# Polynomial Factorization* 

René Thiemann and Akihisa Yamada

September 13, 2023


#### Abstract

Based on existing libraries for polynomial interpolation and matrices, we formalized several factorization algorithms for polynomials, including Kronecker's algorithm for integer polynomials, Yun's squarefree factorization algorithm for field polynomials, and a factorization algorithm which delivers root-free polynomials.

As side products, we developed division algorithms for polynomials over integral domains, as well as primality-testing and primefactorization algorithms for integers.


## Contents

1 Introduction ..... 2
1.1 Missing List ..... 3
1.2 Partitions ..... 7
1.3 merging functions ..... 11
2 Preliminaries ..... 36
2.1 Missing Multiset ..... 36
2.2 Precomputation ..... 39
2.3 Order of Polynomial Roots ..... 41
3 Explicit Formulas for Roots ..... 44
4 Division of Polynomials over Integers ..... 47
5 More on Polynomials ..... 55
6 Gauss Lemma ..... 62
7 Prime Factorization ..... 73
7.1 Definitions ..... 73
7.2 Proofs ..... 75

[^0]8 Rational Root Test ..... 90
9 Kronecker Factorization ..... 93
9.1 Definitions ..... 93
9.2 Code setup for divisors ..... 94
9.3 Proofs ..... 95
10 Polynomial Divisibility ..... 102
10.1 Fundamental Theorem of Algebra for Factorizations ..... 104
11 Square Free Factorization ..... 106
11.1 Yun's factorization algorithm ..... 111
11.2 Yun factorization and homomorphisms ..... 135
12 GCD of rational polynomials via GCD for integer polyno- mials ..... 138
13 Rational Factorization ..... 139

## 1 Introduction

The details of the factorization algorithms have mostly been extracted from Knuth's Art of Computer Programming [1]. Also Wikipedia provided valuable help.

As a first fast preprocessing for factorization we integrated Yun's factorization algorithm which identifies duplicate factors [2]. In contrast to the existing formalized result that the GCD of $p$ and $p^{\prime}$ has no duplicate factors (and the same roots as $p$ ), Yun's algorithm decomposes a polynomial $p$ into $p_{1}^{1} \cdot \ldots \cdot p_{n}^{n}$ such that no $p_{i}$ has a duplicate factor and there is no common factor of $p_{i}$ and $p_{j}$ for $i \neq j$. As a comparison, the GCD of $p$ and $p^{\prime}$ is exactly $p_{1} \cdot \ldots \cdot p_{n}$, but without decomposing this product into the list of $p_{i}$ 's.

Factorization over $\mathbb{Q}$ is reduced to factorization over $\mathbb{Z}$ with the help of Gauss' Lemma.

Kronecker's algorithm for factorization over $\mathbb{Z}$ requires both polynomial interpolation over $\mathbb{Z}$ and prime factorization over $\mathbb{N}$. Whereas the former is available as a separate AFP-entry, for prime factorization we mechanized a simple algorithm depicted in [1]: For a given number $n$, the algorithm iteratively checks divisibility by numbers until $\sqrt{n}$, with some optimizations: it uses a precomputed set of small primes (all primes up to 1000), and if $n \bmod 30=11$, the next test candidates in the range $[n, n+30)$ are only the 8 numbers $n, n+2, n+6, n+8, n+12, n+18, n+20, n+26$.

However, in theory and praxis it turned out that Kronecker's algorithm is too inefficient. Therefore, in a separate AFP-entry we formalized the Berlekamp-Zassenhaus factorization. ${ }^{1}$

There also is a combined factorization algorithm: For polynomials of degree 2 , the closed form for the roots of quadratic polynomials is applied. For polynomials of degree 3 , the rational root test determines whether the polynomial is irreducible or not, and finally for degree 4 and higher, Kronecker's factorization algorithm is applied.

### 1.1 Missing List

The provides some standard algorithms and lemmas on lists.

```
theory Missing-List
imports
    Matrix.Utility
    HOL-Library.Monad-Syntax
begin
fun concat-lists :: 'a list list \(\Rightarrow\) 'a list list where
    concat-lists [] = [[]]
\(\mid\) concat-lists \((\) as \(\# x s)=\) concat \((\operatorname{map}(\lambda v e c . \operatorname{map}(\lambda a . a \#\) vec) as) (concat-lists
xs))
lemma concat-lists-listset: set (concat-lists \(x s\) ) \(=\) listset (map set xs)
    by (induct xs, auto simp: set-Cons-def)
lemma sum-list-concat: sum-list (concat ls) \(=\) sum-list ( map sum-list ls)
    by (induct \(l s\), auto)
```

```
lemma listset: listset \(x s=\{\) ys. length ys \(=\) length \(x s \wedge(\forall i<\) length \(x s . y s!i \in\)
xs!i)\}
proof (induct xs)
    case (Cons x xs)
    let \(? n=\) length \(x s\)
    from Cons
    have ?case \(=(\) set-Cons \(x\{y s\). length ys \(=\) ? \(n \wedge(\forall i<\) ?n. ys \(!i \in x s!i)\}=\)
            \(\{\) ys. length ys \(=\) Suc ? \(n \wedge y s!0 \in x \wedge(\forall i<? n\). ys \(!\) Suc \(i \in x s!i)\})\)
            \((\) is \(-=(? L=? R))\)
            by (auto simp: all-Suc-conv)
    also have \(? L=? R\)
            by (auto simp: set-Cons-def, case-tac xa, auto)
    finally show? case by simp
```

[^1]lemma set-concat-lists[simp]: set (concat-lists xs) $=\{$ as. length as $=$ length $x s \wedge$ $(\forall i<$ length $x s$. as $!i \in \operatorname{set}(x s!i))\}$
unfolding concat-lists-listset listset by simp
declare concat-lists.simps[simp del]
fun find-map-filter :: $\left({ }^{\prime} a \Rightarrow{ }^{\prime} b\right) \Rightarrow\left({ }^{\prime} b \Rightarrow\right.$ bool $) \Rightarrow{ }^{\prime} a$ list $\Rightarrow{ }^{\prime} b$ option where
find-map-filter f $p[]=$ None
$\mid$ find-map-filter f $p(a \# a s)=($ let $b=f a$ in if $p b$ then Some $b$ else find-map-filter
f $p a s$ )
lemma find-map-filter-Some: find-map-filter f $p$ as $=$ Some $b \Longrightarrow p b \wedge b \in f$ 'set as
by (induct $f$ p as rule: find-map-filter.induct, auto simp: Let-def split: if-splits)
lemma find-map-filter-None: find-map-filter f $p$ as $=$ None $\Longrightarrow \forall b \in f$ 'set as. $\neg p b$
by (induct $f$ p as rule: find-map-filter.induct, auto simp: Let-def split: if-splits)
lemma remdups-adj-sorted-distinct $[$ simp $]:$ sorted $x s \Longrightarrow$ distinct (remdups-adj xs) by (induct xs rule: remdups-adj.induct) (auto)
lemma subseqs-length-simple:
assumes $b \in$ set (subseqs xs) shows length $b \leq$ length $x s$
using assms by (induct xs arbitrary:b;auto simp:Let-def Suc-leD)
lemma subseqs-length-simple-False:
assumes $b \in$ set (subseqs xs) length $x s<l e n g t h b$ shows False
using assms subseqs-length-simple by fastforce
lemma empty-subseqs[simp]: []$\in$ set (subseqs xs) by (induct xs, auto simp: Let-def)
lemma full-list-subseqs: $\{y s . y s \in$ set (subseqs xs) $\wedge$ length $y s=$ length $x s\}=\{x s\}$
proof (induct $x s$ )
case (Cons x xs)
have ? case $=\left(\left\{y s \in(\#) x^{\prime}\right.\right.$ set (subseqs xs $) \cup$ set $($ subseqs $x s)$. length ys $=$ Suc $($ length $x s)\}=(\#) x ‘\{x s\})($ is $-=(? l=? r))$ by (auto simp: Let-def)
also have $? l=\{y s \in(\#) x$ 'set (subseqs xs). length ys $=$ Suc (length $x s)\}$ using length-subseqs[of xs] using subseqs-length-simple-False by force
also have $\ldots=(\#) x^{\prime}\{y s \in$ set (subseqs $x$ s). length $y s=$ length $x s\}$
by auto
also have $\ldots=(\#) x$ ' $\{x s\}$ unfolding Cons by auto
finally show ?case by simp
qed $\operatorname{simp}$

```
lemma nth-concat-split: assumes i< length (concat xs)
    shows }\existsjk.j<length xs \wedgek<length (xs!j)^ concat xs ! i=xs!j!
    using assms
proof (induct xs arbitrary: i)
    case (Cons x xs i)
    define I where I= i - length x
    show ?case
    proof (cases i< length x)
        case True note l= this
        hence i:concat (Cons x xs)!i=x!i by (auto simp: nth-append)
        show ?thesis unfolding i
        by (rule exI[of-0], rule exI[of-i], insert Cons l, auto)
    next
        case False note l= this
        from l Cons(2) have i: i = length x + I I < length (concat xs) unfolding
I-def by auto
    hence iI: concat (Cons x xs)!i= concat xs !I by (auto simp: nth-append)
    from Cons(1)[OF i(2)] obtain jk where
            IH:j< length xs ^k<length (xs ! j)^ concat xs ! I = xs ! j!k by auto
        show ?thesis unfolding iI
            by (rule exI[of - Suc j], rule exI[of - k], insert IH, auto)
    qed
qed simp
lemma nth-concat-diff: assumes i1 < length (concat xs) i2 < length (concat xs)
i1 = i2
    shows \exists j1 k1 j2 k2. (j1,k1)\not= (j2,k2) ^j1< length xs }\wedge j2 < length xs
        \wedge <1<length (xs! j1) ^ k\mathcal{L}<length (xs!j2)
        ^concat xs!i1 = xs!j1!k1 ^ concat xs!i2 = xs! j2!k2
    using assms
proof (induct xs arbitrary: i1 i2)
    case (Cons x xs)
    define I1 where I1 = i1 - length }
    define I2 where I2 = i2 - length }
    show ?case
    proof (cases i1 < length x)
        case True note l1 = this
        hence i1: concat (Cons x xs)!i1 = x!i1 by (auto simp: nth-append)
    show ?thesis
    proof (cases i2 < length x)
        case True note l2 = this
        hence i2: concat (Cons x xs)!i2 = x ! i2 by (auto simp: nth-append)
        show ?thesis unfolding i1 i2
                            by (rule exI[of-0], rule exI[of-i1], rule exI[of-0], rule exI[of-i2],
                    insert Cons(4) l1 l2, auto)
    next
                case False note l2 = this
                    from l2 Cons(3) have i22: i2 = length x + I2 I2 < length (concat xs)
```

unfolding I2-def by auto
hence i2: concat (Cons $x x s$ ) ! i2 = concat $x s!$ I2 by (auto simp: nth-append) from nth-concat-split[OF i22(2)] obtain $j 2 \mathrm{k} 2$ where
*: j2 < length $x s \wedge k 2<$ length $(x s!j 2) \wedge$ concat $x s!I 2=x s!j 2!k 2$ by auto
show ?thesis unfolding i1 i2
by (rule exI[of-0], rule exI[of-i1], rule exI[of-Suc j2], rule exI[of-k2], insert * l1, auto)
qed
next
case False note $l 1=$ this
from $l 1 \operatorname{Cons}(2)$ have $111: i 1=$ length $x+I 1$ I1 < length (concat $x s$ ) unfolding I1-def by auto
hence i1: concat (Cons xas)!i1 = concat xs! I1 by (auto simp: nth-append) show ?thesis
proof (cases i2 < length $x$ )
case False note $12=$ this
from 12 Cons(3) have i22: i2 $_{2}=$ length $x+$ I2 I2 $<$ length (concat $x s$ )
unfolding I2-def by auto hence $i 2$ : concat (Cons $x$ xs)! i2 = concat xs! I2 by (auto simp: nth-append) from Cons(4) i11 i22 have diff: I1 $\neq I 2$ by auto from Cons(1)[OF i11(2) i22(2) diff] obtain j1 k1 j2 k2
where $I H:(j 1, k 1) \neq(j 2, k 2) \wedge j 1<$ length $x s \wedge j 2<$ length $x s$
$\wedge k 1<$ length $(x s!j 1) \wedge k 2<$ length $(x s!j 2)$
$\wedge$ concat xs! I1 = xs! j1! k1 $\wedge$ concat $x s!I 2=x s!j 2!k 2$ by auto show ?thesis unfolding i1 i2
by (rule exI[of - Suc j1], rule exI[of - k1], rule exI[of - Suc j2], rule exI[of - k2],
insert $I H$, auto)
next
case True note $l 2=$ this
hence i2: concat (Cons $x$ xs) ! i2 $=x!$ i2 by (auto simp: nth-append)
from nth-concat-split[OF i11(2)] obtain $j 1 k 1$ where
$*: j 1<$ length $x s \wedge k 1<$ length $(x s!j 1) \wedge$ concat $x s!I 1=x s!j 1!k 1$ by auto
show ?thesis unfolding i1 i2
by (rule exI[of - Suc j1], rule exI[of - $k 1]$, rule exI[of-0], rule exI[of-i2], insert * l2, auto)
qed
qed
qed auto
lemma list-all2-map-map: $(\bigwedge x . x \in$ set $x s \Longrightarrow R(f x)(g x)) \Longrightarrow$ list-all2 $R$ (map $f x s)($ map $g x s)$
by (induct xs, auto)

### 1.2 Partitions

Check whether a list of sets forms a partition, i.e., whether the sets are pairwise disjoint.

```
definition is-partition \(::\) ('a set) list \(\Rightarrow\) bool where
    \(i s\)-partition \(c s \longleftrightarrow(\forall j<\) length \(c s . \forall i<j . c s!i \cap c s!j=\{ \})\)
```

definition is-partition-alt :: ('a set) list $\Rightarrow$ bool where
is-partition-alt $c s \longleftrightarrow(\forall i j . i<l e n g t h ~ c s ~ \wedge j<l e n g t h ~ c s ~ \wedge i \neq j \longrightarrow c s!i \cap$
$c s!j=\{ \})$
lemma is-partition-alt: is-partition $=i s$-partition-alt
proof (intro ext)
fix $c s$ :: 'a set list
\{
assume is-partition-alt cs
hence is-partition cs unfolding is-partition-def is-partition-alt-def by auto
\}
moreover
\{
assume part: is-partition cs
have is-partition-alt cs unfolding is-partition-alt-def
proof (intro allI impI)
fix $i j$
assume $i<$ length cs $\wedge j<$ length cs $\wedge i \neq j$
with part show $c s!i \cap c s!j=\{ \}$
unfolding is-partition-def
by (cases $i<j$, simp, cases $j<i$, force, simp)
qed
\}
ultimately
show is-partition $c s=i s$-partition-alt $c s$ by auto
qed
lemma is-partition-Nil:
is-partition [] = True unfolding is-partition-def by auto
lemma is-partition-Cons:
is-partition $(x \# x s) \longleftrightarrow$ is-partition $x s \wedge x \cap \bigcup($ set $x s)=\{ \}($ is ?l $=? r)$
proof
assume ?l
have one: is-partition xs
proof (unfold is-partition-def, intro allI impI)
fix $j i$ assume $j<$ length xs and $i<j$
hence Suc $j<$ length $(x \# x s)$ and Suc $i<S u c j$ by auto
from〈?l〉[unfolded is-partition-def,THEN spec,THEN mp,THEN spec,THEN
$m p, O F$ this]
have $(x \# x s)!($ Suc $i) \cap(x \# x s)!($ Suc $j)=\{ \}$.

```
    thus xs!i\cap xs! j={} by simp
    qed
    have two: }x\cap\bigcup(\mathrm{ set xs)={}
    proof (rule ccontr)
    assume }x\cap\bigcup(\mathrm{ set xs) }={
    then obtain }y\mathrm{ where }y\inx\mathrm{ and }y\in\bigcup(\mathrm{ set xs) by auto
    then obtain z}\mathrm{ where z& set xs and }y\inz\mathrm{ by auto
    then obtain i}\mathrm{ where }i<length xs and xs!i=z using in-set-conv-nth[of z
xs] by auto
    with }\langley\inz\rangle\mathrm{ have }y\in(x#xs)!Suc i by aut
    moreover with }\langley\inx\rangle\mathrm{ have }y\in(x#xs)!0 by sim
    ultimately have (x#xs)!0 \cap (x#xs)!Suc i\not={} by auto
    moreover from <i< length xs` have Suc i< length(x#xs) by simp
    ultimately show False using <?l>[unfolded is-partition-def] by best
    qed
    from one two show ?r ..
next
    assume ?r
    show ?l
    proof (unfold is-partition-def, intro allI impI)
        fix ji
        assume j: j< length (x # xs)
        assume i: i<j
        from i obtain j' where j': j=Suc j' by (cases j, auto)
        with j have j'len: j'<length xs and j'elem: (x# ms)! j=xs! j' by auto
        show (x # xs) ! i\cap(x # xs)! j={}
        proof (cases i)
            case 0
            with j'elem have (x# xs)!i\cap(x# xs)!j=x\cap xs! j' by auto
            also have ...\subseteqx\cap\bigcup(set xs) using j'len by force
            finally show ?thesis using 〈?r〉 by auto
        next
            case (Suc i')
            with }i\mp@subsup{j}{}{\prime}\mathrm{ have }\mp@subsup{i}{}{\prime}\mp@subsup{j}{}{\prime}:\mp@subsup{i}{}{\prime}<\mp@subsup{j}{}{\prime}\mathrm{ by auto
            from Suc j' have (x# xs)!i\cap(x# xs)! j=xs! i'\cap xs! j' by auto
            with\langle?r\rangle i'j' j'len show ?thesis unfolding is-partition-def by auto
        qed
    qed
qed
lemma is-partition-sublist:
    assumes is-partition(us @ xs @ ys @ zs @ vs)
    shows is-partition (xs @ zs)
proof (rule ccontr)
    assume ᄀis-partition (xs @ zs)
    then obtain ij where j:j< length (xs@zs) and i:i<j and *:(xs @zs)!i\cap
(xs@zs)!j\not={}
    unfolding is-partition-def by blast
    then show False
```

```
proof (cases j < length xs)
    case True
    let ?m}=j+ length u
    let ?n = i + length us
    from True have ?m < length (us @ xs @ ys @ zs @ vs) by auto
    moreover from i have ? n<?m by auto
    moreover have(us @ xs @ ys @ zs @ vs)!?n \cap(us@ xs@ys @zs@vs)!
?m}\not={
            using i True * nth-append
    by (metis (no-types, lifting) add-diff-cancel-right' not-add-less2 order.strict-trans)
    ultimately show False using assms unfolding is-partition-def by auto
next
    case False
    let ?m = j + length us + length ys
    from j have m:?m < length (us @ xs @ ys @ zs @ vs) by auto
    have mj:(us@(xs@ys@zs@vs))!?m=(xs@zs)!j unfolding nth-append
using False j by auto
    show False
    proof (cases i< length xs)
        case True
        let ?n = i+ length us
        from i have ?n< ?m by auto
        moreover have(us@ xs @ ys @ zs @ vs)!?n = (xs @ zs)!i by (simp add:
True nth-append)
            ultimately show False using * m assms mj unfolding is-partition-def by
blast
    next
        case False
        let ?n = i + length us + length ys
        from i have i:?n<?m by auto
        moreover have (us @ xs @ ys @ zs @ vs)!?n = (xs @ zs)!i
            unfolding nth-append using False i j less-diff-conv2 by auto
        ultimately show False using * m assms mj unfolding is-partition-def by
blast
    qed
    qed
qed
lemma is-partition-inj-map:
    assumes is-partition xs
    and inj-on f ( \bigcupx\in set xs. x)
    shows is-partition (map ((`)f)xs)
proof (rule ccontr)
    assume }\neg\mathrm{ is-partition (map ((`)f)xs)
    then obtain ij where neq:i\not=j
        and i:i< length (map ((`)f) xs) and j:j< length (map ((`)f)xs)
        and map ((`)f) xs ! i\cap map ((`)f) xs ! j\not={}
        unfolding is-partition-alt is-partition-alt-def by auto
    then obtain x where }x\in\operatorname{map}((`f)xs!i and x\inmap ((`)f) xs!j by aut
```

```
    then obtain }yz\mathrm{ where yi:y }\inxs!i\mathrm{ and yx:f y = x and zj:z f xs!j and zx:f
z=x
    using ij by auto
    show False
    proof (cases y=z)
    case True
            with zj yi neq assms(1) i j show ?thesis by (auto simp: is-partition-alt
is-partition-alt-def)
    next
        case False
        have }y\in(\bigcupx\in\mathrm{ set xs. x) using yi i by force
    moreover have z}\in(\bigcupx\in\mathrm{ set xs. x) using zj j by force
    ultimately show ?thesis using assms(2) inj-on-def[off ( Ux\inset xs. x)] False
zx yx by blast
    qed
qed
context
begin
private fun is-partition-impl :: 'a set list }=>\mathrm{ ''a set option where
    is-partition-impl [] = Some {}
| is-partition-impl (as # rest) = do {
        all }\leftarrow is-partition-impl rest
        if as \cap all = {} then Some (all \cup as) else None
    }
lemma is-partition-code[code]: is-partition as =(is-partition-impl as \not= None)
proof -
    note [simp]= is-partition-Cons is-partition-Nil
    have \ bs. (is-partition as = (is-partition-impl as \not=None))}
    (is-partition-impl as = Some bs \longrightarrowbs=\bigcup (set as))
    proof (induct as)
    case (Cons as rest bs)
    show ?case
    proof (cases is-partition rest)
            case False
            thus ?thesis using Cons by auto
    next
            case True
            with Cons obtain c where rest: is-partition-impl rest = Some c
                by (cases is-partition-impl rest, auto)
            with Cons True show ?thesis by auto
    qed
    qed auto
    thus ?thesis by blast
qed
end
lemma case-prod-partition:
```

```
case-prod f(partition p xs) =f(filter p xs)(filter (Not \circ p) xs)
by simp
```

lemmas map-id $[$ simp $]=$ list.map-id

## 1.3 merging functions

definition fun-merge $::\left({ }^{\prime} a \Rightarrow{ }^{\prime} b\right)$ list $\Rightarrow{ }^{\prime} a$ set list $\Rightarrow{ }^{\prime} a \Rightarrow{ }^{\prime} b$ where fun-merge fs as $a \equiv(f s!(L E A S T i . i<l e n g t h ~ a s ~ \wedge a \in a s!i)) a$
lemma fun-merge: assumes
$i: i<$ length as
and $a: a \in a s!i$
and ident: $\bigwedge i j a . i<$ length $a s \Longrightarrow j<$ length $a s \Longrightarrow a \in a s!i \Longrightarrow a \in a s!j$
$\Longrightarrow\left(f_{s}!i\right) a=\left(f_{s}!j\right) a$
shows fun-merge fs as $a=(f s!i) a$
proof -
let ? $p=\lambda i . i<$ length as $\wedge a \in$ as $!i$
let ?l $=L E A S T i$. ?p $i$
have $p$ : ? $p$ ?l
by (rule LeastI, insert i a, auto)
show ?thesis unfolding fun-merge-def
by (rule $\operatorname{ident}[O F-i-a]$, insert $p$, auto)
qed
lemma fun-merge-part: assumes
part: is-partition as
and $i: i<$ length as
and $a: a \in a s!i$
shows fun-merge fs as $a=\left(f_{s}!i\right) a$
proof (rule fun-merge $[O F \quad i a]$ )
fix $i j a$
assume $i<$ length as and $j<$ length as and $a \in a s!i$ and $a \in a s!j$
hence $i=j$ using part[unfolded is-partition-alt is-partition-alt-def] by (cases $i$ $=j$, auto)
thus $(f s!i) a=(f s!j) a$ by $\operatorname{simp}$
qed
lemma map-nth-conv: map $f$ ss $=$ map $g t s \Longrightarrow \forall i<$ length ss. $f(s s!i)=g(t s!i)$ proof (intro allI impI)
fix $i$ show map $f$ ss $=$ map $g t s \Longrightarrow i<$ length $s s \Longrightarrow f(s s!i)=g(t s!i)$
proof (induct ss arbitrary: $i$ ts)
case Nil thus ?case by (induct ts) auto
next
case (Cons s ss) thus ?case
by (induct ts, simp, (cases $i$, auto))
qed
qed

```
lemma distinct-take-drop:
    assumes dist:distinct vs and len: i < length vs shows distinct(take i vs @ drop
(Suc i) vs)(is distinct(?xs@?ys))
proof -
    from id-take-nth-drop[OF len] have vs[symmetric]: vs = ?xs @ vs!i # ?ys .
    with dist have distinct ?xs and distinct(vs!i#?ys) and set ?xs \cap set(vs!i#?ys)
={} using distinct-append[of ?xs vs!i#?ys] by auto
    hence distinct ?ys and set ?xs \cap set ?ys = {} by auto
    with «distinct ?xs` show ?thesis using distinct-append[of ?xs ?ys] vs by simp
qed
lemma map-nth-eq-conv:
    assumes len: length xs = length ys
    shows (map fxs =ys)=(\foralli<length ys.f (xs!i)=ys!i)(is ?l = ?r)
proof -
    have (map fxs=ys)=(mapfxs=map id ys) by auto
    also have ... =( }\forall\textrm{i}<\mathrm{ length ys. f (xs!i) =id (ys!i))
        using map-nth-conv[of f xs id ys] nth-map-conv[OF len, of f id] unfolding len
        by blast
    finally show ?thesis by auto
qed
lemma map-upt-len-conv:
    map}(\lambdai.f(xs!i))[0..<length xs] = map fx
    by (rule nth-equalityI, auto)
lemma map-upt-add':
    map f[a..<a+b] = map (\lambdai.f(a+i))[0..<b]
    by (induct b, auto)
definition generate-lists :: nat }=>\mathrm{ 'a list }=>\mathrm{ 'a list list
    where generate-lists n xs \equivconcat-lists (map (\lambda -. xs) [0 ..<n])
lemma set-generate-lists[simp]: set (generate-lists n xs) = {as. length as = n ^
set as \subseteq set xs}
proof -
    {
        fix as
        have (length as = n ^(\foralli<n. as !i\in set xs ))=(length as = n ^ set as \subseteq
set xs)
    proof -
            {
            assume length as = n
            hence n: n= length as by auto
            have ( }\foralli<n\mathrm{ . as ! i set xs) = (set as }\subseteq\mathrm{ set xs) unfolding n
                unfolding all-set-conv-all-nth[of as \lambda x. x \in set xs, symmetric] by auto
            }
```

```
        thus ?thesis by auto
        qed
    }
    thus ?thesis unfolding generate-lists-def unfolding set-concat-lists by auto
qed
lemma nth-append-take:
    assumes i\leqlength xs shows (take ixs @ y#ys)!i=y
proof -
    from assms have a:length(take ixs)=i by simp
    have (take ixs @ y#ys)!(length(take ixs)) = y by (rule nth-append-length)
    thus ?thesis unfolding a.
qed
lemma nth-append-take-is-nth-conv:
    assumes i<j and j\leqlength xs shows (take jxs @ ys)!i=xs!i
proof -
    from assms have i< length(take j xs) by simp
    hence (take j xs @ys)!i=take j xs ! i unfolding nth-append by simp
    thus ?thesis unfolding nth-take[OF assms(1)].
qed
lemma nth-append-drop-is-nth-conv:
    assumes j<i and j\leq length xs and i\leq length xs
    shows (take j xs @ y # drop (Suc j) xs)!i = xs!i
proof -
    from <j< i〉 obtain n where ij: Suc(j+n) = i using less-imp-Suc-add by
auto
    with assms have i: i= length(take jxs)+Suc n by auto
    have len:Suc j+n\leqlength xs using assms i by auto
    have (take j xs @ y # drop (Suc j) xs)! i=
            (y # drop (Suc j) xs)!(i - length(take j xs)) unfolding nth-append i by auto
    also have ... = (y # drop (Suc j) xs)!(Suc n) unfolding i by simp
    also have \ldots=(drop (Suc j) xs)!n by simp
    finally show ?thesis using ij len by simp
qed
lemma nth-append-take-drop-is-nth-conv:
    assumes i\leq length xs and j\leq length xs and i\not=j
    shows (take j xs @ y # drop (Suc j) xs)! i = xs!i
proof -
    from assms have i<j\veei>j by auto
    thus ?thesis using assms
    by (auto simp: nth-append-take-is-nth-conv nth-append-drop-is-nth-conv)
qed
lemma take-drop-imp-nth:\llbrackettake i ss @ x # drop (Suc i) ss=ss\rrbracket\Longrightarrowx=ss!i
proof (induct ss arbitrary: i)
    case (Cons s ss)
```

```
    from <take \(i(s \# s s) @ x \#\) drop (Suc \(i)(s \# s s)=(s \# s s)\rangle\) show ?case
    proof (induct \(i\) )
    case (Suc i)
    from Cons have \(I H\) : take \(i\) ss @ \(x \#\) drop (Suc \(i\) ) ss \(=s s \Longrightarrow x=s s!i\) by auto
    from Suc have take iss @ \(x\) \# drop (Suc i) ss =ss by auto
    with \(I H\) show ? case by auto
qed auto
qed auto
lemma take-drop-update-first: assumes \(j<\) length \(d s\) and length \(c s=\) length \(d s\)
    shows (take \(j d s\) @ drop \(j c s)[j:=d s!j]=\) take (Suc j)ds @ drop (Suc j) cs
using assms
proof (induct \(j\) arbitrary: ds cs)
    case 0
    then obtain \(d d d s c c s\) where \(d s: d s=d \# d d s\) and \(c s: c s=c \# c c s\) by
(cases ds, simp, cases cs, auto)
    show ?case unfolding \(d s c s\) by auto
next
    case (Suc j)
    then obtain \(d d d s c c s\) where \(d s: d s=d \# d d s\) and \(c s: c s=c \# c c s\) by
(cases ds, simp, cases cs, auto)
    from \(\operatorname{Suc}(1)[o f d d s c c s] \operatorname{Suc}(2) \operatorname{Suc(3)}\) show ?case unfolding \(d s c s\) by auto
qed
lemma take-drop-update-second: assumes \(j<\) length \(d s\) and length cs \(=\) length
\(d s\)
    shows (take jds@drop jcs) \([j:=c s!j]=\) take \(j d s\) @ drop jcs
using assms
proof (induct \(j\) arbitrary: ds cs)
    case 0
    then obtain \(d d d s c c s\) where \(d s: d s=d \# d d s\) and \(c s: c s=c \# c c s\) by
(cases ds, simp, cases cs, auto)
    show ? case unfolding \(d s c s\) by auto
next
    case (Suc j)
    then obtain \(d d d s c c c s\) where \(d s: d s=d \# d d s\) and \(c s: c s=c \# c c s\) by
(cases ds, simp, cases cs, auto)
    from Suc(1)[of dds ccs] Suc(2) Suc(3) show ?case unfolding \(d s c s\) by auto
qed
lemma nth-take-prefix:
    length \(y s \leq\) length \(x s \Longrightarrow \forall i<\) length \(y s . x s!i=y s!i \Longrightarrow\) take (length \(y s) x s=y s\)
proof (induct xs ys rule: list-induct2')
    case ( \(4 x\) xs y ys)
    have take (length ys) \(x s=y s\)
    by (rule 4(1), insert \(4(2-3)\), auto)
    moreover from 4(3) have \(x=y\) by auto
    ultimately show ?case by auto
```


## lemma take-upt-idx:

assumes $i: i<$ length ls
shows take ils=[ls!j.j৮[0..<i]], ],
proof -
have $e: 0+i \leq i$ by auto
show ?thesis
using take-upt[OF e] take-map map-nth
by (metis (opaque-lifting, no-types) add.left-neutral i nat-less-le take-upt)
qed

```
fun distinct-eq :: (' \(a \Rightarrow{ }^{\prime} a \Rightarrow\) bool \() \Rightarrow{ }^{\prime}\) a list \(\Rightarrow\) bool where
    distinct-eq - [] = True
\(\mid\) distinct-eq eq \((x \# x s)=((\forall y \in\) set \(x s . \neg(e q y x)) \wedge\) distinct-eq eq \(x s)\)
```

lemma distinct-eq-append: distinct-eq eq $(x s$ @ ys) $=$ (distinct-eq eq $x s \wedge$ distinct-eq eq ys $\wedge(\forall x \in$ set $x s . \forall y \in$ set $y s . \neg(e q y x)))$
by (induct xs, auto)
lemma append-Cons-nth-left:
assumes $i<$ length xs
shows (xs@u\#ys)!i=xs!i
using assms nth-append[of xs - $i$ ] by simp
lemma append-Cons-nth-middle:
assumes $i=$ length xs
shows $(x s @ y \# z s)!i=y$
using assms by auto
lemma append-Cons-nth-right:
assumes $i>$ length $x s$
shows $(x s @ u \# y s)!i=(x s @ z \# y s)!i$
by (simp add: assms nth-append)
lemma append-Cons-nth-not-middle:
assumes $i \neq$ length xs
shows (xs@u\#ys)! $i=(x s @ z \# y s)!i$
by (metis assms list-update-length nth-list-update-neq)
lemmas append-Cons-nth = append-Cons-nth-middle append-Cons-nth-not-middle
lemma concat-all-nth:
assumes length $x s=$ length $y s$
and $\bigwedge i . i<$ length $x s \Longrightarrow$ length $(x s!i)=$ length $(y s!i)$
and $\bigwedge i j . i<$ length $x s \Longrightarrow j<$ length $(x s!i) \Longrightarrow P(x s!i!j)(y s!i!j)$
shows $\forall k<$ length (concat xs). $P$ (concat xs!k) (concat ys!k)

```
    using assms
proof (induct xs ys rule: list-induct2)
    case (Cons x xs y ys)
    from Cons(3)[of 0] have xy: length }x=\mathrm{ length }y\mathrm{ by simp
    from Cons(4)[of 0] xy have pxy: \ j. j< length x\LongrightarrowP(x!j) (y!j) by auto
    {
        fix }
        assume i: i< length xs
        with Cons(3)[of Suc i]
        have len: length (xs ! i) = length (ys!i) by simp
        from Cons(4)[of Suc i] i have }\j.j<length (xs!i)\LongrightarrowP(xs!i!j)(ys
i!j)
            by auto
    note len and this
    }
    from Cons(2)[OF this] have ind: \ k. k< length (concat xs) \LongrightarrowP (concat xs
!k)(concat ys!k)
    by auto
    show ?case unfolding concat.simps
    proof (intro allI impI)
    fix }
    assume k: k< length (x@ concat xs)
    show P ((x@ concat xs)!k) ((y @ concat ys)!k)
    proof (cases k< length x)
            case True
            show ?thesis unfolding nth-append using True xy pxy[OF True]
                by simp
    next
            case False
            with }k\mathrm{ have }k-(\mathrm{ length }x\mathrm{ ) < length (concat xs) by auto
            then obtain n where n: k - length x = n and nxs: n<length (concat xs)
by auto
            show ?thesis unfolding nth-append n n[unfolded xy] using False xy ind[OF
nxs]
            by auto
        qed
    qed
qed auto
lemma eq-length-concat-nth:
    assumes length xs = length ys
        and }\bigwedgei.i<length xs \Longrightarrowlength (xs!i)= length (ys!i
    shows length (concat xs) = length (concat ys)
using assms
proof (induct xs ys rule: list-induct2)
    case (Cons x xs y ys)
    from Cons(3)[of 0] have xy: length x length y by simp
    {
        fix }
```

```
        assume i< length xs
        with Cons(3)[of Suc i]
        have length (xs!i) = length (ys ! i) by simp
    }
    from Cons(2)[OF this] have ind: length (concat xs) = length (concat ys) by
simp
    show ?case using xy ind by auto
qed auto
primrec
    list-union :: 'a list }=>\mp@subsup{}{}{\prime
where
    list-union [] ys = ys
| list-union (x# xs) ys = (let zs = list-union xs ys in if x fet zs then zs else x
# zs)
lemma set-list-union[simp]: set (list-union xs ys) = set xs \cup set ys
proof (induct xs)
    case (Cons x xs) thus ?case by (cases x fet (list-union xs ys)) (auto)
qed simp
declare list-union.simps[simp del]
fun list-inter :: 'a list }=>\mathrm{ 'a list }=>\mp@subsup{}{}{\prime}'a list where
    list-inter [] bs = []
| list-inter (a#as) bs =
        (if }a\in\mathrm{ set bs then a # list-inter as bs else list-inter as bs)
lemma set-list-inter[simp]:
    set (list-inter xs ys) = set xs \cap set ys
    by (induct rule: list-inter.induct) simp-all
declare list-inter.simps[simp del]
primrec list-diff :: 'a list }=>\mp@subsup{}{}{\prime
    list-diff [] ys = []
| list-diff (x# xs) ys = (let zs = list-diff xs ys in if x e set ys then zs else x # zs)
lemma set-list-diff[simp]:
    set (list-diff xs ys) = set xs - set ys
proof (induct xs)
    case (Cons x xs) thus ?case by (cases x f set ys) (auto)
qed simp
declare list-diff.simps[simp del]
lemma nth-drop-0:0 < length ss \Longrightarrow (ss!0)#drop (Suc 0) ss = ss
    by (simp add:Cons-nth-drop-Suc)
```

```
lemma set-foldr-remdups-set-map-conv[simp]:
    set \((\) foldr \((\lambda x x s\). remdups \((f x @ x s)) x s[])=\bigcup(\operatorname{set}(\operatorname{map}(\operatorname{set} \circ f) x s))\)
    by (induct xs) auto
lemma subset-set-code[code-unfold]: set \(x s \subseteq\) set \(y s \longleftrightarrow\) list-all \((\lambda x . x \in\) set \(y s)\)
xs
    unfolding list-all-iff by auto
fun union-list-sorted where
    union-list-sorted \((x \# x s)(y \# y s)=\)
        (if \(x=y\) then \(x \#\) union-list-sorted xs ys
        else if \(x<y\) then \(x\) \# union-list-sorted \(x s\) ( \(y \# y s\) )
        else \(y\) \# union-list-sorted ( \(x \#\) xs) ys)
| union-list-sorted [] ys = ys
| union-list-sorted xs [] = xs
lemma [simp]: set (union-list-sorted \(x s\) ys) \(=\) set \(x s \cup\) set \(y s\)
    by (induct xs ys rule: union-list-sorted.induct, auto)
fun subtract-list-sorted :: ('a :: linorder) list \(\Rightarrow{ }^{\prime}\) 'a list \(\Rightarrow\) 'a list where
    subtract-list-sorted \((x \# x s)(y \# y s)=\)
        (if \(x=y\) then subtract-list-sorted xs ( \(y \# y s\) )
        else if \(x<y\) then \(x \#\) subtract-list-sorted \(x s\) ( \(y \# y s\) )
        else subtract-list-sorted ( \(x\) \# xs) ys)
| subtract-list-sorted [] ys = []
| subtract-list-sorted xs [] = xs
lemma set-subtract-list-sorted[simp]: sorted \(x s \Longrightarrow\) sorted \(y s \Longrightarrow\)
    set (subtract-list-sorted xs ys) \(=\) set \(x s-\) set ys
proof (induct xs ys rule: subtract-list-sorted.induct)
    case (1 x xs y ys)
    have \(x x s\) : sorted \((x \# x s)\) by fact
    have yys: sorted ( \(y \# y s\) ) by fact
    have \(x s\) : sorted \(x s\) using xxs by ( \(\operatorname{simp}\) )
    show ?case
    proof (cases \(x=y\) )
        case True
        thus ?thesis using 1 (1)[OF True xs yys] by auto
    next
        case False note neq = this
        note \(I H=1(2-3)[O F\) this \(]\)
        show ?thesis
            by (cases \(x<y\), insert IH xxs yys False, auto)
    qed
qed auto
```

```
lemma subset-subtract-listed-sorted: set (subtract-list-sorted xs ys) \subseteq set xs
    by (induct xs ys rule: subtract-list-sorted.induct, auto)
lemma set-subtract-list-distinct[simp]: distinct xs \Longrightarrow distinct (subtract-list-sorted
xs ys)
    by (induct xs ys rule: subtract-list-sorted.induct, insert subset-subtract-listed-sorted,
auto)
definition remdups-sort xs = remdups-adj (sort xs)
lemma remdups-sort[simp]: sorted (remdups-sort xs) set (remdups-sort xs) = set
xs
    distinct (remdups-sort xs)
    by (simp-all add: remdups-sort-def)
        maximum and minimum
lemma max-list-mono: assumes }\bigwedgex.x\in set xs - set ys \Longrightarrow\existsy.y\in set ys ^
x\leqy
    shows max-list xs \leq max-list ys
    using assms
proof (induct xs)
    case (Cons x xs)
    have }x\leq\mathrm{ max-list ys
    proof (cases x set ys)
        case True
        from max-list[OF this] show ?thesis .
    next
        case False
        with Cons(2)[of x] obtain y where y: y f set ys
            and xy: }x\leqy\mathrm{ by auto
        from xy max-list[OF y] show ?thesis by arith
    qed
    moreover have max-list xs \leq max-list ys
        by (rule Cons(1)[OF Cons(2)], auto)
    ultimately show ?case by auto
qed auto
fun min-list :: (' }a\mathrm{ :: linorder) list }=>\mp@subsup{}{}{\prime}a\mathrm{ where
    min-list [x] = x
|min-list (x # xs) = min x (min-list xs)
lemma min-list: ( }x:: ' a :: linorder ) \in set xs \Longrightarrow min-list xs \leqx
proof (induct xs)
    case oCons:(Cons y ys)
    show ?case
    proof (cases ys)
        case Nil
        thus ?thesis using oCons by auto
    next
```

```
    case (Cons z zs)
    hence min-list (y#ys)=\operatorname{min}y(min-list ys)
    by auto
    then show ?thesis
    using min-le-iff-disj oCons.hyps oCons.prems by auto
    qed
qed simp
lemma min-list-Cons:
    assumes xy: }x\leq
    and len: length xs = length ys
    and xsys: min-list xs \leqmin-list ys
    shows min-list (x # xs) \leqmin-list (y # ys)
    by (metis min-list.simps len length-greater-0-conv min.mono nth-drop-0 xsys xy)
lemma min-list-nth:
    assumes length xs = length ys
    and }\bigwedgei.i<length ys \Longrightarrowxs!i\leqys!
    shows min-list xs \leqmin-list ys
using assms
proof (induct xs arbitrary: ys)
    case (Cons x xs zs)
    from Cons(2) obtain y ys where zs:zs=y # ys by (cases zs,auto)
    note Cons = Cons[unfolded zs]
    from Cons(2) have len: length xs = length ys by simp
    from Cons(3)[of 0] have xy: }x\leqy\mathrm{ by simp
    {
        fix }
        assume i< length xs
        with Cons(3)[of Suc i] Cons(2)
        have xs!i\leqys!i by simp
    }
    from Cons(1)[OF len this] Cons(2) have ind: min-list xs \leqmin-list ys by simp
    show ?case unfolding zs
    by (rule min-list-Cons[OF xy len ind])
qed auto
lemma min-list-ex:
    assumes xs \not=[] shows \existsx\inset xs. min-list xs =x
    using assms
proof (induct xs)
    case oCons:(Cons x xs)
    show ?case
    proof (cases xs)
    case (Cons y ys)
    hence id: min-list (x# xs)=\operatorname{min}x(min-list xs) and nNil: xs \not=[] by auto
    show ?thesis
    proof (cases x \leqmin-list xs)
        case True
```

```
        show ?thesis unfolding id
            by (rule bexI[of - x], insert True, auto simp: min-def)
        next
        case False
        show ?thesis unfolding id min-def
            using oCons(1)[OF nNil] False by auto
        qed
    qed auto
qed auto
lemma min-list-subset:
    assumes subset: set ys \subseteqset xs and mem: min-list xs \in set ys
    shows min-list xs = min-list ys
    by (metis antisym empty-iff empty-set mem min-list min-list-ex subset subsetD)
        Apply a permutation to a list.
primrec permut-aux :: 'a list }=>(nat => nat) = 'a list | 'a list where
    permut-aux [] - - = [] |
    permut-aux (a# as) f bs=(bs!f0) # (permut-aux as (\lambdan.f (Suc n)) bs)
definition permut :: 'a list }=>(nat=>nat) => ' 'a list where
    permut as f}=\mathrm{ permut-aux as f as
declare permut-def[simp]
lemma permut-aux-sound:
    assumes i< length as
    shows permut-aux as f bs ! i=bs! (fi)
using assms proof (induct as arbitrary: if bs)
    case (Cons x xs)
    show ?case
    proof (cases i)
        case (Suc j)
        with Cons(2) have j< length xs by simp
        from Cons(1)[OF this] and Suc show ?thesis by simp
    qed simp
qed simp
lemma permut-sound:
    assumes i< length as
    shows permut as f!i= as ! (fi)
using assms and permut-aux-sound by simp
lemma permut-aux-length:
    assumes bij-betw f {..<length as} {..<length bs}
    shows length (permut-aux as f bs) = length as
by (induct as arbitrary: f bs, simp-all)
lemma permut-length:
    assumes bij-betw f {..< length as} {..< length as }
```

```
    shows length (permut as f) = length as
    using permut-aux-length[OF assms] by simp
declare permut-def[simp del]
lemma foldl-assoc:
```



```
    assumes }\fgh.f\cdot(g\cdoth)=f\cdotg\cdot
    shows foldl (.) (x · y) zs = x foldl ( () y zs
    using assms[symmetric] by (induct zs arbitrary: y) simp-all
lemma foldr-assoc:
    assumes }\fgh.b(bfg)h=bf(bgh
    shows foldr b xs (b y z)=b (foldr b xs y) z
    using assms by (induct xs) simp-all
lemma foldl-foldr-o-id:
    foldl (o) id fs = foldr (०) fs id
proof (induct fs)
    case (Cons ffs)
    have id \circf=f\circid by simp
    with Cons [symmetric] show ?case
    by (simp only: foldl-Cons foldr-Cons o-apply [of - - id] foldl-assoc o-assoc)
qed simp
lemma foldr-o-o-id[simp]:
    foldr ((o)\circf) xs id a= foldr f xs a
    by (induct xs) simp-all
lemma Ex-list-of-length-P:
    assumes }\foralli<n.\existsx.Px
    shows \existsxs.length xs =n\wedge(\foralli<n.P(xs!i)i)
proof -
    from assms have }\foralli.\existsx.i<n\longrightarrowPxi by sim
    from choice[OF this] obtain xs where xs: \bigwedgei. i<n\LongrightarrowP(xs i) i by auto
    show ?thesis
    by (rule exI[of - map xs [0 ..<n]], insert xs,auto)
qed
lemma ex-set-conv-ex-nth: (\existsx\inset xs. P x ) =( }\existsi<
    using in-set-conv-nth[of-xs] by force
lemma map-eq-set-zipD [dest]:
    assumes map f xs = map fys
    and (x,y)\in set (zip xs ys)
    shows fx=fy
using assms
proof (induct xs arbitrary: ys)
    case (Cons x xs)
```

```
    then show ?case by (cases ys) auto
qed simp
fun span :: ('a m bool) = 'a list | 'a list }\times\mathrm{ 'a list where
    span P (x # xs) =
        (if P x then let (ys,zs) = span P xs in (x#ys,zs)
        else ([], x # xs)) |
    span - [] = ([], [])
lemma span[simp]: span P xs = (takeWhile P xs,dropWhile P xs)
    by (induct xs, auto)
declare span.simps[simp del]
lemma parallel-list-update: assumes
    one-update: \xs i y. length xs = n\Longrightarrow \Longrightarrow < n \Longrightarrow r (xs!i) y \Longrightarrowpxs \Longrightarrowp
(xs[i := y])
    and init: length xs = n p xs
    and rel:length ys =n\bigwedge i. i<n\Longrightarrowr(xs!i)(ys!i)
    shows p ys
proof -
    note len = rel(1) init(1)
    {
        fix }
        assume i\leqn
        hence p (take i ys @ drop i xs)
        proof (induct i)
            case 0 with init show ?case by simp
        next
            case (Suc i)
            hence IH:p (take i ys @ drop i xs) by simp
            from Suc have i: i<n by simp
            let ?xs = (take i ys @ drop i xs)
            have length ?xs = n using i len by simp
            from one-update[OF this i-IH, of ys!i] rel(2)[OF i] i len
            show ?case by (simp add: nth-append take-drop-update-first)
        qed
    }
    from this[of n] show ?thesis using len by auto
qed
lemma nth-concat-two-lists:
    i< length (concat (xs :: 'a list list)) \Longrightarrow length (ys :: 'b list list) = length xs
    \Longrightarrow ( \bigwedge ~ i . ~ i < ~ l e n g t h ~ x s ~ \Longrightarrow ~ l e n g t h ~ ( y s ! i ) = ~ l e n g t h ~ ( x s ! i ) ) ~
    \Longrightarrow\existsjk.j< length xs ^k<length (xs!j)^(concat xs)!i=xs!j!k^
        (concat ys)!i=ys!j!k
proof (induct xs arbitrary: i ys)
    case (Cons x xs i yys)
    then obtain y ys where yys: yys = y # ys by (cases yys,auto)
```

```
    note Cons = Cons[unfolded yys]
    from Cons(4)[of 0] have [simp]: length y = length x by simp
    show ?case
    proof (cases i< length x)
    case True
    show ?thesis unfolding yys
        by (rule exI[of-0], rule exI[of-i], insert True Cons(2-4), auto simp:
nth-append)
    next
        case False
        let ?i= i - length }
    from False Cons(2-3) have ?i < length (concat xs) length ys = length xs by
auto
    note IH=Cons(1)[OF this]
    {
        fix }
        assume i< length xs
        with Cons(4)[of Suc i] have length (ys!i)= length (xs!i) by simp
    }
    from IH[OF this]
    obtain jk where IH1: j< length xs k< length (xs ! j)
        concat xs!? ? = xs ! j!k
        concat ys!? i = ys! j! k by auto
    show ?thesis unfolding yys
    by (rule exI[of - Suc j], rule exI[of - k], insert IH1 False, auto simp: nth-append)
    qed
qed simp
```

Removing duplicates w.r.t. some function.
fun remdups-gen $::\left({ }^{\prime} a \Rightarrow{ }^{\prime} b\right) \Rightarrow{ }^{\prime}$ 'a list $\Rightarrow{ }^{\prime}$ 'a list where
remdups-gen $f[]=[]$
$\mid$ remdups-gen $f(x \# x s)=x \#$ remdups-gen $f[y<-x s . \neg f x=f y]$
lemma remdups-gen-subset: set (remdups-gen f xs) $\subseteq$ set xs
by (induct $f$ xs rule: remdups-gen.induct, auto)
lemma remdups-gen-elem-imp-elem: $x \in$ set (remdups-gen $f$ xs) $\Longrightarrow x \in$ set xs
using remdups-gen-subset[of f xs] by blast
lemma elem-imp-remdups-gen-elem: $x \in$ set $x s \Longrightarrow \exists y \in$ set (remdups-gen $f x s$ ).
$f x=f y$
proof (induct $f$ xs rule: remdups-gen.induct)
case ( $2 f z z s$ )
show ?case
proof (cases f $x=f z$ )
case False
with 2(2) have $x \in \operatorname{set}[y \leftarrow z s . f z \neq f y]$ by auto
from 2(1)[OF this] show ?thesis by auto
qed auto

```
lemma take-nth-drop-concat:
    assumes \(i<\) length xss and xss \(!i=y s\)
    and \(j<\) length \(y s\) and \(y s!j=z\)
    shows \(\exists k<\) length (concat xss).
        take \(k\) (concat xss) \(=\) concat (take \(i\) xss \() @\) take \(j\) ys \(\wedge\)
        concat xss ! \(k=x s s!i!j \wedge\)
        drop \((\) Suc \(k)(\) concat xss \()=\) drop \((S u c j) y s @ \operatorname{concat}(d r o p(S u c ~ i) ~ x s s) ~\)
using \(\operatorname{assms}(1,2)\)
proof (induct xss arbitrary: i rule: List.rev-induct)
    case (snoc xs xss)
    then show ?case using assms by (cases \(i<\) length xss) (auto simp: nth-append)
qed simp
lemma concat-map-empty [simp]:
    concat \((\operatorname{map}(\lambda-.[]) x s)=[]\)
    by \(\operatorname{simp}\)
lemma map-upt-len-same-len-conv:
    assumes length \(x s=\) length ys
    shows map \((\lambda i . f(x s!i))[0 . .<\) length \(y s]=\operatorname{map} f x s\)
    unfolding assms [symmetric] by (rule map-upt-len-conv)
lemma concat-map-concat [simp]:
    concat (map concat \(x s\) ) \(=\) concat ( concat \(x s\) )
    by (induct xs) simp-all
lemma concat-concat-map:
    concat \((\) concat \((\operatorname{map} f x s))=\operatorname{concat}(\operatorname{map}(\) concat \(\circ f) x s)\)
    by (induct xs) simp-all
lemma UN-upt-len-conv [simp]:
    length \(x s=n \Longrightarrow(\bigcup i \in\{0 . .<n\} . f(x s!i))=\bigcup(\) set \((\operatorname{map} f x s))\)
    by (force simp: in-set-conv-nth)
lemma Ball-at-Least0LessThan-conv [simp]:
    length \(x s=n \Longrightarrow\)
    \((\forall i \in\{0 . .<n\} . P(x s!i)) \longleftrightarrow(\forall x \in\) set \(x s . P x)\)
    by (metis atLeast0LessThan in-set-conv-nth lessThan-iff)
lemma sum-list-replicate-length [simp]:
    sum-list (replicate (length xs) (Suc 0)) = length xs
    by (induct xs) simp-all
lemma list-all2-in-set2:
    assumes list-all2 \(P\) xs ys and \(y \in\) set ys
    obtains \(x\) where \(x \in\) set \(x s\) and \(P x y\)
```

using assms by (induct) auto
lemma map-eq-conv':
map $f x s=\operatorname{map} g y s \longleftrightarrow$ length $x s=$ length $y s \wedge(\forall i<$ length $x s . f(x s!i)=g$ (ys!i))
using map-equality-iff map-equality-iff nth-map-conv by auto
lemma list-3-cases[case-names Nil 1 2]:
assumes $x s=[] \Longrightarrow P$
and $\bigwedge x . x s=[x] \Longrightarrow P$
and $\bigwedge x y$ ys. $x s=x \# y \# y s \Longrightarrow P$
shows $P$
using assms by (rule remdups-adj.cases)
lemma list-4-cases[case-names Nil 12 3]:
assumes $x s=[] \Longrightarrow P$
and $\bigwedge x . x s=[x] \Longrightarrow P$
and $\bigwedge x y$. $x s=[x, y] \Longrightarrow P$
and $\bigwedge x y z z s . x s=x \# y \# z \# z s \Longrightarrow P$
shows $P$
using assms by (cases xs; cases tl xs; cases tl (tl xs), auto)
lemma foldr-append2 [simp]:
foldr $((@) \circ f) x s(y s$ @ zs) $=$ foldr $((@) \circ f) x s y s$ @ zs
by (induct xs) simp-all
lemma foldr-append2-Nil [simp]:
foldr $((@) \circ f) x s[]$ @ zs = foldr $((@) \circ f) x s z s$
unfolding foldr-append2 [symmetric] by simp
lemma UNION-set-zip:
$(\bigcup x \in \operatorname{set}(z i p[0 . .<$ length $x s]($ map $f x s)) . g x)=(\bigcup i<$ length xs. $g(i, f(x s!$
i)))
by (auto simp: set-conv-nth)
lemma zip-fst: $p \in \operatorname{set}(z i p$ as bs) $\Longrightarrow f s t p \in$ set as
by (metis in-set-zipE prod.collapse)
lemma zip-snd: $p \in \operatorname{set}(z i p$ as bs $) \Longrightarrow$ snd $p \in$ set bs by (metis in-set-zipE prod.collapse)
lemma zip-size-aux: size-list (size o snd) (zip ts ls) $\leq$ (size-list size ls)
proof (induct ls arbitrary: ts)
case (Cons l ls ts)
thus ?case by (cases ts, auto)
qed auto
We definie the function that remove the nth element of a list. It uses take and drop and the soundness is therefore not too hard to prove thanks
to the already existing lemmas.

```
definition remove-nth :: nat \(\Rightarrow\) 'a list \(\Rightarrow\) 'a list where
    remove-nth \(n x s \equiv\) (take \(n x s)\) @ (drop (Suc n) xs)
declare remove-nth-def[simp]
lemma remove-nth-len:
    assumes \(i: i<\) length \(x\)
    shows length \(x s=\) Suc (length (remove-nth \(i x s)\) )
proof -
    show ?thesis unfolding arg-cong[where \(f=\) length, OF id-take-nth-drop[OF i]]
        unfolding remove-nth-def by simp
qed
lemma remove-nth-length :
    assumes \(n\) - \(b d: n<\) length \(x s\)
    shows length (remove-nth \(n\) xs \()=\) length \(x s-1\)
    using \(n\)-bd by force
lemma remove-nth-id : length \(x s \leq n \Longrightarrow\) remove-nth \(n x s=x s\)
    by \(\operatorname{simp}\)
lemma remove-nth-sound-l:
    assumes \(p\) - ub: \(p<n\)
    shows (remove-nth \(n x s\) ) ! \(p=x s!p\)
proof (cases \(n<\) length \(x s\) )
    case True
    from length-take and True have ltk: length (take \(n x s)=n\) by simp
    \{
        assume pltn: \(p<n\)
        from this and ltk have plttk: \(p<\) length (take \(n x s\) ) by simp
        with nth-append[of take \(n\) xs - p]
        have ((take nxs) @ (drop (Suc n) xs)) ! p=take nxs ! p by auto
        with pltn and nth-take have \(((\) take \(n x s) @(d r o p(S u c n) x s))!p=x s!p\)
by \(\operatorname{simp}\)
    \}
    from this and ltk and \(p\)-ub show ?thesis by simp
    next
    case False
    hence length \(x s \leq n\) by arith
    with remove-nth-id show ?thesis by force
qed
lemma remove-nth-sound-r :
    assumes \(n \leq p\) and \(p<\) length xs
    shows (remove-nth \(n x s)!p=x s!(\) Suc \(p)\)
proof -
    from \(\langle n \leq p\rangle\) and \(\langle p<\) length \(x s\rangle\) have \(n\)-ub: \(n<\) length \(x s\) by arith
    from length-take and \(n\)-ub have ltk: length (take \(n x s\) ) \(=n\) by simp
```

```
from <n\leqp> and ltk and nth-append[of take n xs - p]
have Hrew:((take nxs)@ (drop (Suc n) xs))! p=drop (Suc n) xs ! (p-n) by
auto
    from <n\leq p` have idx: Suc n+(p-n)=Suc p by arith
    from <p<length xs〉 have Sp-ub: Suc p \leq length xs by arith
    from idx and Sp-ub and nth-drop have Hrew': drop (Suc n) xs ! (p-n) = xs !
(Suc p) by simp
    from Hrew and Hrew' show ?thesis by simp
qed
lemma nth-remove-nth-conv:
    assumes i< length (remove-nth n xs)
    shows remove-nth n xs ! i=xs!(if i<n then i else Suc i)
using assms remove-nth-sound-l remove-nth-sound-r[of n i xs] by auto
lemma remove-nth-P-compat :
    assumes aslbs: length as = length bs
    and Pab:\foralli.i< length as \longrightarrowP(as!i) (bs!i)
    shows }\foralli.i<length (remove-nth pas)\longrightarrowP(remove-nth p as!i) (remove-nth
pbs!i)
proof (cases p < length as)
    case True
    hence p-ub: p< length as by assumption
    with remove-nth-length have lr-ub: length (remove-nth pas)=length as - 1 by
auto
    {
    fix i assume i-ub:i< length (remove-nth p as)
    have P(remove-nth p as!i) (remove-nth p bs!i)
        proof (cases i<p)
        case True
            from i-ub and lr-ub have i-ub2: i< length as by arith
            from i-ub2 and Pab have P:P(as!i)(bs!i) by blast
            from P and remove-nth-sound-l[OF True, of as] and remove-nth-sound-l[OF
True, of bs]
            show ?thesis by simp
            next
            case False
            hence p-ub2: p\leqi by arith
            from i-ub and lr-ub have Si-ub: Suc i< length as by arith
            with Pab have P: P (as!Suc i) (bs!Suc i) by blast
            from i-ub and lr-ub have i-uba: i< length as by arith
            from i-uba and aslbs have i-ubb: i< length bs by simp
            from P and p-ub and aslbs and remove-nth-sound-r[OF p-ub2 i-uba]
            and remove-nth-sound-r[OF p-ub2 i-ubb]
            show ?thesis by auto
        qed
    }
    thus ?thesis by simp
    next
```

```
case False
    hence p-lba:length as }\leqp\mathrm{ by arith
    with aslbs have p-lbb: length bs \leq p by simp
    from remove-nth-id[OF p-lba] and remove-nth-id[OF p-lbb] and Pab
    show ?thesis by simp
qed
declare remove-nth-def[simp del]
definition adjust-idx :: nat }=>\mathrm{ nat }=>\mathrm{ nat where
    adjust-idx i j \equiv (if j < i then j else (Suc j))
definition adjust-idx-rev :: nat }=>\mathrm{ nat }=>\mathrm{ nat where
    adjust-idx-rev i j\equiv(if j<i then j else j - Suc 0)
lemma adjust-idx-rev1: adjust-idx-rev i (adjust-idx i j) = j
    using adjust-idx-def adjust-idx-rev-def by auto
lemma adjust-idx-rev2:
    assumes j\not=i shows adjust-idx i (adjust-idx-rev i j) =j
    using adjust-idx-def adjust-idx-rev-def assms by auto
lemma adjust-idx-i:
    adjust-idx i j\not=i
    using adjust-idx-def lessI less-irrefl-nat by auto
lemma adjust-idx-nth:
    assumes i: i< length xs
    shows remove-nth i xs ! j = xs ! adjust-idx i j (is ?l = ?r)
proof -
    let ?j = adjust-idx i j
    from i have ltake: length (take i xs) = i by simp
    note nth-xs = arg-cong[where f=\lambda xs. xs !?j, OF id-take-nth-drop[OF i],
unfolded nth-append ltake]
    show ?thesis
    proof (cases j<i)
        case True
        hence j: ?j = j unfolding adjust-idx-def by simp
        show ?thesis unfolding nth-xs unfolding j remove-nth-def nth-append ltake
            using True by simp
    next
        case False
        hence j: ?j = Suc j unfolding adjust-idx-def by simp
    from i have lxs: min (length xs) i=i by simp
    show ?thesis unfolding nth-xs unfolding j remove-nth-def nth-append
        using False by (simp add: lxs)
    qed
qed
```


## lemma adjust-idx-rev-nth:

assumes $i$ : $i<$ length xs and $j i: j \neq i$
shows remove-nth $i x s!$ adjust-idx-rev $i j=x s!j$ (is ?l $=? r)$
by (simp add: adjust-idx-nth adjust-idx-rev2 $i j i)$
lemma adjust-idx-length:
assumes $i: i<$ length $x s$ and $j: j<$ length (remove-nth $i x s$ )
shows adjust-idx $i j<$ length xs
using adjust-idx-def ij remove-nth-len by fastforce
lemma adjust-idx-rev-length:
assumes $i<$ length xs and $j<$ length xs and $j \neq i$
shows adjust-idx-rev $i j<$ length (remove-nth $i$ xs)
by (metis adjust-idx-def adjust-idx-rev2 assms not-less-eq remove-nth-len)
If a binary relation holds on two couples of lists, then it holds on the concatenation of the two couples.

```
lemma \(P\)-as-bs-extend:
    assumes lab: length as \(=\) length \(b s\)
    and lcd: length \(c s=\) length \(d s\)
    and nsab: \(\forall i . i<l e n g t h ~ b s \longrightarrow P(a s!i)(b s!i)\)
    and nscd: \(\forall i . i<\) length \(d s \longrightarrow P(c s!i)(d s!i)\)
    shows \(\forall i . i<l e n g t h(b s @ d s) \longrightarrow P((a s @ c s)!i)((b s @ d s)!i)\)
    by (simp add: lab nsab nscd nth-append)
```

        Extension of filter and partition to binary relations.
    fun filter2 $::\left(' a \Rightarrow{ }^{\prime} b \Rightarrow\right.$ bool $) \Rightarrow{ }^{\prime} a$ list $\Rightarrow{ }^{\prime} b$ list $\Rightarrow\left({ }^{\prime} a\right.$ list $\times$ ' $b$ list $)$ where
filter2 $P[]-=([],[]) \mid$
filter2 $P-[]=([],[]) \mid$
filter2 $P(a \# a s)(b \# b s)=($ if $P a b$
then $(a \#$ fst (filter2 $P$ as bs), $b \#$ snd (filter2 $P$ as $b s)$ )
else filter2 $P$ as $b s$ )
lemma filter2-length:
length $(f s t($ filter2 $P$ as $b s)) \equiv$ length $($ snd $(f i l t e r 2 ~ P a s ~ b s))$
proof (induct as arbitrary: bs)
case Nil
show ?case by simp
next
case (Cons a as) note $I H=$ this
thus ?case proof (cases bs)
case Nil
thus ?thesis by simp
next
case (Cons b bs)

```
        thus ?thesis proof (cases P ab)
        case True
        with Cons and IH show ?thesis by simp
        next
        case False
        with Cons and IH show ?thesis by simp
        qed
    qed
qed
lemma filter2-sound: \foralli.i<length (fst (filter2 P as bs)) \longrightarrowP (fst (filter2 P as
bs)!i)(snd (filter2 P as bs)!i)
proof (induct as arbitrary: bs)
case Nil
    thus ?case by simp
next
case (Cons a as) note IH=this
    thus ?case proof (cases bs)
        case Nil
        thus ?thesis by simp
        next
        case (Cons b bs)
        thus ?thesis proof (cases Pab)
        case False
            with Cons and IH show ?thesis by simp
        next
        case True
            {
                fix }
                assume i-bd: i< length (fst (filter2 P (a# as) (b # bs)))
                    have P(fst (filter2 P (a # as) (b# bs)) ! i) (snd (filter2 P (a # as) (b
# bs))!i)\quad proof (cases i)
                case 0
                    with True show ?thesis by simp
                next
                case (Suc j)
                    with i-bd and True have j< length (fst (filer2 P as bs)) by auto
                with Suc and IH and True show ?thesis by simp
            qed
        }
        with Cons show ?thesis by simp
        qed
    qed
qed
```



```
< ('a list > 'b list) where
    partition2 P as bs \equiv((filter2 P as bs), (filter2 (\lambdaa b.\neg (Pab)) as bs))
```

```
lemma partition2-sound-P: \(\forall i . i<l e n g t h(f s t(f s t(p a r t i t i o n 2 ~ P ~ a s ~ b s))) \longrightarrow\)
    \(P(f s t(f s t(\) partition2 \(P\) as \(b s))!i)(\) snd \((f s t(\) partition2 \(P\) as \(b s))!i)\)
    by (simp add: filter2-sound partition2-def)
```

lemma partition2-sound-nP: $\forall i . i<l e n g t h($ fst $($ snd $($ partition2 $P$ as bs))) $\longrightarrow$
$\neg P($ fst $($ snd $($ partition2 $P$ as bs $))!i)($ snd $($ snd $($ partition2 $P$ as bs $))!i)$
by (metis filter2-sound partition2-def snd-conv)

Membership decision function that actually returns the value of the index where the value can be found.

```
fun mem-idx :: ' \(a \Rightarrow\) 'a list \(\Rightarrow\) nat Option.option where
    mem-idx - [] \(\quad=\) None
    mem-idx \(x(a \#\) as \()=(\) if \(x=a\) then Some 0 else map-option Suc (mem-idx \(x\)
as))
```

lemma mem-idx-sound-output:
assumes mem-idx x as $=$ Some $i$
shows $i<$ length as $\wedge$ as $!i=x$
using assms proof (induct as arbitrary: $i$ )
case Nil thus ?case by simp
next
case (Cons a as) note $I H=$ this
thus ? case proof (cases $x=a$ )
case True with $I H$ (2) show ?thesis by simp
next
case False note neq-x-a=this
show ?thesis proof (cases mem-idx x as)
case None with $I H$ (2) and neq-x-a show ?thesis by simp
next
case (Some j)
with $I H$ (2) and neq-x-a have $i=S u c j$ by simp
with $I H(1)$ and Some show ?thesis by simp
qed
qed
qed
lemma mem-idx-sound-output2:
assumes mem-idx x as $=$ Some $i$
shows $\forall j . j<i \longrightarrow a s!j \neq x$
using assms proof (induct as arbitrary: $i$ )
case Nil thus ?case by simp
next
case (Cons a as) note $I H=$ this
thus ?case proof (cases $x=a$ )
case True with $I H$ show ?thesis by simp
next
case False note neq-x-a $=$ this
show ?thesis proof (cases mem-idx x as)
case None with $I H$ (2) and neq-x-a show ?thesis by simp

```
        next
        case (Some j)
        with IH(2) and neq-x-a have eq-i-Sj:i=Suc j by simp
        {
            fix }k\mathrm{ assume k-bd: k<i
            have (a # as)!k\not=x
            proof (cases k)
            case 0 with neq-x-a show ?thesis by simp
            next
            case (Suc l)
                with }k-bd\mathrm{ and eq-i-Sj have l-bd:l<j by arith
                    with IH(1) and Some have as!l\not=x by simp
                with Suc show ?thesis by simp
            qed
        }
        thus ?thesis by simp
        qed
        qed
qed
lemma mem-idx-sound:
    (x\in set as) = (\existsi. mem-idx x as = Some i)
proof (induct as)
case Nil thus ?case by simp
next
case (Cons a as) note IH = this
    show ?case proof (cases x=a)
    case True thus ?thesis by simp
    next
    case False
    {
        assume x f set (a# as)
            with False have x\in set as by simp
            with IH obtain i where Some-i: mem-idx x as =Some i by auto
            with False have mem-idx x (a# as)=Some (Suc i) by simp
            hence \existsi. mem-idx x (a#as)=Some i by simp
        }
        moreover
        {
            assume \existsi. mem-idx x (a# as)=Some i
            then obtain i where Some-i: mem-idx x (a# as)=Some i by fast
            have }x\in\mathrm{ set as proof (cases i)
                case 0 with mem-idx-sound-output[OF Some-i] and False show ?thesis
by simp
            next
            case (Suc j)
                with Some-i and False have mem-idx x as = Some j by simp
                hence }\existsi\mathrm{ . mem-idx x as = Some i by simp
            with IH show ?thesis by simp
```

```
        qed
        hence }x\in\operatorname{set}(a#\mathrm{ as) by simp
    }
    ultimately show ?thesis by fast
    qed
qed
lemma mem-idx-sound2:
    (x\not\in set as ) = (mem-idx x as = None )
    unfolding mem-idx-sound by auto
lemma sum-list-replicate-mono: assumes w1 \leq (w2 :: nat)
    shows sum-list (replicate n w1)\leq sum-list (replicate n w2)
proof (induct n)
    case (Suc n)
    thus ?case using <w1 \leq w2> by auto
qed simp
lemma all-gt-0-sum-list-map:
    assumes *: \x.fx> (0::nat)
        and x: x\in set xs and len: 1< length xs
    shows f}x<(\sumx\leftarrowxs.fx
    using x len
proof (induct xs)
    case (Cons y xs)
    show ?case
    proof (cases y=x)
        case True
        with *[of hd xs] Cons(3) show ?thesis by (cases xs, auto)
    next
        case False
        with Cons(2) have x: x\in set xs by auto
        then obtain zzs where xs:xs=z#zs by (cases xs,auto)
        show ?thesis
        proof (cases length zs)
            case 0
            with x xs *[of y] show ?thesis by auto
        next
            case (Suc n)
            with xs have 1< length xs by auto
            from Cons(1)[OF x this] show ?thesis by simp
        qed
    qed
qed simp
lemma finite-distinct: finite {xs.distinct xs ^ set xs=X} (is finite (?S X))
proof (cases finite X)
    case False
    with finite-set have id:?S }X={}\mathrm{ by auto
```

```
    show ?thesis unfolding id by auto
next
    case True
    show ?thesis
    proof (induct rule: finite-induct[OF True])
        case (2 x X)
    let ?L }={0..<\operatorname{card}(\mathrm{ insert }xX)}\times?S 
    from 2(3)
    have fin: finite ?L by auto
    let ?f = \lambda (i,xs). take ixs @ x # drop ixs
    show ?case
    proof (rule finite-surj[OF fin, of - ?f], rule)
        fix xs
        assume xs \in?S (insert x X)
        hence dis: distinct xs and set: set xs = insert x X by auto
        from distinct-card[OF dis] have len: length xs = card (set xs) by auto
            from set[unfolded set-conv-nth] obtain i where x: x = xs!i and i:i<
length xs by auto
    from i have min: min (length xs) i=i by simp
    let ?ys= take ixs @ drop (Suc i) xs
    from id-take-nth-drop[OF i] have xsi:xs = take i xs @ xs ! i # drop (Suc i)
xs.
    also have ... = ?f (i,?ys) unfolding split
        by (simp add: min x)
    finally have xs: xs = ?f (i,?ys) .
    show xs\in?f'?L
    proof (rule image-eqI, rule xs, rule SigmaI)
        show i}\in{0..<card (insert x X)} using i[unfolded len] set[symmetric] by
simp
    next
        from dis xsi have disxsi: distinct (take i xs @ xs ! i # drop (Suc i) xs) by
simp
            note disxsi = disxsi[unfolded distinct-append x[symmetric]]
            have xys: x & set ?ys using disxsi by auto
            from distinct-take-drop[OF dis i]
            have disys: distinct ?ys .
            have insert x (set ?ys) = set xs unfolding arg-cong[OF xsi, of set] x by
simp
            hence insert x (set ?ys) = insert x X unfolding set by simp
            from this[unfolded insert-eq-iff[OF xys 2(2)]]
            show ?ys \in?S X using disys by auto
            qed
    qed
    qed simp
qed
lemma finite-distinct-subset:
    assumes finite X
    shows finite { xs . distinct xs ^ set xs\subseteqX} (is finite (?S X))
```

```
proof -
    let ?X ={ {xs.distinct xs ^ set xs = Y}|Y.Y\subseteqX}
    have id: ?S X=U ?X by blast
    show ?thesis unfolding id
    proof (rule finite-Union)
        show finite ?X using assms by auto
    next
        fix M
        assume M \in?X
        with finite-distinct show finite M by auto
    qed
qed
lemma map-of-filter:
    assumes P x
    shows map-of [(x',y)\leftarrowys.P x] x = map-of ys x
proof (induct ys)
    case (Cons xy ys)
    obtain \mp@subsup{x}{}{\prime}y\mathrm{ where xy: xy = ( }\mp@subsup{x}{}{\prime},y)\mathrm{ by force}
    show ?case
        using assms local.Cons by auto
qed simp
lemma set-subset-insertI: set xs \subseteq set (List.insert x xs)
    by auto
lemma set-removeAll-subset: set (removeAll x xs) \subseteq set xs
    by auto
lemma map-of-append-Some:
    map-of xs y = Some z \Longrightarrowmap-of (xs@ ys) y=Some z
    by simp
lemma map-of-append-None:
    map-of xs y = None \Longrightarrow map-of (xs @ ys) y = map-of ys y
    by (simp add: map-add-def)
end
```


## 2 Preliminaries

### 2.1 Missing Multiset

This theory provides some definitions and lemmas on multisets which we did not find in the Isabelle distribution.

```
theory Missing-Multiset
imports
```

HOL-Library.Multiset
Missing-List

## begin

lemma remove-nth-soundness:
assumes $n<$ length as
shows mset (remove-nth $n$ as $)=$ mset as $-\{\#(a s!n) \#\}$
using assms
proof (induct as arbitrary: $n$ )
case (Cons a as)
note $[$ simp $]=$ remove-nth-def
show ?case
proof (cases $n$ )
case (Suc n)
with Cons have $n$-bd: $n<$ length as by auto
with Cons have mset (remove-nth $n$ as) $=$ mset as $-\{\#$ as ! $n \#\}$ by auto
hence $G$ : mset (remove-nth (Suc n) $(a \#$ as $))=$ mset as $-\{\#$ as $!n \#\}+$
\{\#a\#\}
by $\operatorname{simp}$
thus ?thesis
proof (cases $a=a s!n$ )
case True
with $G$ and Suc and insert-DiffM2[symmetric]
and insert-DiffM2[of - \{\#as!n\#\}]
and nth-mem-mset[of $n a s]$ and $n$-bd
show ?thesis by auto
next
case False
from $G$ and Suc and diff-union-swap[OF this[symmetric], symmetric] show
?thesis by simp
qed
qed auto
qed auto
lemma multiset-subset-insert: $\{p s . p s \subseteq \#$ add-mset $x x s\}=$
$\{p s . p s \subseteq \# x s\} \cup$ add-mset $x$ ' $\{p s . p s \subseteq \# x s\}($ is $? l=? r)$
proof -
\{
fix $p s$
have $(p s \in ? l)=(p s \subseteq \# x s+\{\# x \#\})$ by auto
also have $\ldots=(p s \in ? r)$
proof (cases $x \in \#$ ps)
case True
then obtain $q s$ where $p s: p s=q s+\{\# x \#\}$ by (metis insert-DiffM2)
show ?thesis unfolding ps mset-subset-eq-mono-add-right-cancel
by (auto dest: mset-subset-eq-insertD)
next
case False

```
            hence id: (ps\subseteq# xs + {#x#})=(ps\subseteq# xs)
            by (simp add: subset-mset.inf.absorb-iff2 inter-add-left1)
            show ?thesis unfolding id using False by auto
        qed
        finally have (ps\in?l)=(ps\in?r).
    }
    thus ?thesis by auto
qed
lemma multiset-of-subseqs:mset'set (subseqs xs) ={ ps.ps\subseteq# mset xs}
proof (induct xs)
    case (Cons x xs)
    show ?case (is ?l = ?r)
    proof -
        have id: ?r = {ps. ps\subseteq# mset xs} \cup (add-mset x) ' {ps. ps\subseteq# mset xs}
            by (simp add: multiset-subset-insert)
    show ?thesis unfolding id Cons[symmetric]
        by (auto simp add: Let-def) (metis UnCI image-iff mset.simps(2))
    qed
qed simp
lemma remove1-mset: w set vs \Longrightarrow mset (remove1 wvs)+{#w#}= mset vs
    by (induct vs) auto
lemma fold-remove1-mset: mset ws \subseteq# mset vs \Longrightarrowmset (fold remove1 ws vs) +
mset ws = mset vs
proof (induct ws arbitrary: vs)
    case (Cons w ws vs)
    from Cons(2) have w\in set vs using set-mset-mono by force
    from remove1-mset[OF this] have vs: mset vs = mset (remove1 wvs)+{#w#}
by simp
    from Cons(2)[unfolded vs] have mset ws \subseteq# mset (remove1 w vs) by auto
    from Cons(1)[OF this,symmetric]
    show ?case unfolding vs by (simp add: ac-simps)
qed simp
lemma subseqs-sub-mset:ws \in set (subseqs vs) \Longrightarrow mset ws }\subseteq#mset v
proof (induct vs arbitrary: ws)
    case (Cons v vs Ws)
    note mem = Cons(2)
    note IH=Cons(1)
    show ?case
    proof (cases Ws)
    case (Cons w ws)
    show ?thesis
    proof (cases v=w)
        case True
        from mem Cons have ws\in set (subseqs vs) by (auto simp: Let-def Cons-in-subseqsD[of
- ws vs])
```

```
        from IH[OF this]
        show ?thesis unfolding Cons True by simp
    next
        case False
    with mem Cons have Ws set (subseqs vs) by (auto simp: Let-def Cons-in-subseqsD[of
- ws vs])
    note IH = mset-subset-eq-count[OF IH[OF this]]
    with IH[of v] show ?thesis by (intro mset-subset-eqI, auto, linarith)
    qed
    qed simp
qed simp
lemma filter-mset-inequality: filter-mset fxs }\not=xs\Longrightarrow\existsx\in#xs.\negf
    by (induct xs,auto)
end
```


### 2.2 Precomputation

This theory contains precomputation functions, which take another function $f$ and a finite set of inputs, and provide the same function $f$ as output, except that now all values $f i$ are precomputed if $i$ is contained in the set of finite inputs.

```
theory Precomputation
imports
```

    Containers.RBT-Set2
    HOL-Library.RBT-Mapping
    begin
lemma lookup-tabulate: $x \in$ set $x s \Longrightarrow$ Mapping.lookup (Mapping.tabulate xs $f$ ) $x$
$=$ Some ( $f x$ )
by (transfer, simp add: map-of-map-Pair-key)
lemma lookup-tabulate2: Mapping.lookup (Mapping.tabulate xs f) $x=$ Some $y \Longrightarrow$
$y=f x$
by transfer (metis map-of-map-Pair-key option.distinct(1) option.sel)
definition memo-int $::$ int $\Rightarrow$ int $\Rightarrow\left(\right.$ int $\left.\Rightarrow{ }^{\prime} a\right) \Rightarrow\left(\right.$ int $\left.\Rightarrow{ }^{\prime} a\right)$ where
memo-int low up $f \equiv$ let $m=$ Mapping.tabulate [low .. up] $f$
in $(\lambda x$. if $x \geq$ low $\wedge x \leq$ up then the (Mapping.lookup $m x)$ else $f x$ )
lemma memo-int[simp]: memo-int low up $f=f$
proof (intro ext)
fix $x$
show memo-int low up $f x=f x$
proof (cases $x \geq$ low $\wedge x \leq u p$ )
case False
thus ?thesis unfolding memo-int-def by auto
next

```
        case True
        from True have x: x set [low .. up] by auto
        with True lookup-tabulate[OF this, of f]
        show ?thesis unfolding memo-int-def by auto
    qed
qed
definition memo-nat :: nat => nat }=>(nat=>'a)=>(nat => 'a) where
    memo-nat low up f}\equiv\mathrm{ let m}=\mathrm{ Mapping.tabulate [low ..< up] f
        in (\lambdax. if }x\geq\mathrm{ low }\wedgex<up then the (Mapping.lookup m x) else f x)
lemma memo-nat[simp]: memo-nat low up f}=
proof (intro ext)
    fix }
    show memo-nat low up f x = f x
    proof (cases x \geq low }\wedgex<up
        case False
        thus ?thesis unfolding memo-nat-def by auto
    next
        case True
        from True have x: x fet [low ..<up] by auto
        with True lookup-tabulate[OF this, of f]
        show ?thesis unfolding memo-nat-def by auto
    qed
qed
definition memo :: 'a list => (' }a=>\mp@subsup{|}{}{\prime}b)=>('a=>'b) wher
    memo xs f 三 let m= Mapping.tabulate xs f
        in (\lambda x. case Mapping.lookup m x of None }=>fx|\mathrm{ Some y }=>y
lemma memo[simp]: memo xs f=f
proof (intro ext)
    fix }
    show memo xs fx=fx
    proof (cases Mapping.lookup (Mapping.tabulate xs f) x)
        case None
        thus ?thesis unfolding memo-def by auto
    next
        case (Some y)
        with lookup-tabulate2[OF this]
        show ?thesis unfolding memo-def by auto
    qed
qed
end
```


### 2.3 Order of Polynomial Roots

We extend the collection of results on the order of roots of polynomials. Moreover, we provide code-equations to compute the order for a given root and polynomial.

```
theory Order-Polynomial
imports
    Polynomial-Interpolation.Missing-Polynomial
begin
lemma order-linear \([\) simp \(]\) : order \(a[:-a, 1:]=\) Suc 0 unfolding order-def
proof (rule Least-equality, intro notI)
    assume \([:-a, 1:]\) - Suc (Suc 0) dvd \([:-a, 1:]\)
    from dvd-imp-degree-le[OF this] show False by auto
next
    fix \(n\)
    assume \(*: \neg[:-a, 1:]\) Suc \(n\) dvd \([:-a, 1:]\)
    thus Suc \(0 \leq n\)
        by (cases \(n\), auto)
qed
declare order-power- \(n-n[\) simp \(]\)
lemma linear-power-nonzero: [: \(a, 1:] \wedge n \neq 0\)
proof
    assume \([: a, 1:] \widehat{ } n=0\)
    with arg-cong[OF this, of degree, unfolded degree-linear-power]
    show False by auto
qed
```

lemma order-linear-power': order $a([: b, 1:] \uparrow S u c n)=($ if $b=-a$ then Suc $n$ else
0)
proof (cases $b=-a$ )
case True
thus ?thesis unfolding True order-power-n-n by simp
next
case False
let ? $p=[: b, 1:]^{\wedge}$ Suc $n$
from linear-power-nonzero have $? p \neq 0$.
have $p: ? p=\left(\prod a \leftarrow\right.$ replicate $($ Suc $n) b$. $\left.[: a, 1:]\right)$ by auto
\{
assume order a $? p \neq 0$
then obtain $m$ where ord: order $a ? p=$ Suc $m$ by (cases order a ?p, auto)
from order $[O F\langle ? p \neq 0\rangle$, of $a$, unfolded ord $]$ have dvd: $[:-a, 1:]$ ~ Suc $m$ dvd
?p by auto
from poly-linear-exp-linear-factors[OF dvd[unfolded p]] False have False by
auto
\}
hence order $a ? p=0$ by auto

```
    with False show ?thesis by simp
qed
lemma order-linear-power: order a ([:b, 1:]` n)=( if b = - a then n else 0)
proof (cases n)
    case (Suc m)
    show ?thesis unfolding Suc order-linear-power' by simp
qed simp
lemma order-linear': order a [:b, 1:] = (if b=-a then 1 else 0)
    using order-linear-power'[of a b 0] by simp
lemma degree-div-less:
    assumes p:(p::'a::field poly) }=0\mathrm{ and dvd: r dvd p and deg: degree r}\not=
    shows degree ( }p\mathrm{ div r) < degree p
proof -
    from dvd obtain q}\mathrm{ where prq: p=r*q unfolding dvd-def by auto
    have degree p= degree r + degree q
        unfolding prq
        by (rule degree-mult-eq, insert p prq, auto)
    with deg have deg: degree q< degree p by auto
    from prq have q=p div r
        using deg p by auto
    with deg show ?thesis by auto
qed
lemma order-sum-degree: assumes p}=
    shows sum (\lambda a. order a p) {a. poly pa=0}\leqdegree p
proof -
    define n where n= degree p
    have degree p}\leqn\mathrm{ unfolding n-def by auto
    thus ?thesis using <p}\not=0
    proof (induct n arbitrary: p)
        case (0 p)
        define a where a= coeff p 0
        from 0 have degree p=0 by auto
        hence p:p=[:a:] unfolding a-def
            by (metis degree-0-id)
        with 0 have a\not=0 by auto
        thus ?case unfolding p by auto
    next
        case (Suc m p)
        note order = order[OF }\langlep\not=0\rangle
        show ?case
        proof (cases \existsa. poly p a=0)
            case True
            then obtain a where root: poly p a=0 by auto
```

with order-root $[$ of $p$ a] Suc obtain $n$ where orda: order a $p=$ Suc $n$
by (cases order a p, auto)
let ? $a=[:-a, 1:]$ ^Suc $n$
from order-decomp $[O F\langle p \neq 0\rangle$, of $a$, unfolded orda]
obtain $q$ where $p: p=? a * q$ and $n d v d: \neg[:-a, 1:] d v d q$ by auto
from $\langle p \neq 0\rangle[$ unfolded $p]$ have $n z: ? a \neq 0 q \neq 0$ by auto
hence deg: degree $p=$ degree ? $a+$ degree $q$ unfolding $p$
by (subst degree-mult-eq, auto)
have ord: $\bigwedge a$. order a $p=$ order $a ? a+$ order a $q$
unfolding $p$
by (subst order-mult, insert nz, auto)
have roots: $\{a$. poly pa=0\}=insert $a(\{a$. poly $q a=0\}-\{a\})$ using root
unfolding $p$ poly-mult by auto
have fin: finite $\{a$. poly $q a=0\}$ by (rule poly-roots-finite $[O F\langle q \neq 0\rangle]$ )
have Suc $n=$ order a $p$ using orda by simp
also have $\ldots=$ Suc $n+$ order a $q$ unfolding ord order-linear-power' by simp
finally have order a $q=0$ by auto
with order-root $[$ of $q a]\langle q \neq 0\rangle$ have $q a$ : poly $q a \neq 0$ by auto
have $\left(\sum a \in\{a\right.$. poly $q a=0\}-\{a\}$. order $\left.a p\right)=\left(\sum a \in\{a\right.$. poly $q a=0\}$
$-\{a\}$. order $a q)$
proof (rule sum.cong $[$ OF refl $]$ )
fix $b$
assume $b \in\{a$. poly $q a=0\}-\{a\}$
hence $b \neq a$ by auto
hence order $b$ ? $a=0$ unfolding order-linear-power' by simp
thus order $b$ p order $b q$ unfolding ord by simp
qed
also have $\ldots=\left(\sum a \in\{a\right.$. poly $q a=0\}$. order a $\left.q\right)$ using $q$ a by auto
also have $\ldots \leq$ degree $q$
by (rule $\operatorname{Suc}(1)[O F-\langle q \neq 0\rangle]$,
insert deg[unfolded degree-linear-power] Suc(2), auto)
finally have $\left(\sum a \in\{a\right.$. poly $q a=0\}-\{a\}$. order a $\left.p\right) \leq$ degree $q$.
thus ?thesis unfolding roots deg using fin
by (subst sum.insert, simp-all only: degree-linear-power, auto simp: orda)
qed auto
qed
qed
lemma order-code[code]: order ( $a::^{\prime} a::$ idom-divide) $p=$
(if $p=0$ then Code.abort (STR "order of polynomial 0 undefined") ( $\lambda$-. order a
p)
else if poly $p$ a $\neq 0$ then 0 else Suc (order $a(p$ div $[:-a, 1:]))$ )
proof (cases $p=0$ )
case False note $p=$ this
note order $=$ order $[$ OF $p]$
show ?thesis
proof (cases poly pa=0)

```
    case True
    with order-root[of pa] p obtain n where ord: order a p = Suc n
        by (cases order a p, auto)
    from this(1) have [:-a,1 :] dvd p
        using True poly-eq-0-iff-dvd by blast
    then obtain q}\mathrm{ where p:p=[:-a,1:]*q unfolding dvd-def by auto
    have ord: order a p=order a [:-a, 1:] + order a q
        using p False order-mult[of [:-a,1:] q] by auto
    have q: p div [: -a,1 :] = q using False p
        by (metis mult-zero-left nonzero-mult-div-cancel-left)
    show ?thesis unfolding ord q using False True by auto
next
    case False
    with order-root[of p a] p show ?thesis by auto
    qed
qed auto
end
```


## 3 Explicit Formulas for Roots

We provide algorithms which use the explicit formulas to compute the roots of polynomials of degree up to 2 . For polynomials of degree 3 and 4 have a look at the AFP entry "Cubic-Quartic-Equations".

```
theory Explicit-Roots
imports
    Polynomial-Interpolation.Missing-Polynomial
    Sqrt-Babylonian.Sqrt-Babylonian
begin
lemma roots 0 : assumes \(p: p \neq 0\) and \(p 0\) : degree \(p=0\)
    shows \(\{x\). poly \(p x=0\}=\{ \}\)
    using degree0-coeffs[OF p0] \(p\) by auto
definition roots \(1::\) ' \(a\) :: field poly \(\Rightarrow{ }^{\prime} a\) where
    roots1 \(p=(-\) coeff \(p 0 /\) coeff \(p 1)\)
lemma roots1: fixes \(p::\) ' \(a\) :: field poly
    assumes \(p 1\) : degree \(p=1\)
    shows \(\{x\). poly \(p x=0\}=\{\) roots1 \(p\}\)
    using degree1-coeffs[OF p1] unfolding roots1-def
    by (auto simp: add-eq-0-iff nonzero-neg-divide-eq-eq2)
lemma roots2: fixes \(p::{ }^{\prime} a\) :: field-char-0 poly
    assumes \(p 2: p=[: c, b, a:]\) and \(a: a \neq 0\)
    shows \(\{x\). poly \(p x=0\}=\left\{-(b /(2 * a))+e \mid e . e\right.\) ^2 \(=(b /(2 * a))^{\wedge} 2\)
\(-c / a\}\) (is ? \(l=? r\) )
proof -
```

```
define \(b 2 a\) where \(b 2 a=b /(2 * a)\)
\{
    fix \(x\)
    have \((x \in ? l)=(x * x * a+x * b+c=0)\) unfolding \(p 2\) by (simp add:
field-simps)
    also have \(\ldots=((x * x+2 * x * b 2 a)+c / a=0)\) using \(a\) by (auto simp:
b2a-def field-simps)
    also have \(x * x+2 * x * b 2 a=(x * x+2 * x * b 2 a+b 2 a \wedge 2)-b 2 a \wedge 2\) by
simp
    also have \(\ldots=(x+b 2 a)^{\wedge}\) 2 - \(b 2 a{ }^{\text {^2 }}\)
            by (simp add: field-simps power2-eq-square)
    also have \((\ldots+c / a=0)=\left((x+b 2 a)\right.\) へ \(\left.2=b 2 a^{\wedge} 2-c / a\right)\) by algebra
            also have \(\ldots=(x \in ? r)\) unfolding \(b 2 a-d e f[\) symmetric \(]\) by (auto simp:
field-simps)
    finally have \((x \in ? l)=(x \in ? r)\).
    \}
    thus?thesis by auto
qed
definition croots2 :: complex poly \(\Rightarrow\) complex list where
    croots2 \(p=(\) let \(a=\) coeff \(p 2 ; b=\) coeff \(p 1 ; c=\operatorname{coeff} p 0 ; b 2 a=b /(2 * a)\);
    \(b a c=b 2 a \wedge 2-c / a ;\)
    \(e=c s q r t b a c\)
    in
        remdups \([-b 2 a+e,-b 2 a-e])\)
definition complex-rat :: complex \(\Rightarrow\) bool where
    complex-rat \(x=(\) Re \(x \in \mathbb{Q} \wedge \operatorname{Im} x \in \mathbb{Q})\)
lemma croots2: assumes degree \(p=2\)
    shows \(\{x\). poly \(p x=0\}=\operatorname{set}(\operatorname{croots} 2 p)\)
proof -
    from degree2-coeffs[OF assms] obtain abc
    where \(p: p=[: c, b, a:]\) and \(a: a \neq 0\) by auto
    note main \(=\) roots \(2[O F \quad p a]\)
    have \(2: 2=S u c(S u c ~ 0)\) by \(\operatorname{simp}\)
    have coeff: coeff p \(2=a\) coeff \(p 1=b\) coeff \(p 0=c\) unfolding \(p\) by (auto simp:
2)
    let \(? b 2 a=b /(2 * a)\)
    define \(b 2 a\) where \(b 2 a=? b 2 a\)
    let ? \(b a c=b 2 a^{\wedge} 2-c / a\)
    define \(b a c\) where \(b a c=? b a c\)
    have roots: set \((\) croots2 \(p)=\{-b 2 a+c s q r t ~ b a c,-b 2 a-c s q r t b a c\}\)
        unfolding croots2-def Let-def coeff b2a-def[symmetric] bac-def[symmetric]
        by (auto split: if-splits)
    show ?thesis unfolding roots main b2a-def[symmetric] bac-def[symmetric]
        using power2-eq-iff by fastforce
qed
```

definition rroots2 :: real poly $\Rightarrow$ real list where

```
    rroots2 \(p=(\) let \(a=\) coeff \(p\) 2; \(b=\) coeff \(p 1 ; c=\operatorname{coeff} p 0 ; b 2 a=b /(2 * a)\);
        \(b a c=b 2 a \wedge 2-c / a\)
    in if bac \(=0\) then \([-b 2 a]\) else if bac \(<0\) then []
        else let \(e=\) sqrt bac
        in
        \([-b 2 a+e,-b 2 a-e])\)
```

definition rat-roots2 :: rat poly $\Rightarrow$ rat list where
rat-roots2 $p=($ let $a=$ coeff $p 2 ; b=$ coeff $p 1 ; c=\operatorname{coeff} p 0 ; b 2 a=b /(2 * a)$;
$b a c=b 2 a \wedge 2-c / a$
in map $(\lambda e .-b 2 a+e)(s q r t-r a t ~ b a c))$
lemma rroots2: assumes degree $p=2$
shows $\{x$. poly $p x=0\}=\operatorname{set}($ rroots $2 p)$
proof -
from degree2-coeffs[OF assms] obtain $a b c$
where $p: p=[: c, b, a:]$ and $a: a \neq 0$ by auto
note main $=$ roots2[OF $\quad$ a $a$ ]
have 2: $2=$ Suc (Suc 0) by simp
have coeff: coeff p $2=a$ coeff $p 1=b$ coeff $p 0=c$ unfolding $p$ by (auto simp:
2)
let $? b 2 a=b /(2 * a)$
define $b 2 a$ where $b 2 a=? b 2 a$
let $? b a c=b 2 a \wedge 2-c / a$
define $b a c$ where $b a c=? b a c$
have roots: set (rroots2 $p$ ) $=($ if bac $<0$ then $\{ \}$ else $\{-b 2 a+$ sqrt bac, $-b 2 a$

- sqrt bac\})
unfolding rroots2-def Let-def coeff b2a-def[symmetric] bac-def[symmetric]
by (auto split: if-splits)
show ?thesis unfolding roots main b2a-def[symmetric] bac-def[symmetric]
by auto
qed
lemma rat-roots2: assumes degree $p=2$
shows $\{x$. poly $p x=0\}=\operatorname{set}($ rat-roots $2 p)$
proof -
from degree2-coeffs[OF assms] obtain abc
where $p: p=[: c, b, a:]$ and $a: a \neq 0$ by auto
note main $=$ roots $2[O F \quad p a]$
have 2: $2=$ Suc (Suc 0) by simp
have coeff: coeff p $2=a$ coeff $p 1=b$ coeff $p 0=c$ unfolding $p$ by (auto simp:

2) let $? b 2 a=b /(2 * a)$
define $b 2 a$ where $b 2 a=? b 2 a$
let $? b a c=b 2 a$ ヘ2 $-c / a$
define $b a c$ where $b a c=? b a c$
have roots: $($ rat-roots2 $p)=(\operatorname{map}(\lambda e .-b 2 a+e)(s q r t-r a t ~ b a c))$
unfolding rat-roots2-def Let-def coeff b2a-def[symmetric] bac-def[symmetric]
show ?thesis unfolding roots main b2a-def[symmetric] bac-def[symmetric]
by (auto simp: power2-eq-square)
qed

Determinining roots of complex polynomials of degree up to 2 .
definition croots :: complex poly $\Rightarrow$ complex list where
croots $p=($ if $p=0 \vee$ degree $p>2$ then []
else (if degree $p=0$ then [] else if degree $p=1$ then [roots1 $p]$
else croots2 $p$ ))
lemma croots: assumes $p \neq 0$ degree $p \leq 2$
shows set $($ croots $p)=\{x$. poly $p x=0\}$
using assms unfolding croots-def
using roots $0[$ of $p]$ roots $1[$ of $p]$ croots $2[$ of $p]$
by (auto split: if-splits)
Determinining roots of real polynomials of degree up to 2 .
definition rroots $::$ real poly $\Rightarrow$ real list where
rroots $p=$ (if $p=0 \vee$ degree $p>2$ then []
else (if degree $p=0$ then [] else if degree $p=1$ then [roots1 $p$ ]
else rroots2 $p$ ))
lemma rroots: assumes $p \neq 0$ degree $p \leq 2$
shows set (rroots $p$ ) $=\{$ x. poly $p x=0\}$
using assms unfolding rroots-def
using roots $0[$ of $p$ ] roots1 [of $p$ ] rroots2[ of $p$ ]
by (auto split: if-splits)
end

## 4 Division of Polynomials over Integers

This theory contains an algorithm to efficiently compute divisibility of two integer polynomials.

```
theory Dvd-Int-Poly
imports
    Polynomial-Interpolation.Ring-Hom-Poly
    Polynomial-Interpolation.Divmod-Int
    Polynomial-Interpolation.Is-Rat-To-Rat
begin
definition div-int-poly-step :: int poly }=>\mathrm{ int }=>\mathrm{ (int poly }\times\mathrm{ int poly) option }=>\mathrm{ (int
poly }\times\mathrm{ int poly) option where
    div-int-poly-step q}=(\lambdaa\mathrm{ sro. case sro of Some (s,r) #
        let ar = pCons a r; (b,m)= divmod-int (coeff ar (degree q)) (coeff q (degree
q))
    in if m = 0 then Some (pCons b s, ar - smult b q) else None |None }=>\mathrm{ None)
```

```
declare div-int-poly-step-def[code-unfold]
definition div-mod-int-poly :: int poly }=>\mathrm{ int poly }=>\mathrm{ (int poly }\times\mathrm{ int poly) option
where
    div-mod-int-poly p q=(if q=0 then None
    else (let n= degree q;qn= coeff qn
    in fold-coeffs (div-int-poly-step q) p(Some (0,0))))
definition div-int-poly :: int poly }=>\mathrm{ int poly }=>\mathrm{ int poly option where
    div-int-poly p q=
    (case div-mod-int-poly p q of None }=>\mathrm{ None |Some (d,m) = if m=0 then
Some d else None)
definition div-rat-poly-step :: 'a::field poly }=>\mp@subsup{'}{}{\prime}a=>\mp@subsup{|}{}{\prime}a poly x 'a poly = 'a poly x
'a poly where
    div-rat-poly-step q = (\lambdaa (s,r).
        let b = coeff (pCons a r) (degree q) / coeff q (degree q)
        in (pCons b s, pCons a r - smult b q))
lemma foldr-cong-plus:
```



```
    and f'-inj: \ab. f' a= f' b\Longrightarrowa=b
    and f-bit-sur : \bigwedge abc. f' a=fbc\Longrightarrow\exists c'.c= f' c'
    and lst-in-s : set lst \subseteqs
    shows f}\mp@subsup{f}{}{\prime}a=\mathrm{ foldr flst (f'b) ఋ g' a = foldr g lst ( }\mp@subsup{g}{}{\prime}b
using lst-in-s
proof (induct lst arbitrary:a)
    case (Cons x xs)
    have prems: f}\mp@subsup{f}{}{\prime}a=(fx\circfoldr fxs)(f)b) using Cons.prems unfolding
foldr-Cons by auto
    hence \exists}\mp@subsup{c}{}{\prime}.\mp@subsup{f}{}{\prime}\mp@subsup{c}{}{\prime}=\mathrm{ foldr fxs ( f' b) using f-bit-sur by fastforce
    then obtain }\mp@subsup{c}{}{\prime}\mathrm{ where }\mp@subsup{c}{}{\prime}\mathrm{ -def: f}\mp@subsup{f}{}{\prime}\mp@subsup{c}{}{\prime}=\mathrm{ foldr }fxs(\mp@subsup{f}{}{\prime}b)\mathrm{ by blast
    hence f}\mp@subsup{f}{}{\prime}a=fx(\mp@subsup{f}{}{\prime}\mp@subsup{c}{}{\prime})\mathrm{ using prems by simp
    hence g' a = gx ( (g}\mp@subsup{g}{}{\prime})\mathrm{ using f-is-g Cons.prems(2) by simp
    also have g}\mp@subsup{g}{}{\prime}\mp@subsup{c}{}{\prime}=\mathrm{ foldr g xs ( }\mp@subsup{g}{}{\prime}b)\mathrm{ using Cons.hyps[of c ] c'-def Cons.prems(2)
by auto
    finally have g' a = (gx\circ foldr g xs) ( g'b) by simp
    thus ?case using foldr-Cons by simp
qed (insert f'-inj, auto)
abbreviation (input) rp :: int poly }=>\mathrm{ rat poly where
    rp \equiv map-poly rat-of-int
lemma rat-int-poly-step-agree :
    assumes coeff (pCons b c\mathcal{L})(\mathrm{ degree q) mod coeff q (degree q) = 0}00
    shows (rp a1,rp a2) = (div-rat-poly-step (rp q) ○ rat-of-int)b (rp c1,rp c\mathcal{Z})
        Some (a1,a2) = div-int-poly-step qb (Some (c1,c2))
```

```
proof -
    have coeffs: coeff (pCons b c\mathcal{L})(\mathrm{ degree q) mod coeff q (degree q) = 0 using}
assms by auto
    let ?ri = rat-of-int
    let ?withDiv1 = pCons (?ri (coeff (pCons b c\mathcal{L})(degree q) div coeff q (degree
q))) (rp c1)
    let ?withSls1 = pCons (coeff (pCons (?ri b) (rp c2)) (degree q) / coeff (rp q)
(degree q)) (rp c1)
    let ?ident1 = ?withDiv1 = ?withSls1
    let ?withDiv2 = rp (pCons b c\mathcal{Z - smult (coeff (pCons b c\mathcal{Z) (degree q) div coeff}}\mathbf{~}\mathrm{ (d)}
q(degree q)) q)
    let ?withSls2 = pCons (?ri b) (rp c2) - smult (coeff (pCons (?ri b) (rp c2))
(degree q) / coeff (rpq) (degree q)) (rp q)
    let ?ident2 = ?withDiv2 = ?withSls2
    note simps = div-int-poly-step-def option.simps Let-def prod.simps
    have id1:?ri (coeff (pCons b c2) (degree q) div coeff q (degree q)) =
                    ?ri (coeff (pCons b c\mathcal{L})(degree q)) / ?ri (coeff q (degree q)) using coeffs
by auto
    have id2:?ident1 unfolding id1
            by (simp, fold of-int-hom.coeff-map-poly-hom of-int-hom.map-poly-pCons-hom,
simp)
    hence id3:?ident2 using id2 by (auto simp: hom-distribs)
    have c1:((rp (pCons (coeff (pCons b c2) (degree q) div coeff q (degree q)) c1)
                        ,rp (pCons b c2 - smult (coeff (pCons b c2) (degree q) div coeff q (degree
q)) q))
                = div-rat-poly-step (rp q) (?ri b) (rp c1,rp c2)) \longleftrightarrow(?ident1 ^ ?ident2)
        unfolding div-rat-poly-step-def simps
        by (simp add: hom-distribs)
    have ((rpa1, rp a2) = (div-rat-poly-step (rp q) ○ rat-of-int) b (rp c1,rp c\mathcal{L}))
\longleftrightarrow
    (rp a1 = ?withSls1 ^rp a2 = ?withSls2)
        unfolding div-rat-poly-step-def simps by simp
    also have ... \longleftrightarrow
                ((a1 = pCons (coeff (pCons b c\mathcal{L})(\mathrm{ degree q) div coeff q (degree q)) c1) ^}
                (a2 = pCons b c\mathcal{L}
q)) q))
    by (fold id2 id3 of-int-hom.map-poly-pCons-hom, unfold of-int-poly-hom.eq-iff,
auto)
    also have c0:...\longleftrightarrowSome (a1,a2) = div-int-poly-step q b (Some (c1,c2))
        unfolding divmod-int-def div-int-poly-step-def option.simps Let-def prod.simps
        using coeffs by (auto split: option.splits prod.splits if-splits)
    finally show ?thesis .
qed
lemma int-step-then-rat-poly-step :
    assumes Some:Some (a1,a2) = div-int-poly-step q b (Some (c1,c2))
    shows (rp a1,rp a2) = (div-rat-poly-step (rp q) ○ rat-of-int) b (rp c1,rp c2)
proof -
```

note simps $=$ div-int-poly-step-def option.simps Let-def divmod-int-def prod.simps
from Some[unfolded simps] have mod0: coeff ( $p$ Cons $b$ c2) (degree q) mod coeff $q($ degree $q)=0$
by (auto split: option.splits prod.splits if-splits)
thus ?thesis using assms rat-int-poly-step-agree by auto
qed
lemma is-int-rat-division :
assumes $y \neq 0$
shows is-int-rat (rat-of-int $x /$ rat-of-int $y) \longleftrightarrow x \bmod y=0$
proof
assume is-int-rat (rat-of-int x / rat-of-int y)
then obtain $v$ where $v$-def:rat-of-int $v=$ rat-of-int $x /$ rat-of-int $y$
using int-of-rat(2) is-int-rat by fastforce
hence $v=\lfloor$ rat-of-int $x /$ rat-of-int $y\rfloor$ by linarith
hence $v * y=x-x$ mod $y$ using div-is-floor-divide-rat mod-div-equality-int by simp
hence rat-of-int $v *$ rat-of-int $y=r a t-o f-i n t x-r a t-o f-i n t(x \bmod y)$
by (fold hom-distribs, unfold of-int-hom.eq-iff)
hence (rat-of-int $x /$ rat-of-int $y) *$ rat-of-int $y=$ rat-of-int $x$-rat-of-int ( $x$ mod y)
using $v$-def by simp
hence rat-of-int $x=$ rat-of-int $x$-rat-of-int ( $x \bmod y$ ) by (simp add: assms)
thus $x$ mod $y=0$ by $\operatorname{simp}$
qed (force)
lemma $p$ Cons-of-rp-contains-ints :
assumes $r p a=p$ Cons $b c$
shows is-int-rat b
proof -
have $\bigwedge b n$. rp $a=b \Longrightarrow$ is-int-rat (coeff $b n$ ) by auto
hence $r p a=p$ Cons $b c \Longrightarrow$ is-int-rat (coeff (pCons bc) 0).
thus ?thesis using assms by auto
qed
lemma rat-step-then-int-poly-step :
assumes $q \neq 0$
and $(r p a 1, r p a 2)=($ div-rat-poly-step $(r p q) \circ r a t-o f-i n t) b 2(r p c 1, r p c 2)$
shows Some $(a 1, a 2)=$ div-int-poly-step q b2 $($ Some $(c 1, c 2))$
proof -
let ? mustbeint $=$ rat-of-int $($ coeff $(p$ Cons b2 $c \mathcal{L})($ degree $q)) /$ rat-of-int $($ coeff $q$
(degree q))
let ?mustbeint2 $=$ coeff $(p$ Cons (rat-of-int b2) $(r p c \mathcal{Z}))($ degree $(r p q))$
/ coeff (rp q) (degree (rp q))
have mustbeint : ? mustbeint = ? mustbeint2 by (fold hom-distribs of-int-hom.coeff-map-poly-hom, $\operatorname{simp}$ )
note simps $=$ div-int-poly-step-def option.simps Let-def divmod-int-def prod.simps
from assms leading-coeff-neq- $0[$ of $q]$ have $q 0$ :coeff $q($ degree $q) \neq 0$ by simp

```
    have rp a1 = pCons ?mustbeint2 (rp c1)
    using assms(2) unfolding div-rat-poly-step-def by (simp add:div-int-poly-step-def
Let-def)
    hence is-int-rat ?mustbeint2
        unfolding div-rat-poly-step-def using pCons-of-rp-contains-ints by simp
    hence is-int-rat ?mustbeint unfolding mustbeint by simp
    hence coeff (pCons b2 c2) (degree q) mod coeff q (degree q) = 0
        using is-int-rat-division q0 by simp
    thus ?thesis using rat-int-poly-step-agree assms by simp
qed
lemma div-int-poly-step-surjective : Some a = div-int-poly-step q b c \Longrightarrow\exists c'.c
=Some c'
    unfolding div-int-poly-step-def by(cases c, simp-all)
lemma div-mod-int-poly-then-pdivmod:
    assumes div-mod-int-poly p q = Some (r,m)
    shows (rpp div rpq, rp p mod rpq) = (rpr,rpm)
        and q}=
proof -
    let ?rpp = (\lambda (a,b). (rp a,rp b))
    let ?p = rp p
    let ?q = rp q
    let ?r = rp r
    let ?m}=rp
    let ?div-rat-step = div-rat-poly-step ?q
    let ?div-int-step = div-int-poly-step q
    from assms show q0:q\not=0 using div-mod-int-poly-def by auto
    hence div-mod-int-poly p q=Some (r,m)\longleftrightarrowSome (r,m)=foldr (div-int-poly-step
q) (coeffs p)(Some (0,0))
    unfolding div-mod-int-poly-def fold-coeffs-def by (auto split: option.splits prod.splits
if-splits)
    hence innerRes: Some (r,m) = foldr (?div-int-step) (coeffs p) (Some (0, 0))
using assms by simp
    { fix oldRes res :: int poly }\times\mathrm{ int poly
    fix lst :: int list
    have Some res = foldr ?div-int-step lst (Some oldRes) \Longrightarrow
            ?rpp res = foldr (?div-rat-step o rat-of-int) lst (?rpp oldRes)
        using foldr-cong-plus[of set lst Some?div-int-step ?rpp ?div-rat-step o rat-of-int
            lst res oldRes] int-step-then-rat-poly-step div-int-poly-step-surjective by auto
    hence Some res = foldr ?div-int-step lst (Some oldRes)
                \Longrightarrow \text { ?rpp res = foldr ?div-rat-step (map rat-of-int lst) (?rpp oldRes)}
                using foldr-map[of ?div-rat-step rat-of-int lst] by simp
    }
    hence equal-foldr: Some (r,m)= foldr (?div-int-step) (coeffs p) (Some (0,0))
    \Longrightarrow?rpp (r,m) = foldr (?div-rat-step) (map rat-of-int (coeffs p)) (?rpp (0,0)).
    have (map rat-of-int (coeffs p) = coeffs ?p) by simp
    hence (?r,?m)=(foldr (?div-rat-step) (coeffs ?p) (0,0)) using equal-foldr in-
```

```
nerRes by simp
    thus \((? p\) div ?q, ?p mod ?q) \(=(? r, ? m)\)
    using fold-coeffs-def [of ?div-rat-step ? \(p\) ] \(q 0\)
            div-mod-fold-coeffs [of ?p ?q]
    unfolding div-rat-poly-step-def by auto
qed
lemma div-rat-poly-step-sur:
    assumes \((\) case \(a\) of \((a, b) \Rightarrow(r p a, r p b))=(\) div-rat-poly-step \((r p q) \circ r a t-o f-i n t)\)
\(x\) pair
    shows \(\exists c^{\prime}\). pair \(=\left(\right.\) case \(c^{\prime}\) of \(\left.(a, b) \Rightarrow(r p a, r p b)\right)\)
proof -
    obtain b1 b2 where pair: pair \(=(b 1\), b2) by (cases pair) simp
    define \(p 12\) where \(p 12=\) coeff \((p\) Cons (rat-of-int \(x)\) b2) \((\) degree \((r p q)) /\) coeff
\((r p q)(\) degree \((r p q))\)
    obtain a1 a2 where \(a=(a 1, a 2)\) by (cases a) simp
    with assms pair have (rp a1, rp a2) = div-rat-poly-step (rp q) (rat-of-int \(x)(b 1\),
b2)
    by \(\operatorname{simp}\)
    then have a1: rp a1 = pCons p12 b1
    and \(r p a 2=p C o n s(r a t-o f-i n t x) b 2-s m u l t ~ p 12(r p q)\)
    by (auto split: prod.splits simp add: Let-def div-rat-poly-step-def p12-def)
    then obtain p21 p22 where rp p21 \(=\) pCons p22 b2
        apply (simp add: field-simps)
    apply (metis coeff-pCons-0 of-int-hom.map-poly-hom-add of-int-hom.map-poly-hom-smult
of-int-hom.coeff-map-poly-hom)
    done
    moreover obtain \(p 21\) ' \(p 21 q\) where \(p 21=p\) Cons \(p 21\) ' \(p 21 q\)
    by (rule pCons-cases)
    ultimately obtain \(p 2\) where \(b 2=r p p 2\)
        by (auto simp: hom-distribs)
    moreover obtain \(a 1^{\prime} a 1 q\) where \(a 1=p\) Cons \(a 1^{\prime} a 1 q\)
        by (rule pCons-cases)
    with a1 obtain \(p 1\) where \(b 1=r p p 1\)
    by (auto simp: hom-distribs)
    ultimately have pair \(=(r p p 1, r p p 2)\) using pair by simp
    then show ?thesis by auto
qed
lemma pdivmod-then-div-mod-int-poly:
    assumes \(q 0: q \neq 0\) and \((r p p\) div \(r p q, r p p \bmod r p q)=(r p r, r p m)\)
    shows div-mod-int-poly p \(q=\) Some ( \(r, m\) )
proof -
    let ? \(r p p=(\lambda(a, b) .(r p a, r p b))\)
    let \(? p=r p p\)
    let ? \(q=r p q\)
    let \(? r=r p r\)
    let \(? m=r p m\)
    let ?div-rat-step \(=\) div-rat-poly-step ?q
```

let ?div-int-step $=$ div-int-poly-step $q$
\{ fix oldRes res :: int poly $\times$ int poly
fix lst :: int list
have $i n j:(\bigwedge a b$. (case $a$ of $(a, b) \Rightarrow(r p a, r p b))=($ case $b$ of $(a, b) \Rightarrow(r p a$, $r p b)) \Longrightarrow a=b$ )
by auto
have $(\bigwedge a b c . b \in$ set lst $\Longrightarrow$
$($ case $a$ of $(a, b) \Rightarrow($ map-poly rat-of-int $a$, map-poly rat-of-int b) $)=$ (div-rat-poly-step (map-poly rat-of-int $q$ ) ○ rat-of-int) $b$ (case $c$ of $(a, b) \Rightarrow($ map-poly rat-of-int $a$, map-poly rat-of-int $b)) \Longrightarrow$ Some $a=$ div-int-poly-step $q b($ Some $c))$
using rat-step-then-int-poly-step $[O F ~ q 0]$ by auto
hence ?rpp res $=$ foldr (?div-rat-step $\circ$ rat-of-int) lst (?rpp oldRes)
$\Longrightarrow$ Some res $=$ foldr ? div-int-step lst (Some oldRes)
using foldr-cong-plus[of set lst ?rpp ?div-rat-step $\circ$ rat-of-int Some ?div-int-step lst res oldRes]
div-rat-poly-step-sur inj by simp
hence ?rpp res $=$ foldr ? div-rat-step (map rat-of-int lst) (?rpp oldRes)
$\Longrightarrow$ Some res $=$ foldr ? div-int-step lst (Some oldRes)
using foldr-map[of ? div-rat-step rat-of-int lst] by auto
\}
hence equal-foldr : ?rpp $(r, m)=$ foldr (?div-rat-step) (map rat-of-int (coeffs p))
(? ? $\quad$ pp $(0,0)$ )
$\Longrightarrow$ Some $(r, m)=$ foldr $($ ?div-int-step $)($ coeffs $p)($ Some $(0,0))$
by $\operatorname{simp}$
have $(? r, ? m)=($ foldr $($ ? div-rat-step $)($ coeffs ? $p)(0,0))$
using fold-coeffs-def[of ?div-rat-step ?p] assms
div-mod-fold-coeffs [of ?p ? $q$ ]
unfolding div-rat-poly-step-def by auto
hence Some ( $r, m$ ) = foldr (?div-int-step) (coeffs p) (Some ( 0,0 ))
using equal-foldr by simp
thus ?thesis using $q 0$ unfolding div-mod-int-poly-def by (simp add: fold-coeffs-def) qed
lemma div-int-then-rqp:
assumes div-int-poly $p q=$ Some $r$
shows $r * q=p$
and $q \neq 0$
proof -
let $? r p p=(\lambda(a, b) .(r p a, r p b))$
let $? p=r p p$
let $? q=r p q$
let $? r=r p r$
have Some ( $r, 0$ ) $=$ div-mod-int-poly $p$ using assms unfolding div-int-poly-def
by (auto split: option.splits prod.splits if-splits)
with div-mod-int-poly-then-pdivmod[of $p$ q r 0]
have ?p div ? $q=? r \wedge$ ? $p \bmod ? q=0$ by simp
with div-mult-mod-eq[of ?p ?q]
have ? $p=? r *$ ? $q$ by auto

```
    also have ... =rp (r*q) by (simp add: hom-distribs)
    finally have ? p = rp (r*q).
    thus r*q=p by simp
    show q}\not=0\mathrm{ using assms unfolding div-int-poly-def div-mod-int-poly-def
        by (auto split: option.splits prod.splits if-splits)
qed
lemma rqp-then-div-int:
    assumes r*q=p
        and q0:q\not=0
    shows div-int-poly p q = Some r
proof -
    let ?rpp = (\lambda (a,b). (rpa,rpb))
    let ?p = rp p
    let ?q = rpq
    let ?r = rp r
    have ?p = ?r * ?q using assms(1) by (auto simp: hom-distribs)
    hence ?p div ?q=?r and ?p mod ?q=0
        using q0 by simp-all
    hence (rp p div rp q, rp p mod rp q) = (rp r,0) by (auto split: prod.splits)
    hence (rp p div rp q, rp p mod rpq) = (rpr,rp 0) by simp
    hence Some (r,0) = div-mod-int-poly p q
        using pdivmod-then-div-mod-int-poly[OF q0,of p r 0] by simp
    thus ?thesis unfolding div-mod-int-poly-def div-int-poly-def using q0
        by (metis (mono-tags, lifting) option.simps(5) split-conv)
qed
lemma div-int-poly:(div-int-poly p q=Some r)\longleftrightarrow(q\not=0^p=r*q)
    using div-int-then-rqp rqp-then-div-int by blast
definition dvd-int-poly :: int poly }=>\mathrm{ int poly }=>\mathrm{ bool where
    dvd-int-poly q p = (if q=0 then p=0 else div-int-poly p q\not= None)
lemma dvd-int-poly[simp]:dvd-int-poly q p = ( q dvd p)
    unfolding dvd-def dvd-int-poly-def using div-int-poly[of p q]
    by (cases q = 0, auto)
definition dvd-int-poly-non-0 :: int poly }=>\mathrm{ int poly }=>\mathrm{ bool where
    dvd-int-poly-non-0 q p = (div-int-poly p q\not=None)
lemma dvd-int-poly-non-0[simp]: q}=0\Longrightarrowdvd-int-poly-non-0 q p = ( q dvd p
    unfolding dvd-def dvd-int-poly-non-0-def using div-int-poly[of p q] by auto
lemma [code-unfold]: p dvd q\longleftrightarrow dvd-int-poly p q by simp
hide-const rp
end
```


## 5 More on Polynomials

This theory contains several results on content, gcd, primitive part, etc.. Moreover, there is a slightly improved code-equation for computing the gcd.

```
theory Missing-Polynomial-Factorial
    imports HOL-Computational-Algebra.Polynomial-Factorial
        Polynomial-Interpolation.Missing-Polynomial
begin
```

Improved code equation for gcd-poly-code which avoids computing the content twice.
lemma gcd-poly-code-code[code]: gcd-poly-code p $q=$
(if $p=0$ then normalize $q$ else if $q=0$ then normalize $p$ else let
$c 1=$ content $p ;$
$c 2=$ content $q ;$
$p^{\prime}=$ map-poly $(\lambda x . x$ div c1) $p ;$
$q^{\prime}=$ map-poly $(\lambda x . x$ div c2) $q$
in smult (gcd c1 c2) (gcd-poly-code-aux $\left.p^{\prime} q^{\prime}\right)$ )
unfolding gcd-poly-code-def Let-def primitive-part-def by simp
lemma gcd-smult: fixes $f g::{ }^{\prime} a$ :: \{factorial-ring-gcd,semiring-gcd-mult-normalize\}
poly
defines $c f: c f \equiv$ content $f$
and $c g: c g \equiv$ content $g$
shows $g c d$ (smult $a f) g=($ if $a=0 \vee f=0$ then normalize $g$ else
smult $(g c d$ a $(c g \operatorname{div}(g c d c f c g)))(g c d f g))$
proof (cases $a=0 \vee f=0)$
case False
let ?c $=$ content
let $? p p=$ primitive-part
let ? $u a=$ unit-factor $a$
let ? $n a=$ normalize $a$
define $H$ where $H=\operatorname{gcd}(? c f)(? c \quad g)$
have $H$ dvd ?c $f$ unfolding $H$-def by auto
then obtain $F$ where $f h$ : ?c $f=H * F$ unfolding dvd-def by blast
from False have $c f 0:$ ? $c f \neq 0$ by auto
hence $H: H \neq 0$ unfolding $H$-def by auto
from arg-cong[OF fh, of $\lambda f . f$ div $H] H$ have $F: F=$ ?c $f$ div $H$ by auto
have $H$ dvd ?c $g$ unfolding $H$-def by auto
then obtain $G$ where $g h$ : ?c $g=H * G$ unfolding dvd-def by blast
from arg-cong[OF gh, of $\lambda f$. $f$ div $H] H$ have $G: G=$ ?c $g$ div $H$ by auto
have coprime $F G$ using $H$ unfolding $F G H$-def
using cf0 div-gcd-coprime by blast
have is-unit ?ua using False by simp
then have ua: is-unit [: ?ua :]
by (simp add: is-unit-const-poly-iff)
have $g c d($ smult a f) $g=\operatorname{smult}(g c d(? n a * ? c f)(? c g))$
(gcd (smult ?ua (?pp f)) (?pp g))
unfolding gcd-poly-decompose[of smult a $f$ ]
content-smult primitive-part-smult by simp
also have smult ?ua (?ppf) $=$ ?pp $f$ * [: ?ua :] by simp
also have $g c d \ldots(? p p g)=g c d(? p p f)(? p p g)$
unfolding gcd-mult-unit1 [OF ua] ..
also have $g c d(? n a * ? c f)(? c g)=g c d((? n a * F) * H)(G * H)$
unfolding fh gh by (simp add: ac-simps)
also have $\ldots=\operatorname{gcd}($ ?na $* F) G *$ normalize $H$ unfolding gcd-mult-right gcd.commute $[$ of $G]$
by (simp add: normalize-mult)
also have normalize $H=H$ by (metis $H$-def normalize-gcd)
finally
have $g c d($ smult $a f) g=\operatorname{smult}(g c d(? n a * F) G)(s m u l t \quad H(g c d(? p p f)(? p p$ $g))$ ) by $\operatorname{simp}$
also have smult $H(g c d(? p p f)(? p p g))=g c d f g$ unfolding $H$-def
by (rule gcd-poly-decompose[symmetric])
also have $g c d(? n a * F) G=\operatorname{gcd}(F * ? n a) G$ by (simp add: ac-simps)
also have $\ldots=\operatorname{lcd}$ ? $n a G$
using 〈coprime $F G$ by (simp add: gcd-mult-right-left-cancel ac-simps)
finally show ?thesis unfolding $G H$-def $c g$ cf using False by simp
next
case True
hence $g c d$ (smult af) $g=$ normalize $g$ by (cases $a=0$, auto)
thus? ?hesis using True by simp
qed
lemma gcd-smult-ex: assumes $a \neq 0$
shows $\exists b$. gcd (smult af) $g=$ smult $b(g c d f g) \wedge b \neq 0$
proof (cases $f=0$ )
case True
thus ?thesis by (intro exI[of-1], auto)
next
case False
hence $i d:(a=0 \vee f=0)=$ False using assms by auto
show ?thesis unfolding gcd-smult id if-False
by (intro exI conjI, rule refl, insert assms, auto)
qed
lemma primitive-part-idemp[simp]:
fixes $f::$ ' $a$ :: \{semiring-gcd,normalization-semidom-multiplicative $\}$ poly
shows primitive-part (primitive-part $f$ ) $=$ primitive-part $f$
by (metis content-primitive-part[of f] primitive-part-eq-0-iff primitive-part-prim)
lemma content-gcd-primitive:
$f \neq 0 \Longrightarrow$ content $(g c d($ primitive-part $f) g)=1$
$f \neq 0 \Longrightarrow$ content $(g c d$ (primitive-part $f)($ primitive-part $g))=1$
by (metis (no-types, lifting) content-dvd-contentI content-primitive-part gcd-dvd1
is-unit-content-iff)+
lemma content-gcd-content: content $(g c d f g)=g c d($ content $f)($ content $g)$

```
    (is ?l = ?r)
proof -
    let ?c = content
    have ?l = normalize (gcd (?c f) (?c g)) *
        ?c (gcd (primitive-part f) (primitive-part g))
        unfolding gcd-poly-decompose[of f g] content-smult ..
    also have ...=gcd (?c f) (?c g) *
        ?c (gcd (primitive-part f) (primitive-part g)) by simp
    also have ... = ?r using content-gcd-primitive[of f g]
        by (metis (no-types, lifting) content-dvd-contentI content-eq-zero-iff
        content-primitive-part gcd-dvd2 gcd-eq-0-iff is-unit-content-iff mult-cancel-left1)
    finally show ?thesis.
qed
lemma gcd-primitive-part:
    gcd (primitive-part f) (primitive-part g) = normalize (primitive-part (gcd fg))
    proof(cases f=0)
    case True
    show ?thesis unfolding gcd-poly-decompose[of f g] gcd-0-left primitive-part-0
True
        by (simp add: associatedI primitive-part-dvd-primitive-partI)
    next
    case False
    have normalize 1 = normalize (unit-factor (gcd (content f) (content g)))
        by (simp add: False)
    then show ?thesis unfolding gcd-poly-decompose[of f g]
    by (metis (no-types) Polynomial.normalize-smult content-gcd-primitive(1)[OF
False] content-times-primitive-part normalize-gcd primitive-part-smult)
qed
lemma primitive-part-gcd: primitive-part (gcd f g)
    = unit-factor (gcd f g)* gcd (primitive-part f) (primitive-part g)
    unfolding gcd-primitive-part
    by (metis (no-types, lifting)
    content-times-primitive-part gcd.normalize-idem mult-cancel-left2 mult-smult-left
    normalize-eq-0-iff normalize-mult-unit-factor primitive-part-eq-0-iff
    smult-content-normalize-primitive-part unit-factor-mult-normalize)
lemma primitive-part-normalize:
    fixes f :: 'a :: {semiring-gcd,idom-divide,normalization-semidom-multiplicative}
poly
    shows primitive-part (normalize f) = normalize (primitive-part f)
proof (cases f=0)
    case True
    thus ?thesis by simp
next
    case False
    have normalize (content (normalize (primitive-part f))) = 1
        using content-primitive-part[OF False] content-dvd content-const
```

content-dvd-contentI dvd-normalize-iff is-unit-content-iff by (metis (no-types))
then have content (normalize (primitive-part f)) = 1 by fastforce
then have content (normalize $f$ ) $=1 *$ content $f$
by (metis (no-types) content-smult mult.commute normalize-content smult-content-normalize-primitive-part)
then have content $f=$ content (normalize $f$ )
by simp
then show?thesis unfolding smult-content-normalize-primitive-part[of f,symmetric]
by (metis (no-types) False content-times-primitive-part mult.commute mult-cancel-left mult-smult-right smult-content-normalize-primitive-part)
qed
lemma length-coeffs-primitive-part[simp]: length $($ coeffs $($ primitive-part $f))=$ length
(coeffs f)
proof (cases $f=0$ )
case False
hence length $($ coeffs $f) \neq 0$ length (coeffs (primitive-part $f)) \neq 0$ by auto
thus ?thesis using degree-primitive-part[of $f$, unfolded degree-eq-length-coeffs] by linarith
qed $\operatorname{simp}$
lemma degree-unit-factor $[$ simp $]$ : degree (unit-factor $f)=0$
by (simp add: monom-0 unit-factor-poly-def)
lemma degree-normalize[simp]: degree (normalize $f)=$ degree $f$
proof (cases $f=0$ )
case False
have degree $f=$ degree (unit-factor $f *$ normalize $f$ ) by simp
also have $\ldots=$ degree (unit-factor $f$ ) + degree (normalize $f$ )
by (rule degree-mult-eq, insert False, auto)
finally show ?thesis by simp
qed $\operatorname{simp}$
lemma content-iff: $x$ dvd content $p \longleftrightarrow(\forall c \in$ set (coeffs $p) . x d v d c)$
by (simp add: content-def dvd-gcd-list-iff)
lemma is-unit-field-poly[simp]: $\left(p::{ }^{\prime} a::\right.$ field poly) $d v d 1 \longleftrightarrow p \neq 0 \wedge$ degree $p=0$
proof (intro iffI conjI, unfold conj-imp-eq-imp-imp)
assume $i s$-unit $p$
then obtain $q$ where $*: p * q=1$ by (elim dvdE, auto)
from $*$ show $p 0: p \neq 0$ by auto
from $*$ have $q 0: q \neq 0$ by auto
from $*$ degree-mult-eq[OF p0 q0]
show degree $p=0$ by auto
next
assume degree $p=0$
from degree 0-coeffs[OF this]
obtain $c$ where $c: p=[: c:]$ by auto
assume $p \neq 0$
with $c$ have $c \neq 0$ by auto
with $c$ have $1=p *[: 1 / c:]$ by auto
from $d v d I[O F$ this] show is-unit $p$.
qed
definition primitive where
primitive $f \longleftrightarrow(\forall x .(\forall y \in \operatorname{set}(\operatorname{coeffs} f) . x d v d y) \longrightarrow x d v d 1)$
lemma primitiveI:
assumes $(\bigwedge x .(\bigwedge y . y \in \operatorname{set}($ coeffs $f) \Longrightarrow x d v d y) \Longrightarrow x d v d 1)$
shows primitive $f$ by (insert assms, auto simp: primitive-def)
lemma primitiveD:
assumes primitive $f$
shows $(\bigwedge y . y \in \operatorname{set}($ coeffs $f) \Longrightarrow x d v d y) \Longrightarrow x d v d 1$
by (insert assms, auto simp: primitive-def)
lemma not-primitiveE:
assumes $\neg$ primitive $f$
and $\bigwedge x .(\bigwedge y . y \in \operatorname{set}($ coeffs $f) \Longrightarrow x d v d y) \Longrightarrow \neg x$ dvd $1 \Longrightarrow$ thesis
shows thesis by (insert assms, auto simp: primitive-def)
lemma primitive-iff-content-eq-1 [simp]:
fixes $f$ :: ' $a$ :: semiring-gcd poly
shows primitive $f \longleftrightarrow$ content $f=1$
proof (intro iffI primitiveI)
fix $x$
assume $(\bigwedge y . y \in \operatorname{set}($ coeffs $f) \Longrightarrow x d v d y)$
from gcd-list-greatest[of coeffs $f, O F$ this]
have $x$ dvd content $f$ by (simp add: content-def)
also assume content $f=1$
finally show $x$ dvd 1 .
next
assume primitive $f$
from primitive $D[O F$ this list-gcd $[$ of - coeffs $f]$, folded content-def $]$
show content $f=1$ by simp
qed
lemma primitive-prod-list:
fixes $f s$ :: ' $a$ :: \{factorial-semiring,semiring-Gcd,normalization-semidom-multiplicative\} poly list
assumes primitive (prod-list fs) and $f \in$ set fs shows primitive $f$
proof (insert assms, induct fs arbitrary: $f$ )
case (Cons f' fs)
from Cons.prems
have is-unit (content $f^{\prime} *$ content (prod-list fs)) by (auto simp: content-mult)
from this[unfolded is-unit-mult-iff]
have content $f^{\prime}=1$ and content (prod-list fs) $=1$ by auto
moreover from Cons.prems have $f=f^{\prime} \vee f \in$ set fs by auto
ultimately show ?case using Cons.hyps $[$ of $f]$ by auto qed auto
lemma irreducible-imp-primitive:
fixes $f::$ ' $a$ :: \{idom,semiring-gcd $\}$ poly
assumes irr: irreducible $f$ and deg: degree $f \neq 0$ shows primitive $f$
proof (rule ccontr)
assume not: $\neg$ ?thesis
then have $\neg[$ :content $f:]$ dvd 1 by simp
moreover have $f=[$ :content $f:] *$ primitive-part $f$ by simp
note Factorial-Ring.irreducibleD[OF irr this]
ultimately
have primitive-part $f d v d 1$ by auto
from this[unfolded poly-dvd-1] have degree $f=0$ by auto
with deg show False by auto
qed
lemma irreducible-primitive-connect:
fixes $f::$ ' $a::\{$ idom,semiring-gcd $\}$ poly
assumes cf: primitive $f$ shows irreducible $_{d} f \longleftrightarrow$ irreducible $f$ (is ?l $\longleftrightarrow$ ?r)
proof
assume $l$ : ?l show ?r
proof (rule ccontr, elim not-irreducibleE)
from $l$ have deg: degree $f>0$ by (auto dest: irreducible ${ }_{d} D$ )
from cf have $f 0: f \neq 0$ by auto
then show $f=0 \Longrightarrow$ False by auto
show $f$ dvd $1 \Longrightarrow$ False using deg by (auto simp:poly-dvd-1)
fix $a b$ assume $f a b: f=a * b$ and $a 1: \neg a$ dvd 1 and $b 1: \neg b$ dvd 1
then have $a f: a d v d f$ and $b f: b d v d f$ by auto
with $f 0$ have $a 0: a \neq 0$ and $b 0: b \neq 0$ by auto
from irreducible $_{d} D(2)[O F ~ l$, of a] af dvd-imp-degree-le[OF af f0]
have degree $a=0 \vee$ degree $a=$ degree $f$
by (metis degree-smult-le irreducible $d_{d}$-dvd-smult l le-antisym Nat.neq0-conv)
then show False
proof (elim disjE)
assume degree $a=0$
then obtain $c$ where $a c: a=[: c:]$ by (auto dest: degree 0 -coeffs)
from fab[unfolded ac] have $c$ dvd content $f$ by (simp add: content-iff co-effs-smult)
with $c f$ have $c$ dvd 1 by simp
then have $a d v d 1$ by (auto simp: ac)
with a1 show False by auto
next
assume dega: degree $a=$ degree $f$
with f0 degree-mult-eq[OF aO b0] fab have degree $b=0$ by (auto simp: ac-simps)
then obtain $c$ where $b c: b=[: c:]$ by (auto dest: degree $0-c o e f f s$ )
from fab[unfolded bc] have $c$ dvd content $f$ by (simp add: content-iff co-effs-smult)

```
        with cf have c dvd 1 by simp
        then have b dvd 1 by (auto simp: bc)
        with b1 show False by auto
    qed
    qed
next
    assume r:?r
    show ?l
    proof(intro irreducible }\mp@subsup{|}{d}{}I
        show degree f>0
        proof (rule ccontr)
            assume }\neg\mathrm{ degree f>0
            then obtain f0 where f:f=[:f0:] by (auto dest: degree0-coeffs)
            from cf[unfolded this] have normalize f0 = 1 by auto
            then have f0 dvd 1 by (unfold normalize-1-iff)
            with r[unfolded f irreducible-const-poly-iff] show False by auto
        qed
    next
    fix gh assume deg-g: degree g>0 and deg-gf: degree g<degree f and fgh:f
=g*h
    with r have g dvd 1 \vee h dvd 1 by auto
    with deg-g have degree h=0 by (auto simp: poly-dvd-1)
    with deg-gf[unfolded fgh] degree-mult-eq[of g h] show False by (cases g=0 \vee
h=0,auto)
    qed
qed
lemma deg-not-zero-imp-not-unit:
    fixes f:: 'a::{idom-divide,semidom-divide-unit-factor} poly
    assumes deg-f:degree f>0
    shows \neg is-unit f
proof -
    have degree (normalize f) >0
            using deg-f degree-normalize by auto
    hence normalize f}\not=
            by fastforce
    thus \neg is-unit f using normalize-1-iff by auto
qed
lemma content-pCons[simp]: content (pCons a p) = gcd a (content p)
proof(induct p arbitrary:a)
    case 0 show ?case by simp
next
    case (pCons c p)
    then show ?case by (cases p = 0, auto simp: content-def cCons-def)
qed
lemma content-field-poly:
    fixes f :: 'a :: {field,semiring-gcd} poly
```

```
shows content f}=(\mathrm{ if }f=0\mathrm{ then 0 else 1)
by(induct f, auto simp:dvd-field-iff is-unit-normalize)
```

end

## 6 Gauss Lemma

We formalized Gauss Lemma, that the content of a product of two polynomials $p$ and $q$ is the product of the contents of $p$ and $q$. As a corollary we provide an algorithm to convert a rational factor of an integer polynomial into an integer factor.

In contrast to the theory on unique factorization domains - where Gauss Lemma is also proven in a more generic setting - we are here in an executable setting and do not use the unspecified some - gcd function. Moreover, there is a slight difference in the definition of content: in this theory it is only defined for integer-polynomials, whereas in the UFD theory, the content is defined for polynomials in the fraction field.

```
theory Gauss-Lemma
imports
    HOL-Computational-Algebra.Primes
    HOL-Computational-Algebra.Field-as-Ring
    Polynomial-Interpolation.Ring-Hom-Poly
    Missing-Polynomial-Factorial
begin
lemma primitive-part-alt-def:
    primitive-part p = sdiv-poly p (content p)
    by (simp add: primitive-part-def sdiv-poly-def)
definition common-denom :: rat list }=>\mathrm{ int }\times\mathrm{ int list where
    common-denom xs \equivlet
        nds = map quotient-of xs;
        denom = list-lcm (map snd nds);
        ints = map ( }\lambda(n,d).n*\mathrm{ denom div d) nds
        in (denom, ints)
definition rat-to-int-poly :: rat poly }=>\mathrm{ int }\times\mathrm{ int poly where
    rat-to-int-poly p \equivlet
        ais = coeffs p;
        d=fst (common-denom ais)
        in (d, map-poly ( }\lambda\mathrm{ x. case quotient-of }x\mathrm{ of (p,q) # p*d div q) p)
definition rat-to-normalized-int-poly :: rat poly }=>\mathrm{ rat }\times\mathrm{ int poly where
    rat-to-normalized-int-poly p \equiv if p=0 then (1,0) else case rat-to-int-poly p of
(s,q)
    | (of-int (content q) / of-int s, primitive-part q)
```

```
lemma rat-to-normalized-int-poly-code[code]:
    rat-to-normalized-int-poly p = (if p=0 then (1,0) else case rat-to-int-poly p of
(s,q)
    let c = content q in (of-int c / of-int s, sdiv-poly q c))
    unfolding Let-def rat-to-normalized-int-poly-def primitive-part-alt-def ..
lemma common-denom: assumes cd: common-denom xs = (dd,ys)
    shows xs = map ( }\lambdai\mathrm{ . of-int i / of-int dd) ys dd > 0
    \x.x set xs \Longrightarrow rat-of-int (case quotient-of x of ( }n,x)=>n*dd div x)
rat-of-int dd = x
proof -
    let ?nds = map quotient-of xs
    define nds where nds=? nds
    let ?denom = list-lcm (map snd nds)
    let ?ints = map ( }\lambda(n,d).n*dd div d) nd
    from cd[unfolded common-denom-def Let-def]
    have dd:dd=? ?enom and ys: ys = ?ints unfolding nds-def by auto
    show dd0: dd > 0 unfolding dd
    by (intro list-lcm-pos(3), auto simp: nds-def quotient-of-nonzero)
    {
    fix }
    assume x: x f set xs
    obtain pq where quot:quotient-of x = (p,q) by force
    from x have (p,q)\in set nds unfolding nds-def using quot by force
    hence q}\in\mathrm{ set (map snd nds) by force
    from list-lcm[OF this] have q: q dvd dd unfolding dd .
    show rat-of-int (case quotient-of x of ( }n,x)=>n*dd div x) / rat-of-int dd =
x
            unfolding quot split unfolding quotient-of-div[OF quot]
    proof -
        have f1: q * (dd div q) = dd
            using dvd-mult-div-cancel q by blast
            have rat-of-int (dd div q) \not=0
                using dd0 dvd-mult-div-cancel q by fastforce
            thus rat-of-int (p*dd div q) / rat-of-int dd = rat-of-int p / rat-of-int q
                using f1 by (metis (no-types) div-mult-swap mult-divide-mult-cancel-right
of-int-mult q)
    qed
    } note main = this
    show xs = map ( }\lambda\mathrm{ i . of-int i / of-int dd) ys unfolding ys map-map o-def nds-def
    by (rule sym, rule map-idI, rule main)
qed
lemma rat-to-int-poly: assumes rat-to-int-poly p = (d,q)
    shows p = smult (inverse (of-int d)) (map-poly of-int q)d>0
proof -
    let ?f = \lambda x. case quotient-of x of (pa,x) => pa*d div x
    define f}\mathrm{ where f=?f
```

```
    from assms[unfolded rat-to-int-poly-def Let-def]
    obtain xs where cd:common-denom (coeffs p)=(d,xs)
    and q: q = map-poly f p unfolding f-def by (cases common-denom (coeffs p),
auto)
    from common-denom[OF cd] have d:d>0 and
        id: \bigwedgex.x set (coeffs p)\Longrightarrow rat-of-int (f x)/ rat-of-int d = x
        unfolding f-def by auto
    have f0: f 0 = 0 unfolding f-def by auto
    have id: rat-of-int (f (coeff p n)) / rat-of-int d = coeff p n for n
        using id[of coeff p n] f0 range-coeff by (cases coeff p n = 0, auto)
    show d>0 by fact
    show p = smult (inverse (of-int d)) (map-poly of-int q)
        unfolding q smult-as-map-poly using id f0
        by (intro poly-eqI, auto simp: field-simps coeff-map-poly)
qed
lemma content-ge-0-int: content p \geq(0 :: int)
    unfolding content-def
    by (cases coeffs p, auto)
lemma abs-content-int[simp]: fixes p :: int poly
    shows abs (content p) = content p using content-ge-0-int[of p] by auto
lemma content-smult-int: fixes p :: int poly
    shows content (smult a p) = abs a* content p by simp
lemma normalize-non-0-smult: \exists a. ( }a::\mp@subsup{\mp@code{''}}{}{\prime}a:: semiring-gcd) \not= 0 ^ smult a
(primitive-part p) = p
    by (cases p = 0, rule exI[of-1], simp, rule exI[of - content p],auto)
lemma rat-to-normalized-int-poly: assumes rat-to-normalized-int-poly p = (d,q)
    shows p= smult d (map-poly of-int q) d>0 p}=0\Longrightarrow\mathrm{ content }q=1\mathrm{ degree q
= degree p
proof -
    have p= smult d (map-poly of-int q) ^d>0\wedge(p\not=0\longrightarrow content q=1)
    proof (cases p=0)
        case True
        thus ?thesis using assms unfolding rat-to-normalized-int-poly-def
        by (auto simp: eval-poly-def)
    next
        case False
    hence p0: p}\not=0\mathrm{ by auto
    obtain s r where id: rat-to-int-poly p}=(s,r) by forc
    let ?cr = rat-of-int (content r)
    let ?s = rat-of-int s
    let ?q = map-poly rat-of-int q
    from rat-to-int-poly[OF id] have p: p= smult (inverse ?s) (map-poly of-int r)
    and s:s>0 by auto
    let ?q = map-poly rat-of-int q
```

from $p 0$ assms[unfolded rat-to-normalized-int-poly-def id split]
have $d: d=? c r / ? s$ and $q: q=$ primitive-part $r$ by auto
from content-times-primitive-part [of $r$, folded $q$ ] have $q r$ : smult (content $r$ ) $q$ $=r$.
have smult $d ? q=$ smult (?cr / ?s) ?q
unfolding $d$ by simp
also have ?cr / ?s $=$ ?cr $*$ inverse ?s by (rule divide-inverse)
also have $\ldots=$ inverse ? $s *$ ? cr by $\operatorname{simp}$
also have smult (inverse ?s * ?cr) ?q $=$ smult (inverse ?s) (smult ?cr ?q) by simp
also have smult ?cr ? $q=$ map-poly of-int (smult (content r) q) by (simp add: hom-distribs)
also have $\ldots=$ map-poly of-int $r$ unfolding $q r$..
finally have $p q: p=$ smult $d$ ? $q$ unfolding $p$ by simp
from $p p 0$ have $r 0: r \neq 0$ by auto
from content-eq-zero-iff[of r] content-ge-0-int[of r] r0 have cr: ?cr >0 by linarith
with $s$ have $d 0: d>0$ unfolding $d$ by auto
from content-primitive-part $[O F r 0]$ have $c q$ : content $q=1$ unfolding $q$.
from $p q d 0 c q$ show ?thesis by auto
qed
thus $p: p=$ smult $d$ (map-poly of-int $q$ ) and $d: d>0$ and $p \neq 0 \Longrightarrow$ content $q=1$ by auto
show degree $q=$ degree $p$ unfolding $p$ smult-as-map-poly
by (rule sym, subst map-poly-map-poly, force+, rule degree-map-poly, insert $d$, auto)
qed
lemma content-dvd-1:
content $g=1$ if content $f=\left(1::{ }^{\prime} a::\right.$ semiring-gcd $) g d v d f$
proof -
from $\langle g d v d f\rangle$ have content $g$ dvd content $f$
by (rule content-dvd-contentI)
with $\langle$ content $f=1$ 〉show ?thesis
by $\operatorname{simp}$
qed
lemma dvd-smult-int: fixes $c::$ int assumes $c: c \neq 0$
and dvd: $q$ dvd (smult c $p$ )
shows primitive-part $q$ dvd $p$
proof (cases $p=0$ )
case True thus ?thesis by auto
next
case False note $p 0=$ this
let ${ }^{2} c p=$ smult $c p$
from $p 0 c$ have $c p 0: ? c p \neq 0$ by auto
from $d v d$ obtain $r$ where prod: ? $c p=q * r$ unfolding $d v d-d e f$ by auto
from prod cp0 have $q 0: q \neq 0$ and $r 0: r \neq 0$ by auto
let ?c $=$ content $::$ int poly $\Rightarrow$ int
let $? n=$ primitive-part $::$ int poly $\Rightarrow$ int poly
let ? $p n=\lambda p$.smult $(? c p)(? n p)$
have $c q:(? c q=0)=$ False using content-eq-zero-iff $q 0$ by auto
from prod have $i d 1$ : ? $c p=? p n q *$ ? pn $r$ unfolding content-times-primitive-part by $\operatorname{simp}$
from arg-cong[OF this, of content, unfolded content-smult-int content-mult content-primitive-part $[O F r 0]$ content-primitive-part[OF q0], symmetric] $p 0[$ folded content-eq-zero-iff] $c$
have abs c dvd ?c $q$ * ?c $r$ unfolding dvd-def by auto
hence $c d v d$ ? $c q *$ ? $c r$ by auto
then obtain $d$ where $i d$ : ? $c q * ? c r=c * d$ unfolding dvd-def by auto
have ? $c p=$ ? $p n q *$ ? $p n r$ by fact
also have $\ldots=$ smult $(c * d)(? n q * ? n r)$ unfolding id [symmetric]
by (metis content-mult content-times-primitive-part primitive-part-mult)
finally have $i d: ? c p=$ smult $c(? n q *$ smult $d(? n r))$ by (simp add: mult.commute)
interpret map-poly-inj-zero-hom $(*) c$ using $c$ by (unfold-locales, auto)
have $p=? n q *$ smult $d(? n r$ ) using id[unfolded smult-as-map-poly[of c]] by auto
thus $d v d$ : ?n $q$ dvd $p$ unfolding $d v d-d e f$ by blast
qed
lemma irreducible $_{d}$-primitive-part:
fixes $p::$ int poly
shows irreducible $_{d}($ primitive-part $p) \longleftrightarrow$ irreducible $_{d} p$ (is ?l $\longleftrightarrow$ ?r)
proof (rule iffI, rule irreducible ${ }_{d} I$ )
assume $l$ : ?l
show degree $p>0$ using $l$ by auto
have dpp: degree (primitive-part $p$ ) $=$ degree $p$ by simp
fix $q r$
assume deg: degree $q<$ degree $p$ degree $r<$ degree $p$ and $p=q * r$
then have $p$ : primitive-part $p=$ primitive-part $q *$ primitive-part $r$ by (simp add: primitive-part-mult)
have $\neg$ irreducible $_{d}$ (primitive-part p)
apply (intro reducible ${ }_{d} I$, rule exI $[o f$ - primitive-part $q$ ], rule exI[of - primi-tive-part r], unfold dpp)
using deg pp by auto
with $l$ show False by auto
next
show ?r $\Longrightarrow$ ?l by (metis irreducible ${ }_{d}$-smultI normalize-non-0-smult)
qed
lemma irreducible $_{d}$-smult-int:
fixes $c::$ int assumes $c: c \neq 0$
shows irreducible $_{d}($ smult $c p)=$ irreducible $_{d} p($ is ?l $=? r)$
using irreducible $_{d}$-primitive-part[of smult $c$ p, unfolded primitive-part-smult] $c$
apply (cases $c<0$, simp)
apply (metis add.inverse-inverse add.inverse-neutral c irreducible ${ }_{d}$-smultI nor-
malize-non-0-smult smult-1-left smult-minus-left)
apply (simp add: irreducible ${ }_{d}$-primitive-part)

## done

```
lemma irreducible \(_{d}\)-as-irreducible:
    fixes \(p::\) int poly
    shows irreducible \(_{d} p \longleftrightarrow\) irreducible ( primitive-part \(p\) )
    using irreducible-primitive-connect[of primitive-part p]
    by (cases \(p=0\), auto simp: irreducible \(e_{d}\)-primitive-part)
```

lemma rat-to-int-factor-content-1: fixes $p::$ int poly
assumes $c p$ : content $p=1$
and pgh: map-poly rat-of-int $p=g * h$
and $g$ : rat-to-normalized-int-poly $g=(r, r g)$
and $h$ : rat-to-normalized-int-poly $h=(s, s h)$
and $p: p \neq 0$
shows $p=r g * s h$
proof -
let $? r=r a t-o f-i n t$
let $? r p=$ map-poly $? r$
from $p$ have $r p 0$ : ? $r p p \neq 0$ by simp
with pgh have $g 0: g \neq 0$ and $h 0: h \neq 0$ by auto
from rat-to-normalized-int-poly[OF g] g0
have $r: r>0 r \neq 0$ and $g: g=$ smult $r(? r p r g)$ and crg: content $r g=1$ by
auto
from rat-to-normalized-int-poly[OF h] h0
have $s: s>0 s \neq 0$ and $h: h=s m u l t s(? r p s h)$ and $c s h:$ content $s h=1$ by
auto
let ? irs = inverse $(r * s)$
from $r s$ have irs0: ? irs $\neq 0$ by (auto simp: field-simps)
have $?^{r p}(r g * s h)=? r p r g * ? r p$ sh by (simp add: hom-distribs)
also have $\ldots=$ smult ? irs (? rp p) unfolding pgh $g h$ using $r s$
by (simp add: field-simps)
finally have $i d$ : ? $r p(r g * s h)=s m u l t$ ? irs $(? r p p)$ by auto
have $r s Z:$ ?irs $\in \mathbb{Z}$
proof (rule ccontr)
assume not: $\neg$ ? irs $\in \mathbb{Z}$
obtain $n d$ where irs': quotient-of ? irs $=(n, d)$ by force
from quotient-of-denom-pos[OF irs $]$ have $d>0$.
from not quotient-of-div[OF irs $]$ have $d \neq 1 d \neq 0$ and irs: ? irs $=$ ?r $n /$ ?r
$d$ by auto
with irs0 have $n 0: n \neq 0$ by auto
from $\langle d>0\rangle\langle d \neq 1\rangle$ have $d \geq 2$ and $\neg d d v d 1$ by auto
with content-iff [of $d p$, unfolded $c p$ ] obtain $c$ where
$c: c \in \operatorname{set}($ coeffs $p$ ) and $d c: \neg d d v d c$
by auto
from $c$ range-coeff $[$ of $p]$ obtain $i$ where $c=$ coeff $p i$ by auto
from arg-cong $[O F$ id, of $\lambda p$. coeff $p i$,
unfolded coeff-smult of-int-hom.coeff-map-poly-hom this[symmetric] irs]
have ?r $n /$ ? $r d *$ ?r $c \in \mathbb{Z}$ by (metis Ints-of-int)

```
    also have ?r n / ?r d * ?r c = ?r ( }n*c)/\mathrm{ ? ? d by simp
    finally have inZ: ?r }(n*c)/\mathrm{ ? }rd\in\mathbb{Z}\mathrm{ .
    have cop: coprime nd by (rule quotient-of-coprime[OF irs'])
    define prod where prod = ?r ( }n*c)/?r 
    obtain }\mp@subsup{n}{}{\prime}\mp@subsup{d}{}{\prime}\mathrm{ where quot: quotient-of prod = ( }\mp@subsup{n}{}{\prime},\mp@subsup{d}{}{\prime})\mathrm{ by force
    have qr: \bigwedge x. quotient-of (?r x)=(x,1)
        using Rat.of-int-def quotient-of-int by auto
    from quotient-of-denom-pos[OF quot] have d'>0.
    with quotient-of-div[OF quot] inZ[folded prod-def] have d'}=
        by (metis Ints-cases Rat.of-int-def old.prod.inject quot quotient-of-int)
    with quotient-of-div[OF quot] have prod = ?r n' by auto
    from arg-cong[OF this, of quotient-of, unfolded prod-def rat-divide-code qr
Let-def split]
    have Rat.normalize ( }n*c,d)=(\mp@subsup{n}{}{\prime},1)\mathrm{ by simp
    from normalize-crossproduct[OF <d\not=0\rangle, of 1 n*c n', unfolded this]
    have id: }n*c=\mp@subsup{n}{}{\prime}*d\mathrm{ by auto
    from quotient-of-coprime[OF irs ] have coprime n d .
    with id have d dvd c
        by (metis coprime-commute coprime-dvd-mult-right-iff dvd-triv-right)
    with dc show False ..
    qed
    then obtain irs where irs: ?irs = ?r irs unfolding Ints-def by blast
    from id[unfolded irs, folded hom-distribs, unfolded of-int-poly-hom.eq-iff]
    have p:rg*sh= smult irs p by auto
    have content (rg*sh)=1 unfolding content-mult crg csh by auto
    from this[unfolded p content-smult-int cp] have abs irs = 1 by simp
    hence abs ?irs = 1 using irs by auto
    with r s have ?irs=1 by auto
    with irs have irs=1 by auto
    with p show p:p=rg*sh by auto
qed
lemma rat-to-int-factor-explicit: fixes p :: int poly
    assumes pgh: map-poly rat-of-int p=g*h
    and g:rat-to-normalized-int-poly g}=(r,rg
    shows }\existsr.p=rg* smult (content p)
proof -
    show ?thesis
    proof (cases p=0)
    case True
    show ?thesis unfolding True
        by (rule exI[of-0], auto simp: degree-monom-eq)
    next
    case False
    hence p: p\not=0 by auto
    let ?r = rat-of-int
    let ?rp = map-poly ?r
    define q}\mathrm{ where q= primitive-part p
```

from content-times-primitive-part $[o f ~ p$, folded $q$-def] content-eq-zero-iff $[o f ~ p] p$
obtain $a$ where $a: a \neq 0$ and $p q: p=$ smult $a q$ and acp: content $p=a$ by metis
from $a p q p$ have $r a:$ ? $a \neq 0$ and $q 0: q \neq 0$ by auto
from content-primitive-part [OF p, folded $q$-def] have $c q$ : content $q=1$ by auto
obtain $s$ sh where $h$ : rat-to-normalized-int-poly (smult (inverse (?r a)) h) = $(s, s h)$ by force
from arg-cong[OF pgh[unfolded pq], of smult (inverse (?r a))] ra
have ? $r p q=g *$ smult (inverse (?r a)) $h$ by (auto simp: hom-distribs)
from rat-to-int-factor-content-1[OF cq this $g h q 0]$
have $q r s: q=r g * s h$.
show ?thesis unfolding acp unfolding $p q$ qrs
by (rule exI $[o f-s h]$, auto)
qed
qed
lemma rat-to-int-factor: fixes $p::$ int poly
assumes pgh: map-poly rat-of-int $p=g * h$
shows $\exists g^{\prime} h^{\prime} . p=g^{\prime} * h^{\prime} \wedge$ degree $g^{\prime}=$ degree $g \wedge$ degree $h^{\prime}=$ degree $h$
proof (cases $p=0$ )
case True
with $p g h$ have $g=0 \vee h=0$ by auto
then show ?thesis
by (metis True degree-0 mult-hom.hom-zero mult-zero-left rat-to-normalized-int-poly(4) surj-pair)
next
case False
obtain $r$ rg where ri: rat-to-normalized-int-poly (smult (1/ of-int (content p))
$g)=(r, r g)$ by force
obtain $q$ qh where ri2: rat-to-normalized-int-poly $h=(q, q h)$ by force
show ?thesis
proof (intro exI conjI)
have of-int-poly $($ primitive-part $p)=\operatorname{smult}(1 /$ of-int $($ content $p))(g * h)$
apply (auto simp: primitive-part-def pgh[symmetric] smult-map-poly map-poly-map-poly
o-def intro!: map-poly-cong)
by (metis (no-types, lifting) content-dvd-coeffs div-by-0 dvd-mult-div-cancel
floor-of-int nonzero-mult-div-cancel-left of-int-hom.hom-zero of-int-mult)
also have $\ldots=\operatorname{smult}(1 /$ of-int (content $p)) g * h$ by $\operatorname{simp}$
finally have of-int-poly (primitive-part p) $=\ldots$..
note main $=$ rat-to-int-factor-content-1[OF - this ri ri2, simplified, OF False]
show $p=$ smult (content $p) r g * q h$ by (simp add: main[symmetric])
from ri2 show degree $q h=$ degree $h$ by (fact rat-to-normalized-int-poly)
from rat-to-normalized-int-poly(4)[OF ri] False
show degree (smult (content p) rg) $=$ degree $g$ by auto
qed
qed
lemma rat-to-int-factor-normalized-int-poly: fixes $p$ :: rat poly
assumes pgh: $p=g * h$
and $p$ : rat-to-normalized-int-poly $p=(i, i p)$
shows $\exists g^{\prime} h^{\prime}$. ip $=g^{\prime} * h^{\prime} \wedge$ degree $g^{\prime}=$ degree $g$
proof -
from rat-to-normalized-int-poly[OF p]
have $p: p=$ smult $i$ (map-poly rat-of-int $i p$ ) and $i: i \neq 0$ by auto
from arg-cong[OF p, of smult (inverse $i$ ), unfolded pgh] $i$
have map-poly rat-of-int ip $=g *$ smult (inverse $i$ ) $h$ by auto
from rat-to-int-factor[OF this] show ?thesis by auto
qed
lemma irreducible-smult [simp]:
fixes $c::{ }^{\prime} a::$ field
shows irreducible (smult c $p$ ) $\longleftrightarrow$ irreducible $p \wedge c \neq 0$
using irreducible-mult-unit-left $[$ of $[: c:]$, simplified $]$ by force
A polynomial with integer coefficients is irreducible over the rationals, if it is irreducible over the integers.

```
theorem irreducible \(_{d}\)-int-rat: fixes \(p::\) int poly
    assumes \(p\) : irreducible \(_{d} p\)
    shows irreducible \({ }_{d}\) (map-poly rat-of-int p)
proof (rule irreducible \({ }_{d} I\) )
    from irreducible \(_{d} D[\) OF \(p]\)
    have \(p\) : degree \(p \neq 0\) and irr: \(\bigwedge q r\). degree \(q<\) degree \(p \Longrightarrow\) degree \(r<\) degree
\(p \Longrightarrow p \neq q * r\) by auto
    let \(? r=r a t-o f-i n t\)
    let \({ }^{2} r p=\) map-poly \(?\) r
    from \(p\) show \(r p\) : degree (? \(r p p\) ) \(>0\) by auto
    from \(p\) have \(p 0: p \neq 0\) by auto
    fix \(g h\) :: rat poly
    assume deg: degree \(g>0\) degree \(g<\) degree (?rp \(p\) ) degree \(h>0\) degree \(h<\)
degree (?rp p) and pgh: ?rp \(p=g * h\)
    from rat-to-int-factor [OF pgh] obtain \(g^{\prime} h^{\prime}\) where \(p: p=g^{\prime} * h^{\prime}\) and dg: degree
\(g^{\prime}=\) degree \(g\) degree \(h^{\prime}=\) degree \(h\)
            by auto
    from \(\operatorname{irr}\left[\right.\) of \(\left.g^{\prime} h\right] \operatorname{deg}[\) unfolded \(d g]\)
    show False using degree-mult-eq[of \(\left.g^{\prime} h\right]\) by (auto simp: \(p d g\) )
qed
corollary irreducible \(_{d}\)-rat-to-normalized-int-poly:
    assumes rp: rat-to-normalized-int-poly \(r p=(a, i p)\)
    and \(i p:\) irreducible \(_{d}\) ip
    shows irreducible \(_{d} r p\)
proof -
    from rat-to-normalized-int-poly[OF rp]
    have rp: \(r p=\) smult \(a\) (map-poly rat-of-int \(i p\) ) and \(a: a \neq 0\) by auto
    with irreducible \(_{d}\)-int-rat \([O F\) ip] show ?thesis by auto
qed
```

lemma dvd-content-dvd: assumes dvd: content $f d v d$ content $g$ primitive-part $f d v d$ primitive-part $g$

$$
\text { shows } f d v d g
$$

## proof -

let ? $c f=$ content $f$ let $? n f=$ primitive-part $f$
let $? c g=$ content $g$ let $? n g=$ primitive-part $g$
have $f$ dvd $g=$ (smult ?cf ?nf dvd smult ?cg ?ng)
unfolding content-times-primitive-part by auto
from $d v d(1)$ obtain $c h$ where $c g: ? c g=? c f * c h$ unfolding $d v d-d e f$ by auto
from $d v d(2)$ obtain $n h$ where $n g: ? n g=? n f * n h$ unfolding dvd-def by auto have $f$ dvd $g=($ smult ?cf ?nf dvd smult ?cg ?ng)
unfolding content-times-primitive-part $[$ of $f]$ content-times-primitive-part $[o f ~ g]$ by auto
also have $\ldots=($ smult $? c f$ ? $n f$ dvd smult ? $c f$ ? $n f *$ smult ch $n h)$ unfolding $c g$ $n g$
by (metis mult.commute mult-smult-right smult-smult)
also have ... by (rule dvd-triv-left)
finally show ?thesis.
qed
lemma sdiv-poly-smult: $c \neq 0 \Longrightarrow$ sdiv-poly (smult $c f) c=f$
by (intro poly-eqI, unfold coeff-sdiv-poly coeff-smult, auto)
lemma primitive-part-smult-int: fixes $f::$ int poly shows
primitive-part $($ smult $d f)=$ smult $($ sgn $d)($ primitive-part $f)$
proof (cases $d=0 \vee f=0$ )
case False
obtain $c f$ where $c f$ : content $f=c f$ by auto
with False have $0: d \neq 0 f \neq 0 c f \neq 0$ by auto
show ?thesis
proof (rule poly-eqI, unfold primitive-part-alt-def coeff-sdiv-poly content-smult-int coeff-smult cf)
fix $n$
consider (pos) $d>0 \mid$ (neg) $d<0$ using $0(1)$ by linarith
thus $d *$ coeff $f n \operatorname{div}(|d| * c f)=\operatorname{sgn} d *($ coeff $f n \operatorname{div} c f)$
proof cases
case neg
hence ?thesis $=(d *$ coeff $f n$ div $-(d * c f)=-($ coeff $f n$ div cf $))$ by auto
also have $d *$ coeff $f n$ div $-(d * c f)=-(d *$ coeff $f n \operatorname{div}(d * c f))$
by (subst dvd-div-neg, insert $0(1)$, auto simp: cf[symmetric])
also have $d *$ coeff $f n$ div $(d * c f)=$ coeff $f n$ div of using $O(1)$ by auto
finally show? ?thesis by simp
qed auto
qed
qed auto
lemma gcd-smult-left: assumes $c \neq 0$
shows $g c d$ (smult $c f) g=g c d f(g:: ' b::\{$ field- $g c d\}$ poly $)$
proof -

```
    from assms have normalize c = 1
    by (meson dvd-field-iff is-unit-normalize)
    then show ?thesis
    by (metis (no-types) Polynomial.normalize-smult gcd.commute gcd.left-commute
gcd-left-idem gcd-self smult-1-left)
qed
lemma gcd-smult-right: c\not=0\Longrightarrowgcd f(smult c g)=gcd f(g :: 'b :: {field-gcd}
poly)
    using gcd-smult-left[of c g f] by (simp add: gcd.commute)
lemma gcd-rat-to-gcd-int: gcd (of-int-poly f :: rat poly) (of-int-poly g) =
    smult (inverse (of-int (lead-coeff (gcd f g)))) (of-int-poly (gcd fg))
proof (cases f=0^g=0)
    case True
    thus ?thesis by simp
next
    case False
    let ?r = rat-of-int
    let ?rp = map-poly ?r
    from False have gcd0: gcd fg\not=0 by auto
    hence lc0: lead-coeff (gcd fg)\not=0 by auto
    hence inv: inverse (?r (lead-coeff (gcd f g)))\not=0 by auto
    show ?thesis
    proof (rule sym, rule gcdI, goal-cases)
        case 1
        have gcd fg dvd f by auto
    then obtain h where f:f=gcd fg*h unfolding dvd-def by auto
    show ?case by (rule smult-dvd[OF - inv], insert arg-cong[OF f, of ?rp], simp
add: hom-distribs)
    next
        case 2
        have gcd f g dvd g by auto
        then obtain h where g:g=gcd fg*h unfolding dvd-def by auto
        show ?case by (rule smult-dvd[OF - inv], insert arg-cong[OF g, of ?rp], simp
add: hom-distribs)
    next
        case (3 h)
        show ?case
    proof (rule dvd-smult)
            obtain ch ph where h: rat-to-normalized-int-poly h=(ch,ph) by force
            from 3 obtain ff where f: ?rp f}=h*ff\mathrm{ unfolding dvd-def by auto
            from 3 obtain gg where g: ?rp g=h*gg unfolding dvd-def by auto
                from rat-to-int-factor-explicit[OF f h] obtain f' where f:f=ph* f' by
blast
        from rat-to-int-factor-explicit[OF g h] obtain g' where g: g=ph* g' by
blast
            from fg}\mathrm{ have ph dvd gcd fg by auto
            then obtain gg where gcd: gcd fg=ph*gg unfolding dvd-def by auto
```

```
            note * = rat-to-normalized-int-poly[OF h]
            show h dvd ?rp (gcd fg) unfolding gcd *(1)
            by (rule smult-dvd, insert *(2), auto)
        qed
    next
        case 4
        have [simp]:[:1:]=1 by simp
        show ?case unfolding normalize-poly-def
        by (rule poly-eqI, simp)
    qed
qed
end
```


## 7 Prime Factorization

This theory contains not-completely naive algorithms to test primality and to perform prime factorization. More precisely, it corresponds to prime factorization algorithm A in Knuth's textbook [1].

```
theory Prime-Factorization
imports
    HOL-Computational-Algebra.Primes
    Missing-List
    Missing-Multiset
begin
```


### 7.1 Definitions

definition primes-1000 :: nat list where
primes-1000 $=[2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59$, $61,67,71,73,79,83,89,97,101$,
103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199,
211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317,
331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443,
$449,457,461,463,467,479,487,491,499,503,509,521,523,541,547$, 557, 563, 569, 571, 577,
587, 593, 599, 601, 607, 613, 617, 619, 631, 641, 643, 647, 653, 659, 661, 673, 677, 683, 691, 701,

709, 719, 727, 733, 739, 743, 751, 757, 761, 769, 773, 787, 797, 809, 811, 821, 823, 827, 829, 839,
$853,857,859,863,877,881,883,887,907,911,919,929,937,941,947$, 953, 967, 971, 977, 983,
991, 997]
lemma primes-1000: primes-1000 $=$ filter prime $[0 . .<1001]$

## by eval

definition next-candidates :: nat $\Rightarrow$ nat $\times$ nat list where
next-candidates $n=($ if $n=0$ then (1001,primes-1000) else $(n+30$,
$[n, n+2, n+6, n+8, n+12, n+18, n+20, n+26]))$
definition candidate-invariant $n=(n=0 \vee n \bmod 30=(11::$ nat $))$
partial-function (tailrec) remove-prime-factor :: nat $\Rightarrow$ nat $\Rightarrow$ nat list $\Rightarrow$ nat $\times$ nat list where
[code]: remove-prime-factor p $n$ ps $=$ (case Euclidean-Rings.divmod-nat n $p$ of $\left(n^{\prime}, m\right) \Rightarrow$
if $m=0$ then remove-prime-factor $p n^{\prime}(p \# p s)$ else $\left.(n, p s)\right)$
partial-function (tailrec) prime-factorization-nat-main
$::$ nat $\Rightarrow$ nat $\Rightarrow$ nat list $\Rightarrow$ nat list $\Rightarrow$ nat list where
[code]: prime-factorization-nat-main $n j$ is $p s=$ (case is of
[]$\Rightarrow$ (case next-candidates $j$ of $(j, i s) \Rightarrow$ prime-factorization-nat-main $n j$ is ps)
$\mid(i \# i s) \Rightarrow$ (case Euclidean-Rings.divmod-nat $n i$ of $\left(n^{\prime}, m\right) \Rightarrow$ if $m=0$ then case remove-prime-factor in $n^{\prime}(i \# p s)$ of $\left(n^{\prime}, p s^{\prime}\right) \Rightarrow$ if $n^{\prime}=1$ then $p s^{\prime}$ else prime-factorization-nat-main $n^{\prime} j$ is $p s^{\prime}$ else if $i * i \leq n$ then prime-factorization-nat-main $n j$ is $p s$ else $(n \# p s))$ )
partial-function (tailrec) prime-nat-main
$::$ nat $\Rightarrow$ nat $\Rightarrow$ nat list $\Rightarrow$ bool where
[code]: prime-nat-main $n j$ is $=($ case is of []$\Rightarrow$ (case next-candidates $j$ of $(j, i s) \Rightarrow$ prime-nat-main $n j$ is $)$
$\mid(i \#$ is $) \Rightarrow($ if $i$ dvd $n$ then $i \geq n$ else if $i * i \leq n$ then prime-nat-main $n j$ is else True))
definition prime-nat $::$ nat $\Rightarrow$ bool where
prime-nat $n \equiv$ if $n<2$ then False else - TODO: integrate precomputed map case next-candidates 0 of $(j, i s) \Rightarrow$ prime-nat-main $n j$ is
definition prime-factorization-nat $::$ nat $\Rightarrow$ nat list where
prime-factorization-nat $n \equiv \operatorname{rev}$ (if $n<2$ then [] else case next-candidates 0 of $(j, i s) \Rightarrow$ prime-factorization-nat-main $n j$ is [])
definition divisors-nat :: nat $\Rightarrow$ nat list where
divisors-nat $n \equiv$ if $n=0$ then [] else remdups-adj (sort (map prod-list (subseqs (prime-factorization-nat $n)$ ))
definition divisors-int-pos :: int $\Rightarrow$ int list where
divisors-int-pos $x \equiv$ map int (divisors-nat (nat (abs $x)$ ))
definition divisors-int $::$ int $\Rightarrow$ int list where
divisors-int $x \equiv$ let $x s=$ divisors-int-pos $x$ in $x s$ @ (map uminus $x s$ )

### 7.2 Proofs

lemma remove-prime-factor: assumes res: remove-prime-factor in $n s=(m, q s)$ and $i: i>1$ and $n: n \neq 0$
shows $\exists$ rs. qs $=r s @ p s \wedge n=m *$ prod-list rs $\wedge \neg i d v d m \wedge$ set $r s \subseteq\{i\}$ using res $n$
proof (induct $n$ arbitrary: ps rule: less-induct)
case (less n ps)
obtain $n^{\prime}$ mo where $d m$ : Euclidean-Rings.divmod-nat $n i=\left(n^{\prime}, m o\right)$ by force
hence $n^{\prime}: n^{\prime}=n$ div $i$ and mo: mo $=n \bmod i$ by (auto simp: Euclidean-Rings.divmod-nat-def)
from less(2)[unfolded remove-prime-factor.simps $[$ of $i n] d m]$
have res: $\left(\right.$ if $m o=0$ then remove-prime-factor $i n^{\prime}(i \# p s)$ else $\left.(n, p s)\right)=(m$,
$q s)$ by auto
from less(3) have $n: n \neq 0$ by auto
with $n^{\prime} i$ have $n^{\prime}<n$ by auto
note $I H=\operatorname{less}(1)[$ OF this]
show ?case
proof (cases mo $=0$ )
case True
with mo $n^{\prime}$ have $n: n=n^{\prime} * i$ by auto
with $\langle n \neq 0\rangle$ have $n^{\prime}: n^{\prime} \neq 0$ by auto
from True res have remove-prime-factor in' $n^{\prime}(\# p s)=(m, q s)$ by auto
from $I H[O F$ this $n]$ obtain $r s$ where
$q s=r s @ i \# p s$ and $n^{\prime}=m *$ prod-list rs $\wedge \neg i d v d m \wedge$ set $r s \subseteq\{i\}$ by
auto
thus ?thesis
by (intro exI[of -rs @ [i]], unfold n, auto)
next
case False
with mo have $i-n$ : $\neg i d v d n$ by auto
from False res have $i d: m=n q s=p s$ by auto
show ?thesis unfolding id using $i-n$ by auto
qed
qed
lemma prime-sqrtI: assumes $n: n \geq 2$
and small: $\wedge j$. $2 \leq j \Longrightarrow j<i \Longrightarrow \neg j d v d n$
and $i$ : $\neg i * i \leq n$
shows prime ( $n:: n a t$ ) unfolding prime-nat-iff
proof (intro conjI impI allI)
show $1<n$ using $n$ by auto
fix $j$
assume $j n$ : $j$ dvd $n$
from $j n$ obtain $k$ where $n j k: n=j * k$ unfolding dvd-def by auto
with $\langle 1<n\rangle$ have $j n: j \leq n$ by (metis dvd-imp-le jn neq0-conv not-less0)
show $j=1 \vee j=n$

```
    proof (rule ccontr)
    assume \neg ?thesis
    with njk n have j1:j> 1^j\not=n by simp
    have \existsjk. 1<j^j\leqk^n=j*k
    proof (cases j\leqk)
        case True
        thus ?thesis unfolding njk using j1 by blast
    next
        case False
        show ?thesis by (rule exI[of - k], rule exI[of - j], insert <1 < n> j1 njk False,
auto)
            (metis Suc-lessI mult-0-right neq0-conv)
    qed
    then obtain jk where j1: 1<j and jk:j\leqk and njk:n=j*k by auto
    with small[of j] have ji: j\geqi unfolding dvd-def by force
    from mult-mono[OF ji ji] have i*i\leqj*j by auto
    with i have j*j>n by auto
    from this[unfolded njk] have k<j by auto
    with jk show False by auto
    qed
qed
lemma candidate-invariant-0:candidate-invariant 0
    unfolding candidate-invariant-def by auto
lemma next-candidates: assumes res: next-candidates n = (m,ps)
    and n: candidate-invariant n
    shows candidate-invariant m sorted ps {i.prime i\wedgen\leqi^i<m}\subseteq set ps
        set ps\subseteq{2..}\cap{n..<m} distinct ps ps \not=[] n<m
    unfolding candidate-invariant-def
proof -
    note res = res[unfolded next-candidates-def]
    note }n=n[unfolded candidate-invariant-def]
    show m=0 \vee m mod 30=11 using res n by (auto split: if-splits)
    show sorted ps using res n by (auto split: if-splits simp: primes-1000-def sorted2-simps
simp del: sorted-wrt.simps(2))
    show set ps\subseteq{2..} \cap {n..<m} using res n by (auto split: if-splits simp:
primes-1000-def)
    show distinct ps using res n by (auto split: if-splits simp: primes-1000-def)
    show ps }\not=[]\mathrm{ using res n by (auto split: if-splits simp: primes-1000-def)
    show }n<m\mathrm{ using res by (auto split: if-splits)
    show {i.prime i\wedgen\leqi^i<m}\subseteq set ps
    proof (cases n=0)
    case True
    hence *:m=1001 ps= primes-1000 using res by auto
    show ?thesis unfolding * True primes-1000 by auto
    next
    case False
    hence n: n mod 30=11 and m: m=n+30 and ps: ps = [n,n+2,n+6,n+8,n+12,n+18,n+20,n+26]
```

using res $n$ by auto
\{
fix $i$
assume $*$ : prime $i n \leq i i<n+30 i \notin$ set ps
from $n *$ have $i 11: i \geq 11$ by auto
define $j$ where $j=i-n$
have $i: i=n+j$ using $\langle n \leq i\rangle j$-def by auto
have $i \bmod 30=(j+n) \bmod 30$ using $\langle n \leq i\rangle$ unfolding $j$-def by simp
also have $\ldots=(j \bmod 30+n \bmod 30) \bmod 30$
by (simp add: mod-simps)
also have $\ldots=(j \bmod 30+11) \bmod 30$ unfolding $n$ by $\operatorname{simp}$
finally have $i 30: i \bmod 30=(j \bmod 30+11) \bmod 30$ by $\operatorname{simp}$
have 2: 2 dvd ( 30 :: nat) and 112: $11 \bmod (2::$ nat $)=1$ by simp-all
have $(j+11) \bmod 2=(j+1) \bmod 2$
by (rule mod-add-cong) simp-all
with arg-cong [OF i30, of $\lambda j$. $j$ mod 2]
have 2: $i \bmod 2=(j \bmod 2+1) \bmod 2$
by (simp add: mod-simps mod-mod-cancel [OF 2])
have 3: 3 dvd ( 30 :: nat) and 113: $11 \bmod (3::$ nat $)=2$ by simp-all
have $(j+11) \bmod 3=(j+2) \bmod 3$
by (rule mod-add-cong) simp-all
with $\arg -\operatorname{cong}[O F i 30$, of $\lambda j$. $j \bmod 3]$ have $3: i \bmod 3=(j \bmod 3+2)$ mod 3
by (simp add: mod-simps mod-mod-cancel [OF 3])
have $5: 5 \mathrm{dvd}(30::$ nat $)$ and $115: 11 \bmod (5::$ nat $)=1$ by simp-all
have $(j+11) \bmod 5=(j+1) \bmod 5$
by (rule mod-add-cong) simp-all
with arg-cong [OF i30, of $\lambda j$. $j \bmod 5]$ have $5: i \bmod 5=(j \bmod 5+1)$ mod 5
by (simp add: mod-simps mod-mod-cancel [OF 5])
from $n *(2-)$ [unfolded ps $i$, simplified] have

$$
j \in\{1,3,5,7,9,11,13,15,17,19,21,23,25,27,29\} \vee j \in\{4,10,16,22,28\} \vee
$$

$j \in\{14,24\}$
(is $j \in ? ~ j 2 \vee j \in ? j 3 \vee j \in ? j 5$ )
by simp presburger
moreover
\{
assume $j \in ? j 2$
hence $j \bmod 2=1$ by auto
with 2 have $i \bmod 2=0$ by auto
with $i 11$ have 2 dvd $i i \neq 2$ by auto
with $*(1)$ have False unfolding prime-nat-iff by auto
\}
moreover
\{
assume $j \in ? j 3$
hence $j \bmod 3=1$ by auto

```
            with 3 have i mod 3 = 0 by auto
            with i11 have 3 dvd i i\not=3 by auto
            with *(1) have False unfolding prime-nat-iff by auto
        }
        moreover
        {
            assume j }\in
            hence j mod 5 = 4 by auto
            with }5\mathrm{ have i mod 5 = 0 by auto
            with i11 have 5 dvd i i\not=5 by auto
            with *(1) have False unfolding prime-nat-iff by auto
        }
        ultimately have False by blast
        }
        thus ?thesis unfolding m ps by auto
    qed
qed
lemma prime-test-iterate2: assumes small: \j. 2 \leq j \Longrightarrow j < (i :: nat ) \Longrightarrow ᄀ
j dvd n
    and odd: odd n
    and n: n\geq3
    and i:i\geq3 odd i
    and mod: \neg i dvd n
    and j:2 
    shows \neg j dvd n
proof
    assume dvd: j dvd n
    with small[OF j(1)] have j\geqi}\mathrm{ by linarith
    with dvd mod have j>i by (cases i=j,auto)
    with j have j=Suc i by simp
    with i have even j by auto
    with dvd j(1) have 2 dvd n by (metis dvd-trans)
    with odd show False by auto
qed
lemma prime-divisor: assumes j\geq2 and j dvd n shows
    \exists :: nat. prime p}\wedge p dvd j^pdvd 
proof -
    let ?pf = prime-factors j
    from assms have j>0 by auto
    from prime-factorization-nat[OF this]
    have j=(\prodp\in?pf. p ^ multiplicity p j) by auto
    with }\langlej\geq2\rangle have ?pf \not={} by aut
    then obtain p where p:p\in?pf by auto
    hence pr: prime p by auto
    define rem where rem = (\prodp\in?pf - {p}. p^ multiplicity pj)
    from p have mult: multiplicity p j\not=0
```

```
    by (auto simp: prime-factors-multiplicity)
    have finite ?pf by simp
    have j=(\prodp\in?.pf. p^ multiplicity p j) by fact
    also have ?pf = (insert p (?pf - {p})) using p by auto
    also have (\prodp\ininsert p (?pf - {p}). p^ multiplicity p j)=
        p ^ multiplicity p j * rem unfolding rem-def
    by (subst prod.insert, auto)
    also have ...= p*( p^ (multiplicity pj-1)* rem) using mult
    by (cases multiplicity p j, auto)
    finally have pj:p dvd j unfolding dvd-def by blast
    with «j dvd n〉 have p dvd n by (metis dvd-trans)
    with pj pr show ?thesis by blast
qed
lemma prime-nat-main: ni = (n,i,is)\Longrightarrowi\geq2 \Longrightarrow n\geq2 ב
    (\bigwedgej.2 }\leqj\Longrightarrowj<i\Longrightarrow\neg(jdvd n)) 
    (\bigwedgej.i\leqj\Longrightarrowj<jj\Longrightarrow prime j\Longrightarrowj\in set is)\Longrightarrowi\leqjj\Longrightarrow
    sorted is \Longrightarrow distinct is \Longrightarrow candidate-invariant jj \Longrightarrow set is\subseteq{i..<jj}\Longrightarrow
    res = prime-nat-main n jj is \Longrightarrow
    res = prime n
proof (induct ni arbitrary: n i is jj res rule: wf-induct[OF
            wf-measures[of [\lambda(n,i,is). n-i,\lambda(n,i,is). if is = [] then 1 else 0]]])
    case (1 ni n i is jj res)
    note res = 1(12)
    from 1(3-4) have i:i\geq2 and i2:Suc i\geq2 and n: n\geq2 by auto
    from 1(5) have dvd: \ j. 2 }\leqj\Longrightarrowj<i\Longrightarrow\negjdvd n.
    from 1(7) have ijj: i\leqjj.
    note sort-dist = 1(8-9)
    have is: \j. i\leqj\Longrightarrowj<jj\Longrightarrow prime j\Longrightarrowj\in set is by (rule 1(6))
    note simps=prime-nat-main.simps[of n jj is]
    note IH=1(1)[rule-format,unfolded 1(2),OF-refl]
    show ?case
    proof (cases is)
        case Nil
    obtain jjj iis where can: next-candidates jj = (jjj,iis) by force
    from res[unfolded simps, unfolded Nil can split] have res:res = prime-nat-main
n jjj iis by auto
    from next-candidates[OF can 1(10)] have can:
            sorted iis distinct iis candidate-invariant jjj
            {i.prime i}\wedgejj\leqi^i<jjj}\subseteq set iis set iis\subseteq{2..}\cap{jj..<jjj
            iis \not=[] jj < jjj by blast+
    from can ijj have i\leq jjj by auto
    note IH=IH[OF - in dvd - this can(1-3) - res]
    show ?thesis
    proof (rule IH, force simp: Nil can(6))
        fix }
        assume ix: i\leqx and xj:x<jjj and px: prime x
        from is[OF ix - px] Nil have jx: jj \leqx by force
        with can(4) xj px show }x\in\mathrm{ set iis by auto
```

```
    qed (insert can(5) ijj, auto)
    next
    case (Cons i' iis)
    with res[unfolded simps]
    have res: res = (if i' dvd n then n \leq i' else if i'}*\mp@subsup{i}{}{\prime}\leqn\mathrm{ then prime-nat-main
n jj iis else True)
            by simp
    from 1(11) Cons have iis: set iis\subseteq{i..<jj} and i':i\leq i' i'<jjSuc i'
by auto
    from sort-dist have sd-iis: sorted iis distinct iis and }\mp@subsup{i}{}{\prime}\not\in\mathrm{ set iis by(auto simp:
Cons)
    from sort-dist(1) have set iis \subseteq{\mp@subsup{i}{}{\prime}..} by(auto simp:Cons)
    with iis have set iis \subseteq{\mp@subsup{i}{}{\prime}..<jj} by force
    with <i' }\not=\mathrm{ set iis〉 have iis: set iis }\subseteq{Suc i'..<jj
        by (auto, case-tac x = i',}\mathrm{ ,auto)
    {
        fix }
        assume j:2 < j j< i'
        have }\negjdvd
        proof
                assume j dvd n
            from prime-divisor[OF j(1) this] obtain p where
                p: prime p p dvd j p dvd n by auto
                have pj: p \leq j
                    by (rule dvd-imp-le[OF p(2)], insert j, auto)
                have p2: 2 \leq p using p(1) by (rule prime-ge-2-nat)
                from dvd[OF p2] p(3) have pi: p\geqi by force
                from pj j(2) i' is[OF pi-p(1)] have p set is by auto
                with «sorted is` have i'}\mp@subsup{i}{}{\prime
                with pj j(2) show False by arith
        qed
    } note dvd = this
    from i' i have i'2: 2 \leqSuc i' by auto
    note IH=IH[OF-\mp@subsup{i}{}{\prime}2n--\mp@subsup{i}{}{\prime}(3) sd-iis 1(10) iis]
    show ?thesis
    proof (cases i' dvd n)
        case False note dvdi= this
        {
            fix }
            assume j: 2 \leqjj< Suc i'
            have }\negjdvd
            proof (cases j= '')
                case False
                with j have j< i' by auto
                    from dvd[OF j(1) this] show ?thesis .
            qed (insert False, auto)
        } note dvds= this
        show ?thesis
        proof (cases i'* ' }\mp@subsup{i}{}{\prime}\leqn
```

```
            case True note iin = this
            with res False have res:res = prime-nat-main n jj iis by auto
            from ion have i-n: i'<n
            using dvd dvdi n nat-neq-iff dvd-refl by blast
        {
            fix }
            assume Suc i' 
            hence i\leqx x< jj prime x using i' by auto
            from is[OF this] have x\in set is.
            with «Suc \mp@subsup{i}{}{\prime}}\leqx\rangle\mathrm{ have }x\in\mathrm{ set iis unfolding Cons by auto
            } note iis = this
            show ?thesis
            by (rule IH[OF - dvds iis res], insert i-n i', auto)
        next
            case False
            with res dvdi have res: res = True by auto
            have n: prime n
            by (rule prime-sqrtI[OF n dvd False])
            thus ?thesis unfolding res by auto
        qed
    next
        case True
        have }\mp@subsup{i}{}{\prime}\geq2 using i i ' by aut
    from 〈i`}\mp@subsup{}{}{\prime}dvd n\rangle\mathrm{ obtain }k\mathrm{ where n= i'* k..
    with n have k\not=0 by (cases k=0, auto)
    with <n= i'* 
        by auto
    with True res have res \longleftrightarrow < i'=n
        by auto
    also have ... = prime n
    using * proof
        assume i'<n
        with \langlei'\geq 2\rangle\langlei' dvd n\rangle have }\neg\mathrm{ prime n
            by (auto simp add: prime-nat-iff)
        with <i'< n` show ?thesis
            by auto
    next
        assume i' }=
        with dvd n have prime n
            by (simp add: prime-nat-iff')
            with <i' = n` show ?thesis
            by auto
        qed
        finally show ?thesis .
    qed
    qed
qed
```

lemma prime-factorization-nat-main: $n i=(n, i, i s) \Longrightarrow i \geq 2 \Longrightarrow n \geq 2 \Longrightarrow$

$$
(\wedge j .2 \leq j \Longrightarrow j<i \Longrightarrow \neg(j d v d n)) \Longrightarrow
$$

$$
(\bigwedge j . i \leq j \Longrightarrow j<j j \Longrightarrow \text { prime } j \Longrightarrow j \in \text { set } i s) \Longrightarrow i \leq j j \Longrightarrow
$$

$$
\text { sorted is } \Longrightarrow \text { distinct is } \Longrightarrow \text { candidate-invariant } j j \Longrightarrow \text { set is } \subseteq\{i . .<j j\} \Longrightarrow
$$

$$
\text { res }=\text { prime-factorization-nat-main } n \text { jj is } p s \Longrightarrow
$$

$\exists$ qs. res $=q s @ p s \wedge$ Ball (set qs) prime $\wedge n=$ prod-list qs
proof (induct ni arbitrary: $n i$ is $j j$ res ps rule: wf-induct [OF
wf-measures $[$ of $[\lambda(n, i, i s) . n-i, \lambda(n, i$, is $)$. if is $=[]$ then 1 else 0$]]])$
case ( 1 nini is $j \mathrm{j}$ res ps)
note res $=1$ (12)
from 1(3-4) have $i: i \geq 2$ and $i 2:$ Suc $i \geq 2$ and $n: n \geq 2$ by auto
from 1(5) have dvd: $\wedge j$. $2 \leq j \Longrightarrow j<i \Longrightarrow \neg j$ dvd $n$.
from $1(7)$ have $i j j: i \leq j j$.
note sort-dist $=1(8-9)$
have is: $\wedge j . i \leq j \Longrightarrow j<j j \Longrightarrow$ prime $j \Longrightarrow j \in$ set is by (rule 1(6))
note simps $=$ prime-factorization-nat-main.simps $[$ of $n$ ij is]
note $I H=1$ (1)[rule-format, unfolded 1(2), OF - refl]
show ? case
proof (cases is)
case Nil
obtain $j j j$ iis where can: next-candidates $j j=(j j j, i i s)$ by force
from res[unfolded simps, unfolded Nil can split] have res: res = prime-factorization-nat-main $n$ jjj is $p s$ by auto
from next-candidates[OF can 1(10)] have can:
sorted iis distinct iis candidate-invariant jjj
$\{i$. prime $i \wedge j j \leq i \wedge i<j j j\} \subseteq$ set is set iis $\subseteq\{2 ..\} \cap\{j j . .<j j j\}$
iis $\neq[] j j<j j j$ by blast +
from can $i j j$ have $i \leq j j j$ by auto
note $I H=I H[O F-i n d v d-\operatorname{this} \operatorname{can}(1-3)-r e s]$
show ?thesis
proof (rule IH, force simp: Nil can(6))
fix $x$
assume $i x: i \leq x$ and $x j: x<j j j$ and $p x$ : prime $x$
from $i s[O F i x-p x]$ Nil have $j x: j j \leq x$ by force
with can(4) $x j p x$ show $x \in$ set iis by auto
qed (insert can(5) ijj, auto)
next
case (Cons $\left.i^{\prime} i i s\right)$
obtain $n^{\prime} m$ where dm: Euclidean-Rings.divmod-nat $n i^{\prime}=\left(n^{\prime}, m\right)$ by force
hence $n^{\prime}: n^{\prime}=n$ div $i^{\prime}$ and $m: m=n$ mod $i^{\prime}$ by (auto simp: Euclidean-Rings.divmod-nat-def)
have $m:(m=0)=\left(i^{\prime}\right.$ dvd $\left.n\right)$ unfolding $m$ by auto
from Cons res[unfolded simps] $d m m n^{\prime}$
have res: res $=($
if $i^{\prime}$ dvd $n$ then case remove-prime-factor $i^{\prime}\left(n\right.$ div $\left.i^{\prime}\right)\left(i^{\prime} \# p s\right)$ of
$\left(n^{\prime}, p s^{\prime}\right) \Rightarrow$ if $n^{\prime}=1$ then $p s^{\prime}$ else prime-factorization-nat-main $n^{\prime} j j$ iis $p s^{\prime}$
else if $i^{\prime} * i^{\prime} \leq n$ then prime-factorization-nat-main $n j j$ is ps else $n \# p s$ ) by $\operatorname{simp}$
from 1(11) $i$ Cons have iis: set $i$ is $\subseteq\{i . .<j j\}$ and $i^{\prime}: i \leq i^{\prime} i^{\prime}<j j$ Suc $i^{\prime} \leq$ $j j i^{\prime}>1$ by auto
from sort-dist have sd-iis: sorted iis distinct iis and $i^{\prime} \notin$ set iis by (auto simp: Cons)
from sort-dist(1) Cons have set iis $\subseteq\left\{i^{\prime} ..\right\}$ by (auto)
with iis have set iis $\subseteq\left\{i^{\prime} . .<j j\right\}$ by force
with $\left\langle i^{\prime} \notin\right.$ set iis〉 have iis: set iis $\subseteq\left\{\right.$ Suc $\left.i^{\prime}{ }^{\prime} .<j j\right\}$
by (auto, case-tac $x=i^{\prime}$, auto)
\{
fix $j$
assume $j: 2 \leq j j<i^{\prime}$
have $\neg j d v d n$
proof
assume $j d v d n$
from prime-divisor $[O F j(1)$ this $]$ obtain $p$ where
$p$ : prime $p$ pdvd $j p d v d n$ by auto
have $p j: p \leq j$
by (rule dvd-imp-le[OF p(2)], insert $j$, auto)
have $p 2$ : $2 \leq p$ using $p(1)$ by (rule prime-ge-2-nat)
from $d v d[O F p 2] p(3)$ have $p i$ : $p \geq i$ by force
from $p j j(2) i^{\prime}$ is $[$ OF $p i-p(1)]$ have $p \in$ set is by auto
with «sorted is〉 have $i^{\prime} \leq p$ by (auto simp: Cons)
with $p j j(2)$ show False by arith
qed
$\}$ note $d v d=t h i s$
from $i^{\prime} i$ have $i^{\prime} 2: 2 \leq$ Suc $i^{\prime}$ by auto
note $I H=I H\left[O F-i^{\prime} 2--i^{\prime}(3)\right.$ sd-iis 1 (10) iis]
\{
fix $x$
assume Suc $i^{\prime} \leq x x<j$ prime $x$
hence $i \leq x x<j j$ prime $x$ using $i^{\prime}$ by auto
from is $[O F$ this] have $x \in$ set is.
with $\left\langle S u c i^{\prime} \leq x\right\rangle$ have $x \in$ set iis unfolding Cons by auto
\} note $i i s=$ this
show ?thesis
proof (cases $i^{\prime}$ dvd $n$ )
case False note $d v d i=t h i s$
\{
fix $j$
assume $j$ : $2 \leq j j<$ Suc $i^{\prime}$
have $\neg j d v d n$
proof (cases $j=i^{\prime}$ )
case False
with $j$ have $j<i^{\prime}$ by auto
from $d v d[O F j(1)$ this] show ?thesis.
qed (insert False, auto)
$\}$ note $d v d s=t h i s$
show ?thesis
proof (cases $i^{\prime} * i^{\prime} \leq n$ )
case True note $i$ iin $=$ this
with res False have res: res $=$ prime-factorization-nat-main $n j j$ iis $p s$ by
auto
from iin have $i-n$ : $i^{\prime}<n$ using dvd dvdi $n$ nat-neq-iff dvd-refl by blast show ?thesis
by (rule $I H\left[O F-n d v d s\right.$ iis res], insert $i-n i^{\prime}$, auto)
next
case False
with res dvdi have res: res $=n \# p s$ by auto
have $n$ : prime $n$
by (rule prime-sqrtI[OF $n$ dvd False])
thus ?thesis unfolding res by auto
qed
next
case True note $i-n=$ this
obtain $n^{\prime \prime} q s$ where rp: remove-prime-factor $i^{\prime}\left(n\right.$ div $\left.i^{\prime}\right)\left(i^{\prime} \# p s\right)=\left(n^{\prime \prime}, q s\right)$
by force
with res True
have res: res $=\left(\right.$ if $n^{\prime \prime}=1$ then qs else prime-factorization-nat-main $n^{\prime \prime}$ jj iis
$q s)$ by auto
have pi: prime $i^{\prime}$ unfolding prime-nat-iff
proof (intro conjI allI impI)
show $1<i^{\prime}$ using $i^{\prime} i$ by auto
fix $j$
assume $j i$ : $j d v d i^{\prime}$
with $i^{\prime} i$ have $j 0: j \neq 0$ by (cases $j=0$, auto)
from $j i i-n$ have $j n$ : $j d v d n$ by (metis dvd-trans)
with $\operatorname{dvd}[o f j]$ have $j$ : $2>j \vee j \geq i^{\prime}$ by linarith
from $j i\left\langle 1<i^{\prime}\right\rangle$ have $j \leq i^{\prime}$ unfolding $d v d$-def
by (simp add: dvd-imp-le $j i$ )
with $j j 0$ show $j=1 \vee j=i^{\prime}$ by linarith
qed
from True $n^{\prime}$ have $i d: n=n^{\prime} * i^{\prime}$ by auto
from $n$ id have $n^{\prime} \neq 0$ by (cases $n=0$, auto)
with $i d$ have $i^{\prime} \leq n$ by auto
from remove-prime-factor $\left[O F \operatorname{rp}[\right.$ folded $\left.n]\left\langle 1<i^{\prime}\right\rangle\left\langle n^{\prime} \neq 0\right\rangle\right]$ obtain rs
where $q s: q s=r s @ i^{\prime} \# p s$ and $n^{\prime}: n^{\prime}=n^{\prime \prime} *$ prod-list rs and $i-n^{\prime \prime}: \neg i^{\prime}$ dvd $n^{\prime \prime}$
and $r s:$ set $r s \subseteq\left\{i^{\prime}\right\}$ by auto
\{
fix $j$
assume $j d v d n^{\prime \prime}$
hence $j d v d n$ unfolding $i d n^{\prime}$ by auto
\} note $d v d^{\prime}=t h i s$
show ?thesis
proof $\left(\right.$ cases $\left.n^{\prime \prime}=1\right)$
case False
with res have res: res $=$ prime-factorization-nat-main $n^{\prime \prime}$ jj iis qs by $\operatorname{simp}$
from $i i^{\prime}$ have $i^{\prime} \geq 2$ by $\operatorname{simp}$
from False $n^{\prime}\left\langle n^{\prime} \neq 0\right\rangle$ have n2: $n^{\prime \prime} \geq 2$ by (cases $n^{\prime \prime}=0$; auto)

```
            have lrs: prod-list rs \not=0 using n' }\langle\mp@subsup{n}{}{\prime}\not=0\rangle\mathrm{ by (cases prod-list rs = 0,
auto)
            with <\mp@subsup{i}{}{\prime}\geq2\ have prod-list rs * ' ' ' 2 2 by (cases prod-list rs, auto)
            hence nn':}n>>n'| unfolding id n' using n2 by sim
            have }\mp@subsup{i}{}{\prime}\not=n\mathrm{ unfolding id n' using pi False by fastforce
            with }\langle\mp@subsup{i}{}{\prime}\leqn\rangle\mp@subsup{i}{}{\prime}\mathrm{ have }n>i\mathrm{ by auto
            with nn"\prime i i' have less: n-i> n'\prime - Suc i' by simp
            {
                fix }
                assume 2: 2 \leq j and ji: j< Suc i'
                have }\negjdvd n"
            proof (cases j= i')
                    case False
                    with ji have j< i' by auto
                    from dvd' dvd[OF 2 this] show ?thesis by blast
                qed (insert i-n'", auto)
            }
            from IH[OF - n2 this iis res] less obtain ss where
                res:res =ss@ qs ^ Ball (set ss) prime }\wedge \mp@subsup{n}{}{\prime\prime}=\mathrm{ prod-list ss by auto
            thus ?thesis unfolding id n' qs using pi rs by auto
        next
            case True
            with res have res: res = qs by auto
            show ?thesis unfolding id n' res qs True using rs <prime i`>
                by (intro exI[of - rs @ [i`], auto)
            qed
    qed
    qed
qed
lemma prime-nat[simp]: prime-nat n = prime n
proof (cases n<2)
    case True
    thus ?thesis unfolding prime-nat-def prime-nat-iff by auto
next
    case False
    hence n: n \geq2 by auto
    obtain jj is where can: next-candidates 0 = (jj,is) by force
    from next-candidates[OF this candidate-invariant-0]
    have cann: sorted is distinct is candidate-invariant jj
        {i.prime i}\wedge0\leqi\wedgei<jj}\subseteq set is
        set is \subseteq{2..}\cap{0..<jj} distinct is is }\not=[] by aut
    from cann have sub: set is \subseteq{2..<jj} by force
```



```
    from n can have res: prime-nat n = prime-nat-main n jj is
    unfolding prime-nat-def by auto
    show ?thesis using prime-nat-main[OF refl le-refl n - jj cann(1-3) sub res]
cann(4) by auto
qed
```

```
lemma prime-factorization-nat: fixes n :: nat
    defines pf \equiv prime-factorization-nat n
    shows Ball (set pf) prime
    and n\not=0\Longrightarrow prod-list pf = n
    and n=0\Longrightarrowpf=[]
proof -
    note pf = pf-def[unfolded prime-factorization-nat-def]
    have Ball (set pf) prime ^(n\not=0\longrightarrow prod-list pf = n)^(n=0\longrightarrowpf=[])
    proof (cases n<2)
        case True
        thus ?thesis using pf by auto
    next
        case False
        hence n: n \geq2 by auto
        obtain jj is where can: next-candidates 0 = (jj,is) by force
    from next-candidates[OF this candidate-invariant-0]
    have cann: sorted is distinct is candidate-invariant jj
            {i.prime i}\wedge0\leqi\wedgei<jj}\subseteq set i
            set is \subseteq{2..}\cap{0..<jj} distinct is is }\not=[]\mathrm{ by auto
    from cann have sub: set is \subseteq{2..<jj} by force
    with «is \not= []〉 have jj: jj \geq2 by (cases is, auto)
    let ?pfm = prime-factorization-nat-main n jj is []
    from pf[unfolded can] False
    have res: pf = rev ?pfm by simp
    from prime-factorization-nat-main[OF refl le-refl n--jj cann(1-3) sub refl,
of Nil] cann(4)
    have Ball (set ?pfm) prime n = prod-list ?pfm by auto
    thus ?thesis unfolding res using n}\mathrm{ by auto
    qed
    thus Ball (set pf) prime n}\not=0\Longrightarrow\mathrm{ prod-list pf = n n=0 p pf=[] by auto
qed
lemma prod-mset-multiset-prime-factorization-nat [simp]:
    (x::nat) }=0\Longrightarrow\mathrm{ prod-mset (prime-factorization }x\mathrm{ ) = x
    by simp
lemma prime-factorization-unique":
    fixes A :: 'a :: {factorial-semiring-multiplicative} multiset
    assumes }\p.p\in#A\Longrightarrow\mathrm{ prime p
    assumes prod-mset A = normalize x
    shows prime-factorization x}=
proof -
    have prod-mset A\not=0 by (auto dest: assms(1))
    with assms(2) have x\not=0 by simp
    hence prod-mset (prime-factorization x) = prod-mset A
    by (simp add: assms prod-mset-prime-factorization)
    with assms show ?thesis
```

```
    by (intro prime-factorization-unique) auto
qed
lemma multiset-prime-factorization-nat-correct:
    prime-factorization n = mset (prime-factorization-nat n)
proof -
    note pf = prime-factorization-nat[of n]
    show ?thesis
    proof (cases n=0)
        case True
        thus ?thesis using pf(3) by simp
    next
        case False
        note pf=pf(1) pf(2)[OF False]
        show ?thesis
        proof (rule prime-factorization-unique")
            show prime p if p\in# mset (prime-factorization-nat n) for p
                using pf(1) that by simp
            let ?l = \i\in#prime-factorization n. i
            let ?r = \i\in#mset (prime-factorization-nat n). i
            show prod-mset (mset (prime-factorization-nat n)) = normalize n
                by (simp add: pf(2) prod-mset-prod-list)
    qed
    qed
qed
lemma multiset-prime-factorization-code[code-unfold]:
    prime-factorization = (\lambdan. mset (prime-factorization-nat n))
    by (intro ext multiset-prime-factorization-nat-correct)
lemma divisors-nat:
    n\not=0\Longrightarrowset (divisors-nat n)={p.pdvd n} distinct (divisors-nat n) divisors-nat
O=[]
proof -
    show distinct (divisors-nat n) divisors-nat 0 = [] unfolding divisors-nat-def by
auto
    assume n: n\not=0
    from n have n>0 by auto
    {
        fix }
        have (x dvd n) =(x\not=0^(\forallp. multiplicity p x \leq multiplicity p n))
        proof (cases x = 0)
            case False
            with \langlen> 0\rangle show ?thesis by (auto simp: dvd-multiplicity-eq)
        next
            case True
            with n show ?thesis by auto
        qed
    } note dvd = this
```

```
    let ?dn = set (divisors-nat n)
    let ?mf = \lambda(n :: nat). prime-factorization n
    have ?dn = prod-list'set (subseqs (prime-factorization-nat n)) unfolding divi-
sors-nat-def
    using n by auto
    also have ... = prod-mset' mset'set (subseqs (prime-factorization-nat n))
    by (force simp: prod-mset-prod-list)
    also have mset'set (subseqs (prime-factorization-nat n))
    ={ ps.ps\subseteq# mset (prime-factorization-nat n)}
    unfolding multiset-of-subseqs by simp
    also have ...={ ps.ps\subseteq# ?mf n}
    thm multiset-prime-factorization-code[symmetric]
    unfolding multiset-prime-factorization-nat-correct[symmetric] by auto
    also have prod-mset ' }\ldots={p.pdvdn} (is ?l = ?r
    proof -
    {
        fix }
        assume x dvd n
        from this[unfolded dvd] have }x:x\not=0\mathrm{ by auto
        from }\langlex\mathrm{ dvd n><x}\not=0\rangle\langlen\not=0\rangle have sub: ?mf x\subseteq# ?mf 
            by (subst prime-factorization-subset-iff-dvd) auto
        have prod-mset (?mf x)=x using }
            by (simp add: prime-factorization-nat)
        hence }x\in\mathrm{ ?l using sub by force
    }
    moreover
    {
        fix }
        assume }x\in?
        then obtain ps where x: x= prod-mset ps and sub:ps\subseteq# ?mf n by auto
        have x dvd n using prod-mset-subset-imp-dvd[OF sub] n x by simp
    }
    ultimately show ?thesis by blast
    qed
    finally show set (divisors-nat n)={p.p dvd n} .
qed
lemma divisors-int-pos: x\not=0\Longrightarrow set(divisors-int-pos x)={i. i dvd x\wedgei>0}
distinct (divisors-int-pos x)
    divisors-int-pos 0 = []
proof -
    show divisors-int-pos 0 = [] by code-simp
    show distinct (divisors-int-pos x)
    unfolding divisors-int-pos-def using divisors-nat(2)[of nat (abs x)]
    by (simp add: distinct-map inj-on-def)
    assume x: x\not=0
    let ?x = nat (abs x)
    from }x\mathrm{ have }xx:?x\not=0\mathrm{ by auto
    from }x\mathrm{ have 0: \ y.y dvd x ב y =0 by auto
```

```
    have id: int ' {p. int p dvd x}={i.idvd x\wedge0<i} (is ?l = ?r)
    proof -
    {
        fix y
        assume y f?l
        then obtain p where y: y= int p and dvd: int p dvd x by auto
        have }y\in?r\mathrm{ unfolding y using dvd O[OF dvd] by auto
    }
    moreover
    {
        fix y
        assume y \in?r
        hence dvd:y dvd x and y0:y>0 by auto
        define }n\mathrm{ where }n=\mathrm{ nat }
        from y0 have y:}y=\mathrm{ int n unfolding n-def by auto
        with dvd have }y\in?l\mathrm{ by auto
    }
    ultimately show ?thesis by blast
    qed
    from xx show set (divisors-int-pos }x\mathrm{ ) = {i. i dvd x}\wedgei>0
    by (simp add: divisors-int-pos-def divisors-nat id)
qed
lemma divisors-int: x\not=0\Longrightarrow set (divisors-int x)={i.i dvd x} distinct (divisors-int
x)
    divisors-int 0 = []
proof -
    show divisors-int 0 = [] by code-simp
    show distinct (divisors-int x)
    proof (cases x=0)
        case True
        show ?thesis unfolding True by code-simp
    next
        case False
        from divisors-int-pos(1)[OF False] divisors-int-pos(2)
            show ?thesis unfolding divisors-int-def Let-def distinct-append distinct-map
inj-on-def by auto
    qed
    assume x: x\not=0
    show set (divisors-int x) = {i.i dvd x}
    unfolding divisors-int-def Let-def set-append set-map divisors-int-pos(1)[OF x]
using }
    by auto (metis (no-types, lifting) dvd-mult-div-cancel image-eqI linorder-neqE-linordered-idom
    mem-Collect-eq minus-dvd-iff minus-minus mult-zero-left neg-less-0-iff-less)
qed
definition divisors-fun ::('a m ('a :: {comm-monoid-mult,zero}) list) => bool
where
```

divisors-fun $d f \equiv(\forall x . x \neq 0 \longrightarrow$ set $(d f x)=\{d . d$ dvd $x\}) \wedge(\forall x$. distinct (df $x)$ )
lemma divisors-funD: divisors-fun $d f \Longrightarrow x \neq 0 \Longrightarrow d d v d x \Longrightarrow d \in \operatorname{set}(d f x)$ unfolding divisors-fun-def by auto
definition divisors-pos-fun :: (' $a \Rightarrow\left({ }^{\prime} a::\{\right.$ comm-monoid-mult,zero,ord $\left.\}\right)$ list $) \Rightarrow$ bool where
divisors-pos-fun $d f \equiv(\forall x . x \neq 0 \longrightarrow \operatorname{set}(d f x)=\{d . d d v d x \wedge d>0\}) \wedge(\forall$ x. distinct (df $x)$ )
lemma divisors-pos-funD: divisors-pos-fun $d f \Longrightarrow x \neq 0 \Longrightarrow d$ dvd $x \Longrightarrow d>0$ $\Longrightarrow d \in \operatorname{set}(d f x)$
unfolding divisors-pos-fun-def by auto
lemma divisors-fun-nat: divisors-fun divisors-nat unfolding divisors-fun-def using divisors-nat by auto
lemma divisors-fun-int: divisors-fun divisors-int
unfolding divisors-fun-def using divisors-int by auto
lemma divisors-pos-fun-int: divisors-pos-fun divisors-int-pos
unfolding divisors-pos-fun-def using divisors-int-pos by auto
end

## 8 Rational Root Test

This theory contains a formalization of the rational root test, i.e., a decision procedure to test whether a polynomial over the rational numbers has a rational root.

## theory Rational-Root-Test

 importsGauss-Lemma
Missing-List
Prime-Factorization

## begin

definition rational-root-test-main ::

```
    (int \(\Rightarrow\) int list \() \Rightarrow(\) int \(\Rightarrow\) int list \() \Rightarrow\) rat poly \(\Rightarrow\) rat option where
    rational-root-test-main df dp \(p \equiv\) let ip = snd (rat-to-normalized-int-poly \(p\) );
        \(a 0=\) coeff ip 0; an \(=\) coeff \(i p(\) degree \(i p)\)
        in if \(a 0=0\) then Some 0 else
        let \(d 0=d f a 0 ; d n=d p a n\)
        in map-option fst
        (find-map-filter \((\lambda x .(x, p o l y p x))\)
        \((\lambda(-\), res \()\). res \(=0)[\) rat-of-int b0 / of-int bn.b0 \(<-d 0, b n<-d n\), coprime
b0 bn ])
```

```
definition rational-root-test :: rat poly \(\Rightarrow\) rat option where
    rational-root-test \(p=\)
        rational-root-test-main divisors-int divisors-int-pos \(p\)
lemma rational-root-test-main:
    rational-root-test-main df dp \(p=\) Some \(x \Longrightarrow\) poly \(p x=0\)
    divisors-fun \(d f \Longrightarrow\) divisors-pos-fun \(d p \Longrightarrow\) rational-root-test-main \(d f d p \quad p=\)
None \(\Longrightarrow \neg(\exists\) x. poly \(p x=0)\)
proof -
    let \(? r=r a t-o f-i n t\)
    let ?rp = map-poly ?r
    obtain \(a\) ip where rp: rat-to-normalized-int-poly \(p=(a, i p)\) by force
    from rat-to-normalized-int-poly \([\) OF this \(]\) have \(p: p=s m u l t ~ a(? r p ~ i p) ~ a n d ~ a 00: ~\)
\(a \neq 0\)
    and cip: \(p \neq 0 \Longrightarrow\) content \(i p=1\) by auto
    let ? \(a 0=\) coeff ip 0
    let ?an \(=\) coeff \(i p(\) degree \(i p)\)
    let \(? d 0=d f ? a 0\)
    let ? \(d n=d p\) ?an
    let ? ip = ? \(r p\) ip
    define tests where tests \(=[\) rat-of-int b0 / rat-of-int bn . b0 \(<-\) ? \(d 0, b n<-\)
?dn, coprime b0 bn ]
    let \(? f=(\lambda x .(x\), poly \(p x))\)
    let ?test \(=(\lambda(-\), res \()\). res \(=0)\)
    define mo where mo = find-map-filter ?f ?test tests
    note \(d=\) rational-root-test-main-def[of df \(d p\) p, unfolded Let-def rp snd-conv
mo-def[symmetric] tests-def[symmetric]]
    \{
        assume rational-root-test-main \(d f\) dp \(p=\) Some \(x\)
        from this[unfolded d] have ? \(a 0=0 \wedge x=0 \vee\) map-option fst mo \(=\) Some \(x\)
by (auto split: if-splits)
    thus poly \(p x=0\)
    proof
            assume \(*: ~ ? a 0=0 \wedge x=0\)
            hence coeff p \(0=0\) unfolding \(p\) coeff-smult by simp
            hence poly \(p 0=0\) by (cases \(p\), auto)
            with \(*\) show ?thesis by auto
    next
            assume map-option fst mo \(=\) Some \(x\)
            then obtain pair where find: find-map-filter ?f ?test tests \(=\) Some pair and
\(x: x=\) fst pair
            unfolding mo-def by (auto split: option.splits)
            then obtain \(z\) where pair: pair \(=(x, z)\) by (cases pair, auto)
            from find-map-filter-Some[OF find, unfolded pair split] show poly p \(x=0\)
by auto
    qed
    \}
assume \(d f\) : divisors-fun \(d f\) and \(d p\) : divisors-pos-fun \(d p\) and res: rational-root-test-main
```

$d f d p p=$ None
note $d f=$ divisors-fun $D\left[\begin{array}{ll}O F & d f\end{array}\right]$ note $d p=$ divisors-pos-fun $D\left[\begin{array}{ll}O F & d p\end{array}\right]$
from res[unfolded $d]$ have $a 0: ? a 0 \neq 0$ and res: map-option fst mo $=$ None by (auto split: if-splits)
from res[unfolded mo-def] have find: find-map-filter ?f ?test tests $=$ None by auto
show $\neg(\exists x$. poly $p x=0)$
proof
assume $\exists$ x. poly $p x=0$
then obtain $x$ where poly $p x=0$ by auto
from this[unfolded $p$ ] a00 have poly (? ? ip) $x=0$ by auto
from this[unfolded poly-eq-0-iff-dvd] have $[:-x, 1$ :] dvd ?ip by auto
then obtain $q$ where $i p-i d:$ ? $i p=[:-x, 1:] * q$ unfolding dvd-def by auto
obtain $c q$ where $x 1$ : rat-to-normalized-int-poly $[:-x, 1:]=(c, q)$ by force
from rat-to-int-factor-explicit[OF ip-id x1] obtain $r$ where $i p: i p=q * r$ by blast
from rat-to-normalized-int-poly(4)[OF x1] have deg: degree $q=1$ by auto
from degree1-coeffs[OF deg] obtain $a b$ where $q: q=[: b, a:]$ and $a: a \neq 0$ by auto
have $i p r: i p=[: b, a:] * r$ using $i p q$ by auto
from arg-cong[OF ipr, of $\lambda$ p. coeff $p$ 0] have ba0: b dvd ?a0 by auto
have $r p q$ : ? $r p ~ q=[:$ ? $r b$, ?r $a:]$ unfolding $q$
proof (rule poly-eqI, unfold of-int-hom.coeff-map-poly-hom)
fix $n$
show ?r (coeff $[: b, a:] n)=\operatorname{coeff}[:$ ?r $b$, ?r $a:] n$
unfolding coeff-pCons
by (cases $n$, force, cases $n-1$, auto)
qed
from arg-cong[OF ip, of ?rp, unfolded of-int-poly-hom.hom-mult rpq] have [: ?r $b$, ?r $a:] d v d$ ?rp ip
unfolding dvd-def by blast
hence smult (inverse (?r a)) [: ?r b , ?r a :] dvd ?rp ip by (rule smult-dvd, insert a, auto)
also have smult (inverse (?r a) ) $[:$ ?r $b$, ?r $a:]=[:$ ?r b / ?r $a, 1$ :] using $a$ by (simp add: field-simps)
finally have $[:-(-$ ?r $b /$ ? $r ~ a), 1$ :] dvd ? $r p$ ip by simp
from this[unfolded poly-eq-O-iff-dvd[symmetric]]
have rt: poly (?rp ip) $(-$ ?r $b /$ ? $r ~ a)=0$.
hence rt: poly $p(-$ ?r $b /$ ? $r ~ a)=0$
unfolding $p$ using a00 by simp
obtain $a a b b$ where quot: quotient-of $(-? r b / ? r a)=(b b, a a)$ by force
hence quotient-of $(? r(-b) /$ ?r $a)=(b b, a a)$ by simp
from quotient-of-int-div[OF this $\langle a \neq 0\rangle$ ] obtain $z$ where
$z: z \neq 0$ and $b:-b=z * b b$ and $a: a=z * a a$ by auto
from $r$ t[unfolded quotient-of-div[OF quot $]$ ] have $r$ : poly $p($ ?r $b b /$ ?r $a a)=0$ by auto
from quotient-of-coprime $[O F$ quot $]$ have cop: coprime bb aa coprime $(-b b)$ aa by auto
from quotient-of-denom-pos[OF quot $]$ have $a a: a a>0$ by auto

```
    from ba0 arg-cong[OF b, of uminus] z have bba0: bb dvd ?a0 unfolding dvd-def
        by (metis ba0 dvdE dvd-mult-right minus-dvd-iff)
    hence bb0: bb\not=0 using a0 by auto
    from df[OF a0 bba0] have bb: bb\in set ? d0 by auto
    from a0 have ip0:ip\not=0 by auto
    hence an0: ?an \not=0 by auto
    from ipr ip0 have r}\not=0\mathrm{ by auto
    from degree-mult-eq[OF - this, of [:b,a:], folded ipr]<a\not=0\rangleipr
    have deg: degree ip =Suc (degree r) by auto
    from arg-cong[OF ipr, of \lambda p. coeff p (degree ip)] have ba0: a dvd ?an
        unfolding deg by (auto simp: coeff-eq-0)
    hence aa dvd ?an using <a\not=0\rangle unfolding a by (auto simp: dvd-def)
    from dp[OF an0 this aa] have aa: aa \in set ?dn .
    from find-map-filter-None[OF find] rt have (?r bb / ?r aa) & set tests by auto
    note test = this[unfolded tests-def, simplified, rule-format, of - aa]
    from this[of bb] cop bb aa
    show False by auto
    qed
qed
lemma rational-root-test:
    rational-root-test p=Some x moly p x = 0
    rational-root-test p}=\mathrm{ None }\Longrightarrow\neg(\existsx\mathrm{ . poly p x = 0)
    using rational-root-test-main(1) rational-root-test-main(2)[OF divisors-fun-int
divisors-pos-fun-int]
    unfolding rational-root-test-def by blast+
```

end

## 9 Kronecker Factorization

This theory contains Kronecker's factorization algorithm to factor integer or rational polynomials.

```
theory Kronecker-Factorization
imports
    Polynomial-Interpolation.Polynomial-Interpolation
    Sqrt-Babylonian.Sqrt-Babylonian-Auxiliary
    Missing-List
    Prime-Factorization
    Precomputation
    Gauss-Lemma
    Dvd-Int-Poly
begin
```


### 9.1 Definitions

context

$$
\begin{aligned}
& \text { fixes } d f:: \text { int } \Rightarrow \text { int list } \\
& \text { and } d p:: \text { int } \Rightarrow \text { int list } \\
& \text { and } b n d:: \text { nat } \\
& \text { begin }
\end{aligned}
$$

definition kronecker-samples :: nat $\Rightarrow$ int list where
kronecker-samples $n \equiv$ let min $=-\operatorname{int}(n \operatorname{div} 2)$ in $[\min . . \min +\operatorname{int} n]$
lemma kronecker-samples-0: $0 \in$ set (kronecker-samples n) unfolding kronecker-samples-def by auto

Since 0 is always a samples value, we make a case analysis: we only take positive divisors of $p(0)$, and consider all divisors for other $p(j)$.
definition kronecker-factorization-main :: int poly $\Rightarrow$ int poly option where
kronecker-factorization-main $p \equiv$ if degree $p \leq 1$ then None else let
$p=$ primitive-part $p$;
$j s=k r o n e c k e r-s a m p l e s ~ b n d ;$
$c j s=\operatorname{map}(\lambda j .($ poly $p j, j)) j s$
in (case map-of cjs 0 of
Some $j \Rightarrow$ Some ([:- j, 1 :])
$\mid$ None $\Rightarrow$ let djs $=\operatorname{map}(\lambda(v, j) . \operatorname{map}($ Pair $j)($ if $j=0$ then $d p v$ else df $v))$ cjs in
map-option the (find-map-filter newton-interpolation-poly-int
( $\lambda$ go. case go of None $\Rightarrow$ False $\mid$ Some $g \Rightarrow$ dvd-int-poly-non-0 g $p \wedge$ degree $g$
$\geq 1$ )
(concat-lists djs)))
definition kronecker-factorization-rat-main :: rat poly $\Rightarrow$ rat poly option where kronecker-factorization-rat-main $p \equiv$ map-option (map-poly of-int) (kronecker-factorization-main (snd (rat-to-normalized-int-poly p)))
end
definition kronecker-factorization :: int poly $\Rightarrow$ int poly option where
kronecker-factorization $p=$ kronecker-factorization-main divisors-int divisors-int-pos (degree p div 2) $p$
definition kronecker-factorization-rat :: rat poly $\Rightarrow$ rat poly option where kronecker-factorization-rat $p=$ kronecker-factorization-rat-main divisors-int divisors-int-pos (degree $p$ div 2) $p$

### 9.2 Code setup for divisors

definition divisors-nat-copy $n \equiv$ if $n=0$ then [] else remdups-adj (sort (map prod-list (subseqs (prime-factorization-nat n))))
lemma divisors-nat-copy[simp]: divisors-nat-copy $=$ divisors-nat unfolding divisors-nat-def[abs-def] divisors-nat-copy-def[abs-def] ..
definition memo-divisors-nat $\equiv$ memo-nat 0100 divisors-nat-copy
lemma memo-divisors-nat[code-unfold]: divisors-nat $=$ memo-divisors-nat unfolding memo-divisors-nat-def by simp

### 9.3 Proofs

## context

begin
lemma rat-to-int-poly-of-int: assumes rp: rat-to-int-poly (map-poly of-int p) = $(c, q)$
shows $c=1 q=p$
proof -
define $x s$ where $x s=\operatorname{map}($ snd $\circ$ quotient-of) (coeffs (map-poly rat-of-int p))
have $x s$ : set $x s \subseteq\{1\}$ unfolding $x s$-def by auto
from assms[unfolded rat-to-int-poly-def Let-def]
have $c: c=f s t$ (common-denom (coeffs (map-poly rat-of-int $p$ )) ) by auto
also have $\ldots=$ list-lcm xs
unfolding common-denom-def Let-def xs-def by (simp add: o-assoc)
also have $\ldots=1$ using $x s$
by (induct xs, auto)
finally show $c: c=1$ by auto
from rat-to-int-poly[OF rp, unfolded $c]$ show $q=p$ by auto
qed
lemma rat-to-normalized-int-poly-of-int: assumes rat-to-normalized-int-poly (map-poly of-int $p)=(c, q)$
shows $c \in \mathbb{Z} p \neq 0 \Longrightarrow c=$ of-int (content $p) \wedge q=$ primitive-part $p$ proof -
obtain $d r$ where ri: rat-to-int-poly (map-poly rat-of-int $p)=(d, r)$ by force
from rat-to-int-poly-of-int[OF ri]
assms[unfolded rat-to-normalized-int-poly-def ri split]
show $c \in \mathbb{Z} p \neq 0 \Longrightarrow c=o f-i n t($ content $p) \wedge q=$ primitive-part $p$
by (auto split: if-splits)
qed
lemma dvd-poly-int-content-1: assumes $c$ - $x$ : content $x=1$
shows $(x$ dvd $y)=($ map-poly rat-of-int $x$ dvd map-poly of-int $y)$
proof -
let $? r=r a t-o f-i n t$
let ? $r p=$ map-poly ?r
show ?thesis
proof
assume $x d v d y$
then obtain $z$ where $y=x * z$ unfolding dvd-def by auto
from $\arg$-cong[OF this, of ? $r p]$
show ?rp $x$ dvd ?rp y by auto
next
assume dvd: ? $r p x d v d$ ? $r p y$

```
    show x dvd y
    proof (cases y=0)
    case True
    thus ?thesis by auto
    next
    case False note y0 = this
    hence ?rp y}=0\mathrm{ by simp
    hence rx0: ?rp x\not=0 using dvd by auto
    hence x0: x\not=0 by simp
    from dvd obtain z}\mathrm{ where prod: ?rp y = ?rp }x*z\mathrm{ unfolding dvd-def by
auto
    obtain cx xx where x: rat-to-normalized-int-poly (?rp x) = (cx, xx) by force
    from rat-to-int-factor-explicit[OF prod x] obtain z where y: y=xx* smult
(content y) z by auto
    from rat-to-normalized-int-poly[OF x] rx0 have xx: ?rp x = smult cx (?rp
xx)
            and cxx: content }xx=1\mathrm{ and cx0:cx> 0 by auto
    obtain cn cd where quot: quotient-of cx = (cn,cd) by force
    from quotient-of-div[OF quot] have cx: cx = ?r cn / ?r cd by auto
    from quotient-of-denom-pos[OF quot] have cd0:cd > 0 by auto
    with cx cx0 have cn0: cn > 0 by (simp add: zero-less-divide-iff)
    from arg-cong[OF xx, of smult (?r cd)] have smult (?r cd) (?rp x) = smult
(?r cn) (?rp xx )
            unfolding cx using cd0 by (auto simp: field-simps)
            from this have id: smult cd x = smult cn xx by (fold hom-distribs, unfold
of-int-poly-hom.eq-iff)
            from arg-cong[OF this, of content, unfolded content-smult-int cxx] cn0 cd0
            have cn:cn =cd * content x by auto
            from quotient-of-coprime[OF quot, unfolded cn] cd0 have cd =1 by auto
            with cx have cx:cx=?r cn by auto
            from xx[unfolded this] have x: x= smult cn xx by (fold hom-distribs, simp)
            from arg-cong[OF this, of content, unfolded content-smult-int c-x cxx] cn0
have }cn=1\mathrm{ by auto
            with }x\mathrm{ have }xx:xx=x\mathrm{ by auto
            show }x\mathrm{ dvd y using y[unfolded xx] unfolding dvd-def by blast
    qed
    qed
qed
lemma content-x-minus-const-int[simp]: content [:c, 1 :] = (1 :: int)
    unfolding content-def by auto
lemma length-upto-add-nat[simp]:length [a .. a + int n] = Suc n
proof (induct n arbitrary: a)
    case (0 a)
    show ?case using upto.simps[of a a] by auto
next
    case (Suc n a)
    from Suc[of a + 1]
```

show ?case using upto.simps[of a a $\operatorname{lint}$ (Suc n)] by (auto simp: ac-simps) qed
lemma kronecker-samples: distinct (kronecker-samples n) length (kronecker-samples $n)=$ Suc $n$
unfolding kronecker-samples-def Let-def length-upto-add-nat by auto
lemma dvd-int-poly-non-0-degree-1[simp]: degree $q \geq 1 \Longrightarrow$ dvd-int-poly-non-0 $q$ $p=(q d v d p)$
by (intro dvd-int-poly-non-0, auto)
context fixes $d f d p::$ int $\Rightarrow$ int list
and bnd :: nat
begin
lemma kronecker-factorization-main-sound: assumes some: kronecker-factorization-main $d f$ dp bnd $p=$ Some $q$
and bnd: degree $p \geq 2 \Longrightarrow b n d \geq 1$
shows degree $q \geq 1$ degree $q \leq b n d$ $q$ dvd $p$
proof -
let ?r $=$ rat-of-int
let ${ }^{2} r p=$ map-poly $?$ r
note res $=$ some[unfolded kronecker-factorization-main-def Let-def]
from res have $d p$ : degree $p \geq 2$ and (degree $p \leq 1$ ) $=$ False by (auto split:
if-splits)
note res $=$ res[unfolded this if-False]
note $b n d=b n d[O F d p]$
define $P$ where $P=$ primitive-part $p$
have deg $P$ : degree $P=$ degree $p$ unfolding $P$-def by simp
define $j s$ where $j s=k r o n e c k e r-s a m p l e s ~ b n d ~$
define filt where filt $=$ (case-option False ( $\lambda$ g. dvd-int-poly-non-0 g $P \wedge 1 \leq$ degree g))
define tests where tests $=$ concat-lists $(\operatorname{map}(\lambda(v, j) . \operatorname{map}($ Pair $j)($ if $j=0$ then $d p v$ else $d f v)$ ) (map $(\lambda j$. (poly $P j, j)) j s))$
note res $=$ res[folded $P$-def, folded js-def filt-def, folded tests-def]
let ?zero $=\operatorname{map}(\lambda j .($ poly $P j, j)) j s$
from res have res: (case map-of ?zero 0 of
None $\Rightarrow$ map-option the (find-map-filter newton-interpolation-poly-int filt tests)
Some $j \Rightarrow$ Some $[:-j, 1:])=$
Some $q$ by auto
have degree $q \geq 1 \wedge$ degree $q \leq b n d \wedge q$ dvd $P$
proof (cases map-of ?zero 0)
case (Some $j$ )
with res have $q: q=[:-j, 1:]$ by auto
from map-of-SomeD[OF Some] have 0: poly $P j=0$ by auto
hence poly $($ ?rp $P)(? r j)=0$ by simp
hence [: - ?r j, 1 :] dvd ?rp $P$ using poly-eq- 0 -iff-dvd by blast
also have $[:-? r j, 1:]=$ ? $r p q$ unfolding $q$ by simp
finally have $d v d:$ ? $r p q d v d$ ? $r p P$.
have $q$ dvd $P$
by (subst dvd-poly-int-content-1, insert dvd q, auto)
with $q d p$ bnd show ?thesis by auto
next
case None
from res[unfolded None]
have res: map-option the (find-map-filter newton-interpolation-poly-int filt tests)
$=$ Some $q$ by auto
then obtain $q q$ where
res: find-map-filter newton-interpolation-poly-int filt tests $=$ Some $q q$ and $q$ :
$q=$ the $q q$
by (auto split: option.splits)
from find-map-filter-Some[OF res]
have filt: filt $q q$ and tests: $q q \in$ newton-interpolation-poly-int' set tests by auto
from filt[unfolded filt-def] $q$ obtain $g$ where $d v d: g d v d P$ and $d g: 1 \leq$ degree
$g$ and $q q: q q=$ Some $g$
by (cases qq, auto)
from $q q q$ have $g q: g=q$ by auto
from tests obtain $t$ where $t: t \in$ set tests and $l$ : newton-interpolation-poly-int $t=$ Some $g$ unfolding $q q$
by auto
from $t[u n f o l d e d ~ t e s t s-d e f] ~$
have lent: length $t=$ length $j s$ and $\wedge i . i<$ length $j s \Longrightarrow$ map fst $t!i=j s!$
$i$ by auto
hence $i d$ : map fst $t=j s$
by (intro nth-equalityI, auto)
have dist: distinct $j$ s and lenj: length js $=$ Suc bnd unfolding js-def degP using kronecker-samples by auto
from newton-interpolation-poly-int-Some[OF dist[folded id] l, unfolded lent lenj]
have degree $g \leq b n d$ by auto
with $d v d d g$ show ?thesis unfolding $g q$ by auto
qed note main $=$ this
thus degree $q \geq 1$ degree $q \leq b n d$ by auto
from content-times-primitive-part $[$ of $p]$ have $p=\operatorname{smult}$ (content $p$ ) $P$ unfolding $P$-def by auto
with main show $q$ dvd $p$ by (metis dvd-smult)
qed
lemma kronecker-factorization-rat-main-sound: assumes
some: kronecker-factorization-rat-main df $d p$ bnd $p=$ Some $q$
and $b n d$ : degree $p \geq 2 \Longrightarrow b n d \geq 1$
shows degree $q \geq 1$ degree $q \leq$ bnd $q$ dvd $p$

## proof -

let $? r=$ rat-of-int
let $? r p=$ map-poly $? r$
let ? $p=$ rat-to-normalized-int-poly $p$
obtain a $P$ where $r p$ : $p=(a, P)$ by force
from rat-to-normalized-int-poly[OF this] have $p: p=s m u l t a(? r p P)$ and $a: a$ $\neq 0$
and deg: degree $P=$ degree $p$ by auto
from some[unfolded kronecker-factorization-rat-main-def rp]
obtain $Q$ where some: kronecker-factorization-main df $d p$ bnd $P=$ Some $Q$ and $q: q=? r p Q$ by auto from kronecker-factorization-main-sound $[$ OF some bnd $]$ have $d Q: 1 \leq$ degree $Q$
degree $Q \leq b n d$
and dvd: $Q$ dvd $P$ unfolding deg by auto
from $d v d$ obtain $R$ where $P Q R: P=Q * R$ unfolding dvd-def by auto
from $p[$ unfolded arg-cong[OF this, of ? $r p]]$
have $p=q *$ smult a (? $r p$ ) unfolding $q$ by (auto simp: hom-distribs)
thus $q$ dvd $p$ unfolding dvd-def by blast
from $q d Q$ show degree $q \geq 1$ degree $q \leq b n d$ by auto
qed

## context

assumes $d f$ : divisors-fun $d f$ and $d p f$ : divisors-pos-fun $d p$
begin
lemma kronecker-factorization-main-complete: assumes none: kronecker-factorization-main $d f d p$ bnd $p=$ None and $d p$ : degree $p \geq 2$ shows $\neg(\exists q .1 \leq$ degree $q \wedge$ degree $q \leq b n d \wedge q d v d p)$
proof -
let $? r=r a t-o f-i n t$
let $? r p=$ map-poly $? r$
from $d p$ have (degree $p \leq 1$ ) = False by auto
note res $=$ none[unfolded kronecker-factorization-main-def Let-def this if-False]
define $P$ where $P=$ primitive-part $p$
have $\operatorname{deg} P$ : degree $P=$ degree $p$ unfolding $P$-def by simp

define filt where filt $=($ case-option False $(\lambda g$. dvd-int-poly-non-0 g $P \wedge 1 \leq$ degree g))
define tests where tests $=$ concat-lists $(\operatorname{map}(\lambda(v, j) . \operatorname{map}(\operatorname{Pair} j)($ if $j=0$
then $d p v$ else df $v))(\operatorname{map}(\lambda j$. (poly $P j, j)) j s))$
note res $=$ res $[$ folded $P$-def, folded js-def filt-def, folded tests-def]
let ?zero $=\operatorname{map}(\lambda j$. $($ poly $P j, j)) j s$
from res have res: (case map-of ?zero 0 of
None $\Rightarrow$ map-option the (find-map-filter newton-interpolation-poly-int filt tests)
| Some $j \Rightarrow$ Some $[:-j, 1:])=$
None by auto
hence zero: map-of ?zero $0=$ None by (auto split: option.splits)
with res have res: find-map-filter newton-interpolation-poly-int filt tests $=$ None by auto
\{ fix $q q$
assume $q q: 1 \leq$ degree $q q$ degree $q q \leq b n d$ and $d v d: q q$ dvd $p$
define $q^{\prime}$ where $q^{\prime}=$ primitive-part $q q$
define $q$ where $q=\left(\right.$ if poly $q^{\prime} 0>0$ then $q^{\prime}$ else $-q^{\prime}$ )
from $q q$ have $q^{\prime}: 1 \leq$ degree $q^{\prime}$ degree $q^{\prime} \leq$ bnd unfolding $q^{\prime}$-def by auto
hence $q: 1 \leq$ degree $q$ degree $q \leq$ bnd unfolding $q$-def by auto
from $d v d$ have $q q d v d$ (smult (content $p$ ) P)
using content-times-primitive-part $[$ of $p]$ unfolding $P$-def by simp
from dvd-smult-int [OF - this] $d p$ have $q^{\prime} d v d P$ unfolding $q^{\prime}$-def
by force
hence $d v d: q$ dvd $P$ unfolding $q$-def by auto
then obtain $r$ where $P: P=q * r$ unfolding dvd-def by auto
\{
fix $j$
assume $j: j \in$ set $j s$
from $P$ have $i d$ : poly $P j=$ poly $q j *$ poly $r j$ by auto
hence dvd: poly $q j$ dvd poly $P j$ by auto
from $j$ have (poly $P j, j) \in$ set ?zero by auto
with zero have zero: poly $P j \neq 0$ unfolding map-of-eq-None-iff by force
with id have poly $q j \neq 0$ by auto
hence $j=0 \Longrightarrow$ poly $q j>0$ unfolding $q$-def by auto
from divisors-funD[OF df zero dvd] divisors-pos-funD[OF dpf zero dvd this]
have poly $q j \in \operatorname{set}(d f($ poly $P j)) j=0 \Longrightarrow \operatorname{poly} q j \in \operatorname{set}(d p(p o l y P j))$.
$\}$ note $m e m 1=$ this
define $t$ where $t=\operatorname{map}(\lambda j$. $(j$, poly $q j))$ js
have $t: t \in$ set tests unfolding tests-def concat-lists-listset listset length-map
map-map o-def
proof (rule, intro conjI allI impI)
show length $t=$ length $j s$ unfolding $t$-def by simp
fix $i$
assume $i: i<$ length $j s$
hence $j s i: j s!i \in$ set $j s$ by auto
have $t i: t!i=(j s!i$, poly $q(j s!i))$ unfolding $t$-def using $i$ by auto
let ?f $=(\lambda x$. set (case (poly P $x, x)$ of $(v, j) \Rightarrow \operatorname{map}$ (Pair j) (if $j=0$ then
$d p \quad v$ else df $v)$ ))
show $t!i \in$ map ?f $j s!i$
unfolding ti nth-map[OF i] split using mem1[OF jsi] by auto
qed
have dist: distinct js and lenj: length $j s=$ Suc bnd unfolding $j s$-def degP using kronecker-samples by auto
have map-fst: map fst $t=j s$ unfolding $t$-def
by (rule nth-equalityI, auto)
with dist have dist: distinct (map fst $t$ ) by simp
from lenj $q \operatorname{deg} P$ have degq: degree $q<$ length $t$ unfolding $t$-def by auto
from find-map-filter-None[OF res] $t$
have nfilt: $\neg$ filt (newton-interpolation-poly-int $t$ ) by auto
have $q t: \bigwedge x y .(x, y) \in$ set $t \Longrightarrow$ poly $q x=y$ unfolding $t$-def by auto
from interpolation-poly-int-None[OF dist - qt degq, of Newton] have newton-interpolation-poly-int $t \neq$ None by auto
then obtain $g$ where $l t$ : newton-interpolation-poly-int $t=S o m e g$ by auto

```
    from newton-interpolation-poly-int-Some[OF dist lt]
    have gt: \bigwedgexy. (x,y)\in set t\Longrightarrow poly g x = y and degg: degree g< length t
        using degq by auto
    from uniqueness-of-interpolation-point-list[OF dist qt degq gt degg]
    have g: g=q by auto
    from nfilt[unfolded lt g] have ᄀ filt (Some q).
    from this[unfolded filt-def] q dvd have False by auto
    } note main = this
    thus ?thesis by auto
qed
lemma kronecker-factorization-rat-main-complete: assumes
    none: kronecker-factorization-rat-main df dp bnd p = None
    and dp: degree p\geq2
    shows }\neg(\existsq.1\leq\mathrm{ degree }q\wedge\mathrm{ degree }q\leqbnd ^q dvd p
proof
    assume \exists q. 1 \leq degree q ^ degree q \leqbnd ^qdvd p
    then obtain q}\mathrm{ where q:1 土 degree q degree q}\leqbnd and dvd:q dvd p by aut
    from dvd obtain r where prod: p=q*r unfolding dvd-def by auto
    let ?r = rat-of-int
    let ?rp = map-poly ?r
    let ?p = rat-to-normalized-int-poly p
    obtain a P where rp: ?p = (a,P) by force
    from rat-to-normalized-int-poly[OF this] have deg: degree P = degree p by auto
    from rat-to-int-factor-normalized-int-poly[OF prod rp]
    obtain }\mp@subsup{g}{}{\prime}\mathrm{ where dvd: g}\mp@subsup{g}{}{\prime}dvdP\mathrm{ and dg: degree }\mp@subsup{g}{}{\prime}=\mathrm{ degree q by (auto intro:
dvdI)
    have kronecker-factorization-main df dp bnd P = None
        using none[unfolded kronecker-factorization-rat-main-def rp] by auto
    from kronecker-factorization-main-complete[OF this dp[folded deg]] dg dvd q show
False by auto
qed
end
end
lemma kronecker-factorization:
    kronecker-factorization p=Some q\Longrightarrow
        degree q}\geq1\wedge\mathrm{ degree }q<\mathrm{ degree }p\wedgeq\mathrm{ dvd p
    kronecker-factorization p=None \Longrightarrow degree p \geq1\Longrightarrow irreducible e}
proof -
    note d}=kronecker-factorization-def
    {
        assume kronecker-factorization p = Some q
        from kronecker-factorization-main-sound[OF this[unfolded d]]
        show degree q}\geq1\wedge\mathrm{ degree q< degree p}\wedgeqdvd p by auto linarith
    }
    assume kf: kronecker-factorization p=None and deg: degree p \geq1
    show irreducibled p
    proof (cases degree p=1)
```

```
    case True
    thus ?thesis by (rule linear-irreducible }\mp@subsup{}{d}{}
    next
        case False
        with deg have degree p\geq2 by auto
    with kronecker-factorization-main-complete[OF divisors-fun-int divisors-pos-fun-int
kf[unfolded d] this]
    show ?thesis
        by (intro irreducible ed2, auto)
    qed
qed
lemma kronecker-factorization-rat:
    kronecker-factorization-rat p = Some q \Longrightarrow
        degree q\geq1^degree q< degree p\wedgeqdvd p
    kronecker-factorization-rat p = None \Longrightarrow degree p \geq1 \Longrightarrow irreducible d}
proof -
    note d = kronecker-factorization-rat-def
    {
        assume kronecker-factorization-rat p = Some q
        from kronecker-factorization-rat-main-sound[OF this[unfolded d]]
        show degree q\geq1^ degree q< degree p\wedgeqdvd p by auto linarith
    }
    assume kf: kronecker-factorization-rat p=None and deg: degree p \geq1
    show irreducibled p
    proof (cases degree p=1)
        case True
        thus ?thesis by (rule linear-irreducible d)
    next
        case False
        with deg have degree p\geq2 by auto
    with kronecker-factorization-rat-main-complete[OF divisors-fun-int divisors-pos-fun-int
kf[unfolded d] this]
    show ?thesis
        by (intro irreducible ed I2, auto)
    qed
qed
end
end
```


## 10 Polynomial Divisibility

We make a connection between irreducibility of Missing-Polynomial and Factorial-Ring.
theory Polynomial-Divisibility
imports
Polynomial-Interpolation.Missing-Polynomial

## begin

lemma $d v d$-gcd-mult: fixes $p::{ }^{\prime} a$ :: semiring-gcd
assumes $d v d: k d v d p * q k d v d p * r$
shows $k d v d p * \operatorname{gcd} q r$
by (rule dvd-trans, rule gcd-greatest $[O F$ dvd])
(auto intro!: mult-dvd-mono simp: gcd-mult-left)
lemma poly-gcd-monic-factor:
monic $p \Longrightarrow \operatorname{gcd}(p * q)(p * r)=p * \operatorname{gcd} q r$
by (rule gcdI [symmetric]) (simp-all add: normalize-mult normalize-monic dvd-gcd-mult)

## context

assumes SORT-CONSTRAINT('a :: field)
begin
lemma field-poly-irreducible-dvd-mult[simp]:
assumes irr: irreducible ( $p$ :: 'a poly)
shows $p d v d q * r \longleftrightarrow p d v d q \vee p d v d r$
using field-poly-irreducible-imp-prime[OF irr] by (simp add: prime-elem-dvd-mult-iff)
lemma irreducible-dvd-pow:
fixes $p$ :: 'a poly
assumes irr: irreducible $p$
shows $p$ dvd $q{ }^{\wedge} n \Longrightarrow p d v d q$
using field-poly-irreducible-imp-prime $[O F$ irr $]$ by (rule prime-elem-dvd-power)
lemma irreducible-dvd-prod: fixes $p$ :: 'a poly
assumes irr: irreducible $p$
and $d v d: p$ dvd prod $f$ as
shows $\exists a \in a s . p d v d f a$
by (insert dvd, induct as rule: infinite-finite-induct, insert irr, auto)
lemma irreducible-dvd-prod-list: fixes $p$ :: ' $a$ poly
assumes irr: irreducible $p$
and dvd: $p$ dvd prod-list as
shows $\exists a \in$ set as. $p$ dvd $a$
by (insert dvd, induct as, insert irr, auto)
lemma dvd-mult-imp-degree: fixes $p::{ }^{\prime} a$ poly
assumes $p$ dvd $q * r$
and degree $p>0$
shows $\exists s t$. irreducible $s \wedge p=s * t \wedge(s d v d q \vee s d v d r)$
proof -
from irreducible $_{d}$-factor $[O F \operatorname{assms}$ (2) $]$ obtain $s t$
where irred: irreducible $s$ and $p: p=s * t$ by auto
from $\langle p$ dvd $q * r\rangle p$ have $s$ : $s d v d q * r$ unfolding $d v d-d e f$ by auto
from $s p$ irred show ?thesis by auto

## qed

end
end

### 10.1 Fundamental Theorem of Algebra for Factorizations

Via the existing formulation of the fundamental theorem of algebra, we prove that we always get a linear factorization of a complex polynomial. Using this factorization we show that root-square-freeness of complex polynomial is identical to the statement that the cardinality of the set of all roots is equal to the degree of the polynomial.

```
theory Fundamental-Theorem-Algebra-Factorized
imports
    Order-Polynomial
    HOL-Computational-Algebra.Fundamental-Theorem-Algebra
begin
lemma fundamental-theorem-algebra-factorized: fixes p :: complex poly
    shows \exists as. smult (coeff p (degree p)) (П a\leftarrowas. [:- a, 1:]) = p^ length as
= degree p
proof -
    define n}\mathrm{ where }n=\mathrm{ degree }
    have degree p=n unfolding n-def by simp
    thus ?thesis
    proof (induct n arbitrary: p)
        case (0 p)
        hence \exists c. p=[:c:] by (cases p, auto split: if-splits)
        thus ?case by (intro exI[of-Nil], auto)
    next
        case (Suc n p)
        have dp: degree p =Suc n by fact
        hence }\neg\mathrm{ constant (poly p) by (simp add: constant-degree)
        from fundamental-theorem-of-algebra[OF this] obtain c where rt: poly p c=
O by auto
    hence [:-c,1 :] dvd p by (simp add: dvd-iff-poly-eq-0)
    then obtain q}\mathrm{ where p:p=q*[:-c,1:] by (metis dvd-def mult.commute)
    from «degree p=Suc n〉 have dq: degree q}=n\mathrm{ using p
            by simp (metis add.right-neutral degree-synthetic-div diff-Suc-1 mult.commute
mult-left-cancel p pCons-eq-0-iff rt synthetic-div-correct' zero-neq-one)
    from Suc(1)[OF this] obtain as where q: [:coeff q (degree q):] * (\proda\leftarrowas. [:-
a, 1:]) = q
            and deg: length as = degree q by auto
    have dc: degree p = degree q + degree [: -c, 1 :] unfolding dq dp by simp
            have cq: coeff q (degree q) = coeff p (degree p) unfolding dc unfolding p
coeff-mult-degree-sum unfolding dq by simp
    show ?case using p[unfolded q[unfolded cq, symmetric]]
```

```
        by (intro exI[of-c# as], auto simp: ac-simps, insert deg dc, auto)
    qed
qed
lemma rsquarefree-card-degree: assumes p0:(p :: complex poly) }=
    shows rsquarefree p}=(\mathrm{ card {x. poly p x = 0} = degree p)
proof -
    from fundamental-theorem-algebra-factorized[of p] obtain c as
        where p:p=smult c (Пa\leftarrowas. [:- a, 1:]) and pas: degree p = length as
        and c:c= coeff p (degree p) by metis
    let ?prod = (\proda\leftarrowas. [:-a,1:])
    from p0 have c:c\not=0 unfolding c by auto
    have roots: {x. poly p x=0}= set as unfolding p poly-smult-zero-iff poly-prod-list
prod-list-zero-iff
        using c by auto
    have idr: (card {x. poly px=0} = degree p)= distinct as unfolding roots pas
        using card-distinct distinct-card by blast
    have id: }\q.(p\not=0\wedgeq)=q\mathrm{ using p0 by simp
    have dist: distinct as = (\foralla. (\sumx\leftarrowas. if x =a then 1 else 0)\leqSuc 0) (is ?l
=(}\forall\mp@code{a. ?r a ) )
    proof (cases distinct as)
    case False
    from not-distinct-decomp[OF this] obtain xs ys zs a where as=xs@ @a]@
ys @ [a]@ zs by auto
    hence \neg? ?r a by auto
    thus ?thesis using False by auto
    next
        case True
    {
        fix }
        from True have ?r a
        proof (induct as)
                case (Cons b bs)
                show ?case
                proof (cases a = b)
                        case False
                        with Cons show ?thesis by auto
                next
                case True
                    with Cons(2) have a\not\in set bs by auto
                hence }(\sumx\leftarrowbs.if x=a then 1 else 0) = (0 :: nat) by (induct bs,auto
                    thus ?thesis unfolding True by auto
                qed
            qed simp
        }
        thus ?thesis using True by auto
    qed
    have rsquarefree p = distinct as unfolding rsquarefree-def' id unfolding p
order-smult[OF c]
```

```
    by (subst order-prod-list, auto simp: o-def order-linear' dist)
    thus ?thesis unfolding idr by simp
qed
```

end

## 11 Square Free Factorization

We implemented Yun's algorithm to perform a square-free factorization of a polynomial. We further show properties of a square-free factorization, namely that the exponents in the square-free factorization are exactly the orders of the roots. We also show that factorizing the result of square-free factorization further will again result in a square-free factorization, and that square-free factorizations can be lifted homomorphically.

```
theory Square-Free-Factorization
imports
    Matrix.Utility
    Polynomial-Divisibility
    Order-Polynomial
    Fundamental-Theorem-Algebra-Factorized
    Polynomial-Interpolation.Ring-Hom-Poly
begin
definition square-free :: 'a :: comm-semiring-1 poly \(\Rightarrow\) bool where
    square-free \(p=(p \neq 0 \wedge(\forall q\). degree \(q>0 \longrightarrow \neg(q * q\) dvd \(p)))\)
lemma square-freeI:
    assumes \(\wedge q\). degree \(q>0 \Longrightarrow q \neq 0 \Longrightarrow q * q\) dvd \(p \Longrightarrow\) False
    and \(p: p \neq 0\)
    shows square-free \(p\) unfolding square-free-def
proof (intro allI conjI[OF p] impI notI, goal-cases)
    case (1 q)
    from \(\operatorname{assms}(1)[\) OF 1(1)-1(2)] 1(1) show False by (cases \(q=0\), auto)
qed
lemma square-free-multD:
    assumes sf: square-free \((f * g)\)
    shows \(h\) dvd \(f \Longrightarrow h\) dvd \(g \Longrightarrow\) degree \(h=0\) square-free \(f\) square-free \(g\)
proof -
    from \(s f[\) unfolded square-free-def] have \(0: f \neq 0 g \neq 0\)
        and dvd: \(\bigwedge q . q * q d v d f * g \Longrightarrow\) degree \(q=0\) by auto
    then show square-free \(f\) square-free \(g\) by (auto simp: square-free-def)
    assume \(h d v d f h d v d g\)
    then have \(h * h\) dvd \(f * g\) by (rule mult-dvd-mono)
    from dvd [OF this] show degree \(h=0\).
qed
```

```
lemma irreducible \(_{d}\)-square-free:
    fixes \(p:: ' a::\{\) comm-semiring-1, semiring-no-zero-divisors \(\}\) poly
    shows irreducible \(_{d} p \Longrightarrow\) square-free \(p\)
    by (metis degree-0 degree-mult-eq degree-mult-eq-0 irreducible \({ }_{d} D\) (1) irreducible \({ }_{d} D\) (2)
irreducible \(_{d}\)-dvd-smult irreducible \(d_{d}\)-smultI less-add-same-cancel2 not-gr-zero square-free-def)
lemma square-free-factor: assumes \(d v d: a d v d p\)
    and sf: square-free \(p\)
    shows square-free a
proof (intro square-freeI)
    fix \(q\)
    assume \(q\) : degree \(q>0\) and \(q * q\) dvd \(a\)
    hence \(q * q d v d p\) using dvd dvd-trans sf square-free-def by blast
    with \(s f[\) unfolded square-free-def] \(q\) show False by auto
qed (insert dvd sf, auto simp: square-free-def)
lemma square-free-prod-list-distinct:
    assumes sf: square-free (prod-list us :: 'a :: idom poly)
    and \(u s: \bigwedge u . u \in\) set \(u s \Longrightarrow\) degree \(u>0\)
    shows distinct us
proof (rule ccontr)
    assume \(\neg\) distinct us
    from not-distinct-decomp[OF this] obtain xs ys zs \(u\) where
        \(u s=x s @ u \# y s\) @ \(u \# z s\) by auto
    hence \(d v d: u * u\) dvd prod-list us and \(u: u \in\) set us by auto
    from \(d v d u s[O F \quad u]\) sf have prod-list us \(=0\) unfolding square-free-def by auto
    hence \(0 \in\) set us by (simp add: prod-list-zero-iff)
    from \(u s[O F\) this \(]\) show False by auto
qed
definition separable where
    separable \(f=\) coprime \(f(\) pderiv \(f)\)
lemma separable-imp-square-free:
    assumes sep: separable ( \(f\) :: 'a::\{field, factorial-ring-gcd, semiring-gcd-mult-normalize\}
poly)
    shows square-free \(f\)
proof (rule ccontr)
    note sep \(=\) sep[unfolded separable-def]
    from sep have \(f 0: f \neq 0\) by (cases \(f\), auto)
    assume \(\neg\) square-free \(f\)
    then obtain \(g\) where \(g\) : degree \(g \neq 0\) and \(g * g d v d f\) using \(f 0\) unfolding
square-free-def by auto
    then obtain \(h\) where \(f: f=g *(g * h)\) unfolding dvd-def by (auto simp:
ac-simps)
    have pderiv \(f=g *((g *\) pderiv \(h+h *\) pderiv \(g)+h *\) pderiv \(g)\)
        unfolding \(f\) pderiv-mult \([o f g]\) by (simp add: field-simps)
    hence \(g\) dvd pderiv \(f\) unfolding dvd-def by blast
    moreover have \(g d v d f\) unfolding \(f d v d\)-def by blast
```

ultimately have $d v d: g d v d(g c d f(p d e r i v f))$ by $\operatorname{simp}$
have $g c d f($ pderiv $f) \neq 0$ using f0 by simp
with $g$ dvd have degree $(g c d f($ pderiv $f)) \neq 0$
by ( simp add: sep poly-dvd-1)
hence $\neg$ coprime $f(p d e r i v f)$ by auto
with sep show False by simp
qed
lemma square-free-rsquarefree: assumes $f$ : square-free $f$
shows rsquarefree $f$
unfolding rsquarefree-def
proof (intro conjI allI)
fix $x$
show order $x f=0 \vee$ order $x f=1$
proof (rule ccontr)
assume $\neg$ ?thesis
then obtain $n$ where ord: order $x f=$ Suc (Suc n)
by (cases order $x f$; cases order $x f-1$; auto)
define $p$ where $p=[:-x, 1:]$
from order-divides[of $x$ Suc (Suc 0) $f$, unfolded ord]
have $p * p$ dvd $f$ degree $p \neq 0$ unfolding $p$-def by auto
hence $\neg$ square-free $f$ using $f(1)$ unfolding square-free-def by auto
with assms show False by auto
qed
qed (insert $f$, auto simp: square-free-def)
lemma square-free-prodD:
fixes $f s$ :: ' $a$ :: \{field,euclidean-ring-gcd,semiring-gcd-mult-normalize $\}$ poly set
assumes sf: square-free ( $\Pi$ fs)
and fin: finite $f_{s}$
and $f: f \in f s$
and $g: g \in f_{s}$
and $f g: f \neq g$
shows coprime $f g$
proof -
have $\left(\prod f s\right)=f *\left(\prod(f s-\{f\})\right)$
by (rule prod.remove $[O F$ fin $f]$ )
also have $\left(\prod\left(f_{s}-\{f\}\right)\right)=g *\left(\prod\left(f_{s}-\{f\}-\{g\}\right)\right)$
by (rule prod.remove, insert fin $g$ fg, auto)
finally obtain $k$ where sf: square-free $(f * g * k)$ using sf by (simp add: ac-simps)
from $s f[$ unfolded square-free-def] have $0: f \neq 0 g \neq 0$
and $d v d: \bigwedge q . q * q d v d f * g * k \Longrightarrow$ degree $q=0$
by auto
have $g c d f g * g c d f g d v d f * g * k$ by (simp add: mult-dvd-mono)
from $d v d[O F$ this $]$ have degree $(g c d f g)=0$.
moreover have $g c d f g \neq 0$ using 0 by auto
ultimately show coprime $f g$ using is-unit-gcd[offg]is-unit-iff-degree[of gcd $f$
$g]$ by $\operatorname{simp}$

## qed

lemma rsquarefree-square-free-complex: assumes rsquarefree ( $p$ :: complex poly)
shows square-free $p$
proof (rule square-freeI)
fix $q$
assume $d$ : degree $q>0$ and $d v d: q * q d v d p$
from $d$ have $\neg$ constant (poly q) by (simp add: constant-degree)
from fundamental-theorem-of-algebra[OF this] obtain $x$ where poly $q x=0$ by auto
hence $[:-x, 1:]$ dvd $q$ by (simp add: poly-eq- 0 -iff-dvd)
then obtain $k$ where $q: q=[:-x, 1:] * k$ unfolding dvd-def by auto
from $d v d$ obtain $l$ where $p: p=q * q * l$ unfolding dvd-def by auto
from $p$ [unfolded $q$ ] have $p=[:-x, 1:]^{\wedge} 2 *(k * k * l)$ by algebra
hence [:-x, 1:] 2 dvd $p$ unfolding dvd-def by blast
from this[unfolded order-divides] have $p=0 \vee \neg$ order $x p \leq 1$ by auto
thus False using assms unfolding rsquarefree-def' by auto
qed (insert assms, auto simp: rsquarefree-def)
lemma square-free-separable-main:
fixes $f::$ ' $a$ :: \{field,factorial-ring-gcd,semiring-gcd-mult-normalize $\}$ poly
assumes square-free $f$
and sep: $\neg$ separable $f$
shows $\exists g k . f=g * k \wedge$ degree $g \neq 0 \wedge$ pderiv $g=0$
proof -
note $c o p=$ sep[unfolded separable-def]
from assms have $f: f \neq 0$ unfolding square-free-def by auto
let $? g=g c d f($ pderiv $f)$
define $G$ where $G=$ ? $g$
from poly-gcd-monic[of f pderiv $f] f$ have mon: monic ? $g$
by auto
have deg: degree $G>0$
proof (cases degree $G$ )
case 0
from degree 0 -coeffs [OF this] cop mon show? ?thesis
by (auto simp: G-def coprime-iff-gcd-eq-1)
qed auto
have $g f$ : $G d v d f$ unfolding $G$-def by auto
have $g f^{\prime}: G$ dvd pderiv $f$ unfolding $G$-def by auto
from irreducible $_{d}$-factor [OF deg] obtain $g r$ where $g$ : irreducible $g$ and $G$ : $G$
$=g * r$ by auto
from $g f$ have $g f: g d v d f$ unfolding $G$ by (rule dvd-mult-left)
from $g f^{\prime}$ have $g f^{\prime}: g$ dvd pderiv $f$ unfolding $G$ by (rule dvd-mult-left)
have $g 0$ : degree $g \neq 0$ using $g$ unfolding irreducible $d_{d}$-def by auto
from $g f$ obtain $k$ where $f g k$ : $f=g * k$ unfolding dvd-def by auto
have id1: pderiv $f=g *$ pderiv $k+k *$ pderiv $g$ unfolding fgk pderiv-mult by simp
from $g f^{\prime}$ obtain $h$ where pderiv $f=g * h$ unfolding $d v d$-def by auto
from id1[unfolded this] have $k *$ pderiv $g=g *(h-$ pderiv $k)$ by (simp add:

```
field-simps)
    hence dvd: g dvd k* pderiv g unfolding dvd-def by auto
    {
        assume g dvd k
        then obtain h where k: k=g*h unfolding dvd-def by auto
        with fgk have g*g dvd f by auto
        with g0 have }\neg\mathrm{ square-free f}\mathrm{ unfolding square-free-def using f by auto
        with assms have False by simp
    }
    with g dvd
    have g dvd pderiv g by auto
    from divides-degree[OF this] degree-pderiv-le[of g] g0
    have pderiv g=0 by linarith
    with fgk g0 show ?thesis by auto
qed
lemma square-free-imp-separable: fixes f :: 'a :: {field-char-0,factorial-ring-gcd,semiring-gcd-mult-normalize}
poly
    assumes square-free f
    shows separable f
proof (rule ccontr)
    assume \neg separable f
    from square-free-separable-main[OF assms this]
    obtain gk where *: f=g*k degree g}\not=0\mathrm{ pderiv }g=0\mathrm{ by auto
    hence g dvd pderiv g}\mathrm{ by auto
    thus False unfolding dvd-pderiv-iff using * by auto
qed
lemma square-free-iff-separable:
    square-free (f :: 'a :: {field-char-0,factorial-ring-gcd,semiring-gcd-mult-normalize}
poly) = separable f
    using separable-imp-square-free[of f] square-free-imp-separable[of f] by auto
context
    assumes SORT-CONSTRAINT('a::{field,factorial-ring-gcd})
begin
lemma square-free-smult: c\not=0\Longrightarrow square-free ( }f\mathrm{ :: 'a poly) > square-free (smult
c f)
    by (unfold square-free-def, insert dvd-smult-cancel[of-c], auto)
lemma square-free-smult-iff[simp]: c\not=0\Longrightarrow square-free (smult c f)=square-free
(f :: 'a poly)
    using square-free-smult[of cf] square-free-smult[of inverse c smult c f] by auto
end
context
    assumes SORT-CONSTRAINT('a::factorial-ring-gcd)
begin
```

definition square-free-factorization $::$ ' $a$ poly $\Rightarrow{ }^{\prime} a \times\left({ }^{\prime} a\right.$ poly $\times$ nat $)$ list $\Rightarrow$ bool where
square-free-factorization $p$ cbs $\equiv$ case cbs of $(c, b s) \Rightarrow$
( $p=$ smult $c\left(\Pi(a, i) \in\right.$ set bs. $a^{\wedge}$ Suc $\left.\left.i\right)\right)$
$\wedge(p=0 \longrightarrow c=0 \wedge b s=[])$
$\wedge(\forall a i .(a, i) \in$ set $b s \longrightarrow$ square-free $a \wedge$ degree $a>0)$
$\wedge(\forall$ a ibj. $(a, i) \in$ set $b s \longrightarrow(b, j) \in$ set $b s \longrightarrow(a, i) \neq(b, j) \longrightarrow$ coprime a $b)$
$\wedge$ distinct bs
lemma square-free-factorizationD: assumes square-free-factorization $p(c, b s)$
shows $p=$ smult $c\left(\prod(a, i) \in\right.$ set bs. $a^{\wedge}$ Suc $\left.i\right)$
$(a, i) \in$ set $b s \Longrightarrow$ square-free $a \wedge$ degree $a \neq 0$
$(a, i) \in$ set $b s \Longrightarrow(b, j) \in$ set $b s \Longrightarrow(a, i) \neq(b, j) \Longrightarrow$ coprime $a b$
$p=0 \Longrightarrow c=0 \wedge b s=[]$
distinct bs
using assms unfolding square-free-factorization-def split by blast+
lemma square-free-factorization-prod-list: assumes square-free-factorization $p(c, b s)$ shows $p=$ smult $c($ prod-list $(\operatorname{map}(\lambda(a, i), a \wedge$ Suc i) bs $))$
proof -
note sff $=$ square-free-factorizationD[OF assms]
show ?thesis unfolding sff (1)
by (simp add: prod.distinct-set-conv-list[OF sff (5)])
qed
end

### 11.1 Yun's factorization algorithm

locale $y u n-g c d=$
fixes Gcd :: ' $a$ :: factorial-ring-gcd poly $\Rightarrow$ ' $a$ poly $\Rightarrow$ 'a poly
begin
partial-function (tailrec) yun-factorization-main ::
'a poly $\Rightarrow$ 'a poly $\Rightarrow$
nat $\Rightarrow\left({ }^{\prime}\right.$ a poly $\times$ nat $)$ list $\Rightarrow\left({ }^{\prime}\right.$ a poly $\times$ nat $)$ list where
[code]: yun-factorization-main bn cn i sqr $=($
if $b n=1$ then $s q r$
else (
let
$d n=c n-$ pderiv $b n ;$
$a n=G c d b n d n$
in yun-factorization-main (bn div an) (dn div an) (Suc i) ((an,i) \# sqr)))
definition yun-monic-factorization :: 'a poly $\Rightarrow$ ('a poly $\times$ nat)list where yun-monic-factorization $p=$ (let
$p p=$ pderiv $p$;
$u=G c d p p p ;$
$b 0=p$ div $u ;$
$c 0=p p \operatorname{div} u$

```
in
(filter }(\lambda(a,i).a\not=1)(yun-factorization-main b0 c0 0 []))
```

definition square-free-monic-poly :: 'a poly $\Rightarrow$ 'a poly where square-free-monic-poly $p=(p \operatorname{div}(G c d p(p d e r i v ~ p)))$
end
declare yun-gcd.yun-monic-factorization-def [code]
declare yun-gcd.yun-factorization-main.simps [code]
declare yun-gcd.square-free-monic-poly-def [code]

## context

fixes Gcd :: 'a :: \{field-char-0, euclidean-ring-gcd\} poly $\Rightarrow{ }^{\prime}$ a poly $\Rightarrow$ ' $a$ poly
begin
interpretation yun-gcd Gcd.
definition square-free-poly $::$ ' $a$ poly $\Rightarrow$ ' $a$ poly where
square-free-poly $p=$ (if $p=0$ then 0 else
square-free-monic-poly $($ smult $($ inverse $(\operatorname{coeff} p($ degree $p))) p))$
definition yun-factorization :: 'a poly $\Rightarrow$ ' $a \times($ 'a poly $\times$ nat $)$ list where
yun-factorization $p=($ if $p=0$
then $(0,[])$ else (let
$c=$ coeff $p($ degree $p)$;
$q=$ smult (inverse $c$ ) $p$
in $(c$, yun-monic-factorization $q))$ )
lemma yun-factorization- $0[$ simp $]$ : yun-factorization $0=(0,[])$
unfolding yun-factorization-def by simp
end
locale monic-factorization =
fixes as :: ('a :: \{field-char-0,euclidean-ring-gcd,semiring-gcd-mult-normalize\}
poly $\times$ nat) set
and $p::$ 'a poly
assumes $p: p=\operatorname{prod}(\lambda(a, i) \cdot a `$ Suc $i)$ as
and fin: finite as
assumes as-distinct: $\bigwedge a i b j .(a, i) \in a s \Longrightarrow(b, j) \in a s \Longrightarrow(a, i) \neq(b, j) \Longrightarrow$
$a \neq b$
and as-irred: $\bigwedge$ a i. $(a, i) \in$ as $\Longrightarrow$ irreducible $_{d} a$
and as-monic: $\bigwedge a i .(a, i) \in a s \Longrightarrow$ monic $a$
begin
lemma poly-exp-expand:
$p=\left(\operatorname{prod}\left(\lambda(a, i) \cdot a{ }^{\wedge} i\right) a s\right) * \operatorname{prod}(\lambda(a, i) \cdot a)$ as unfolding $p$ prod.distrib[symmetric]
by (rule prod.cong, auto)

```
lemma pderiv-exp-prod:
    pderiv p = (prod ( }\lambda(a,i).a^i)as*\operatorname{sum}(\lambda(a,i)
    prod (\lambda (b,j).b) (as - {(a,i)})* smult (of-nat (Suc i)) (pderiv a)) as)
    unfolding p pderiv-prod sum-distrib-left
proof (rule sum.cong[OF refl])
    fix }
    assume x\inas
    then obtain a i where x:x=(a,i) and mem: (a,i)\inas by (cases x,auto)
    let ?si = smult (of-nat (Suc i)) :: 'a poly }=>\mathrm{ ' 'a poly
    show (\prod(a,i)\inas - {x}. a^ Suc i)* pderiv (case x of (a,i) => a^ Suc i)=
        (\prod(a,i)\inas. a ^
            (case x of (a,i) =>(\prod(a,i)\inas - {(a,i)}.a)* smult (of-nat (Suc i))
(pderiv a))
    unfolding x split pderiv-power-Suc
    proof -
    let ?prod = \(a,i)\inas - {(a,i)}.a^ Suc i
    let ?l = ?prod * (?si ( a ^ i)* pderiv a)
    let ?r = (\Pi(a,i)\inas. a ^ i)*((\Pi(a,i)\inas - {(a,i)}.a)* ?si (pderiv a))
    have ?r = a^ i * ((\Pi (a,i)\inas - {(a,i)}.a^ i)* (\Pi(a,i)\inas - {(a,i)}.
a) * ?si (pderiv a))
            unfolding prod.remove[OF fin mem] by (simp add: ac-simps)
    also have (\Pi (a,i)\inas - {(a,i)}.a^ `)*(\prod (a,i)\inas - {(a,i)}.a)
            = ?prod unfolding prod.distrib[symmetric]
        by (rule prod.cong[OF refl],auto)
    finally show ?l = ?r
                by (simp add: ac-simps)
    qed
qed
lemma monic-gen: assumes \(b s \subseteq\) as shows monic \(\left(\prod(a, i) \in b s\right.\). \(\left.a\right)\)
by (rule monic-prod, insert assms as-monic, auto)
lemma nonzero-gen: assumes \(b s \subseteq a s\) shows \(\left(\prod(a, i) \in b s . a\right) \neq 0\)
using monic-gen [OF assms] by auto
lemma monic-Prod: monic \(\left(\left(\prod(a, i) \in\right.\right.\) as. \(\left.\left.a^{\wedge} i\right)\right)\)
by (rule monic-prod, insert as-monic, auto intro: monic-power)
lemma coprime-generic:
assumes \(b s: b s \subseteq a s\)
and \(f: \wedge a i .(a, i) \in b s \Longrightarrow f i>0\)
shows coprime \(\left(\prod(a, i) \in b s . a\right)\)
\(\left(\sum(a, i) \in b s .\left(\prod(b, j) \in b s-\{(a, i)\} . b\right) * \operatorname{smult}(\right.\) of-nat \(\left.(f i))(p d e r i v a)\right)\)
(is coprime ?single? onederiv)
proof -
have single: ?single \(\neq 0\) by (rule nonzero-gen \([O F \quad b s]\) )
show ?thesis
```

```
proof (rule gcd-eq-1-imp-coprime, rule gcdI [symmetric])
    fix }
    assume dvd: k dvd ?single k dvd ?onederiv
    note bs-monic = as-monic[OF subsetD[OF bs]]
    from dvd(1) single have k: k\not=0 by auto
    show k dvd 1
    proof (cases degree k>0)
        case False
        with k obtain c where k=[:c:]
        by (auto dest: degree0-coeffs)
        with }k\mathrm{ have c}=
            by auto
    with 〈k=[:c:]> show is-unit k
        using dvdI [of 1 [:c:] [:1 / c:]] by auto
    next
        case True
        from irreducible d-factor[OF this]
        obtain p q where k: k=p*q and p: irreducible p by auto
        from k dvd have dvd: p dvd ?single p dvd ?onederiv unfolding dvd-def by
auto
    from irreducible-dvd-prod[OF p dvd(1)] obtain a i where ai: (a,i)\inbs and
pa: p dvd a
                by force
    then obtain q where a:a}=p*q\mathrm{ unfolding dvd-def by auto
    from p[unfolded irreducible edef] have p0: degree p>0 by auto
    from irreducible d
            obtain c where c:c\not=0 and ap: a = smult c p by auto
    hence ap': p= smult (1/c) a by auto
    let ?prod = \lambda a i. (\prod (b,j)\inbs-{(a,i)}.b)* smult (of-nat (fi)) (pderiv a)
    let ?prod'=\lambda aa ii a i. (\prod(b,j)\inbs-{(a,i),(aa,ii)}.b)* smult (of-nat (f
i)) (pderiv a)
    define factor where factor = sum ( }\lambda(b,j).?\mathrm{ ?prod' a i b j ) (bs - {(a,i)})
    define fac where fac =q* factor
    from fin finite-subset[OF bs] have fin: finite bs by auto
    have ?onederiv = ?prod a i + sum (\lambda (b,j). ?prod b j) (bs - {(a,i)})
        by (subst sum.remove[OF fin ai], auto)
    also have sum (\lambda (b,j). ?prod b j) (bs - {(a,i)})
        =a* factor
        unfolding factor-def sum-distrib-left
    proof (rule sum.cong[OF refl])
        fix bj
        assume mem:bj \in bs - {(a,i)}
        obtain b j where bj: bj = (b,j) by force
        from mem bj ai have ai: (a,i)\inbs-{(b,j)} by auto
        have id:bs - {(b,j)}-{(a,i)}=bs-{(b,j),(a,i)} by auto
        show (\lambda(b,j). ?prod b j) bj=a*(\lambda(b,j). ?prod' a ibj) bj
            unfolding bj split
            by (subst prod.remove[OF - ai], insert fin, auto simp: id ac-simps)
    qed
```

```
    finally have ?onederiv = ?prod a i+p* fac unfolding fac-def a by simp
    from dvd(2)[unfolded this] have p dvd ?prod a i by algebra
    from this[unfolded field-poly-irreducible-dvd-mult[OF p]]
    have False
    proof
    assume p dvd (\prod(b,j)\inbs - {(a,i)}.b)
    from irreducible-dvd-prod[OF p this] obtain bj where bj':(b,j) \inbs -
{(a,i)}
            and pb: p dvd b by auto
            hence bj: (b,j)\inbs by auto
            from as-irred bj bs have irreducible d b by auto
            from irreducible e}\mp@subsup{|}{d}{-dvd-smult[OF p0 this pb] obtain d where d:d\not=0
            and b:b=smult d p by auto
    with ap c have id: smult (c/d) b=a and deg: degree a=degree b by auto
                from coeff-smult[of c/d b degree b, unfolded id] deg bs-monic[OF ai]
bs-monic[OF bj]
    have c / d=1 by simp
    from id[unfolded this] have a=b by simp
    with as-distinct[OF subsetD[OF bs ai] subsetD[OF bs bj]] bj'
    show False by auto
    next
    from f[OF ai] obtain k where fi: fi=Suc k by (cases f i,auto)
    assume p dvd smult (of-nat (f i)) (pderiv a)
    hence pdvd (pderiv a) unfolding fi using dvd-smult-cancel of-nat-eq-0-iff
by blast
    from this[unfolded ap] have p dvd pderiv p using c
            by (metis <p dvd pderiv a> ap' dvd-trans dvd-triv-right mult.left-neutral
pderiv-smult smult-dvd-cancel)
            with not-dvd-pderiv p0 show False by auto
        qed
        thus k dvd 1 by simp
        qed
    qed (insert<?single }\not=0\mathrm{ \, auto)
qed
lemma pderiv-exp-gcd:
    gcd p (pderiv p)=(\prod(a,i)\inas. a^ i)(is - = ?prod)
proof -
    let ?sum = (\sum(a,i)\inas. (\prod(b,j)\inas - {(a,i)}.b)* smult (of-nat (Suc i))
(pderiv a))
    let ?single = (\prod(a,i)\inas.a)
    let ?prd = \lambda a i. (\prod(b,j)\inas - {(a,i)}.b)* smult (of-nat (Suc i)) (pderiv a)
    let ?onederiv = \sum(a,i)\inas. ?prd a i
    have pp: pderiv p = ?prod * ?sum by (rule pderiv-exp-prod)
    have p:p=?prod * ?single by (rule poly-exp-expand)
    have monic: monic ?prod by (rule monic-Prod)
    have gcd: coprime ?single ?onederiv
        by (rule coprime-generic, auto)
    then have gcd: gcd ?single ?onederiv = 1
```

by $\operatorname{simp}$
show ?thesis unfolding $p p$ unfolding $p$ poly-gcd-monic-factor [OF monic] gcd by $\operatorname{simp}$
qed
lemma $p$-div-gcd- $p$-pderiv: $p \operatorname{div}(\operatorname{gcd} p($ pderiv $p))=\left(\prod(a, i) \in a s . a\right)$
unfolding pderiv-exp-gcd unfolding poly-exp-expand
by (rule nonzero-mult-div-cancel-left, insert monic-Prod, auto)
fun $A B C D$ :: nat $\Rightarrow$ 'a poly where
$A n=g c d(B n)(D n)$
$\mid B 0=p \operatorname{div}(g c d p($ pderiv $p))$
| $B($ Suc $n)=B n \operatorname{div} A n$
| $C 0=$ pderiv $p \operatorname{div}($ gcd $p($ pderiv $p))$
| $C($ Suc $n)=D n \operatorname{div} A n$
| $D n=C n-p d e r i v(B n)$
lemma $A-B-C-D: A n=\left(\prod(a, i) \in a s \cap U N I V \times\{n\} . a\right)$
$B n=\left(\prod(a, i) \in a s-U N I V \times\{0 . .<n\} . a\right)$
$C n=\left(\sum(a, i) \in a s-U N I V \times\{0 . .<n\}\right.$.
$\left(\prod(b, j) \in a s-U N I V \times\{0 . .<n\}-\{(a, i)\} . b\right) * \operatorname{smult}($ of-nat $(S u c i-n))$
(pderiv a))
$D n=\left(\prod(a, i) \in a s \cap\right.$ UNIV $\left.\times\{n\} . a\right) *$
$\left(\sum_{(a, i) \in a s-U N I V} \times\{0 . .<\right.$ Suc $n\}$.
$\left(\prod(b, j) \in a s-U N I V \times\{0 . .<S u c n\}-\{(a, i)\} . b\right) *($ smult $($ of-nat $(i-$
n)) (pderiv a)))
proof (induct $n$ and $n$ and $n$ and $n$ rule: $A-B-C$-D.induct)
case (1n)
note $B n=1(1)$
note $D n=1$ (2)
have $\left(\prod(a, i) \in a s-U N I V \times\{0 . .<n\} . a\right)=\left(\prod(a, i) \in a s \cap U N I V \times\{n\} . a\right)$

* $\left(\prod(a, i) \in a s-U N I V \times\{0 . .<\right.$ Suc $\left.n\} . a\right)$
by (subst prod.union-disjoint[symmetric], auto, insert fin, auto intro: prod.cong)
note $B n^{\prime}=B n[u n f o l d e d$ this]
let ? $a n=\left(\prod(a, i) \in a s \cap U N I V \times\{n\} . a\right)$
let $? b n=\left(\prod(a, i) \in a s-U N I V \times\{0 . .<\right.$ Suc $\left.n\} . a\right)$
show $A n=$ ? an unfolding A.simps
proof (rule gcdI[symmetric, OF - - normalize-monic[OF monic-gen]])
have monB1: monic ( $B n$ ) unfolding $B n$ by (rule monic-gen, auto)
hence $B n \neq 0$ by auto
let ? $d n=\left(\sum(a, i) \in a s-U N I V \times\{0 . .<\right.$ Suc $n\}$.
$\left(\prod(b, j) \in a s-U N I V \times\{0 . .<S u c n\}-\{(a, i)\} . b\right) *$ (smult (of-nat $(i$
$-n))(p d e r i v ~ a)))$
have $D n$ : $D n=? a n * ? d n$ unfolding $D n$ by auto
show dvd1: ?an dvd $B$ unfolding $B n^{\prime} d v d$-def by blast
show dvd2: ?an dvd $D n$ unfolding $D n d v d-d e f$ by blast \{
fix $k$
assume $k$ dvd $B n k d v d D n$

```
        from dvd-gcd-mult[OF this[unfolded Bn' Dn]]
        have k dvd ?an * (gcd ?bn ?dn).
        moreover have coprime ?bn ?dn
            by (rule coprime-generic, auto)
        ultimately show }k\mathrm{ dvd ?an by simp
    }
    qed auto
next
    case 2
    have as: as - UNIV }\times{0..<0}=as by aut
    show ?case unfolding B.simps as p-div-gcd-p-pderiv by auto
next
    case (3 n)
    have id: (\prod(a,i)\inas - UNIV }\times{0..<n}.a)=(\prod(a,i)\inas-UNIV \times
{0..<Suc n}.a)*(\prod(a,i)\inas\capUNIV }\times{n}.a
    by (subst prod.union-disjoint[symmetric], auto, insert fin, auto intro: prod.cong)
    show ?case unfolding B.simps 3 id
        by (subst nonzero-mult-div-cancel-right[OF nonzero-gen], auto)
next
    case 4
    have as: as - UNIV }\times{0..<0}= as \bigwedge i.Suc i-0=Suc i by aut
    show ?case unfolding C.simps pderiv-exp-gcd unfolding pderiv-exp-prod as
        by (rule nonzero-mult-div-cancel-left, insert monic-Prod, auto)
next
    case (5 n)
    show ?case unfolding C.simps 5
        by (subst nonzero-mult-div-cancel-left, rule nonzero-gen, auto)
next
    case (6 n)
    let ?f = \lambda (a,i).(\prod(b,j)\inas - UNIV }\times{0..<n}-{(a,i)}.b)*(smult
(of-nat (i - n)) (pderiv a))
```



```
..<n}-{(a,i)}.b)*
        (smult (of-nat (Suc i - n)) (pderiv a) - pderiv a))
        unfolding D.simps 6 pderiv-prod sum-subtractf[symmetric] right-diff-distrib
        by (rule sum.cong, auto)
    also have ... = sum ?f (as - UNIV }\times{0..<n}
    proof (rule sum.cong[OF refl])
    fix }
    assume x fas - UNIV }\times{0..<n
    then obtain a i where x: x = (a,i) and i:Suci>n by (cases x, auto)
    hence id: Suc i - n=Suc (i-n) by arith
    have id:of-nat (Suc i - n) =of-nat ( }i-n)+(1 ::'a) unfolding id by sim
    have id: smult (of-nat (Suc i - n)) (pderiv a) - pderiv a = smult (of-nat (i
- n))(pderiv a)
            unfolding id smult-add-left by auto
    have cong: \ x y z:: 'a poly. }x=y\Longrightarrowx*z=y*z\mathrm{ by auto
    show (case x of
        (a,i) =>
```

```
            (\prod(b,j)\inas - UNIV }\times{0..<n}-{(a,i)}.b)
            (smult (of-nat (Suc i - n)) (pderiv a) - pderiv a)) =
            (case x of
            (a,i)=>(\prod(b,j)\inas - UNIV }\times{0..<n}-{(a,i)}.b)* smult (of-na
(i - n)) (pderiv a))
    unfolding }x\mathrm{ split id
    by (rule cong, auto)
    qed
    also have ... = sum ?f (as - UNIV }\times{0..<Suc n})+sum ?f (as \capUNI
< {n})
    by (subst sum.union-disjoint[symmetric], insert fin, auto intro: sum.cong)
    also have sum?f (as\capUNIV }\times{n})=
    by (rule sum.neutral, auto)
    finally have id: D n= sum ?f (as - UNIV }\times{0..<Suc n}) by sim
    show ?case unfolding id sum-distrib-left
    proof (rule sum.cong[OF refl])
    fix }
    assume mem: }x\in\mathrm{ as - UNIV }\times{0..<Suc n
    obtain a i where x: x=(a,i) by force
    with mem have i:i>n by auto
    have cong: \bigwedgexyzv :: 'a poly. }x=y*v\Longrightarrowx*z=y*(v*z) by aut
    show (case x of
            (a,i)=>(\prod(b,j)\inas - UNIV }\times{0..<n}-{(a,i)}.b)* smult (of-na
(i-n))(pderiv a))=
            (\prod(a,i)\inas\capUNIV }\times{n}.a)
            (case x of (a,i) =>
                        (\prod(b,j)\inas - UNIV }\times{0..<Suc n}-{(a,i)}.b)* smult (of-nat (
- n)) (pderiv a))
        unfolding x split
        by (rule cong, subst prod.union-disjoint[symmetric], insert fin, (auto)[3],
            rule prod.cong, insert i, auto)
    qed
qed
lemmas A=A-B-C-D(1)
lemmas }B=A-B-C-D(2
lemmas ABCD-simps = A.simps B.simps C.simps D.simps
declare ABCD-simps[simp del]
lemma prod-A:
    (\prodi=0..<n.A i^Suc i)=(\prod(a,i)\inas\capUNIV ×{0..<n}.a^ Suc i)
proof (induct n)
    case (Suc n)
    have id:{0 ..<Suc n}= insert n{0 ..< n} by auto
    have id2: as \capUNIV }\times{0..<Suc n}= as\capUNIV > {n}\cup as\capUNIV 
{0..<n} by auto
    have cong: \}xyz.x=y\Longrightarrowx*z=y*z\mathrm{ by auto
    show ?case unfolding id2 unfolding id
```

```
    proof (subst prod.insert; (subst prod.union-disjoint) ?; (unfold Suc)?;
    (unfold A, rule cong)?)
    show \(\left(\prod(a, i) \in\right.\) as \(\cap\) UNIV \(\left.\times\{n\} . a\right){ }^{\wedge}\) Suc \(n=\left(\prod(a, i) \in a s \cap\right.\) UNIV \(\times\{n\}\).
\(a^{\wedge}\) Suc i)
            unfolding prod-power-distrib
            by (rule prod.cong, auto)
    qed (insert fin, auto)
qed \(\operatorname{simp}\)
lemma prod-A-is-p-unknown: assumes \(\bigwedge a i .(a, i) \in a s \Longrightarrow i<n\)
    shows \(p=\left(\prod i=0 . .<n . A{ }^{\wedge}\right.\) Suc \(\left.i\right)\)
proof -
    have \(p=\left(\Pi(a, i) \in\right.\) as. \(a^{\wedge}\) Suc \(\left.i\right)\) by (rule \(\left.p\right)\)
    also have \(\ldots=\left(\prod i=0 . .<n . A i^{\wedge} S u c i\right)\) unfolding prod- \(A\)
    by (rule prod.cong, insert assms, auto)
    finally show? thesis.
qed
definition bound :: nat where
    bound \(=\) Suc \((\operatorname{Max}(\) snd'as \())\)
lemma bound: assumes \(m: m \geq\) bound
    shows \(B m=1\)
proof -
    let ?set \(=a s-U N I V \times\{0 . .<m\}\)
    \{
        fix \(a i\)
        assume ai: \((a, i) \in\) ?set
        hence \(i \in\) snd ' as by force
        from Max-ge [OF - this] fin have \(i \leq \operatorname{Max}\) (snd'as) by auto
        with ai m[unfolded bound-def] have False by auto
    \}
    hence \(i d\) : ? set \(=\{ \}\) by force
    show \(B m=1\) unfolding \(B\) id by simp
qed
lemma coprime- \(A-A\) : assumes \(i \neq j\)
    shows coprime \((A i)(A j)\)
proof (rule coprimeI)
    fix \(k\)
    assume \(d v d: k d v d A i k d v d A j\)
    have \(A i\) : \(A i \neq 0\) unfolding \(A\)
        by (rule nonzero-gen, auto)
    with \(d v d\) have \(k: k \neq 0\) by auto
    show is-unit \(k\)
    proof (cases degree \(k>0\) )
    case False
    then obtain \(c\) where \(k c: k=[: c:]\) by (auto dest: degree0-coeffs)
    with \(k\) have \(1=k *[: 1 / c:]\)
```

```
    by simp
    thus ?thesis unfolding dvd-def by blast
    next
    case True
    from irreducible-monic-factor[OF this]
    obtain qr where k: k=q*r and q: irreducible q and mq: monic q by auto
    with dvd have dvd: q dvd A i q dvd A j unfolding dvd-def by auto
    from q have q0: degree q>0 unfolding irreducible }\mp@subsup{d}{d}{}\mathrm{ -def by auto
    from irreducible-dvd-prod[OF q dvd(1)[unfolded A]]
        obtain }a\mathrm{ where ai: (a,i) f as and qa:qdvd a by auto
    from irreducible-dvd-prod[OF q dvd(2)[unfolded A]]
        obtain b where bj: (b,j) \inas and qb: q dvd b by auto
    from as-distinct[OF ai bj] assms have neq: a\not=b by auto
    from irreducible }\mp@subsup{d}{d}{}\mathrm{ -dvd-smult[OF q0 as-irred[OF ai] qa]
            irreducible d-dvd-smult[OF q0 as-irred[OF bj] qb]
    obtain c d where c\not=0d\not=0 a = smult c q b = smult d q by auto
    hence ab:a smult (c/d)b and c/d\not=0 by auto
    with as-monic[OF bj] as-monic[OF ai] arg-cong[OF ab, of \lambda p.coeff p (degree
p)]
    have }a=b\mathrm{ unfolding coeff-smult degree-smult-eq by auto
    with neq show ?thesis by auto
    qed
qed
lemma A-monic: monic (A i)
    unfolding A by (rule monic-gen, auto)
lemma A-square-free: square-free (A i)
proof (rule square-freeI)
    fix qk
    have mon: monic (A i) by (rule A-monic)
    hence Ai: A i\not=0 by auto
    assume q: degree q>0 and dvd: q* q dvd A i
    from irreducible-monic-factor[OF q] obtain rs where q: q=r*s and
        irr: irreducible r and mr: monic r by auto
    from dvd[unfolded q] have dvd2: r * r dvd A i and dvd1: r dvd A i unfolding
dvd-def by auto
    from irreducible-dvd-prod[OF irr dvd1 [unfolded A]]
        obtain a where ai: (a,i)\inas and ra:r dvd a by auto
    let ?rem=(\prod(a,i)\inas\capUNIV }\times{i}-{(a,i)}.a
    have a:\mp@subsup{\mathrm{ irreducible }}{d}{}a\mathrm{ by (rule as-irred[OF ai])}
    from irreducible d-dvd-smult[OF - a ra] irr
        obtain c where ar: a = smult c r and c\not=0 by force
    with mr as-monic[OF ai] arg-cong[OF ar, of \lambda p. coeff p (degree p)]
    have }a=r\mathrm{ unfolding coeff-smult degree-smult-eq by auto
    with dvd2 have dvd: a*a dvd A i by simp
    have id: A i=a* ?rem unfolding A
    by (subst prod.remove[of-(a,i)], insert ai fin, auto)
    with dvd have a dvd ?rem using a id Ai by auto
```

```
    from irreducible-dvd-prod[OF - this] a obtain b where bi: (b,i) \inas
        and neq: }b\not=a\mathrm{ and }ab:advdb\mathrm{ by auto
    from as-irred[OF bi] have b: irreducible d b .
    from irreducible }\mp@subsup{d}{d}{-dvd-smult[OF - b ab] a[unfolded irreducible d}\mp@subsup{d}{}{-}def
    obtain c where c\not=0 and ba: b= smult c a by auto
    with as-monic[OF bi] as-monic[OF ai] arg-cong[OF ba, of \lambda p. coeff p (degree
p)]
    have }a=b\mathrm{ unfolding coeff-smult degree-smult-eq by auto
    with neq show False by auto
qed (insert A-monic[of i], auto)
lemma prod-A-is-p-B-bound: assumes B n =1
    shows p=(\prodi=0..<n.A i^ Suc i)
proof (rule prod-A-is-p-unknown)
    fix a i
    assume ai:(a,i)\inas
    let ?rem = (\prod(a,i)\inas-UNIV }\times{0..<n}-{(a,i)}.a
    have rem: ? rem }=
        by (rule nonzero-gen, auto)
    have irreducible d a using as-irred[OF ai] .
    hence a: a\not=0 degree a\not=0 unfolding irreducible d-def by auto
    show i<n
    proof (rule ccontr)
        assume \neg ?thesis
        hence }i\geqn\mathrm{ by auto
    with ai have mem: (a,i)\in as - UNIV }\times{0..<n} by aut
    have 0= degree (\prod(a,i)\inas - UNIV }\times{0..<n}.a) using assms unfoldin
B by simp
    also have ... = degree (a*?rem)
                            by (subst prod.remove[OF - mem], insert fin, auto)
    also have ... = degree a+degree ?rem
        by (rule degree-mult-eq[OF a(1) rem])
    finally show False using a(2) by auto
    qed
qed
interpretation yun-gcd gcd .
lemma square-free-monic-poly: (poly (square-free-monic-poly p) x=0) =(poly p
x=0)
proof -
    show ?thesis unfolding square-free-monic-poly-def unfolding p-div-gcd-p-pderiv
        unfolding p poly-prod prod-zero-iff[OF fin] by force
qed
lemma yun-factorization-induct: assumes base: }\\mathrm{ bn cn. bn =1 בP bn cn
    and step: }\bigwedgebn cn.bn\not=1\LongrightarrowP(bn div (gcd bn (cn - pderiv bn)))
        ((cn - pderiv bn) div (gcd bn (cn - pderiv bn))) \LongrightarrowPbn cn
```

```
    and \(i d: b n=p\) div \(g c d p(p d e r i v ~ p) c n=\) pderiv \(p\) div gcd \(p(p d e r i v p)\)
    shows \(P b n c n\)
proof -
    define \(n\) where \(n=(0::\) nat \()\)
    let \(? m=\lambda n\). bound \(-n\)
    have \(P(B n)(C n)\)
    proof (induct \(n\) rule: wf-induct[OF wf-measure \([\) of ? \(m\) ]])
        case (1 \(n\) )
    note \(I H=1(1)[\) rule-format \(]\)
    show ?case
    proof (cases B \(n=1\) )
        case True
        with base show ?thesis by auto
    next
        case False note \(B n=\) this
        with bound \([\) of \(n\) ] have \(\neg\) bound \(\leq n\) by auto
        hence (Suc \(n, n\) ) \(\in\) measure ? \(m\) by auto
        note \(I H=I H[\) OF this]
        show ?thesis
            by (rule step \([\) OF Bn], insert \(I H\), simp add: D.simps C.simps B.simps
A.simps)
    qed
    qed
    thus ?thesis unfolding id \(n\)-def B.simps C.simps .
qed
lemma yun-factorization-main: assumes yun-factorization-main ( \(B n\) ) ( \(C\) n) \(n\)
\(b s=c s\)
    set \(b s=\{(A i, i) \mid i . i<n\}\) distinct (map snd bs)
    shows \(\exists m\). set \(c s=\{(A i, i) \mid i . i<m\} \wedge B m=1 \wedge \operatorname{distinct}\) (map snd cs)
    using assms
proof -
    let \(? m=\lambda n\). bound \(-n\)
    show ?thesis using assms
    proof (induct \(n\) arbitrary: bs rule: wf-induct[OF wf-measure[of ?m]])
        case (1 n)
        note \(I H=1(1)[\) rule-format \(]\)
    have res: yun-factorization-main \((B n)(C n) n b s=c s\) by fact
    note res \(=\) res[unfolded yun-factorization-main.simps[of \(B \quad n]]\)
    have bs: set bs \(=\{(A i, i) \mid i . i<n\}\) distinct (map snd bs) by fact+
    show ?case
    proof (cases B \(n=1\) )
        case True
        with res have \(b s=c s\) by auto
        with True bs show ?thesis by auto
    next
        case False note \(B n=\) this
        with bound \([\) of \(n]\) have \(\neg\) bound \(\leq n\) by auto
        hence (Suc \(n, n\) ) \(\in\) measure ?m by auto
```

```
    note IH=IH[OF this]
    from Bn res[unfolded Let-def, folded D.simps C.simps B.simps A.simps]
    have res: yun-factorization-main (B (Suc n)) (C (Suc n)) (Suc n) ((A n,n)
# bs)=cs
            by simp
    note IH = IH[OF this]
    {
            fix }
            assume }i<\mathrm{ Suc n ᄀi<n
            hence }n=i\mathrm{ by arith
            } note missing = this
            have set ((A n, n) #bs) ={(A i,i) |i.i<Suc n}
            unfolding list.simps bs by (auto, subst missing, auto)
            note IH=IH[OF this]
            from bs have distinct (map snd ((A n, n) # bs)) by auto
            note IH=IH[OF this]
            show ?thesis by (rule IH)
    qed
    qed
qed
lemma yun-monic-factorization-res: assumes res: yun-monic-factorization \(p=b s\) shows \(\exists m\). set \(b s=\{(A i, i) \mid i . i<m \wedge A i \neq 1\} \wedge B m=1 \wedge\) distinct
(map snd bs)
proof -
    from res[unfolded yun-monic-factorization-def Let-def,
        folded B.simps C.simps]
    obtain cs where yun: yun-factorization-main (B 0) ( }\begin{array}{l}{C}\end{array}0)0[]=cs and bs:b
= filter ( }\lambda(a,i).a\not=1) c
    by auto
    from yun-factorization-main[OF yun] show ?thesis unfolding bs
        by (auto simp: distinct-map-filter)
qed
lemma yun-monic-factorization: assumes yun: yun-monic-factorization p =bs
    shows square-free-factorization p}(1,bs)(b,i)\in\mathrm{ set bs \ monic b distinct (map
snd bs)
proof -
    from yun-monic-factorization-res[OF yun]
    obtain m where bs: set bs={(A i,i)| i.i<m\wedgeAi\not=1} and B:B m=1
    and dist: distinct (map snd bs) by auto
    have id: {0..<m}={i.i<m\wedgeAi=1}\cup{i.i<m\wedgeAi\not=1} (is - =
?ignore \cup -) by auto
    have p=(\prodi=0..<m.A i^Suc i)
    by (rule prod-A-is-p-B-bound[OF B])
    also have ...= prod (\lambdai.A i^ Suc i) {i.i<m^Ai\not=1}
        unfolding id
        by (subst prod.union-disjoint, (force+)[3],
    subst prod.neutral[of ?ignore], auto)
```

```
    also have ... = (\Pi(a,i)\in set bs. a^ Suc i) unfolding bs
    by (rule prod.reindex-cong[of snd], auto simp: inj-on-def, force)
    finally have 1:p=(\Pi(a,i)\in set bs. a ^ Suc i) .
{
    fix a i
    assume (a,i) & set bs
    hence A: a=A i A i\not=1 unfolding bs by auto
    with }A\mathrm{ -square-free[of i] A-monic[of i] have square-free a }\wedge\mathrm{ degree a}=0\mathrm{ monic
a
    by (auto simp: monic-degree-0)
} note 2 = this
{
    fix aibj
    assume ai: (a,i) set bs and bj:(b,j)\in set bs and neq: (a,i)\not=(b,j)
    hence a:a=A i and b:b=A j unfolding bs by auto
    from neq dist ai bj have neq:i\not=j using ab}\mathrm{ by blast
    from coprime-A-A [OF neq] have coprime a b unfolding ab.
} note 3= this
have monic punfolding p
    by (rule monic-prod, insert as-monic, auto intro: monic-power monic-mult)
    hence 4:p\not=0 by auto
    from dist have 5: distinct bs unfolding distinct-map ..
    show square-free-factorization p (1,bs)
    unfolding square-free-factorization-def using 12 345
    by auto
    show (b,i)\in set bs \Longrightarrow monic b using 2 by auto
    show distinct (map snd bs) by fact
qed
end
lemma monic-factorization: assumes monic p
    shows \exists as. monic-factorization as p
proof -
    from monic-irreducible-factorization[OF assms]
    obtain as f where fin: finite as and p:p=(\proda\inas.a^ Suc (fa))
        and as:as\subseteq{q. irreducible e}q|\wedge monic q
        by auto
    define cs where cs={(a,fa)|a.a\inas}
    show ?thesis
    proof (rule exI, standard)
        show finite cs unfolding cs-def using fin by auto
        {
        fix a i
        assume (a,i) \incs
        thus irreducible d a monic a unfolding cs-def using as by auto
    } note irr = this
    show \a ibj. (a,i) \incs \Longrightarrow(b,j) \incs \Longrightarrow(a,i)\not=(b,j)\Longrightarrowa\not=b
unfolding cs-def by auto
    show p}=(\prod(a,i)\incs.a^ Suc i) unfolding p cs-def
```

```
        by (rule prod.reindex-cong, auto, auto simp: inj-on-def)
    qed
qed
lemma square-free-monic-poly:
    assumes monic ( }p::\mp@subsup{|}{}{\prime}a:: {field-char-0, euclidean-ring-gcd,semiring-gcd-mult-normalize
poly)
    shows (poly (yun-gcd.square-free-monic-poly gcd p) x=0)=(poly px=0)
proof -
    from monic-factorization[OF assms] obtain as where monic-factorization as p
    from monic-factorization.square-free-monic-poly[OF this] show ?thesis .
qed
lemma yun-factorization-induct:
    assumes base: \ bn cn.bn=1\LongrightarrowPbn cn
    and step: \bigwedge bn cn.bn\not=1\LongrightarrowP(bn div (gcd bn (cn - pderiv bn)))
        ((cn - pderiv bn) div (gcd bn (cn - pderiv bn))) \LongrightarrowP bn cn
    and id:bn = p div gcd p (pderiv p) cn= pderiv p div gcd p (pderiv p)
    and monic: monic ( }p:: 'a :: {field-char-0,euclidean-ring-gcd,semiring-gcd-mult-normalize}
poly)
    shows Pbn cn
proof -
    from monic-factorization[OF monic] obtain as where monic-factorization as p
    from monic-factorization.yun-factorization-induct[OF this base step id] show
?thesis.
qed
lemma square-free-poly:
    (poly (square-free-poly gcd p)x=0)=(poly p x = 0)
proof (cases p=0)
    case True
    thus ?thesis unfolding square-free-poly-def by auto
next
    case False
    let ?c = coeff p (degree p)
    let ?ic = inverse ?c
    have id: square-free-poly gcd p = yun-gcd.square-free-monic-poly gcd (smult ?ic
p)
    unfolding square-free-poly-def using False by auto
    from False have mon: monic (smult ?ic p) and ic: ?ic }\not=0\mathrm{ by auto
    show ?thesis unfolding id square-free-monic-poly[OF mon]
        using ic by simp
qed
lemma yun-monic-factorization:
fixes \(p\) :: ' \(a\) :: \{field-char-0,euclidean-ring-gcd,semiring-gcd-mult-normalize\} poly
```

assumes res: yun-gcd.yun-monic-factorization gcd $p=b s$
and monic: monic $p$
shows square-free-factorization $p(1, b s)(b, i) \in$ set $b s \Longrightarrow$ monic $b$ distinct (map snd bs)
proof -
from monic-factorization[OF monic] obtain as where monic-factorization as $p$
from monic-factorization.yun-monic-factorization[OF this res]
show square-free-factorization $p(1, b s)(b, i) \in$ set $b s \Longrightarrow$ monic $b$ distinct (map snd bs)
by auto
qed
lemma square-free-factorization-smult: assumes $c: c \neq 0$
and $s f$ : square-free-factorization $p(d, b s)$
shows square-free-factorization (smult c p) $(c * d, b s)$
proof -
from $s f$ [unfolded square-free-factorization-def split]
have $p: p=$ smult $d\left(\prod(a, i) \in\right.$ set bs. $a^{\wedge}$ Suc $\left.i\right)$
and eq: $p=0 \longrightarrow d=0 \wedge b s=[]$ by blast +
from eq $c$ have eq: smult $c p=0 \longrightarrow c * d=0 \wedge b s=[]$ by auto
from $p$ have $p$ : smult $c p=$ smult $(c * d)\left(\prod(a, i) \in\right.$ set bs. $a^{\wedge}$ Suc $\left.i\right)$ by auto from eq $p$ sf show ?thesis unfolding square-free-factorization-def by blast qed
lemma yun-factorization: assumes res: yun-factorization gcd $p=c$-bs shows square-free-factorization $p c$-bs $(b, i) \in \operatorname{set}(s n d c-b s) \Longrightarrow$ monic $b$ proof -
interpret yun-gcd gcd.
note res $=$ res[unfolded yun-factorization-def Let-def]
have square-free-factorization p c-bs $\wedge((b, i) \in$ set $($ snd $c$-bs $) \longrightarrow$ monic $b)$
proof (cases $p=0$ )
case True
with res have $c$-bs $=(0,[])$ by auto
thus ?thesis unfolding True by (auto simp: square-free-factorization-def)
next
case False
let ?c $=$ coeff $p($ degree $p)$
let $?$ ic $=$ inverse ?c
obtain $c$ bs where $c b s$ : $c-b s=(c, b s)$ by force
with False res
have $c: c=? c ? c \neq 0$ and fact: yun-monic-factorization (smult ? ic $p$ ) $=b s$
by auto
from False have mon: monic (smult ?ic p) by auto
from yun-monic-factorization[OF fact mon]
have sff: square-free-factorization (smult ?ic $p)(1, b s)(b, i) \in$ set $b s \Longrightarrow$ monic $b$ by auto
have id: smult ?c (smult ?ic $p$ ) $=p$ using False by auto

```
    from square-free-factorization-smult[OF c(2) sff(1), unfolded id] sff
    show ?thesis unfolding cbs c by simp
    qed
    thus square-free-factorization p c-bs (b,i) \in set (snd c-bs)\Longrightarrow monic b by blast+
qed
lemma prod-list-pow-suc: (\x\leftarrowbs. (x ::' 'a :: comm-monoid-mult) * x ^ i)
    = prod-list bs * prod-list bs ` i
    by (induct bs, auto simp: field-simps)
declare irreducible-linear-field-poly[intro!]
context
    assumes SORT-CONSTRAINT(' ' :: {field, factorial-ring-gcd,semiring-gcd-mult-normalize})
begin
lemma square-free-factorization-order-root-mem:
    assumes sff: square-free-factorization p (c,bs)
        and p: p\not=(0 :: 'a poly)
        and ai: (a,i)\in set bs and rt: poly a x=0
    shows order x p Suc i
proof -
    note sff = square-free-factorizationD[OF sff]
    let ?prod = \\(a,i)\inset bs. a ^ Suc i)
    from sff have pf:p=smult c ?prod by blast
    with p have c:c\not=0 by auto
    have ord: order x p = order x ?prod unfolding pf
        using order-smult[OF c] by auto
    define q}\mathrm{ where q=[:-x, 1:]
    have q0: q}\not=0\mathrm{ unfolding q-def by auto
    have iq: irreducible q by (auto simp: q-def)
    from rt have qa: q dvd a unfolding q-def poly-eq-0-iff-dvd .
    then obtain b where aqb: a=q*b unfolding dvd-def by auto
    from sff(2)[OF ai] have sq: square-free a and mon: degree a}\not=0\mathrm{ by auto
    let ?rem = (\Pi(a,i)\inset bs - {(a,i)}.a^ Suc i)
    have p0:?prod }\not=0\mathrm{ using p pf by auto
    have ?prod =a^ Suc i* ?rem
    by (subst prod.remove[OF - ai], auto)
    also have a` Suc i= q^ Suc i*b` Suc i unfolding aqb by (simp add:
field-simps)
    finally have id: ?prod = q^ Suci*(b` Suci* ?rem) by simp
    hence dvd: q^ Suc i dvd ?prod by auto
    {
    assume q^ Suc (Suc i) dvd ?prod
    hence q dvd ?prod div q^Suc i
            by (metis dvd dvd-0-left-iff dvd-div-iff-mult p0 power-Suc)
    also have ?prod div q^ Suc i=b^\Suc i* ?rem
            unfolding id by (rule nonzero-mult-div-cancel-left, insert q0, auto)
```

```
    finally have qdvd b \veeqdvd ?rem
        using iq irreducible-dvd-pow[OF iq] by auto
    hence False
    proof
        assume qdvd b
        with aqb have q*q dvd a by auto
        with sq[unfolded square-free-def] mon iq show False
        unfolding irreducible e}\mp@subsup{d}{}{-}\mathrm{ def by auto
    next
        assume q dvd ?rem
        from irreducible-dvd-prod[OF iq this]
        obtain b j where bj: (b,j) \in set bs and neq: (a,i)\not=(b,j) and dvd:q dvd b
^Suc j by auto
            from irreducible-dvd-pow[OF iq dvd] have qb: q dvd b .
            from sff(3)[OF ai bj neq] have gcd: coprime a b .
            from qb qa have q dvd gcd a b by simp
            from dvd-imp-degree-le[OF this[unfolded gcd]] iq q0 show False
                using gcd by auto
    qed
}
hence ndvd: ᄀ q` Suc (Suc i) dvd ?prod by blast
with dvd have order x ?prod = Suc i unfolding q-def
    by (metis order-unique-lemma)
    thus ?thesis unfolding ord.
qed
lemma square-free-factorization-order-root-no-mem:
    assumes sff: square-free-factorization p (c,bs)
    and p:p\not=(0 :: 'a poly)
    and no-root: \ a i. (a,i) \in set bs \Longrightarrow poly a x \not=0
    shows order x p=0
proof (rule ccontr)
    assume o0: order x p}\not=
    with order-root[of px] p have 0: poly p x = 0 by auto
    note sff = square-free-factorizationD[OF sff]
    let ?prod = (\prod(a,i)\inset bs. a ^ Suc i)
    from sff have pf:p= smult c ?prod by blast
    with p have c:c\not=0 by auto
    with 0 have 0: poly ?prod x=0 unfolding pf by auto
    define }q\mathrm{ where }q=[:-x,1:
    from 0 have dvd: q dvd}?\mathrm{ ?prod unfolding poly-eq-0-iff-dvd by (simp add: q-def)
    have q0:q\not=0 unfolding q-def by auto
    have iq: irreducible q by (unfold q-def, auto intro:)
from irreducible-dvd-prod[OF iq dvd]
obtain a i where ai: (a,i)\in set bs and dvd: q dvd a}^ Suc i by aut
from irreducible-dvd-pow[OF iq dvd] have dvd: q dvd a .
hence poly a x = 0 unfolding q-def by (simp add: poly-eq-0-iff-dvd q-def)
with no-root[OF ai] show False by simp
```


## qed

lemma square-free-factorization-order-root:
assumes sff: square-free-factorization $p(c, b s)$
and $p: p \neq(0::$ 'a poly $)$
shows order $x p=i \longleftrightarrow(i=0 \wedge(\forall a j .(a, j) \in$ set $b s \longrightarrow$ poly a $x \neq 0)$
$\vee(\exists a j .(a, j) \in$ set $b s \wedge$ poly $a x=0 \wedge i=S u c j))($ is ?l $=(? r 1 \vee$ ? $r 2))$
proof -
note $m e m=$ square-free-factorization-order-root-mem $[O F$ sff $p]$
note no-mem $=$ square-free-factorization-order-root-no-mem[OF sff p]
show ?thesis
proof
assume ? $r 1 \vee$ ? $r 2$
thus ?l
proof
assume ?r2
then obtain $a j$ where $a j:(a, j) \in$ set bs poly a $x=0$ and $i: i=S u c j$ by
auto
from $\operatorname{mem}[O F a j] i$ show ?l by simp
next
assume ?r1
with no-mem[of $x$ ] show ?l by auto
qed
next
assume ?l
show ?r1 V ?r2
proof (cases $\exists a j$. $(a, j) \in$ set bs $\wedge$ poly $a x=0)$
case True
then obtain $a j$ where $(a, j) \in$ set bs poly $a x=0$ by auto
with mem $[$ OF this]〈?l〉
have ?r2 by auto
thus ?thesis ..
next
case False
with no-mem $[$ of $x]\langle ? l\rangle$ have ? $r 1$ by auto
thus ?thesis ..
qed
qed
qed
lemma square-free-factorization-root:
assumes sff: square-free-factorization $p(c, b s)$
and $p: p \neq(0::$ 'a poly $)$
shows $\{x$. poly $p x=0\}=\{x . \exists$ a $i .(a, i) \in$ set $b s \wedge$ poly a $x=0\}$
using square-free-factorization-order-root [OF sff $p] p$
unfolding order-root by auto
lemma square-free-factorizationD': fixes $p::$ ' $a$ poly
assumes sf: square-free-factorization $p(c, b s)$

```
    shows \(p=\) smult \(c\left(\prod(a, i) \leftarrow b s . a \wedge\right.\) Suc \(\left.i\right)\)
    and square-free (prod-list (map fst bs))
    and \(\bigwedge b i .(b, i) \in\) set \(b s \Longrightarrow\) degree \(b \neq 0\)
    and \(p=0 \Longrightarrow c=0 \wedge b s=[]\)
proof -
    note \(s f=\) square-free-factorizationD[OF sf]
    show \(p=\) smult \(c\left(\prod(a, i) \leftarrow b s . a^{\wedge}\right.\) Suc \(\left.i\right)\) unfolding \(s f(1)\) using \(s f(5)\)
    by (simp add: prod.distinct-set-conv-list)
    show \(b s\) : \(\bigwedge b i .(b, i) \in\) set \(b s \Longrightarrow\) degree \(b \neq 0\) using \(s f(2)\) by auto
    show \(p=0 \Longrightarrow c=0 \wedge b s=[]\) using \(s f(4)\).
    show square-free (prod-list (map fst bs))
    proof (rule square-freeI)
    from \(b s\) have \(\bigwedge b . b \in\) set (map fst \(b s) \Longrightarrow b \neq 0\) by fastforce
    thus prod-list (map fst bs) \(\neq 0\) unfolding prod-list-zero-iff by auto
    fix \(q\)
    assume degree \(q>0 q * q\) dvd prod-list (map fst bs)
    from irreducible \(_{d}\)-factor \([O F\) this(1)] this(2) obtain \(q\) where
        irr: irreducible \(q\) and \(d v d: q * q\) dvd prod-list (map fst bs) unfolding dvd-def
by auto
    hence \(d v d^{\prime}: q\) dvd prod-list (map fst bs) unfolding dvd-def by auto
    from irreducible-dvd-prod-list[OF irr dvd'] obtain \(b i\) where
        mem: \((b, i) \in\) set \(b s\) and dvd1: q dvd \(b\) by auto
    from dvd1 obtain \(k\) where \(b: b=q * k\) unfolding \(d v d\)-def by auto
    from split-list \([O F m e m] b\) obtain \(b s 1 b s 2\) where \(b s\) : \(b s=b s 1 @(b, i) \# b s 2\)
by auto
    from \(\operatorname{irr}\) have \(q 0: q \neq 0\) and \(d q\) : degree \(q>0\) unfolding irreducible \(_{d}-\) def by
auto
    from \(s f(2)[O F\) mem, unfolded \(b]\) have square-free \((q * k)\) by auto
    from this[unfolded square-free-def, THEN conjunct2, rule-format, OF dq]
    have \(q k\) : \(\neg q d v d k\) by simp
        from dvd[unfolded bs b] have \(q * q\) dvd \(q *(k *\) prod-list (map fst (bs1 @
bs2)))
            by (auto simp: ac-simps)
    with \(q 0\) have \(q d v d k\) * prod-list (map fst (bs1 @ bs2)) by auto
    with irr \(q k\) have \(q\) dvd prod-list (map fst (bs1 @ bs2)) by auto
    from irreducible-dvd-prod-list[OF irr this] obtain \(b^{\prime} i^{\prime}\) where
            \(m^{\prime}:\left(b^{\prime}, i^{\prime}\right) \in \operatorname{set}\left(b s 1\right.\) @ bs2) and dvd2: \(q\) dvd \(b^{\prime}\) by fastforce
    from dvd1 dvd2 have \(q d v d g c d b b^{\prime}\) by auto
    with dq is-unit-iff-degree [OF q0] have cop: ᄀ coprime b bl by force
    from mem \({ }^{\prime}\) have \(\left(b^{\prime}, i^{\prime}\right) \in\) set \(b s\) unfolding \(b s\) by auto
    from \(s f(3)\left[\right.\) OF mem this] cop have \(b^{\prime}:\left(b^{\prime}, i^{\prime}\right)=(b, i)\)
            by (auto simp add: coprime-iff-gcd-eq-1)
    with mem' \(s f(5)\) [unfolded bs] show False by auto
    qed
qed
lemma square-free-factorization \(I^{\prime}\) : fixes \(p::\) 'a poly
    assumes prod: \(p=\) smult \(c\left(\prod(a, i) \leftarrow b s . a \wedge\right.\) Suc \(\left.i\right)\)
```

```
    and sf: square-free (prod-list (map fst bs))
    and deg: \bigwedge bi.(b,i) \in set bs \Longrightarrow degree b>0
    and 0:p=0\Longrightarrowc=0^bs=[]
    shows square-free-factorization p (c,bs)
    unfolding square-free-factorization-def split
proof (intro conjI impI allI)
    show }p=0\Longrightarrowc=0p=0\Longrightarrowbs=[] using 0 by aut
    {
        fix b i
        assume bi:(b,i)\in set bs
        from deg[OF this] show degree b>0.
    have b dvd prod-list (map fst bs)
        by (intro prod-list-dvd, insert bi, force)
    from square-free-factor[OF this sf] show square-free b .
    }
    show dist: distinct bs
    proof (rule ccontr)
    assume \neg ?thesis
    from not-distinct-decomp[OF this] obtain bs1 bs2 bs3 b i where
            bs:bs=bs1 @ [(b,i)] @ bs2 @ [(b,i)] @ bs3 by force
    hence b*b dvd prod-list (map fst bs) by auto
    with sf[unfolded square-free-def, THEN conjunct2, rule-format, of b]
    have db: degree b=0 by auto
    from bs have (b,i)\in set bs by auto
    from deg[OF this] db show False by auto
    qed
    show p = smult c (\prod(a,i)\inset bs. a ^ Suc i) unfolding prod using dist
    by (simp add: prod.distinct-set-conv-list)
{
    fix a ibj
    assume ai: (a,i)\in set bs and bj: (b,j)\in set bs and diff: (a,i)\not=(b,j)
    from split-list[OF ai] obtain bs1 bs2 where bs:bs = bs1 @ (a,i) # bs2 by
auto
    with bj diff have (b,j) \in set (bs1 @ bs2) by auto
    from split-list[OF this] obtain cs1 cs2 where cs: bs1 @ bs2=cs1 @ (b,j)#
cs2 by auto
    have prod-list (map fst bs) =a* prod-list (map fst (bs1 @ bs2)) unfolding bs
by simp
    also have ...=a*b* prod-list (map fst (cs1@ cs2)) unfolding cs by simp
    finally obtain c where lp: prod-list (map fst bs) =a*b*c by auto
    from deg[OF ai] have 0: gcd a b}=0\mathrm{ by auto
    have gcd: gcd a b * gcd a b dvd prod-list (map fst bs)
        unfolding lp by (simp add: mult-dvd-mono)
    {
    assume degree (gcd a b)>0
    from sf[unfolded square-free-def, THEN conjunct2, rule-format, OF this] gcd
    have False by simp
    }
    hence degree (gcd a b)=0 by auto
```

with 0 show coprime a $b$ using is-unit-gcd is-unit-iff-degree by blast \}

## qed

lemma square-free-factorization-def': fixes $p$ :: 'a poly
shows square-free-factorization $p(c, b s) \longleftrightarrow$
$\left(p=\operatorname{smult} c\left(\prod(a, i) \leftarrow b s . a^{\wedge}\right.\right.$ Suc $\left.\left.i\right)\right) \wedge$
(square-free (prod-list (map fst bs))) $\wedge$
$(\forall b i .(b, i) \in$ set $b s \longrightarrow$ degree $b>0) \wedge$
$(p=0 \longrightarrow c=0 \wedge b s=[])$
using square-free-factorization ${ }^{\prime}\left[\begin{array}{lll}o f & p & c\end{array} b s\right]$
square-free-factorizationI' $[$ of $p c c s]$ by blast
lemma square-free-factorization-smult-prod-listI: fixes $p$ :: ' $a$ poly
assumes sff: square-free-factorization $p$ (c, bs1 @ (smult b (prod-list bs),i) \# bs2)
and $b s: \bigwedge b . b \in$ set $b s \Longrightarrow$ degree $b>0$
shows square-free-factorization $p\left(c * b^{\wedge}(S u c i)\right.$,bs1 @ map $(\lambda b .(b, i)) b s @$ bs2)
proof -
from square-free-factorizationD ${ }^{\prime}(3)[$ OF sff, of smult $b$ (prod-list bs) $i]$
have $b: b \neq 0$ by auto
note $s f f=$ square-free-factorization $D^{\prime}[O F$ sff $]$
show ?thesis
proof (intro square-free-factorizationI', goal-cases)
case 1
thus ?case unfolding sff(1) by (simp add: o-def field-simps smult-power prod-list-pow-suc)

## next

case 2
show ?case using sff(2) by (simp add: ac-simps o-def square-free-smult-iff[OF b])
next
case 3
with $s f f(3)$ bs show ?case by auto
next
case 4
from sff(4)[OF this] show ?case by simp
qed
qed
lemma square-free-factorization-further-factorization: fixes $p$ :: 'a poly
assumes sff: square-free-factorization $p(c, b s)$
and $b s: \bigwedge b i d f s .(b, i) \in$ set $b s \Longrightarrow f b=(d, f s)$
$\Longrightarrow b=$ smult $d($ prod-list $f s) \wedge(\forall f \in$ set fs. degree $f>0)$
and $h: h=\left(\lambda(b, i)\right.$. case $f b$ of $\left.(d, f s) \Rightarrow\left(d^{\wedge} S u c i, \operatorname{map}(\lambda f .(f, i)) f s\right)\right)$
and $g s: g s=\operatorname{map} h b s$
and $d: d=c *$ prod-list (map fst gs)
and es: es $=$ concat (map snd gs)
shows square-free-factorization $p$ (d, es) proof -
note $s f f=$ square-free-factorizationD ${ }^{\prime}[$ OF sff $]$
show ?thesis
proof (rule square-free-factorization $I^{\prime}$ )
assume $p=0$
from $\operatorname{sff}(4)[O F$ this $]$ show $d=0 \wedge e s=[]$ unfolding $d$ es gs by auto
next
have $i d:\left(\prod(a, i) \leftarrow b s . a * a \wedge i\right)=$ smult $($ prod-list $($ map fst gs $))\left(\prod(a, i) \leftarrow e s\right.$. $\left.a * a{ }^{\wedge} i\right)$
unfolding es gs $h$ map-map o-def using $b s$
proof (induct bs)
case (Cons bi bs)
obtain $b i$ where $b i: b i=(b, i)$ by force
obtain $d f s$ where $f: f b=(d, f s)$ by force
from Cons(2)[OF-f, of $i]$ have $b: b=s m u l t d$ (prod-list $f s$ ) unfolding $b i$
by auto
note $I H=\operatorname{Cons}(1)\left[O F \operatorname{Cons}(2)\right.$, of $\left.\lambda-i-{ }^{-} \cdot i\right]$
show ?case unfolding $b i$
by (simp add: fo-def, simp add: bac-simps, subst IH,
auto simp: smult-power prod-list-pow-suc ac-simps)
qed $\operatorname{simp}$
show $p=$ smult $d\left(\prod(a, i) \leftarrow e s . a^{\wedge} S u c i\right)$ unfolding sff(1) using id by (simp add: $d$ )
next
fix $f i$
assume $f:(f, i) \in$ set es
from this[unfolded es] obtain $G$ where $G: G \in$ snd' set gs and $f i:(f i, i) \in$ set $G$ by auto
from $G[$ unfolded $g s]$ obtain $b i$ where $b i:(b, i) \in$ set $b s$
and $G: G=\operatorname{snd}(h(b, i))$ by auto
obtain $d f s$ where $f: f b=(d, f s)$ by force
show degree $f i>0$ by (rule bs[THEN conjunct2, rule-format, OF bif], insert fi $G f$, unfold $h$, auto)
next
have id: $\exists$ c. prod-list (map fst bs) $=$ smult $c($ prod-list (map fst es) $)$
unfolding es gs map-map o-def using bs
proof (induct bs)
case (Cons bi bs)
obtain $b i$ where $b i$ : $b i=(b, i)$ by force
obtain $d f s$ where $f: f b=(d, f s)$ by force
from Cons(2)[OF-f, of $i]$ have $b: b=s m u l t d$ (prod-list $f s$ ) unfolding $b i$ by auto
have $\exists$ c. prod-list (map fst bs) $=$ smult $c($ prod-list (map fst (concat (map $(\lambda x$. snd (h x)) bs $)$ )) )
by (rule Cons(1), rule Cons(2), auto)
then obtain $c$ where
IH: prod-list $($ map fst bs) $)=$ smult $c($ prod-list $($ map fst $($ concat $(\operatorname{map})(\lambda x$.

```
snd (h x)) bs)))) by auto
    show ?case unfolding bi
        by (intro exI[of - c*d], auto simp: b IH, auto simp: h f[unfolded b] o-def)
    qed (intro exI[of-1], auto)
    then obtain c where prod-list (map fst bs)=smult c(prod-list (map fst es))
by blast
    from sff(2)[unfolded this] show square-free (prod-list (map fst es))
        by (metis smult-eq-0-iff square-free-def square-free-smult-iff)
    qed
qed
lemma square-free-factorization-prod-listI: fixes p :: 'a poly
    assumes sff:square-free-factorization p (c,bs1 @ ((prod-list bs),i) # bs2)
    and bs:\bigwedge b. b set bs \Longrightarrow degree b>0
    shows square-free-factorization p (c,bs1 @ map (\lambda b. (b,i)) bs @ bs2)
    using square-free-factorization-smult-prod-listI[of p c bs1 1 bs i bs2] sff bs by
auto
lemma square-free-factorization-factorI: fixes p :: 'a poly
    assumes sff:square-free-factorization p (c,bs1 @ (a,i) # bs2)
    and r: degree r}\not=0\mathrm{ and }s:\mathrm{ degree }s\not=
    and a:a=r*s
    shows square-free-factorization p (c,bs1 @ ((r,i) # (s,i) # bsQ))
    using square-free-factorization-prod-listI[of p c bs1 [r,s]i bsQ] sff r s a by auto
end
lemma monic-square-free-irreducible-factorization: assumes mon: monic ( \(f\) :: ' \(b\)
:: field poly)
    and sf: square-free f
    shows \existsP. finite }P\wedgef=\PiP\wedgeP\subseteq{q. irreducible q\wedge monic q
proof -
    from mon have f0: f}\not=0\mathrm{ by auto
    from monic-irreducible-factorization[OF assms(1)] obtain P n where
        P: finite P P\subseteq{q. irreducible }\mp@subsup{|}{|}{}q\wedge\mathrm{ monic q} and f:f=(\a<P.a^ Suc (n
a)) by auto
    have *: }\foralla\inP.na=
    proof (rule ccontr)
    assume \neg ?thesis
    then obtain }a\mathrm{ where }a:a\inP\mathrm{ and n: n a}\not=0\mathrm{ by auto
    have f=a^}(Suc(na))*(\prodb\inP-{a}.b^\Suc (nb)
            unfolding f by (rule prod.remove[OF P(1) a])
    with n have a*a dvd f by (cases n a, auto)
    with sf[unfolded square-free-def] f0 have degree a=0 by auto
    with a P(2)[unfolded irreducibled}\mp@subsup{d}{-}{-def] show False by auto
qed
have f}=\P\mathrm{ unfolding }
    by (rule prod.cong[OF refl], insert *, auto)
    with P show ?thesis by auto
```


## qed

```
context
    assumes SORT-CONSTRAINT(' a :: {field, factorial-ring-gcd})
begin
lemma monic-factorization-uniqueness:
fixes P::'a poly set
assumes finite-P: finite P
    and }PQ:\prodP=\prod
    and P:P\subseteq{q. irreducible }\mp@subsup{|}{d}{}q\wedge\mathrm{ monic q}
and finite-Q: finite Q
    and Q:Q\subseteq{q. irreducible }\mp@subsup{\mp@code{d}}{}{q}\wedge\mathrm{ ^ monic q}
shows }P=
proof (rule; rule subsetI)
    fix }x\mathrm{ assume }x:x\in
    have irr-x: irreducible x using x P by auto
    then have \existsa\inQ.x dvd id a
    proof (rule irreducible-dvd-prod)
            show }x\mathrm{ dvd prod id Q using PQ x
            by (metis dvd-refl dvd-prod finite-P id-apply prod.cong)
    qed
    from this obtain a where a: a\inQ and x-dvd-a: x dvd a unfolding id-def by
blast
    have }x=a\mathrm{ using x P a Q irreducibled}\mp@subsup{d}{d}{}dvd-eq[OF - - x-dvd-a] by fas
    thus }x\inQ\mathrm{ using a by simp
next
    fix x assume x: x \inQ
    have irr-x: irreducible x using x Q by auto
    then have }\existsa\inP.x dvd id 
    proof (rule irreducible-dvd-prod)
        show x dvd prod id P using PQ x
            by (metis dvd-refl dvd-prod finite-Q id-apply prod.cong)
    qed
    from this obtain a where a: a\inP and x-dvd-a: x dvd a unfolding id-def by
blast
    have x=a using x P a Q irreducible }\mp@subsup{d}{d}{}-dvd-eq[OF - x-dvd-a] by fas
    thus }x\inP\mathrm{ using a by simp
qed
end
```


### 11.2 Yun factorization and homomorphisms

locale field-hom-0' $=$ field-hom hom
for hom :: ' $a::\{$ field-char-0, field-gcd $\} \Rightarrow$ 'b :: \{field-char-0,field-gcd $\}$
begin
sublocale field-hom' ..
end

```
lemma (in field-hom-0') yun-factorization-main-hom:
    defines hp: hp\equiv map-poly hom
    defines hpi: hpi\equiv\operatorname{map}(\lambda(f,i).(hpf,i:: nat))
    assumes monic: monic p and f:f=p div gcd p (pderiv p) and g:g=pderiv p
div gcd p (pderiv p)
    shows yun-gcd.yun-factorization-main gcd (hpf) (hp g)i(hpi as) = hpi(yun-gcd.yun-factorization-main
gcd fgi as)
proof -
    let ?P = \lambda f g.\forall i as. yun-gcd.yun-factorization-main gcd (hp f) (hp g) i (hpi
as)=hpi(yun-gcd.yun-factorization-main gcd fgi as)
    note ind = yun-factorization-induct[OF --fg monic, of ?P, rule-format]
    interpret map-poly-hom: map-poly-inj-comm-ring-hom..
    interpret p: inj-comm-ring-hom hp unfolding hp..
    note homs = map-poly-gcd[folded hp]
        map-poly-pderiv[folded hp]
        p.hom-minus
        map-poly-div[folded hp]
    show ?thesis
    proof (induct rule: ind)
        case (1fgias)
    show ?case unfolding yun-gcd.yun-factorization-main.simps[of - hp f] yun-gcd.yun-factorization-main.simp
- f]
    unfolding 1 by simp
    next
        case (2fgias)
        have id: \fi fis. hpi ((f,i) # fis)=(hpf,i) # hpi fis unfolding hpi by auto
    show ?case unfolding yun-gcd.yun-factorization-main.simps[of - hp f] yun-gcd.yun-factorization-main.simp
-f]
        unfolding p.hom-1-iff
        unfolding Let-def
        unfolding homs[symmetric] id[symmetric]
        unfolding 2(2) by simp
    qed
qed
lemma square-free-square-free-factorization:
    square-free ( }p::\mathrm{ 'a :: {field,factorial-ring-gcd,semiring-gcd-mult-normalize} poly)
    degree p}=0\Longrightarrow\mathrm{ square-free-factorization p (1,[(p,0)])
    by (intro square-free-factorizationI', auto)
lemma constant-square-free-factorization:
    degree p=0\Longrightarrow square-free-factorization p (coeff p 0,[])
    by (drule degree0-coeffs [of p]) (auto simp: square-free-factorization-def)
lemma (in field-hom-0') yun-monic-factorization:
    defines hp: hp\equiv map-poly hom
    defines hpi: hpi\equiv\operatorname{map}(\lambda(f,i).(hpf,i :: nat))
    assumes monic: monic f
```

```
shows yun-gcd.yun-monic-factorization gcd (hp f) = hpi (yun-gcd.yun-monic-factorization
gcd f)
proof -
    interpret map-poly-hom: map-poly-inj-comm-ring-hom..
    interpret p: inj-ring-hom hp unfolding hp..
    have hpiN: hpi [] = [] unfolding hpi by simp
    obtain res where res =
        yun-gcd.yun-factorization-main gcd (f div gcd f(pderiv f))(pderiv f div gcd f
(pderiv f)) 0 [] by auto
    note homs = map-poly-gcd[folded hp]
        map-poly-pderiv[folded hp]
        p.hom-minus
        map-poly-div[folded hp]
        yun-factorization-main-hom[folded hp, folded hpi, symmetric, OF monic refl
refl, of - Nil, unfolded hpiN]
    this
    show ?thesis
    unfolding yun-gcd.yun-monic-factorization-def Let-def
    unfolding homs[symmetric]
    unfolding hpi
        by (induct res,auto)
qed
lemma (in field-hom-0') yun-factorization-hom:
    defines hp: hp\equiv map-poly hom
    defines hpi: hpi\equiv\operatorname{map}(\lambda(f,i).(hpf,i:: nat))
    shows yun-factorization gcd (hp f) = map-prod hom hpi (yun-factorization gcd
f)
    using yun-monic-factorization[of smult (inverse (coeff f (degree f))) f]
    unfolding yun-factorization-def Let-def hp hpi
    by (auto simp: hom-distribs)
lemma (in field-hom-0') square-free-map-poly:
    square-free (map-poly hom f)=square-free f
proof -
    interpret map-poly-hom: map-poly-inj-comm-ring-hom..
    show ?thesis unfolding square-free-iff-separable separable-def
    by (simp only: hom-distribs [symmetric] )
        (simp add: coprime-iff-gcd-eq-1 map-poly-gcd [symmetric])
qed
end
```


## 12 GCD of rational polynomials via GCD for integer polynomials

This theory contains an algorithm to compute GCDs of rational polynomials via a conversion to integer polynomials and then invoking the integer polynomial GCD algorithm.

```
theory Gcd-Rat-Poly
imports
    Gauss-Lemma
    HOL-Computational-Algebra.Field-as-Ring
begin
definition gcd-rat-poly :: rat poly }=>\mathrm{ rat poly }=>\mathrm{ rat poly where
    gcd-rat-poly fg=(let
        f}=\mp@code{snd (rat-to-int-poly f);
        g' = snd (rat-to-int-poly g);
        h = map-poly rat-of-int (gcd f' g')
    in smult (inverse (lead-coeff h)) h)
lemma gcd-rat-poly[simp]: gcd-rat-poly = gcd
proof (intro ext)
    fix fg
    let ?ri = map-poly rat-of-int
    obtain a' f}\mp@subsup{f}{}{\prime}\mathrm{ where faf': rat-to-int-poly f}=(\mp@subsup{a}{}{\prime},\mp@subsup{f}{}{\prime})\mathrm{ by force
    from rat-to-int-poly[OF this] obtain a where
        f:f= smult a (?ri f ) and a: a}=0\mathrm{ by auto
    obtain b}\mp@subsup{b}{}{\prime}\mp@subsup{g}{}{\prime}\mathrm{ where gbg': rat-to-int-poly g}=(\mp@subsup{b}{}{\prime},\mp@subsup{g}{}{\prime})\mathrm{ by force
    from rat-to-int-poly[OF this] obtain b where
        g:g= smult b (?ri g') and b: b\not=0 by auto
    define }h\mathrm{ where }h=gcd\mp@subsup{f}{}{\prime}\mp@subsup{g}{}{\prime
    let ?h = ?ri h
    define lc where lc= inverse (coeff ?h (degree ?h))
    let ?gcd = smult lc ?h
    have id: gcd-rat-poly f g=?gcd
        unfolding lc-def h-def gcd-rat-poly-def Let-def faf' gbg' snd-conv by auto
    show gcd-rat-poly f g}=gcdfg\mathrm{ unfolding id
    proof (rule gcdI)
        have h dvd f' unfolding h-def by auto
        hence ?h dvd ?ri f' unfolding dvd-def by (auto simp: hom-distribs)
        hence ?h dvd f}\mathrm{ unfolding f by (rule dvd-smult)
        thus dvd-f: ? gcd dvd f
            by (metis dvdE inverse-zero-imp-zero lc-def leading-coeff-neq-0 mult-eq-0-iff
smult-dvd-iff)
        have h dvd g' unfolding h-def by auto
        hence ?h dvd ?ri g' unfolding dvd-def by (auto simp: hom-distribs)
        hence ?h dvd g unfolding g by (rule dvd-smult)
        thus dvd-g: ? gcd dvd g
            by (metis dvdE inverse-zero-imp-zero lc-def leading-coeff-neq-0 mult-eq-0-iff
```

```
smult-dvd-iff)
    show normalize ?gcd = ?gcd
        by (cases lc = 0)
            (simp-all add: normalize-poly-def pCons-one field-simps lc-def)
    fix }
    assume dvd: k dvd fkdvd g
    obtain }\mp@subsup{k}{}{\prime}c\mathrm{ where kck: rat-to-normalized-int-poly k=(c,k') by force
    from rat-to-normalized-int-poly[OF this] have k: k=smult c (?ri k') and c:c
\not=0}\mathrm{ by auto
    from dvd(1) have kf: k dvd ?rif'unfolding f using a by (rule dvd-smult-cancel)
    from dvd(2) have kg: kdvd ?ri g' unfolding g using b by (rule dvd-smult-cancel)
        from kf kg obtain kf kg where kf: ?ri f' =k*kf and kg: ?ri g' = k* kg
unfolding dvd-def by auto
    from rat-to-int-factor-explicit[OF kf kck] have kf: k' dvd f' unfolding dvd-def
by blast
    from rat-to-int-factor-explicit[OF kg kck] have kg: k' dvd g' unfolding dvd-def
by blast
    from kf kg have k' dvd h unfolding h-def by simp
    hence ?ri k' dvd ?ri h unfolding dvd-def by (auto simp: hom-distribs)
    hence k dvd ?ri h unfolding k using c by (rule smult-dvd)
    thus k dvd?gcd by (rule dvd-smult)
    qed
qed
lemma gcd-rat-poly-unfold[code-unfold]: gcd \(=\) gcd-rat-poly by simp
end
```


## 13 Rational Factorization

We combine the rational root test, the formulas for explicit roots, and the Kronecker's factorization algorithm to provide a basic factorization algorithm for polynomial over rational numbers. Moreover, also the roots of a rational polynomial can be determined.

```
theory Rational-Factorization
imports
    Explicit-Roots
    Kronecker-Factorization
    Square-Free-Factorization
    Rational-Root-Test
    Gcd-Rat-Poly
    Show.Show-Poly
begin
function roots-of-rat-poly-main :: rat poly \(\Rightarrow\) rat list where
    roots-of-rat-poly-main \(p=(\) let \(n=\) degree \(p\) in if \(n=0\) then [] else if \(n=1\) then
[roots1 p]
    else if \(n=2\) then rat-roots2 \(p\) else
    case rational-root-test \(p\) of None \(\Rightarrow[] \mid\) Some \(x \Rightarrow x \#\) roots-of-rat-poly-main ( \(p\)
```

```
div [:-x,1:]))
    by pat-completeness auto
termination by (relation measure degree,
    auto dest: rational-root-test(1) intro!: degree-div-less simp: poly-eq-0-iff-dvd)
lemma roots-of-rat-poly-main-code[code]: roots-of-rat-poly-main p = (let n = degree
p in if n=0 then [] else if n=1 then [roots1 p]
    else if n=2 then rat-roots2 p else
    case rational-root-test p of None }=>[]|\mathrm{ Some x }=>x#\mathrm{ roots-of-rat-poly-main (p
div [:-x,1:]))
proof -
    note d = roots-of-rat-poly-main.simps[of p] Let-def
    show ?thesis
    proof (cases rational-root-test p)
    case (Some x)
    let ? }x=[:-x,1:
    from rational-root-test(1)[OF Some] have ?x dvd p
            by (simp add: poly-eq-0-iff-dvd)
    from dvd-mult-div-cancel[OF this]
    have pp: p div ? }x=?x*(p\mathrm{ div ?x) div ?x by simp
    then show ?thesis unfolding d Some by auto
    qed (simp add:d)
qed
lemma roots-of-rat-poly-main: p}\not=0\Longrightarrow\mathrm{ set (roots-of-rat-poly-main p) ={x.poly
px=0}
proof (induct p rule: roots-of-rat-poly-main.induct)
    case (1 p)
    note IH=1(1)
    note p=1(2)
    let ? n = degree p
    let ?rr = roots-of-rat-poly-main
    show ?case
    proof (cases ?n=0)
    case True
    from roots0[OF p True] True show ?thesis by simp
    next
    case False note 0 = this
    show ?thesis
    proof (cases ?n = 1)
            case True
            from roots1[OF True] True show ?thesis by simp
    next
            case False note 1 = this
            show ?thesis
            proof (cases ? n = 2)
                    case True
                    from rat-roots2[OF True] True show ?thesis by simp
```

```
        next
            case False note 2 = this
            from 012 have id: ?rr p = (case rational-root-test p of None }=>[]|\mathrm{ Some
x 
                x # ?rr ( p div [: -x, 1 :])) by simp
            show ?thesis
            proof (cases rational-root-test p)
                case None
                from rational-root-test(2)[OF None] None id show ?thesis by simp
            next
                case (Some x)
                    from rational-root-test(1)[OF Some] have [: -x, 1:] dvd p
                    by (simp add: poly-eq-0-iff-dvd)
                    from dvd-mult-div-cancel[OF this]
                    have pp: p=[:-x,1:]*(p div [:-x,1:]) by simp
                    with p have p:p div [:- x, 1:] }\not=0\mathrm{ by auto
                    from arg-cong[OF pp, of \lambda p. {x. poly p x = 0}]
                    rational-root-test(1)[OF Some] IH[OF refl 0 1 2 Some p] show ?thesis
                unfolding id Some by auto
            qed
        qed
    qed
    qed
qed
declare roots-of-rat-poly-main.simps[simp del]
definition roots-of-rat-poly :: rat poly }=>\mathrm{ rat list where
    roots-of-rat-poly p \equiv let (c,pis) = yun-factorization gcd-rat-poly p in
        concat (map (roots-of-rat-poly-main o fst) pis)
lemma roots-of-rat-poly: assumes p: p\not=0
    shows set (roots-of-rat-poly p)={x. poly p x=0}
proof -
    obtain c pis where yun: yun-factorization gcd p=(c,pis) by force
    from yun
    have res: roots-of-rat-poly p = concat (map (roots-of-rat-poly-main \circ fst) pis)
    by (auto simp: roots-of-rat-poly-def split: if-splits)
    note yun = square-free-factorizationD (1,2,4)[OF yun-factorization(1)[OF yun]]
    from yun(1) p have c:c\not=0 by auto
    from yun(1) have p:p=smult c (\Pi(a,i)\inset pis.a^ Suc i).
    have {x. poly p x=0} ={x.poly }(\Pi(a,i)\in\mathrm{ set pis.a^ Suc i) x=0}
        unfolding p using c by auto
    also have ...= \bigcup ((\lambda p.{x. poly p x=0})'fst'set pis)(is - = ?r)
    by (subst poly-prod-0, force+)
    finally have r:{x. poly px=0}=?r.
    {
    fix pi
    assume p:(p,i)\in set pis
```

```
    have set (roots-of-rat-poly-main p)={x.poly p x = 0}
            by (rule roots-of-rat-poly-main, insert yun(2) p, force)
    } note main = this
    have set (roots-of-rat-poly p)=\bigcup((\lambda(p,i). set (roots-of-rat-poly-main p))'set
pis)
    unfolding res o-def by auto
    also have ... = ?r using main by auto
    finally show ?thesis unfolding r by simp
qed
definition root-free :: ' }a\mathrm{ :: comm-semiring-0 poly }=>\mathrm{ bool where
    root-free p=(degree p=1\vee (\forall x. poly p x = 0))
lemma irreducible-root-free:
    fixes p :: 'a :: idom poly
    assumes irreducible p shows root-free p
proof-
    from assms have p0:p\not=0 by auto
    {
        fix }
        assume poly p x=0 and degp: degree p}\not=
        hence [:-x,1:] dvd p using poly-eq-0-iff-dvd by blast
        then obtain q where p:p=[:-x,1:]*q by (elim dvdE)
        with p0 have q0: q*=0 by auto
        from irreducibleD[OF assms p]
        have q dvd 1 by (metis one-neq-zero poly-1 poly-eq-0-iff-dvd)
        then have degree q=0 by (simp add: poly-dvd-1)
        with degree-mult-eq[of [:-x,1:] q, folded p] q0 degp
        have False by auto
    }
    thus ?thesis unfolding root-free-def by auto
qed
partial-function (tailrec) factorize-root-free-main :: rat poly }=>\mathrm{ rat list }=>\mathrm{ rat poly
list }=>\mathrm{ rat }\times\mathrm{ rat poly list where
    [code]: factorize-root-free-main p xs fs = (case xs of Nil }
        let l= coeff p (degree p);q= smult (inverse l) p in (l, (if q=1 then fs else q
# fs) )
    | x # xs =>
        if poly px=0 then factorize-root-free-main (p div [:-x,1:]) (x # xs)([:-x,1:]
# fs)
    else factorize-root-free-main p xs fs)
definition factorize-root-free :: rat poly }=>\mathrm{ rat }\times\mathrm{ rat poly list where
    factorize-root-free p=(if degree p=0 then (coeff p 0,[]) else
        factorize-root-free-main p (roots-of-rat-poly p) [])
lemma factorize-root-free-0 [simp]: factorize-root-free 0 = (0,[])
    unfolding factorize-root-free-def by simp
```

lemma factorize-root-free: assumes res: factorize-root-free $p=(c, q s)$
shows $p=$ smult $c$ (prod-list $q s$ )
$\wedge q . q \in$ set $q s \Longrightarrow$ root-free $q \wedge$ monic $q \wedge$ degree $q \neq 0$
proof -
have $p=$ smult $c($ prod-list $q s) \wedge(\forall q \in$ set qs. root-free $q \wedge$ monic $q \wedge$ degree $q \neq 0$ )
proof (cases degree $p=0$ )
case True
thus ?thesis using res unfolding factorize-root-free-def by (auto dest: de-
gree0-coeffs)
next
case False
hence $p 0: p \neq 0$ by auto
define $f s$ where $f s=$ ([] :: rat poly list)
define $x s$ where $x s=$ roots-of-rat-poly $p$
define $q$ where $q=p$
obtain $n$ where $n$ : $n=$ degree $q+$ length $x s$ by auto
have prod: $p=q *$ prod-list fs unfolding $q$-def fs-def by auto
have sub: $\{x$. poly $q x=0\} \subseteq$ set $x s$ using roots-of-rat-poly[OF p0] unfolding $q$-def $x s$-def by auto
have $f s: \bigwedge q . q \in$ set $f s \Longrightarrow$ root-free $q \wedge$ monic $q \wedge$ degree $q \neq 0$ unfolding $f s$-def by auto
have res: factorize-root-free-main $q$ xs $f s=(c, q s)$ using res False
unfolding $x s$-def $f s$-def $q$-def factorize-root-free-def by auto
from False have $q \neq 0$ unfolding $q$-def by auto
from prod sub fs res $n$ this show ?thesis
proof (induct $n$ arbitrary: $q$ fs xs rule: wf-induct[OF wf-less])
case ( $1 n q f s x s$ )
note $\operatorname{simp}=$ factorize-root-free-main.simps $[$ of $q$ xs $f s]$
note $I H=1(1)[$ rule-format $]$
note $0=1$ (2-) [unfolded simp]
show ?case
proof (cases xs)
case Nil
note $0=0$ [unfolded Nil Let-def]
hence no-rt: $\bigwedge x$. poly $q x \neq 0$ by auto
hence $q: q \neq 0$ by auto
let ?r $=$ smult $($ inverse $c) q$
define $r$ where $r=$ ? $r$
from $0(4-5)$ have $c: c=$ coeff $q$ (degree $q)$ and $q s: q s=($ if $r=1$ then $f s$ else $r \# f s$ ) by (auto simp: $r$-def)
from $q$ c qs $0(1)$ have $c 0: c \neq 0$ and $p: p=\operatorname{smult} c($ prod-list $(r \# f s))$
by (auto simp: r-def)
from $p$ have $p: p=$ smult $c(p r o d-l i s t ~ q s)$ unfolding $q s$ by auto
from $0(2,5) c 0 c$ have root-free ?r monic ? $r$
unfolding root-free-def by auto
with $0(3)$ have $\wedge q . q \in$ set $q s \Longrightarrow$ root-free $q \wedge$ monic $q \wedge$ degree $q \neq 0$ unfolding $q s$
by (cases degree $q=0$, insert degree 0 -coeffs $[$ of $q]$, auto split: if-splits simp: $r-d e f)$
with $p$ show ?thesis by auto
next
case (Cons $x$ xs)
note $0=0$ [unfolded Cons]
show ?thesis
proof (cases poly $q x=0$ )
case True
let $? q=q \operatorname{div}[:-x, 1:]$
let $? x=[:-x, 1:]$
let ? $f s=$ ? $x \# f s$
let ? $x s=x \# x s$
from True have $q: q=? q * ? x$
by (metis dvd-mult-div-cancel mult.commute poly-eq-0-iff-dvd)
with $0(6)$ have $q^{\prime}: ? q \neq 0$ by auto
have deg: degree $q=$ Suc (degree ? $q$ ) unfolding arg-cong[OF $q$, of degree] by (subst degree-mult-eq[OF q], auto)
hence $n$ : degree ? $q+$ length ? $x s<n$ unfolding $0(5)$ by auto
from arg-cong[OF $q$, of poly] 0 (2) have $r t:\{x$. poly ? $q x=0\} \subseteq$ set ? xs by auto
have $p: p=? q *$ prod-list ?fs unfolding prod-list.Cons 0 (1) mult.assoc[symmetric] $q[$ symmetric $]$..
have root-free? $x$ unfolding root-free-def by auto
with $O(3)$ have $r f: \bigwedge f . f \in$ set ? fs $\Longrightarrow$ root-free $f \wedge$ monic $f \wedge$ degree $f$ $\neq 0$ by auto
from True $0(4)$ have res: factorize-root-free-main ?q ? xs ? $f s=(c, q s)$ by simp
show ?thesis
by (rule $I H[O F-p$ rt rf res refl $q$ ], insert $n$, auto)
next
case False
with 0 (4) have res: factorize-root-free-main $q$ xs $f s=(c, q s)$ by simp
from $0(5)$ obtain $m$ where $m: m=$ degree $q+$ length $x s$ and $n: n=$ Suc $m$ by auto
from False $0(2)$ have $r t$ : $\{x$. poly $q x=0\} \subseteq$ set xs by auto
show ?thesis by (rule $\operatorname{IH}[O F-O(1)$ rt $0(3)$ res $m 0(6)]$, unfold $n$, auto)
qed
qed
qed
qed
thus $p=$ smult $c($ prod-list $q s)$
$\wedge q . q \in$ set $q s \Longrightarrow$ root-free $q \wedge$ monic $q \wedge$ degree $q \neq 0$ by auto
qed
definition rational-proper-factor :: rat poly $\Rightarrow$ rat poly option where
rational-proper-factor $p=$ (if degree $p \leq 1$ then None
else if degree $p=2$ then (case rat-roots2 $p$ of Nil $\Rightarrow$ None $\mid$ Cons $x$ xs $\Rightarrow$ Some

```
[:-x,1 :])
    else if degree p=3 then (case rational-root-test p of None }=>\mathrm{ None | Some x
A Some [:-x,1:])
    else kronecker-factorization-rat p)
lemma degree-1-dvd-root: assumes q: degree ( }q:: 'a :: field poly) = 1
    and rt: \bigwedge x. poly p x\not=0
    shows }\negqdvd
proof -
    from degree1-coeffs[OF q] obtain ab where q:q=[:b,a:] and a:a\not=0 by
auto
    have q: q = smult a [:-(-b/a), 1 :] unfolding q
        by (rule poly-eqI, unfold coeff-smult, insert a, auto simp: field-simps coeff-pCons
        split: nat.splits)
    show ?thesis unfolding q smult-dvd-iff poly-eq-0-iff-dvd[symmetric, of - p] using
a rt by auto
qed
```

lemma rational-proper-factor:
degree $p>0 \Longrightarrow$ rational-proper-factor $p=$ None $\Longrightarrow$ irreducible $_{d} p$ rational-proper-factor $p=$ Some $q \Longrightarrow q$ dvd $p \wedge$ degree $q \geq 1 \wedge$ degree $q<$
degree $p$
proof -
let ? $r p=$ rational-proper-factor $p$
let $? r r=$ rational-root-test
note $d=$ rational-proper-factor-def $[$ of $p]$
have (degree $p>0 \longrightarrow$ ? $r p=$ None $\longrightarrow$ irreducible $\left._{d} p\right) \wedge$
$($ ?rp $=$ Some $q \longrightarrow q$ dvd $p \wedge$ degree $q \geq 1 \wedge$ degree $q<$ degree $p)$
proof (cases degree $p=0$ )
case True
thus ?thesis unfolding $d$ by auto
next
case False note $0=$ this
show ?thesis
proof (cases degree $p=1$ )
case True
hence ${ }^{2} r p=$ None unfolding $d$ by auto
with linear-irreducible $e_{d}[$ OF True $]$ show ?thesis by auto
next
case False note $1=$ this
show ?thesis
proof (cases degree $p=2$ )
case True
hence rp: ? $r p=($ case rat-roots2 $p$ of Nil $\Rightarrow$ None $\mid$ Cons x xs $\Rightarrow$ Some
[:-x, 1 :]) unfolding $d$ by auto
show ?thesis

```
        proof (cases rat-roots2 p)
            case Nil
            with rp have rp: ?rp = None by auto
            from Nil rat-roots2[OF True] have nex: \neg (\exists x. poly p x = 0) by auto
            have irreducible ed
            proof (rule irreducible }\mp@subsup{|}{d}{}\mathrm{ )
            fix q r :: rat poly
            assume degree q>0 degree q< degree p and p:p=q*r
            with True have dq: degree q}=1\mathrm{ by auto
            have \negq dvd p by (rule degree-1-dvd-root[OF dq], insert nex, auto)
            with p show False by auto
            qed (insert True, auto)
            with rp show ?thesis by auto
        next
            case (Cons x xs)
            from Cons rat-roots2[OF True] have poly p x=0 by auto
            from this[unfolded poly-eq-0-iff-dvd] have x: [:-x, 1 :] dvd p by auto
            from Cons rp have rp: ?rp = Some ([: - x, 1 :]) by auto
            show ?thesis using True }x\mathrm{ unfolding rp by auto
        qed
next
case False note 2 = this
show ?thesis
proof (cases degree p=3)
    case True
    hence rp: ?rp = (case ?rr p of None }=>\mathrm{ N None | Some x }=>\mathrm{ Some [:- x,
1:]) unfolding d by auto
    show ?thesis
    proof (cases ?rr p)
        case None
            from rational-root-test(2)[OF None] have nex: }\neg(\exists\mathrm{ x. poly p x = 0)
            from rp[unfolded None] have rp: ?rp = None by auto
            have irreducible d p
            proof (rule irreducible d}I\mathrm{ I)
            fix q :: rat poly
            assume degree q>0 degree q}\leq\mathrm{ degree p div 2
            with True have dq: degree q=1 by auto
            show \negq dvd p
                    by (rule degree-1-dvd-root[OF dq], insert nex, auto)
        qed (insert True, auto)
        with rp show ?thesis by auto
    next
        case (Some x)
        from rational-root-test(1)[OF Some] have poly px=0.
        from this[unfolded poly-eq-0-iff-dvd] have x:[:-x, 1:] dvd p by auto
        from Some rp have rp: ?rp = Some ([:-x,1 :]) by auto
        show ?thesis using True x unfolding rp by auto
    qed
```

by auto

```
            next
                    case False note 3 = this
            let ?kp = kronecker-factorization-rat p
            from 012 3 have d4: degree p\geq4 and d1: degree p\geq1 by auto
            hence rp: ? rp = ? kp using d4 d by auto
            show ?thesis
            proof (cases ?kp)
                case None
                    with rp kronecker-factorization-rat(2)[OF None d1] show ?thesis by
auto
                    next
                    case (Some q)
                    with rp kronecker-factorization-rat(1)[OF Some] show ?thesis by auto
                    qed
            qed
        qed
    qed
qed
thus degree p>0\Longrightarrow rational-proper-factor p=None \Longrightarrow irreducible d}
    rational-proper-factor p=Some q\Longrightarrowqdvd p^degree q\geq1^ degree q<
degree p by auto
qed
function factorize-rat-poly-main :: rat }=>\mathrm{ rat poly list }=>\mathrm{ rat poly list }=>\mathrm{ rat }\times\mathrm{ rat
poly list where
    factorize-rat-poly-main c irr [] = (c,irr)
| factorize-rat-poly-main c irr ( 
    then factorize-rat-poly-main (c* coeff p 0) irr ps
    else (case rational-proper-factor p of
        None }=>\mathrm{ factorize-rat-poly-main c (p# irr) ps
    | Some q = factorize-rat-poly-main c irr (q#p div q # ps)))
    by pat-completeness auto
definition factorize-rat-poly-main-wf-rel = inv-image (mult1 {(x,y).x<y}) (\lambda(c,
irr, ps).mset (map degree ps))
lemma wf-factorize-rat-poly-main-wf-rel:wf factorize-rat-poly-main-wf-rel
    unfolding factorize-rat-poly-main-wf-rel-def using wf-mult1[OF wf-less] by auto
lemma factorize-rat-poly-main-wf-rel-sub:
    ((a,b,ps), (c,d, p# ps)) \in factorize-rat-poly-main-wf-rel
    unfolding factorize-rat-poly-main-wf-rel-def
    by (auto intro: mult1I [of -- - {#}])
lemma factorize-rat-poly-main-wf-rel-two: assumes degree q < degree p degree r
< degree p
    shows ((a,b,q #r # ps), (c,d,p # ps)) \in factorize-rat-poly-main-wf-rel
    unfolding factorize-rat-poly-main-wf-rel-def mult1-def
    using add-eq-conv-ex assms ab-semigroup-add-class.add-ac
```


## by fastforce

## termination

proof (relation factorize-rat-poly-main-wf-rel,
rule wf-factorize-rat-poly-main-wf-rel, rule factorize-rat-poly-main-wf-rel-sub, rule factorize-rat-poly-main-wf-rel-sub, rule factorize-rat-poly-main-wf-rel-two)
fix $p q$
assume $r f$ : rational-proper-factor $p=$ Some $q$ and dp: degree $p \neq 0$
from rational-proper-factor (2)[OF rf]
have dvd: $q d v d p$ and deg: $1 \leq$ degree $q$ degree $q<$ degree $p$ by auto
show degree $q<$ degree $p$ by fact
from dvd have $p=q *(p$ div $q)$ by auto
from arg-cong[OF this, of degree]
have degree $p=$ degree $q+$ degree ( $p$ div $q$ )
by (subst degree-mult-eq[symmetric], insert dp, auto)
with deg
show degree ( $p$ div $q$ ) < degree $p$ by simp

## qed

declare factorize-rat-poly-main.simps[simp del]
lemma factorize-rat-poly-main:
assumes factorize-rat-poly-main c irr ps $=(d, q s)$
and Ball (set irr) irreducible ${ }_{d}$
shows Ball (set qs) irreducible ${ }_{d}$ (is ?g1)
and smult $c($ prod-list $($ irr @ ps $))=$ smult d (prod-list qs) (is ?g2)
proof (atomize(full), insert assms, induct c irr ps rule: factorize-rat-poly-main.induct)
case (1 c irr)
thus ?case by (auto simp: factorize-rat-poly-main.simps)
next
case (2 c irr pps)
note $I H=2(1-3)$
note res $=2(4)[$ unfolded factorize-rat-poly-main.simps(2)[of cirr pess]]
note $\mathrm{irr}=2(5)$
let ?f = factorize-rat-poly-main
show ?case
proof (cases degree $p=0$ )
case True
with res have res: ?f $(c *$ coeff $p 0)$ irr $p s=(d, q s)$ by simp
from degree 0 -coeffs $[$ OF True $]$ obtain $a$ where $p: p=[: a:]$ by auto
from $\operatorname{IH}(1)[$ OF True res irr]
show ?thesis using $p$ by simp
next
case False
note $I H=I H(2-)[$ OF False $]$
from False have (degree $p=0$ ) = False by auto
note res $=$ res[unfolded this if-False]
let ?rf $=$ rational-proper-factor $p$
show ?thesis

```
    proof (cases ?rf)
            case None
            with res have res: ?f c ( }p#\mathrm{ irr) ps=(d,qs) by auto
            from rational-proper-factor(1)[OF - None] False
            have irp: irreducibled p by auto
            note IH(1)[OF None res, unfolded atomize-imp imp-conjR, simplified]
            note 1 = conjunct1 [OF this, rule-format] conjunct2[OF this, rule-format]
            from irr irp show ?thesis by (auto intro:1 simp: ac-simps)
    next
        case (Some q)
        define pq where pq=p div q
        from Some res have res: ?f c irr ( }q#pq# #s)=(d,qs) unfolding pq-def
by auto
            from rational-proper-factor(2)[OF Some] have q dvd p by auto
            hence p:p=q*pq unfolding pq-def by auto
            from IH(2)[OF Some, folded pq-def, OF res irr] show ?thesis unfolding p
                by (auto simp: ac-simps)
    qed
    qed
qed
definition factorize-rat-poly-basic p=factorize-rat-poly-main 1[] [p]
lemma factorize-rat-poly-basic: assumes res: factorize-rat-poly-basic p = (c,qs)
    shows p= smult c (prod-list qs)
    \q.q\in set qs \Longrightarrow \mp@subsup{\mathrm{ irreducible }}{d}{}q
    using factorize-rat-poly-main[OF res[unfolded factorize-rat-poly-basic-def]] by
auto
We removed the factorize-rat-poly function from this theory, since the one in Berlekamp-Zassenhaus is easier to use and implements a more efficient algorithm.
```

end

## References

[1] D. E. Knuth. The Art of Computer Programming, Volume II: Seminumerical Algorithms, 2nd Edition. Addison-Wesley, 1981.
[2] D. Yun. On square-free decomposition algorithms. In Proc. the third ACM symposium on Symbolic and Algebraic Computation, pages 26-35, 1976.


[^0]:    *Supported by FWF (Austrian Science Fund) project Y757.

[^1]:    ${ }^{1}$ The Berlekamp-Zassenhaus AFP-entry was originally not present and at that time, this AFP-entry contained an implementation of Berlekamp-Zassenhaus as a non-certified function.

