Schönhage-Strassen Multiplication on Integers

Jakob Schulz

September 4, 2026

Abstract

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

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

Contents

1 Preliminaries **2**
- 1.1 Some Running Time Formalizations . . . 4
- 1.2 Auxiliary Lemmas for Landau Notation . . . 6
- 1.3 Multiplicative Subgroups . . . 8
- 1.4 Additive Subgroups . . . 10

2 Number Theoretic Transforms in Rings **10**
- 2.1 Roots of Unity . . . 11
- 2.2 Primitive Roots . . . 13
- 2.3 Number Theoretic Transforms . . . 13
  - 2.3.1 Inversion Rule . . . 14
  - 2.3.2 Convolution Theorem . . . 15
- 2.4 Fast Number Theoretic Transforms in Rings . . . 16

3 The Schoenhage-Strassen Algorithm **20**
- 3.1 Representing $\mathbb{Z}_{2^n}$ . . . 20
- 3.2 Representing $\mathbb{Z}_{F_n}$ . . . 23
- 3.3 Implementing FNTT in $\mathbb{Z}_{F_n}$ . . . 29
- 3.4 Final Preparations . . . 40
  - 3.4.1 A special residue problem . . . 46
  - 3.4.2 Subroutine for combining the final result . . . 47

3.5 Schoenhage-Strassen Multiplication in $\mathbb{Z}_{F_m}$ 48
3.6 Schoenhage-Strassen Multiplication in $\mathbb{N}$ 52

4 Running Time Formalization **53**
4.1 Multiplication in $\mathbb{N}$ 60

1 Preliminaries

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

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

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

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

lemma *const-diff-mod-idem*:
assumes *m* ≥ (*n* :: *nat*)
f = (λ*i*. (*m* − *i*) *mod n*)
shows (⋀*i*. *i* ∈ {*0*..<*n*} ⟹ *f* (*f i*) = *i*)
⟨*proof*⟩

lemma *const-diff-mod-bij*: *m* ≥ (*n* :: *nat*) ⟹ *bij-betw* (λ*i*. (*m* − *i*) *mod n*) {*0*..<*n*} {*0*..<*n*}
⟨*proof*⟩

lemma *const-add-mod-bij*: *bij-betw* (λ*i*. ((*m* :: *nat*) + *i*) *mod n*) {*0*..<*n*} {*0*..<*n*}
⟨*proof*⟩

lemma *mod-diff-add-eq*: (*a* − *b* + *c*) *mod* (*m* :: *int*) = (*a mod m* − *b mod m* + *c mod m*) *mod m*
⟨*proof*⟩

lemma *set-map-subseteqI*:
assumes ⋀*x*. *x* ∈ *A* ⟹ *f x* ∈ *B*
assumes *set xs* ⊆ *A*
shows *set* (*map f xs*) ⊆ *B*
⟨*proof*⟩

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*⟩

lemma *set-map2*:
assumes $z \in set\ (map2\ f\ xs\ ys)$
shows $\exists x\ y.\ x \in set\ xs \wedge y \in set\ ys \wedge z = f\ x\ y$
⟨*proof*⟩

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*⟩

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*⟩

lemma *set-map3*:
assumes $w \in set\ (map3\ f\ xs\ ys\ zs)$
shows $\exists x\ y\ z.\ x \in set\ xs \wedge y \in set\ ys \wedge z \in set\ zs \wedge w = f\ x\ y\ z$
⟨*proof*⟩

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*⟩

lemma *map3-compose3*: $map3\ (\lambda x\ y\ z.\ f\ x\ y\ (g\ z))\ xs\ ys\ zs = map3\ f\ xs\ ys\ (map\ g\ zs)$
⟨*proof*⟩

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*]: $rotate\text{-}left\ k\ xs = rotate\ k\ xs$
⟨*proof*⟩

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

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

lemma *rotate-right-rotate*[*simp*]: *rotate-right k* (*rotate k xs*) = *xs*
⟨*proof*⟩
lemma *rotate-rotate-right*[*simp*]: *rotate k* (*rotate-right k xs*) = *xs*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *rotate-right-conv-mod*: *rotate-right n xs* = *rotate-right* (*n mod length xs*) *xs*
⟨*proof*⟩

lemma *mod-diff-right-eq-nat*:
assumes (*a*::*nat*) ≥ *b*
shows (*a* − *b*) *mod m* = (*a* − (*b mod m*)) *mod m*
⟨*proof*⟩

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

lemma *nth-rotate-right*: *n* < *length xs* ⟹ *m* < *length xs* ⟹ *rotate-right m xs* ! *n* = *xs* ! ((*n* + *length xs* − *m*) *mod length xs*)
⟨*proof*⟩

end

1.1 Some Running Time Formalizations

theory *Schoenhage-Strassen-Runtime-Preliminaries*
imports
Main
Karatsuba.Time-Monad-Extended
Karatsuba.Main-TM
Karatsuba.Karatsuba-Preliminaries
Karatsuba.Nat-LSBF
Karatsuba.Nat-LSBF-TM
Karatsuba.Estimation-Method
Schoenhage-Strassen-Preliminaries
Akra-Bazzi.Akra-Bazzi
HOL−Library.Landau-Symbols
begin

fun *zip-tm* :: *'a list* ⇒ *'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*
⟨*proof*⟩

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

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* (λ*x y z*. *val* (*f x y z*)) *xs ys zs*
⟨*proof*⟩

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*
⟨*proof*⟩

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* (λ*x y z w*. *val* (*f x y z w*)) *xs ys zs ws*
⟨*proof*⟩

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*
⟨*proof*⟩

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* (λ*x y*. *val* (*f x y*)) *xs ys*
⟨*proof*⟩

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*⟩

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*
⟨*proof*⟩

lemma *time-rotate-left-tm-le*: *time* (*rotate-left-tm k xs*) ≤ *13* + *14* ∗ *max k* (*length xs*)
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *time-rotate-right-tm-le*: *time* (*rotate-right-tm k xs*) ≤ *23* + *26* ∗ *max k* (*length xs*)
⟨*proof*⟩

1.2 Auxiliary Lemmas for Landau Notation

lemma *eventually-early-nat*:
fixes *f g* :: *nat* ⇒ *nat*

assumes $f \in O(g)$
assumes $\bigwedge x.\ x \geq n0 \Longrightarrow g\ x > 0$
shows $\exists\, c.\ (\forall\, x.\ x \geq n0 \longrightarrow f\ x \leq c * g\ x)$
⟨*proof*⟩

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

lemma *floor-in-nat-iff*: $floor\ x \in \mathbb{N} \longleftrightarrow x \geq 0$
⟨*proof*⟩

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

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 $a \ominus (b \ominus c) = a \ominus b \oplus c$
⟨*proof*⟩
lemma *minus-eq-mult-one*:
assumes $a \in carrier\ R$
shows $\ominus\ a = (\ominus\ \mathbf{1}) \otimes a$
⟨*proof*⟩
lemma *diff-eq-add-mult-one*:
assumes $a \in carrier\ R\ b \in carrier\ R$
shows $a \ominus b = a \oplus (\ominus\ \mathbf{1}) \otimes b$
⟨*proof*⟩
lemma *minus-cancel*:
assumes $a \in carrier\ R\ b \in carrier\ R$
shows $a \ominus b \oplus b = a$
⟨*proof*⟩
lemma *assoc4*:
assumes $a \in carrier\ R\ b \in carrier\ R\ c \in carrier\ R\ d \in carrier\ R$

shows $a \otimes (b \otimes (c \otimes d)) = a \otimes b \otimes c \otimes d$
⟨*proof*⟩
lemma *diff-sum*:
assumes $a \in carrier\ R$ $b \in carrier\ R$ $c \in carrier\ R$ $d \in carrier\ R$
shows $(a \ominus c) \oplus (b \ominus d) = (a \oplus b) \ominus (c \oplus d)$
⟨*proof*⟩

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 $inv\ z \otimes x = y$
⟨*proof*⟩

lemma (**in** *ring*) *r-distr-diff*:
assumes $x \in carrier\ R$
assumes $y \in carrier\ R$
assumes $z \in carrier\ R$
shows $x \otimes (y \ominus z) = x \otimes y \ominus x \otimes z$
⟨*proof*⟩

lemma (**in** *group*)
assumes $x \in carrier\ G$
shows $\bigwedge i.\ i \in \{1..<ord\ x\} \Longrightarrow x\ [\hat{}]\ i \neq \mathbf{1}$
⟨*proof*⟩

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*]: $carrier\ M = X$ ⟨*proof*⟩

lemma *one-eq*: $\mathbf{1}_M = \mathbf{1}$ ⟨*proof*⟩

lemma *mult-eq*: $a \otimes_M b = a \otimes b$ ⟨*proof*⟩

lemma *inv-eq*:
assumes $x \in X$
shows $inv_M\ x = inv\ x$
⟨*proof*⟩

lemma *nat-pow-eq*: $x\ [\widehat{}]_M\ (m :: nat) = x\ [\widehat{}]\ m$
⟨*proof*⟩

lemma *int-pow-eq*:
assumes $x \in X$
shows $x\ [\widehat{}]_M\ (i :: int) = x\ [\widehat{}]\ i$
⟨*proof*⟩

end

context *cring*
begin

interpretation *units-group*: *group units-of R*
⟨*proof*⟩

lemma *units-subgroup*: *multiplicative-subgroup R* (*Units R*) (*units-of R*)
⟨*proof*⟩

interpretation *units-subgroup*: *multiplicative-subgroup R Units R units-of R*
⟨*proof*⟩

lemma *inv-nat-pow*:
assumes $a \in Units\ R$
shows $inv\ (a\ [\widehat{}]\ (b :: nat)) = inv\ a\ [\widehat{}]\ b$
⟨*proof*⟩
lemma *int-pow-mult*:
fixes *m1 m2* :: *int*
assumes $x \in Units\ R$
shows $x\ [\widehat{}]\ m1 \otimes x\ [\widehat{}]\ m2 = x\ [\widehat{}]\ (m1 + m2)$
⟨*proof*⟩
lemma *int-pow-pow*:
fixes *m1 m2* :: *int*
assumes $x \in Units\ R$
shows $(x\ [\widehat{}]\ m1)\ [\widehat{}]\ m2 = x\ [\widehat{}]\ (m1 * m2)$
⟨*proof*⟩
lemma *int-pow-one*:
$\mathbf{1}\ [\widehat{}]\ (i :: int) = \mathbf{1}$
⟨*proof*⟩
lemma *int-pow-closed*:
assumes $x \in Units\ R$
shows $x\ [\widehat{}]\ (i :: int) \in Units\ R$
⟨*proof*⟩

lemma *units-of-int-pow*: $\mu \in Units\ R \Longrightarrow \mu\ [\widehat{}]_{(units\text{-}of\ R)}\ i = \mu\ [\widehat{}]\ (i :: int)$
⟨*proof*⟩

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

⟨*proof*⟩

lemma *units-inv-int-pow*: $\mu \in \mathit{Units}\ R \Longrightarrow \mathit{inv}\ \mu = \mu\ [\widehat{}]\ (-\ (1 :: \mathit{int}))$
⟨*proof*⟩

lemma *inv-prod*: $\mu \in \mathit{Units}\ R \Longrightarrow \nu \in \mathit{Units}\ R \Longrightarrow \mathit{inv}\ (\mu \otimes \nu) = \mathit{inv}\ \nu \otimes \mathit{inv}\ \mu$
⟨*proof*⟩

lemma *powers-of-negative*:
fixes $r :: \mathit{nat}$
assumes $x \in \mathit{carrier}\ R$
shows $\mathit{even}\ r \Longrightarrow (\ominus x)\ [\widehat{}]\ r = x\ [\widehat{}]\ r\ \mathit{odd}\ r \Longrightarrow (\ominus x)\ [\widehat{}]\ r = \ominus\ (x\ [\widehat{}]\ r)$
⟨*proof*⟩

end

1.4 Additive Subgroups

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

lemma *carrier-M*[*simp*]: *carrier M* = *X*
⟨*proof*⟩

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

lemma *mult-eq*: $a \otimes_M b = a \oplus b$
⟨*proof*⟩

lemma *inv-eq*:
assumes $a \in X$
shows $\mathit{inv}_M\ a = \ominus\ a$
⟨*proof*⟩

end

end

2 Number Theoretic Transforms in Rings

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

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

lemma *max-dividing-power-factorization*:
fixes $a :: nat$
assumes $a \neq 0$
assumes $k = Max\ \{s.\ p\ \hat{}\ s\ dvd\ a\}$
assumes $r = a\ div\ (p\ \hat{}\ k)$
assumes *prime p*
shows $a = r * p\ \hat{}\ k$ *coprime p r*
⟨*proof*⟩

context *cring*
begin

interpretation *units-group*: *group units-of R*
⟨*proof*⟩

interpretation *units-subgroup*: *multiplicative-subgroup R Units R units-of R*
⟨*proof*⟩

2.1 Roots of Unity

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

lemma *root-of-unityI*[*intro*]: $\mu \in carrier\ R \Longrightarrow \mu\ [\hat{}]\ n = \mathbf{1} \Longrightarrow$ *root-of-unity n μ*
⟨*proof*⟩

lemma *root-of-unityD*[*simp*]: *root-of-unity n μ* $\Longrightarrow \mu\ [\hat{}]\ n = \mathbf{1}$
⟨*proof*⟩

lemma *root-of-unity-closed*[*simp*]: *root-of-unity n μ* $\Longrightarrow \mu \in carrier\ R$
⟨*proof*⟩

context
fixes $n :: nat$
assumes $n > 0$
begin

lemma *roots-Units*[*simp*]:
assumes *root-of-unity n μ*
shows $\mu \in Units\ R$
⟨*proof*⟩

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

roots-of-unity-group ≡ ⦇ *carrier* = {μ. *root-of-unity n* μ}, *monoid.mult* = ($\otimes$), *one* = **1** ⦈

lemma *roots-of-unity-group-is-group*:
  shows *group roots-of-unity-group*
  ⟨*proof*⟩

interpretation *root-group* : *group roots-of-unity-group*
  ⟨*proof*⟩

interpretation *root-subgroup* : *multiplicative-subgroup R* {μ. *root-of-unity n* μ} *roots-of-unity-group*
  ⟨*proof*⟩

lemma *root-of-unity-inv*:
  assumes *root-of-unity n* μ
  shows *root-of-unity n* (*inv* μ)
  ⟨*proof*⟩

lemma *inv-root-of-unity*:
  assumes *root-of-unity n* μ
  shows *inv* μ = μ [^] (*n* − *1*)
⟨*proof*⟩

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*⟩

lemma *root-of-unity-nat-pow-closed*:
  assumes *root-of-unity n* μ
  shows *root-of-unity n* (μ [^] (*m* :: *nat*))
  ⟨*proof*⟩

lemma *root-of-unity-powers*:
  assumes *root-of-unity n* μ
  shows μ [^] *i* = μ [^] (*i mod n*)
⟨*proof*⟩

lemma *root-of-unity-powers-modint*:
  assumes *root-of-unity n* μ
  shows μ [^] (*i* :: *int*) = μ [^] (*i mod int n*)
⟨*proof*⟩

lemma *root-of-unity-powers-nat*:
  assumes *root-of-unity n* μ
  assumes *i mod n* = *j mod n*
  shows μ [^] *i* = μ [^] *j*

⟨*proof*⟩

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*
⟨*proof*⟩

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 ≠ 1)

lemma *primitive-rootI*[*intro*]:
assumes *μ ∈ carrier R*
assumes *μ [^] n = 1*
assumes *⋀i. i > 0 ⟹ i < n ⟹ μ [^] i ≠ 1*
shows *primitive-root n μ*
⟨*proof*⟩

lemma *primitive-root-is-root-of-unity*[*simp*]: *primitive-root n μ ⟹ root-of-unity n μ*
⟨*proof*⟩

lemma *primitive-root-recursion*:
assumes *even n*
assumes *primitive-root n μ*
shows *primitive-root (n div 2) (μ [^] (2 :: nat))*
⟨*proof*⟩

lemma *primitive-root-inv*:
assumes *n > 0*
assumes *primitive-root n μ*
shows *primitive-root n (inv μ)*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *NTT-nth*:
assumes *length a = n*
assumes *i < n*

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

lemma *NTT-nth-2*:
assumes *length a* = *n*
assumes $i < n$
assumes $\mu \in$ *carrier R*
shows $NTT\ \mu\ a\ !\ i = (\bigoplus j \leftarrow [0..<n].\ (a\ !\ j) \otimes (\mu\ [\widehat{}]\ (i * j)))$
⟨*proof*⟩

lemma *NTT-nth-closed*:
assumes *set a* ⊆ *carrier R*
assumes $\mu \in$ *carrier R*
assumes *length a* = *n*
assumes $i < n$
shows $NTT\ \mu\ a\ !\ i \in$ *carrier R*
⟨*proof*⟩

lemma *NTT-closed*:
assumes *set a* ⊆ *carrier R*
assumes $\mu \in$ *carrier R*
shows *set* $(NTT\ \mu\ a) \subseteq$ *carrier R*
⟨*proof*⟩

lemma *primitive-root 1* **1**
⟨*proof*⟩

lemma $(\ominus\ \mathbf{1})\ [\widehat{}]\ (2::nat) = \mathbf{1}$
⟨*proof*⟩
lemma $\mathbf{1} \oplus \mathbf{1} \neq \mathbf{0} \Longrightarrow$ *primitive-root 2* $(\ominus\ \mathbf{1})$
⟨*proof*⟩

2.3.1 Inversion Rule

theorem *inversion-rule*:
fixes $\mu :: 'a$
fixes $n ::$ *nat*
assumes $n > 0$
assumes *primitive-root n* 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* ⊆ *carrier R*
shows $NTT\ (inv\ \mu)\ (NTT\ \mu\ a) = map\ (\lambda x.\ nat\text{-}embedding\ n \otimes x)\ a$
⟨*proof*⟩

lemma *inv-good*:
assumes $n > 0$
assumes *primitive-root n* μ
assumes *good*: $\bigwedge i.\ i \in \{1..<n\} \Longrightarrow (\bigoplus j \leftarrow [0..<n].\ (\mu\ [\widehat{}]\ i)\ [\widehat{}]\ j) = \mathbf{0}$

shows *primitive-root n (inv μ)*
$\bigwedge i.\ i \in \{1..<n\} \Longrightarrow (\bigoplus j \leftarrow [0..<n].\ ((inv\ \mu)\ [\widehat{}]\ i)\ [\widehat{}]\ j) = \mathbf{0}$
⟨*proof*⟩

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

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

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*⟩

corollary *inversion-rule-inv*:
fixes $\mu :: 'a$
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 ⊆ carrier R*
shows $NTT\ \mu\ (NTT\ (inv\ \mu)\ a) = map\ (\lambda x.\ nat\text{-}embedding\ n \otimes x)\ a$
⟨*proof*⟩

2.3.2 Convolution Theorem

lemma *root-of-unity-power-sum-product*:
assumes *root-of-unity n x*
assumes[*simp*]: $\bigwedge i.\ i < n \Longrightarrow f\ i \in carrier\ R$
assumes[*simp*]: $\bigwedge i.\ i < n \Longrightarrow g\ i \in carrier\ R$
shows $(\bigoplus i \leftarrow [0..<n].\ f\ i \otimes x\ [\widehat{}]\ i) \otimes (\bigoplus i \leftarrow [0..<n].\ g\ i \otimes x\ [\widehat{}]\ i) =$
$(\bigoplus k \leftarrow [0..<n].\ (\bigoplus i \leftarrow [0..<n].\ f\ i \otimes g\ ((n + k - i)\ mod\ n)) \otimes x\ [\widehat{}]\ k)$
⟨*proof*⟩

context
fixes $n :: nat$

begin

definition *cyclic-convolution* :: *'a list* ⇒ *'a list* ⇒ *'a list* (**infixl** ‹⋆› *70*) **where**
cyclic-convolution a b ≡ [(⨁ σ ← [*0*..<*n*]. (*a* ! σ ⊗ *b* ! ((*n* + *i* − σ) *mod n*))). *i* ← [*0*..<*n*]]

lemma *cyclic-convolution-length*[*simp*]:
length (*a* ⋆ *b*) = *n* ⟨*proof*⟩

lemma *cyclic-convolution-nth*:
i < *n* ⟹ (*a* ⋆ *b*) ! *i* = (⨁ σ ← [*0*..<*n*]. (*a* ! σ ⊗ *b* ! ((*n* + *i* − σ) *mod n*)))
⟨*proof*⟩

lemma *cyclic-convolution-closed*:
assumes *length a* = *n* *length b* = *n*
assumes *set a* ⊆ *carrier R* *set b* ⊆ *carrier R*
shows *set* (*a* ⋆ *b*) ⊆ *carrier R*
⟨*proof*⟩

theorem *convolution-rule*:
assumes *length a* = *n*
assumes *length b* = *n*
assumes *set a* ⊆ *carrier R*
assumes *set b* ⊆ *carrier R*
assumes *root-of-unity n* μ
assumes *i* < *n*
shows *NTT* μ *a* ! *i* ⊗ *NTT* μ *b* ! *i* = *NTT* μ (*a* ⋆ *b*) ! *i*
⟨*proof*⟩

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* $\subseteq$ *carrier R*
defines $j' \equiv$ (*if* $j < n$ *div 2 then j else* $j - n$ *div 2*)
shows $j' < n$ *div 2* $j =$ (*if* $j < n$ *div 2 then* j' *else* $j' + n$ *div 2*)
and (*NTT* μ *a*) ! j = (*NTT* (μ $\lceil^\frown\rceil$ (2::*nat*)) [*a* ! *i*. *i* $\leftarrow$ *filter even* [0..<*n*]]) ! j'
$\oplus$ μ $\lceil^\frown\rceil$ j $\otimes$ (*NTT* (μ $\lceil^\frown\rceil$ (2::*nat*)) [*a* ! *i*. *i* $\leftarrow$ *filter odd* [0..<*n*]]) ! j'
⟨*proof*⟩

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* $\subseteq$ *carrier R*
shows (*NTT* μ *a*) ! j =
(*NTT* (μ $\lceil^\frown\rceil$ (2::*nat*)) [*a* ! *i*. *i* $\leftarrow$ *filter even* [0..<*n*]]) ! j
$\oplus$ μ $\lceil^\frown\rceil$ j $\otimes$ (*NTT* (μ $\lceil^\frown\rceil$ (2::*nat*)) [*a* ! *i*. *i* $\leftarrow$ *filter odd* [0..<*n*]]) ! j
⟨*proof*⟩

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* $\subseteq$ *carrier R*
assumes *halfway-property*: μ $\lceil^\frown\rceil$ (*n div 2*) = $\ominus$ **1**
shows (*NTT* μ *a*) ! (*n div 2* + j) =
(*NTT* (μ $\lceil^\frown\rceil$ (2::*nat*)) [*a* ! *i*. *i* $\leftarrow$ *filter even* [0..<*n*]]) ! j
$\ominus$ μ $\lceil^\frown\rceil$ j $\otimes$ (*NTT* (μ $\lceil^\frown\rceil$ (2::*nat*)) [*a* ! *i*. *i* $\leftarrow$ *filter odd* [0..<*n*]]) ! j
⟨*proof*⟩

lemma *NTT-diffs*:
assumes *even n*
assumes *primitive-root n* μ
assumes *length a* = *n*
assumes $j < n$ *div 2*
assumes *set a* $\subseteq$ *carrier R*
assumes μ $\lceil^\frown\rceil$ (*n div 2*) = $\ominus$ **1**
shows *NTT* μ *a* ! j $\ominus$ *NTT* μ *a* ! (*n div 2* + j) = *nat-embedding 2* $\otimes$ (μ $\lceil^\frown\rceil$ j
$\otimes$ *NTT* (μ $\lceil^\frown\rceil$ (2::*nat*)) (*map* ((!) *a*) (*filter odd* [0..<*n*])) ! j)
⟨*proof*⟩

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

lemma *FNTT-term-aux*[*simp*]: *length* (*filter P* [0..<*l*]) < *Suc l*
⟨*proof*⟩
fun *FNTT* :: *'a* $\Rightarrow$ *'a list* $\Rightarrow$ *'a list* **where**
FNTT μ [] = []
| *FNTT* μ [*x*] = [*x*]
| *FNTT* μ [*x*, *y*] = [$x \oplus y$, $x \ominus y$]

```
| 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
  ⟨proof⟩

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
  ⟨proof⟩

end
```

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

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

lemma map-filter-shift:  map f (filter even [0..<Suc g]) =
       f 0 #  map (λ x. f (x+1)) (filter odd [0..<g])
  ⟨proof⟩

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

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
  ⟨proof⟩
```

lemma *FNTT'-termination-aux*[*simp*]: *length* (*evens-odds True xs*) < *Suc* (*length xs*)
length (*evens-odds False xs*) < *Suc* (*length xs*)
⟨*proof*⟩

(End of copy)

lemma *map-evens-odds*: *map f* (*evens-odds x a*) = *evens-odds x* (*map f a*)
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *set-evens-odds*:
set (*evens-odds x a*) ⊆ *set a*
⟨*proof*⟩

context *cring* **begin**

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

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

lemma *FNTT'-FNTT*: *FNTT' μ xs* = *FNTT μ xs*
⟨*proof*⟩

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

$g = map2\ (\oplus)\ b\ (map2\ (\otimes)\ [\mu\ [\frown]\ i.\ i \leftarrow [0..<(n\ div\ 2)]]\ c);$
$h = map2\ (\lambda x\ y.\ x \ominus y)\ b\ (map2\ (\otimes)\ [\mu\ [\frown]\ i.\ i \leftarrow [0..<(n\ div\ 2)]]$
$c)$
$in\ g@h)$

lemma *FNTT''-FNTT'*:
assumes *length a* = 2 ^ *k*
shows *FNTT'' μ a* = *FNTT' μ a*
⟨*proof*⟩

end

end

3 The Schoenhage-Strassen Algorithm

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

theory *Z-mod-power-of-2*
imports
Karatsuba.Nat-LSBF-TM
Finite-Fields.Ring-Characteristic
Karatsuba.Abstract-Representations-2
HOL−Number-Theory.Number-Theory
begin

context *cring* **begin**
lemma *pow-one-imp-unit*:
(*n*::*nat*) > 0 ⟹ *a* ∈ *carrier R* ⟹ *a* [^] *n* = **1** ⟹ *a* ∈ *Units R*
⟨*proof*⟩
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* ⟨*proof*⟩
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*
⟨*proof*⟩

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
⟨*proof*⟩

sublocale *residues n Zn*
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *to-nat-reduce*: *Nat-LSBF.to-nat* (*reduce xs*) = *Nat-LSBF.to-nat xs mod n*
⟨*proof*⟩

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*
⟨*proof*⟩

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*⟩

lemma *length-subtract-mod*: *length xs* ≤ *k* ⟹ *length ys* ≤ *k* ⟹ *length* (*subtract-mod xs ys*) ≤ *k*
⟨*proof*⟩

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

⟨*proof*⟩

lemma *subtract-mod-correct*:
  assumes *length x* ≤ *k*
  assumes *length y* ≤ *k*
  assumes *n* > *1*
  shows *to-residue-ring* (*subtract-mod x y*) = *to-residue-ring x* ⊖$_{Zn}$ *to-residue-ring y*
⟨*proof*⟩

lemma *length-from-residue-ring*: *x* < *2* ^ *k* ⟹ *length* (*from-residue-ring x*) = *k*
⟨*proof*⟩

interpretation *int-lsbf-mod*: *abstract-representation-2 from-residue-ring to-residue-ring* {*0..*<*int n*}
  rewrites *int-lsbf-mod.reduce* = *reduce*
  and *int-lsbf-mod.representations* = {*x* :: *bool list*. *length x* = *k*}
⟨*proof*⟩

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

end

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

definition *ensure-length-tm* :: *nat* ⇒ *nat-lsbf* ⇒ *nat-lsbf tm* **where**
ensure-length-tm k xs =*1 fill-tm k xs* ⤜ *take-tm k*

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

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

context *int-lsbf-mod*
begin

definition *reduce-tm* :: *nat-lsbf* ⇒ *nat-lsbf tm* **where**
reduce-tm xs =*1 ensure-length-tm k xs*

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

lemma *time-reduce-tm*[*simp*]: *time* (*reduce-tm xs*) = *8* + *2* ∗ *length xs* + *2* ∗ *k*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *time-add-mod-tm-le*: *time* (*add-mod-tm xs ys*) ≤ *14* + *4* ∗ *max* (*length xs*) (*length ys*) + *2* ∗ *k*
⟨*proof*⟩

definition *subtract-mod-tm* :: *nat-lsbf* ⇒ *nat-lsbf* ⇒ *nat-lsbf tm* **where**
subtract-mod-tm xs ys =*1 do* {
b ← *xs* ≤$_{nt}$ *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*
⟨*proof*⟩

lemma *time-subtract-mod-tm-le*: *time* (*subtract-mod-tm xs ys*) ≤ *118* + *51* ∗ *max k* (*max* (*length xs*) (*length ys*))
⟨*proof*⟩

end

end

3.2 Representing $\mathbb{Z}_{F_n}$

theory *Z-mod-Fermat*
imports
Z-mod-power-of-2
../NTT-Rings/FNTT-Rings
../Preliminaries/Schoenhage-Strassen-Preliminaries
Karatsuba.Estimation-Method
begin

lemma *to-nat-replicate-True2*:
assumes *Nat-LSBF.to-nat xs* = *2* ^ (*length xs*) − *1*
shows *xs* = *replicate* (*length xs*) *True*
⟨*proof*⟩

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

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

locale *int-lsbf-fermat* =
fixes $k :: nat$
begin

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

lemma *n-positive*[*simp*]: $n > 0$ ⟨*proof*⟩
lemma *n-gt-1*[*simp*]: $n > 1$ ⟨*proof*⟩
lemma *n-gt-2*[*simp*]: $n > 2$
⟨*proof*⟩

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

sublocale *residues n Fn*
⟨*proof*⟩

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

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

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

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

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

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

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

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

$(2::int)\ [^]_{Fn}\ (2::nat)\ \hat{}\ (k + 1) = \mathbf{1}_{Fn}$
⟨*proof*⟩

corollary *two-is-unit*: $2 \in Units\ Fn$
⟨*proof*⟩

corollary *two-in-carrier*: $2 \in carrier\ Fn$
⟨*proof*⟩

lemma *nat-mod-eqE*: $(a::nat)\ mod\ m = b\ mod\ m \Longrightarrow \exists\, i\ j.\ a + i * m = b + j * m$
⟨*proof*⟩

corollary *pow-mod-carrier-length*:
assumes $(a::nat)\ mod\ 2\ \hat{}\ (k + 1) = b\ mod\ 2\ \hat{}\ (k + 1)$
shows $2\ [^]_{Fn}\ a = 2\ [^]_{Fn}\ b$
⟨*proof*⟩
lemma *two-powers-trivial*:
assumes $s \leq 2\ \hat{}\ k$
shows $2\ [^]_{Fn}\ s = 2\ \hat{}\ s$
⟨*proof*⟩

lemma *two-powers-Units*:
assumes $s \leq 2\ \hat{}\ k$
shows $2\ \hat{}\ s \in Units\ Fn$
⟨*proof*⟩
corollary *two-powers-in-carrier*:
assumes $s \leq 2\ \hat{}\ k$
shows $2\ \hat{}\ s \in carrier\ Fn$
⟨*proof*⟩

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

interpretation *z-mod-fermat-unit-group*: *group units-of Fn*
⟨*proof*⟩

lemma *inv-of-2*[*simp*]:
$inv_{Fn}\ 2 = 2\ [^]_{Fn}\ ((2::nat)\ \hat{}\ (k + 1) - 1)$
⟨*proof*⟩

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

lemma *inv-pow-mod-carrier-length*:
assumes $(a::nat)\ mod\ 2\ \hat{}\ (k + 1) = b\ mod\ 2\ \hat{}\ (k + 1)$
shows $(inv_{Fn}\ 2)\ [\hat{}]_{Fn}\ a = (inv_{Fn}\ 2)\ [\hat{}]_{Fn}\ b$
⟨*proof*⟩

lemma
assumes $m > 0$
shows $\exists\, i\ j.\ (a::nat) = j + i * m \wedge j < m$
⟨*proof*⟩

corollary *two-powers*: $(2::nat)\ \hat{}\ a\ mod\ n = (2::nat)\ \hat{}\ (a\ mod\ (2\ \hat{}\ (k + 1)))\ mod\ n$
⟨*proof*⟩

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

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*
⟨*proof*⟩

lemma *to-residue-ring-eq-to-nat*: *Nat-LSBF.to-nat xs* < *n* ⟹ *to-residue-ring xs* = *int* (*Nat-LSBF.to-nat xs*)
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *length-divide-by-power-of-2*[*simp*]: *length* (*divide-by-power-of-2 xs m*) = *length xs*
⟨*proof*⟩

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

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*⟩

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*⟩

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

corollary *multiply-with-power-of-2-correct*:
assumes $xs \in fermat\text{-}non\text{-}unique\text{-}carrier$
shows $to\text{-}residue\text{-}ring\ (multiply\text{-}with\text{-}power\text{-}of\text{-}2\ xs\ m) = to\text{-}residue\text{-}ring\ xs \otimes_{Fn}$ $2\ [\hat{}]_{Fn}\ m$
⟨*proof*⟩

lemma
assumes $xs \in fermat\text{-}non\text{-}unique\text{-}carrier$
shows *divide-by-power-of-2-correct*: $to\text{-}residue\text{-}ring\ (divide\text{-}by\text{-}power\text{-}of\text{-}2\ xs\ m)$ $= to\text{-}residue\text{-}ring\ xs \otimes_{Fn} (inv_{Fn}\ 2)\ [\hat{}]_{Fn}\ m$
and *divide-by-power-of-2-closed*: $divide\text{-}by\text{-}power\text{-}of\text{-}2\ xs\ m \in fermat\text{-}non\text{-}unique\text{-}carrier$
⟨*proof*⟩

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

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

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

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

ys
⟨*proof*⟩

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

lemma *subtract-fermat-correct′*:
assumes *xs* ∈ *fermat-non-unique-carrier*
assumes *ys* ∈ *fermat-non-unique-carrier*
shows *subtract-fermat xs ys* ∈ *fermat-non-unique-carrier* ∧ *int* (*Nat-LSBF.to-nat* (*subtract-fermat xs ys*)) *mod n* = (*int* (*Nat-LSBF.to-nat xs*) − *int* (*Nat-LSBF.to-nat ys*)) *mod n*
⟨*proof*⟩

corollary *subtract-fermat-closed*:
assumes *xs* ∈ *fermat-non-unique-carrier*
assumes *ys* ∈ *fermat-non-unique-carrier*
shows *subtract-fermat xs ys* ∈ *fermat-non-unique-carrier*
⟨*proof*⟩

corollary *subtract-fermat-correct*:
assumes *xs* ∈ *fermat-non-unique-carrier*
assumes *ys* ∈ *fermat-non-unique-carrier*
shows *to-residue-ring* (*subtract-fermat xs ys*) = *to-residue-ring xs* $\ominus_{Fn}$ *to-residue-ring ys*
⟨*proof*⟩

end

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

definition *reduce* :: *nat-lsbf* ⇒ *nat-lsbf* **where**
reduce xs = (*let* (*ys*, *zs*) = *split xs in*
if compare-nat zs ys then
subtract-nat ys zs
else
subtract-nat (*add-nat* (*True* # *replicate* (2 ^ *k* − 1) *False* @ [*True*]) *ys*) *zs*)

lemma *reduce-correct′*:
assumes *xs* ∈ *fermat-non-unique-carrier*
shows *Nat-LSBF.to-nat* (*reduce xs*) < *n* ∧ *Nat-LSBF.to-nat* (*reduce xs*) *mod n* = *Nat-LSBF.to-nat xs mod n* **and** *length* (*reduce xs*) ≤ 2 ^ *k* + 2
⟨*proof*⟩

lemma *reduce-correct*:
assumes *xs* ∈ *fermat-non-unique-carrier*
shows *Nat-LSBF.to-nat xs mod n* = *Nat-LSBF.to-nat* (*reduce xs*)
⟨*proof*⟩

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

function *from-nat-lsbf* :: *nat-lsbf* ⇒ *nat-lsbf* **where**
$from\text{-}nat\text{-}lsbf\ xs = (if\ length\ xs \leq 2\ \widehat{}\ (k + 1)\ then\ fill\ (2\ \widehat{}\ (k + 1))\ xs$
    $else\ from\text{-}nat\text{-}lsbf\ (add\text{-}nat\ (take\ (2\ \widehat{}\ (k + 1))\ xs)\ (drop\ (2\ \widehat{}\ (k + 1))\ xs)))$
  ⟨*proof*⟩
lemma *from-nat-lsbf-dom-termination*: *All from-nat-lsbf-dom*
⟨*proof*⟩
termination ⟨*proof*⟩

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

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

lemma *length-from-nat-lsbf*: $length\ (from\text{-}nat\text{-}lsbf\ xs) = 2\ \widehat{}\ (k + 1)$
  ⟨*proof*⟩

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

lemma *n-odd*: *odd n*
  ⟨*proof*⟩

lemma *ord-2*: $ord\ n\ 2 = 2\ \widehat{}\ (k + 1)$
⟨*proof*⟩
corollary *ord-2-int*: $ord\ (int\ n)\ 2 = 2\ \widehat{}\ (k + 1)$
  ⟨*proof*⟩

lemma *two-is-primitive-root*: $primitive\text{-}root\ (2\ \widehat{}\ (k + 1))\ 2$
  ⟨*proof*⟩

lemma *two-inv-is-primitive-root*: $primitive\text{-}root\ (2\ \widehat{}\ (k + 1))\ (inv_{Fn}\ 2)$
  ⟨*proof*⟩

lemma *two-powers-primitive-root*:
  assumes $i + s = k + 1$
  assumes $i \leq k$
  shows $primitive\text{-}root\ (2\ \widehat{}\ s)\ (2\ [\widehat{}]_{Fn}\ (2{::}nat)\ \widehat{}\ i)$
⟨*proof*⟩

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*]
⟨*proof*⟩

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* (λ*x 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*]
⟨*proof*⟩

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* (λ*x 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*]
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *fft-ifft-combine-b-c-subtract-carrier*:
assumes *length bs* = *len-bc length cs* = *len-bc*

assumes *set bs* $\subseteq$ *fermat-non-unique-carrier*
assumes *set cs* $\subseteq$ *fermat-non-unique-carrier*
shows *set* (*fft-ifft-combine-b-c-subtract it l bs cs*) $\subseteq$ *fermat-non-unique-carrier*
⟨*proof*⟩

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* ⟨*proof*⟩
lemma *e-ge-2*: *e* ≥ 2
⟨*proof*⟩
lemma *e-pos*: *e* > 0 ⟨*proof*⟩
lemma *two-pow-e-div-2*: (2::*nat*) ^ *e div* 2 = 2 ^ (*e* − 1)
⟨*proof*⟩
lemma *two-pow-e-as-sum*: (2::*nat*) ^ *e* = 2 ^ (*e* − 1) + 2 ^ (*e* − 1)
⟨*proof*⟩

lemma
shows *length-nums1*: *length nums1* = *2* ^ (*e* − *1*)
and *length-nums2*: *length nums2* = *2* ^ (*e* − *1*)
⟨*proof*⟩

lemma *result-eq*: *fft-ifft it l a* = *g* @ *h*
⟨*proof*⟩

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*
⟨*proof*⟩

end

context *int-lsbf-fermat*
begin

lemma *length-fft-ifft*:
assumes *length a* = *2* ^ *e*
shows *length* (*fft-ifft it l a*) = *2* ^ *e*
⟨*proof*⟩

lemma *length-fft*:
assumes *length a* = *2* ^ *e*
shows *length* (*fft l a*) = *2* ^ *e*
⟨*proof*⟩

lemma *length-ifft*:
assumes *length a* = *2* ^ *e*
shows *length* (*ifft l a*) = *2* ^ *e*
⟨*proof*⟩

end

context *fft-context* **begin**

lemma *length-b*: *length b* = *2* ^ (*e* − *1*)
⟨*proof*⟩
lemma *length-c*: *length c* = *2* ^ (*e* − *1*)
⟨*proof*⟩
lemma *length-g*: *length g* = *2* ^ (*e* − *1*)
⟨*proof*⟩
lemma *length-h*: *length h* = *2* ^ (*e* − *1*)
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *fft-carrier*:
assumes *length a* = *2* ^ *l*
assumes *set a* ⊆ *fermat-non-unique-carrier*
shows *set* (*fft s a*) ⊆ *fermat-non-unique-carrier*
⟨*proof*⟩

lemma *ifft-carrier*:
assumes *length a* = *2* ^ *l*
assumes *set a* ⊆ *fermat-non-unique-carrier*
shows *set* (*ifft s a*) ⊆ *fermat-non-unique-carrier*
⟨*proof*⟩

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*) $[\widehat{\ }]_{Fn}$ *s*) (*map to-residue-ring a*)
⟨*proof*⟩

lemma *fft-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* (*fft-ifft it s a*) = *NTT* ((*if it then* inv_{Fn} *2 else 2*) $[\widehat{\ }]_{Fn}$ *s*) (*map to-residue-ring a*)
⟨*proof*⟩

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* $[\widehat{\ }]_{Fn}$ *s*) (*map to-residue-ring a*)
⟨*proof*⟩

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)*
⟨*proof*⟩

end

end
theory *Z-mod-Fermat-TM*
imports
Z-mod-Fermat
Z-mod-power-of-2-TM
../Preliminaries/Schoenhage-Strassen-Runtime-Preliminaries
begin

fun *evens-odds-tm* :: *bool ⇒ 'a list ⇒ 'a list tm* **where**
evens-odds-tm b [] =1 return []
| *evens-odds-tm True (x # xs) =1 do {*
rs ← evens-odds-tm False xs;
return (x # rs)
}
| *evens-odds-tm False (x # xs) =1 evens-odds-tm True xs*

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

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

context *int-lsbf-fermat*
begin

definition *multiply-with-power-of-2-tm* :: *nat-lsbf ⇒ nat ⇒ nat-lsbf tm* **where**
multiply-with-power-of-2-tm xs m =1 rotate-right-tm m xs

lemma *val-multiply-with-power-of-2-tm*[*simp*, *val-simp*]:
val (multiply-with-power-of-2-tm xs m) = multiply-with-power-of-2 xs m
⟨*proof*⟩

lemma *time-multiply-with-power-of-2-tm-le*:
*time (multiply-with-power-of-2-tm xs m) ≤ 24 + 26 * max m (length xs)*
⟨*proof*⟩

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

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

lemma *val-divide-by-power-of-2-tm*[*simp*, *val-simp*]:
val (*divide-by-power-of-2-tm xs m*) = *divide-by-power-of-2 xs m*
⟨*proof*⟩

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

definition *add-fermat-tm* :: *nat-lsbf* ⇒ *nat-lsbf* ⇒ *nat-lsbf tm* **where**
add-fermat-tm xs ys =1 do {
zs ← *xs* +$_{nt}$ *ys*;
lenzs ← *length-tm zs*;
k1 ← *k* +$_t$ *1*;
powk ← *2* ^$_t$ *k1*;
powk1 ← *powk* +$_t$ *1*;
b ← *lenzs* =$_t$ *powk1*;
if b then do {
zsr ← *butlast-tm zs*;
inc-nat-tm zsr
} *else return zs*
}

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

lemma *time-add-fermat-tm-le*: *time* (*add-fermat-tm xs ys*) ≤ *13* + *7* ∗ *max* (*length xs*) (*length ys*) + *28* ∗ *2* ^ *k*
⟨*proof*⟩

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

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

lemma *time-subtract-fermat-tm-le*: *time* (*subtract-fermat-tm xs ys*) ≤
38 + *66* ∗ *2* ^ *k* + *26* ∗ *length ys* + *7* ∗ *max* (*length xs*) (*length ys*)
⟨*proof*⟩

definition *reduce-tm* :: *nat-lsbf* ⇒ *nat-lsbf tm* **where**
reduce-tm xs =1 do {

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

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

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

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

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

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

abbreviation e :: nat where e ≡ 2 ^ (k + 1)
lemma e-ge-1: e ≥ 1 ⟨proof⟩
lemma e-ge-2: e ≥ 2 ⟨proof⟩
lemma e-ge-4: k > 0 ⟹ e ≥ 4 ⟨proof⟩

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

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

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

function *time-from-nat-lsbf-tm-bound* **where**
$time\text{-}from\text{-}nat\text{-}lsbf\text{-}tm\text{-}bound\ l = 88 * e + 4 * l + 48 +$
$(if\ l \leq 2 * e\ then\ 0\ else\ time\text{-}from\text{-}nat\text{-}lsbf\text{-}tm\text{-}bound\ (l - (e - 1)))$
⟨*proof*⟩
termination
⟨*proof*⟩
declare *time-from-nat-lsbf-tm-bound.simps*[*simp del*]

lemma *time-from-nat-lsbf-tm-le-bound*:
assumes $length\ xs \leq l$
shows $time\ (from\text{-}nat\text{-}lsbf\text{-}tm\ xs) \leq time\text{-}from\text{-}nat\text{-}lsbf\text{-}tm\text{-}bound\ l$
⟨*proof*⟩

lemma *time-from-nat-lsbf-tm-bound-closed*:
assumes $x \leq 2 * e$
assumes $x \geq e + 2$
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*⟩

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

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 ← $2\ \widehat{}_t\ k1$;
s-new-mod ← $s\text{-}new\ mod_t\ powk1$;
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 (λ*x y*. *val* (*f x y*)) (λ*x y*. *val* (*g x y*)) *l* (*revs*, *s*) *bs cs*
⟨*proof*⟩

lemma *time-fft-combine-b-c-aux-tm-le*:
assumes *length bs = length cs*
assumes ⋀*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*)
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *time-fft-combine-b-c-add-tm-le*:
assumes *length bs = length cs*
assumes ⋀*b*. *b* ∈ *set bs* ⟹ *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*) $\leq 4 + length\ bs * (72 + 116 * e + 8 * l)$
⟨*proof*⟩

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*) $\leq 4 + length\ bs * (72 + 116 * e + 8 * l)$
⟨*proof*⟩

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*⟩

lemma *time-fft-ifft-tm-le-aux*:
assumes $\bigwedge x.\ x \in set\ a \Longrightarrow length\ x = e$
assumes *length a* = 2 ^ *m*
shows *time* (*fft-ifft-tm it l a*) $\leq 2\ \hat{}\ (m - 1) * (52 + 87 * e) + (m - 1) * 2\ \hat{}\ m * (76 + 116 * e) + (\sum i \leftarrow [0..<m-1].\ 2\ \hat{}\ i) * (8 * 2\ \hat{}\ m * l + 13)$
⟨*proof*⟩

lemma *time-fft-ifft-tm-le*:
assumes $\bigwedge x.\ x \in set\ a \Longrightarrow length\ x = e$
assumes *length a* = 2 ^ *m*
shows *time* (*fft-ifft-tm it l a*) $\leq 2\ \hat{}\ m * (65 + 87 * e) + m * 2\ \hat{}\ m * (76 + 116 * e) + (8 * l) * 2\ \hat{}\ (2 * m)$
⟨*proof*⟩

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 ^ m ⟹ val (fft-tm l a) = fft l a*
⟨*proof*⟩
lemma *val-ifft-tm*[*simp*, *val-simp*]: *length a = 2 ^ m ⟹ val (ifft-tm l a) = ifft l a*
⟨*proof*⟩

lemma *time-fft-tm-le*:
assumes ⋀*x*. *x ∈ set a ⟹ length x = e*
assumes *length a = 2 ^ m*
shows *time (fft-tm l a) ≤ 2 ^ m * (66 + 87 * e) + m * 2 ^ m * (76 + 116 * e) + (8 * l) * 2 ^ (2 * m)*
⟨*proof*⟩

lemma *time-ifft-tm-le*:
assumes ⋀*x*. *x ∈ set a ⟹ length x = e*
assumes *length a = 2 ^ m*
shows *time (ifft-tm l a) ≤ 2 ^ m * (66 + 87 * e) + m * 2 ^ m * (76 + 116 * e) + (8 * l) * 2 ^ (2 * m)*
⟨*proof*⟩

end

end

3.4 Final Preparations

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

lemma *aux-ineq-1*: *n > 1 ⟹ 2 ^ (2 * n − 1) > n + 1 + 2 ^ n*

⟨*proof*⟩

lemma *aux-ineq-2*: $n > 2 \Longrightarrow 2 \hat{} (2 * n - 2) > n + 2 \hat{} n$
⟨*proof*⟩
lemma *aux-ineq-3*: $n > 1 \Longrightarrow 2 \hat{} n \geq n + 2$
⟨*proof*⟩

lemma (**in** *residues*) *nat-embedding-eq*: *ring.nat-embedding R x* = *int x mod m*
⟨*proof*⟩

lemma (**in** *residues*) *carrier-mod-eq*: $x \in$ *carrier R* $\Longrightarrow x$ *mod m* = *x*
⟨*proof*⟩

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

locale *m-lemmas* =
fixes *m* :: *nat*
assumes *m-ge-3*: ¬ *m* < *3*
begin

Lemmas in *nat* resp. *int*.

lemma *m-gt-0*: *m* > *0* ⟨*proof*⟩

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

definition *oe-n* :: *nat* **where**
oe-n ≡ (*if odd m then n* + *1 else n*)

lemma *n-gt-1*: *n* > *1* ⟨*proof*⟩
lemma *n-ge-2*: *n* ≥ *2* ⟨*proof*⟩
lemma *n-gt-0*: *n* > *0* ⟨*proof*⟩
lemma *even-m-imp-n-ge-3*: *even m* ⟹ *n* ≥ *3* ⟨*proof*⟩

lemma *n-lt-m*: *n* < *m* ⟨*proof*⟩

lemma *oe-n-gt-1*: *oe-n* > *1* ⟨*proof*⟩
lemma *oe-n-gt-0*: *oe-n* > *0* ⟨*proof*⟩

lemma *oe-n-le-n*: *oe-n* ≤ *n* + *1* ⟨*proof*⟩
lemma *oe-n-minus-1-le-n*: *oe-n* − *1* ≤ *n* ⟨*proof*⟩

lemma *two-pow-oe-n-div-2*: (*2*::*nat*) ^ *oe-n div 2* = *2* ^ (*oe-n* − *1*)
⟨*proof*⟩
lemma *two-pow-oe-n-as-halves*: (*2*::*nat*) ^ *oe-n* = *2* ^ (*oe-n* − *1*) + *2* ^ (*oe-n* − *1*)
⟨*proof*⟩
lemma *two-pow-Suc-oe-n-as-prod*: (*2*::*nat*) ^ (*oe-n* + *1*) = *4* ∗ *2* ^ (*oe-n* − *1*)

⟨*proof*⟩

lemma *index-intros*:
fixes i :: *nat*
assumes $i < 2 \hat{} (oe\text{-}n - 1)$
shows $i < 2 \hat{} oe\text{-}n\ 2 \hat{} (oe\text{-}n - 1) + i < 2 \hat{} oe\text{-}n$
⟨*proof*⟩

lemma *index-decomp*:
assumes $j < (2::nat) \hat{} (oe\text{-}n + 1)$
shows
$j\ div\ 2 \hat{} (oe\text{-}n - 1) < 4$
$j\ mod\ 2 \hat{} (oe\text{-}n - 1) < 2 \hat{} (oe\text{-}n - 1)$
$j = (j\ div\ 2 \hat{} (oe\text{-}n - 1)) * 2 \hat{} (oe\text{-}n - 1) + (j\ mod\ 2 \hat{} (oe\text{-}n - 1))$
⟨*proof*⟩
lemma *index-comp*:
fixes $i\ j$ :: *nat*
assumes $i < 4\ j < 2 \hat{} (oe\text{-}n - 1)$
shows
$i * 2 \hat{} (oe\text{-}n - 1) + j < 2 \hat{} (oe\text{-}n + 1)$
$(i * 2 \hat{} (oe\text{-}n - 1) + j)\ div\ 2 \hat{} (oe\text{-}n - 1) = i$
$(i * 2 \hat{} (oe\text{-}n - 1) + j)\ mod\ 2 \hat{} (oe\text{-}n - 1) = j$
⟨*proof*⟩

lemma *mn*:
$odd\ m \Longrightarrow m = 2 * n - 1$
$even\ m \Longrightarrow m = 2 * n - 2$
⟨*proof*⟩

lemma *m0*: $m = (n - 1) + (oe\text{-}n - 1)$
⟨*proof*⟩
lemma *m1*: $m + 1 = (n - 1) + oe\text{-}n$
⟨*proof*⟩

lemma *two-pow-m1-as-prod*: $(2::nat) \hat{} (m + 1) = 2 \hat{} (n - 1) * 2 \hat{} oe\text{-}n$
⟨*proof*⟩
lemma *two-pow-m0-as-prod*: $(2::nat) \hat{} m = 2 \hat{} (n - 1) * 2 \hat{} (oe\text{-}n - 1)$
⟨*proof*⟩

lemma *two-pow-two-n-le*: $(2::nat) \hat{} (2 * n) \leq 2 * 2 \hat{} (m + 1)$
⟨*proof*⟩

lemma *oe-n-m-bound-0*: $oe\text{-}n + 2 \hat{} n < 2 \hat{} m$
⟨*proof*⟩
lemma *oe-n-m-bound-1*: $oe\text{-}n + 1 + 2 \hat{} n \leq 2 \hat{} m$
⟨*proof*⟩
lemma *two-pow-oe-n-m-bound-1*: $(2::'a::linordered\text{-}semidom) \hat{} (oe\text{-}n + 1 + 2 \hat{} n) \leq 2 \hat{} 2 \hat{} m$
⟨*proof*⟩

lemma *two-pow-oe-n-m-bound-0-int*: $2 \hat{} (oe\text{-}n + 2 \hat{} n) < int\text{-}lsbf\text{-}fermat.n\ m$
⟨*proof*⟩
lemma *two-pow-oe-n-m-bound-1-int*: $2 \hat{} (oe\text{-}n + 1 + 2 \hat{} n) < int\text{-}lsbf\text{-}fermat.n\ m$
⟨*proof*⟩

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

definition *pad-length* **where** $pad\text{-}length = 3 * n + 5$

Lemmas using residue rings.

definition *Zn* **where** $Zn = residue\text{-}ring\ (int\text{-}lsbf\text{-}mod.n\ (n + 2))$
definition *Fn* **where** $Fn = residue\text{-}ring\ (int\text{-}lsbf\text{-}fermat.n\ n)$
definition *Fm* **where** $Fm = residue\text{-}ring\ (int\text{-}lsbf\text{-}fermat.n\ m)$

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

sublocale *Znr* : $int\text{-}lsbf\text{-}mod\ n + 2$
rewrites $Znr.Zn \equiv Zn$
⟨*proof*⟩

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

sublocale *Fnr* : $int\text{-}lsbf\text{-}fermat\ n$
rewrites $Fnr.Fn \equiv Fn$
⟨*proof*⟩

sublocale *Fnr-M* : $multiplicative\text{-}subgroup\ Fn\ Units\ Fn\ units\text{-}of\ Fn$
⟨*proof*⟩

sublocale *Fmr* : $int\text{-}lsbf\text{-}fermat\ m$
rewrites $Fmr.Fn \equiv Fm$
⟨*proof*⟩

sublocale *Fmr-M* : $multiplicative\text{-}subgroup\ Fm\ Units\ Fm\ units\text{-}of\ Fm$
⟨*proof*⟩

lemma *two-pow-oe-n-primitive-root-Fm*:
$Fmr.primitive\text{-}root\ (2 \hat{} oe\text{-}n)\ (2\ \lceil\rceil_{Fm}\ (2{::}nat)\ \hat{}\ (n - 1))$
⟨*proof*⟩

lemma *two-pow-oe-n-root-of-unity-Fm*:
$Fmr.root\text{-}of\text{-}unity\ (2 \hat{} oe\text{-}n)\ (2\ \lceil\rceil_{Fm}\ (2{::}nat)\ \hat{}\ (n - 1))$
⟨*proof*⟩

lemma *four-prim-root-Fn*: $Fnr.primitive\text{-}root\ (2 \hat{} n)\ (2\ \lceil\rceil_{Fn}\ (2{::}nat))$
⟨*proof*⟩

lemma *two-oe-n*: $2\ \lceil\rceil_{Fn}\ oe\text{-}n = 2 \hat{} oe\text{-}n$
⟨*proof*⟩

lemma *two-oe-n-Units-Fn*: $2 \mathbin{\hat{}} \textit{oe-n} \in \textit{Units Fn}$
⟨*proof*⟩
lemma *two-oe-n-carrier-Fn*: $2 \mathbin{\hat{}} \textit{oe-n} \in \textit{carrier Fn}$
⟨*proof*⟩

definition *prim-root-exponent* :: *nat* **where** *prim-root-exponent* = (*if odd m then 1 else 2*)
definition μ **where** $\mu = 2\ [\hat{}]_{Fn}$ *prim-root-exponent*

lemma μ-*Units-Fn*: $\mu \in \textit{Units Fn}$
⟨*proof*⟩
lemma μ-*carrier-Fn*: $\mu \in \textit{carrier Fn}$
⟨*proof*⟩

lemma μ-*prim-root*: *Fnr.primitive-root* $(2 \mathbin{\hat{}} \textit{oe-n})\ \mu$
⟨*proof*⟩
lemma μ-*root-of-unity*: *Fnr.root-of-unity* $(2 \mathbin{\hat{}} \textit{oe-n})\ \mu$
⟨*proof*⟩
lemma μ-*halfway-property*: $\mu\ [\hat{}]_{Fn}\ ((2{::}nat) \mathbin{\hat{}} \textit{oe-n div } 2) = \ominus_{Fn}\ \mathbf{1}_{Fn}$
⟨*proof*⟩

end

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

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

definition *num-blocks* **where** *num-blocks* = *subdivide* $(2 \mathbin{\hat{}} (n - 1))$ *num*
definition *num-blocks-carrier* **where** *num-blocks-carrier* = *map* (*fill* $(2 \mathbin{\hat{}} (n + 1))$) *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* $= 2 \mathbin{\hat{}} (m + 1)$
⟨*proof*⟩

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

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

shows *length* (*num-blocks* ! *i*) = 2 ^ (*n* − 1)
⟨*proof*⟩

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)
⟨*proof*⟩
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*
⟨*proof*⟩

lemma *length-num-blocks-carrier*[*simp*]: *length num-blocks-carrier* = 2 ^ *oe-n*
⟨*proof*⟩

lemma *to-res-num*: *Fmr.to-residue-ring num* = (⨁$_{Fm}$*j* ← [*0*..<*2* ^ *oe-n*].
map (*int* ∘ *Nat-LSBF.to-nat*) *num-blocks* ! *j* ⊗$_{Fm}$ ((*2* [^]$_{Fm}$ ((*2*::*nat*) ^ (*n* − *1*))) [^]$_{Fm}$ *j*))
⟨*proof*⟩

lemma *length-num-Zn*[*simp*]: *length num-Zn* = 2 ^ *oe-n*
⟨*proof*⟩
lemma *length-nth-num-Zn*[*simp*]:
fixes *i* :: *nat*
assumes *i* < 2 ^ *oe-n*
shows *length* (*num-Zn* ! *i*) = *n* + *2*
⟨*proof*⟩

lemma *length-num-Zn-pad*[*simp*]: *length num-Zn-pad* = *pad-length* ∗ 2 ^ *oe-n*
⟨*proof*⟩

lemma *to-nat-num-Zn-pad*:
Nat-LSBF.to-nat num-Zn-pad = (∑ *i* ← [*0*..<*2* ^ *oe-n*]. *Nat-LSBF.to-nat* (*num-Zn* ! *i*) ∗ 2 ^ (*i* ∗ *pad-length*))
⟨*proof*⟩

lemma *length-num-dft*[*simp*]: *length num-dft* = 2 ^ *oe-n*
⟨*proof*⟩

lemma *fill-num-blocks-carrier*[*simp*]: *set num-blocks-carrier* ⊆ *Fnr.fermat-non-unique-carrier*
⟨*proof*⟩

lemma *num-dft-carrier*[*simp*]: *set num-dft* ⊆ *Fnr.fermat-non-unique-carrier*
⟨*proof*⟩

lemma *to-res-num-dft*:
map Fnr.to-residue-ring num-dft = *Fnr.NTT* μ (*map Fnr.to-residue-ring num-blocks-carrier*)

⟨*proof*⟩

lemma *length-num-dft-odds*[*simp*]: *length num-dft-odds* = *2* ^ (*oe-n* − *1*)
⟨*proof*⟩
lemma *num-dft-odds-carrier*[*simp*]: *set num-dft-odds* ⊆ *Fnr.fermat-non-unique-carrier*
⟨*proof*⟩

end

3.4.1 A special residue problem

definition *solve-special-residue-problem* **where**
solve-special-residue-problem n ξ η =
(*let δ* = *int-lsbf-mod.subtract-mod* (*n* + *2*) *η* (*take* (*n* + *2*) *ξ*) *in*
add-nat ξ (*add-nat* (*δ* >>$_n$ (*2* ^ *n*)) *δ*))

lemma *two-pow-n-geq-n-plus-2*: *n* ≥ *2* ⟹ *2* ^ *n* ≥ *n* + *2*
⟨*proof*⟩

lemma *fermat-mod-pow-2-aux*: *n* ≥ *2* ⟹ (*2*::*nat*) ^ (*2* ^ *n*) *mod* *2* ^ (*n* + *2*) = *0*
⟨*proof*⟩

definition *solves-special-residue-problem* **where**
solves-special-residue-problem z n ξ η ≡
z < *2* ^ (*n* + *2*) ∗ *int-lsbf-fermat.n n*
∧ *z mod int-lsbf-fermat.n n* = *ξ*
∧ *z mod* (*2* ^ (*n* + *2*)) = *η*

lemma *solve-special-residue-problem-correct*:
fixes *n* :: *nat*
fixes *ξ η* :: *nat-lsbf*
assumes *n* ≥ *2*
assumes *length η* ≤ *n* + *2*
assumes *Nat-LSBF.to-nat ξ* < *int-lsbf-fermat.n n*
assumes *z* = *solve-special-residue-problem n ξ η*
shows *solves-special-residue-problem* (*Nat-LSBF.to-nat z*) *n* (*Nat-LSBF.to-nat ξ*) (*Nat-LSBF.to-nat η*)
⟨*proof*⟩

lemma *fn-zn-coprime*: *coprime* (*int-lsbf-fermat.n n*) (*2* ^ (*n* + *2*))
⟨*proof*⟩

lemma *int-ideal-add*: $Idl_{\mathcal{Z}}$ {*m*} <+>$_{\mathcal{Z}}$ $Idl_{\mathcal{Z}}$ {*n*} = $Idl_{\mathcal{Z}}$ {*gcd m n*}
⟨*proof*⟩

lemma *int-ideal-inter*: $Idl_{\mathcal{Z}}$ {*m*} ∩ $Idl_{\mathcal{Z}}$ {*n*} = $Idl_{\mathcal{Z}}$ {*lcm m n*}
⟨*proof*⟩

corollary *coprime m n* $\Longrightarrow$ $Idl_{\mathcal{Z}}\ \{m\} <+>_{\mathcal{Z}} Idl_{\mathcal{Z}}\ \{n\} = carrier\ \mathcal{Z}$
⟨*proof*⟩

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

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

corollary *coprime m n* $\Longrightarrow$ $Idl_{\mathcal{Z}}\ \{m\} \cap Idl_{\mathcal{Z}}\ \{n\} = Idl_{\mathcal{Z}}\ \{m * n\}$
⟨*proof*⟩

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

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*⟩

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)*
⟨*proof*⟩

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

3.4.2 Subroutine for combining the final result

fun *combine-z-aux* **where**
combine-z-aux l acc [] = *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 $\bigwedge z.\ z \in set\ zs \Longrightarrow length\ z \geq l$
shows *Nat-LSBF.to-nat (combine-z-aux l acc zs) = Nat-LSBF.to-nat (concat (rev acc)) +*
*2 ^ (length (concat acc)) * (∑ i ← [0..<length zs]. Nat-LSBF.to-nat (zs ! i) * 2 ^ (i * l))*
⟨*proof*⟩

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) = (∑ i ← [0..<length zs]. Nat-LSBF.to-nat (zs ! i) * 2 ^ (i * l))*
⟨*proof*⟩

lemma *length-combine-z-aux-le*:
assumes $\bigwedge z.\ z \in set\ zs \Longrightarrow length\ z \leq lz$
assumes *length z ≤ lz + 1*
assumes *l > 0*
shows *length (combine-z-aux l acc (z # zs)) ≤ (lz + 1) * (length zs + 1) + length (concat acc)*
⟨*proof*⟩

lemma *length-combine-z-le*:
assumes $\bigwedge z.\ z \in set\ zs \Longrightarrow length\ z \leq lz$
assumes *l > 0*
shows *length (combine-z l zs) ≤ (lz + 1) * length zs*
⟨*proof*⟩

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)

 ⟨proof⟩

termination
 ⟨proof⟩

declare schoenhage-strassen.simps[simp del]

locale schoenhage-strassen-context =
 fixes m :: nat
 fixes a :: nat-lsbf
 fixes b :: nat-lsbf
 assumes m-ge-3: ¬ m < 3
 assumes a-carrier: a ∈ int-lsbf-fermat.fermat-non-unique-carrier m
 assumes b-carrier: b ∈ int-lsbf-fermat.fermat-non-unique-carrier m
begin
```

sublocale *m-lemmas*
⟨*proof*⟩

sublocale *A*: *carrier-input m a*
⟨*proof*⟩

sublocale *B*: *carrier-input m b*
⟨*proof*⟩

definition *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*
⟨*proof*⟩

lemma *length-uv*: *length uv* = *uv-length*
⟨*proof*⟩

lemma *pad-length-gt-0*: *pad-length* > *0* ⟨*proof*⟩

lemma *scuv*:
length (*subdivide pad-length uv*) = 2 ^ (*oe-n* + 1)
x ∈ *set* (*subdivide pad-length uv*) ⟹ *length x* = *pad-length*
⟨*proof*⟩

lemma *length-c-dft-odds*: *length c-dft-odds* = 2 ^ (*oe-n* − 1)
⟨*proof*⟩
lemma *length-c-diffs*: *length c-diffs* = 2 ^ (*oe-n* − 1)
⟨*proof*⟩
lemma *length-ξ′*: *length ξ′* = 2 ^ (*oe-n* − 1)
⟨*proof*⟩
lemma *length-ξ*: *length ξ* = 2 ^ (*oe-n* − 1)
⟨*proof*⟩

lemma *γ-nth*: *i* < 4 ⟹ *j* < 2 ^ (*oe-n* − 1) ⟹ *γ* ! *i* ! *j* = (*subdivide pad-length uv*) ! (*i* ∗ 2 ^ (*oe-n* − 1) + *j*) **for** *i j*
⟨*proof*⟩
lemma *γ-nth′*: ⋀*j*. *j* < 2 ^ (*oe-n* + 1) ⟹ *γ* ! (*j div* 2 ^ (*oe-n* − 1)) ! (*j mod* 2 ^ (*oe-n* − 1)) = *subdivide pad-length uv* ! *j*
⟨*proof*⟩
lemma *scγ*: *length γ* = 4 ⋀*i*. *i* < 4 ⟹ *length* (*γ* ! *i*) = 2 ^ (*oe-n* − 1)
⟨*proof*⟩
lemmas *length-γ* = *scγ*(1)
lemmas *length-γ-i* = *scγ*(2)

lemma *length-γ-nth*: *i* < 4 ⟹ *j* < 2 ^ (*oe-n* − 1) ⟹ *length* (*γ* ! *i* ! *j*) = *pad-length*
⟨*proof*⟩

lemma *length-η*: *length η* = 2 ^ (*oe-n* − 1) ⟨*proof*⟩
lemma *length-z*: *length z* = 2 ^ (*oe-n* − 1)
⟨*proof*⟩
lemma *nth-z*: *z* ! *j* = *solve-special-residue-problem n* (*ξ* ! *j*) (*η* ! *j*) **if** *j* < 2 ^ (*oe-n* − 1) **for** *j*
⟨*proof*⟩
lemma *length-z-filled*: *length z-filled* = 2 ^ (*oe-n* − 1)
⟨*proof*⟩
lemma *length-z-consts*: *length z-consts* = 2 ^ (*oe-n* − 1)
⟨*proof*⟩

end

theorem *schoenhage-strassen-correct′*:
assumes *a* ∈ *int-lsbf-fermat.fermat-non-unique-carrier m*
assumes *b* ∈ *int-lsbf-fermat.fermat-non-unique-carrier m*
shows *int-lsbf-fermat.to-residue-ring m* (*schoenhage-strassen m a b*)
= *int-lsbf-fermat.to-residue-ring m a* ⊗*int-lsbf-fermat.Fn m int-lsbf-fermat.to-residue-ring m b* ∧ *schoenhage-strassen m a b* ∈ *int-lsbf-fermat.fermat-non-unique-carrier m*
⟨*proof*⟩

3.6 Schoenhage-Strassen Multiplication in $\mathbb{N}$

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

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

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*)
⟨*proof*⟩

lemma
shows *length-a'*: *length a* ≤ *2* ^ (*m* + *1*)
and *length-b'*: *length b* ≤ *2* ^ (*m* + *1*)
⟨*proof*⟩

lemma *length-fill-a*: *length fill-a* = *2* ^ (*m* + *1*)
⟨*proof*⟩

lemma *length-fill-b*: *length fill-b* = *2* ^ (*m* + *1*)
⟨*proof*⟩

sublocale *fm*: *int-lsbf-fermat m* ⟨*proof*⟩

definition *Fm* **where** *Fm* = *residue-ring* (*int-lsbf-fermat.n m*)
sublocale *Fmr*: *residues fm.n Fm*
rewrites *fm-Fm*: *fm.Fn* ≡ *Fm*

⟨*proof*⟩

lemma *fill-a-carrier*[*simp*, *intro*]: *fill-a* ∈ *fm.fermat-non-unique-carrier*
⟨*proof*⟩
lemma *fill-b-carrier*[*simp*, *intro*]: *fill-b* ∈ *fm.fermat-non-unique-carrier*
⟨*proof*⟩

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

lemma *ssc'*: *fm.to-residue-ring fm-result* = *fm.to-residue-ring fill-a* $\otimes_{Fm}$ *fm.to-residue-ring fill-b*
and *fm-result* ∈ *int-lsbf-fermat.fermat-non-unique-carrier m*
⟨*proof*⟩

end

theorem *schoenhage-strassen-mul-correct*: *Nat-LSBF.to-nat* (*schoenhage-strassen-mul a b*) = *Nat-LSBF.to-nat a* ∗ *Nat-LSBF.to-nat b*
⟨*proof*⟩

end

4 Running Time Formalization

theory *Schoenhage-Strassen-TM*
imports
Schoenhage-Strassen
../Preliminaries/Schoenhage-Strassen-Preliminaries
Z-mod-Fermat-TM
Karatsuba.Karatsuba-TM
Landau-Symbols.Landau-More
begin

definition *solve-special-residue-problem-tm* **where**
solve-special-residue-problem-tm n ξ η =1 do {
n2 ← *n* +$_t$ *2*;
ξmod ← *take-tm n2 ξ*;
δ ← *int-lsbf-mod.subtract-mod-tm n2 η ξmod*;
pown ← *2* ^$_t$ *n*;
δ-shifted ← *δ* >>$_{nt}$ *pown*;
δ1 ← *δ-shifted* +$_{nt}$ *δ*;
ξ +$_{nt}$ *δ1*
}

lemma *val-solve-special-residue-problem-tm*[*simp*, *val-simp*]:
val (*solve-special-residue-problem-tm n ξ η*) = *solve-special-residue-problem n ξ η*
⟨*proof*⟩

lemma *time-solve-special-residue-problem-tm-le*:
time (*solve-special-residue-problem-tm n ξ η*) ≤ *245* + *74* ∗ *2* ^ *n* + *55* ∗ *length η* + *2* ∗ *length ξ*
⟨*proof*⟩

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

lemma *val-combine-z-aux-tm*[*simp*, *val-simp*]: *val* (*combine-z-aux-tm l acc zs*) = *combine-z-aux l acc zs*
⟨*proof*⟩

lemma *time-combine-z-aux-tm-le*:
assumes ⋀*z*. *z* ∈ *set zs* ⟹ *length z* ≤ *lz*
assumes *length z* ≤ *lz* + *1*
assumes *l* > *0*
shows *time* (*combine-z-aux-tm l acc* (*z* # *zs*)) ≤ (*2* ∗ *l* + *2* ∗ *lz* + *7*) ∗ *length zs* + *3* ∗ (*length acc* + *length zs*) + *length* (*concat acc*) + *length zs* ∗ *l* + *lz* + *9*
⟨*proof*⟩

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

lemma *val-combine-z-tm*[*simp*, *val-simp*]: *val* (*combine-z-tm l zs*) = *combine-z l zs*
⟨*proof*⟩

lemma *time-combine-z-tm-le*:
assumes ⋀*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*⟩

lemma *schoenhage-strassen-tm-termination-aux*: ¬ *m* < *3* ⟹ *Suc* (*m div 2*) < *m*
⟨*proof*⟩

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 ^_t n-minus-1;
a′ ← subdivide-tm segment-lens a;
b′ ← subdivide-tm segment-lens b;
α ← map-tm (int-lsbf-mod.reduce-tm n-plus-2) a′;
three-n ← 3 *_t n;
pad-length ← three-n +_t 5;
α-padded ← map-tm (fill-tm pad-length) α;
u ← concat-tm α-padded;
β ← map-tm (int-lsbf-mod.reduce-tm n-plus-2) b′;
β-padded ← map-tm (fill-tm pad-length) β;
v ← concat-tm β-padded;
oe-n-plus-1 ← oe-n +_t 1;
two-pow-oe-n-plus-1 ← 2 ^_t oe-n-plus-1;
uv-length ← pad-length *_t two-pow-oe-n-plus-1;
uv-unpadded ← karatsuba-mul-nat-tm u v;
uv ← ensure-length-tm uv-length uv-unpadded;
oe-n-minus-1 ← oe-n −_t 1;
two-pow-oe-n-minus-1 ← 2 ^_t oe-n-minus-1;
γs ← subdivide-tm pad-length uv;
γ ← subdivide-tm two-pow-oe-n-minus-1 γs;
γ0 ← nth-tm γ 0;
γ1 ← nth-tm γ 1;
γ2 ← nth-tm γ 2;
γ3 ← nth-tm γ 3;
η ← map4-tm
  (λx y z w. do {
    xmod ← take-tm n-plus-2 x;
    ymod ← take-tm n-plus-2 y;
    zmod ← take-tm n-plus-2 z;
    wmod ← take-tm n-plus-2 w;
    xy ← int-lsbf-mod.subtract-mod-tm n-plus-2 xmod ymod;
    zw ← int-lsbf-mod.subtract-mod-tm n-plus-2 zmod wmod;
    int-lsbf-mod.add-mod-tm n-plus-2 xy zw
  })
  γ0 γ1 γ2 γ3;
prim-root-exponent ← if odd-m then return 1 else return 2;
fn-carrier-len ← 2 ^_t n-plus-1;
a′-carrier ← map-tm (fill-tm fn-carrier-len) a′;
```

```
    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
  }
}
  ⟨proof⟩
termination
  ⟨proof⟩

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

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

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

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

lemma *val-nth-γ*[*simp*, *val-simp*]:
val (nth-tm γ 0) = γ ! 0
val (nth-tm γ 1) = γ ! 1
val (nth-tm γ 2) = γ ! 2
val (nth-tm γ 3) = γ ! 3
⟨*proof*⟩

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

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
⟨*proof*⟩

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*
⟨*proof*⟩

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 $a \in$ *int-lsbf-fermat.fermat-non-unique-carrier m*
assumes $b \in$ *int-lsbf-fermat.fermat-non-unique-carrier m*
shows *time* (*schoenhage-strassen-tm m a b*) ≤ *schoenhage-strassen-Fm-bound m*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *powr-unbounded*:
assumes (*c* :: *real*) > *0*
shows *eventually* (λ*x. d* ≤ *x powr c*) *at-top*
⟨*proof*⟩

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*⟩

lemma *poly-smallo-exp*:
assumes *c* > *1*
shows (λ*n.* (*real n*) *powr d*) ∈ *o*(λ*n. c powr* (*real n*))
⟨*proof*⟩

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

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

lemma *kar-aux-lem-le*:
assumes $n > 0$
shows *real* $(n * 2\ \widehat{}\ n)\ powr\ log\ 2\ 3 \leq kar\text{-}aux\text{-}const * real\ (2\ \widehat{}\ (2 * n))$
⟨*proof*⟩

lemma *kar-aux-const-gt-0*: *kar-aux-const* > 0
⟨*proof*⟩

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

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* $\Rightarrow$ *nat* **where**
$m < 3 \Longrightarrow$ *schoenhage-strassen-Fm-bound'* $m = 5336$
$\mid m \geq 3 \Longrightarrow$ *schoenhage-strassen-Fm-bound'* $m =$ *s-const1* $* (m * 2\ \widehat{}\ m) +$ *s-const2* $+$ *schoenhage-strassen-Fm-bound'* $((m + 2)\ div\ 2) * 2\ \widehat{}\ ((m + 1)\ div\ 2)$
⟨*proof*⟩
termination
⟨*proof*⟩

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

lemma *schoenhage-strassen-Fm-bound-le-schoenhage-strassen-Fm-bound'*:
shows *schoenhage-strassen-Fm-bound* $m \leq$ *schoenhage-strassen-Fm-bound'* m
⟨*proof*⟩

definition γ-*0* **where** γ-*0* $= 2 *$ *s-const1* $+$ *s-const2*

lemma *schoenhage-strassen-Fm-bound'-oe-rec*:
assumes $n \geq 3$
shows *schoenhage-strassen-Fm-bound'* $(2 * n - 2) \leq \gamma\text{-}0 * n * 2\ \widehat{}\ (2 * n - 2) +$ *schoenhage-strassen-Fm-bound'* $n * 2\ \widehat{}\ (n - 1)$
and *schoenhage-strassen-Fm-bound'* $(2 * n - 1) \leq \gamma\text{-}0 * n * 2\ \widehat{}\ (2 * n - 1) +$ *schoenhage-strassen-Fm-bound'* $n * 2\ \widehat{}\ n$
⟨*proof*⟩

definition γ **where** $\gamma = Max\ \{\gamma\text{-}0,$ *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 $m \leq 2 \mathbin{\hat{}} Suc\ k + 1$
shows *schoenhage-strassen-Fm-bound'* $m \leq \gamma * Suc\ k * 2 \mathbin{\hat{}} (Suc\ k + m)$
⟨*proof*⟩

lemma *schoenhage-strassen-Fm-bound'-le-aux2*:
assumes $k \geq 1$
assumes $m \leq 2 \mathbin{\hat{}} k + 1$
shows *schoenhage-strassen-Fm-bound'* $m \leq \gamma * k * 2 \mathbin{\hat{}} (k + m)$
⟨*proof*⟩

4.1 Multiplication in $\mathbb{N}$

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 ← $2 \mathbin{\hat{}}_t$ *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*⟩

lemma *real-power*: $a > 0 \Longrightarrow real\ ((a :: nat) \mathbin{\hat{}} x) = real\ a\ powr\ real\ x$
⟨*proof*⟩

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

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*⟩

lemma *real-diff*: $a \leq b \Longrightarrow real\ (b - a) = real\ b - real\ a$
⟨*proof*⟩

lemma *bitsize-le-log*: $n > 0 \Longrightarrow real\ (bitsize\ n) \leq log\ 2\ (real\ n) + 1$

⟨*proof*⟩

lemma *powr-mono-base2*: $a \leq b \Longrightarrow 2\ powr\ (a :: real) \leq 2\ powr\ b$
⟨*proof*⟩

lemma *log-mono-base2*: $a > 0 \Longrightarrow b > 0 \Longrightarrow a \leq b \Longrightarrow log\ 2\ a \leq log\ 2\ b$
⟨*proof*⟩

lemma *log-nonneg-base2*: $x \geq 1 \Longrightarrow log\ 2\ x \geq 0$
⟨*proof*⟩

lemma *powr-log-cancel-base2*: $x > 0 \Longrightarrow 2\ powr\ (log\ 2\ x) = x$
⟨*proof*⟩

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

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

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

References

[1] T. Ammer and K. Kreuzer. Number theoretic transform. *Archive of Formal Proofs*, August 2022. https://isa-afp.org/entries/Number_Theoretic_Transform.html, Formal proof development.

[2] T. Nipkow. Verified root-balanced trees. In B.-Y. E. Chang, editor, *Asian Symposium on Programming Languages and Systems, APLAS 2017*, volume 10695 of *LNCS*, pages 255–272. Springer, 2017. https://www21.in.tum.de/~nipkow/pubs/aplas17.pdf.

[3] A. Schönhage and V. Strassen. Schnelle multiplikation großer zahlen. *Computing*, 7:281–292, 1971.