfolding** *fermat-non-unique-carrier-def* **by** *simp*

**lemma** *fermat-non-unique-carrierE*[*elim*]:  
 $xs \in \text{fermat-non-unique-carrier} \implies (\text{length } xs = 2^{\wedge}(k + 1) \implies P) \implies P$   
**unfolding** *fermat-non-unique-carrier-def* **by** *simp*

**lemma** *two-pow-half-carrier-length*[*simp*]:  $(\text{int } 2^{\wedge}(2^{\wedge}k)) \bmod n = -1 \bmod n$   
**apply** *simp*  
**using** *zmod-minus1* [of *int n*] *n-positive*  
**by** (*metis add-diff-cancel-left' diff-eq-eq of-nat-0-less-iff of-nat-numeral pos2 zero-less-power zless-add1-eq zmod-minus1*)

**lemma** *two-pow-half-carrier-length-neq-1*:  $2^{\wedge}(2^{\wedge}k) \bmod n \neq 1$   
**by** *simp*

**lemma** *two-pow-carrier-length*[*simp*]:  $(2 :: \text{nat})^{\wedge}(2^{\wedge}(k + 1)) \bmod n = 1$   
**proof** –  
**have**  $\text{int } 2^{\wedge}(2^{\wedge}(k + 1)) \bmod n = 1$   
**proof** –  
**have**  $\text{int } 2^{\wedge}(2^{\wedge}(k + 1)) \bmod n = ((\text{int } 2)^{\wedge}(2 * 2^{\wedge}k)) \bmod n$   
**by** *simp*  
**also have**  $\dots = ((\text{int } 2)^{\wedge}(2^{\wedge}k))^{\wedge}2 \bmod n$   
**using** *power-mult* [of *int 2 2^{\wedge}k 2*]  
**by** (*simp add: mult.commute*)  
**also have**  $\dots = (\text{int } 2^{\wedge}(2^{\wedge}k) * \text{int } 2^{\wedge}(2^{\wedge}k)) \bmod n$   
**by** (*simp add: power2-eq-square*)  
**also have**  $\dots = (((\text{int } 2^{\wedge}(2^{\wedge}k)) \bmod n) * ((\text{int } 2^{\wedge}(2^{\wedge}k)) \bmod n)) \bmod n$   
**by** *simp*  
**also have**  $(\text{int } 2^{\wedge}(2^{\wedge}k)) \bmod n = -1 \bmod n$   
**using** *two-pow-half-carrier-length* .  
**finally have**  $\text{int } 2^{\wedge}(2^{\wedge}(k + 1)) \bmod n = \text{int } 1 \bmod n$   
**by** (*simp add: mod-simps*)  
**thus** *?thesis* **by** *simp*  
**qed**  
**then show** *?thesis*  
**by** (*metis int-ops(2) of-nat-eq-iff of-nat-power zmod-int*)  
**qed**

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

**proof** –  
**have**  $(2::int) [\bigwedge]_{Fn} (2::nat) ^ k = (2::int) ^ ((2::nat) ^ k) \text{ mod } n$   
**by**  $(\text{intro pow-nat-eq})$   
**also have**  $\dots = -1 \text{ mod } n$  **using**  $\text{two-pow-half-carrier-length}$  **by**  $\text{simp}$   
**also have**  $\dots = \ominus_{Fn} \mathbf{1}_{Fn}$   
**using**  $\text{res-neg-eq res-one-eq}$  **by**  $\text{algebra}$   
**finally show**  $?thesis$  .  
**qed**

**lemma**  $\text{two-pow-carrier-length-residue-ring}[\text{simp}]$ :  
 $(2::int) [\bigwedge]_{Fn} (2::nat) ^ (k + 1) = \mathbf{1}_{Fn}$   
**proof** –  
**have**  $(2::int) [\bigwedge]_{Fn} (2::nat) ^ (k + 1) = (2::int) ^ ((2::nat) ^ (k + 1)) \text{ mod } n$   
**by**  $(\text{intro pow-nat-eq})$   
**also have**  $\dots = 1$  **using**  $\text{two-pow-carrier-length zmod-int}$   
**by**  $(\text{metis int-exp-hom int-ops(2) int-ops(3)})$   
**also have**  $\dots = \mathbf{1}_{Fn}$  **by**  $(\text{simp only: res-one-eq})$   
**finally show**  $?thesis$  .  
**qed**

**corollary**  $\text{two-is-unit: } 2 \in \text{Units } Fn$   
**apply**  $(\text{intro pow-one-imp-unit}[\text{of } 2 ^ (k + 1)])$   
**subgoal by**  $\text{simp}$   
**subgoal using**  $\text{res-carrier-eq}$  **by**  $(\text{simp add: self-le-power})$   
**subgoal using**  $\text{two-pow-carrier-length-residue-ring}$  .  
**done**

**corollary**  $\text{two-in-carrier: } 2 \in \text{carrier } Fn$   
**using**  $\text{Units-closed}[\text{OF two-is-unit}]$  .

**lemma**  $\text{nat-mod-eqE: } (a::nat) \text{ mod } m = b \text{ mod } m \implies \exists i j. a + i * m = b + j * m$   
**proof** –  
**assume**  $a \text{ mod } m = b \text{ mod } m$   
**then have**  $\text{int } a \text{ mod int } m = \text{int } b \text{ mod int } m$  **using**  $\text{zmod-int}$  **by**  $\text{metis}$   
**then obtain**  $l$  **where**  $\text{int } a = \text{int } b + l * \text{int } m$  **by**  $(\text{metis mod-eqE mult.commute})$   
**define**  $i j$  **where**  $i = (\text{if } l \geq 0 \text{ then } 0 \text{ else nat } (-l))$   $j = (\text{if } l \geq 0 \text{ then nat } l \text{ else } 0)$   
**then have**  $\text{int } a + \text{int } i * \text{int } m = \text{int } b + \text{int } j * \text{int } m$   
**using**  $\langle \text{int } a = \text{int } b + l * \text{int } m \rangle$  **by**  $\text{simp}$   
**then have**  $a + i * m = b + j * m$  **by**  $(\text{metis int-ops(7) nat-int-add})$   
**then show**  $?thesis$  **by**  $\text{blast}$   
**qed**

**corollary**  $\text{pow-mod-carrier-length}$ :  
**assumes**  $(a::nat) \text{ mod } 2 ^ (k + 1) = b \text{ mod } 2 ^ (k + 1)$   
**shows**  $2 [\bigwedge]_{Fn} a = 2 [\bigwedge]_{Fn} b$   
**proof** –  
**from**  $\text{assms}$  **obtain**  $i j$  **where**  $0: a + i * 2 ^ (k + 1) = b + j * 2 ^ (k + 1)$

**using** *nat-mod-eqE* **by** *blast*  
**have**  $2 \lceil_{Fn} a = 2 \lceil_{Fn} a \otimes_{Fn} (2 \lceil_{Fn} ((2::nat) \wedge (k + 1))) \lceil_{Fn} i$   
**using** *two-pow-carrier-length-residue-ring two-in-carrier nat-pow-closed*  
**using** *nat-pow-one* **by** *algebra*  
**also have**  $\dots = 2 \lceil_{Fn} (a + i * 2 \wedge (k + 1))$   
**using** *nat-pow-pow nat-pow-mult two-in-carrier*  
**using** *mult commute* **by** *metis*  
**also have**  $\dots = 2 \lceil_{Fn} (b + j * 2 \wedge (k + 1))$   
**using** *0* **by** *argo*  
**also have**  $\dots = 2 \lceil_{Fn} b \otimes_{Fn} (2 \lceil_{Fn} ((2::nat) \wedge (k + 1))) \lceil_{Fn} j$   
**using** *nat-pow-pow nat-pow-mult two-in-carrier*  
**using** *mult commute* **by** *metis*  
**also have**  $\dots = 2 \lceil_{Fn} b$   
**using** *two-pow-carrier-length-residue-ring two-in-carrier nat-pow-closed*  
**using** *nat-pow-one* **by** *algebra*  
**finally show** *?thesis* .

**qed**

**lemma** *two-powers-trivial*:

**assumes**  $s \leq 2 \wedge k$

**shows**  $2 \lceil_{Fn} s = 2 \wedge s$

**proof** –

**from** *assms* **have**  $2 \wedge s \leq \text{int } n - 1$  **by** *simp*

**then have**  $2 \wedge s < \text{int } n$  **using** *n-positive* **by** *linarith*

**then have**  $2 \wedge s = 2 \wedge s \text{ mod } \text{int } n$  **by** *simp*

**also have**  $\dots = 2 \lceil_{Fn} s$  **using** *pow-nat-eq* **by** *simp*

**finally show** *?thesis* **by** *argo*

**qed**

**lemma** *two-powers-Units*:

**assumes**  $s \leq 2 \wedge k$

**shows**  $2 \wedge s \in \text{Units } Fn$

**unfolding** *two-powers-trivial*[*OF assms, symmetric*]

**by** (*intro Units-pow-closed two-is-unit*)

**corollary** *two-powers-in-carrier*:

**assumes**  $s \leq 2 \wedge k$

**shows**  $2 \wedge s \in \text{carrier } Fn$

**using** *assms two-powers-Units Units-closed* **by** *simp*

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

**assumes**  $i + s = k$

**shows**  $(2 \wedge 2 \wedge i) \lceil_{Fn} (2::nat) \wedge s = \ominus_{Fn} \mathbf{1}_{Fn}$

**proof** –

**from** *assms* **have**  $i \leq k$  **by** *simp*

**then have**  $(2 \wedge 2 \wedge i) \lceil_{Fn} (2::nat) \wedge s =$

$(2 \lceil_{Fn} ((2::nat) \wedge i)) \lceil_{Fn} (2::nat) \wedge s$

**using** *two-powers-trivial*[*of 2 \wedge i, symmetric*] **by** *simp*

**also have**  $\dots = 2 \lceil_{Fn} ((2::nat) \wedge (i + s))$

**using** *monoid.nat-pow-pow*[*OF - two-in-carrier*] *cring*

**using** *power-add*[*symmetric, of 2::nat i s*]

**using** *monoid-axioms* **by** *auto*  
**also have**  $\dots = \ominus_{Fn} \mathbf{1}_{Fn}$   
**using**  $\langle i + s = k \rangle$  *two-pow-half-carrier-length-residue-ring* **by** *argo*  
**finally show** *?thesis* .  
**qed**

**interpretation** *z-mod-fermat-unit-group: group units-of Fn*  
**by** (*rule units-group*)

**lemma** *inv-of-2[simp]*:  
 $inv_{Fn} 2 = 2 [\wedge]_{Fn} ((2::nat) \wedge (k + 1) - 1)$   
**proof** –  
**have**  $\mathbf{1}_{Fn} = 2 \otimes_{Fn} 2 [\wedge]_{Fn} ((2::nat) \wedge (k + 1) - 1)$   
**by** (*metis two-is-unit two-pow-carrier-length-residue-ring Units-closed Units-r-inv inv-root-of-unity root-of-unityI zero-less-numeral zero-less-power*)  
**moreover have**  $\mathbf{1}_{Fn} = 2 [\wedge]_{Fn} ((2::nat) \wedge (k + 1) - 1) \otimes_{Fn} 2$   
**by** (*metis two-is-unit two-pow-carrier-length-residue-ring Units-closed Units-l-inv inv-root-of-unity root-of-unityI zero-less-numeral zero-less-power*)  
**ultimately show**  $inv_{Fn} 2 = 2 [\wedge]_{Fn} ((2::nat) \wedge (k + 1) - 1)$   
**using** *less-2-cases-iff two-pow-carrier-length-residue-ring two-in-carrier inv-root-of-unity root-of-unityI* **by** *presburger*  
**qed**

**lemma** *inv-of-2-powers*:  
**assumes**  $s \leq 2 \wedge k$   
**shows**  $inv_{Fn} (2 \wedge s) = 2 [\wedge]_{Fn} (2 \wedge (k + 1) - s)$   
**proof** (*cases s = 0*)  
**case** *True*  
**then show** *?thesis*  
**using** *inv-one res-one-eq*  
**using** *two-pow-carrier-length-residue-ring*  
**by** *simp*  
**next**  
**case** *False*  
**then have**  $s > 0$  **by** *simp*  
**interpret**  $m : multiplicative-subgroup Fn Units Fn units-of Fn$   
**apply** *unfold-locales*  
**subgoal by** *simp*  
**subgoal by** (*simp add: units-of-def*)  
**done**  
**have**  $inv_{Fn} (2 \wedge s) = inv_{Fn} (2 [\wedge]_{Fn} s)$   
**using** *two-powers-trivial[OF  $\langle s \leq 2 \wedge k \rangle$ ]* **by** *argo*  
**also have**  $\dots = (inv_{Fn} 2) [\wedge]_{Fn} s$   
**using** *two-is-unit group.nat-pow-inv[OF m.M-group] m.inv-eq m.M-group m.carrier-M*  
**using** *m.nat-pow-eq Units-pow-closed* **by** *algebra*  
**also have**  $\dots = (2 [\wedge]_{Fn} ((2::nat) \wedge (k + 1) - 1)) [\wedge]_{Fn} s$   
**using** *inv-of-2*  
**by** *argo*

**also have** ... =  $2 \llbracket \_ \rrbracket_{Fn} (((2::nat) \wedge (k + 1) - 1) * s)$   
**using** *two-in-carrier nat-pow-pow* **by** *presburger*  
**also have**  $((2::nat) \wedge (k + 1) - 1) * s = (2::nat) \wedge (k + 1) * s - s$   
**using** *diff-mult-distrib* **by** *simp*  
**also have** ... =  $2 \wedge (k + 1) * (s - 1) + 2 \wedge (k + 1) - s$   
**using**  $\langle s > 0 \rangle$  **by** (*metis add commute mult commute mult-eq-if zero-less-iff-neq-zero*)  
**also have** ... =  $2 \wedge (k + 1) * (s - 1) + (2 \wedge (k + 1) - s)$   
**apply** (*intro diff-add-assoc*) **using** *assms* **by** *simp*  
**also have**  $2 \llbracket \_ \rrbracket_{Fn} (2 \wedge (k + 1) * (s - 1) + (2 \wedge (k + 1) - s)) =$   
 $2 \llbracket \_ \rrbracket_{Fn} (2 \wedge (k + 1) - s)$   
**apply** (*intro pow-mod-carrier-length*) **by** *simp*  
**finally show** *?thesis* .  
**qed**

**lemma** *inv-pow-mod-carrier-length*:  
**assumes**  $(a::nat) \bmod 2 \wedge (k + 1) = b \bmod 2 \wedge (k + 1)$   
**shows**  $(inv_{Fn} 2) \llbracket \_ \rrbracket_{Fn} a = (inv_{Fn} 2) \llbracket \_ \rrbracket_{Fn} b$   
**unfolding** *inv-of-2 nat-pow-pow[OF two-in-carrier]*  
**apply** (*intro pow-mod-carrier-length*)  
**using** *assms mod-mult-cong* **by** *blast*

**lemma**  
**assumes**  $m > 0$   
**shows**  $\exists i j. (a::nat) = j + i * m \wedge j < m$   
**using** *mod-div-mult-eq[of a m, symmetric] pos-mod-bound[of m a] assms mod-less-divisor*  
**by** *blast*

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

**proof** –  
**define** *i* **where**  $i = a \bmod 2 \wedge (k + 1)$   
**define** *j* **where**  $j = a \bmod 2 \wedge (k + 1)$   
**have**  $a = i + j * 2 \wedge (k + 1)$  **using** *mod-div-mult-eq[of a 2 \wedge (k + 1)] i-def j-def*  
**by** *simp*  
**hence**  $(2::nat) \wedge a \bmod n = 2 \wedge i * (2 \wedge (2 \wedge (k + 1))) \wedge j \bmod n$   
**using** *power-add[of 2::nat i j \* 2 \wedge (k + 1)]*  
**using** *power-mult[of 2::nat 2 \wedge (k + 1) j]*  
**using** *mult commute[of j 2 \wedge (k + 1)]*  
**by** *argo*  
**also have** ... =  $2 \wedge i * ((2 \wedge (2 \wedge (k + 1))) \wedge j \bmod n) \bmod n$   
**using** *mod-mult-right-eq* **by** *metis*  
**also have** ... =  $2 \wedge i * ((2 \wedge (2 \wedge (k + 1)) \bmod n) \wedge j \bmod n) \bmod n$   
**using** *power-mod* **by** *metis*  
**also have** ... =  $2 \wedge i * ((1::nat) \wedge j \bmod n) \bmod n$   
**using** *two-pow-carrier-length* **by** *simp*  
**also have** ... =  $2 \wedge i \bmod n$  **by** *simp*  
**finally show** *?thesis* **using** *i-def* **by** *simp*  
**qed**

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

**unfolding** *fermat-non-unique-carrier-def* **by** *simp*

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

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

**using** *zmod-int[of - n, symmetric]*

**by** (*simp add: res-carrier-eq*)

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

**using** *zmod-int*

**by** (*metis to-residue-ring.simps mod-less*)

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

**unfolding** *multiply-with-power-of-2-def* **by** *simp*

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

**unfolding** *divide-by-power-of-2-def* **by** *simp*

**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** (*induction xs*)

**case** *Nil*

**then show** *?case* **by** *simp*

**next**

**case** (*Cons a xs*)

**have**  $(\sum i \leftarrow (a \ \# \ xs). f \ i) \ \bmod \ m = (f \ a + (\sum i \leftarrow xs. f \ i)) \ \bmod \ m$

**by** *simp*

**also have**  $\dots = (f \ a \ \bmod \ m + (\sum i \leftarrow xs. f \ i) \ \bmod \ m) \ \bmod \ m$

**using** *mod-add-eq[symmetric, of f a]* **by** *simp*

**also have**  $\dots = (f \ a \ \bmod \ m + (\sum i \leftarrow xs. f \ i \ \bmod \ m) \ \bmod \ m) \ \bmod \ m$

**using** *Cons.IH* **by** *argo*

**also have**  $\dots = (f \ a \ \bmod \ m + (\sum i \leftarrow xs. f \ i \ \bmod \ m)) \ \bmod \ m$

**using** *mod-add-right-eq* **by** *blast*

**also have**  $\dots = (\sum i \leftarrow (a \# xs). f i \text{ mod } m) \text{ mod } m$   
**by** *simp*  
**finally show** *?case by argo*  
**qed**

**lemma** (*in euclidean-semiring-cancel*) *sum-list-mod'*:  
**assumes**  $\bigwedge i. i \in \text{set } xs \implies f i \text{ mod } m = g i \text{ mod } m$   
**shows**  $(\sum i \leftarrow xs. f i) \text{ mod } m = (\sum i \leftarrow xs. g i) \text{ mod } m$   
**proof** –  
**have**  $(\sum i \leftarrow xs. f i) \text{ mod } m = (\sum i \leftarrow xs. f i \text{ mod } m) \text{ mod } m$   
**by** (*intro sum-list-mod[symmetric]*)  
**also have**  $\dots = (\sum i \leftarrow xs. g i \text{ mod } m) \text{ mod } m$   
**apply** (*intro-cong [cong-tag-1 ( $\lambda i. i \text{ mod } m$ )]*)  
**apply** (*intro-cong [cong-tag-1 sum-list] more: map-cong refl*)  
**using** *assms by assumption*  
**also have**  $\dots = (\sum i \leftarrow xs. g i) \text{ mod } m$   
**by** (*intro sum-list-mod*)  
**finally show** *?thesis .*  
**qed**

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

**proof** (*intro conjI*)  
**assume**  $xs \in \text{fermat-non-unique-carrier}$   
**then have** *length-xs: length xs = 2<sup>(k + 1)</sup>* **by** *simp*  
**then have** *length xs > 0* **by** *simp*

**let**  $?m = \text{length } xs - m \text{ mod } \text{length } xs$

**define** *ys zs where*  $ys = \text{take } ?m \ xs \ \text{and} \ zs = \text{drop } ?m \ xs$   
**then have**  $xs = ys @ zs$   
**and** *length-ys: length ys = ?m*  
**and** *length-zs: length zs = m mod length xs*  
**using**  $\langle \text{length } xs = 2^{k+1} \rangle$  **by** *simp-all*

**have** *1: Nat-LSBF.to-nat xs = Nat-LSBF.to-nat ys + 2<sup>?m</sup> \* Nat-LSBF.to-nat*  
 $zs$  **(is - = ?y + - \* ?z)**  
**apply** (*unfold  $\langle xs = ys @ zs \rangle$  to-nat-app*)  
**apply** (*unfold  $\langle xs = ys @ zs \rangle$  [symmetric] length-ys*)  
**apply** (*rule refl*)  
**done**

**have** *2: multiply-with-power-of-2 xs m = zs @ ys*

**proof** –  
**have** *multiply-with-power-of-2 xs m = rotate-right (m mod length xs) xs*  
**unfolding** *multiply-with-power-of-2-def*  
**by** (*rule rotate-right-conv-mod*)  
**also have**  $\dots = \text{rotate-right } (\text{length } zs) \ (ys @ zs)$

**using**  $\langle xs = ys @ zs \rangle$  *length-zs* **by** *simp*  
**also have**  $\dots = zs @ ys$   
**by** (*rule rotate-right-append*)  
**finally show** *?thesis* .  
**qed**  
**then have**  $\exists: \text{Nat-LSBF.to-nat } (\text{multiply-with-power-of-2 } xs \ m)$   
 $= ?z + 2^{(m \text{ mod } \text{length } xs)} * ?y$   
**by** (*simp add: to-nat-app length-zs*)  
  
**from**  $1$  **have**  $\text{Nat-LSBF.to-nat } xs * 2^m \text{ mod } n = (?y + 2^{?m} * ?z) * 2^{m \text{ mod } n}$   
**by** *argo*  
**also have**  $\dots = (?y + 2^{?m} * ?z) * (2^{m \text{ mod } n} \text{ mod } n)$   
**by** (*simp add: mod-simps*)  
**also have**  $\dots = (?y + 2^{?m} * ?z) * (2^{(m \text{ mod } \text{length } xs)} \text{ mod } n) \text{ mod } n$   
**using** *length-xs two-powers* **by** *algebra*  
**also have**  $\dots = (?y + 2^{?m} * ?z) * 2^{(m \text{ mod } \text{length } xs)} \text{ mod } n$   
**by** (*simp add: mod-simps*)  
**also have**  $\dots = (?y * 2^{(m \text{ mod } \text{length } xs)} + 2^{(?m + (m \text{ mod } \text{length } xs))} * ?z) \text{ mod } n$   
**by** (*simp add: algebra-simps power-add*)  
**also have**  $\dots = (?y * 2^{(m \text{ mod } \text{length } xs)} + 2^{\text{length } xs} * ?z) \text{ mod } n$   
**by** (*simp add: length-xs*)  
**also have**  $\dots = (?y * 2^{(m \text{ mod } \text{length } xs)} + (2^{\text{length } xs} \text{ mod } n) * ?z \text{ mod } n) \text{ mod } n$   
**by** (*simp add: mod-simps*)  
**also have**  $\dots = (?y * 2^{(m \text{ mod } \text{length } xs)} + 1 * ?z \text{ mod } n) \text{ mod } n$   
**by** (*simp only: length-xs two-pow-carrier-length*)  
**also have**  $\dots = (?z + 2^{(m \text{ mod } \text{length } xs)} * ?y) \text{ mod } n$   
**by** (*simp add: mod-simps algebra-simps*)  
**also have**  $\dots = \text{Nat-LSBF.to-nat } (\text{multiply-with-power-of-2 } xs \ m) \text{ mod } n$   
**using**  $\exists$  **by** *argo*  
**finally show**  $\text{Nat-LSBF.to-nat } (\text{multiply-with-power-of-2 } xs \ m) \text{ mod } n = \text{Nat-LSBF.to-nat } xs * 2^m \text{ mod } n$   
**by** *argo*

**have**  $\text{length } (\text{multiply-with-power-of-2 } xs \ m) = \text{length } xs$   
**using**  $2 \langle xs = ys @ zs \rangle$  **by** *simp*  
**then show**  $\text{multiply-with-power-of-2 } xs \ m \in \text{fermat-non-unique-carrier}$   
**apply** (*intro fermat-non-unique-carrierI*)  
**using** *length-xs* **by** *argo*  
**qed**

**corollary** *multiply-with-power-of-2-closed:*  
**assumes**  $xs \in \text{fermat-non-unique-carrier}$   
**shows**  $\text{multiply-with-power-of-2 } xs \ m \in \text{fermat-non-unique-carrier}$   
**by** (*intro conjunct2[OF multiply-with-power-of-2-correct'] assms*)

**corollary** *multiply-with-power-of-2-correct:*

**assumes**  $xs \in \text{fermat-non-unique-carrier}$   
**shows**  $\text{to-residue-ring (multiply-with-power-of-2 } xs \ m) = \text{to-residue-ring } xs \otimes_{F_n} 2 \ [\frown]_{F_n} m$   
**proof** –  
**have**  $\text{to-residue-ring (multiply-with-power-of-2 } xs \ m)$   
 $= \text{int (Nat-LSBF.to-nat (multiply-with-power-of-2 } xs \ m) \ \text{mod } n)$   
**using**  $\text{zmod-int by simp}$   
**also have**  $\dots = \text{int (Nat-LSBF.to-nat } xs \ * \ 2 \ ^m \ \text{mod } n)$   
**using**  $\text{multiply-with-power-of-2-correct'[OF assms] by simp}$   
**also have**  $\dots = (\text{int (Nat-LSBF.to-nat } xs)) \ * \ (2 \ ^m) \ \text{mod } \text{int } n$   
**using**  $\text{zmod-int by simp}$   
**also have**  $\dots = (\text{int (Nat-LSBF.to-nat } xs) \ \text{mod } \text{int } n) \ * \ ((2 \ ^m) \ \text{mod } \text{int } n) \ \text{mod } \text{int } n$   
**by**  $(\text{simp add: mod-mult-eq})$   
**also have**  $\dots = (\text{to-residue-ring } xs) \ \otimes_{F_n} \ ((2 \ ^m) \ \text{mod } \text{int } n)$   
**using**  $\text{res-mult-eq by simp}$   
**also have**  $(2 \ ^m) \ \text{mod } \text{int } n = 2 \ [\frown]_{F_n} m$   
**using**  $\text{pow-nat-eq by simp}$   
**finally show**  $?thesis$  .  
**qed**

**lemma**

**assumes**  $xs \in \text{fermat-non-unique-carrier}$   
**shows**  $\text{divide-by-power-of-2-correct: to-residue-ring (divide-by-power-of-2 } xs \ m)$   
 $= \text{to-residue-ring } xs \ \otimes_{F_n} (\text{inv}_{F_n} \ 2) \ [\frown]_{F_n} m$   
**and**  $\text{divide-by-power-of-2-closed: divide-by-power-of-2 } xs \ m \in \text{fermat-non-unique-carrier}$   
**unfolding**  $\text{atomize-conj}$   
**proof**  $(\text{intro conjI})$   
**from**  $\text{assms show } c: \text{divide-by-power-of-2 } xs \ m \in \text{fermat-non-unique-carrier}$   
**unfolding**  $\text{fermat-non-unique-carrier-def by simp}$   
**define**  $\text{divxs where divxs} = \text{divide-by-power-of-2 } xs \ m$   
**define**  $\text{mulxs where mulxs} = \text{multiply-with-power-of-2 } xs \ m$   
  
**have**  $\text{multiply-with-power-of-2 } \text{divxs } m = xs$   
**unfolding**  $\text{divxs-def multiply-with-power-of-2-def divide-by-power-of-2-def by simp}$   
**then have**  $\text{to-residue-ring } xs = \text{to-residue-ring (multiply-with-power-of-2 } \text{divxs } m)$   
**by**  $\text{simp}$   
**also have**  $\dots = \text{to-residue-ring } \text{divxs} \ \otimes_{F_n} \ 2 \ [\frown]_{F_n} m$   
**apply**  $(\text{intro multiply-with-power-of-2-correct})$   
**unfolding**  $\text{divxs-def by (rule c)}$   
**finally have**  $\text{to-residue-ring } xs \ \otimes_{F_n} (\text{inv}_{F_n} \ 2) \ [\frown]_{F_n} m = \text{to-residue-ring } \text{divxs}$   
 $\otimes_{F_n} \ 2 \ [\frown]_{F_n} m \ \otimes_{F_n} (\text{inv}_{F_n} \ 2) \ [\frown]_{F_n} m$   
**by**  $\text{simp}$   
**also have**  $\dots = \text{to-residue-ring } \text{divxs} \ \otimes_{F_n} (2 \ [\frown]_{F_n} m \ \otimes_{F_n} (\text{inv}_{F_n} \ 2) \ [\frown]_{F_n} m)$   
**apply**  $(\text{intro m-assoc to-residue-ring-in-carrier nat-pow-closed two-in-carrier})$   
**using**  $\text{two-is-unit by auto}$   
**also have**  $(2 \ [\frown]_{F_n} m \ \otimes_{F_n} (\text{inv}_{F_n} \ 2) \ [\frown]_{F_n} m) = (2 \ \otimes_{F_n} (\text{inv}_{F_n} \ 2)) \ [\frown]_{F_n} m$

**apply** (*intro pow-mult-distrib[symmetric] m-comm two-in-carrier*)  
**using** *two-is-unit by auto*  
**also have**  $\dots = \mathbf{1}_{F_n} [\wedge]_{F_n} m$   
**by** (*intro arg-cong2[where  $f = ([\wedge]_{F_n})$ ] refl Units-r-inv two-is-unit*)  
**also have**  $\dots = \mathbf{1}_{F_n}$  **by** *simp*  
**also have** *to-residue-ring  $\text{divxs} \otimes_{F_n} \mathbf{1}_{F_n} = \text{to-residue-ring divxs}$*   
**by** (*intro r-one to-residue-ring-in-carrier*)  
**finally show** *to-residue-ring  $\text{divxs} = \text{to-residue-ring } xs \otimes_{F_n} \text{inv}_{F_n} 2 [\wedge]_{F_n} m$  by*  
*simp*  
**qed**

**definition** *add-fermat where*

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

**define**  *$zs$  where  $zs = \text{add-nat } xs\ ys$*   
**show** *?thesis*  
**proof** (*cases length  $zs = 2^{\wedge}(k + 1) + 1$* )  
**case** *True*  
**then have**  *$\text{add-fermat } xs\ ys = \text{inc-nat} (\text{butlast } zs)$*   
**using** *zs-def unfolding add-fermat-def by simp*  
**then have** *1:  $\text{Nat-LSBF.to-nat} (\text{add-fermat } xs\ ys) = 1 + \text{Nat-LSBF.to-nat}$*   
*(butlast zs) by (simp add: inc-nat-correct)*  
**from** *True obtain  $zs'$  where  $zs = zs' @ [True]$*   
**using** *add-nat-last-bit-True assms zs-def by fastforce*  
**then have**  *$\text{butlast } zs = zs'$  by simp*  
**then have**  *$\text{Nat-LSBF.to-nat} (\text{add-fermat } xs\ ys) = 1 + \text{Nat-LSBF.to-nat } zs'$*   
**using** *1 by simp*  
**moreover have**  *$\text{Nat-LSBF.to-nat } zs = \text{Nat-LSBF.to-nat } zs' + 2^{\wedge}(2^{\wedge}(k + 1))$*   
**using**  *$\langle zs = zs' @ [True] \rangle$  True by (simp add: to-nat-app)*  
**hence**  *$\text{Nat-LSBF.to-nat } zs \bmod n = (\text{Nat-LSBF.to-nat } zs' + 1) \bmod n$*   
**using** *two-pow-carrier-length by (metis mod-add-right-eq)*  
**ultimately have** *2:  $\text{Nat-LSBF.to-nat} (\text{add-fermat } xs\ ys) \bmod n = (\text{Nat-LSBF.to-nat}$*   
 *$xs + \text{Nat-LSBF.to-nat } ys) \bmod n$*   
**using** *add-nat-correct[of xs ys] zs-def by auto*

**have**  *$\text{length } zs' = 2^{\wedge}(k + 1)$  using True  $\langle zs = zs' @ [True] \rangle$  by simp*

**have**  *$\text{Nat-LSBF.to-nat } zs = \text{Nat-LSBF.to-nat } xs + \text{Nat-LSBF.to-nat } ys$  using*  
*zs-def by (simp add: add-nat-correct)*

**also have**  *$\dots \leq (2^{\wedge} \text{length } xs - 1) + (2^{\wedge} \text{length } ys - 1)$*   
**using** *to-nat-length-upper-bound add-le-mono by algebra*

**also have**  $\dots = (2^{\wedge}(2^{\wedge}(k+1)) - 1) + (2^{\wedge}(2^{\wedge}(k+1)) - 1)$   
**using** *assms* **by** *simp*  
**also have**  $\dots < (2^{\wedge}(2^{\wedge}(k+1)) - 1) + (2^{\wedge}(2^{\wedge}(k+1)))$   
**by** (*meson add-strict-left-mono diff-less pos2 zero-less-one zero-less-power*)  
**finally have**  $\text{Nat-LSBF.to-nat } zs' < 2^{\wedge}(2^{\wedge}(k+1)) - 1$   
**using**  $\langle \text{Nat-LSBF.to-nat } zs = \text{Nat-LSBF.to-nat } zs' + 2^{\wedge}(2^{\wedge}(k+1)) \rangle$  **by**  
*simp*  
**then have**  $\text{length } (\text{inc-nat } zs') = \text{length } zs'$   
**using** *length-inc-nat'*  $\langle \text{length } zs' = 2^{\wedge}(k+1) \rangle$  **by** *simp*  
**then have**  $\text{length } (\text{add-fermat } xs \ ys) = 2^{\wedge}(k+1)$   
**using**  $\langle \text{add-fermat } xs \ ys = \text{inc-nat } (\text{butlast } zs) \rangle$   $\langle \text{butlast } zs = zs' \rangle$   $\langle \text{length } zs' = 2^{\wedge}(k+1) \rangle$   
**by** *simp*  
**with** 2 **show** *?thesis* **unfolding** *fermat-non-unique-carrier-def* **by** *simp*  
**next**  
**case** *False*  
**have**  $\text{length } zs \geq 2^{\wedge}(k+1)$   
**using** *assms* *zs-def* *length-add-nat-lower*[*of xs ys*] **by** *simp*  
**moreover have**  $\text{length } zs \leq 2^{\wedge}(k+1) + 1$   
**using** *assms* *zs-def* *length-add-nat-upper*[*of xs ys*] **by** *simp*  
**ultimately have**  $\text{length } zs = 2^{\wedge}(k+1)$  **using** *False* **by** *simp*  
**then have**  $\text{add-fermat } xs \ ys \in \text{fermat-non-unique-carrier}$   
**unfolding** *fermat-non-unique-carrier-def* *add-fermat-def*  
**by** (*simp add: Let-def zs-def*)  
**moreover have**  $\text{Nat-LSBF.to-nat } zs = \text{Nat-LSBF.to-nat } xs + \text{Nat-LSBF.to-nat } ys$   
**by** (*simp add: zs-def add-nat-correct*)  
**moreover have**  $\text{add-fermat } xs \ ys = zs$   
**unfolding** *add-fermat-def* **using** *False* *zs-def* **by** *simp*  
**ultimately show** *?thesis* **by** *algebra*  
**qed**  
**qed**

**corollary** *add-fermat-closed*:  
**assumes**  $xs \in \text{fermat-non-unique-carrier}$   
**assumes**  $ys \in \text{fermat-non-unique-carrier}$   
**shows**  $\text{add-fermat } xs \ ys \in \text{fermat-non-unique-carrier}$   
**by** (*intro conjunct1*[*OF add-fermat-correct*] *assms*)

**corollary** *add-fermat-correct*:  
**assumes**  $xs \in \text{fermat-non-unique-carrier}$   
**assumes**  $ys \in \text{fermat-non-unique-carrier}$   
**shows**  $\text{to-residue-ring } (\text{add-fermat } xs \ ys) = \text{to-residue-ring } xs \oplus_{\mathbb{F}_n} \text{to-residue-ring } ys$   
**proof** –  
**have**  $\text{to-residue-ring } (\text{add-fermat } xs \ ys) = (\text{int } (\text{Nat-LSBF.to-nat } xs) + \text{int } (\text{Nat-LSBF.to-nat } ys)) \bmod \text{int } n$   
**using** *add-fermat-correct*'[*OF assms*]  
**by** (*metis of-nat-add of-nat-mod to-residue-ring.simps*)

**also have** ... = (int (Nat-LSBF.to-nat xs) mod int n + int (Nat-LSBF.to-nat ys) mod int n) mod int n  
**using** mod-add-eq **by** presburger  
**also have** ... = (int (Nat-LSBF.to-nat xs mod n) + int (Nat-LSBF.to-nat ys mod n)) mod int n  
**using** zmod-int **by** simp  
**also have** ... = to-residue-ring xs  $\oplus_{F_n}$  to-residue-ring ys  
**by** (simp add: res-add-eq zmod-int)  
**finally show** ?thesis .  
**qed**

**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  $\in$  fermat-non-unique-carrier  
**assumes** ys  $\in$  fermat-non-unique-carrier  
**shows** subtract-fermat xs ys  $\in$  fermat-non-unique-carrier  $\wedge$  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** -  
**from** assms(2) **have** multiply-with-power-of-2 ys (2 ^ k)  $\in$  fermat-non-unique-carrier  
**unfolding** fermat-non-unique-carrier-def multiply-with-power-of-2-def rotate-right-def **by** simp  
**with** assms(1) **have** 1: subtract-fermat xs ys  $\in$  fermat-non-unique-carrier  
**unfolding** subtract-fermat-def **using** add-fermat-correct' **by** simp  
**have** int (Nat-LSBF.to-nat (subtract-fermat xs ys)) mod n = int (Nat-LSBF.to-nat (subtract-fermat xs ys) mod n)  
**using** zmod-int **by** presburger  
**also have** ... = int ((Nat-LSBF.to-nat xs + Nat-LSBF.to-nat (multiply-with-power-of-2 ys (2 ^ k))) mod n)  
**using** add-fermat-correct'  
**using**  $\langle$ multiply-with-power-of-2 ys (2 ^ k)  $\in$  fermat-non-unique-carrier $\rangle$   
**using** assms(1) subtract-fermat-def **by** presburger  
**also have** ... = int ((Nat-LSBF.to-nat xs + Nat-LSBF.to-nat (multiply-with-power-of-2 ys (2 ^ k)) mod n) mod n)  
**by** presburger  
**also have** ... = int ((Nat-LSBF.to-nat xs + (Nat-LSBF.to-nat ys \* 2 ^ (2 ^ k)) mod n) mod n)  
**using** multiply-with-power-of-2-correct' assms(2) **by** presburger  
**also have** ... = (int (Nat-LSBF.to-nat xs) + int (Nat-LSBF.to-nat ys) \* (int (2 ^ (2 ^ k)) mod n)) mod n  
**using** zmod-int int-ops(7) int-plus  
**by** (simp add: mod-add-right-eq mod-mult-right-eq)  
**also have** ... = (int (Nat-LSBF.to-nat xs) + int (Nat-LSBF.to-nat ys) \* ((-1) mod n)) mod n  
**using** two-pow-half-carrier-length **by** simp  
**also have** ... = (int (Nat-LSBF.to-nat xs) - int (Nat-LSBF.to-nat ys)) mod n  
**by** (simp add: mod-add-cong mod-mult-right-eq)

finally show *?thesis* using 1 by blast  
qed

**corollary** *subtract-fermat-closed*:

assumes  $xs \in \text{fermat-non-unique-carrier}$   
 assumes  $ys \in \text{fermat-non-unique-carrier}$   
 shows *subtract-fermat*  $xs\ ys \in \text{fermat-non-unique-carrier}$   
 by (intro conjunct1[OF *subtract-fermat-correct'*] *assms*)

**corollary** *subtract-fermat-correct*:

assumes  $xs \in \text{fermat-non-unique-carrier}$   
 assumes  $ys \in \text{fermat-non-unique-carrier}$   
 shows *to-residue-ring* (*subtract-fermat*  $xs\ ys$ ) = *to-residue-ring*  $xs \ominus_{F_n}$  *to-residue-ring*  $ys$   
 proof –

have *to-residue-ring* (*subtract-fermat*  $xs\ ys$ ) = (*int* (*Nat-LSBF.to-nat*  $xs$ ) – *int* (*Nat-LSBF.to-nat*  $ys$ )) mod *int*  $n$

using *zmod-int subtract-fermat-correct' assms* by *simp*

also have ... = (*int* (*Nat-LSBF.to-nat*  $xs$ ) mod *int*  $n$  – *int* (*Nat-LSBF.to-nat*  $ys$ ) mod *int*  $n$ ) mod *int*  $n$

using *mod-diff-eq* by *metis*

also have ... = (*int* (*Nat-LSBF.to-nat*  $xs$  mod  $n$ ) – *int* (*Nat-LSBF.to-nat*  $ys$  mod  $n$ )) mod *int*  $n$

using *zmod-int* by *simp*

also have ... = *to-residue-ring*  $xs \ominus_{F_n}$  *to-residue-ring*  $ys$

using *residues-minus-eq* by (*simp add: zmod-int*)

finally show *?thesis* .

qed

end

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

**definition** *reduce* :: *nat-lsbf*  $\Rightarrow$  *nat-lsbf* **where**

*reduce*  $xs = (\text{let } (ys, zs) = \text{split } xs \text{ in}$

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  $xs \in \text{fermat-non-unique-carrier}$

shows *Nat-LSBF.to-nat* (*reduce*  $xs$ ) <  $n \wedge$  *Nat-LSBF.to-nat* (*reduce*  $xs$ ) mod  $n$  = *Nat-LSBF.to-nat*  $xs$  mod  $n$  **and** *length* (*reduce*  $xs$ )  $\leq 2^k + 2$

**proof** –

**obtain**  $ys\ zs$  **where** *split*  $xs = (ys, zs)$  **by** *fastforce*

**then** have *length*  $ys = 2^k$  *length*  $zs = 2^k$  **using** *assms* **by** (*auto simp: split-def Let-def*)

**then** have *Nat-LSBF.to-nat*  $ys < n$  *Nat-LSBF.to-nat*  $zs < n$

**using** *to-nat-length-upper-bound*  
**by** (*metis add.commute add-strict-increasing le-Suc-ex nat-le-linear nat-zero-less-power-iff not-add-less1 power-0 to-nat-bound-to-length-bound*)  
**have**  $(\text{int } (\text{Nat-LSBF.to-nat } ys) - \text{int } (\text{Nat-LSBF.to-nat } zs)) \bmod n = (\text{int } (\text{Nat-LSBF.to-nat } ys) + (-1) \bmod n * \text{int } (\text{Nat-LSBF.to-nat } zs)) \bmod n$   
**by** (*metis diff-minus-eq-add left-minus-one-mult-self mod-add-right-eq mod-mult-left-eq mult-minus1 power-one-right*)  
**also have**  $\dots = (\text{int } (\text{Nat-LSBF.to-nat } ys) + 2^{(2^k)} \bmod n * \text{int } (\text{Nat-LSBF.to-nat } zs)) \bmod n$   
**using** *two-pow-half-carrier-length* **by** *simp*  
**also have**  $\dots = (\text{int } (\text{Nat-LSBF.to-nat } ys + 2^{(2^k)} * \text{Nat-LSBF.to-nat } zs)) \bmod n$   
**by** *auto*  
**also have**  $\dots = (\text{int } (\text{Nat-LSBF.to-nat } (ys @ zs))) \bmod n$   
**using**  $\langle \text{length } ys = 2^k \rangle$  *to-nat-app* **by** *presburger*  
**also have**  $\dots = (\text{int } (\text{Nat-LSBF.to-nat } xs)) \bmod n$   
**using**  $\langle \text{split } xs = (ys, zs) \rangle$  *app-split* **by** *presburger*  
**finally have**  $0: (\text{int } (\text{Nat-LSBF.to-nat } ys) - \text{int } (\text{Nat-LSBF.to-nat } zs)) \bmod n = (\text{int } (\text{Nat-LSBF.to-nat } xs)) \bmod n$  .  
**have**  $\text{Nat-LSBF.to-nat } (\text{reduce } xs) < n \wedge \text{Nat-LSBF.to-nat } (\text{reduce } xs) \bmod n = \text{Nat-LSBF.to-nat } xs \bmod n \wedge \text{length } (\text{reduce } xs) \leq 2^k + 2$   
**proof** (*cases compare-nat zs ys*)  
**case** *True*  
**then have**  $\text{reduce } xs = \text{subtract-nat } ys \ zs$   
**unfolding** *reduce-def*  $\langle \text{split } xs = (ys, zs) \rangle$  **by** *simp*  
**then have**  $1: \text{Nat-LSBF.to-nat } (\text{reduce } xs) = \text{Nat-LSBF.to-nat } ys - \text{Nat-LSBF.to-nat } zs$   
**using** *subtract-nat-correct* **by** *presburger*  
**from** *True* **have**  $\text{Nat-LSBF.to-nat } zs \leq \text{Nat-LSBF.to-nat } ys$   
**using** *compare-nat-correct* **by** *blast*  
**with**  $1$  **have**  $\text{int } (\text{Nat-LSBF.to-nat } (\text{reduce } xs)) = \text{int } (\text{Nat-LSBF.to-nat } ys) - \text{int } (\text{Nat-LSBF.to-nat } zs)$   
**by** *linarith*  
**then have**  $\text{int } (\text{Nat-LSBF.to-nat } (\text{reduce } xs)) \bmod n = (\text{int } (\text{Nat-LSBF.to-nat } xs)) \bmod n$   
**using**  $0$  **by** *presburger*  
**then have**  $\text{Nat-LSBF.to-nat } (\text{reduce } xs) \bmod n = \text{Nat-LSBF.to-nat } xs \bmod n$   
**using** *zmod-int* **by** (*metis of-nat-eq-iff*)  
  
**have**  $\text{Nat-LSBF.to-nat } (\text{reduce } xs) \leq \text{Nat-LSBF.to-nat } ys$  **using**  $1$  **by** *linarith*  
**also have**  $\dots < n$  **using**  $\langle \text{Nat-LSBF.to-nat } ys < n \rangle$  .  
**finally have**  $\text{Nat-LSBF.to-nat } (\text{reduce } xs) < n \wedge \text{Nat-LSBF.to-nat } (\text{reduce } xs) \bmod n = \text{Nat-LSBF.to-nat } xs \bmod n$   
**using**  $\langle \text{Nat-LSBF.to-nat } (\text{reduce } xs) \bmod n = \text{Nat-LSBF.to-nat } xs \bmod n \rangle$  **by** *blast*  
**moreover have**  $\text{length } (\text{reduce } xs) \leq 2^k + 2$  **unfolding**  $\langle \text{reduce } xs = \text{subtract-nat } ys \ zs \rangle$   
**apply** (*estimation estimate: conjunct2[OF subtract-nat-ax]*)  
**using**  $\langle \text{length } zs = 2^k \rangle$   $\langle \text{length } ys = 2^k \rangle$  **by** *simp*

```

ultimately show ?thesis by simp
next
case False
then have reduce-eq: reduce xs = subtract-nat (add-nat (True # replicate (2 ^
k - 1) False @ [True]) ys) zs
  unfolding reduce-def ⟨split xs = (ys, zs)⟩ by simp
then have Nat-LSBF.to-nat (reduce xs) = 1 + 2 * (2 ^ (2 ^ k - 1)) +
Nat-LSBF.to-nat ys - Nat-LSBF.to-nat zs
  by (simp add: subtract-nat-correct add-nat-correct to-nat-app)
also have (1::nat) + 2 * (2 ^ (2 ^ k - 1)) = 1 + 2 ^ (2 ^ k - 1 + 1)
  by (metis add.commute power-add power-one-right)
also have ... = n
  by simp
finally have 1: Nat-LSBF.to-nat (reduce xs) = n + Nat-LSBF.to-nat ys -
Nat-LSBF.to-nat zs .
then have Nat-LSBF.to-nat (reduce xs) < n
  using False ⟨Nat-LSBF.to-nat ys < n⟩ ⟨Nat-LSBF.to-nat zs < n⟩ unfolding
compare-nat-correct
  by linarith
from 1 have int (Nat-LSBF.to-nat (reduce xs)) = int n + int (Nat-LSBF.to-nat
ys) - int (Nat-LSBF.to-nat zs)
  using ⟨Nat-LSBF.to-nat zs < n⟩ by linarith
also have ... mod n = ((int n) mod n + (int (Nat-LSBF.to-nat ys) - int
(Nat-LSBF.to-nat zs))) mod n
  using add-diff-eq
  using mod-add-left-eq[of int n int n int (Nat-LSBF.to-nat ys) - int (Nat-LSBF.to-nat
zs), symmetric]
  by metis
also have ... = (int (Nat-LSBF.to-nat ys) - int (Nat-LSBF.to-nat zs)) mod n
  using mod-self[of int n]
  by simp
finally have int (Nat-LSBF.to-nat (reduce xs)) mod n = int (Nat-LSBF.to-nat
xs) mod n using 0 by presburger
then have Nat-LSBF.to-nat (reduce xs) < n ∧ Nat-LSBF.to-nat (reduce xs)
mod n = Nat-LSBF.to-nat xs mod n
  using ⟨Nat-LSBF.to-nat (reduce xs) < n⟩ zmod-int nat-int-comparison(1) by
presburger
moreover have length (reduce xs) ≤ 2 ^ k + 2
  unfolding reduce-eq
  apply (estimation estimate: conjunct2[OF subtract-nat-ax])
  apply (estimation estimate: length-add-nat-upper)
  unfolding ⟨length ys = 2 ^ k⟩ ⟨length zs = 2 ^ k⟩ by simp
ultimately show ?thesis by simp
qed
then show Nat-LSBF.to-nat (reduce xs) < n ∧ Nat-LSBF.to-nat (reduce xs)
mod n = Nat-LSBF.to-nat xs mod n length (reduce xs) ≤ 2 ^ k + 2
  by simp-all
qed

```

**lemma** *reduce-correct*:  
**assumes**  $xs \in \text{fermat-non-unique-carrier}$   
**shows**  $\text{Nat-LSBF.to-nat } xs \text{ mod } n = \text{Nat-LSBF.to-nat } (\text{reduce } xs)$   
**using**  $\text{reduce-correct}'[OF \text{ assms}] \text{ mod-less}$  **by** *metis*

**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**  $\text{length } xs' \leq e \vee (xs' = \text{replicate } e \text{ False} @ [\text{True}] \wedge xs = \text{replicate } e \text{ True} @ [\text{True}])$   
**proof** (*intro* *verit-and-neg*(3))  
**assume**  $a: \neg (\text{length } xs' \leq e)$   
**then have**  $\text{length } xs' \geq e + 1$  **by** *simp*  
**moreover have**  $\text{length } xs' \leq e + 1$   
**unfolding** *assms*(1)  
**apply** (*estimation estimate: length-add-nat-upper*)  
**using** *assms* **by** *simp*  
**ultimately have**  $\text{len-}xs': \text{length } xs' = e + 1$  **by** *simp*  
**moreover have**  $\max (\text{length } (\text{take } e \text{ } xs)) (\text{length } (\text{drop } e \text{ } xs)) = e$   
**using** *assms* **by** *simp*  
**ultimately have**  $\exists zs. xs' = zs @ [\text{True}]$   
**unfolding** *assms*(1) **by** (*intro* *add-nat-last-bit-True*, *argo*)  
**then obtain**  $zs$  **where**  $zs\text{-def}: xs' = zs @ [\text{True}]$  **and**  $\text{len-}zs: \text{length } zs = e$  **using** *len-}xs'* **by** *auto*

**have**  $\text{Nat-LSBF.to-nat } xs' = \text{Nat-LSBF.to-nat } xs \text{ mod } 2^e + \text{Nat-LSBF.to-nat } xs \text{ div } 2^e$   
**unfolding** *assms*(1) **by** (*simp* *add: add-nat-correct to-nat-take to-nat-drop*)  
**also have**  $\dots < (2^e - 1) + (2^e (e + 1)) \text{ div } 2^e$   
**apply** (*intro* *add-le-less-mono*)  
**subgoal using**  $\text{pos-mod-bound}[\text{of } 2^e \text{ Nat-LSBF.to-nat } xs] \text{ two-pow-pos}$   
**by** (*metis* *Suc-mask-eq-exp mask-eq-exp-minus-1 mod-Suc-le-divisor*)  
**subgoal using**  $\text{to-nat-length-upper-bound}[\text{of } xs] \text{ assms div-le-mono}$   
**by** (*metis* *add-diff-cancel-left' le-add1 less-mult-imp-div-less power-add power-commutes power-diff power-one-right to-nat-length-bound zero-neq-numeral*)  
**done**  
**also have**  $\dots = 2^e + 1$  **by** *simp*  
**finally have**  $\text{Nat-LSBF.to-nat } xs' \leq 2^e$  **by** *simp*  
**moreover have**  $\text{Nat-LSBF.to-nat } xs' = \text{Nat-LSBF.to-nat } zs + 2^e$   
**unfolding** *zs-def* **by** (*simp* *add: to-nat-app len-zs*)  
**ultimately have**  $\text{Nat-LSBF.to-nat } zs = 0$  **by** *simp*  
**then have**  $zs = \text{replicate } e \text{ False}$   $\text{Nat-LSBF.to-nat } xs' = 2^e$   
**using**  $\text{len-zs to-nat-zero-iff truncate-Nil-iff } \langle \text{Nat-LSBF.to-nat } xs' = \text{Nat-LSBF.to-nat } zs + 2^e \rangle$   
**by** *auto*  
**then have**  $xs' = \text{replicate } e \text{ False} @ [\text{True}]$  **using** *zs-def* **by** *simp*  
**from** *assms*(2) **obtain**  $xst \text{ } xsh$  **where**  $xs\text{-decomp}: xs = xst @ [xsh]$   $\text{length } xst = e$

by (*metis Suc-eq-plus1 length-Suc-conv-rev*)  
 then have  $\text{take } e \text{ } xs = xst \text{ drop } e \text{ } xs = [xsh]$  using *assms* by *simp-all*  
 moreover have<sub>[simp]</sub>:  $xsh = \text{True}$   
 proof (*rule ccontr*)  
   assume  $xsh \neq \text{True}$   
   then have  $\text{drop } e \text{ } xs = [\text{False}]$  using *xs-decomp* by *simp*  
   then have  $\text{Nat-LSBF.to-nat } xs' = \text{Nat-LSBF.to-nat } (\text{take } e \text{ } xs)$   
     unfolding *assms(1)* *add-nat-correct* by *simp*  
   also have  $\dots < 2^e$   
     using *assms(2)* *to-nat-length-bound*[of *take e xs*] by *simp*  
   finally show *False* using  $\langle \text{Nat-LSBF.to-nat } xs' = 2^e \rangle$  by *simp*  
 qed  
 ultimately have  $\text{Nat-LSBF.to-nat } xs' = \text{Nat-LSBF.to-nat } xst + 1$  unfolding  
*assms(1)* *add-nat-correct*  
 by *simp*  
 then have  $\text{Nat-LSBF.to-nat } xst = 2^e - 1$  using  $\langle \text{Nat-LSBF.to-nat } xs' = 2^e \rangle$  by *simp*  
 then have  $xst = \text{replicate } e \text{ True}$  using *to-nat-replicate-True2*[of *xst*]  $\langle \text{length } xst = e \rangle$  by *argo*  
 then have  $xs = \text{replicate } e \text{ True} @ [\text{True}]$   
   using  $\langle xs = xst @ [xsh] \rangle$  by *simp*  
 then show  $xs' = \text{replicate } e \text{ False} @ [\text{True}] \wedge xs = \text{replicate } e \text{ True} @ [\text{True}]$   
   using  $\langle xs' = \text{replicate } e \text{ False} @ [\text{True}] \rangle$   
   by (*simp add: replicate-append-same*)  
 qed

**function** *from-nat-lsbf* :: *nat-lsbf*  $\Rightarrow$  *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* (*add-nat* (*take* ( $2^{k+1}$ ) *xs*) (*drop* ( $2^{k+1}$ ) *xs*)))

by *pat-completeness auto*

**lemma** *from-nat-lsbf-dom-termination*: *All from-nat-lsbf-dom*

**proof** (*relation measures [length, Nat-LSBF.to-nat]*)  
 show *wf* (*measures [length, Nat-LSBF.to-nat]*) by *simp*  
 fix  $xs :: \text{nat-lsbf}$   
 define  $e :: \text{nat}$  where  $e = 2^{k+1}$   
 then have *e-ge-1*:  $e \geq 1$  and *e-ge-2*:  $e \geq 2$  by *simp-all*  
 define  $xs'$  where  $xs' = \text{add-nat } (\text{take } e \text{ } xs) (\text{drop } e \text{ } xs)$   
 assume  $\neg \text{length } xs \leq 2^{k+1}$   
 then have  $a: \text{length } xs \geq e + 1$  unfolding *e-def* by *simp*  
 then consider  $\text{length } xs = e + 1 \wedge \text{length } xs' \leq e \mid$   
    $\text{length } xs = e + 1 \wedge \text{length } xs' \geq e + 1 \mid$   
    $\text{length } xs \geq e + 2$   
 by *linarith*  
 then show  $(\text{add-nat } (\text{take } (2^{k+1}) \text{ } xs) (\text{drop } (2^{k+1}) \text{ } xs), xs)$   
    $\in \text{measures [length, Nat-LSBF.to-nat]}$   
   unfolding *e-def*[*symmetric*] *xs'-def*[*symmetric*]  
**proof** *cases*  
   case 1  
   then show  $(xs', xs) \in \text{measures [length, Nat-LSBF.to-nat]}$  by *simp*

```

next
  case 2
  with add-take-drop-carry-aux[OF xs'-def - e-ge-1] have
    xs'-rep: xs' = replicate e False @ [True] and
    xs-rep: xs = replicate e True @ [True]
  by simp-all
  then have Nat-LSBF.to-nat xs' < Nat-LSBF.to-nat xs  $\longleftrightarrow$  (0::nat) < 2 ^ e
- 1
  by (auto simp: to-nat-app)
  also have ... using e-ge-1
  by (metis One-nat-def Suc-le-lessD less-2-cases-iff one-less-power zero-less-diff)
  finally show (xs', xs)  $\in$  measures [length, Nat-LSBF.to-nat]
    using 2 xs'-rep by simp
next
  case 3
  have length xs'  $\leq$  max e (length xs - e) + 1
  unfolding xs'-def
  apply (estimation estimate: length-add-nat-upper)
  by simp
  also have ... < length xs using 3 e-ge-2 by simp
  finally show (xs', xs)  $\in$  measures [length, Nat-LSBF.to-nat] by simp
qed
qed
termination by (rule from-nat-lsbf-dom-termination)

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

lemma from-nat-lsbf-correct:
  shows from-nat-lsbf xs  $\in$  fermat-non-unique-carrier
    to-residue-ring (from-nat-lsbf xs) = to-residue-ring xs
proof (induction xs rule: from-nat-lsbf.induct)
  case (1 xs)
  then show from-nat-lsbf xs  $\in$  fermat-non-unique-carrier
  apply (cases length xs  $\leq$  2 ^ (k + 1))
  subgoal
    unfolding fermat-non-unique-carrier-def
    by (simp add: from-nat-lsbf.simps[of xs] length-fill)
  subgoal
    by (simp add: from-nat-lsbf.simps[of xs])
  done
  show to-residue-ring (from-nat-lsbf xs) = to-residue-ring xs
proof (cases length xs  $\leq$  2 ^ (k + 1))
  case True
  then show ?thesis
  by (simp add: from-nat-lsbf.simps[of xs])
next
  case False
  let ?xs1 = take (2 ^ (k + 1)) xs
  let ?xs2 = drop (2 ^ (k + 1)) xs

```

```

from False have  $xs = ?xs1 @ ?xs2$  by simp
from False have  $from\text{-}nat\text{-}lsbf\ xs = from\text{-}nat\text{-}lsbf\ (add\text{-}nat\ ?xs1\ ?xs2)$ 
  by (simp add: from-nat-lsbf.simps[of xs])
then have  $to\text{-}residue\text{-}ring\ (from\text{-}nat\text{-}lsbf\ xs) = to\text{-}residue\text{-}ring\ (add\text{-}nat\ ?xs1\ ?xs2)$ 
  using 1[OF False] by argo
also have  $\dots = (Nat\text{-}LSBF.to\text{-}nat\ ?xs1 + Nat\text{-}LSBF.to\text{-}nat\ ?xs2) \bmod n$  by
(simp add: add-nat-correct zmod-int)
also have  $\dots = (Nat\text{-}LSBF.to\text{-}nat\ ?xs1 + (2^{(2^{(k+1)})}) * Nat\text{-}LSBF.to\text{-}nat\ ?xs2) \bmod n$ 
  using two-pow-carrier-length mod-add-right-eq mod-mult-left-eq
  by (metis (no-types, opaque-lifting) mult-numeral-1 numerals(1))
also have  $\dots = (Nat\text{-}LSBF.to\text{-}nat\ xs) \bmod n$ 
  by (intro-cong [cong-tag-1 int, cong-tag-2 (mod)] more: refl to-nat-drop-take[symmetric])
finally show ?thesis by (simp add: zmod-int)
qed
qed

```

```

lemma length-from-nat-lsbf:  $length\ (from\text{-}nat\text{-}lsbf\ xs) = 2^{(k+1)}$ 
  using fermat-carrier-length[OF from-nat-lsbf-correct(1)] .

```

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

```

lemma n-odd: odd n
  by simp

```

```

lemma ord-2:  $ord\ n\ 2 = 2^{(k+1)}$ 

```

```

proof -

```

```

  have  $ord\ n\ 2\ dvd\ 2^{(k+1)}$ 
    using ord-divides[of  $2::nat\ 2^{(k+1)}\ n$ ]
    using two-pow-carrier-length
    by (simp add: cong-def)

```

```

  then obtain i where  $ord\ n\ 2 = 2^i$   $i \leq k + 1$ 

```

```

    using divides-primepow-nat[OF two-is-prime-nat]
    by blast

```

```

  have  $i = k + 1$ 

```

```

  proof (rule ccontr)

```

```

    assume  $i \neq k + 1$ 

```

```

    then have  $i \leq k$  using  $\langle i \leq k + 1 \rangle$  by linarith

```

```

    have  $1 \neq (2::nat)^{(2^k)} \bmod n$  using two-pow-half-carrier-length-neq-1[symmetric]

```

```

  moreover have  $(2::nat)^{(2^k)} \bmod n = 1$ 

```

```

  proof -

```

```

    have  $(2::nat)^{(2^k)} \bmod n = (2^{(2^i)})^{(2^{(k-i)})} \bmod n$ 

```

```

      by (simp add:  $\langle i \leq k \rangle$  power-add[symmetric] power-mult[symmetric])

```

```

    also have  $\dots = (2^{(2^i)} \bmod n)^{(2^{(k-i)})} \bmod n$ 

```

```

      by (simp add: power-mod)

```

```

    also have  $2^{(2^i)} \bmod n = 1$  using  $\langle ord\ n\ 2 = 2^i \rangle$ 

```

```

      using ord[of  $2\ n$ ] n-gt-1 unfolding cong-def by simp

```

```

    finally show ?thesis by simp
  qed
  ultimately show False by argo
  qed
  then show ?thesis using ⟨ord n 2 = 2 ^ i⟩ by argo
  qed
  corollary ord-2-int: ord (int n) 2 = 2 ^ (k + 1)
    using ord-2 ord-int[of n 2] by simp

lemma two-is-primitive-root: primitive-root (2 ^ (k + 1)) 2
  apply (intro primitive-rootI)
  subgoal
    using two-in-carrier .
  subgoal
    using two-pow-carrier-length-residue-ring .
  subgoal for i
    using ord-2-int unfolding ord-def
    using pow-nat-eq not-less-Least cong-def
    by (metis (no-types, lifting) less-nat-zero-code one-cong)
  done

lemma two-inv-is-primitive-root: primitive-root (2 ^ (k + 1)) (inv_Fn 2)
  using primitive-root-inv[OF two-is-primitive-root] by simp

lemma two-powers-primitive-root:
  assumes i + s = k + 1
  assumes i ≤ k
  shows primitive-root (2 ^ s) (2 [^]_Fn (2::nat) ^ i)
  proof (intro primitive-rootI nat-pow-closed two-in-carrier)

    have (2 [^]_Fn (2::nat) ^ i) [^]_Fn (2::nat) ^ s = 2 [^]_Fn ((2::nat) ^ (i + s))
      by (simp add: nat-pow-pow[OF two-in-carrier] power-add)
    also have ... = 1_Fn
      unfolding assms(1) by (rule two-pow-carrier-length-residue-ring)
    finally show (2 [^]_Fn (2::nat) ^ i) [^]_Fn (2::nat) ^ s = 1_Fn .

  fix j :: nat
  assume 0 < j j < 2 ^ s
  then have 2 ^ i * j < 2 ^ (k + 1)
    using power-add assms(1)
    by (metis nat-mult-less-cancel1 pos2 zero-less-power)
  have 2 ^ i * j > 0 using ⟨j > 0⟩ by simp
  have 1: (∀ l ∈ {1..<(2::nat) ^ (k + 1)}. 2 [^]_Fn l ≠ 1_Fn)
    using two-is-primitive-root unfolding primitive-root-def by simp
  have (2 [^]_Fn (2::nat) ^ i) [^]_Fn j = 2 [^]_Fn (2 ^ i * j)
    by (simp add: nat-pow-pow[OF two-in-carrier])
  also have ... ≠ 1_Fn
    using 1 ⟨2 ^ i * j > 0⟩ ⟨2 ^ i * j < 2 ^ (k + 1)⟩ by simp
  finally show (2 [^]_Fn (2::nat) ^ i) [^]_Fn j ≠ 1_Fn .

```

qed

**fun** *fft-combine-b-c-aux* :: (nat-lsbf  $\Rightarrow$  nat-lsbf  $\Rightarrow$  nat-lsbf)  $\Rightarrow$  (nat-lsbf  $\Rightarrow$  nat  $\Rightarrow$  nat-lsbf)  $\Rightarrow$  nat  $\Rightarrow$  nat-lsbf list  $\times$  nat  $\Rightarrow$  nat-lsbf list  $\Rightarrow$  nat-lsbf list  $\Rightarrow$  nat-lsbf list

**where**

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

**fun** *fft-iff-combine-b-c-add* **where**

*fft-iff-combine-b-c-add* True l bs cs = *fft-combine-b-c-aux* add-fermat divide-by-power-of-2

l ([], 0) bs cs

| *fft-iff-combine-b-c-add* False l bs cs = *fft-combine-b-c-aux* add-fermat multiply-with-power-of-2

l ([], 0) bs cs

**fun** *fft-iff-combine-b-c-subtract* **where**

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

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

**lemma** *fft-combine-b-c-aux-correct*:

**assumes** length bs = len-bc length cs = len-bc

**assumes** e < 2  $^{\wedge}$  (k + 1)

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

**using** *assms* **proof** (induction len-bc arbitrary: bs cs revs e)

**case** 0

**then have** bs = [] cs = [] **by** *simp-all*

**then show** ?*case* **by** *simp*

**next**

**case** (Suc len-bc)

**then obtain** b bs' c cs' **where** bcs: bs = b # bs' cs = c # cs' **by** (*meson length-Suc-conv*)

**with** *Suc.prem*s **have** len-bcs': length bs' = len-bc length cs' = len-bc **by** *simp-all*

**have** (e + l \* i) mod 2  $^{\wedge}$  (k + 1) < 2  $^{\wedge}$  (k + 1) **for** i **by** *simp*

**note** ih = *Suc.IH*[*OF* len-bcs' *this*]

**have** *fft-combine-b-c-aux* f g l (revs, e) bs cs =

*fft-combine-b-c-aux* f g l (f b (g c e) # revs, (e + l) mod (2 \* 2  $^{\wedge}$  k)) bs' cs'

**unfolding** bcs **by** *simp*

**also have** ... = rev (f b (g c e) # revs) @

    map3 ( $\lambda$ x y i. f x (g y (((e + l \* 1) mod 2  $^{\wedge}$  (k + 1) + l \* i) mod 2  $^{\wedge}$  (k + 1)))) bs' cs'

    [0..*len-bc*]

**using** ih[*of* f b (g c e) # revs 1] **by** *simp*

**also have** ... = rev revs @ (f b (g c e) #

    map3 ( $\lambda$ x y i. f x (g y (((e + l \* 1) mod 2  $^{\wedge}$  (k + 1) + l \* i) mod 2  $^{\wedge}$  (k + 1)))) bs' cs'

```

    [0..<len-bc])
  by simp
  finally have r: fft-combine-b-c-aux f g l (revs, e) bs cs = ... .
  show ?case unfolding r
  proof (intro arg-cong2[where f = (@)] refl)
    have f b (g c e) #
      map3 (λx y i. f x (g y (((e + l * 1) mod 2 ^ (k + 1) + l * i) mod 2 ^ (k +
1)))) bs' cs' [0..<len-bc] =
      f b (g c (e + l * 0)) #
      map3 (λx y i. f x (g y ((e + l * Suc i) mod 2 ^ (k + 1)))) bs' cs' [0..<len-bc]
    (is ?l = ?f # ?m3)
    apply (intro arg-cong2[where f = (#)])
    subgoal by simp
    subgoal
      unfolding append.append-Nil
      apply (intro arg-cong[where f = λi. map3 i - - -])
      by (simp add: add.assoc mod-add-left-eq)
    done
    also have ?m3 = map3 (λx y i. f x (g y ((e + l * i) mod 2 ^ (k + 1)))) bs'
cs' (map Suc [0..<len-bc])
    by (rule map3-compose3)
    also have ... = map3 (λx y i. f x (g y ((e + l * i) mod 2 ^ (k + 1)))) bs' cs'
[0..<Suc len-bc]
    by (subst map-Suc-upt) (rule refl)
    also have ?f # ... = map3 (λx y i. f x (g y ((e + l * i) mod 2 ^ (k + 1))))
bs cs [0..<Suc len-bc]
    unfolding upt-conv-Cons[OF zero-less-Suc[of len-bc]] bcs using Suc.premis
  by simp
  finally show ?l = ... .
  qed
qed

```

**lemma** *fft-iff-fft-combine-b-c-add-correct*:

```

  assumes length bs = len-bc length cs = len-bc
  shows fft-iff-fft-combine-b-c-add it l bs cs = map3 (λx y i. add-fermat x ((if it then
divide-by-power-of-2 else multiply-with-power-of-2) y ((l * i) mod 2 ^ (k + 1))))
bs cs [0..<len-bc]
  by (cases it; simp add: fft-combine-b-c-aux-correct[OF assms])

```

**lemma** *fft-iff-fft-combine-b-c-subtract-correct*:

```

  assumes length bs = len-bc length cs = len-bc
  shows fft-iff-fft-combine-b-c-subtract it l bs cs = map3 (λx y i. subtract-fermat x
((if it then divide-by-power-of-2 else multiply-with-power-of-2) y ((l * i) mod 2 ^
(k + 1)))) bs cs [0..<len-bc]
  by (cases it; simp add: fft-combine-b-c-aux-correct[OF assms])

```

**lemma** *fft-iff-fft-combine-b-c-add-carrier*:

```

  assumes length bs = len-bc length cs = len-bc
  assumes set bs ⊆ fermat-non-unique-carrier

```

```

assumes set cs  $\subseteq$  fermat-non-unique-carrier
shows set (fft-iff-combine-b-c-add it l bs cs)  $\subseteq$  fermat-non-unique-carrier
unfolding fft-iff-combine-b-c-add-correct[OF assms(1) assms(2)]
apply (intro set-map3-subseteqI[OF - assms(3) assms(4) subset-refl] add-fermat-closed)
apply (simp-all add: divide-by-power-of-2-closed multiply-with-power-of-2-closed)
done

```

```

lemma fft-iff-combine-b-c-subtract-carrier:
assumes length bs = len-bc length cs = 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
unfolding fft-iff-combine-b-c-subtract-correct[OF assms(1) assms(2)]
apply (intro set-map3-subseteqI[OF - assms(3) assms(4) subset-refl] subtract-fermat-closed)
apply (simp-all add: divide-by-power-of-2-closed multiply-with-power-of-2-closed)
done

```

```

fun fft-iff :: bool  $\Rightarrow$  nat  $\Rightarrow$  nat-lsbf list  $\Rightarrow$  nat-lsbf 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-lsbf-fermat +
  fixes it :: bool
  fixes l e :: nat
  fixes a1 a2 a3 :: nat-lsbf
  fixes as :: nat-lsbf 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 =  $2^e$  unfolding a-def by (rule length-a')
lemma e-ge-2:  $e \geq 2$ 
proof (rule ccontr)
  assume  $\neg e \geq 2$ 
  then have  $e \leq 1$  by simp
  have  $(2::nat)^e \leq 2$  using power-increasing[OF <e ≤ 1>, of 2::nat] by simp
  then show False using length-a' by simp
qed
lemma e-pos:  $e > 0$  using e-ge-2 by simp
lemma two-pow-e-div-2:  $(2::nat)^e \text{ div } 2 = 2^{e-1}$ 
  using gr0-implies-Suc[OF e-pos] by auto
lemma two-pow-e-as-sum:  $(2::nat)^e = 2^{e-1} + 2^{e-1}$ 
  by (metis e-pos two-pow-e-div-2 even-power even-two-times-div-two gcd-nat.eq-iff mult-2)

lemma
  shows length-nums1: length nums1 =  $2^{e-1}$ 
  and length-nums2: length nums2 =  $2^{e-1}$ 
  unfolding nums1-def nums2-def length-evens-odds length-a
  using two-pow-e-div-2 by simp-all

lemma result-eq: fft-iff it l a = g @ h
  unfolding a-def fft-iff.simps[of it l] Let-def
  unfolding defs[symmetric] by (rule refl)

lemma
  assumes set a  $\subseteq$  fermat-non-unique-carrier
  shows nums1-carrier: set nums1  $\subseteq$  fermat-non-unique-carrier
  and nums2-carrier: set nums2  $\subseteq$  fermat-non-unique-carrier
  unfolding nums1-def nums2-def atomize-conj
  by (intro conjI subset-trans[OF set-evens-odds] assms)

end

context int-lsbf-fermat
begin

lemma length-fft-iff:
  assumes length a =  $2^e$ 
  shows length (fft-iff it l a) =  $2^e$ 
  using assms
proof (induction it l a arbitrary: e rule: fft-iff.induct)
  case (4 it l a1 a2 a3 as)
  interpret fft-context k it l e a1 a2 a3 as
  apply unfold-locales
  using 4 by argo

```

```

have len-b: length b = 2 ^ (e - 1)
  unfolding b-def
  apply (intro 4.IH[of nums1 nums2])
  unfolding defs[symmetric] length-nums1
  by (rule refl)+
have len-c: length c = 2 ^ (e - 1)
  unfolding c-def
  apply (intro 4.IH(2)[of nums1 nums2 b])
  unfolding defs[symmetric] length-nums2
  by (rule refl)+
have len-g: length g = 2 ^ (e - 1)
  unfolding g-def fft-iff-combine-b-c-add-correct[OF len-b len-c] map3-as-map
  by (simp add: len-b len-c)
have len-h: length h = 2 ^ (e - 1)
  unfolding h-def fft-iff-combine-b-c-subtract-correct[OF len-b len-c] map3-as-map
  by (simp add: len-b len-c)
show ?case
  unfolding a-def[symmetric] result-eq
  by (simp add: len-g len-h e-pos two-pow-e-as-sum)
qed simp-all

```

```

lemma length-fft:
  assumes length a = 2 ^ e
  shows length (fft l a) = 2 ^ e
  unfolding fft.simps length-fft-iff[OF assms] by (rule refl)

```

```

lemma length-iff:
  assumes length a = 2 ^ e
  shows length (iff l a) = 2 ^ e
  unfolding iff.simps length-fft-iff[OF assms] by (rule refl)

```

**end**

**context** fft-context **begin**

```

lemma length-b: length b = 2 ^ (e - 1)
  unfolding b-def by (intro length-fft-iff length-nums1)
lemma length-c: length c = 2 ^ (e - 1)
  unfolding c-def by (intro length-fft-iff length-nums2)
lemma length-g: length g = 2 ^ (e - 1)
  unfolding g-def fft-iff-combine-b-c-add-correct[OF length-b length-c] map3-as-map
  by (simp add: length-b length-c)
lemma length-h: length h = 2 ^ (e - 1)
  unfolding h-def fft-iff-combine-b-c-subtract-correct[OF length-b length-c] map3-as-map
  by (simp add: length-b length-c)

```

**end**

```

context int-lsbfermat
begin

lemma fft-iff-carrier:
  assumes length a = 2 ^ l
  assumes set a ⊆ feramat-non-unique-carrier
  shows set (fft-iff it s a) ⊆ feramat-non-unique-carrier
using assms proof (induction it s a arbitrary: l rule: fft-iff.induct)
  case (1 it s)
  then show ?case by simp
next
  case (2 it s x)
  then show ?case by simp
next
  case (3 it s x y)
  then show ?case by (simp add: add-fermat-closed subtract-fermat-closed)
next
  case (4 it s a1 a2 a3 as)
  interpret fft-context k it s l a1 a2 a3 as
  apply unfold-locales using 4 by simp

have b-carrier: set b ⊆ feramat-non-unique-carrier
  unfolding b-def
  apply (intro 4.IH(1)[of nums1 nums2 l - 1])
  unfolding defs[symmetric]
  using nums1-carrier length-nums1 4.prem1 a-def by simp-all
have c-carrier: set c ⊆ feramat-non-unique-carrier
  unfolding c-def
  apply (intro 4.IH(2)[of nums1 nums2 b l - 1])
  unfolding defs[symmetric]
  using nums2-carrier length-nums2 4.prem2 a-def by simp-all

have g-carrier: set g ⊆ feramat-non-unique-carrier
  unfolding g-def
  apply (intro fft-iff-combine-b-c-add-carrier)
  using length-b length-c b-carrier c-carrier by simp-all
have h-carrier: set h ⊆ feramat-non-unique-carrier
  unfolding h-def
  apply (intro fft-iff-combine-b-c-subtract-carrier)
  using length-b length-c b-carrier c-carrier by simp-all

show ?case
  unfolding defs[symmetric] result-eq
  using g-carrier h-carrier by simp
qed

lemma fft-carrier:
  assumes length a = 2 ^ l

```

**assumes**  $set\ a \subseteq fermap-non-unique-carrier$   
**shows**  $set\ (fft\ s\ a) \subseteq fermap-non-unique-carrier$   
**using**  $fft-iffm-carrier[OF\ assms]$  **by**  $simp$

**lemma**  $iffm-carrier$ :

**assumes**  $length\ a = 2^l$   
**assumes**  $set\ a \subseteq fermap-non-unique-carrier$   
**shows**  $set\ (iffm\ s\ a) \subseteq fermap-non-unique-carrier$   
**using**  $fft-iffm-carrier[OF\ assms]$  **by**  $simp$

**lemma**  $fft-iffm-correct'$ :

**assumes**  $length\ a = 2^l$   
**assumes**  $l \leq k + 1$   
**assumes**  $set\ a \subseteq fermap-non-unique-carrier$   
**shows**  $map\ to-residue-ring\ (fft-iffm\ it\ s\ a) = FNTT''\ ((if\ it\ then\ inv_{Fn}\ 2\ else\ 2)\ [\wedge]_{Fn}\ s)\ (map\ to-residue-ring\ a)$   
**using**  $assms$   
**proof** ( $induction\ it\ s\ a\ arbitrary$ :  $l$   $rule$ :  $fft-iffm.induct$ )  
**case** ( $1\ it\ s$ )  
**then** **show**  $?case$  **by**  $simp$   
**next**  
**case** ( $2\ it\ s\ x$ )  
**then** **show**  $?case$  **by**  $simp$   
**next**  
**case** ( $3\ it\ s\ x\ y$ )  
**then** **show**  $?case$  **using**  $add-fermap-correct\ subtract-fermap-correct$  **by**  $simp$   
**next**  
**case** ( $4\ it\ s\ a1\ a2\ a3\ as$ )  
**interpret**  $fft-context\ k\ it\ s\ l\ a1\ a2\ a3\ as$   
**apply**  $unfold-locales$  **using**  $4$  **by**  $simp$

**define**  $nums1'$  **where**  $nums1' = evens-odds\ True\ (map\ to-residue-ring\ local.a)$   
**define**  $nums2'$  **where**  $nums2' = evens-odds\ False\ (map\ to-residue-ring\ local.a)$   
**define**  $b'$  **where**  $b' = FNTT''\ (((if\ it\ then\ inv_{Fn}\ 2\ else\ 2)\ [\wedge]_{Fn}\ s)\ [\wedge]_{Fn}\ (2::nat))\ nums1'$   
**define**  $c'$  **where**  $c' = FNTT''\ (((if\ it\ then\ inv_{Fn}\ 2\ else\ 2)\ [\wedge]_{Fn}\ s)\ [\wedge]_{Fn}\ (2::nat))\ nums2'$   
**define**  $g'$  **where**  $g' = map2\ (\oplus_{Fn})\ b'$   
 $(map2\ (\otimes_{Fn})\ (map\ (([\wedge]_{Fn})\ ((if\ it\ then\ inv_{Fn}\ 2\ else\ 2)\ [\wedge]_{Fn}\ s))\ [0..<length\ local.a\ div\ 2])\ c')$   
**define**  $h'$  **where**  $h' = map2\ (a-minus\ Fn)\ b'$   
 $(map2\ (\otimes_{Fn})\ (map\ (([\wedge]_{Fn})\ ((if\ it\ then\ inv_{Fn}\ 2\ else\ 2)\ [\wedge]_{Fn}\ s))\ [0..<length\ local.a\ div\ 2])\ c')$   
**note**  $defs' = a-def\ nums1'-def\ nums2'-def\ b'-def\ c'-def\ g'-def\ h'-def$

**have**  $fntt-def$ :  $FNTT''\ ((if\ it\ then\ inv_{Fn}\ 2\ else\ 2)\ [\wedge]_{Fn}\ s)\ (map\ to-residue-ring$

```

(a1 # a2 # a3 # as)
  = g' @ h'
  using length-map[of to-residue-ring local.a]
  by (simp only: list.map(2) FNTT''.simps Let-def defs')

from two-is-primitive-root have root-of-unity (2 ^ (k + 1)) 2
  unfolding primitive-root-def by blast

from e-ge-2 have Suc (k + 1 - l) ≤ k using ⟨l ≤ k + 1⟩ by linarith

have pr-unit: (if it then invFn 2 else 2) ∈ Units Fn
  using two-is-unit Units-inv-Units by algebra
then have pr-carrier: (if it then invFn 2 else 2) ∈ carrier Fn
  by (rule Units-closed)
have pow-2s: ((if it then invFn 2 else 2) [∧]Fn s) [∧]Fn (2::nat) = (if it then
invFn 2 else 2) [∧]Fn (2 * s)
  using nat-pow-pow[OF pr-carrier, of s 2] mult.commute[of s 2] by argo

from e-ge-2 obtain l' where l'-def[simp]: l = Suc l'
  by (metis not-numeral-le-zero old.nat.exhaust)

have l'-le: l' ≤ k + 1
  using ⟨l ≤ k + 1⟩ ⟨l = Suc l'⟩ by linarith

have nums1'-def': nums1' = map to-residue-ring nums1
  by (simp add: nums1'-def nums1-def map-evens-odds)
then have length-nums1': length nums1' = 2 ^ l' using length-nums1 ⟨l = Suc
l'⟩ by simp
have nums2'-def': nums2' = map to-residue-ring nums2
  by (simp add: nums2'-def nums2-def map-evens-odds)
then have length-nums2': length nums2' = 2 ^ l' using length-nums2 ⟨l = Suc
l'⟩ by simp

have set local.a ⊆ fermat-non-unique-carrier using 4 by (simp only: a-def)
have nums1-carrier: set nums1 ⊆ fermat-non-unique-carrier
  unfolding nums1-def using ⟨set local.a ⊆ fermat-non-unique-carrier⟩ set-evens-odds
by fastforce

have b-b': b' = map to-residue-ring b
unfolding b'-def b-def nums1'-def map-evens-odds[symmetric] pow-2s nums1-def
  apply (intro 4(1)[symmetric, of - nums2 l'])
  subgoal unfolding a-def by (rule refl)
  subgoal unfolding nums2-def a-def by (rule refl)
  subgoal unfolding defs[symmetric] length-nums1 by simp
  subgoal by (rule l'-le)
  subgoal unfolding defs[symmetric] by (rule nums1-carrier)
  done
then have length-b': length b' = 2 ^ l'

```

```

using length-b by simp

have nums2-carrier: set nums2  $\subseteq$  fermat-non-unique-carrier
  unfolding nums2-def using ⟨set local.a  $\subseteq$  fermat-non-unique-carrier⟩ set-evens-odds
by fastforce

have c-c': c' = map to-residue-ring c
  unfolding c'-def c-def nums2'-def map-evens-odds[symmetric] pow-2s nums2-def
  apply (intro 4(2)[symmetric, of nums1 - b l'])
  subgoal unfolding defs by (rule refl)
  subgoal unfolding a-def by (rule refl)
  subgoal unfolding defs[symmetric] by (rule refl)
  subgoal unfolding defs[symmetric] length-nums2 by simp
  subgoal by (rule l'-le)
  subgoal unfolding defs[symmetric] by (rule nums2-carrier)
done
then have length-c': length c' =  $2^{\wedge} l'$ 
  using length-c by simp

have b-carrier: set b  $\subseteq$  fermat-non-unique-carrier
  unfolding b-def
  apply (intro fft-iff-carrier nums1-carrier) using length-nums1 by simp
have c-carrier: set c  $\subseteq$  fermat-non-unique-carrier
  unfolding c-def
  apply (intro fft-iff-carrier nums2-carrier) using length-nums2 by simp

have length-nums1': length nums1' =  $2^{\wedge} l'$ 
  using length-nums1 nums1'-def' by simp
have length-nums2': length nums2' =  $2^{\wedge} l'$ 
  using length-nums2 nums2'-def' by simp

have length-g': length g' =  $2^{\wedge} l'$ 
  unfolding g'-def by (simp add: length-a length-b' length-c')
have length-h': length h' =  $2^{\wedge} l'$ 
  unfolding h'-def by (simp add: length-a length-b' length-c')

have g-g': g' = map to-residue-ring g
proof (intro nth-equalityI)
  show length g' = length (map to-residue-ring g)
    by (simp add: length-g length-g')
next
fix i
assume i < length g'
then have i-less: i <  $2^{\wedge} (l - 1)$  unfolding length-g' using ⟨l = Suc l'⟩ by
simp

have bi-carrier: b ! i  $\in$  fermat-non-unique-carrier
  using set-subseteqD[OF b-carrier] length-b i-less by simp
have ci-carrier: c ! i  $\in$  fermat-non-unique-carrier

```

```

using set-subseteqD[OF c-carrier] length-c i-less by simp

have bi-b'i: to-residue-ring (b ! i) = b' ! i
  unfolding b-b' by (intro nth-map[symmetric]; simp add: length-b i-less del:
    ‹l = Suc l'› One-nat-def)
  have ci-c'i: to-residue-ring (c ! i) = c' ! i
    unfolding c-c' by (intro nth-map[symmetric]; simp add: length-c i-less del:
      ‹l = Suc l'› One-nat-def)

show g' ! i = (map to-residue-ring g) ! i
proof (cases it)
  case True
  then have it-op: (if it then divide-by-power-of-2 else multiply-with-power-of-2)
    = divide-by-power-of-2 by argo
  have map to-residue-ring g ! i = to-residue-ring (g ! i)
    apply (intro nth-map)
    unfolding length-g using i-less by simp
  also have ... = to-residue-ring (add-fermat (b ! i) (divide-by-power-of-2 (c !
    i) (s * ([0..<2 ^ (l - 1)] ! i) mod 2 ^ (k + 1))))
    unfolding g-def fft-iff-combine-b-c-add-correct[OF length-b length-c] it-op
    apply (intro arg-cong[where f = to-residue-ring] nth-map3)
    unfolding length-b length-c using i-less by simp-all
  also have ... = to-residue-ring (add-fermat (b ! i) (divide-by-power-of-2 (c !
    i) (s * i mod 2 ^ (k + 1))))
    using i-less by simp
  also have ... = to-residue-ring (b ! i) ⊕Fn to-residue-ring (divide-by-power-of-2
    (c ! i) (s * i mod 2 ^ (k + 1)))
    by (intro add-fermat-correct bi-carrier divide-by-power-of-2-closed ci-carrier)
  also have ... = to-residue-ring (b ! i) ⊕Fn to-residue-ring (c ! i) ⊗Fn invFn
    2 [∧]Fn (s * i mod 2 ^ (k + 1))
    by (intro arg-cong2[where f = (⊕Fn)] divide-by-power-of-2-correct refl
      ci-carrier)
  also have ... = (b' ! i) ⊕Fn (c' ! i) ⊗Fn invFn 2 [∧]Fn (s * i mod 2 ^ (k +
    1))
    unfolding bi-b'i ci-c'i by (rule refl)
  also have ... = (b' ! i) ⊕Fn (c' ! i) ⊗Fn invFn 2 [∧]Fn (s * i)
    by (intro-cong [cong-tag-2 (⊕Fn), cong-tag-2 (⊗Fn)] more: inv-pow-mod-carrier-length
      mod-mod-trivial)
  also have ... = (b' ! i) ⊕Fn (c' ! i) ⊗Fn ((invFn 2 [∧]Fn s) [∧]Fn i)
    by (intro-cong [cong-tag-2 (⊕Fn), cong-tag-2 (⊗Fn)] more: nat-pow-pow[symmetric]
      Units-inv-closed two-is-unit)
  finally have 1: map to-residue-ring g ! i = ... .
  have g' ! i = map3 (λx y z. x ⊕Fn y ⊗Fn z) b' (map (([∧]Fn) (invFn 2 [∧]Fn
    s)) [0..<length local.a div 2]) c' ! i
    unfolding g'-def eqTrueI[OF True] if-True map2-of-map2-r by (rule refl)
  also have ... = (b' ! i) ⊕Fn ((map (([∧]Fn) (invFn 2 [∧]Fn s)) [0..<length
    local.a div 2]) ! i) ⊗Fn (c' ! i)
    using i-less length-b' length-c' ‹l = Suc l'› length-a by (intro nth-map3)
  simp-all

```

**also have** ... =  $(b' ! i) \oplus_{F_n} (\text{inv}_{F_n} 2 [\lceil_{F_n} s] [\lceil_{F_n} i \otimes_{F_n} (c' ! i)])$   
**apply** (*intro-cong* [*cong-tag-2* ( $\oplus_{F_n}$ ), *cong-tag-2* ( $\otimes_{F_n}$ )])  
**using** *nth-map length-a*  $\langle l = \text{Suc } l' \rangle$  *i-less* **by** *simp*  
**also have** ... =  $(b' ! i) \oplus_{F_n} (c' ! i) \otimes_{F_n} (\text{inv}_{F_n} 2 [\lceil_{F_n} s] [\lceil_{F_n} i])$   
**apply** (*intro arg-cong2*[**where**  $f = (\oplus_{F_n})$ ] *refl m-comm nat-pow-closed*  
*Units-inv-closed two-is-unit*)  
**using** *to-residue-ring-in-carrier ci-c'i[symmetric]* **by** *simp*  
**finally show** *?thesis unfolding 1* .  
**next**  
**case** *False*  
**then have** *it-op: (if it then divide-by-power-of-2 else multiply-with-power-of-2)*  
= *multiply-with-power-of-2* **by** *argo*  
**have** *map to-residue-ring g ! i = to-residue-ring (g ! i)*  
**apply** (*intro nth-map*)  
**unfolding** *length-g* **using** *i-less* **by** *simp*  
**also have** ... = *to-residue-ring (add-fermat (b ! i) (multiply-with-power-of-2*  
*(c ! i) (s \* ([0..<2 ^ (l - 1)] ! i) mod 2 ^ (k + 1))))*  
**unfolding** *g-def fft-iff-combine-b-c-add-correct[OF length-b length-c]* *it-op*  
**apply** (*intro arg-cong*[**where**  $f = \text{to-residue-ring}$ ] *nth-map3*)  
**unfolding** *length-b length-c* **using** *i-less* **by** *simp-all*  
**also have** ... = *to-residue-ring (add-fermat (b ! i) (multiply-with-power-of-2*  
*(c ! i) (s \* i mod 2 ^ (k + 1))))*  
**using** *i-less* **by** *simp*  
**also have** ... = *to-residue-ring (b ! i) \oplus\_{F\_n} to-residue-ring (multiply-with-power-of-2*  
*(c ! i) (s \* i mod 2 ^ (k + 1))))*  
**by** (*intro add-fermat-correct bi-carrier multiply-with-power-of-2-closed ci-carrier*)  
**also have** ... = *to-residue-ring (b ! i) \oplus\_{F\_n} to-residue-ring (c ! i) \otimes\_{F\_n} 2 [\lceil\_{F\_n}*  
*(s \* i mod 2 ^ (k + 1))*)  
**by** (*intro arg-cong2*[**where**  $f = (\oplus_{F_n})$ ] *multiply-with-power-of-2-correct refl*  
*ci-carrier*)  
**also have** ... =  $(b' ! i) \oplus_{F_n} (c' ! i) \otimes_{F_n} 2 [\lceil_{F_n} (s * i \text{ mod } 2 ^ (k + 1))]$   
**unfolding** *bi-b'i ci-c'i* **by** (*rule refl*)  
**also have** ... =  $(b' ! i) \oplus_{F_n} (c' ! i) \otimes_{F_n} 2 [\lceil_{F_n} (s * i)]$   
**by** (*intro-cong* [*cong-tag-2* ( $\oplus_{F_n}$ ), *cong-tag-2* ( $\otimes_{F_n}$ )] *more: pow-mod-carrier-length*  
*mod-mod-trivial*)  
**also have** ... =  $(b' ! i) \oplus_{F_n} (c' ! i) \otimes_{F_n} ((2 [\lceil_{F_n} s] [\lceil_{F_n} i])$   
**by** (*intro-cong* [*cong-tag-2* ( $\oplus_{F_n}$ ), *cong-tag-2* ( $\otimes_{F_n}$ )] *more: nat-pow-pow[symmetric]*  
*two-in-carrier*)  
**finally have** *1: map to-residue-ring g ! i = ...* .  
**have**  $g' ! i = \text{map3 } (\lambda x y z. x \oplus_{F_n} y \otimes_{F_n} z) b' (\text{map } (([\lceil_{F_n}]) (2 [\lceil_{F_n} s])$   
 $[0..<\text{length local.a div 2}]) c' ! i$   
**unfolding** *g'-def if-False map2-of-map2-r* **by** (*simp add: False*)  
**also have** ... =  $(b' ! i) \oplus_{F_n} ((\text{map } (([\lceil_{F_n}]) (2 [\lceil_{F_n} s]) [0..<\text{length local.a}$   
 $\text{div 2}]) ! i) \otimes_{F_n} (c' ! i)$   
**using** *i-less length-b' length-c'*  $\langle l = \text{Suc } l' \rangle$  *length-a* **by** (*intro nth-map3*)  
*simp-all*  
**also have** ... =  $(b' ! i) \oplus_{F_n} (2 [\lceil_{F_n} s] [\lceil_{F_n} i] \otimes_{F_n} (c' ! i))$   
**apply** (*intro-cong* [*cong-tag-2* ( $\oplus_{F_n}$ ), *cong-tag-2* ( $\otimes_{F_n}$ )])  
**using** *nth-map length-a*  $\langle l = \text{Suc } l' \rangle$  *i-less* **by** *simp*

**also have**  $\dots = (b' ! i) \oplus_{F_n} (c' ! i) \otimes_{F_n} (2 [\bigwedge_{F_n} s) [\bigwedge_{F_n} i$   
**apply** (*intro arg-cong2*[**where**  $f = (\oplus_{F_n})$ ] *refl m-comm nat-pow-closed*  
*two-in-carrier*)  
**using** *to-residue-ring-in-carrier ci-c'i[symmetric]* **by** *simp*  
**finally show** *?thesis unfolding 1 .*  
**qed**  
**qed**

**have**  $h-h'$ :  $h' = \text{map to-residue-ring } h$   
**proof** (*intro nth-equalityI*)  
**show**  $\text{length } h' = \text{length } (\text{map to-residue-ring } h)$   
**by** (*simp add: length-h length-h'*)  
**next**  
**fix**  $i$   
**assume**  $i < \text{length } h'$   
**then have**  $i\text{-less}$ :  $i < 2 \wedge (l - 1)$  **unfolding**  $\text{length-h}'$  **using**  $\langle l = \text{Suc } l' \rangle$  **by**  
*simp*

**have**  $bi\text{-carrier}$ :  $b ! i \in \text{fermat-non-unique-carrier}$   
**using** *set-subseteqD[OF b-carrier] length-b i-less* **by** *simp*  
**have**  $ci\text{-carrier}$ :  $c ! i \in \text{fermat-non-unique-carrier}$   
**using** *set-subseteqD[OF c-carrier] length-c i-less* **by** *simp*

**have**  $bi-b'i$ :  $\text{to-residue-ring } (b ! i) = b' ! i$   
**unfolding**  $b-b'$  **by** (*intro nth-map[symmetric]; simp add: length-b i-less del:*  
 $\langle l = \text{Suc } l' \rangle \text{One-nat-def}$ )  
**have**  $ci-c'i$ :  $\text{to-residue-ring } (c ! i) = c' ! i$   
**unfolding**  $c-c'$  **by** (*intro nth-map[symmetric]; simp add: length-c i-less del:*  
 $\langle l = \text{Suc } l' \rangle \text{One-nat-def}$ )

**show**  $h' ! i = (\text{map to-residue-ring } h) ! i$   
**proof** (*cases it*)  
**case** *True*  
**then have**  $it\text{-op}$ : (*if it then divide-by-power-of-2 else multiply-with-power-of-2*)  
 $= \text{divide-by-power-of-2}$  **by** *argo*  
**have**  $\text{map to-residue-ring } h ! i = \text{to-residue-ring } (h ! i)$   
**apply** (*intro nth-map*)  
**unfolding**  $\text{length-h}$  **using**  $i\text{-less}$  **by** *simp*  
**also have**  $\dots = \text{to-residue-ring } (\text{subtract-fermat } (b ! i) (\text{divide-by-power-of-2}$   
 $(c ! i) (s * ([0..<2 \wedge (l - 1)] ! i) \text{ mod } 2 \wedge (k + 1))))$   
**unfolding**  $h\text{-def}$  *fft-iff-combine-b-c-subtract-correct*[*OF length-b length-c*]  
*it-op*  
**apply** (*intro arg-cong*[**where**  $f = \text{to-residue-ring}$ ] *nth-map3*)  
**unfolding**  $\text{length-b}$   $\text{length-c}$  **using**  $i\text{-less}$  **by** *simp-all*  
**also have**  $\dots = \text{to-residue-ring } (\text{subtract-fermat } (b ! i) (\text{divide-by-power-of-2}$   
 $(c ! i) (s * i \text{ mod } 2 \wedge (k + 1))))$   
**using**  $i\text{-less}$  **by** *simp*  
**also have**  $\dots = \text{to-residue-ring } (b ! i) \ominus_{F_n} \text{to-residue-ring } (\text{divide-by-power-of-2}$   
 $(c ! i) (s * i \text{ mod } 2 \wedge (k + 1))))$

**by** (*intro subtract-fermat-correct bi-carrier divide-by-power-of-2-closed ci-carrier*)  
**also have** ... = *to-residue-ring* (b ! i)  $\ominus_{F_n}$  *to-residue-ring* (c ! i)  $\otimes_{F_n}$  *inv*  $F_n$   
 $2 \ [\frown]_{F_n} (s * i \bmod 2 \wedge (k + 1))$   
**by** (*intro arg-cong2*[**where** f = ( $\lambda x y. x \ominus_{F_n} y$ )] *divide-by-power-of-2-correct refl ci-carrier*)  
**also have** ... = (b' ! i)  $\ominus_{F_n}$  (c' ! i)  $\otimes_{F_n}$  *inv*  $F_n$   $2 \ [\frown]_{F_n} (s * i \bmod 2 \wedge (k + 1))$   
**unfolding** *bi-b'i ci-c'i* **by** (*rule refl*)  
**also have** ... = (b' ! i)  $\ominus_{F_n}$  (c' ! i)  $\otimes_{F_n}$  *inv*  $F_n$   $2 \ [\frown]_{F_n} (s * i)$   
**by** (*intro-cong* [*cong-tag-2* ( $\lambda x y. x \ominus_{F_n} y$ ), *cong-tag-2* ( $\otimes_{F_n}$ )] *more: inv-pow-mod-carrier-length mod-mod-trivial*)  
**also have** ... = (b' ! i)  $\ominus_{F_n}$  (c' ! i)  $\otimes_{F_n}$  ((*inv*  $F_n$   $2 \ [\frown]_{F_n} s$ ) [ $\frown]_{F_n} i$ )  
**by** (*intro-cong* [*cong-tag-2* ( $\lambda x y. x \ominus_{F_n} y$ ), *cong-tag-2* ( $\otimes_{F_n}$ )] *more: nat-pow-pow[symmetric] Units-inv-closed two-is-unit*)  
**finally have** 1: *map to-residue-ring* h ! i = ... .  
**have** h' ! i = *map3* ( $\lambda x y z. x \ominus_{F_n} y \otimes_{F_n} z$ ) b' (*map* (( $\frown]_{F_n}$ ) (*inv*  $F_n$   $2 \ [\frown]_{F_n} s$ )) [ $0..<length\ local.a\ div\ 2$ ]) c' ! i  
**unfolding** *h'-def eqTrueI[OF True] if-True map2-of-map2-r* **by** (*rule refl*)  
**also have** ... = (b' ! i)  $\ominus_{F_n}$  ((*map* (( $\frown]_{F_n}$ ) (*inv*  $F_n$   $2 \ [\frown]_{F_n} s$ )) [ $0..<length\ local.a\ div\ 2$ ]) ! i)  $\otimes_{F_n}$  (c' ! i)  
**using** *i-less length-b' length-c' <l = Suc l'> length-a* **by** (*intro nth-map3*)  
*simp-all*  
**also have** ... = (b' ! i)  $\ominus_{F_n}$  (*inv*  $F_n$   $2 \ [\frown]_{F_n} s$ ) [ $\frown]_{F_n} i \otimes_{F_n} (c' ! i)$   
**apply** (*intro-cong* [*cong-tag-2* ( $\lambda x y. x \ominus_{F_n} y$ ), *cong-tag-2* ( $\otimes_{F_n}$ )])  
**using** *nth-map length-a <l = Suc l'> i-less* **by** *simp*  
**also have** ... = (b' ! i)  $\ominus_{F_n}$  (c' ! i)  $\otimes_{F_n}$  (*inv*  $F_n$   $2 \ [\frown]_{F_n} s$ ) [ $\frown]_{F_n} i$   
**apply** (*intro arg-cong2*[**where** f = ( $\lambda x y. x \ominus_{F_n} y$ )] *refl m-comm nat-pow-closed Units-inv-closed two-is-unit*)  
**using** *to-residue-ring-in-carrier ci-c'i[symmetric]* **by** *simp*  
**finally show** ?thesis **unfolding** 1 .  
**next**  
**case** *False*  
**then have** *it-op*: (*if it then divide-by-power-of-2 else multiply-with-power-of-2*)  
= *multiply-with-power-of-2* **by** *argo*  
**have** *map to-residue-ring* h ! i = *to-residue-ring* (h ! i)  
**apply** (*intro nth-map*)  
**unfolding** *length-h* **using** *i-less* **by** *simp*  
**also have** ... = *to-residue-ring* (*subtract-fermat* (b ! i) (*multiply-with-power-of-2* (c ! i) (s \* ([ $0..<2 \wedge (l - 1)$ ] ! i)  $\bmod 2 \wedge (k + 1)$ )))  
**unfolding** *h-def fft-iff-combine-b-c-subtract-correct[OF length-b length-c]*  
*it-op*  
**apply** (*intro arg-cong*[**where** f = *to-residue-ring*] *nth-map3*)  
**unfolding** *length-b length-c* **using** *i-less* **by** *simp-all*  
**also have** ... = *to-residue-ring* (*subtract-fermat* (b ! i) (*multiply-with-power-of-2* (c ! i) (s \* i  $\bmod 2 \wedge (k + 1)$ )))  
**using** *i-less* **by** *simp*  
**also have** ... = *to-residue-ring* (b ! i)  $\ominus_{F_n}$  *to-residue-ring* (*multiply-with-power-of-2* (c ! i) (s \* i  $\bmod 2 \wedge (k + 1)$ )))  
**by** (*intro subtract-fermat-correct bi-carrier multiply-with-power-of-2-closed*)

*ci-carrier*)  
**also have** ... = *to-residue-ring* (b ! i)  $\ominus_{F_n}$  *to-residue-ring* (c ! i)  $\otimes_{F_n} 2 [\uparrow]_{F_n}$   
(s \* i mod 2  $\wedge$  (k + 1))  
**by** (*intro arg-cong2*[**where** f = ( $\lambda x y. x \ominus_{F_n} y$ )] *multiply-with-power-of-2-correct*  
*refl ci-carrier*)  
**also have** ... = (b' ! i)  $\ominus_{F_n}$  (c' ! i)  $\otimes_{F_n} 2 [\uparrow]_{F_n}$  (s \* i mod 2  $\wedge$  (k + 1))  
**unfolding** *bi-b'i ci-c'i* **by** (*rule refl*)  
**also have** ... = (b' ! i)  $\ominus_{F_n}$  (c' ! i)  $\otimes_{F_n} 2 [\uparrow]_{F_n}$  (s \* i)  
**by** (*intro-cong* [*cong-tag-2* ( $\lambda x y. x \ominus_{F_n} y$ ), *cong-tag-2* ( $\otimes_{F_n}$ )] *more:*  
*pow-mod-carrier-length mod-mod-trivial*)  
**also have** ... = (b' ! i)  $\ominus_{F_n}$  (c' ! i)  $\otimes_{F_n}$  ((2 [ $\uparrow]_{F_n}$  s) [ $\uparrow]_{F_n}$  i)  
**by** (*intro-cong* [*cong-tag-2* ( $\lambda x y. x \ominus_{F_n} y$ ), *cong-tag-2* ( $\otimes_{F_n}$ )] *more:*  
*nat-pow-pow[symmetric] two-in-carrier*)  
**finally have** 1: *map to-residue-ring h ! i = ...* .  
**have** h' ! i = *map3* ( $\lambda x y z. x \ominus_{F_n} y \otimes_{F_n} z$ ) b' (map (([ $\uparrow]_{F_n}$ ) (2 [ $\uparrow]_{F_n}$  s))  
[0..*length local.a div 2*]) c' ! i  
**unfolding** *h'-def if-False map2-of-map2-r* **by** (*simp add: False*)  
**also have** ... = (b' ! i)  $\ominus_{F_n}$  ((map (([ $\uparrow]_{F_n}$ ) (2 [ $\uparrow]_{F_n}$  s)) [0..*length local.a*  
*div 2*]) ! i)  $\otimes_{F_n}$  (c' ! i)  
**using** *i-less length-b' length-c' < l = Suc l' > length-a* **by** (*intro nth-map3*)  
*simp-all*  
**also have** ... = (b' ! i)  $\ominus_{F_n}$  (2 [ $\uparrow]_{F_n}$  s) [ $\uparrow]_{F_n}$  i  $\otimes_{F_n}$  (c' ! i)  
**apply** (*intro-cong* [*cong-tag-2* ( $\lambda x y. x \ominus_{F_n} y$ ), *cong-tag-2* ( $\otimes_{F_n}$ )]  
**using** *nth-map length-a < l = Suc l' > i-less* **by** *simp*  
**also have** ... = (b' ! i)  $\ominus_{F_n}$  (c' ! i)  $\otimes_{F_n}$  (2 [ $\uparrow]_{F_n}$  s) [ $\uparrow]_{F_n}$  i  
**apply** (*intro arg-cong2*[**where** f = ( $\lambda x y. x \ominus_{F_n} y$ )] *refl m-comm nat-pow-closed*  
*two-in-carrier*)  
**using** *to-residue-ring-in-carrier ci-c'i[symmetric]* **by** *simp*  
**finally show** *?thesis unfolding 1* .  
**qed**  
**qed**  
**show** *?case*  
**using** *fntt-def*  
**unfolding** *a-def[symmetric] result-eq map-append g-g'[symmetric] h-h'[symmetric]*  
**by** *argo*  
**qed**

**lemma** *fft-iff-correct:*

**assumes** *length a = 2  $\wedge$  l*  
**assumes** *s = 2  $\wedge$  i*  
**assumes** *i + l = k + 1*  
**assumes** *l > 0*  
**assumes** *set a  $\subseteq$  fermat-non-unique-carrier*  
**shows** *map to-residue-ring (fft-iff it s a) = NTT ((if it then inv $_{F_n}$  2 else 2)*  
[ $\uparrow]_{F_n}$  s) (*map to-residue-ring a*)  
**proof** –  
**let** *? $\mu$  = (if it then inv $_{F_n}$  2 else 2) [ $\uparrow]_{F_n}$  s*  
**have** *inv2s: (inv $_{F_n}$  2 [ $\uparrow]_{F_n}$  s) = inv $_{F_n}$  (2 [ $\uparrow]_{F_n}$  s)*  
**by** (*intro inv-nat-pow[symmetric] two-is-unit*)

```

then have it-unfold:  $it \implies ?\mu = \text{inv}_{Fn} (2 [\wedge]_{Fn} s) \neg it \implies ?\mu = 2 [\wedge]_{Fn} s$ 
  by simp-all
from assms have  $l \leq k + 1$  by simp
from assms have  $i \leq k$  by simp
then have  $(2::nat) \wedge i \leq 2 \wedge k$  by simp

  have  $2 \wedge l \text{ div } 2 = (2::nat) \wedge (l - 1)$  using  $\langle l > 0 \rangle$  by (simp add: Suc-leI
power-diff)
then have pow-2sl:  $(2 [\wedge]_{Fn} s) [\wedge]_{Fn} ((2::nat) \wedge l \text{ div } 2) = \ominus_{Fn} \mathbf{1}_{Fn}$ 
  using two-powers-half-carrier-length-residue-ring[of i l - 1]
  using  $\langle l > 0 \rangle \langle i + l = k + 1 \rangle \langle s = 2 \wedge i \rangle$  two-powers-trivial[of 2 \wedge i]  $\langle i \leq k \rangle$ 
  by simp
have pr: primitive-root  $(2 \wedge l) (2 [\wedge]_{Fn} s)$ 
  unfolding assms(2) by (intro two-powers-primitive-root assms  $\langle i \leq k \rangle$ )

from fft-iff-correct'[OF  $\langle \text{length } a = 2 \wedge l \rangle \langle l \leq k + 1 \rangle \langle \text{set } a \subseteq \text{fermat-non-unique-carrier} \rangle$ ]
have map to-residue-ring  $(\text{fft-iff } it \ s \ a) = \text{FNTT}'' ?\mu (\text{map to-residue-ring } a)$ 
  by simp
also have  $\dots = \text{FNTT}' ?\mu (\text{map to-residue-ring } a)$ 
  apply (intro FNTT''-FNTT')
  using assms(1) by simp
also have  $\dots = \text{FNTT} ?\mu (\text{map to-residue-ring } a)$ 
  by (intro FNTT'-FNTT)
also have  $\dots = \text{NTT} ?\mu (\text{map to-residue-ring } a)$ 
  apply (intro FNTT-NTT[of - 2 \wedge l l])
  subgoal by (intro nat-pow-closed two-in-carrier prop-iff[where  $P = \lambda x. x \in$ 
carrier Fn] Units-inv-closed two-is-unit)
  subgoal by argo
  subgoal
  proof (cases it)
    case True
      then show ?thesis unfolding it-unfold(1)[OF True]
      apply (intro primitive-root-inv)
      subgoal by simp
      subgoal by (rule pr)
      done
    next
      case False
      then show ?thesis unfolding it-unfold(2)[OF False] by (intro pr)
  qed
  subgoal
  proof (cases it)
    case True
      show ?thesis unfolding it-unfold(1)[OF True]
      by (intro inv-halfway-property Units-pow-closed two-is-unit pow-2sl)
    next
      case False
      then show ?thesis

```

```

    unfolding it-unfold(2)[OF False] by (intro pow-2sl)
  qed
  subgoal using assms(1) by simp
  subgoal apply (intro set-subseteqI) using to-residue-ring-in-carrier by simp
  done
  finally show ?thesis .
qed

lemma fft-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 (fft s a) = NTT (2 [∧]Fn s) (map to-residue-ring a)
  unfolding fft.simps using fft-fft-correct[OF assms] by simp

lemma ifft-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)
  unfolding ifft.simps using ifft-ifft-correct[OF assms] by simp

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
  by (induction b xs rule: evens-odds-tm.induct; simp)

```

**lemma** *time-evens-odds-tm-le*:  $\text{time } (\text{evens-odds-tm } b \text{ } xs) \leq \text{length } xs + 1$   
**by** (*induction* *b xs rule: evens-odds-tm.induct; simp*)

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

**definition** *multiply-with-power-of-2-tm* ::  $\text{nat-lsbf} \Rightarrow \text{nat} \Rightarrow \text{nat-lsbf } tm$  **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*]:  
 $\text{val } (\text{multiply-with-power-of-2-tm } xs \text{ } m) = \text{multiply-with-power-of-2 } xs \text{ } m$   
**unfolding** *multiply-with-power-of-2-tm-def multiply-with-power-of-2-def* **by** *simp*

**lemma** *time-multiply-with-power-of-2-tm-le*:  
 $\text{time } (\text{multiply-with-power-of-2-tm } xs \text{ } m) \leq 24 + 26 * \text{max } m (\text{length } xs)$   
**unfolding** *multiply-with-power-of-2-tm-def tm-time-simps*  
**by** (*estimation estimate: time-rotate-right-tm-le*) *simp*

**definition** *divide-by-power-of-2-tm* ::  $\text{nat-lsbf} \Rightarrow \text{nat} \Rightarrow \text{nat-lsbf } tm$  **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*]:  
 $\text{val } (\text{divide-by-power-of-2-tm } xs \text{ } m) = \text{divide-by-power-of-2 } xs \text{ } m$   
**unfolding** *divide-by-power-of-2-tm-def divide-by-power-of-2-def* **by** *simp*

**lemma** *time-divide-by-power-of-2-tm-le*:  
 $\text{time } (\text{divide-by-power-of-2-tm } xs \text{ } m) \leq 24 + 26 * \text{max } m (\text{length } xs)$   
**unfolding** *divide-by-power-of-2-tm-def tm-time-simps*  
**by** (*estimation estimate: time-rotate-left-tm-le*) *simp*

**definition** *add-fermat-tm* ::  $\text{nat-lsbf} \Rightarrow \text{nat-lsbf} \Rightarrow \text{nat-lsbf } tm$  **where**  
*add-fermat-tm* *xs ys =1 do {*  
 $zs \leftarrow xs +_{nt} ys;$   
 $lensz \leftarrow \text{length-tm } zs;$   
 $k1 \leftarrow k +_t 1;$   
 $\text{powk} \leftarrow 2 \hat{=} _t k1;$   
 $\text{powk1} \leftarrow \text{powk} +_t 1;$   
 $b \leftarrow \text{lensz} =_t \text{powk1};$   
*if* *b* *then do {*  
 $zsr \leftarrow \text{butlast-tm } zs;$   
 $\text{inc-nat-tm } zsr$   
*}* *else return* *zs*  
*}*

**lemma** *val-add-fermat-tm*[*simp, val-simp*]:  $\text{val } (\text{add-fermat-tm } xs \text{ } ys) = \text{add-fermat } xs \text{ } ys$   
**unfolding** *add-fermat-tm-def add-fermat-def* **by** (*simp add: Let-def*)

**lemma** *time-add-fermat-tm-le*:  $\text{time } (\text{add-fermat-tm } xs \text{ } ys) \leq 13 + 7 * \text{max } (\text{length } xs \text{ } \text{length } ys)$

```

xs) (length ys) + 28 * 2 ^ k
proof -
  define m where m = max (length xs) (length ys)
  have time (add-fermat-tm xs ys) =
    time (xs +nt ys) +
    time (length-tm (add-nat xs ys)) +
    time (k +t 1) +
    time (2 ^t (k + 1)) +
    time (2 ^ (k + 1) +t 1) +
    time (length (xs +n ys) =t 2 ^ (k + 1) + 1) +
    (if length (xs +n ys) = 2 ^ (k + 1) + 1
     then time (butlast-tm (xs +n ys)) +
       time (inc-nat-tm (butlast (xs +n ys)))
     else 0) + 1
  unfolding add-fermat-tm-def tm-time-simps val-add-nat-tm val-plus-nat-tm
    val-power-nat-tm val-length-tm val-equal-nat-tm val-butlast-tm by simp
  also have ... ≤
    (2 * m + 3) +
    (m + 2) +
    (2 ^ Suc k) +
    12 * 2 ^ Suc k +
    (2 ^ Suc k + 1) +
    (m + 2) +
    (3 * m + 4) + 1
  apply (intro add-mono order.refl)
  subgoal apply (estimation estimate: time-add-nat-tm-le) unfolding m-def by
simp
  subgoal
    unfolding time-length-tm
    apply (estimation estimate: length-add-nat-upper) unfolding m-def by simp
  subgoal using less-exp[of Suc k] by auto
  subgoal apply (estimation estimate: time-power-nat-tm-2-le) by simp
  subgoal by simp
  subgoal unfolding time-equal-nat-tm
    apply (estimation estimate: length-add-nat-upper)
    unfolding m-def[symmetric] by simp
  subgoal
    apply (estimation estimate: time-butlast-tm-le)
    apply (estimation estimate: time-inc-nat-tm-le)
    apply (intro if-leqI)
    subgoal
      apply (subst length-butlast)
      apply (estimation estimate: length-add-nat-upper)
      subgoal using length-add-nat-upper[of xs ys] by simp
      subgoal unfolding m-def[symmetric] by simp
    done
  subgoal by simp
  done
done

```

also have ... =  $13 + 7 * m + 28 * 2^k$  by *simp*  
 finally show *?thesis unfolding m-def* .  
**qed**

**definition** *subtract-fermat-tm* :: *nat-lsbf*  $\Rightarrow$  *nat-lsbf*  $\Rightarrow$  *nat-lsbf tm* **where**  
*subtract-fermat-tm xs ys =1* do {  
   *powk*  $\leftarrow$   $2^k$ ;  
   *minusy*  $\leftarrow$  *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*  
**unfolding** *subtract-fermat-tm-def subtract-fermat-def* **by** *simp*

**lemma** *time-subtract-fermat-tm-le*: *time (subtract-fermat-tm xs ys)  $\leq$*   
 $38 + 66 * 2^k + 26 * \text{length } ys + 7 * \max(\text{length } xs) (\text{length } ys)$   
**unfolding** *subtract-fermat-tm-def tm-time-simps val-power-nat-tm*  
*val-multiply-with-power-of-2-tm*  
**apply** (*estimation estimate: time-power-nat-tm-2-le*)  
**apply** (*estimation estimate: time-multiply-with-power-of-2-tm-le*)  
**apply** (*estimation estimate: time-add-fermat-tm-le*)  
**apply** (*subst length-multiply-with-power-of-2*)  
**apply** (*estimation estimate: Nat-max-le-sum*[of  $2^k$ ])  
**by** *simp*

**definition** *reduce-tm* :: *nat-lsbf*  $\Rightarrow$  *nat-lsbf tm* **where**  
*reduce-tm xs =1* do {  
   (*ys, zs*)  $\leftarrow$  *split-tm xs*;  
   *b*  $\leftarrow$  *zs  $\leq_{nt}$  ys*;  
   if *b* then *ys -<sub>nt</sub> zs*  
   else do {  
   *kpow*  $\leftarrow$   $2^k$ ;  
   *kpow1*  $\leftarrow$  *kpow -<sub>t</sub> 1*;  
   *zeros*  $\leftarrow$  *replicate-tm kpow1 False*;  
   *a1*  $\leftarrow$  *zeros @<sub>t</sub> [True]*;  
   *s*  $\leftarrow$  (*True # a1*)  $+_{nt}$  *ys*;  
   *s -<sub>nt</sub> zs*  
   }  
 }

**lemma** *val-reduce-tm*[*simp, val-simp*]: *val (reduce-tm xs) = reduce xs*  
**unfolding** *reduce-tm-def reduce-def* **by** (*simp split: prod.splits*)

**lemma** *time-reduce-tm-le*: *time (reduce-tm xs)  $\leq$  155 + 85 \* length xs + 46 \* 2<sup>k</sup>*

**proof** –

**obtain** *ys zs* **where** *split-xs: split xs = (ys, zs)* **by** *fastforce*  
**note** *lens = length-split-le*[*OF split-xs*]

```

define b where b = compare-nat zs ys
define kpow1 :: nat where kpow1 =  $2^k - 1$ 
define zeros where zeros = replicate kpow1 False
define a1 where a1 = zeros @ [True]
define s where s = add-nat (True # a1) ys

note defs = b-def kpow1-def zeros-def a1-def s-def

have len-a1: length a1 =  $2^k$ 
  unfolding a1-def zeros-def kpow1-def by simp
have len-s-le: length s ≤  $2^k + \text{length } xs + 2$ 
  unfolding s-def
  apply (estimation estimate: length-add-nat-upper)
  apply (estimation estimate: Nat-max-le-sum)
  apply (estimation estimate: lens(1))
  using len-a1 by simp

have time (reduce-tm xs) =
  time (split-tm xs) +
  time (zs ≤nt ys) +
  (if b then time (ys -nt zs)
  else time ( $2^k$ ) +
  time ( $(2^k) -_t 1$ ) +
  time (replicate-tm kpow1 False) +
  time (zeros @t [True]) +
  time ((True # a1) +nt ys) +
  time (s -nt zs)) + 1
  unfolding reduce-tm-def tm-time-simps val-split-tm split-xs
  Product-Type.prod.case val-compare-nat-tm val-power-nat-tm val-replicate-tm
  val-minus-nat-tm val-append-tm val-add-nat-tm defs[symmetric] by simp
also have ... ≤
  ( $10 * \text{length } xs + 16$ ) + ( $13 * \text{length } xs + 23$ ) +
  (if b then ( $30 * \text{length } xs + 48$ )
  else  $12 * 2^k +$ 
   $2 +$ 
   $2^k +$ 
   $2^k +$ 
  ( $2 * 2^k + 2 * \text{length } xs + 5$ ) +
  ( $30 * 2^k + 60 * \text{length } xs + 108$ )) + 1
  apply (intro add-mono if-prop-cong[where P = (≤)] order.refl refl)
subgoal using time-split-tm-le by simp
subgoal
  apply (estimation estimate: time-compare-nat-tm-le) using lens by simp
subgoal
  apply (estimation estimate: time-subtract-nat-tm-le) using lens by simp
subgoal using time-power-nat-tm-2-le by simp
subgoal by simp
subgoal unfolding time-replicate-tm kpow1-def by simp
subgoal by (simp add: zeros-def kpow1-def)

```

```

subgoal
  apply (estimation estimate: time-add-nat-tm-le)
  apply (estimation estimate: Nat-max-le-sum)
  apply (estimation estimate: lens(1))
  using len-a1 by simp
subgoal
  apply (estimation estimate: time-subtract-nat-tm-le)
  apply (estimation estimate: Nat-max-le-sum)
  apply (estimation estimate: len-s-le)
  apply (estimation estimate: lens(2))
  by simp
done
also have  $\dots \leq 155 + 85 * \text{length } xs + 46 * 2 \wedge k$ 
  by simp
finally show ?thesis .
qed

function (domintros) from-nat-lsbf-tm :: nat-lsbf  $\Rightarrow$  nat-lsbf tm where
from-nat-lsbf-tm xs =1 do {
  k1  $\leftarrow$  k +t 1;
  powk  $\leftarrow$  2  $\wedge_t$  k1;
  lenxs  $\leftarrow$  length-tm xs;
  b  $\leftarrow$  lenxs  $\leq_t$  powk;
  if b then fill-tm powk xs else do {
    xs1  $\leftarrow$  take-tm powk xs;
    xs2  $\leftarrow$  drop-tm powk xs;
    a  $\leftarrow$  xs1 +nt xs2;
    from-nat-lsbf-tm a
  }
}
by pat-completeness auto
termination
apply (intro allI)
subgoal for xs
  apply (induction xs rule: from-nat-lsbf.induct)
  subgoal for xs
    using from-nat-lsbf-tm.domintros[of xs] from-nat-lsbf-dom-termination
    by simp
  done
done

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 (induction xs rule: from-nat-lsbf.induct)
  case (1 xs)
  then show ?case
    unfolding from-nat-lsbf-tm.simps[of xs] val-simps from-nat-lsbf.simps[of xs]

```

unfolding *Let-def* by *simp*  
 qed

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

lemma *e-ge-1*:  $e \geq 1$  by *simp*

lemma *e-ge-2*:  $e \geq 2$  by *simp*

lemma *e-ge-4*:  $k > 0 \implies e \geq 4$  using *power-increasing*[of  $2\ k + 1\ 2::\text{nat}$ ] by *simp*

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

assumes  $xs' = \text{add-nat } (\text{take } e\ xs) (\text{drop } e\ xs)$

shows  $\text{time } (\text{from-nat-lsbf-tm } xs) \leq 18 * e + 4 * \text{length } xs + 9 +$   
 (*if length xs ≤ e then 0 else time (from-nat-lsbf-tm xs')*)

using *assms* proof (*induction xs rule: from-nat-lsbf.induct*)

case ( $1\ xs$ )

have  $\text{time } (\text{from-nat-lsbf-tm } xs) = \text{time } (k +_t 1) +$

$\text{time } (2^{\wedge}_t (k + 1)) +$

$\text{time } (\text{length-tm } xs) +$

$\text{time } (\text{length } xs \leq_t e) +$

(*if length xs ≤ e then time (fill-tm e xs)*

*else time (take-tm e xs) +*

*time (drop-tm e xs) +*

*time (take e xs +<sub>nt</sub> drop e xs) +*

*time (from-nat-lsbf-tm xs')*) + 1

unfolding *from-nat-lsbf-tm.simps*[of  $xs$ ] *tm-time-simps val-simp 1(2)*[*symmetric*]

by *simp*

also have  $\dots \leq e +$

$(12 * e) +$

$(\text{length } xs + 1) +$

$(2 * e + 2) +$

(*if length xs ≤ e then 3 \* (e + length xs) + 5*

*else (e + 1) +*

*(e + 1) +*

*(2 \* length xs + 3) +*

*time (from-nat-lsbf-tm xs')*) + 1

apply (*intro add-mono order.refl if-prop-cong*[where  $P = (\leq)$ ] *refl*)

subgoal unfolding *time-plus-nat-tm 1(2)* using *less-exp*[of  $k + 1$ ] by *simp*

subgoal unfolding *1(2)* by (*rule time-power-nat-tm-2-le*)

subgoal by *simp*

subgoal apply (*estimation estimate: time-less-eq-nat-tm-le*) by *simp*

subgoal apply (*estimation estimate: time-fill-tm-le*) by *simp*

subgoal apply (*estimation estimate: time-take-tm-le*) by *simp*

subgoal apply (*estimation estimate: time-drop-tm-le*) by *simp*

subgoal apply (*estimation estimate: time-add-nat-tm-le*) by *simp*

done

also have  $\dots = 15 * e + \text{length } xs + 4 +$

(*if length xs ≤ e then 3 \* e + 3 \* length xs + 5*

*else 2 \* e + 2 \* length xs + 5 +*

```

    time (from-nat-lsbj-tm xs')
  by simp
  also have ... ≤ 18 * e + 4 * length xs + 9 +
    (if length xs ≤ e then 0 else time (from-nat-lsbj-tm xs'))
  by (cases length xs ≤ e; simp)
  finally show ?case .
qed

lemma time-from-nat-lsbj-tm-le-aux':
  assumes xs' = add-nat (take e xs) (drop e xs)
  shows time (from-nat-lsbj-tm xs) ≤ 66 * e + 4 * length xs + 35 +
    (if length xs ≤ e + 1 then 0 else time (from-nat-lsbj-tm xs'))
proof -
  consider length xs ≤ e | length xs = e + 1 | length xs ≥ e + 2
  by linarith
  then show ?thesis
proof cases
  case 1
  then show ?thesis
  using time-from-nat-lsbj-tm-le-aux[OF assms] by simp
next
  case 2
  consider (2-1) length xs' ≤ e | (2-2) xs' = replicate e False @ [True]
  using add-take-drop-carry-aux[OF assms 2 e-ge-1] by argo
  then show ?thesis
proof cases
  case 2-1
  then have time (from-nat-lsbj-tm xs') ≤ 22 * e + 9
  using time-from-nat-lsbj-tm-le-aux[OF refl, of xs']
  by simp
  then have time (from-nat-lsbj-tm xs) ≤ 44 * e + 22
  using time-from-nat-lsbj-tm-le-aux[OF assms] 2
  by simp
  then show ?thesis by simp
next
  case 2-2
  then have len-xs': length xs' = e + 1 by simp
  define xs'' where xs'' = add-nat (take e xs') (drop e xs')
  from 2-2 have take e xs' = replicate e False drop e xs' = [True] by simp-all
  then have xs'' = True # replicate (e - 1) False
  unfolding add-nat-def xs''-def using add-carry.simps
  by (metis Suc-eq-plus1 Suc-le-D add-carry-True-inc-nat diff-Suc-1 inc-nat.simps(1)
  inc-nat.simps(2) le-refl replicate-Suc self-le-ge2-pow)
  then have len-xs'': length xs'' = e using e-ge-1 by simp
  then have time (from-nat-lsbj-tm xs'') ≤ 22 * e + 9
  using time-from-nat-lsbj-tm-le-aux[OF refl, of xs''] by simp
  then have time (from-nat-lsbj-tm xs) ≤ 44 * e + 22
  using time-from-nat-lsbj-tm-le-aux[OF xs''-def] len-xs'
  by simp

```

```

    then have time (from-nat-lsbf-tm xs) ≤ 66 * e + 35
      using time-from-nat-lsbf-tm-le-aux[OF assms] 2
      by simp
    then show ?thesis by simp
  qed
next
case 3
then show ?thesis
  using time-from-nat-lsbf-tm-le-aux[OF assms] by simp
qed
qed

function time-from-nat-lsbf-tm-bound where
time-from-nat-lsbf-tm-bound l = 88 * e + 4 * l + 48 +
  (if l ≤ 2 * e then 0 else time-from-nat-lsbf-tm-bound (l - (e - 1)))
  by pat-completeness auto
termination
  apply (relation Wellfounded.measure id)
  subgoal by simp
  subgoal for l unfolding in-measure id-def using e-ge-2 by linarith
done
declare time-from-nat-lsbf-tm-bound.simps[simp del]

lemma time-from-nat-lsbf-tm-le-bound:
  assumes length xs ≤ l
  shows time (from-nat-lsbf-tm xs) ≤ time-from-nat-lsbf-tm-bound l
using assms proof (induction l arbitrary: xs rule: time-from-nat-lsbf-tm-bound.induct)
  case (1 l)
  consider length xs ≤ e + 1 | length xs ≥ e + 2 ∧ length xs ≤ 2 * e | length xs
  > 2 * e
  by linarith
  then show ?case
  proof cases
  case 1
  then show ?thesis
    unfolding time-from-nat-lsbf-tm-bound.simps[of l]
    using time-from-nat-lsbf-tm-le-aux'[OF refl, of xs]
    by simp
  next
  case 2
  define xs' where xs' = add-nat (take e xs) (drop e xs)
  have length xs' ≤ e + 1
    unfolding xs'-def
    apply (estimation estimate: length-add-nat-upper)
    using 2 by auto
  then have time (from-nat-lsbf-tm xs') ≤ 70 * e + 39
    using time-from-nat-lsbf-tm-le-aux'[OF refl, of xs'] by auto
  then have time (from-nat-lsbf-tm xs) ≤ 88 * e + 4 * length xs + 48
    using time-from-nat-lsbf-tm-le-aux[OF xs'-def] 2 by auto

```

```

then show ?thesis
  unfolding time-from-nat-lsbf-tm-bound.simps[of l] using 2 1 by linarith
next
case 3
define xs' where xs' = add-nat (take e xs) (drop e xs)
have length (take e xs) = e length (drop e xs) = length xs - e
  using 3 by simp-all
then have max (length (take e xs)) (length (drop e xs)) = length xs - e
  using 3 by linarith
then have length xs' ≤ length xs - e + 1
  unfolding xs'-def
  using length-add-nat-upper[of take e xs drop e xs] by argo
then have len-xs': length xs' ≤ l - (e - 1) using 1.prem1 e-ge-1 3 by linarith

  have ih: time (from-nat-lsbf-tm xs') ≤ time-from-nat-lsbf-tm-bound (l - (e -
1))
  apply (intro 1.IH)
  subgoal using 1.prem3 3 by linarith
  subgoal by (fact len-xs')
  done
then have time (from-nat-lsbf-tm xs) ≤ 18 * e + 4 * length xs + 9 +
  time-from-nat-lsbf-tm-bound (l - (e - 1))
  using time-from-nat-lsbf-tm-le-aux[OF xs'-def] 3 by simp
then show ?thesis
  unfolding time-from-nat-lsbf-tm-bound.simps[of l]
  using 3 1.prem1 by simp
qed
qed

lemma time-from-nat-lsbf-tm-bound-closed:
  assumes x ≤ 2 * e
  assumes x ≥ e + 2
  shows time-from-nat-lsbf-tm-bound (x + l * (e - 1)) =
    (l + 1) * (88 * e + 4 * x + 48) + 4 * (∑ {0..l}) * (e - 1)
proof (induction l)
case 0
then show ?case
  unfolding time-from-nat-lsbf-tm-bound.simps[of x + 0 * (e - 1)]
  using assms by simp
next
case (Suc l)
have x + Suc l * (e - 1) > 2 * e
  using assms e-ge-1 by simp
then have time-from-nat-lsbf-tm-bound (x + Suc l * (e - 1)) =
  88 * e + 4 * (x + Suc l * (e - 1)) + 48 +
  time-from-nat-lsbf-tm-bound (x + Suc l * (e - 1) - (e - 1)) (is - = ?t + ?r)
  unfolding time-from-nat-lsbf-tm-bound.simps[of x + Suc l * (e - 1)]
  apply (intro-cong [cong-tag-2 (+)] more: refl)
  using iffD2[OF not-le] by metis

```

**also have**  $?r = \text{time-from-nat-lsbf-tm-bound } (x + l * (e - 1))$   
**apply** (*intro arg-cong*[**where**  $f = \text{time-from-nat-lsbf-tm-bound}$ ])  
**using** *assms* **by** *simp*  
**also have**  $\dots = (l + 1) * (88 * e + 4 * x + 48) + 4 * (\sum \{0..l\}) * (e - 1)$   
**(is**  $\dots = ?r'$ **)**  
**by** (*rule Suc.IH*)  
**also have**  $?t + ?r' = (\text{Suc } l + 1) * (88 * e + 4 * x + 48) + 4 * \sum \{0..\text{Suc } l\} * (e - 1)$   
**by** (*simp add: add-mult-distrib*)  
**finally show**  $?case$  .  
**qed**

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

**assumes**  $e \geq 4$   
**assumes**  $\text{length } xs \leq c * e$   
**shows**  $\text{time } (\text{from-nat-lsbf-tm } xs) \leq (288 * c + 144) + (96 + 192 * c + 8 * c * c) * e$   
**proof** (*cases length xs ≤ 2 \* e*)  
**case** *True*  
**have**  $\text{time } (\text{from-nat-lsbf-tm } xs) \leq \text{time-from-nat-lsbf-tm-bound } (\text{length } xs)$   
**by** (*intro time-from-nat-lsbf-tm-le-bound order.refl*)  
**also have**  $\dots = 88 * e + 4 * \text{length } xs + 48$   
**unfolding** *time-from-nat-lsbf-tm-bound.simps*[*of length xs*]  
**using** *True* **by** *simp*  
**also have**  $\dots \leq 96 * e + 48$   
**using** *True* **by** *auto*  
**also have**  $\dots \leq (96 + 192 * c + 8 * c * c) * e + (288 * c + 144)$   
**apply** (*intro add-mono mult-le-mono order.refl*)  
**by** *simp-all*  
**finally show**  $?thesis$  **by** *simp*

**next**

**case** *False*  
**define**  $x'$  **where**  $x' = \text{length } xs \bmod (e - 1)$   
**define**  $y'$  **where**  $y' = \text{length } xs \text{ div } (e - 1)$   
**from**  $x'$ -*def*  $y'$ -*def* **have**  $\text{len-}xs'$ :  $\text{length } xs = y' * (e - 1) + x'$  **by** *presburger*  
**from** *False* **have**  $\text{length } xs \geq 2 * (e - 1)$  **by** *simp*  
**then have**  $y' \geq 2$  **unfolding**  $y'$ -*def*  
**by** (*metis add-gr-0 e-ge-1 even-power gcd-nat.eq-iff le-neq-implies-less less-eq-div-iff-mult-less-eq odd-one odd-pos zero-less-diff*)  
**define**  $x$  **where**  $x = (\text{if } x' \leq 2 \text{ then } x' + 2 * (e - 1) \text{ else } x' + (e - 1))$   
**define**  $y$  **where**  $y = (\text{if } x' \leq 2 \text{ then } y' - 2 \text{ else } y' - 1)$   
**have**  $\text{len-}xs$ :  $\text{length } xs = x + y * (e - 1)$   
**unfolding**  $\text{len-}xs'$   
**apply** (*cases x' ≤ 2*)  
**subgoal unfolding**  $x$ -*def*  $y$ -*def* **using**  $\langle y' \geq 2 \rangle$  **by** (*simp add: diff-mult-distrib*)  
**subgoal premises** *prems*  
**proof** –  
**have**  $y' * (e - 1) + x' = (y' - 1 + 1) * (e - 1) + x'$   
**using**  $\langle y' \geq 2 \rangle$  **by** *simp*

```

    also have ... = x' + (e - 1) + (y' - 1) * (e - 1)
      by (simp only: add-mult-distrib)
    also have ... = x + y * (e - 1)
      unfolding x-def y-def using prems by simp
    finally show ?thesis .
  qed
done
have x-lower: x ≥ e + 2
proof (cases x' ≤ 2)
  case True
  then show ?thesis unfolding x-def using assms by simp
next
  case False
  then show ?thesis unfolding x-def by simp
qed
have e - 1 > 0 using e-ge-2 by linarith
have x'-upper: x' < e - 1
  using x'-def pos-mod-bound[of <e - 1> <e - 1 > 0> less-eq-Suc-le mod-less-divisor]
by blast
have x-upper: x ≤ 2 * e
proof (cases x' ≤ 2)
  case True
  then show ?thesis unfolding x-def using x'-upper by simp
next
  case False
  then show ?thesis unfolding x-def using x'-upper by simp
qed
have y ≤ y' unfolding y-def using <y' ≥ 2> by simp
also have ... ≤ (c * e) div (e - 1)
  unfolding y'-def using assms(2) div-le-mono by simp
also have ... = (c * ((e - 1) + 1)) div (e - 1)
  using e-ge-1 by simp
also have ... = c + c div (e - 1)
  unfolding add-mult-distrib2 using div-mult-self3[of e - 1 c]
  using <0 < e - 1> by simp
also have ... ≤ 2 * c using div-le-dividend by simp
finally have y ≤ 2 * c .
have y * (e - 1) ≤ y' * (e - 1)
  unfolding y-def using <y' ≥ 2> by simp
also have ... ≤ length xs unfolding y'-def by simp
finally have ye-le: y * (e - 1) ≤ length xs .
have time (from-nat-lsbf-tm xs) ≤ time-from-nat-lsbf-tm-bound (length xs)
  by (intro time-from-nat-lsbf-tm-le-bound order.refl)
also have ... = (y + 1) * (88 * e + 4 * x + 48) + 4 * ∑ {(0::nat)..y} * (e
- 1)
  unfolding len-xs
  by (intro time-from-nat-lsbf-tm-bound-closed x-lower x-upper)
also have ... ≤ (y + 1) * (88 * e + 4 * x + 48) + 4 * y * y * (e - 1)
  using euler-sum-bound[of y] atMost-atLeast0[of y] by simp

```

```

also have ... ≤ (y + 1) * (96 * e + 48) + 4 * y * y * (e - 1)
  by (estimation estimate: x-upper, simp)
also have ... = (y + 1) * (96 * (e - 1) + 144) + 4 * y * y * (e - 1)
  using e-ge-1 by (simp add: diff-mult-distrib2 add.commute)
also have ... = 96 * (e - 1) + 144 * (y + 1) + 96 * y * (e - 1) + 4 * y * y
* (e - 1)
  by (simp add: add-mult-distrib2)
also have ... = 96 * (e - 1) + 144 * y + 144 + (96 + 4 * y) * (y * (e - 1))
  by (simp add: add-mult-distrib)
also have ... ≤ 96 * length xs + 144 * (2 * c) + 144 + (96 + 4 * (2 * c)) *
length xs
  apply (intro add-mono order.refl mult-le-mono ye-le ⟨y ≤ 2 * c⟩)
  subgoal using False by simp
  done
also have ... = (288 * c + 144) + (192 + 8 * c) * length xs
  by (simp add: add-mult-distrib)
also have ... ≤ (288 * c + 144) + (192 * c + 8 * c * c) * e
  apply (estimation estimate: assms(2))
  by (simp add: add-mult-distrib)
also have ... ≤ (288 * c + 144) + (96 + 192 * c + 8 * c * c) * e
  by (intro add-mono mult-le-mono order.refl; simp)
finally show ?thesis .
qed

```

```

fun fft-combine-b-c-aux-tm where
fft-combine-b-c-aux-tm f g l (revs, s) [] [] = 1 rev-tm revs
| fft-combine-b-c-aux-tm f g l (revs, s) (b # bs) (c # cs) = 1 do {
  c-shifted ← g c s;
  r ← f b c-shifted;
  s-new ← s +t l;
  k1 ← k +t 1;
  powk1 ← 2 ^t k1;
  s-new-mod ← s-new modt powk1;
  fft-combine-b-c-aux-tm f g l (r # revs, s-new-mod) bs cs
}
| 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
using assms apply (induction bs arbitrary: cs revs s)
subgoal for cs revs s by (cases cs; simp)
subgoal for b bs cs revs s by (cases cs; simp)
done

```

```

lemma time-fft-combine-b-c-aux-tm-le:
assumes length bs = length cs
assumes ∧ b. b ∈ set bs ⇒ length b = e

```

**assumes**  $\bigwedge c. c \in \text{set } cs \implies \text{length } c = e$   
**assumes**  $\bigwedge xs \ ys. \text{time } (f \ xs \ ys) \leq 38 + 66 * 2^{\wedge k} + 26 * \text{length } ys + 7 * \max$   
 $(\text{length } xs) (\text{length } ys)$   
**assumes**  $s < e$   
**assumes**  $g = \text{multiply-with-power-of-2-tm} \vee g = \text{divide-by-power-of-2-tm}$   
**shows**  $\text{time } (\text{fft-combine-b-c-aux-tm } f \ g \ l \ (\text{revs}, s) \ bs \ cs) \leq \text{length } \text{revs} + 3 +$   
 $\text{length } bs * (72 + 116 * e + 8 * l)$   
**using** *assms*  
**proof** (*induction bs arbitrary: revs s cs*)  
**case** *Nil*  
**then have**  $cs = []$  **by** *simp*  
**then have**  $\text{time } (\text{fft-combine-b-c-aux-tm } f \ g \ l \ (\text{revs}, s) [] \ cs) = \text{length } \text{revs} + 3$   
**by** *simp*  
**then show** *?case* **by** *simp*  
**next**  
**case** (*Cons b bs*)  
**from** *Cons.prem*s **have** *len-b: length b = e* **by** *simp*  
**from** *Cons.prem*s(1) **obtain**  $c \ cs'$  **where**  $cs = c \# \ cs'$  **by** (*metis length-Suc-conv*)  
**with** *Cons.prem*s **have** *len-c: length c = e* **by** *simp*  
**have** *sl-less: (s + l) mod e < e* **using** *e-ge-1 mod-less-divisor[OF iffD2[OF*  
*less-eq-Suc-le, of 0 e]* **by** *simp*  
**have** *ih: time (fft-combine-b-c-aux-tm f g l (revs', s') bs cs') ≤*  
 $\text{length } \text{revs}' + 3 + \text{length } bs * (72 + 116 * e + 8 * l)$   
**if**  $s' < e$  **for**  $\text{revs}' \ s'$   
**apply** (*intro Cons.IH*)  
**subgoal using** *Cons.prem*s  $\langle cs = c \# \ cs' \rangle$  **by** *simp*  
**subgoal using** *Cons.prem*s **by** *simp*  
**subgoal using** *Cons.prem*s  $\langle cs = c \# \ cs' \rangle$  **by** *simp*  
**subgoal by** (*rule Cons.prem*s)  
**subgoal by** (*fact that*)  
**subgoal by** (*rule Cons.prem*s)  
**done**  
**have** *val-gcs: val (g c s) = multiply-with-power-of-2 c s*  $\vee$  *val (g c s) = di-*  
*vide-by-power-of-2 c s*  
**using** *Cons.prem*s **by** *fastforce*  
**from**  $\langle cs = c \# \ cs' \rangle$  **have**  $\text{time } (\text{fft-combine-b-c-aux-tm } f \ g \ l \ (\text{revs}, s) (b \# \ bs)$   
 $cs) =$   
 $\text{time } (g \ c \ s) +$   
 $\text{time } (f \ b \ (\text{val } (g \ c \ s))) +$   
 $\text{time } (2^{\wedge_t} (k + 1)) +$   
 $\text{time } ((s + l) \text{ mod}_t \ e) +$   
 $\text{time } (\text{fft-combine-b-c-aux-tm } f \ g \ l \ (\text{val } (f \ b \ (\text{val } (g \ c \ s))) \# \ \text{revs}, (s + l) \text{ mod}$   
 $e) \ bs \ cs') +$   
 $s + k + 3$   
**by** (*simp del: One-nat-def*)  
**also have**  $\dots \leq$   
 $(24 + 26 * \max s (\text{length } c)) +$   
 $(38 + 33 * e + 26 * \text{length } c + 7 * \max (\text{length } b) (\text{length } c)) +$   
 $12 * e + (8 * (s + l) + 2 * e + 7) +$

```

    (length revs + 4 + length bs * (72 + 116 * e + 8 * l)) + s + k + 3 (is ...
≤ ?t + k + 3)
  apply (intro add-mono order.refl)
  subgoal
    using time-multiply-with-power-of-2-tm-le[of c s]
    using time-divide-by-power-of-2-tm-le[of c s]
    using Cons.prem1 by fastforce
  subgoal
    using val-gcs Cons.prem1[4][of b val (g c s)]
    using length-multiply-with-power-of-2[of c s] length-divide-by-power-of-2[of c
s]
  by auto
  subgoal by (rule time-power-nat-tm-2-le)
  subgoal by (rule time-mod-nat-tm-le)
  subgoal apply (estimation estimate: ih[OF sl-less]) by simp
  done
  also have ?t + k + 3 = ?t + (k + 3) by (rule add.assoc[of ?t k 3])
  also have ... ≤ ?t + e + 2
    using less-exp[of k] iffD1[OF less-eq-Suc-le, OF less-exp[of k]] by simp
  also have ?t + e + 2 = 75 + 107 * e + 9 * s + 8 * l + length revs + length
bs * (72 + 116 * e + 8 * l)
    unfolding len-b len-c using ⟨s < e⟩ by simp
  also have ... ≤ 75 + 116 * e + 8 * l + length revs + length bs * (72 + 116 *
e + 8 * l)
    using ⟨s < e⟩ by simp
  also have ... = length revs + 3 + length (b # bs) * (72 + 116 * e + 8 * l)
    by simp
  finally show ?case .
qed

```

```

fun fft-iffcombine-b-c-add-tm :: bool ⇒ nat ⇒ nat-lsbf list ⇒ nat-lsbf list ⇒
nat-lsbf list tm where
fft-iffcombine-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-iffcombine-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-iffcombine-b-c-subtract-tm :: bool ⇒ nat ⇒ nat-lsbf list ⇒ nat-lsbf list ⇒
nat-lsbf list tm where
fft-iffcombine-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-iffcombine-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-iffcombine-b-c-add-tm[simp, val-simp]:
assumes length bs = length cs
shows val (fft-iffcombine-b-c-add-tm it l bs cs) = fft-iffcombine-b-c-add it l bs
cs
by (cases it; simp add: assms)

```

**lemma** *val-fft-iff-combine-b-c-subtract-tm*[*simp*, *val-simp*]:  
**assumes**  $\text{length } bs = \text{length } cs$   
**shows**  $\text{val } (\text{fft-iff-combine-b-c-subtract-tm } it \ l \ bs \ cs) = \text{fft-iff-combine-b-c-subtract } it \ l \ bs \ cs$   
**by** (*cases it*; *simp add: assms*)

**lemma** *time-fft-combine-b-c-add-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**  $\text{time } (\text{fft-iff-combine-b-c-add-tm } it \ l \ bs \ cs) \leq 4 + \text{length } bs * (72 + 116 * e + 8 * l)$   
**proof** –  
**have**  
 $\text{time } (\text{fft-combine-b-c-aux-tm } \text{add-fermat-tm } g \ l \ (\[], 0) \ bs \ cs)$   
 $\leq \text{length } (\[] :: \text{nat-lsbf list}) + 3 + \text{length } bs * (72 + 116 * e + 8 * l)$   
**if**  $g = \text{multiply-with-power-of-2-tm} \vee g = \text{divide-by-power-of-2-tm}$  **for**  $g$   
**apply** (*intro time-fft-combine-b-c-aux-tm-le*)  
**subgoal by** (*intro assms*)  
**subgoal using** *assms by simp*  
**subgoal using** *assms by simp*  
**subgoal by** (*estimation estimate: time-add-fermat-tm-le; simp*)  
**subgoal using** *e-ge-1 by simp*  
**subgoal using** *that .*  
**done**  
**then show** *?thesis by (cases it; simp)*  
**qed**

**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**  $\text{time } (\text{fft-iff-combine-b-c-subtract-tm } it \ l \ bs \ cs) \leq 4 + \text{length } bs * (72 + 116 * e + 8 * l)$   
**proof** –  
**have**  
 $\text{time } (\text{fft-combine-b-c-aux-tm } \text{subtract-fermat-tm } g \ l \ (\[], 0) \ bs \ cs)$   
 $\leq \text{length } (\[] :: \text{nat-lsbf list}) + 3 + \text{length } bs * (72 + 116 * e + 8 * l)$   
**if**  $g = \text{multiply-with-power-of-2-tm} \vee g = \text{divide-by-power-of-2-tm}$  **for**  $g$   
**apply** (*intro time-fft-combine-b-c-aux-tm-le*)  
**subgoal by** (*intro assms*)  
**subgoal using** *assms by simp*  
**subgoal using** *assms by simp*  
**subgoal by** (*estimation estimate: time-subtract-fermat-tm-le; simp*)  
**subgoal using** *e-ge-1 by simp*  
**subgoal using** *that .*  
**done**  
**then show** *?thesis by (cases it; simp)*

qed

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

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

**proof** (*induction it l a arbitrary: m rule: fft-iff.induct*)

**case** (1 *it l*)

**then show** ?*case* **by** *simp*

**next**

**case** (2 *it l x*)

**then show** ?*case* **by** *simp*

**next**

**case** (3 *it l x y*)

**then show** ?*case* **by** *simp*

**next**

**case** (4 *it l a1 a2 a3 as*)

**interpret** *fft-context* *k* *it l m a1 a2 a3 as*

**apply** *unfold-locales* **using** 4 **by** *simp*

**obtain** *m'* **where** *m* = *Suc* (*Suc* *m'*) **using** *nat-le-iff-add e-ge-2* **by** *auto*

**have** *len-eo*:  $\text{length } (\text{evens-odds } b \text{ local.a}) = 2^{\text{Suc } m'}$  **for** *b*

**apply** (*cases* *b*)

**subgoal using** *length-evens-odds*(1)[*of local.a*] 4.*prems* **unfolding** *a-def*[*symmetric*]  
<*m* = *Suc* (*Suc* *m'*)>

**by** *simp*

**subgoal using** *length-evens-odds*(2)[*of local.a*] 4.*prems* **unfolding** *a-def*[*symmetric*]  
<*m* = *Suc* (*Suc* *m'*)>

**by** *simp*

**done**

**have** *len-eq*:  $\text{length } (\text{fft-iff } \textit{it} \textit{l} \textit{a}) = \text{length } (\text{fft-iff } \textit{it} \textit{l} \textit{a})$

**using** *length-fft-iff*[*OF len-eo*] **by** *simp*

```

have ih1: val (fft-iff-tm it (2 * l) (evens-odds True local.a)) = fft-iff it (2 * l)
(evens-odds True local.a)
  using len-eo by (intro 4.IH[OF - refl], subst a-def[symmetric], intro refl,
fastforce)
have ih2: val (fft-iff-tm it (2 * l) (evens-odds False local.a)) = fft-iff it (2 * l)
(evens-odds False local.a)
  by (intro 4.IH(2)[OF refl - refl len-eo], subst a-def[symmetric], rule refl)

show ?case unfolding fft-iff-tm.simps fft-iff.simps unfolding a-def[symmetric]
unfolding Let-def val-simp ih1 ih2
unfolding val-fft-iff-combine-b-c-add-tm[OF len-eq] val-fft-iff-combine-b-c-subtract-tm[OF
len-eq]
  by (rule refl)
qed

```

```

lemma time-fft-iff-tm-le-aux:
  assumes  $\bigwedge x. x \in \text{set } a \implies \text{length } x = e$ 
  assumes  $\text{length } a = 2^m$ 
  shows  $\text{time } (\text{fft-iff-tm it } l \ a) \leq 2^{(m-1)} * (52 + 87 * e) + (m-1) * 2^{m-1} * (76 + 116 * e) + (\sum i \leftarrow [0..<m-1]. 2^i) * (8 * 2^{m * l} + 13)$ 
  using assms proof (induction it l a arbitrary: m rule: fft-iff.induct)
    case (1 it l)
      then show ?case by simp
    next
      case (2 it l x)
        then show ?case by simp
    next
      case (3 it l x y)
        have  $\text{time } (\text{fft-iff-tm it } l \ [x, y]) \leq 52 + 87 * e$ 
          unfolding fft-iff-tm.simps tm-time-simps
          apply (estimation estimate: time-add-fermat-tm-le)
          apply (estimation estimate: time-subtract-fermat-tm-le)
          by (simp add: 3)
        also have  $\dots \leq 2^{(m - \text{Suc } 0)} * (52 + 87 * e)$  by simp
        finally show ?case by simp
    next
      case (4 it l a1 a2 a3 as)
        interpret fft-context k it l m a1 a2 a3 as
        apply unfold-locales using 4 by simp
        obtain m' where  $m = \text{Suc } (\text{Suc } m')$  using nat-le-iff-add e-ge-2 by auto
        then have  $\text{Suc } m' = m - 1$  by simp
        have len-eo:  $\text{length } (\text{evens-odds } b \ \text{local.a}) = 2^{\text{Suc } m'}$  for b
          apply (cases b)
          subgoal using length-evens-odds(1)[of local.a] length-a  $\langle m = \text{Suc } (\text{Suc } m') \rangle$ 
            by simp
          subgoal using length-evens-odds(2)[of local.a] length-a  $\langle m = \text{Suc } (\text{Suc } m') \rangle$ 
            by simp
          done
        have len-eo-nth:  $\text{length } x = e$  if  $x \in \text{set } (\text{evens-odds } b \ \text{local.a})$  for b x

```

```

using set-evens-odds[of b local.a] that 4.prem1 unfolding a-def[symmetric] by
auto
define ih-bound where ih-bound = 2 ^ (Suc m' - 1) * (52 + 87 * e) + (Suc
m' - 1) * 2 ^ Suc m' * (76 + 116 * e) +
  (∑ i ← [0..<Suc m' - 1]. 2 ^ i) * (8 * 2 ^ Suc m' * (2 * l) + 13)
have ih1: time (fft-iff-tm it (2 * l) nums1) ≤ ih-bound
unfolding a-def nums1-def ih-bound-def
apply (intro 4.IH(1)[OF refl refl, of Suc m'])
subgoal for x unfolding a-def[symmetric] using len-eo-nth by simp
subgoal unfolding a-def[symmetric] by (rule len-eo)
done
have ih2: time (fft-iff-tm it (2 * l) nums2) ≤ ih-bound
unfolding a-def nums2-def ih-bound-def
apply (intro 4.IH(2)[OF refl refl refl, of Suc m'])
subgoal for x unfolding a-def[symmetric] using len-eo-nth by simp
subgoal unfolding a-def[symmetric] by (rule len-eo)
done

have val-fft1: val (fft-iff-tm it (2 * l) nums1) = fft-iff it (2 * l) nums1
apply (intro val-fft-iff-tm[of - Suc m'])
unfolding nums1-def by (rule len-eo)
have val-fft2: val (fft-iff-tm it (2 * l) nums2) = fft-iff it (2 * l) nums2
apply (intro val-fft-iff-tm[of - Suc m'])
unfolding nums2-def by (rule len-eo)
from length-b length-c have len-bc: length b = length c by simp
have val-add: val (fft-iff-combine-b-c-add-tm it l b c) = fft-iff-combine-b-c-add
it l b c
by (rule val-fft-iff-combine-b-c-add-tm[OF len-bc])
have val-sub: val (fft-iff-combine-b-c-subtract-tm it l b c) = fft-iff-combine-b-c-subtract
it l b c
by (rule val-fft-iff-combine-b-c-subtract-tm[OF len-bc])

have nums-carrier: set nums1 ⊆ fermat-non-unique-carrier set nums2 ⊆ fer-
mat-non-unique-carrier
using 4.prem1 unfolding a-def[symmetric] nums1-def nums2-def using set-evens-odds
by fast+

have b-carrier: set b ⊆ fermat-non-unique-carrier
unfolding b-def apply (intro fft-iff-carrier nums-carrier) unfolding nums1-def
len-eo by fast
have c-carrier: set c ⊆ fermat-non-unique-carrier
unfolding c-def apply (intro fft-iff-carrier nums-carrier) unfolding nums2-def
len-eo by fast

have time (fft-iff-tm it l (a1 # a2 # a3 # as)) =
time (evens-odds-tm True local.a) +
time (evens-odds-tm False local.a) +
time (fft-iff-tm it (2 * l) nums1) +
time (fft-iff-tm it (2 * l) nums2) +

```

```

time (fft-iff-combine-b-c-add-tm it l b c) +
time (fft-iff-combine-b-c-subtract-tm it l b c) +
time (g @t h) + 1
unfolding fft-iff-tm.simps tm-time-simps val-evens-odds-tm
unfolding defs[symmetric] val-fft1 val-fft2 val-add val-sub
by simp
also have ... ≤
  (length local.a + 1) +
  (length local.a + 1) +
  ih-bound +
  ih-bound +
  (4 + length b * (72 + 116 * e + 8 * l)) +
  (4 + length b * (72 + 116 * e + 8 * l)) +
  (length g + 1) + 1
apply (intro add-mono order.refl)
subgoal by (rule time-evens-odds-tm-le)
subgoal by (rule time-evens-odds-tm-le)
subgoal using ih1 .
subgoal using ih2 .
subgoal
  apply (intro time-fft-combine-b-c-add-tm-le[OF len-bc])
  subgoal using b-carrier unfolding fermat-non-unique-carrier-def by auto
  subgoal using c-carrier unfolding fermat-non-unique-carrier-def by auto
  done
subgoal
  apply (intro time-fft-combine-b-c-subtract-tm-le[OF len-bc])
  subgoal using b-carrier unfolding fermat-non-unique-carrier-def by auto
  subgoal using c-carrier unfolding fermat-non-unique-carrier-def by auto
  done
subgoal by simp
done
also have ... = 2 * length local.a + 2 * ih-bound + (2 * length b) * (72 + 116
* e + 8 * l) + length g + 12
  by simp
also have ... = 5 * 2 ^ Suc m' + 2 * ih-bound + 2 * 2 ^ Suc m' * (72 + 116
* e + 8 * l) + 12
  unfolding length-a length-b length-g ‹m = Suc (Suc m')› by simp
also have ... ≤ 8 * 2 ^ Suc m' + 2 * ih-bound + 2 * 2 ^ Suc m' * (72 + 116
* e + 8 * l) + 13
  by simp
also have ... = 2 * ih-bound + 2 ^ m * (76 + 116 * e + 8 * l) + 13
  unfolding ‹m = Suc (Suc m')› by (simp add: add-mult-distrib2)
also have ... = 2 * ih-bound + 2 ^ m * (76 + 116 * e) + 8 * 2 ^ m * l + 13
  by (simp add: add-mult-distrib2)
also have ... = (2 * 2 ^ (Suc m' - 1)) * (52 + 87 * e) +
  (Suc m' - 1) * (2 * 2 ^ Suc m') * (76 + 116 * e) +
  2 * (∑ i ← [0..<Suc m' - 1]. 2 ^ i) * (8 * 2 ^ Suc m' * (2 * l) + 13) +
  2 ^ m * (76 + 116 * e) + 8 * 2 ^ m * l + 13
  unfolding ih-bound-def by simp

```

**also have** ... =  $2^{m-1} * (52 + 87 * e) +$   
 $(Suc\ m' - 1) * 2^m * (76 + 116 * e) +$   
 $2 * (\sum i \leftarrow [0..<Suc\ m' - 1]. 2^i) * (8 * 2^m * l + 13) +$   
 $2^m * (76 + 116 * e) + 8 * 2^m * l + 13$   
**apply** (intro arg-cong2[where f = (+)] refl)  
**subgoal unfolding**  $\langle m = Suc\ (Suc\ m') \rangle$  by simp  
**subgoal unfolding**  $\langle m = Suc\ (Suc\ m') \rangle$  by simp  
**subgoal unfolding**  $\langle m = Suc\ (Suc\ m') \rangle$  by simp  
**done**  
**also have** ... =  $2^{m-1} * (52 + 87 * e) +$   
 $((Suc\ m' - 1) + 1) * 2^m * (76 + 116 * e) +$   
 $(2 * (\sum i \leftarrow [0..<Suc\ m' - 1]. 2^i) + 1) * (8 * 2^m * l + 13)$   
**by** (simp add: add-mult-distrib)  
**also have** ... =  $2^{m-1} * (52 + 87 * e) +$   
 $(m - 1) * 2^m * (76 + 116 * e) +$   
 $(\sum i \leftarrow [0..<Suc\ m']. 2^i) * (8 * 2^m * l + 13)$   
**apply** (intro arg-cong2[where f = (+)] arg-cong2[where f = (\*)] refl)  
**subgoal unfolding**  $\langle m = Suc\ (Suc\ m') \rangle$  by simp  
**subgoal unfolding** sum-list-const-mult[symmetric] power-Suc[symmetric]  
**unfolding** sum-list-split-0 sum-list-index-trafo[of power 2 Suc [0..<Suc\ m' -

1]] map-Suc-upt  
**by** simp  
**done**  
**finally show** ?case **unfolding**  $\langle Suc\ m' = m - 1 \rangle$  .  
**qed**

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

**assumes**  $\bigwedge x. x \in set\ a \implies length\ x = e$   
**assumes**  $length\ a = 2^m$   
**shows**  $time\ (fft-iff-tm\ it\ l\ a) \leq 2^m * (65 + 87 * e) + m * 2^m * (76 +$   
 $116 * e) + (8 * l) * 2^{(2 * m)}$   
**proof** -  
**from** time-fft-iff-tm-le-aux[OF assms]  
**have**  $time\ (fft-iff-tm\ it\ l\ a) \leq 2^{m-1} * (52 + 87 * e) + (m - 1) * 2^m * (76 +$   
 $116 * e) + (\sum i \leftarrow [0..<m-1]. 2^i) * (8 * 2^m * l + 13)$   
**by** simp  
**also have** ...  $\leq 2^m * (52 + 87 * e) + m * 2^m * (76 + 116 * e) + (2^{m-1} -$   
 $1) * (8 * 2^m * l + 13)$   
**apply** (intro add-mono mult-le-mono order.refl)  
**subgoal by** simp  
**subgoal by** simp  
**subgoal using** geo-sum-nat[of 2 m - 1] **by** simp  
**done**  
**also have** ...  $\leq 2^m * (52 + 87 * e) + m * 2^m * (76 + 116 * e) + 2^m * (8 * 2^m * l + 13)$   
**apply** (intro add-mono mult-le-mono order.refl)  
**by** (meson diff-le-self le-trans one-le-numeral power-increasing)  
**also have** ... =  $2^m * (65 + 87 * e) + m * 2^m * (76 + 116 * e) + (8 * l) * 2^{(2 * m)}$

by (*simp add: add-mult-distrib2 power-add[symmetric]*)  
 finally show ?thesis .  
 qed

**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]*:  $\text{length } a = 2^m \implies \text{val } (\text{fft-tm } l a) = \text{fft } l a$   
 by *simp*  
**lemma** *val-iff-tm[simp, val-simp]*:  $\text{length } a = 2^m \implies \text{val } (\text{iff-tm } l a) = \text{iff } l a$   
 by *simp*

**lemma** *time-fft-tm-le*:  
 assumes  $\bigwedge x. x \in \text{set } a \implies \text{length } x = e$   
 assumes  $\text{length } a = 2^m$   
 shows  $\text{time } (\text{fft-tm } l a) \leq 2^m * (66 + 87 * e) + m * 2^m * (76 + 116 * e) + (8 * l) * 2^{(2 * m)}$   
**proof** –  
 have  $\text{time } (\text{fft-tm } l a) = 1 + \text{time } (\text{fft-iff-tm } \text{False } l a)$   
 by *simp*  
 also have  $\dots \leq 1 + (2^m * (65 + 87 * e) + m * 2^m * (76 + 116 * e) + (8 * l) * 2^{(2 * m)})$   
 by (*intro add-mono order.refl time-fft-iff-tm-le assms; assumption*)  
 also have  $\dots \leq 2^m + (2^m * (65 + 87 * e) + m * 2^m * (76 + 116 * e) + (8 * l) * 2^{(2 * m)})$   
 by (*intro add-mono order.refl; simp*)  
 finally show ?thesis by (*simp add: algebra-simps*)  
 qed

**lemma** *time-iff-tm-le*:  
 assumes  $\bigwedge x. x \in \text{set } a \implies \text{length } x = e$   
 assumes  $\text{length } a = 2^m$   
 shows  $\text{time } (\text{iff-tm } l a) \leq 2^m * (66 + 87 * e) + m * 2^m * (76 + 116 * e) + (8 * l) * 2^{(2 * m)}$   
**proof** –  
 have  $\text{time } (\text{iff-tm } l a) = 1 + \text{time } (\text{fft-iff-tm } \text{True } l a)$   
 by *simp*  
 also have  $\dots \leq 1 + (2^m * (65 + 87 * e) + m * 2^m * (76 + 116 * e) + (8 * l) * 2^{(2 * m)})$   
 by (*intro add-mono order.refl time-fft-iff-tm-le assms; assumption*)  
 also have  $\dots \leq 2^m + (2^m * (65 + 87 * e) + m * 2^m * (76 + 116 * e) + (8 * l) * 2^{(2 * m)})$   
 by (*intro add-mono order.refl; simp*)  
 finally show ?thesis by (*simp add: algebra-simps*)  
 qed

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** –

**have** 1:  $2^{(2 * (k + 2) - 1)} > (k + 2) + 1 + 2^{(k + 2)}$  **for**  $k$

**by** (*induction k simp-all*)

**assume**  $\langle n > 1 \rangle$

**then obtain**  $k$  **where**  $n = k + 2$

**by** (*metis Suc-eq-plus1-left add-2-eq-Suc' less-natE*)

**then show** *?thesis* **using** 1 **by** *blast*

**qed**

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

**proof** –

**have** 1:  $2^{(2 * (k + 3) - 2)} \geq (k + 3) + 2^{(k + 3)} + 1$  **for**  $k$

**proof** (*induction k*)

**case** (*Suc k*)

**have**  $2^{(Suc\ k + 3)} \geq Suc\ k + 3$  **by** *simp*

**then have**  $4 * k + 16 + 2^{(Suc\ k + 3)} \geq (Suc\ k + 3) + 1$

**by** *simp*

**then have**  $(Suc\ k + 3) + 2^{(Suc\ k + 3)} + 1 \leq 4 * k + 16 + 2 * 2^{(Suc\ k + 3)}$

**by** *simp*

**also have**  $\dots = 4 * k + 4 * 4 + 2 * 2^{(Suc\ (k + 3))}$  **by** *simp*

**also have**  $\dots = 4 * k + 4 * 4 + 2 * 2 * 2^{(k + 3)}$

**by** (*metis mult.assoc power-Suc*)

**also have**  $\dots \leq 4 * 2^{(2 * (k + 3) - 2)}$  **using** *Suc.IH* **by** *simp*

**also have**  $\dots = 2^{((2 * (k + 3)) - 2 + 2)}$  **by** (*simp add: power-add*)

```

    also have ... = 2 ^ (2 * (Suc k + 3) - 2) by simp
    finally show ?case .
qed simp
assume n > 2
then have n ≥ 3 by simp
then obtain k where n = k + 3
  by (metis add.commute le-Suc-ex)
then show ?thesis using 1
  by (metis add-lessD1 le-eq-less-or-eq less-add-one)
qed
lemma aux-ineq-3: n > 1 ⇒ 2 ^ n ≥ n + 2
proof -
  have 1: 2 ^ (k + 2) ≥ (k + 2) + 2 for k
    by (induction k) simp-all
  assume ⟨n > 1⟩
  then obtain k where n = k + 2
    by (metis Suc-eq-plus1-left add-2-eq-Suc' less-natE)
  then show ?thesis using 1 by blast
qed

```

```

lemma (in residues) nat-embedding-eq: ring.nat-embedding R x = int x mod m
proof (induction x)
  case 0
  then show ?case by (simp add: zero-cong)
next
  case (Suc x)
  then show ?case by (simp add: res-add-eq one-cong mod-add-eq add.commute)
qed

```

```

lemma (in residues) carrier-mod-eq: x ∈ carrier R ⇒ x mod m = x
  unfolding res-carrier-eq by simp

```

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 :: nat
  assumes m-ge-3: ¬ m < 3
begin

```

Lemmas in *nat* resp. *int*.

```

lemma m-gt-0: m > 0 using m-ge-3 by simp

```

```

definition n :: nat where
n ≡ (if odd m then (m + 1) div 2 else (m + 2) div 2)

```

```

definition oe-n :: nat where
oe-n ≡ (if odd m then n + 1 else n)

```

**lemma** *n-gt-1*:  $n > 1$  **unfolding** *n-def* **using** *m-ge-3* **by** *simp*  
**lemma** *n-ge-2*:  $n \geq 2$  **using** *n-gt-1* **by** *simp*  
**lemma** *n-gt-0*:  $n > 0$  **using** *n-gt-1* **by** *simp*  
**lemma** *even-m-imp-n-ge-3*:  $\text{even } m \implies n \geq 3$  **unfolding** *n-def* **using** *m-ge-3* **by** *auto*

**lemma** *n-lt-m*:  $n < m$  **unfolding** *n-def* **using** *m-ge-3* **by** *auto*

**lemma** *oe-n-gt-1*:  $oe-n > 1$  **unfolding** *oe-n-def* **using** *n-gt-1* **by** *simp*  
**lemma** *oe-n-gt-0*:  $oe-n > 0$  **using** *oe-n-gt-1* **by** *simp*

**lemma** *oe-n-le-n*:  $oe-n \leq n + 1$  **unfolding** *oe-n-def* **by** *simp*  
**lemma** *oe-n-minus-1-le-n*:  $oe-n - 1 \leq n$  **unfolding** *oe-n-def* **by** *simp*

**lemma** *two-pow-oe-n-div-2*:  $(2::\text{nat})^{\wedge} oe-n \text{ div } 2 = 2^{\wedge}(oe-n - 1)$   
**by** (*simp add: Suc-leI power-diff oe-n-gt-0*)  
**lemma** *two-pow-oe-n-as-halves*:  $(2::\text{nat})^{\wedge} oe-n = 2^{\wedge}(oe-n - 1) + 2^{\wedge}(oe-n - 1)$   
**using** *two-pow-oe-n-div-2 oe-n-gt-0*  
**by** (*metis add-self-div-2 div-add dvd-power*)  
**lemma** *two-pow-Suc-oe-n-as-prod*:  $(2::\text{nat})^{\wedge}(oe-n + 1) = 4 * 2^{\wedge}(oe-n - 1)$   
**using** *oe-n-gt-0* **by** (*simp add: power-eq-if*)

**lemma** *index-intros*:  
**fixes**  $i :: \text{nat}$   
**assumes**  $i < 2^{\wedge}(oe-n - 1)$   
**shows**  $i < 2^{\wedge} oe-n \ 2^{\wedge}(oe-n - 1) + i < 2^{\wedge} oe-n$   
**using** *assms two-pow-oe-n-as-halves* **by** *simp-all*

**lemma** *index-decomp*:  
**assumes**  $j < (2::\text{nat})^{\wedge}(oe-n + 1)$   
**shows**  
 $j \text{ div } 2^{\wedge}(oe-n - 1) < 4$   
 $j \text{ mod } 2^{\wedge}(oe-n - 1) < 2^{\wedge}(oe-n - 1)$   
 $j = (j \text{ div } 2^{\wedge}(oe-n - 1)) * 2^{\wedge}(oe-n - 1) + (j \text{ mod } 2^{\wedge}(oe-n - 1))$   
**using** *assms two-pow-Suc-oe-n-as-prod*  
**by** (*simp-all add: less-mult-imp-div-less div-mod-decomp*)

**lemma** *index-comp*:  
**fixes**  $i \ j :: \text{nat}$   
**assumes**  $i < 4 \ j < 2^{\wedge}(oe-n - 1)$   
**shows**  
 $i * 2^{\wedge}(oe-n - 1) + j < 2^{\wedge}(oe-n + 1)$   
 $(i * 2^{\wedge}(oe-n - 1) + j) \text{ div } 2^{\wedge}(oe-n - 1) = i$   
 $(i * 2^{\wedge}(oe-n - 1) + j) \text{ mod } 2^{\wedge}(oe-n - 1) = j$   
**proof** –  
**from** *assms* **have**  $i \leq 3$  **by** *simp*  
**then** **have**  $i * 2^{\wedge}(oe-n - 1) + j < 3 * 2^{\wedge}(oe-n - 1) + 2^{\wedge}(oe-n - 1)$   
**using**  $\langle j < 2^{\wedge}(oe-n - 1) \rangle$   
**using** *nat-less-add-iff2 trans-less-add2* **by** *blast*

**then show**  $i * 2^{(oe-n - 1)} + j < 2^{(oe-n + 1)}$   
**unfolding** *two-pow-Suc-oe-n-as-prod* **by** *simp*  
**show**  $(i * 2^{(oe-n - 1)} + j) \text{ div } 2^{(oe-n - 1)} = i$   
**using** *assms* **by** *simp*  
**show**  $(i * 2^{(oe-n - 1)} + j) \text{ mod } 2^{(oe-n - 1)} = j$   
**using** *assms* **by** *simp*  
**qed**

**lemma** *mn*:  
*odd m*  $\implies m = 2 * n - 1$   
*even m*  $\implies m = 2 * n - 2$   
**using** *n-def* **by** *simp-all*

**lemma** *m0*:  $m = (n - 1) + (oe-n - 1)$   
**unfolding** *oe-n-def* **using** *n-gt-0 mn*  
**by** *auto*

**lemma** *m1*:  $m + 1 = (n - 1) + oe-n$   
**using** *m0 oe-n-gt-0* **by** *linarith*

**lemma** *two-pow-m1-as-prod*:  $(2::nat)^{(m + 1)} = 2^{(n - 1)} * 2^{oe-n}$   
**by** (*simp only: power-add[symmetric] m1*)

**lemma** *two-pow-m0-as-prod*:  $(2::nat)^m = 2^{(n - 1)} * 2^{(oe-n - 1)}$   
**using** *m0* **by** (*simp only: power-add[symmetric]*)

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

**have**  $(2::nat)^{(2 * n)} = 2^{(2 * n - 2 + 2)}$   
**apply** (*intro arg-cong2[where f = power] refl*)  
**using** *n-gt-1* **by** *linarith*  
**also have**  $\dots = 2^2 * 2^{(2 * n - 2)}$  **by** *simp*  
**also have**  $\dots \leq 2^2 * 2^m$  **using** *mn* **by** (*cases odd m; simp*)  
**finally show** *?thesis* **by** *simp*

**qed**

**lemma** *oe-n-m-bound-0*:  $oe-n + 2^n < 2^m$

**proof** (*cases odd m*)

**case** *True*

**then have**  $m = 2 * n - 1$   $oe-n = n + 1$  **using** *mn oe-n-def* **by** *simp-all*

**then show** *?thesis* **using** *aux-ineq-1[OF n-gt-1]* **by** *argo*

**next**

**case** *False*

**then have**  $m = 2 * n - 2$   $oe-n = n$   $n > 2$  **using** *mn oe-n-def even-m-imp-n-ge-3*  
**by** *simp-all*

**then show** *?thesis* **using** *aux-ineq-2[OF <n > 2>]* **by** *argo*

**qed**

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

**using** *oe-n-m-bound-0* **by** *simp*

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

by (intro power-increasing oe-n-m-bound-1) simp  
**lemma** two-pow-oe-n-m-bound-0-int:  $2^{\wedge}(oe-n + 2^{\wedge}n) < \text{int-lsb-f-fermat}.n\ m$   
 by (metis oe-n-m-bound-0 one-less-numeral-iff power-strict-increasing-iff semiring-norm(76) trans-less-add1)  
**lemma** two-pow-oe-n-m-bound-1-int:  $2^{\wedge}(oe-n + 1 + 2^{\wedge}n) < \text{int-lsb-f-fermat}.n\ m$   
 using two-pow-oe-n-m-bound-1  
 by (metis le-eq-less-or-eq less-add-one trans-less-add1)

**lemma** oe-n-n-bound-1:  $oe-n + 1 + 2^{\wedge}n \leq 2^{\wedge}(n + 1)$   
**proof** –  
 have  $oe-n + 1 + 2^{\wedge}n \leq n + 2 + 2^{\wedge}n$  **unfolding** oe-n-def **by** simp  
 also have  $\dots \leq 2^{\wedge}n + 2^{\wedge}n$   
 by (intro add-mono order.refl aux-ineq-3 n-gt-1)  
 also have  $\dots = 2^{\wedge}(n + 1)$  **by** simp  
 finally show ?thesis .  
**qed**

**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** –  
 show int-lsb-f-mod (n + 2) **by** unfold-locales simp  
 then **interpret** A : int-lsb-f-mod n + 2 .  
 show A.Zn  $\equiv$  Zn **unfolding** Zn-def A.Zn-def .  
**qed**

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

**sublocale** Fnr : int-lsb-f-fermat n  
 rewrites Fnr.Fn  $\equiv$  Fn  
**unfolding** int-lsb-f-fermat.Fn-def Fn-def .

**sublocale** Fnr-M : multiplicative-subgroup Fn Units Fn units-of Fn  
**by** (rule Fnr.units-subgroup)

**sublocale** Fmr : int-lsb-f-fermat m  
 rewrites Fmr.Fn  $\equiv$  Fm  
**unfolding** int-lsb-f-fermat.Fn-def Fm-def .

**sublocale** Fmr-M : multiplicative-subgroup Fm Units Fm units-of Fm  
**by** (rule Fmr.units-subgroup)

**lemma** *two-pow-oe-n-primitive-root-Fm*:  
*Fmr.primitive-root* ( $2 \wedge \text{oe-n}$ ) ( $2 \lceil_{Fm} (2::\text{nat}) \wedge (n - 1)$ )  
**using** *Fmr.two-powers-primitive-root m0 m1* **by** *force*

**lemma** *two-pow-oe-n-root-of-unity-Fm*:  
*Fmr.root-of-unity* ( $2 \wedge \text{oe-n}$ ) ( $2 \lceil_{Fm} (2::\text{nat}) \wedge (n - 1)$ )  
**using** *two-pow-oe-n-primitive-root-Fm* **by** *simp*

**lemma** *four-prim-root-Fn*: *Fnr.primitive-root* ( $2 \wedge n$ ) ( $2 \lceil_{Fn} (2::\text{nat})$ )  
**using** *Fnr.primitive-root-recursion[OF - Fnr.two-is-primitive-root]* **by** *simp*

**lemma** *two-oe-n*:  $2 \lceil_{Fn} \text{oe-n} = 2 \wedge \text{oe-n}$   
**proof** –  
**have**  $2 \wedge n \geq n + 1$  **using** *aux-ineq-3[OF n-gt-1]* **by** *simp*  
**then have**  $2 \wedge n \geq \text{oe-n}$  **unfolding** *oe-n-def* **by** *simp*  
**then have**  $(2::\text{int}) \wedge \text{oe-n} \leq 2 \wedge 2 \wedge n$  **by** *simp*  
**then have** *two-oe-n-mod-Fn*:  $2 \wedge \text{oe-n} \bmod \text{int } Fnr.n = 2 \wedge \text{oe-n}$   
**using** *zle-iff-zadd* **by** *auto*  
**then show** *?thesis* **unfolding** *Fnr.pow-nat-eq* .  
**qed**

**lemma** *two-oe-n-Units-Fn*:  $2 \wedge \text{oe-n} \in \text{Units } Fn$   
**apply** (*intro Fnr.two-powers-Units*)  
**unfolding** *oe-n-def* **using** *aux-ineq-3[OF n-gt-1]* **by** *simp*

**lemma** *two-oe-n-carrier-Fn*:  $2 \wedge \text{oe-n} \in \text{carrier } Fn$   
**by** (*intro Fnr.Units-closed two-oe-n-Units-Fn*)

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

**definition**  $\mu$  **where**  $\mu = 2 \lceil_{Fn} \text{prim-root-exponent}$

**lemma**  $\mu$ -*Units-Fn*:  $\mu \in \text{Units } Fn$   
**unfolding**  $\mu$ -*def* **by** (*intro Fnr.Units-pow-closed Fnr.two-is-unit*)

**lemma**  $\mu$ -*carrier-Fn*:  $\mu \in \text{carrier } Fn$   
**by** (*intro Fnr.Units-closed  $\mu$ -Units-Fn*)

**lemma**  $\mu$ -*prim-root*: *Fnr.primitive-root* ( $2 \wedge \text{oe-n}$ )  $\mu$   
**proof** (*cases odd m*)  
**case** *True*  
**then show** *?thesis* **unfolding** *oe-n-def  $\mu$ -def prim-root-exponent-def*  
**using** *Fnr.two-in-carrier Fnr.two-is-primitive-root* **by** *simp*  
**next**  
**case** *False*  
**then show** *?thesis* **unfolding** *oe-n-def  $\mu$ -def prim-root-exponent-def*  
**using** *four-prim-root-Fn* **by** *simp*  
**qed**

**lemma**  $\mu$ -*root-of-unity*: *Fnr.root-of-unity* ( $2 \wedge \text{oe-n}$ )  $\mu$   
**using**  $\mu$ -*prim-root* **by** *simp*

**lemma**  $\mu$ -*halfway-property*:  $\mu \lceil_{Fn} ((2::\text{nat}) \wedge \text{oe-n} \text{ div } 2) = \ominus_{Fn} \mathbf{1}_{Fn}$   
**proof** –

```

have prim-root-exponent * (2 ^ oe-n div 2) = 2 ^ n
  unfolding prim-root-exponent-def oe-n-def
  using n-gt-1 by simp
then have  $\mu \ [\wedge]_{F_n} ((2::nat) \wedge \text{oe-n div } 2) = 2 \ [\wedge]_{F_n} ((2::nat) \wedge n)$ 
  unfolding μ-def by (simp add: Fnr.nat-pow-pow[OF Fnr.two-in-carrier])
then show ?thesis
  using Fnr.two-pow-half-carrier-length-residue-ring
  unfolding Fn-def[symmetric] by argo
qed

```

**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 ∈ int-lsbf-fermat.fermat-non-unique-carrier m
begin

```

```

definition num-blocks where num-blocks = subdivide (2 ^ (n - 1)) num

```

```

definition num-blocks-carrier where num-blocks-carrier = map (fill (2 ^ (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 ^ (m + 1)
  using num-carrier by (elim Fmr.fermat-non-unique-carrierE)

```

```

lemma length-num-blocks[simp]: length num-blocks = 2 ^ oe-n
  apply (unfold num-blocks-def)
  apply (intro conjunct1[OF subdivide-correct])
  using two-pow-m1-as-prod by simp-all

```

```

lemma length-nth-num-blocks[simp]:
  fixes i :: nat
  assumes i < 2 ^ oe-n
  shows length (num-blocks ! i) = 2 ^ (n - 1)
  apply (intro mp[OF conjunct2[OF subdivide-correct[of 2 ^ (n - 1) num 2 ^ oe-n]]])
  subgoal by simp
  subgoal using length-num two-pow-m1-as-prod by argo
  subgoal using assms length-num-blocks unfolding num-blocks-def[symmetric]
by simp
done

```

**lemma** *num-blocks-bound*[simp]:  
**fixes**  $i :: \text{nat}$   
**assumes**  $i < 2 \wedge \text{oe-}n$   
**shows**  $\text{Nat-LSBF.to-nat } (\text{num-blocks } ! i) < 2 \wedge 2 \wedge (n - 1)$   
**using** *length-nth-num-blocks*[OF *assms*] *to-nat-length-bound* **by** *metis*

**lemma** *num-blocks-carrier-Fm*[simp]:  
**fixes**  $i :: \text{nat}$   
**assumes**  $i < 2 \wedge \text{oe-}n$   
**shows**  $\text{int } (\text{Nat-LSBF.to-nat } (\text{num-blocks } ! i)) \in \text{carrier } Fm$   
**unfolding** *Fmr.res-carrier-eq* *atLeastAtMost-iff*

**proof** (*intro conjI*)  
**show**  $0 \leq \text{int } (\text{Nat-LSBF.to-nat } (\text{num-blocks } ! i))$  **by** *simp*  
**have**  $\text{int } (\text{Nat-LSBF.to-nat } (\text{num-blocks } ! i)) < 2 \wedge 2 \wedge (n - 1)$   
**using** *num-blocks-bound*[OF *assms*] **by** *simp*  
**also have**  $\dots < 2 \wedge 2 \wedge m$  **using** *n-lt-m* **by** *simp*  
**finally show**  $\text{int } (\text{Nat-LSBF.to-nat } (\text{num-blocks } ! i)) \leq \text{int } (2 \wedge 2 \wedge m + 1) - 1$  **by** *simp*

**qed**

**lemma** *length-num-blocks-carrier*[simp]:  $\text{length } \text{num-blocks-carrier} = 2 \wedge \text{oe-}n$   
**unfolding** *num-blocks-carrier-def* **by** *simp*

**lemma** *to-res-num*:  $Fmr.\text{to-residue-ring } \text{num} = (\bigoplus_{Fm} j \leftarrow [0..<2 \wedge \text{oe-}n].$   
 $\text{map } (\text{int } \circ \text{Nat-LSBF.to-nat}) \text{ num-blocks } ! j \otimes_{Fm} ((2 \uparrow_{Fm} ((2::\text{nat}) \wedge (n - 1)))) \uparrow_{Fm} j))$

**proof** –  
**let**  $?m = \text{int } Fmr.n$   
**have**  $(\bigoplus_{Fm} j \leftarrow [0..<2 \wedge \text{oe-}n].$   
 $\text{map } (\text{int } \circ \text{Nat-LSBF.to-nat}) \text{ num-blocks } ! j \otimes_{Fm} ((2 \uparrow_{Fm} ((2::\text{nat}) \wedge (n - 1)))) \uparrow_{Fm} j) =$   
 $(\bigoplus_{Fm} j \leftarrow [0..<2 \wedge \text{oe-}n].$   
 $\text{map } (\text{int } \circ \text{Nat-LSBF.to-nat}) \text{ num-blocks } ! j \otimes_{Fm} (2 \uparrow_{Fm} (j * (2::\text{nat}) \wedge (n - 1))))$

**apply** (*intro-cong* [*cong-tag-2* ( $\otimes_{Fm}$ )] *more: refl Fmr.monoid-sum-list-cong*)  
**unfolding** *Fmr.nat-pow-pow*[OF *Fmr.two-in-carrier*]  
**by** (*intro arg-cong2*[**where**  $f = (\uparrow_{Fm})$ ] *refl mult commute*)

**also have**  $\dots = (\sum j \leftarrow [0..<2 \wedge \text{oe-}n].$   
 $\text{map } (\text{int } \circ \text{Nat-LSBF.to-nat}) \text{ num-blocks } ! j * (2 \wedge (j * 2 \wedge (n - 1))) \text{ mod } ?m) \text{ mod } ?m) \text{ mod } ?m$   
**unfolding** *Fmr.monoid-sum-list-eq-sum-list* *Fmr.res-mult-eq* *Fmr.pow-nat-eq*

**by** (*rule refl*)

**also have**  $\dots = (\sum j \leftarrow [0..<2 \wedge \text{oe-}n].$   
 $(\text{map } (\text{int } \circ \text{Nat-LSBF.to-nat}) \text{ num-blocks } ! j * 2 \wedge (j * 2 \wedge (n - 1)))) \text{ mod } ?m$   
**by** (*simp only: mod-mult-right-eq sum-list-mod*)

**also have**  $\dots = (\sum j \leftarrow [0..<2 \wedge \text{oe-}n].$   
 $(\text{int } (\text{Nat-LSBF.to-nat } (\text{num-blocks } ! j)) * (2 \wedge (j * 2 \wedge (n - 1)))) \text{ mod } ?m$   
**by** (*intro-cong* [*cong-tag-2* (*mod*), *cong-tag-2* (*\**)] *more: refl semiring-1-sum-list-eq*)

```

      simp-all
    also have ... = int (∑ j ← [0..<2 ^ oe-n].
      (Nat-LSBF.to-nat (num-blocks ! j) * (2 ^ (j * 2 ^ (n - 1)))))) mod ?m
      by (simp add: sum-list-int)
    also have ... = int (Nat-LSBF.to-nat num) mod ?m
      unfolding num-blocks-def
      apply (intro arg-cong[where f = λi. i mod ?m] arg-cong[where f = int])
      apply (intro to-nat-subdivide[symmetric])
      subgoal by simp
      subgoal by (simp only: length-num two-pow-m1-as-prod)
      done
    finally show ?thesis unfolding Fmr.to-residue-ring.simps by argo
  qed

lemma length-num-Zn[simp]: length num-Zn = 2 ^ oe-n
  unfolding num-Zn-def using length-num-blocks by simp
lemma length-nth-num-Zn[simp]:
  fixes i :: nat
  assumes i < 2 ^ oe-n
  shows length (num-Zn ! i) = n + 2
    unfolding num-Zn-def using length-num-blocks Znr.length-reduce asms by
  simp

lemma length-num-Zn-pad[simp]: length num-Zn-pad = pad-length * 2 ^ oe-n
  unfolding num-Zn-pad-def length-concat
proof -
  have sum-list (map length (map (fill pad-length) num-Zn)) =
    sum-list (map (length ∘ (fill pad-length)) num-Zn)
    by simp
  also have ... = sum-list (map (λj. pad-length) num-Zn)
  proof (intro arg-cong[where f = sum-list] map-cong refl)
    fix x
    assume x ∈ set num-Zn
    then obtain i where i < 2 ^ oe-n x = num-Zn ! i using length-num-Zn
      by (metis in-set-conv-nth)
    then have length x = n + 2 using length-nth-num-Zn by simp
    then show (length ∘ fill pad-length) x = pad-length using length-fill pad-length-def
  by simp
  qed
  also have ... = pad-length * 2 ^ oe-n
    using length-num-Zn sum-list-triv[of pad-length num-Zn] by simp
  finally show sum-list (map length (map (fill pad-length) num-Zn)) = ... .
  qed

lemma to-nat-num-Zn-pad:
  Nat-LSBF.to-nat num-Zn-pad = (∑ i ← [0..<2 ^ oe-n]. Nat-LSBF.to-nat (num-Zn
! i) * 2 ^ (i * pad-length))
proof -
  have Nat-LSBF.to-nat num-Zn-pad = (∑ i ← [0..<2 ^ oe-n]. Nat-LSBF.to-nat

```

(*subdivide pad-length num-Zn-pad ! i*) \*  $2^{(i * \text{pad-length})}$   
**using** *length-num-Zn-pad* **by** (*intro to-nat-subdivide length-num-Zn-pad*) (*simp add: pad-length-def*)  
**also have** *subdivide pad-length num-Zn-pad = map (fill pad-length) num-Zn*  
**unfolding** *num-Zn-pad-def*  
**apply** (*intro subdivide-concat*)  
**by** (*simp-all add: Znr.length-reduce length-fill pad-length-def*)  
**also have** ( $\sum i \leftarrow [0..<2^{oe-n}]$ . *Nat-LSBF.to-nat (map (fill pad-length) num-Zn ! i) \* 2^{(i \* pad-length)}*)  
 = ( $\sum i \leftarrow [0..<2^{oe-n}]$ . *Nat-LSBF.to-nat (num-Zn ! i) \* 2^{(i \* pad-length)}*)  
**apply** (*intro semiring-1-sum-list-eq arg-cong2[where f = (\*)] refl*)  
**using** *length-num-Zn* **by** *simp*  
**finally show** *?thesis* .  
**qed**

**lemma** *length-num-dft[simp]: length num-dft = 2^{oe-n}*  
**unfolding** *num-dft-def*  
**by** (*intro Fnr.length-fft simp*)

**lemma** *fill-num-blocks-carrier[simp]: set num-blocks-carrier  $\subseteq$  Fnr.fermat-non-unique-carrier*  
**apply** (*intro set-subseteqI Fnr.fermat-non-unique-carrierI*)  
**by** (*simp only: num-blocks-carrier-def length-num-blocks length-map nth-map length-fill length-nth-num-blocks power-increasing[of n - 1 n + 1 2::nat]*)

**lemma** *num-dft-carrier[simp]: set num-dft  $\subseteq$  Fnr.fermat-non-unique-carrier*  
**unfolding** *num-dft-def*  
**apply** (*intro Fnr.fft-carrier[of - oe-n]*)  
**subgoal by** *simp*  
**subgoal by** (*rule fill-num-blocks-carrier*)  
**done**

**lemma** *to-res-num-dft:*  
*map Fnr.to-residue-ring num-dft = Fnr.NTT  $\mu$  (map Fnr.to-residue-ring num-blocks-carrier)*  
**unfolding** *num-dft-def  $\mu$ -def prim-root-exponent-def*  
**apply** (*intro Fnr.fft-correct[of - oe-n - if odd m then 0 else 1]*)  
**subgoal by** *simp*  
**subgoal unfolding** *prim-root-exponent-def* **by** *simp*  
**subgoal unfolding** *oe-n-def* **by** *simp*  
**subgoal by** (*rule oe-n-gt-0*)  
**subgoal by** (*rule fill-num-blocks-carrier*)  
**done**

**lemma** *length-num-dft-odds[simp]: length num-dft-odds = 2^{(oe-n - 1)}*  
**unfolding** *num-dft-odds-def*  
**by** (*simp add: length-evens-odds two-pow-oe-n-as-halves*)  
**lemma** *num-dft-odds-carrier[simp]: set num-dft-odds  $\subseteq$  Fnr.fermat-non-unique-carrier*  
**unfolding** *num-dft-odds-def* **using** *set-evens-odds num-dft-carrier* **by** *fastforce*

**end**

### 3.4.1 A special residue problem

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

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

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

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

**proof** –

**have**  $\text{aux}: 2 \wedge (k + 2) \geq k + 4$  for  $k$

by (*induction*  $k$ ) *simp-all*

**assume**  $n \geq 2$

**then obtain**  $k$  where  $n = k + 2$  by (*metis* *le-add-diff-inverse2*)

**then show** *?thesis* using  $\text{aux}[\text{of } k]$  by *presburger*

**qed**

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

**proof** –

**assume**  $n \geq 2$

**then show** *?thesis* using *two-pow-n-geq-n-plus-2*[*of*  $n$ ]

by (*meson* *dvd-imp-mod-0* *le-imp-power-dvd*)

**qed**

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

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

$z < 2 \wedge (n + 2) * \text{int-lsbf-fermat.n } n$

$\wedge z \text{ mod } \text{int-lsbf-fermat.n } n = \xi$

$\wedge z \text{ mod } (2 \wedge (n + 2)) = \eta$

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

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

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

**assumes**  $n \geq 2$

**assumes** *length*  $\eta \leq n + 2$

**assumes** *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* (*Nat-LSBF.to-nat*  $z$ )  $n$  (*Nat-LSBF.to-nat*  $\xi$ ) (*Nat-LSBF.to-nat*  $\eta$ )

**unfolding** *solves-special-residue-problem-def*

**proof** (*intro conjI*)

**define**  $\delta$  where  $\delta = \text{int-lsbf-mod.subtract-mod } (n + 2) \eta$  (take  $(n + 2) \xi$ )

**then have**  $z = \xi +_n ((\delta \gg_n (2 \wedge n)) +_n \delta)$

using *assms(4)* by (*simp add: Let-def solve-special-residue-problem-def*)

**then have** *Nat-LSBF.to-nat*  $z = \text{Nat-LSBF.to-nat } \xi + (2 \wedge (2 \wedge n)) * \text{Nat-LSBF.to-nat } \delta + \text{Nat-LSBF.to-nat } \delta$

by (*simp add: add-nat-correct to-nat-app*)

**then have**  $0: \text{Nat-LSBF.to-nat } z = \text{Nat-LSBF.to-nat } \xi + \text{int-lsbf-fermat.n } n * \text{Nat-LSBF.to-nat } \delta$

by *simp*

**then have**  $\text{Nat-LSBF.to-nat } z \text{ mod int-lsbf-fermat.n } n = \text{Nat-LSBF.to-nat } \xi \text{ mod int-lsbf-fermat.n } n$   
**by** *presburger*  
**also have**  $\dots = \text{Nat-LSBF.to-nat } \xi$   
**using** *assms(3)* **by** *simp*  
**finally show**  $\text{Nat-LSBF.to-nat } z \text{ mod int-lsbf-fermat.n } n = \text{Nat-LSBF.to-nat } \xi .$

**have**  $\text{int-lsbf-fermat.n } n \text{ mod } 2^{(n+2)} = 1$   
**using** *assms(1)* *fermat-mod-pow-2-aux[of n]*  
**by** (*metis* *Nat.add-0-right* *add.left-commute* *add-lessD1* *less-exp* *mod-add-right-eq* *mod-less* *nat-1-add-1*)  
**then have**  $1: \text{int } (\text{int-lsbf-fermat.n } n) \text{ mod } 2^{(n+2)} = 1$   
**by** (*metis* *int-ops(2)* *of-nat-numeral* *of-nat-power* *zmod-int*)

**interpret** *Znr: int-lsbf-mod*  $n + 2$   
**apply** *unfold-locales* **by** *simp*

**have**  $\text{int } (\text{Nat-LSBF.to-nat } \delta) \text{ mod int } \text{Znr.n} = \text{Znr.to-residue-ring } \delta$   
**by** (*rule* *Znr.to-residue-ring-def[symmetric]*)  
**also have**  $\dots = \text{Znr.to-residue-ring } \eta \ominus_{\text{Znr.Zn}} \text{Znr.to-residue-ring } (\text{take } (n + 2) \xi)$   
**unfolding** *δ-def*  
**apply** (*intro* *Znr.subtract-mod-correct*)  
**subgoal using** *assms* **by** *argo*  
**subgoal by** *simp*  
**subgoal using** *Znr.m-gt-one* **by** *linarith*  
**done**  
**also have**  $\dots = (\text{int } (\text{Nat-LSBF.to-nat } \eta) - \text{int } (\text{Nat-LSBF.to-nat } \xi)) \text{ mod int } \text{Znr.n}$   
**unfolding** *Znr.residues-minus-eq* *Znr.to-residue-ring-def* *to-nat-take*  
**by** (*simp* *add: mod-diff-eq* *zmod-int*)  
**finally have**  $2: \text{int } (\text{Nat-LSBF.to-nat } \delta) \text{ mod int } \text{Znr.n} = \dots .$

**have**  $\text{Nat-LSBF.to-nat } \eta < 2^{(n+2)}$  **using**  $\langle \text{length } \eta \leq n + 2 \rangle$  *to-nat-length-bound[of η]* *power-increasing[of length η n + 2 2::nat]*  
**by** *linarith*  
**from**  $0$  **have**  $\text{int } (\text{Nat-LSBF.to-nat } z) \text{ mod } \text{Znr.n} = (\text{int } (\text{Nat-LSBF.to-nat } \xi) + \text{int-lsbf-fermat.n } n * \text{int } (\text{Nat-LSBF.to-nat } \delta)) \text{ mod } \text{Znr.n}$   
**using** *int-ops(7)* *int-plus* **by** *presburger*  
**also have**  $\dots = (\text{int } (\text{Nat-LSBF.to-nat } \xi) \text{ mod } \text{Znr.n} + (\text{int } (\text{int-lsbf-fermat.n } n) \text{ mod } \text{Znr.n}) * (\text{int } (\text{Nat-LSBF.to-nat } \delta) \text{ mod } \text{Znr.n})) \text{ mod } \text{Znr.n}$   
**by** (*simp* *only: mod-add-eq[of int (Nat-LSBF.to-nat ξ) Znr.n, symmetric]* *mod-mult-eq[of int (int-lsbf-fermat.n n) Znr.n, symmetric]* *mod-add-right-eq*)  
**also have**  $\dots = (\text{int } (\text{Nat-LSBF.to-nat } \xi) \text{ mod } \text{Znr.n} + (\text{int } (\text{Nat-LSBF.to-nat } \delta) \text{ mod } \text{Znr.n})) \text{ mod } \text{Znr.n}$   
**apply** (*intro-cong* [*cong-tag-2* (*mod*), *cong-tag-2* (*+*)] *more: refl*)  
**using**  $1$  **by** *simp*

**also have** ... = (int (Nat-LSBF.to-nat  $\xi$ ) mod Znr.n + (int (Nat-LSBF.to-nat  $\eta$ ) mod Znr.n - int (Nat-LSBF.to-nat  $\xi$ ) mod Znr.n)) mod Znr.n  
**using** 2 **by** (simp add: mod-simps)  
**also have** ... = int (Nat-LSBF.to-nat  $\eta$ ) mod  $2^{(n+2)}$   
**by** simp  
**also have** ... = int (Nat-LSBF.to-nat  $\eta$ ) **using** <Nat-LSBF.to-nat  $\eta < 2^{(n+2)}$ >  
**by** (metis mod-less of-nat-mod real-of-nat-eq-numeral-power-cancel-iff)  
**finally have** int (Nat-LSBF.to-nat  $z$ ) mod Znr.n = int (Nat-LSBF.to-nat  $\eta$ ) .  
**then show** Nat-LSBF.to-nat  $z$  mod  $2^{(n+2)}$  = Nat-LSBF.to-nat  $\eta$   
**by** (metis nat-int-comparison(1) zmod-int)

**show** Nat-LSBF.to-nat  $z < 2^{(n+2)} * \text{int-lsbf-fermat.n } n$   
**proof** -  
**have** int (Nat-LSBF.to-nat  $z$ ) = int (Nat-LSBF.to-nat  $\xi$ ) + int (int-lsbf-fermat.n  $n$ ) \* int (Nat-LSBF.to-nat  $\delta$ )  
**using** 0  
**using** int-ops(7) int-plus **by** presburger  
**also have** ...  $\leq (2::\text{int})^{(2^n)} + \text{int (int-lsbf-fermat.n } n) * \text{int (Nat-LSBF.to-nat } \delta)$   
**using** assms(3) **by** simp  
**also have** int (int-lsbf-fermat.n  $n$ ) \* int (Nat-LSBF.to-nat  $\delta$ )  $\leq \text{int (int-lsbf-fermat.n } n) * ((2::\text{int})^{(n+2)} - 1)$   
**proof** -  
**have** length  $\delta \leq n + 2$   
**unfolding**  $\delta$ -def  
**apply** (intro Znr.length-subtract-mod <length  $\eta \leq n + 2$ >)  
**using** Znr.length-reduce **by** simp  
**have** Nat-LSBF.to-nat  $\delta \leq 2^{(n+2)} - 1$   
**using** to-nat-length-upper-bound[of  $\delta$ ] power-increasing[OF <length  $\delta \leq n + 2$ >, of  $2::\text{nat}$ ]  
**using** diff-le-mono **by** fastforce  
**then have** int (Nat-LSBF.to-nat  $\delta$ )  $\leq (2::\text{int})^{(n+2)} - 1$   
**using** nat-int-comparison(3)[of Nat-LSBF.to-nat  $\delta$   $2^{(n+2)} - 1$ ]  
**by** (simp add: of-nat-diff)  
**then show** ?thesis  
**using** int-lsbf-fermat.n-positive[of  $n$ ]  
**by** (meson mult-left-mono of-nat-0-le-iff)

**qed**  
**finally have** int (Nat-LSBF.to-nat  $z$ )  $\leq (2::\text{int})^{(2^n)} + (2^{(2^n)} + 1) * ((2::\text{int})^{(n+2)} - 1)$   
**by** force  
**also have** ... =  $((2::\text{int})^{(2^n)} + 1) * 2^{(n+2)} - 1$   
**apply** (simp add: distrib-right)  
**apply** (simp only: diff-conv-add-uminus[of  $(4::\text{int}) * 2^n$  1])  
**apply** (simp only: distrib-left)  
**done**  
**finally have** int (Nat-LSBF.to-nat  $z$ )  $< 2^{(n+2)} * \text{int (int-lsbf-fermat.n } n)$

by (simp add: add commute mult commute)  
 thus  $\text{Nat-LSBF.to-nat } z < 2^{(n+2)} * \text{int-lsbfermat.n } n$   
 by (metis (mono-tags, lifting) of-nat-less-imp-less of-nat-mult of-nat-numeral  
 of-nat-power)  
 qed  
 qed

lemma *fn-zn-coprime*: coprime (int-lsbfermat.n n) ( $2^{(n+2)}$ )

proof –

consider  $n = 0 \mid n = 1 \mid n \geq 2$  by linarith

then show ?thesis

proof cases

case 1

have  $\text{gcd } (3::\text{nat}) 4 = \text{nat } (\text{gcd } (3::\text{int}) 4)$  using gcd-int-int-eq[of 3 4] by simp

also have  $\dots = \text{gcd } 1 3$  using gcd-diff1[of 4::int 3, symmetric] gcd commute[of 4::int 3]

by simp

also have  $\dots = 1$  by simp

finally show ?thesis unfolding coprime-iff-gcd-eq-1 by (simp add: 1)

next

case 2

have  $\text{gcd } (5::\text{nat}) 8 = \text{nat } (\text{gcd } (5::\text{int}) 8)$  using gcd-int-int-eq[of 5 8] by simp

also have  $\dots = \text{nat } (\text{gcd } 3 5)$  using gcd-diff1[of 8::int 5] by (simp add: gcd commute)

also have  $\dots = \text{nat } (\text{gcd } 2 3)$  using gcd-diff1[of 5::int 3] by (simp add: gcd commute)

also have  $\dots = \text{nat } (\text{gcd } 1 2)$  using gcd-diff1[of 3::int 2] by (simp add: gcd commute)

also have  $\dots = 1$  by simp

finally show ?thesis unfolding coprime-iff-gcd-eq-1 by (simp add: 2)

next

case 3

then have  $2^n \geq n + 2$  by (rule two-pow-n-geq-n-plus-2)

then obtain  $k$  where  $2^n = (n + 2) + k$  by (meson le-iff-add)

then have  $0: (2::\text{nat})^2^n = 2^{(n+2)} * 2^k$  by (simp add: power-add)

show ?thesis

unfolding coprime-iff-gcd-eq-1 gcd-red-nat[of  $2^2^n + 1$   $2^{(n+2)}$ ]

unfolding 0 mod-mult-self4

by simp

qed

qed

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

proof (intro equalityI subsetI)

fix  $x$

assume  $x \in \text{Idl}_{\mathcal{Z}} \{m\} \langle + \rangle_{\mathcal{Z}} \text{Idl}_{\mathcal{Z}} \{n\}$

then obtain  $y \ z$  where  $y \in \text{Idl}_{\mathcal{Z}} \{m\} \ z \in \text{Idl}_{\mathcal{Z}} \{n\} \ x = y \oplus_{\mathcal{Z}} z$

unfolding AbelCoset.set-add-def Coset.set-mult-def by auto

then obtain  $y' \ z'$  where  $y = y' * m \ z = z' * n$

using *int-Idl* by *fastforce*  
 then have 1:  $x = y' * m + z' * n$  using  $\langle x = y \oplus_{\mathcal{Z}} z \rangle$  by *simp*  
 obtain  $m'$  where 2:  $m = m' * \text{gcd } m \ n$   
 by (*metis dvdE gcd-dvd1 mult.commute*)  
 obtain  $n'$  where 3:  $n = n' * \text{gcd } m \ n$   
 by (*metis dvdE gcd-dvd2 mult.commute*)  
 from 1 2 3 have  $x = (y' * m' + z' * n') * \text{gcd } m \ n$   
 by (*simp add: int-distrib(1) mult.assoc*)  
 then show  $x \in \text{Idl}_{\mathcal{Z}} \{\text{gcd } m \ n\}$  using *int-Idl* by *blast*  
**next**  
 fix  $x$   
 assume  $x \in \text{Idl}_{\mathcal{Z}} \{\text{gcd } m \ n\}$   
 then obtain  $x'$  where 1:  $x = x' * \text{gcd } m \ n$  using *int-Idl* by *fastforce*  
 obtain  $s \ t$  where  $\text{gcd } m \ n = s * m + t * n$  using *bezout-int* by *metis*  
 with 1 have  $x = (x' * s) * m \oplus_{\mathcal{Z}} (x' * t) * n$   
 by (*simp add: int-distrib*)  
 moreover have  $(x' * s) * m \in \text{Idl}_{\mathcal{Z}} \{m\}$   $(x' * t) * n \in \text{Idl}_{\mathcal{Z}} \{n\}$   
 using *int-Idl* by *simp-all*  
 ultimately show  $x \in \text{Idl}_{\mathcal{Z}} \{m\} \langle + \rangle_{\mathcal{Z}} \text{Idl}_{\mathcal{Z}} \{n\}$   
 unfolding *AbelCoset.set-add-def Coset.set-mult-def* by *auto*  
**qed**

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

**proof** –

have  $\text{Idl}_{\mathcal{Z}} \{m\} \cap \text{Idl}_{\mathcal{Z}} \{n\} = \{u. \exists x. u = x * m\} \cap \{u. \exists x. u = x * n\}$   
 unfolding *int-Idl* by *simp*  
 also have  $\dots = \{u. m \ \text{dvd } u\} \cap \{u. n \ \text{dvd } u\}$   
 using *dvd-def[symmetric, of - m]*  
 using *dvd-def[symmetric, of - n]*  
 using *mult.commute[of m] mult.commute[of n]*  
 by *algebra*  
 also have  $\dots = \{u. m \ \text{dvd } u \wedge n \ \text{dvd } u\}$  by *blast*  
 also have  $\dots = \{u. \text{lcm } m \ n \ \text{dvd } u\}$  using *lcm-least-iff[of m n]* by *blast*  
 also have  $\dots = \{u. \exists x. u = x * \text{lcm } m \ n\}$   
 using *dvd-def[symmetric, of - lcm m n]*  
 using *mult.commute[of lcm m n]*  
 by *algebra*  
 also have  $\dots = \text{Idl}_{\mathcal{Z}} \{\text{lcm } m \ n\}$  unfolding *int-Idl* by *simp*  
 finally show *?thesis* .

**qed**

**corollary** *coprime m n*  $\implies \text{Idl}_{\mathcal{Z}} \{m\} \langle + \rangle_{\mathcal{Z}} \text{Idl}_{\mathcal{Z}} \{n\} = \text{carrier } \mathcal{Z}$   
 using *int-ideal-add coprime-imp-gcd-eq-1 int.genideal-one* by *simp*

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

unfolding *int-Idl*

by (*metis minus-mult-commute minus-mult-minus*)

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

**apply** (*cases*  $x \geq 0$ )  
**unfolding** *normalize-int-def* **using** *genideal-uminus* **by** *simp-all*

**corollary** *coprime*  $m\ n \implies \text{Idl}_{\mathcal{Z}}\ \{m\} \cap \text{Idl}_{\mathcal{Z}}\ \{n\} = \text{Idl}_{\mathcal{Z}}\ \{m * n\}$   
**using** *int-ideal-inter lcm-coprime genideal-normalize* **by** *metis*

**lemma** *int-ideal-inter-a-r-coset-distrib*:  $(\text{Idl}_{\mathcal{Z}}\ \{m\} \cap \text{Idl}_{\mathcal{Z}}\ \{n\}) +>_{\mathcal{Z}}\ x = (\text{Idl}_{\mathcal{Z}}\ \{m\} +>_{\mathcal{Z}}\ x) \cap (\text{Idl}_{\mathcal{Z}}\ \{n\} +>_{\mathcal{Z}}\ x)$   
**by** (*auto simp add: a-r-coset-def r-coset-def*)

**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** –  
**have**  $?thesis \iff \text{Idl}_{\mathcal{Z}}\ \{\text{lcm}\ m\ n\} +>_{\mathcal{Z}}\ x = \text{Idl}_{\mathcal{Z}}\ \{\text{lcm}\ m\ n\} +>_{\mathcal{Z}}\ y$   
**using** *ZMod-def ZMod-eq-mod* **by** *algebra*  
**also have**  $\dots \iff (\text{Idl}_{\mathcal{Z}}\ \{m\} \cap \text{Idl}_{\mathcal{Z}}\ \{n\}) +>_{\mathcal{Z}}\ x = (\text{Idl}_{\mathcal{Z}}\ \{m\} \cap \text{Idl}_{\mathcal{Z}}\ \{n\}) +>_{\mathcal{Z}}\ y$   
**using** *int-ideal-inter* **by** *presburger*  
**also have**  $\dots \iff (\text{Idl}_{\mathcal{Z}}\ \{m\} +>_{\mathcal{Z}}\ x) \cap (\text{Idl}_{\mathcal{Z}}\ \{n\} +>_{\mathcal{Z}}\ x) = (\text{Idl}_{\mathcal{Z}}\ \{m\} +>_{\mathcal{Z}}\ y) \cap (\text{Idl}_{\mathcal{Z}}\ \{n\} +>_{\mathcal{Z}}\ y)$   
**by** (*simp only: int-ideal-inter-a-r-coset-distrib*)  
**also have**  $\dots$  **using** *assms ZMod-def ZMod-eq-mod* **by** *blast*  
**finally show**  $?thesis$  **by** *blast*  
**qed**

**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**  $x \bmod (\text{lcm}\ m\ n) = y \bmod (\text{lcm}\ m\ n)$   
**using** *assms chinese-remainder-very-simple-int*  
**by** (*meson lcm-unique-nat mod-eq-iff-dvd-symdiff-nat*)

**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** –  
**from** *assms* **have**  $z1 \bmod (\text{lcm}\ (\text{int-lsbf-fermat}.n\ n)\ (2^{(n+2)})) = z2 \bmod (\text{lcm}\ (\text{int-lsbf-fermat}.n\ n)\ (2^{(n+2)}))$   
**unfolding** *solves-special-residue-problem-def*  
**using** *chinese-remainder-very-simple-nat* **by** *presburger*  
**moreover have** *coprime*  $(\text{int-lsbf-fermat}.n\ n)\ (2^{(n+2)})$   
**using** *fn-zn-coprime* .

**hence**  $\text{lcm} (\text{int-lsbf-fermat}.n\ n) (2 \wedge (n + 2)) = (\text{int-lsbf-fermat}.n\ n) * (2 \wedge (n + 2))$   
**by** (*simp add: lcm-coprime*)  
**ultimately show**  $z1 = z2$  **using** *assms unfolding solves-special-residue-problem-def*  
**by** (*metis mod-less mult commute*)  
**qed**

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

**definition** *combine-z*  $::\ \text{nat} \Rightarrow \text{nat-lsbf}\ \text{list} \Rightarrow \text{nat-lsbf}$  **where**  
*combine-z*  $l\ zs = \text{combine-z-aux}\ l\ []\ zs$

**lemma** *combine-z-aux-correct*:

**assumes**  $l > 0$   
**assumes**  $\bigwedge z. z \in \text{set}\ zs \implies \text{length}\ z \geq l$   
**shows**  $\text{Nat-LSBF.to-nat} (\text{combine-z-aux}\ l\ acc\ zs) = \text{Nat-LSBF.to-nat} (\text{concat} (\text{rev}\ acc)) +$   
 $2 \wedge (\text{length} (\text{concat}\ acc)) * (\sum i \leftarrow [0..<\text{length}\ zs]. \text{Nat-LSBF.to-nat} (zs\ !\ i) * 2 \wedge (i * l))$   
**using** *assms*  
**proof** (*induction*  $l\ acc\ zs$  **rule:** *combine-z-aux.induct*)  
  **case** ( $1\ l\ acc$ )  
  **then show** *?case* **by** *simp*  
**next**  
  **case** ( $2\ l\ acc\ z$ )  
  **then show** *?case* **by** (*simp add: to-nat-app*)  
**next**  
  **case** ( $3\ l\ acc\ z1\ z2\ zs$ )  
  **define**  $z1h\ z1t$  **where**  $z1h = \text{take}\ l\ z1\ z1t = \text{drop}\ l\ z1$   
  **have**  $\text{lens} : l \leq \text{length} (\text{add-nat}\ z1t\ z2)$   
  **using** *length-add-nat-lower*[*of*  $z1t\ z2$ ]  $3.\text{prems}$  **by** *force*  
  **from** *z1h-z1t-def* **have**  $\text{combine-z-aux}\ l\ acc\ (z1\ \# z2\ \# zs) = \text{combine-z-aux}\ l$   
 $(z1h\ \# acc)\ ((\text{add-nat}\ z1t\ z2)\ \# zs)$   
  **by** *simp*  
  **then have**  $\text{Nat-LSBF.to-nat} (\text{combine-z-aux}\ l\ acc\ (z1\ \# z2\ \# zs)) = \text{Nat-LSBF.to-nat}$   
... **by** *argo*  
  **also have** ...  $= \text{Nat-LSBF.to-nat} (\text{concat} (\text{rev}\ (z1h\ \# acc))) +$   
 $2 \wedge \text{length} (\text{concat}\ (z1h\ \# acc)) * (\sum i \leftarrow [0..<\text{length} (\text{add-nat}\ z1t\ z2\ \# zs)]. \text{Nat-LSBF.to-nat} ((\text{add-nat}\ z1t\ z2\ \#$   
 $zs)\ !\ i) * 2 \wedge (i * l))$   
  **(is** ...  $=\ ?t1 + ?p * ?t2)$

```

apply (intro 3.IH[OF refl])
subgoal unfolding split-at.simps using z1h-z1t-def by simp
subgoal by (rule 3.prem1)
subgoal using 3.prem1 lena by auto
done
also have ?t1 = Nat-LSBF.to-nat (concat (rev acc) @ z1h)
by simp
also have ... = Nat-LSBF.to-nat (concat (rev acc)) + 2 ^ length (concat acc) *
Nat-LSBF.to-nat z1h (is ... = ?ta + ?tb)
by (simp add: to-nat-app)
also have (?ta + ?tb) + ?p * ?t2 = ?ta + (?tb + ?p * ?t2)
by simp
also have ?p = 2 ^ length (concat acc) * 2 ^ length z1h
by (simp add: power-add)
also have length z1h = l using z1h-z1t-def 3.prem1 by simp
also have ?tb + (2 ^ length (concat acc) * 2 ^ l) * ?t2 = 2 ^ length (concat acc)
* (Nat-LSBF.to-nat z1h + 2 ^ l *
(∑ i←[0..<length (add-nat z1t z2 # zs)]. Nat-LSBF.to-nat ((add-nat z1t z2 #
zs) ! i) * 2 ^ (i * l)))
(is - = - * ?t3)
by (simp add: add-mult-distrib2)
also have ?t3 = Nat-LSBF.to-nat z1h +
2 ^ l * (Nat-LSBF.to-nat (add-nat z1t z2) + (∑ i←[1..<Suc (length zs)].
Nat-LSBF.to-nat ((add-nat z1t z2 # zs) ! i) * 2 ^ (i * l)))
(is - = - + - * (- + ?sum))
using sum-list-split-0[of λi. Nat-LSBF.to-nat ((add-nat z1t z2 # zs) ! i) * 2
^ (i * l) length zs] by simp
also have ... = Nat-LSBF.to-nat z1h + 2 ^ l * Nat-LSBF.to-nat z1t + 2 ^ l *
(Nat-LSBF.to-nat z2 + ?sum)
by (simp only: add-mult-distrib2 add-nat-correct)
also have ... = Nat-LSBF.to-nat (z1h @ z1t) + 2 ^ l * (Nat-LSBF.to-nat z2 +
?sum)
by (simp add: to-nat-app ⟨length z1h = l⟩)
also have ... = Nat-LSBF.to-nat z1 + 2 ^ l * (Nat-LSBF.to-nat z2 + ?sum)
using z1h-z1t-def by simp
also have ... = Nat-LSBF.to-nat z1 + 2 ^ l * (Nat-LSBF.to-nat z2 + (∑ i←[1..<Suc
(length zs)]. Nat-LSBF.to-nat ((z2 # zs) ! i) * 2 ^ (i * l)))
apply (intro-cong [cong-tag-2 (+), cong-tag-2 (*)] more: refl sum-list-eq)
subgoal premises prem1 for x
proof -
from prem1 obtain x' where x = Suc x'
by (metis atLeastAtMost-iff atLeastAtMost-upt not0-implies-Suc not-one-le-zero)
then show ?thesis by simp
qed
done
also have ... = Nat-LSBF.to-nat z1 + 2 ^ l * (∑ i ← [0..<Suc (length zs)].
Nat-LSBF.to-nat ((z2 # zs) ! i) * 2 ^ (i * l))
using sum-list-split-0[of λi. Nat-LSBF.to-nat ((z2 # zs) ! i) * 2 ^ (i * l)] by
simp

```

**also have** ... =  $\text{Nat-LSBF.to-nat } z1 + (\sum i \leftarrow [0..<\text{Suc } (\text{length } zs)]. 2^{\wedge} l * (\text{Nat-LSBF.to-nat } ((z2 \# zs) ! i) * 2^{\wedge} (i * l)))$   
**by** (*intro arg-cong2*[**where**  $f = (+)$ ] *refl sum-list-const-mult*[*symmetric*])  
**also have** ... =  $\text{Nat-LSBF.to-nat } z1 + (\sum i \leftarrow [0..<\text{Suc } (\text{length } zs)]. \text{Nat-LSBF.to-nat } ((z2 \# zs) ! i) * 2^{\wedge} (\text{Suc } i * l))$   
**apply** (*intro arg-cong2*[**where**  $f = (+)$ ] *refl sum-list-eq*)  
**by** (*simp add: power-add*)  
**also have** ... =  $\text{Nat-LSBF.to-nat } z1 + (\sum i \leftarrow [0..<\text{Suc } (\text{length } zs)]. \text{Nat-LSBF.to-nat } ((z1 \# z2 \# zs) ! \text{Suc } i) * 2^{\wedge} (\text{Suc } i * l))$   
**by** *simp*  
**also have** ... =  $\text{Nat-LSBF.to-nat } z1 + (\sum i \leftarrow [1..<\text{Suc } (\text{Suc } (\text{length } zs))]. \text{Nat-LSBF.to-nat } ((z1 \# z2 \# zs) ! i) * 2^{\wedge} (i * l))$   
**unfolding** *sum-list-index-trafo*[*of*  $\lambda i. \text{Nat-LSBF.to-nat } ((z1 \# z2 \# zs) ! i) * 2^{\wedge} (i * l) \text{Suc } [0..<\text{Suc } (\text{length } zs)]$ ]  
**unfolding** *map-Suc-upt* **by** *simp*  
**also have** ... =  $\text{Nat-LSBF.to-nat } ((z1 \# z2 \# zs) ! 0) * 2^{\wedge} (0 * l) + (\sum i \leftarrow [1..<\text{length } (z1 \# z2 \# zs)]. \text{Nat-LSBF.to-nat } ((z1 \# z2 \# zs) ! i) * 2^{\wedge} (i * l))$   
**by** *simp*  
**also have** ... =  $(\sum i \leftarrow [0..<\text{length } (z1 \# z2 \# zs)]. \text{Nat-LSBF.to-nat } ((z1 \# z2 \# zs) ! i) * 2^{\wedge} (i * l))$   
**using** *sum-list-split-0*[**where**  $f = \lambda i. \text{Nat-LSBF.to-nat } ((z1 \# z2 \# zs) ! i) * 2^{\wedge} (i * l)$ ] **by** *simp*  
**finally show** ?*case* .  
**qed**

**lemma** *combine-z-correct*:

**assumes**  $l > 0$   
**assumes**  $\bigwedge z. z \in \text{set } zs \implies \text{length } z \geq l$   
**shows**  $\text{Nat-LSBF.to-nat } (\text{combine-z } l \text{ } zs) = (\sum i \leftarrow [0..<\text{length } zs]. \text{Nat-LSBF.to-nat } (zs ! i) * 2^{\wedge} (i * l))$   
**unfolding** *combine-z-def* **using** *combine-z-aux-correct*[*OF* *assms*] **by** *simp*

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

**assumes**  $\bigwedge z. z \in \text{set } zs \implies \text{length } z \leq lz$   
**assumes**  $\text{length } z \leq lz + 1$   
**assumes**  $l > 0$   
**shows**  $\text{length } (\text{combine-z-aux } l \text{ } \text{acc } (z \# zs)) \leq (lz + 1) * (\text{length } zs + 1) + \text{length } (\text{concat } \text{acc})$   
**using** *assms* **proof** (*induction* *zs* *arbitrary: acc z*)  
**case** *Nil*  
**then show** ?*case* **by** *simp*  
**next**  
**case** (*Cons*  $z1 \text{ } zs$ )  
**then have** *len-drop-z*:  $\text{length } (\text{drop } l \text{ } z) \leq lz$  **by** *simp*  
**have** *lena*:  $\text{length } (\text{add-nat } (\text{drop } l \text{ } z) \text{ } z1) \leq lz + 1$   
**apply** (*estimation estimate: length-add-nat-upper*)  
**using** *len-drop-z* *Cons.prem*s **by** *simp*  
**have**  $\text{length } (\text{combine-z-aux } l \text{ } \text{acc } (z \# z1 \# zs)) = \text{length } (\text{combine-z-aux } l \text{ } (\text{take } l \text{ } z \# \text{acc}) (\text{add-nat } (\text{drop } l \text{ } z) \text{ } z1 \# zs))$

```

    by simp
  also have ... ≤ (lz + 1) * (length zs + 1) + length (concat (take l z # acc))
    apply (intro Cons.IH)
    subgoal using Cons.prem1 by simp
    subgoal using lena .
    subgoal using Cons.prem2 by simp
  done
  also have ... = (lz + 1) * (length (z1 # zs)) + length (take l z) + length (concat
acc)
    by simp
  also have ... ≤ (lz + 1) * length (z1 # zs) + (lz + 1) + length (concat acc)
    apply (intro add-mono mult-le-mono order.refl)
    using Cons.prem1 by simp
  also have ... = (lz + 1) * (length (z1 # zs) + 1) + length (concat acc)
    by simp
  finally show ?case .
qed

```

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

```

  assumes  $\bigwedge z. z \in \text{set } zs \implies \text{length } z \leq lz$ 
  assumes  $l > 0$ 
  shows  $\text{length } (\text{combine-z } l \text{ } zs) \leq (lz + 1) * \text{length } zs$ 
proof (cases zs)
  case Nil
  then show ?thesis by (simp add: combine-z-def)
next
  case (Cons z zs')
  have  $\text{length } (\text{combine-z } l \text{ } zs) \leq (lz + 1) * (\text{length } zs' + 1) + \text{length } (\text{concat } ([]$ 
:: nat-lsbf list)
    unfolding Cons combine-z-def
    apply (intro length-combine-z-aux-le)
    subgoal using assms Cons by simp
    subgoal using assms Cons by fastforce
    subgoal using assms by simp
  done
  also have ... = (lz + 1) * length zs
    unfolding Cons by simp
  finally show ?thesis .
qed

```

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

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

```

  (if  $m < 3$  then int-lsbf-fermat.from-nat-lsbf m (grid-mul-nat a b) else
  let
     $n = (\text{if odd } m \text{ then } (m + 1) \text{ div } 2 \text{ else } (m + 2) \text{ div } 2);$ 
     $oe-n = (\text{if odd } m \text{ then } n + 1 \text{ else } n);$ 
     $a' = \text{subdivide } (2 \wedge (n - 1)) \text{ } a;$ 

```

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

— residue mod  $2^{n+2}$

```
 $\alpha = \text{map } (\text{int-lsbj-mod.reduce } (n + 2)) a';$   
 $u = \text{concat } (\text{map } (\text{fill } (3 * n + 5)) \alpha);$   
 $\beta = \text{map } (\text{int-lsbj-mod.reduce } (n + 2)) b';$   
 $v = \text{concat } (\text{map } (\text{fill } (3 * n + 5)) \beta);$   
 $uw = \text{ensure-length } ((3 * n + 5) * 2^{(oe-n + 1)}) (\text{karatsuba-mul-nat } u v);$   
 $\gamma = \text{subdivide } (2^{(oe-n - 1)}) (\text{subdivide } (3 * n + 5) uw);$   
 $\eta = \text{map4 } (\lambda x y z w.$   
   $\text{int-lsbj-mod.add-mod } (n + 2)$   
   $(\text{int-lsbj-mod.subtract-mod } (n + 2) (\text{take } (n + 2) x) (\text{take } (n + 2) y))$   
   $(\text{int-lsbj-mod.subtract-mod } (n + 2) (\text{take } (n + 2) z) (\text{take } (n + 2) w))$   
  )  
 $(\gamma ! 0) (\gamma ! 1) (\gamma ! 2) (\gamma ! 3);$ 
```

— residue mod  $F_n$

```
 $\text{prim-root-exponent} = (\text{if odd } m \text{ then } 1 \text{ else } 2);$   
 $a'\text{-carrier} = \text{map } (\text{fill } (2^{(n + 1)})) a';$   
 $b'\text{-carrier} = \text{map } (\text{fill } (2^{(n + 1)})) b';$   
 $a\text{-dft} = \text{int-lsbj-fermat.fft } n \text{ prim-root-exponent } a'\text{-carrier};$   
 $b\text{-dft} = \text{int-lsbj-fermat.fft } n \text{ prim-root-exponent } b'\text{-carrier};$   
 $a\text{-dft-odds} = \text{evens-odds False } a\text{-dft};$   
 $b\text{-dft-odds} = \text{evens-odds False } b\text{-dft};$   
 $c\text{-dft-odds} = \text{map2 } (\text{schoenhage-strassen } n) a\text{-dft-odds } b\text{-dft-odds};$   
 $c\text{-diffs} = \text{int-lsbj-fermat.iff } n (\text{prim-root-exponent} * 2) c\text{-dft-odds};$   
 $\xi' = \text{map2 } (\lambda c j. \text{int-lsbj-fermat.add-fermat } n$   
   $(\text{int-lsbj-fermat.divide-by-power-of-2 } c j (\text{oe-n} + \text{prim-root-exponent} * j - 1))$   
   $(\text{int-lsbj-fermat.from-nat-lsbj } n (\text{replicate } (\text{oe-n} + 2^n) \text{False @ [True]})))$   
   $c\text{-diffs } [0..<2^{(oe-n - 1)}];$   
 $\xi = \text{map } (\text{int-lsbj-fermat.reduce } n) \xi';$ 
```

— calculate  $z_j$  for  $j < 2^n$

```
 $z = \text{map2 } (\text{solve-special-residue-problem } n) \xi \eta;$   
 $z\text{-filled} = \text{map } (\text{fill } (2^{(n - 1)})) z;$   
 $z\text{-consts} = \text{replicate } (2^{(oe-n - 1)}) (\text{replicate } (\text{oe-n} + 2^n) \text{False @ [True]});$   
 $z\text{-sum} = \text{combine-z } (2^{(n - 1)}) (z\text{-filled @ } z\text{-consts});$   
 $\text{result} = \text{int-lsbj-fermat.from-nat-lsbj } m z\text{-sum}$ 
```

— return the resulting sum  
*in result*)

**by** *pat-completeness auto*

**termination**

**apply** (*relation Wellfounded.measure*  $(\lambda(n, a, b). n)$ )

**subgoal by** *blast*

**by** *fastforce*

```

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:  $\neg m < 3$ 
  assumes a-carrier:  $a \in \text{int-lsbf-fermat.fermat-non-unique-carrier } m$ 
  assumes b-carrier:  $b \in \text{int-lsbf-fermat.fermat-non-unique-carrier } m$ 
begin

sublocale m-lemmas
  using m-ge-3 by unfold-locales simp

sublocale A: carrier-input m a
  by unfold-locales (rule a-carrier)

sublocale B: carrier-input m b
  by unfold-locales (rule b-carrier)

definition uv-length where uv-length = pad-length *  $2^{\wedge}(\text{oe-n} + 1)$ 
definition uv-unpadded where uv-unpadded = karatsuba-mul-nat A.num-Zn-pad
  B.num-Zn-pad
definition uv where uv = ensure-length uv-length uv-unpadded
definition  $\gamma$ s where  $\gamma$ s = subdivide pad-length uv
definition  $\gamma$  where  $\gamma$  = subdivide ( $2^{\wedge}(\text{oe-n} - 1)$ )  $\gamma$ s
definition  $\eta$  where  $\eta$  = map4 ( $\lambda x y z w.$  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))
  ) ( $\gamma ! 0$ ) ( $\gamma ! 1$ ) ( $\gamma ! 2$ ) ( $\gamma ! 3$ )
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-lsbf-fermat.ifft n (prim-root-exponent * 2)
  c-dft-odds
definition  $\xi'$  where  $\xi'$  = map2 ( $\lambda c j.$  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 +  $2^{\wedge}n$ ) False @ [True])))
  c-diffs [0.. $2^{\wedge}(\text{oe-n} - 1)$ ]
definition  $\xi$  where  $\xi$  = map (int-lsbf-fermat.reduce n)  $\xi'$ 
definition z where z = map2 (solve-special-residue-problem n)  $\xi$   $\eta$ 
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-lsbf-fermat.from-nat-lsbf m z-sum

lemmas defs = n-def oe-n-def A.defs B.defs pad-length-def uv-length-def uv-unpadded-def
  uv-def
   $\gamma$ s-def  $\gamma$ -def  $\eta$ -def c-dft-odds-def c-diffs-def  $\xi'$ -def  $\xi$ -def z-def z-filled-def z-consts-def

```

*z-sum-def result-def prim-root-exponent-def  $\mu$ -def*

**lemma** *result-eq: schoenhage-strassen*  $m a b = \text{result}$   
**unfolding** *schoenhage-strassen.simps*[of  $m a b$ ]  
**unfolding** *iffD2*[*OF eq-False m-ge-3*] *if-False Let-def defs*[*symmetric*]  
**by** (*rule refl*)

**lemma** *length-uv: length uv = uv-length*  
**unfolding** *uv-def* **by** (*intro ensure-length-correct*)

**lemma** *pad-length-gt-0: pad-length > 0* **unfolding** *pad-length-def* **by** *simp*

**lemma** *scuv:*  
 $\text{length} (\text{subdivide } \text{pad-length } uv) = 2^{\wedge} (\text{oe-n} + 1)$   
 $x \in \text{set} (\text{subdivide } \text{pad-length } uv) \implies \text{length } x = \text{pad-length}$   
**using** *subdivide-correct*[*OF pad-length-gt-0*] *length-uv uv-length-def*  
**by** *auto*

**lemma** *length-c-dft-odds: length c-dft-odds = 2^{\wedge} (oe-n - 1)*  
**unfolding** *c-dft-odds-def*  
**using** *A.length-num-dft-odds B.length-num-dft-odds* **by** *simp*

**lemma** *length-c-diffs: length c-diffs = 2^{\wedge} (oe-n - 1)*  
**unfolding** *c-diffs-def*

**by** (*intro Fnr.length-iff length-c-dft-odds*)  
**lemma** *length- $\xi'$ : length  $\xi' = 2^{\wedge} (oe-n - 1)$*   
**unfolding**  *$\xi'$ -def* **by** (*simp add: length-c-diffs*)

**lemma** *length- $\xi$ : length  $\xi = 2^{\wedge} (oe-n - 1)$*   
**unfolding**  *$\xi$ -def* **by** (*simp add: length- $\xi'$* )

**lemma**  *$\gamma$ -nth:  $i < 4 \implies j < 2^{\wedge} (oe-n - 1) \implies \gamma ! i ! j = (\text{subdivide } \text{pad-length } uv) ! (i * 2^{\wedge} (oe-n - 1) + j)$  for  $i j$*   
**unfolding**  *$\gamma$ -def  $\gamma$ s-def*  
**apply** (*intro nth-nth-subdivide*[**where**  $k = 4$ ])  
**subgoal** **by** *simp*  
**subgoal**  
**apply** (*intro conjunct1*[*OF subdivide-correct*])  
**subgoal** **unfolding** *pad-length-def* **by** *simp*  
**subgoal** **using** *length-uv two-pow-Suc-oe-n-as-prod uv-length-def*  
**by** *simp*  
**done**

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

**using** *index-decomp  $\gamma$ -nth* **by** *algebra*  
**lemma** *sc $\gamma$ : length  $\gamma = 4 \bigwedge i. i < 4 \implies \text{length} (\gamma ! i) = 2^{\wedge} (oe-n - 1)$*

**proof** –  
**have** *1:  $(2::\text{nat})^{\wedge} (oe-n - 1) > 0$*  **by** *simp*  
**have** *2: length (subdivide pad-length uv) =  $2^{\wedge} (oe-n - 1) * 4$*   
**using** *two-pow-Suc-oe-n-as-prod scuv(1)* **by** *simp*

```

show length  $\gamma = 4 \wedge i. i < 4 \implies \text{length } (\gamma ! i) = 2^{\wedge} (oe-n - 1)$ 
  using subdivide-correct[OF 1 2]
  unfolding  $\gamma$ -def[symmetric]  $\gamma$ s-def[symmetric] by simp-all
qed
lemmas length- $\gamma = sc\gamma(1)$ 
lemmas length- $\gamma$ -i =  $sc\gamma(2)$ 

lemma length- $\gamma$ -nth:  $i < 4 \implies j < 2^{\wedge} (oe-n - 1) \implies \text{length } (\gamma ! i ! j) =$ 
  pad-length
  using scuv  $\gamma$ -nth index-comp[of i j] by fastforce

lemma length- $\eta$ : length  $\eta = 2^{\wedge} (oe-n - 1)$  unfolding  $\eta$ -def
  using length- $\gamma$ -i by (simp add: map4-as-map)
  lemma length-z: length  $z = 2^{\wedge} (oe-n - 1)$ 
  unfolding z-def using length- $\xi$  length- $\eta$  by simp
lemma nth-z:  $z ! j = \text{solve-special-residue-problem } n (\xi ! j) (\eta ! j)$  if  $j < 2^{\wedge} (oe-n - 1)$  for  $j$ 
  unfolding z-def using length-z that length- $\xi$  length- $\eta$  by simp
lemma length-z-filled: length z-filled =  $2^{\wedge} (oe-n - 1)$ 
  unfolding z-filled-def by (simp add: length-z)
lemma length-z-consts: length z-consts =  $2^{\wedge} (oe-n - 1)$ 
  unfolding z-consts-def by simp

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  $\text{int-lsbf-fermat.to-residue-ring } m (\text{schoenhage-strassen } m a b)$ 
    =  $\text{int-lsbf-fermat.to-residue-ring } m a \otimes_{\text{int-lsbf-fermat.Fn } m} \text{int-lsbf-fermat.to-residue-ring}$ 
     $m b \wedge \text{schoenhage-strassen } m a b \in \text{int-lsbf-fermat.fermat-non-unique-carrier } m$ 
  using assms
proof (induction m arbitrary: a b rule: less-induct)
  case (less m)
  then show ?case
  proof (cases m < 3)
  case True
  then have def:  $\text{schoenhage-strassen } m a b = \text{int-lsbf-fermat.from-nat-lsbf } m$ 
    ( $\text{grid-mul-nat } a b$ )
  by (simp add: schoenhage-strassen.simps)
  then have  $\text{int-lsbf-fermat.to-residue-ring } m (\text{schoenhage-strassen } m a b)$ 
    =  $\text{int-lsbf-fermat.to-residue-ring } m (\text{grid-mul-nat } a b)$ 
  using  $\text{int-lsbf-fermat.from-nat-lsbf-correct}$  by simp
  also have ... =  $\text{int } (\text{Nat-LSBF.to-nat } (\text{grid-mul-nat } a b)) \bmod \text{int } (2^{\wedge} (2^{\wedge} m$ 
    + 1))
  unfolding  $\text{int-lsbf-fermat.to-residue-ring.simps}$  by argo
  also have ... =  $\text{int } (\text{Nat-LSBF.to-nat } a * \text{Nat-LSBF.to-nat } b) \bmod \text{int } (2^{\wedge} (2^{\wedge} m$ 
    + 1))
  by (simp add: grid-mul-nat-correct)

```

**also have** ... = *int-lsbf-fermat.to-residue-ring*  $m$   $a \otimes_{\text{residue-ring}} (2 \wedge (2 \wedge m) + 1)$   
*int-lsbf-fermat.to-residue-ring*  $m$   $b$   
**apply** (*simp add: residue-ring-def int-lsbf-fermat.to-residue-ring.simps*)  
**using** *mod-mult-eq*  
**by** (*metis add.commute*)  
**finally show** *?thesis unfolding int-lsbf-fermat.Fn-def using def int-lsbf-fermat.from-nat-lsbf-correct(1)*  
**by** (*simp add: add.commute*)  
**next**  
**case** *False*

**interpret** *schoenhage-strassen-context*  $m$   $a$   $b$   
**using** *False less.premis by unfold-locales assumption+*

**have** *Fn-def'*:  $F_n = \text{residue-ring } (2 \wedge 2 \wedge n + 1)$   
**unfolding** *Fn-def* **by** (*simp add: int-ops add.commute*)  
**have** *fn-Fn[simp]*:  $\text{int-lsbf-fermat.Fn } n = F_n$   
**unfolding** *Fn-def int-lsbf-fermat.Fn-def* **by** (*rule refl*)

**from** *Fmr.res-carrier-eq* **have** *Fm-carrierI*:  $\bigwedge i. 0 \leq i \implies i < 2 \wedge 2 \wedge m + 1$   
 $\implies i \in \text{carrier } F_m$   
**by** *simp*

**define** *c'* **where**  $c' = \text{map } (\lambda j. \sum \sigma \leftarrow [0..<2 \wedge \text{oe-n}]. (\text{int } (\text{Nat-LSBF.to-nat } (A.\text{num-blocks } ! \sigma)) * \text{int } (\text{Nat-LSBF.to-nat } (B.\text{num-blocks } ! ((2 \wedge \text{oe-n} + j - \sigma) \bmod 2 \wedge \text{oe-n})))) [0..<2 \wedge \text{oe-n}]$   
**define** *z'* **where**  $z' = (\lambda j. \text{if } j < 2 \wedge (\text{oe-n} - 1) \text{ then } c' ! j - c' ! (2 \wedge (\text{oe-n} - 1) + j) + 2 \wedge (\text{oe-n} + 2 \wedge n) \text{ else } 2 \wedge (\text{oe-n} + 2 \wedge n))$   
**define** *z''* **where**  $z'' = (\lambda j. \text{if } j < 2 \wedge (\text{oe-n} - 1) \text{ then } c' ! j \oplus_{F_m} c' ! (2 \wedge (\text{oe-n} - 1) + j) \oplus_{F_m} 2 \lceil_{F_m} (\text{oe-n} + 2 \wedge n) \text{ else } 2 \lceil_{F_m} (\text{oe-n} + 2 \wedge n))$

**have** *length-c'*:  $\text{length } c' = 2 \wedge \text{oe-n}$  **unfolding** *c'-def* **by** *simp*  
**have** *c'-nth*:  $c' ! j = (\sum \sigma \leftarrow [0..<2 \wedge \text{oe-n}]. (\text{int } (\text{Nat-LSBF.to-nat } (A.\text{num-blocks } ! \sigma)) * \text{int } (\text{Nat-LSBF.to-nat } (B.\text{num-blocks } ! ((2 \wedge \text{oe-n} + j - \sigma) \bmod 2 \wedge \text{oe-n}))))$   
**if**  $j < 2 \wedge \text{oe-n}$  **for**  $j$   
**unfolding** *c'-def* **using** *that* **by** *simp*  
**have** *c'-nth-nat*:  $c' ! j = \text{int } (\sum \sigma \leftarrow [0..<2 \wedge \text{oe-n}]. (\text{Nat-LSBF.to-nat } (A.\text{num-blocks } ! \sigma) * \text{Nat-LSBF.to-nat } (B.\text{num-blocks } ! ((2 \wedge \text{oe-n} + j - \sigma) \bmod 2 \wedge \text{oe-n}))))$   
**if**  $j < 2 \wedge \text{oe-n}$  **for**  $j$   
**proof** –  
**have**  $c' ! j = (\sum \sigma \leftarrow [0..<2 \wedge \text{oe-n}]. (\text{int } (\text{Nat-LSBF.to-nat } (A.\text{num-blocks } ! \sigma) * \text{Nat-LSBF.to-nat } (B.\text{num-blocks } ! ((2 \wedge \text{oe-n} + j - \sigma) \bmod 2 \wedge \text{oe-n}))))$   
**unfolding** *c'-nth[OF that]* **by** *simp*  
**also have** ... =  $\text{int } (\sum \sigma \leftarrow [0..<2 \wedge \text{oe-n}]. \text{Nat-LSBF.to-nat } (A.\text{num-blocks } ! \sigma) * \text{Nat-LSBF.to-nat } (B.\text{num-blocks } ! ((2 \wedge \text{oe-n} + j - \sigma) \bmod 2 \wedge \text{oe-n})))$   
**by** (*intro sum-list-int[symmetric]*)  
**finally show**  $c' ! j = \dots$   
**qed**

**have**  $c'$ -lower-bound:  $c' ! j \geq 0$  **if**  $j < 2^{\wedge} oe-n$  **for**  $j$   
**unfolding**  $c'$ -nth[OF that]  
**apply** (intro sum-list-nonneg) **by** fastforce  
**have**  $c'$ -upper-bound:  $c' ! j < 2^{\wedge}(oe-n + 2^{\wedge} n)$  **if**  $j < 2^{\wedge} oe-n$  **for**  $j$   
**proof** –  
**have**  $Nat-LSBF.to-nat (A.num-blocks ! \sigma) * Nat-LSBF.to-nat (B.num-blocks ! ((2^{\wedge} oe-n + j - \sigma) \bmod 2^{\wedge} oe-n)) < 2^{\wedge} 2^{\wedge} n$   
**if**  $\sigma < 2^{\wedge} oe-n$  **for**  $\sigma$   
**proof** –  
**have**  $length (A.num-blocks ! \sigma) = 2^{\wedge}(n - 1)$  **using**  $A.length-nth-num-blocks$   
that **by** simp  
**then** **have**  $Nat-LSBF.to-nat (A.num-blocks ! \sigma) < 2^{\wedge} 2^{\wedge}(n - 1)$   
**using** to-nat-length-bound **by** metis  
**moreover** **have**  $length (B.num-blocks ! ((2^{\wedge} oe-n + j - \sigma) \bmod 2^{\wedge} oe-n)) = 2^{\wedge}(n - 1)$   
**using**  $B.length-nth-num-blocks$  **by** simp  
**then** **have**  $Nat-LSBF.to-nat (B.num-blocks ! ((2^{\wedge} oe-n + j - \sigma) \bmod 2^{\wedge} oe-n)) < 2^{\wedge} 2^{\wedge}(n - 1)$   
**using** to-nat-length-bound **by** metis  
**ultimately** **have**  $Nat-LSBF.to-nat (A.num-blocks ! \sigma) * Nat-LSBF.to-nat (B.num-blocks ! ((2^{\wedge} oe-n + j - \sigma) \bmod 2^{\wedge} oe-n)) < 2^{\wedge} 2^{\wedge}(n - 1) * 2^{\wedge} 2^{\wedge}(n - 1)$   
**by** (intro mult-strict-mono) simp-all  
**also** **have**  $\dots = 2^{\wedge} 2^{\wedge} n$  **using** n-gt-1  
**by** (simp add: power-add[symmetric] mult-2[symmetric] power-Suc[symmetric])  
**finally** **show** ?thesis .  
**qed**  
**then** **have**  $(\sum \sigma \leftarrow [0..<2^{\wedge} oe-n]. (Nat-LSBF.to-nat (A.num-blocks ! \sigma) * Nat-LSBF.to-nat (B.num-blocks ! ((2^{\wedge} oe-n + j - \sigma) \bmod 2^{\wedge} oe-n)))) < length [0..<2^{\wedge} oe-n] * 2^{\wedge} 2^{\wedge} n$   
**by** (intro sum-list-estimation-le) simp-all  
**then** **have**  $c' ! j < length [0..<2^{\wedge} oe-n] * 2^{\wedge} 2^{\wedge} n$   
**unfolding**  $c'$ -nth-nat[OF that]  
**using** nat-int-comparison(2)[symmetric] **by** blast  
**also** **have**  $\dots = 2^{\wedge}(oe-n + 2^{\wedge} n)$   
**by** (simp add: power-add)  
**finally** **show**  $c' ! j < 2^{\wedge}(oe-n + 2^{\wedge} n)$  .  
**qed**  
**have**  $c'$ -carrier:  $c' ! j \in carrier Fm$  **if**  $j < 2^{\wedge} oe-n$  **for**  $j$   
**proof** –  
**have**  $c' ! j < 2^{\wedge}(oe-n + 2^{\wedge} n)$  **using**  $c'$ -upper-bound[OF that] .  
**also** **have**  $\dots < 2^{\wedge}(oe-n + 1 + 2^{\wedge} n)$  **by** simp  
**also** **have**  $\dots \leq 2^{\wedge} 2^{\wedge} m$  **using** iffD2[OF zle-int two-pow-oe-n-m-bound-1]  
**by** simp  
**finally** **show** ?thesis  
**by** (simp add: Fm-def residue-ring-def  $c'$ -lower-bound[OF that])  
**qed**  
**have**  $c'$ -alt:  $c' ! j = (\sum \sigma \leftarrow [0..<2^{\wedge} oe-n]. \sum \varrho \leftarrow [0..<2^{\wedge} oe-n]. of-bool ([j = \sigma + \varrho] \bmod 2^{\wedge} oe-n)) * (int (Nat-LSBF.to-nat (A.num-blocks ! \sigma)) * int$

```

(Nat-LSBF.to-nat (B.num-blocks !  $\varrho$ )))
  if  $j < 2^{\wedge} oe-n$  for  $j$ 
  proof -
    have  $c' ! j = (\sum \sigma \leftarrow [0..<2^{\wedge} oe-n]. \text{int} (\text{Nat-LSBF.to-nat} (A.\text{num-blocks} ! \sigma))) * \text{int} (\text{Nat-LSBF.to-nat} (B.\text{num-blocks} ! ((2^{\wedge} oe-n + j - \sigma) \bmod 2^{\wedge} oe-n)))$ 
    using  $c'\text{-nth}[OF \text{that}]$  .
    also have  $\dots = (\sum \sigma \leftarrow [0..<2^{\wedge} oe-n]. \sum \varrho \leftarrow [0..<2^{\wedge} oe-n]. \text{of-bool} (\varrho = (2^{\wedge} oe-n + j - \sigma) \bmod 2^{\wedge} oe-n) * (\text{int} (\text{Nat-LSBF.to-nat} (A.\text{num-blocks} ! \sigma))) * \text{int} (\text{Nat-LSBF.to-nat} (B.\text{num-blocks} ! \varrho)))$ 
    by ( $\text{intro semiring-1-sum-list-eq of-bool-distinct-in}[\text{symmetric}] \text{simp-all}$ )
    also have  $\dots = (\sum \sigma \leftarrow [0..<2^{\wedge} oe-n]. \sum \varrho \leftarrow [0..<2^{\wedge} oe-n]. \text{of-bool} ([j = \sigma + \varrho] \bmod 2^{\wedge} oe-n) * (\text{int} (\text{Nat-LSBF.to-nat} (A.\text{num-blocks} ! \sigma))) * \text{int} (\text{Nat-LSBF.to-nat} (B.\text{num-blocks} ! \varrho)))$ 
    apply ( $\text{intro-cong} [\text{cong-tag-2} (*), \text{cong-tag-1 of-bool}] \text{more: semiring-1-sum-list-eq refl}$ )
    subgoal premises  $\text{prems}$  for  $\sigma \varrho$ 
      unfolding  $\text{cong-def}$ 
      using  $\text{cyclic-index-lemma}[of \sigma 2^{\wedge} oe-n \varrho j, \text{symmetric}] \text{that prems}$ 
      by  $\text{auto}$ 
    done
  finally show  $?thesis$  .
qed

have  $z'-z''$ :  $z' j = z'' j$  if  $j < 2^{\wedge} oe-n$  for  $j$ 
proof -
  have  $(2 :: \text{int})^{\wedge} (oe-n + 2^{\wedge} n) = \text{int} (2^{\wedge} (oe-n + 2^{\wedge} n))$  by  $\text{simp}$ 
  also have  $\dots = \text{int} (2^{\wedge} (oe-n + 2^{\wedge} n) \bmod \text{Fmr}.n)$ 
  apply ( $\text{intro arg-cong}[\text{where } f = \text{int}] \text{mod-less}[\text{symmetric}]$ )
  using  $oe-n\text{-m-bound-0}$ 
  by ( $\text{meson one-less-numeral-iff power-strict-increasing semiring-norm}(76) \text{trans-less-add1}$ )
  also have  $\dots = 2 [\wedge]_{\text{Fm}} (oe-n + 2^{\wedge} n)$ 
  by ( $\text{simp add: Fmr.pow-nat-eq zmod-int}$ )
  finally have  $\text{twopow: } (2 :: \text{int})^{\wedge} (oe-n + 2^{\wedge} n) = 2 [\wedge]_{\text{Fm}} (oe-n + 2^{\wedge} n)$  .
  show  $z' j = z'' j$ 
  proof ( $\text{cases } j < 2^{\wedge} (oe-n - 1)$ )
    case  $\text{True}$ 
    then have  $z' j = c' ! j - c' ! (2^{\wedge} (oe-n - 1) + j) + 2^{\wedge} (oe-n + 2^{\wedge} n)$ 
    unfolding  $z'\text{-def}$  by  $\text{simp}$ 
    moreover have  $\dots \geq 0 \dots < \text{Fmr}.n$ 
    subgoal using  $c'\text{-upper-bound}[of 2^{\wedge} (oe-n - 1) + j] c'\text{-lower-bound}[of j]$ 
      using  $\langle j < 2^{\wedge} oe-n \rangle \text{index-intros}(2)[of j] \text{True}$  by  $\text{simp}$ 
    subgoal
      proof -
        have  $c' ! j - c' ! (2^{\wedge} (oe-n - 1) + j) < 2^{\wedge} (oe-n + 2^{\wedge} n)$ 
        using  $c'\text{-upper-bound}[OF \langle j < 2^{\wedge} oe-n \rangle] c'\text{-lower-bound}[OF \text{index-intros}(2)[OF \langle j < 2^{\wedge} (oe-n - 1) \rangle]]$ 

```

```

    by simp
    then have  $c'! j - c'! (2^{\wedge}(oe-n - 1) + j) + 2^{\wedge}(oe-n + 2^{\wedge}n) < 2^{\wedge}(oe-n + 1 + 2^{\wedge}n)$ 
    by simp
    also have  $\dots < 2^{\wedge}2^{\wedge}m + 1$ 
    using two-pow-oe-n-m-bound-1 by simp
    finally show ?thesis by simp
  qed
done
ultimately have  $z' j = z' j \text{ mod } Fmr.n$  by simp
have  $z'' j = c'! j \ominus_{Fm} c'! (2^{\wedge}(oe-n - 1) + j) \oplus_{Fm} 2 [\wedge]_{Fm} (oe-n + 2^{\wedge}n)$ 
  unfolding  $z''\text{-def}$  using True by simp
  also have  $\dots = ((c'! j \ominus_{Fm} c'! (2^{\wedge}(oe-n - 1) + j)) + 2 [\wedge]_{Fm} (oe-n + 2^{\wedge}n)) \text{ mod } Fmr.n$ 
  by (intro Fmr.res-add-eq)
  also have  $\dots = ((c'! j \ominus_{Fm} c'! (2^{\wedge}(oe-n - 1) + j)) + 2^{\wedge}(oe-n + 2^{\wedge}n)) \text{ mod } Fmr.n$ 
  using  $\langle 2^{\wedge}(oe-n + 2^{\wedge}n) = 2 [\wedge]_{Fm} (oe-n + 2^{\wedge}n) \rangle$  by argo
  also have  $\dots = ((c'! j - c'! (2^{\wedge}(oe-n - 1) + j)) \text{ mod } Fmr.n + 2^{\wedge}(oe-n + 2^{\wedge}n)) \text{ mod } Fmr.n$ 
  using Fmr.residues-minus-eq by simp
  also have  $\dots = ((c'! j - c'! (2^{\wedge}(oe-n - 1) + j)) + 2^{\wedge}(oe-n + 2^{\wedge}n)) \text{ mod } Fmr.n$ 
  by (simp add: mod-add-left-eq)
  also have  $\dots = z' j \text{ mod } Fmr.n$ 
  unfolding  $\langle z' j = c'! j - c'! (2^{\wedge}(oe-n - 1) + j) + 2^{\wedge}(oe-n + 2^{\wedge}n) \rangle$  by (intro refl)
  finally show ?thesis using  $\langle z' j = z' j \text{ mod } Fmr.n \rangle$  by argo
next
case False
then show ?thesis unfolding  $z'\text{-def}$   $z''\text{-def}$  using two-pow by simp
qed
qed

have  $z'\text{-carrier}: z'' j \in \text{carrier } Fm \text{ if } j < 2^{\wedge}oe-n \text{ for } j$ 
  unfolding  $z''\text{-def}$ 
  apply (intro prop-iffI[where  $P = \lambda p. p \in \text{carrier } Fm$ ] Fmr.a-closed Fmr.minus-closed Fmr.nat-pow-closed  $c'\text{-carrier}$  Fmr.two-in-carrier])
  using index-intros by simp-all

have Fmr.to-residue-ring  $a \otimes_{Fm} Fmr.to-residue-ring b =$ 
   $(\bigoplus_{Fm} j \leftarrow [0..<2^{\wedge}oe-n]. (\bigoplus_{Fm} k \leftarrow [0..<2^{\wedge}oe-n].$ 
   $\text{map } (int \circ \text{Nat-LSBF.to-nat}) A.\text{num-blocks}! k \otimes_{Fm} \text{map } (int \circ \text{Nat-LSBF.to-nat})$ 
   $B.\text{num-blocks}! ((2^{\wedge}oe-n + j - k) \text{ mod } 2^{\wedge}oe-n)) \otimes_{Fm} ((2 [\wedge]_{Fm} (2::nat)^{\wedge}(n$ 
   $- 1)) [\wedge]_{Fm} j))$ 
  unfolding A.to-res-num B.to-res-num
  apply (intro Fmr.root-of-unity-power-sum-product)
  apply (intro Fmr.root-of-unity-power-sum-product two-pow-oe-n-root-of-unity-Fm
```

*A.num-blocks-carrier-Fm*)  
**subgoal for  $j$  using  $A.num-blocks-carrier-Fm[of\ j]$   $A.length-num-blocks$  by *simp***  
**subgoal for  $j$  using  $B.num-blocks-carrier-Fm[of\ j]$   $B.length-num-blocks$  by *simp***  
**done**  
**also have ... =  $(\bigoplus_{Fm} i \leftarrow [0..<2 \wedge oe-n]. (c' ! i) \otimes_{Fm} 2 [\lceil]_{Fm} (i * 2 \wedge (n - 1)))$**   
**apply (intro *Fmr.monoid-sum-list-cong arg-cong2*[where  $f = (\otimes_{Fm})$ ])**  
**subgoal premises *prems* for  $j$**   
**proof –**  
**from *prems* have  $j < 2 \wedge oe-n$  by *simp***  
**have  $(\bigoplus_{Fm} k \leftarrow [0..<2 \wedge oe-n]. \text{map } (int \circ Nat-LSBF.to-nat) A.num-blocks ! k \otimes_{Fm} \text{map } (int \circ Nat-LSBF.to-nat) B.num-blocks ! ((2 \wedge oe-n + j - k) \bmod 2 \wedge oe-n)) = (\bigoplus_{Fm} k \leftarrow [0..<2 \wedge oe-n]. (\text{map } (int \circ Nat-LSBF.to-nat) A.num-blocks ! k * \text{map } (int \circ Nat-LSBF.to-nat) B.num-blocks ! ((2 \wedge oe-n + j - k) \bmod 2 \wedge oe-n)) \bmod Fmr.n)$**   
**by (intro *Fmr.monoid-sum-list-cong Fmr.res-mult-eq*)**  
**also have ... =  $(\sum k \leftarrow [0..<2 \wedge oe-n]. (\text{map } (int \circ Nat-LSBF.to-nat) A.num-blocks ! k * \text{map } (int \circ Nat-LSBF.to-nat) B.num-blocks ! ((2 \wedge oe-n + j - k) \bmod 2 \wedge oe-n))) \bmod Fmr.n$**   
**by (intro *Fmr.monoid-sum-list-eq-sum-list'*)**  
**also have ... =  $c' ! j \bmod Fmr.n$**   
**unfolding  $c'-nth[OF \langle j < 2 \wedge oe-n \rangle]$**   
**apply (intro-cong [cong-tag-2 (mod)] more: *refl semiring-1-sum-list-eq*)**  
**using  $A.length-num-blocks B.length-num-blocks$  by *simp-all***  
**also have ... =  $c' ! j$**   
**using *Fmr.carrier-mod-eq*[ $OF\ c'-carrier[OF \langle j < 2 \wedge oe-n \rangle]$ ].**  
**finally show ?thesis .**  
**qed**  
**subgoal for  $j$  unfolding *Fmr.nat-pow-pow*[ $OF\ Fmr.two-in-carrier$ ]**  
**by (intro *arg-cong2*[where  $f = ([\lceil]_{Fm})$ ] *refl mult.commute*)**  
**done**  
**also have ... =  $(\bigoplus_{Fm} i \leftarrow [0..<2 \wedge oe-n]. (z' i) \otimes_{Fm} 2 [\lceil]_{Fm} (i * 2 \wedge (n - 1)))$**   
**proof –**  
**have  $(\bigoplus_{Fm} i \leftarrow [0..<2 \wedge oe-n]. (z' i) \otimes_{Fm} 2 [\lceil]_{Fm} (i * 2 \wedge (n - 1))) = (\bigoplus_{Fm} i \leftarrow [0..<2 \wedge oe-n]. (z'' i) \otimes_{Fm} 2 [\lceil]_{Fm} (i * 2 \wedge (n - 1)))$**   
**apply (intro-cong [cong-tag-2 ( $\otimes_{Fm}$ )] more: *Fmr.monoid-sum-list-cong refl*)**  
**using  $z'-z''$  by *simp***  
**also have ... =  $(\bigoplus_{Fm} i \leftarrow [0..<2 \wedge (oe-n - 1)]. (z'' i) \otimes_{Fm} 2 [\lceil]_{Fm} (i * 2 \wedge (n - 1)))$**   
 $\oplus_{Fm} 2 [\lceil]_{Fm} ((2::nat) \wedge (oe-n - 1) * 2 \wedge (n - 1)) \otimes_{Fm} (\bigoplus_{Fm} i \leftarrow [0..<2 \wedge (oe-n - 1)]. (z'' (2 \wedge (oe-n - 1) + i)) \otimes_{Fm} 2 [\lceil]_{Fm} (i * 2 \wedge (n - 1)))$

**apply** (*intro Fmr.monoid-pow-sum-split two-pow-oe-n-as-halves[symmetric]*  
*z'-carrier Fmr.two-in-carrier*)  
**by** *assumption*  
**also have** ... =  $(\bigoplus_{Fm} i \leftarrow [0..<2 \wedge (oe-n - 1)]. (c'! i \ominus_{Fm} c'! (2 \wedge (oe-n - 1) + i) \oplus_{Fm} 2 [\ulcorner]_{Fm} (oe-n + 2 \wedge n) \otimes_{Fm} 2 [\ulcorner]_{Fm} (i * 2 \wedge (n - 1))) \oplus_{Fm} 2 [\ulcorner]_{Fm} ((2::nat) \wedge (oe-n - 1) * 2 \wedge (n - 1)) \otimes_{Fm} (\bigoplus_{Fm} i \leftarrow [0..<2 \wedge (oe-n - 1)]. 2 [\ulcorner]_{Fm} (oe-n + 2 \wedge n) \otimes_{Fm} 2 [\ulcorner]_{Fm} (i * 2 \wedge (n - 1)))$   
**apply** (*intro-cong [cong-tag-2 ( $\oplus_{Fm}$ ), cong-tag-2 ( $\otimes_{Fm}$ )] more: Fmr.monoid-sum-list-cong refl*)  
**by** (*simp-all add: z''-def*)  
**also have** ... =  $(\bigoplus_{Fm} i \leftarrow [0..<2 \wedge (oe-n - 1)]. (c'! i \ominus_{Fm} c'! (2 \wedge (oe-n - 1) + i) \oplus_{Fm} 2 [\ulcorner]_{Fm} (oe-n + 2 \wedge n) \otimes_{Fm} 2 [\ulcorner]_{Fm} (i * 2 \wedge (n - 1))) \oplus_{Fm} 2 [\ulcorner]_{Fm} ((2::nat) \wedge m) \otimes_{Fm} (\bigoplus_{Fm} i \leftarrow [0..<2 \wedge (oe-n - 1)]. 2 [\ulcorner]_{Fm} (oe-n + 2 \wedge n) \otimes_{Fm} 2 [\ulcorner]_{Fm} (i * 2 \wedge (n - 1)))$   
**apply** (*intro-cong [cong-tag-2 ( $\oplus_{Fm}$ ), cong-tag-2 ( $\otimes_{Fm}$ ), cong-tag-2 ( $[\ulcorner]_{Fm}$ )] more: refl*)  
**using** *two-pow-m0-as-prod by simp*  
**also have** ... =  $(\bigoplus_{Fm} i \leftarrow [0..<2 \wedge (oe-n - 1)]. (c'! i \ominus_{Fm} (c'! (2 \wedge (oe-n - 1) + i) \ominus_{Fm} 2 [\ulcorner]_{Fm} (oe-n + 2 \wedge n))) \otimes_{Fm} 2 [\ulcorner]_{Fm} (i * 2 \wedge (n - 1))) \oplus_{Fm} 2 [\ulcorner]_{Fm} ((2::nat) \wedge m) \otimes_{Fm} (\bigoplus_{Fm} i \leftarrow [0..<2 \wedge (oe-n - 1)]. 2 [\ulcorner]_{Fm} (oe-n + 2 \wedge n) \otimes_{Fm} 2 [\ulcorner]_{Fm} (i * 2 \wedge (n - 1)))$   
**apply** (*intro-cong [cong-tag-2 ( $\oplus_{Fm}$ ), cong-tag-2 ( $\otimes_{Fm}$ )] more: refl Fmr.monoid-sum-list-cong Fmr.diff-diff[symmetric] Fmr.nat-pow-closed c'-carrier Fmr.two-in-carrier*)  
**using** *index-intros by simp-all*  
**also have** ... =  $(\bigoplus_{Fm} i \leftarrow [0..<2 \wedge (oe-n - 1)]. c'! i \otimes_{Fm} 2 [\ulcorner]_{Fm} (i * 2 \wedge (n - 1))) \ominus_{Fm} (\bigoplus_{Fm} i \leftarrow [0..<2 \wedge (oe-n - 1)]. (c'! (2 \wedge (oe-n - 1) + i) \ominus_{Fm} 2 [\ulcorner]_{Fm} (oe-n + 2 \wedge n) \otimes_{Fm} 2 [\ulcorner]_{Fm} (i * 2 \wedge (n - 1))) \oplus_{Fm} 2 [\ulcorner]_{Fm} ((2::nat) \wedge m) \otimes_{Fm} (\bigoplus_{Fm} i \leftarrow [0..<2 \wedge (oe-n - 1)]. 2 [\ulcorner]_{Fm} (oe-n + 2 \wedge n) \otimes_{Fm} 2 [\ulcorner]_{Fm} (i * 2 \wedge (n - 1)))$   
**apply** (*intro-cong [cong-tag-2 ( $\oplus_{Fm}$ )] more: refl Fmr.monoid-pow-sum-diff'[symmetric] Fmr.minus-closed Fmr.nat-pow-closed c'-carrier Fmr.two-in-carrier*)  
**using** *index-intros by simp-all*  
**also have** ... =  $(\bigoplus_{Fm} i \leftarrow [0..<2 \wedge (oe-n - 1)]. c'! i \otimes_{Fm} 2 [\ulcorner]_{Fm} (i * 2 \wedge (n - 1))) \oplus_{Fm} (\ominus_{Fm} \mathbf{1}_{Fm}) \otimes_{Fm} (\bigoplus_{Fm} i \leftarrow [0..<2 \wedge (oe-n - 1)]. (c'! (2 \wedge (oe-n - 1) + i) \ominus_{Fm} 2 [\ulcorner]_{Fm} (oe-n + 2 \wedge n) \otimes_{Fm} 2 [\ulcorner]_{Fm} (i * 2 \wedge (n - 1))) \oplus_{Fm} 2 [\ulcorner]_{Fm} ((2::nat) \wedge m) \otimes_{Fm} (\bigoplus_{Fm} i \leftarrow [0..<2 \wedge (oe-n - 1)]. 2 [\ulcorner]_{Fm} (oe-n + 2 \wedge n) \otimes_{Fm} 2 [\ulcorner]_{Fm} (i * 2 \wedge (n - 1)))$

**apply** (*intro-cong* [*cong-tag-2* ( $\oplus_{Fm}$ )] *more: refl Fmr.diff-eq-add-mult-one Fmr.monoid-sum-list-closed Fmr.m-closed Fmr.nat-pow-closed Fmr.minus-closed c'-carrier Fmr.two-in-carrier*)

**using** *index-intros by simp-all*

**also have** ... = ( $\oplus_{Fm} i \leftarrow [0..<2 \wedge (oe-n - 1)]. c' ! i \otimes_{Fm} 2 [\ulcorner]_{Fm} (i * 2 \wedge (n - 1))$ )

$\oplus_{Fm} (2 [\ulcorner]_{Fm} ((2::nat) \wedge m)) \otimes_{Fm}$

$(\oplus_{Fm} i \leftarrow [0..<2 \wedge (oe-n - 1)].$

$(c' ! (2 \wedge (oe-n - 1) + i) \ominus_{Fm} 2 [\ulcorner]_{Fm} (oe-n + 2 \wedge n)) \otimes_{Fm} 2 [\ulcorner]_{Fm} (i * 2 \wedge (n - 1))$ )

$\oplus_{Fm} 2 [\ulcorner]_{Fm} ((2::nat) \wedge m) \otimes_{Fm}$

$(\oplus_{Fm} i \leftarrow [0..<2 \wedge (oe-n - 1)].$

$2 [\ulcorner]_{Fm} (oe-n + 2 \wedge n) \otimes_{Fm} 2 [\ulcorner]_{Fm} (i * 2 \wedge (n - 1))$ )

**using** *Fmr.two-pow-half-carrier-length-residue-ring by argo*

**also have** ... = ( $\oplus_{Fm} i \leftarrow [0..<2 \wedge (oe-n - 1)]. c' ! i \otimes_{Fm} 2 [\ulcorner]_{Fm} (i * 2 \wedge (n - 1))$ )

$\oplus_{Fm} ((2 [\ulcorner]_{Fm} ((2::nat) \wedge m)) \otimes_{Fm}$

$(\oplus_{Fm} i \leftarrow [0..<2 \wedge (oe-n - 1)].$

$(c' ! (2 \wedge (oe-n - 1) + i) \ominus_{Fm} 2 [\ulcorner]_{Fm} (oe-n + 2 \wedge n)) \otimes_{Fm} 2 [\ulcorner]_{Fm} (i * 2 \wedge (n - 1))$ )

$\oplus_{Fm} 2 [\ulcorner]_{Fm} ((2::nat) \wedge m) \otimes_{Fm}$

$(\oplus_{Fm} i \leftarrow [0..<2 \wedge (oe-n - 1)].$

$2 [\ulcorner]_{Fm} (oe-n + 2 \wedge n) \otimes_{Fm} 2 [\ulcorner]_{Fm} (i * 2 \wedge (n - 1))$ )

**apply** (*intro Fmr.a-assoc Fmr.m-closed Fmr.nat-pow-closed Fmr.monoid-sum-list-closed Fmr.minus-closed c'-carrier Fmr.two-in-carrier*)

**using** *index-intros by simp-all*

**also have** ... = ( $\oplus_{Fm} i \leftarrow [0..<2 \wedge (oe-n - 1)]. c' ! i \otimes_{Fm} 2 [\ulcorner]_{Fm} (i * 2 \wedge (n - 1))$ )

$\oplus_{Fm} ((2 [\ulcorner]_{Fm} ((2::nat) \wedge m)) \otimes_{Fm}$

$((\oplus_{Fm} i \leftarrow [0..<2 \wedge (oe-n - 1)].$

$(c' ! (2 \wedge (oe-n - 1) + i) \ominus_{Fm} 2 [\ulcorner]_{Fm} (oe-n + 2 \wedge n)) \otimes_{Fm} 2 [\ulcorner]_{Fm} (i * 2 \wedge (n - 1))$ )

$\oplus_{Fm}$

$(\oplus_{Fm} i \leftarrow [0..<2 \wedge (oe-n - 1)].$

$2 [\ulcorner]_{Fm} (oe-n + 2 \wedge n) \otimes_{Fm} 2 [\ulcorner]_{Fm} (i * 2 \wedge (n - 1))))$ )

**apply** (*intro-cong* [*cong-tag-2* ( $\oplus_{Fm}$ )] *more: refl Fmr.r-distr[symmetric] Fmr.monoid-sum-list-closed Fmr.m-closed Fmr.nat-pow-closed c'-carrier Fmr.two-in-carrier Fmr.minus-closed*)

**using** *index-intros by simp*

**also have** ... = ( $\oplus_{Fm} i \leftarrow [0..<2 \wedge (oe-n - 1)]. c' ! i \otimes_{Fm} 2 [\ulcorner]_{Fm} (i * 2 \wedge (n - 1))$ )

$\oplus_{Fm} ((2 [\ulcorner]_{Fm} ((2::nat) \wedge m)) \otimes_{Fm}$

$(\oplus_{Fm} i \leftarrow [0..<2 \wedge (oe-n - 1)].$

$(c' ! (2 \wedge (oe-n - 1) + i) \ominus_{Fm} 2 [\ulcorner]_{Fm} (oe-n + 2 \wedge n) \oplus_{Fm} 2 [\ulcorner]_{Fm} (oe-n + 2 \wedge n)) \otimes_{Fm} 2 [\ulcorner]_{Fm} (i * 2 \wedge (n - 1))$ )

**apply** (*intro-cong* [*cong-tag-2* ( $\oplus_{Fm}$ ), *cong-tag-2* ( $\otimes_{Fm}$ )] *more: refl Fmr.monoid-pow-sum-add' Fmr.minus-closed Fmr.nat-pow-closed c'-carrier Fmr.two-in-carrier*)

**using** *index-intros by simp*

**also have** ... = ( $\oplus_{Fm} i \leftarrow [0..<2 \wedge (oe-n - 1)]. c' ! i \otimes_{Fm} 2 [\ulcorner]_{Fm} (i * 2 \wedge (n - 1))$ )

```

 $\wedge (n - 1)))$ 
   $\oplus_{Fm} ((2 [\wedge]_{Fm} ((2::nat) \wedge m)) \otimes_{Fm}$ 
   $(\oplus_{Fm} i \leftarrow [0..<2 \wedge (oe-n - 1)].$ 
   $(c'! (2 \wedge (oe-n - 1) + i)) \otimes_{Fm} 2 [\wedge]_{Fm} (i * 2 \wedge (n - 1))))$ 
  apply (intro-cong [cong-tag-2 ( $\oplus_{Fm}$ ), cong-tag-2 ( $\otimes_{Fm}$ )] more: refl Fmr.monoid-sum-list-cong
  Fmr.minus-cancel Fmr.nat-pow-closed c'-carrier Fmr.two-in-carrier)
  using index-intros by simp
  also have ... = ( $\oplus_{Fm} i \leftarrow [0..<2 \wedge (oe-n - 1)]. c'! i \otimes_{Fm} 2 [\wedge]_{Fm} (i * 2$ 
 $\wedge (n - 1)))$ 
   $\oplus_{Fm} ((2 [\wedge]_{Fm} ((2::nat) \wedge (oe-n - 1) * 2 \wedge (n - 1))) \otimes_{Fm}$ 
   $(\oplus_{Fm} i \leftarrow [0..<2 \wedge (oe-n - 1)].$ 
   $(c'! (2 \wedge (oe-n - 1) + i)) \otimes_{Fm} 2 [\wedge]_{Fm} (i * 2 \wedge (n - 1))))$ 
  using two-pow-m0-as-prod by (simp add: mult.commute)
  also have ... = ( $\oplus_{Fm} i \leftarrow [0..<2 \wedge oe-n]. c'! i \otimes_{Fm} 2 [\wedge]_{Fm} (i * 2 \wedge (n -$ 
  1)))
  apply (intro Fmr.monoid-pow-sum-split[symmetric] two-pow-oe-n-as-halves[symmetric]
  c'-carrier Fmr.two-in-carrier)
  by assumption
  finally show ?thesis by argo
qed
finally have result0: Fmr.to-residue-ring a  $\otimes_{Fm}$  Fmr.to-residue-ring b
  = ( $\oplus_{Fm} i \leftarrow [0..<2 \wedge oe-n]. (z' i) \otimes_{Fm} 2 [\wedge]_{Fm} (i * 2 \wedge (n - 1)))$  .

have Nat-LSBF.to-nat uv = Nat-LSBF.to-nat A.num-Zn-pad * Nat-LSBF.to-nat
  B.num-Zn-pad
proof (cases length (karatsuba-mul-nat A.num-Zn-pad B.num-Zn-pad)  $\leq$  uv-length)
  case True
  have uv = fill uv-length (karatsuba-mul-nat A.num-Zn-pad B.num-Zn-pad)
  unfolding uv-def ensure-length-def uv-unpadded-def
  apply (intro take-id)
  using True unfolding length-fill' by linarith
  then have Nat-LSBF.to-nat uv = Nat-LSBF.to-nat (karatsuba-mul-nat
  A.num-Zn-pad B.num-Zn-pad) by simp
  also have ... = Nat-LSBF.to-nat A.num-Zn-pad * Nat-LSBF.to-nat B.num-Zn-pad
by (rule karatsuba-mul-nat-correct)
  finally show ?thesis .
next
  case False
  define uv' where uv' = take uv-length (karatsuba-mul-nat A.num-Zn-pad
  B.num-Zn-pad)
  define f where f = drop uv-length (karatsuba-mul-nat A.num-Zn-pad B.num-Zn-pad)
  have karatsuba-mul-nat A.num-Zn-pad B.num-Zn-pad = uv' @ f
  unfolding uv'-def f-def
  by (rule append-take-drop-id[symmetric])
  from uv'-def False have length uv' = uv-length
  unfolding uv'-def length-take using False
  by (intro min.absorb2) linarith

```

```

have f = replicate (length f) False
proof (rule ccontr)
  assume f ≠ replicate (length f) False
  with list-is-replicate-iff obtain i where i < length f f ! i = True by force
  define j where j = uv-length + i
  then have karatsuba-mul-nat A.num-Zn-pad B.num-Zn-pad ! j = True
  using ⟨karatsuba-mul-nat A.num-Zn-pad B.num-Zn-pad = uv' @ f⟩ ⟨length
uv' = uv-length⟩
  using ⟨f ! i = True⟩ by (metis nth-append-length-plus)
  then have Nat-LSBF.to-nat (karatsuba-mul-nat A.num-Zn-pad B.num-Zn-pad)
≥ 2 ^ j
  apply (intro to-nat-nth-True-bound)
  subgoal using j-def ⟨i < length f⟩ ⟨length uv' = uv-length⟩
  using ⟨karatsuba-mul-nat A.num-Zn-pad B.num-Zn-pad = uv' @ f⟩ by
simp
  subgoal .
  done
  moreover have (2::nat) ^ j ≥ 2 ^ uv-length
  apply (intro power-increasing) using j-def by simp-all
  ultimately have G: Nat-LSBF.to-nat (karatsuba-mul-nat A.num-Zn-pad
B.num-Zn-pad) ≥ 2 ^ uv-length
  by linarith
  have Nat-LSBF.to-nat (karatsuba-mul-nat A.num-Zn-pad B.num-Zn-pad)
= Nat-LSBF.to-nat A.num-Zn-pad * Nat-LSBF.to-nat B.num-Zn-pad
  by (intro karatsuba-mul-nat-correct)
  also have ... < 2 ^ length A.num-Zn-pad * 2 ^ length B.num-Zn-pad
  by (intro mult-strict-mono to-nat-length-bound) simp-all
  also have ... = 2 ^ (length A.num-Zn-pad + length B.num-Zn-pad)
  by (simp only: power-add)
  also have ... = 2 ^ (pad-length * 2 ^ oe-n + pad-length * 2 ^ oe-n)
  using A.length-num-Zn-pad B.length-num-Zn-pad
  by simp
  also have ... = 2 ^ uv-length
  unfolding uv-length-def
  by (intro arg-cong[where f = power 2]) simp
  finally show False using G by linarith
qed
then have Nat-LSBF.to-nat (karatsuba-mul-nat A.num-Zn-pad B.num-Zn-pad)
= Nat-LSBF.to-nat uv'
  using ⟨karatsuba-mul-nat A.num-Zn-pad B.num-Zn-pad = uv' @ f⟩
  using to-nat-app-replicate by metis
moreover have uv' = uv
  unfolding uv'-def uv-def ensure-length-def uv-unpadded-def
  apply (intro arg-cong2[where f = take] refl fill-id[symmetric])
  using False by linarith
ultimately show ?thesis unfolding karatsuba-mul-nat-correct by simp
qed

define γ' where γ' ≡ λτ. (∑ σ ← [0..<2 ^ oe-n]. ∑ ρ ← [0..<2 ^ oe-n]. of-bool

```

$(\tau = \sigma + \varrho) * (\text{Nat-LSBF.to-nat } (A.\text{num-Zn } ! \sigma) * \text{Nat-LSBF.to-nat } (B.\text{num-Zn } ! \varrho))$

```

have to-nat- $\gamma$ : Nat-LSBF.to-nat ( $\gamma$  !  $i$  !  $j$ ) =  $\gamma'$  ( $i * 2^{\wedge}(\text{oe-n} - 1) + j$ )
if  $i < 4 j < 2^{\wedge}(\text{oe-n} - 1)$  for  $i j$ 
proof -
  have Nat-LSBF.to-nat  $uv = (\sum j \leftarrow [0..<2^{\wedge}(\text{oe-n} + 1)]. \text{Nat-LSBF.to-nat}$ 
    ( $\text{subdivide pad-length } uv$  !  $j$ ) *  $2^{\wedge}(j * \text{pad-length})$ )
    apply (intro to-nat-subdivide pad-length-gt-0)
    unfolding length-uv uv-length-def by (rule refl)
  also have ... =  $(\sum j \leftarrow [0..<2^{\wedge}(\text{oe-n} + 1)].$ 
    Nat-LSBF.to-nat ( $\gamma$  !  $(j \text{ div } 2^{\wedge}(\text{oe-n} - 1))$  !  $(j \text{ mod } 2^{\wedge}(\text{oe-n} - 1))$ )
    *  $2^{\wedge}(j * \text{pad-length})$ )
    apply (intro-cong [cong-tag-2 (*), cong-tag-1 Nat-LSBF.to-nat] more: semir-
      ing-1-sum-list-eq refl)
    apply (intro  $\gamma$ -nth'[symmetric]) by simp
  finally have 1: Nat-LSBF.to-nat  $uv = \dots$  .

  let ?exp =  $\lambda i. 2^{\wedge}(i * \text{pad-length})$ 
  let ?a =  $\lambda i. \text{Nat-LSBF.to-nat } (A.\text{num-Zn } ! i)$ 
  let ?b =  $\lambda i. \text{Nat-LSBF.to-nat } (B.\text{num-Zn } ! i)$ 
  from A.to-nat-num-Zn-pad B.to-nat-num-Zn-pad
  have Nat-LSBF.to-nat  $uv =$ 
     $(\sum i \leftarrow [0..<2^{\wedge} \text{oe-n}]. ?a i * ?exp i) *$ 
     $(\sum j \leftarrow [0..<2^{\wedge} \text{oe-n}]. ?b j * ?exp j)$ 
  using  $\langle \text{Nat-LSBF.to-nat } uv = \text{Nat-LSBF.to-nat } A.\text{num-Zn-pad} * \text{Nat-LSBF.to-nat}$ 
     $B.\text{num-Zn-pad} \rangle$ 
    by argo
  also have ... =  $(\sum i \leftarrow [0..<2^{\wedge} \text{oe-n}]. \sum j \leftarrow [0..<2^{\wedge} \text{oe-n}]. (?a i * ?exp$ 
     $i) * (?b j * ?exp j))$ 
    by (rule sum-list-mult-sum-list)
  also have ... =  $(\sum i \leftarrow [0..<2^{\wedge} \text{oe-n}]. \sum j \leftarrow [0..<2^{\wedge} \text{oe-n}]. (?a i * ?b j) *$ 
     $?exp (i + j))$ 
    by (intro sum-list-eq; simp add: algebra-simps power-add)
  also have ... =  $(\sum i \leftarrow [0..<2^{\wedge} \text{oe-n}]. \sum j \leftarrow [0..<2^{\wedge} \text{oe-n}]. \sum k \leftarrow [0..<2^{\wedge}$ 
     $(\text{oe-n} + 1) - 1].$ 
     $\text{of-bool } (k = i + j) * ((?a i * ?b j) * ?exp (i + j)))$ 
    by (intro sum-list-eq of-bool-distinct-in[symmetric]; simp)
  also have ... =  $(\sum i \leftarrow [0..<2^{\wedge} \text{oe-n}]. \sum j \leftarrow [0..<2^{\wedge} \text{oe-n}]. \sum k \leftarrow [0..<2^{\wedge}$ 
     $(\text{oe-n} + 1) - 1].$ 
     $\text{of-bool } (k = i + j) * ((?a i * ?b j) * ?exp k))$ 
    by (intro sum-list-eq[where xs = [0..<2^{\wedge} \text{oe-n}] of-bool-var-swap[symmetric]])
  also have ... =  $(\sum k \leftarrow [0..<2^{\wedge}(\text{oe-n} + 1) - 1]. \sum i \leftarrow [0..<2^{\wedge} \text{oe-n}].$ 
     $\sum j \leftarrow [0..<2^{\wedge} \text{oe-n}].$ 
     $\text{of-bool } (k = i + j) * ((?a i * ?b j) * ?exp k))$ 
    by (simp only: sum-swap[where ys = [0..<2^{\wedge}(\text{oe-n} + 1) - 1]])
  also have ... =  $(\sum k \leftarrow [0..<2^{\wedge}(\text{oe-n} + 1) - 1]. \gamma' k * ?exp k)$ 
    apply (unfold  $\gamma'$ -def)
    apply (intro sum-list-eq)

```

```

apply (unfold sum-list-mult-const[symmetric])
apply (intro sum-list-eq)
apply (simp only: algebra-simps)
done
also have ... =  $(\sum k \leftarrow [0..<2^{\wedge}(oe-n + 1)]. \gamma' k * 2^{\wedge}(k * pad-length))$ 
proof -
  have  $(\sum k \leftarrow [0..<2^{\wedge}(oe-n + 1)]. \gamma' k * 2^{\wedge}(k * pad-length)) =$ 
 $(\sum k \leftarrow [0..<2^{\wedge}(oe-n + 1) - 1]. \gamma' k * 2^{\wedge}(k * pad-length)) + \gamma' (2^{\wedge}(oe-n + 1) - 1) * 2^{\wedge}((2^{\wedge}(oe-n + 1) - 1) * pad-length)$ 
  apply (intro sum-list-split-Suc) by simp
  also have  $\gamma' (2^{\wedge}(oe-n + 1) - 1) = (\sum i \leftarrow [0..<2^{\wedge}oe-n]. \sum j \leftarrow [0..<2^{\wedge}oe-n]. 0)$ 
  unfolding  $\gamma'$ -def
  proof (intro semiring-1-sum-list-eq)
    fix  $i j :: nat$ 
    assume  $i \in set [0..<2^{\wedge}oe-n] j \in set [0..<2^{\wedge}oe-n]$ 
    then have  $i + j \leq (2^{\wedge}oe-n - 1) + (2^{\wedge}oe-n - 1)$  by simp
    also have ... =  $2^{\wedge}(oe-n + 1) - 2$  by simp
    also have ... <  $2^{\wedge}(oe-n + 1) - 1$  using oe-n-gt-0
    by (meson diff-less-mono2 one-less-numeral-iff one-less-power pos-add-strict
semiring-norm(76) zero-less-one)
    finally have  $2^{\wedge}(oe-n + 1) - 1 \neq i + j$  by simp
    then have of-bool  $(2^{\wedge}(oe-n + 1) - 1 = i + j) = 0$  by simp
    then show of-bool  $(2^{\wedge}(oe-n + 1) - 1 = i + j) * (Nat-LSBF.to-nat$ 
 $(A.num-Zn ! i) * Nat-LSBF.to-nat (B.num-Zn ! j)) = 0$ 
    using mult-zero-left by metis
    qed
  also have ... = 0 by simp
  finally show ?thesis by simp
qed
finally have Nat-LSBF.to-nat  $uv = \dots$  .
with 1 have 2:  $(\sum j \leftarrow [0..<2^{\wedge}(oe-n + 1)].$ 
 $Nat-LSBF.to-nat (\gamma ! (j \text{ div } 2^{\wedge}(oe-n - 1)) ! (j \text{ mod } 2^{\wedge}(oe-n - 1)))$ 
 $* 2^{\wedge}(j * pad-length)) = \dots$  by argo
have  $\bigwedge j. j < 2^{\wedge}(oe-n + 1) \implies$ 
 $Nat-LSBF.to-nat (\gamma ! (j \text{ div } 2^{\wedge}(oe-n - 1)) ! (j \text{ mod } 2^{\wedge}(oe-n - 1))) = \gamma' j$ 
apply (intro power-sum-nat-eq[where  $n = 2^{\wedge}(oe-n + 1)$  and  $g = \gamma'$  and
 $x = 2$  and  $c = pad-length$ ])
subgoal by simp
subgoal by (rule pad-length-gt-0)
subgoal for  $i j$ 
proof -
  assume  $j < 2^{\wedge}(oe-n + 1)$ 
  then have  $Nat-LSBF.to-nat (\gamma ! (j \text{ div } 2^{\wedge}(oe-n - 1)) ! (j \text{ mod } 2^{\wedge}(oe-n - 1))) = Nat-LSBF.to-nat (subdivide pad-length uv ! j)$ 
  apply (intro arg-cong[where  $f = Nat-LSBF.to-nat$ ]  $\gamma$ -nth') .
  also have ... <  $2^{\wedge}(length (subdivide pad-length uv ! j))$ 
  by (intro to-nat-length-bound)
  also have ... =  $2^{\wedge}pad-length$ 

```

```

    apply (intro arg-cong[where f = power 2] scuv(2) nth-mem)
    using ⟨j < 2 ^ (oe-n + 1)⟩ scuv(1) by argo
    finally show Nat-LSBF.to-nat (γ ! (j div 2 ^ (oe-n - 1)) ! (j mod 2 ^
(oe-n - 1))) < 2 ^ pad-length .
  qed
  subgoal for i j
  proof -
    have γ' j = (∑ σ←[0..<2 ^ oe-n]. ∑ ρ←[0..<2 ^ oe-n]. of-bool (j = σ +
ρ) * (Nat-LSBF.to-nat (A.num-Zn ! σ) * Nat-LSBF.to-nat (B.num-Zn ! ρ)))
    unfolding γ'-def by argo
    also have ... ≤ (∑ σ←[0..<2 ^ oe-n]. ∑ ρ←[0..<2 ^ oe-n]. of-bool (j =
σ + ρ) * ((2 ^ (n + 2) - 1) * (2 ^ (n + 2) - 1)))
    apply (intro sum-list-mono mult-le-mono2 mult-le-mono)
    subgoal for σ ρ unfolding A.num-Zn-def
    using A.length-num-blocks to-nat-length-upper-bound[of map Znr.reduce
A.num-blocks ! σ] Znr.length-reduce
    by simp
    subgoal for σ ρ unfolding B.num-Zn-def
    using B.length-num-blocks to-nat-length-upper-bound[of map Znr.reduce
B.num-blocks ! ρ] Znr.length-reduce
    by simp
  done
  also have ... = (∑ σ←[0..<2 ^ oe-n]. ∑ ρ←[0..<2 ^ oe-n]. of-bool (j =
σ + ρ) * ((2 ^ (n + 2) - 1) * (2 ^ (n + 2) - 1)))
  by (simp add: sum-list-mult-const)
  also have ... ≤ (∑ σ←[0..<2 ^ oe-n]. 1) * ((2 ^ (n + 2) - 1) * (2 ^ (n
+ 2) - 1))
  apply (intro mult-le-mono1 sum-list-mono)
  subgoal for σ
  by (intro of-bool-sum-leq-1) simp-all
  done
  also have ... = 2 ^ oe-n * ((2 ^ (n + 2) - 1) * (2 ^ (n + 2) - 1))
  apply (intro arg-cong2[where f = (*)] refl)
  using sum-list-triv[of (1::nat) [0..<2 ^ oe-n]] by simp
  also have ... < 2 ^ oe-n * (2 ^ (n + 2) * 2 ^ (n + 2))
  apply (intro iffD2[OF mult-less-cancel1] conjI)
  subgoal by simp
  subgoal by (intro mult-strict-mono) simp-all
  done
  also have ... = 2 ^ (oe-n + 2 * n + 4) by (simp add: power-add
power2-eq-square power-even-eq)
  finally show ?thesis unfolding oe-n-def apply (cases odd m)
  subgoal by (simp add: add commute pad-length-def)
  subgoal by (simp add: power-add pad-length-def)
  done
  qed
  subgoal for j using 2 .
  subgoal by assumption
  done

```

**then show**  $\text{Nat-LSBF.to-nat } (\gamma ! i ! j) = \gamma' (i * 2^{\wedge} (oe-n - 1) + j)$   
**using** *index-comp that by metis*  
**qed**  
**have**  $\gamma c: [\text{int } (\gamma' \tau) + \text{int } (\gamma' (2^{\wedge} oe-n + \tau)) = c' ! \tau] \pmod{2^{\wedge} (n + 2)}$   
**if**  $\tau < 2^{\wedge} oe-n$  **for**  $\tau$   
**proof** –  
**have**  $c' ! \tau \pmod{2^{\wedge} (n + 2)} = (\sum \sigma \leftarrow [0..<2^{\wedge} oe-n]. \sum \varrho \leftarrow [0..<2^{\wedge} oe-n].$   
 $\text{of-bool } [\tau = \sigma + \varrho] \pmod{2^{\wedge} oe-n} * (\text{int } (\text{Nat-LSBF.to-nat } (A.\text{num-blocks} ! \sigma)) * \text{int } (\text{Nat-LSBF.to-nat } (B.\text{num-blocks}$   
 $! \varrho))) \pmod{2^{\wedge} (n + 2)}$   
**by** (*intro arg-cong[where  $f = \lambda j. j \pmod{-}$ ] c'-alt[OF that]*)  
**also have**  $\dots = (\sum \sigma \leftarrow [0..<2^{\wedge} oe-n]. \sum \varrho \leftarrow [0..<2^{\wedge} oe-n].$   
 $\text{of-bool } [\tau = \sigma + \varrho] \pmod{2^{\wedge} oe-n} * ((\text{int } (\text{Nat-LSBF.to-nat } (A.\text{num-blocks} ! \sigma)) \pmod{2^{\wedge} (n + 2)}) * (\text{int}$   
 $(\text{Nat-LSBF.to-nat } (B.\text{num-blocks} ! \varrho)) \pmod{2^{\wedge} (n + 2)})) \pmod{2^{\wedge} (n + 2)}$   
**apply** (*intro sum-list-mod'*)  
**using** *mod-mult-right-eq[of of-bool -] mod-mult-eq[of int (Nat-LSBF.to-nat*  
 $(A.\text{num-blocks} ! -)) - \text{int } (\text{Nat-LSBF.to-nat } (B.\text{num-blocks} ! -))]$   
**by** *metis*  
**also have**  $(\sum \sigma \leftarrow [0..<2^{\wedge} oe-n]. \sum \varrho \leftarrow [0..<2^{\wedge} oe-n].$   
 $\text{of-bool } [\tau = \sigma + \varrho] \pmod{2^{\wedge} oe-n} * ((\text{int } (\text{Nat-LSBF.to-nat } (A.\text{num-blocks} ! \sigma)) \pmod{2^{\wedge} (n + 2)}) * (\text{int}$   
 $(\text{Nat-LSBF.to-nat } (B.\text{num-blocks} ! \varrho)) \pmod{2^{\wedge} (n + 2)})) =$   
 $(\sum \sigma \leftarrow [0..<2^{\wedge} oe-n]. \sum \varrho \leftarrow [0..<2^{\wedge} oe-n].$   
 $\text{of-bool } [\tau = \sigma + \varrho] \pmod{2^{\wedge} oe-n} * ((\text{int } (\text{Nat-LSBF.to-nat } (A.\text{num-Zn} ! \sigma))) * (\text{int } (\text{Nat-LSBF.to-nat } (B.\text{num-Zn}$   
 $! \varrho))))))$   
**apply** (*intro-cong [cong-tag-2 (\*)] more: semiring-1-sum-list-eq refl*)  
**unfolding**  $A.\text{num-Zn-def } B.\text{num-Zn-def}$   
**subgoal for**  $\sigma \varrho$   
**using**  $A.\text{length-num-blocks}$   
**using**  $Znr.\text{to-nat-reduce}$   
**by** (*simp add: zmod-int*)  
**subgoal for**  $\sigma \varrho$   
**using**  $B.\text{length-num-blocks } Znr.\text{to-nat-reduce}$   
**by** (*simp add: zmod-int*)  
**done**  
**also have**  $\dots = (\sum \sigma \leftarrow [0..<2^{\wedge} oe-n]. \sum \varrho \leftarrow [0..<2^{\wedge} oe-n].$   
 $\text{of-bool } (\tau = \sigma + \varrho \vee \tau + 2^{\wedge} oe-n = \sigma + \varrho) * (\text{int } (\text{Nat-LSBF.to-nat } (A.\text{num-Zn} ! \sigma)) * \text{int } (\text{Nat-LSBF.to-nat } (B.\text{num-Zn}$   
 $! \varrho)))$   
**proof** (*intro-cong [cong-tag-2 (\*), cong-tag-1 of-bool] more: semiring-1-sum-list-eq*  
*refl*)  
**fix**  $\sigma \varrho :: \text{nat}$   
**assume**  $a: \sigma \in \text{set } [0..<2^{\wedge} oe-n] \varrho \in \text{set } [0..<2^{\wedge} oe-n]$   
**have**  $[\tau = \sigma + \varrho] \pmod{2^{\wedge} oe-n} \implies \tau = \sigma + \varrho \vee \tau + 2^{\wedge} oe-n = \sigma + \varrho$   
**proof** –  
**assume**  $[\tau = \sigma + \varrho] \pmod{2^{\wedge} oe-n}$   
**then have**  $\tau \pmod{2^{\wedge} oe-n} = (\sigma + \varrho) \pmod{2^{\wedge} oe-n}$

**unfolding** *cong-def* .  
**then have**  $\tau = (\sigma + \varrho) \bmod (2^{\wedge} oe-n)$   
**using** *mod-less*  $\langle \tau < 2^{\wedge} oe-n \rangle$  **by** *simp*  
**define** *i* **where**  $i = (\sigma + \varrho) \text{ div } (2^{\wedge} oe-n)$   
**have**  $\tau + i * 2^{\wedge} oe-n = \sigma + \varrho$   
**unfolding**  $\langle \tau = (\sigma + \varrho) \bmod (2^{\wedge} oe-n) \rangle$  *i-def*  
**by** (*simp add: mod-div-mult-eq*)  
**moreover have**  $i \leq 1$   
**proof** (*rule ccontr*)  
**assume**  $\neg i \leq 1$   
**then have**  $i \geq 2$  **by** *simp*  
**then have**  $\sigma + \varrho \geq 2^{\wedge} (oe-n + 1)$   
**using**  $\langle \tau + i * 2^{\wedge} oe-n = \sigma + \varrho \rangle$   
**by** (*metis div-exp-eq div-greater-zero-iff i-def pos2 power-one-right*)  
**then show** *False* **using** *a* **by** *simp*  
**qed**  
**hence**  $i = 0 \vee i = 1$  **by** *linarith*  
**ultimately show** *?thesis* **by** *auto*  
**qed**  
**moreover have**  $\tau = \sigma + \varrho \vee \tau + 2^{\wedge} oe-n = \sigma + \varrho \implies [\tau = \sigma + \varrho]$   
*(mod*  $2^{\wedge} oe-n$ *)*  
**by** (*metis cong-add-lcancel-0-nat cong-refl cong-sym cong-to-1'-nat mult-1*)  
**ultimately show**  $[\tau = \sigma + \varrho] \text{ (mod } 2^{\wedge} oe-n) = (\tau = \sigma + \varrho \vee \tau + 2^{\wedge} oe-n = \sigma + \varrho)$  **by** *argo*  
**qed**  
**also have**  $\dots = (\sum \sigma \leftarrow [0..<2^{\wedge} oe-n]. \sum \varrho \leftarrow [0..<2^{\wedge} oe-n].$   
*(of-bool*  $(\tau = \sigma + \varrho)$  *+ of-bool*  $(\tau + 2^{\wedge} oe-n = \sigma + \varrho)) *$   
*(int*  $(\text{Nat-LSBF.to-nat } (A.\text{num-Zn } ! \sigma)) * \text{int } (\text{Nat-LSBF.to-nat } (B.\text{num-Zn } ! \varrho)))$ *)*  
**apply** (*intro-cong [cong-tag-2 (\*)] more: semiring-1-sum-list-eq refl of-bool-disj-excl*)  
**by** *simp*  
**also have**  $\dots = (\sum \sigma \leftarrow [0..<2^{\wedge} oe-n]. \sum \varrho \leftarrow [0..<2^{\wedge} oe-n].$   
*of-bool*  $(\tau = \sigma + \varrho) * (\text{int } (\text{Nat-LSBF.to-nat } (A.\text{num-Zn } ! \sigma)) * \text{int } (\text{Nat-LSBF.to-nat } (B.\text{num-Zn } ! \varrho)))$ *)* +  
 $(\sum \sigma \leftarrow [0..<2^{\wedge} oe-n]. \sum \varrho \leftarrow [0..<2^{\wedge} oe-n].$   
*of-bool*  $(\tau + 2^{\wedge} oe-n = \sigma + \varrho) * (\text{int } (\text{Nat-LSBF.to-nat } (A.\text{num-Zn } ! \sigma))$   
 $* \text{int } (\text{Nat-LSBF.to-nat } (B.\text{num-Zn } ! \varrho)))$ *)*  
**by** (*simp add: int-distrib(1) sum-list-addr*)  
**also have**  $\dots = (\sum \sigma \leftarrow [0..<2^{\wedge} oe-n]. \sum \varrho \leftarrow [0..<2^{\wedge} oe-n].$   
*int*  $(\text{of-bool } (\tau = \sigma + \varrho) * ((\text{Nat-LSBF.to-nat } (A.\text{num-Zn } ! \sigma) * \text{Nat-LSBF.to-nat } (B.\text{num-Zn } ! \varrho))))$ *)* +  
 $(\sum \sigma \leftarrow [0..<2^{\wedge} oe-n]. \sum \varrho \leftarrow [0..<2^{\wedge} oe-n].$   
*int*  $(\text{of-bool } (\tau + 2^{\wedge} oe-n = \sigma + \varrho) * ((\text{Nat-LSBF.to-nat } (A.\text{num-Zn } ! \sigma)$   
 $* (\text{Nat-LSBF.to-nat } (B.\text{num-Zn } ! \varrho))))$ *)*  
**apply** (*intro-cong [cong-tag-2 (+)] more: semiring-1-sum-list-eq*)  
**by** *simp-all*  
**also have**  $\dots = \text{int } (\sum \sigma \leftarrow [0..<2^{\wedge} oe-n]. \sum \varrho \leftarrow [0..<2^{\wedge} oe-n].$   
*of-bool*  $(\tau = \sigma + \varrho) * ((\text{Nat-LSBF.to-nat } (A.\text{num-Zn } ! \sigma) * \text{Nat-LSBF.to-nat } (B.\text{num-Zn } ! \varrho)))$ *)* +  
 $(\sum \sigma \leftarrow [0..<2^{\wedge} oe-n]. \sum \varrho \leftarrow [0..<2^{\wedge} oe-n].$   
*of-bool*  $(\tau + 2^{\wedge} oe-n = \sigma + \varrho) * ((\text{Nat-LSBF.to-nat } (A.\text{num-Zn } ! \sigma)$   
 $* (\text{Nat-LSBF.to-nat } (B.\text{num-Zn } ! \varrho))))$ *)* +

```

      int (∑ σ←[0.. $2^{\wedge}oe-n$ ]. ∑ ρ←[0.. $2^{\wedge}oe-n$ ].
      of-bool (τ +  $2^{\wedge}oe-n = \sigma + \rho$ ) * ((Nat-LSBF.to-nat (A.num-Zn ! σ) *
(Nat-LSBF.to-nat (B.num-Zn ! ρ))))))
    by (simp add: sum-list-int)
  also have ... = int (γ' τ) + int (γ' (τ +  $2^{\wedge}oe-n$ ))
    unfolding γ'-def by argo
  finally show [int (γ' τ) + int (γ' ( $2^{\wedge}oe-n + \tau$ )) = c' ! τ] (mod  $2^{\wedge}(n + 2)$ )
    unfolding cong-def by (simp add: add.commute)
qed
have η-carrier: length (η ! j) = n + 2 if j <  $2^{\wedge}(oe-n - 1)$  for j
proof -
  have η ! j = Znr.add-mod
    (Znr.subtract-mod (take (n + 2) (γ ! 0 ! j)) (take (n + 2) (γ ! 1 ! j)))
    (Znr.subtract-mod (take (n + 2) (γ ! 2 ! j)) (take (n + 2) (γ ! 3 ! j)))
    unfolding η-def apply (intro nth-map4) using scγ that by simp-all
  then show ?thesis using Znr.add-mod-closed by simp
qed
have η-residues: Znr.to-residue-ring (η ! j) = z' j mod  $2^{\wedge}(n + 2)$ 
  if j <  $2^{\wedge}(oe-n - 1)$  for j
proof -
  have Znr.to-residue-ring (η ! j) =
    Znr.to-residue-ring (
      Znr.add-mod
        (Znr.subtract-mod (take (n + 2) (γ ! 0 ! j)) (take (n + 2) (γ ! 1 ! j)))
        (Znr.subtract-mod (take (n + 2) (γ ! 2 ! j)) (take (n + 2) (γ ! 3 ! j))))
    unfolding η-def
  apply (intro arg-cong[where f = Znr.to-residue-ring] nth-map4)
  using ⟨j <  $2^{\wedge}(oe-n - 1)$ ⟩ scγ
  by simp-all
  also have ... =
    Znr.to-residue-ring (Znr.subtract-mod (take (n + 2) (γ ! 0 ! j)) (take (n + 2)
2) (γ ! 1 ! j))) ⊕Zn
    Znr.to-residue-ring (Znr.subtract-mod (take (n + 2) (γ ! 2 ! j)) (take (n + 2)
2) (γ ! 3 ! j)))
    by (intro Znr.add-mod-correct)
  also have ... =
    (Znr.to-residue-ring (take (n + 2) (γ ! 0 ! j)) ⊕Zn
    Znr.to-residue-ring (take (n + 2) (γ ! 1 ! j))) ⊕Zn
    (Znr.to-residue-ring (take (n + 2) (γ ! 2 ! j)) ⊕Zn
    Znr.to-residue-ring (take (n + 2) (γ ! 3 ! j)))
  apply (intro arg-cong2[where f = (⊕Zn)]])
  subgoal
    using less-exp[of n + 2] by (intro Znr.subtract-mod-correct) simp-all
  subgoal
    using less-exp[of n + 2] by (intro Znr.subtract-mod-correct) simp-all
  done
  also have ... =
    (Znr.to-residue-ring (take (n + 2) (γ ! 0 ! j)) ⊕Zn

```

```

Znr.to-residue-ring (take (n + 2) (γ ! 2 ! j))) ⊖ Zn
(Znr.to-residue-ring (take (n + 2) (γ ! 1 ! j))) ⊕ Zn
Znr.to-residue-ring (take (n + 2) (γ ! 3 ! j)))
apply (intro Znr.diff-sum)
using Znr.to-residue-ring-in-carrier by simp-all
also have ... =
(int (Nat-LSBF.to-nat (take (n + 2) (γ ! 0 ! j))) mod 2^(n + 2) ⊕ Zn
int (Nat-LSBF.to-nat (take (n + 2) (γ ! 2 ! j))) mod 2^(n + 2)) ⊖ Zn
(int (Nat-LSBF.to-nat (take (n + 2) (γ ! 1 ! j))) mod 2^(n + 2) ⊕ Zn
int (Nat-LSBF.to-nat (take (n + 2) (γ ! 3 ! j))) mod 2^(n + 2))
unfolding Znr.to-residue-ring-def by simp
also have ... =
(int (Nat-LSBF.to-nat (γ ! 0 ! j)) mod 2^(n + 2) ⊕ Zn
int (Nat-LSBF.to-nat (γ ! 2 ! j)) mod 2^(n + 2)) ⊖ Zn
(int (Nat-LSBF.to-nat (γ ! 1 ! j)) mod 2^(n + 2) ⊕ Zn
int (Nat-LSBF.to-nat (γ ! 3 ! j)) mod 2^(n + 2))
apply (intro-cong [cong-tag-2 (λ i j. i ⊖ Zn j), cong-tag-2 (⊕ Zn)])
by (simp-all add: to-nat-take zmod-int)
also have ... =
(int (γ' (0 * 2^(oe-n - 1) + j) mod 2^(n + 2) ⊕ Zn
int (γ' (2 * 2^(oe-n - 1) + j) mod 2^(n + 2)) ⊖ Zn
(int (γ' (1 * 2^(oe-n - 1) + j) mod 2^(n + 2) ⊕ Zn
int (γ' (3 * 2^(oe-n - 1) + j) mod 2^(n + 2))
apply (intro-cong [cong-tag-2 (λ i j. i ⊖ Zn j), cong-tag-2 (⊕ Zn), cong-tag-2
(mod), cong-tag-1 int] more: refl to-nat-γ ⟨j < 2^(oe-n - 1)⟩)
by simp-all
also have ... =
(int (γ' j) mod 2^(n + 2) ⊕ Zn
int (γ' (2^oe-n + j)) mod 2^(n + 2)) ⊖ Zn
(int (γ' (2^(oe-n - 1) + j) mod 2^(n + 2) ⊕ Zn
int (γ' (2^oe-n + (2^(oe-n - 1) + j))) mod 2^(n + 2))
apply (intro-cong [cong-tag-2 (λ i j. i ⊖ Zn j), cong-tag-2 (⊕ Zn), cong-tag-2
(mod), cong-tag-1 int, cong-tag-1 γ'] more: refl)
using two-pow-oe-n-as-halves by simp-all
also have ... =
(int (γ' j) mod 2^(n + 2) +
int (γ' (2^oe-n + j)) mod 2^(n + 2)) mod 2^(n + 2) ⊖ Zn
(int (γ' (2^(oe-n - 1) + j) mod 2^(n + 2) +
int (γ' (2^oe-n + (2^(oe-n - 1) + j))) mod 2^(n + 2)) mod 2^(n +
2))
apply (intro-cong [cong-tag-2 (λ i j. i ⊖ Zn j)])
unfolding Znr.res-add-eq
subgoal by (intro arg-cong[where f = λ i. - mod i], unfold int-exp-hom[symmetric],
simp)
subgoal by (intro arg-cong[where f = λ i. - mod i], simp)
done
also have ... =
(int (γ' j) + int (γ' (2^oe-n + j))) mod 2^(n + 2) ⊖ Zn
(int (γ' (2^(oe-n - 1) + j) +

```

```

    int (γ' (2 ^ oe-n + (2 ^ (oe-n - 1) + j))) mod 2 ^ (n + 2)
  by (intro-cong [cong-tag-2 (λi j. i ⊖Zn j)] more: mod-add-eq)
  also have ... = ((c' ! j) mod 2 ^ (n + 2)) ⊖Zn ((c' ! (2 ^ (oe-n - 1) + j))
mod 2 ^ (n + 2))
  apply (intro arg-cong2[where f = λi j. i ⊖Zn j])
  using γc unfolding cong-def using ⟨j < 2 ^ (oe-n - 1)⟩ index-intros[of j]
  by simp-all
  also have ... = (c' ! j - c' ! (2 ^ (oe-n - 1) + j)) mod 2 ^ (n + 2)
  unfolding Znr.residues-minus-eq
  by (simp add: mod-diff-eq)
  also have ... = (c' ! j - c' ! (2 ^ (oe-n - 1) + j) + 2 ^ (oe-n + 2 ^ n)) mod
2 ^ (n + 2)
  proof -
    have oe-n ≥ n unfolding oe-n-def by simp
    moreover have 2 ^ n ≥ n + 2 using aux-ineq-3[OF n-gt-1] .
    ultimately have oe-n + 2 ^ n ≥ n + 2
      by simp
    then have (2::int) ^ (n + 2) dvd 2 ^ (oe-n + 2 ^ n)
      apply (intro le-imp-power-dvd) .
    then have (2::int) ^ (oe-n + 2 ^ n) mod 2 ^ (n + 2) = 0
      using dvd-imp-mod-0 by blast
    then show ?thesis using mod-add-right-eq by fastforce
  qed
  also have ... = z' j mod 2 ^ (n + 2)
    unfolding z'-def using ⟨j < 2 ^ (oe-n - 1)⟩ by presburger
  finally show Znr.to-residue-ring (η ! j) = z' j mod 2 ^ (n + 2) .
  qed

```

```

define c'-mod where c'-mod = map (λi. i mod Fnr.n) c'
then have length c'-mod = 2 ^ oe-n using ⟨length c' = 2 ^ oe-n⟩ by simp
have c'-mod-carrier: ∧j. j < 2 ^ oe-n ⇒ c'-mod ! j ∈ carrier Fnr
  unfolding c'-mod-def
  using Fnr.mod-in-carrier ⟨length c' = 2 ^ oe-n⟩ by simp
have c'-mod-eq: c'-mod = Fnr.cyclic-convolution (2 ^ oe-n) (map Fnr.to-residue-ring
A.num-blocks) (map Fnr.to-residue-ring B.num-blocks)
  proof (intro nth-equalityI)
    show length c'-mod = length
      (Fnr.cyclic-convolution (2 ^ oe-n) (map Fnr.to-residue-ring A.num-blocks)
      (map Fnr.to-residue-ring B.num-blocks))
    by (simp add: ⟨length c'-mod = 2 ^ oe-n⟩)
  fix i
  assume i < length c'-mod
  then have i < 2 ^ oe-n using ⟨length c'-mod = 2 ^ oe-n⟩ by simp
  then have c'-mod ! i = (∑ σ ← [0..<2 ^ oe-n]. int (Nat-LSBF.to-nat (A.num-blocks
! σ)) *
    int (Nat-LSBF.to-nat (B.num-blocks ! ((2 ^ oe-n + i - σ) mod 2 ^
oe-n)))) mod int Fnr.n

```

**unfolding**  $c'$ -mod-def **using**  $c'$ -nth[OF  $\langle i < 2^{\wedge} oe-n \rangle$ ]  $\langle \text{length } c' = 2^{\wedge} oe-n \rangle$  **by** *simp*  
**also have** ... =  $(\bigoplus_{F_n \sigma \leftarrow [0.. < 2^{\wedge} oe-n]}$ .  $(\text{int } (\text{Nat-LSBF.to-nat } (A.\text{num-blocks } ! \sigma))) * \text{int } (\text{Nat-LSBF.to-nat } (B.\text{num-blocks } ! ((2^{\wedge} oe-n + i - \sigma) \bmod 2^{\wedge} oe-n)))) \bmod \text{int } F_n.n$ )  
**by** (*intro*  $F_n.\text{monoid-sum-list-eq-sum-list}'[\text{symmetric}]$ )  
**also have** ... =  $(\bigoplus_{F_n \sigma \leftarrow [0.. < 2^{\wedge} oe-n]}$ .  $((\text{int } (\text{Nat-LSBF.to-nat } (A.\text{num-blocks } ! \sigma))) \bmod \text{int } F_n.n) * (\text{int } (\text{Nat-LSBF.to-nat } (B.\text{num-blocks } ! ((2^{\wedge} oe-n + i - \sigma) \bmod 2^{\wedge} oe-n)))) \bmod \text{int } F_n.n$ )  
**mod int**  $F_n.n$   
**by** (*intro*  $F_n.\text{monoid-sum-list-cong mod-mult-eq}[\text{symmetric}]$ )  
**also have** ... =  $(\bigoplus_{F_n \sigma \leftarrow [0.. < 2^{\wedge} oe-n]}$ .  $(F_n.\text{to-residue-ring } (A.\text{num-blocks } ! \sigma) * F_n.\text{to-residue-ring } (B.\text{num-blocks } ! ((2^{\wedge} oe-n + i - \sigma) \bmod 2^{\wedge} oe-n))) \bmod \text{int } F_n.n$ )  
**unfolding**  $F_n.\text{to-residue-ring.simps}$  **by** *argo*  
**also have** ... =  $(\bigoplus_{F_n \sigma \leftarrow [0.. < 2^{\wedge} oe-n]}$ .  $F_n.\text{to-residue-ring } (A.\text{num-blocks } ! \sigma) \otimes_{F_n} F_n.\text{to-residue-ring } (B.\text{num-blocks } ! ((2^{\wedge} oe-n + i - \sigma) \bmod 2^{\wedge} oe-n)))$   
**unfolding**  $F_n.\text{res-mult-eq}$  **by** *argo*  
**also have** ... =  $(\bigoplus_{F_n \sigma \leftarrow [0.. < 2^{\wedge} oe-n]}$ .  $\text{map } F_n.\text{to-residue-ring } A.\text{num-blocks } ! \sigma \otimes_{F_n} \text{map } F_n.\text{to-residue-ring } B.\text{num-blocks } ! ((2^{\wedge} oe-n + i - \sigma) \bmod 2^{\wedge} oe-n))$   
**apply** (*intro-cong* [*cong-tag-2* ( $\otimes_{F_n}$ )] *more:*  $F_n.\text{monoid-sum-list-cong nth-map}[\text{symmetric}]$ )  
**subgoal using**  $A.\text{length-num-blocks}$  **by** *simp*  
**subgoal using**  $B.\text{length-num-blocks}$  **by** *simp*  
**done**  
**also have** ... =  $F_n.\text{cyclic-convolution } (2^{\wedge} oe-n) (\text{map } F_n.\text{to-residue-ring } A.\text{num-blocks})$   
 $(\text{map } F_n.\text{to-residue-ring } B.\text{num-blocks}) ! i$   
**by** (*intro*  $F_n.\text{cyclic-convolution-nth}[\text{symmetric}] \langle i < 2^{\wedge} oe-n \rangle$ )  
**finally show**  $c'$ -mod !  $i = F_n.\text{cyclic-convolution } (2^{\wedge} oe-n) (\text{map } F_n.\text{to-residue-ring } A.\text{num-blocks})$   
 $(\text{map } F_n.\text{to-residue-ring } B.\text{num-blocks}) ! i$  .  
**qed**  
**have** *fill-a'*:  $\text{map } F_n.\text{to-residue-ring } A.\text{num-blocks} = \text{map } F_n.\text{to-residue-ring } (\text{map } (\text{fill } (2^{\wedge} (n + 1))) A.\text{num-blocks})$   
**apply** (*intro nth-equalityI*)  
**subgoal by** *simp*  
**subgoal for**  $i$   
**unfolding**  $F_n.\text{to-residue-ring.simps}$  **by** *simp*  
**done**  
**have** *fill-b'*:  $\text{map } F_n.\text{to-residue-ring } B.\text{num-blocks} = \text{map } F_n.\text{to-residue-ring } (\text{map } (\text{fill } (2^{\wedge} (n + 1))) B.\text{num-blocks})$

```

apply (intro nth-equalityI)
subgoal by simp
subgoal for i
  unfolding Fnr.to-residue-ring.simps by simp
done
have aux0: Fnr.NTT  $\mu$  c'-mod = map2 ( $\otimes_{F_n}$ ) (Fnr.NTT  $\mu$  (map Fnr.to-residue-ring
(map (fill (2 ^ (n + 1))) A.num-blocks)))
(Fnr.NTT  $\mu$  (map Fnr.to-residue-ring (map (fill (2 ^ (n + 1)))
B.num-blocks))))
proof (intro nth-equalityI)
  show length (Fnr.NTT  $\mu$  c'-mod) = length (map2 ( $\otimes_{F_n}$ )
(Fnr.NTT  $\mu$  (map Fnr.to-residue-ring (map (fill (2 ^ (n + 1))) A.num-blocks)))
(Fnr.NTT  $\mu$  (map Fnr.to-residue-ring (map (fill (2 ^ (n + 1))) B.num-blocks))))
  using <length c'-mod = 2 ^ oe-n> A.length-num-blocks B.length-num-blocks
by simp
  fix i :: nat
  assume i < length (Fnr.NTT  $\mu$  c'-mod)
  then have i < 2 ^ oe-n using <length c'-mod = 2 ^ oe-n> by simp
  have Fnr.NTT  $\mu$  c'-mod ! i =
    Fnr.NTT  $\mu$  (map Fnr.to-residue-ring A.num-blocks) ! i  $\otimes_{F_n}$ 
    Fnr.NTT  $\mu$  (map Fnr.to-residue-ring B.num-blocks) ! i
  unfolding c'-mod-eq
  apply (intro Fnr.convolution-rule[symmetric] set-subseteqI)
  subgoal using A.length-num-blocks by simp
  subgoal using B.length-num-blocks by simp
  subgoal using Fnr.to-residue-ring-in-carrier by simp
  subgoal using Fnr.to-residue-ring-in-carrier by simp
  subgoal using  $\mu$ -root-of-unity .
  subgoal using <i < 2 ^ oe-n> .
  done
  then show Fnr.NTT  $\mu$  c'-mod ! i = map2 ( $\otimes_{F_n}$ )
(Fnr.NTT  $\mu$  (map Fnr.to-residue-ring (map (fill (2 ^ (n + 1))) A.num-blocks)))
(Fnr.NTT  $\mu$  (map Fnr.to-residue-ring (map (fill (2 ^ (n + 1))) B.num-blocks)))
! i
  unfolding fill-a' fill-b'
  using A.length-num-blocks B.length-num-blocks <length c'-mod = 2 ^ oe-n>
by (simp add: <i < 2 ^ oe-n>)
qed
have IH-inst: Fnr.to-residue-ring (schoenhage-strassen n (evens-odds False
A.num-dft ! i) (evens-odds False B.num-dft ! i)) =
  Fnr.to-residue-ring (evens-odds False A.num-dft ! i)  $\otimes_{\text{int-lsbfermat.Fn } n}$ 
  Fnr.to-residue-ring (evens-odds False B.num-dft ! i)  $\wedge$ 
  schoenhage-strassen n (evens-odds False A.num-dft ! i) (evens-odds False
B.num-dft ! i)  $\in$  Fnr.fermat-non-unique-carrier
if i < 2 ^ (oe-n - 1) for i
apply (intro less.IH n-lt-m)
subgoal
  apply (rule set-mp[OF A.num-dft-carrier])
  apply (rule set-mp[OF set-evens-odds])

```

```

    apply (rule nth-mem)
    apply (unfold A.num-dft-odds-def[symmetric] A.length-num-dft-odds)
    apply (rule that)
  done
subgoal
  apply (rule set-mp[OF B.num-dft-carrier])
  apply (rule set-mp[OF set-evens-odds])
  apply (rule nth-mem)
  apply (unfold B.num-dft-odds-def[symmetric] B.length-num-dft-odds)
  apply (rule that)
  done
done
have aux4: Fnr.to-residue-ring (c-dft-odds ! i) =
  Fnr.to-residue-ring (A.num-dft-odds ! i)  $\otimes_{F_n}$ 
  Fnr.to-residue-ring (B.num-dft-odds ! i)
  c-dft-odds ! i  $\in$  Fnr.fermat-non-unique-carrier
if i < 2 ^ (oe-n - 1) for i
proof -
  from that have i < length c-dft-odds using length-c-dft-odds by simp
  then have c-dft-odds ! i = schoenhage-strassen n (A.num-dft-odds ! i)
(B.num-dft-odds ! i)
  unfolding c-dft-odds-def by simp
  also have Fnr.to-residue-ring ... =
    Fnr.to-residue-ring (A.num-dft-odds ! i)  $\otimes_{\text{int-lsbf-fermat.Fn } n}$ 
    Fnr.to-residue-ring (B.num-dft-odds ! i)  $\wedge$ 
    ...  $\in$  Fnr.fermat-non-unique-carrier
  using IH-inst[OF that]
  using A.num-dft-odds-def B.num-dft-odds-def Fn-def
  by argo
  finally show Fnr.to-residue-ring (c-dft-odds ! i) =
    Fnr.to-residue-ring (A.num-dft-odds ! i)  $\otimes_{F_n}$ 
    Fnr.to-residue-ring (B.num-dft-odds ! i)
    c-dft-odds ! i  $\in$  Fnr.fermat-non-unique-carrier
  by simp-all
qed
then have to-res-c-dft-odds: map Fnr.to-residue-ring c-dft-odds = map2 ( $\otimes_{F_n}$ )
  (map Fnr.to-residue-ring A.num-dft-odds)
  (map Fnr.to-residue-ring B.num-dft-odds)
  apply (intro nth-equalityI)
  using length-c-dft-odds A.length-num-dft-odds B.length-num-dft-odds
  by auto
have set c'-mod  $\subseteq$  carrier Fn
  apply (intro set-subseteqI)
  using c'-mod-carrier <length c'-mod = 2 ^ oe-n> by simp
have Fnr.NTT (invFn  $\mu$ ) (Fnr.NTT  $\mu$  c'-mod) =
  map (( $\otimes_{F_n}$ ) (Fnr.nat-embedding (2 ^ oe-n))) c'-mod
  apply (intro Fnr.inversion-rule)
  subgoal by simp
  subgoal using  $\mu$ -prim-root .

```

```

subgoal premises prems for i
  apply (intro Fnr.sufficiently-good[of - - oe-n])
  subgoal using  $\mu$ -prim-root .
  subgoal using  $\mu$ -halfway-property by blast
  subgoal by (fact prems)
  done
subgoal using  $\langle \text{length } c'\text{-mod} = 2^{\wedge} \text{oe-n} \rangle$  by simp
subgoal using  $\langle \text{set } c'\text{-mod} \subseteq \text{carrier } F_n \rangle$  .
done
also have ... = map (( $\otimes_{F_n}$ ) ( $2^{\wedge} \text{oe-n mod int } F_n$ )) c'-mod
  unfolding Fnr.nat-embedding-eq by simp

also have ... = map (( $\otimes_{F_n}$ ) ( $2^{\wedge} \text{oe-n}$ )) c'-mod
  unfolding Fnr.pow-nat-eq[symmetric] two-oe-n by argo
finally have aux1: Fnr.NTT (invFn  $\mu$ ) (Fnr.NTT  $\mu$  c'-mod) ! j =
  ( $2^{\wedge} \text{oe-n}$ )  $\otimes_{F_n}$  (c'-mod ! j) if j <  $2^{\wedge} \text{oe-n}$  for j
  using  $\langle \text{length } c'\text{-mod} = 2^{\wedge} \text{oe-n} \rangle$  that by simp
  have c'-NTT-NTT-carrier: Fnr.NTT (invFn  $\mu$ ) (Fnr.NTT  $\mu$  c'-mod) ! j  $\in$ 
carrier Fn if j <  $2^{\wedge} \text{oe-n}$  for j
  apply (intro set-subseteqD[OF Fnr.NTT-closed] Fnr.NTT-closed Fnr.Units-inv-closed
 $\mu$ -Units-Fn  $\mu$ -carrier-Fn  $\langle \text{set } c'\text{-mod} \subseteq \text{carrier } F_n \rangle$ )
  by (simp add:  $\langle \text{length } c'\text{-mod} = 2^{\wedge} \text{oe-n} \rangle$  that)
  have aux2: invFn ( $2^{\wedge} \text{oe-n}$ )  $\otimes_{F_n}$  Fnr.NTT (invFn  $\mu$ ) (Fnr.NTT  $\mu$  c'-mod) !
j =
  (c'-mod ! j) if j <  $2^{\wedge} \text{oe-n}$  for j
  apply (intro Fnr.inv-cancel-left)
  subgoal using c'-NTT-NTT-carrier that by presburger
  subgoal using c'-mod-carrier[OF that] by simp
  subgoal unfolding two-oe-n[symmetric] by (intro Fnr.Units-pow-closed
Fnr.two-is-unit)
  subgoal using aux1[OF that] .
  done
have aux3: c'-mod ! j  $\ominus_{F_n}$  c'-mod ! ( $2^{\wedge} (\text{oe-n} - 1) + j$ ) =
  (invFn  $2$ ) [ $\bigwedge_{F_n}$ ] (oe-n + prim-root-exponent * j - 1)  $\otimes_{F_n}$ 
Fnr.NTT (invFn  $\mu$  [ $\bigwedge_{F_n}$ ] ( $2::\text{nat}$ )) (map Fnr.to-residue-ring c-dft-odds) ! j
  if j <  $2^{\wedge} (\text{oe-n} - 1)$  for j
proof -
  have odd-indices:  $\bigwedge i. i < 2^{\wedge} (\text{oe-n} - 1) \implies (2::\text{nat}) * i + 1 < 2^{\wedge} \text{oe-n}$ 
proof -
  fix i :: nat
  assume i <  $2^{\wedge} (\text{oe-n} - 1)$ 
  then have i + 1  $\leq 2^{\wedge} (\text{oe-n} - 1)$  by simp
  then have  $2 * i + 2 \leq 2 * 2^{\wedge} (\text{oe-n} - 1)$  by simp
  also have ... =  $2^{\wedge} \text{oe-n}$  using two-pow-oe-n-as-halves by simp
  finally show  $2 * i + 1 < 2^{\wedge} \text{oe-n}$  by simp
qed
have c'-mod ! j  $\ominus_{F_n}$  c'-mod ! ( $2^{\wedge} (\text{oe-n} - 1) + j$ ) =
  invFn ( $2^{\wedge} \text{oe-n}$ )  $\otimes_{F_n}$  (Fnr.NTT (invFn  $\mu$ ) (Fnr.NTT  $\mu$  c'-mod) ! j)  $\ominus_{F_n}$ 
invFn ( $2^{\wedge} \text{oe-n}$ )  $\otimes_{F_n}$  (Fnr.NTT (invFn  $\mu$ ) (Fnr.NTT  $\mu$  c'-mod) ! ( $2^{\wedge} (\text{oe-n}$ 

```

- 1) + j))  
**apply** (intro arg-cong2[**where**  $f = \lambda i j. i \ominus_{F_n} j$ ])  
**using** aux2 index-intros[OF that] **by** simp-all  
**also have** ... =  $inv_{F_n} (2 \wedge oe-n) \otimes_{F_n} (Fnr.NTT (inv_{F_n} \mu) (Fnr.NTT \mu$   
 $c'-mod) ! j \ominus_{F_n}$   
 $Fnr.NTT (inv_{F_n} \mu) (Fnr.NTT \mu c'-mod) ! (2 \wedge (oe-n - 1) +$   
 $j))$   
**apply** (intro Fnr.r-distr-diff[symmetric])  
**subgoal by** (intro Fnr.Units-closed Fnr.Units-inv-Units two-oe-n-Units-Fn)  
**subgoal using** index-intros[OF that] c'-NTT-NTT-carrier **by** presburger  
**subgoal using** index-intros[OF that] c'-NTT-NTT-carrier **by** presburger  
**done**  
**also have** ... =  $inv_{F_n} (2 \wedge oe-n) \otimes_{F_n} (Fnr.nat-embedding 2 \otimes_{F_n} (inv_{F_n} \mu$   
 $[\bigwedge]_{F_n} j \otimes_{F_n}$   
 $Fnr.NTT (inv_{F_n} \mu [\bigwedge]_{F_n} (2::nat))$   
 $(map (!) (Fnr.NTT \mu c'-mod)) (filter odd [0..<2 \wedge oe-n])) ! j))$   
**unfolding** two-pow-oe-n-div-2[symmetric]  
**apply** (intro arg-cong2[**where**  $f = (\otimes_{F_n})$ ] refl Fnr.NTT-diffs)  
**subgoal using** oe-n-gt-0 **by** simp  
**subgoal by** (intro Fnr.primitive-root-inv  $\mu$ -prim-root) simp  
**subgoal by** (simp add:  $\langle length\ c'-mod = 2 \wedge oe-n \rangle$ )  
**subgoal using**  $\langle j < 2 \wedge (oe-n - 1) \rangle$  **unfolding**  $\langle 2 \wedge (oe-n - 1) = 2 \wedge oe-n$   
 $div\ 2 \rangle$  .  
**subgoal by** (intro Fnr.NTT-closed  $\langle set\ c'-mod \subseteq carrier\ F_n \rangle$   $\mu$ -carrier-Fn)  
**subgoal by** (intro Fnr.inv-halfway-property  $\mu$ -Units-Fn  $\mu$ -halfway-property)  
**done**  
**also have** ... =  $inv_{F_n} (2 \wedge oe-n) \otimes_{F_n} (2 \otimes_{F_n} (inv_{F_n} \mu [\bigwedge]_{F_n} j \otimes_{F_n}$   
 $Fnr.NTT (inv_{F_n} \mu [\bigwedge]_{F_n} (2::nat))$   
 $(map (!) (Fnr.NTT \mu c'-mod)) (filter odd [0..<2 \wedge oe-n])) ! j))$   
**using** Fnr.nat-embedding-eq Fnr.carrier-mod-eq[OF Fnr.two-in-carrier] **by**  
simp  
**also have**  $Fnr.NTT (inv_{F_n} \mu [\bigwedge]_{F_n} (2::nat)) (map (!) (Fnr.NTT \mu c'-mod))$   
 $(filter\ odd\ [0..<2 \wedge\ oe-n]) ! j =$   
 $Fnr.NTT (inv_{F_n} \mu [\bigwedge]_{F_n} (2::nat)) ($   
 $map (!) (map2 (\otimes_{F_n})$   
 $(Fnr.NTT \mu (map\ Fnr.to-residue-ring\ A.num-blocks-carrier))$   
 $(Fnr.NTT \mu (map\ Fnr.to-residue-ring\ B.num-blocks-carrier))$   
 $))$   
 $(filter\ odd\ [0..<2 \wedge\ oe-n])$   
 $) ! j$   
**using** aux0 **unfolding** A.num-blocks-carrier-def B.num-blocks-carrier-def  
**by** argo  
**also have** ... =  $Fnr.NTT (inv_{F_n} \mu [\bigwedge]_{F_n} (2::nat)) ($   
 $map (!) (map2 (\otimes_{F_n})$   
 $(map\ Fnr.to-residue-ring\ A.num-dft)$   
 $(map\ Fnr.to-residue-ring\ B.num-dft)$   
 $))$   
 $(filter\ odd\ [0..<2 \wedge\ oe-n])$   
 $) ! j$

by (intro-cong [cong-tag-2 (!), cong-tag-2 Fnr.NTT, cong-tag-2 map,  
 cong-tag-1 (!), cong-tag-2 zip] more: refl A.to-res-num-dft[symmetric] B.to-res-num-dft[symmetric])  
 also have map (!) (map2 ( $\otimes_{F_n}$ )  
 (map Fnr.to-residue-ring A.num-dft)  
 (map Fnr.to-residue-ring B.num-dft)  
 ))  
 (filter odd [0..<2 ^ oe-n]) =  
 map2 ( $\otimes_{F_n}$ ) (map Fnr.to-residue-ring (map (!) A.num-dft) (filter odd  
 [0..<length A.num-dft]))  
 (map Fnr.to-residue-ring (map (!) B.num-dft) (filter odd [0..<length  
 B.num-dft]))  
 apply (intro nth-equalityI)  
 subgoal by (simp add: length-filter-odd)  
 subgoal for i  
 using odd-indices[of i] length-filter-odd[of 2 ^ oe-n] filter-odd-nth[of i 2 ^  
 oe-n] A.length-num-dft B.length-num-dft two-pow-oe-n-as-halves  
 by simp  
 done  
 also have ... =  
 map2 ( $\otimes_{F_n}$ ) (map Fnr.to-residue-ring (evens-odds False A.num-dft))  
 (map Fnr.to-residue-ring (evens-odds False B.num-dft))  
 using filter-comprehension-evens-odds by metis  
 also have ... = map Fnr.to-residue-ring c-dft-odds  
 using to-res-c-dft-odds[symmetric] unfolding A.num-dft-odds-def B.num-dft-odds-def  
 .  
 also have  $inv_{F_n} ((2::int) ^ oe-n) \otimes_{F_n} (2 \otimes_{F_n} (inv_{F_n} \mu [\ ]_{F_n} j \otimes_{F_n}$   
 $Fnr.NTT (inv_{F_n} \mu [\ ]_{F_n} (2::nat)) (map Fnr.to-residue-ring c-dft-odds) !$   
 $j)) =$   
 $(inv_{F_n} 2) [\ ]_{F_n} oe-n \otimes_{F_n} ((inv_{F_n} 2) [\ ]_{F_n} (-1::int) \otimes_{F_n} ((inv_{F_n} 2)$   
 $[\ ]_{F_n} (prim-root-exponent * j) \otimes_{F_n}$   
 $Fnr.NTT (inv_{F_n} \mu [\ ]_{F_n} (2::nat)) (map Fnr.to-residue-ring c-dft-odds) !$   
 $j))$   
 apply (intro-cong [cong-tag-2 ( $\otimes_{F_n}$ )] more: refl)  
 subgoal  
 unfolding two-oe-n[symmetric] by (intro Fnr.inv-nat-pow Fnr.two-is-unit)  
 subgoal by (metis Fnr.Units-inv-Units Fnr.Units-inv-inv Fnr.units-inv-int-pow  
 Fnr.two-is-unit)  
 subgoal  
 proof -  
 have  $inv_{F_n} \mu [\ ]_{F_n} j = inv_{F_n} (\mu [\ ]_{F_n} j)$   
 using Fnr.inv-nat-pow[OF  $\mu$ -Units- $F_n$ , symmetric] .  
 also have ... =  $inv_{F_n} (2 [\ ]_{F_n} (prim-root-exponent * j))$   
 unfolding  $\mu$ -def Fnr.nat-pow-pow[OF Fnr.two-in-carrier] by argo  
 also have ... =  $inv_{F_n} 2 [\ ]_{F_n} (prim-root-exponent * j)$   
 using Fnr.inv-nat-pow[OF Fnr.two-is-unit] .  
 finally show ?thesis .  
 qed  
 done  
 also have ... =  $inv_{F_n} 2 [\ ]_{F_n} oe-n \otimes_{F_n} inv_{F_n} 2 [\ ]_{F_n} (-1::int) \otimes_{F_n} inv_{F_n}$

$2 \ [\frown]_{F_n} (\text{prim-root-exponent} * j) \otimes_{F_n}$   
 $Fnr.NTT (\text{inv}_{F_n} \mu \ [\frown]_{F_n} (2::\text{nat})) (\text{map } Fnr.\text{to-residue-ring } c\text{-dft-odds}) ! j$   
**apply** (*intro*  $Fnr.assoc4$ )  
**subgoal by** (*intro*  $Fnr.nat\text{-pow-closed}$   $Fnr.Units\text{-inv-closed}$   $Fnr.two\text{-is-unit}$ )  
**subgoal by** (*intro*  $Fnr.Units\text{-closed}$   $Fnr.int\text{-pow-closed}$   $Fnr.Units\text{-inv-Units}$   
 $Fnr.two\text{-is-unit}$ )  
**subgoal by** (*intro*  $Fnr.nat\text{-pow-closed}$   $Fnr.Units\text{-inv-closed}$   $Fnr.two\text{-is-unit}$ )  
**subgoal**  
**apply** (*intro*  $set\text{-subteqD}[OF Fnr.NTT\text{-closed}]$ )  
**subgoal**  
**apply** (*intro*  $set\text{-subteqI}$ )  
**using**  $Fnr.\text{to-residue-ring-in-carrier}$   
**by** *simp*  
**subgoal by** (*intro*  $Fnr.Units\text{-closed}$   $Fnr.Units\text{-pow-closed}$   $Fnr.Units\text{-inv-Units}$   
 $\mu\text{-Units-}F_n$ )  
**subgoal using**  $\langle j < 2 \wedge (oe-n - 1) \rangle$  **by** (*simp add: length-c-dft-odds*)  
**done**  
**done**  
**also have**  $\text{inv}_{F_n} 2 \ [\frown]_{F_n} oe-n \otimes_{F_n} \text{inv}_{F_n} 2 \ [\frown]_{F_n} (-1::\text{int}) \otimes_{F_n} \text{inv}_{F_n} 2 \ [\frown]_{F_n}$   
 $(\text{prim-root-exponent} * j) =$   
 $\text{inv}_{F_n} 2 \ [\frown]_{F_n} (oe-n + \text{prim-root-exponent} * j - 1)$   
**unfolding**  $int\text{-pow-int}[\text{symmetric}] Fnr.int\text{-pow-mult}[OF Fnr.Units\text{-inv-Units}[OF$   
 $Fnr.two\text{-is-unit}]$   
**apply** (*intro*  $arg\text{-cong}[\text{where } f = ([\frown]_{F_n} -)]$ )  
**using**  $oe-n\text{-gt-0}$  **by** *linarith*  
**finally show**  $c'\text{-mod} ! j \ominus_{F_n} c'\text{-mod} ! (2 \wedge (oe-n - 1) + j) =$   
 $\text{inv}_{F_n} 2 \ [\frown]_{F_n} (oe-n + \text{prim-root-exponent} * j - 1) \otimes_{F_n}$   
 $Fnr.NTT (\text{inv}_{F_n} \mu \ [\frown]_{F_n} (2::\text{nat})) (\text{map } Fnr.\text{to-residue-ring } c\text{-dft-odds}) ! j$   
.

**qed**  
**have**  $c\text{-dft-odds-carrier}: set\ c\text{-dft-odds} \subseteq Fnr.\text{fermat-non-unique-carrier}$   
**unfolding**  $c\text{-dft-odds-def}$   
**apply** (*intro*  $set\text{-subteqI}$ )  
**using**  $conjunct2[OF IH\text{-inst}] A.length\text{-num-dft-odds } B.length\text{-num-dft-odds}$   
**unfolding**  $A.num\text{-dft-odds-def } B.num\text{-dft-odds-def}$   
**by** *simp*  
**have**  $c\text{-diffs-carrier}: c\text{-diffs} ! i \in Fnr.\text{fermat-non-unique-carrier}$  **if**  $i < 2 \wedge (oe-n$   
 $- 1)$  **for**  $i$   
**unfolding**  $c\text{-diffs-def } Fnr.iff\text{-simps}$   
**apply** (*intro*  $set\text{-subteqD}[OF Fnr.fft\text{-iff-carrier}[of - oe-n - 1]])$   
**subgoal using**  $length\text{-c-dft-odds}$  .  
**subgoal using**  $c\text{-dft-odds-carrier}$  .  
**subgoal using**  $Fnr.length\text{-iff}[OF length\text{-c-dft-odds}]$  **that** **by** *simp*  
**done**  
**have**  $\xi'\text{-residues}: Fnr.\text{to-residue-ring} (\xi' ! j) = z' j \text{ mod } Fnr.n$  **if**  $j < 2 \wedge (oe-n$   
 $- 1)$  **for**  $j$   
**proof** –  
**from that have**  $Fnr.\text{to-residue-ring} (\xi' ! j) = Fnr.\text{to-residue-ring}$   
 $(Fnr.add\text{-fermat}$

```

    (Fnr.divide-by-power-of-2 (c-diffs ! j) (oe-n + prim-root-exponent * j
- 1))
    (Fnr.from-nat-lsbf (replicate (oe-n + 2 ^ n) False @ [True])))
  unfolding  $\xi'$ -def by (simp add: length-c-diffs)
  also have ... = Fnr.to-residue-ring (Fnr.divide-by-power-of-2 (c-diffs ! j)
(oe-n + prim-root-exponent * j - 1))  $\oplus_{F_n}$ 
    Fnr.to-residue-ring (Fnr.from-nat-lsbf (replicate (oe-n + 2 ^ n) False @
[True]))
  apply (intro Fnr.add-fermat-correct)
  subgoal by (intro Fnr.divide-by-power-of-2-closed c-diffs-carrier that)
  subgoal by (intro Fnr.from-nat-lsbf-correct(1))
  done
  also have ... = Fnr.to-residue-ring (c-diffs ! j)  $\otimes_{F_n}$  (invFn 2) [ $\bigwedge_{F_n}$  (oe-n +
prim-root-exponent * j - 1)  $\oplus_{F_n}$ 
    Fnr.to-residue-ring (replicate (oe-n + 2 ^ n) False @ [True]))
  apply (intro arg-cong2[where f = ( $\oplus_{F_n}$ )])
  subgoal by (intro Fnr.divide-by-power-of-2-correct c-diffs-carrier that)
  subgoal by (intro Fnr.from-nat-lsbf-correct(2))
  done
  also have Fnr.to-residue-ring (replicate (oe-n + 2 ^ n) False @ [True]) =
    (2 ^ (oe-n + 2 ^ n)) mod Fnr.n
  by (simp add: zmod-int)
  also have ... = 2 [ $\bigwedge_{F_n}$  (oe-n + 2 ^ n)
    using Fnr.pow-nat-eq[symmetric] by (simp add: zmod-int)
  also have Fnr.to-residue-ring (c-diffs ! j) =
    map Fnr.to-residue-ring
      (Fnr.ifft (prim-root-exponent * 2) c-dft-odds) ! j
  unfolding c-diffs-def using length-c-dft-odds ⟨j < 2 ^ (oe-n - 1)⟩
  apply (intro nth-map[symmetric])
  by (simp add: Fnr.length-fft-iff)
  also have ... = Fnr.NTT ((invFn 2) [ $\bigwedge_{F_n}$  (prim-root-exponent * 2)]) (map
Fnr.to-residue-ring c-dft-odds) ! j
  apply (intro arg-cong2[where f = (!) refl Fnr.ifft-correct[of - oe-n - 1 -
prim-root-exponent]])
  subgoal using length-c-dft-odds .
  subgoal unfolding prim-root-exponent-def by simp
  subgoal unfolding prim-root-exponent-def oe-n-def using n-gt-1 by simp
  subgoal using oe-n-gt-1 by simp
  subgoal apply (intro set-subseteqI) using aux4 ⟨length c-dft-odds = 2 ^
(oe-n - 1)⟩ by fastforce
  done
  also have ... = Fnr.NTT
    (invFn (2 [ $\bigwedge_{F_n}$  (prim-root-exponent * 2)]))
    (map Fnr.to-residue-ring c-dft-odds) ! j
  by (intro arg-cong[where f =  $\lambda a.$  Fnr.NTT a - ! -] Fnr.inv-nat-pow[symmetric]
Fnr.two-is-unit)
  also have ... = Fnr.NTT (invFn ( $\mu$  [ $\bigwedge_{F_n}$  (2::nat)]))
    (map Fnr.to-residue-ring c-dft-odds) ! j
  apply (intro-cong [cong-tag-2 (!), cong-tag-2 Fnr.NTT, cong-tag-1 ( $\lambda j.$ 

```

$inv_{Fn} j]$  more: refl)  
**unfolding**  $\mu$ -def  
**by** (intro Fnr.nat-pow-pow[symmetric] Fnr.two-in-carrier)  
**also have** ...  $\otimes_{Fn} (inv_{Fn} 2) [\uparrow]_{Fn} (oe-n + prim-root-exponent * j - 1) =$   
 $(inv_{Fn} 2) [\uparrow]_{Fn} (oe-n + prim-root-exponent * j - 1) \otimes_{Fn} \dots$   
**apply** (intro Fnr.m-comm)  
**subgoal**  
**apply** (intro set-subseteqD[OF Fnr.NTT-closed])  
**subgoal apply** (intro set-subseteqI) **using** Fnr.to-residue-ring-in-carrier  
**by simp**  
**subgoal by** (intro Fnr.Units-closed Fnr.Units-inv-Units Fnr.Units-pow-closed  
 $\mu$ -Units-Fn)  
**subgoal using**  $\langle j < 2^{\wedge}(oe-n - 1) \rangle$  **by** (simp add: length-c-dft-odds)  
**done**  
**subgoal**  
**by** (intro Fnr.nat-pow-closed Fnr.Units-inv-closed Fnr.two-is-unit)  
**done**  
**finally have** Fnr.to-residue-ring ( $\xi' ! j$ ) =  
 $(c' \text{-mod} ! j \ominus_{Fn} c' \text{-mod} ! (2^{\wedge}(oe-n - 1) + j)) \oplus_{Fn}$   
 $2 [\uparrow]_{Fn} (oe-n + 2^{\wedge} n)$   
**unfolding** aux3[OF  $\langle j < 2^{\wedge}(oe-n - 1) \rangle$ ]  
**using** Fnr.inv-nat-pow[OF  $\mu$ -Units-Fn] **by presburger**  
**also have** ... =  $((c' \text{-mod} ! j \ominus_{Fn} c' \text{-mod} ! (2^{\wedge}(oe-n - 1) + j)) +$   
 $2 [\uparrow]_{Fn} (oe-n + 2^{\wedge} n)) \text{ mod } Fnr.n$   
**by** (intro Fnr.res-add-eq)  
**also have** ... =  $((c' \text{-mod} ! j - (c' \text{-mod} ! (2^{\wedge}(oe-n - 1) + j))) \text{ mod } Fnr.n +$   
 $2 [\uparrow]_{Fn} (oe-n + 2^{\wedge} n)) \text{ mod } Fnr.n$   
**by** (intro-cong [cong-tag-2 (mod), cong-tag-2 (+)] more: refl Fnr.residues-minus-eq)  
**also have** ... =  $((c' ! j) \text{ mod } Fnr.n - (c' ! (2^{\wedge}(oe-n - 1) + j)) \text{ mod } Fnr.n)$   
 $\text{ mod } Fnr.n +$   
 $2 [\uparrow]_{Fn} (oe-n + 2^{\wedge} n)) \text{ mod } Fnr.n$   
**apply** (intro-cong [cong-tag-2 (mod), cong-tag-2 (+), cong-tag-2 (-)] more:  
refl)  
**unfolding**  $c' \text{-mod-def}$  **using**  $\langle j < 2^{\wedge}(oe-n - 1) \rangle$  index-intros[of j]  $\langle \text{length}$   
 $c' = 2^{\wedge} oe-n \rangle$   
**by simp-all**  
**also have** ... =  $(c' ! j - c' ! (2^{\wedge}(oe-n - 1) + j) +$   
 $2 [\uparrow]_{Fn} (oe-n + 2^{\wedge} n)) \text{ mod } Fnr.n$   
**by** (simp only: mod-diff-eq mod-add-left-eq)  
**also have** ... =  $(c' ! j - c' ! (2^{\wedge}(oe-n - 1) + j) +$   
 $2^{\wedge}(oe-n + 2^{\wedge} n)) \text{ mod } Fnr.n$   
**by** (simp only: Fnr.pow-nat-eq mod-add-right-eq)  
**also have** ... =  $z' j \text{ mod } Fnr.n$   
**unfolding**  $z' \text{-def}$  **using**  $\langle j < 2^{\wedge}(oe-n - 1) \rangle$  **by presburger**  
**finally show** Fnr.to-residue-ring ( $\xi' ! j$ ) =  $z' j \text{ mod } Fnr.n$  .  
**qed**

**have**  $\xi' \text{-carrier}$ :  $\xi' ! i \in Fnr.fermat-non-unique-carrier$  **if**  $i < 2^{\wedge}(oe-n - 1)$

```

for i
  proof -
    from that have  $\xi' ! i = \text{Fnr.add-fermat}$ 
      ( $\text{Fnr.divide-by-power-of-2}$  ( $c\text{-diffs} ! i$ )
        ( $oe\text{-}n + \text{prim-root-exponent} * ([0..<2^{\wedge}(oe\text{-}n - 1)] ! i) - 1$ ))
      ( $\text{Fnr.from-nat-lsbf}$  ( $\text{replicate}$  ( $oe\text{-}n + 2^{\wedge}n$ )  $\text{False}$  @  $[\text{True}]$ )) unfolding
 $\xi'$ -def
    apply (intro nth-map2)
    using length-c-diffs by simp-all
    also have ...  $\in \text{Fnr.fermat-non-unique-carrier}$ 
    apply (intro  $\text{Fnr.add-fermat-closed}$ )
    subgoal
      by (intro  $\text{Fnr.divide-by-power-of-2-closed}$  that  $c\text{-diffs-carrier}$ )
    subgoal by (intro  $\text{Fnr.from-nat-lsbf-correct}(1)$ )
    done
    finally show  $\xi' ! i \in \text{Fnr.fermat-non-unique-carrier}$  .
qed
have  $\xi\text{-}\xi'$ :  $\text{Nat-LSBF.to-nat} (\xi ! i) = \text{Nat-LSBF.to-nat} (\xi' ! i) \text{ mod } \text{Fnr.n}$ 
if  $i < 2^{\wedge}(oe\text{-}n - 1)$  for i
  unfolding  $\xi\text{-def}$  using  $\text{Fnr.reduce-correct}[OF \xi'\text{-carrier}[OF that]]$ 
  using that length- $\xi'$  by simp
have  $z'\text{-bounds}$ :  $z' j \geq 0$   $z' j < 2^{\wedge}(oe\text{-}n + 1) * 2^{\wedge}2^{\wedge}n$  if  $j < 2^{\wedge}(oe\text{-}n - 1)$  for j
  proof -
    have  $z' j = c' ! j - c' ! (2^{\wedge}(oe\text{-}n - 1) + j) + 2^{\wedge}(oe\text{-}n + 2^{\wedge}n)$  (is - =
    ?z)
    unfolding  $z'\text{-def}$  using that by simp
    have  $c' ! j \geq 0$   $c' ! j < 2^{\wedge}(oe\text{-}n + 2^{\wedge}n)$ 
      using  $c'\text{-upper-bound}[of j]$   $c'\text{-lower-bound}[of j]$   $\text{index-intros}[of j]$   $\langle j < 2^{\wedge}(oe\text{-}n - 1) \rangle$ 
      by simp-all
    moreover have  $c' ! (2^{\wedge}(oe\text{-}n - 1) + j) \geq 0$   $c' ! (2^{\wedge}(oe\text{-}n - 1) + j) < 2^{\wedge}(oe\text{-}n + 2^{\wedge}n)$ 
      using  $c'\text{-upper-bound}$   $c'\text{-lower-bound}$   $\text{index-intros}[of j]$   $\langle j < 2^{\wedge}(oe\text{-}n - 1) \rangle$ 
    by simp-all
    ultimately have  $?z \geq 0$   $?z < 2^{\wedge}(oe\text{-}n + 2^{\wedge}n) + 2^{\wedge}(oe\text{-}n + 2^{\wedge}n)$ 
      by linarith+
    then have  $?z < 2^{\wedge}(oe\text{-}n + 1 + 2^{\wedge}n)$ 
      by simp
    also have ...  $= 2^{\wedge}(oe\text{-}n + 1) * 2^{\wedge}2^{\wedge}n$  by (simp add: power-add)
    finally show  $z' j \geq 0$   $z' j < 2^{\wedge}(oe\text{-}n + 1) * 2^{\wedge}2^{\wedge}n$  using  $\langle z' j = ?z \rangle$ 
     $\langle ?z \geq 0 \rangle$  by simp-all
  qed
have  $z\text{-}z'$ :  $\text{Fmr.to-residue-ring} (z ! j) = z' j$  if  $j < 2^{\wedge}(oe\text{-}n - 1)$  for j
  proof -
    from that have  $z ! j = \text{solve-special-residue-problem } n (\xi ! j) (\eta ! j)$ 
    unfolding  $z\text{-def}$  using length- $\xi$  length- $\eta$  by simp
    then have  $\text{solves-special-residue-problem} (\text{Nat-LSBF.to-nat} (z ! j)) n (\text{Nat-LSBF.to-nat} (\xi ! j)) (\text{Nat-LSBF.to-nat} (\eta ! j))$ 

```

```

apply (intro solve-special-residue-problem-correct)
subgoal using n-gt-1 by simp
subgoal using η-carrier[OF that] by simp
subgoal using ξ-ξ'[OF that] by simp
subgoal .
done
moreover have solves-special-residue-problem (nat (z' j)) n (Nat-LSBF.to-nat
(ξ ! j)) (Nat-LSBF.to-nat (η ! j))
unfolding solves-special-residue-problem-def
apply (intro conjI)
subgoal
apply (intro iffD2[OF nat-int-comparison(2)])
unfolding nat-0-le[of z' j, OF z'-bounds(1)][OF that]
unfolding int-ops
apply (intro order.strict-trans2[OF z'-bounds(2)][OF that])
apply (intro mult-mono)
unfolding oe-n-def by simp-all
subgoal unfolding ξ-ξ'[OF that] using ξ'-residues[OF that, symmetric]
apply (intro iffD1[OF int-int-eq])
using nat-0-le[OF z'-bounds(1)][OF that], symmetric] zmod-int
by simp
subgoal
proof -
have z' j mod 2 ^ (n + 2) = int (Nat-LSBF.to-nat (η ! j)) mod 2 ^ (n +
2)
using η-residues[OF that] unfolding Znr.to-residue-ring-def by simp
also have ... = int (Nat-LSBF.to-nat (η ! j) mod 2 ^ (n + 2))
by (simp add: zmod-int)
also have ... = int (Nat-LSBF.to-nat (η ! j))
apply (intro arg-cong[where f = int])
using to-nat-length-bound η-carrier[OF that] mod-less by metis
finally show ?thesis
apply (intro iffD1[OF int-int-eq])
using nat-0-le[OF z'-bounds(1)][OF that] zmod-int by simp
qed
done
ultimately have nat (z' j) = Nat-LSBF.to-nat (z ! j)
using special-residue-problem-unique-solution by simp
then have int (Nat-LSBF.to-nat (z ! j)) = z' j using nat-0-le[OF z'-bounds(1)][OF
that]] by argo
have z' j ∈ carrier Fm
using z'-carrier z'-z'' index-intros that by simp
then have z' j mod Fmr.n = z' j
apply (intro Fmr.carrier-mod-eq) .
with ⟨int (Nat-LSBF.to-nat (z ! j)) = z' j⟩ show Fmr.to-residue-ring (z ! j)
= z' j
by simp
qed

```

```

have result-value: Fmr.to-residue-ring result = ( $\bigoplus_{Fm} i \leftarrow [0..<2^{\wedge} oe-n]. (z'$ 
i)  $\otimes_{Fm} 2 [\bigwedge]_{Fm} (i * 2^{\wedge} (n - 1))$ )
proof -
  have Fmr.to-residue-ring result = Fmr.to-residue-ring z-sum
    unfolding result-def by (rule Fmr.from-nat-lsbf-correct(2))
  also have ... = int (Nat-LSBF.to-nat z-sum) mod int Fmr.n
    unfolding Fmr.to-residue-ring.simps by simp
  also have Nat-LSBF.to-nat z-sum = ( $\sum_{i \leftarrow [0..<length (z-filled @ z-consts)].$ 
Nat-LSBF.to-nat ((z-filled @ z-consts) ! i) *  $2^{\wedge} (i * 2^{\wedge} (n - 1))$ )
    unfolding z-sum-def
    apply (intro combine-z-correct)
  subgoal by simp
  subgoal premises prems for zi
    unfolding set-append
  proof (cases zi  $\in$  set z-filled)
    case True
      then obtain i where zi = fill ( $2^{\wedge} (n - 1)$ ) i i  $\in$  set z
        unfolding z-filled-def set-map by auto
        then show ?thesis using length-fill' by simp
    next
      case False
        then have zi = replicate (oe-n +  $2^{\wedge} n$ ) False @ [True]
          using prems unfolding z-consts-def by simp
        then have length zi  $\geq 2^{\wedge} n$  by simp
        moreover have  $2^{\wedge} n \geq (2::nat)^{\wedge} (n - 1)$  by simp
        ultimately show ?thesis by linarith
    qed
  done
  finally have Fmr.to-residue-ring result = ( $\bigoplus_{Fm} i \leftarrow [0..<length (z-filled @$ 
z-consts)]. int (Nat-LSBF.to-nat ((z-filled @ z-consts) ! i) *  $2^{\wedge} (i * 2^{\wedge} (n - 1))$ )
mod Fmr.n)
    unfolding Fmr.monoid-sum-list-eq-sum-list' sum-list-int .
  also have ... = ( $\bigoplus_{Fm} i \leftarrow [0..<2^{\wedge} oe-n].$  int (Nat-LSBF.to-nat ((z-filled @
z-consts) ! i) *  $2^{\wedge} (i * 2^{\wedge} (n - 1))$ ) mod Fmr.n)
    apply (intro arg-cong2[where f = Fmr.monoid-sum-list] refl arg-cong[where
f =  $\lambda i. [0..<i]$ ])
    by (simp add: length-z-filled length-z-consts two-pow-oe-n-as-halves)
  also have ... = ( $\bigoplus_{Fm} i \leftarrow [0..<2^{\wedge} oe-n]. (z' i) \otimes_{Fm} 2 [\bigwedge]_{Fm} (i * 2^{\wedge} (n -$ 
1)))
    apply (intro Fmr.monoid-sum-list-cong)
  subgoal premises prems for i
  proof (cases i <  $2^{\wedge} (oe-n - 1)$ )
    case True
      then have int (Nat-LSBF.to-nat ((z-filled @ z-consts) ! i) *  $2^{\wedge} (i * 2^{\wedge}$ 
(n - 1))) mod int Fmr.n
        = int (Nat-LSBF.to-nat (z-filled ! i) *  $2^{\wedge} (i * 2^{\wedge} (n - 1))$ ) mod int
Fmr.n
      using length-z-filled nth-append by metis

```

**also have** ... =  $\text{int } (\text{Nat-LSBF.to-nat } (z ! i) * 2 ^{(i * 2 ^{(n - 1)})}) \text{ mod int } Fmr.n$   
**unfolding** *z-filled-def* **using** *length-z* *True* **by** *simp*  
**also have** ... =  $(\text{int } (\text{Nat-LSBF.to-nat } (z ! i)) \text{ mod int } Fmr.n * 2 ^{(i * 2 ^{(n - 1)})}) \text{ mod int } Fmr.n$   
**by** (*simp add: mod-mult-left-eq*)  
**also have** ... =  $(z' i * 2 ^{(i * 2 ^{(n - 1)})}) \text{ mod int } Fmr.n$   
**using** *z-z'[OF True]* **unfolding** *Fmr.to-residue-ring.simps* **by** *argo*  
**also have** ... =  $(z' i * (2 ^{(i * 2 ^{(n - 1)})}) \text{ mod int } Fmr.n)) \text{ mod int } Fmr.n$   
**by** (*simp add: mod-mult-right-eq*)  
**also have** ... =  $z' i \otimes_{Fm} (2 ^{(i * 2 ^{(n - 1)})}) \text{ mod int } Fmr.n$   
**by** (*rule Fmr.res-mult-eq[symmetric]*)  
**also have**  $2 ^{(i * 2 ^{(n - 1)})} \text{ mod int } Fmr.n = 2 \lceil_{Fm} (i * 2 ^{(n - 1)})$   
**by** (*rule Fmr.pow-nat-eq[symmetric]*)  
**finally show** *?thesis* .  
**next**  
**case** *False*  
**define** *j* **where**  $j = i - 2 ^{(oe-n - 1)}$   
**then have**  $i = 2 ^{(oe-n - 1)} + j$   $j < 2 ^{(oe-n - 1)}$   
**subgoal using** *False* **by** *linarith*  
**subgoal using** *j-def* *prems* *two-pow-oe-n-as-halves* **by** *simp*  
**done**  
**then have**  $\text{int } (\text{Nat-LSBF.to-nat } ((z\text{-filled } @ z\text{-consts}) ! i) * 2 ^{(i * 2 ^{(n - 1)})}) \text{ mod int } Fmr.n =$   
 $\text{int } (\text{Nat-LSBF.to-nat } (z\text{-consts } ! j) * 2 ^{(i * 2 ^{(n - 1)})}) \text{ mod int } Fmr.n$   
**using** *length-z-filled* *nth-append-length-plus* **by** *metis*  
**also have** ... =  $\text{int } (\text{Nat-LSBF.to-nat } (\text{replicate } (oe-n + 2 ^{n}) \text{ False } @ [\text{True}]) * 2 ^{(i * 2 ^{(n - 1)})}) \text{ mod int } Fmr.n$   
**unfolding** *z-consts-def* **using**  $\langle j < 2 ^{(oe-n - 1)} \rangle$  **by** *simp*  
**also have** ... =  $\text{int } (2 ^{(oe-n + 2 ^{n})}) * 2 ^{(i * 2 ^{(n - 1)})} \text{ mod int } Fmr.n$   
**by** *simp*  
**also have** ... =  $z' i * 2 ^{(i * 2 ^{(n - 1)})} \text{ mod int } Fmr.n$   
**unfolding** *z'-def* **using** *False* **by** *simp*  
**also have** ... =  $z' i \otimes_{Fm} (2 ^{(i * 2 ^{(n - 1)})}) \text{ mod int } Fmr.n$   
**by** (*simp add: mod-mult-right-eq Fmr.res-mult-eq*)  
**also have**  $2 ^{(i * 2 ^{(n - 1)})} \text{ mod int } Fmr.n = 2 \lceil_{Fm} (i * 2 ^{(n - 1)})$   
**by** (*rule Fmr.pow-nat-eq[symmetric]*)  
**finally show** *?thesis* .  
**qed**  
**done**  
**finally show** *?thesis* .  
**qed**

**have** *Fmr.to-residue-ring result* = *Fmr.to-residue-ring a*  $\otimes_{Fm}$  *Fmr.to-residue-ring*

**b**  
**using** *result-value result0* **by** *argo*  
  
**moreover have** *result*  $\in$  *Fmr.fermat-non-unique-carrier*  
**unfolding** *result-def*  
**by** (*rule Fmr.from-nat-lsbf-correct(1)*)  
  
**ultimately show** *?thesis*  
**unfolding** *result-eq Fm-def int-lsbf-fermat.Fn-def* **by** *simp*  
**qed**  
**qed**

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

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

**using** *m-def bitsize-length defs* **by** *fastforce+*

**lemma**

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

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

**using** *length-a length-b* **by** (*simp-all add: m-def nat-le-real-less nat-less-real-le*)

**lemma** *length-fill-a*:  $\text{length fill-a} = 2 \wedge (m + 1)$   
**unfolding** *fill-a-def car-len-def*  
**by** (*intro length-fill length-a'*)

**lemma** *length-fill-b*:  $\text{length fill-b} = 2 \wedge (m + 1)$   
**unfolding** *fill-b-def car-len-def*  
**by** (*intro length-fill length-b'*)

**sublocale** *fm*: *int-lsbfermat m* .

**definition** *Fm* **where** *Fm* = *residue-ring (int-lsbfermat.n m)*  
**sublocale** *Fmr*: *residues fm.n Fm*  
**rewrites** *fm-Fm*: *fm.Fn*  $\equiv$  *Fm*  
**unfolding** *Fm-def fm.Fn-def* **by** (*rule fm.residues-axioms reflexive*) $+$

**lemma** *fill-a-carrier*[*simp, intro*]:  $\text{fill-a} \in \text{fm.fermat-non-unique-carrier}$   
**by** (*intro fm.fermat-non-unique-carrierI length-fill-a*)

**lemma** *fill-b-carrier*[*simp, intro*]:  $\text{fill-b} \in \text{fm.fermat-non-unique-carrier}$   
**by** (*intro fm.fermat-non-unique-carrierI length-fill-b*)

**lemma** *fm-result-carrier*[*simp, intro*]:  $\text{fm-result} \in \text{fm.fermat-non-unique-carrier}$   
**unfolding** *fm-result-def*  
**by** (*intro conjunct2[OF schoenhage-strassen-correct'] fill-a-carrier fill-b-carrier*)

**lemma** *ssc'*:  $\text{fm.to-residue-ring fm-result} = \text{fm.to-residue-ring fill-a} \otimes_{Fm} \text{fm.to-residue-ring fill-b}$   
**and**  $\text{fm-result} \in \text{int-lsbfermat.fermat-non-unique-carrier } m$   
**unfolding** *atomize-conj fm-result-def fm-Fm[symmetric]*  
**by** (*intro schoenhage-strassen-correct' fill-a-carrier fill-b-carrier*)

**end**

**theorem** *schoenhage-strassen-mul-correct*:  $\text{Nat-LSBF.to-nat (schoenhage-strassen-mul } a \text{ } b) = \text{Nat-LSBF.to-nat } a * \text{Nat-LSBF.to-nat } b$   
**proof** –  
**interpret** *schoenhage-strassen-mul-context a b* .

**have**  $\text{int (Nat-LSBF.to-nat } a) * \text{int (Nat-LSBF.to-nat } b) < \text{int-lsbfermat.n } m$   
**proof** –  
**have**  $\text{Nat-LSBF.to-nat } a < 2 \wedge \text{length } a$   $\text{Nat-LSBF.to-nat } b < 2 \wedge \text{length } b$  **by**  
(*intro to-nat-length-bound*) $+$   
**moreover have**  $(2::\text{nat}) \wedge \text{length } a < 2 \wedge 2 \wedge (m - 1)$   $(2::\text{nat}) \wedge \text{length } b <$   
 $2 \wedge 2 \wedge (m - 1)$   
**using** *length-a length-b* **by** *simp-all*  
**ultimately have**  $\text{Nat-LSBF.to-nat } a * \text{Nat-LSBF.to-nat } b < 2 \wedge 2 \wedge (m - 1)$   
 $* 2 \wedge 2 \wedge (m - 1)$   
**by** (*metis bot-nat-0.extremum max.absorb3 max-less-iff-conj mult-strict-mono pos2 zero-less-power*)

```

also have ... = 2 ^ 2 ^ m by (simp add: power2-eq-square power-even-eq m-def)
finally show ?thesis by (simp add: nat-int-comparison(2))
qed
then have int-lsbfermat.to-residue-ring m a ⊗Fm int-lsbfermat.to-residue-ring
m b =
  int (Nat-LSBF.to-nat a) * int (Nat-LSBF.to-nat b)
by (simp add: Fmr.res-mult-eq int-lsbfermat.to-residue-ring.simps mod-mult-eq)
also have int-lsbfermat.to-residue-ring m a = int-lsbfermat.to-residue-ring m
fill-a
  unfolding int-lsbfermat.to-residue-ring.simps defs by simp
also have int-lsbfermat.to-residue-ring m b = int-lsbfermat.to-residue-ring m
fill-b
  unfolding int-lsbfermat.to-residue-ring.simps defs by simp
finally have c: int-lsbfermat.to-residue-ring m fill-a ⊗Fm int-lsbfermat.to-residue-ring
m fill-b =
  int (Nat-LSBF.to-nat a) * int (Nat-LSBF.to-nat b) .

have schoenhage-strassen-mul a b = int-lsbfermat.reduce m (schoenhage-strassen
m fill-a fill-b)
  by (simp only: schoenhage-strassen-mul-def Let-def defs)
then have Nat-LSBF.to-nat (schoenhage-strassen-mul a b) = Nat-LSBF.to-nat
(schoenhage-strassen m fill-a fill-b) mod int-lsbfermat.n m
  using fm.reduce-correct[OF fm-result-carrier] fm-result-def by algebra
also have ... = nat (int (Nat-LSBF.to-nat (schoenhage-strassen m fill-a fill-b)
mod int-lsbfermat.n m))
  by simp
also have ... = nat (int-lsbfermat.to-residue-ring m (schoenhage-strassen m
fill-a fill-b))
  unfolding int-lsbfermat.to-residue-ring.simps
by (intro arg-cong[where f = nat] zmod-int)
also have ... = nat (
  int-lsbfermat.to-residue-ring m fill-a ⊗Fm
  int-lsbfermat.to-residue-ring m fill-b)
  apply (intro arg-cong[where f = nat]) using ssc' unfolding fm-result-def .
also have ... = nat (int (Nat-LSBF.to-nat a) * int (Nat-LSBF.to-nat b))
  by (intro arg-cong[where f = nat] c)
also have ... = Nat-LSBF.to-nat a * Nat-LSBF.to-nat b
  by (simp add: nat-mult-distrib)
finally show ?thesis .
qed
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  $\xi$   $\eta = 1$  do {
  n2  $\leftarrow$  n +t 2;
   $\xi$ mod  $\leftarrow$  take-tm n2  $\xi$ ;
   $\delta$   $\leftarrow$  int-lsbf-mod.subtract-mod-tm n2  $\eta$   $\xi$ mod;
  pown  $\leftarrow$  2 $\hat{t}$  n;
   $\delta$ -shifted  $\leftarrow$   $\delta$  >>nt pown;
   $\delta$ 1  $\leftarrow$   $\delta$ -shifted +nt  $\delta$ ;
   $\xi$  +nt  $\delta$ 1
}

lemma val-solve-special-residue-problem-tm[simp, val-simp]:
  val (solve-special-residue-problem-tm n  $\xi$   $\eta$ ) = solve-special-residue-problem n  $\xi$   $\eta$ 
proof –
  have a: n + 2 > 0 by simp
  show ?thesis
  unfolding solve-special-residue-problem-tm-def solve-special-residue-problem-def
  using int-lsbf-mod.val-subtract-mod-tm[OF int-lsbf-mod.intro[OF a]]
  by (simp add: Let-def)
qed

lemma time-solve-special-residue-problem-tm-le:
  time (solve-special-residue-problem-tm n  $\xi$   $\eta$ )  $\leq$  245 + 74 * 2 $\hat{n}$  + 55 * length
 $\eta$  + 2 * length  $\xi$ 
proof –
  define n2 where n2 = n + 2
  define  $\xi$ mod where  $\xi$ mod = take n2  $\xi$ 
  define  $\delta$  where  $\delta$  = int-lsbf-mod.subtract-mod n2  $\eta$   $\xi$ mod
  define pown where pown = (2::nat) $\hat{n}$ 
  define  $\delta$ -shifted where  $\delta$ -shifted =  $\delta$  >>n pown
  define  $\delta$ 1 where  $\delta$ 1 = add-nat  $\delta$ -shifted  $\delta$ 
  note defs = n2-def  $\xi$ mod-def  $\delta$ -def pown-def  $\delta$ -shifted-def  $\delta$ 1-def

  interpret mr: int-lsbf-mod n2 apply (intro int-lsbf-mod.intro) unfolding n2-def
by simp

  have length- $\xi$ mod-le: length  $\xi$ mod  $\leq$  n2 unfolding  $\xi$ mod-def by simp
  have length- $\delta$ -le: length  $\delta$   $\leq$  max n2 (length  $\eta$ )
  unfolding  $\delta$ -def mr.subtract-mod-def if-distrib[where f = length] mr.length-reduce
  apply (estimation estimate: conjunct2[OF subtract-nat-aux])
  using length- $\xi$ mod-le by auto
  have length- $\delta$ 1add-le: max (length  $\delta$ -shifted) (length  $\delta$ )  $\leq$  2 $\hat{n}$  + (n + 2) +
length  $\eta$ 
  unfolding  $\delta$ -shifted-def pown-def

```

```

using length-δ-le unfolding n2-def by simp

have time (solve-special-residue-problem-tm n ξ η) =
  n + 1 + time (take-tm n2 ξ) + time (int-lsbf-mod.subtract-mod-tm n2 η ξ mod)
+
  time (2 ^ t n) +
  time (δ >>_nt pown) +
  time (δ-shifted +_nt δ) +
  time (ξ +_nt δ1) +
  1
unfolding solve-special-residue-problem-tm-def tm-time-simps
by (simp del: One-nat-def add-2-eq-Suc' add: add.assoc[symmetric] defs[symmetric])
also have ... ≤ n + 1 + (n + 3) + (118 + 51 * (n + 2 + length η)) +
  (3 * 2 ^ Suc n + 5 * n + 1) +
  (2 * 2 ^ n + 3) +
  (2 * 2 ^ n + 2 * length η + 2 * n + 7) +
  (2 * length ξ + 2 * 2 ^ n + 2 * n + 2 * length η + 9) +
  1
apply (intro add-mono order.refl)
subgoal apply (estimation estimate: time-take-tm-le) unfolding n2-def by
simp
subgoal
  apply (estimation estimate: mr.time-subtract-mod-tm-le)
  apply (estimation estimate: length-ξmod-le)
  apply (estimation estimate: Nat-max-le-sum[of length η])
  by (simp add: n2-def Nat-max-le-sum)
subgoal by (rule time-power-nat-tm-le)
subgoal unfolding time-shift-right-tm pown-def by simp
subgoal
  apply (estimation estimate: time-add-nat-tm-le)
  apply (estimation estimate: length-δ1add-le)
  by simp
subgoal
  apply (estimation estimate: time-add-nat-tm-le)
  unfolding δ1-def
  apply (estimation estimate: length-add-nat-upper)
  apply (estimation estimate: length-δ1add-le)
  apply (estimation estimate: Nat-max-le-sum)
  by simp
done
also have ... = 245 + 12 * 2 ^ n + 62 * n + 55 * length η + 2 * length ξ
unfolding n2-def by simp
also have ... ≤ 245 + 74 * 2 ^ n + 55 * length η + 2 * length ξ
  using less-exp[of n] by simp
finally show ?thesis .
qed

fun combine-z-aux-tm where
combine-z-aux-tm l acc [] = 1 rev-tm acc ≫= 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) ← split-at-tm l z1;
  r ← 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

**by** (*induction l acc zs rule: combine-z-aux.induct; simp*)

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

**assumes**  $\bigwedge z. z \in \text{set } zs \implies \text{length } z \leq lz$

**assumes**  $\text{length } z \leq lz + 1$

**assumes**  $l > 0$

**shows**  $\text{time } (\text{combine-z-aux-tm } l \text{ acc } (z \# zs)) \leq (2 * l + 2 * lz + 7) * \text{length } zs + 3 * (\text{length } \text{acc} + \text{length } zs) + \text{length } (\text{concat } \text{acc}) + \text{length } zs * l + lz + 9$

**using** *assms proof* (*induction zs arbitrary: acc z*)

**case** *Nil*

**then show** *?case*

**by** (*simp del: One-nat-def*)

**next**

**case** (*Cons z1 zs*)

**then have** *len-drop-z*:  $\text{length } (\text{drop } l \ z) \leq lz$  **by** *simp*

**have** *lena*:  $\text{length } (\text{add-nat } (\text{drop } l \ z) \ z1) \leq lz + 1$

**apply** (*estimation estimate: length-add-nat-upper*)

**using** *len-drop-z Cons.prem*s **by** *simp*

**have**  $\text{time } (\text{combine-z-aux-tm } l \ \text{acc } (z \# z1 \# zs)) =$

$\text{time } (\text{split-at-tm } l \ z) +$

$\text{time } (\text{drop } l \ z +_{nt} \ z1) +$

$\text{time } (\text{combine-z-aux-tm } l \ (\text{take } l \ z \# \text{acc}) ((\text{drop } l \ z +_n \ z1) \# zs)) + 1$

**by** *simp*

**also have**  $\dots \leq$

$(2 * l + 3) +$

$(2 * lz + 3) +$

$((2 * l + 2 * lz + 7) * \text{length } zs + 3 * (\text{length } (\text{take } l \ z \# \text{acc}) + \text{length } zs)$

+

$\text{length } (\text{concat } (\text{take } l \ z \# \text{acc})) + \text{length } zs * l + lz + 9) + 1$

**apply** (*intro add-mono order.refl*)

**subgoal by** (*simp add: time-split-at-tm*)

**subgoal**

**apply** (*estimation estimate: time-add-nat-tm-le*)

**using** *len-drop-z Cons.prem*s **by** *simp*

**subgoal**

**apply** (*intro Cons.IH*)

**subgoal using** *Cons.prem*s **by** *simp*

**subgoal using** *lena* .

**subgoal using** *Cons.prem*s(3) .

**done**

**done**  
**also have** ... =  $(2 * l + 2 * lz + 7) * \text{length } (z1 \# zs) + 3 * (\text{length } acc + 1 + \text{length } zs) +$   
 $\text{length } (\text{concat } acc) + \text{length } (\text{take } l z) + \text{length } zs * l + lz + 9$   
**by** *simp*  
**also have** ...  $\leq (2 * l + 2 * lz + 7) * \text{length } (z1 \# zs) + 3 * (\text{length } acc + 1 + \text{length } zs) +$   
 $\text{length } (\text{concat } acc) + l + \text{length } zs * l + lz + 9$   
**apply** (*intro add-mono order.refl*) **by** *simp*  
**also have** ... =  $(2 * l + 2 * lz + 7) * \text{length } (z1 \# zs) + 3 * (\text{length } acc + \text{length } (z1 \# zs)) +$   
 $\text{length } (\text{concat } acc) + \text{length } (z1 \# zs) * l + lz + 9$   
**by** *simp*  
**finally show** ?*case* .  
**qed**

**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*  
**unfolding** *combine-z-tm-def combine-z-def* **by** *simp*

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

**assumes**  $\bigwedge z. z \in \text{set } zs \implies \text{length } z \leq lz$

**assumes**  $l > 0$

**shows**  $\text{time } (\text{combine-z-tm } l \text{ } zs) \leq 10 + (3 * l + 2 * lz + 10) * \text{length } zs$

**proof** (*cases zs*)

**case** *Nil*

**then have**  $\text{time } (\text{combine-z-tm } l \text{ } zs) = 5$

**unfolding** *combine-z-tm-def* **by** *simp*

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

**next**

**case** (*Cons z zs'*)

**then have**  $\text{time } (\text{combine-z-tm } l \text{ } zs) = \text{time } (\text{combine-z-aux-tm } l \text{ } [] (z \# zs')) + 1$

**unfolding** *combine-z-tm-def* **by** *simp*

**also have** ...  $\leq (2 * l + 2 * lz + 7) * \text{length } zs' + 3 * (\text{length } ([] :: \text{nat-lsbf list}) + \text{length } zs') +$   
 $\text{length } (\text{concat } ([] :: \text{nat-lsbf list})) +$

$\text{length } zs' * l + lz + 9 + 1$

**apply** (*intro add-mono time-combine-z-aux-tm-le order.refl*)

**subgoal using** *Cons assms* **by** *simp*

**subgoal using** *Cons assms* **by** *force*

**subgoal using** *assms(2)* .

**done**

**also have** ... =  $10 + (3 * l + 2 * lz + 10) * \text{length } zs' + lz$

**by** (*simp add: add-mult-distrib*)

**also have** ...  $\leq 10 + (3 * l + 2 * lz + 10) * \text{length } zs$

**unfolding** *Cons* **by** *simp*

**finally show** ?*thesis* .

**qed**

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

by *linarith*

**function** *schoenhage-strassen-tm* ::  $\text{nat} \Rightarrow \text{nat-lsbf} \Rightarrow \text{nat-lsbf} \Rightarrow \text{nat-lsbf tm}$  **where**  
*schoenhage-strassen-tm* *m a b =1* do {

```

  m-le-3  $\leftarrow m <_t 3$ ;
  if m-le-3 then do {
    ab  $\leftarrow a *_t b$ ;
    int-lsbf-fermat.from-nat-lsbf-tm m ab
  } else do {
    odd-m  $\leftarrow \text{odd-tm } m$ ;
    n  $\leftarrow$  (if odd-m then do {
      m1  $\leftarrow m +_t 1$ ;
      m1  $\text{div}_t 2$ 
    } else do {
      m2  $\leftarrow m +_t 2$ ;
      m2  $\text{div}_t 2$ 
    });
    n-plus-1  $\leftarrow n +_t 1$ ;
    n-minus-1  $\leftarrow n -_t 1$ ;
    n-plus-2  $\leftarrow n +_t 2$ ;
    oe-n  $\leftarrow$  (if odd-m then return n-plus-1 else return n);
    segment-lens  $\leftarrow 2 \hat{^}_t n\text{-minus-1}$ ;
    a'  $\leftarrow \text{subdivide-tm } \text{segment-lens } a$ ;
    b'  $\leftarrow \text{subdivide-tm } \text{segment-lens } b$ ;
     $\alpha \leftarrow \text{map-tm } (\text{int-lsbf-mod.reduce-tm } n\text{-plus-2}) a'$ ;
    three-n  $\leftarrow 3 *_t n$ ;
    pad-length  $\leftarrow \text{three-n} +_t 5$ ;
     $\alpha\text{-padded} \leftarrow \text{map-tm } (\text{fill-tm } \text{pad-length}) \alpha$ ;
    u  $\leftarrow \text{concat-tm } \alpha\text{-padded}$ ;
     $\beta \leftarrow \text{map-tm } (\text{int-lsbf-mod.reduce-tm } n\text{-plus-2}) b'$ ;
     $\beta\text{-padded} \leftarrow \text{map-tm } (\text{fill-tm } \text{pad-length}) \beta$ ;
    v  $\leftarrow \text{concat-tm } \beta\text{-padded}$ ;
    oe-n-plus-1  $\leftarrow \text{oe-n} +_t 1$ ;
    two-pow-oe-n-plus-1  $\leftarrow 2 \hat{^}_t \text{oe-n-plus-1}$ ;
    uv-length  $\leftarrow \text{pad-length} *_t \text{two-pow-oe-n-plus-1}$ ;
    uv-unpadded  $\leftarrow \text{karatsuba-mul-nat-tm } u v$ ;
    uv  $\leftarrow \text{ensure-length-tm } \text{uv-length } \text{uv-unpadded}$ ;
    oe-n-minus-1  $\leftarrow \text{oe-n} -_t 1$ ;
    two-pow-oe-n-minus-1  $\leftarrow 2 \hat{^}_t \text{oe-n-minus-1}$ ;
     $\gamma s \leftarrow \text{subdivide-tm } \text{pad-length } uv$ ;
     $\gamma \leftarrow \text{subdivide-tm } \text{two-pow-oe-n-minus-1 } \gamma s$ ;
     $\gamma 0 \leftarrow \text{nth-tm } \gamma 0$ ;
     $\gamma 1 \leftarrow \text{nth-tm } \gamma 1$ ;
     $\gamma 2 \leftarrow \text{nth-tm } \gamma 2$ ;
     $\gamma 3 \leftarrow \text{nth-tm } \gamma 3$ ;
     $\eta \leftarrow \text{map4-tm}$ 

```

```

(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';
b'-carrier ← map-tm (fill-tm fn-carrier-len) b';
a-dft ← int-lsbf-fermat.fft-tm n prim-root-exponent a'-carrier;
b-dft ← int-lsbf-fermat.fft-tm n prim-root-exponent b'-carrier;
a-dft-odds ← evens-odds-tm False a-dft;
b-dft-odds ← evens-odds-tm False b-dft;
c-dft-odds ← map2-tm (schoenhage-strassen-tm n) a-dft-odds b-dft-odds;
prim-root-exponent-2 ← prim-root-exponent *t 2;
c-diffs ← int-lsbf-fermat.ifft-tm n prim-root-exponent-2 c-dft-odds;
two-pow-oe-n ← 2 ^t oe-n;
interval1 ← upt-tm 0 two-pow-oe-n-minus-1;
interval2 ← upt-tm two-pow-oe-n-minus-1 two-pow-oe-n;
two-pow-n ← 2 ^t n;
oe-n-plus-two-pow-n ← oe-n +t two-pow-n;
oe-n-plus-two-pow-n-zeros ← replicate-tm oe-n-plus-two-pow-n False;
oe-n-plus-two-pow-n-one ← oe-n-plus-two-pow-n-zeros @t [True];
ξ' ← map2-tm (λx y. do {
  v1 ← prim-root-exponent *t y;
  v2 ← oe-n +t v1;
  v3 ← v2 -t 1;
  summand1 ← int-lsbf-fermat.divide-by-power-of-2-tm x v3;
  summand2 ← int-lsbf-fermat.from-nat-lsbf-tm n oe-n-plus-two-pow-n-one;
  int-lsbf-fermat.add-fermat-tm n summand1 summand2
})
c-diffs interval1;
ξ ← map-tm (int-lsbf-fermat.reduce-tm n) ξ';
z ← map2-tm (solve-special-residue-problem-tm n) ξ η;
z-filled ← map-tm (fill-tm segment-lens) z;
z-consts ← replicate-tm two-pow-oe-n-minus-1 oe-n-plus-two-pow-n-one;
z-complete ← z-filled @t z-consts;
z-sum ← combine-z-tm segment-lens z-complete;
result ← int-lsbf-fermat.from-nat-lsbf-tm m z-sum;
return result
}
}
by pat-completeness auto
termination

```



```

unfolding z-filled-def defs'[symmetric] by (rule refl)
lemma z-sum-def': z-sum = combine-z segment-lens z-complete
unfolding z-sum-def defs'[symmetric] by (rule refl)

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

lemma segment-lens-pos: segment-lens > 0 unfolding segment-lens-def by simp

lemma length-γs: length γs = 2 ^ (oe-n + 1)
using scwv(1) unfolding defs[symmetric] .
lemma length-γs': length γs = 2 ^ (oe-n - 1) * 4
using two-pow-Suc-oe-n-as-prod length-γs unfolding defs[symmetric]
by simp

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
unfolding defs' using scγ by simp-all

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
by (intro int-lsbfermat.val-fft-tm[where m = oe-n] A.length-num-blocks-carrier)
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
by (intro int-lsbfermat.val-fft-tm[where m = oe-n] B.length-num-blocks-carrier)

lemma val-iff1[simp, val-simp]: val (int-lsbfermat.iff1-tm n (prim-root-exponent *
2) c-dft-odds) =
  int-lsbfermat.iff1 n (prim-root-exponent * 2) c-dft-odds
apply (intro int-lsbfermat.val-iff1-tm[where m = oe-n - 1])
apply (simp add: c-dft-odds-def)
done

end

lemma val-schoenhage-strassen-tm[simp, val-simp]:
assumes a ∈ int-lsbfermat.fermat-non-unique-carrier m
assumes b ∈ int-lsbfermat.fermat-non-unique-carrier m
shows val (schoenhage-strassen-tm m a b) = schoenhage-strassen m a b
using assms proof (induction m arbitrary: a b rule: less-induct)
case (less m)
show ?case
proof (cases m < 3)
case True

```

```

then show ?thesis
  unfolding schoenhage-strassen-tm.simps[of m a b] val-simps
  unfolding schoenhage-strassen.simps[of m a b]
  using int-lsbfermat.val-from-nat-lsbfermat by simp
next
case False

interpret schoenhage-strassen-context m a b
  apply unfold-locales using False less.prem by simp-all

have val-ih: map2 (λx y. val (schoenhage-strassen-tm n x y)) A.num-dft-odds
B.num-dft-odds =
  map2 (λx y. schoenhage-strassen n x y) A.num-dft-odds B.num-dft-odds
  apply (intro map-cong refl)
  subgoal premises prems for p
  proof -
    from prems set-zip obtain i
      where i-le: i < min (length A.num-dft-odds) (length B.num-dft-odds)
        and p-i: p = (A.num-dft-odds ! i, B.num-dft-odds ! i)
      by blast
    then have i < 2 ^ (oe-n - 1)
      using A.length-num-dft-odds by simp
    show ?thesis unfolding p-i prod.case
    apply (intro less.IH n-lt-m set-subseteqD A.num-dft-odds-carrier B.num-dft-odds-carrier)
      using i-le by simp-all
  qed
done

have val (schoenhage-strassen-tm m a b) = result
  unfolding schoenhage-strassen-tm.simps[of m a b]
  unfolding val-simp
  val-times-nat-tm
  val-subdivide-tm[OF segment-lens-pos] val-subdivide-tm[OF pad-length-gt-0]
  Znr.val-reduce-tm Znr.val-subtract-mod-tm Znr.val-add-mod-tm
  val-nth-γ val-subdivide-tm[OF two-pow-pos] val-fft1 val-fft2 val-ih val-iff
  defs[symmetric] Let-def
  val-subdivide-tm[OF two-pow-pos] Fnr.val-iff-tm[OF length-c-dft-odds]
  using False by argo
then show ?thesis using result-eq by argo
qed
qed

fun schoenhage-strassen-Fm-bound where
schoenhage-strassen-Fm-bound m = (if m < 3 then 5336 else
  let n = (if odd m then (m + 1) div 2 else (m + 2) div 2);
  oe-n = (if odd m then n + 1 else n) in
  23525 * 2 ^ m + 8093 * (n * 2 ^ (2 * n)) + 8410 +
  time-karatsuba-mul-nat-bound ((3 * n + 5) * 2 ^ oe-n) +
  4 * karatsuba-lower-bound +

```

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

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

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

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

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

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

**using** *assms* **proof** (*induction m arbitrary: a b rule: less-induct*)

**case** (*less m*)

**consider**  $m = 0 \mid m \geq 1 \wedge m < 3 \mid \neg m < 3$  **by** *linarith*

**then show** *?case*

**proof** *cases*

**case** *1*

**from** *less.prem*s *int-lsbfermat.fermat-carrier-length*

**have** *len-ab*:  $\text{length } a = 2 \ \text{length } b = 2$  **unfolding** *1* **by** *simp-all*

**then have** *len-mul-ab*:  $\text{length}(\text{grid-mul-nat } a \ b) \leq 4$

**using** *length-grid-mul-nat*[*of a b*] **by** *simp*

**from** *1* **have**  $\text{time}(\text{schoenhage-strassen-tm } m \ a \ b) =$

$\text{time}(m <_t 3) +$

$\text{time}(a *_{nt} b) +$

$\text{time}(\text{int-lsbfermat.from-nat-lsbfermat-tm } m \ (\text{grid-mul-nat } a \ b)) + 1$

**unfolding** *schoenhage-strassen-tm.simps*[*of m a b*] *time-bind-tm val-less-nat-tm*

**by** (*simp del: One-nat-def*)

**also have**  $\dots \leq (2 * m + 2) +$

$(8 * \text{length } a * \max(\text{length } a) (\text{length } b) + 1) +$

$\text{int-lsbfermat.time-from-nat-lsbfermat-tm-bound } m \ (\text{length}(\text{grid-mul-nat } a \ b)) + 1$

**apply** (*intro add-mono order.refl*)

**subgoal by** (*simp add: time-less-nat-tm 1*)

**subgoal by** (*rule time-grid-mul-nat-tm-le*)

**subgoal by** (*intro int-lsbfermat.time-from-nat-lsbfermat-tm-le-bound order.refl*)

**done**

**also have**  $\dots \leq 2 + 33 + 240 + 1$

**apply** (*intro add-mono order.refl*)

**subgoal unfolding** *1* **by** *simp*

**subgoal unfolding** *len-ab* **by** *simp*

**subgoal unfolding** *int-lsbfermat.time-from-nat-lsbfermat-tm-bound.simps*[*of 0 (length (grid-mul-nat a b))*]] *1*

**using** *len-mul-ab* **by** *simp*

**done**

**also have**  $\dots = 276$  **by** *simp*

**finally show** *?thesis* **unfolding** *schoenhage-strassen-Fm-bound.simps*[*of m*]

**using** *1* **by** *simp*

**next**

**case** *2*

**then have**  $(2::\text{nat})^{\wedge}(m + 1) \geq 4$

**using** *power-increasing*[*of 2 m + 1 2::nat*] **by** *simp*

**from** *2* **have**  $(2::\text{nat})^{\wedge}(m + 1) \leq 8$

**using** *power-increasing*[*of m + 1 3 2::nat*] **by** *simp*

```

from less.prems have len-ab: length a =  $2^{m+1}$  length b =  $2^{m+1}$ 
  using int-lsbfermat.fermat-carrier-length by simp-all
then have len-ab-le: length a ≤ 8 length b ≤ 8
  using  $\langle 2^{m+1} \leq 8 \rangle$  by linarith+
have len-mul-ab-le: length (grid-mul-nat a b) ≤  $2 * 2^{m+1}$ 
  using length-grid-mul-nat[of a b] len-ab by simp
from 2 have time (schoenhage-strassen-tm m a b) =
  time (m <_t 3) +
  time (a *_nt b) +
  time (int-lsbfermat.from-nat-lsbfermat m (grid-mul-nat a b)) + 1
unfolding schoenhage-strassen-tm.simps[of m a b] time-bind-tm val-less-nat-tm
  by (simp del: One-nat-def)
also have ... ≤  $(2 * m + 2) +$ 
   $(8 * \text{length } a * \max(\text{length } a) (\text{length } b) + 1) +$ 
   $(720 + 512 * 2^{m+1}) + 1$ 
  apply (intro add-mono order.refl)
  subgoal by (simp add: time-less-nat-tm 2)
  subgoal by (rule time-grid-mul-nat-tm-le)
  subgoal using int-lsbfermat.time-from-nat-lsbfermat-le[OF  $\langle 4 \leq 2^{m+1}$ 
1)  $\rangle$  len-mul-ab-le]
  by simp
  done
also have ... ≤  $6 + 513 + (720 + 512 * 8) + 1$ 
  apply (intro add-mono mult-le-mono order.refl)
  subgoal using 2 by simp
  subgoal
    apply (estimation estimate: max.boundedI[OF len-ab-le])
    using len-ab-le by simp
  subgoal using  $\langle 2^{m+1} \leq 8 \rangle$  .
  done
also have ... = 5336 by simp
  finally show ?thesis unfolding schoenhage-strassen-Fm-bound.simps[of m]
using 2 by simp
next
  case 3

```

```

interpret schoenhage-strassen-context m a b
  apply unfold-locales using 3 less.prems by simp-all

```

```

define time-η where time-η = time (map4-tm
  ( $\lambda x y z w.$  do {
    xmod ← take-tm (n + 2) x;
    ymod ← take-tm (n + 2) y;
    zmod ← take-tm (n + 2) z;
    wmod ← take-tm (n + 2) w;
    xy ← Znr.subtract-mod-tm xmod ymod;
    zw ← Znr.subtract-mod-tm zmod wmod;
    Znr.add-mod-tm xy zw
  })

```

```

  γ0 γ1 γ2 γ3) (is time-η = time (map4-tm ?η-fun - - -))
define time-ξ' where time-ξ' = time (map2-tm (λx y. do {
  v1 ← prim-root-exponent *t y;
  v2 ← oe-n +t v1;
  v3 ← v2 -t 1;
  summand1 ← Fnr.divide-by-power-of-2-tm x v3;
  summand2 ← Fnr.from-nat-lsbf-tm oe-n-plus-two-pow-n-one;
  Fnr.add-fermat-tm summand1 summand2
}))
c-diffs interval1)
define time-ξ where time-ξ = time (map-tm (int-lsbf-fermat.reduce-tm n) ξ')
define time-z where time-z = time (map2-tm (solve-special-residue-problem-tm
n) ξ η)
define time-z-filled where time-z-filled = time (map-tm (fill-tm segment-lens)
z)

note map-time-defs = time-η-def time-ξ'-def time-ξ-def time-z-def time-z-filled-def

from Fmr.res-carrier-eq have Fm-carrierI:  $\bigwedge i. 0 \leq i \implies i < 2^{2^m} + 1$ 
 $\implies i \in \text{carrier } Fm$ 
by simp

have length-uv-unpadded-le: length uv-unpadded  $\leq 12 * (3 * n + 5) * 2^{oe-n}$ 
+
(6 + 2 * karatsuba-lower-bound)
unfolding uv-unpadded-def
apply (estimation estimate: length-karatsuba-mul-nat-le)
unfolding A.length-num-Zn-pad B.length-num-Zn-pad pad-length-def by simp

have prim-root-exponent-le: prim-root-exponent  $\leq 2$  unfolding prim-root-exponent-def
by simp
then have prim-root-exponent-2-le: prim-root-exponent * 2  $\leq 4$ 
by simp

have length-interval1: length interval1 =  $2^{(oe-n - 1)}$ 
unfolding interval1-def by simp
have length-interval2: length interval2 =  $2^{(oe-n - 1)}$ 
unfolding interval2-def using two-pow-oe-n-as-halves by simp
have length-oe-n-plus-two-pow-n-zeros: length oe-n-plus-two-pow-n-zeros = oe-n
+  $2^n$ 
unfolding oe-n-plus-two-pow-n-zeros-def by simp
have length-oe-n-plus-two-pow-n-one: length oe-n-plus-two-pow-n-one = oe-n +
 $2^{n+1}$ 
unfolding oe-n-plus-two-pow-n-one-def
using length-oe-n-plus-two-pow-n-zeros by simp
have c-dft-odds-carrier: set c-dft-odds  $\subseteq$  Fnr.fermat-non-unique-carrier
unfolding c-dft-odds-def
apply (intro set-subseteqI)
subgoal premises prems for i

```

```

proof –
  have map2 (schoenhage-strassen n) A.num-dft-odds B.num-dft-odds ! i =
    schoenhage-strassen n (A.num-dft-odds ! i) (B.num-dft-odds ! i)
  using nth-map prems by simp
  also have ... ∈ Fnr.fermat-non-unique-carrier
  apply (intro conjunct2[OF schoenhage-strassen-correct])
  subgoal
    apply (intro set-subseteqD[OF A.num-dft-odds-carrier])
    using prems by simp
  subgoal
    apply (intro set-subseteqD[OF B.num-dft-odds-carrier])
    using prems by simp
  done
  finally show ?thesis .
qed
done
have c-diffs-carrier: c-diffs ! i ∈ Fnr.fermat-non-unique-carrier if  $i < 2^{oe-n}$ 
– 1) for i
  unfolding c-diffs-def Fnr.ifft.simps
  apply (intro set-subseteqD[OF Fnr.fft-iff-carrier[of - oe-n - 1]])
  subgoal using length-c-dft-odds .
  subgoal using c-dft-odds-carrier .
  subgoal using Fnr.length-iff[OF length-c-dft-odds] that by simp
  done
have  $\xi'$ -carrier:  $\xi' ! i \in Fnr.fermat-non-unique-carrier$  if  $i < 2^{oe-n - 1}$ 
for i
proof –
  from that have  $\xi' ! i = Fnr.add-fermat$ 
    (Fnr.divide-by-power-of-2 (c-diffs ! i)
      (oe-n + prim-root-exponent * ([0..<2^{oe-n - 1}] ! i) - 1))
    (Fnr.from-nat-lsb (replicate (oe-n + 2^n) False @ [True]))
  unfolding  $\xi'$ -def using nth-map2 that length-c-diffs by simp
  also have ... ∈ Fnr.fermat-non-unique-carrier
  apply (intro Fnr.add-fermat-closed)
  subgoal
    by (intro Fnr.divide-by-power-of-2-closed that c-diffs-carrier)
  subgoal by (intro Fnr.from-nat-lsb-correct(1))
  done
  finally show  $\xi' ! i \in Fnr.fermat-non-unique-carrier$  .
qed
have  $\xi'$ -carrier': set  $\xi' \subseteq Fnr.fermat-non-unique-carrier$ 
  apply (intro set-subseteqI  $\xi'$ -carrier) unfolding length- $\xi'$  .
have length- $\xi'$ -entries: length  $x \leq 2^n + 2$  if  $x \in$  set  $\xi$  for x
proof –
  from that obtain  $x'$  where  $x' \in$  set  $\xi' x = Fnr.reduce x'$  unfolding  $\xi'$ -def
    by auto
  from that show ?thesis unfolding  $\langle x = Fnr.reduce x' \rangle$ 
  apply (intro Fnr.reduce-correct'(2))
  using  $\langle x' \in$  set  $\xi' \rangle$   $\xi'$ -carrier' by auto

```

```

qed
have length- $\eta$ -entries: length ( $\eta ! i$ ) =  $n + 2$  if  $i < 2^{(oe-n-1)}$  for  $i$ 
proof -
  have  $\eta ! i = \text{Znr.add-mod } (\text{Znr.subtract-mod } (\text{take } (n + 2) (\gamma 0 ! i)) (\text{take } (n + 2) (\gamma 1 ! i)))$ 
    ( $\text{Znr.subtract-mod } (\text{take } (n + 2) (\gamma 2 ! i)) (\text{take } (n + 2) (\gamma 3 ! i)))$ 
  unfolding  $\eta$ -def Let-def defs'[symmetric]
  apply (intro nth-map4)
  unfolding length- $\gamma$ s defs' using length- $\gamma$ -i that by simp-all
  then show ?thesis using Znr.add-mod-closed by simp
qed
have length-z-entries: length ( $z ! i$ )  $\leq 2^n + n + 4$  if  $i < 2^{(oe-n-1)}$  for
 $i$ 
proof -
  have  $z ! i = \text{solve-special-residue-problem } n (\xi ! i) (\eta ! i)$ 
  unfolding z-def apply (intro nth-map2) using that length- $\xi$  length- $\eta$  by
simp-all
  also have length ...  $\leq \max (\text{length } (\xi ! i))$ 
    ( $2^n + \text{length } (\text{Znr.subtract-mod } (\eta ! i) (\text{take } (n + 2) (\xi ! i))) + 1) + 1$ 
  unfolding solve-special-residue-problem-def Let-def defs[symmetric]
  apply (estimation estimate: length-add-nat-upper)
  apply (estimation estimate: length-add-nat-upper)
  by (simp del: One-nat-def)
  also have ...  $\leq \max (2^n + 2) ((2^n + (n + 2)) + 1) + 1$ 
  apply (intro add-mono order.refl max.mono)
  subgoal using length- $\xi$ -entries nth-mem[of  $i \xi$ ] length- $\xi$  that by simp
  subgoal apply (intro Znr.length-subtract-mod)
  subgoal using length- $\eta$ -entries[OF that] by simp
  subgoal by simp
  done
  done
  also have ... =  $2^n + n + 4$  by simp
  finally show ?thesis .
qed
have length-z-filled-entries: length ( $z\text{-filled} ! i$ )  $\leq 2^n + n + 4$  if  $i < 2^{(oe-n-1)}$  for  $i$ 
proof -
  have  $z\text{-filled} ! i = \text{fill } (2^{(n-1)}) (z ! i)$ 
  unfolding z-filled-def segment-lens-def
  using nth-map[of  $i z$ ] unfolding length-z
  using that by auto
  also have length ...  $\leq \max (2^{(n-1)}) (2^n + n + 4)$ 
  using length-z-entries[OF that] unfolding length-fill' by simp
  also have ...  $\leq 2^n + n + 4$ 
  apply (intro max.boundedI order.refl)
  using power-increasing[of  $n - 1 n 2::nat$ ] by linarith
  finally show ?thesis .
qed

```

```

have length-z-complete-entries: length i ≤ 2 ^ n + n + 4 if i ∈ set z-complete
for i
proof –
  from that consider i ∈ set z-filled | i ∈ set z-consts
  unfolding z-complete-def by auto
  then show ?thesis
  proof cases
    case 1
    show ?thesis
    using iffD1[OF in-set-conv-nth 1] length-z-filled-entries length-z-filled
    by auto
  next
  case 2
  then have i-eq: i = oe-n-plus-two-pow-n-one
  unfolding z-consts-def defs'
  by simp
  show ?thesis unfolding i-eq length-oe-n-plus-two-pow-n-one
  using oe-n-le-n by simp
qed
qed
have length-z-complete: length z-complete = 2 ^ oe-n
  unfolding z-complete-def
  by (simp add: length-z-filled length-z-consts two-pow-oe-n-as-halves)
have length-z-sum-le: length z-sum ≤ 28 * Fmr.e
proof –
  have length z-sum ≤ ((2 ^ n + n + 4) + 1) * length z-complete
  unfolding z-sum-def z-complete-def
  apply (intro length-combine-z-le segment-lens-pos)
  using length-z-complete-entries z-complete-def by simp-all
  also have ... = (2 ^ n + n + 5) * 2 ^ oe-n
  unfolding length-z-complete by simp
  also have ... ≤ (2 ^ n + 2 ^ n + 5 * 2 ^ n) * (2 * 2 ^ n)
  apply (intro mult-le-mono add-mono order.refl)
  subgoal using less-exp by simp
  subgoal by simp
  subgoal by (estimation estimate: oe-n-le-n; simp)
  done
  also have ... = 14 * 2 ^ (2 * n)
  by (simp add: mult-2[of n] power-add)
  also have ... ≤ 28 * Fmr.e
  using two-pow-two-n-le by simp
  finally show ?thesis .
qed

have val-ih: map2 (λx y. val (schoenhage-strassen-tm n x y)) A.num-dft-odds
B.num-dft-odds =
  c-dft-odds
  unfolding c-dft-odds-def
  apply (intro map-cong ext refl)

```

```

subgoal premises prems for p
proof –
  from prems obtain i where p-decomp: i < length A.num-dft-odds i < length
B.num-dft-odds
    p = (A.num-dft-odds ! i, B.num-dft-odds ! i)
    using set-zip[of A.num-dft-odds B.num-dft-odds] by auto
    show ?thesis unfolding p-decomp prod.case
    apply (intro val-schoenhage-strassen-tm)
    subgoal using set-subseteqD[OF A.num-dft-odds-carrier]
      using p-decomp by simp
    subgoal using set-subseteqD[OF B.num-dft-odds-carrier]
      using p-decomp by simp
    done
qed
done

have  $\xi'$ -alt: map2
  ( $\lambda$  x y. Fnr.add-fermat
    (Fmr.divide-by-power-of-2 x (oe-n + prim-root-exponent * y –
1))
    (Fnr.from-nat-lsb oe-n-plus-two-pow-n-one))
  c-diffs interval1 =  $\xi'$ 
unfolding  $\xi'$ -def Let-def defs'[symmetric] by (rule refl)

have time- $\eta$   $\leq$  ((112 * (n + 2) + 254) + 1) * min (min (min (length  $\gamma$ 0)
(length  $\gamma$ 1)) (length  $\gamma$ 2)) (length  $\gamma$ 3) + 1
unfolding time- $\eta$ -def
apply (intro time-map4-tm-bounded)
unfolding tm-time-simps add.assoc[symmetric] val-take-tm Znr.val-subtract-mod-tm
Znr.val-add-mod-tm
subgoal premises prems for x y z w
proof –
  have time (take-tm (n + 2) x) + time (take-tm (n + 2) y) + time (take-tm
(n + 2) z) + time (take-tm (n + 2) w) +
    time (Znr.subtract-mod-tm (take (n + 2) x) (take (n + 2) y)) +
    time (Znr.subtract-mod-tm (take (n + 2) z) (take (n + 2) w)) +
    time (Znr.add-mod-tm (Znr.subtract-mod (take (n + 2) x) (take (n + 2)
y)) (Znr.subtract-mod (take (n + 2) z) (take (n + 2) w)))  $\leq$ 
    ((n + 2) + 1) + ((n + 2) + 1) + ((n + 2) + 1) + ((n + 2) + 1) +
    (118 + 51 * (n + 2)) +
    (118 + 51 * (n + 2)) +
    (14 + 4 * (n + 2) + 2 * (n + 2))
apply (intro add-mono time-take-tm-le)
subgoal
apply (estimation estimate: Znr.time-subtract-mod-tm-le)
unfolding length-take
apply (estimation estimate: min.cobounded2)
apply (estimation estimate: min.cobounded2)
by (simp add: defs')

```

```

subgoal
  apply (estimation estimate: Znr.time-subtract-mod-tm-le)
  unfolding length-take
  apply (estimation estimate: min.cobounded2)
  apply (estimation estimate: min.cobounded2)
  by (simp add: defs')
subgoal
  apply (estimation estimate: Znr.time-add-mod-tm-le)
  apply (estimation estimate: Znr.length-subtract-mod[OF length-take-cobounded1
length-take-cobounded1])
  apply (estimation estimate: Znr.length-subtract-mod[OF length-take-cobounded1
length-take-cobounded1])
  apply simp
done
done
also have ... = 112 * (n + 2) + 254 by simp
finally show ?thesis .
qed
done
also have ... = (255 + 112 * (n + 2)) * 2 ^ (oe-n - 1) + 1
  unfolding length-γs defs' using length-γ-i by simp
also have ... ≤ (255 + 112 * (n + 2)) * 2 ^ n + 1 * 2 ^ n
  apply (intro add-mono mult-le-mono order.refl)
  unfolding oe-n-def by simp-all
also have ... = (256 + 112 * (n + 2)) * 2 ^ n
  by (simp add: add-mult-distrib)
also have ... ≤ (128 * (n + 2) + 112 * (n + 2)) * 2 ^ n
  apply (intro add-mono mult-le-mono order.refl)
  by simp
finally have time-η-le: time-η ≤ 240 * (n + 2) * 2 ^ n by simp

have oe-n-prim-root-le: oe-n + prim-root-exponent * y - 1 ≤ fn-carrier-len if
y ∈ set interval1 for y
proof -
  have oe-n + prim-root-exponent * y - 1 ≤ n + prim-root-exponent * y
    using oe-n-minus-1-le-n by simp
  also have ... ≤ n + prim-root-exponent * 2 ^ (oe-n - 1)
    using that unfolding interval1-def defs' by simp
  also have ... = n + 2 ^ n
    unfolding oe-n-def prim-root-exponent-def
    by (cases odd m; simp add: n-gt-0 power-Suc[symmetric])
  also have ... ≤ 2 ^ n + 2 ^ n
    by simp
  also have ... = fn-carrier-len
    unfolding defs' by simp
  finally show ?thesis .
qed

have time-ξ' ≤ ((475 + 378 * Fnr.e) + 2) * length c-diffs + 3

```

```

unfolding time-ξl-def
apply (intro time-map2-tm-bounded)
subgoal unfolding length-c-diffs length-interval1 by (rule refl)
subgoal premises prems for x y
unfolding tm-time-simps add.assoc[symmetric] val-times-nat-tm defs[symmetric]
  val-plus-nat-tm val-minus-nat-tm Fmr.val-divide-by-power-of-2-tm
  Fnr.val-from-nat-lsbf-tm
proof –
  have time (prim-root-exponent *t y) +
    time (oe-n +t (prim-root-exponent * y)) +
    time ((oe-n + prim-root-exponent * y) –t 1) +
    time (Fmr.divide-by-power-of-2-tm x (oe-n + prim-root-exponent * y –
1)) +
    time (Fnr.from-nat-lsbf-tm oe-n-plus-two-pow-n-one) +
    time
    (Fnr.add-fermat-tm
    (Fmr.divide-by-power-of-2 x (oe-n + prim-root-exponent * y – 1))
    (Fnr.from-nat-lsbf oe-n-plus-two-pow-n-one)) ≤
    (2 * y + 5) + (oe-n + 1) + 2 + (24 + 26 * fn-carrier-len + 26 * length
x) +
    (288 * 1 + 144 + (96 + 192 * 1 + 8 * 1 * 1) * Fnr.e) +
    (13 + 7 * length x + 21 * Fnr.e) (is ?t ≤ -)
  apply (intro add-mono)
  subgoal unfolding time-times-nat-tm
    apply (estimation estimate: prim-root-exponent-le)
    by simp
  subgoal unfolding time-plus-nat-tm by simp
  subgoal unfolding time-minus-nat-tm by simp
  subgoal apply (estimation estimate: Fmr.time-divide-by-power-of-2-tm-le)
    apply (estimation estimate: oe-n-prim-root-le[OF prems(2)])
    apply (estimation estimate: Nat-max-le-sum)
    by simp
  subgoal
    apply (intro Fnr.time-from-nat-lsbf-tm-le Fnr.e-ge-4 n-gt-0)
  unfolding length-oe-n-plus-two-pow-n-one using oe-n-n-bound-1 by simp
  subgoal
    apply (estimation estimate: Fnr.time-add-fermat-tm-le)
    unfolding Fmr.length-multiply-with-power-of-2 Fnr.length-from-nat-lsbf
    apply (estimation estimate: Nat-max-le-sum)
    by simp
  done
  also have ... = 477 + 2 * y + oe-n + 343 * Fnr.e + 33 * length x
    unfolding fn-carrier-len-def by simp
  also have ... = 477 + 2 * y + oe-n + 376 * Fnr.e
    using prems set-subseteqI[OF c-diffs-carrier] length-c-diffs by auto
  also have ... ≤ 477 + 2 * 2(oe-n - 1) + oe-n + 376 * Fnr.e
    using prems unfolding interval1-def
    by simp
  also have ... ≤ 477 + oe-n + 377 * Fnr.e

```

```

    unfolding oe-n-def by simp
    also have ... ≤ 475 + 378 * Fnr.e
    using oe-n-n-bound-1 by simp
    finally show ?t ≤ 475 + 378 * Fnr.e unfolding defs[symmetric] .
  qed
done
also have ... ≤ (475 + 758 * 2 ^ n) * 2 ^ n + 3
  apply (intro add-mono[of - - 3] order.refl mult-le-mono)
  subgoal by simp
  subgoal unfolding length-c-diffs oe-n-def by simp
  done
also have ... = 3 + 475 * 2 ^ n + 758 * 2 ^ (2 * n)
  by (simp add: add-mult-distrib power-add mult-2)
finally have time-ξ'-le: time-ξ' ≤ ... .

have time-reduce-ξ'-nth: time (Fnr.reduce-tm i) ≤ 155 + 216 * 2 ^ n if i ∈
set ξ' for i
proof -
  have length i = Fnr.e
  using iffD1[OF in-set-conv-nth that]
  Fnr.fermat-carrier-length[OF ξ'-carrier] length-ξ' by auto
  show ?thesis
  by (estimation estimate: Fnr.time-reduce-tm-le)
  (simp add: ⟨length i = Fnr.e⟩)
qed

have time-ξ ≤ ((155 + 216 * 2 ^ n) + 1) * length ξ' + 1
  unfolding time-ξ-def
  by (intro time-map-tm-bounded time-reduce-ξ'-nth)
also have ... ≤ (156 + 216 * 2 ^ n) * 2 ^ n + 1
  unfolding length-ξ' oe-n-def by simp
also have ... = 1 + 156 * 2 ^ n + 216 * 2 ^ (2 * n)
  by (simp add: add-mult-distrib power-add mult-2)
finally have time-ξ-le: time-ξ ≤ ... .

have time-z ≤ ((245 + 74 * 2 ^ n + 55 * (n + 2) + 2 * (2 ^ n + 2)) + 2)
* length ξ + 3
  unfolding time-z-def
  apply (intro time-map2-tm-bounded)
  subgoal unfolding length-ξ length-η by (rule refl)
  subgoal premises prems for x y
  apply (estimation estimate: time-solve-special-residue-problem-tm-le)
  apply (intro add-mono mult-le-mono order.refl)
  subgoal using length-η-entries length-η iffD1[OF in-set-conv-nth ⟨y ∈ set
η⟩] by auto
  subgoal using length-ξ-entries[OF ⟨x ∈ set ξ⟩] .
  done
done
also have ... = (361 + 76 * 2 ^ n + 55 * n) * 2 ^ (oe-n - 1) + 3

```

```

    unfolding length-ξ by simp
  also have ... ≤ (361 + 76 * 2 ^ n + 55 * 2 ^ n) * 2 ^ n + 3
  apply (intro add-mono order.refl mult-le-mono)
  subgoal using less-exp by simp
  subgoal unfolding oe-n-def by simp
  done
  also have ... = 131 * 2 ^ (2 * n) + 361 * 2 ^ n + 3
  by (simp add: add-mult-distrib mult-2 power-add)
  finally have time-z-le: time-z ≤ ... .

  have time-z-filled ≤ ((2 * (2 ^ n + n + 4) + 2 ^ (n - 1) + 5) + 1) * length
z + 1
  unfolding time-z-filled-def
  apply (intro time-map-tm-bounded)
  unfolding time-fill-tm segment-lens-def
  using length-z-entries in-set-conv-nth[of - z] unfolding length-z
  by fastforce
  also have ... ≤ (2 * 2 ^ n + 2 * n + 2 ^ (n - 1) + 14) * 2 ^ n + 1
  apply (intro add-mono[of - - 1] mult-le-mono order.refl)
  subgoal by simp
  subgoal unfolding length-z oe-n-def by simp
  done
  also have ... ≤ (5 * 2 ^ n + 14) * 2 ^ n + 1
  apply (intro add-mono[of - - 1] mult-le-mono order.refl)
  using less-exp[of n] power-increasing[of n - 1 n 2::nat] by linarith
  also have ... = 5 * 2 ^ (2 * n) + 14 * 2 ^ n + 1
  by (simp add: add-mult-distrib mult-2 power-add)
  finally have time-z-filled-le: time-z-filled ≤ ... .

  have time (map2-tm (schoenhage-strassen-tm n) A.num-dft-odds B.num-dft-odds)
≤
  (schoenhage-strassen-Fm-bound n + 2) * length A.num-dft-odds + 3
  apply (intro time-map2-tm-bounded)
  subgoal unfolding A.length-num-dft-odds B.length-num-dft-odds by (rule
refl)
  subgoal premises prems for x y
  apply (intro less.IH[OF n-lt-m])
  subgoal using prems A.num-dft-odds-carrier by blast
  subgoal using prems B.num-dft-odds-carrier by blast
  done
  done
  also have ... ≤ schoenhage-strassen-Fm-bound n * 2 ^ (oe-n - 1) + 2 * 2 ^
n + 3
  unfolding A.length-num-dft-odds
  using oe-n-minus-1-le-n
  by simp
  finally have recursive-time: time (map2-tm (schoenhage-strassen-tm n) A.num-dft-odds
B.num-dft-odds) ≤
  ... .

```

**have** *two-pow-pos*:  $(2::\text{nat}) \hat{\ } x > 0$  **for**  $x$   
**by** *simp*

**have** *time (schoenhage-strassen-tm m a b)* =  
*time (m <\_t 3) + time (odd-tm m) +*  
*(if odd m then time (m +\_t 1) + time ((m + 1) div\_t 2)*  
*else time (m +\_t 2) + time ((m + 2) div\_t 2)) +*  
*time (n +\_t 1) +*  
*time (n -\_t 1) +*  
*time (n +\_t 2) +*  
*(if odd m then 0 else 0) +*  
*time (2 ^\_t (n - 1)) +*  
*time (subdivide-tm segment-lens a) +*  
*time (subdivide-tm segment-lens b) +*  
*time (map-tm Znr.reduce-tm A.num-blocks) +*  
*time (3 \*\_t n) +*  
*time ((3 \* n) +\_t 5) +*  
*time (map-tm (fill-tm pad-length) A.num-Zn) +*  
*time (concat-tm (map (fill pad-length) A.num-Zn)) +*  
*time (map-tm Znr.reduce-tm B.num-blocks) +*  
*time (map-tm (fill-tm pad-length) B.num-Zn) +*  
*time (concat-tm (map (fill pad-length) B.num-Zn)) +*  
*time (oe-n +\_t 1) +*  
*time (2 ^\_t (oe-n + 1)) +*  
*time (pad-length \*\_t 2 ^ (oe-n + 1)) +*  
*time (karatsuba-mul-nat-tm A.num-Zn-pad B.num-Zn-pad) +*  
*time (ensure-length-tm uv-length uv-unpadded) +*  
*time (oe-n -\_t 1) +*  
*time (2 ^\_t (oe-n - 1)) +*  
*time (subdivide-tm pad-length uv) +*  
*time (subdivide-tm (2 ^ (oe-n - 1))  $\gamma$ s) +*  
*time (nth-tm  $\gamma$  0) +*  
*time (nth-tm  $\gamma$  1) +*  
*time (nth-tm  $\gamma$  2) +*  
*time (nth-tm  $\gamma$  3) +*  
*time- $\eta$  +*  
*(if odd m then 0 else 0) +*  
*time (2 ^\_t (n + 1)) +*  
*time (map-tm (fill-tm fn-carrier-len) A.num-blocks) +*  
*time (map-tm (fill-tm fn-carrier-len) B.num-blocks) +*  
*time (Fnr.fft-tm prim-root-exponent A.num-blocks-carrier) +*  
*time (Fnr.fft-tm prim-root-exponent B.num-blocks-carrier) +*  
*time (evens-odds-tm False A.num-dft) +*  
*time (evens-odds-tm False B.num-dft) +*  
*time (map2-tm (schoenhage-strassen-tm n) A.num-dft-odds B.num-dft-odds) +*  
*time (prim-root-exponent \*\_t 2) +*  
*time (Fnr.ifft-tm (prim-root-exponent \* 2) c-dft-odds) +*  
*time (2 ^\_t oe-n) +*

```

time (upt-tm 0 (2 ^ (oe-n - 1))) +
time (upt-tm (2 ^ (oe-n - 1)) (2 ^ oe-n)) +
time (2 ^ t n) +
time (oe-n + t 2 ^ n) +
time (replicate-tm (oe-n + 2 ^ n) False) +
time (oe-n-plus-two-pow-n-zeros @t [True]) +
time-ξ' +
time-ξ +
time-z +
time (map-tm (fill-tm segment-lens) z) +
time (replicate-tm (2 ^ (oe-n - 1)) oe-n-plus-two-pow-n-one) +
time (z-filled @t z-consts) +
time (combine-z-tm segment-lens z-complete) +
time (Fmr.from-nat-lsbf-tm z-sum) +
0 +
1
  unfolding schoenhage-strassen-tm.simps[of m a b] tm-time-simps
  unfolding val-simp val-times-nat-tm val-subdivide-tm[OF two-pow-pos] val-subdivide-tm[OF
pad-length-gt-0] Znr.val-reduce-tm defs[symmetric]
  Let-def val-nth-γ val-fft1 val-fft2 val-iff1 val-ih Fnr.val-iff1-tm[OF length-c-dft-odds]
  unfolding Eq-FalseI[OF 3] if-False add.assoc[symmetric] time-z-filled-def[symmetric]
  apply (intro arg-cong2[where f = (+)] refl)
  unfolding defs'[symmetric] time-ξ'-def[symmetric] time-η-def[symmetric]
time-ξ-def[symmetric]
  time-z-def[symmetric] time-z-filled-def[symmetric]
  by (intro refl)+
  also have ... ≤ 8 + (8 * m + 14) +
(28 + 9 * m) +
(n + 1) +
2 +
(n + 1) +
0 +
(8 * 2 ^ n + 1) +
(10 * 2 ^ m + 2 ^ n + 4) +
(10 * 2 ^ m + 2 ^ n + 4) +
(((2 ^ n + 2 * n + 12) + 1) * length A.num-blocks + 1) +
(7 + 3 * n) +
(6 + 3 * n) +
(((5 * n + 14) + 1) * length A.num-Zn + 1) +
(14 * 2 ^ n + 6 * (n * 2 ^ n) + 1) +
(((2 ^ n + 2 * n + 12) + 1) * length B.num-blocks + 1) +
(((5 * n + 14) + 1) * length B.num-Zn + 1) +
(14 * 2 ^ n + 6 * (n * 2 ^ n) + 1) +
(n + 2) +
(24 * 2 ^ n + 5 * n + 11) +
(12 * (n * 2 ^ n) + 20 * 2 ^ n + 6 * n + 11) +
time (karatsuba-mul-nat-tm A.num-Zn-pad B.num-Zn-pad) +
(168 * (n * 2 ^ n) + 280 * 2 ^ n + (4 * karatsuba-lower-bound + 19)) +
2 +

```

$12 * 2^n +$   
 $(14 + 60 * (n * 2^n) + (100 * 2^n + 6 * n)) +$   
 $(22 * 2^n + 4) +$   
 $(0 + 1) +$   
 $(1 + 1) +$   
 $(2 + 1) +$   
 $(3 + 1) +$   
 $(480 * 2^n + 240 * (n * 2^n)) +$   
 $0 +$   
 $24 * 2^n +$   
 $((3 * 2^n + 5) + 1) * \text{length } A.\text{num-blocks} + 1) +$   
 $((3 * 2^n + 5) + 1) * \text{length } B.\text{num-blocks} + 1) +$   
 $(2^{oe-n} * (66 + 87 * \text{Fnr.e}) + oe-n * 2^{oe-n} * (76 + 116 * \text{Fnr.e}) +$   
 $8 * \text{prim-root-exponent} * 2^{(2 * oe-n)}) +$   
 $(2^{oe-n} * (66 + 87 * \text{Fnr.e}) + oe-n * 2^{oe-n} * (76 + 116 * \text{Fnr.e}) +$   
 $8 * \text{prim-root-exponent} * 2^{(2 * oe-n)}) +$   
 $(2 * 2^n + 1) +$   
 $(2 * 2^n + 1) +$   
 $(\text{schoenhage-strassen-Fm-bound } n * 2^{(oe-n - 1)} + 2 * 2^n + 3) +$   
 $9 +$   
 $(2^{(oe-n - 1)} * (66 + 87 * \text{Fnr.e}) +$   
 $(oe-n - 1) * 2^{(oe-n - 1)} * (76 + 116 * \text{Fnr.e}) +$   
 $8 * (\text{prim-root-exponent} * 2) * 2^{(2 * (oe-n - 1))}) +$   
 $24 * 2^n +$   
 $(2 * 2^{(2 * n)} + 5 * 2^n + 2) +$   
 $(8 * 2^{(2 * n)} + 10 * 2^n + 2) +$   
 $12 * 2^n +$   
 $(n + 2) +$   
 $(2^n + n + 2) +$   
 $(2^n + n + 2) +$   
 $(3 + 475 * 2^n + 758 * 2^{(2 * n)}) +$   
 $(1 + 156 * 2^n + 216 * 2^{(2 * n)}) +$   
 $(131 * 2^{(2 * n)} + 361 * 2^n + 3) +$   
 $(5 * 2^{(2 * n)} + 14 * 2^n + 1) +$   
 $(2^n + 1) +$   
 $(2^n + 1) +$   
 $(10 + (3 * \text{segment-lens} + 2 * (2^n + n + 4) + 10) * \text{length } z\text{-complete}) +$   
 $(8208 + 23488 * 2^m) + 0 + 1$   
**apply** (*intro add-mono*)  
**subgoal unfolding** *time-less-nat-tm* **by** *simp*  
**subgoal by** (*rule time-odd-tm-le*)  
**subgoal**  
**apply** (*estimation estimate: if-le-max*)  
**unfolding** *time-plus-nat-tm*  
**apply** (*estimation estimate: time-divide-nat-tm-le*)  
**apply** (*estimation estimate: time-divide-nat-tm-le*)  
**by** *simp*  
**subgoal unfolding** *time-plus-nat-tm* **by** (*rule order.refl*)  
**subgoal unfolding** *time-minus-nat-tm* **by** *simp*

```

subgoal unfolding time-plus-nat-tm by (rule order.refl)
subgoal by simp
subgoal
  apply (estimation estimate: time-power-nat-tm-le)
  unfolding Suc-diff-1[OF n-gt-0]
  using less-exp[of n - 1] power-increasing[of n - 1 n 2::nat]
  by linarith
subgoal
  apply (estimation estimate: time-subdivide-tm-le[OF segment-lens-pos])
  unfolding A.length-num segment-lens-def power-Suc[symmetric]
  Suc-diff-1[OF n-gt-0] by simp
subgoal
  apply (estimation estimate: time-subdivide-tm-le[OF segment-lens-pos])
  unfolding B.length-num segment-lens-def power-Suc[symmetric]
  Suc-diff-1[OF n-gt-0] by simp
subgoal
  apply (intro time-map-tm-bounded)
  subgoal premises prems for i
  proof -
    have time (Znr.reduce-tm i) = 8 + 2 * length i + 2 * n + 4
      unfolding Znr.time-reduce-tm by simp
    also have ... = 8 + 2 * 2 ^ (n - 1) + 2 * n + 4
      apply (intro arg-cong2[where f = (+)] arg-cong2[where f = (*)] refl)
      using A.length-nth-num-blocks iffD1[OF in-set-conv-nth prems]
      unfolding A.length-num-blocks by auto
    also have ... = 2 ^ n + 2 * n + 12
      unfolding power-Suc[symmetric] Suc-diff-1[OF n-gt-0] by simp
    finally show ?thesis by simp
  qed
done
subgoal by (simp del: One-nat-def)
subgoal by simp
subgoal apply (intro time-map-tm-bounded)
  subgoal premises prems for i
  proof -
    have time (fill-tm pad-length i) = 2 * length i + 3 * n + 10
      unfolding time-fill-tm pad-length-def by simp
    also have ... = 2 * (n + 2) + 3 * n + 10
      apply (intro arg-cong2[where f = (+)] arg-cong2[where f = (*)] refl)
      using A.length-nth-num-Zn iffD1[OF in-set-conv-nth prems]
      unfolding A.length-num-Zn by auto
    also have ... = 5 * n + 14
      by simp
    finally show ?thesis by simp
  qed
done
subgoal unfolding time-concat-tm length-map A.num-Zn-pad-def[symmetric]
  A.length-num-Zn-pad
  unfolding A.length-num-Zn pad-length-def

```

```

    apply (estimation estimate: oe-n-le-n)
    by (simp add: add-mult-distrib)
  subgoal
    apply (intro time-map-tm-bounded)
    subgoal premises prems for i
    proof -
      have time (Znr.reduce-tm i) = 8 + 2 * length i + 2 * n + 4
      unfolding Znr.time-reduce-tm by simp
      also have ... = 8 + 2 * 2 ^ (n - 1) + 2 * n + 4
      apply (intro arg-cong2[where f = (+)] arg-cong2[where f = (*)] refl)
      using B.length-nth-num-blocks iffD1[OF in-set-conv-nth prems]
      unfolding B.length-num-blocks by auto
      also have ... = 2 ^ n + 2 * n + 12
      unfolding power-Suc[symmetric] Suc-diff-1[OF n-gt-0] by simp
      finally show ?thesis by simp
    qed
  done
  subgoal apply (intro time-map-tm-bounded)
  subgoal premises prems for i
  proof -
    have time (fill-tm pad-length i) = 2 * length i + 3 * n + 10
    unfolding time-fill-tm pad-length-def by simp
    also have ... = 2 * (n + 2) + 3 * n + 10
    apply (intro arg-cong2[where f = (+)] arg-cong2[where f = (*)] refl)
    using B.length-nth-num-Zn iffD1[OF in-set-conv-nth prems]
    unfolding B.length-num-Zn by auto
    also have ... = 5 * n + 14
    by simp
    finally show ?thesis by simp
  qed
  done
  subgoal unfolding time-concat-tm length-map B.num-Zn-pad-def[symmetric]
  B.length-num-Zn-pad
  unfolding B.length-num-Zn pad-length-def
  apply (estimation estimate: oe-n-le-n)
  by (simp add: add-mult-distrib)
  subgoal unfolding oe-n-def by simp
  subgoal
    apply (estimation estimate: time-power-nat-tm-le)
    apply (estimation estimate: oe-n-le-n)
    by simp-all
  subgoal
    unfolding time-times-nat-tm pad-length-def
    unfolding add-mult-distrib add-mult-distrib2
    apply (estimation estimate: oe-n-le-n)
    by simp-all
  subgoal by (rule order.refl)
  subgoal unfolding time-ensure-length-tm
  apply (estimation estimate: length-uv-unpadded-le)

```

```

    unfolding uv-length-def pad-length-def
    apply (estimation estimate: oe-n-le-n)
    by (simp-all add: add-mult-distrib)
subgoal by simp
subgoal
  apply (estimation estimate: time-power-nat-tm-2-le)
  apply (estimation estimate: oe-n-minus-1-le-n)
  by simp-all
subgoal
  apply (estimation estimate: time-subdivide-tm-le[OF pad-length-gt-0])
  unfolding uv-length-def length-uv pad-length-def
  apply (estimation estimate: oe-n-le-n)
  by (simp-all add: add-mult-distrib)
subgoal
  apply (estimation estimate: time-subdivide-tm-le[OF two-pow-pos])
  unfolding length- $\gamma$ s'
  apply (estimation estimate: oe-n-minus-1-le-n)
  by simp-all
subgoal using time-nth-tm[of 0  $\gamma$ ] sc $\gamma$ (1) by simp
subgoal using time-nth-tm[of 1  $\gamma$ ] sc $\gamma$ (1) by simp
subgoal using time-nth-tm[of 2  $\gamma$ ] sc $\gamma$ (1) by simp
subgoal using time-nth-tm[of 3  $\gamma$ ] sc $\gamma$ (1) by simp
subgoal
  apply (estimation estimate: time- $\eta$ -le)
  by (simp add: add-mult-distrib)
subgoal by simp
subgoal
  apply (estimation estimate: time-power-nat-tm-2-le)
  by simp
subgoal apply (intro time-map-tm-bounded)
  unfolding time-fill-tm
  subgoal premises prems for i
  proof -
    have leni: length i =  $2^{(n-1)}$ 
      using iffD1[OF in-set-conv-nth prems]
      unfolding A.length-num-blocks
      using A.length-nth-num-blocks by auto
    show ?thesis unfolding leni power-Suc[symmetric] Suc-diff-1[OF n-gt-0]
      unfolding fn-carrier-len-def
      by simp
  qed
done
subgoal apply (intro time-map-tm-bounded)
  unfolding time-fill-tm
  subgoal premises prems for i
  proof -
    have leni: length i =  $2^{(n-1)}$ 
      using iffD1[OF in-set-conv-nth prems]
      unfolding B.length-num-blocks

```

```

    using B.length-nth-num-blocks by auto
    show ?thesis unfolding leni power-Suc[symmetric] Suc-diff-1[OF n-gt-0]
    unfolding fn-carrier-len-def
    by simp
qed
done
subgoal apply (intro Fnr.time-fft-tm-le A.length-num-blocks-carrier)
  using A.fill-num-blocks-carrier
  using Fnr.fermat-carrier-length
  unfolding defs[symmetric] by blast
subgoal apply (intro Fnr.time-fft-tm-le B.length-num-blocks-carrier)
  using B.fill-num-blocks-carrier
  using Fnr.fermat-carrier-length
  unfolding defs[symmetric] by blast
subgoal
  apply (estimation estimate: time-evens-odds-tm-le)
  unfolding A.length-num-dft
  apply (estimation estimate: oe-n-le-n)
  by simp-all
subgoal
  apply (estimation estimate: time-evens-odds-tm-le)
  unfolding B.length-num-dft
  apply (estimation estimate: oe-n-le-n)
  by simp-all
subgoal by (rule recursive-time)
subgoal using prim-root-exponent-le by simp
subgoal apply (intro Fnr.time-iff-tm-le length-c-dft-odds)
  using c-dft-odds-carrier Fnr.fermat-carrier-length by auto
subgoal
  apply (estimation estimate: time-power-nat-tm-2-le)
  apply (estimation estimate: oe-n-le-n)
  by simp-all
subgoal apply (estimation estimate: time-upt-tm-le')
  apply (estimation estimate: oe-n-minus-1-le-n)
  by (simp-all only: power-add[symmetric] mult.assoc mult-2[symmetric])
subgoal apply (estimation estimate: time-upt-tm-le')
  apply (estimation estimate: oe-n-le-n)
  by (simp-all add: power-add[symmetric] mult-2[symmetric])
subgoal apply (estimation estimate: time-power-nat-tm-2-le)
  by (rule order.refl)
subgoal using oe-n-le-n by simp
subgoal unfolding time-replicate-tm
  using oe-n-le-n by simp
subgoal using oe-n-le-n
  by (simp add: oe-n-plus-two-pow-n-zeros-def)
subgoal by (rule time-ξ'-le)
subgoal by (rule time-ξ-le)
subgoal by (rule time-z-le)
subgoal unfolding time-z-filled-def[symmetric] by (rule time-z-filled-le)

```

```

subgoal unfolding time-replicate-tm
  using oe-n-minus-1-le-n by simp
subgoal using oe-n-minus-1-le-n by (simp add: length-z-filled)
subgoal apply (intro time-combine-z-tm-le[OF - segment-lens-pos])
  using length-z-complete-entries .
subgoal
  apply (estimation estimate: Fmr.time-from-nat-lsbf-tm-le[OF Fmr.e-ge-4, OF
m-gt-0 length-z-sum-le])
    by simp
  subgoal by (rule order.refl)
  subgoal by (rule order.refl)
done
also have ...  $\leq 8410 + 23508 * 2^m + 2069 * 2^n + 1141 * 2^{(2 * n)}$ 
+  $29 * n +$ 
   $32 * 2^{(2 * oe-n)} +$ 
   $2 * (oe-n * (2^{oe-n} * (76 + 232 * 2^n))) +$ 
   $2 * (2^{oe-n} * (66 + 174 * 2^n)) +$ 
   $2 * (2^{oe-n} * (6 + 3 * 2^n)) +$ 
   $492 * (n * 2^n) +$ 
   $2 * (2^{oe-n} * (15 + 5 * n)) +$ 
   $2 * (2^{oe-n} * (13 + 2^n + 2 * n)) +$ 
   $17 * m +$ 
time (karatsuba-mul-nat-tm A.num-Zn-pad B.num-Zn-pad) +
   $4 * karatsuba-lower-bound +$ 
schoenhage-strassen-Fm-bound  $n * 2^{(oe-n - 1)} +$ 
   $2^{(oe-n - 1)} * (66 + 174 * 2^n) +$ 
   $(oe-n - 1) * 2^{(oe-n - 1)} * (76 + 232 * 2^n) +$ 
   $32 * 2^{(2 * (oe-n - 1))} +$ 
   $(18 + 3 * 2^{(n - 1)} + 2 * 2^n + 2 * n) * 2^{oe-n}$ 
unfolding A.length-num-blocks A.length-num-Zn B.length-num-blocks B.length-num-Zn
apply (estimation estimate: prim-root-exponent-le)
apply (estimation estimate: prim-root-exponent-2-le)
unfolding segment-lens-def length-z-complete
by (simp add: add.assoc[symmetric])
also have ...  $\leq 8410 + 23508 * 2^m + 2069 * 2^n + 1141 * 2^{(2 * n)}$ 
+  $29 * n +$ 
   $128 * 2^{(2 * n)} +$ 
   $2464 * (n * 2^{(2 * n)}) +$ 
   $(264 * 2^n + 696 * 2^{(2 * n)}) +$ 
   $(24 * 2^n + 12 * 2^{(2 * n)}) +$ 
   $492 * (n * 2^n) +$ 
   $(60 * 2^n + 20 * (n * 2^n)) +$ 
   $(52 * 2^n + 4 * 2^{(2 * n)} + 8 * (n * 2^n)) +$ 
   $17 * m +$ 
time (karatsuba-mul-nat-tm A.num-Zn-pad B.num-Zn-pad) +
   $4 * karatsuba-lower-bound +$ 
schoenhage-strassen-Fm-bound  $n * 2^{(oe-n - 1)} +$ 
   $(66 * 2^n + 174 * 2^{(2 * n)}) +$ 
   $(76 * (n * 2^n) + n * (232 * 2^{(2 * n)})) +$ 

```

```

32 * 2 ^ (2 * n) +
(36 * 2 ^ n + 10 * 2 ^ (2 * n) + 4 * (n * 2 ^ n))
apply (intro add-mono order.refl)
subgoal apply (estimation estimate: oe-n-le-n) by simp-all
subgoal
proof -
  have 2 * (oe-n * (2 ^ oe-n * (76 + 232 * 2 ^ n))) ≤
    2 * ((2 * n) * (2 ^ (n + 1) * (76 + 232 * 2 ^ n)))
  apply (intro add-mono mult-le-mono order.refl)
  subgoal apply (estimation estimate: oe-n-le-n)
    unfolding mult-2 using n-gt-0 by simp
  subgoal by (estimation estimate: oe-n-le-n; simp)
  done
  also have ... = 8 * n * 2 ^ n * (76 + 232 * 2 ^ n)
    by simp
  also have ... ≤ 8 * n * 2 ^ n * (76 * 2 ^ n + 232 * 2 ^ n)
    by (intro add-mono mult-le-mono order.refl; simp)
  also have ... = 2464 * (n * 2 ^ (2 * n))
    by (simp add: mult-2 power-add)
  finally show ?thesis .
qed
subgoal apply (estimation estimate: oe-n-le-n)
  by (simp add: add-mult-distrib2 mult-2 power-add)
subgoal apply (estimation estimate: oe-n-le-n)
  by (simp add: add-mult-distrib2 mult-2 power-add)
subgoal apply (estimation estimate: oe-n-le-n)
  by (simp add: add-mult-distrib2 mult-2 power-add)
subgoal apply (estimation estimate: oe-n-le-n)
  by (simp add: add-mult-distrib2 power-add[symmetric])
subgoal apply (estimation estimate: oe-n-minus-1-le-n)
  by (simp add: add-mult-distrib2 mult-2 power-add)
subgoal apply (estimation estimate: oe-n-minus-1-le-n)
  by (simp add: add-mult-distrib2 mult-2 power-add)
subgoal apply (estimation estimate: oe-n-minus-1-le-n)
  by (simp add: add-mult-distrib2 mult-2 power-add)
subgoal apply (estimation estimate: oe-n-le-n)
  using power-increasing[of n - 1 n 2::nat]
  by (simp add: add-mult-distrib2 add-mult-distrib mult-2[of n, symmetric]
power-add[symmetric])
done
also have ... = 600 * (n * 2 ^ n) + 2197 * 2 ^ (2 * n) + 2571 * 2 ^ n +
2696 * (n * 2 ^ (2 * n)) +
8410 +
23508 * 2 ^ m +
29 * n +
17 * m +
time (karatsuba-mul-nat-tm A.num-Zn-pad B.num-Zn-pad) +
4 * karatsuba-lower-bound +
schoenhage-strassen-Fm-bound n * 2 ^ (oe-n - 1)

```

**by** (*simp add: add.assoc[symmetric]*)  
**also have** ...  $\leq 600 * (n * 2^{\wedge} n) + 2197 * 2^{\wedge}(2 * n) + 2571 * 2^{\wedge} n +$   
 $2696 * (n * 2^{\wedge}(2 * n)) +$   
 $8410 +$   
 $23508 * 2^{\wedge} m +$   
 $29 * n +$   
 $17 * m +$   
*time-karatsuba-mul-nat-bound* ((3 \* n + 5) \* 2<sup>oe-n</sup>) +  
 $4 * \textit{karatsuba-lower-bound} +$   
*schoenhage-strassen-Fm-bound* n \* 2<sup>(oe-n - 1)</sup>  
**apply** (*intro add-mono order.refl time-karatsuba-mul-nat-tm-le*)  
**unfolding** *A.length-num-Zn-pad B.length-num-Zn-pad pad-length-def* **by** *simp*  
**also have** ...  $\leq 600 * (n * 2^{\wedge}(2 * n)) + 2197 * (n * 2^{\wedge}(2 * n)) + 2571 *$   
 $(n * 2^{\wedge}(2 * n)) +$   
 $2696 * (n * 2^{\wedge}(2 * n)) +$   
 $8410 +$   
 $23508 * 2^{\wedge} m +$   
 $29 * (n * 2^{\wedge}(2 * n)) +$   
 $17 * 2^{\wedge} m +$   
*time-karatsuba-mul-nat-bound* ((3 \* n + 5) \* 2<sup>oe-n</sup>) +  
 $4 * \textit{karatsuba-lower-bound} +$   
*schoenhage-strassen-Fm-bound* n \* 2<sup>(oe-n - 1)</sup>  
**apply** (*intro add-mono mult-le-mono order.refl power-increasing*)  
**subgoal by** *simp*  
**subgoal by** *simp*  
**subgoal using** *n-gt-0* **by** *simp*  
**subgoal using** *power-increasing*[of n 2 \* n 2::nat]  $\langle 2^{\wedge}(2 * n) \leq n * 2^{\wedge}(2$   
 $* n) \rangle$  **by** *linarith*  
**subgoal by** *simp*  
**subgoal by** *simp*  
**done**  
**also have** ... =  $23525 * 2^{\wedge} m + 8093 * (n * 2^{\wedge}(2 * n)) + 8410 +$   
*time-karatsuba-mul-nat-bound* ((3 \* n + 5) \* 2<sup>oe-n</sup>) +  
 $4 * \textit{karatsuba-lower-bound} +$   
*schoenhage-strassen-Fm-bound* n \* 2<sup>(oe-n - 1)</sup>  
**by** *simp*  
**also have** ... = *schoenhage-strassen-Fm-bound* m  
**unfolding** *schoenhage-strassen-Fm-bound.simps*[of m] *Let-def defs*[*symmetric*]  
**using** 3 **by** *argo*  
**finally show** *?thesis* .  
**qed**  
**qed**

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

**lemma** *real-divide-mult-eq*:  
**assumes** (c :: real)  $\neq 0$

```

shows  $a / c * c = a$ 
using assms by simp

lemma powr-unbounded:
  assumes  $(c :: \text{real}) > 0$ 
  shows eventually  $(\lambda x. d \leq x \text{ powr } c)$  at-top
proof (cases  $d > 0$ )
  case True
  define N where  $N = d \text{ powr } (1 / c)$ 
  have  $d \leq x \text{ powr } c$  if  $x \geq N$  for x
  proof -
    have  $d = d \text{ powr } 1$  apply (intro powr-one[symmetric]) using True by simp
    also have  $\dots = (d \text{ powr } (1 / c)) \text{ powr } c$ 
      unfolding powr-powr
    apply (intro arg-cong2[where  $f = (\text{powr})$ ] refl real-divide-mult-eq[symmetric])
  using assms by simp
  also have  $\dots = N \text{ powr } c$  unfolding N-def by (rule refl)
  also have  $\dots \leq x \text{ powr } c$ 
    apply (intro powr-mono2)
    subgoal using assms by simp
    subgoal unfolding N-def by (rule powr-ge-zero)
    subgoal by (rule that)
  done
  finally show ?thesis .
qed
then show ?thesis unfolding eventually-at-top-linorder by blast
next
  case False
  then show ?thesis
    apply (intro always-eventually allI)
    subgoal for x using powr-ge-zero[of x c] by argo
  done
qed

lemma time-kar-le-kar-const:
  assumes  $x > 0$ 
  shows time-karatsuba-mul-nat-bound  $x \leq \text{karatsuba-const} * \text{nat} (\text{floor} (\text{real } x \text{ powr } \log 2 3))$ 
proof -
  have  $\exists c. (\forall x. x \geq 1 \longrightarrow \text{time-karatsuba-mul-nat-bound } x \leq c * \text{nat} (\text{floor} (\text{real } x \text{ powr } \log 2 3)))$ 
  apply (intro eventually-early-nat)
  subgoal
    apply (intro bigo-floor)
    subgoal by (rule time-karatsuba-mul-nat-bound-bigo)
    subgoal apply (intro eventually-nat-real[OF powr-unbounded[of  $\log 2 3 1$ ]])
  by simp
  done
  subgoal premises prems for x

```

```

proof –
  have  $real\ x \geq 1$  using prems by simp
  then have  $real\ x\ powr\ log\ 2\ 3 \geq 1$  powr log 2 3
    by (intro powr-mono2; simp)
  then have  $real\ x\ powr\ log\ 2\ 3 \geq 1$  by simp
  then have  $floor\ (real\ x\ powr\ log\ 2\ 3) \geq 1$  by simp
  then show ?thesis by simp
qed
done
then have  $\forall x > 0. time-karatsuba-mul-nat-bound\ x \leq karatsuba-const * nat$ 
 $[real\ x\ powr\ log\ 2\ 3]$ 
  unfolding karatsuba-const-def
  apply (intro someI-ex[of  $\lambda c. \forall x > 0. time-karatsuba-mul-nat-bound\ x \leq c * nat$ 
 $[real\ x\ powr\ log\ 2\ 3]]$ )
  by (metis int-one-le-iff-zero-less nat-int nat-mono nat-one-as-int of-nat-0-less-iff)
  then show ?thesis using assms by blast
qed

lemma poly-smallo-exp:
  assumes  $c > 1$ 
  shows  $(\lambda n. (real\ n)\ powr\ d) \in o(\lambda n. c\ powr\ (real\ n))$ 
  by (intro smallo-real-nat-transfer power-smallo-exponential assms)

lemma kar-aux-lem:  $(\lambda n. real\ (n * 2^{\wedge} n)\ powr\ log\ 2\ 3) \in O(\lambda n. real\ (2^{\wedge} (2 * n)))$ 
proof –
  define c where  $c = 2\ powr\ (2 / log\ 2\ 3 - 1)$ 
  have  $c > 1$  unfolding c-def
    apply (intro gr-one-powr)
    subgoal by simp
    subgoal apply simp using less-powr-iff[of 2 3 2] by simp
  done
  have  $1: (log\ 2\ c + 1) * log\ 2\ 3 = 2$ 
  proof –
    have  $log\ 2\ c = 2 / log\ 2\ 3 - 1$ 
      unfolding c-def by (intro log-powr-cancel; simp)
    then have  $log\ 2\ c + 1 = 2 / log\ 2\ 3$  by simp
    then have  $(log\ 2\ c + 1) * log\ 2\ 3 = 2 / log\ 2\ 3 * log\ 2\ 3$  by simp
    also have  $\dots = 2$  apply (intro real-divide-mult-eq)
      using zero-less-log-cancel-iff[of 2 3] by linarith
    finally show ?thesis .
  qed
from poly-smallo-exp[OF  $\langle c > 1 \rangle$ , of 1] have  $real \in o(\lambda n. c\ powr\ real\ n)$  by
simp
  then have  $(\lambda n. real\ (n * 2^{\wedge} n)) \in o(\lambda n. (c\ powr\ real\ n) * real\ (2^{\wedge} n))$ 
    by simp
  then have  $(\lambda n. real\ (n * 2^{\wedge} n)) \in O(\lambda n. (c\ powr\ real\ n) * real\ (2^{\wedge} n))$ 
    using landau-o.small-imp-big by blast
  then have  $(\lambda n. real\ (n * 2^{\wedge} n)\ powr\ log\ 2\ 3) \in O(\lambda n. ((c\ powr\ real\ n) * real$ 

```

$(2 \wedge n)$  *powr log 2 3*)  
**by** (*intro iffD2*[*OF bigo-powr-iff*]; *simp*)  
**also have** ... =  $O(\lambda n. ((c \text{ powr } \text{real } n) * 2 \text{ powr } (\text{real } n)) \text{ powr } \log 2 3)$   
**using** *powr-realpow*[*of 2*] **by** *simp*  
**also have** ... =  $O(\lambda n. (((2 \text{ powr } \log 2 c) \text{ powr } \text{real } n) * 2 \text{ powr } (\text{real } n)) \text{ powr } \log 2 3)$   
**using** *powr-log-cancel*[*of 2 c*]  $\langle c > 1 \rangle$  **by** *simp*  
**also have** ... =  $O(\lambda n. 2 \text{ powr } ((\log 2 c * \text{real } n + \text{real } n) * \log 2 3))$   
**unfolding** *powr-powr powr-add*[*symmetric*] **by** (*rule refl*)  
**also have** ... =  $O(\lambda n. 2 \text{ powr } (\text{real } n * (\log 2 c + 1) * \log 2 3))$   
**apply** (*intro-cong* [*cong-tag-1* ( $\lambda f. O(f)$ ), *cong-tag-2* (*powr*), *cong-tag-2* (\*)]  
*more: refl ext*)  
**by** *argo*  
**also have** ... =  $O(\lambda n. 2 \text{ powr } (\text{real } n * 2))$   
**apply** (*intro-cong* [*cong-tag-1* ( $\lambda f. O(f)$ ), *cong-tag-2* (*powr*)] *more: ext refl*)  
**using** 1 **by** *simp*  
**also have** ... =  $O(\lambda n. \text{real } (2 \wedge (2 * n)))$   
**apply** (*intro-cong* [*cong-tag-1* ( $\lambda f. O(f)$ )] *more: ext*)  
**subgoal for** *n*  
**using** *powr-realpow*[*of 2 2 \* n, symmetric*]  
**by** (*simp add: mult.commute*)  
**done**  
**finally show** ?*thesis* .  
**qed**

**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** –  
**have** ( $\exists c. \forall n \geq 1. \text{real } (n * 2 \wedge n) \text{ powr } \log 2 3 \leq c * \text{real } (2 \wedge (2 * n))$ )  
**using** *eventually-early-real*[*OF kar-aux-lem*] **by** *simp*  
**then have**  $\forall n \geq 1. \text{real } (n * 2 \wedge n) \text{ powr } \log 2 3 \leq \text{kar-aux-const} * \text{real } (2 \wedge (2 * n))$   
**unfolding** *kar-aux-const-def* **apply** (*intro someI-ex*[*of*  $\lambda c. \forall n \geq 1. \text{real } (n * 2 \wedge n) \text{ powr } \log 2 3 \leq c * \text{real } (2 \wedge (2 * n))$ ]) .  
**then show** ?*thesis* **using** *assms* **by** *simp*  
**qed**

**lemma** *kar-aux-const-gt-0*: *kar-aux-const* > 0

**proof** (*rule ccontr*)  
**assume**  $\neg \text{kar-aux-const} > 0$   
**then have** *kar-aux-const*  $\leq 0$  **by** *simp*  
**then show** *False* **using** *kar-aux-lem-le*[*of 1*] **by** *simp*  
**qed**

**definition** *kar-aux-const-nat* **where** *kar-aux-const-nat* = *karatsuba-const* \* *nat*

[16 powr log 2 3] \* nat [kar-aux-const]

**definition** *s-const1* **where** *s-const1* = 55897 + 4 \* *kar-aux-const-nat*

**definition** *s-const2* **where** *s-const2* = 8410 + 4 \* *karatsuba-lower-bound*

**function** *schoenhage-strassen-Fm-bound'* :: nat ⇒ nat **where**

*m* < 3 ⇒ *schoenhage-strassen-Fm-bound'* *m* = 5336

| *m* ≥ 3 ⇒ *schoenhage-strassen-Fm-bound'* *m* = *s-const1* \* (*m* \* 2 ^ *m*) +  
*s-const2* + *schoenhage-strassen-Fm-bound'* ((*m* + 2) div 2) \* 2 ^ ((*m* + 1) div 2)

**by** *fastforce+*

**termination**

**by** (*relation Wellfounded.measure* (λ*m*. *m*); *simp*)

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

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

**shows** *schoenhage-strassen-Fm-bound* *m* ≤ *schoenhage-strassen-Fm-bound'* *m*

**proof** (*induction m rule: less-induct*)

**case** (*less m*)

**show** *?case*

**proof** (*cases m < 3*)

**case** *True*

**from** *True* **have** *schoenhage-strassen-Fm-bound* *m* = 5336 **unfolding** *schoenhage-strassen-Fm-bound.simps*[*of m*] **by** *simp*

**also** **have** ... = *schoenhage-strassen-Fm-bound'* *m* **using** *schoenhage-strassen-Fm-bound'.simps*  
*True* **by** *simp*

**finally** **show** *?thesis* **by** *simp*

**next**

**case** *False*

**then** **interpret** *m-lemmas: m-lemmas m*

**by** (*unfold-locales; simp*)

**from** *False* **have** *m* ≥ 3 **by** *simp*

**define** *n* **where** *n* = *m-lemmas.n*

**define** *oe-n* **where** *oe-n* = *m-lemmas.oe-n*

**have** *kar-arg-pos*: (*3* \* *n* + 5) \* 2 ^ *oe-n* > 0 **by** *simp*

**have** *fm*: *schoenhage-strassen-Fm-bound* *m* = 23525 \* 2 ^ *m* + 8093 \* (*n* \* 2 ^ (2 \* *n*)) + 8410 +

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

4 \* *karatsuba-lower-bound* +

*schoenhage-strassen-Fm-bound* *n* \* 2 ^ (*oe-n* - 1) (**is** - = *?t1* + *?t2* + *?t3* + *?t4* + *?t5* + *?t6*)

**unfolding** *schoenhage-strassen-Fm-bound.simps*[*of m*] *n-def* *oe-n-def* **using**  
*False m-lemmas.n-def m-lemmas.oe-n-def*

**by** *simp*

**have** *?t4* ≤ *karatsuba-const* \* nat [real ((*3* \* *n* + 5) \* 2 ^ *oe-n*) powr log 2 3]

**by** (*intro time-kar-le-kar-const*[*OF kar-arg-pos*])

```

also have ... ≤ karatsuba-const * nat [real ((8 * n) * 2 ^ (n + 1)) powr log 2
3]
apply (intro add-mono order.refl mult-le-mono nat-mono floor-mono powr-mono2
iffD1[OF real-mono] power-increasing)
using m-lemmas.oe-n-gt-0 m-lemmas.n-gt-0 m-lemmas.oe-n-le-n by (simp-all
add: n-def oe-n-def)
also have ... = karatsuba-const * nat [real (16 * (n * 2 ^ n)) powr log 2 3]
by simp
also have ... = karatsuba-const * nat [(16 powr log 2 3) * ((n * 2 ^ n) powr
log 2 3)]
unfolding real-multiplicative using powr-mult[of real 16 real n * real (2 ^ n)
log 2 3]
by simp
also have ... ≤ karatsuba-const * nat [(16 powr log 2 3) * (kar-aux-const * real
(2 ^ (2 * n)))]
apply (intro mult-le-mono order.refl nat-mono floor-mono mult-mono kar-aux-lem-le)
subgoal using m-lemmas.n-gt-0 unfolding n-def .
subgoal by simp
subgoal by simp
done
also have ... ≤ karatsuba-const * nat [(16 powr log 2 3) * (kar-aux-const * real
(2 ^ (2 * n)))]
by (intro mult-le-mono order.refl nat-mono floor-le-ceiling)
also have ... ≤ karatsuba-const * (nat ([16 powr log 2 3] * [kar-aux-const *
real (2 ^ (2 * n))]))
using kar-aux-const-gt-0 by (intro mult-le-mono order.refl nat-mono mult-ceiling-le;
simp)
also have ... = karatsuba-const * (nat [16 powr log 2 3] * nat [kar-aux-const
* real (2 ^ (2 * n))])
apply (intro arg-cong2[where f = (*)] refl nat-mult-distrib)
using powr-ge-zero[of 16 log 2 3] by linarith
also have ... ≤ karatsuba-const * (nat [16 powr log 2 3] * nat ([kar-aux-const]
* [real (2 ^ (2 * n))]))
apply (intro mult-le-mono order.refl nat-mono mult-ceiling-le)
using kar-aux-const-gt-0 by simp-all
also have ... = karatsuba-const * (nat [16 powr log 2 3] * (nat [kar-aux-const]
* nat [real (2 ^ (2 * n))]))
apply (intro arg-cong2[where f = (*)] refl nat-mult-distrib)
using kar-aux-const-gt-0 by simp
also have ... = karatsuba-const * nat [16 powr log 2 3] * nat [kar-aux-const]
* (2 ^ (2 * n))
by simp
also have ... = kar-aux-const-nat * 2 ^ (2 * n)
unfolding kar-aux-const-nat-def[symmetric] by (rule refl)
also have ... ≤ kar-aux-const-nat * (n * 2 ^ (2 * n))
using m-lemmas.n-gt-0 n-def by simp
finally have t4-le: ?t4 ≤ ... .
have schoenhage-strassen-Fm-bound m ≤ ?t1 + ?t2 + ?t3 + kar-aux-const-nat
* (n * 2 ^ (2 * n)) + ?t5 + ?t6

```

**unfolding** *fm*  
**by** (*intro add-mono order.refl t4-le*)  
**also have** ... = ?*t1* + (8093 + *kar-aux-const-nat*) \* (*n* \* 2<sup>(2 \* *n*)</sup>) + 8410  
+ 4 \* *karatsuba-lower-bound* + *schoenhage-strassen-Fm-bound* *n* \* 2<sup>(*oe-n* - 1)</sup>  
**by** (*simp add: add-mult-distrib*)  
**also have** ... ≤ 23525 \* (*m* \* 2<sup>*m*</sup>) + (8093 + *kar-aux-const-nat*) \* (*m* \* (2 \* 2<sup>(*m* + 1)</sup>)) + 8410 + 4 \* *karatsuba-lower-bound* + *schoenhage-strassen-Fm-bound*  
*n* \* 2<sup>(*oe-n* - 1)</sup>  
**apply** (*intro add-mono order.refl mult-le-mono*)  
**subgoal using** *m-lemmas.m-gt-0* **by** *simp*  
**subgoal using** *m-lemmas.n-lt-m n-def* **by** *simp*  
**subgoal using** *m-lemmas.two-pow-two-n-le n-def* **by** *simp*  
**done**  
**also have** ... = (55897 + 4 \* *kar-aux-const-nat*) \* (*m* \* 2<sup>*m*</sup>) + (8410 + 4  
\* *karatsuba-lower-bound*) + *schoenhage-strassen-Fm-bound* *n* \* 2<sup>(*oe-n* - 1)</sup>  
**by** (*simp add: add-mult-distrib*)  
**also have** ... ≤ (55897 + 4 \* *kar-aux-const-nat*) \* (*m* \* 2<sup>*m*</sup>) + (8410 + 4  
\* *karatsuba-lower-bound*) + *schoenhage-strassen-Fm-bound'* *n* \* 2<sup>(*oe-n* - 1)</sup>  
**apply** (*intro add-mono order.refl mult-le-mono less.IH*)  
**unfolding** *n-def* **using** *m-lemmas.n-lt-m* .  
**also have** ... = (55897 + 4 \* *kar-aux-const-nat*) \* (*m* \* 2<sup>*m*</sup>) + (8410 + 4  
\* *karatsuba-lower-bound*) + *schoenhage-strassen-Fm-bound'* ((*m* + 2) div 2) \* 2<sup>((*m* + 1) div 2)</sup>  
**apply** (*intro-cong [cong-tag-2 (+), cong-tag-2 (\*), cong-tag-2 (∧), cong-tag-1*  
*schoenhage-strassen-Fm-bound'] more: refl*)  
**subgoal unfolding** *n-def m-lemmas.n-def* **by** (*cases odd m; simp*)  
**subgoal unfolding** *oe-n-def m-lemmas.oe-n-def m-lemmas.n-def* **by** (*cases*  
*odd m; simp*)  
**done**  
**also have** ... = *schoenhage-strassen-Fm-bound'* *m* **using** *schoenhage-strassen-Fm-bound'.simps*[*of*  
*m*] *False* **unfolding** *s-const1-def[symmetric] s-const2-def[symmetric]* **by** *simp*  
**finally show** ?*thesis* .  
**qed**  
**qed**

**definition** *γ-0* **where** *γ-0* = 2 \* *s-const1* + *s-const2*

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

**assumes** *n* ≥ 3

**shows** *schoenhage-strassen-Fm-bound'* (2 \* *n* - 2) ≤ *γ-0* \* *n* \* 2<sup>(2 \* *n* - 2)</sup> + *schoenhage-strassen-Fm-bound'* *n* \* 2<sup>(*n* - 1)</sup>

**and** *schoenhage-strassen-Fm-bound'* (2 \* *n* - 1) ≤ *γ-0* \* *n* \* 2<sup>(2 \* *n* - 1)</sup> + *schoenhage-strassen-Fm-bound'* *n* \* 2<sup>*n*</sup>

**proof** –

**from** *assms* **have** *r*: 2 \* *n* - 2 ≥ 4 **by** *linarith*

**from** *r* **have** *schoenhage-strassen-Fm-bound'* (2 \* *n* - 1) = *s-const1* \* (2 \* *n* - 1) \* 2<sup>(2 \* *n* - 1)</sup> + *s-const2* + *schoenhage-strassen-Fm-bound'* *n* \* 2<sup>*n*</sup>

**using** *schoenhage-strassen-Fm-bound'.simps*[*of* 2 \* *n* - 1] **by** *auto*

**also have** ... ≤ *s-const1* \* (2 \* *n*) \* 2<sup>(2 \* *n* - 1)</sup> + *s-const2* \* (*n* \* 2<sup>(2 \* *n* - 1)</sup>)

$* n - 1)) + \text{schoenhage-strassen-Fm-bound}' n * 2^{\wedge} n$   
**apply** (*intro add-mono order.refl mult-le-mono*)  
**subgoal by simp**  
**subgoal using *assms* by simp**  
**done**  
**also have**  $\dots = \gamma-0 * n * 2^{\wedge} (2 * n - 1) + \text{schoenhage-strassen-Fm-bound}' n$   
 $* 2^{\wedge} n$   
**unfolding**  $\gamma-0\text{-def}$  **by** (*simp add: add-mult-distrib*)  
**finally show**  $\text{schoenhage-strassen-Fm-bound}' (2 * n - 1) \leq \dots$  .  
**from**  $r$  **have**  $\text{schoenhage-strassen-Fm-bound}' (2 * n - 2) = s\text{-const1} * ((2 * n$   
 $- 2) * 2^{\wedge} (2 * n - 2)) + s\text{-const2} +$   
 $\text{schoenhage-strassen-Fm-bound}' ((2 * n - 2 + 2) \text{div } 2) * 2^{\wedge} ((2 * n - 2 +$   
 $1) \text{div } 2)$   
**using**  $\text{schoenhage-strassen-Fm-bound}'.\text{simps}(2)[\text{of } 2 * n - 2]$  **by fastforce**  
**also have**  $\dots = s\text{-const1} * ((2 * n - 2) * 2^{\wedge} (2 * n - 2)) + s\text{-const2} +$   
 $\text{schoenhage-strassen-Fm-bound}' n * 2^{\wedge} (n - 1)$   
**apply** (*intro-cong [cong-tag-2 (+), cong-tag-2 (\*), cong-tag-2 ( $\wedge$ ), cong-tag-1*  
 $\text{schoenhage-strassen-Fm-bound}']$  *more: refl*)  
**subgoal using**  $r$  **by linarith**  
**subgoal using**  $r$  **by linarith**  
**done**  
**also have**  $\dots \leq s\text{-const1} * ((2 * n) * 2^{\wedge} (2 * n - 2)) + s\text{-const2} * (n * 2^{\wedge} (2$   
 $* n - 2)) + \text{schoenhage-strassen-Fm-bound}' n * 2^{\wedge} (n - 1)$   
**apply** (*intro add-mono order.refl mult-le-mono*)  
**subgoal by simp**  
**subgoal using *assms* by simp**  
**done**  
**also have**  $\dots = \gamma-0 * n * 2^{\wedge} (2 * n - 2) + \text{schoenhage-strassen-Fm-bound}' n$   
 $* 2^{\wedge} (n - 1)$   
**unfolding**  $\gamma-0\text{-def}$  **by** (*simp add: add-mult-distrib*)  
**finally show**  $\text{schoenhage-strassen-Fm-bound}' (2 * n - 2) \leq \dots$  .  
**qed**

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

**lemma**  $\text{schoenhage-strassen-Fm-bound}'\text{-le-aux1}$ :  
**assumes**  $m \leq 2^{\wedge} \text{Suc } k + 1$   
**shows**  $\text{schoenhage-strassen-Fm-bound}' m \leq \gamma * \text{Suc } k * 2^{\wedge} (\text{Suc } k + m)$   
**using** *assms* **proof** (*induction k arbitrary: m rule: less-induct*)  
**case** (*less k*)  
**consider**  $m \leq 3 \mid m \geq 4$  **by linarith**  
**then show** *?case*  
**proof** *cases*  
**case** 1  
**then have**  $m \in \{0, 1, 2, 3\}$  **by auto**  
**then have**  $\text{schoenhage-strassen-Fm-bound}' m \in \{\gamma-0, \text{schoenhage-strassen-Fm-bound}'$   
 $0, \text{schoenhage-strassen-Fm-bound}' 1, \text{schoenhage-strassen-Fm-bound}' 2, \text{schoen-}$

```

hage-strassen-Fm-bound' 3} by auto
  then have schoenhage-strassen-Fm-bound'  $m \leq \gamma$  unfolding  $\gamma$ -def by (intro
Max.coboundedI; simp)
  also have ... =  $\gamma * 1 * 1$  by simp
  also have ...  $\leq \gamma * \text{Suc } k * 2^{\wedge}(k + m)$ 
    by (intro mult-le-mono order.refl; simp)
  finally show ?thesis .
next
case 2
have  $k > 0$ 
proof (rule ccontr)
  assume  $\neg k > 0$ 
  with less.premis have  $m \leq 3$  by simp
  thus False using 2 by simp
qed
then obtain  $k'$  where  $k = \text{Suc } k' \ k' < k$ 
  using gr0-conv-Suc by auto
  have ih': schoenhage-strassen-Fm-bound'  $m \leq \gamma * k * 2^{\wedge}(k + m)$  if  $m \leq 2^{\wedge}k + 1$  for  $m$ 
    using less.IH[OF  $\langle k' < k \rangle$ ] unfolding  $\langle k = \text{Suc } k' \rangle$ [symmetric] using that
  by simp

interpret ml: m-lemmas m
  apply unfold-locales
  using 2 by simp

define  $n'$  where  $n' = (\text{if odd } m \text{ then } ml.n \text{ else } ml.n - 1)$ 
have  $n' = ml.oe-n - 1$ 
  unfolding  $n'$ -def ml.oe-n-def by simp
have  $ml.n + n' = m + 1$ 
  unfolding ml.m1  $\langle n' = ml.oe-n - 1 \rangle$ 
  using Nat.add-diff-assoc[of 1 ml.oe-n ml.n]
  using Nat.diff-add-assoc2[of 1 ml.n ml.oe-n]
  using ml.oe-n-gt-0 ml.n-gt-0
  by simp

have  $ml.n \geq 3$  using 2 ml.mn by (cases odd m; simp)
have  $ml.n \leq 2^{\wedge}k + 1$ 
  using less.premis ml.mn by (cases odd m; simp)
note ih = ih'[OF this]

  have schoenhage-strassen-Fm-bound'  $m \leq \gamma * 0 * ml.n * 2^{\wedge}m + \text{schoen-}$ 
hage-strassen-Fm-bound'  $ml.n * 2^{\wedge}n'$ 
  unfolding  $n'$ -def
  using schoenhage-strassen-Fm-bound'-oe-rec[OF  $\langle ml.n \geq 3 \rangle$ ] ml.mn
  by (cases odd m; algebra)
also have ...  $\leq \gamma * ml.n * 2^{\wedge}m + (\gamma * k * 2^{\wedge}(k + ml.n)) * 2^{\wedge}n'$ 
  apply (intro add-mono mult-le-mono order.refl ih)
  apply (unfold  $\gamma$ -def)

```

```

    apply simp
  done
  also have ... =  $\gamma * ml.n * 2^m + \gamma * k * 2^{(k + ml.n + n')}$ 
    by (simp add: power-add)
  also have ... =  $\gamma * ml.n * 2^m + \gamma * k * 2^{(k + m + 1)}$ 
    using  $\langle ml.n + n' = m + 1 \rangle$  by (simp add: add.assoc)
  also have ... =  $\gamma * 2^m * (ml.n + k * 2^{(k + 1)})$ 
    by (simp add: Nat.add-mult-distrib2 power-add)
  also have ...  $\leq \gamma * 2^m * (2^{(k + 1)} + k * 2^{(k + 1)})$ 
    apply (intro mono-intros)
    apply (estimation estimate:  $\langle ml.n \leq 2^{k + 1} \rangle$ )
    apply simp
  done
  also have ... =  $\gamma * 2^m * (k + 1) * 2^{(k + 1)}$ 
    by (simp add: Nat.add-mult-distrib2 Nat.add-mult-distrib)
  also have ... =  $\gamma * (k + 1) * 2^{(k + 1 + m)}$ 
    by (simp add: power-add Nat.add-mult-distrib)
  finally show ?thesis by simp
qed
qed

lemma schoenhage-strassen-Fm-bound'-le-aux2:
  assumes  $k \geq 1$ 
  assumes  $m \leq 2^{k + 1}$ 
  shows schoenhage-strassen-Fm-bound'  $m \leq \gamma * k * 2^{(k + m)}$ 
proof -
  from assms(1) obtain  $k'$  where  $k = \text{Suc } k'$ 
  by (metis Suc-le-D numeral-nat(7))
  then show ?thesis using schoenhage-strassen-Fm-bound'-le-aux1 [of  $m k'$ ] assms(2)
  by argo
qed

```

## 4.1 Multiplication in $\mathbb{N}$

```

definition schoenhage-strassen-mul-tm where
schoenhage-strassen-mul-tm a b = 1 do {
  bits-a  $\leftarrow$  length-tm a  $\gg$  bitsize-tm;
  bits-b  $\leftarrow$  length-tm b  $\gg$  bitsize-tm;
  m'  $\leftarrow$  max-nat-tm bits-a bits-b;
  m  $\leftarrow$  m' +t 1;
  m-plus-1  $\leftarrow$  m +t 1;
  car-len  $\leftarrow$  2t m-plus-1;
  fill-a  $\leftarrow$  fill-tm car-len a;
  fill-b  $\leftarrow$  fill-tm car-len b;
  fm-result  $\leftarrow$  schoenhage-strassen-tm m fill-a fill-b;
  int-lsbf-fermat.reduce-tm m fm-result
}

```

```

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

```

```

    val (schoenhage-strassen-mul-tm a b) = schoenhage-strassen-mul a b
  proof -
    interpret schoenhage-strassen-mul-context a b .

    have val-fm[val-simp]: val (schoenhage-strassen-tm m fill-a fill-b) = schoen-
hage-strassen m fill-a fill-b
      apply (intro val-schoenhage-strassen-tm)
      subgoal unfolding fill-a-def car-len-def
        by (intro int-lsbfermat.fermat-non-unique-carrierI length-fill length-a')
      subgoal unfolding fill-b-def car-len-def
        by (intro int-lsbfermat.fermat-non-unique-carrierI length-fill length-b')
      done

    show ?thesis
    unfolding schoenhage-strassen-mul-tm-def schoenhage-strassen-mul-def
    unfolding val-simp Let-def int-lsbfermat.val-reduce-tm defs[symmetric]
    by (rule refl)
  qed

  lemma real-power:  $a > 0 \implies \text{real } ((a :: \text{nat}) \wedge x) = \text{real } a \text{ powr } \text{real } x$ 
    using powr-realpow[of real a x] by simp

  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 length a  $\leq n$  length b  $\leq n$ 
    shows time (schoenhage-strassen-mul-tm a b)  $\leq$  schoenhage-strassen-bound n
  proof -
    interpret schoenhage-strassen-mul-context a b .

    have m-le:  $m \leq \text{bitsize } n + 1$ 
      unfolding defs
      by (intro add-mono order.refl max.boundedI bitsize-mono assms)

    have m-gt-0:  $m > 0$  unfolding m-def by simp

    have bits-a-le:  $\text{bits-a} \leq m - 1$ 
      unfolding bits-a-def
      by (intro iffD2[OF bitsize-length] length-a)
    have bits-b-le:  $\text{bits-b} \leq m - 1$ 
      unfolding bits-b-def
      by (intro iffD2[OF bitsize-length] length-b)

    have a-carrier:  $\text{fill-a} \in \text{int-lsbfermat.fermat-non-unique-carrier } m$ 
      unfolding fill-a-def car-len-def
      by (intro int-lsbfermat.fermat-non-unique-carrierI length-fill length-a')

```

**have** *b-carrier*:  $\text{fill-b} \in \text{int-lsbj-fermat.fermat-non-unique-carrier } m$   
**unfolding** *fill-b-def car-len-def*  
**by** (*intro int-lsbj-fermat.fermat-non-unique-carrierI length-fill length-b'*)

**have** *val-fm*:  $\text{val } (\text{schoenhage-strassen-tm } m \text{ fill-a fill-b}) = \text{schoenhage-strassen } m \text{ fill-a fill-b}$   
**by** (*intro val-schoenhage-strassen-tm a-carrier b-carrier*)

**have**  $\text{time } (\text{schoenhage-strassen-mul-tm } a \ b) = \text{time } (\text{length-tm } a) + \text{time } (\text{bitsize-tm } (\text{length } a)) + \text{time } (\text{length-tm } b) + \text{time } (\text{bitsize-tm } (\text{length } b)) + \text{time } (\text{max-nat-tm } \text{bits-a } \text{bits-b}) + \text{time } (m' +_t 1) + \text{time } (m +_t 1) + \text{time } (2 \hat{^}_t (m + 1)) + \text{time } (\text{fill-tm } \text{car-len } a) + \text{time } (\text{fill-tm } \text{car-len } b) + \text{time } (\text{schoenhage-strassen-tm } m \text{ fill-a fill-b}) + \text{time } (\text{int-lsbj-fermat.reduce-tm } m \ (\text{schoenhage-strassen } m \ \text{fill-a fill-b})) + 1$   
**unfolding** *schoenhage-strassen-mul-tm-def*  
**unfolding** *tm-time-simps defs[symmetric] val-length-tm val-bitsize-tm val-simps val-max-nat-tm Let-def val-plus-nat-tm val-power-nat-tm val-fill-tm val-fm add.assoc[symmetric]*  
**by** (*rule refl*)

**also have**  $\dots \leq (n + 1) + (72 * n + 23) + (n + 1) + (72 * n + 23) + (2 * (m - 1) + 3) + m + (m + 1) + 12 * 2 \hat{^}(m + 1) + (3 * 2 \hat{^}(m + 1) + 5) + (3 * 2 \hat{^}(m + 1) + 5) + \text{schoenhage-strassen-Fm-bound}' m + (155 + 108 * 2 \hat{^}(m + 1)) + 1$   
**apply** (*intro add-mono order.refl*)  
**subgoal using** *assms by simp*  
**subgoal apply** (*estimation estimate: time-bitsize-tm-le*) **using** *assms by simp*  
**subgoal using** *assms by simp*  
**subgoal apply** (*estimation estimate: time-bitsize-tm-le*) **using** *assms by simp*  
**subgoal apply** (*estimation estimate: time-max-nat-tm-le*)  
**apply** (*estimation estimate: min.cobounded1*)  
**apply** (*estimation estimate: bits-a-le*)  
**by** (*rule order.refl*)  
**subgoal by** (*simp add: m-def*)  
**subgoal by** *simp*  
**subgoal apply** (*estimation estimate: time-power-nat-tm-2-le*)  
**unfolding** *defs[symmetric]* **by** (*rule order.refl*)  
**subgoal apply** (*estimation estimate: time-fill-tm-le*)

**apply** (*estimation estimate: length-a'*)  
**unfolding** *defs[symmetric]* **by** *simp*  
**subgoal apply** (*estimation estimate: time-fill-tm-le*)  
**apply** (*estimation estimate: length-b'*)  
**unfolding** *defs[symmetric]* **by** *simp*  
**subgoal**  
**apply** (*estimation estimate: time-schoenhage-strassen-tm-le[OF a-carrier*  
*b-carrier]*)  
**apply** (*estimation estimate: schoenhage-strassen-Fm-bound-le-schoenhage-strassen-Fm-bound'*)  
**by** (*rule order.refl*)  
**subgoal**  
**apply** (*estimation estimate: int-lsbfermat.time-reduce-tm-le*)  
**unfolding** *int-lsbfermat.fermat-carrier-length[OF conjunct2[OF schoen-*  
*hage-strassen-correct'[OF a-carrier b-carrier]]]*  
**by** *simp*  
**done**  
**also have**  $\dots = 146 * n + 218 +$   
 $2 * (m - 1) + 2 * m + 126 * 2^{(m + 1)} + \text{schoenhage-strassen-Fm-bound}'$   
 $m$   
**by** *simp*  
**also have**  $\dots \leq 146 * n + 218 +$   
 $4 * m + 126 * 2^{(m + 1)} + \text{schoenhage-strassen-Fm-bound}' m$   
**by** *simp*  
**also have**  $\dots \leq 146 * n + 218 +$   
 $4 * m + 126 * 2^{(m + 1)} + (\gamma * \text{bitsize } m * 2^{(\text{bitsize } m + m)})$   
**apply** (*intro add-mono order.refl schoenhage-strassen-Fm-bound'-le-aux2*)  
**subgoal using** *bitsize-zero-iff[of m] iffD2[OF neq0-conv m-gt-0]* **by** *simp*  
**subgoal using** *iffD1[OF bitsize-length order.refl[of bitsize m]]*  
**by** *simp*  
**done**  
**also have**  $\dots \leq 146 * n + 218 + 4 * (\text{bitsize } n + 1) + 126 * 2^{(\text{bitsize } n +$   
 $2)} +$   
 $\gamma * \text{bitsize } (\text{bitsize } n + 1) * 2^{(\text{bitsize } (\text{bitsize } n + 1) + (\text{bitsize } n + 1))}$   
**apply** (*estimation estimate: m-le*)  
**by** (*intro bitsize-mono m-le order.refl*) **+** *simp*  
**finally show** *?thesis* **unfolding** *schoenhage-strassen-bound-def[symmetric]* .  
**qed**

**lemma** *real-diff*:  $a \leq b \implies \text{real } (b - a) = \text{real } b - \text{real } a$   
**by** *simp*

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

**proof** –

**assume**  $n > 0$   
**then have**  $\text{bitsize } n > 0$  **using** *bitsize-zero-iff[of n]* **by** *simp*  
**then have**  $\neg (\text{bitsize } n \leq \text{bitsize } n - 1)$  **by** *simp*  
**then have**  $n \geq 2^{(\text{bitsize } n - 1)}$  **using** *bitsize-length[of n bitsize n - 1]* **by**  
*simp*  
**then have**  $\log 2 (\text{real } n) \geq \text{real } (\text{bitsize } n - 1)$

**using** *le-log2-of-power* **by** *simp*  
**then show** *?thesis* **by** *simp*  
**qed**

**lemma** *powr-mono-base2*:  $a \leq b \implies 2 \text{ powr } (a :: \text{real}) \leq 2 \text{ powr } b$   
**by** (*intro powr-mono; simp*)

**lemma** *log-mono-base2*:  $a > 0 \implies b > 0 \implies a \leq b \implies \log 2 a \leq \log 2 b$   
**using** *log-le-cancel-iff[of 2 a b]* **by** *simp*

**lemma** *log-nonneg-base2*:  $x \geq 1 \implies \log 2 x \geq 0$   
**using** *zero-le-log-cancel-iff[of 2 x]* **by** *simp*

**lemma** *powr-log-cancel-base2*:  $x > 0 \implies 2 \text{ powr } (\log 2 x) = x$   
**by** (*intro powr-log-cancel; simp*)

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

**proof** –

**have**  $0: \log 2 x \geq 1$  **if**  $x \geq 2$  **for**  $x$

**using** *log-mono-base2[of 2 x]* **that** **by** *simp*

**show** *?thesis* **apply** (*intro landau-o.bigI[where c = 1]*)

**subgoal** **by** *simp*

**subgoal unfolding** *eventually-at-top-linorder* **using**  $0$  **by** *fastforce*

**done**

**qed**

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

**proof** –

**have**  $\log 2 4 = 2$

**by** (*metis log2-of-power-eq mult-2 numeral-Bit0 of-nat-numeral power2-eq-square*)

**then have**  $0: \log 2 x \geq 2$  **if**  $x \geq 4$  **for**  $x$

**using** *log-mono-base2[of 4 x]* **that** **by** *simp*

**have**  $1: \log 2 (\log 2 x) \geq 1$  **if**  $x \geq 4$  **for**  $x$

**using** *log-mono-base2[of 2 log 2 x]* **that** *0[OF that]* **by** *simp*

**show** *?thesis* **apply** (*intro landau-o.bigI[where c = 1]*)

**subgoal** **by** *simp*

**subgoal unfolding** *eventually-at-top-linorder* **using**  $1$  **by** *fastforce*

**done**

**qed**

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

**proof** –

**define** *explicit-bound* **where** *explicit-bound* =  $(\lambda x. 1154 * x + 226 + 4 * \log 2 x + (\text{real } \gamma * 24) * x * \log 2 x * \log 2 (\log 2 x) + (\text{real } \gamma * 24 * (1 + \log 2 3)) * x * \log 2 x)$

**have** *le*:  $\text{real } (\text{schoenhage-strassen-bound } n) \leq \text{explicit-bound } (\text{real } n)$  **if**  $n \geq 2$

```

for n
proof -
  have (2::nat) > 0 by simp
  from that have n ≥ 1 n > 0 by simp-all

  have 0: bitsize n + 1 > 0 by simp
  define x where x = real n
  then have x ≥ 2 x ≥ 1 x > 0 using ⟨n ≥ 2⟩ ⟨n ≥ 1⟩ ⟨n > 0⟩ by simp-all

  have log-ge: log 2 x ≥ 1 using log-mono-base2[of 2 x] using ⟨x ≥ 2⟩ by simp
  then have log-log-ge: log 2 (log 2 x) ≥ 0 and log 2 x > 0 by simp-all

  have log-n: real (bitsize n) ≤ log 2 x + 1
    unfolding x-def by (rule bitsize-le-log[OF ⟨n > 0⟩])

  have log-log-n: real (bitsize (bitsize n + 1)) ≤ log 2 (log 2 x) + (1 + log 2 3)
  proof -
    have real (bitsize (bitsize n + 1)) ≤ log 2 (real (bitsize n + 1)) + 1
      apply (intro bitsize-le-log) by simp
    also have ... = log 2 (real (bitsize n) + 1) + 1
      unfolding real-linear by simp
    also have ... ≤ log 2 (log 2 (real n) + 1 + 1) + 1
      apply (intro add-mono order.refl log-mono-base2 bitsize-le-log ⟨n > 0⟩)
      subgoal by simp
      subgoal using log-nonneg-base2[of real n] ⟨n ≥ 1⟩ by linarith
      done
    also have ... = log 2 (log 2 x + 2 * 1) + 1 unfolding x-def by argo
    also have ... ≤ log 2 (log 2 x + 2 * log 2 x) + 1
      apply (intro add-mono order.refl log-mono-base2 mult-mono)
      using log-ge by simp-all
    also have ... = log 2 (3 * log 2 x) + 1 by simp
    also have ... = (log 2 3 + log 2 (log 2 x)) + 1
      apply (intro arg-cong2[where f = (+)] refl log-mult-pos)
      using log-ge by simp-all
    also have ... = log 2 (log 2 x) + (1 + log 2 3) by simp
    finally show ?thesis .
  qed

  have 1: 0 ≤ log 2 (log 2 x) + (1 + log 2 3)
    using log-log-ge by simp

  have real (schoenhage-strassen-bound n) = 146 * x + 218 + 4 * (real (bitsize
n) + 1) + 126 * 2 powr (real (bitsize n) + 2) +
    real γ * real (bitsize (bitsize n + 1)) * 2 powr (real (bitsize (bitsize n + 1))
+ (real (bitsize n) + 1))
    unfolding schoenhage-strassen-bound-def real-linear real-multiplicative x-def
real-power[OF ⟨2 > 0⟩]
    by (intro-cong [cong-tag-2 (+), cong-tag-2 (*), cong-tag-2 (powr)] more: refl;
simp)

```

**also have** ...  $\leq 146 * x + 218 + 4 * ((\log 2 x + 1) + 1) + 126 * 2 \text{ powr } ((\log 2 x + 1) + 2) +$   
 $\text{ real } \gamma * (\log 2 (\log 2 x) + (1 + \log 2 3)) * 2 \text{ powr } ((\log 2 (\log 2 x) + (1 + \log 2 3)) + ((\log 2 x + 1) + 1))$   
**apply** (intro add-mono mult-mono order.refl powr-mono-base2 log-n log-log-n mult-nonneg-nonneg 1)  
**unfolding** x-def by simp-all  
**also have** ...  $= 1154 * x + (226 + 4 * \log 2 x) + \text{ real } \gamma * (\log 2 (\log 2 x) + (1 + \log 2 3)) * (24 * (\log 2 x * x))$   
**unfolding** powr-add powr-log-cancel-base2[OF ‹x > 0›] powr-log-cancel-base2[OF ‹log 2 x > 0›] by simp  
**also have** ...  $= 1154 * x + 226 + 4 * \log 2 x + (\text{ real } \gamma * 24) * x * \log 2 x * \log 2 (\log 2 x) + (\text{ real } \gamma * 24 * (1 + \log 2 3)) * x * \log 2 x$   
**unfolding** distrib-left distrib-right add.assoc[symmetric] mult.assoc[symmetric] by simp  
**also have** ... = explicit-bound x  
**unfolding** explicit-bound-def by (rule refl)  
**finally show** ?thesis **unfolding** x-def .  
**qed**

**have** le-bigo: schoenhage-strassen-bound  $\in O(\text{explicit-bound})$   
**apply** (intro landau-o.bigI[where c = 1])  
**subgoal** by simp  
**subgoal unfolding** eventually-at-top-linorder **using** le **by** fastforce  
**done**

**have** bigo: explicit-bound  $\in O(\lambda n. n * \log 2 n * \log 2 (\log 2 n))$   
**unfolding** explicit-bound-def  
**apply** (intro sum-in-bigo(1))  
**subgoal**  
**proof** –  
**have** (\*)  $1154 \in O(\lambda x. x)$  **by** simp  
**moreover** **have**  $1 \in O(\lambda x. \log 2 x)$  **by** (rule const-bigo-log)  
**moreover** **have**  $1 \in O(\lambda x. \log 2 (\log 2 x))$  **by** (rule const-bigo-log-log)  
**ultimately show** ?thesis **using** landau-o.big-mult[of 1 - - 1] **by** auto  
**qed**  
**subgoal**  
**proof** –  
**have** a:  $(\lambda x. 225) \in O(\lambda x. x :: \text{real})$  **by** simp  
**have** b:  $1 \in O(\lambda x. \log 2 x)$  **by** (rule const-bigo-log)  
**have** c:  $(\lambda x. 225) \in O(\lambda x. x * \log 2 x)$   
**using** landau-o.big-mult[OF a b] **by** simp  
**have** d:  $1 \in O(\lambda x. \log 2 (\log 2 x))$  **by** (rule const-bigo-log-log)  
**show** ?thesis **using** landau-o.big-mult[OF c d] **by** simp  
**qed**  
**subgoal**  
**proof** –  
**have** a:  $(\lambda x. 4) \in O(\lambda x. x :: \text{real})$  **by** simp  
**have** b:  $(\lambda x. 4 * \log 2 x) \in O(\lambda x. x * \log 2 x)$

```

    by (rule landau-o.big.mult-right[OF a])
  have c:  $1 \in O(\lambda x. \log 2 (\log 2 x))$  by (rule const-bigo-log-log)
  show ?thesis using landau-o.big-mult[OF b c] by simp
qed
subgoal
proof -
  have a:  $(\lambda x. \text{real } \gamma * 24 * x) \in O(\lambda x. x :: \text{real})$  by simp
  show ?thesis by (intro landau-o.big.mult-right a)
qed
subgoal
proof -
  have a:  $(\lambda x. \text{real } \gamma * 24 * (1 + \log 2 3) * x) \in O(\lambda x. x :: \text{real})$  by simp
  have b:  $(\lambda x. \text{real } \gamma * 24 * (1 + \log 2 3) * x * \log 2 x) \in O(\lambda x. x * \log 2 x)$ 
    by (intro landau-o.big.mult-right a)
  show ?thesis using landau-o.big-mult[OF b const-bigo-log-log] by simp
qed
done

show ?thesis using bigo landau-o.big-trans[OF le-bigo] by blast
qed
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](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.