Schönhage-Strassen Multiplication on Integers

Jakob Schulz

March 19, 2025

Abstract

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

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

Contents

1 Preliminaries

theory *Schoenhage-Strassen-Preliminaries*
imports
  Main
  HOL−Library.FuncSet
  Karatsuba.Karatsuba-Preliminaries
  Karatsuba.Nat-LSBF
begin

lemma *two-pow-pos*: $(2 :: nat)$ ^ $x > 0$
  by *simp*

lemma *length-take-cobounded1*: *length* (*take n xs*) $\leq n$
  by *simp*

lemma *const-diff-mod-idem*:
  assumes $m \geq (n :: nat)$
    $f = (\lambda i.\ (m - i)\ mod\ n)$
  shows $(\bigwedge i.\ i \in \{0..<n\} \Longrightarrow f\ (f\ i) = i)$
proof −
  fix i
  assume $i \in \{0..<n\}$
  then have $i < n$ **by** *simp*
  then have $i \leq m$ **using** ‹$n \leq m$› **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 ‹$n \leq m$› *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*: $m \geq (n :: nat) \Longrightarrow$ *bij-betw* $(\lambda i.\ (m - i)\ mod\ n)$ $\{0..<n\}$
$\{0..<n\}$

```
proof (intro bij-betwI)
  show (λi. (m − i) mod n) ∈ {0..<n} → {0..<n} by simp
  show (λi. (m − i) mod n) ∈ {0..<n} → {0..<n} by simp
  show ⋀x. n ≤ m ⟹ x ∈ {0..<n} ⟹ (m − (m − x) mod n) mod n = x
    using const-diff-mod-idem[of n] by blast
  show ⋀x. n ≤ m ⟹ x ∈ {0..<n} ⟹ (m − (m − x) mod n) mod n = x
    using const-diff-mod-idem[of n] by blast
qed

lemma const-add-mod-bij: bij-betw (λi. ((m :: nat) + i) mod n) {0..<n} {0..<n}
proof (intro bij-betwI)
  show (λi. (m + i) mod n) ∈ {0..<n} → {0..<n} by simp
  show (λi. (n − (m mod n) + i) mod n) ∈ {0..<n} → {0..<n} by simp
  show ⋀x. x ∈ {0..<n} ⟹ (n − m mod n + (m + x) mod n) mod n = x
  proof −
    fix x
    assume x ∈ {0..<n}
    then have x < n by simp
    have int ((n − m mod n + (m + x) mod n) mod n) = (int n − int m mod int n + (int m + int x) mod int n) mod int n
      using ‹x < n› zmod-int
      by (metis less-nat-zero-code mod-le-divisor not-gr-zero of-nat-add of-nat-diff)
    also have ... = (int n + (int x) mod int n) mod 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 ... = int x using ‹x < n› mod-add-self1 by simp
    finally show (n − m mod n + (m + x) mod n) mod n = x by linarith
  qed
  show ⋀y. y ∈ {0..<n} ⟹ (m + (n − m mod n + y) mod n) mod n = y
  proof −
    fix y
    assume y ∈ {0..<n}
    then have y < n by simp
    then show (m + (n − m mod n + y) mod n) mod n = y
      by (metis ‹⋀x. x ∈ {0..<n} ⟹ (n − m mod n + (m + x) mod n) mod n = x› ‹y ∈ {0..<n}› add.left-commute mod-add-right-eq)
  qed
qed

lemma mod-diff-add-eq: (a − b + c) mod (m :: int) = (a mod m − b mod m + c mod m) mod m
proof −
  have (a − b + c) mod m = ((a − b) mod m + c mod m) mod m
    by (intro mod-add-eq[symmetric])
  also have ... = ((a mod m − b mod m) mod m + c mod m) mod m
    by (simp only: mod-diff-eq)
  also have ... = (a mod m − b mod m + c mod m) mod m
    by (simp only: mod-add-left-eq)
  finally show (a − b + c) mod m = (a mod m − b mod m + c mod m) mod m .
```

qed

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

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

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

lemma *set-map2-subseteqI*:
  assumes $\bigwedge x\ y.\ x \in A \Longrightarrow y \in B \Longrightarrow f\ x\ y \in C$
  assumes *set xs* $\subseteq A$ *set ys* $\subseteq B$
  shows *set* (*map2 f xs ys*) $\subseteq C$
proof
  fix *z*
  assume $z \in$ *set* (*map2 f xs ys*)
  then obtain *x y* **where** $z = f\ x\ y\ x \in$ *set xs* $y \in$ *set ys*
    using *set-map2* **by** *meson*
  then show $z \in C$ **using** *assms* **by** *auto*
qed

lemma *in-set-conv-nth-map3*:
  assumes $w \in$ *set* (*map3 f xs ys zs*)
  shows $\exists i.\ i <$ *min* (*min* (*length xs*) (*length ys*)) (*length zs*) $\wedge w = f$ (*xs* ! *i*) (*ys* ! *i*) (*zs* ! *i*)
proof −
  from *assms* **obtain** *i* **where** $i <$ *length* (*map3 f xs ys zs*) $w =$ *map3 f xs ys zs* ! *i*
    by (*metis in-set-conv-nth*)
  have $i <$ *min* (*min* (*length xs*) (*length ys*)) (*length zs*)
    using ‹$i <$ *length* (*map3 f xs ys zs*)›
    unfolding *map3-as-map* **by** *simp*

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

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

lemma *set-map3-subseteqI*:
assumes $\bigwedge x\ y\ z.\ x \in A \Longrightarrow y \in B \Longrightarrow z \in C \Longrightarrow f\ x\ y\ z \in D$
assumes $set\ xs \subseteq A\ set\ ys \subseteq B\ set\ zs \subseteq C$
shows $set\ (map3\ f\ xs\ ys\ zs) \subseteq D$
proof
fix w
assume $w \in set\ (map3\ f\ xs\ ys\ zs)$
then obtain $x\ y\ z$ **where** $w = f\ x\ y\ z\ x \in set\ xs\ y \in set\ ys\ z \in set\ zs$
using *set-map3* **by** *meson*
then show $w \in D$ **using** *assms* **by** *fastforce*
qed

lemma *map3-compose3*: $map3\ (\lambda x\ y\ z.\ f\ x\ y\ (g\ z))\ xs\ ys\ zs = map3\ f\ xs\ ys\ (map\ g\ zs)$
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* :: $nat \Rightarrow 'a\ list \Rightarrow 'a\ list$ **where**
$rotate\text{-}left\ k\ xs = (let\ (xs1,\ xs2) = split\text{-}at\ (k\ mod\ length\ xs)\ xs\ in\ xs2\ @\ xs1)$

lemma *rotate-left-rotate*[*simp*]: $rotate\text{-}left\ k\ xs = rotate\ k\ xs$
unfolding *rotate-left-def* **by** (*simp add*: *rotate-drop-take*)

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

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

lemma *rotate-right-rotate*[*simp*]: $rotate\text{-}right\ k\ (rotate\ k\ xs) = xs$
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* < *length xs* ⟹ *m* < *length xs* ⟹ *rotate-right m xs* ! *n* = *xs* ! ((*n* + *length xs* − *m*) *mod 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 ⇒ 'b list ⇒ ('a × 'b) list tm where
zip-tm xs [] =1 return []
| zip-tm [] ys =1 return []
| zip-tm (x # xs) (y # ys) =1 do { rs ← 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 ⇒ 'b ⇒ 'c ⇒ 'd tm) ⇒ 'a list ⇒ 'b list ⇒ 'c list ⇒ 'd list tm
where
map3-tm f (x # xs) (y # ys) (z # zs) =1 do {
  r ← f x y z;
  rs ← 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 (λx 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 ⋀x y z. x ∈ set xs ⟹ y ∈ set ys ⟹ z ∈ set zs ⟹ time (f x y z) ≤ c
  shows time (map3-tm f xs ys zs) ≤ (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) ≤ (c + 1) ∗ min (min (length xs)
(length ys)) (length zs) + 1
    by simp
  from 1.prems have fxyz: time (f x y z) ≤ 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.prems 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 length xs = length ys
  assumes ⋀x y. x ∈ set xs ⟹ y ∈ set ys ⟹ time (f x y) ≤ c
```

```
  shows time (map2-tm f xs ys) ≤ (c + 2) ∗ length xs + 3
proof −
  have time (map2-tm f xs ys) = length xs + 2 + time (map-tm (λ(x, y). f x y)
(zip xs ys))
    unfolding map2-tm-def by (simp add: assms)
  also have ... ≤ length xs + 2 + ((c + 1) ∗ length (zip xs ys) + 1)
    apply (intro add-mono order.refl time-map-tm-bounded)
    using assms by (auto split: prod.splits elim: in-set-zipE)
  also have ... = (c + 2) ∗ length xs + 3
    using assms by simp
  finally show ?thesis .
qed

definition rotate-left-tm :: nat ⇒ 'a list ⇒ 'a list tm where
rotate-left-tm k xs =1 do {
  lenxs ← length-tm xs;
  kmod ← k mod_t lenxs;
  (xs1, xs2) ← split-at-tm kmod xs;
  xs2 @_t xs1
}

lemma val-rotate-left-tm[simp, val-simp]: val (rotate-left-tm k xs) = rotate-left k
xs
  unfolding rotate-left-tm-def rotate-left-def by (simp add: Let-def)

lemma time-rotate-left-tm-le: time (rotate-left-tm k xs) ≤ 13 + 14 ∗ max k (length
xs)
proof −
  obtain xs1 xs2 where 1: (xs1, xs2) = split-at (k mod length xs) xs
    by simp
  then have 2: length xs2 ≤ length xs by simp
  have time (rotate-left-tm k xs) =
    time (length-tm xs) +
    time (k mod_t (length xs)) +
    time (split-at-tm (k mod length xs) xs) + time (xs2 @_t xs1) + 1
 unfolding rotate-left-tm-def tm-time-simps val-length-tm val-mod-nat-tm val-split-at-tm
 Product-Type.prod.case 1[symmetric] by simp
  also have ... ≤ (length xs + 1) + (8 ∗ k + 2 ∗ length xs + 7) + (2 ∗ length xs
+ 3) + (length xs + 1) + 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 add: time-split-at-tm)
    subgoal by (simp add: 2)
    done
  also have ... = 13 + 6 ∗ length xs + 8 ∗ k by simp
  finally show ?thesis by simp
qed
```

definition *rotate-right-tm* :: *nat* ⇒ *'a list* ⇒ *'a list tm* **where**
rotate-right-tm k xs =1 *do* {
  lenxs ← *length-tm xs*;
  kmod ← *k* mod_t *lenxs*;
  rk ← *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*) ≤ *23* + *26* ∗ *max k* (*length xs*)
proof −
  have *time* (*rotate-right-tm k xs*) =
    time (*length-tm xs*) +
    time (*k* mod_t *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 ... ≤ (*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* ⇒ *nat*
  assumes *f* ∈ *O*(*g*)
  assumes ⋀*x*. *x* ≥ *n0* ⟹ *g x* > *0*
  shows ∃ *c*. (∀ *x*. *x* ≥ *n0* ⟶ *f x* ≤ *c* ∗ *g x*)
proof −
  from *landau-o.bigE*[*OF* ‹*f* ∈ *O*(*g*)›]
  obtain *c-real* **where** *eventually* (λ*x*. *norm* (*f x*) ≤ *c-real* ∗ *norm* (*g x*)) *sequentially*
    by *auto*
  then have *eventually* (λ*x*. *f x* ≤ *c-real* ∗ *g x*) *at-top* **by** *simp*

```
  then obtain n1 where f-le-g-real: f x ≤ c-real ∗ g x if x ≥ n1 for x
    using eventually-at-top-linorder by meson
  define c where c = nat (ceiling c-real)
  then have f-le-g: f x ≤ c ∗ g x if x ≥ n1 for x
  proof −
    have real (f x) ≤ c-real ∗ real (g x) using f-le-g-real[OF that] .
    also have ... ≤ real c ∗ real (g x) unfolding c-def
      by (simp add: mult-mono real-nat-ceiling-ge)
    also have ... = real (c ∗ g x) by simp
    finally show ?thesis by linarith
  qed
  consider n1 ≤ n0 | 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 = Max (f ‘ {n0..<n1})
    define C where C = (max M 1) ∗ (max c 1)
    have f x ≤ C ∗ g x if x ≥ n0 for x
    proof (cases x < n1)
      case True
      then have f x ≤ M
        unfolding M-def using 2
        by (intro Max.coboundedI; simp add: that)
      also have ... ≤ C unfolding C-def
        using nat-mult-max-right by auto
      also have ... ≤ C ∗ g x
        using assms(2)[OF that] by simp
      finally show ?thesis .
    next
      case False
      then have f x ≤ c ∗ g x using f-le-g by simp
      also have ... ≤ 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 :: nat ⇒ real
  assumes f ∈ O(g)
  assumes ⋀x. x ≥ n0 ⟹ f x ≥ 0 ∧ g x ≥ 1
  shows ∃ c. (∀ x ≥ n0. f x ≤ c ∗ g x)
proof −
```

```
  from landau-o.bigE[OF ‹f ∈ O(g)›]
  obtain c where eventually (λx. norm (f x) ≤ c ∗ norm (g x)) at-top
    by auto
  then obtain n1 where f-le-g: norm (f x) ≤ c ∗ norm (g x) if x ≥ n1 for x
    using eventually-at-top-linorder by meson
  consider n1 ≤ n0 | 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 ≥ 1 using mult-mono[OF max.cobounded2[of 1 M] max.cobounded2[of
1 c]] by argo
    have C ≥ c unfolding C-def using mult-mono[OF max.cobounded2[of 1 M]
max.cobounded1[of c 1]]
      by linarith
    have f x ≤ C ∗ g x if x ≥ n0 for x
    proof (cases x < n1)
      case True
      then have f x ≤ M
        unfolding M-def using 2
        by (intro Max.coboundedI; simp add: that)
      also have ... ≤ C unfolding C-def
        using mult-mono[OF max.cobounded1[of M 1] max.cobounded2[of 1 c]] by
simp
      also have ... ≤ C ∗ g x
        using assms(2)[OF that] mult-left-mono[of 1 g x C] ‹C ≥ 1› by argo
      finally show ?thesis .
    next
      case False
      then have f x ≤ c ∗ g x using f-le-g[of x] assms(2)[OF that] by simp
      also have ... ≤ C ∗ g x apply (intro mult-mono[OF ‹C ≥ c›])
        subgoal by (rule order.refl)
        subgoal using ‹C ≥ 1› 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: floor x ∈ ℕ ⟷ x ≥ 0
```

```
proof
  assume floor x ∈ ℕ
  then obtain n where floor x = of-nat n unfolding Nats-def by auto
  then have floor x ≥ 0 using of-nat-0-le-iff by simp
  then show x ≥ 0 by simp
next
  assume 0 ≤ x
  then have floor x ≥ 0 by simp
  then obtain n where floor x = of-nat n using nat-0-le by metis
  then show floor x ∈ ℕ unfolding Nats-def by simp
qed

lemma bigo-floor:
  fixes f :: nat ⇒ nat
  fixes g :: nat ⇒ real
  assumes (λx. real (f x)) ∈ O(g)
  assumes eventually (λx. g x ≥ 1) at-top
  shows (λx. real (f x)) ∈ O(λx. real (nat (floor (g x))))
proof −
  have ineq: x ≤ 2 ∗ real-of-int (floor x) if x ≥ 1 for x :: real
  proof −
    have x ≤ real-of-int (floor x) + 1
      by (rule real-of-int-floor-add-one-ge)
    also have ... ≤ 2 ∗ real-of-int (floor x)
      using that by simp
    finally show ?thesis .
  qed
  obtain c where c > 0 and f-le-g: eventually (λx. real (f x) ≤ c ∗ norm (g x))
at-top
    using landau-o.bigE[OF assms(1)] by auto
  have eventually (λx. g x ≤ 2 ∗ of-int (floor (g x))) at-top
    using eventually-rev-mp[OF assms(2), of λx. g x ≤ 2 ∗ of-int (floor (g x))]
    using assms(2) ineq by simp
  then have 1: eventually (λx. c ∗ g x ≤ (2 ∗ c) ∗ of-int (floor (g x))) at-top
    using eventually-mp[of λx. g x ≤ 2 ∗ of-int (floor (g x)) λx. c ∗ g x ≤ (2 ∗
c) ∗ of-int (floor (g x))]
    using ‹c > 0› by simp
  have 2: eventually (λx. c ∗ norm (g x) = c ∗ g x) at-top
    using eventually-rev-mp[OF assms(2)] by simp
  have 3: eventually (λx. c ∗ norm (g x) ≤ (2 ∗ c) ∗ of-int (floor (g x))) at-top
    apply (intro eventually-rev-mp[OF eventually-conj[OF 1 2], of λx. c ∗ norm
(g x) ≤ (2 ∗ c) ∗ of-int (floor (g x))])
    apply (intro always-eventually allI impI)
    by argo
  have 4: eventually (λx. real (f x) ≤ (2 ∗ c) ∗ of-int (floor (g x))) at-top
    apply (intro eventually-rev-mp[OF eventually-conj[OF f-le-g 3], where Q =
λx. real (f x) ≤ (2 ∗ c) ∗ of-int (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-rev-mp*[*OF eventually-conj*[*OF 4 assms*(2)], **where** $Q = \lambda x.\ norm\ (real\ (f\ x)) \le (2 * c) * norm\ (real\ (nat\ \lfloor g\ x \rfloor))$]])
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 $a \in carrier\ R$ $b \in carrier\ R$ $c \in carrier\ R$
shows $a \ominus (b \ominus c) = a \ominus b \oplus c$
using *assms* **by** *algebra*
lemma *minus-eq-mult-one*:
assumes $a \in carrier\ R$
shows $\ominus\ a = (\ominus\ \mathbf{1}) \otimes a$
using *assms* **by** *algebra*
lemma *diff-eq-add-mult-one*:
assumes $a \in carrier\ R$ $b \in carrier\ R$
shows $a \ominus b = a \oplus (\ominus\ \mathbf{1}) \otimes b$
using *assms* **by** *algebra*
lemma *minus-cancel*:
assumes $a \in carrier\ R$ $b \in carrier\ R$
shows $a \ominus b \oplus b = a$
using *assms* **by** *algebra*
lemma *assoc4*:
assumes $a \in carrier\ R$ $b \in carrier\ R$ $c \in carrier\ R$ $d \in carrier\ R$
shows $a \otimes (b \otimes (c \otimes d)) = a \otimes b \otimes c \otimes d$
using *assms* **by** *algebra*
lemma *diff-sum*:
assumes $a \in carrier\ R$ $b \in carrier\ R$ $c \in carrier\ R$ $d \in carrier\ R$
shows $(a \ominus c) \oplus (b \ominus d) = (a \oplus b) \ominus (c \oplus d)$
using *assms* **by** *algebra*

end

lemma (**in** *ring*) *inv-cancel-left*:
assumes $x \in carrier\ R$
assumes $y \in carrier\ R$
assumes $z \in Units\ R$
assumes $x = z \otimes y$
shows $inv\ z \otimes x = y$

using *assms*
by (*metis Units-closed Units-inv-closed Units-l-inv l-one m-assoc*)

lemma (**in** *ring*) *r-distr-diff*:
assumes $x \in$ *carrier R*
assumes $y \in$ *carrier R*
assumes $z \in$ *carrier R*
shows $x \otimes (y \ominus z) = x \otimes y \ominus x \otimes z$
using *assms* **by** *algebra*

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

1.3 Multiplicative Subgroups

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

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

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

lemma *int-pow-eq*:

```
  assumes x ∈ X
  shows x [^]_M (i :: int) = x [^] i
proof (cases i ≥ 0)
  case True
  then have x [^]_M i = x [^]_M (nat i)
    by simp
  also have ... = x [^] (nat i)
    using nat-pow-eq by simp
  also have ... = x [^] i
    using True by simp
  finally show ?thesis .
next
  case False
  then have x [^]_M i = inv_M (x [^]_M (nat (− i)))
    using int-pow-def2[of M] by presburger
  also have ... = inv (x [^]_M (nat (− i)))
    apply (intro inv-eq)
   using monoid.nat-pow-closed[OF group.is-monoid[OF M-group]] assms by simp
  also have ... = inv (x [^] (nat (− i)))
    by (simp add: nat-pow-eq)
  also have ... = x [^] 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 ∈ Units R
  shows inv (a [^] (b :: nat)) = inv a [^] b
proof −
  have inv (a [^] b) = inv_units-of R (a [^]_units-of R b)
     using assms units-subgroup.nat-pow-eq units-subgroup.inv-eq Units-pow-closed
by simp
  also have ... = inv_units-of R a [^]_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 $x \in$ *Units R*
shows *x* [^] *m1* $\otimes$ *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 $x \in$ *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 $x \in$ *Units R*
shows *x* [^] (*i* :: *int*) $\in$ *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*: $\mu \in$ *Units R* $\Longrightarrow$ μ [^]$_{(\textit{units-of } R)}$ *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*: $\mu \in$ *Units R* $\Longrightarrow$ (*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*: $\mu \in$ *Units R* $\Longrightarrow$ *inv* μ = μ [^] ($-$ (*1* :: *int*))
using *units-int-pow-neg*[*of* μ *1* :: *int*]
by (*simp add*: *int-pow-def2*)

lemma *inv-prod*: $\mu \in$ *Units R* $\Longrightarrow$ $\nu \in$ *Units R* $\Longrightarrow$ *inv* ($\mu \otimes \nu$) = *inv* $\nu \otimes$ *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* :: *nat*

assumes $x \in$ *carrier R*
shows *even r* $\Longrightarrow (\ominus\ x)\ [\urcorner]\ r = x\ [\urcorner]\ r$ *odd r* $\Longrightarrow (\ominus\ x)\ [\urcorner]\ r = \ominus\ (x\ [\urcorner]\ r)$
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*: $X \subseteq$ *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 $inv_M\ a = \ominus\ a$
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 $a \neq 0$
  assumes $k = Max \{s.\ p \mathbin{\char`\^} s\ dvd\ a\}$
  assumes $r = a\ div\ (p \mathbin{\char`\^} k)$
  assumes *prime p*
  shows $a = r * p \mathbin{\char`\^} k$ *coprime p r*
  subgoal
  proof −
    have $p \mathbin{\char`\^} 0\ dvd\ a$ **by** *simp*
    then have $\{s.\ p \mathbin{\char`\^} s\ dvd\ a\} \neq \{\}$ **by** *blast*
    with *assms* **have** $p \mathbin{\char`\^} k\ dvd\ a$
      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 $p \mathbin{\char`\^} (k + 1)\ dvd\ a$ **using** ‹$a = r * p \mathbin{\char`\^} k$› **by** *simp*
    then have $k \geq k + 1$
      using *assms Max-ge*[*of* $\{s.\ p \mathbin{\char`\^} s\ dvd\ a\}$ *k*] *Max-in*[*of* $\{s.\ p \mathbin{\char`\^} s\ dvd\ a\}$]
    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 $\mu \equiv \mu \in$ *carrier R* $\wedge\ \mu\ [\widehat{\ }]\ n = \mathbf{1}$

lemma *root-of-unityI*[*intro*]: $\mu \in$ *carrier R* $\Longrightarrow \mu\ [\widehat{\ }]\ n = \mathbf{1} \Longrightarrow$ *root-of-unity n* μ
  unfolding *root-of-unity-def* **by** *simp*

lemma *root-of-unityD*[*simp*]: *root-of-unity n* $\mu \Longrightarrow \mu\ [\widehat{\ }]\ n = \mathbf{1}$
  unfolding *root-of-unity-def* **by** *simp*

lemma *root-of-unity-closed*[*simp*]: *root-of-unity n* $\mu \Longrightarrow \mu \in$ *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* μ
  shows $\mu \in$ *Units R*
proof −
  from ‹$n > 0$› **obtain** n' **where** $n = Suc\ n'$
    using *gr0-implies-Suc* **by** *auto*
  then have $\mathbf{1} = \mu \otimes (\mu\ [\widehat{\ }]\ n')$
    using *assms nat-pow-Suc2* **unfolding** *root-of-unity-def* **by** *auto*
  then show $\mu \in$ *Units R* **using** *assms m-comm*[*of* μ $\mu\ [\widehat{\ }]\ n'$] *nat-pow-closed*[*of* μ n']
    unfolding *Units-def root-of-unity-def* **by** *auto*
qed

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

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 ‹$n > 0$›
    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* $\{\mu.$ *root-of-unity n* $\mu\}$ *roots-of-unity-group*
  apply *unfold-locales*
  subgoal using *roots-Units* ‹$n > 0$› **by** *blast*
  subgoal unfolding *roots-of-unity-group-def* **by** *simp*
  done

lemma *root-of-unity-inv*:
  assumes *root-of-unity n* μ

shows *root-of-unity n* (*inv μ*)
using *assms root-group.inv-closed*[*of μ*] *root-subgroup.carrier-M root-subgroup.inv-eq*[*of μ*] **by** *simp*

lemma *inv-root-of-unity*:
assumes *root-of-unity n μ*
shows *inv μ* = *μ* [^] (*n* − *1*)
proof −
have *μ* ∈ *Units R* **using** *assms*
using *roots-Units* **by** *blast*
then have *inv μ* = *μ* [^] (−*1* :: *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 ... = **1** ⊗ *μ* [^] (−*1* :: *int*)
apply (*intro l-one*[*symmetric*])
using ‹*μ* ∈ *Units R*› **by** (*metis Units-inv-closed calculation*)
also have ... = *μ* [^] *n* ⊗ *μ* [^] (−*1* :: *int*)
using *assms* **by** *simp*
also have ... = *μ* [^] (*int n*) ⊗ *μ* [^] (−*1* :: *int*)
using *Units-closed*[*OF* ‹*μ* ∈ *Units R*›]
by (*simp add*: *int-pow-int*)
also have ... = *μ* [^] (*int n* − *1*)
using *units-group.int-pow-mult*[*of μ*] ‹*μ* ∈ *Units R*› *units-subgroup.int-pow-eq*[*of μ*]
using *units-of-mult units-subgroup.carrier-M*
by (*metis add.commute uminus-add-conv-diff*)
also have ... = *μ* [^] (*n* − *1*)
using ‹*n* > *0*› *Units-closed*[*OF* ‹*μ* ∈ *Units R*›]
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 μ*
assumes *i* ∈ {*1*..<*n*}
shows (*inv μ*) [^] *i* = *μ* [^] (*n* − *i*) *n* − *i* ∈ {*1*..<*n*}
proof −
have (*inv μ*) [^] *i* = (*μ* [^] (*n* − (*1*::*nat*))) [^] *i*
using *assms inv-root-of-unity* **by** *algebra*
also have ... = *μ* [^] ((*n* − *1*) ∗ *i*)
apply (*intro nat-pow-pow*) **using** *assms roots-Units Units-closed* **by** *blast*
also have ... = *μ* [^] *n* ⊗ *μ* [^] ((*n* − *1*) ∗ *i*)
using *assms root-of-unity-def*[*of n μ*] **by** *fastforce*
also have ... = *μ* [^] (*n* + (*n* − *1*) ∗ *i*)
apply (*intro nat-pow-mult*) **using** *assms roots-Units Units-closed* **by** *blast*
also have ... = *μ* [^] (*n* ∗ *i* + (*n* − *i*))
proof (*intro arg-cong*[**where** *f* = ([^]) *μ*])
have *int* (*n* + (*n* − *1*) ∗ *i*) = *int* (*n* ∗ *i* + (*n* − *i*))

```
    proof −
      have int (n + (n − 1) ∗ i) = int n + int (n − 1) ∗ int i
        by simp
      also have ... = int n + (int n − int 1) ∗ int i
        using ‹n > 0› by fastforce
      also have ... = int n + int n ∗ int i − int i
        by (simp add: left-diff-distrib′)
      also have ... = int n ∗ int i + (int n − int i)
        by simp
      also have ... = int (n ∗ i) + 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 ... = (μ [^] n) [^] i ⊗ μ [^] (n − i)
    using nat-pow-mult nat-pow-pow
    using assms roots-Units Units-closed by algebra
  also have ... = μ [^] (n − i)
    using assms unfolding root-of-unity-def by simp
  finally show (inv μ) [^] i = μ [^] (n − i) by blast
  show n − i ∈ {1..<n} using assms by auto
qed

lemma root-of-unity-nat-pow-closed:
  assumes root-of-unity n μ
  shows root-of-unity n (μ [^] (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 μ
  shows μ [^] i = μ [^] (i mod n)
proof −
  have[simp]: μ ∈ carrier R using assms by simp
  define s t where s = i div n t = i mod n
  then have i = s ∗ n + t t < n using ‹n > 0› by simp-all
  then have μ [^] i = μ [^] (s ∗ n) ⊗ μ [^] t by (simp add: nat-pow-mult)
  also have μ [^] (s ∗ n) = (μ [^] n) [^] s by (simp add: nat-pow-pow mult.commute)
  also have ... = 1 using assms by simp
  finally show ?thesis using ‹t = i mod n› by simp
qed

lemma root-of-unity-powers-modint:
  assumes root-of-unity n μ
  shows μ [^] (i :: int) = μ [^] (i mod int n)
proof −
  have μ ∈ Units R μ [^] n = 1 using assms by simp-all
  define s t where s = i div int n t = i mod int n
  then have i = s ∗ int n + t t ≥ 0 t < int n using ‹n > 0› by simp-all
```

then have μ [^] i = μ [^] (s * *int* n) ⊗ μ [^] t
using *int-pow-mult*[*OF* ‹μ ∈ *Units R*›] **by** *simp*
also have ... = (μ [^] *int* n) [^] s ⊗ μ [^] t
by (*intro-cong* [*cong-tag-2* (⊗)] *more*: *refl*) (*simp add*: *int-pow-pow* ‹μ ∈ *Units R*› *mult.commute*)
also have ... = (μ [^] n) [^] s ⊗ μ [^] t
apply (*intro-cong* [*cong-tag-2* (⊗), *cong-tag-1* (λi. i [^] s)] *more*: *refl*)
using ‹$n > 0$› **by** (*simp add*: *int-pow-int*)
also have ... = μ [^] t
using *int-pow-closed*[*OF* ‹μ ∈ *Units R*›] *Units-closed l-one*
by (*simp add*: ‹μ [^] n = **1**› *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 μ
assumes i *mod* n = j *mod* n
shows μ [^] i = μ [^] j
using *assms root-of-unity-powers* **by** *metis*

lemma *root-of-unity-powers-int*:
assumes *root-of-unity* n μ
assumes i *mod int* n = j *mod int* n
shows μ [^] i = μ [^] j
using *assms root-of-unity-powers-modint* **by** *metis*

end

2.2 Primitive Roots

definition *primitive-root* :: *nat* ⇒ *'a* ⇒ *bool* **where**
primitive-root n μ ≡ *root-of-unity* n μ ∧ (∀ i ∈ {*1*..<n}. μ [^] $i \neq$ **1**)

lemma *primitive-rootI*[*intro*]:
assumes μ ∈ *carrier R*
assumes μ [^] n = **1**
assumes $\bigwedge i.\ i > 0 \Longrightarrow i < n \Longrightarrow$ μ [^] $i \neq$ **1**
shows *primitive-root* n μ
unfolding *primitive-root-def root-of-unity-def* **using** *assms* **by** *simp*

lemma *primitive-root-is-root-of-unity*[*simp*]: *primitive-root* n μ ⟹ *root-of-unity* n μ
unfolding *primitive-root-def* **by** *simp*

lemma *primitive-root-recursion*:
assumes *even* n
assumes *primitive-root* n μ
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 $\mu \in$ *carrier R*
  shows *NTT* μ *a* ! *i* = ($\bigoplus j \leftarrow [0..<n]$. (*a* ! *j*) $\otimes$ (μ [^] (*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* $\subseteq$ *carrier R*
  assumes $\mu \in$ *carrier R*
  assumes *length a* = *n*
  assumes *i* < *n*
  shows *NTT* μ *a* ! *i* $\in$ *carrier R*
proof −
  have *NTT* μ *a* ! *i* = ($\bigoplus j \leftarrow [0..<length\ a]$. (*a* ! *j*) $\otimes$ (μ [^] *i*) [^] *j*)
    using *NTT-nth assms* **by** *blast*
  also have ... $\in$ *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* $\subseteq$ *carrier R*
  assumes $\mu \in$ *carrier R*
  shows *set* (*NTT* μ *a*) $\subseteq$ *carrier R*
  using *assms NTT-nth-closed*[*of a* μ]
  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**) [^] (*2*::*nat*) = **1**
  by (*simp add*: *numeral-2-eq-2*) *algebra*
lemma **1** $\oplus$ **1** $\neq$ **0** $\Longrightarrow$ *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 ¬ $\ominus$ **1** [^] *i* $\neq$ **1**
    then have $\ominus$ **1** = **1** **using** ‹*i* = *1*› **by** *simp*
    then have **1** $\oplus$ **1** = **0** **using** *l-neg* **by** *fastforce*

thus *False* **using** ‹$\mathbf{1} \oplus \mathbf{1} \neq \mathbf{0}$› **by** *simp*
qed
done

2.3.1 Inversion Rule

theorem *inversion-rule*:
fixes μ :: *'a*
fixes *n* :: *nat*
assumes $n > 0$
assumes *primitive-root n* μ
assumes *good*: $\bigwedge i.\ i \in \{1..<n\} \Longrightarrow (\bigoplus j \leftarrow [0..<n].\ (\mu\ [\widehat{\ }]\ i)\ [\widehat{\ }]\ j) = \mathbf{0}$
assumes[*simp*]: *length a = n*
assumes[*simp*]: *set a* $\subseteq$ *carrier R*
shows *NTT* (*inv* μ) (*NTT* μ *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* μ) (*NTT* μ *a*)) = *length* (*map* (($\otimes$) (*nat-embedding n*)) *a*) **using** *NTT-length*
by *simp*
fix *i*
assume *i* < *length* (*NTT* (*inv* μ) (*NTT* μ *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* μ) $[\widehat{\ }]$ *i* $\in$ *carrier R* **by** *simp*

have *0*: *NTT* (*inv* μ) (*NTT* μ *a*) ! *i* = ($\bigoplus j \leftarrow [0..<n]$. (*NTT* μ *a* ! *j*) $\otimes$ ((*inv* μ) $[\widehat{\ }]$ *i*) $[\widehat{\ }]$ *j*)
using *NTT-nth*
using *assms NTT-length* ‹*i* < *n*› **by** *blast*
also have ... = ($\bigoplus j \leftarrow [0..<n]$. ($\bigoplus k \leftarrow [0..<n]$. *a* ! *k* $\otimes$ μ $[\widehat{\ }]$ ((*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* μ *a* ! *j*) = ($\bigoplus k \leftarrow [0..<n]$. *a* ! *k* $\otimes$ (μ $[\widehat{\ }]$ *j*) $[\widehat{\ }]$ *k*)
using *NTT-nth* **by** *simp*
have ... $\otimes$ ((*inv* μ) $[\widehat{\ }]$ *i*) $[\widehat{\ }]$ *j* = ($\bigoplus k \leftarrow [0..<n]$. *a* ! *k* $\otimes$ ((μ $[\widehat{\ }]$ *j*) $[\widehat{\ }]$ *k*) $\otimes$ ((*inv* μ) $[\widehat{\ }]$ *i*) $[\widehat{\ }]$ *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 ... = ($\bigoplus k \leftarrow [0..<n]$. *a* ! *k* $\otimes$ μ $[\widehat{\ }]$ ((*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 (⨁*j* ← [*0*..<*n*]. *μ* [^] ((*int k* − *int i*) ∗ *int j*)) = (*if i* = *k then nat-embedding n else* **0**)

proof (*cases*)
  case *1*
  then have $\bigwedge j.\ j < n \Longrightarrow \mu\ [\urcorner]\ ((int\ k - int\ i) * int\ j) = (\mu\ [\urcorner]\ (k - i))\ [\urcorner]\ j$
  proof –
    fix *j*
    assume $j < n$
    have $(int\ k - int\ i) * int\ j = int\ ((k - i) * j)$ **using** *1* **by** *auto*
    then have $\mu\ [\urcorner]\ ((int\ k - int\ i) * int\ j) = \mu\ [\urcorner]\ int\ ((k - i) * j)$
      by *argo*
    also have ... $= \mu\ [\urcorner]\ ((k - i) * j)$
      by (*intro int-pow-int*)
    also have ... $= (\mu\ [\urcorner]\ (k - i))\ [\urcorner]\ j$
      by (*intro nat-pow-pow*[*symmetric*] ‹$\mu \in$ *carrier R*›)
    finally show $\mu\ [\urcorner]\ ((int\ k - int\ i) * int\ j) = (\mu\ [\urcorner]\ (k - i))\ [\urcorner]\ j$ **.**
  qed
  then have $(\bigoplus j \leftarrow [0..<n].\ \mu\ [\urcorner]\ ((int\ k - int\ i) * int\ j)) = (\bigoplus j \leftarrow [0..<n].$
$(\mu\ [\urcorner]\ (k - i))\ [\urcorner]\ j)$
    by (*intro monoid-sum-list-cong*, *simp*)
  also have ... $= \mathbf{0}$
    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 $\bigwedge j.\ j < n \Longrightarrow \mu\ [\urcorner]\ ((int\ k - int\ i) * int\ j) = \mathbf{1}$ **by** *simp*
  then have $(\bigoplus j \leftarrow [0..<n].\ \mu\ [\urcorner]\ ((int\ k - int\ i) * int\ j)) =$ *nat-embedding n*
    using *monoid-sum-list-const*[*of* $\mathbf{1}$ $[0..<n]$]
    using *monoid-sum-list-cong*[*of* $[0..<n]$ $\lambda j.\ \mu\ [\urcorner]\ ((int\ k - int\ i) * int\ j)$ $\lambda j.$
$\mathbf{1}$]
    by *simp*
  then show *?thesis* **using** *2* **by** *simp*
next
  case *3*
  then have $\bigwedge j.\ j < n \Longrightarrow \mu\ [\urcorner]\ ((int\ k - int\ i) * int\ j) = (\mu\ [\urcorner]\ (n + k - i))$
$[\urcorner]\ j$
  proof –
    fix *j*
    assume $j < n$
    have $\mu\ [\urcorner]\ ((int\ k - int\ i) * int\ j) = (\mu\ [\urcorner]\ (int\ k - int\ i))\ [\urcorner]\ j$
      using *int-pow-pow* **by** (*metis* ‹$\mu \in$ *Units R*› *int-pow-int*)
    also have ... $= (\mu\ [\urcorner]\ n \otimes \mu\ [\urcorner]\ (int\ k - int\ i))\ [\urcorner]\ j$
    proof –
      have $\mu\ [\urcorner]\ (int\ k - int\ i) \in$ *carrier R*
        using ‹$\mu \in$ *Units R*› *int-pow-closed Units-closed* **by** *simp*
      then have $\mu\ [\urcorner]\ (int\ k - int\ i) = \mu\ [\urcorner]\ n \otimes \mu\ [\urcorner]\ (int\ k - int\ i)$
        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 μ
  assumes good: ⋀i. i ∈ {1..<n} ⟹ (⨁j ← [0..<n]. (μ [^] i) [^] j) = 𝟬
  shows primitive-root n (inv μ)
     ⋀i. i ∈ {1..<n} ⟹ (⨁j ← [0..<n]. ((inv μ) [^] 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 $(\bigoplus j \leftarrow [0..<n]. (\mu [\urcorner] (n - i)) [\urcorner] j) = \mathbf{0}$
using *assms* **by** *blast*
moreover have $\mu [\urcorner] (n - i) = inv\ \mu [\urcorner] i$
using *assms inv-pow-root-of-unity*[*of n* μ *i*] ‹$i \in \{1..<n\}$›
by *auto*
ultimately show $(\bigoplus j \leftarrow [0..<n]. ((inv\ \mu) [\urcorner] i) [\urcorner] j) = \mathbf{0}$ **by** *simp*
qed
done

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

lemma *sufficiently-good-aux*:
assumes *primitive-root* $m\ \eta$
assumes $m = 2\ \hat{}\ j$
assumes $\eta [\urcorner] (m\ div\ 2) = \ominus \mathbf{1}$
assumes *odd* r
assumes $r * 2\ \hat{}\ k < m$
shows $(\bigoplus l \leftarrow [0..<m]. (\eta [\urcorner] (r * 2\ \hat{}\ k)) [\urcorner] l) = \mathbf{0}$
using *assms*
proof (*induction k arbitrary*: $\eta\ m\ j$)
case *0*
then have *root-of-unity* $m\ \eta$ **by** *simp*
then have $\eta \in carrier\ R$ **by** *simp*
have $j > 0$

  proof (*rule ccontr*)
    assume ¬ *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 div 2* + *m div 2* **by** *auto*
  then have (⨁ *l* ← [*0*..<*m*]. (*η* [^] (*r* ∗ *2* ^ *0*)) [^] *l*) = (⨁ *l* ← [*0*..<*m div 2* + *m div 2*]. (*η* [^] *r*) [^] *l*)
    by *simp*
  also have ... = (⨁ *l* ← [*0*..<*m div 2*]. (*η* [^] *r*) [^] *l*) ⊕ (⨁ *l* ← [*m div 2*..<*m div 2* + *m div 2*]. (*η* [^] *r*) [^] *l*)
   by (*intro monoid-sum-list-split*[*symmetric*] *nat-pow-closed*, *rule* ‹*η* ∈ *carrier R*›)
  also have ... = (⨁ *l* ← [*0*..<*m div 2*]. (*η* [^] *r*) [^] *l*) ⊕ (⨁ *l* ← [*0*..<*m div 2*]. (*η* [^] *r*) [^] (*m div 2* + *l*))
    by (*intro arg-cong*[**where** *f* = (⊕) -] *monoid-sum-list-index-shift-0*)
  also have ... = (⨁ *l* ← [*0*..<*m div 2*]. (*η* [^] *r*) [^] *l* ⊕ (*η* [^] *r*) [^] (*m div 2* + *l*))
    by (*intro monoid-sum-list-add-in nat-pow-closed*; *rule* ‹*η* ∈ *carrier R*›)
  also have ... = (⨁ *l* ← [*0*..<*m div 2*]. (*η* [^] *r*) [^] *l* ⊖ (*η* [^] *r*) [^] *l*)
  proof (*intro monoid-sum-list-cong*)
    fix *l*
    have (*η* [^] *r*) [^] (*m div 2* + *l*) = (*η* [^] *r*) [^] (*m div 2*) ⊗ (*η* [^] *r*) [^] *l*
      by (*intro nat-pow-mult*[*symmetric*] *nat-pow-closed*, *rule* ‹*η* ∈ *carrier R*›)
    also have (*η* [^] *r*) [^] (*m div 2*) = (⊖ **1**) [^] *r*
      unfolding *nat-pow-pow*[*OF* ‹*η* ∈ *carrier R*›] *mult.commute*[*of r* -]
      by (*simp only*: *nat-pow-pow*[*symmetric*] ‹*η* ∈ *carrier R*› ‹*η* [^] (*m div 2*) = ⊖ **1**›)
    also have ... = ⊖ **1** **using** ‹*odd r*›
      by (*simp add*: *powers-of-negative*)
    finally have (*η* [^] *r*) [^] (*m div 2* + *l*) = ⊖ ((*η* [^] *r*) [^] *l*)
      using ‹*η* ∈ *carrier R*› *nat-pow-closed* **by** *algebra*
    then show (*η* [^] *r*) [^] *l* ⊕ (*η* [^] *r*) [^] (*m div 2* + *l*) = (*η* [^] *r*) [^] *l* ⊖ (*η* [^] *r*) [^] *l*
      unfolding *minus-eq*
      by (*intro arg-cong*[**where** *f* = (⊕) -])
  qed
  also have ... = (⨁ *l* ← [*0*..<*m div 2*]. **0**)
    by (*intro monoid-sum-list-cong*) (*simp add*: ‹*η* ∈ *carrier R*›)
  also have ... = **0** **by** *simp*
  finally show *?case* **.**
next
  case (*Suc k*)
  have *j* > *0*
  proof (*rule ccontr*)
    assume ¬ *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 ‹odd r› by simp
  qed
  then have even m using Suc by simp
  then have m = m div 2 + m div 2 by auto
  have root-of-unity m η using ‹primitive-root m η› by simp
  then have η ∈ carrier R by simp
  from ‹j > 0› obtain j' where j = Suc j'
    using gr0-implies-Suc by blast
  then have m div 2 = 2 ^ j' using ‹m = 2 ^ j› by simp
  have j' > 0
  proof (rule ccontr)
    assume ¬ j' > 0
    then have j' = 0 by simp
    then have m = 2 using ‹m = 2 ^ j› ‹j = Suc j'› by simp
    then have r * 2 ^ Suc k < 2 using Suc by simp
    then show False using ‹odd r› by simp
  qed
  then have even (m div 2) using ‹m div 2 = 2 ^ j'› by simp
  have IH': (⨁ l ← [0..<m div 2]. ((η [^] (2::nat)) [^] (r * 2 ^ k)) [^] l) = 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 ‹m = 2 ^ j› ‹j = Suc j'› by simp
    subgoal
      by (simp add: ‹η ∈ carrier R› nat-pow-pow ‹even (m div 2)› ‹η [^] (m div 2)
= ⊖ 𝟭›)
    subgoal using ‹odd r› .
    subgoal using ‹r * 2 ^ (Suc k) < m› ‹even m› by auto
    done
  have (⨁ l ← [0..<m]. (η [^] (r * 2 ^ (Suc k))) [^] l) = (⨁ l ← [0..<m]. ((η [^]
(2::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 ... = (⨁ l ← [0..<m div 2 + m div 2]. ((η [^] (2::nat)) [^] (r * 2 ^
k)) [^] l)
    using ‹m = m div 2 + m div 2› by argo
  also have ... = (⨁ l ← [0..<m div 2]. ((η [^] (2::nat)) [^] (r * 2 ^ k)) [^] l) ⊕
(⨁ l ← [m div 2..<m div 2 + m div 2]. ((η [^] (2::nat)) [^] (r * 2 ^ k)) [^] l)
   by (intro monoid-sum-list-split[symmetric] nat-pow-closed, rule ‹η ∈ carrier R›)
  also have ... = 𝟬 ⊕ (⨁ l ← [m div 2..<m div 2 + m div 2]. ((η [^] (2::nat)) [^]
(r * 2 ^ k)) [^] l)
    using IH' by argo
  also have ... = (⨁ l ← [m div 2..<m div 2 + m div 2]. ((η [^] (2::nat)) [^] (r
```

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

lemma *sufficiently-good*:
assumes *primitive-root n* μ
assumes *domain R* $\vee (n = 2\ \widehat{}\ k \wedge \mu\ [\widehat{}]\ (n\ div\ 2) = \ominus \mathbf{1})$
shows *good*: $\bigwedge i.\ i \in \{1..<n\} \Longrightarrow (\bigoplus j \leftarrow [0..<n].\ (\mu\ [\widehat{}]\ i)\ [\widehat{}]\ j) = \mathbf{0}$
proof (*cases domain R*)
case *True*
fix *i*
assume $i \in \{1..<n\}$

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

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

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

```
  then have n = 2 ^ k μ [^] (n div 2) = ⊖ 1 using assms(2) by auto

  have root-of-unity n μ using ‹primitive-root n μ› by simp
  then have μ ∈ carrier R μ [^] n = 1 by simp-all

  fix i
  assume i ∈ {1..<n}
  define l where l = Max {s. 2 ^ s dvd i}
  define r where r = i div 2 ^ l
  from ‹i ∈ {1..<n}› have i ≠ 0 by simp
  then have i = r * 2 ^ l 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 (⨁j ← [0..<n]. (μ [^] i) [^] j) = 0
    apply (simp only: ‹i = r * 2 ^ l›)
    apply (intro sufficiently-good-aux[of n μ k r l, OF ‹primitive-root n μ› ‹n = 2
^ k› ‹μ [^] (n div 2) = ⊖ 1› ‹odd r›])
    using ‹i = r * 2 ^ l› ‹i ∈ {1..<n}› by simp
qed

corollary inversion-rule-inv:
  fixes μ :: 'a
  fixes n :: nat
  assumes n > 0
  assumes primitive-root n μ
  assumes good: ⋀i. i ∈ {1..<n} ⟹ (⨁j ← [0..<n]. (μ [^] i) [^] j) = 0
  assumes[simp]: length a = n
  assumes[simp]: set a ⊆ carrier R
  shows NTT μ (NTT (inv μ) a) = map (λx. nat-embedding n ⊗ x) a
  using assms inv-good[of n μ] inversion-rule[of n inv μ a]
  using Units-inv-inv[of μ]
  using roots-Units[of n μ]
  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]: ⋀i. i < n ⟹ f i ∈ carrier R
  assumes[simp]: ⋀i. i < n ⟹ g i ∈ carrier R
  shows (⨁i ← [0..<n]. f i ⊗ x [^] i) ⊗ (⨁i ← [0..<n]. g i ⊗ x [^] i) =
    (⨁k ← [0..<n]. (⨁i ← [0..<n]. f i ⊗ g ((n + k − i) mod n)) ⊗ x [^] k)
proof (cases n > 0)
  case False
  then have n = 0 by simp
  then show ?thesis by simp
next
  case True
```

have[*simp*]: $x \in$ *carrier R* **using** ‹*root-of-unity n x*› **by** *simp*

have $(\bigoplus k \leftarrow [0..<n].\ (\bigoplus i \leftarrow [0..<n].\ f\ i \otimes g\ ((n + k - i)\ mod\ n)) \otimes x\ [^\frown]\ k)$
$=$
$(\bigoplus k \leftarrow [0..<n].\ (\bigoplus i \leftarrow [0..<n].\ f\ i \otimes g\ ((n + k - i)\ mod\ n) \otimes x\ [^\frown]\ k))$
by (*intro monoid-sum-list-cong monoid-sum-list-in-right*[*symmetric*] *nat-pow-closed m-closed*)
simp-all
also have ... $= (\bigoplus k \leftarrow [0..<n].\ (\bigoplus i \leftarrow [0..<n].\ f\ i \otimes g\ ((n + k - i)\ mod\ n) \otimes x\ [^\frown]\ ((n + k - i)\ mod\ n + i)))$
apply (*intro monoid-sum-list-cong arg-cong*[**where** $f = (\otimes)$ -])
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 ... $= (\bigoplus k \leftarrow [0..<n].\ (\bigoplus i \leftarrow [0..<n].\ f\ i \otimes g\ ((n + k - i)\ mod\ n) \otimes (x\ [^\frown]\ ((n + k - i)\ mod\ n) \otimes x\ [^\frown]\ i)))$
by (*intro monoid-sum-list-cong arg-cong*[**where** $f = (\otimes)$ -] *nat-pow-mult*[*symmetric*])
simp
also have ... $= (\bigoplus k \leftarrow [0..<n].\ (\bigoplus i \leftarrow [0..<n].\ f\ i \otimes x\ [^\frown]\ i \otimes (g\ ((n + k - i)\ mod\ n) \otimes x\ [^\frown]\ ((n + k - i)\ mod\ n))))$
proof –
have *reorder*: $\bigwedge a\ b\ c\ d.\ [\![\ a \in$ *carrier R*; $b \in$ *carrier R*; $c \in$ *carrier R*; $d \in$ *carrier R* $]\!] \Longrightarrow$
$a \otimes b \otimes (c \otimes d) = a \otimes d \otimes (b \otimes c)$
using *m-comm m-assoc* **by** *algebra*
show *?thesis*
by (*intro monoid-sum-list-cong reorder nat-pow-closed*) *simp-all*
qed
also have ... $= (\bigoplus i \leftarrow [0..<n].\ (\bigoplus k \leftarrow [0..<n].\ f\ i \otimes x\ [^\frown]\ i \otimes (g\ ((n + k - i)\ mod\ n) \otimes x\ [^\frown]\ ((n + k - i)\ mod\ n))))$
by (*intro monoid-sum-list-swap m-closed nat-pow-closed*) *simp-all*
also have ... $= (\bigoplus i \leftarrow [0..<n].\ f\ i \otimes x\ [^\frown]\ i \otimes (\bigoplus k \leftarrow [0..<n].\ (g\ ((n + k - i)\ mod\ n) \otimes x\ [^\frown]\ ((n + k - i)\ mod\ n))))$
by (*intro monoid-sum-list-cong monoid-sum-list-in-left m-closed nat-pow-closed*)
simp-all
also have ... $= (\bigoplus i \leftarrow [0..<n].\ f\ i \otimes x\ [^\frown]\ i \otimes (\bigoplus j \leftarrow [0..<n].\ (g\ j \otimes x\ [^\frown]\ j)))$
(**is** $(\bigoplus i \leftarrow$ -. - $\otimes$ *?lhs i*$) = (\bigoplus i \leftarrow$ -. - $\otimes$ *?rhs i*))
proof –
have $\bigwedge i.\ i \in$ *set* $[0..<n] \Longrightarrow$ *?lhs i* $=$ *?rhs i*
proof (*intro monoid-sum-list-index-permutation*[*symmetric*] *m-closed nat-pow-closed*)
fix *i*
assume $i \in$ *set* $[0..<n]$
have *bij-betw* $(\lambda ia.\ (n - i + ia)\ mod\ n)\ \{0..<n\}\ \{0..<n\}$
by (*intro const-add-mod-bij*)
also have *bij-betw* $(\lambda ia.\ (n - i + ia)\ mod\ n)\ \{0..<n\}\ \{0..<n\} =$
bij-betw $(\lambda ia.\ (n + ia - i)\ mod\ n)\ \{0..<n\}\ \{0..<n\}$
apply (*intro bij-betw-cong*)
using ‹$i \in$ *set* $[0..<n]$› **by** *simp*
finally show *bij-betw* $(\lambda ia.\ (n + ia - i)\ mod\ n)$ (*set* $[0..<n]$) (*set* $[0..<n]$)
by *simp*

qed *simp-all*
then show *?thesis*
by (*intro monoid-sum-list-cong*) (*intro arg-cong*[**where** $f = (\otimes)$ -])
qed
also have ... $= (\bigoplus i \leftarrow [0..<n].\ f\ i \otimes x\ [\urcorner]\ i) \otimes (\bigoplus j \leftarrow [0..<n].\ (g\ j \otimes x\ [\urcorner]\ j))$
by (*intro monoid-sum-list-in-right monoid-sum-list-closed*) *simp-all*
finally show *?thesis* **by** *argo*
qed

context
fixes n :: *nat*
begin

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

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

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

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

theorem *convolution-rule*:
assumes *length* $a = n$
assumes *length* $b = n$
assumes *set* $a \subseteq$ *carrier R*
assumes *set* $b \subseteq$ *carrier R*
assumes *root-of-unity* n μ
assumes $i < n$

shows *NTT* μ *a* ! *i* $\otimes$ *NTT* μ *b* ! *i* = *NTT* μ (*a* $\star$ *b*) ! *i*
proof (*cases n* > *0*)
case *False*
then show *?thesis* **using** ‹*i* < *n*› **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* {μ. *root-of-unity n* μ} *roots-of-unity-group n*
apply *unfold-locales*
subgoal using *roots-Units* ‹*n* > *0*› **by** *blast*
subgoal unfolding *roots-of-unity-group-def*[*OF* ‹*n* > *0*›] **by** *simp*
done

have $\mu \in$ *carrier R* **using** *assms*(*5*) **by** *simp*
have *NTT* μ *a* ! *i* $\otimes$ *NTT* μ *b* ! *i* =
($\bigoplus$*j* ← [*0*..<*n*]. *a* ! *j* $\otimes$ (μ [^] *i*) [^] *j*) $\otimes$ ($\bigoplus$*j* ← [*0*..<*n*]. *b* ! *j* $\otimes$ (μ [^] *i*) [^] *j*)
unfolding *NTT-nth*[*OF assms*(*1*) ‹*i* < *n*›] *NTT-nth*[*OF assms*(*2*) ‹*i* < *n*›] **by** *argo*
also have ... = ($\bigoplus$*j* ← [*0*..<*n*]. ($\bigoplus$*k* ← [*0*..<*n*]. (*a* ! *k*) $\otimes$ (*b* ! ((*n* + *j* − *k*) *mod n*))) $\otimes$ (μ [^] *i*) [^] *j*)
apply (*intro root-of-unity-power-sum-product root-of-unity-nat-pow-closed*)
using *True* ‹*root-of-unity n* μ› *set-subseteqD*[*OF assms*(*3*)] *set-subseteqD*[*OF assms*(*4*)] *assms*(*1*) *assms*(*2*)
by *simp-all*
also have ... = ($\bigoplus$*j* ← [*0*..<*n*]. (*a* $\star$ *b*) ! *j* $\otimes$ (μ [^] *i*) [^] *j*)
apply (*intro monoid-sum-list-cong arg-cong*[**where** *f* = λ*j*. *j* $\otimes$ -] *cyclic-convolution-nth*[*symmetric*])
by *simp*
also have ... = *NTT* μ (*a* $\star$ *b*) ! *i*
apply (*intro NTT-nth*[*symmetric*]) **using** ‹*i* < *n*› **by** *simp-all*
finally show *?thesis* **.**
qed

end

end

end

2.4 Fast Number Theoretic Transforms in Rings

theory *FNTT-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[*simp*]: *length a* = *n*
  assumes[*simp*]: *j* < *n*
  assumes[*simp*]: *set a* ⊆ *carrier R*
  defines *j′* ≡ (*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 μ a*) ! *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′*
proof −
  from *assms* **have** *n* > *0* **by** *linarith*
  have[*simp*]: *μ* ∈ *carrier R* **using** ‹*primitive-root n μ*› **unfolding** *primitive-root-def root-of-unity-def* **by** *blast*
  then have *μ-pow-carrier*[*simp*]: *μ* [^] *i* ∈ *carrier R* **for** *i* :: *nat* **by** *simp*
  show *j′* < *n div 2* **unfolding** *j′-def* **using** ‹*j* < *n*› ‹*even n*› **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*[*simp*]: *a* ! (*2* ∗ *i*) ∈ *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*[*simp*]: *a* ! (*2* ∗ *i* + *1*) ∈ *carrier R* **if** *i* < *n div 2* **for** *i*
    using *set-subseteqD*[*OF* ‹*set a* ⊆ *carrier R*›] *assms that* **by** *simp*

  have *μ-pow*: *μ* [^] (*j* ∗ (*2* ∗ *i*)) = (*μ* [^] (*2::nat*)) [^] (*j′* ∗ *i*) **for** *i*
  proof −
    have *μ* [^] (*j* ∗ (*2* ∗ *i*)) = (*μ* [^] (*j* ∗ *2*)) [^] *i*
      using *mult.assoc nat-pow-pow*[*symmetric*] **by** *simp*
    also have *μ* [^] (*j* ∗ *2*) = *μ* [^] (*j′* ∗ *2*)
    proof (*cases j* < *n div 2*)
      case *True*
      then show *?thesis* **unfolding** *j′-def* **by** *simp*
    next
      case *False*
      then have *μ* [^] (*j* ∗ *2*) = *μ* [^] (*j′* ∗ *2* + *n*)
        using *j′-alt* **by** (*simp add*: ‹*even n*›)
      also have ... = *μ* [^] (*j′* ∗ *2*)
        using ‹*n* > *0*› ‹*primitive-root n μ*›
        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 $(NTT\ \mu\ a)\ !\ j = (\bigoplus i \leftarrow [0..<n].\ a\ !\ i \otimes (\mu\ [\ulcorner]\ (j * i)))$
using *NTT-nth-2*[*of a n j* μ] **by** *simp*
also have ... $= (\bigoplus i \leftarrow [0..<n\ div\ 2].\ a\ !\ (2 * i) \otimes (\mu\ [\ulcorner]\ (j * (2 * i))))$
$\oplus (\bigoplus i \leftarrow [0..<n\ div\ 2].\ a\ !\ (2 * i + 1) \otimes (\mu\ [\ulcorner]\ (j * (2 * i + 1))))$
using ‹*even n*›
by (*intro monoid-sum-list-even-odd-split m-closed nat-pow-closed set-subseteqD*) *simp-all*
also have $(\bigoplus i \leftarrow [0..<n\ div\ 2].\ a\ !\ (2 * i + 1) \otimes (\mu\ [\ulcorner]\ (j * (2 * i + 1))))$
$= (\bigoplus i \leftarrow [0..<n\ div\ 2].\ \mu\ [\ulcorner]\ j \otimes (a\ !\ (2 * i + 1) \otimes (\mu\ [\ulcorner]\ (j * (2 * i)))))$
proof (*intro monoid-sum-list-cong*)
fix *i*
assume $i \in set\ [0..<n\ div\ 2]$
then have[*simp*]: $i < n\ div\ 2$ **by** *simp*
have $a\ !\ (2 * i + 1) \otimes (\mu\ [\ulcorner]\ (j * (2 * i + 1))) =$
$a\ !\ (2 * i + 1) \otimes (\mu\ [\ulcorner]\ (j * (2 * i)) \otimes \mu\ [\ulcorner]\ j)$
unfolding *distrib-left mult-1-right*
unfolding *nat-pow-mult*[*symmetric*, *OF* ‹$\mu \in carrier\ R$›]
by (*rule refl*)
also have ... $= (a\ !\ (2 * i + 1) \otimes \mu\ [\ulcorner]\ (j * (2 * i))) \otimes \mu\ [\ulcorner]\ j$
using *a-odd-carrier*[*OF* ‹$i < n\ div\ 2$›]
by (*intro m-assoc*[*symmetric*]; *simp*)
also have ... $= \mu\ [\ulcorner]\ j \otimes (a\ !\ (2 * i + 1) \otimes \mu\ [\ulcorner]\ (j * (2 * i)))$
using *a-odd-carrier*[*OF* ‹$i < n\ div\ 2$›]
by (*intro m-comm*; *simp*)
finally show $a\ !\ (2 * i + 1) \otimes \mu\ [\ulcorner]\ (j * (2 * i + 1)) =$ ... **.**
qed
also have ... $= \mu\ [\ulcorner]\ j \otimes (\bigoplus i \leftarrow [0..<n\ div\ 2].\ a\ !\ (2 * i + 1) \otimes (\mu\ [\ulcorner]\ (j * (2 * i))))$
using *a-odd-carrier* **by** (*intro monoid-sum-list-in-left*; *simp*)
finally have $(NTT\ \mu\ a)\ !\ j = (\bigoplus i \leftarrow [0..<n\ div\ 2].\ a\ !\ (2 * i) \otimes (\mu\ [\ulcorner]\ (2::nat))\ [\ulcorner]\ (j' * i))$
$\oplus\ \mu\ [\ulcorner]\ j \otimes (\bigoplus i \leftarrow [0..<n\ div\ 2].\ a\ !\ (2 * i + 1) \otimes (\mu\ [\ulcorner]\ (2::nat))\ [\ulcorner]\ (j' * i))$
unfolding μ*-pow* **.**
also have ... $= (\bigoplus i \leftarrow [0..<n\ div\ 2].\ [a\ !\ i.\ i \leftarrow filter\ even\ [0..<n]]\ !\ i \otimes (\mu\ [\ulcorner]\ (2::nat))\ [\ulcorner]\ (j' * i))$
$\oplus\ \mu\ [\ulcorner]\ j \otimes (\bigoplus i \leftarrow [0..<n\ div\ 2].\ [a\ !\ i.\ i \leftarrow filter\ odd\ [0..<n]]\ !\ i \otimes (\mu\ [\ulcorner]\ (2::nat))\ [\ulcorner]\ (j' * i))$
by (*intro-cong* [*cong-tag-2* ($\oplus$), *cong-tag-2* ($\otimes$)] *more*: *monoid-sum-list-cong*)
(*simp-all add*: *filter-even-nth length-filter-even length-filter-odd filter-odd-nth*)
also have ... $= (NTT\ (\mu\ [\ulcorner]\ (2::nat))\ [a\ !\ i.\ i \leftarrow filter\ even\ [0..<n]])\ !\ j'$
$\oplus\ \mu\ [\ulcorner]\ j \otimes (NTT\ (\mu\ [\ulcorner]\ (2::nat))\ [a\ !\ i.\ i \leftarrow filter\ odd\ [0..<n]])\ !\ j'$
by (*intro-cong* [*cong-tag-2* ($\oplus$), *cong-tag-2* ($\otimes$)] *more*: *NTT-nth-2*[*symmetric*])
(*simp-all add*: *length-filter-even length-filter-odd* ‹*even n*› ‹$j' < n\ div\ 2$›)
finally show $(NTT\ \mu\ a)\ !\ j =$ ... **.**
qed

lemma *NTT-recursion-1*:

assumes *even n*
assumes *primitive-root n μ*
assumes[*simp*]: *length a = n*
assumes[*simp*]: *j < n div 2*
assumes[*simp*]: *set a ⊆ carrier R*
shows (*NTT μ a*) ! *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*
proof −
have *j < n* **using** ‹*j < n div 2*› **by** *linarith*
show *?thesis*
using *NTT-recursion*[*OF* ‹*even n*› ‹*primitive-root n μ*› ‹*length a = n*› ‹*j < n*› ‹*set a ⊆ carrier R*›]
using ‹*j < n div 2*› **by** *presburger*
qed

lemma *NTT-recursion-2*:
assumes *even n*
assumes *primitive-root n μ*
assumes[*simp*]: *length a = n*
assumes[*simp*]: *j < n div 2*
assumes[*simp*]: *set a ⊆ carrier R*
assumes *halfway-property*: *μ* [^] (*n div 2*) = ⊖ **1**
shows (*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*
proof −
from *assms* **have** *μ ∈ carrier R* **unfolding** *primitive-root-def root-of-unity-def* **by** *simp*
then have *carrier-1*: *μ* [^] *j ∈ carrier R*
by *simp*
have *carrier-2*: *NTT* (*μ* [^] (*2::nat*)) (*map* ((!) *a*) (*filter odd* [*0..<n*])) ! *j ∈ carrier R*
apply (*intro NTT-nth-closed*[**where** *n = n div 2*])
subgoal using ‹*set a ⊆ carrier R*› ‹*length a = n*› **by** *fastforce*
subgoal using ‹*μ ∈ carrier R*› **by** *simp*
subgoal by (*simp add*: *length-filter-odd*)
subgoal using ‹*j < n div 2*› **.**
done
have *n div 2 + j < n* **using** ‹*j < n div 2*› ‹*even n*› **by** *linarith*
then have (*NTT μ a*) ! (*n div 2* + *j*) =
(*NTT* (*μ* [^] (*2::nat*)) [*a* ! *i*. *i* ← *filter even* [*0..<n*]]) ! *j*
⊕ *μ* [^] (*n div 2* + *j*) ⊗ (*NTT* (*μ* [^] (*2::nat*)) [*a* ! *i*. *i* ← *filter odd* [*0..<n*]]) ! *j*
using *NTT-recursion*[*OF* ‹*even n*› ‹*primitive-root n μ*› ‹*length a = n*› ‹*n div 2 + j < n*› ‹*set a ⊆ carrier R*›]
by *simp*
also have *μ* [^] (*n div 2* + *j*) = ⊖ (*μ* [^] *j*)
unfolding *nat-pow-mult*[*symmetric*, *OF* ‹*μ ∈ carrier R*›] *halfway-property*

```
    by (intro minus-eq-mult-one[symmetric]; simp add: ‹μ ∈ carrier R›)
  finally show ?thesis unfolding minus-eq l-minus[OF carrier-1 carrier-2] .
qed

lemma NTT-diffs:
  assumes even n
  assumes primitive-root n μ
  assumes length a = n
  assumes j < n div 2
  assumes set a ⊆ carrier R
  assumes μ [^] (n div 2) = ⊖ 1
  shows NTT μ a ! j ⊖ NTT μ a ! (n div 2 + j) = nat-embedding 2 ⊗ (μ [^] j
⊗ NTT (μ [^] (2::nat)) (map ((!) a) (filter odd [0..<n])) ! j)
proof −
  have[simp]: μ ∈ carrier R using ‹primitive-root n μ› unfolding primitive-root-def
root-of-unity-def by blast
  define ntt1 where ntt1 = NTT (μ [^] (2::nat)) (map ((!) a) (filter even [0..<n]))
! j
  have ntt1 ∈ carrier R unfolding ntt1-def
    apply (intro set-subseteqD[OF NTT-closed] set-subseteqI)
    subgoal for i
      using set-subseteqD[OF ‹set a ⊆ carrier R›]
      by (simp add: filter-even-nth ‹length a = n› ‹even n› length-filter-even)
    subgoal by simp
    subgoal using assms by (simp add: length-filter-even ‹even n›)
    done
  define ntt2 where ntt2 = NTT (μ [^] (2::nat)) (map ((!) a) (filter odd [0..<n]))
! j
  have ntt2 ∈ carrier R unfolding ntt2-def
    apply (intro set-subseteqD[OF NTT-closed] set-subseteqI)
    subgoal for i
      using set-subseteqD[OF ‹set a ⊆ carrier R›]
      by (simp add: filter-odd-nth ‹length a = n› ‹even n› length-filter-odd)
    subgoal by simp
    subgoal using assms by (simp add: length-filter-odd ‹even n›)
    done
  have NTT μ a ! j ⊖ NTT μ a ! (n div 2 + j) =
    (ntt1 ⊕ μ [^] j ⊗ ntt2) ⊖ (ntt1 ⊖ μ [^] j ⊗ ntt2)
    apply (intro arg-cong2[where f = λi j. i ⊖ j])
    unfolding ntt1-def ntt2-def
    subgoal by (intro NTT-recursion-1 assms)
    subgoal by (intro NTT-recursion-2 assms)
    done
  also have ... = μ [^] j ⊗ (ntt2 ⊕ ntt2)
    using ‹ntt1 ∈ carrier R› ‹ntt2 ∈ carrier R› nat-pow-closed[OF ‹μ ∈ carrier
R›]
    by algebra
  also have ... = μ [^] j ⊗ ((1 ⊕ 1) ⊗ ntt2)
    using ‹ntt2 ∈ carrier R› one-closed by algebra
```

```
  also have ... = μ [^] j ⊗ (nat-embedding 2 ⊗ ntt2)
    by (simp add: numeral-2-eq-2)
  also have ... = nat-embedding 2 ⊗ (μ [^] j ⊗ ntt2)
   using nat-pow-closed[OF ‹μ ∈ carrier R›] ‹ntt2 ∈ carrier R› 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 ⇒ 'a list ⇒ 'a list where
FNTT μ [] = []
| FNTT μ [x] = [x]
| FNTT μ [x, y] = [x ⊕ y, x ⊖ y]
| FNTT μ a = (let n = length a;
                  nums1 = [a!i.  i ← filter even [0..<n]];
                  nums2 = [a!i.  i ← filter odd [0..<n]];
                  b = FNTT (μ [^] (2::nat)) nums1;
                  c = FNTT (μ [^] (2::nat)) nums2;
                  g = [b!i ⊕ (μ [^] i) ⊗ c!i. i ← [0..<(n div 2)]];
                  h = [b!i ⊖ (μ [^] i) ⊗ c!i. i ← [0..<(n div 2)]]
               in g@h)
lemmas [simp del] = FNTT-term-aux

declare FNTT.simps[simp del]

lemma length-FNTT[simp]:
  assumes length a = 2 ^ k
  shows length (FNTT μ a) = length a
  using assms
proof (induction rule: FNTT.induct)
  case (1 μ)
  then show ?case by simp
next
  case (2 μ x)
  then show ?case by (simp add: FNTT.simps)
next
  case (3 μ x y)
  then show ?case by (simp add: FNTT.simps)
next
  case (4 μ 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 ← 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 (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)]]
then have length g = n div 2 by simp
define h where h = [b!i ⊖ (μ [^] i) ⊗ 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 ‹length (a1 # a2 # a3 # as) = n› ‹n = 2 ^ k›
  by (cases k = 0) simp-all
then have even n n div 2 = 2 ^(k − 1)
  using ‹n = 2 ^ k› by (simp-all add: power-diff)

have FNTT μ (a1 # a2 # a3 # as) = g @ h
  unfolding FNTT.simps
  using ‹length (a1 # a2 # a3 # as) = n› by (simp only: Let-def defs)
then have FNTT μ a = g @ h using a-def by simp

have recursive-halfway: (μ [^] (2 :: nat)) [^] (n div 2 div 2) = ⊖ 1
proof −
  have n ≥ 3
    using ‹length (a1 # a2 # a3 # as) = n› by simp
  then have k ≥ 2 using ‹n = 2 ^ k› by (cases k ∈ {0, 1}) auto
  then have even (n div 2) using ‹n div 2 = 2 ^(k − 1)› by fastforce
  then show ?thesis
    by (simp add: nat-pow-pow ‹μ ∈ carrier R› ‹μ [^] (n div 2) = ⊖ 1›)
qed

have b = NTT (μ [^] (2::nat)) nums1
  unfolding b-def
  apply (intro 4(1)[of n nums1 nums2 n div 2 k − 1])
  subgoal using ‹length (a1 # a2 # a3 # as) = n› by simp
  subgoal using nums1-def a-def by simp
  subgoal using nums2-def a-def by simp
  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 using nums1-def length-filter-even ‹even n› by simp
  subgoal
    unfolding nums1-def
    apply (intro set-subseteqI)
    using set-subseteqD[OF ‹set (a1 # a2 # a3 # as) ⊆ carrier R›]
    by (simp add: a-def[symmetric] filter-even-nth length-filter-even ‹even n›)
  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 ‹$\mu \in$ *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)* $\subseteq$ *carrier R*›]
by (*simp add: a-def[symmetric] filter-odd-nth length-filter-odd*)
done

show *?case*
proof (*intro nth-equalityI*)
have[*simp*]: *length (FNTT* μ *(a1 # a2 # a3 # as)) = n*
using ‹*length (a1 # a2 # a3 # as) = n*› ‹*n = 2 ^ k*› *length-FNTT[of a1 # a2 # a3 # as]*
by *blast*
then show *length (FNTT* 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 (FNTT* μ *(a1 # a2 # a3 # as))*
then have *i < n* **by** *simp*

have *FNTT* 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*
$\oplus$ μ [^] *i* $\otimes$ (*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)* $\subseteq$ *carrier R*› *a-def*
using ‹$\mu \in$ *carrier R*› ‹*length (a1 # a2 # a3 # as) = n*›
by *simp-all*

also have ... = (*NTT* (μ [^] (*2::nat*)) *nums1*) ! *i*
$\oplus$ μ [^] *i* $\otimes$ (*NTT* (μ [^] (*2::nat*)) *nums2*) ! *i*
unfolding *nums1-def nums2-def* **by** *blast*
also have ... = *b ! i* $\oplus$ μ [^] *i* $\otimes$ *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) ⊆ 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 ⇒ 'a list ⇒ 'a list where
FNTT' μ [] = []
| FNTT' μ [x] = [x]
| FNTT' μ [x, y] = [x ⊕ y, x ⊖ y]
| FNTT' μ a = (let n = length a;
                  nums1 = evens-odds True a;
                  nums2 = evens-odds False a;
                  b = FNTT' (μ [^] (2::nat)) nums1;
                  c = FNTT' (μ [^] (2::nat)) nums2;
                  g = [b!i ⊕ (μ [^] i) ⊗ c!i. i ← [0..<(n div 2)]];
                  h = [b!i ⊖ (μ [^] i) ⊗ c!i. i ← [0..<(n div 2)]]
               in g@h)

lemma FNTT'-FNTT: FNTT' μ xs = FNTT μ xs
  apply (induction μ xs rule: FNTT'.induct)
  subgoal by (simp add: FNTT.simps)
  subgoal by (simp add: FNTT.simps)
  subgoal by (simp add: FNTT.simps)
  subgoal for μ 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 ⇒ 'a list ⇒ 'a list where
FNTT'' μ [] = []
| FNTT'' μ [x] = [x]
| FNTT'' μ [x, y] = [x ⊕ y, x ⊖ y]
| FNTT'' μ a = (let n = length a;
                  nums1 = evens-odds True a;
                  nums2 = evens-odds False a;
                  b = FNTT'' (μ [^] (2::nat)) nums1;
                  c = FNTT'' (μ [^] (2::nat)) nums2;
                  g = map2 (⊕) b (map2 (⊗) [μ [^] i. i ← [0..<(n div 2)]] c);
                  h = map2 (λx y. x ⊖ y) b (map2 (⊗) [μ [^] i. i ← [0..<(n div 2)]]
c)
               in g@h)
```

```
lemma FNTT''-FNTT':
  assumes length a = 2 ^ k
  shows FNTT'' μ a = FNTT' μ a
  using assms
proof (induction μ a arbitrary: k rule: FNTT''.induct)
  case (4 μ a1 a2 a3 as)
  define a where a = a1 # a2 # a3 # as
  define n where n = length a
  then have n = 2 ^ k using 4 a-def by simp
  then have k ≥ 2 using n-def a-def by (cases k = 0; cases k = 1) simp-all
  then have even n using ‹n = 2 ^ k› by simp
  have n div 2 = 2 ^(k − 1) using ‹n = 2 ^ k› ‹k ≥ 2› by (simp add: power-diff)
  then have even (n div 2) using ‹k ≥ 2› by simp
  define nums1 where nums1 = evens-odds True a
  then have length nums1 = n div 2
    using length-filter-even[of n] filter-comprehension-evens-odds[of a] n-def ‹even n›
    by (metis length-map)
  define nums2 where nums2 = evens-odds False a
  then have length nums2 = n div 2
    using length-filter-odd[of n] filter-comprehension-evens-odds[of a] n-def
    by (metis length-map)
  define b where b = FNTT' (μ [^] (2::nat)) nums1
  then have length b = n div 2
    by (simp add: FNTT'-FNTT ‹length nums1 = n div 2› ‹n div 2 = 2 ^(k − 1)›)
  define c where c = FNTT' (μ [^] (2::nat)) nums2
  then have length c = n div 2
    by (simp add: FNTT'-FNTT ‹length nums2 = n div 2› ‹n div 2 = 2 ^(k − 1)›)
  define g1 where g1 = [b!i ⊕ (μ [^] i) ⊗ c!i. i ← [0..<(n div 2)]]
  then have length g1 = n div 2 by simp
  define h1 where h1 = [b!i ⊖ (μ [^] i) ⊗ c!i. i ← [0..<(n div 2)]]
  then have length h1 = n div 2 by simp
  define g2 where g2 = map2 (⊕) b (map2 (⊗) [μ [^] i. i ← [0..<(n div 2)]] c)
  then have length g2 = n div 2
    by (simp add: ‹length b = n div 2› ‹length c = n div 2›)

  have g1 = g2
    apply (intro nth-equalityI)
    subgoal by (simp only: ‹length g1 = n div 2› ‹length g2 = n div 2›)
    subgoal for i
      by (simp add: g1-def g2-def ‹length b = n div 2› ‹length c = n div 2›)
    done

  define h2 where h2 = map2 (λx y. x ⊖ y) b (map2 (⊗) [μ [^] i. i ← [0..<(n 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* ⟹ *a* ∈ *carrier R* ⟹ *a* [^] *n* = **1** ⟹ *a* ∈ *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* ^ *n* ⟹ *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* ≡ (*2*::*nat*) ^ *k*

definition *Zn* **where** *Zn* ≡ *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* ⇒ *int* **where**
to-residue-ring xs = *int* (*Nat-LSBF.to-nat xs*) *mod int n*

lemma *to-residue-ring-in-carrier*: *to-residue-ring xs* ∈ *carrier Zn*
  unfolding *to-residue-ring-def res-carrier-eq* **by** *simp*

definition *from-residue-ring* :: *int* ⇒ *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 ... = *xs* @ (*replicate* (*k* − *length xs*) *False*) **using** *fill-def* **by** *simp*
  finally have *Nat-LSBF.to-nat* (*reduce xs*) = *Nat-LSBF.to-nat xs* **by** *simp*
  moreover have *Nat-LSBF.to-nat xs* ≤ *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 *Nat-LSBF.to-nat xs* $<$ *2* ^ *k*
  using *Nat.le-diff-conv2* **by** *auto*
 ultimately show *?thesis* **by** *simp*
next
 case *False*
 then have *length* (*take k xs*) = *k fill k xs* = *xs xs* = (*take k xs*) @ (*drop k xs*)
using *fill-def* **by** *simp-all*
 then have *Nat-LSBF.to-nat xs* = *Nat-LSBF.to-nat* (*take k xs*) + *n* ∗ *Nat-LSBF.to-nat* (*drop k xs*)
  using *to-nat-app*[*of take k xs drop k xs*] **by** *simp*
 moreover have *Nat-LSBF.to-nat* (*take k xs*) ≤ *2* ^ *k* − *1*
  using *to-nat-length-upper-bound*[*of take k xs*] ‹*length* (*take k xs*) = *k*› **by** *simp*
 hence *Nat-LSBF.to-nat* (*take k 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**
add-mod x y = *reduce* (*add-nat x y*)

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

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

lemma *to-nat-subtract-mod*: *length xs* ≤ *k* ⟹ *length ys* ≤ *k* ⟹ *int* (*Nat-LSBF.to-nat* (*subtract-mod xs ys*)) = (*int* (*Nat-LSBF.to-nat xs*) − *int* (*Nat-LSBF.to-nat ys*)) *mod n*
proof (*cases Nat-LSBF.to-nat xs* ≤ *Nat-LSBF.to-nat ys*)
 case *True*
 assume *length xs* ≤ *k*
 assume *length ys* ≤ *k*
 then have *Nat-LSBF.to-nat ys* ≤ *n* − *1*
  using *to-nat-length-upper-bound*[*of ys*]
  by (*meson diff-le-mono le-trans one-le-numeral power-increasing*)
 then have *Nat-LSBF.to-nat ys* ≤ *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*: *to-residue-ring* (*add-mod x y*) = *to-residue-ring x* $\oplus_{Zn}$ *to-residue-ring y*
proof −
  have *to-residue-ring* (*add-mod x y*) = *to-residue-ring* (*reduce* (*add-nat x y*))
    unfolding *add-mod-def* **by** *simp*
  also have ... = (*Nat-LSBF.to-nat x* + *Nat-LSBF.to-nat y*) *mod n*
    using *to-nat-reduce add-nat-correct to-residue-ring-def* **by** *simp*
  also have ... = (*int* (*Nat-LSBF.to-nat x*) *mod n* + (*int* (*Nat-LSBF.to-nat y*) *mod n*)) *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 *length x* $\leq$ *k*
  assumes *length y* $\leq$ *k*
  assumes *n* > *1*
  shows *to-residue-ring* (*subtract-mod x y*) = *to-residue-ring x* $\ominus_{Zn}$ *to-residue-ring y*
proof −
  have *to-residue-ring* (*subtract-mod x y*) = *int* (*Nat-LSBF.to-nat* (*subtract-mod x y*)) *mod int n*
    unfolding *to-residue-ring-def* **by** *argo*
  also have ... = (*int* (*Nat-LSBF.to-nat x*) − (*int* (*Nat-LSBF.to-nat y*))) *mod int n*
    by (*simp add*: *to-nat-subtract-mod assms*)
  also have ... = (*to-residue-ring x* + (− *to-residue-ring y mod n*)) *mod n*
    unfolding *diff-conv-add-uminus to-residue-ring-def*
    by (*simp add*: *mod-add-eq mod-diff-right-eq*)
  also have ... = (*to-residue-ring x* + ($\ominus_{residue\text{-}ring\ n}$ (*to-residue-ring y mod n*))) *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** ‹*n* > *1*›
    by (*metis int-ops*(*2*) *nat-int-comparison*(*2*))
  also have ... = *to-residue-ring x* $\ominus_{residue\text{-}ring\ n}$ (*to-residue-ring y mod n*)
    unfolding *a-minus-def*
    by (*simp add*: *residue-ring-def*)
  also have *to-residue-ring y mod n* = *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* ^ *k* $\Longrightarrow$ *length* (*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)) ≤ k using ‹x < 2 ^ k›
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 :: 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 :: 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* ⇒ *nat-lsbf* ⇒ *nat-lsbf tm* **where**
add-mod-tm xs ys =*1 xs* $+_{nt}$ *ys* ⋙ *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*) ≤ *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* ⇒ *nat-lsbf* ⇒ *nat-lsbf tm* **where**
subtract-mod-tm xs ys =*1 do* {
  b ← *xs* $\leq_{nt}$ *ys*;
  if b then do {
    fillx ← *fill-tm k xs*;
    fillx1 ← *fillx* $@_t$ [*True*];
    fillx1 $-_{nt}$ *ys* ⋙ *reduce-tm*
  } *else xs* $-_{nt}$ *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*) ≤ *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*) ≤ *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*) $@_t$ [*True*]) +
          time ((*fill k xs* @ [*True*]) $-_{nt}$ *ys*) +
          time (*reduce-tm* ((*fill k xs* @ [*True*]) $-_n$ *ys*))
     else time (*xs* $-_{nt}$ *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 ¬ (∀ i∈{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 ... ≤ 2 ^ (length (xs1 @ True # xs2)) − 1
    by (intro to-nat-length-upper-bound)
  also have ... = 2 ^ (length xs) − 1
    using ‹xs = xs1 @ False # xs2› by simp
  finally show False using assms by simp
qed

lemma residue-ring-pow: n > 1 ⟹ a [^]residue-ring n b = (a ^ b) mod n
 by (induction b) (simp-all add: residue-ring-def mod-mult-right-eq mult.commute)

lemma (in residues) pow-nat-eq:
a [^]R (n :: nat) = a ^ n 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 ≡ (2::nat) ^ (2 ^ 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 ≡ 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**
$fermat\text{-}non\text{-}unique\text{-}carrier \equiv \{xs :: nat\text{-}lsbf.\ length\ xs = 2\ \hat{}\ (k + 1)\}$

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

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

lemma *two-pow-half-carrier-length*[*simp*]: $(int\ 2\ \hat{}\ (2\ \hat{}\ k))\ mod\ n = -1\ mod\ n$
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*: $2\ \hat{}\ (2\ \hat{}\ k)\ mod\ n \neq 1$
by *simp*

lemma *two-pow-carrier-length*[*simp*]: $(2::nat)\ \hat{}\ (2\ \hat{}\ (k + 1))\ mod\ n = 1$
proof −
have $int\ 2\ \hat{}\ (2\ \hat{}\ (k + 1))\ mod\ n = 1$
proof −
have $int\ 2\ \hat{}\ (2\ \hat{}\ (k + 1))\ mod\ n = ((int\ 2)\ \hat{}\ (2 * 2\ \hat{}\ k))\ mod\ n$
by *simp*
also have ... $= ((int\ 2)\ \hat{}\ (2\ \hat{}\ k))\ \hat{}\ 2\ mod\ n$
using *power-mult*[*of int 2 2* ^ *k 2*]
by (*simp add*: *mult.commute*)
also have ... $= (int\ 2\ \hat{}\ (2\ \hat{}\ k) * int\ 2\ \hat{}\ (2\ \hat{}\ k))\ mod\ n$
by (*simp add*: *power2-eq-square*)
also have ... $= (((int\ 2\ \hat{}\ (2\ \hat{}\ k))\ mod\ n) * ((int\ 2\ \hat{}\ (2\ \hat{}\ k))\ mod\ n))\ mod\ n$
by *simp*
also have $(int\ 2\ \hat{}\ (2\ \hat{}\ k))\ mod\ n = -1\ mod\ n$
using *two-pow-half-carrier-length* **.**
finally have $int\ 2\ \hat{}\ (2\ \hat{}\ (k + 1))\ mod\ n = int\ 1\ mod\ n$
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*]:
$(2::int)\ [\hat{}]_{Fn}\ (2::nat)\ \hat{}\ k = \ominus_{Fn}\ \mathbf{1}_{Fn}$

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

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

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

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

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

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

using *nat-mod-eqE* **by** *blast*
have $2\ [\hat{}]_{Fn}\ a = 2\ [\hat{}]_{Fn}\ a \otimes_{Fn} (2\ [\hat{}]_{Fn}\ ((2{::}nat)\ \hat{}\ (k + 1)))\ [\hat{}]_{Fn}\ i$
using *two-pow-carrier-length-residue-ring two-in-carrier nat-pow-closed*
using *nat-pow-one* **by** *algebra*
also have ... $= 2\ [\hat{}]_{Fn}\ (a + i * 2\ \hat{}\ (k + 1))$
using *nat-pow-pow nat-pow-mult two-in-carrier*
using *mult.commute* **by** *metis*
also have ... $= 2\ [\hat{}]_{Fn}\ (b + j * 2\ \hat{}\ (k + 1))$
using *0* **by** *argo*
also have ... $= 2\ [\hat{}]_{Fn}\ b \otimes_{Fn} (2\ [\hat{}]_{Fn}\ ((2{::}nat)\ \hat{}\ (k + 1)))\ [\hat{}]_{Fn}\ j$
using *nat-pow-pow nat-pow-mult two-in-carrier*
using *mult.commute* **by** *metis*
also have ... $= 2\ [\hat{}]_{Fn}\ b$
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 $s \le 2\ \hat{}\ k$
shows $2\ [\hat{}]_{Fn}\ s = 2\ \hat{}\ s$
proof −
from *assms* **have** $2\ \hat{}\ s \le int\ n - 1$ **by** *simp*
then have $2\ \hat{}\ s < int\ n$ **using** *n-positive* **by** *linarith*
then have $2\ \hat{}\ s = 2\ \hat{}\ s\ mod\ int\ n$ **by** *simp*
also have ... $= 2\ [\hat{}]_{Fn}\ s$ **using** *pow-nat-eq* **by** *simp*
finally show *?thesis* **by** *argo*
qed

lemma *two-powers-Units*:
assumes $s \le 2\ \hat{}\ k$
shows $2\ \hat{}\ s \in Units\ Fn$
unfolding *two-powers-trivial*[*OF assms*, *symmetric*]
by (*intro Units-pow-closed two-is-unit*)
corollary *two-powers-in-carrier*:
assumes $s \le 2\ \hat{}\ k$
shows $2\ \hat{}\ s \in carrier\ Fn$
using *assms two-powers-Units Units-closed* **by** *simp*

lemma *two-powers-half-carrier-length-residue-ring*[*simp*]:
assumes $i + s = k$
shows $(2\ \hat{}\ 2\ \hat{}\ i)\ [\hat{}]_{Fn}\ (2{::}nat)\ \hat{}\ s = \ominus_{Fn} \mathbf{1}_{Fn}$
proof −
from *assms* **have** $i \le k$ **by** *simp*
then have $(2\ \hat{}\ 2\ \hat{}\ i)\ [\hat{}]_{Fn}\ (2{::}nat)\ \hat{}\ s =$
$(2\ [\hat{}]_{Fn}\ ((2{::}nat)\ \hat{}\ i))\ [\hat{}]_{Fn}\ (2{::}nat)\ \hat{}\ s$
using *two-powers-trivial*[*of* $2\ \hat{}\ i$, *symmetric*] **by** *simp*
also have ... $= 2\ [\hat{}]_{Fn}\ ((2{::}nat)\ \hat{}\ (i + s))$
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 ... = $\ominus_{Fn}$ $\mathbf{1}_{Fn}$
using ‹$i + s = k$› *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* $[\char94]_{Fn}$ ((*2*::*nat*) ^ (*k* + *1*) − *1*)
proof −
have $\mathbf{1}_{Fn}$ = *2* $\otimes_{Fn}$ *2* $[\char94]_{Fn}$ ((*2*::*nat*) ^ (*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* $[\char94]_{Fn}$ ((*2*::*nat*) ^ (*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* $[\char94]_{Fn}$ ((*2*::*nat*) ^ (*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* ≤ *2* ^ *k*
shows inv_{Fn} (*2* ^ *s*) = *2* $[\char94]_{Fn}$ (*2* ^ (*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* ^ *s*) = inv_{Fn} (*2* $[\char94]_{Fn}$ *s*)
using *two-powers-trivial*[*OF* ‹*s* ≤ *2* ^ *k*›] **by** *argo*
also have ... = (inv_{Fn} *2*) $[\char94]_{Fn}$ *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 ... = (*2* $[\char94]_{Fn}$ ((*2*::*nat*) ^ (*k* + *1*) − *1*)) $[\char94]_{Fn}$ *s*
using *inv-of-2*
by *argo*

```
  also have ... = 2 [^]_Fn (((2::nat) ^ (k + 1) − 1) ∗ s)
    using two-in-carrier nat-pow-pow by presburger
  also have ((2::nat) ^ (k + 1) − 1) ∗ s = (2::nat) ^ (k + 1) ∗ s − s
    using diff-mult-distrib by simp
  also have ... = 2 ^ (k + 1) ∗ (s − 1) + 2 ^ (k + 1) − s
   using ‹s > 0› by (metis add.commute mult.commute mult-eq-if zero-less-iff-neq-zero)
  also have ... = 2 ^ (k + 1) ∗ (s − 1) + (2 ^ (k + 1) − s)
    apply (intro diff-add-assoc) using assms by simp
  also have 2 [^]_Fn (2 ^ (k + 1) ∗ (s − 1) + (2 ^ (k + 1) − s)) =
    2 [^]_Fn (2 ^ (k + 1) − s)
    apply (intro pow-mod-carrier-length) by simp
  finally show ?thesis .
qed

lemma inv-pow-mod-carrier-length:
  assumes (a::nat) mod 2 ^ (k + 1) = b mod 2 ^ (k + 1)
  shows (inv_Fn 2) [^]_Fn a = (inv_Fn 2) [^]_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 ∃ i j. (a::nat) = j + i ∗ m ∧ 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) ^ a mod n = (2::nat) ^ (a mod (2 ^ (k + 1))) mod
n
proof −
  define i where i = a mod 2 ^ (k + 1)
  define j where j = a div 2 ^ (k + 1)
  have a = i + j ∗ 2 ^ (k + 1) using mod-div-mult-eq[of a 2 ^ (k + 1)] i-def j-def
    by simp
  hence (2::nat) ^ a mod n = 2 ^ i ∗ (2 ^ (2 ^ (k + 1))) ^ j mod n
    using power-add[of 2::nat i j ∗ 2 ^ (k + 1)]
    using power-mult[of 2::nat 2 ^ (k + 1) j]
    using mult.commute[of j 2 ^ (k + 1)]
    by argo
  also have ... = 2 ^ i ∗ ((2 ^ (2 ^ (k + 1))) ^ j mod n) mod n
    using mod-mult-right-eq by metis
  also have ... = 2 ^ i ∗ ((2 ^ (2 ^ (k + 1)) mod n) ^ j mod n) mod n
    using power-mod by metis
  also have ... = 2 ^ i ∗ ((1::nat) ^ j mod n) mod n
    using two-pow-carrier-length by simp
  also have ... = 2 ^ i mod n by simp
  finally show ?thesis using i-def by simp
qed
```

lemma *fermat-carrier-length*[*simp*]: *xs* ∈ *fermat-non-unique-carrier* ⟹ *length xs* = *2* ^ (*k* + *1*)
  unfolding *fermat-non-unique-carrier-def* **by** *simp*

fun *to-residue-ring* :: *nat-lsbf* ⇒ *int* **where**
to-residue-ring xs = *int* (*Nat-LSBF.to-nat xs*) *mod int n*
fun *from-residue-ring* :: *int* ⇒ *nat-lsbf* **where**
from-residue-ring x = *fill* (*2* ^ (*k* + *1*)) (*Nat-LSBF.from-nat* (*nat x*))

lemma *to-residue-ring-in-carrier*[*simp*]: *to-residue-ring xs* ∈ *carrier Fn*
  using *zmod-int*[*of* - *n*, *symmetric*]
  by (*simp add*: *res-carrier-eq*)

lemma *to-residue-ring-eq-to-nat*: *Nat-LSBF.to-nat xs* < *n* ⟹ *to-residue-ring xs* = *int* (*Nat-LSBF.to-nat xs*)
  using *zmod-int*
  by (*metis to-residue-ring.simps mod-less*)

definition *multiply-with-power-of-2* :: *nat-lsbf* ⇒ *nat* ⇒ *nat-lsbf* **where**
multiply-with-power-of-2 xs m = *rotate-right m xs*

definition *divide-by-power-of-2* :: *nat-lsbf* ⇒ *nat* ⇒ *nat-lsbf* **where**
divide-by-power-of-2 xs m = *rotate-left m xs*

lemma *length-multiply-with-power-of-2*[*simp*]: *length* (*multiply-with-power-of-2 xs m*) = *length xs*
  unfolding *multiply-with-power-of-2-def* **by** *simp*

lemma *length-divide-by-power-of-2*[*simp*]: *length* (*divide-by-power-of-2 xs m*) = *length xs*
  unfolding *divide-by-power-of-2-def* **by** *simp*

lemma (**in** *euclidean-semiring-cancel*) *sum-list-mod*: (∑*i* ← *xs*. (*f i mod m*)) *mod m* = (∑*i* ← *xs*. *f i*) *mod m*
proof (*induction xs*)
  case *Nil*
  then show *?case* **by** *simp*
next
  case (*Cons a xs*)
  have (∑*i* ← (*a* # *xs*). *f i*) *mod m* = (*f a* + (∑*i* ← *xs*. *f i*)) *mod m*
    by *simp*
  also have ... = (*f a mod m* + (∑*i* ← *xs*. *f i*) *mod m*) *mod m*
    using *mod-add-eq*[*symmetric*, *of f a*] **by** *simp*
  also have ... = (*f a mod m* + (∑*i* ← *xs*. *f i mod m*) *mod m*) *mod m*
    using *Cons.IH* **by** *argo*
  also have ... = (*f a mod m* + (∑*i* ← *xs*. *f i mod m*)) *mod m*
    using *mod-add-right-eq* **by** *blast*

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

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

lemma *multiply-with-power-of-2-correct'*: $xs \in fermat\text{-}non\text{-}unique\text{-}carrier \Longrightarrow Nat\text{-}LSBF.to\text{-}nat$ $(multiply\text{-}with\text{-}power\text{-}of\text{-}2\ xs\ m)\ mod\ n = Nat\text{-}LSBF.to\text{-}nat\ xs * 2 \ \hat{}\ m\ mod\ n \wedge$ $multiply\text{-}with\text{-}power\text{-}of\text{-}2\ xs\ m \in fermat\text{-}non\text{-}unique\text{-}carrier$
proof (*intro conjI*)
  assume $xs \in fermat\text{-}non\text{-}unique\text{-}carrier$
  then have *length-xs*: $length\ xs = 2 \ \hat{}\ (k + 1)$ **by** *simp*
  then have $length\ xs > 0$ **by** *simp*

  let $?m = length\ xs - m\ mod\ length\ xs$

  define *ys zs* **where** $ys = take\ ?m\ xs\ zs = drop\ ?m\ xs$
  then have $xs = ys\ @\ zs$
    and *length-ys*: $length\ ys = ?m$
    and *length-zs*: $length\ zs = m\ mod\ length\ xs$
    using ‹$length\ xs = 2 \ \hat{}\ (k + 1)$› **by** *simp-all*

  have *1*: $Nat\text{-}LSBF.to\text{-}nat\ xs = Nat\text{-}LSBF.to\text{-}nat\ ys + 2 \ \hat{}\ ?m * Nat\text{-}LSBF.to\text{-}nat$ zs (**is** $- = ?y + - * ?z$)
    apply (*unfold* ‹$xs = ys\ @\ zs$› *to-nat-app*)
    apply (*unfold* ‹$xs = ys\ @\ zs$›[*symmetric*] *length-ys*)
    apply (*rule refl*)
    done

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

    using ‹*xs* = *ys* @ *zs*› *length-zs* **by** *simp*
  also have ... = *zs* @ *ys*
    by (*rule rotate-right-append*)
  finally show *?thesis* **.**
 qed
 then have *3*: *Nat-LSBF.to-nat* (*multiply-with-power-of-2 xs m*)
  = *?z* + *2* ^ (*m mod length xs*) ∗ *?y*
  by (*simp add*: *to-nat-app length-zs*)

 from *1* **have** *Nat-LSBF.to-nat xs* ∗ *2* ^ *m mod n* = (*?y* + *2* ^ *?m* ∗ *?z*) ∗ *2* ^
m mod n
  by *argo*
 also have ... = (*?y* + *2* ^ *?m* ∗ *?z*) ∗ (*2* ^ *m mod n*) *mod n*
  by (*simp add*: *mod-simps*)
 also have ... = (*?y* + *2* ^ *?m* ∗ *?z*) ∗ (*2* ^ (*m mod length xs*) *mod n*) *mod n*
  using *length-xs two-powers* **by** *algebra*
 also have ... = (*?y* + *2* ^ *?m* ∗ *?z*) ∗ *2* ^ (*m mod length xs*) *mod n*
  by (*simp add*: *mod-simps*)
 also have ... = (*?y* ∗ *2* ^ (*m mod length xs*) + *2* ^ (*?m* + (*m mod length xs*)) ∗
?z) *mod n*
  by (*simp add*: *algebra-simps power-add*)
 also have ... = (*?y* ∗ *2* ^ (*m mod length xs*) + *2* ^ *length xs* ∗ *?z*) *mod n*
  by (*simp add*: *length-xs*)
 also have ... = (*?y* ∗ *2* ^ (*m mod length xs*) + (*2* ^ *length xs mod n*) ∗ *?z mod
n*) *mod n*
  by (*simp add*: *mod-simps*)
 also have ... = (*?y* ∗ *2* ^ (*m mod length xs*) + *1* ∗ *?z mod n*) *mod n*
  by (*simp only*: *length-xs two-pow-carrier-length*)
 also have ... = (*?z* + *2* ^ (*m mod length xs*) ∗ *?y*) *mod n*
  by (*simp add*: *mod-simps algebra-simps*)
 also have ... = *Nat-LSBF.to-nat* (*multiply-with-power-of-2 xs m*) *mod n*
  using *3* **by** *argo*
 finally show *Nat-LSBF.to-nat* (*multiply-with-power-of-2 xs m*) *mod n* = *Nat-LSBF.to-nat*
xs ∗ *2* ^ *m mod n*
  by *argo*

 have *length* (*multiply-with-power-of-2 xs m*) = *length xs*
  using *2* ‹*xs* = *ys* @ *zs*› **by** *simp*
 then show *multiply-with-power-of-2 xs m* ∈ *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* ∈ *fermat-non-unique-carrier*
 shows *multiply-with-power-of-2 xs m* ∈ *fermat-non-unique-carrier*
 by (*intro conjunct2*[*OF multiply-with-power-of-2-correct'*] *assms*)

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

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

lemma
assumes *xs* ∈ *fermat-non-unique-carrier*
shows *divide-by-power-of-2-correct*: *to-residue-ring* (*divide-by-power-of-2 xs m*) = *to-residue-ring xs* $\otimes_{Fn}$ (*inv*$_{Fn}$ *2*) [^]$_{Fn}$ *m*
and *divide-by-power-of-2-closed*: *divide-by-power-of-2 xs m* ∈ *fermat-non-unique-carrier*
unfolding *atomize-conj*
proof (*intro conjI*)
from *assms* **show** *c*: *divide-by-power-of-2 xs m* ∈ *fermat-non-unique-carrier*
unfolding *fermat-non-unique-carrier-def* **by** *simp*
define *divxs* **where** *divxs* = *divide-by-power-of-2 xs m*
define *mulxs* **where** *mulxs* = *multiply-with-power-of-2 xs m*

have *multiply-with-power-of-2 divxs m* = *xs*
unfolding *divxs-def multiply-with-power-of-2-def divide-by-power-of-2-def* **by** *simp*
then have *to-residue-ring xs* = *to-residue-ring* (*multiply-with-power-of-2 divxs m*)
by *simp*
also have ... = *to-residue-ring divxs* $\otimes_{Fn}$ *2* [^]$_{Fn}$ *m*
apply (*intro multiply-with-power-of-2-correct*)
unfolding *divxs-def* **by** (*rule c*)
finally have *to-residue-ring xs* $\otimes_{Fn}$ (*inv*$_{Fn}$ *2*) [^]$_{Fn}$ *m* = *to-residue-ring divxs* $\otimes_{Fn}$ *2* [^]$_{Fn}$ *m* $\otimes_{Fn}$ (*inv*$_{Fn}$ *2*) [^]$_{Fn}$ *m*
by *simp*
also have ... = *to-residue-ring divxs* $\otimes_{Fn}$ (*2* [^]$_{Fn}$ *m* $\otimes_{Fn}$ (*inv*$_{Fn}$ *2*) [^]$_{Fn}$ *m*)
apply (*intro m-assoc to-residue-ring-in-carrier nat-pow-closed two-in-carrier*)
using *two-is-unit* **by** *auto*
also have (*2* [^]$_{Fn}$ *m* $\otimes_{Fn}$ (*inv*$_{Fn}$ *2*) [^]$_{Fn}$ *m*) = (*2* $\otimes_{Fn}$ (*inv*$_{Fn}$ *2*)) [^]$_{Fn}$ *m*

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

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

lemma *add-fermat-correct′*:
assumes *xs* ∈ *fermat-non-unique-carrier*
assumes *ys* ∈ *fermat-non-unique-carrier*
shows *add-fermat xs ys* ∈ *fermat-non-unique-carrier* ∧ *Nat-LSBF.to-nat* (*add-fermat xs ys*) *mod n* = (*Nat-LSBF.to-nat xs* + *Nat-LSBF.to-nat ys*) *mod n*
proof −
define *zs* **where** *zs* = *add-nat xs ys*
show *?thesis*
proof (*cases length zs* = *2* ^ (*k* + *1*) + *1*)
case *True*
then have *add-fermat xs ys* = *inc-nat* (*butlast zs*)
using *zs-def* **unfolding** *add-fermat-def* **by** *simp*
then have *1*: *Nat-LSBF.to-nat* (*add-fermat xs ys*) = *1* + *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 *butlast zs* = *zs′* **by** *simp*
then have *Nat-LSBF.to-nat* (*add-fermat xs ys*) = *1* + *Nat-LSBF.to-nat zs′* **using** *1* **by** *simp*
moreover have *Nat-LSBF.to-nat zs* = *Nat-LSBF.to-nat zs′* + *2* ^ (*2* ^ (*k* + *1*))
using ‹*zs* = *zs′* @ [*True*]› *True* **by** (*simp add*: *to-nat-app*)
hence *Nat-LSBF.to-nat zs mod n* = (*Nat-LSBF.to-nat zs′* + *1*) *mod n*
using *two-pow-carrier-length* **by** (*metis mod-add-right-eq*)
ultimately have *2*: *Nat-LSBF.to-nat* (*add-fermat xs ys*) *mod n* = (*Nat-LSBF.to-nat xs* + *Nat-LSBF.to-nat ys*) *mod n*
using *add-nat-correct*[*of xs ys*] *zs-def* **by** *auto*

have *length zs′* = *2* ^ (*k* + *1*) **using** *True* ‹*zs* = *zs′* @ [*True*]› **by** *simp*

have *Nat-LSBF.to-nat zs* = *Nat-LSBF.to-nat xs* + *Nat-LSBF.to-nat ys* **using** *zs-def* **by** (*simp add*: *add-nat-correct*)
also have ... ≤ (*2* ^ *length xs* − *1*) + (*2* ^ *length ys* − *1*)
using *to-nat-length-upper-bound add-le-mono* **by** *algebra*

also have ... = (2 ^ (2 ^ (k + 1)) − 1) + (2 ^ (2 ^ (k + 1)) − 1)
using *assms* **by** *simp*
also have ... < (2 ^ (2 ^ (k + 1)) − 1) + (2 ^ (2 ^ (k + 1)))
by (*meson add-strict-left-mono diff-less pos2 zero-less-one zero-less-power*)
finally have *Nat-LSBF.to-nat zs′* < 2 ^ (2 ^ (k + 1)) − 1
using ‹*Nat-LSBF.to-nat zs* = *Nat-LSBF.to-nat zs′* + 2 ^ (2 ^ (k + 1))› **by** *simp*
then have *length* (*inc-nat zs′*) = *length zs′*
using *length-inc-nat′* ‹*length zs′* = 2 ^ (k + 1)› **by** *simp*
then have *length* (*add-fermat xs ys*) = 2 ^ (k + 1)
using ‹*add-fermat xs ys* = *inc-nat* (*butlast zs*)› ‹*butlast zs* = *zs′*› ‹*length zs′* = 2 ^ (k + 1)›
by *simp*
with *2* **show** *?thesis* **unfolding** *fermat-non-unique-carrier-def* **by** *simp*
next
case *False*
have *length zs* ≥ 2 ^ (k + 1)
using *assms zs-def length-add-nat-lower*[*of xs ys*] **by** *simp*
moreover have *length zs* ≤ 2 ^ (k + 1) + 1
using *assms zs-def length-add-nat-upper*[*of xs ys*] **by** *simp*
ultimately have *length zs* = 2 ^ (k + 1) **using** *False* **by** *simp*
then have *add-fermat xs ys* ∈ *fermat-non-unique-carrier*
unfolding *fermat-non-unique-carrier-def add-fermat-def*
by (*simp add*: *Let-def zs-def*)
moreover have *Nat-LSBF.to-nat zs* = *Nat-LSBF.to-nat xs* + *Nat-LSBF.to-nat ys*
by (*simp add*: *zs-def add-nat-correct*)
moreover have *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* ∈ *fermat-non-unique-carrier*
assumes *ys* ∈ *fermat-non-unique-carrier*
shows *add-fermat xs ys* ∈ *fermat-non-unique-carrier*
by (*intro conjunct1*[*OF add-fermat-correct′*] *assms*)

corollary *add-fermat-correct*:
assumes *xs* ∈ *fermat-non-unique-carrier*
assumes *ys* ∈ *fermat-non-unique-carrier*
shows *to-residue-ring* (*add-fermat xs ys*) = *to-residue-ring xs* $\oplus_{Fn}$ *to-residue-ring ys*
proof −
have *to-residue-ring* (*add-fermat xs ys*) = (*int* (*Nat-LSBF.to-nat xs*) + *int* (*Nat-LSBF.to-nat ys*)) *mod 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* $\oplus_{Fn}$ *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* ∈ *fermat-non-unique-carrier*
assumes *ys* ∈ *fermat-non-unique-carrier*
shows *subtract-fermat xs ys* ∈ *fermat-non-unique-carrier* ∧ *int* (*Nat-LSBF.to-nat* (*subtract-fermat xs ys*)) *mod n* = (*int* (*Nat-LSBF.to-nat xs*) − *int* (*Nat-LSBF.to-nat ys*)) *mod n*
proof −
from *assms*(*2*) **have** *multiply-with-power-of-2 ys* (*2* ^ *k*) ∈ *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* ∈ *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 ‹*multiply-with-power-of-2 ys* (*2* ^ *k*) ∈ *fermat-non-unique-carrier*›
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 $xs \in$ *fermat-non-unique-carrier*
assumes $ys \in$ *fermat-non-unique-carrier*
shows *subtract-fermat xs ys* $\in$ *fermat-non-unique-carrier*
by (*intro conjunct1*[*OF subtract-fermat-correct'*] *assms*)

corollary *subtract-fermat-correct*:
assumes $xs \in$ *fermat-non-unique-carrier*
assumes $ys \in$ *fermat-non-unique-carrier*
shows *to-residue-ring* (*subtract-fermat xs ys*) = *to-residue-ring xs* $\ominus_{Fn}$ *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* $\ominus_{Fn}$ *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* ⇒ *nat-lsbf* **where**
reduce xs = (*let* (*ys*, *zs*) = *split xs in*
if compare-nat zs ys then
subtract-nat ys zs
else
subtract-nat (*add-nat* (*True* # *replicate* (*2* ^ *k* − *1*) *False* @ [*True*]) *ys*) *zs*)

lemma *reduce-correct'*:
assumes $xs \in$ *fermat-non-unique-carrier*
shows *Nat-LSBF.to-nat* (*reduce xs*) < *n* ∧ *Nat-LSBF.to-nat* (*reduce xs*) *mod n* = *Nat-LSBF.to-nat xs mod n* **and** *length* (*reduce xs*) ≤ *2* ^ *k* + *2*
proof −
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 (*int* (*Nat-LSBF.to-nat ys*) − *int* (*Nat-LSBF.to-nat zs*)) *mod n* = (*int* (*Nat-LSBF.to-nat ys*) + (−*1*) *mod n* ∗ *int* (*Nat-LSBF.to-nat zs*)) *mod 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 ... = (*int* (*Nat-LSBF.to-nat ys*) + *2* ^ (*2* ^ *k*) *mod n* ∗ *int* (*Nat-LSBF.to-nat zs*)) *mod n*
using *two-pow-half-carrier-length* **by** *simp*
also have ... = (*int* (*Nat-LSBF.to-nat ys* + *2* ^ (*2* ^ *k*) ∗ *Nat-LSBF.to-nat zs*)) *mod n*
by *auto*
also have ... = (*int* (*Nat-LSBF.to-nat* (*ys* @ *zs*))) *mod n*
using ‹*length ys* = *2* ^ *k*› *to-nat-app* **by** *presburger*
also have ... = (*int* (*Nat-LSBF.to-nat xs*)) *mod n*
using ‹*split xs* = (*ys*, *zs*)› *app-split* **by** *presburger*
finally have *0*: (*int* (*Nat-LSBF.to-nat ys*) − *int* (*Nat-LSBF.to-nat zs*)) *mod n* = (*int* (*Nat-LSBF.to-nat xs*)) *mod n* **.**
have *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*
proof (*cases compare-nat zs ys*)
case *True*
then have *reduce xs* = *subtract-nat ys zs*
unfolding *reduce-def* ‹*split xs* = (*ys*, *zs*)› **by** *simp*
then have *1*: *Nat-LSBF.to-nat* (*reduce xs*) = *Nat-LSBF.to-nat ys* − *Nat-LSBF.to-nat zs*
using *subtract-nat-correct* **by** *presburger*
from *True* **have** *Nat-LSBF.to-nat zs* ≤ *Nat-LSBF.to-nat ys*
using *compare-nat-correct* **by** *blast*
with *1* **have** *int* (*Nat-LSBF.to-nat* (*reduce xs*)) = *int* (*Nat-LSBF.to-nat ys*) − *int* (*Nat-LSBF.to-nat zs*)
by *linarith*
then 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*) *mod n* = *Nat-LSBF.to-nat xs mod n*
using *zmod-int* **by** (*metis of-nat-eq-iff*)

have *Nat-LSBF.to-nat* (*reduce xs*) ≤ *Nat-LSBF.to-nat ys* **using** *1* **by** *linarith*
also have ... < *n* **using** ‹*Nat-LSBF.to-nat ys* < *n*› **.**
finally 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*) *mod n* = *Nat-LSBF.to-nat xs mod n*› **by** *blast*
moreover have *length* (*reduce xs*) ≤ *2* ^ *k* + *2* **unfolding** ‹*reduce xs* = *subtract-nat ys zs*›
apply (*estimation estimate*: *conjunct2*[*OF subtract-nat-aux*])
using ‹*length zs* = *2* ^ *k*› ‹*length ys* = *2* ^ *k*› **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-aux*])
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* ∈ *fermat-non-unique-carrier*
  shows *Nat-LSBF.to-nat xs mod n* = *Nat-LSBF.to-nat* (*reduce xs*)
  using *reduce-correct*′[*OF assms*] *mod-less* **by** *metis*

lemma *add-take-drop-carry-aux*:
  assumes *xs*′ = *add-nat* (*take e xs*) (*drop e xs*)
  assumes *length xs* = *e* + *1*
  assumes *e* ≥ *1*
  shows *length xs*′ ≤ *e* ∨ (*xs*′ = *replicate e False* @ [*True*] ∧ *xs* = *replicate e True* @ [*True*])
proof (*intro verit-and-neg*(*3*))
  assume *a*: ¬ (*length xs*′ ≤ *e*)
  then have *length xs*′ ≥ *e* + *1* **by** *simp*
  moreover have *length xs*′ ≤ *e* + *1*
    unfolding *assms*(*1*)
    apply (*estimation estimate*: *length-add-nat-upper*)
    using *assms* **by** *simp*
  ultimately have *len-xs*′: *length xs*′ = *e* + *1* **by** *simp*
  moreover have *max* (*length* (*take e xs*)) (*length* (*drop e xs*)) = *e*
    using *assms* **by** *simp*
  ultimately have ∃ *zs*. *xs*′ = *zs* @ [*True*]
    unfolding *assms*(*1*) **by** (*intro add-nat-last-bit-True*, *argo*)
  then obtain *zs* **where** *zs-def*: *xs*′ = *zs* @ [*True*] **and** *len-zs*: *length zs* = *e* **using** *len-xs*′ **by** *auto*

  have *Nat-LSBF.to-nat xs*′ = *Nat-LSBF.to-nat xs mod 2* ^ *e* + *Nat-LSBF.to-nat xs div 2* ^ *e*
    unfolding *assms*(*1*) **by** (*simp add*: *add-nat-correct to-nat-take to-nat-drop*)
  also have ... < (*2* ^ *e* − *1*) + (*2* ^ (*e* + *1*)) *div 2* ^ *e*
    apply (*intro add-le-less-mono*)
    subgoal using *pos-mod-bound*[*of 2* ^ *e Nat-LSBF.to-nat xs*] *two-pow-pos*
      by (*metis Suc-mask-eq-exp mask-eq-exp-minus-1 mod-Suc-le-divisor*)
    subgoal using *to-nat-length-upper-bound*[*of xs*] *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 ... = *2* ^ *e* + *1* **by** *simp*
  finally have *Nat-LSBF.to-nat xs*′ ≤ *2* ^ *e* **by** *simp*
  moreover have *Nat-LSBF.to-nat xs*′ = *Nat-LSBF.to-nat zs* + *2* ^ *e*
    unfolding *zs-def* **by** (*simp add*: *to-nat-app len-zs*)
  ultimately have *Nat-LSBF.to-nat zs* = *0* **by** *simp*
  then have *zs* = *replicate e False Nat-LSBF.to-nat xs*′ = *2* ^ *e*
   using *len-zs to-nat-zero-iff truncate-Nil-iff* ‹*Nat-LSBF.to-nat xs*′ = *Nat-LSBF.to-nat zs* + *2* ^ *e*›
    by *auto*
  then have *xs*′ = *replicate e False* @ [*True*] **using** *zs-def* **by** *simp*
  from *assms*(*2*) **obtain** *xst xsh* **where** *xs-decomp*: *xs* = *xst* @ [*xsh*] *length xst* = *e*

by (*metis Suc-eq-plus1 length-Suc-conv-rev*)
then have *take e xs* = *xst drop e xs* = [*xsh*] **using** *assms* **by** *simp-all*
moreover have[*simp*]: *xsh* = *True*
proof (*rule ccontr*)
assume *xsh* ≠ *True*
then have *drop e xs* = [*False*] **using** *xs-decomp* **by** *simp*
then have *Nat-LSBF.to-nat xs'* = *Nat-LSBF.to-nat* (*take e xs*)
unfolding *assms*(*1*) *add-nat-correct* **by** *simp*
also have ... < *2* ^ *e*
using *assms*(*2*) *to-nat-length-bound*[*of take e xs*] **by** *simp*
finally show *False* **using** ‹*Nat-LSBF.to-nat xs'* = *2* ^ *e*› **by** *simp*
qed
ultimately have *Nat-LSBF.to-nat xs'* = *Nat-LSBF.to-nat xst* + *1* **unfolding** *assms*(*1*) *add-nat-correct*
by *simp*
then have *Nat-LSBF.to-nat xst* = *2* ^ *e* − *1* **using** ‹*Nat-LSBF.to-nat xs'* = *2* ^ *e*› **by** *simp*
then have *xst* = *replicate e True* **using** *to-nat-replicate-True2*[*of xst*] ‹*length xst* = *e*› **by** *argo*
then have *xs* = *replicate e True* @ [*True*]
using ‹*xs* = *xst* @ [*xsh*]› **by** *simp*
then show *xs'* = *replicate e False* @ [*True*] ∧ *xs* = *replicate e True* @ [*True*]
using ‹*xs'* = *replicate e False* @ [*True*]›
by (*simp add*: *replicate-append-same*)
qed

function *from-nat-lsbf* :: *nat-lsbf* ⇒ *nat-lsbf* **where**
from-nat-lsbf xs = (*if length xs* ≤ *2* ^ (*k* + *1*) *then fill* (*2* ^ (*k* + *1*)) *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* :: *nat-lsbf*
define *e* :: *nat* **where** *e* = *2* ^ (*k* + *1*)
then have *e-ge-1*: *e* ≥ *1* **and** *e-ge-2*: *e* ≥ *2* **by** *simp-all*
define *xs'* **where** *xs'* = *add-nat* (*take e xs*) (*drop e xs*)
assume ¬ *length xs* ≤ *2* ^ (*k* + *1*)
then have *a*: *length xs* ≥ *e* + *1* **unfolding** *e-def* **by** *simp*
then consider *length xs* = *e* + *1* ∧ *length xs'* ≤ *e* |
length xs = *e* + *1* ∧ *length xs'* ≥ *e* + *1* |
length xs ≥ *e* + *2*
by *linarith*
then show (*add-nat* (*take* (*2* ^ (*k* + *1*)) *xs*) (*drop* (*2* ^ (*k* + *1*)) *xs*), *xs*)
∈ *measures* [*length*, *Nat-LSBF.to-nat*]
unfolding *e-def*[*symmetric*] *xs'-def*[*symmetric*]
proof *cases*
case *1*
then show (*xs'*, *xs*) ∈ *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* ⟷ (*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*) ∈ *measures* [*length*, *Nat-LSBF.to-nat*]
    using *2 xs′-rep* **by** *simp*
next
  case *3*
  have *length xs′* ≤ *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*) ∈ *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* ∈ *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* ∈ *fermat-non-unique-carrier*
    apply (*cases length xs* ≤ *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* ≤ *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-nat-lsbf xs* = *from-nat-lsbf* (*add-nat ?xs1 ?xs2*)
by (*simp add*: *from-nat-lsbf.simps*[*of xs*])
then have *to-residue-ring* (*from-nat-lsbf xs*) = *to-residue-ring* (*add-nat ?xs1 ?xs2*)
using *1*[*OF False*] **by** *argo*
also have ... = (*Nat-LSBF.to-nat ?xs1* + *Nat-LSBF.to-nat ?xs2*) *mod n* **by** (*simp add*: *add-nat-correct zmod-int*)
also have ... = (*Nat-LSBF.to-nat ?xs1* + (*2* ^ (*2* ^ (*k* + *1*))) ∗ *Nat-LSBF.to-nat ?xs2*) *mod 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 ... = (*Nat-LSBF.to-nat xs*) *mod 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-nat-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* ≤ *k* + *1*
using *divides-primepow-nat*[*OF two-is-prime-nat*]
by *blast*
have *i* = *k* + *1*
proof (*rule ccontr*)
assume *i* ≠ *k* + *1*
then have *i* ≤ *k* **using** ‹*i* ≤ *k* + *1*› **by** *linarith*
have *1* ≠ (*2*::*nat*) ^ (*2* ^ *k*) *mod n* **using** *two-pow-half-carrier-length-neq-1*[*symmetric*]
.
moreover have (*2*::*nat*) ^ (*2* ^ *k*) *mod n* = *1*
proof −
have (*2*::*nat*) ^ (*2* ^ *k*) *mod n* = (*2* ^ (*2* ^ *i*)) ^ (*2* ^ (*k* − *i*)) *mod n*
by (*simp add*: ‹*i* ≤ *k*› *power-add*[*symmetric*] *power-mult*[*symmetric*])
also have ... = (*2* ^ (*2* ^ *i*) *mod n*) ^ (*2* ^ (*k* − *i*)) *mod n*
by (*simp add*: *power-mod*)
also have *2* ^ (*2* ^ *i*) *mod n* = *1* **using** ‹*ord n 2* = *2* ^ *i*›
using *ord*[*of 2 n*] **unfolding** *cong-def* **using** *n-gt-1* **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* ⇒ *nat-lsbf* ⇒ *nat-lsbf*) ⇒ (*nat-lsbf* ⇒ *nat* ⇒ *nat-lsbf*) ⇒ *nat* ⇒ *nat-lsbf list* × *nat* ⇒ *nat-lsbf list* ⇒ *nat-lsbf list* ⇒ *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* ^ (*k* + *1*)) *bs cs*
| *fft-combine-b-c-aux f g l - - -* = *undefined*

fun *fft-ifft-combine-b-c-add* **where**
fft-ifft-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-ifft-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-ifft-combine-b-c-subtract* **where**
fft-ifft-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-ifft-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* ^ (*k* + *1*)
shows *fft-combine-b-c-aux f g l* (*revs*, *e*) *bs cs* = *rev revs* @ *map3* (λ*x y i*. *f x* (*g y* ((*e* + *l* ∗ *i*) *mod 2* ^ (*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.prems* **have** *len-bcs′*: *length bs′* = *len-bc length cs′* = *len-bc* **by** *simp-all*
have (*e* + *l* ∗ *i*) *mod 2* ^ (*k* + *1*) < *2* ^ (*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* ^ *k*)) *bs′ cs′*
unfolding *bcs* **by** *simp*
also have ... = *rev* (*f b* (*g c e*) # *revs*) @
map3 (λ*x y i*. *f x* (*g y* (((*e* + *l* ∗ *1*) *mod 2* ^ (*k* + *1*) + *l* ∗ *i*) *mod 2* ^ (*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 (λ*x y i*. *f x* (*g y* (((*e* + *l* ∗ *1*) *mod 2* ^ (*k* + *1*) + *l* ∗ *i*) *mod 2* ^ (*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′* [*Suc 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.prems* **by** *simp*
finally show *?l* = ... **.**
qed
qed

lemma *fft-ifft-combine-b-c-add-correct*:
assumes *length bs* = *len-bc length cs* = *len-bc*
shows *fft-ifft-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-ifft-combine-b-c-subtract-correct*:
assumes *length bs* = *len-bc length cs* = *len-bc*
shows *fft-ifft-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-ifft-combine-b-c-add-carrier*:
assumes *length bs* = *len-bc length cs* = *len-bc*
assumes *set bs* ⊆ *fermat-non-unique-carrier*

```
  assumes set cs ⊆ fermat-non-unique-carrier
  shows set (fft-ifft-combine-b-c-add it l bs cs) ⊆ fermat-non-unique-carrier
  unfolding fft-ifft-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-ifft-combine-b-c-subtract-carrier:
  assumes length bs = len-bc length cs = len-bc
  assumes set bs ⊆ fermat-non-unique-carrier
  assumes set cs ⊆ fermat-non-unique-carrier
  shows set (fft-ifft-combine-b-c-subtract it l bs cs) ⊆ fermat-non-unique-carrier
  unfolding fft-ifft-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-ifft :: bool ⇒ nat ⇒ nat-lsbf list ⇒ nat-lsbf list where
fft-ifft it l [] = []
| fft-ifft it l [x] = [x]
| fft-ifft it l [x, y] = [add-fermat x y, subtract-fermat x y]
| fft-ifft it l a = (let nums1 = evens-odds True a;
                   nums2 = evens-odds False a;
                   b = fft-ifft it (2 * l) nums1;
                   c = fft-ifft it (2 * l) nums2;
                   g = fft-ifft-combine-b-c-add it l b c;
                   h = fft-ifft-combine-b-c-subtract it l b c
                    in g@h)


fun fft where fft l xs = fft-ifft False l xs
fun ifft where ifft l xs = fft-ifft 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-ifft it (2 * l) nums1
definition c where c = fft-ifft it (2 * l) nums2
definition g where g = fft-ifft-combine-b-c-add it l b c
```

definition *h* **where** *h* = *fft-ifft-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* ≥ *2*
proof (*rule ccontr*)
  assume ¬ *e* ≥ *2*
  then have *e* ≤ *1* **by** *simp*
  have (*2*::*nat*) ^ *e* ≤ *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 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-ifft it l a* = *g* @ *h*
  unfolding *a-def fft-ifft.simps*[*of it l*] *Let-def*
  unfolding *defs*[*symmetric*] **by** (*rule refl*)

lemma
  assumes *set a* ⊆ *fermat-non-unique-carrier*
  shows *nums1-carrier*: *set nums1* ⊆ *fermat-non-unique-carrier*
  and *nums2-carrier*: *set nums2* ⊆ *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-ifft*:
  assumes *length a* = *2* ^ *e*
  shows *length* (*fft-ifft it l a*) = *2* ^ *e*
  using *assms*
proof (*induction it l a arbitrary*: *e rule*: *fft-ifft.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-ifft-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-ifft-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-ifft*[*OF assms*] **by** (*rule refl*)

lemma *length-ifft*:
assumes *length a* = *2* ^ *e*
shows *length* (*ifft l a*) = *2* ^ *e*
unfolding *ifft.simps length-fft-ifft*[*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-ifft length-nums1*)
lemma *length-c*: *length c* = *2* ^ (*e* − *1*)
unfolding *c-def* **by** (*intro length-fft-ifft length-nums2*)
lemma *length-g*: *length g* = *2* ^ (*e* − *1*)
unfolding *g-def fft-ifft-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-ifft-combine-b-c-subtract-correct*[*OF length-b length-c*] *map3-as-map*
by (*simp add*: *length-b length-c*)

end

context *int-lsbf-fermat*
begin

lemma *fft-ifft-carrier*:
  assumes *length a* = *2* ^ *l*
  assumes *set a* ⊆ *fermat-non-unique-carrier*
  shows *set* (*fft-ifft it s a*) ⊆ *fermat-non-unique-carrier*
using *assms* **proof** (*induction it s a arbitrary*: *l rule*: *fft-ifft.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* ⊆ *fermat-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.prems a-def* **by** *simp-all*
  have *c-carrier*: *set c* ⊆ *fermat-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.prems a-def* **by** *simp-all*

  have *g-carrier*: *set g* ⊆ *fermat-non-unique-carrier*
    unfolding *g-def*
    apply (*intro fft-ifft-combine-b-c-add-carrier*)
    using *length-b length-c b-carrier c-carrier* **by** *simp-all*
  have *h-carrier*: *set h* ⊆ *fermat-non-unique-carrier*
    unfolding *h-def*
    apply (*intro fft-ifft-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* ⊆ *fermat-non-unique-carrier*
shows *set* (*fft s a*) ⊆ *fermat-non-unique-carrier*
using *fft-ifft-carrier*[*OF assms*] **by** *simp*

lemma *ifft-carrier*:
assumes *length a* = *2* ^ *l*
assumes *set a* ⊆ *fermat-non-unique-carrier*
shows *set* (*ifft s a*) ⊆ *fermat-non-unique-carrier*
using *fft-ifft-carrier*[*OF assms*] **by** *simp*

lemma *fft-ifft-correct'*:
assumes *length a* = *2* ^ *l*
assumes *l* ≤ *k* + *1*
assumes *set a* ⊆ *fermat-non-unique-carrier*
shows *map to-residue-ring* (*fft-ifft it s a*) = *FNTT''* ((*if it then inv*$_{Fn}$ *2 else 2*) [^]$_{Fn}$ *s*) (*map to-residue-ring a*)
using *assms*
proof (*induction it s a arbitrary*: *l rule*: *fft-ifft.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-fermat-correct subtract-fermat-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*) [^]$_{Fn}$ *s*) [^]$_{Fn}$ (*2*::*nat*)) *nums1'*
define *c'* **where** *c'* = *FNTT''* (((*if it then inv*$_{Fn}$ *2 else 2*) [^]$_{Fn}$ *s*) [^]$_{Fn}$ (*2*::*nat*)) *nums2'*
define *g'* **where** *g'* = *map2* (⊕$_{Fn}$) *b'*
(*map2* (⊗$_{Fn}$)
(*map* (([^]$_{Fn}$) ((*if it then inv*$_{Fn}$ *2 else 2*) [^]$_{Fn}$ *s*)) [*0*..<*length local.a div 2*]) *c'*)
define *h'* **where** *h'* = *map2* (*a-minus Fn*) *b'*
(*map2* (⊗$_{Fn}$)
(*map* (([^]$_{Fn}$) ((*if it then inv*$_{Fn}$ *2 else 2*) [^]$_{Fn}$ *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*) [^]$_{Fn}$ *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* inv_{Fn} *2 else 2*) ∈ *Units Fn*
  using *two-is-unit Units-inv-Units* **by** *algebra*
then have *pr-carrier*: (*if it then* inv_{Fn} *2 else 2*) ∈ *carrier Fn*
  by (*rule Units-closed*)
have *pow-2s*: ((*if it then* inv_{Fn} *2 else 2*) $[^]_{Fn}$ *s*) $[^]_{Fn}$ (*2*::*nat*) = (*if it then* inv_{Fn} *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* ⊆ *fermat-non-unique-carrier*
unfolding *nums2-def* **using** ‹*set local.a* ⊆ *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* ^ *l′*
using *length-c* **by** *simp*

have *b-carrier*: *set b* ⊆ *fermat-non-unique-carrier*
unfolding *b-def*
apply (*intro fft-ifft-carrier nums1-carrier*) **using** *length-nums1* **by** *simp*
have *c-carrier*: *set c* ⊆ *fermat-non-unique-carrier*
unfolding *c-def*
apply (*intro fft-ifft-carrier nums2-carrier*) **using** *length-nums2* **by** *simp*

have *length-nums1′*: *length nums1′* = *2* ^ *l′*
using *length-nums1 nums1′-def′* **by** *simp*
have *length-nums2′*: *length nums2′* = *2* ^ *l′*
using *length-nums2 nums2′-def′* **by** *simp*

have *length-g′*: *length g′* = *2* ^ *l′*
unfolding *g′-def* **by** (*simp add*: *length-a length-b′ length-c′*)
have *length-h′*: *length h′* = *2* ^ *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* ^ (*l* − *1*) **unfolding** *length-g′* **using** ‹*l* = *Suc l′*› **by** *simp*

have *bi-carrier*: *b* ! *i* ∈ *fermat-non-unique-carrier*
using *set-subseteqD*[*OF b-carrier*] *length-b i-less* **by** *simp*
have *ci-carrier*: *c* ! *i* ∈ *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-ifft-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*) $\oplus_{Fn}$ *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*) $\oplus_{Fn}$ *to-residue-ring* (*c* ! *i*) $\otimes_{Fn}$ inv_{Fn} *2* $[^]_{Fn}$ (*s* ∗ *i mod 2* ^ (*k* + *1*))
by (*intro arg-cong2*[**where** *f* = ($\oplus_{Fn}$)] *divide-by-power-of-2-correct refl ci-carrier*)
also have ... = (*b′* ! *i*) $\oplus_{Fn}$ (*c′* ! *i*) $\otimes_{Fn}$ inv_{Fn} *2* $[^]_{Fn}$ (*s* ∗ *i mod 2* ^ (*k* + *1*))
unfolding *bi-b′i ci-c′i* **by** (*rule refl*)
also have ... = (*b′* ! *i*) $\oplus_{Fn}$ (*c′* ! *i*) $\otimes_{Fn}$ inv_{Fn} *2* $[^]_{Fn}$ (*s* ∗ *i*)
by (*intro-cong* [*cong-tag-2* ($\oplus_{Fn}$), *cong-tag-2* ($\otimes_{Fn}$)] *more*: *inv-pow-mod-carrier-length mod-mod-trivial*)
also have ... = (*b′* ! *i*) $\oplus_{Fn}$ (*c′* ! *i*) $\otimes_{Fn}$ ((inv_{Fn} *2* $[^]_{Fn}$ *s*) $[^]_{Fn}$ *i*)
by (*intro-cong* [*cong-tag-2* ($\oplus_{Fn}$), *cong-tag-2* ($\otimes_{Fn}$)] *more*: *nat-pow-pow*[*symmetric*] *Units-inv-closed two-is-unit*)
finally have *1*: *map to-residue-ring g* ! *i* = ... **.**
have *g′* ! *i* = *map3* (λ*x y z*. *x* $\oplus_{Fn}$ *y* $\otimes_{Fn}$ *z*) *b′* (*map* (($[^]_{Fn}$) (inv_{Fn} *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*) $\oplus_{Fn}$ ((*map* (($[^]_{Fn}$) (inv_{Fn} *2* $[^]_{Fn}$ *s*)) [*0*..<*length local.a div 2*]) ! *i*) $\otimes_{Fn}$ (*c′* ! *i*)
using *i-less length-b′ length-c′* ‹*l* = *Suc l′*› *length-a* **by** (*intro nth-map3*) *simp-all*

also have ... = $(b' ! i) \oplus_{Fn} (inv_{Fn}\ 2\ [\widehat{\ }]_{Fn}\ s)\ [\widehat{\ }]_{Fn}\ i \otimes_{Fn} (c' ! i)$
apply (*intro-cong* [*cong-tag-2* ($\oplus_{Fn}$), *cong-tag-2* ($\otimes_{Fn}$)])
using *nth-map length-a* ‹*l* = *Suc l'*› *i-less* **by** *simp*
also have ... = $(b' ! i) \oplus_{Fn} (c' ! i) \otimes_{Fn} (inv_{Fn}\ 2\ [\widehat{\ }]_{Fn}\ s)\ [\widehat{\ }]_{Fn}\ i$
apply (*intro arg-cong2*[**where** *f* = ($\oplus_{Fn}$)] *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*) ($s * ([0..<2 \widehat{\ } (l - 1)] ! i)$ *mod* $2 \widehat{\ } (k + 1)$)))
unfolding *g-def fft-ifft-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*) (*multiply-with-power-of-2* (*c* ! *i*) ($s * i$ *mod* $2 \widehat{\ } (k + 1)$)))
using *i-less* **by** *simp*
also have ... = *to-residue-ring* (*b* ! *i*) $\oplus_{Fn}$ *to-residue-ring* (*multiply-with-power-of-2* (*c* ! *i*) ($s * i$ *mod* $2 \widehat{\ } (k + 1)$))
by (*intro add-fermat-correct bi-carrier multiply-with-power-of-2-closed ci-carrier*)
also have ... = *to-residue-ring* (*b* ! *i*) $\oplus_{Fn}$ *to-residue-ring* (*c* ! *i*) $\otimes_{Fn}\ 2\ [\widehat{\ }]_{Fn}$ ($s * i$ *mod* $2 \widehat{\ } (k + 1)$)
by (*intro arg-cong2*[**where** *f* = ($\oplus_{Fn}$)] *multiply-with-power-of-2-correct refl ci-carrier*)
also have ... = $(b' ! i) \oplus_{Fn} (c' ! i) \otimes_{Fn}\ 2\ [\widehat{\ }]_{Fn}$ ($s * i$ *mod* $2 \widehat{\ } (k + 1)$)
unfolding *bi-b'i ci-c'i* **by** (*rule refl*)
also have ... = $(b' ! i) \oplus_{Fn} (c' ! i) \otimes_{Fn}\ 2\ [\widehat{\ }]_{Fn}\ (s * i)$
by (*intro-cong* [*cong-tag-2* ($\oplus_{Fn}$), *cong-tag-2* ($\otimes_{Fn}$)] *more*: *pow-mod-carrier-length mod-mod-trivial*)
also have ... = $(b' ! i) \oplus_{Fn} (c' ! i) \otimes_{Fn} ((2\ [\widehat{\ }]_{Fn}\ s)\ [\widehat{\ }]_{Fn}\ i)$
by (*intro-cong* [*cong-tag-2* ($\oplus_{Fn}$), *cong-tag-2* ($\otimes_{Fn}$)] *more*: *nat-pow-pow*[*symmetric*] *two-in-carrier*)
finally have *1*: *map to-residue-ring g* ! *i* = ... **.**
have *g'* ! *i* = *map3* ($\lambda x\ y\ z.\ x \oplus_{Fn} y \otimes_{Fn} z$) *b'* (*map* (($[\widehat{\ }]_{Fn}$) (2 $[\widehat{\ }]_{Fn}$ *s*)) [*0..<length local.a div 2*]) *c'* ! *i*
unfolding *g'-def if-False map2-of-map2-r* **by** (*simp add*: *False*)
also have ... = $(b' ! i) \oplus_{Fn}$ ((*map* (($[\widehat{\ }]_{Fn}$) (2 $[\widehat{\ }]_{Fn}$ *s*)) [*0..<length local.a div 2*]) ! *i*) $\otimes_{Fn} (c' ! i)$
using *i-less length-b' length-c'* ‹*l* = *Suc l'*› *length-a* **by** (*intro nth-map3*) *simp-all*
also have ... = $(b' ! i) \oplus_{Fn} (2\ [\widehat{\ }]_{Fn}\ s)\ [\widehat{\ }]_{Fn}\ i \otimes_{Fn} (c' ! i)$
apply (*intro-cong* [*cong-tag-2* ($\oplus_{Fn}$), *cong-tag-2* ($\otimes_{Fn}$)])
using *nth-map length-a* ‹*l* = *Suc l'*› *i-less* **by** *simp*

also have ... = $(b' \mathbin{!} i) \oplus_{Fn} (c' \mathbin{!} i) \otimes_{Fn} (2 \,[\urcorner]_{Fn}\, s) \,[\urcorner]_{Fn}\, i$
apply (*intro arg-cong2*[**where** $f = (\oplus_{Fn})$] *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' = $ *map to-residue-ring h*
proof (*intro nth-equalityI*)
show *length* h' = *length* (*map to-residue-ring h*)
by (*simp add*: *length-h length-h'*)
next
fix i
assume $i <$ *length* h'
then have *i-less*: $i < 2 \,\hat{}\,(l - 1)$ **unfolding** *length-h'* **using** ‹l = *Suc* l'› **by** *simp*

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

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

show $h' \mathbin{!} i$ = (*map to-residue-ring h*) ! 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 h* ! i = *to-residue-ring* $(h \mathbin{!} i)$
apply (*intro nth-map*)
unfolding *length-h* **using** *i-less* **by** *simp*
also have ... = *to-residue-ring* (*subtract-fermat* $(b \mathbin{!} i)$ (*divide-by-power-of-2* $(c \mathbin{!} i)$ $(s * ([0..{<}2 \,\hat{}\,(l - 1)] \mathbin{!} i)$ *mod* $2 \,\hat{}\,(k + 1))))$
unfolding *h-def fft-ifft-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* $(b \mathbin{!} i)$ (*divide-by-power-of-2* $(c \mathbin{!} i)$ $(s * i$ *mod* $2 \,\hat{}\,(k + 1))))$
using *i-less* **by** *simp*
also have ... = *to-residue-ring* $(b \mathbin{!} i) \ominus_{Fn}$ *to-residue-ring* (*divide-by-power-of-2* $(c \mathbin{!} i)$ $(s * i$ *mod* $2 \,\hat{}\,(k + 1)))$

by (*intro subtract-fermat-correct bi-carrier divide-by-power-of-2-closed ci-carrier*)
also have ... = *to-residue-ring* (*b* ! *i*) $\ominus_{Fn}$ *to-residue-ring* (*c* ! *i*) $\otimes_{Fn}$ inv_{Fn} *2* $[\urcorner]_{Fn}$ (*s* ∗ *i mod 2* ^ (*k* + *1*))
by (*intro arg-cong2*[**where** *f* = ($\lambda x\ y.\ x \ominus_{Fn} y$)] *divide-by-power-of-2-correct refl ci-carrier*)
also have ... = (*b′* ! *i*) $\ominus_{Fn}$ (*c′* ! *i*) $\otimes_{Fn}$ inv_{Fn} *2* $[\urcorner]_{Fn}$ (*s* ∗ *i mod 2* ^ (*k* + *1*))
unfolding *bi-b′i ci-c′i* **by** (*rule refl*)
also have ... = (*b′* ! *i*) $\ominus_{Fn}$ (*c′* ! *i*) $\otimes_{Fn}$ inv_{Fn} *2* $[\urcorner]_{Fn}$ (*s* ∗ *i*)
by (*intro-cong* [*cong-tag-2* ($\lambda x\ y.\ x \ominus_{Fn} y$), *cong-tag-2* ($\otimes_{Fn}$)] *more*: *inv-pow-mod-carrier-length mod-mod-trivial*)
also have ... = (*b′* ! *i*) $\ominus_{Fn}$ (*c′* ! *i*) $\otimes_{Fn}$ ((inv_{Fn} *2* $[\urcorner]_{Fn}$ *s*) $[\urcorner]_{Fn}$ *i*)
by (*intro-cong* [*cong-tag-2* ($\lambda x\ y.\ x \ominus_{Fn} y$), *cong-tag-2* ($\otimes_{Fn}$)] *more*: *nat-pow-pow*[*symmetric*] *Units-inv-closed two-is-unit*)
finally have *1*: *map to-residue-ring h* ! *i* = ... **.**
have *h′* ! *i* = *map3* ($\lambda x\ y\ z.\ x \ominus_{Fn} y \otimes_{Fn} z$) *b′* (*map* (($[\urcorner]_{Fn}$) ($inv_{Fn}$ *2* $[\urcorner]_{Fn}$ *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*) $\ominus_{Fn}$ ((*map* (($[\urcorner]_{Fn}$) (inv_{Fn} *2* $[\urcorner]_{Fn}$ *s*)) [*0*..<*length local.a div 2*]) ! *i*) $\otimes_{Fn}$ (*c′* ! *i*)
using *i-less length-b′ length-c′* ‹*l* = *Suc l′*› *length-a* **by** (*intro nth-map3*) *simp-all*
also have ... = (*b′* ! *i*) $\ominus_{Fn}$ (inv_{Fn} *2* $[\urcorner]_{Fn}$ *s*) $[\urcorner]_{Fn}$ *i* $\otimes_{Fn}$ (*c′* ! *i*)
apply (*intro-cong* [*cong-tag-2* ($\lambda x\ y.\ x \ominus_{Fn} y$), *cong-tag-2* ($\otimes_{Fn}$)])
using *nth-map length-a* ‹*l* = *Suc l′*› *i-less* **by** *simp*
also have ... = (*b′* ! *i*) $\ominus_{Fn}$ (*c′* ! *i*) $\otimes_{Fn}$ (inv_{Fn} *2* $[\urcorner]_{Fn}$ *s*) $[\urcorner]_{Fn}$ *i*
apply (*intro arg-cong2*[**where** *f* = ($\lambda x\ y.\ x \ominus_{Fn} y$)] *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* ^ (*l* − *1*)] ! *i*) *mod 2* ^ (*k* + *1*))))
unfolding *h-def fft-ifft-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 mod 2* ^ (*k* + *1*))))
using *i-less* **by** *simp*
also have ... = *to-residue-ring* (*b* ! *i*) $\ominus_{Fn}$ *to-residue-ring* (*multiply-with-power-of-2* (*c* ! *i*) (*s* ∗ *i mod 2* ^ (*k* + *1*)))
by (*intro subtract-fermat-correct bi-carrier multiply-with-power-of-2-closed*

ci-carrier)
also have ... = *to-residue-ring* (*b* ! *i*) $\ominus_{Fn}$ *to-residue-ring* (*c* ! *i*) $\otimes_{Fn}$ *2* $[^\frown]_{Fn}$ (*s* $*$ *i mod 2* ^ (*k* + *1*))
by (*intro arg-cong2*[**where** *f* = (λ*x y. x* $\ominus_{Fn}$ *y*)] *multiply-with-power-of-2-correct refl ci-carrier*)
also have ... = (*b'* ! *i*) $\ominus_{Fn}$ (*c'* ! *i*) $\otimes_{Fn}$ *2* $[^\frown]_{Fn}$ (*s* $*$ *i mod 2* ^ (*k* + *1*))
unfolding *bi-b'i ci-c'i* **by** (*rule refl*)
also have ... = (*b'* ! *i*) $\ominus_{Fn}$ (*c'* ! *i*) $\otimes_{Fn}$ *2* $[^\frown]_{Fn}$ (*s* $*$ *i*)
by (*intro-cong* [*cong-tag-2* (λ*x y. x* $\ominus_{Fn}$ *y*), *cong-tag-2* ($\otimes_{Fn}$)] *more*: *pow-mod-carrier-length mod-mod-trivial*)
also have ... = (*b'* ! *i*) $\ominus_{Fn}$ (*c'* ! *i*) $\otimes_{Fn}$ ((*2* $[^\frown]_{Fn}$ *s*) $[^\frown]_{Fn}$ *i*)
by (*intro-cong* [*cong-tag-2* (λ*x y. x* $\ominus_{Fn}$ *y*), *cong-tag-2* ($\otimes_{Fn}$)] *more*: *nat-pow-pow*[*symmetric*] *two-in-carrier*)
finally have *1*: *map to-residue-ring h* ! *i* = ... **.**
have *h'* ! *i* = *map3* (λ*x y z. x* $\ominus_{Fn}$ *y* $\otimes_{Fn}$ *z*) *b'* (*map* (($[^\frown]_{Fn}$) (*2* $[^\frown]_{Fn}$ *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*) $\ominus_{Fn}$ ((*map* (($[^\frown]_{Fn}$) (*2* $[^\frown]_{Fn}$ *s*)) [*0*..<*length local.a div 2*]) ! *i*) $\otimes_{Fn}$ (*c'* ! *i*)
using *i-less length-b' length-c'* ‹*l* = *Suc l'*› *length-a* **by** (*intro nth-map3*) *simp-all*
also have ... = (*b'* ! *i*) $\ominus_{Fn}$ (*2* $[^\frown]_{Fn}$ *s*) $[^\frown]_{Fn}$ *i* $\otimes_{Fn}$ (*c'* ! *i*)
apply (*intro-cong* [*cong-tag-2* (λ*x y. x* $\ominus_{Fn}$ *y*), *cong-tag-2* ($\otimes_{Fn}$)])
using *nth-map length-a* ‹*l* = *Suc l'*› *i-less* **by** *simp*
also have ... = (*b'* ! *i*) $\ominus_{Fn}$ (*c'* ! *i*) $\otimes_{Fn}$ (*2* $[^\frown]_{Fn}$ *s*) $[^\frown]_{Fn}$ *i*
apply (*intro arg-cong2*[**where** *f* = (λ*x y. x* $\ominus_{Fn}$ *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-ifft-correct*:
assumes *length a* = *2* ^ *l*
assumes *s* = *2* ^ *i*
assumes *i* + *l* = *k* + *1*
assumes *l* > *0*
assumes *set a* $\subseteq$ *fermat-non-unique-carrier*
shows *map to-residue-ring* (*fft-ifft it s a*) = *NTT* ((*if it then* inv_{Fn} *2 else 2*) $[^\frown]_{Fn}$ *s*) (*map to-residue-ring a*)
proof −
let *?*μ = (*if it then* inv_{Fn} *2 else 2*) $[^\frown]_{Fn}$ *s*
have *inv2s*: (inv_{Fn} *2* $[^\frown]_{Fn}$ *s*) = inv_{Fn} (*2* $[^\frown]_{Fn}$ *s*)
by (*intro inv-nat-pow*[*symmetric*] *two-is-unit*)

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

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

from *fft-ifft-correct′*[*OF* ‹*length a* = $2\ \widehat{\ }\ l$› ‹$l \leq k + 1$› ‹*set a* ⊆ *fermat-non-unique-carrier*›]
have *map to-residue-ring* (*fft-ifft it s a*) = *FNTT′′* ?μ (*map to-residue-ring a*)
  by *simp*
also have ... = *FNTT′* ?μ (*map to-residue-ring a*)
  apply (*intro FNTT′′-FNTT′*)
  using *assms*(*1*) **by** *simp*
also have ... = *FNTT* ?μ (*map to-residue-ring a*)
  by (*intro FNTT′-FNTT*)
also have ... = *NTT* ?μ (*map to-residue-ring a*)
  apply (*intro FNTT-NTT*[*of* - $2\ \widehat{\ }\ l\ l$]])
  subgoal by (*intro nat-pow-closed two-in-carrier prop-ifI*[**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-ifft-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* ((*inv*$_{Fn}$ *2*) [^]$_{Fn}$ *s*) (*map to-residue-ring a*)
unfolding *ifft.simps* **using** *fft-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*: *time* (*evens-odds-tm b xs*) ≤ *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* :: *nat-lsbf* ⇒ *nat* ⇒ *nat-lsbf tm* **where**
multiply-with-power-of-2-tm xs m =*1* *rotate-right-tm m xs*

lemma *val-multiply-with-power-of-2-tm*[*simp*, *val-simp*]:
  val (*multiply-with-power-of-2-tm xs m*) = *multiply-with-power-of-2 xs m*
  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*:
time (*multiply-with-power-of-2-tm xs m*) ≤ *24* + *26* ∗ *max m* (*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* :: *nat-lsbf* ⇒ *nat* ⇒ *nat-lsbf tm* **where**
divide-by-power-of-2-tm xs m =*1* *rotate-left-tm m xs*

lemma *val-divide-by-power-of-2-tm*[*simp*, *val-simp*]:
  val (*divide-by-power-of-2-tm xs m*) = *divide-by-power-of-2 xs m*
  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*:
time (*divide-by-power-of-2-tm xs m*) ≤ *24* + *26* ∗ *max m* (*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* :: *nat-lsbf* ⇒ *nat-lsbf* ⇒ *nat-lsbf tm* **where**
add-fermat-tm xs ys =*1* *do* {
  zs ← *xs* $+_{nt}$ *ys*;
  lenzs ← *length-tm zs*;
  k1 ← *k* $+_t$ *1*;
  powk ← *2* $\widehat{\ }_t$ *k1*;
  powk1 ← *powk* $+_t$ *1*;
  b ← *lenzs* $=_t$ *powk1*;
  if b then do {
    zsr ← *butlast-tm zs*;
    inc-nat-tm zsr
  } *else return zs*
}

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

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

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* $\hat{}_t$ (*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* ⇒ *nat-lsbf* ⇒ *nat-lsbf tm* **where**
subtract-fermat-tm xs ys =1 do {
powk ← *2* $\hat{}_t$ *k*;
minusy ← *multiply-with-power-of-2-tm ys powk*;
add-fermat-tm xs minusy
}

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

lemma *time-subtract-fermat-tm-le*: *time* (*subtract-fermat-tm xs ys*) ≤
38 + 66 ∗ 2 ^ k + 26 ∗ length ys + 7 ∗ max (*length xs*) (*length ys*)
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* ⇒ *nat-lsbf tm* **where**
reduce-tm xs =1 do {
(*ys*, *zs*) ← *split-tm xs*;
b ← *zs* $\le_{nt}$ *ys*;
if b then ys $-_{nt}$ *zs*
else do {
kpow ← *2* $\hat{}_t$ *k*;
kpow1 ← *kpow* $-_t$ *1*;
zeros ← *replicate-tm kpow1 False*;
a1 ← *zeros* $@_t$ [*True*];
s ← (*True # a1*) $+_{nt}$ *ys*;
s $-_{nt}$ *zs*
}
}

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

lemma *time-reduce-tm-le*: *time* (*reduce-tm xs*) ≤ *155 + 85 ∗ length xs + 46 ∗ 2 ^ k*
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 + 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 ^t 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 ∗ length xs + 16) + (13 ∗ length xs + 23) +
  (if b then (30 ∗ length xs + 48)
  else 12 ∗ 2 ^ k +
    2 +
    2 ^ k +
    2 ^ k +
    (2 ∗ 2 ^ k + 2 ∗ length xs + 5) +
    (30 ∗ 2 ^ k + 60 ∗ 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 ... ≤ 155 + 85 ∗ length xs + 46 ∗ 2 ^ k
    by simp
  finally show ?thesis .
qed

function (domintros) from-nat-lsbf-tm :: nat-lsbf ⇒ nat-lsbf tm where
from-nat-lsbf-tm xs =1 do {
  k1 ← k +_t 1;
  powk ← 2 ^_t k1;
  lenxs ← length-tm xs;
  b ← lenxs ≤_t powk;
  if b then fill-tm powk xs else do {
    xs1 ← take-tm powk xs;
    xs2 ← drop-tm powk xs;
    a ← xs1 +_nt xs2;
    from-nat-lsbf-tm a
  }
}
  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 :: *nat* **where** $e \equiv 2 \mathbin{\char`\^} (k + 1)$
lemma *e-ge-1*: $e \geq 1$ **by** *simp*
lemma *e-ge-2*: $e \geq 2$ **by** *simp*
lemma *e-ge-4*: $k > 0 \Longrightarrow e \geq 4$ **using** *power-increasing*[*of* 2 $k + 1$ 2::*nat*] **by** *simp*

lemma *time-from-nat-lsbf-tm-le-aux*:
assumes $xs' = $ *add-nat* (*take e xs*) (*drop e xs*)
shows *time* (*from-nat-lsbf-tm xs*) $\leq 18 * e + 4 *$ *length xs* $+ 9 +$
(*if length xs* $\leq e$ *then* 0 *else time* (*from-nat-lsbf-tm xs'*))
using *assms* **proof** (*induction xs rule*: *from-nat-lsbf.induct*)
case (*1 xs*)

have *time* (*from-nat-lsbf-tm xs*) = *time* ($k +_t 1$) +
time ($2 \mathbin{\char`\^}_t (k + 1)$) +
time (*length-tm xs*) +
time (*length xs* $\leq_t e$) +
(*if length xs* $\leq e$ *then time* (*fill-tm e xs*)
else time (*take-tm e xs*) +
time (*drop-tm e xs*) +
time (*take e xs* $+_{nt}$ *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 ... $\leq e +$
($12 * e$) +
(*length xs* + 1) +
($2 * e + 2$) +
(*if length xs* $\leq e$ *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 ... = $15 * e +$ *length xs* $+ 4 +$
(*if length xs* $\leq e$ *then* $3 * e + 3 *$ *length xs* $+ 5$
else $2 * e + 2 *$ *length xs* $+ 5 +$

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

lemma *time-from-nat-lsbf-tm-le-aux'*:
assumes *xs'* = *add-nat* (*take e xs*) (*drop e xs*)
shows *time* (*from-nat-lsbf-tm xs*) $\leq$ *66* $*$ *e* + *4* $*$ *length xs* + *35* +
(*if length xs* $\leq$ *e* + *1 then 0 else time* (*from-nat-lsbf-tm xs'*))
proof −
consider *length xs* $\leq$ *e* | *length xs* = *e* + *1* | *length xs* $\geq$ *e* + *2*
by *linarith*
then show *?thesis*
proof *cases*
case *1*
then show *?thesis*
using *time-from-nat-lsbf-tm-le-aux*[*OF assms*] **by** *simp*
next
case *2*
consider (*2-1*) *length xs'* $\leq$ *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-lsbf-tm xs'*) $\leq$ *22* $*$ *e* + *9*
using *time-from-nat-lsbf-tm-le-aux*[*OF refl*, *of xs'*]
by *simp*
then have *time* (*from-nat-lsbf-tm xs*) $\leq$ *44* $*$ *e* + *22*
using *time-from-nat-lsbf-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-lsbf-tm xs''*) $\leq$ *22* $*$ *e* + *9*
using *time-from-nat-lsbf-tm-le-aux*[*OF refl*, *of xs''*] **by** *simp*
then have *time* (*from-nat-lsbf-tm xs'*) $\leq$ *44* $*$ *e* + *22*
using *time-from-nat-lsbf-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** $xs' = add\text{-}nat\ (take\ e\ xs)\ (drop\ e\ xs)$
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 $length\ xs' \le length\ xs - e + 1$
unfolding xs'-*def*
using *length-add-nat-upper*[*of take e xs drop e xs*] **by** *argo*
then have *len-xs'*: $length\ xs' \le l - (e - 1)$ **using** *1.prems e-ge-1 3* **by** *linarith*

have *ih*: $time\ (from\text{-}nat\text{-}lsbf\text{-}tm\ xs') \le time\text{-}from\text{-}nat\text{-}lsbf\text{-}tm\text{-}bound\ (l - (e - 1))$
apply (*intro 1.IH*)
subgoal using *1.prems 3* **by** *linarith*
subgoal by (*fact len-xs'*)
done
then have $time\ (from\text{-}nat\text{-}lsbf\text{-}tm\ xs) \le 18 * e + 4 * length\ xs + 9 + time\text{-}from\text{-}nat\text{-}lsbf\text{-}tm\text{-}bound\ (l - (e - 1))$
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.prems* **by** *simp*
qed
qed

lemma *time-from-nat-lsbf-tm-bound-closed*:
assumes $x \le 2 * e$
assumes $x \ge e + 2$
shows $time\text{-}from\text{-}nat\text{-}lsbf\text{-}tm\text{-}bound\ (x + l * (e - 1)) = (l + 1) * (88 * e + 4 * x + 48) + 4 * (\sum\ \{0..l\}) * (e - 1)$
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 $time\text{-}from\text{-}nat\text{-}lsbf\text{-}tm\text{-}bound\ (x + Suc\ l * (e - 1)) = 88 * e + 4 * (x + Suc\ l * (e - 1)) + 48 + time\text{-}from\text{-}nat\text{-}lsbf\text{-}tm\text{-}bound\ (x + Suc\ l * (e - 1) - (e - 1))$ (**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 = time-from-nat-lsbf-tm-bound (x + l ∗ (e − 1))
    apply (intro arg-cong[where f = time-from-nat-lsbf-tm-bound])
    using assms by simp
  also have ... = (l + 1) ∗ (88 ∗ e + 4 ∗ x + 48) + 4 ∗ (∑ {0..l}) ∗ (e − 1)
    (is ... = ?r′)
    by (rule Suc.IH)
  also have ?t + ?r′ = (Suc l + 1) ∗ (88 ∗ e + 4 ∗ x + 48) + 4 ∗ ∑ {0..Suc
l} ∗ (e − 1)
    by (simp add: add-mult-distrib)
  finally show ?case .
qed

lemma time-from-nat-lsbf-tm-le:
  assumes e ≥ 4
  assumes length xs ≤ c ∗ e
  shows time (from-nat-lsbf-tm xs) ≤ (288 ∗ c + 144) + (96 + 192 ∗ c + 8 ∗ c
∗ c) ∗ e
proof (cases length xs ≤ 2 ∗ e)
  case True
  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 ... = 88 ∗ e + 4 ∗ length xs + 48
    unfolding time-from-nat-lsbf-tm-bound.simps[of length xs]
    using True by simp
  also have ... ≤ 96 ∗ e + 48
    using True by auto
  also have ... ≤ (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′ = length xs mod (e − 1)
  define y′ where y′ = length xs div (e − 1)
  from x′-def y′-def have len-xs′: length xs = y′ ∗ (e − 1) + x′ by presburger
  from False have length xs ≥ 2 ∗ (e − 1) by simp
  then have y′ ≥ 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 = (if x′ ≤ 2 then x′ + 2 ∗ (e − 1) else x′ + (e − 1))
  define y where y = (if x′ ≤ 2 then y′ − 2 else y′ − 1)
  have len-xs: length xs = x + y ∗ (e − 1)
    unfolding len-xs′
    apply (cases x′ ≤ 2)
   subgoal unfolding x-def y-def using ‹y′ ≥ 2› by (simp add: diff-mult-distrib)
    subgoal premises prems
    proof −
      have y′ ∗ (e − 1) + x′ = (y′ − 1 + 1) ∗ (e − 1) + x′
        using ‹y′ ≥ 2› 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 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 ⋀*c*. *c* ∈ *set cs* ⟹ *length c* = *e*
assumes ⋀*xs ys*. *time* (*f xs ys*) ≤ *38* + *66* ∗ *2* ^ *k* + *26* ∗ *length ys* + *7* ∗ *max* (*length xs*) (*length ys*)
assumes *s* < *e*
assumes *g* = *multiply-with-power-of-2-tm* ∨ *g* = *divide-by-power-of-2-tm*
shows *time* (*fft-combine-b-c-aux-tm f g l* (*revs*, *s*) *bs cs*) ≤ *length revs* + *3* + *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 *time* (*fft-combine-b-c-aux-tm f g l* (*revs*, *s*) [] *cs*) = *length revs* + *3* **by** *simp*
then show *?case* **by** *simp*
next
case (*Cons b bs*)
from *Cons.prems* **have** *len-b*: *length b* = *e* **by** *simp*
from *Cons.prems*(*1*) **obtain** *c cs'* **where** *cs* = *c* # *cs'* **by** (*metis length-Suc-conv*)
with *Cons.prems* **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'*) ≤
length revs' + *3* + *length bs* ∗ (*72* + *116* ∗ *e* + *8* ∗ *l*)
if *s'* < *e* **for** *revs' s'*
apply (*intro Cons.IH*)
subgoal using *Cons.prems* ‹*cs* = *c* # *cs'*› **by** *simp*
subgoal using *Cons.prems* **by** *simp*
subgoal using *Cons.prems* ‹*cs* = *c* # *cs'*› **by** *simp*
subgoal by (*rule Cons.prems*)
subgoal by (*fact that*)
subgoal by (*rule Cons.prems*)
done
have *val-gcs*: *val* (*g c s*) = *multiply-with-power-of-2 c s* ∨ *val* (*g c s*) = *divide-by-power-of-2 c s*
using *Cons.prems* **by** *fastforce*
from ‹*cs* = *c* # *cs'*› **have** *time* (*fft-combine-b-c-aux-tm f g l* (*revs*, *s*) (*b* # *bs*) *cs*) =
time (*g c s*) +
time (*f b* (*val* (*g c s*))) +
time (*2* $\widehat{\ }_t$ (*k* + *1*)) +
time ((*s* + *l*) mod_t *e*) +
time (*fft-combine-b-c-aux-tm f g l* (*val* (*f b* (*val* (*g c s*))) # *revs*, (*s* + *l*) *mod e*) *bs cs'*) +
s + *k* + *3*
by (*simp del*: *One-nat-def*)
also have ... ≤
(*24* + *26* ∗ *max s* (*length c*)) +
(*38* + *33* ∗ *e* + *26* ∗ *length c* + *7* ∗ *max* (*length b*) (*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.prems* **by** *fastforce*
subgoal
using *val-gcs Cons.prems*(*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-ifft-combine-b-c-add-tm* :: *bool* ⇒ *nat* ⇒ *nat-lsbf list* ⇒ *nat-lsbf list* ⇒ *nat-lsbf list tm* **where**
fft-ifft-combine-b-c-add-tm True l bs cs =*1 fft-combine-b-c-aux-tm add-fermat-tm divide-by-power-of-2-tm l* ([], *0*) *bs cs*
| *fft-ifft-combine-b-c-add-tm False l bs cs* =*1 fft-combine-b-c-aux-tm add-fermat-tm multiply-with-power-of-2-tm l* ([], *0*) *bs cs*

fun *fft-ifft-combine-b-c-subtract-tm* :: *bool* ⇒ *nat* ⇒ *nat-lsbf list* ⇒ *nat-lsbf list* ⇒ *nat-lsbf list tm* **where**
fft-ifft-combine-b-c-subtract-tm True l bs cs =*1 fft-combine-b-c-aux-tm subtract-fermat-tm divide-by-power-of-2-tm l* ([], *0*) *bs cs*
| *fft-ifft-combine-b-c-subtract-tm False l bs cs* =*1 fft-combine-b-c-aux-tm subtract-fermat-tm multiply-with-power-of-2-tm l* ([], *0*) *bs cs*

lemma *val-fft-ifft-combine-b-c-add-tm*[*simp*, *val-simp*]:
assumes *length bs* = *length cs*
shows *val* (*fft-ifft-combine-b-c-add-tm it l bs cs*) = *fft-ifft-combine-b-c-add it l bs cs*
by (*cases it*; *simp add*: *assms*)

lemma *val-fft-ifft-combine-b-c-subtract-tm*[*simp*, *val-simp*]:
  assumes *length bs* = *length cs*
  shows *val* (*fft-ifft-combine-b-c-subtract-tm it l bs cs*) = *fft-ifft-combine-b-c-subtract it l bs cs*
  by (*cases it*; *simp add*: *assms*)

lemma *time-fft-combine-b-c-add-tm-le*:
  assumes *length bs* = *length cs*
  assumes $\bigwedge b.\ b \in set\ bs \Longrightarrow length\ b = e$
  assumes $\bigwedge c.\ c \in set\ cs \Longrightarrow length\ c = e$
  shows *time* (*fft-ifft-combine-b-c-add-tm it l bs cs*) ≤ *4* + *length bs* ∗ (*72* + *116* ∗ *e* + *8* ∗ *l*)
proof −
  have
    time (*fft-combine-b-c-aux-tm add-fermat-tm g l* ([], *0*) *bs cs*)
      ≤ *length* ([] :: *nat-lsbf list*) + *3* + *length bs* ∗ (*72* + *116* ∗ *e* + *8* ∗ *l*)
    if *g* = *multiply-with-power-of-2-tm* ∨ *g* = *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 *length bs* = *length cs*
  assumes $\bigwedge b.\ b \in set\ bs \Longrightarrow length\ b = e$
  assumes $\bigwedge c.\ c \in set\ cs \Longrightarrow length\ c = e$
  shows *time* (*fft-ifft-combine-b-c-subtract-tm it l bs cs*) ≤ *4* + *length bs* ∗ (*72* + *116* ∗ *e* + *8* ∗ *l*)
proof −
  have
    time (*fft-combine-b-c-aux-tm subtract-fermat-tm g l* ([], *0*) *bs cs*)
      ≤ *length* ([] :: *nat-lsbf list*) + *3* + *length bs* ∗ (*72* + *116* ∗ *e* + *8* ∗ *l*)
    if *g* = *multiply-with-power-of-2-tm* ∨ *g* = *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-ifft-tm where
fft-ifft-tm it l [] =1 return []
| fft-ifft-tm it l [x] =1 return [x]
| fft-ifft-tm it l [x, y] =1 do {
    r1 ← add-fermat-tm x y;
    r2 ← subtract-fermat-tm x y;
    return [r1, r2]
  }
| fft-ifft-tm it l a =1 do {
    nums1 ← evens-odds-tm True a;
    nums2 ← evens-odds-tm False a;
    b ← fft-ifft-tm it (2 ∗ l) nums1;
    c ← fft-ifft-tm it (2 ∗ l) nums2;
    g ← fft-ifft-combine-b-c-add-tm it l b c;
    h ← fft-ifft-combine-b-c-subtract-tm it l b c;
    g @t h
  }

lemma val-fft-ifft-tm[simp, val-simp]: length a = 2 ^ m ⟹ val (fft-ifft-tm it l a)
= fft-ifft it l a
proof (induction it l a arbitrary: m rule: fft-ifft.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: length (evens-odds b local.a) = 2 ^ 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: length (fft-ifft it (2 ∗ l) (evens-odds True local.a)) = length (fft-ifft
it (2 ∗ l) (evens-odds False local.a))
    using length-fft-ifft[OF len-eo] by simp
```

have *ih1*: *val* (*fft-ifft-tm it* (*2* ∗ *l*) (*evens-odds True local.a*)) = *fft-ifft 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-ifft-tm it* (*2* ∗ *l*) (*evens-odds False local.a*)) = *fft-ifft 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-ifft-tm.simps fft-ifft.simps* **unfolding** *a-def*[*symmetric*]
unfolding *Let-def val-simp ih1 ih2*
unfolding *val-fft-ifft-combine-b-c-add-tm*[*OF len-eq*] *val-fft-ifft-combine-b-c-subtract-tm*[*OF len-eq*]
by (*rule refl*)
qed

lemma *time-fft-ifft-tm-le-aux*:
assumes $\bigwedge x.\ x \in set\ a \Longrightarrow length\ x = e$
assumes $length\ a = 2\ \widehat{}\ m$
shows $time\ (fft\text{-}ifft\text{-}tm\ it\ l\ a) \leq 2\ \widehat{}\ (m - 1) * (52 + 87 * e) + (m - 1) * 2\ \widehat{}\ m * (76 + 116 * e) + (\sum i \leftarrow [0..<m-1].\ 2\ \widehat{}\ i) * (8 * 2\ \widehat{}\ m * l + 13)$
using *assms* **proof** (*induction it l a arbitrary*: *m rule*: *fft-ifft.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 $time\ (fft\text{-}ifft\text{-}tm\ it\ l\ [x, y]) \leq 52 + 87 * e$
unfolding *fft-ifft-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 ... $\leq 2\ \widehat{}\ (m - Suc\ 0) * (52 + 87 * e)$ **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** $m = Suc\ (Suc\ m')$ **using** *nat-le-iff-add e-ge-2* **by** *auto*
then have $Suc\ m' = m - 1$ **by** *simp*
have *len-eo*: $length\ (evens\text{-}odds\ b\ local.a) = 2\ \widehat{}\ Suc\ m'$ **for** *b*
apply (*cases b*)
subgoal using *length-evens-odds*(*1*)[*of local.a*] *length-a* ‹$m = Suc\ (Suc\ m')$›
by *simp*
subgoal using *length-evens-odds*(*2*)[*of local.a*] *length-a* ‹$m = Suc\ (Suc\ m')$›
by *simp*
done
have *len-eo-nth*: $length\ x = e$ **if** $x \in set\ (evens\text{-}odds\ b\ local.a)$ **for** *b x*

using *set-evens-odds*[*of b local.a*] *that 4.prems* **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-ifft-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-ifft-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-ifft-tm it* (*2* ∗ *l*) *nums1*) = *fft-ifft it* (*2* ∗ *l*) *nums1*
apply (*intro val-fft-ifft-tm*[*of - Suc m′*])
unfolding *nums1-def* **by** (*rule len-eo*)
have *val-fft2*: *val* (*fft-ifft-tm it* (*2* ∗ *l*) *nums2*) = *fft-ifft it* (*2* ∗ *l*) *nums2*
apply (*intro val-fft-ifft-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-ifft-combine-b-c-add-tm it l b c*) = *fft-ifft-combine-b-c-add it l b c*
by (*rule val-fft-ifft-combine-b-c-add-tm*[*OF len-bc*])
have *val-sub*: *val* (*fft-ifft-combine-b-c-subtract-tm it l b c*) = *fft-ifft-combine-b-c-subtract it l b c*
by (*rule val-fft-ifft-combine-b-c-subtract-tm*[*OF len-bc*])

have *nums-carrier*: *set nums1* ⊆ *fermat-non-unique-carrier set nums2* ⊆ *fermat-non-unique-carrier*
using *4.prems* **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-ifft-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-ifft-carrier nums-carrier*) **unfolding** *nums2-def len-eo* **by** *fast*

have *time* (*fft-ifft-tm it l* (*a1* # *a2* # *a3* # *as*)) =
time (*evens-odds-tm True local.a*) +
time (*evens-odds-tm False local.a*) +
time (*fft-ifft-tm it* (*2* ∗ *l*) *nums1*) +
time (*fft-ifft-tm it* (*2* ∗ *l*) *nums2*) +

```
      time (fft-ifft-combine-b-c-add-tm it l b c) +
      time (fft-ifft-combine-b-c-subtract-tm it l b c) +
      time (g @t h) + 1
    unfolding fft-ifft-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 ... = $2 \hat{} (m - 1) * (52 + 87 * e) +$
$(Suc\ m' - 1) * 2 \hat{} m * (76 + 116 * e) +$
$2 * (\sum i \leftarrow [0..<Suc\ m' - 1].\ 2 \hat{} i) * (8 * 2 \hat{} m * l + 13) +$
$2 \hat{} m * (76 + 116 * e) + 8 * 2 \hat{} m * l + 13$
apply (*intro arg-cong2*[**where** $f = (+)$] *refl*)
subgoal unfolding ‹$m = Suc\ (Suc\ m')$› **by** *simp*
subgoal unfolding ‹$m = Suc\ (Suc\ m')$› **by** *simp*
subgoal unfolding ‹$m = Suc\ (Suc\ m')$› **by** *simp*
done
also have ... = $2 \hat{} (m - 1) * (52 + 87 * e) +$
$((Suc\ m' - 1) + 1) * 2 \hat{} m * (76 + 116 * e) +$
$(2 * (\sum i \leftarrow [0..<Suc\ m' - 1].\ 2 \hat{} i) + 1) * (8 * 2 \hat{} m * l + 13)$
by (*simp add*: *add-mult-distrib*)
also have ... = $2 \hat{} (m - 1) * (52 + 87 * e) +$
$(m - 1) * 2 \hat{} m * (76 + 116 * e) +$
$(\sum i \leftarrow [0..<Suc\ m'].\ 2 \hat{} i) * (8 * 2 \hat{} m * l + 13)$
apply (*intro arg-cong2*[**where** $f = (+)$] *arg-cong2*[**where** $f = (*)$] *refl*)
subgoal unfolding ‹$m = Suc\ (Suc\ m')$› **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** ‹$Suc\ m' = m - 1$› **.**
qed

lemma *time-fft-ifft-tm-le*:
assumes $\bigwedge x.\ x \in set\ a \Longrightarrow length\ x = e$
assumes $length\ a = 2 \hat{} m$
shows $time\ (fft\text{-}ifft\text{-}tm\ it\ l\ a) \le 2 \hat{} m * (65 + 87 * e) + m * 2 \hat{} m * (76 + 116 * e) + (8 * l) * 2 \hat{} (2 * m)$
proof –
from *time-fft-ifft-tm-le-aux*[*OF assms*]
have $time\ (fft\text{-}ifft\text{-}tm\ it\ l\ a) \le 2 \hat{} (m - 1) * (52 + 87 * e) + (m - 1) * 2 \hat{} m * (76 + 116 * e) + (\sum i \leftarrow [0..<m-1].\ 2 \hat{} i) * (8 * 2 \hat{} m * l + 13)$
by *simp*
also have ... $\le 2 \hat{} m * (52 + 87 * e) + m * 2 \hat{} m * (76 + 116 * e) + (2 \hat{} (m - 1) - 1) * (8 * 2 \hat{} m * l + 13)$
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 ... $\le 2 \hat{} m * (52 + 87 * e) + m * 2 \hat{} m * (76 + 116 * e) + 2 \hat{} m * (8 * 2 \hat{} m * l + 13)$
apply (*intro add-mono mult-le-mono order.refl*)
by (*meson diff-le-self le-trans one-le-numeral power-increasing*)
also have ... = $2 \hat{} m * (65 + 87 * e) + m * 2 \hat{} m * (76 + 116 * e) + (8 * l) * 2 \hat{} (2 * m)$

by (*simp add*: *add-mult-distrib2 power-add*[*symmetric*])
finally show *?thesis* **.**
qed

fun *fft-tm* **where**
fft-tm l a =1 fft-ifft-tm False l a
fun *ifft-tm* **where**
ifft-tm l a =1 fft-ifft-tm True l a

lemma *val-fft-tm*[*simp*, *val-simp*]: $length\ a = 2 \ \hat{}\ m \Longrightarrow val\ (fft\text{-}tm\ l\ a) = fft\ l\ a$
by *simp*
lemma *val-ifft-tm*[*simp*, *val-simp*]: $length\ a = 2 \ \hat{}\ m \Longrightarrow val\ (ifft\text{-}tm\ l\ a) = ifft\ l\ a$
by *simp*

lemma *time-fft-tm-le*:
assumes $\bigwedge x.\ x \in set\ a \Longrightarrow length\ x = e$
assumes $length\ a = 2 \ \hat{}\ m$
shows $time\ (fft\text{-}tm\ l\ a) \leq 2 \ \hat{}\ m * (66 + 87 * e) + m * 2 \ \hat{}\ m * (76 + 116 * e) + (8 * l) * 2 \ \hat{}\ (2 * m)$
proof −
have $time\ (fft\text{-}tm\ l\ a) = 1 + time\ (fft\text{-}ifft\text{-}tm\ False\ l\ a)$
by *simp*
also have ... $\leq 1 + (2 \ \hat{}\ m * (65 + 87 * e) + m * 2 \ \hat{}\ m * (76 + 116 * e) + (8 * l) * 2 \ \hat{}\ (2 * m))$
by (*intro add-mono order.refl time-fft-ifft-tm-le assms*; *assumption*)
also have ... $\leq 2 \ \hat{}\ m + (2 \ \hat{}\ m * (65 + 87 * e) + m * 2 \ \hat{}\ m * (76 + 116 * e) + (8 * l) * 2 \ \hat{}\ (2 * m))$
by (*intro add-mono order.refl*; *simp*)
finally show *?thesis* **by** (*simp add*: *algebra-simps*)
qed

lemma *time-ifft-tm-le*:
assumes $\bigwedge x.\ x \in set\ a \Longrightarrow length\ x = e$
assumes $length\ a = 2 \ \hat{}\ m$
shows $time\ (ifft\text{-}tm\ l\ a) \leq 2 \ \hat{}\ m * (66 + 87 * e) + m * 2 \ \hat{}\ m * (76 + 116 * e) + (8 * l) * 2 \ \hat{}\ (2 * m)$
proof −
have $time\ (ifft\text{-}tm\ l\ a) = 1 + time\ (fft\text{-}ifft\text{-}tm\ True\ l\ a)$
by *simp*
also have ... $\leq 1 + (2 \ \hat{}\ m * (65 + 87 * e) + m * 2 \ \hat{}\ m * (76 + 116 * e) + (8 * l) * 2 \ \hat{}\ (2 * m))$
by (*intro add-mono order.refl time-fft-ifft-tm-le assms*; *assumption*)
also have ... $\leq 2 \ \hat{}\ m + (2 \ \hat{}\ m * (65 + 87 * e) + m * 2 \ \hat{}\ m * (76 + 116 * e) + (8 * l) * 2 \ \hat{}\ (2 * m))$
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 ⟹ 2 ^ (2 ∗ n − 1) > n + 1 + 2 ^ n
proof −
  have 1: ⋀k. 2 ^ (2 ∗ (k + 2) − 1) > (k + 2) + 1 + 2 ^ (k + 2)
    subgoal for k
      by (induction k) simp-all
    done
  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 aux-ineq-2: n > 2 ⟹ 2 ^ (2 ∗ n − 2) > n + 2 ^ n
proof −
  have 1: ⋀k. 2 ^ (2 ∗ (k + 3) − 2) ≥ (k + 3) + 2 ^ (k + 3) + 1
    subgoal for k
    proof (induction k)
      case (Suc k)
      have 2 ^ (Suc k + 3) ≥ Suc k + 3 by simp
      then have 4 ∗ k + 16 + 2 ^ (Suc k + 3) ≥ (Suc k + 3) + 1
        by simp
       then have (Suc k + 3) + 2 ^ (Suc k + 3) + 1 ≤ 4 ∗ k + 16 + 2 ∗ 2 ^
(Suc k + 3)
        by simp
      also have ... = 4 ∗ k + 4 ∗ 4 + 2 ∗ 2 ^ (Suc (k + 3)) by simp
      also have ... = 4 ∗ k + 4 ∗ 4 + 2 ∗ 2 ∗ 2 ^ (k + 3)
        apply (intro arg-cong2[where f = (+)] refl)
```

using *power-Suc mult.assoc* **by** *metis*
also have ... = $4 * (k + 3 + 2 \hat{} (k + 3) + 1)$ **by** *simp*
also have ... $\leq 4 * 2 \hat{} (2 * (k + 3) - 2)$ **using** *Suc.IH* **by** *simp*
also have ... = $2 \hat{} ((2 * (k + 3)) - 2 + 2)$ **by** (*simp add*: *power-add*)
also have ... = $2 \hat{} (2 * (Suc\ k + 3) - 2)$ **by** *simp*
finally show *?case* **.**
qed *simp*
done
assume $n > 2$
then have $n \geq 3$ **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*: $n > 1 \Longrightarrow 2 \hat{} n \geq n + 2$
proof –
have *1*: $\bigwedge k.\ 2 \hat{} (k + 2) \geq (k + 2) + 2$
subgoal for *k*
by (*induction k*) *simp-all*
done
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*
apply (*induction x*)
subgoal by (*simp add*: *zero-cong*)
subgoal for *x* **by** (*simp add*: *res-add-eq one-cong mod-add-eq add.commute*)
done

lemma (**in** *residues*) *carrier-mod-eq*: $x \in carrier\ R \Longrightarrow x\ mod\ m = x$
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*: $\neg\ m < 3$
begin

Lemmas in *nat* resp. *int*.

lemma *m-gt-0*: $m > 0$ **using** *m-ge-3* **by** *simp*

definition *n* :: *nat* **where**
$n \equiv (if\ odd\ m\ then\ (m + 1)\ div\ 2\ else\ (m + 2)\ div\ 2)$

definition *oe-n* :: *nat* **where**
oe-n $\equiv$ (*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*: *even m* $\Longrightarrow n \geq 3$ **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*::*nat*) ^ *oe-n div 2* = *2* ^ (*oe-n* − *1*)
  by (*simp add*: *Suc-leI power-diff oe-n-gt-0*)
lemma *two-pow-oe-n-as-halves*: (*2*::*nat*) ^ *oe-n* = *2* ^ (*oe-n* − *1*) + *2* ^ (*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*::*nat*) ^ (*oe-n* + *1*) = *4* ∗ *2* ^ (*oe-n* − *1*)
  using *oe-n-gt-0* **by** (*simp add*: *power-eq-if*)

lemma *index-intros*:
  fixes *i* :: *nat*
  assumes *i* < *2* ^ (*oe-n* − *1*)
  shows *i* < *2* ^ *oe-n* *2* ^ (*oe-n* − *1*) + *i* < *2* ^ *oe-n*
  using *assms two-pow-oe-n-as-halves* **by** *simp-all*

lemma *index-decomp*:
  assumes *j* < (*2*::*nat*) ^ (*oe-n* + *1*)
  shows
    j div 2 ^ (*oe-n* − *1*) < *4*
    j mod 2 ^ (*oe-n* − *1*) < *2* ^ (*oe-n* − *1*)
    j = (*j div 2* ^ (*oe-n* − *1*)) ∗ *2* ^ (*oe-n* − *1*) + (*j mod 2* ^ (*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* :: *nat*
  assumes *i* < *4* *j* < *2* ^ (*oe-n* − *1*)
  shows
    i ∗ *2* ^ (*oe-n* − *1*) + *j* < *2* ^ (*oe-n* + *1*)
    (*i* ∗ *2* ^ (*oe-n* − *1*) + *j*) *div 2* ^ (*oe-n* − *1*) = *i*
    (*i* ∗ *2* ^ (*oe-n* − *1*) + *j*) *mod 2* ^ (*oe-n* − *1*) = *j*
proof −

```
  from assms have i ≤ 3 by simp
  then have i ∗ 2 ^ (oe-n − 1) + j < 3 ∗ 2 ^ (oe-n − 1) + 2 ^ (oe-n − 1)
    using ‹j < 2 ^ (oe-n − 1)›
    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) div 2 ^ (oe-n − 1) = i
    using assms by simp
  show (i ∗ 2 ^ (oe-n − 1) + j) mod 2 ^ (oe-n − 1) = j
    using assms by simp
qed

lemma mn:
  odd m ⟹ m = 2 ∗ n − 1
  even m ⟹ 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) ≤ 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 ... = 2 ^ 2 ∗ 2 ^ (2 ∗ n − 2) by simp
  also have ... ≤ 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* ≤ *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*) ≤ *2* ^ *2* ^ *m*
  by (*intro power-increasing oe-n-m-bound-1*) *simp*
lemma *two-pow-oe-n-m-bound-0-int*: *2* ^ (*oe-n* + *2* ^ *n*) < *int-lsbf-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* ^ (*oe-n* + *1* + *2* ^ *n*) < *int-lsbf-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* ^ *n* ≤ *2* ^ (*n* + *1*)
proof −
  have *oe-n* + *1* + *2* ^ *n* ≤ *n* + *2* + *2* ^ *n* **unfolding** *oe-n-def* **by** *simp*
  also have ... ≤ *2* ^ *n* + *2* ^ *n*
    by (*intro add-mono order.refl aux-ineq-3 n-gt-1*)
  also have ... = *2* ^ (*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-lsbf-mod.n* (*n* + *2*))
definition *Fn* **where** *Fn* = *residue-ring* (*int-lsbf-fermat.n n*)
definition *Fm* **where** *Fm* = *residue-ring* (*int-lsbf-fermat.n m*)

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

sublocale *Znr* : *int-lsbf-mod n* + *2*
  rewrites *Znr.Zn* ≡ *Zn*
proof −
  show *int-lsbf-mod* (*n* + *2*) **by** *unfold-locales simp*
  then interpret *A* : *int-lsbf-mod n* + *2* **.**
  show *A.Zn* ≡ *Zn* **unfolding** *Zn-def A.Zn-def* **.**
qed

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

sublocale *Fnr* : *int-lsbf-fermat n*
  rewrites *Fnr.Fn* ≡ *Fn*
  subgoal unfolding *int-lsbf-fermat.Fn-def Fn-def* **.**
  done

sublocale *Fnr-M* : *multiplicative-subgroup Fn Units Fn units-of Fn*
  by (*rule Fnr.units-subgroup*)

sublocale *Fmr* : *int-lsbf-fermat m*
  rewrites *Fmr.Fn* ≡ *Fm*

```
  subgoal unfolding int-lsbf-fermat.Fn-def Fm-def .
  done

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 ^ oe-n) (2 [^]Fm (2::nat) ^(n − 1))
  apply (intro Fmr.two-powers-primitive-root)
  subgoal using m1 by argo
  subgoal using n-lt-m by simp
  done
lemma two-pow-oe-n-root-of-unity-Fm:
  Fmr.root-of-unity (2 ^ oe-n) (2 [^]Fm (2::nat) ^(n − 1))
  using two-pow-oe-n-primitive-root-Fm by simp

lemma four-prim-root-Fn: Fnr.primitive-root (2 ^ n) (2 [^]Fn (2::nat))
  using Fnr.primitive-root-recursion[OF - Fnr.two-is-primitive-root] by simp

lemma two-oe-n: 2 [^]Fn oe-n = 2 ^ oe-n
proof −
  have 2 ^ n ≥ n + 1 using aux-ineq-3[OF n-gt-1] by simp
  then have 2 ^ n ≥ oe-n unfolding oe-n-def by simp
  then have (2::int) ^ oe-n ≤ 2 ^ 2 ^ n by simp
  then have two-oe-n-mod-Fn: 2 ^ oe-n mod int Fnr.n = 2 ^ oe-n
    using zle-iff-zadd by auto
  then show ?thesis unfolding Fnr.pow-nat-eq .
qed
lemma two-oe-n-Units-Fn: 2 ^ oe-n ∈ 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 ^ oe-n ∈ 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 μ where μ = 2 [^]Fn prim-root-exponent

lemma μ-Units-Fn: μ ∈ Units Fn
  unfolding μ-def by (intro Fnr.Units-pow-closed Fnr.two-is-unit)
lemma μ-carrier-Fn: μ ∈ carrier Fn
  by (intro Fnr.Units-closed μ-Units-Fn)

lemma μ-prim-root: Fnr.primitive-root (2 ^ oe-n) μ
proof (cases odd m)
  case True
  then show ?thesis unfolding oe-n-def μ-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* μ*-def prim-root-exponent-def*
using *four-prim-root-Fn* **by** *simp*
qed
lemma μ*-root-of-unity*: *Fnr.root-of-unity* (*2* ^ *oe-n*) μ
using μ*-prim-root* **by** *simp*
lemma μ*-halfway-property*: μ $[\widehat{\ }]_{Fn}$ ((*2*::*nat*) ^ *oe-n 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 μ $[\widehat{\ }]_{Fn}$ ((*2*::*nat*) ^ *oe-n div 2*) = *2* $[\widehat{\ }]_{Fn}$ ((*2*::*nat*) ^ *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* :: *nat*
  assumes *i* < *2* ^ *oe-n*
  shows *Nat-LSBF.to-nat* (*num-blocks* ! *i*) < *2* ^ *2* ^ (*n* − *1*)
  using *length-nth-num-blocks*[*OF assms*] *to-nat-length-bound* **by** *metis*
lemma *num-blocks-carrier-Fm*[*simp*]:
  fixes *i* :: *nat*
  assumes *i* < *2* ^ *oe-n*
  shows *int* (*Nat-LSBF.to-nat* (*num-blocks* ! *i*)) ∈ *carrier Fm*
  unfolding *Fmr.res-carrier-eq atLeastAtMost-iff*
proof (*intro conjI*)
  show *0* ≤ *int* (*Nat-LSBF.to-nat* (*num-blocks* ! *i*)) **by** *simp*
  have *int* (*Nat-LSBF.to-nat* (*num-blocks* ! *i*)) < *2* ^ *2* ^ (*n* − *1*)
    using *num-blocks-bound*[*OF assms*] **by** *simp*
  also have ... < *2* ^ *2* ^ *m* **using** *n-lt-m* **by** *simp*
  finally show *int* (*Nat-LSBF.to-nat* (*num-blocks* ! *i*)) ≤ *int* (*2* ^ *2* ^ *m* + *1*) − *1* **by** *simp*
qed

lemma *length-num-blocks-carrier*[*simp*]: *length num-blocks-carrier* = *2* ^ *oe-n*
  unfolding *num-blocks-carrier-def* **by** *simp*

lemma *to-res-num*: *Fmr.to-residue-ring num* = ($\bigoplus_{Fm}$*j* ← [*0*..<*2* ^ *oe-n*].
      map (*int* ∘ *Nat-LSBF.to-nat*) *num-blocks* ! *j* $\otimes_{Fm}$ ((*2* $[\urcorner]_{Fm}$ ((*2*::*nat*) ^ (*n* − *1*))) $[\urcorner]_{Fm}$ *j*))
proof −
  let *?m* = *int Fmr.n*
  have ($\bigoplus_{Fm}$*j* ← [*0*..<*2* ^ *oe-n*].
      map (*int* ∘ *Nat-LSBF.to-nat*) *num-blocks* ! *j* $\otimes_{Fm}$ ((*2* $[\urcorner]_{Fm}$ ((*2*::*nat*) ^ (*n* − *1*))) $[\urcorner]_{Fm}$ *j*)) =
    ($\bigoplus_{Fm}$*j* ← [*0*..<*2* ^ *oe-n*].
     map (*int* ∘ *Nat-LSBF.to-nat*) *num-blocks* ! *j* $\otimes_{Fm}$ (*2* $[\urcorner]_{Fm}$ (*j* ∗ (*2*::*nat*) ^ (*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* = ($[\urcorner]_{Fm}$)] *refl mult.commute*)
  also have ... = (∑*j*←[*0*..<*2* ^ *oe-n*].
      map (*int* ∘ *Nat-LSBF.to-nat*) *num-blocks* ! *j* ∗ (*2* ^ (*j* ∗ *2* ^ (*n* − *1*)) *mod ?m*) *mod ?m*) *mod ?m*
    unfolding *Fmr.monoid-sum-list-eq-sum-list Fmr.res-mult-eq Fmr.pow-nat-eq* **by** (*rule refl*)

  also have ... = (∑*j* ← [*0*..<*2* ^ *oe-n*].

(*map* (*int* ∘ *Nat-LSBF.to-nat*) *num-blocks* ! *j* ∗ *2* ^ (*j* ∗ *2* ^ (*n* − *1*)))) *mod* *?m*
by (*simp only*: *mod-mult-right-eq sum-list-mod*)
also have ... = (∑ *j* ← [*0*..<*2* ^ *oe-n*].
(*int* (*Nat-LSBF.to-nat* (*num-blocks* ! *j*)) ∗ (*2* ^ (*j* ∗ *2* ^ (*n* − *1*))))) *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* = λ*i*. *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 assms* **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* (λ*j*. *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 = ($\sum$ *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* = ($\sum$ *i* ← [*0..<2* ^ *oe-n*]. *Nat-LSBF.to-nat* (*subdivide pad-length num-Zn-pad* ! *i*) ∗ *2* ^ (*i* ∗ *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* ← [*0..<2* ^ *oe-n*]. *Nat-LSBF.to-nat* (*map* (*fill pad-length*) *num-Zn* ! *i*) ∗ *2* ^ (*i* ∗ *pad-length*))
= ($\sum$ *i* ← [*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* ⊆ *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* ⊆ *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* μ (*map Fnr.to-residue-ring num-blocks-carrier*)
unfolding *num-dft-def* μ*-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* δ = *int-lsbf-mod.subtract-mod* $(n + 2)\ \eta$ (*take* $(n + 2)\ \xi$) *in*
add-nat ξ (*add-nat* $(\delta >>_n (2 \mathbin{\hat{}} n))\ \delta$))

lemma *two-pow-n-geq-n-plus-2*: $n \geq 2 \Longrightarrow 2 \mathbin{\hat{}} n \geq n + 2$
proof –
have *aux*: $\bigwedge k.\ 2 \mathbin{\hat{}} (k + 2) \geq k + 4$
subgoal for k
by (*induction* k) *simp-all*
done
assume $n \geq 2$
then obtain k **where** $n = k + 2$ **by** (*metis le-add-diff-inverse2*)
then show *?thesis* **using** *aux*[*of* k] **by** *presburger*
qed

lemma *fermat-mod-pow-2-aux*: $n \geq 2 \Longrightarrow (2{::}nat) \mathbin{\hat{}} (2 \mathbin{\hat{}} n)$ *mod* $2 \mathbin{\hat{}} (n + 2) =$
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 $z\ n\ \xi\ \eta \equiv$
$z < 2 \mathbin{\hat{}} (n + 2) *$ *int-lsbf-fermat.n* n
$\wedge$ z *mod int-lsbf-fermat.n* $n = \xi$
$\wedge$ z *mod* $(2 \mathbin{\hat{}} (n + 2)) = \eta$

lemma *solve-special-residue-problem-correct*:
fixes n :: *nat*
fixes $\xi\ \eta$ :: *nat-lsbf*
assumes $n \geq 2$
assumes *length* $\eta \leq n + 2$
assumes *Nat-LSBF.to-nat* $\xi <$ *int-lsbf-fermat.n* n
assumes z = *solve-special-residue-problem* $n\ \xi\ \eta$
shows *solves-special-residue-problem* (*Nat-LSBF.to-nat* z) n (*Nat-LSBF.to-nat*
ξ) (*Nat-LSBF.to-nat* η)
unfolding *solves-special-residue-problem-def*

```
proof (intro conjI)
  define δ where δ = int-lsbf-mod.subtract-mod (n + 2) η (take (n + 2) ξ)
  then have z = ξ +ₙ ((δ >>ₙ (2 ^ n)) +ₙ δ)
    using assms(4) by (simp add: Let-def solve-special-residue-problem-def)
  then have Nat-LSBF.to-nat z = Nat-LSBF.to-nat ξ + (2 ^(2 ^n) * Nat-LSBF.to-nat
δ + Nat-LSBF.to-nat δ)
    by (simp add: add-nat-correct to-nat-app)
  then have 0: Nat-LSBF.to-nat z = Nat-LSBF.to-nat ξ + int-lsbf-fermat.n n *
Nat-LSBF.to-nat δ
    by simp

  then have Nat-LSBF.to-nat z mod int-lsbf-fermat.n n = Nat-LSBF.to-nat ξ mod
int-lsbf-fermat.n n
    by presburger
  also have ... = Nat-LSBF.to-nat ξ
    using assms(3) by simp
  finally show Nat-LSBF.to-nat z mod int-lsbf-fermat.n n = Nat-LSBF.to-nat ξ .

  have int-lsbf-fermat.n n 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: int (int-lsbf-fermat.n n) 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 int (Nat-LSBF.to-nat δ) mod int Znr.n = Znr.to-residue-ring δ
    by (rule Znr.to-residue-ring-def[symmetric])
  also have ... = Znr.to-residue-ring η ⊖Znr.Zn Znr.to-residue-ring (take (n + 2)
ξ)
    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 ... = (int (Nat-LSBF.to-nat η) − int (Nat-LSBF.to-nat ξ)) mod int
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: int (Nat-LSBF.to-nat δ) mod int Znr.n = ... .

  have Nat-LSBF.to-nat η < 2 ^(n + 2) using ‹length η ≤ n + 2› to-nat-length-bound[of
η] power-increasing[of length η n + 2 2::nat]
    by linarith
  from 0 have int (Nat-LSBF.to-nat z) mod Znr.n = (int (Nat-LSBF.to-nat ξ)
+ int-lsbf-fermat.n n * int (Nat-LSBF.to-nat δ)) mod Znr.n
```

using *int-ops*(*7*) *int-plus* **by** *presburger*
also have ... = (*int* (*Nat-LSBF.to-nat* ξ) *mod Znr.n* + (*int* (*int-lsbf-fermat.n n*) *mod Znr.n*) $*$ (*int* (*Nat-LSBF.to-nat* δ) *mod Znr.n*)) *mod 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 ... = (*int* (*Nat-LSBF.to-nat* ξ) *mod Znr.n* + (*int* (*Nat-LSBF.to-nat* δ) *mod Znr.n*)) *mod 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* ξ) *mod Znr.n* + (*int* (*Nat-LSBF.to-nat* η) *mod Znr.n* − *int* (*Nat-LSBF.to-nat* ξ) *mod Znr.n*)) *mod Znr.n*
using *2* **by** (*simp add*: *mod-simps*)
also have ... = *int* (*Nat-LSBF.to-nat* η) *mod 2* ^ (*n* + *2*)
by *simp*
also have ... = *int* (*Nat-LSBF.to-nat* η) **using** ‹*Nat-LSBF.to-nat* η < *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* η) **.**
then show *Nat-LSBF.to-nat z mod 2* ^ (*n* + *2*) = *Nat-LSBF.to-nat* η
by (*metis nat-int-comparison*(*1*) *zmod-int*)

show *Nat-LSBF.to-nat z* < *2* ^ (*n* + *2*) $*$ *int-lsbf-fermat.n n*
proof −
have *int* (*Nat-LSBF.to-nat z*) = *int* (*Nat-LSBF.to-nat* ξ) + *int* (*int-lsbf-fermat.n n*) $*$ *int* (*Nat-LSBF.to-nat* δ)
using *0*
using *int-ops*(*7*) *int-plus* **by** *presburger*
also have ... ≤ (*2*::*int*) ^ (*2* ^ *n*) + *int* (*int-lsbf-fermat.n n*) $*$ *int* (*Nat-LSBF.to-nat* δ)
using *assms*(*3*) **by** *simp*
also have *int* (*int-lsbf-fermat.n n*) $*$ *int* (*Nat-LSBF.to-nat* δ) ≤ *int* (*int-lsbf-fermat.n n*) $*$ ((*2*::*int*) ^ (*n* + *2*) − *1*)
proof −
have *length* δ ≤ *n* + *2*
unfolding δ*-def*
apply (*intro Znr.length-subtract-mod* ‹*length* η ≤ *n* + *2*›)
using *Znr.length-reduce* **by** *simp*
have *Nat-LSBF.to-nat* δ ≤ *2* ^ (*n* + *2*) − *1*
using *to-nat-length-upper-bound*[*of* δ] *power-increasing*[*OF* ‹*length* δ ≤ *n* + *2*›, *of 2*]
using *diff-le-mono* **by** *fastforce*
then have *int* (*Nat-LSBF.to-nat* δ) ≤ (*2*::*int*) ^ (*n* + *2*) − *1*
using *nat-int-comparison*(*3*)[*of Nat-LSBF.to-nat* δ *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*::*int*) ^ (*2* ^ *n*) + (*2* ^ (*2* ^ *n*) + *1*) $*$ ((*2*::*int*) ^ (*n* + *2*) − *1*)
by *force*
also have ... = ((*2*::*int*) ^ (*2* ^ *n*) + *1*) $*$ *2* ^ (*n* + *2*) − *1*
apply (*simp add*: *distrib-right*)
apply (*simp only*: *diff-conv-add-uminus*[*of* (*4*::*int*) $*$ *2* ^ *n 1*])
apply (*simp only*: *distrib-left*)
done
finally have *int* (*Nat-LSBF.to-nat z*) < *2* ^ (*n* + *2*) $*$ (*int* (*int-lsbf-fermat.n n*))
by (*simp add*: *add.commute mult.commute*)
thus *Nat-LSBF.to-nat z* < *2* ^ (*n* + *2*) $*$ *int-lsbf-fermat.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-lsbf-fermat.n n*) (*2* ^ (*n* + *2*))
proof −
consider *n* = *0* | *n* = *1* | *n* $\geq$ *2* **by** *linarith*
then show *?thesis*
proof *cases*
case *1*
have *gcd* (*3*::*nat*) *4* = *nat* (*gcd* (*3*::*int*) *4*) **using** *gcd-int-int-eq*[*of 3 4*] **by** *simp*
also have ... = *gcd 1 3* **using** *gcd-diff1*[*of 4*::*int 3*, *symmetric*] *gcd.commute*[*of 4*::*int 3*]
by *simp*
also have ... = *1* **by** *simp*
finally show *?thesis* **unfolding** *coprime-iff-gcd-eq-1* **by** (*simp add*: *1*)
next
case *2*
have *gcd* (*5*::*nat*) *8* = *nat* (*gcd* (*5*::*int*) *8*) **using** *gcd-int-int-eq*[*of 5 8*] **by** *simp*
also have ... = *nat* (*gcd 3 5*) **using** *gcd-diff1*[*of 8*::*int 5*] **by** (*simp add*: *gcd.commute*)
also have ... = *nat* (*gcd 2 3*) **using** *gcd-diff1*[*of 5*::*int 3*] **by** (*simp add*: *gcd.commute*)
also have ... = *nat* (*gcd 1 2*) **using** *gcd-diff1*[*of 3*::*int 2*] **by** (*simp add*: *gcd.commute*)
also have ... = *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*::*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*: $Idl_{\mathcal{Z}}\ \{m\}\ {<}{+}{>}_{\mathcal{Z}}\ Idl_{\mathcal{Z}}\ \{n\} = Idl_{\mathcal{Z}}\ \{gcd\ m\ n\}$
proof (*intro equalityI subsetI*)
fix x
assume $x \in Idl_{\mathcal{Z}}\ \{m\}\ {<}{+}{>}_{\mathcal{Z}}\ Idl_{\mathcal{Z}}\ \{n\}$
then obtain $y\ z$ **where** $y \in Idl_{\mathcal{Z}}\ \{m\}$ $z \in Idl_{\mathcal{Z}}\ \{n\}$ $x = y \oplus_{\mathcal{Z}} z$
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** ‹$x = y \oplus_{\mathcal{Z}} z$› **by** *simp*
obtain m' **where** *2*: $m = m' * gcd\ m\ n$
by (*metis dvdE gcd-dvd1 mult.commute*)
obtain n' **where** *3*: $n = n' * gcd\ m\ n$
by (*metis dvdE gcd-dvd2 mult.commute*)
from *1 2 3* **have** $x = (y' * m' + z' * n') * gcd\ m\ n$
by (*simp add*: *int-distrib*(*1*) *mult.assoc*)
then show $x \in Idl_{\mathcal{Z}}\ \{gcd\ m\ n\}$ **using** *int-Idl* **by** *blast*
next
fix x
assume $x \in Idl_{\mathcal{Z}}\ \{gcd\ m\ n\}$
then obtain x' **where** *1*: $x = x' * gcd\ m\ n$ **using** *int-Idl* **by** *fastforce*
obtain $s\ t$ **where** $gcd\ m\ n = s * m + t * n$ **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 $(x' * s) * m \in Idl_{\mathcal{Z}}\ \{m\}$ $(x' * t) * n \in Idl_{\mathcal{Z}}\ \{n\}$
using *int-Idl* **by** *simp-all*
ultimately show $x \in Idl_{\mathcal{Z}}\ \{m\}\ {<}{+}{>}_{\mathcal{Z}}\ Idl_{\mathcal{Z}}\ \{n\}$
unfolding *AbelCoset.set-add-def Coset.set-mult-def* **by** *auto*
qed

lemma *int-ideal-inter*: $Idl_{\mathcal{Z}}\ \{m\} \cap Idl_{\mathcal{Z}}\ \{n\} = Idl_{\mathcal{Z}}\ \{lcm\ m\ n\}$
proof −
have $Idl_{\mathcal{Z}}\ \{m\} \cap Idl_{\mathcal{Z}}\ \{n\} = \{u.\ \exists x.\ u = x * m\} \cap \{u.\ \exists x.\ u = x * n\}$
unfolding *int-Idl* **by** *simp*
also have ... $= \{u.\ m\ dvd\ u\} \cap \{u.\ n\ dvd\ u\}$
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 ... $= \{u.\ m\ dvd\ u \wedge n\ dvd\ u\}$ **by** *blast*
also have ... $= \{u.\ lcm\ m\ n\ dvd\ u\}$ **using** *lcm-least-iff*[*of m n*] **by** *blast*
also have ... $= \{u.\ \exists x.\ u = x * lcm\ m\ n\}$
using *dvd-def*[*symmetric*, *of - lcm m n*]
using *mult.commute*[*of lcm m n*]
by *algebra*
also have ... $= Idl_{\mathcal{Z}}\ \{lcm\ m\ n\}$ **unfolding** *int-Idl* **by** *simp*
finally show *?thesis* **.**

qed

corollary $coprime\ m\ n \Longrightarrow Idl_{\mathcal{Z}}\ \{m\} <+>_{\mathcal{Z}} Idl_{\mathcal{Z}}\ \{n\} = carrier\ \mathcal{Z}$
  using *int-ideal-add coprime-imp-gcd-eq-1 int.genideal-one* **by** *simp*

lemma *genideal-uminus*: $Idl_{\mathcal{Z}}\ \{-x\} = Idl_{\mathcal{Z}}\ \{x\}$
  unfolding *int-Idl*
  by (*metis minus-mult-commute minus-mult-minus*)

lemma *genideal-normalize*: $Idl_{\mathcal{Z}}\ \{x\} = Idl_{\mathcal{Z}}\ \{normalize\ x\}$
  apply ($cases\ x \geq 0$)
  unfolding *normalize-int-def* **using** *genideal-uminus* **by** *simp-all*

corollary $coprime\ m\ n \Longrightarrow Idl_{\mathcal{Z}}\ \{m\} \cap Idl_{\mathcal{Z}}\ \{n\} = Idl_{\mathcal{Z}}\ \{m * n\}$
  using *int-ideal-inter lcm-coprime genideal-normalize* **by** *metis*

lemma *int-ideal-inter-a-r-coset-distrib*: $(Idl_{\mathcal{Z}}\ \{m\} \cap Idl_{\mathcal{Z}}\ \{n\}) +>_{\mathcal{Z}} x = (Idl_{\mathcal{Z}}$
$\{m\} +>_{\mathcal{Z}} x) \cap (Idl_{\mathcal{Z}}\ \{n\} +>_{\mathcal{Z}} x)$
  by (*auto simp add*: *a-r-coset-def r-coset-def*)

lemma *chinese-remainder-very-simple-int*:
  fixes $x\ y\ m\ n$ :: *int*
  assumes $x\ mod\ m = y\ mod\ m$
  assumes $x\ mod\ n = y\ mod\ n$
  shows $x\ mod\ (lcm\ m\ n) = y\ mod\ (lcm\ m\ n)$
proof −
  have $?thesis \longleftrightarrow Idl_{\mathcal{Z}}\ \{lcm\ m\ n\} +>_{\mathcal{Z}} x = Idl_{\mathcal{Z}}\ \{lcm\ m\ n\} +>_{\mathcal{Z}} y$
    using *ZMod-def ZMod-eq-mod* **by** *algebra*
  also have ... $\longleftrightarrow (Idl_{\mathcal{Z}}\ \{m\} \cap Idl_{\mathcal{Z}}\ \{n\}) +>_{\mathcal{Z}} x = (Idl_{\mathcal{Z}}\ \{m\} \cap Idl_{\mathcal{Z}}\ \{n\})$
$+>_{\mathcal{Z}} y$
    using *int-ideal-inter* **by** *presburger*
  also have ... $\longleftrightarrow (Idl_{\mathcal{Z}}\ \{m\} +>_{\mathcal{Z}} x) \cap (Idl_{\mathcal{Z}}\ \{n\} +>_{\mathcal{Z}} x) = (Idl_{\mathcal{Z}}\ \{m\} +>_{\mathcal{Z}}$
$y) \cap (Idl_{\mathcal{Z}}\ \{n\} +>_{\mathcal{Z}} y)$
    by (*simp only*: *int-ideal-inter-a-r-coset-distrib*)
  also have ... **using** *assms ZMod-def ZMod-eq-mod* **by** *blast*
  finally show *?thesis* **by** *blast*
qed

lemma *chinese-remainder-very-simple-nat*:
  fixes $x\ y\ m\ n$ :: *nat*
  assumes $x\ mod\ m = y\ mod\ m$
  assumes $x\ mod\ n = y\ mod\ n$
  shows $x\ mod\ (lcm\ m\ n) = y\ mod\ (lcm\ m\ n)$
  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 :: *nat*
  fixes $\xi\ \eta$ :: *nat*

assumes *solves-special-residue-problem z1 n* ξ η
assumes *solves-special-residue-problem z2 n* ξ η
shows *z1* = *z2*
proof −
from *assms* **have** *z1 mod* (*lcm* (*int-lsbf-fermat.n n*) (*2* ^ (*n* + *2*))) = *z2 mod* (*lcm* (*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* (*int-lsbf-fermat.n n*) (*2* ^ (*n* + *2*))
using *fn-zn-coprime* **.**
hence *lcm* (*int-lsbf-fermat.n n*) (*2* ^ (*n* + *2*)) = (*int-lsbf-fermat.n n*) ∗ (*2* ^ (*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 [] = *concat* (*rev acc*)
| *combine-z-aux l acc* [*z*] = *combine-z-aux l* (*z* # *acc*) []
| *combine-z-aux l acc* (*z1* # *z2* # *zs*) = (*let*
(*z1h*, *z1t*) = *split-at l z1 in*
combine-z-aux l (*z1h* # *acc*) ((*add-nat z1t z2*) # *zs*)
)

definition *combine-z* :: *nat* ⇒ *nat-lsbf list* ⇒ *nat-lsbf* **where**
combine-z l zs = *combine-z-aux l* [] *zs*

lemma *combine-z-aux-correct*:
assumes *l* > *0*
assumes $\bigwedge$*z*. *z* ∈ *set zs* ⟹ *length z* ≥ *l*
shows *Nat-LSBF.to-nat* (*combine-z-aux l acc zs*) = *Nat-LSBF.to-nat* (*concat* (*rev acc*)) +
2 ^ (*length* (*concat acc*)) ∗ ($\sum$ *i* ← [*0*..<*length zs*]. *Nat-LSBF.to-nat* (*zs* ! *i*) ∗ *2* ^ (*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* = *take l z1 z1t* = *drop l z1*
have *lena*: *l* ≤ *length* (*add-nat z1t z2*)
using *length-add-nat-lower*[*of z1t z2*] *3.prems* **by** *force*

from *z1h-z1t-def* **have** *combine-z-aux l acc* (*z1* # *z2* # *zs*) = *combine-z-aux l* (*z1h* # *acc*) ((*add-nat z1t z2*) # *zs*)
  by *simp*
then have *Nat-LSBF.to-nat* (*combine-z-aux l acc* (*z1* # *z2* # *zs*)) = *Nat-LSBF.to-nat* ... **by** *argo*
also have ... = *Nat-LSBF.to-nat* (*concat* (*rev* (*z1h* # *acc*))) +
  2 ^ *length* (*concat* (*z1h* # *acc*)) ∗
  (∑ *i*←[*0*..<*length* (*add-nat z1t z2* # *zs*)]. *Nat-LSBF.to-nat* ((*add-nat z1t z2* # *zs*) ! *i*) ∗ *2* ^ (*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.prems*)
  subgoal using *3.prems 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.prems* **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* λ*i*. *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 *prems* **for** *x*

proof −
from *prems* **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 $*$ ($\sum i \leftarrow [0..<Suc\ (length\ zs)]$. *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 ... $=$ *Nat-LSBF.to-nat z1* $+$ ($\sum i \leftarrow [0..<Suc\ (length\ zs)]$. 2 ^ l $*$ (*Nat-LSBF.to-nat* $((z2\ \#\ zs)\ !\ i) * 2$ ^ $(i * l)$))
by (*intro arg-cong2*[**where** $f = (+)$] *refl sum-list-const-mult*[*symmetric*])
also have ... $=$ *Nat-LSBF.to-nat z1* $+$ ($\sum i \leftarrow [0..<Suc\ (length\ zs)]$. *Nat-LSBF.to-nat* $((z2\ \#\ zs)\ !\ i) * 2$ ^ $(Suc\ i * l)$)
apply (*intro arg-cong2*[**where** $f = (+)$] *refl sum-list-eq*)
by (*simp add*: *power-add*)
also have ... $=$ *Nat-LSBF.to-nat z1* $+$ ($\sum i \leftarrow [0..<Suc\ (length\ zs)]$. *Nat-LSBF.to-nat* $((z1\ \#\ z2\ \#\ zs)\ !\ Suc\ i) * 2$ ^ $(Suc\ i * l)$)
by *simp*
also have ... $=$ *Nat-LSBF.to-nat z1* $+$ ($\sum i \leftarrow [1..<Suc\ (Suc\ (length\ zs))]$. *Nat-LSBF.to-nat* $((z1\ \#\ z2\ \#\ zs)\ !\ i) * 2$ ^ $(i * l)$)
unfolding *sum-list-index-trafo*[*of* λi. *Nat-LSBF.to-nat* $((z1\ \#\ z2\ \#\ zs)\ !\ i) * 2$ ^ $(i * l)$ *Suc* $[0..<Suc\ (length\ zs)]$]
unfolding *map-Suc-upt* **by** *simp*
also have ... $=$ *Nat-LSBF.to-nat* $((z1\ \#\ z2\ \#\ zs)\ !\ 0) * 2$ ^ $(0 * l)$ $+$ ($\sum i \leftarrow [1..<length\ (z1\ \#\ z2\ \#\ zs)]$. *Nat-LSBF.to-nat* $((z1\ \#\ z2\ \#\ zs)\ !\ i) * 2$ ^ $(i * l)$)
by *simp*
also have ... $=$ ($\sum i \leftarrow [0..<length\ (z1\ \#\ z2\ \#\ zs)]$. *Nat-LSBF.to-nat* $((z1\ \#\ z2\ \#\ zs)\ !\ i) * 2$ ^ $(i * l)$)
using *sum-list-split-0*[**where** $f = \lambda i$. *Nat-LSBF.to-nat* $((z1\ \#\ z2\ \#\ zs)\ !\ i) * 2$ ^ $(i * l)$] **by** *simp*
finally show *?case* **.**
qed

lemma *combine-z-correct*:
assumes $l > 0$
assumes $\bigwedge z.\ z \in set\ zs \Longrightarrow length\ z \geq l$
shows *Nat-LSBF.to-nat* (*combine-z l zs*) $=$ ($\sum i \leftarrow [0..<length\ zs]$. *Nat-LSBF.to-nat* $(zs\ !\ i) * 2$ ^ $(i * l)$)
unfolding *combine-z-def* **using** *combine-z-aux-correct*[*OF assms*] **by** *simp*

lemma *length-combine-z-aux-le*:
assumes $\bigwedge z.\ z \in set\ zs \Longrightarrow length\ z \leq lz$
assumes $length\ z \leq lz + 1$
assumes $l > 0$
shows $length\ (combine\text{-}z\text{-}aux\ l\ acc\ (z\ \#\ zs)) \leq (lz + 1) * (length\ zs + 1) + length\ (concat\ acc)$
using *assms* **proof** (*induction zs arbitrary*: *acc z*)

case *Nil*
then show *?case* **by** *simp*
next
case (*Cons z1 zs*)
then have *len-drop-z*: *length* (*drop l z*) ≤ *lz* **by** *simp*
have *lena*: *length* (*add-nat* (*drop l z*) *z1*) ≤ *lz* + *1*
apply (*estimation estimate*: *length-add-nat-upper*)
using *len-drop-z Cons.prems* **by** *simp*
have *length* (*combine-z-aux l acc* (*z* # *z1* # *zs*)) =
length (*combine-z-aux l* (*take l z* # *acc*) (*add-nat* (*drop l z*) *z1* # *zs*))
by *simp*
also have ... ≤ (*lz* + *1*) ∗ (*length zs* + *1*) + *length* (*concat* (*take l z* # *acc*))
apply (*intro Cons.IH*)
subgoal using *Cons.prems* **by** *simp*
subgoal using *lena* **.**
subgoal using *Cons.prems* **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.prems* **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 ⋀*z*. *z* ∈ *set zs* ⟹ *length z* ≤ *lz*
assumes *l* > *0*
shows *length* (*combine-z l zs*) ≤ (*lz* + *1*) ∗ *length zs*
proof (*cases zs*)
case *Nil*
then show *?thesis* **by** (*simp add*: *combine-z-def*)
next
case (*Cons z zs'*)
have *length* (*combine-z l zs*) ≤ (*lz* + *1*) ∗ (*length zs'* + *1*) + *length* (*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 ⇒ nat-lsbf ⇒ nat-lsbf ⇒ 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 = (if odd m then (m + 1) div 2 else (m + 2) div 2);
    oe-n = (if odd m then n + 1 else n);
    a′ = subdivide (2 ^ (n − 1)) a;
    b′ = subdivide (2 ^ (n − 1)) b;

    — residue mod 2^(n+2)
    α = map (int-lsbf-mod.reduce (n + 2)) a′;
    u = concat (map (fill (3∗n + 5)) α);
    β = map (int-lsbf-mod.reduce (n + 2)) b′;
    v = concat (map (fill (3∗n + 5)) β);
    uv = ensure-length ((3∗n + 5) ∗ 2 ^ (oe-n + 1)) (karatsuba-mul-nat u v);
    γ = subdivide (2 ^ (oe-n − 1)) (subdivide (3∗n + 5) uv);
    η = map4 (λx y z w.
        int-lsbf-mod.add-mod (n + 2)
        (int-lsbf-mod.subtract-mod (n + 2) (take (n + 2) x) (take (n + 2) y))
        (int-lsbf-mod.subtract-mod (n + 2) (take (n + 2) z) (take (n + 2) w))
      )
      (γ ! 0) (γ ! 1) (γ ! 2) (γ ! 3);

    — residue mod F_n
    prim-root-exponent = (if odd m then 1 else 2);
    a′-carrier = map (fill (2 ^ (n + 1))) a′;
    b′-carrier = map (fill (2 ^ (n + 1))) b′;
    a-dft = int-lsbf-fermat.fft n prim-root-exponent a′-carrier;
    b-dft = int-lsbf-fermat.fft n prim-root-exponent b′-carrier;
    a-dft-odds = evens-odds False a-dft;
    b-dft-odds = evens-odds False b-dft;
    c-dft-odds = map2 (schoenhage-strassen n) a-dft-odds b-dft-odds;
    c-diffs = int-lsbf-fermat.ifft n (prim-root-exponent ∗ 2) c-dft-odds;
    ξ′ = map2 (λcj j. int-lsbf-fermat.add-fermat n
       (int-lsbf-fermat.divide-by-power-of-2 cj (oe-n + prim-root-exponent ∗ j − 1))
        (int-lsbf-fermat.from-nat-lsbf n (replicate (oe-n + 2 ^ n) False @ [True])))
      c-diffs [0..<2 ^ (oe-n − 1)];
    ξ = map (int-lsbf-fermat.reduce n) ξ′;

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

  — return the resulting sum
  in result)
```

by *pat-completeness auto*

termination
apply (*relation Wellfounded.measure* (λ(*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*: ¬ *m* < *3*
assumes *a-carrier*: *a* ∈ *int-lsbf-fermat.fermat-non-unique-carrier m*
assumes *b-carrier*: *b* ∈ *int-lsbf-fermat.fermat-non-unique-carrier m*
begin

sublocale *m-lemmas*
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* ^ (*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 γs **where** γs = *subdivide pad-length uv*
definition γ **where** γ = *subdivide* (*2* ^ (*oe-n* − *1*)) γs
definition η **where** η = *map4* (λ*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*))
) (γ ! *0*) (γ ! *1*) (γ ! *2*) (γ ! *3*)
definition *c-dft-odds* **where** *c-dft-odds* = *map2* (*schoenhage-strassen n*) *A.num-dft-odds B.num-dft-odds*
definition *c-diffs* **where** *c-diffs* = *int-lsbf-fermat.ifft n* (*prim-root-exponent* ∗ *2*) *c-dft-odds*
definition ξ' **where** ξ' = *map2* (λ*cj j*. *int-lsbf-fermat.add-fermat n*
(*int-lsbf-fermat.divide-by-power-of-2 cj* (*oe-n* + *prim-root-exponent* ∗ *j* − *1*))
(*int-lsbf-fermat.from-nat-lsbf n* (*replicate* (*oe-n* + *2* ^ *n*) *False* @ [*True*])))
c-diffs [*0*..<*2* ^ (*oe-n* − *1*)]
definition ξ **where** ξ = *map* (*int-lsbf-fermat.reduce n*) ξ'
definition *z* **where** *z* = *map2* (*solve-special-residue-problem n*) ξ η
definition *z-filled* **where** *z-filled* = *map* (*fill* (*2* ^ (*n* − *1*))) *z*

definition *z-consts* **where** *z-consts* = *replicate* (*2* ^ (*oe-n* − *1*)) (*replicate* (*oe-n* + *2* ^ *n*) *False* @ [*True*])
definition *z-sum* **where** *z-sum* = *combine-z* (*2* ^ (*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*
  γs-def γ-def η-def c-dft-odds-def c-diffs-def ξ′-def ξ-def z-def z-filled-def z-consts-def
   z-sum-def result-def prim-root-exponent-def μ-def

lemma *result-eq*: *schoenhage-strassen m a b* = *result*
 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*:
 length (*subdivide pad-length uv*) = *2* ^ (*oe-n* + *1*)
 x ∈ *set* (*subdivide pad-length uv*) ⟹ *length x* = *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* ^ (*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* ^ (*oe-n* − *1*)
 unfolding *c-diffs-def*
 by (*intro Fnr.length-ifft length-c-dft-odds*)
lemma *length-ξ′*: *length ξ′* = *2* ^ (*oe-n* − *1*)
 unfolding *ξ′-def* **by** (*simp add*: *length-c-diffs*)
lemma *length-ξ*: *length ξ* = *2* ^ (*oe-n* − *1*)
 unfolding *ξ-def* **by** (*simp add*: *length-ξ′*)

lemma *γ-nth*: ⋀*i j*. *i* < *4* ⟹ *j* < *2* ^ (*oe-n* − *1*) ⟹ *γ* ! *i* ! *j* = (*subdivide pad-length uv*) ! (*i* ∗ *2* ^ (*oe-n* − *1*) + *j*)
 subgoal for *i j*
  unfolding *γ-def γs-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

.
done
lemma *γ-nth′*: $\bigwedge j.\ j < 2\ \hat{}\ (oe\text{-}n + 1) \Longrightarrow \gamma\ !\ (j\ div\ 2\ \hat{}\ (oe\text{-}n - 1))\ !\ (j\ mod\ 2\ \hat{}\ (oe\text{-}n - 1)) = subdivide\ pad\text{-}length\ uv\ !\ j$
using *index-decomp γ-nth* **by** *algebra*
lemma *scγ*: $length\ \gamma = 4\ \bigwedge i.\ i < 4 \Longrightarrow length\ (\gamma\ !\ i) = 2\ \hat{}\ (oe\text{-}n - 1)$
proof −
have *1*: $(2::nat)\ \hat{}\ (oe\text{-}n - 1) > 0$ **by** *simp*
have *2*: $length\ (subdivide\ pad\text{-}length\ uv) = 2\ \hat{}\ (oe\text{-}n - 1) * 4$
using *two-pow-Suc-oe-n-as-prod scuv*(*1*) **by** *simp*
show $length\ \gamma = 4\ \bigwedge i.\ i < 4 \Longrightarrow length\ (\gamma\ !\ i) = 2\ \hat{}\ (oe\text{-}n - 1)$
using *subdivide-correct*[*OF 1 2*]
unfolding *γ-def*[*symmetric*] *γs-def*[*symmetric*] **by** *simp-all*
qed
lemmas $length\text{-}\gamma = sc\gamma(1)$
lemmas $length\text{-}\gamma\text{-}i = sc\gamma(2)$
lemma *length-γ-nth*: $\bigwedge i\ j.\ i < 4 \Longrightarrow j < 2\ \hat{}\ (oe\text{-}n - 1) \Longrightarrow length\ (\gamma\ !\ i\ !\ j) = pad\text{-}length$
subgoal for *i j*
using *scuv γ-nth index-comp*[*of i j*] **by** *fastforce*
done
lemma *length-η*: $length\ \eta = 2\ \hat{}\ (oe\text{-}n - 1)$ **unfolding** *η-def*
using *length-γ-i* **by** (*simp add*: *map4-as-map*)
lemma *length-z*: $length\ z = 2\ \hat{}\ (oe\text{-}n - 1)$
unfolding *z-def* **using** *length-ξ length-η* **by** *simp*
lemma *nth-z*: $z\ !\ j = solve\text{-}special\text{-}residue\text{-}problem\ n\ (\xi\ !\ j)\ (\eta\ !\ j)$ **if** $j < 2\ \hat{}\ (oe\text{-}n - 1)$ **for** *j*
unfolding *z-def* **using** *length-z that length-ξ length-η* **by** *simp*
lemma *length-z-filled*: $length\ z\text{-}filled = 2\ \hat{}\ (oe\text{-}n - 1)$
unfolding *z-filled-def* **by** (*simp add*: *length-z*)
lemma *length-z-consts*: $length\ z\text{-}consts = 2\ \hat{}\ (oe\text{-}n - 1)$
unfolding *z-consts-def* **by** *simp*

end

theorem *schoenhage-strassen-correct′*:
assumes $a \in int\text{-}lsbf\text{-}fermat.fermat\text{-}non\text{-}unique\text{-}carrier\ m$
assumes $b \in int\text{-}lsbf\text{-}fermat.fermat\text{-}non\text{-}unique\text{-}carrier\ m$
shows $int\text{-}lsbf\text{-}fermat.to\text{-}residue\text{-}ring\ m\ (schoenhage\text{-}strassen\ m\ a\ b)$
$= int\text{-}lsbf\text{-}fermat.to\text{-}residue\text{-}ring\ m\ a \otimes_{int\text{-}lsbf\text{-}fermat.Fn\ m} int\text{-}lsbf\text{-}fermat.to\text{-}residue\text{-}ring\ m\ b \wedge schoenhage\text{-}strassen\ m\ a\ b \in int\text{-}lsbf\text{-}fermat.fermat\text{-}non\text{-}unique\text{-}carrier\ m$
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*: $schoenhage\text{-}strassen\ m\ a\ b = int\text{-}lsbf\text{-}fermat.from\text{-}nat\text{-}lsbf\ m\ (grid\text{-}mul\text{-}nat\ a\ b)$

by (*simp add*: *schoenhage-strassen.simps*)
then have *int-lsbf-fermat.to-residue-ring m* (*schoenhage-strassen m a b*) = *int-lsbf-fermat.to-residue-ring m* (*grid-mul-nat a b*)
using *int-lsbf-fermat.from-nat-lsbf-correct* **by** *simp*
also have ... = *int* (*Nat-LSBF.to-nat* (*grid-mul-nat a b*)) *mod int* (*2* ^ (*2* ^ *m*) + *1*)
unfolding *int-lsbf-fermat.to-residue-ring.simps* **by** *argo*
also have ... = *int* (*Nat-LSBF.to-nat a* $*$ *Nat-LSBF.to-nat b*) *mod int* (*2* ^ (*2* ^ *m*) + *1*)
by (*simp add*: *grid-mul-nat-correct*)
also have ... = *int-lsbf-fermat.to-residue-ring m a* $\otimes_{residue\text{-}ring\ (2\ \hat{}\ (2\ \hat{}\ m)\ +\ 1)}$ *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.prems* **by** *unfold-locales assumption+*

have *Fn-def′*: *Fn* = *residue-ring* (*2* ^ *2* ^ *n* + *1*)
unfolding *Fn-def* **by** (*simp add*: *int-ops add.commute*)
have *fn-Fn*[*simp*]: *int-lsbf-fermat.Fn n* = *Fn*
unfolding *Fn-def int-lsbf-fermat.Fn-def* **by** (*rule refl*)

from *Fmr.res-carrier-eq* **have** *Fm-carrierI*: $\bigwedge i.\ 0 \le i \Longrightarrow i < 2\ \hat{}\ 2\ \hat{}\ m + 1 \Longrightarrow i \in$ *carrier Fm*
by *simp*

define c' **where** c' = *map* ($\lambda j.$ $\sum\sigma \leftarrow$ [*0..<2* ^ *oe-n*]. (*int* (*Nat-LSBF.to-nat* (*A.num-blocks* ! σ)) $*$ *int* (*Nat-LSBF.to-nat* (*B.num-blocks* ! ((*2* ^ *oe-n* + *j* − σ) *mod 2* ^ *oe-n*))))) [*0..<2* ^ *oe-n*]
define z' **where** z' = ($\lambda j.$ *if j* < *2* ^ (*oe-n* − *1*) *then* c' ! *j* − c' ! (*2* ^ (*oe-n* − *1*) + *j*) + *2* ^ (*oe-n* + *2* ^ *n*) *else 2* ^ (*oe-n* + *2* ^ *n*))
define z'' **where** z'' = ($\lambda j.$ *if j* < *2* ^ (*oe-n* − *1*) *then* c' ! *j* $\ominus_{Fm}$ c' ! (*2* ^ (*oe-n* − *1*) + *j*) $\oplus_{Fm}$ *2* [^]$_{Fm}$ (*oe-n* + *2* ^ *n*) *else 2* [^]$_{Fm}$ (*oe-n* + *2* ^ *n*))

have *length-c′*: *length* c' = *2* ^ *oe-n* **unfolding** *c′-def* **by** *simp*
have *c′-nth*: c' ! *j* = ($\sum\sigma \leftarrow$ [*0..<2* ^ *oe-n*]. (*int* (*Nat-LSBF.to-nat* (*A.num-blocks* ! σ)) $*$ *int* (*Nat-LSBF.to-nat* (*B.num-blocks* ! ((*2* ^ *oe-n* + *j* − σ) *mod 2* ^ *oe-n*)))))
if *j* < *2* ^ *oe-n* **for** *j*
unfolding *c′-def* **using** *that* **by** *simp*
have *c′-nth-nat*: c' ! *j* = *int* ($\sum\sigma \leftarrow$ [*0..<2* ^ *oe-n*]. (*Nat-LSBF.to-nat* (*A.num-blocks* ! σ) $*$ *Nat-LSBF.to-nat* (*B.num-blocks* ! ((*2* ^ *oe-n* + *j* − σ) *mod 2* ^ *oe-n*))))

if *j* < *2* ^ *oe-n* **for** *j*
proof −
have *c*′ ! *j* = (∑*σ* ← [*0*..<*2* ^ *oe-n*]. (*int* (*Nat-LSBF.to-nat* (*A.num-blocks* ! *σ*) ∗ *Nat-LSBF.to-nat* (*B.num-blocks* ! ((*2* ^ *oe-n* + *j* − *σ*) *mod* *2* ^ *oe-n*)))))
unfolding *c*′*-nth*[*OF that*] **by** *simp*
also have ... = *int* (∑*σ* ← [*0*..<*2* ^ *oe-n*]. *Nat-LSBF.to-nat* (*A.num-blocks* ! *σ*) ∗ *Nat-LSBF.to-nat* (*B.num-blocks* ! ((*2* ^ *oe-n* + *j* − *σ*) *mod* *2* ^ *oe-n*)))
by (*intro sum-list-int*[*symmetric*])
finally show *c*′ ! *j* = ... **.**
qed
have *c*′*-lower-bound*: *c*′ ! *j* ≥ *0* **if** *j* < *2* ^ *oe-n* **for** *j*
unfolding *c*′*-nth*[*OF that*]
apply (*intro sum-list-nonneg*) **by** *fastforce*
have *c*′*-upper-bound*: *c*′ ! *j* < *2* ^ (*oe-n* + *2* ^ *n*) **if** *j* < *2* ^ *oe-n* **for** *j*
proof −
have *Nat-LSBF.to-nat* (*A.num-blocks* ! *σ*) ∗ *Nat-LSBF.to-nat* (*B.num-blocks* ! ((*2* ^ *oe-n* + *j* − *σ*) *mod* *2* ^ *oe-n*)) < *2* ^ *2* ^ *n*
if *σ* < *2* ^ *oe-n* **for** *σ*
proof −
have *length* (*A.num-blocks* ! *σ*) = *2* ^ (*n* − *1*) **using** *A.length-nth-num-blocks that* **by** *simp*
then have *Nat-LSBF.to-nat* (*A.num-blocks* ! *σ*) < *2* ^ *2* ^ (*n* − *1*)
using *to-nat-length-bound* **by** *metis*
moreover have *length* (*B.num-blocks* ! ((*2* ^ *oe-n* + *j* − *σ*) *mod* *2* ^ *oe-n*)) = *2* ^ (*n* − *1*)
using *B.length-nth-num-blocks* **by** *simp*
then have *Nat-LSBF.to-nat* (*B.num-blocks* ! ((*2* ^ *oe-n* + *j* − *σ*) *mod* *2* ^ *oe-n*)) < *2* ^ *2* ^ (*n* − *1*)
using *to-nat-length-bound* **by** *metis*
ultimately have *Nat-LSBF.to-nat* (*A.num-blocks* ! *σ*) ∗ *Nat-LSBF.to-nat* (*B.num-blocks* ! ((*2* ^ *oe-n* + *j* − *σ*) *mod* *2* ^ *oe-n*)) <
2 ^ *2* ^ (*n* − *1*) ∗ *2* ^ *2* ^ (*n* − *1*)
by (*intro mult-strict-mono*) *simp-all*
also have ... = *2* ^ *2* ^ *n* **using** *n-gt-1*
by (*simp add*: *power-add*[*symmetric*] *mult-2*[*symmetric*] *power-Suc*[*symmetric*])
finally show *?thesis* **.**
qed
then have (∑*σ* ← [*0*..<*2* ^ *oe-n*]. (*Nat-LSBF.to-nat* (*A.num-blocks* ! *σ*) ∗ *Nat-LSBF.to-nat* (*B.num-blocks* ! ((*2* ^ *oe-n* + *j* − *σ*) *mod* *2* ^ *oe-n*)))) < *length* [*0*..<*2* ^ *oe-n*] ∗ *2* ^ *2* ^ *n*
by (*intro sum-list-estimation-le*) *simp-all*
then have *c*′ ! *j* < *length* [*0*..<*2* ^ *oe-n*] ∗ *2* ^ *2* ^ *n*
unfolding *c*′*-nth-nat*[*OF that*]
using *nat-int-comparison*(*2*)[*symmetric*] **by** *blast*
also have ... = *2* ^ (*oe-n* + *2* ^ *n*)
by (*simp add*: *power-add*)
finally show *c*′ ! *j* < *2* ^ (*oe-n* + *2* ^ *n*) **.**
qed
have *c*′*-carrier*: *c*′ ! *j* ∈ *carrier Fm* **if** *j* < *2* ^ *oe-n* **for** *j*

proof −
have c' ! $j < 2$ ^ $(oe\text{-}n + 2$ ^ $n)$ **using** c'*-upper-bound*[*OF that*] **.**
also have ... < 2 ^ $(oe\text{-}n + 1 + 2$ ^ $n)$ **by** *simp*
also have ... ≤ 2 ^ 2 ^ 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' ! $j = (\sum \sigma \leftarrow [0..<2$ ^ $oe\text{-}n]. \sum \varrho \leftarrow [0..<2$ ^ $oe\text{-}n]$. *of-bool* $([j = \sigma + \varrho]$ $(mod\ 2$ ^ $oe\text{-}n)) * (int\ (Nat\text{-}LSBF.to\text{-}nat\ (A.num\text{-}blocks\ !\ \sigma)) * int\ (Nat\text{-}LSBF.to\text{-}nat\ (B.num\text{-}blocks\ !\ \varrho))))$
if $j < 2$ ^ $oe\text{-}n$ **for** j
proof −
have c' ! $j = (\sum \sigma \leftarrow [0..<2$ ^ $oe\text{-}n]$. $int\ (Nat\text{-}LSBF.to\text{-}nat\ (A.num\text{-}blocks\ !\ \sigma)) *$
$int\ (Nat\text{-}LSBF.to\text{-}nat\ (B.num\text{-}blocks\ !\ ((2$ ^ $oe\text{-}n + j - \sigma)\ mod\ 2$ ^ $oe\text{-}n))))$
using c'*-nth*[*OF that*] **.**
also have ... $= (\sum \sigma \leftarrow [0..<2$ ^ $oe\text{-}n]. \sum \varrho \leftarrow [0..<2$ ^ $oe\text{-}n]$. *of-bool* $(\varrho = (2$ ^ $oe\text{-}n + j - \sigma)\ mod\ 2$ ^ $oe\text{-}n) * (int\ (Nat\text{-}LSBF.to\text{-}nat\ (A.num\text{-}blocks\ !\ \sigma)) * int\ (Nat\text{-}LSBF.to\text{-}nat\ (B.num\text{-}blocks\ !\ \varrho))))$
by (*intro semiring-1-sum-list-eq of-bool-distinct-in*[*symmetric*]) *simp-all*
also have ... $= (\sum \sigma \leftarrow [0..<2$ ^ $oe\text{-}n]. \sum \varrho \leftarrow [0..<2$ ^ $oe\text{-}n]$. *of-bool* $([j = \sigma + \varrho]$ $(mod\ 2$ ^ $oe\text{-}n)) * (int\ (Nat\text{-}LSBF.to\text{-}nat\ (A.num\text{-}blocks\ !\ \sigma)) * int\ (Nat\text{-}LSBF.to\text{-}nat\ (B.num\text{-}blocks\ !\ \varrho))))$
apply (*intro-cong* [*cong-tag-2* (∗), *cong-tag-1 of-bool*] *more*: *semiring-1-sum-list-eq refl*)
subgoal premises *prems* **for** σ ϱ
unfolding *cong-def*
using *cyclic-index-lemma*[*of* σ 2 ^ $oe\text{-}n$ ϱ j, *symmetric*] *that prems*
by *auto*
done
finally show *?thesis* **.**
qed

have z'-z'': $z'\ j = z''\ j$ **if** $j < 2$ ^ $oe\text{-}n$ **for** j
proof −
have $(2 :: int)$ ^ $(oe\text{-}n + 2$ ^ $n) = int\ (2$ ^ $(oe\text{-}n + 2$ ^ $n))$ **by** *simp*
also have ... $= int\ (2$ ^ $(oe\text{-}n + 2$ ^ $n)\ mod\ Fmr.n)$
apply (*intro arg-cong*[**where** $f = int$] *mod-less*[*symmetric*])
using *oe-n-m-bound-0*
by (*meson one-less-numeral-iff power-strict-increasing semiring-norm*(*76*) *trans-less-add1*)
also have ... $= 2\ [\urcorner]_{Fm}\ (oe\text{-}n + 2$ ^ $n)$
by (*simp add*: *Fmr.pow-nat-eq zmod-int*)
finally have *twopow*: $(2 :: int)$ ^ $(oe\text{-}n + 2$ ^ $n) = 2\ [\urcorner]_{Fm}\ (oe\text{-}n + 2$ ^ $n)$ **.**
show $z'\ j = z''\ j$
proof (*cases* $j < 2$ ^ $(oe\text{-}n - 1)$)
case *True*

then have z' j = c' ! j − c' ! (2 ^ (oe-n − 1) + j) + 2 ^ (oe-n + 2 ^ n)
unfolding z'-*def* **by** *simp*
moreover have ... ≥ 0 ... < *Fmr.n*
subgoal using c'-*upper-bound*[*of* 2 ^ (oe-n − 1) + j] c'-*lower-bound*[*of* j]
using ‹j < 2 ^ *oe-n*› *index-intros*(2)[*of* j] *True* **by** *simp*
subgoal
proof −
have c' ! j − c' ! (2 ^ (oe-n − 1) + j) < 2 ^ (oe-n + 2 ^ n)
using c'-*upper-bound*[*OF* ‹j < 2 ^ *oe-n*›] c'-*lower-bound*[*OF* *index-intros*(2)[*OF* ‹j < 2 ^ (oe-n − 1)›]]
by *simp*
then have c' ! j − c' ! (2 ^ (oe-n − 1) + j) + 2 ^ (oe-n + 2 ^ n) < 2 ^ (oe-n + 1 + 2 ^ n)
by *simp*
also have ... < 2 ^ 2 ^ 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 *mod Fmr.n* **by** *simp*
have z'' j = c' ! j $\ominus_{Fm}$ c' ! (2 ^ (oe-n − 1) + j) $\oplus_{Fm}$ 2 $[\urcorner]_{Fm}$ (oe-n + 2 ^ n)
unfolding z''-*def* **using** *True* **by** *simp*
also have ... = ((c' ! j $\ominus_{Fm}$ c' ! (2 ^ (oe-n − 1) + j)) + 2 $[\urcorner]_{Fm}$ (oe-n + 2 ^ n)) *mod Fmr.n*
by (*intro Fmr.res-add-eq*)
also have ... = ((c' ! j $\ominus_{Fm}$ c' ! (2 ^ (oe-n − 1) + j)) + 2 ^ (oe-n + 2 ^ n)) *mod Fmr.n*
using ‹2 ^ (oe-n + 2 ^ n) = 2 $[\urcorner]_{Fm}$ (oe-n + 2 ^ n)› **by** *argo*
also have ... = ((c' ! j − c' ! (2 ^ (oe-n − 1) + j)) *mod Fmr.n* + 2 ^ (oe-n + 2 ^ n)) *mod Fmr.n*
using *Fmr.residues-minus-eq* **by** *simp*
also have ... = ((c' ! j − c' ! (2 ^ (oe-n − 1) + j)) + 2 ^ (oe-n + 2 ^ n)) *mod Fmr.n*
by (*simp add*: *mod-add-left-eq*)
also have ... = z' j *mod Fmr.n*
unfolding ‹z' j = c' ! j − c' ! (2 ^ (oe-n − 1) + j) + 2 ^ (oe-n + 2 ^ n)› **by** (*intro refl*)
finally show *?thesis* **using** ‹z' j = z' j *mod Fmr.n*› **by** *argo*
next
case *False*
then show *?thesis* **unfolding** z'-*def* z''-*def* **using** *twopow* **by** *simp*
qed
qed

have z'-*carrier*: z'' j ∈ *carrier Fm* **if** j < 2 ^ *oe-n* **for** j
unfolding z''-*def*
apply (*intro prop-ifI*[**where** $P = \lambda p.\ p \in$ *carrier Fm*] *Fmr.a-closed Fmr.minus-closed Fmr.nat-pow-closed* c'-*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* ← [*0*..<*2* ^ *oe-n*]. ($\bigoplus_{Fm}$*k* ← [*0*..<*2* ^ *oe-n*].
map (*int* ∘ *Nat-LSBF.to-nat*) *A.num-blocks* ! *k* $\otimes_{Fm}$ *map* (*int* ∘ *Nat-LSBF.to-nat*) *B.num-blocks* ! ((*2* ^ *oe-n* + *j* − *k*) *mod 2* ^ *oe-n*)) $\otimes_{Fm}$ ((*2* [^]$_{Fm}$ (*2*::*nat*) ^(*n* − *1*)) [^]$_{Fm}$ *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* ← [*0*..<*2* ^ *oe-n*]. (*c′* ! *i*) $\otimes_{Fm}$ *2* [^]$_{Fm}$ (*i* ∗ *2* ^(*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* ^ *oe-n* **by** *simp*
have ($\bigoplus_{Fm}$*k* ← [*0*..<*2* ^ *oe-n*]. *map* (*int* ∘ *Nat-LSBF.to-nat*) *A.num-blocks* ! *k* $\otimes_{Fm}$
map (*int* ∘ *Nat-LSBF.to-nat*) *B.num-blocks* ! ((*2* ^ *oe-n* + *j* − *k*) *mod 2* ^ *oe-n*)) =
($\bigoplus_{Fm}$*k* ← [*0*..<*2* ^ *oe-n*]. (*map* (*int* ∘ *Nat-LSBF.to-nat*) *A.num-blocks* ! *k* ∗
map (*int* ∘ *Nat-LSBF.to-nat*) *B.num-blocks* ! ((*2* ^ *oe-n* + *j* − *k*) *mod 2* ^ *oe-n*)) *mod Fmr.n*)
by (*intro Fmr.monoid-sum-list-cong Fmr.res-mult-eq*)
also have ... = ($\sum$ *k* ← [*0*..<*2* ^ *oe-n*]. (*map* (*int* ∘ *Nat-LSBF.to-nat*) *A.num-blocks* ! *k* ∗
map (*int* ∘ *Nat-LSBF.to-nat*) *B.num-blocks* ! ((*2* ^ *oe-n* + *j* − *k*) *mod 2* ^ *oe-n*))) *mod Fmr.n*
by (*intro Fmr.monoid-sum-list-eq-sum-list′*)
also have ... = *c′* ! *j mod Fmr.n*
unfolding *c′-nth*[*OF* ‹*j* < *2* ^ *oe-n*›]
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* ‹*j* < *2* ^ *oe-n*›]] **.**
finally show *?thesis* **.**
qed
subgoal for *j* **unfolding** *Fmr.nat-pow-pow*[*OF Fmr.two-in-carrier*]
by (*intro arg-cong2*[**where** *f* = ([^]$_{Fm}$)] *refl mult.commute*)
done
also have ... = ($\bigoplus_{Fm}$*i* ← [*0*..<*2* ^ *oe-n*]. (*z′ i*) $\otimes_{Fm}$ *2* [^]$_{Fm}$ (*i* ∗ *2* ^(*n* − *1*)))

proof −
have $(\bigoplus_{Fm} i \leftarrow [0..<2 \mathbin{\hat{}} oe\text{-}n].\ (z'\ i) \otimes_{Fm} 2\ [\urcorner]_{Fm}\ (i * 2 \mathbin{\hat{}} (n - 1))) =$
$(\bigoplus_{Fm} i \leftarrow [0..<2 \mathbin{\hat{}} oe\text{-}n].\ (z''\ i) \otimes_{Fm} 2\ [\urcorner]_{Fm}\ (i * 2 \mathbin{\hat{}} (n - 1)))$
apply (*intro-cong* [*cong-tag-2* $(\otimes_{Fm})$] *more*: *Fmr.monoid-sum-list-cong refl*)
using z'-z'' **by** *simp*
also have ... =
$(\bigoplus_{Fm} i \leftarrow [0..<2 \mathbin{\hat{}} (oe\text{-}n - 1)].\ (z''\ i) \otimes_{Fm} 2\ [\urcorner]_{Fm}\ (i * 2 \mathbin{\hat{}} (n - 1)))$
$\oplus_{Fm}$
$2\ [\urcorner]_{Fm}\ ((2::nat) \mathbin{\hat{}} (oe\text{-}n - 1) * 2 \mathbin{\hat{}} (n - 1)) \otimes_{Fm} (\bigoplus_{Fm} i \leftarrow [0..<2 \mathbin{\hat{}} (oe\text{-}n - 1)].\ (z''\ (2 \mathbin{\hat{}} (oe\text{-}n - 1) + i)) \otimes_{Fm} 2\ [\urcorner]_{Fm}\ (i * 2 \mathbin{\hat{}} (n - 1)))$
apply (*intro Fmr.monoid-pow-sum-split two-pow-oe-n-as-halves*[*symmetric*] z'*-carrier Fmr.two-in-carrier*)
by *assumption*
also have ... = $(\bigoplus_{Fm} i \leftarrow [0..<2 \mathbin{\hat{}} (oe\text{-}n - 1)].\ (c'\ !\ i \ominus_{Fm} c'\ !\ (2 \mathbin{\hat{}} (oe\text{-}n - 1) + i) \oplus_{Fm} 2\ [\urcorner]_{Fm}\ (oe\text{-}n + 2 \mathbin{\hat{}} n)) \otimes_{Fm} 2\ [\urcorner]_{Fm}\ (i * 2 \mathbin{\hat{}} (n - 1))) \oplus_{Fm}$
$2\ [\urcorner]_{Fm}\ ((2::nat) \mathbin{\hat{}} (oe\text{-}n - 1) * 2 \mathbin{\hat{}} (n - 1)) \otimes_{Fm} (\bigoplus_{Fm} i \leftarrow [0..<2 \mathbin{\hat{}} (oe\text{-}n - 1)].\ 2\ [\urcorner]_{Fm}\ (oe\text{-}n + 2 \mathbin{\hat{}} n) \otimes_{Fm} 2\ [\urcorner]_{Fm}\ (i * 2 \mathbin{\hat{}} (n - 1)))$
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 ... = $(\bigoplus_{Fm} i \leftarrow [0..<2 \mathbin{\hat{}} (oe\text{-}n - 1)].$
$(c'\ !\ i \ominus_{Fm} c'\ !\ (2 \mathbin{\hat{}} (oe\text{-}n - 1) + i) \oplus_{Fm} 2\ [\urcorner]_{Fm}\ (oe\text{-}n + 2 \mathbin{\hat{}} n)) \otimes_{Fm} 2\ [\urcorner]_{Fm}\ (i * 2 \mathbin{\hat{}} (n - 1)))$
$\oplus_{Fm} 2\ [\urcorner]_{Fm}\ ((2::nat) \mathbin{\hat{}} m) \otimes_{Fm}$
$(\bigoplus_{Fm} i \leftarrow [0..<2 \mathbin{\hat{}} (oe\text{-}n - 1)].$
$2\ [\urcorner]_{Fm}\ (oe\text{-}n + 2 \mathbin{\hat{}} n) \otimes_{Fm} 2\ [\urcorner]_{Fm}\ (i * 2 \mathbin{\hat{}} (n - 1)))$
apply (*intro-cong* [*cong-tag-2* $(\oplus_{Fm})$, *cong-tag-2* $(\otimes_{Fm})$, *cong-tag-2* $([\urcorner]_{Fm})$] *more*: *refl*)
using *two-pow-m0-as-prod* **by** *simp*
also have ... = $(\bigoplus_{Fm} i \leftarrow [0..<2 \mathbin{\hat{}} (oe\text{-}n - 1)].$
$(c'\ !\ i \ominus_{Fm} (c'\ !\ (2 \mathbin{\hat{}} (oe\text{-}n - 1) + i) \ominus_{Fm} 2\ [\urcorner]_{Fm}\ (oe\text{-}n + 2 \mathbin{\hat{}} n))) \otimes_{Fm} 2\ [\urcorner]_{Fm}\ (i * 2 \mathbin{\hat{}} (n - 1)))$
$\oplus_{Fm} 2\ [\urcorner]_{Fm}\ ((2::nat) \mathbin{\hat{}} m) \otimes_{Fm}$
$(\bigoplus_{Fm} i \leftarrow [0..<2 \mathbin{\hat{}} (oe\text{-}n - 1)].$
$2\ [\urcorner]_{Fm}\ (oe\text{-}n + 2 \mathbin{\hat{}} n) \otimes_{Fm} 2\ [\urcorner]_{Fm}\ (i * 2 \mathbin{\hat{}} (n - 1)))$
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 ... = $(\bigoplus_{Fm} i \leftarrow [0..<2 \mathbin{\hat{}} (oe\text{-}n - 1)].\ c'\ !\ i \otimes_{Fm} 2\ [\urcorner]_{Fm}\ (i * 2 \mathbin{\hat{}} (n - 1)))$
$\ominus_{Fm}$
$(\bigoplus_{Fm} i \leftarrow [0..<2 \mathbin{\hat{}} (oe\text{-}n - 1)].$
$(c'\ !\ (2 \mathbin{\hat{}} (oe\text{-}n - 1) + i) \ominus_{Fm} 2\ [\urcorner]_{Fm}\ (oe\text{-}n + 2 \mathbin{\hat{}} n)) \otimes_{Fm} 2\ [\urcorner]_{Fm}\ (i * 2 \mathbin{\hat{}} (n - 1)))$
$\oplus_{Fm} 2\ [\urcorner]_{Fm}\ ((2::nat) \mathbin{\hat{}} m) \otimes_{Fm}$
$(\bigoplus_{Fm} i \leftarrow [0..<2 \mathbin{\hat{}} (oe\text{-}n - 1)].$
$2\ [\urcorner]_{Fm}\ (oe\text{-}n + 2 \mathbin{\hat{}} n) \otimes_{Fm} 2\ [\urcorner]_{Fm}\ (i * 2 \mathbin{\hat{}} (n - 1)))$
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 ... = ($\bigoplus_{Fm} i \leftarrow [0..<2\ \widehat{}\ (oe\text{-}n - 1)].\ c'\ !\ i \otimes_{Fm} 2\ [\widehat{}]_{Fm}\ (i * 2\ \widehat{}\ (n - 1)))$
$\oplus_{Fm} (\ominus_{Fm} \mathbf{1}_{Fm}) \otimes_{Fm}$
$(\bigoplus_{Fm} i \leftarrow [0..<2\ \widehat{}\ (oe\text{-}n - 1)].$
$(c'\ !\ (2\ \widehat{}\ (oe\text{-}n - 1) + i) \ominus_{Fm} 2\ [\widehat{}]_{Fm}\ (oe\text{-}n + 2\ \widehat{}\ n)) \otimes_{Fm} 2\ [\widehat{}]_{Fm}\ (i * 2\ \widehat{}\ (n - 1)))$
$\oplus_{Fm} 2\ [\widehat{}]_{Fm}\ ((2::nat)\ \widehat{}\ m) \otimes_{Fm}$
$(\bigoplus_{Fm} i \leftarrow [0..<2\ \widehat{}\ (oe\text{-}n - 1)].$
$2\ [\widehat{}]_{Fm}\ (oe\text{-}n + 2\ \widehat{}\ n) \otimes_{Fm} 2\ [\widehat{}]_{Fm}\ (i * 2\ \widehat{}\ (n - 1)))$
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 ... = ($\bigoplus_{Fm} i \leftarrow [0..<2\ \widehat{}\ (oe\text{-}n - 1)].\ c'\ !\ i \otimes_{Fm} 2\ [\widehat{}]_{Fm}\ (i * 2\ \widehat{}\ (n - 1)))$
$\oplus_{Fm} (2\ [\widehat{}]_{Fm}\ ((2::nat)\ \widehat{}\ m)) \otimes_{Fm}$
$(\bigoplus_{Fm} i \leftarrow [0..<2\ \widehat{}\ (oe\text{-}n - 1)].$
$(c'\ !\ (2\ \widehat{}\ (oe\text{-}n - 1) + i) \ominus_{Fm} 2\ [\widehat{}]_{Fm}\ (oe\text{-}n + 2\ \widehat{}\ n)) \otimes_{Fm} 2\ [\widehat{}]_{Fm}\ (i * 2\ \widehat{}\ (n - 1)))$
$\oplus_{Fm} 2\ [\widehat{}]_{Fm}\ ((2::nat)\ \widehat{}\ m) \otimes_{Fm}$
$(\bigoplus_{Fm} i \leftarrow [0..<2\ \widehat{}\ (oe\text{-}n - 1)].$
$2\ [\widehat{}]_{Fm}\ (oe\text{-}n + 2\ \widehat{}\ n) \otimes_{Fm} 2\ [\widehat{}]_{Fm}\ (i * 2\ \widehat{}\ (n - 1)))$
using *Fmr.two-pow-half-carrier-length-residue-ring* **by** *argo*
also have ... = ($\bigoplus_{Fm} i \leftarrow [0..<2\ \widehat{}\ (oe\text{-}n - 1)].\ c'\ !\ i \otimes_{Fm} 2\ [\widehat{}]_{Fm}\ (i * 2\ \widehat{}\ (n - 1)))$
$\oplus_{Fm} ((2\ [\widehat{}]_{Fm}\ ((2::nat)\ \widehat{}\ m)) \otimes_{Fm}$
$(\bigoplus_{Fm} i \leftarrow [0..<2\ \widehat{}\ (oe\text{-}n - 1)].$
$(c'\ !\ (2\ \widehat{}\ (oe\text{-}n - 1) + i) \ominus_{Fm} 2\ [\widehat{}]_{Fm}\ (oe\text{-}n + 2\ \widehat{}\ n)) \otimes_{Fm} 2\ [\widehat{}]_{Fm}\ (i * 2\ \widehat{}\ (n - 1)))$
$\oplus_{Fm} 2\ [\widehat{}]_{Fm}\ ((2::nat)\ \widehat{}\ m) \otimes_{Fm}$
$(\bigoplus_{Fm} i \leftarrow [0..<2\ \widehat{}\ (oe\text{-}n - 1)].$
$2\ [\widehat{}]_{Fm}\ (oe\text{-}n + 2\ \widehat{}\ n) \otimes_{Fm} 2\ [\widehat{}]_{Fm}\ (i * 2\ \widehat{}\ (n - 1))))$
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 ... = ($\bigoplus_{Fm} i \leftarrow [0..<2\ \widehat{}\ (oe\text{-}n - 1)].\ c'\ !\ i \otimes_{Fm} 2\ [\widehat{}]_{Fm}\ (i * 2\ \widehat{}\ (n - 1)))$
$\oplus_{Fm} ((2\ [\widehat{}]_{Fm}\ ((2::nat)\ \widehat{}\ m)) \otimes_{Fm}$
$((\bigoplus_{Fm} i \leftarrow [0..<2\ \widehat{}\ (oe\text{-}n - 1)].$
$(c'\ !\ (2\ \widehat{}\ (oe\text{-}n - 1) + i) \ominus_{Fm} 2\ [\widehat{}]_{Fm}\ (oe\text{-}n + 2\ \widehat{}\ n)) \otimes_{Fm} 2\ [\widehat{}]_{Fm}\ (i * 2\ \widehat{}\ (n - 1)))$
$\oplus_{Fm}$
$(\bigoplus_{Fm} i \leftarrow [0..<2\ \widehat{}\ (oe\text{-}n - 1)].$
$2\ [\widehat{}]_{Fm}\ (oe\text{-}n + 2\ \widehat{}\ n) \otimes_{Fm} 2\ [\widehat{}]_{Fm}\ (i * 2\ \widehat{}\ (n - 1)))))$
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 ... = ($\bigoplus_{Fm}$*i* ← [*0..<2* ^ (*oe-n* − *1*)]. *c′* ! *i* $\otimes_{Fm}$ *2* [^]$_{Fm}$ (*i* ∗ *2* ^ (*n* − *1*)))
$\oplus_{Fm}$ ((*2* [^]$_{Fm}$ ((*2::nat*) ^ *m*)) $\otimes_{Fm}$
($\bigoplus_{Fm}$*i* ← [*0..<2* ^ (*oe-n* − *1*)].
(*c′* ! (*2* ^ (*oe-n* − *1*) + *i*) $\ominus_{Fm}$ *2* [^]$_{Fm}$ (*oe-n* + *2* ^ *n*) $\oplus_{Fm}$ *2* [^]$_{Fm}$ (*oe-n* + *2* ^ *n*)) $\otimes_{Fm}$ *2* [^]$_{Fm}$ (*i* ∗ *2* ^ (*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 ... = ($\bigoplus_{Fm}$*i* ← [*0..<2* ^ (*oe-n* − *1*)]. *c′* ! *i* $\otimes_{Fm}$ *2* [^]$_{Fm}$ (*i* ∗ *2* ^ (*n* − *1*)))
$\oplus_{Fm}$ ((*2* [^]$_{Fm}$ ((*2::nat*) ^ *m*)) $\otimes_{Fm}$
($\bigoplus_{Fm}$*i* ← [*0..<2* ^ (*oe-n* − *1*)].
(*c′* ! (*2* ^ (*oe-n* − *1*) + *i*)) $\otimes_{Fm}$ *2* [^]$_{Fm}$ (*i* ∗ *2* ^ (*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 ... = ($\bigoplus_{Fm}$*i* ← [*0..<2* ^ (*oe-n* − *1*)]. *c′* ! *i* $\otimes_{Fm}$ *2* [^]$_{Fm}$ (*i* ∗ *2* ^ (*n* − *1*)))
$\oplus_{Fm}$ ((*2* [^]$_{Fm}$ ((*2::nat*) ^ (*oe-n* − *1*) ∗ *2* ^ (*n* − *1*))) $\otimes_{Fm}$
($\bigoplus_{Fm}$*i* ← [*0..<2* ^ (*oe-n* − *1*)].
(*c′* ! (*2* ^ (*oe-n* − *1*) + *i*)) $\otimes_{Fm}$ *2* [^]$_{Fm}$ (*i* ∗ *2* ^ (*n* − *1*))))
using *two-pow-m0-as-prod* **by** (*simp add*: *mult.commute*)
also have ... = ($\bigoplus_{Fm}$*i* ← [*0..<2* ^ *oe-n*]. *c′* ! *i* $\otimes_{Fm}$ *2* [^]$_{Fm}$ (*i* ∗ *2* ^ (*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*
= ($\bigoplus_{Fm}$*i* ← [*0..<2* ^ *oe-n*]. (*z′ i*) $\otimes_{Fm}$ *2* [^]$_{Fm}$ (*i* ∗ *2* ^ (*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*) ≤ *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 ‹$karatsuba\text{-}mul\text{-}nat\ A.num\text{-}Zn\text{-}pad\ B.num\text{-}Zn\text{-}pad = uv'\ @\ f$›
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** $\gamma' \equiv \lambda\tau.\ (\sum \sigma \leftarrow [0..<2\ \hat{}\ oe\text{-}n].\ \sum \varrho \leftarrow [0..<2\ \hat{}\ oe\text{-}n].\ of\text{-}bool\ (\tau = \sigma + \varrho) * (Nat\text{-}LSBF.to\text{-}nat\ (A.num\text{-}Zn\ !\ \sigma) * Nat\text{-}LSBF.to\text{-}nat\ (B.num\text{-}Zn\ !\ \varrho)))$

have *to-nat-γ*: $Nat\text{-}LSBF.to\text{-}nat\ (\gamma\ !\ i\ !\ j) = \gamma'\ (i * 2\ \hat{}\ (oe\text{-}n - 1) + j)$
if $i < 4\ j < 2\ \hat{}\ (oe\text{-}n - 1)$ **for** $i\ j$
proof −
have $Nat\text{-}LSBF.to\text{-}nat\ uv = (\sum j \leftarrow [0..<2\ \hat{}\ (oe\text{-}n + 1)].\ Nat\text{-}LSBF.to\text{-}nat\ (subdivide\ pad\text{-}length\ uv\ !\ j) * 2\ \hat{}\ (j * pad\text{-}length))$
apply (*intro to-nat-subdivide pad-length-gt-0*)
unfolding *length-uv uv-length-def* **by** (*rule refl*)
also have ... $= (\sum j \leftarrow [0..<2\ \hat{}\ (oe\text{-}n + 1)].$
$Nat\text{-}LSBF.to\text{-}nat\ (\gamma\ !\ (j\ div\ 2\ \hat{}\ (oe\text{-}n - 1))\ !\ (j\ mod\ 2\ \hat{}\ (oe\text{-}n - 1)))$
$* 2\ \hat{}\ (j * pad\text{-}length))$
apply (*intro-cong* [*cong-tag-2* (∗), *cong-tag-1 Nat-LSBF.to-nat*] *more*: *semiring-1-sum-list-eq refl*)
apply (*intro γ-nth'*[*symmetric*]) **by** *simp*
finally have *1*: $Nat\text{-}LSBF.to\text{-}nat\ uv = ...$ **.**

let $?exp = \lambda i.\ 2\ \hat{}\ (i * pad\text{-}length)$
let $?a = \lambda i.\ Nat\text{-}LSBF.to\text{-}nat\ (A.num\text{-}Zn\ !\ i)$
let $?b = \lambda i.\ Nat\text{-}LSBF.to\text{-}nat\ (B.num\text{-}Zn\ !\ i)$
from *A.to-nat-num-Zn-pad B.to-nat-num-Zn-pad*
have $Nat\text{-}LSBF.to\text{-}nat\ uv =$
$(\sum i \leftarrow [0..<2\ \hat{}\ oe\text{-}n].\ ?a\ i * ?exp\ i) *$
$(\sum j \leftarrow [0..<2\ \hat{}\ oe\text{-}n].\ ?b\ j * ?exp\ j)$
using ‹$Nat\text{-}LSBF.to\text{-}nat\ uv = Nat\text{-}LSBF.to\text{-}nat\ A.num\text{-}Zn\text{-}pad * Nat\text{-}LSBF.to\text{-}nat\ B.num\text{-}Zn\text{-}pad$›
by *argo*
also have ... $= (\sum i \leftarrow [0..<2\ \hat{}\ oe\text{-}n].\ \sum j \leftarrow [0..<2\ \hat{}\ oe\text{-}n].\ (?a\ i * ?exp\ i) * (?b\ j * ?exp\ j))$
by (*rule sum-list-mult-sum-list*)
also have ... $= (\sum i \leftarrow [0..<2\ \hat{}\ oe\text{-}n].\ \sum j \leftarrow [0..<2\ \hat{}\ oe\text{-}n].\ (?a\ i * ?b\ j) * ?exp\ (i + j))$
by (*intro sum-list-eq*; *simp add*: *algebra-simps power-add*)
also have ... $= (\sum i \leftarrow [0..<2\ \hat{}\ oe\text{-}n].\ \sum j \leftarrow [0..<2\ \hat{}\ oe\text{-}n].\ \sum k \leftarrow [0..<2\ \hat{}\ (oe\text{-}n + 1) - 1].$
$of\text{-}bool\ (k = i + j) * ((?a\ i * ?b\ j) * ?exp\ (i + j)))$
by (*intro sum-list-eq of-bool-distinct-in*[*symmetric*]; *simp*)
also have ... $= (\sum i \leftarrow [0..<2\ \hat{}\ oe\text{-}n].\ \sum j \leftarrow [0..<2\ \hat{}\ oe\text{-}n].\ \sum k \leftarrow [0..<2$

^ (oe-n + 1) − 1].
of-bool (k = i + j) ∗ ((?a i ∗ ?b j) ∗ ?exp k))
by (intro sum-list-eq[**where** xs = [0..<2 ^ oe-n]] of-bool-var-swap[symmetric])
also have ... = (∑ k ← [0..<2 ^ (oe-n + 1) − 1]. ∑ i ← [0..<2 ^ oe-n].
∑ j ← [0..<2 ^ oe-n].
of-bool (k = i + j) ∗ ((?a i ∗ ?b j) ∗ ?exp k))
by (simp only: sum-swap[**where** ys = [0..<2 ^ (oe-n + 1) − 1]])
also have ... = (∑ k ← [0..<2 ^ (oe-n + 1) − 1]. γ′ k ∗ ?exp k)
apply (unfold γ′-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 ... = (∑ k ← [0..<2 ^ (oe-n + 1)]. γ′ k ∗ 2 ^ (k ∗ pad-length))
proof −
have (∑ k ← [0..<2 ^ (oe-n + 1)]. γ′ k ∗ 2 ^ (k ∗ pad-length)) =
(∑ k ← [0..<2 ^ (oe-n + 1) − 1]. γ′ k ∗ 2 ^ (k ∗ pad-length)) + γ′ (2 ^
(oe-n + 1) − 1) ∗ 2 ^ ((2 ^ (oe-n + 1) − 1) ∗ pad-length)
apply (intro sum-list-split-Suc) **by** simp
also have γ′ (2 ^ (oe-n + 1) − 1) = (∑ i ← [0..<2 ^ oe-n]. ∑ j ← [0..<2
^ oe-n]. 0)
unfolding γ′-def
proof (intro semiring-1-sum-list-eq)
fix i j :: nat
assume i ∈ set [0..<2 ^ oe-n] j ∈ set [0..<2 ^ oe-n]
then have i + j ≤ (2 ^ oe-n − 1) + (2 ^ oe-n − 1) **by** simp
also have ... = 2 ^ (oe-n + 1) − 2 **by** simp
also have ... < 2 ^ (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 ^ (oe-n + 1) − 1 ≠ i + j **by** simp
then have of-bool (2 ^ (oe-n + 1) − 1 = i + j) = 0 **by** simp
then show of-bool (2 ^ (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 = ... **.**
with 1 **have** 2: (∑ j ← [0..<2 ^ (oe-n + 1)].
Nat-LSBF.to-nat (γ ! (j div 2 ^ (oe-n − 1)) ! (j mod 2 ^ (oe-n − 1)))
∗ 2 ^ (j ∗ pad-length)) = ... **by** argo
have ⋀j. j < 2 ^ (oe-n + 1) ⟹
Nat-LSBF.to-nat (γ ! (j div 2 ^ (oe-n − 1)) ! (j mod 2 ^ (oe-n − 1))) = γ′ j
apply (intro power-sum-nat-eq[**where** n = 2 ^ (oe-n + 1) **and** g = γ′ **and**
x = 2 **and** c = pad-length])
subgoal by simp

subgoal by (*rule pad-length-gt-0*)
subgoal for *i j*
proof −
assume $j < 2 \widehat{\ } (oe\text{-}n + 1)$
then have *Nat-LSBF.to-nat* (γ ! (j *div* $2 \widehat{\ } (oe\text{-}n - 1)$) ! ($j$ *mod* $2 \widehat{\ } (oe\text{-}n - 1)$)) = *Nat-LSBF.to-nat* (*subdivide pad-length uv* ! *j*)
apply (*intro arg-cong*[**where** *f* = *Nat-LSBF.to-nat*] γ*-nth*$'$) **.**
also have ... $< 2 \widehat{\ }$(*length* (*subdivide pad-length uv* ! *j*))
by (*intro to-nat-length-bound*)
also have ... $= 2 \widehat{\ }$ *pad-length*
apply (*intro arg-cong*[**where** *f* = *power 2*] *scuv*(*2*) *nth-mem*)
using ‹$j < 2 \widehat{\ } (oe\text{-}n + 1)$› *scuv*(*1*) **by** *argo*
finally show *Nat-LSBF.to-nat* (γ ! (j *div* $2 \widehat{\ } (oe\text{-}n - 1)$) ! ($j$ *mod* $2 \widehat{\ } (oe\text{-}n - 1)$)) $< 2 \widehat{\ }$ *pad-length* **.**
qed
subgoal for *i j*
proof −
have γ' *j* = ($\sum\sigma\leftarrow[0..<2 \widehat{\ } oe\text{-}n]$. $\sum\varrho\leftarrow[0..<2 \widehat{\ } oe\text{-}n]$. *of-bool* ($j = \sigma + \varrho$) $*$ (*Nat-LSBF.to-nat* (*A.num-Zn* ! σ) $*$ *Nat-LSBF.to-nat* (*B.num-Zn* ! ϱ)))
unfolding γ'*-def* **by** *argo*
also have ... $\leq$ ($\sum\sigma\leftarrow[0..<2 \widehat{\ } oe\text{-}n]$. $\sum\varrho\leftarrow[0..<2 \widehat{\ } oe\text{-}n]$. *of-bool* ($j = \sigma + \varrho$) $*$ (($2 \widehat{\ } (n + 2) - 1$) $*$ ($2 \widehat{\ } (n + 2) - 1$)))
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 ... = ($\sum\sigma\leftarrow[0..<2 \widehat{\ } oe\text{-}n]$. $\sum\varrho\leftarrow[0..<2 \widehat{\ } oe\text{-}n]$. *of-bool* ($j = \sigma + \varrho$)) $*$ (($2 \widehat{\ } (n + 2) - 1$) $*$ ($2 \widehat{\ } (n + 2) - 1$))
by (*simp add*: *sum-list-mult-const*)
also have ... $\leq$ ($\sum\sigma\leftarrow[0..<2 \widehat{\ } oe\text{-}n]$. *1*) $*$ (($2 \widehat{\ } (n + 2) - 1$) $*$ ($2 \widehat{\ } (n + 2) - 1$))
apply (*intro mult-le-mono1 sum-list-mono*)
subgoal for σ
by (*intro of-bool-sum-leq-1*) *simp-all*
done
also have ... $= 2 \widehat{\ } oe\text{-}n * ((2 \widehat{\ } (n + 2) - 1) * (2 \widehat{\ } (n + 2) - 1))$
apply (*intro arg-cong2*[**where** *f* = ($*$)] *refl*)
using *sum-list-triv*[*of* (*1*::*nat*) [$0..<2 \widehat{\ } oe\text{-}n$]] **by** *simp*
also have ... $< 2 \widehat{\ } oe\text{-}n * (2 \widehat{\ } (n + 2) * 2 \widehat{\ } (n + 2))$
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 Nat-LSBF.to-nat (γ ! i ! j) = γ' (i * 2 ^ (oe-n − 1) + j)
      using index-comp that by metis
  qed
  have γc: [int (γ' τ) + int (γ' (2 ^ oe-n + τ)) = c' ! τ] (mod 2 ^ (n + 2))
    if τ < 2 ^ oe-n for τ
  proof −
    have c' ! τ mod 2 ^ (n + 2) = (∑σ←[0..<2 ^ oe-n]. ∑ϱ←[0..<2 ^ oe-n].
        of-bool [τ = σ + ϱ] (mod 2 ^ oe-n) *
      (int (Nat-LSBF.to-nat (A.num-blocks ! σ)) * int (Nat-LSBF.to-nat (B.num-blocks
! ϱ)))) mod 2 ^ (n + 2)
      by (intro arg-cong[where f = λj. j mod -] c'-alt[OF that])
    also have ... = (∑σ←[0..<2 ^ oe-n]. ∑ϱ←[0..<2 ^ oe-n].
        (of-bool [τ = σ + ϱ] (mod 2 ^ oe-n) *
          ((int (Nat-LSBF.to-nat (A.num-blocks ! σ)) mod 2 ^ (n + 2)) * (int
(Nat-LSBF.to-nat (B.num-blocks ! ϱ)) mod 2 ^ (n + 2))))) mod 2 ^ (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.num-blocks ! -)) - int (Nat-LSBF.to-nat (B.num-blocks ! -))]
      by metis
    also have (∑σ←[0..<2 ^ oe-n]. ∑ϱ←[0..<2 ^ oe-n].
        (of-bool [τ = σ + ϱ] (mod 2 ^ oe-n) *
          ((int (Nat-LSBF.to-nat (A.num-blocks ! σ)) mod 2 ^ (n + 2)) * (int
(Nat-LSBF.to-nat (B.num-blocks ! ϱ)) mod 2 ^ (n + 2))))) =
      (∑σ←[0..<2 ^ oe-n]. ∑ϱ←[0..<2 ^ oe-n].
        (of-bool [τ = σ + ϱ] (mod 2 ^ oe-n) *
      ((int (Nat-LSBF.to-nat (A.num-Zn ! σ))) * (int (Nat-LSBF.to-nat (B.num-Zn
! ϱ))))))
      apply (intro-cong [cong-tag-2 (*)] more: semiring-1-sum-list-eq refl)
      unfolding A.num-Zn-def B.num-Zn-def
      subgoal for σ ϱ
        using A.length-num-blocks
        using Znr.to-nat-reduce
        by (simp add: zmod-int)
      subgoal for σ ϱ
        using B.length-num-blocks Znr.to-nat-reduce
        by (simp add: zmod-int)
      done
    also have ... = (∑σ←[0..<2 ^ oe-n]. ∑ϱ←[0..<2 ^ oe-n].
        of-bool (τ = σ + ϱ ∨ τ + 2 ^ oe-n = σ + ϱ) *
```

(*int* (*Nat-LSBF.to-nat* (*A.num-Zn* ! σ)) $*$ *int* (*Nat-LSBF.to-nat* (*B.num-Zn* ! ϱ))))
proof (*intro-cong* [*cong-tag-2* ($*$), *cong-tag-1 of-bool*] *more*: *semiring-1-sum-list-eq refl*)
fix σ ϱ :: *nat*
assume *a*: $\sigma \in$ *set* [*0*..<*2* ^ *oe-n*] $\varrho \in$ *set* [*0*..<*2* ^ *oe-n*]
have [$\tau = \sigma + \varrho$] (*mod 2* ^ *oe-n*) $\Longrightarrow \tau = \sigma + \varrho \vee \tau +$ *2* ^ *oe-n* $= \sigma + \varrho$
proof −
assume [$\tau = \sigma + \varrho$] (*mod 2* ^ *oe-n*)
then have τ *mod* (*2* ^ *oe-n*) = ($\sigma + \varrho$) *mod* (*2* ^ *oe-n*)
unfolding *cong-def* **.**
then have τ = ($\sigma + \varrho$) *mod* (*2* ^ *oe-n*)
using *mod-less* ‹τ < *2* ^ *oe-n*› **by** *simp*
define *i* **where** *i* = ($\sigma + \varrho$) *div* (*2* ^ *oe-n*)
have τ + *i* $*$ *2* ^ *oe-n* = $\sigma + \varrho$
unfolding ‹τ = ($\sigma + \varrho$) *mod* (*2* ^ *oe-n*)› *i-def*
by (*simp add*: *mod-div-mult-eq*)
moreover have *i* ≤ *1*
proof (*rule ccontr*)
assume ¬ *i* ≤ *1*
then have *i* ≥ *2* **by** *simp*
then have $\sigma + \varrho \geq$ *2* ^(*oe-n* + *1*)
using ‹τ + *i* $*$ *2* ^ *oe-n* = $\sigma + \varrho$›
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* ∨ *i* = *1* **by** *linarith*
ultimately show *?thesis* **by** *auto*
qed
moreover have $\tau = \sigma + \varrho \vee \tau +$ *2* ^ *oe-n* $= \sigma + \varrho \Longrightarrow$ [$\tau = \sigma + \varrho$] (*mod 2* ^ *oe-n*)
by (*metis cong-add-lcancel-0-nat cong-refl cong-sym cong-to-1′-nat mult-1*)
ultimately show [$\tau = \sigma + \varrho$] (*mod 2* ^ *oe-n*) = ($\tau = \sigma + \varrho \vee \tau +$ *2* ^ *oe-n* $= \sigma + \varrho$) **by** *argo*
qed
also have ... = ($\sum\sigma\leftarrow$[*0*..<*2* ^ *oe-n*]. $\sum\varrho\leftarrow$[*0*..<*2* ^ *oe-n*].
(*of-bool* ($\tau = \sigma + \varrho$) + *of-bool* (τ + *2* ^ *oe-n* = $\sigma + \varrho$)) $*$
(*int* (*Nat-LSBF.to-nat* (*A.num-Zn* ! σ)) $*$ *int* (*Nat-LSBF.to-nat* (*B.num-Zn* ! ϱ))))
apply (*intro-cong* [*cong-tag-2* ($*$)] *more*: *semiring-1-sum-list-eq refl of-bool-disj-excl*)
by *simp*
also have ... = ($\sum\sigma\leftarrow$[*0*..<*2* ^ *oe-n*]. $\sum\varrho\leftarrow$[*0*..<*2* ^ *oe-n*].
of-bool ($\tau = \sigma + \varrho$) $*$ (*int* (*Nat-LSBF.to-nat* (*A.num-Zn* ! σ)) $*$ *int* (*Nat-LSBF.to-nat* (*B.num-Zn* ! ϱ)))) +
($\sum\sigma\leftarrow$[*0*..<*2* ^ *oe-n*]. $\sum\varrho\leftarrow$[*0*..<*2* ^ *oe-n*].
of-bool (τ + *2* ^ *oe-n* = $\sigma + \varrho$) $*$ (*int* (*Nat-LSBF.to-nat* (*A.num-Zn* ! σ)) $*$ *int* (*Nat-LSBF.to-nat* (*B.num-Zn* ! ϱ))))
by (*simp add*: *int-distrib*(*1*) *sum-list-addf*)
also have ... = ($\sum\sigma\leftarrow$[*0*..<*2* ^ *oe-n*]. $\sum\varrho\leftarrow$[*0*..<*2* ^ *oe-n*].

int (*of-bool* (τ = σ + ϱ) $*$ ((*Nat-LSBF.to-nat* (*A.num-Zn* ! σ) $*$ *Nat-LSBF.to-nat* (*B.num-Zn* ! ϱ))))) +
($\sum$ σ←[*0*..<*2* ^ *oe-n*]. $\sum$ ϱ←[*0*..<*2* ^ *oe-n*].
int (*of-bool* (τ + *2* ^ *oe-n* = σ + ϱ) $*$ ((*Nat-LSBF.to-nat* (*A.num-Zn* ! σ) $*$ (*Nat-LSBF.to-nat* (*B.num-Zn* ! ϱ))))))
apply (*intro-cong* [*cong-tag-2* (+)] *more*: *semiring-1-sum-list-eq*)
by *simp-all*
also have ... = *int* ($\sum$ σ←[*0*..<*2* ^ *oe-n*]. $\sum$ ϱ←[*0*..<*2* ^ *oe-n*].
of-bool (τ = σ + ϱ) $*$ ((*Nat-LSBF.to-nat* (*A.num-Zn* ! σ) $*$ *Nat-LSBF.to-nat* (*B.num-Zn* ! ϱ)))) +
int ($\sum$ σ←[*0*..<*2* ^ *oe-n*]. $\sum$ ϱ←[*0*..<*2* ^ *oe-n*].
of-bool (τ + *2* ^ *oe-n* = σ + ϱ) $*$ ((*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* ^ *oe-n*))
unfolding γ'*-def* **by** *argo*
finally show [*int* (γ' τ) + *int* (γ' (*2* ^ *oe-n* + τ)) = c' ! τ] (*mod 2* ^(*n* + *2*))
unfolding *cong-def* **by** (*simp add*: *add.commute*)
qed
have η*-carrier*: *length* (η ! *j*) = *n* + *2* **if** *j* < *2* ^(*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** *sc*γ *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* ^(*n* + *2*)
if *j* < *2* ^(*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* ^(*oe-n* − *1*)› *sc*γ
by *simp-all*
also have ... =
Znr.to-residue-ring (*Znr.subtract-mod* (*take* (*n* + *2*) (γ ! *0* ! *j*)) (*take* (*n* + *2*) (γ ! *1* ! *j*))) $\oplus_{Zn}$
Znr.to-residue-ring (*Znr.subtract-mod* (*take* (*n* + *2*) (γ ! *2* ! *j*)) (*take* (*n* + *2*) (γ ! *3* ! *j*)))
by (*intro Znr.add-mod-correct*)
also have ... =
(*Znr.to-residue-ring* (*take* (*n* + *2*) (γ ! *0* ! *j*)) $\ominus_{Zn}$
Znr.to-residue-ring (*take* (*n* + *2*) (γ ! *1* ! *j*))) $\oplus_{Zn}$

$(Znr.to\text{-}residue\text{-}ring\ (take\ (n + 2)\ (\gamma\ !\ 2\ !\ j)) \ominus_{Zn}$
$Znr.to\text{-}residue\text{-}ring\ (take\ (n + 2)\ (\gamma\ !\ 3\ !\ j)))$
apply (*intro arg-cong2*[**where** $f = (\oplus_{Zn})$])
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 ... =
$(Znr.to\text{-}residue\text{-}ring\ (take\ (n + 2)\ (\gamma\ !\ 0\ !\ j)) \oplus_{Zn}$
$Znr.to\text{-}residue\text{-}ring\ (take\ (n + 2)\ (\gamma\ !\ 2\ !\ j))) \ominus_{Zn}$
$(Znr.to\text{-}residue\text{-}ring\ (take\ (n + 2)\ (\gamma\ !\ 1\ !\ j)) \oplus_{Zn}$
$Znr.to\text{-}residue\text{-}ring\ (take\ (n + 2)\ (\gamma\ !\ 3\ !\ j)))$
apply (*intro Znr.diff-sum*)
using *Znr.to-residue-ring-in-carrier* **by** *simp-all*
also have ... =
$(int\ (Nat\text{-}LSBF.to\text{-}nat\ (take\ (n + 2)\ (\gamma\ !\ 0\ !\ j)))\ mod\ 2\ \hat{}\ (n + 2) \oplus_{Zn}$
$int\ (Nat\text{-}LSBF.to\text{-}nat\ (take\ (n + 2)\ (\gamma\ !\ 2\ !\ j)))\ mod\ 2\ \hat{}\ (n + 2)) \ominus_{Zn}$
$(int\ (Nat\text{-}LSBF.to\text{-}nat\ (take\ (n + 2)\ (\gamma\ !\ 1\ !\ j)))\ mod\ 2\ \hat{}\ (n + 2) \oplus_{Zn}$
$int\ (Nat\text{-}LSBF.to\text{-}nat\ (take\ (n + 2)\ (\gamma\ !\ 3\ !\ j)))\ mod\ 2\ \hat{}\ (n + 2))$
unfolding *Znr.to-residue-ring-def* **by** *simp*
also have ... =
$(int\ (Nat\text{-}LSBF.to\text{-}nat\ (\gamma\ !\ 0\ !\ j))\ mod\ 2\ \hat{}\ (n + 2) \oplus_{Zn}$
$int\ (Nat\text{-}LSBF.to\text{-}nat\ (\gamma\ !\ 2\ !\ j))\ mod\ 2\ \hat{}\ (n + 2)) \ominus_{Zn}$
$(int\ (Nat\text{-}LSBF.to\text{-}nat\ (\gamma\ !\ 1\ !\ j))\ mod\ 2\ \hat{}\ (n + 2) \oplus_{Zn}$
$int\ (Nat\text{-}LSBF.to\text{-}nat\ (\gamma\ !\ 3\ !\ j))\ mod\ 2\ \hat{}\ (n + 2))$
apply (*intro-cong* [*cong-tag-2* $(\lambda i\ j.\ i \ominus_{Zn} j)$, *cong-tag-2* $(\oplus_{Zn})$])
by (*simp-all add*: *to-nat-take zmod-int*)
also have ... =
$(int\ (\gamma'\ (0 * 2\ \hat{}\ (oe\text{-}n - 1) + j))\ mod\ 2\ \hat{}\ (n + 2) \oplus_{Zn}$
$int\ (\gamma'\ (2 * 2\ \hat{}\ (oe\text{-}n - 1) + j))\ mod\ 2\ \hat{}\ (n + 2)) \ominus_{Zn}$
$(int\ (\gamma'\ (1 * 2\ \hat{}\ (oe\text{-}n - 1) + j))\ mod\ 2\ \hat{}\ (n + 2) \oplus_{Zn}$
$int\ (\gamma'\ (3 * 2\ \hat{}\ (oe\text{-}n - 1) + j))\ mod\ 2\ \hat{}\ (n + 2))$
apply (*intro-cong* [*cong-tag-2* $(\lambda i\ j.\ i \ominus_{Zn} j)$, *cong-tag-2* $(\oplus_{Zn})$, *cong-tag-2* (*mod*), *cong-tag-1 int*] *more*: *refl to-nat-γ* ‹$j < 2\ \hat{}\ (oe\text{-}n - 1)$›)
by *simp-all*
also have ... =
$(int\ (\gamma'\ j)\ mod\ 2\ \hat{}\ (n + 2) \oplus_{Zn}$
$int\ (\gamma'\ (2\ \hat{}\ oe\text{-}n + j))\ mod\ 2\ \hat{}\ (n + 2)) \ominus_{Zn}$
$(int\ (\gamma'\ (2\ \hat{}\ (oe\text{-}n - 1) + j))\ mod\ 2\ \hat{}\ (n + 2) \oplus_{Zn}$
$int\ (\gamma'\ (2\ \hat{}\ oe\text{-}n + (2\ \hat{}\ (oe\text{-}n - 1) + j)))\ mod\ 2\ \hat{}\ (n + 2))$
apply (*intro-cong* [*cong-tag-2* $(\lambda i\ j.\ i \ominus_{Zn} j)$, *cong-tag-2* $(\oplus_{Zn})$, *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 ... =
$(int\ (\gamma'\ j)\ mod\ 2\ \hat{}\ (n + 2) +$
$int\ (\gamma'\ (2\ \hat{}\ oe\text{-}n + j))\ mod\ 2\ \hat{}\ (n + 2))\ mod\ 2\ \hat{}\ (n + 2) \ominus_{Zn}$
$(int\ (\gamma'\ (2\ \hat{}\ (oe\text{-}n - 1) + j))\ mod\ 2\ \hat{}\ (n + 2) +$
$int\ (\gamma'\ (2\ \hat{}\ oe\text{-}n + (2\ \hat{}\ (oe\text{-}n - 1) + j)))\ mod\ 2\ \hat{}\ (n + 2))\ mod\ 2\ \hat{}\ (n +$

2)
apply (*intro-cong* [*cong-tag-2* (λ*i j*. *i* $\ominus_{Zn}$ *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) $\ominus_{Zn}$
(*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* (λ*i j*. *i* $\ominus_{Zn}$ *j*)] *more*: *mod-add-eq*)
also have ... = ((*c*′ ! *j*) *mod* 2 ^ (*n* + 2)) $\ominus_{Zn}$ ((*c*′ ! (2 ^ (*oe-n* − 1) + *j*)) *mod* 2 ^ (*n* + 2))
apply (*intro arg-cong2*[**where** *f* = λ*i j*. *i* $\ominus_{Zn}$ *j*])
using γ*c* **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* (λ*i*. *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 Fn*
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* = ($\sum$ 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* ‹*i* < *2* ^ *oe-n*›] ‹*length c′* = *2* ^ *oe-n*› **by** *simp*
also have ... = ($\bigoplus_{Fn}$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*)
by (*intro Fnr.monoid-sum-list-eq-sum-list′*[*symmetric*])
also have ... = ($\bigoplus_{Fn}$σ←[*0*..< *2* ^ *oe-n*]. ((*int* (*Nat-LSBF.to-nat* (*A.num-blocks* ! σ)) *mod int Fnr.n*) ∗
(*int* (*Nat-LSBF.to-nat* (*B.num-blocks* ! ((*2* ^ *oe-n* + *i* − σ) *mod 2* ^ *oe-n*))) *mod int Fnr.n*))
mod int Fnr.n)
by (*intro Fnr.monoid-sum-list-cong mod-mult-eq*[*symmetric*])
also have ... = ($\bigoplus_{Fn}$σ←[*0*..< *2* ^ *oe-n*]. (*Fnr.to-residue-ring* (*A.num-blocks* ! σ) ∗
Fnr.to-residue-ring (*B.num-blocks* ! ((*2* ^ *oe-n* + *i* − σ) *mod 2* ^ *oe-n*))) *mod int Fnr.n*)
unfolding *Fnr.to-residue-ring.simps* **by** *argo*
also have ... = ($\bigoplus_{Fn}$σ←[*0*..< *2* ^ *oe-n*]. *Fnr.to-residue-ring* (*A.num-blocks* ! σ) $\otimes_{Fn}$
Fnr.to-residue-ring (*B.num-blocks* ! ((*2* ^ *oe-n* + *i* − σ) *mod 2* ^ *oe-n*)))
unfolding *Fnr.res-mult-eq* **by** *argo*
also have ... = ($\bigoplus_{Fn}$σ←[*0*..< *2* ^ *oe-n*]. *map Fnr.to-residue-ring A.num-blocks* ! σ $\otimes_{Fn}$
map Fnr.to-residue-ring B.num-blocks ! ((*2* ^ *oe-n* + *i* − σ) *mod 2* ^ *oe-n*))
apply (*intro-cong* [*cong-tag-2* ($\otimes_{Fn}$)] *more*: *Fnr.monoid-sum-list-cong nth-map*[*symmetric*])
subgoal using *A.length-num-blocks* **by** *simp*
subgoal using *B.length-num-blocks* **by** *simp*
done
also have ... = *Fnr.cyclic-convolution* (*2* ^ *oe-n*) (*map Fnr.to-residue-ring A.num-blocks*)
(*map Fnr.to-residue-ring B.num-blocks*) ! *i*
by (*intro Fnr.cyclic-convolution-nth*[*symmetric*] ‹*i* < *2* ^ *oe-n*›)
finally show *c′-mod* ! *i* =
Fnr.cyclic-convolution (*2* ^ *oe-n*) (*map Fnr.to-residue-ring A.num-blocks*)
(*map Fnr.to-residue-ring B.num-blocks*) ! *i* **.**

```
    qed
    have fill-a': map Fnr.to-residue-ring A.num-blocks = map Fnr.to-residue-ring
(map (fill (2 ^ (n + 1))) A.num-blocks)
      apply (intro nth-equalityI)
      subgoal by simp
      subgoal for i
        unfolding Fnr.to-residue-ring.simps by simp
      done
    have fill-b': map Fnr.to-residue-ring B.num-blocks = map Fnr.to-residue-ring
(map (fill (2 ^ (n + 1))) B.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 μ c'-mod = map2 (⊗Fn) (Fnr.NTT μ (map Fnr.to-residue-ring
(map (fill (2 ^ (n + 1))) A.num-blocks)))
                (Fnr.NTT μ (map Fnr.to-residue-ring (map (fill (2 ^ (n + 1)))
B.num-blocks)))
    proof (intro nth-equalityI)
      show length (Fnr.NTT μ c'-mod) = length (map2 (⊗Fn)
      (Fnr.NTT μ (map Fnr.to-residue-ring (map (fill (2 ^ (n + 1))) A.num-blocks)))
      (Fnr.NTT μ (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 μ c'-mod)
      then have i < 2 ^ oe-n using ‹length c'-mod = 2 ^ oe-n› by simp
      have Fnr.NTT μ c'-mod ! i =
          Fnr.NTT μ (map Fnr.to-residue-ring A.num-blocks) ! i ⊗Fn
          Fnr.NTT μ (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 μ-root-of-unity .
        subgoal using ‹i < 2 ^ oe-n› .
        done
      then show Fnr.NTT μ c'-mod ! i = map2 (⊗Fn)
       (Fnr.NTT μ (map Fnr.to-residue-ring (map (fill (2 ^ (n + 1))) A.num-blocks)))
       (Fnr.NTT μ (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) ⊗int-lsbf-fermat.Fn n
      Fnr.to-residue-ring (evens-odds False B.num-dft ! i) ∧
        schoenhage-strassen n (evens-odds False A.num-dft ! i) (evens-odds False
B.num-dft ! i) ∈ 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) ⊗Fn
          Fnr.to-residue-ring (B.num-dft-odds ! i)
          c-dft-odds ! i ∈ 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) ⊗int-lsbf-fermat.Fn n
        Fnr.to-residue-ring (B.num-dft-odds ! i) ∧
        ... ∈ 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) ⊗Fn
          Fnr.to-residue-ring (B.num-dft-odds ! i)
          c-dft-odds ! i ∈ 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 (⊗Fn)
      (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* (*inv*$_{Fn}$ μ) (*Fnr.NTT* μ c'*-mod*) =
map (($\otimes_{Fn}$) (*Fnr.nat-embedding* (*2* ^ *oe-n*))) c'*-mod*
apply (*intro Fnr.inversion-rule*)
subgoal by *simp*
subgoal using μ*-prim-root* **.**
subgoal premises *prems* **for** *i*
apply (*intro Fnr.sufficiently-good*[*of - - oe-n*])
subgoal using μ*-prim-root* **.**
subgoal using μ*-halfway-property* **by** *blast*
subgoal by (*fact prems*)
done
subgoal using ‹*length* c'*-mod* = *2* ^ *oe-n*› **by** *simp*
subgoal using ‹*set* c'*-mod* $\subseteq$ *carrier Fn*› **.**
done
also have ... = *map* (($\otimes_{Fn}$) (*2* ^ *oe-n mod int Fnr.n*)) c'*-mod*
unfolding *Fnr.nat-embedding-eq* **by** *simp*

also have ... = *map* (($\otimes_{Fn}$) (*2* ^ *oe-n*)) c'*-mod*
unfolding *Fnr.pow-nat-eq*[*symmetric*] *two-oe-n* **by** *argo*
finally have *aux1*: *Fnr.NTT* (*inv*$_{Fn}$ μ) (*Fnr.NTT* μ c'*-mod*) ! *j* =
(*2* ^ *oe-n*) $\otimes_{Fn}$ (c'*-mod* ! *j*) **if** *j* < *2* ^ *oe-n* **for** *j*
using ‹*length* c'*-mod* = *2* ^ *oe-n*› *that* **by** *simp*
have c'*-NTT-NTT-carrier*: *Fnr.NTT* (*inv*$_{Fn}$ μ) (*Fnr.NTT* μ c'*-mod*) ! *j* $\in$ *carrier Fn* **if** *j* < *2* ^ *oe-n* **for** *j*
apply (*intro set-subseteqD*[*OF Fnr.NTT-closed*] *Fnr.NTT-closed Fnr.Units-inv-closed* μ*-Units-Fn* μ*-carrier-Fn* ‹*set* c'*-mod* $\subseteq$ *carrier Fn*›)
by (*simp add*: ‹*length* c'*-mod* = *2* ^ *oe-n*› *that*)
have *aux2*: *inv*$_{Fn}$ (*2* ^ *oe-n*) $\otimes_{Fn}$ *Fnr.NTT* (*inv*$_{Fn}$ μ) (*Fnr.NTT* μ c'*-mod*) ! *j* =
(c'*-mod* ! *j*) **if** *j* < *2* ^ *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* $\ominus_{Fn}$ c'*-mod* ! (*2* ^ (*oe-n* − *1*) + *j*) =
(*inv*$_{Fn}$ *2*) [^]$_{Fn}$ (*oe-n* + *prim-root-exponent* ∗ *j* − *1*) $\otimes_{Fn}$
Fnr.NTT (*inv*$_{Fn}$ μ [^]$_{Fn}$ (*2::nat*)) (*map Fnr.to-residue-ring c-dft-odds*) ! *j*
if *j* < *2* ^ (*oe-n* − *1*) **for** *j*
proof −
have *odd-indices*: $\bigwedge$*i*. *i* < *2* ^ (*oe-n* − *1*) $\Longrightarrow$ (*2::nat*) ∗ *i* + *1* < *2* ^ *oe-n*
proof −

fix i :: *nat*
assume $i < 2 \,\hat{}\,(oe\text{-}n - 1)$
then have $i + 1 \le 2 \,\hat{}\,(oe\text{-}n - 1)$ **by** *simp*
then have $2 * i + 2 \le 2 * 2 \,\hat{}\,(oe\text{-}n - 1)$ **by** *simp*
also have ... $= 2 \,\hat{}\, oe\text{-}n$ **using** *two-pow-oe-n-as-halves* **by** *simp*
finally show $2 * i + 1 < 2 \,\hat{}\, oe\text{-}n$ **by** *simp*
qed
have $c'\text{-}mod$! $j \ominus_{Fn}$ $c'\text{-}mod$! $(2 \,\hat{}\,(oe\text{-}n - 1) + j) =$
inv_{Fn} $(2 \,\hat{}\, oe\text{-}n) \otimes_{Fn}$ $(Fnr.NTT\ (inv_{Fn}\ \mu)\ (Fnr.NTT\ \mu\ c'\text{-}mod)$! $j) \ominus_{Fn}$
inv_{Fn} $(2 \,\hat{}\, oe\text{-}n) \otimes_{Fn}$ $(Fnr.NTT\ (inv_{Fn}\ \mu)\ (Fnr.NTT\ \mu\ c'\text{-}mod)$! $(2 \,\hat{}\,(oe\text{-}n$ $- 1) + j))$
apply (*intro arg-cong2*[**where** $f = \lambda i\ j.\ i \ominus_{Fn} j$])
using *aux2 index-intros*[*OF that*] **by** *simp-all*
also have ... $= inv_{Fn}$ $(2 \,\hat{}\, oe\text{-}n) \otimes_{Fn}$ $(Fnr.NTT\ (inv_{Fn}\ \mu)\ (Fnr.NTT\ \mu$ $c'\text{-}mod)$! $j \ominus_{Fn}$
$Fnr.NTT\ (inv_{Fn}\ \mu)\ (Fnr.NTT\ \mu\ c'\text{-}mod)$! $(2 \,\hat{}\,(oe\text{-}n - 1) +$ $j))$
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 ... $= inv_{Fn}$ $(2 \,\hat{}\, oe\text{-}n) \otimes_{Fn}$ $(Fnr.nat\text{-}embedding\ 2 \otimes_{Fn} (inv_{Fn}\ \mu$ $[\urcorner]_{Fn}\ j \otimes_{Fn}$
$Fnr.NTT\ (inv_{Fn}\ \mu\ [\urcorner]_{Fn}\ (2{::}nat))$
$(map\ ((!)\ (Fnr.NTT\ \mu\ c'\text{-}mod))\ (filter\ odd\ [0..{<}2 \,\hat{}\, oe\text{-}n]))$! $j))$
unfolding *two-pow-oe-n-div-2*[*symmetric*]
apply (*intro arg-cong2*[**where** $f = (\otimes_{Fn})$] *refl Fnr.NTT-diffs*)
subgoal using *oe-n-gt-0* **by** *simp*
subgoal by (*intro Fnr.primitive-root-inv* μ*-prim-root*) *simp*
subgoal by (*simp add*: ‹*length* $c'\text{-}mod = 2 \,\hat{}\, oe\text{-}n$›)
subgoal using ‹$j < 2 \,\hat{}\,(oe\text{-}n - 1)$› **unfolding** ‹$2 \,\hat{}\,(oe\text{-}n - 1) = 2 \,\hat{}\, oe\text{-}n$ $div\ 2$› **.**
subgoal by (*intro Fnr.NTT-closed* ‹*set* $c'\text{-}mod \subseteq$ *carrier Fn*› μ*-carrier-Fn*)
subgoal by (*intro Fnr.inv-halfway-property* μ*-Units-Fn* μ*-halfway-property*)
done
also have ... $= inv_{Fn}$ $(2 \,\hat{}\, oe\text{-}n) \otimes_{Fn}$ $(2 \otimes_{Fn} (inv_{Fn}\ \mu\ [\urcorner]_{Fn}\ j \otimes_{Fn}$
$Fnr.NTT\ (inv_{Fn}\ \mu\ [\urcorner]_{Fn}\ (2{::}nat))$
$(map\ ((!)\ (Fnr.NTT\ \mu\ c'\text{-}mod))\ (filter\ odd\ [0..{<}2 \,\hat{}\, oe\text{-}n]))$! $j))$
using *Fnr.nat-embedding-eq Fnr.carrier-mod-eq*[*OF Fnr.two-in-carrier*] **by** *simp*
also have $Fnr.NTT\ (inv_{Fn}\ \mu\ [\urcorner]_{Fn}\ (2{::}nat))\ (map\ ((!)\ (Fnr.NTT\ \mu\ c'\text{-}mod))$ $(filter\ odd\ [0..{<}2 \,\hat{}\, oe\text{-}n]))$! $j =$
$Fnr.NTT\ (inv_{Fn}\ \mu\ [\urcorner]_{Fn}\ (2{::}nat))$ (
$map\ ((!)\ (map2\ (\otimes_{Fn})$
$(Fnr.NTT\ \mu\ (map\ Fnr.to\text{-}residue\text{-}ring\ A.num\text{-}blocks\text{-}carrier))$
$(Fnr.NTT\ \mu\ (map\ Fnr.to\text{-}residue\text{-}ring\ B.num\text{-}blocks\text{-}carrier))$
))
$(filter\ odd\ [0..{<}2 \,\hat{}\, oe\text{-}n])$

) ! *j*
using *aux0* **unfolding** *A.num-blocks-carrier-def B.num-blocks-carrier-def*
by *argo*
also have ... = *Fnr.NTT* (*inv*$_{Fn}$ μ [^]$_{Fn}$ (*2*::*nat*)) (
map ((!) (*map2* ($\otimes_{Fn}$)
(*map Fnr.to-residue-ring A.num-dft*)
(*map Fnr.to-residue-ring B.num-dft*)
))
(*filter odd* [*0*..<*2* ^ *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* ($\otimes_{Fn}$)
(*map Fnr.to-residue-ring A.num-dft*)
(*map Fnr.to-residue-ring B.num-dft*)
))
(*filter odd* [*0*..<*2* ^ *oe-n*]) =
map2 ($\otimes_{Fn}$) (*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 ($\otimes_{Fn}$) (*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*$_{Fn}$ ((*2*::*int*) ^ *oe-n*) $\otimes_{Fn}$ (*2* $\otimes_{Fn}$ (*inv*$_{Fn}$ μ [^]$_{Fn}$ *j* $\otimes_{Fn}$
Fnr.NTT (*inv*$_{Fn}$ μ [^]$_{Fn}$ (*2*::*nat*)) (*map Fnr.to-residue-ring c-dft-odds*) ! *j*)) =
(*inv*$_{Fn}$ *2*) [^]$_{Fn}$ *oe-n* $\otimes_{Fn}$ ((*inv*$_{Fn}$ *2*) [^]$_{Fn}$ (−*1*::*int*) $\otimes_{Fn}$ ((*inv*$_{Fn}$ *2*) [^]$_{Fn}$ (*prim-root-exponent* ∗ *j*) $\otimes_{Fn}$
Fnr.NTT (*inv*$_{Fn}$ μ [^]$_{Fn}$ (*2*::*nat*)) (*map Fnr.to-residue-ring c-dft-odds*) ! *j*))
apply (*intro-cong* [*cong-tag-2* ($\otimes_{Fn}$)] *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_{Fn} μ $[\urcorner]_{Fn}$ j = inv_{Fn} (μ $[\urcorner]_{Fn}$ j)
using *Fnr.inv-nat-pow*[*OF* μ-*Units-Fn*, *symmetric*] **.**
also have ... = inv_{Fn} (*2* $[\urcorner]_{Fn}$ (*prim-root-exponent* ∗ *j*))
unfolding μ-*def Fnr.nat-pow-pow*[*OF Fnr.two-in-carrier*] **by** *argo*
also have ... = inv_{Fn} *2* $[\urcorner]_{Fn}$ (*prim-root-exponent* ∗ *j*)
using *Fnr.inv-nat-pow*[*OF Fnr.two-is-unit*] **.**
finally show *?thesis* **.**
qed
done
also have ... = inv_{Fn} *2* $[\urcorner]_{Fn}$ *oe-n* $\otimes_{Fn}$ inv_{Fn} *2* $[\urcorner]_{Fn}$ (−*1*::*int*) $\otimes_{Fn}$ inv_{Fn} *2* $[\urcorner]_{Fn}$ (*prim-root-exponent* ∗ *j*) $\otimes_{Fn}$
Fnr.NTT (inv_{Fn} μ $[\urcorner]_{Fn}$ (*2*::*nat*)) (*map Fnr.to-residue-ring c-dft-odds*) ! *j*
apply (*intro Fnr.assoc4*)
subgoal by (*intro Fnr.nat-pow-closed Fnr.Units-inv-closed Fnr.two-is-unit*)
subgoal by (*intro Fnr.Units-closed Fnr.int-pow-closed Fnr.Units-inv-Units Fnr.two-is-unit*)
subgoal by (*intro Fnr.nat-pow-closed Fnr.Units-inv-closed Fnr.two-is-unit*)
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-pow-closed Fnr.Units-inv-Units* μ-*Units-Fn*)
subgoal using ‹*j* < *2* ^(*oe-n* − *1*)› **by** (*simp add*: *length-c-dft-odds*)
done
done
also have inv_{Fn} *2* $[\urcorner]_{Fn}$ *oe-n* $\otimes_{Fn}$ inv_{Fn} *2* $[\urcorner]_{Fn}$ (−*1*::*int*) $\otimes_{Fn}$ inv_{Fn} *2* $[\urcorner]_{Fn}$ (*prim-root-exponent* ∗ *j*) =
inv_{Fn} *2* $[\urcorner]_{Fn}$ (*oe-n* + *prim-root-exponent* ∗ *j* − *1*)
unfolding *int-pow-int*[*symmetric*] *Fnr.int-pow-mult*[*OF Fnr.Units-inv-Units*[*OF Fnr.two-is-unit*]]
apply (*intro arg-cong*[**where** *f* = ($[\urcorner]_{Fn}$) -])
using *oe-n-gt-0* **by** *linarith*
finally show *c′-mod* ! *j* $\ominus_{Fn}$ *c′-mod* ! (*2* ^(*oe-n* − *1*) + *j*) =
inv_{Fn} *2* $[\urcorner]_{Fn}$ (*oe-n* + *prim-root-exponent* ∗ *j* − *1*) $\otimes_{Fn}$
Fnr.NTT (inv_{Fn} μ $[\urcorner]_{Fn}$ (*2*::*nat*)) (*map Fnr.to-residue-ring c-dft-odds*) ! *j*
.
qed
have *c-dft-odds-carrier*: *set c-dft-odds* ⊆ *Fnr.fermat-non-unique-carrier*
unfolding *c-dft-odds-def*
apply (*intro set-subseteqI*)
using *conjunct2*[*OF IH-inst*] *A.length-num-dft-odds B.length-num-dft-odds*
unfolding *A.num-dft-odds-def B.num-dft-odds-def*
by *simp*
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-ifft-carrier*[*of - oe-n − 1*]])
subgoal using *length-c-dft-odds* **.**
subgoal using *c-dft-odds-carrier* **.**
subgoal using *Fnr.length-ifft*[*OF length-c-dft-odds*] *that* **by** *simp*
done
have ξ'*-residues*: *Fnr.to-residue-ring* (ξ' ! *j*) = *z' j mod Fnr.n* **if** *j* < *2* ^ (*oe-n* − *1*) **for** *j*
proof −
from *that* **have** *Fnr.to-residue-ring* (ξ' ! *j*) = *Fnr.to-residue-ring* (*Fnr.add-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 ξ'*-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*)) $\oplus_{Fn}$
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*) $\otimes_{Fn}$ (*inv*$_{Fn}$ *2*) [^]$_{Fn}$ (*oe-n* + *prim-root-exponent* ∗ *j* − *1*) $\oplus_{Fn}$
Fnr.to-residue-ring (*replicate* (*oe-n* + *2* ^ *n*) *False* @ [*True*])
apply (*intro arg-cong2*[**where** *f* = ($\oplus_{Fn}$)])
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* [^]$_{Fn}$ (*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-ifft*)
also have ... = *Fnr.NTT* ((*inv*$_{Fn}$ *2*) [^]$_{Fn}$ (*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*
(*inv*$_{Fn}$ (*2* [^]$_{Fn}$ (*prim-root-exponent* ∗ *2*)))
(*map Fnr.to-residue-ring c-dft-odds*) ! *j*
by (*intro arg-cong*[**where** *f* = λ*a*. *Fnr.NTT a* - ! -] *Fnr.inv-nat-pow*[*symmetric*] *Fnr.two-is-unit*)
also have ... = *Fnr.NTT* (*inv*$_{Fn}$ (μ [^]$_{Fn}$ (*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* (λ*j*. *inv*$_{Fn}$ *j*)] *more*: *refl*)
unfolding *μ-def*
by (*intro Fnr.nat-pow-pow*[*symmetric*] *Fnr.two-in-carrier*)
also have ... ⊗$_{Fn}$ (*inv*$_{Fn}$ *2*) [^]$_{Fn}$ (*oe-n* + *prim-root-exponent* ∗ *j* − *1*) =
(*inv*$_{Fn}$ *2*) [^]$_{Fn}$ (*oe-n* + *prim-root-exponent* ∗ *j* − *1*) ⊗$_{Fn}$ ...
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 μ-Units-Fn*)
subgoal using ‹*j* < *2* ^ (*oe-n* − *1*)› **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* (ξ′ ! *j*) =
(*c*′*-mod* ! *j* ⊖$_{Fn}$ *c*′*-mod* ! (*2* ^ (*oe-n* − *1*) + *j*)) ⊕$_{Fn}$
2 [^]$_{Fn}$ (*oe-n* + *2* ^ *n*)
unfolding *aux3*[*OF* ‹*j* < *2* ^ (*oe-n* − *1*)›]
using *Fnr.inv-nat-pow*[*OF μ-Units-Fn*] **by** *presburger*
also have ... = ((*c*′*-mod* ! *j* ⊖$_{Fn}$ *c*′*-mod* ! (*2* ^ (*oe-n* − *1*) + *j*)) +
2 [^]$_{Fn}$ (*oe-n* + *2* ^ *n*)) *mod Fnr.n*
by (*intro Fnr.res-add-eq*)
also have ... = ((*c*′*-mod* ! *j* − (*c*′*-mod* ! (*2* ^ (*oe-n* − *1*) + *j*))) *mod Fnr.n* +
2 [^]$_{Fn}$ (*oe-n* + *2* ^ *n*)) *mod Fnr.n*
by (*intro-cong* [*cong-tag-2* (*mod*), *cong-tag-2* (+)] *more*: *refl Fnr.residues-minus-eq*)
also have ... = (((*c*′ ! *j*) *mod Fnr.n* − (*c*′ ! (*2* ^ (*oe-n* − *1*) + *j*)) *mod Fnr.n*) *mod Fnr.n* +
2 [^]$_{Fn}$ (*oe-n* + *2* ^ *n*)) *mod Fnr.n*
apply (*intro-cong* [*cong-tag-2* (*mod*), *cong-tag-2* (+), *cong-tag-2* (−)] *more*: *refl*)
unfolding *c*′*-mod-def* **using** ‹*j* < *2* ^ (*oe-n* − *1*)› *index-intros*[*of j*] ‹*length c*′ = *2* ^ *oe-n*›
by *simp-all*
also have ... = (*c*′ ! *j* − *c*′ ! (*2* ^ (*oe-n* − *1*) + *j*) +
2 [^]$_{Fn}$ (*oe-n* + *2* ^ *n*)) *mod Fnr.n*
by (*simp only*: *mod-diff-eq mod-add-left-eq*)

also have ... = $(c'$! $j - c'$! $(2$ ^ $(oe\text{-}n - 1) + j) +$
2 ^ $(oe\text{-}n + 2$ ^ $n))$ *mod Fnr.n*
by (*simp only*: *Fnr.pow-nat-eq mod-add-right-eq*)
also have ... = z' *j mod Fnr.n*
unfolding *z'-def* **using** ‹$j < 2$ ^ $(oe\text{-}n - 1)$› **by** *presburger*
finally show *Fnr.to-residue-ring* (ξ' ! j) = z' *j mod Fnr.n* **.**
qed

have ξ'*-carrier*: ξ' ! $i \in$ *Fnr.fermat-non-unique-carrier* **if** $i < 2$ ^ $(oe\text{-}n - 1)$ **for** *i*
proof −
from *that* **have** ξ' ! 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-lsbf* (*replicate* (*oe-n* + *2* ^ *n*) *False* @ [*True*])) **unfolding** ξ'*-def*
apply (*intro nth-map2*)
using *length-c-diffs* **by** *simp-all*
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-lsbf-correct*(*1*))
done
finally show ξ' ! $i \in$ *Fnr.fermat-non-unique-carrier* **.**
qed
have ξ*-*ξ': *Nat-LSBF.to-nat* (ξ ! i) = *Nat-LSBF.to-nat* (ξ' ! i) *mod Fnr.n*
if $i < 2$ ^ $(oe\text{-}n - 1)$ **for** *i*
unfolding ξ*-def* **using** *Fnr.reduce-correct*[*OF* ξ'*-carrier*[*OF that*]]
using *that length-*ξ' **by** *simp*
have *z'-bounds*: z' $j \geq 0$ z' $j < 2$ ^ $(oe\text{-}n + 1) * 2$ ^ 2 ^ n **if** $j < 2$ ^ $(oe\text{-}n - 1)$ **for** *j*
proof −
have z' $j = c'$! $j - c'$! $(2$ ^ $(oe\text{-}n - 1) + j) + 2$ ^ $(oe\text{-}n + 2$ ^ $n)$ (**is** - = *?z*)
unfolding *z'-def* **using** *that* **by** *simp*
have c' ! $j \geq 0$ c' ! $j < 2$ ^ $(oe\text{-}n + 2$ ^ $n)$
using *c'-upper-bound*[*of j*] *c'-lower-bound*[*of j*] *index-intros*[*of j*] ‹$j < 2$ ^ $(oe\text{-}n - 1)$›
by *simp-all*
moreover have c' ! $(2$ ^ $(oe\text{-}n - 1) + j) \geq 0$ c' ! $(2$ ^ $(oe\text{-}n - 1) + j) < 2$ ^ $(oe\text{-}n + 2$ ^ $n)$
using *c'-upper-bound c'-lower-bound index-intros*[*of j*] ‹$j < 2$ ^ $(oe\text{-}n - 1)$› **by** *simp-all*
ultimately have *?z* ≥ *0* *?z* < 2 ^ $(oe\text{-}n + 2$ ^ $n) + 2$ ^ $(oe\text{-}n + 2$ ^ $n)$
by *linarith+*
then have *?z* < 2 ^ $(oe\text{-}n + 1 + 2$ ^ $n)$
by *simp*

also have ... = 2 ^ $(oe\text{-}n + 1) * 2$ ^ 2 ^ n **by** (*simp add*: *power-add*)
finally show $z'\ j \geq 0$ $z'\ j < 2$ ^ $(oe\text{-}n + 1) * 2$ ^ 2 ^ n **using** ‹$z'\ j = ?z$› ‹$?z \geq 0$› **by** *simp-all*
qed
have *z-z'*: *Fmr.to-residue-ring* $(z\ !\ j) = z'\ j$ **if** $j < 2$ ^ $(oe\text{-}n - 1)$ **for** j
proof −
from *that* **have** $z\ !\ j$ = *solve-special-residue-problem* n $(\xi\ !\ j)$ $(\eta\ !\ j)$
unfolding *z-def* **using** *length-ξ length-η* **by** *simp*
then have *solves-special-residue-problem* (*Nat-LSBF.to-nat* $(z\ !\ j)$) n (*Nat-LSBF.to-nat* $(\xi\ !\ j)$) (*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* $(\xi\ !\ j)$) (*Nat-LSBF.to-nat* $(\eta\ !\ j)$)
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* $(\eta\ !\ j)$) *mod* 2 ^ $(n + 2)$
using *η-residues*[*OF that*] **unfolding** *Znr.to-residue-ring-def* **by** *simp*
also have ... = *int* (*Nat-LSBF.to-nat* $(\eta\ !\ j)$ *mod* 2 ^ $(n + 2)$)
by (*simp add*: *zmod-int*)
also have ... = *int* (*Nat-LSBF.to-nat* $(\eta\ !\ j)$)
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 $z'\ j \in$ *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* $= (\bigoplus_{Fm} i \leftarrow [0..<2\ \hat{}\ oe\text{-}n].\ (z'\ i) \otimes_{Fm} 2\ [\hat{}]_{Fm}\ (i * 2\ \hat{}\ (n - 1)))$
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* $= (\sum i \leftarrow [0..<length\ (z\text{-}filled\ @\ z\text{-}consts)].\ Nat\text{-}LSBF.to\text{-}nat\ ((z\text{-}filled\ @\ z\text{-}consts)\ !\ i) * 2\ \hat{}\ (i * 2\ \hat{}\ (n - 1)))$
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* $(2\ \hat{}\ (n - 1))$ *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* $(oe\text{-}n + 2\ \hat{}\ n)$ *False* @ [*True*]
using *prems* **unfolding** *z-consts-def* **by** *simp*
then have *length zi* $\geq 2\ \hat{}\ n$ **by** *simp*
moreover have $2\ \hat{}\ n \geq (2{::}nat)\ \hat{}\ (n - 1)$ **by** *simp*
ultimately show *?thesis* **by** *linarith*
qed
done
finally have *Fmr.to-residue-ring result* $= (\bigoplus_{Fm} i \leftarrow [0..<length\ (z\text{-}filled\ @\ z\text{-}consts)].\ int\ (Nat\text{-}LSBF.to\text{-}nat\ ((z\text{-}filled\ @\ z\text{-}consts)\ !\ i) * 2\ \hat{}\ (i * 2\ \hat{}\ (n - 1)))\ mod\ Fmr.n)$
unfolding *Fmr.monoid-sum-list-eq-sum-list' sum-list-int* **.**
also have ... $= (\bigoplus_{Fm} i \leftarrow [0..<2\ \hat{}\ oe\text{-}n].\ int\ (Nat\text{-}LSBF.to\text{-}nat\ ((z\text{-}filled\ @\ z\text{-}consts)\ !\ i) * 2\ \hat{}\ (i * 2\ \hat{}\ (n - 1)))\ mod\ Fmr.n)$
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 ... $= (\bigoplus_{Fm} i \leftarrow [0..<2\ \hat{}\ oe\text{-}n].\ (z'\ i) \otimes_{Fm} 2\ [\hat{}]_{Fm}\ (i * 2\ \hat{}\ (n -$

```
1)))
        apply (intro Fmr.monoid-sum-list-cong)
        subgoal premises prems for i
        proof (cases i < 2 ^ (oe-n − 1))
          case True
          then have int (Nat-LSBF.to-nat ((z-filled @ z-consts) ! i) ∗ 2 ^ (i ∗ 2 ^
(n − 1))) mod int Fmr.n
             = int (Nat-LSBF.to-nat (z-filled ! i) ∗ 2 ^ (i ∗ 2 ^ (n − 1))) mod int
Fmr.n
            using length-z-filled nth-append by metis
         also have ... = int (Nat-LSBF.to-nat (z ! i) ∗ 2 ^ (i ∗ 2 ^ (n − 1))) mod
int Fmr.n
            unfolding z-filled-def using length-z True by simp
          also have ... = (int (Nat-LSBF.to-nat (z ! i)) mod int Fmr.n ∗ 2 ^ (i ∗
2 ^ (n − 1))) mod int Fmr.n
            by (simp add: mod-mult-left-eq)
          also have ... = (z' i ∗ 2 ^ (i ∗ 2 ^ (n − 1))) 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)) mod int Fmr.n)) mod int
Fmr.n
            by (simp add: mod-mult-right-eq)
          also have ... = z' i ⊗Fm (2 ^ (i ∗ 2 ^ (n − 1)) mod int Fmr.n)
            by (rule Fmr.res-mult-eq[symmetric])
          also have 2 ^ (i ∗ 2 ^ (n − 1)) mod int Fmr.n = 2 [^]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 int (Nat-LSBF.to-nat ((z-filled @ z-consts) ! i) ∗ 2 ^ (i ∗ 2 ^
(n − 1))) mod int Fmr.n =
              int (Nat-LSBF.to-nat (z-consts ! j) ∗ 2 ^ (i ∗ 2 ^ (n − 1))) mod int
Fmr.n
            using length-z-filled nth-append-length-plus by metis
          also have ... = int (Nat-LSBF.to-nat (replicate (oe-n + 2 ^ n) False @
[True]) ∗ 2 ^ (i ∗ 2 ^ (n − 1))) mod int Fmr.n
            unfolding z-consts-def using ‹j < 2 ^ (oe-n − 1)› by simp
          also have ... = int (2 ^ (oe-n + 2 ^ n)) ∗ 2 ^ (i ∗ 2 ^ (n − 1)) mod int
Fmr.n
            by simp
          also have ... = z' i ∗ 2 ^ (i ∗ 2 ^ (n − 1)) mod int Fmr.n
            unfolding z'-def using False by simp
          also have ... = z' i ⊗Fm (2 ^ (i ∗ 2 ^ (n − 1)) mod int Fmr.n)
            by (simp add: mod-mult-right-eq Fmr.res-mult-eq)
```

also have $2\ \hat{}\ (i * 2\ \hat{}\ (n - 1))$ *mod int Fmr.n* $=$ $2\ [\hat{}]_{Fm}\ (i * 2\ \hat{}\ (n - 1))$
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* ∈ *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 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 $m = max\ (bitsize\ (length\ a))\ (bitsize\ (length\ b)) + 1$.

definition *schoenhage-strassen-mul* **where**
schoenhage-strassen-mul a b = (*let m* = *max* (*bitsize* (*length a*)) (*bitsize* (*length b*)) + *1 in*
int-lsbf-fermat.reduce m (*schoenhage-strassen m* (*fill* (*2* ^ (*m* + *1*)) *a*) (*fill* (*2* ^ (*m* + *1*)) *b*))
)

locale *schoenhage-strassen-mul-context* =
fixes *a b* :: *nat-lsbf*
begin

definition *bits-a* **where** *bits-a* = *bitsize* (*length a*)
definition *bits-b* **where** *bits-b* = *bitsize* (*length b*)
definition *m′* **where** *m′* = *max bits-a bits-b*
definition *m* **where** *m* = *m′* + *1*
definition *car-len* **where** *car-len* = (*2*::*nat*) ^ (*m* + *1*)
definition *fill-a* **where** *fill-a* = *fill car-len a*
definition *fill-b* **where** *fill-b* = *fill car-len b*
definition *fm-result* **where** *fm-result* = *schoenhage-strassen m 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*: *length a* $<$ *2* ^ $(m - 1)$
    and *length-b*: *length b* $<$ *2* ^ $(m - 1)$
  using *m-def bitsize-length defs* **by** *fastforce*+

lemma
  shows *length-a′*: *length a* $\leq$ *2* ^ $(m + 1)$
    and *length-b′*: *length b* $\leq$ *2* ^ $(m + 1)$
  using *length-a length-b* **by** (*simp-all add*: *m-def nat-le-real-less nat-less-real-le*)

lemma *length-fill-a*: *length fill-a* = *2* ^ $(m + 1)$
  unfolding *fill-a-def car-len-def*
  by (*intro length-fill length-a′*)

lemma *length-fill-b*: *length fill-b* = *2* ^ $(m + 1)$
  unfolding *fill-b-def car-len-def*
  by (*intro length-fill length-b′*)

sublocale *fm*: *int-lsbf-fermat m* **.**

definition *Fm* **where** *Fm* = *residue-ring* (*int-lsbf-fermat.n m*)
sublocale *Fmr*: *residues fm.n Fm*
  rewrites *fm-Fm*: *fm.Fn* ≡ *Fm*
  unfolding *Fm-def fm.Fn-def* **by** (*rule fm.residues-axioms reflexive*)+

lemma *fill-a-carrier*[*simp*, *intro*]: *fill-a* ∈ *fm.fermat-non-unique-carrier*
  by (*intro fm.fermat-non-unique-carrierI length-fill-a*)
lemma *fill-b-carrier*[*simp*, *intro*]: *fill-b* ∈ *fm.fermat-non-unique-carrier*
  by (*intro fm.fermat-non-unique-carrierI length-fill-b*)

lemma *fm-result-carrier*[*simp*, *intro*]: *fm-result* ∈ *fm.fermat-non-unique-carrier*
  unfolding *fm-result-def*
  by (*intro conjunct2*[*OF schoenhage-strassen-correct′*] *fill-a-carrier fill-b-carrier*)

lemma *ssc′*: *fm.to-residue-ring fm-result* = *fm.to-residue-ring fill-a* $\otimes_{Fm}$ *fm.to-residue-ring fill-b*
and *fm-result* ∈ *int-lsbf-fermat.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*: *Nat-LSBF.to-nat* (*schoenhage-strassen-mul a b*) = *Nat-LSBF.to-nat a* ∗ *Nat-LSBF.to-nat b*
proof −
  interpret *schoenhage-strassen-mul-context a b* **.**

  have *int* (*Nat-LSBF.to-nat a*) ∗ *int* (*Nat-LSBF.to-nat b*) $<$ *int-lsbf-fermat.n m*

proof −
have *Nat-LSBF.to-nat a* $<$ *2* ^ *length a Nat-LSBF.to-nat b* $<$ *2* ^ *length b* **by** (*intro to-nat-length-bound*)+
moreover have (*2*::*nat*) ^ *length a* $<$ *2* ^ *2* ^ (*m* − *1*) (*2*::*nat*) ^ *length b* $<$ *2* ^ *2* ^ (*m* − *1*)
using *length-a length-b* **by** *simp-all*
ultimately have *Nat-LSBF.to-nat a* ∗ *Nat-LSBF.to-nat b* $<$ *2* ^ *2* ^ (*m* − *1*) ∗ *2* ^ *2* ^ (*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-lsbf-fermat.to-residue-ring m a* $\otimes_{Fm}$ *int-lsbf-fermat.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-lsbf-fermat.to-residue-ring.simps mod-mult-eq*)
also have *int-lsbf-fermat.to-residue-ring m a* = *int-lsbf-fermat.to-residue-ring m fill-a*
unfolding *int-lsbf-fermat.to-residue-ring.simps defs* **by** *simp*
also have *int-lsbf-fermat.to-residue-ring m b* = *int-lsbf-fermat.to-residue-ring m fill-b*
unfolding *int-lsbf-fermat.to-residue-ring.simps defs* **by** *simp*
finally have *c*: *int-lsbf-fermat.to-residue-ring m fill-a* $\otimes_{Fm}$ *int-lsbf-fermat.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-lsbf-fermat.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-lsbf-fermat.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-lsbf-fermat.n m*))
by *simp*
also have ... = *nat* (*int-lsbf-fermat.to-residue-ring m* (*schoenhage-strassen m fill-a fill-b*))
unfolding *int-lsbf-fermat.to-residue-ring.simps*
by (*intro arg-cong*[**where** *f* = *nat*] *zmod-int*)
also have ... = *nat* (
int-lsbf-fermat.to-residue-ring m fill-a $\otimes_{Fm}$
int-lsbf-fermat.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**
$solve\text{-}special\text{-}residue\text{-}problem\text{-}tm\ n\ \xi\ \eta = 1\ do\ \{$
  $n2 \leftarrow n +_t 2;$
  $\xi mod \leftarrow take\text{-}tm\ n2\ \xi;$
  $\delta \leftarrow int\text{-}lsbf\text{-}mod.subtract\text{-}mod\text{-}tm\ n2\ \eta\ \xi mod;$
  $pown \leftarrow 2\ \widehat{}_t\ n;$
  $\delta\text{-}shifted \leftarrow \delta >>_{nt} pown;$
  $\delta 1 \leftarrow \delta\text{-}shifted +_{nt} \delta;$
  $\xi +_{nt} \delta 1$
$\}$

lemma *val-solve-special-residue-problem-tm*[*simp*, *val-simp*]:
  $val\ (solve\text{-}special\text{-}residue\text{-}problem\text{-}tm\ n\ \xi\ \eta) = solve\text{-}special\text{-}residue\text{-}problem\ n\ \xi\ \eta$
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*:
  $time\ (solve\text{-}special\text{-}residue\text{-}problem\text{-}tm\ n\ \xi\ \eta) \le 245 + 74 * 2\ \widehat{}\ n + 55 * length\ \eta + 2 * length\ \xi$
proof −
  define *n2* **where** $n2 = n + 2$
  define ξmod **where** $\xi mod = take\ n2\ \xi$
  define δ **where** $\delta = int\text{-}lsbf\text{-}mod.subtract\text{-}mod\ n2\ \eta\ \xi mod$
  define *pown* **where** $pown = (2::nat)\ \widehat{}\ n$
  define δ*-shifted* **where** $\delta\text{-}shifted = \delta >>_n pown$
  define $\delta 1$ **where** $\delta 1 = add\text{-}nat\ \delta\text{-}shifted\ \delta$
  note *defs* = *n2-def* ξ*mod-def* δ*-def pown-def* δ*-shifted-def* δ*1-def*

  interpret *mr*: *int-lsbf-mod n2* **apply** (*intro int-lsbf-mod.intro*) **unfolding** *n2-def*

by *simp*

have *length-ξmod-le*: *length ξmod* $\leq$ *n2* **unfolding** *ξmod-def* **by** *simp*
have *length-δ-le*: *length δ* $\leq$ *max n2* (*length η*)
unfolding *δ-def mr.subtract-mod-def if-distrib*[**where** *f* = *length*] *mr.length-reduce*
apply (*estimation estimate*: *conjunct2*[*OF subtract-nat-aux*])
using *length-ξmod-le* **by** *auto*
have *length-δ1add-le*: *max* (*length δ-shifted*) (*length δ*) $\leq$ *2* ^ *n* + (*n* + *2*) + *length η*
unfolding *δ-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 ξmod*) +
time (*2* $\hat{}_t$ *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 ... $\leq$ *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 ... = $245 + 12 * 2 \mathbin{\hat{}} n + 62 * n + 55 * \mathit{length}\ \eta + 2 * \mathit{length}\ \xi$
unfolding *n2-def* **by** *simp*
also have ... $\leq 245 + 74 * 2 \mathbin{\hat{}} n + 55 * \mathit{length}\ \eta + 2 * \mathit{length}\ \xi$
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* $\gg=$ *concat-tm*
| *combine-z-aux-tm l acc* [*z*] =*1 combine-z-aux-tm l* (*z* # *acc*) []
| *combine-z-aux-tm l acc* (*z1* # *z2* # *zs*) =*1 do* {
(*z1h*, *z1t*) $\leftarrow$ *split-at-tm l z1*;
$r \leftarrow z1t +_{nt} z2$;
combine-z-aux-tm l (*z1h* # *acc*) (*r* # *zs*)
}

lemma *val-combine-z-aux-tm*[*simp*, *val-simp*]: *val* (*combine-z-aux-tm l acc zs*) = *combine-z-aux l acc zs*
by (*induction l acc zs rule*: *combine-z-aux.induct*; *simp*)

lemma *time-combine-z-aux-tm-le*:
assumes $\bigwedge z.\ z \in \mathit{set}\ zs \Longrightarrow \mathit{length}\ z \leq lz$
assumes $\mathit{length}\ z \leq lz + 1$
assumes $l > 0$
shows *time* (*combine-z-aux-tm l acc* (*z* # *zs*)) $\leq (2 * l + 2 * lz + 7) * \mathit{length}$ $zs + 3 * (\mathit{length}\ acc + \mathit{length}\ zs) + \mathit{length}\ (\mathit{concat}\ acc) + \mathit{length}\ zs * l + lz + 9$
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*: *length* (*drop l z*) $\leq$ *lz* **by** *simp*
have *lena*: *length* (*add-nat* (*drop l z*) *z1*) $\leq lz + 1$
apply (*estimation estimate*: *length-add-nat-upper*)
using *len-drop-z Cons.prems* **by** *simp*
have *time* (*combine-z-aux-tm l acc* (*z* # *z1* # *zs*)) =
time (*split-at-tm l z*) +
time ($\mathit{drop}\ l\ z +_{nt} z1$) +
time (*combine-z-aux-tm l* (*take l z* # *acc*) (($\mathit{drop}\ l\ z +_{n} z1$) # *zs*)) + *1*
by *simp*
also have ... $\leq$
$(2 * l + 3) +$
$(2 * lz + 3) +$
$((2 * l + 2 * lz + 7) * \mathit{length}\ zs + 3 * (\mathit{length}\ (\mathit{take}\ l\ z \mathbin{\#} acc) + \mathit{length}\ zs)$
+
$\mathit{length}\ (\mathit{concat}\ (\mathit{take}\ l\ z \mathbin{\#} acc)) + \mathit{length}\ zs * l + lz + 9) + 1$
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.prems by simp
    subgoal
      apply (intro Cons.IH)
      subgoal using Cons.prems by simp
      subgoal using lena .
      subgoal using Cons.prems(3) .
      done
    done
  also have ... = (2 ∗ l + 2 ∗ lz + 7) ∗ length (z1 # zs) + 3 ∗ (length acc + 1
+ length zs) +
      length (concat acc) + length (take l z) + length zs ∗ l + lz + 9
    by simp
  also have ... ≤ (2 ∗ l + 2 ∗ lz + 7) ∗ length (z1 # zs) + 3 ∗ (length acc + 1
+ length zs) +
      length (concat acc) + l + length zs ∗ l + lz + 9
    apply (intro add-mono order.refl) by simp
  also have ... = (2 ∗ l + 2 ∗ lz + 7) ∗ length (z1 # zs) + 3 ∗ (length acc +
length (z1 # zs)) +
      length (concat acc) + 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 ⋀z. z ∈ set zs ⟹ length z ≤ lz
  assumes l > 0
  shows time (combine-z-tm l zs) ≤ 10 + (3 ∗ l + 2 ∗ lz + 10) ∗ length zs
proof (cases zs)
  case Nil
  then have time (combine-z-tm l zs) = 5
    unfolding combine-z-tm-def by simp
  then show ?thesis by simp
next
  case (Cons z zs')
  then have time (combine-z-tm l zs) = time (combine-z-aux-tm l [] (z # zs')) +
1
    unfolding combine-z-tm-def by simp
  also have ... ≤ (2 ∗ l + 2 ∗ lz + 7) ∗ length zs' + 3 ∗ (length ([] :: nat-lsbf list)
+ length zs') + length (concat ([] :: nat-lsbf list)) +
     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*) ∗ *length zs′* + *lz*
by (*simp add*: *add-mult-distrib*)
also have ... ≤ *10* + (*3* ∗ *l* + *2* ∗ *lz* + *10*) ∗ *length zs*
unfolding *Cons* **by** *simp*
finally show *?thesis* **.**
qed

lemma *schoenhage-strassen-tm-termination-aux*: ¬ *m* < *3* ⟹ *Suc* (*m div 2*) < *m*
by *linarith*

function *schoenhage-strassen-tm* :: *nat* ⇒ *nat-lsbf* ⇒ *nat-lsbf* ⇒ *nat-lsbf tm* **where**
schoenhage-strassen-tm m a b =*1 do* {
m-le-3 ← *m* $<_t$ *3*;
if m-le-3 then do {
ab ← *a* $*_{nt}$ *b*;
int-lsbf-fermat.from-nat-lsbf-tm m ab
} *else do* {
odd-m ← *odd-tm m*;
n ← (*if odd-m then do* {
m1 ← *m* $+_t$ *1*;
m1 div_t *2*
} *else do* {
m2 ← *m* $+_t$ *2*;
m2 div_t *2*
});
n-plus-1 ← *n* $+_t$ *1*;
n-minus-1 ← *n* $-_t$ *1*;
n-plus-2 ← *n* $+_t$ *2*;
oe-n ← (*if odd-m then return n-plus-1 else return n*);
segment-lens ← *2* $\widehat{}_t$ *n-minus-1*;
a′ ← *subdivide-tm segment-lens a*;
b′ ← *subdivide-tm segment-lens b*;
α ← *map-tm* (*int-lsbf-mod.reduce-tm n-plus-2*) *a′*;
three-n ← *3* $*_t$ *n*;
pad-length ← *three-n* $+_t$ *5*;
α-padded ← *map-tm* (*fill-tm pad-length*) α;
u ← *concat-tm α-padded*;
β ← *map-tm* (*int-lsbf-mod.reduce-tm n-plus-2*) *b′*;
β-padded ← *map-tm* (*fill-tm pad-length*) β;
v ← *concat-tm β-padded*;
oe-n-plus-1 ← *oe-n* $+_t$ *1*;
two-pow-oe-n-plus-1 ← *2* $\widehat{}_t$ *oe-n-plus-1*;
uv-length ← *pad-length* $*_t$ *two-pow-oe-n-plus-1*;
uv-unpadded ← *karatsuba-mul-nat-tm u v*;

```
uv ← ensure-length-tm uv-length uv-unpadded;
oe-n-minus-1 ← oe-n −_t 1;
two-pow-oe-n-minus-1 ← 2 ^_t oe-n-minus-1;
γs ← subdivide-tm pad-length uv;
γ ← subdivide-tm two-pow-oe-n-minus-1 γs;
γ0 ← nth-tm γ 0;
γ1 ← nth-tm γ 1;
γ2 ← nth-tm γ 2;
γ3 ← nth-tm γ 3;
η ← map4-tm
   (λx y z w. do {
      xmod ← take-tm n-plus-2 x;
      ymod ← take-tm n-plus-2 y;
      zmod ← take-tm n-plus-2 z;
      wmod ← take-tm n-plus-2 w;
      xy ← int-lsbf-mod.subtract-mod-tm n-plus-2 xmod ymod;
      zw ← int-lsbf-mod.subtract-mod-tm n-plus-2 zmod wmod;
      int-lsbf-mod.add-mod-tm n-plus-2 xy zw
   })
   γ0 γ1 γ2 γ3;
prim-root-exponent ← if odd-m then return 1 else return 2;
fn-carrier-len ← 2 ^_t n-plus-1;
a′-carrier ← map-tm (fill-tm fn-carrier-len) a′;
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
apply (*relation Wellfounded.measure* (λ(*n*, *a*, *b*). *n*))
subgoal by *blast*
subgoal for *m* **by** (*cases odd m*; *simp*)
subgoal for *m* **by** (*cases odd m*; *simp*)
subgoal for *m* **by** (*cases odd m*; *simp*)
subgoal for *m* **by** (*cases odd m*; *simp*)
subgoal for *m* **by** (*cases odd m*; *simp*)
subgoal for *m* **by** (*cases odd m*; *simp*)
subgoal for *m* **by** (*cases odd m*; *simp*)
subgoal for *m* **by** (*cases odd m*; *simp*)
subgoal for *m* **by** (*cases odd m*; *simp*)
subgoal for *m* **by** (*cases odd m*; *simp*)
subgoal for *m* **by** (*cases odd m*; *simp*)
subgoal for *m* **by** (*cases odd m*; *simp*)
subgoal for *m* **by** (*cases odd m*; *simp*)
subgoal for *m* **by** (*cases odd m*; *simp*)
subgoal for *m* **by** (*cases odd m*; *simp*)
subgoal for *m* **by** (*cases odd m*; *simp*)
subgoal for *m* **by** (*cases odd m*; *simp*)
subgoal for *m* **by** (*cases odd m*; *simp*)
subgoal for *m* **by** (*cases odd m*; *simp*)
subgoal for *m* **by** (*cases odd m*; *simp*)
done

context *schoenhage-strassen-context* **begin**

abbreviation *γ0* **where** *γ0* ≡ *γ* ! *0*
abbreviation *γ1* **where** *γ1* ≡ *γ* ! *1*
abbreviation *γ2* **where** *γ2* ≡ *γ* ! *2*
abbreviation *γ3* **where** *γ3* ≡ *γ* ! *3*

definition *fn-carrier-len* **where** *fn-carrier-len* = (*2*::*nat*) ^ (*n* + *1*)
definition *segment-lens* **where** *segment-lens* = (*2*::*nat*) ^ (*n* − *1*)
definition *interval1* **where** *interval1* = [*0*..<*2* ^ (*oe-n* − *1*)]
definition *interval2* **where** *interval2* = [*2* ^ (*oe-n* − *1*)..<*2* ^ *oe-n*]
definition *oe-n-plus-two-pow-n-zeros* **where** *oe-n-plus-two-pow-n-zeros* = *replicate* (*oe-n* + *2* ^ *n*) *False*
definition *oe-n-plus-two-pow-n-one* **where** *oe-n-plus-two-pow-n-one* = *append oe-n-plus-two-pow-n-zeros* [*True*]

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

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

lemma *z-filled-def′*: *z-filled* = *map* (*fill segment-lens*) *z*
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 *scuv*(*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-lsbf-fermat.fft-tm n prim-root-exponent A.num-blocks-carrier*) =
int-lsbf-fermat.fft n prim-root-exponent A.num-blocks-carrier
by (*intro int-lsbf-fermat.val-fft-tm*[**where** *m* = *oe-n*] *A.length-num-blocks-carrier*)
lemma *val-fft2*[*simp*, *val-simp*]: *val* (*int-lsbf-fermat.fft-tm n prim-root-exponent B.num-blocks-carrier*) =
int-lsbf-fermat.fft n prim-root-exponent B.num-blocks-carrier
by (*intro int-lsbf-fermat.val-fft-tm*[**where** *m* = *oe-n*] *B.length-num-blocks-carrier*)

lemma *val-ifft*[*simp*, *val-simp*]: *val* (*int-lsbf-fermat.ifft-tm n* (*prim-root-exponent* ∗ *2*) *c-dft-odds*) =
int-lsbf-fermat.ifft n (*prim-root-exponent* ∗ *2*) *c-dft-odds*
apply (*intro int-lsbf-fermat.val-ifft-tm*[**where** *m* = *oe-n* − *1*])
apply (*simp add*: *c-dft-odds-def*)
done

end

lemma *val-schoenhage-strassen-tm*[*simp*, *val-simp*]:
  assumes $a \in$ *int-lsbf-fermat.fermat-non-unique-carrier m*
  assumes $b \in$ *int-lsbf-fermat.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-lsbf-fermat.val-from-nat-lsbf-tm* **by** *simp*
  next
    case *False*

    interpret *schoenhage-strassen-context m a b*
      apply *unfold-locales* **using** *False less.prems* **by** *simp-all*

    have *val-ih*: *map2* ($\lambda x\ y.$ *val* (*schoenhage-strassen-tm n x y*)) *A.num-dft-odds*
B.num-dft-odds =
      map2 ($\lambda x\ y.$ *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-ifft*
        defs[*symmetric*] *Let-def*
        val-subdivide-tm[*OF two-pow-pos*] *Fnr.val-ifft-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 ^ (oe-n − 1))

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

lemma time-schoenhage-strassen-tm-le:
  assumes a ∈ int-lsbf-fermat.fermat-non-unique-carrier m
  assumes b ∈ int-lsbf-fermat.fermat-non-unique-carrier m
  shows time (schoenhage-strassen-tm m a b) ≤ schoenhage-strassen-Fm-bound m
  using assms proof (induction m arbitrary: a b rule: less-induct)
  case (less m)
  consider m = 0 | m ≥ 1 ∧ m < 3 | ¬ m < 3 by linarith
  then show ?case
  proof cases
    case 1
    from less.prems int-lsbf-fermat.fermat-carrier-length
    have len-ab: length a = 2 length b = 2 unfolding 1 by simp-all
    then have len-mul-ab: length (grid-mul-nat a b) ≤ 4
      using length-grid-mul-nat[of a b] by simp
    from 1 have time (schoenhage-strassen-tm m a b) =
      time (m <_t 3) +
      time (a *_nt b) +
      time (int-lsbf-fermat.from-nat-lsbf-tm 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 * length a * max (length a) (length b) + 1) +
      int-lsbf-fermat.time-from-nat-lsbf-tm-bound m (length (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-lsbf-fermat.time-from-nat-lsbf-tm-le-bound order.refl)
      done
    also have ... ≤ 2 + 33 + 240 + 1
      apply (intro add-mono order.refl)
      subgoal unfolding 1 by simp
      subgoal unfolding len-ab by simp
        subgoal unfolding int-lsbf-fermat.time-from-nat-lsbf-tm-bound.simps[of 0
(length (grid-mul-nat a b))] 1
        using len-mul-ab by simp
```

done
also have ... = *276* **by** *simp*
finally show *?thesis* **unfolding** *schoenhage-strassen-Fm-bound.simps*[*of m*] **using** *1* **by** *simp*
next
case *2*
then have $(2::nat) \widehat{\ } (m + 1) \geq 4$
using *power-increasing*[*of 2 m + 1 2::nat*] **by** *simp*
from *2* **have** $(2::nat) \widehat{\ } (m + 1) \leq 8$
using *power-increasing*[*of m + 1 3 2::nat*] **by** *simp*
from *less.prems* **have** *len-ab*: $length\ a = 2 \widehat{\ } (m + 1)$ $length\ b = 2 \widehat{\ } (m + 1)$
using *int-lsbf-fermat.fermat-carrier-length* **by** *simp-all*
then have *len-ab-le*: $length\ a \leq 8$ $length\ b \leq 8$
using ‹$2 \widehat{\ } (m + 1) \leq 8$› **by** *linarith+*
have *len-mul-ab-le*: $length\ (grid\text{-}mul\text{-}nat\ a\ b) \leq 2 * 2 \widehat{\ } (m + 1)$
using *length-grid-mul-nat*[*of a b*] *len-ab* **by** *simp*
from *2* **have** $time\ (schoenhage\text{-}strassen\text{-}tm\ m\ a\ b) =$
$time\ (m <_t 3) +$
$time\ (a *_{nt} b) +$
$time\ (int\text{-}lsbf\text{-}fermat.from\text{-}nat\text{-}lsbf\text{-}tm\ m\ (grid\text{-}mul\text{-}nat\ a\ b)) + 1$
unfolding *schoenhage-strassen-tm.simps*[*of m a b*] *time-bind-tm val-less-nat-tm*
by (*simp del*: *One-nat-def*)
also have ... $\leq (2 * m + 2) +$
$(8 * length\ a * max\ (length\ a)\ (length\ b) + 1) +$
$(720 + 512 * 2 \widehat{\ } (m + 1)) + 1$
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-lsbf-fermat.time-from-nat-lsbf-tm-le*[*OF* ‹$4 \leq 2 \widehat{\ } (m + 1)$› *len-mul-ab-le*]
by *simp*
done
also have ... $\leq 6 + 513 + (720 + 512 * 8) + 1$
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 ‹$2 \widehat{\ } (m + 1) \leq 8$› **.**
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*
(λ*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
})
γ*0* γ*1* γ*2* γ*3*) (**is** *time-η* = *time* (*map4-tm ?η-fun - - - -*))
define *time-ξ′* **where** *time-ξ′* = *time* (*map2-tm* (λ*x 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*: $\bigwedge i.\ 0 \le i \Longrightarrow i < 2\ \hat{}\ 2\ \hat{}\ m + 1 \Longrightarrow i \in$ *carrier Fm*
by *simp*

have *length-uv-unpadded-le*: *length uv-unpadded* ≤ *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* ≤ *2* **unfolding** *prim-root-exponent-def* **by** *simp*
then have *prim-root-exponent-2-le*: *prim-root-exponent* ∗ *2* ≤ *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* ⊆ *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-ifft-carrier*[*of - oe-n* − *1*]])
  subgoal using *length-c-dft-odds* **.**
  subgoal using *c-dft-odds-carrier* **.**
  subgoal using *Fnr.length-ifft*[*OF length-c-dft-odds*] *that* **by** *simp*
  done
 have *ξ'-carrier*: *ξ'* ! *i* ∈ *Fnr.fermat-non-unique-carrier* **if** *i* < *2* ^ (*oe-n* − *1*) **for** *i*
 proof −
  from *that* **have** *ξ'* ! *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-lsbf* (*replicate* (*oe-n* + *2* ^ *n*) *False* @ [*True*]))
   unfolding *ξ'-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-lsbf-correct*(*1*))
   done
  finally show *ξ'* ! *i* ∈ *Fnr.fermat-non-unique-carrier* **.**

qed
have ξ'*-carrier'*: *set* $\xi' \subseteq$ *Fnr.fermat-non-unique-carrier*
apply (*intro set-subseteqI* ξ'*-carrier*) **unfolding** *length-*ξ' **.**
have *length-*ξ*-entries*: *length* $x \leq 2$ ^ $n + 2$ **if** $x \in$ *set* ξ **for** x
proof −
from *that* **obtain** x' **where** $x' \in$ *set* ξ' $x =$ *Fnr.reduce* x' **unfolding** ξ*-def*
by *auto*
from *that* **show** *?thesis* **unfolding** ‹$x =$ *Fnr.reduce* x'›
apply (*intro Fnr.reduce-correct'*(*2*))
using ‹$x' \in$ *set* ξ'› ξ'*-carrier'* **by** *auto*
qed
have *length-*η*-entries*: *length* (η ! i) = $n + 2$ **if** $i < 2$ ^(*oe-n* − *1*) **for** i
proof −
have η ! i = *Znr.add-mod* (*Znr.subtract-mod* (*take* (n + *2*) (γ*0* ! i)) (*take* (n + *2*) (γ*1* ! i)))
(*Znr.subtract-mod* (*take* (n + *2*) (γ*2* ! i)) (*take* (n + *2*) (γ*3* ! i)))
unfolding η*-def Let-def defs'*[*symmetric*]
apply (*intro nth-map4*)
unfolding *length-*γ*s defs'* **using** *length-*γ*-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 = *solve-special-residue-problem* n (ξ ! i) (η ! i)
unfolding *z-def* **apply** (*intro nth-map2*) **using** *that length-*ξ *length-*η **by** *simp-all*
also have *length* ... $\leq$ *max* (*length* (ξ ! i))
(*2* ^ n + *length* (*Znr.subtract-mod* (η ! i) (*take* (n + *2*) (ξ ! 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-*ξ*-entries nth-mem*[*of i* ξ] *length-*ξ *that* **by** *simp*
subgoal apply (*intro Znr.length-subtract-mod*)
subgoal using *length-*η*-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-filled* ! i) $\leq$ *2* ^ n + n + *4* **if** $i < 2$ ^ (*oe-n* − *1*) **for** i
proof −
have *z-filled* ! i = *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* $\leq$ *2* ^ *n* + *n* + *4* **if** *i* $\in$ *set z-complete*
for *i*
proof −
from *that* **consider** *i* $\in$ *set z-filled* | *i* $\in$ *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* $\leq$ *28* ∗ *Fmr.e*
proof −
have *length z-sum* $\leq$ ((*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 ... $\leq$ (*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 ... $\leq$ *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 ξ'*-alt*: *map2*
(λx *y*. *Fnr.add-fermat*
(*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*))
c-diffs interval1 = ξ'
unfolding ξ'*-def Let-def defs'*[*symmetric*] **by** (*rule refl*)

have *time-*η $\leq$ ((*112* $*$ (*n* + *2*) + *254*) + *1*) $*$ *min* (*min* (*min* (*length* $\gamma 0$) (*length* $\gamma 1$)) (*length* $\gamma 2$)) (*length* $\gamma 3$) + *1*
unfolding *time-*η*-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 ... $= (255 + 112 * (n + 2)) * 2 \,\hat{}\, (oe\text{-}n - 1) + 1$
unfolding *length-γs defs′* **using** *length-γ-i* **by** *simp*
also have ... $\leq (255 + 112 * (n + 2)) * 2 \,\hat{}\, n + 1 * 2 \,\hat{}\, n$
apply (*intro add-mono mult-le-mono order.refl*)
unfolding *oe-n-def* **by** *simp-all*
also have ... $= (256 + 112 * (n + 2)) * 2 \,\hat{}\, n$
by (*simp add*: *add-mult-distrib*)
also have ... $\leq (128 * (n + 2) + 112 * (n + 2)) * 2 \,\hat{}\, n$
apply (*intro add-mono mult-le-mono order.refl*)
by *simp*
finally have *time-η-le*: $time\text{-}\eta \leq 240 * (n + 2) * 2 \,\hat{}\, n$ **by** *simp*

have *oe-n-prim-root-le*: $oe\text{-}n + prim\text{-}root\text{-}exponent * y - 1 \leq fn\text{-}carrier\text{-}len$ **if** $y \in set\ interval1$ **for** *y*
proof −
have $oe\text{-}n + prim\text{-}root\text{-}exponent * y - 1 \leq n + prim\text{-}root\text{-}exponent * y$
using *oe-n-minus-1-le-n* **by** *simp*
also have ... $\leq n + prim\text{-}root\text{-}exponent * 2 \,\hat{}\, (oe\text{-}n - 1)$
using *that* **unfolding** *interval1-def defs′* **by** *simp*
also have ... $= n + 2 \,\hat{}\, n$

unfolding *oe-n-def prim-root-exponent-def*
by (*cases odd m*; *simp add*: *n-gt-0 power-Suc*[*symmetric*])
also have ... $\leq$ *2* ^ *n* + *2* ^ *n*
by *simp*
also have ... = *fn-carrier-len*
unfolding *defs'* **by** *simp*
finally show *?thesis* **.**
qed

have *time-ξ'* $\leq$ ((*475* + *378* ∗ *Fnr.e*) + *2*) ∗ *length c-diffs* + *3*
unfolding *time-ξ'-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*)) $\leq$
(*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* $\leq$ -)
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 ... = $477 + 2 * y + oe\text{-}n + 343 * Fnr.e + 33 * length\ x$
unfolding *fn-carrier-len-def* **by** *simp*
also have ... = $477 + 2 * y + oe\text{-}n + 376 * Fnr.e$
using *prems set-subseteqI*[*OF c-diffs-carrier*] *length-c-diffs* **by** *auto*
also have ... $\leq 477 + 2 * 2$ ^ $(oe\text{-}n - 1) + oe\text{-}n + 376 * Fnr.e$
using *prems* **unfolding** *interval1-def*
by *simp*
also have ... $\leq 477 + oe\text{-}n + 377 * Fnr.e$
unfolding *oe-n-def* **by** *simp*
also have ... $\leq 475 + 378 * Fnr.e$
using *oe-n-n-bound-1* **by** *simp*
finally show *?t* $\leq 475 + 378 * Fnr.e$ **unfolding** *defs*[*symmetric*] **.**
qed
done
also have ... $\leq (475 + 758 * 2$ ^ $n) * 2$ ^ $n + 3$
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*) $\leq 155 + 216 * 2$ ^ n **if** $i \in$ *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* $\leq ((155 + 216 * 2$ ^ $n) + 1) * length\ \xi' + 1$
unfolding *time-ξ-def*
by (*intro time-map-tm-bounded time-reduce-ξ′-nth*)
also have ... $\leq (156 + 216 * 2$ ^ $n) * 2$ ^ $n + 1$
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* $\leq ((245 + 74 * 2$ ^ $n + 55 * (n + 2) + 2 * (2$ ^ $n + 2)) + 2) * length\ \xi + 3$
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 ... $\leq$ *schoenhage-strassen-Fm-bound* $n * 2 \hat{}\ (oe\text{-}n - 1) + 2 * 2 \hat{}$
$n + 3$
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) $\leq$
... **.**

have *two-pow-pos*: $(2{::}nat)\ \hat{}\ x > 0$ **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 ($2\ \hat{}_t\ (n - 1)$) +
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 ($oe\text{-}n +_t 1$) +
time ($2\ \hat{}_t\ (oe\text{-}n + 1)$) +
time ($pad\text{-}length *_t 2\ \hat{}\ (oe\text{-}n + 1)$) +
time (*karatsuba-mul-nat-tm A.num-Zn-pad B.num-Zn-pad*) +
time (*ensure-length-tm uv-length uv-unpadded*) +
time ($oe\text{-}n -_t 1$) +
time ($2\ \hat{}_t\ (oe\text{-}n - 1)$) +
time (*subdivide-tm pad-length uv*) +
time (*subdivide-tm* ($2\ \hat{}\ (oe\text{-}n - 1)$) γs) +
time (*nth-tm* γ *0*) +
time (*nth-tm* γ *1*) +
time (*nth-tm* γ *2*) +
time (*nth-tm* γ *3*) +
time-η +
(*if odd m then 0 else 0*) +
time ($2\ \hat{}_t\ (n + 1)$) +

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* $\widehat{\ }_t$ *oe-n*) +
time (*upt-tm 0* (*2* ^ (*oe-n* − *1*))) +
time (*upt-tm* (*2* ^ (*oe-n* − *1*)) (*2* ^ *oe-n*)) +
time (*2* $\widehat{\ }_t$ *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-ifft val-ih Fnr.val-ifft-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) * length A.num-blocks + 1) +
(((3 * 2 ^ n + 5) + 1) * length B.num-blocks + 1) +
(2 ^ oe-n * (66 + 87 * Fnr.e) + oe-n * 2 ^ oe-n * (76 + 116 * Fnr.e) +
 8 * prim-root-exponent * 2 ^ (2 * oe-n)) +
(2 ^ oe-n * (66 + 87 * Fnr.e) + oe-n * 2 ^ oe-n * (76 + 116 * Fnr.e) +
 8 * prim-root-exponent * 2 ^ (2 * oe-n)) +
(2 * 2 ^ n + 1) +
(2 * 2 ^ n + 1) +
(schoenhage-strassen-Fm-bound n * 2 ^ (oe-n − 1) + 2 * 2 ^ n + 3) +
9 +
(2 ^ (oe-n − 1) * (66 + 87 * Fnr.e) +
 (oe-n − 1) * 2 ^ (oe-n − 1) * (76 + 116 * Fnr.e) +
 8 * (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 * segment-lens + 2 * (2 ^ n + n + 4) + 10) * length z-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-γs'
  apply (estimation estimate: oe-n-minus-1-le-n)
  by simp-all
subgoal using time-nth-tm[of 0 γ] scγ(1) by simp
subgoal using time-nth-tm[of 1 γ] scγ(1) by simp
subgoal using time-nth-tm[of 2 γ] scγ(1) by simp
subgoal using time-nth-tm[of 3 γ] scγ(1) by simp
subgoal
  apply (estimation estimate: time-η-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-ifft-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 ... ≤ *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-bound ((*3* ∗ *n* + *5*) ∗ *2* ^ *oe-n*) +
4 ∗ *karatsuba-lower-bound* +
schoenhage-strassen-Fm-bound n ∗ *2* ^ (*oe-n* − *1*)
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 ... ≤ *600* ∗ (*n* ∗ *2* ^ (*2* ∗ *n*)) + *2197* ∗ (*n* ∗ *2* ^ (*2* ∗ *n*)) + *2571* ∗ (*n* ∗ *2* ^ (*2* ∗ *n*)) +
2696 ∗ (*n* ∗ *2* ^ (*2* ∗ *n*)) +
8410 +
23508 ∗ *2* ^ *m* +
29 ∗ (*n* ∗ *2* ^ (*2* ∗ *n*)) +
17 ∗ *2* ^ *m* +
time-karatsuba-mul-nat-bound ((*3* ∗ *n* + *5*) ∗ *2* ^ *oe-n*) +
4 ∗ *karatsuba-lower-bound* +
schoenhage-strassen-Fm-bound n ∗ *2* ^ (*oe-n* − *1*)
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*] ‹*2* ^ (*2* ∗ *n*) ≤ *n* ∗ *2* ^ (*2* ∗ *n*)› **by** *linarith*
subgoal by *simp*
subgoal by *simp*
done
also have ... = *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*)
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*. (∀ *x*. *x* > *0* ⟶ *time-karatsuba-mul-nat-bound x* ≤ *c* ∗ *nat* (*floor* (*real x powr log 2 3*))))

lemma *real-divide-mult-eq*:
assumes (*c* :: *real*) ≠ *0*
shows *a* / *c* ∗ *c* = *a*
using *assms* **by** *simp*

lemma *powr-unbounded*:
assumes (*c* :: *real*) > *0*
shows *eventually* (λ*x*. *d* ≤ *x powr c*) *at-top*
proof (*cases d* > *0*)
case *True*
define *N* **where** *N* = *d powr* (*1* / *c*)
have *d* ≤ *x powr c* **if** *x* ≥ *N* **for** *x*
proof −
have *d* = *d powr 1* **apply** (*intro powr-one*[*symmetric*]) **using** *True* **by** *simp*
also have ... = (*d powr* (*1* / *c*)) *powr c*
unfolding *powr-powr*
apply (*intro arg-cong2*[**where** *f* = (*powr*)] *refl real-divide-mult-eq*[*symmetric*])
using *assms* **by** *simp*
also have ... = *N powr c* **unfolding** *N-def* **by** (*rule refl*)
also have ... ≤ *x 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* ≤ *karatsuba-const* ∗ *nat* (*floor* (*real x powr log 2 3*))
proof −

have $\exists\, c.\ (\forall\, x.\ x \geq 1 \longrightarrow$ *time-karatsuba-mul-nat-bound* $x \leq c * nat$ (*floor* (*real* x *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* ≥ 1 **by** *simp*
    then have *floor* (*real x powr log 2 3*) ≥ 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* $\lfloor$*real x powr log 2 3*$\rfloor$
  unfolding *karatsuba-const-def*
  apply (*intro someI-ex*[*of* $\lambda c.\ \forall\, x{>}0.$ *time-karatsuba-mul-nat-bound* $x \leq c * nat$ $\lfloor$*real x powr log 2 3*$\rfloor$])
  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* ($n * 2$ ^ n) *powr log 2 3*) $\in O(\lambda n.$ *real* (2 ^ (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 ... = 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* ‹*c* > *1*›, *of 1*] **have** *real* ∈ *o*(λ*n*. *c powr real n*) **by** *simp*
then have (λ*n*. *real* (*n* ∗ *2* ^ *n*)) ∈ *o*(λ*n*. (*c powr real n*) ∗ *real* (*2* ^ *n*))
by *simp*
then have (λ*n*. *real* (*n* ∗ *2* ^ *n*)) ∈ *O*(λ*n*. (*c powr real n*) ∗ *real* (*2* ^ *n*))
using *landau-o.small-imp-big* **by** *blast*
then have (λ*n*. *real* (*n* ∗ *2* ^ *n*) *powr log 2 3*) ∈ *O*(λ*n*. ((*c powr real n*) ∗ *real* (*2* ^ *n*)) *powr log 2 3*)
by (*intro iffD2*[*OF bigo-powr-iff*]; *simp*)
also have ... = *O*(λ*n*. ((*c powr real n*) ∗ *2 powr* (*real n*)) *powr log 2 3*)
using *powr-realpow*[*of 2*] **by** *simp*
also have ... = *O*(λ*n*. (((*2 powr log 2 c*) *powr real n*) ∗ *2 powr* (*real n*)) *powr log 2 3*)
using *powr-log-cancel*[*of 2 c*] ‹*c* > *1*› **by** *simp*
also have ... = *O*(λ*n*. *2 powr* ((*log 2 c* ∗ *real n* + *real n*) ∗ *log 2 3*))
unfolding *powr-powr powr-add*[*symmetric*] **by** (*rule refl*)
also have ... = *O*(λ*n*. *2 powr* (*real n* ∗ (*log 2 c* + *1*) ∗ *log 2 3*))
apply (*intro-cong* [*cong-tag-1* (λ*f*. *O*(*f*)), *cong-tag-2* (*powr*), *cong-tag-2* (∗)] *more*: *refl ext*)
by *argo*
also have ... = *O*(λ*n*. *2 powr* (*real n* ∗ *2*))
apply (*intro-cong* [*cong-tag-1* (λ*f*. *O*(*f*)), *cong-tag-2* (*powr*)] *more*: *ext refl*)
using *1* **by** *simp*
also have ... = *O*(λ*n*. *real* (*2* ^ (*2* ∗ *n*)))
apply (*intro-cong* [*cong-tag-1* (λ*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*. ∀ *n* ≥ *1*. *real* (*n* ∗ *2* ^ *n*) *powr log 2 3* ≤ *c* ∗ *real* (*2* ^ (*2* ∗ *n*)))

lemma *kar-aux-lem-le*:
assumes *n* > *0*
shows *real* (*n* ∗ *2* ^ *n*) *powr log 2 3* ≤ *kar-aux-const* ∗ *real* (*2* ^ (*2* ∗ *n*))
proof −
have (∃ *c*. ∀ *n* ≥ *1*. *real* (*n* ∗ *2* ^ *n*) *powr log 2 3* ≤ *c* ∗ *real* (*2* ^ (*2* ∗ *n*)))
using *eventually-early-real*[*OF kar-aux-lem*] **by** *simp*
then have ∀ *n* ≥ *1*. *real* (*n* ∗ *2* ^ *n*) *powr log 2 3* ≤ *kar-aux-const* ∗ *real* (*2* ^ (*2* ∗ *n*))
unfolding *kar-aux-const-def* **apply** (*intro someI-ex*[*of* λ*c*. ∀ *n* ≥ *1*. *real* (*n* ∗ *2* ^ *n*) *powr log 2 3* ≤ *c* ∗ *real* (*2* ^ (*2* ∗ *n*))]) **.**
then show *?thesis* **using** *assms* **by** *simp*

qed

lemma *kar-aux-const-gt-0*: *kar-aux-const* > *0*
proof (*rule ccontr*)
  assume ¬ *kar-aux-const* > *0*
  then have *kar-aux-const* ≤ *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* ^ (*2* ∗ *n*)) + *8410* + *4* ∗ *karatsuba-lower-bound* + *schoenhage-strassen-Fm-bound n* ∗ *2* ^ (*oe-n* − *1*)
by (*simp add*: *add-mult-distrib*)
also have ... ≤ *23525* ∗ (*m* ∗ *2* ^ *m*) + (*8093* + *kar-aux-const-nat*) ∗ (*m* ∗ (*2* ∗ *2* ^ (*m* + *1*))) + *8410* + *4* ∗ *karatsuba-lower-bound* + *schoenhage-strassen-Fm-bound n* ∗ *2* ^ (*oe-n* − *1*)
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* ^ *m*) + (*8410* + *4* ∗ *karatsuba-lower-bound*) + *schoenhage-strassen-Fm-bound n* ∗ *2* ^ (*oe-n* − *1*)
by (*simp add*: *add-mult-distrib*)
also have ... ≤ (*55897* + *4* ∗ *kar-aux-const-nat*) ∗ (*m* ∗ *2* ^ *m*) + (*8410* + *4* ∗ *karatsuba-lower-bound*) + *schoenhage-strassen-Fm-bound′ n* ∗ *2* ^ (*oe-n* − *1*)
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* ^ *m*) + (*8410* + *4* ∗ *karatsuba-lower-bound*) + *schoenhage-strassen-Fm-bound′* ((*m* + *2*) *div 2*) ∗ *2* ^ ((*m* + *1*) *div 2*)
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* ^ (*2* ∗ *n* − *2*) + *schoenhage-strassen-Fm-bound′ n* ∗ *2* ^ (*n* − *1*)
and *schoenhage-strassen-Fm-bound′* (*2* ∗ *n* − *1*) ≤ *γ-0* ∗ *n* ∗ *2* ^ (*2* ∗ *n* − *1*) + *schoenhage-strassen-Fm-bound′ n* ∗ *2* ^ *n*
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* ^ (*2* ∗ *n* − *1*) + *s-const2* + *schoenhage-strassen-Fm-bound′ n* ∗ *2* ^ *n*
using *schoenhage-strassen-Fm-bound′.simps*[*of 2* ∗ *n* − *1*] **by** *auto*
also have ... ≤ *s-const1* ∗ (*2* ∗ *n*) ∗ *2* ^ (*2* ∗ *n* − *1*) + *s-const2* ∗ (*n* ∗ *2* ^ (*2* ∗ *n* − *1*)) + *schoenhage-strassen-Fm-bound′ n* ∗ *2* ^ *n*
apply (*intro add-mono order.refl mult-le-mono*)
subgoal by *simp*
subgoal using *assms* **by** *simp*
done
also have ... = *γ-0* ∗ *n* ∗ *2* ^ (*2* ∗ *n* − *1*) + *schoenhage-strassen-Fm-bound′ n* ∗ *2* ^ *n*
unfolding *γ-0-def* **by** (*simp add*: *add-mult-distrib*)
finally show *schoenhage-strassen-Fm-bound′* (*2* ∗ *n* − *1*) ≤ ... **.**
from *r* **have** *schoenhage-strassen-Fm-bound′* (*2* ∗ *n* − *2*) = *s-const1* ∗ ((*2* ∗ *n* − *2*) ∗ *2* ^ (*2* ∗ *n* − *2*)) + *s-const2* +
schoenhage-strassen-Fm-bound′ ((*2* ∗ *n* − *2* + *2*) *div 2*) ∗ *2* ^ ((*2* ∗ *n* − *2* + *1*) *div 2*)
using *schoenhage-strassen-Fm-bound′.simps*(*2*)[*of 2* ∗ *n* − *2*] **by** *fastforce*
also have ... = *s-const1* ∗ ((*2* ∗ *n* − *2*) ∗ *2* ^ (*2* ∗ *n* − *2*)) + *s-const2* +
schoenhage-strassen-Fm-bound′ n ∗ *2* ^ (*n* − *1*)
apply (*intro-cong* [*cong-tag-2* (+), *cong-tag-2* (∗), *cong-tag-2* (^), *cong-tag-1 schoenhage-strassen-Fm-bound′*] *more*: *refl*)
subgoal using *r* **by** *linarith*
subgoal using *r* **by** *linarith*
done
also have ... ≤ *s-const1* ∗ ((*2* ∗ *n*) ∗ *2* ^ (*2* ∗ *n* − *2*)) + *s-const2* ∗ (*n* ∗ *2* ^ (*2* ∗ *n* − *2*)) + *schoenhage-strassen-Fm-bound′ n* ∗ *2* ^ (*n* − *1*)
apply (*intro add-mono order.refl mult-le-mono*)
subgoal by *simp*
subgoal using *assms* **by** *simp*
done
also have ... = *γ-0* ∗ *n* ∗ *2* ^ (*2* ∗ *n* − *2*) + *schoenhage-strassen-Fm-bound′ n* ∗ *2* ^ (*n* − *1*)
unfolding *γ-0-def* **by** (*simp add*: *add-mult-distrib*)
finally show *schoenhage-strassen-Fm-bound′* (*2* ∗ *n* − *2*) ≤ ... **.**
qed

definition *γ* **where** *γ* = *Max* {*γ-0*, *schoenhage-strassen-Fm-bound′ 0*, *schoenhage-strassen-Fm-bound′ 1*, *schoenhage-strassen-Fm-bound′ 2*, *schoenhage-strassen-Fm-bound′ 3*}

lemma *schoenhage-strassen-Fm-bound′-le-aux1*:
assumes *m* ≤ *2* ^ *Suc k* + *1*

```
  shows schoenhage-strassen-Fm-bound' m ≤ γ ∗ Suc k ∗ 2 ^ (Suc k + m)
  using assms proof (induction k arbitrary: m rule: less-induct)
  case (less k)
  consider m ≤ 3 | m ≥ 4 by linarith
  then show ?case
  proof cases
    case 1
    then have m ∈ {0, 1, 2, 3} by auto
    then have schoenhage-strassen-Fm-bound' m ∈ {γ-0, schoenhage-strassen-Fm-bound' 0, schoenhage-strassen-Fm-bound' 1, schoenhage-strassen-Fm-bound' 2, schoenhage-strassen-Fm-bound' 3} by auto
    then have schoenhage-strassen-Fm-bound' m ≤ γ unfolding γ-def by (intro Max.coboundedI; simp)
    also have ... = γ ∗ 1 ∗ 1 by simp
    also have ... ≤ γ ∗ Suc k ∗ 2 ^ (Suc k + m)
      by (intro mult-le-mono order.refl; simp)
    finally show ?thesis .
  next
    case 2
    have k > 0
    proof (rule ccontr)
      assume ¬ k > 0
      with less.prems have m ≤ 3 by simp
      thus False using 2 by simp
    qed
    then obtain k' where k = Suc k' k' < k
      using gr0-conv-Suc by auto
    have ih': schoenhage-strassen-Fm-bound' m ≤ γ ∗ k ∗ 2 ^ (k + m) if m ≤ 2 ^ k + 1 for m
      using less.IH[OF ‹k' < k›] unfolding ‹k = Suc k'›[symmetric] using that by simp

    interpret ml: m-lemmas m
      apply unfold-locales
      using 2 by simp

    define n' where n' = (if odd m then ml.n 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 ‹n' = ml.oe-n − 1›
      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 ≥ 3 using 2 ml.mn by (cases odd m; simp)
    have ml.n ≤ 2 ^ k + 1
      using less.prems ml.mn by (cases odd m; simp)
```

note *ih* = *ih*′[*OF this*]

have *schoenhage-strassen-Fm-bound*′ *m* ≤ *γ-0* ∗ *ml.n* ∗ *2* ^ *m* + *schoenhage-strassen-Fm-bound*′ *ml.n* ∗ *2* ^ *n*′
unfolding *n*′*-def*
using *schoenhage-strassen-Fm-bound*′*-oe-rec*[*OF* ‹*ml.n* ≥ *3*›] *ml.mn*
by (*cases odd m*; *algebra*)
also have ... ≤ *γ* ∗ *ml.n* ∗ *2* ^ *m* + (*γ* ∗ *k* ∗ *2* ^ (*k* + *ml.n*)) ∗ *2* ^ *n*′
apply (*intro add-mono mult-le-mono order.refl ih*)
apply (*unfold γ-def*)
apply *simp*
done
also have ... = *γ* ∗ *ml.n* ∗ *2* ^ *m* + *γ* ∗ *k* ∗ *2* ^ (*k* + *ml.n* + *n*′)
by (*simp add*: *power-add*)
also have ... = *γ* ∗ *ml.n* ∗ *2* ^ *m* + *γ* ∗ *k* ∗ *2* ^ (*k* + *m* + *1*)
using ‹*ml.n* + *n*′ = *m* + *1*› **by** (*simp add*: *add.assoc*)
also have ... = *γ* ∗ *2* ^ *m* ∗ (*ml.n* + *k* ∗ *2* ^ (*k* + *1*))
by (*simp add*: *Nat.add-mult-distrib2 power-add*)
also have ... ≤ *γ* ∗ *2* ^ *m* ∗ (*2* ^ (*k* + *1*) + *k* ∗ *2* ^ (*k* + *1*))
apply (*intro mono-intros*)
apply (*estimation estimate*: ‹*ml.n* ≤ *2* ^ *k* + *1*›)
apply *simp*
done
also have ... = *γ* ∗ *2* ^ *m* ∗ (*k* + *1*) ∗ *2* ^ (*k* + *1*)
by (*simp add*: *Nat.add-mult-distrib2 Nat.add-mult-distrib*)
also have ... = *γ* ∗ (*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* ≥ *1*
assumes *m* ≤ *2* ^ *k* + *1*
shows *schoenhage-strassen-Fm-bound*′ *m* ≤ *γ* ∗ *k* ∗ *2* ^ (*k* + *m*)
proof −
from *assms*(*1*) **obtain** *k*′ **where** *k* = *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 ℕ

definition *schoenhage-strassen-mul-tm* **where**
schoenhage-strassen-mul-tm a b =*1 do* {
bits-a ← *length-tm a* ≫= *bitsize-tm*;
bits-b ← *length-tm b* ≫= *bitsize-tm*;
m′ ← *max-nat-tm bits-a bits-b*;

```
  m ← m′ +t 1;
  m-plus-1 ← m +t 1;
  car-len ← 2 ^t m-plus-1;
  fill-a ← fill-tm car-len a;
  fill-b ← fill-tm car-len b;
  fm-result ← 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-lsbf-fermat.fermat-non-unique-carrierI length-fill length-a′)
    subgoal unfolding fill-b-def car-len-def
      by (intro int-lsbf-fermat.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-lsbf-fermat.val-reduce-tm defs[symmetric]
  by (rule refl)
qed

lemma real-power: a > 0 ⟹ real ((a :: nat) ^ x) = real a powr real x
  using powr-realpow[of real a x] by simp

definition schoenhage-strassen-bound where
schoenhage-strassen-bound n = 146 ∗ n + 218 + 4 ∗ (bitsize n + 1) + 126 ∗ 2 ^
(bitsize n + 2) +
    γ ∗ bitsize (bitsize n + 1) ∗ 2 ^ (bitsize (bitsize n + 1) + (bitsize n + 1))

theorem time-schoenhage-strassen-mul-tm-le:
  assumes length a ≤ n length b ≤ n
  shows time (schoenhage-strassen-mul-tm a b) ≤ schoenhage-strassen-bound n
proof −
  interpret schoenhage-strassen-mul-context a b .

  have m-le: m ≤ 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*: *bits-a* ≤ *m* − *1*
unfolding *bits-a-def*
by (*intro iffD2*[*OF bitsize-length*] *length-a*)
have *bits-b-le*: *bits-b* ≤ *m* − *1*
unfolding *bits-b-def*
by (*intro iffD2*[*OF bitsize-length*] *length-b*)

have *a-carrier*: *fill-a* ∈ *int-lsbf-fermat.fermat-non-unique-carrier m*
unfolding *fill-a-def car-len-def*
by (*intro int-lsbf-fermat.fermat-non-unique-carrierI length-fill length-a′*)
have *b-carrier*: *fill-b* ∈ *int-lsbf-fermat.fermat-non-unique-carrier m*
unfolding *fill-b-def car-len-def*
by (*intro int-lsbf-fermat.fermat-non-unique-carrierI length-fill length-b′*)

have *val-fm*: *val* (*schoenhage-strassen-tm m fill-a fill-b*) = *schoenhage-strassen m fill-a fill-b*
by (*intro val-schoenhage-strassen-tm a-carrier b-carrier*)

have *time* (*schoenhage-strassen-mul-tm a b*) = *time* (*length-tm a*) + *time* (*bitsize-tm* (*length a*)) + *time* (*length-tm b*) +
time (*bitsize-tm* (*length b*)) +
time (*max-nat-tm bits-a bits-b*) +
time ($m' +_t 1$) +
time ($m +_t 1$) +
time ($2\ \widehat{}_t\ (m + 1)$) +
time (*fill-tm car-len a*) +
time (*fill-tm car-len b*) +
time (*schoenhage-strassen-tm m fill-a fill-b*) +
time (*int-lsbf-fermat.reduce-tm m* (*schoenhage-strassen m 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 ... ≤ (*n* + *1*) + (*72* ∗ *n* + *23*) + (*n* + *1*) +
(*72* ∗ *n* + *23*) +
(*2* ∗ (*m* − *1*) + *3*) +
m +
(*m* + *1*) +
12 ∗ *2* ^ (*m* + *1*) +
(*3* ∗ *2* ^ (*m* + *1*) + *5*) +
(*3* ∗ *2* ^ (*m* + *1*) + *5*) +
schoenhage-strassen-Fm-bound′ m +
(*155* + *108* ∗ *2* ^ (*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-lsbf-fermat.time-reduce-tm-le*)
unfolding *int-lsbf-fermat.fermat-carrier-length*[*OF conjunct2*[*OF schoenhage-strassen-correct'*[*OF a-carrier b-carrier*]]]
by *simp*
done
also have ... $= 146 * n + 218 +$
$2 * (m - 1) + 2 * m + 126 * 2 \,\hat{}\, (m + 1) +$ *schoenhage-strassen-Fm-bound'* m
by *simp*
also have ... $\leq 146 * n + 218 +$
$4 * m + 126 * 2 \,\hat{}\, (m + 1) +$ *schoenhage-strassen-Fm-bound'* m
by *simp*
also have ... $\leq 146 * n + 218 +$
$4 * m + 126 * 2 \,\hat{}\, (m + 1) + (\gamma * \mathit{bitsize}\ m * 2 \,\hat{}\, (\mathit{bitsize}\ m + m))$
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 ... $\leq 146 * n + 218 + 4 * (\mathit{bitsize}\ n + 1) + 126 * 2 \,\hat{}\, (\mathit{bitsize}\ n + 2) +$
$\gamma * \mathit{bitsize}\ (\mathit{bitsize}\ n + 1) * 2 \,\hat{}\, (\mathit{bitsize}\ (\mathit{bitsize}\ n + 1) + (\mathit{bitsize}\ n + 1))$
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*: $a \leq b \Longrightarrow \mathit{real}\ (b - a) = \mathit{real}\ b - \mathit{real}\ a$

by *simp*

lemma *bitsize-le-log*: $n > 0 \Longrightarrow real\ (bitsize\ n) \leq log\ 2\ (real\ n) + 1$
proof −
assume $n > 0$
then have $bitsize\ n > 0$ **using** *bitsize-zero-iff*[*of n*] **by** *simp*
then have $\neg\ (bitsize\ n \leq bitsize\ n - 1)$ **by** *simp*
then have $n \geq 2\ \hat{}\ (bitsize\ n - 1)$ **using** *bitsize-length*[*of n bitsize n* − *1*] **by** *simp*
then have $log\ 2\ (real\ n) \geq real\ (bitsize\ n - 1)$
using *le-log2-of-power* **by** *simp*
then show *?thesis* **by** *simp*
qed

lemma *powr-mono-base2*: $a \leq b \Longrightarrow 2\ powr\ (a :: real) \leq 2\ powr\ b$
by (*intro powr-mono*; *simp*)

lemma *log-mono-base2*: $a > 0 \Longrightarrow b > 0 \Longrightarrow a \leq b \Longrightarrow log\ 2\ a \leq log\ 2\ b$
using *log-le-cancel-iff*[*of 2 a b*] **by** *simp*

lemma *log-nonneg-base2*: $x \geq 1 \Longrightarrow log\ 2\ x \geq 0$
using *zero-le-log-cancel-iff*[*of 2 x*] **by** *simp*

lemma *powr-log-cancel-base2*: $x > 0 \Longrightarrow 2\ powr\ (log\ 2\ x) = x$
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*: *schoenhage-strassen-bound* $\in O(\lambda n.\ n * log\ 2\ n * log\ 2\ (log\ 2\ n))$
proof −

define *explicit-bound* **where** *explicit-bound* $= (\lambda x.\ 1154 * x + 226 + 4 * log\ 2\ x + (real\ \gamma * 24) * x * log\ 2\ x * log\ 2\ (log\ 2\ x) + (real\ \gamma * 24 * (1 + log\ 2\ 3)) * x * log\ 2\ x)$

have *le*: *real* (*schoenhage-strassen-bound n*) $\leq$ *explicit-bound* (*real n*) **if** $n \geq 2$ **for** *n*
proof −
have (*2::nat*) > 0 **by** *simp*
from *that* **have** $n \geq 1$ $n > 0$ **by** *simp-all*

have *0*: *bitsize n* $+ 1 > 0$ **by** *simp*
define *x* **where** *x* = *real n*
then have $x \geq 2$ $x \geq 1$ $x > 0$ **using** ‹$n \geq 2$› ‹$n \geq 1$› ‹$n > 0$› **by** *simp-all*

have *log-ge*: $log\ 2\ x \geq 1$ **using** *log-mono-base2*[*of 2 x*] **using** ‹$x \geq 2$› **by** *simp*
then have *log-log-ge*: $log\ 2\ (log\ 2\ x) \geq 0$ **and** $log\ 2\ x > 0$ **by** *simp-all*

have *log-n*: *real* (*bitsize n*) $\leq log\ 2\ x + 1$
unfolding *x-def* **by** (*rule bitsize-le-log*[*OF* ‹$n > 0$›])

have *log-log-n*: *real* (*bitsize* (*bitsize n* + *1*)) $\leq log\ 2\ (log\ 2\ x) + (1 + log\ 2\ 3)$
proof −
have *real* (*bitsize* (*bitsize n* + *1*)) $\leq$ *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 ... $\leq$ *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*] ‹$n \geq 1$› **by** *linarith*
done
also have ... $= log\ 2\ (log\ 2\ x + 2 * 1) + 1$ **unfolding** *x-def* **by** *argo*
also have ... $\leq log\ 2\ (log\ 2\ x + 2 * log\ 2\ x) + 1$
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*: $0 \leq log\ 2\ (log\ 2\ x) + (1 + log\ 2\ 3)$

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 ... ≤ *146* ∗ *x* + *218* + *4* ∗ ((*log 2 x* + *1*) + *1*) + *126* ∗ *2 powr* ((*log 2 x* + *1*) + *2*) +
real γ ∗ (*log 2* (*log 2 x*) + (*1* + *log 2 3*)) ∗ *2 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*) + *real* γ ∗ (*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* + (*real* γ ∗ *24*) ∗ *x* ∗ *log 2 x* ∗ *log 2* (*log 2 x*) + (*real* γ ∗ *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* ∈ *O*(*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* ∈ *O*(λ*n*. *n* ∗ *log 2 n* ∗ *log 2* (*log 2 n*))
unfolding *explicit-bound-def*
apply (*intro sum-in-bigo*(*1*))
subgoal
proof −
have (∗) *1154* ∈ *O*(λ*x*. *x*) **by** *simp*
moreover have *1* ∈ *O*(λ*x*. *log 2 x*) **by** (*rule const-bigo-log*)
moreover have *1* ∈ *O*(λ*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*: (λ*x*. *225*) ∈ *O*(λ*x*. *x* :: *real*) **by** *simp*

```
      have b: 1 ∈ O(λx. log 2 x) by (rule const-bigo-log)
      have c: (λx. 225) ∈ O(λx. x * log 2 x)
        using landau-o.big-mult[OF a b] by simp
      have d: 1 ∈ O(λx. 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: (λx. 4) ∈ O(λx. x :: real) by simp
      have b: (λx. 4 * log 2 x) ∈ O(λx. x * log 2 x)
        by (rule landau-o.big.mult-right[OF a])
      have c: 1 ∈ O(λx. 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: (λx. real γ * 24 * x) ∈ O(λx. x :: real) by simp
      show ?thesis by (intro landau-o.big.mult-right a)
    qed
    subgoal
    proof –
      have a: (λx. real γ * 24 * (1 + log 2 3) * x) ∈ O(λx. x :: real) by simp
      have b: (λx. real γ * 24 * (1 + log 2 3) * x * log 2 x) ∈ O(λx. 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, 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.