

# Fast Fourier Transformation

Clemens Ballarin

12th December 2009

## Contents

<b>1 Preliminaries</b>	<b>1</b>
<b>2 Complex Roots of Unity</b>	<b>3</b>
2.1 Basic Lemmas . . . . .	4
2.2 Derived Lemmas . . . . .	6
<b>3 Discrete Fourier Transformation</b>	<b>7</b>
<b>4 Discrete, Fast Fourier Transformation</b>	<b>12</b>

```
theory FFT
imports Complex_Main
begin
```

We formalise a functional implementation of the FFT algorithm over the complex numbers, and its inverse. Both are shown equivalent to the usual definitions of these operations through Vandermonde matrices. They are also shown to be inverse to each other, more precisely, that composition of the inverse and the transformation yield the identity up to a scalar.

The presentation closely follows Section 30.2 of Cormen *et al.*, *Introduction to Algorithms*, 2nd edition, MIT Press, 2003.

## 1 Preliminaries

```
lemma of_nat_cplx:
  "of_nat n = Complex (of_nat n) 0"
  by (induct n) (simp_all add: complex_one_def)
```

The following two lemmas are useful for experimenting with the transformations, at a vector length of four.

**lemma Iv14:**

```
"{0..<4::nat} = {0, 1, 2, 3}"
```

**proof -**

```
have "{0..<4::nat} = {0..<Suc (Suc (Suc (Suc 0)))}" by (simp add: nat_number)
```

```
also have "... = {0, 1, 2, 3}"
```

```
by (simp add: atLeastLessThanSuc nat_number insert_commute)
```

```
finally show ?thesis .
```

**qed**

**lemma Sum4:**

```
"(∑ i=0..<4::nat. x i) = x 0 + x 1 + x 2 + x 3"
```

```
by (simp add: Iv14 nat_number)
```

A number of specialised lemmas for the summation operator, where the index set is the natural numbers

**lemma setsum\_add\_nat\_ivl\_singleton:**

```
assumes less: "m < (n::nat)"
```

```
shows "f m + setsum f {m<..

```

**proof -**

```
have "f m + setsum f {m<..

```

```
by (simp add: setsum_Un_disjoint ivl_disj_int)
```

```
also from less have "... = setsum f {m..

```

```
by (simp only: ivl_disj_un)
```

```
finally show ?thesis .
```

**qed**

**lemma setsum\_add\_split\_nat\_ivl\_singleton:**

```
assumes less: "m < (n::nat)"
```

```
and g: "!!i. [| m < i; i < n |] ==> g i = f i"
```

```
shows "f m + setsum g {m<..

```

```
using less g
```

```
by(simp add: setsum_add_nat_ivl_singleton cong: strong_setsum_cong)
```

**lemma setsum\_add\_split\_nat\_ivl:**

```
assumes le: "m <= (k::nat)" "k <= n"
```

```
and g: "!!i. [| m <= i; i < k |] ==> g i = f i"
```

```
and h: "!!i. [| k <= i; i < n |] ==> h i = f i"
```

```
shows "setsum g {m..

```

```
using le g h by (simp add: setsum_add_nat_ivl cong: strong_setsum_cong)
```

**lemma ivl\_splice\_Un:**

```

"{0..<2*n::nat} = (op * 2 ' {0..<n}) ∪ ((%i. Suc (2*i)) ' {0..<n})"
apply (unfold image_def Bex_def)
apply auto
apply arith
done

lemma ivl_splice_Int:
  "(op * 2 ' {0..<n}) ∩ ((%i. Suc (2*i)) ' {0..<n}) = {}"
  by auto arith

lemma double_inj_on:
  "inj_on (%i. 2*i::nat) A"
  by (simp add: inj_onI)

lemma Suc_double_inj_on:
  "inj_on (%i. Suc (2*i)) A"
  by (rule inj_onI) simp

lemma setsum_splice:
  "(\sum i::nat = 0..<2*n. f i) = (\sum i = 0..<n. f (2*i)) + (\sum i = 0..<n. f (2*i+1))"
proof -
  have "(\sum i::nat = 0..<2*n. f i) =
    setsum f (op * 2 ' {0..<n}) + setsum f ((%i. 2*i+1) ' {0..<n})"
    by (simp add: ivl_splice_Un ivl_splice_Int setsum_Un_disjoint)
  also have "... = (\sum i = 0..<n. f (2*i)) + (\sum i = 0..<n. f (2*i+1))"
    by (simp add: setsum_reindex [OF double_inj_on]
      setsum_reindex [OF Suc_double_inj_on])
  finally show ?thesis .
qed

```

## 2 Complex Roots of Unity

The function `cis` from the complex library returns the point on the unity circle corresponding to the argument angle. It is the base for our definition of `root`. The main property, De Moirve's formula is already there in the library.

**constdefs**

```

root :: "nat => complex"
"root n == cis (2*pi/(real (n::nat)))"

```

```

lemma sin_periodic_pi_diff [simp]: "sin (x - pi) = - sin x"
  by (simp add: sin_diff)

```

```

lemma sin_cos_between_zero_two_pi:
  assumes 0: "0 < x" and pi: "x < 2 * pi"
  shows "sin x ≠ 0 ∨ cos x ≠ 1"
proof -
  { assume "0 < x" and "x < pi"
    then have "sin x ≠ 0" by (auto dest: sin_gt_zero_pi) }
  moreover
  { assume "x = pi"
    then have "cos x ≠ 1" by simp }
  moreover
  { assume pi1: "pi < x" and pi2: "x < 2 * pi"
    then have "0 < x - pi" and "x - pi < pi" by arith+
    then have "sin (x - pi) ≠ 0" by (auto dest: sin_gt_zero_pi)
    with pi1 pi2 have "sin x ≠ 0" by simp }
  ultimately show ?thesis using 0 pi by arith
qed

```

## 2.1 Basic Lemmas

```

lemma root_nonzero:
  "root n ~≠ 0"
  apply (unfold root_def)
  apply (unfold cis_def)
  apply auto
  apply (drule sin_zero_abs_cos_one)
  apply arith
  done

```

```

lemma root_unity:
  "root n ^ n = 1"
  apply (unfold root_def)
  apply (simp add: DeMoivre)
  apply (simp add: cis_def)
  done

```

```

lemma root_cancel:
  "0 < d ==> root (d * n) ^ (d * k) = root n ^ k"
  apply (unfold root_def)
  apply (simp add: DeMoivre)
  done

```

```

lemma root_summation:
  assumes k: "0 < k" "k < n"

```

```

shows "( $\sum_{i=0..<n}. (\text{root } n \wedge k) \wedge i) = 0"$ 
proof -
from k have real0: "0 < real k * (2 * pi) / real n"
  by (simp add: zero_less_divide_iff
    mult_strict_right_mono [where a = 0, simplified])
from k mult_strict_right_mono [where a = "real k" and
  b = "real n" and c = "2 * pi / real n", simplified]
have realk: "real k * (2 * pi) / real n < 2 * pi"
  by (simp add: zero_less_divide_iff)

```

Main part of the proof

```

have "( $\sum_{i=0..<n}. (\text{root } n \wedge k) \wedge i) =$ 
  (( $\text{root } n \wedge k) \wedge n - 1) / (\text{root } n \wedge k - 1)"$ 
  apply (rule geometric_sum)
  apply (unfold root_def)
  apply (simp add: DeMoivre)
  using real0 realk sin_cos_between_zero_two_pi
  apply (auto simp add: cis_def complex_one_def)
  done
also have "... = (( $\text{root } n \wedge n) \wedge k - 1) / (\text{root } n \wedge k - 1)"$ 
  by (simp add: power_mult [THEN sym] mult_ac)
also have "... = 0"
  by (simp add: root_unity)
finally show ?thesis .

```

qed

lemma root\_summation\_inv:

```

assumes k: "0 < k" "k < n"
shows "( $\sum_{i=0..<n}. ((1 / \text{root } n) \wedge k) \wedge i) = 0"$ 

```

**proof** -

```

from k have real0: "0 < real k * (2 * pi) / real n"
  by (simp add: zero_less_divide_iff
    mult_strict_right_mono [where a = 0, simplified])
from k mult_strict_right_mono [where a = "real k" and
  b = "real n" and c = "2 * pi / real n", simplified]
have realk: "real k * (2 * pi) / real n < 2 * pi"
  by (simp add: zero_less_divide_iff)

```

Main part of the proof

```

have "( $\sum_{i=0..<n}. ((1 / \text{root } n) \wedge k) \wedge i) =$ 
  ((( $1 / \text{root } n) \wedge k) \wedge n - 1) / ((1 / \text{root } n) \wedge k - 1)"$ 
  apply (rule geometric_sum)
  apply (simp add: nonzero_inverse_eq_divide [THEN sym] root_nonzero)
  apply (unfold root_def)

```

```

    apply (simp add: DeMoivre)
    using real0 realk sin_cos_between_zero_two_pi
    apply (auto simp add: cis_def complex_one_def)
    done
  also have "... = (((1 / root n) ^ n) ^ k - 1) / ((1 / root n) ^ k - 1)"
    by (simp add: power_mult [THEN sym] mult_ac)
  also have "... = 0"
    by (simp add: power_divide root_unity)
  finally show ?thesis .
qed

```

```

lemma root0 [simp]:
  "root 0 = 1"
  by (simp add: root_def cis_def)

```

```

lemma root1 [simp]:
  "root 1 = 1"
  by (simp add: root_def cis_def)

```

```

lemma root2 [simp]:
  "root 2 = Complex -1 0"
  by (simp add: root_def cis_def)

```

```

lemma root4 [simp]:
  "root 4 = ii"
  by (simp add: root_def cis_def)

```

## 2.2 Derived Lemmas

```

lemma root_cancel1:
  "root (2 * m) ^ (i * (2 * j)) = root m ^ (i * j)"
proof -
  have "root (2 * m) ^ (i * (2 * j)) = root (2 * m) ^ (2 * (i * j))"
    by (simp add: mult_ac)
  also have "... = root m ^ (i * j)"
    by (simp add: root_cancel)
  finally show ?thesis .
qed

```

```

lemma root_cancel2:
  "0 < n ==> root (2 * n) ^ n = - 1"

```

Note the space between - and 1.

```

using root_cancel [where n = 2 and k = 1]
apply (simp only: mult_ac)
apply (simp add: complex_one_def)
done

```

### 3 Discrete Fourier Transformation

We define operations DFT and IDFT for the discrete Fourier Transform and its inverse. Vectors are simply functions of type  $\text{nat} \Rightarrow \text{complex}$ .

DFT  $n$   $a$  is the transform of vector  $a$  of length  $n$ , IDFT its inverse.

**constdefs**

```

DFT :: "nat => (nat => complex) => (nat => complex)"
"DFT n a == (%i.  $\sum_{j=0..<n. (\text{root } n)^{(i * j)} * (a j)}$ )"
IDFT :: "nat => (nat => complex) => (nat => complex)"
"IDFT n a == (%i.  $(\sum_{k=0..<n. (a k) / (\text{root } n)^{(i * k)})$ )"

```

```

lemma "map (DFT 4 a) [0, 1, 2, 3] = ?x"
  by(simp add: DFT_def Sum4)

```

Lemmas for the correctness proof.

**lemma DFT\_lower:**

```

"DFT (2 * m) a i =
DFT m (%i. a (2 * i)) i +
(\text{root } (2 * m)) ^ i * DFT m (%i. a (2 * i + 1)) i"

```

**proof** (unfold DFT\_def)

```

have " $(\sum_{j = 0..<2 * m. \text{root } (2 * m)^{(i * j)} * a j) =$ 
 $(\sum_{j = 0..<m. \text{root } (2 * m)^{(i * (2 * j))} * a (2 * j)) +$ 
 $(\sum_{j = 0..<m. \text{root } (2 * m)^{(i * (2 * j + 1))} * a (2 * j + 1))$ "
(is "?s = _")
  by (simp add: setsum_splice)
also have "... =  $(\sum_{j = 0..<m. \text{root } m^{(i * j)} * a (2 * j)) +$ 
 $\text{root } (2 * m)^i *$ 
 $(\sum_{j = 0..<m. \text{root } m^{(i * j)} * a (2 * j + 1))$ "
(is "_ = ?t")

```

First pair of sums

```

apply (simp add: root_cancel1)

```

Second pair of sums

```

apply (simp add: setsum_right_distrib)
apply (simp add: power_add)

```

```

    apply (simp add: root_cancel1)
    apply (simp add: mult_ac)
  done
  finally show "?s = ?t" .
qed

```

lemma DFT\_upper:

```

  assumes mbound: "0 < m" and ibound: "m <= i"
  shows "DFT (2 * m) a i =
    DFT m (%i. a (2 * i)) (i - m) -
    root (2 * m) ^ (i - m) * DFT m (%i. a (2 * i + 1)) (i - m)"
proof (unfold DFT_def)

```

```

  have "( $\sum j = 0..<2 * m. \text{root } (2 * m) ^ (i * j) * a j =$ 
    ( $\sum j = 0..<m. \text{root } (2 * m) ^ (i * (2 * j)) * a (2 * j)) +$ 
    ( $\sum j = 0..<m. \text{root } (2 * m) ^ (i * (2 * j + 1)) * a (2 * j + 1))$ )"
    (is "?s = _")
  by (simp add: setsum_splice)
  also have "... =
    ( $\sum j = 0..<m. \text{root } m ^ ((i - m) * j) * a (2 * j) -$ 
     $\text{root } (2 * m) ^ (i - m) *$ 
    ( $\sum j = 0..<m. \text{root } m ^ ((i - m) * j) * a (2 * j + 1)$ )"
    (is "_ = ?t")

```

First pair of sums

```

  apply (simp add: root_cancel1)
  apply (simp add: root_unity ibound root_nonzero power_diff power_mult)

```

Second pair of sums

```

  apply (simp add: mbound root_cancel2)
  apply (simp add: setsum_right_distrib)
  apply (simp add: power_add)
  apply (simp add: root_cancel1)
  apply (simp add: power_mult)
  apply (simp add: mult_ac)
  done

```

finally show "?s = ?t" .

qed

lemma IDFT\_lower:

```

  "IDFT (2 * m) a i =
  IDFT m (%i. a (2 * i)) i +
  (1 / root (2 * m)) ^ i * IDFT m (%i. a (2 * i + 1)) i"

```

proof (unfold IDFT\_def)

```

  have "( $\sum j = 0..<2 * m. a j / \text{root } (2 * m) ^ (i * j) =$ 

```

```

( $\sum_{j=0}^{<m} a (2 * j) / \text{root } (2 * m) ^ (i * (2 * j))$ ) +
( $\sum_{j=0}^{<m} a (2 * j + 1) / \text{root } (2 * m) ^ (i * (2 * j + 1))$ )"
(is "?s = _")
by (simp add: setsum_splice)
also have "... = ( $\sum_{j=0}^{<m} a (2 * j) / \text{root } m ^ (i * j)$ ) +
(1 /  $\text{root } (2 * m)$ ) ^ i *
( $\sum_{j=0}^{<m} a (2 * j + 1) / \text{root } m ^ (i * j)$ )"
(is "_ = ?t")

```

First pair of sums

```
apply (simp add: root_cancel1)
```

Second pair of sums

```

apply (simp add: setsum_right_distrib)
apply (simp add: power_add)
apply (simp add: nonzero_power_divide root_nonzero)
apply (simp add: root_cancel1)
done

```

```
finally show "?s = ?t" .
```

qed

lemma IDFT\_upper:

```
assumes mbound: "0 < m" and ibound: "m <= i"
```

```
shows "IDFT (2 * m) a i =
```

```

IDFT m (%i. a (2 * i)) (i - m) -
(1 /  $\text{root } (2 * m)$ ) ^ (i - m) *
IDFT m (%i. a (2 * i + 1)) (i - m)"

```

proof (unfold IDFT\_def)

```

have "( $\sum_{j=0}^{<2 * m} a j / \text{root } (2 * m) ^ (i * j)$ ) =
( $\sum_{j=0}^{<m} a (2 * j) / \text{root } (2 * m) ^ (i * (2 * j))$ ) +
( $\sum_{j=0}^{<m} a (2 * j + 1) / \text{root } (2 * m) ^ (i * (2 * j + 1))$ )"
(is "?s = _")

```

```
by (simp add: setsum_splice)
```

```
also have "... =
```

```

( $\sum_{j=0}^{<m} a (2 * j) / \text{root } m ^ ((i - m) * j)$ ) -
(1 /  $\text{root } (2 * m)$ ) ^ (i - m) *
( $\sum_{j=0}^{<m} a (2 * j + 1) / \text{root } m ^ ((i - m) * j)$ )"
(is "_ = ?t")

```

First pair of sums

```

apply (simp add: root_cancel1)
apply (simp add: root_unity ibound root_nonzero power_diff power_mult)

```

Second pair of sums

```

    apply (simp add: nonzero_power_divide root_nonzero)
    apply (simp add: mbound root_cancel2)
    apply (simp add: setsum_divide_distrib)
    apply (simp add: power_add)
    apply (simp add: root_cancel1)
    apply (simp add: power_mult)
    apply (simp add: mult_ac)
  done
  finally show "?s = ?t" .
qed

DFT und IDFT are inverses.

declare divide_divide_eq_right [simp del]
  divide_divide_eq_left [simp del]

lemma power_diff_inverse:
  assumes nz: "(a::'a::field) ~= 0"
  shows "m <= n ==> (inverse a) ^ (n-m) = (a^m) / (a^n)"
  apply (induct n m rule: diff_induct)
  apply (simp add: nonzero_power_inverse
    nonzero_inverse_eq_divide [THEN sym] nz)
  apply simp
  apply (simp add: power_Suc nonzero_mult_divide_cancel_left nz)
  done

lemma power_diff_rev_if:
  assumes nz: "(a::'a::field) ~= 0"
  shows "(a^m) / (a^n) = (if n <= m then a ^ (m-n) else (1/a) ^ (n-m))"
proof (cases "n <= m")
  case True with nz show ?thesis
    by (simp add: power_diff)
next
  case False with nz show ?thesis
    by (simp add: power_diff_inverse nonzero_inverse_eq_divide [THEN sym])
qed

lemma power_divides_special:
  "(a::'a::field) ~= 0 ==>
  a ^ (i * j) / a ^ (k * i) = (a ^ j / a ^ k) ^ i"
  by (simp add: nonzero_power_divide power_mult [THEN sym] mult_ac)

theorem DFT_inverse:
  assumes i_less: "i < n"

```

```

shows "IDFT n (DFT n a) i = of_nat n * a i"
using [[linarith_split_limit = 0]]
apply (unfold DFT_def IDFT_def)
apply (simp add: setsum_divide_distrib)
apply (subst setsum_commute)
apply (simp only: times_divide_eq_left [THEN sym])
apply (simp only: power_divides_special [OF root_nonzero])
apply (simp add: power_diff_rev_if root_nonzero)
apply (simp add: setsum_divide_distrib [THEN sym]
  setsum_left_distrib [THEN sym])
proof -
  from i_less have i_diff: "!!k. i - k < n" by arith
  have diff_i: "!!k. k < n ==> k - i < n" by arith

  let ?sum = "%i j n. setsum (op ^ (if i <= j then root n ^ (j - i)
    else (1 / root n) ^ (i - j))) {0..<n} * a j"
  let ?sum1 = "%i j n. setsum (op ^ (root n ^ (j - i))) {0..<n} * a j"
  let ?sum2 = "%i j n. setsum (op ^ ((1 / root n) ^ (i - j))) {0..<n} * a j"

  from i_less have "( $\sum j = 0..<n. ?sum i j n$ ) =
    ( $\sum j = 0..<i. ?sum2 i j n$ ) + ( $\sum j = i..<n. ?sum1 i j n$ )"
    (is "?s = _")
    by (simp add: root_summation_inv nat_dvd_not_less
      setsum_add_split_nat_ivl [where f = "%j. ?sum i j n"])
  also from i_less i_diff
  have "... = ( $\sum j = i..<n. ?sum1 i j n$ )"
    by (simp add: root_summation_inv nat_dvd_not_less)
  also from i_less have "... =
    ( $\sum j \in \{i\} \cup \{i < .. <n\}. ?sum1 i j n$ )"
    by (simp only: ivl_disj_un)
  also have "... =
    (?sum1 i i n + ( $\sum j \in \{i < .. <n\}. ?sum1 i j n$ ))"
    by (simp add: setsum_Un_disjoint ivl_disj_int)
  also from i_less diff_i have "... = ?sum1 i i n"
    by (simp add: root_summation nat_dvd_not_less)
  also from i_less have "... = of_nat n * a i" (is "_ = ?t")
    by (simp add: of_nat_cplx)
  finally show "?s = ?t" .
qed

```

## 4 Discrete, Fast Fourier Transformation

FFT  $k$   $a$  is the transform of vector  $a$  of length  $2^k$ , IFFT its inverse.

**consts**

FFT :: "nat => (nat => complex) => (nat => complex)"

IFFT :: "nat => (nat => complex) => (nat => complex)"

**primrec**

"FFT 0 a = a"

"FFT (Suc k) a =

(let (x, y) = (FFT k (%i. a (2\*i)), FFT k (%i. a (2\*i+1)))

in (%i. if i < 2<sup>k</sup>

then x i + (root (2 ^ (Suc k))) ^ i \* y i

else x (i - 2<sup>k</sup>) - (root (2 ^ (Suc k))) ^ (i - 2<sup>k</sup>) \* y (i - 2<sup>k</sup>)))"

**primrec**

"IFFT 0 a = a"

"IFFT (Suc k) a =

(let (x, y) = (IFFT k (%i. a (2\*i)), IFFT k (%i. a (2\*i+1)))

in (%i. if i < 2<sup>k</sup>

then x i + (1 / root (2 ^ (Suc k))) ^ i \* y i

else x (i - 2<sup>k</sup>) -

(1 / root (2 ^ (Suc k))) ^ (i - 2<sup>k</sup>) \* y (i - 2<sup>k</sup>)))"

Finally, for vectors of length  $2^k$ , DFT and FFT, and IDFT and IFFT are equivalent.

**theorem DFT\_FFT:**

"!!a i. i < 2<sup>k</sup> ==> DFT (2<sup>k</sup>) a i = FFT k a i"

**proof** (induct k)

case 0

then show ?case by (simp add: DFT\_def)

**next**

case (Suc k)

assume i: "i < 2<sup>Suc k</sup>"

show ?case **proof** (cases "i < 2<sup>k</sup>")

case True

then show ?thesis **apply** simp **apply** (simp add: DFT\_lower)

**apply** (simp add: Suc) **done**

**next**

case False

from i have "i - 2<sup>k</sup> < 2<sup>k</sup>" by simp

with False i show ?thesis **apply** simp **apply** (simp add: DFT\_upper)

**apply** (simp add: Suc) **done**

qed

qed

```

theorem IDFT_IFFT:
  "!!a i. i < 2 ^ k ==> IDFT (2 ^ k) a i = IFFT k a i"
proof (induct k)
  case 0
  then show ?case by (simp add: IDFT_def)
next
  case (Suc k)
  assume i: "i < 2 ^ Suc k"
  show ?case proof (cases "i < 2 ^ k")
    case True
    then show ?thesis apply simp apply (simp add: IDFT_lower)
      apply (simp add: Suc) done
  next
    case False
    from i have "i - 2 ^ k < 2 ^ k" by simp
    with False i show ?thesis apply simp apply (simp add: IDFT_upper)
      apply (simp add: Suc) done
  qed
qed

lemma "map (FFT (Suc (Suc 0))) a [0, 1, 2, 3] = ?x"
  by simp

end

```