A verified factorization algorithm for integer polynomials with polynomial complexity*

Jose Divasón Sebastiaan Joosten René Thiemann
Akihisa Yamada

March 19, 2025

Abstract

Short vectors in lattices and factors of integer polynomials are related. Each factor of an integer polynomial belongs to a certain lattice. When factoring polynomials, the condition that we are looking for an irreducible polynomial means that we must look for a *small* element in a lattice, which can be done by a basis reduction algorithm. In this development we formalize this connection and thereby one main application of the LLL basis reduction algorithm: an algorithm to factor square-free integer polynomials which runs in polynomial time. The work is based on our previous Berlekamp–Zassenhaus development, where the exponential reconstruction phase has been replaced by the polynomial-time basis reduction algorithm. Thanks to this formalization we found a serious flaw in a textbook.

Contents

*Supported by FWF (Austrian Science Fund) project Y757. Jose Divasón is partially funded by the Spanish project MTM2017-88804-P.

1 Introduction

In order to factor an integer polynomial f, we may assume a *modular* factorization of f into several monic factors u_i: $f \equiv \mathsf{lc}(f) \cdot \prod_i u_i$ modulo m where $m = p^l$ is some prime power for user-specified l. In Isabelle, we just reuse our verified modular factorization algorithm [1] to obtain the modular factorization of f.

We briefly explain how to compute non-trivial integer factors of f. The key is the following lemma [2, Lemma 16.20].

Lemma 1 ([2, Lemma 16.20]) *Let f, g, u be non-constant integer polynomials. Let u be monic. If u divides f modulo m, u divides g modulo m, and $\|f\|^{degree(g)} \cdot \|g\|^{degree(f)} < m$, then $h = gcd(f, g)$ is non-constant.*

Let f be a polynomial of degree n. Let u be any degree-d factor of f modulo m. Now assume that f is reducible, so $f = f_1 \cdot f_2$ where w.l.o.g., we assume that u divides f_1 modulo m and that $0 < degree(f_1) < n$. Let us further assume that a lattice $L_{u,k}$ encodes the set of all polynomials of degree below $d+k$ (as vectors of length $d+k$) which are divisible by u modulo m. Fix $k = n - d$. Then clearly, $f_1 \in L_{u,k}$.

In order to instantiate Lemma 1, it now suffices to take g as the polynomial corresponding to any short vector in $L_{u,k}$: u will divide g modulo m by definition of $L_{u,k}$ and moreover $degree(g) < n$. The short vector requirement will provide an upper bound to satisfy the assumption $\|f\|^{degree(g)} \cdot \|g\|^{degree(f)} < m$.

$$\|g\| \leq 2^{(n-1)/2} \cdot \|f_1\| \leq 2^{(n-1)/2} \cdot 2^{n-1}\|f\| = 2^{3(n-1)/2}\|f\| \tag{1}$$

$$\|f\|^{degree(g)} \cdot \|g\|^{degree(f)} \leq \|f\|^{n-1} \cdot (2^{3(n-1)/2}\|f\|)^n = \|f\|^{2n-1} \cdot 2^{3n(n-1)/2} \tag{2}$$

Here, the first inequality in (1) is the short vector approximation ($f_1 \in L_{u,k}$). The second inequality in (1) is Mignotte's factor bound (f_1 is a factor of f). Finally, (1) is used as an approximation of $\|g\|$ in (2).

Hence, if l is chosen large enough so that $m = p^l > \|f\|^{2n-1} \cdot 2^{3n(n-1)/2}$ then all preconditions of Lemma 1 are satisfied, and $h = gcd(f, g)$ will be a non-constant factor of f. Since the degree of h will be strictly less than n, h is also a proper factor of f, i.e., in particular $h \notin \{1, f\}$.

The textbook [2] also describes the general idea of the factorization algorithm based on the previous lemma in prose, and then presents an algorithm in pseudo-code which slightly extends the idea by directly splitting off *irreducible* factors [2, Algorithm 16.22]. We initially implemented and tried to verify this pseudo-code algorithm (see files `Factorization_Algorithm_16_22.thy` and `Modern_Computer_Algebra_Problem.thy`). After some work, we had only one remaining goal to prove: the content of the polynomial g corresponding to the short vector is not divisible by the chosen prime p. However, we were unable to figure out how to discharge this goal and then also started to search for inputs where the algorithm delivers wrong results. After a while we realized that Algorithm 16.22 indeed has a serious flaw as demonstrated in the upcoming example.

Example 1 *Consider the square-free and content-free polynomial* $f = (1 + x) \cdot (1 + x + x^3)$. *Then according to Algorithm 16.22 we determine*

- *the prime* $p = 2$
- *the exponent* $l = 61$
  (our new formalized algorithm uses a tighter bound which results in $l = 41$*)*
- *the leading coefficient* $b = 1$
- *the value* $B = 96$
- *the factorization mod* p *via* $h_1 = 1 + x$, $h_2 = 1 + x + x^3$

- *the factorization mod* p^l *via* $g_1 = 1 + x$, $g_2 = 1 + x + x^3$
- $f^* = f$, $T = \{1, 2\}$, $G = \emptyset$.
- *we enter the loop and in the first iteration choose*
- $u = 1 + x + x^3$, $d = 3$, $j = 4$
- *we consider the lattice generated by* $(1, 1, 0, 1)$, $(p^l, 0, 0, 0)$, $(0, p^l, 0, 0)$, $(0, 0, p^l, 0)$.
- *now we obtain a short vector in the lattice:* $g^* = (2, 2, 0, 2)$.
  Note that g^* *has not really been computed by Algorithm 16.10, but it satisfies the soundness criterion, i.e., it is a sufficiently short vector in the lattice.*

  To see this, note that a shortest vector in the lattice is $(1, 1, 0, 1)$.

  $$\|g^*\| = 2 \cdot \sqrt{3} \leq 2 \cdot \sqrt{2} \cdot \sqrt{3} = 2^{(j-1)/2} \cdot \|(1, 1, 0, 1)\|$$

  So g^* *has the required precision that was assumed by the short-vector calculation.*
- *the problem at this point is that* p *divides the content of* g^*. *Consequently, every polynomial divides* g^* *mod* p. *Thus in step 9 we compute* $S = T$, $h = 1$, *enter the then-branch and update* $T = \emptyset$, $G = G \cup \{1 + x + x^3\}$, $f^* = 1$, $b = 1$.
- *Then in step 10 we update* $G = \{1 + x + x^3, 1\}$ *and finally return that the factorization of* f *is* $(1 + x + x^3) \cdot 1$.

More details about the bug and some other wrong results presented in the book are shown in the file `Modern_Computer_Algebra_Problem.thy`.

Once we realized the problem, we derived another algorithm based on Lemma 1, which also runs in polynomial-time, and prove its soundness in Isabelle/HOL. The corresponding Isabelle statement is as follows:

Theorem 1 (LLL Factorization Algorithm)

> **assumes** *square_free* (f :: *int poly*)
> **and** *degree* $f \neq 0$
> **and** *LLL_factorization* $f = gs$
> **shows** $f =$ *prod_list* gs
> **and** $\forall g_i \in$ *set* gs. *irreducible* g_i

Finally, we also have been able to fix Algorithm 16.22 and provide a formal correctness proof of the the slightly modified version. It can be seen as an implementation of the pseudo-code factorization algorithm given by Lenstra, Lenstra, and Lovász [3].

2 Factor bound

This theory extends the work about factor bounds which was carried out in the Berlekamp-Zassenhaus development.

theory *Factor-Bound-2*
imports *Berlekamp-Zassenhaus.Factor-Bound*
LLL-Basis-Reduction.Norms
begin

lemma *norm-1-bound-mignotte*: $norm1\ f \leq 2\hat{}(degree\ f) * mahler\text{-}measure\ f$
⟨*proof*⟩

lemma *mahler-measure-l2norm*: $mahler\text{-}measure\ f \leq sqrt\ (of\text{-}int\ \|f\|^2)$
⟨*proof*⟩

lemma *sq-norm-factor-bound*:
fixes $f\ h :: int\ poly$
assumes *dvd*: $h\ dvd\ f$ **and** *f0*: $f \neq 0$
shows $\|h\|^2 \leq 2\ \hat{}\ (2 * degree\ h) * \|f\|^2$
⟨*proof*⟩

end

3 Executable dvdm operation

This theory contains some results about division of integer polynomials which are not part of Polynomial_Factorization.Dvd_Int_Poly.thy.

Essentially, we give an executable implementation of division modulo m.

theory *Missing-Dvd-Int-Poly*
imports
Berlekamp-Zassenhaus.Poly-Mod-Finite-Field
Berlekamp-Zassenhaus.Polynomial-Record-Based
Berlekamp-Zassenhaus.Hensel-Lifting
Subresultants.Subresultant
Perron-Frobenius.Cancel-Card-Constraint
begin

lemma *degree-div-mod-smult*:
fixes $g{::}int\ poly$
assumes *g*: $degree\ g < j$
and *r*: $degree\ r < d$
and *u*: $degree\ u = d$
and *g1*: $g = q * u + smult\ m\ r$
and *q*: $q \neq 0$ **and** *m-not0*: $m \neq 0$
shows $degree\ q < j - d$
⟨*proof*⟩

3.1 Uniqueness of division algorithm for polynomials

lemma *uniqueness-algorithm-division-poly*:
fixes f::'*a*::{*comm-ring*,*semiring-1-no-zero-divisors*} *poly*
assumes *f1*: $f = g * q1 + r1$
and *f2*: $f = g * q2 + r2$
and *g*: $g \neq 0$
and *r1*: $r1 = 0 \vee degree\ r1 < degree\ g$
and *r2*: $r2 = 0 \vee degree\ r2 < degree\ g$
shows $q1 = q2 \wedge r1 = r2$
⟨*proof*⟩

lemma *pdivmod-eq-pdivmod-monic*:
assumes *g*: *monic g*
shows *pdivmod f g* = *pdivmod-monic f g*
⟨*proof*⟩

context *poly-mod*
begin

definition *pdivmod2 f g* = (*if Mp g* = *0 then* (*0*, *f*)
else let ilc = *inverse-p m* ((*lead-coeff* (*Mp g*)));
h = *Polynomial.smult ilc* (*Mp g*); (*q*, *r*) = *pseudo-divmod* (*Mp f*) (*Mp h*)
in (*Polynomial.smult ilc q*, *r*))
end

context *poly-mod-prime-type*
begin

lemma *dvdm-iff-pdivmod0*:
assumes *f*: (*F* :: '*a mod-ring poly*) = *of-int-poly f*
and *g*: (*G* :: '*a mod-ring poly*) = *of-int-poly g*
shows *g dvdm f* = (*snd* (*pdivmod F G*) = *0*)
⟨*proof*⟩

lemma *of-int-poly-Mp-0*[*simp*]: (*of-int-poly* (*Mp a*) = (*0*:: '*a mod-ring poly*)) = (*Mp a* = *0*)
⟨*proof*⟩

lemma *uniqueness-algorithm-division-of-int-poly*:
assumes *g0*: *Mp g* ≠ *0*
and *f*: (*F* :: '*a mod-ring poly*) = *of-int-poly f*
and *g*: (*G* :: '*a mod-ring poly*) = *of-int-poly g*
and *F*: $F = G * Q + R$
and *R*: $R = 0 \vee degree\ R < degree\ G$
and *Mp-f*: $Mp\ f = Mp\ g * q + r$
and *r*: $r = 0 \vee degree\ r < degree\ (Mp\ g)$
shows $Q = of\text{-}int\text{-}poly\ q \wedge R = of\text{-}int\text{-}poly\ r$
⟨*proof*⟩

corollary *uniqueness-algorithm-division-to-int-poly*:
  assumes *g0*: *Mp g* $\neq$ *0*
  and *f*: (*F* :: *'a mod-ring poly*) = *of-int-poly f*
  and *g*: (*G* :: *'a mod-ring poly*) = *of-int-poly g*
  and *F*: *F* = *G* * *Q* + *R*
  and *R*: *R* = *0* ∨ *degree R* < *degree G*
  and *Mp-f*: *Mp f* = *Mp g* * *q* + *r*
  and *r*: *r* = *0* ∨ *degree r* < *degree* (*Mp g*)
  shows *Mp q* = *to-int-poly Q* ∧ *Mp r* = *to-int-poly R*
  ⟨*proof*⟩

lemma *uniqueness-algorithm-division-Mp-Rel*:
  assumes *monic-Mpg*: *monic* (*Mp g*)
    and *f*: (*F* :: *'a mod-ring poly*) = *of-int-poly f*
  and *g*: (*G* :: *'a mod-ring poly*) = *of-int-poly g*
  and *qr*: *pseudo-divmod* (*Mp f*) (*Mp g*) = (*q*,*r*)
  and *QR*: *pseudo-divmod F G* = (*Q*,*R*)
shows *MP-Rel q Q* ∧ *MP-Rel r R*
⟨*proof*⟩

definition *MP-Rel-Pair A B* ≡ (*let* (*a*,*b*) = *A*; (*c*,*d*) = *B in MP-Rel a c* ∧ *MP-Rel b d*)

lemma *pdivmod2-rel*[*transfer-rule*]:
  (*MP-Rel* ===> *MP-Rel* ===> *MP-Rel-Pair*) (*pdivmod2*) (*pdivmod*)
⟨*proof*⟩

3.2 Executable division operation modulo m for polynomials

lemma *dvdm-iff-Mp-pdivmod2*:
  shows *g dvdm f* = (*Mp* (*snd* (*pdivmod2 f g*)) = *0*)
⟨*proof*⟩

end

lemmas (**in** *poly-mod-prime*) *dvdm-pdivmod* = *poly-mod-prime-type.dvdm-iff-Mp-pdivmod2*
  [*unfolded poly-mod-type-simps*, *internalize-sort 'a* :: *prime-card*, *OF type-to-set*,
   unfolded remove-duplicate-premise, *cancel-type-definition*, *OF non-empty*]

lemma (**in** *poly-mod*) *dvdm-code*:
   g dvdm f = (*if prime m then Mp* (*snd* (*pdivmod2 f g*)) = *0*
    else Code.abort (*STR ''dvdm error*: *m is not a prime number''*) (λ -. *g dvdm f*))
  ⟨*proof*⟩

declare *poly-mod.pdivmod2-def*[*code*]
declare *poly-mod.dvdm-code*[*code*]

end

4 The LLL factorization algorithm

This theory contains an implementation of a polynomial time factorization algorithm. It first constructs a modular factorization. Afterwards it recursively invokes the LLL basis reduction algorithm on one lattice to either split a polynomial into two non-trivial factors, or to deduce irreducibility.

theory *LLL-Factorization-Impl*
  imports *LLL-Basis-Reduction.LLL-Certification*
    Factor-Bound-2
    Missing-Dvd-Int-Poly
    Berlekamp-Zassenhaus.Berlekamp-Zassenhaus
begin

hide-const (**open**) *up-ring.coeff up-ring.monom*
  Unique-Factorization.factors Divisibility.factors
  Unique-Factorization.factor Divisibility.factor
  Divisibility.prime

definition *factorization-lattice* **where** *factorization-lattice u k m* ≡
    map (λ*i*. *vec-of-poly-n* (*u* ∗ *monom 1 i*) (*degree u* + *k*)) [*k*>..*0*] @
    map (λ*i*. *vec-of-poly-n* (*monom m i*) (*degree u* + *k*)) [*degree u* >..*0*]

fun *min-degree-poly* :: *int poly* ⇒ *int poly* ⇒ *int poly*
  where *min-degree-poly a b* = (*if degree a* ≤ *degree b then a else b*)

fun *choose-u* :: *int poly list* ⇒ *int poly*
  where *choose-u* [] = *undefined*
  | *choose-u* [*gi*] = *gi*
  | *choose-u* (*gi* # *gj* # *gs*) = *min-degree-poly gi* (*choose-u* (*gj* # *gs*))

lemma *factorization-lattice-code*[*code*]: *factorization-lattice u k m* = (
  let n = *degree u in*
 map
  (λ*i*. *vec-of-poly-n* (*monom-mult i u*) (*n*+*k*)) [*k*>..*0*]
  @ *map* (λ*i*. *vec-of-poly-n* (*monom m i*) (*n*+*k*)) [*n*>..*0*]
) ⟨*proof*⟩

Optimization: directly try to minimize coefficients of polynomial *u*.

definition *LLL-short-polynomial* **where**
  LLL-short-polynomial pl n u = *poly-of-vec* (*short-vector-hybrid 2* (*factorization-lattice*
    (*poly-mod.inv-Mp pl* (*poly-mod.Mp pl u*)) (*n* − *degree u*) *pl*))

locale *LLL-implementation* =
  fixes *p pl* :: *int*

begin

function *LLL-many-reconstruction* **where**
LLL-many-reconstruction f us = (*let*
d = *degree f*;
d2 = *d div 2*;
f2-opt = *find-map-filter*
(λ *u*. *gcd f* (*LLL-short-polynomial pl* (*Suc d2*) *u*))
(λ *f2*. *let deg* = *degree f2 in deg* > *0* $\wedge$ *deg* < *d*)
(*filter* (λ *u*. *degree u* $\leq$ *d2*) *us*)
in case f2-opt of None $\Rightarrow$ [*f*]
| *Some f2* $\Rightarrow$ *let f1* = *f div f2*;
(*us1*, *us2*) = *List.partition* (λ *gi*. *poly-mod.dvdm p gi f1*) *us*
in LLL-many-reconstruction f1 us1 @ *LLL-many-reconstruction f2 us2*)
⟨*proof*⟩

termination
⟨*proof*⟩

function *LLL-reconstruction* **where**
LLL-reconstruction f us = (*let*
d = *degree f*;
u = *choose-u us*;
g = *LLL-short-polynomial pl d u*;
f2 = *gcd f g*;
deg = *degree f2*
in if deg = *0* $\vee$ *deg* $\geq$ *d then* [*f*]
else let f1 = *f div f2*;
(*us1*, *us2*) = *List.partition* (λ *gi*. *poly-mod.dvdm p gi f1*) *us*
in LLL-reconstruction f1 us1 @ *LLL-reconstruction f2 us2*)
⟨*proof*⟩

termination
⟨*proof*⟩
end

declare *LLL-implementation.LLL-reconstruction.simps*[*code*]
declare *LLL-implementation.LLL-many-reconstruction.simps*[*code*]

definition *LLL-factorization* :: *int poly* $\Rightarrow$ *int poly list* **where**
LLL-factorization f = (*let*
— find suitable prime
p = *suitable-prime-bz f*;
— compute finite field factorization
(-, *fs*) = *finite-field-factorization-int p f*;
— determine exponent l and B
n = *degree f*;
no = $\|f\|^2$;
B = *sqrt-int-ceiling* ($2\hat{}(5 * (n - 1) * (n - 1)) * no\hat{}(2 * (n - 1))$);

l = *find-exponent p B*;
— perform hensel lifting to lift factorization to mod p^l
us = *hensel-lifting p l f fs*;
— reconstruct integer factors via LLL algorithm
pl = p^l
in LLL-implementation.LLL-reconstruction p pl f us)

definition *LLL-many-factorization* :: *int poly* ⇒ *int poly list* **where**
LLL-many-factorization f = (*let*
— find suitable prime
p = *suitable-prime-bz f*;
— compute finite field factorization
(-, *fs*) = *finite-field-factorization-int p f*;
— determine exponent l and B
n = *degree f*;
no = $\|f\|^2$;
B = *sqrt-int-ceiling* (*2*^(*5* ∗ (*n div 2*) ∗ (*n div 2*)) ∗ *no*^(*2* ∗ (*n div 2*)));
l = *find-exponent p B*;
— perform hensel lifting to lift factorization to mod p^l
us = *hensel-lifting p l f fs*;
— reconstruct integer factors via LLL algorithm
pl = p^l
in LLL-implementation.LLL-many-reconstruction p pl f us)

end

5 Correctness of the LLL factorization algorithm

This theory connects short vectors of lattices and factors of polynomials. From this connection, we derive soundness of the lattice based factorization algorithm.

theory *LLL-Factorization*
imports
LLL-Factorization-Impl
Berlekamp-Zassenhaus.Factorize-Int-Poly
begin

5.1 Basic facts about the auxiliary functions

hide-const (**open**) *module.smult*

lemma *nth-factorization-lattice*:
fixes *u* **and** *d*
defines *n* ≡ *degree u*
assumes *i* < *n* + *d*
shows *factorization-lattice u d m* ! *i* =
vec-of-poly-n (*if i* < *d then u* ∗ *monom 1* (*d* − *Suc i*) *else monom m* (*n*+*d*−*Suc i*)) (*n*+*d*)

⟨*proof*⟩

lemma *length-factorization-lattice*[*simp*]:
shows *length* (*factorization-lattice u d m*) = *degree u* + *d*
⟨*proof*⟩

lemma *dim-factorization-lattice*:
assumes $x <$ *degree u* + *d*
shows *dim-vec* (*factorization-lattice u d m* ! *x*) = *degree u* + *d*
⟨*proof*⟩

lemma *dim-factorization-lattice-element*:
assumes $x \in$ *set* (*factorization-lattice u d m*) **shows** *dim-vec x* = *degree u* + *d*
⟨*proof*⟩

lemma *set-factorization-lattice-in-carrier*[*simp*]: *set* (*factorization-lattice u d m*) $\subseteq$ *carrier-vec* (*degree u* + *d*)
⟨*proof*⟩

lemma *choose-u-Cons*: *choose-u* (*x*#*xs*) =
(*if xs* = [] *then x else min-degree-poly x* (*choose-u xs*))
⟨*proof*⟩

lemma *choose-u-member*: *xs* $\neq$ [] $\Longrightarrow$ *choose-u xs* $\in$ *set xs*
⟨*proof*⟩

declare *choose-u.simps*[*simp del*]

5.2 Facts about Sylvester matrices and norms

lemma (**in** *LLL*) *lattice-is-span* [*simp*]: *lattice-of xs* = *span-list xs*
⟨*proof*⟩

lemma *sq-norm-row-sylvester-mat1*:
fixes *f g* :: *'a* :: *conjugatable-ring poly*
assumes *i*: $i <$ *degree g*
shows $\|$(*row* (*sylvester-mat f g*) *i*)$\|^2 = \|f\|^2$
⟨*proof*⟩

lemma *sq-norm-row-sylvester-mat2*:
fixes *f g* :: *'a* :: *conjugatable-ring poly*
assumes *i1*: *degree g* $\leq i$ **and** *i2*: $i <$ *degree f* + *degree g*
shows $\|$*row* (*sylvester-mat f g*) *i*$\|^2 = \|g\|^2$
⟨*proof*⟩

lemma *Hadamard's-inequality-int*:
fixes *A*::*int mat*
assumes *A*: $A \in$ *carrier-mat n n*

shows $|det\ A| \leq sqrt\ (of\text{-}int\ (prod\text{-}list\ (map\ sq\text{-}norm\ (rows\ A))))$
⟨*proof*⟩

lemma *resultant-le-prod-sq-norm*:
fixes $f\ g$::*int poly*
defines $n \equiv degree\ f$ **and** $k \equiv degree\ g$
shows $|resultant\ f\ g| \leq sqrt\ (of\text{-}int\ (\|f\|^2 \widehat{\ } k * \|g\|^2 \widehat{\ } n))$
⟨*proof*⟩

5.3 Proof of the key lemma 16.20

lemma *common-factor-via-short*:
fixes $f\ g\ u$ :: *int poly*
defines $n \equiv degree\ f$ **and** $k \equiv degree\ g$
assumes *n0*: $n > 0$ **and** *k0*: $k > 0$
and *monic*: *monic u* **and** *deg-u*: $degree\ u > 0$
and *uf*: *poly-mod.dvdm m u f* **and** *ug*: *poly-mod.dvdm m u g*
and *short*: $\|f\|^2 \widehat{\ } k * \|g\|^2 \widehat{\ } n < m^2$
and *m*: $m \geq 0$
shows $degree\ (gcd\ f\ g) > 0$
⟨*proof*⟩

5.4 Properties of the computed lattice and its connection with Sylvester matrices

lemma *factorization-lattice-as-sylvester*:
fixes p :: $'a$ :: *semidom poly*
assumes *dj*: $d \leq j$ **and** *d*: $degree\ p = d$
shows $mat\text{-}of\text{-}rows\ j\ (factorization\text{-}lattice\ p\ (j-d)\ m) = sylvester\text{-}mat\text{-}sub\ d\ (j-d)\ p\ [:m:]$
⟨*proof*⟩

context *inj-comm-semiring-hom* **begin**

lemma *map-poly-hom-mult-monom* [*hom-distribs*]:
$map\text{-}poly\ hom\ (p * monom\ a\ n) = map\text{-}poly\ hom\ p * monom\ (hom\ a)\ n$
⟨*proof*⟩

lemma *hom-vec-of-poly-n* [*hom-distribs*]:
$map\text{-}vec\ hom\ (vec\text{-}of\text{-}poly\text{-}n\ p\ n) = vec\text{-}of\text{-}poly\text{-}n\ (map\text{-}poly\ hom\ p)\ n$
⟨*proof*⟩

lemma *hom-factorization-lattice* [*hom-distribs*]:
shows $map\ (map\text{-}vec\ hom)\ (factorization\text{-}lattice\ u\ k\ m) = factorization\text{-}lattice\ (map\text{-}poly\ hom\ u)\ k\ (hom\ m)$
⟨*proof*⟩

end

5.5 Proving that *factorization-lattice* returns a basis of the lattice

context *LLL*
begin

sublocale *idom-vec n TYPE*(*int*)⟨*proof*⟩

lemma *upper-triangular-factorization-lattice*:
fixes $u :: 'a :: semidom\ poly$ **and** $d :: nat$
assumes d: $d \leq n$ **and** du: $d = degree\ u$
shows *upper-triangular* (*mat-of-rows n* (*factorization-lattice u* $(n-d)$ *k*))
(**is** *upper-triangular ?M*)
⟨*proof*⟩

lemma *factorization-lattice-diag-nonzero*:
fixes $u :: 'a :: semidom\ poly$ **and** d
assumes d: $d=degree\ u$
and dn: $d \leq n$
and u: $u \neq 0$
and $m0$: $k \neq 0$
and i: $i<n$
shows (*factorization-lattice u* $(n-d)$ *k*) ! *i* \$ $i \neq 0$
⟨*proof*⟩

corollary *factorization-lattice-diag-nonzero-RAT*: **fixes** d
assumes $d=degree\ u$
and $d \leq n$
and $u \neq 0$
and $k \neq 0$
and $i<n$
shows *RAT* (*factorization-lattice u* $(n-d)$ *k*) ! *i* \$ $i \neq 0$
⟨*proof*⟩

sublocale *gs*: *vec-space TYPE*(*rat*) *n*⟨*proof*⟩

lemma *lin-indpt-list-factorization-lattice*: **fixes** d
assumes d: $d = degree\ u$ **and** dn: $d \leq n$ **and** u: $u \neq 0$ **and** k: $k \neq 0$
shows *gs.lin-indpt-list* (*RAT* (*factorization-lattice u* $(n-d)$ *k*)) (**is** *gs.lin-indpt-list* (*RAT ?vs*))
⟨*proof*⟩

end

5.6 Being in the lattice is being a multiple modulo

lemma (**in** *semiring-hom*) *hom-poly-of-vec*: *map-poly hom* (*poly-of-vec v*) = *poly-of-vec* (*map-vec hom v*)
⟨*proof*⟩

abbreviation *of-int-vec* $\equiv$ *map-vec of-int*

context *LLL*
begin

lemma *lincomb-to-dvd-modulo*:
  fixes *u d*
  defines $d \equiv$ *degree u*
  assumes *d*: $d \leq n$
    and *lincomb*: *lincomb-list c* (*factorization-lattice u* $(n-d)$ *k*) = *g* (**is** *?l* = *?r*)
  shows *poly-mod.dvdm k u* (*poly-of-vec g*)
⟨*proof*⟩

lemma *dvd-modulo-to-lincomb*:
  fixes *u* :: *int poly* **and** *d*
  defines $d \equiv$ *degree u*
  assumes *d*: $d < n$
    and *dvd*: *poly-mod.dvdm k u* (*poly-of-vec g*)
    and *k-not0*: $k \neq 0$
    and *monic-u*: *monic u*
    and *dim-g*: *dim-vec g* = *n*
    and *deg-u*: *degree u* > *0*
  shows $\exists$ *c. lincomb-list c* (*factorization-lattice u* $(n-d)$ *k*) = *g*
⟨*proof*⟩

The factorization lattice precisely characterises the polynomials of a certain degree which divide u modulo M.

lemma *factorization-lattice*: **fixes** *M* **assumes**
  deg-u: *degree u* $\neq$ *0* **and** *M*: $M \neq 0$
shows *degree u* $\leq n \Longrightarrow n \neq 0 \Longrightarrow f \in$ *poly-of-vec* ‘ *lattice-of* (*factorization-lattice u* (*n* − *degree u*) *M*) $\Longrightarrow$
  degree f < *n* $\wedge$ *poly-mod.dvdm M u f*
  monic u $\Longrightarrow$ *degree u* < *n* $\Longrightarrow$
  degree f < *n* $\Longrightarrow$ *poly-mod.dvdm M u f* $\Longrightarrow f \in$ *poly-of-vec* ‘ *lattice-of* (*factorization-lattice u* (*n* − *degree u*) *M*)
⟨*proof*⟩
end

5.7 Soundness of the LLL factorization algorithm

lemma *LLL-short-polynomial*: **assumes** *deg-u-0*: *degree u* $\neq$ *0* **and** *deg-le*: *degree u* $\leq n$
  and *pl1*: *pl* > *1*
  and *monic*: *monic u*
shows *degree* (*LLL-short-polynomial pl n u*) < *n*
  and *LLL-short-polynomial pl n u* $\neq$ *0*
  and *poly-mod.dvdm pl u* (*LLL-short-polynomial pl n u*)

and *degree u* $<$ *n* $\Longrightarrow$ *f* $\neq$ *0* $\Longrightarrow$
poly-mod.dvdm pl u f $\Longrightarrow$ *degree f* $<$ *n* $\Longrightarrow$ $\|$*LLL-short-polynomial pl n u*$\|^2 \leq$
2 $\widehat{\ }$ (*n* $-$ *1*) $*$ $\|f\|^2$
⟨*proof*⟩

context *LLL-implementation*
begin

lemma *LLL-reconstruction*: **assumes** *LLL-reconstruction f us* = *fs*
and *degree f* $\neq$ *0*
and *poly-mod.unique-factorization-m pl f* (*lead-coeff f*, *mset us*)
and *f dvd F*
and $\bigwedge$ *ui*. *ui* $\in$ *set us* $\Longrightarrow$ *poly-mod.Mp pl ui* = *ui*
and *F0*: *F* $\neq$ *0*
and *cop*: *coprime* (*lead-coeff F*) *p*
and *sf*: *poly-mod.square-free-m p F*
and *pl1*: *pl* $>$ *1*
and *plp*: *pl* = *p*$\widehat{\ }$*l*
and *p*: *prime p*
and *large*: *2*$\widehat{\ }$(*5* $*$ (*degree F* $-$ *1*) $*$ (*degree F* $-$ *1*)) $*$ $\|F\|^2$$\widehat{\ }$(*2* $*$ (*degree F* $-$ *1*)) $<$ pl^2
shows *f* = *prod-list fs* $\wedge$ ($\forall$ *fi* $\in$ *set fs*. $irreducible_d$ *fi*)
⟨*proof*⟩

lemma *LLL-many-reconstruction*: **assumes** *LLL-many-reconstruction f us* = *fs*
and *degree f* $\neq$ *0*
and *poly-mod.unique-factorization-m pl f* (*lead-coeff f*, *mset us*)
and *f dvd F*
and $\bigwedge$ *ui*. *ui* $\in$ *set us* $\Longrightarrow$ *poly-mod.Mp pl ui* = *ui*
and *F0*: *F* $\neq$ *0*
and *cop*: *coprime* (*lead-coeff F*) *p*
and *sf*: *poly-mod.square-free-m p F*
and *pl1*: *pl* $>$ *1*
and *plp*: *pl* = *p*$\widehat{\ }$*l*
and *p*: *prime p*
and *large*: *2*$\widehat{\ }$(*5* $*$ (*degree F div 2*) $*$ (*degree F div 2*)) $*$ $\|F\|^2$$\widehat{\ }$(*2* $*$ (*degree F div 2*)) $<$ pl^2
shows *f* = *prod-list fs* $\wedge$ ($\forall$ *fi* $\in$ *set fs*. $irreducible_d$ *fi*)
⟨*proof*⟩

end

lemma *LLL-factorization*:
assumes *res*: *LLL-factorization f* = *gs*
and *sff*: *square-free f*
and *deg*: *degree f* $\neq$ *0*
shows *f* = *prod-list gs* $\wedge$ ($\forall$ *g*$\in$*set gs*. $irreducible_d$ *g*)
⟨*proof*⟩

lemma *LLL-many-factorization*:
  assumes *res*: *LLL-many-factorization f* = *gs*
  and *sff*: *square-free f*
  and *deg*: *degree f* ≠ *0*
  shows *f* = *prod-list gs* ∧ (∀ *g*∈*set gs*. *irreducible*$_d$ *g*)
⟨*proof*⟩

lift-definition *one-lattice-LLL-factorization* :: *int-poly-factorization-algorithm*
  is *LLL-factorization* ⟨*proof*⟩

lift-definition *many-lattice-LLL-factorization* :: *int-poly-factorization-algorithm*
  is *LLL-many-factorization* ⟨*proof*⟩

lemma *LLL-factorization-primitive*: **assumes** *LLL-factorization f* = *fs*
  square-free f
  0 < *degree f*
  primitive f
shows *f* = *prod-list fs* ∧ (∀ *fi*∈*set fs*. *irreducible fi* ∧ *0* < *degree fi* ∧ *primitive fi*)
  ⟨*proof*⟩

thm *factorize-int-poly*[*of one-lattice-LLL-factorization*]
thm *factorize-int-poly*[*of many-lattice-LLL-factorization*]
end

6 Calculating All Possible Sums of Sub-Multisets

theory *Sub-Sums*
  imports
    Main
    HOL−Library.Multiset
begin

fun *sub-mset-sums* :: *'a* :: *comm-monoid-add list* ⇒ *'a set* **where**
  sub-mset-sums [] = {*0*}
| *sub-mset-sums* (*x* # *xs*) = (*let S* = *sub-mset-sums xs in S* ∪ ((+) *x*) ‘ *S*)

lemma *subset-add-mset*: *ys* ⊆# *add-mset x zs* ⟷ (*ys* ⊆# *zs* ∨ (∃ *xs*. *xs* ⊆# *zs*
∧ *ys* = *add-mset x xs*))
  (**is** *?l* = *?r*)
⟨*proof*⟩

lemma *sub-mset-sums*[*simp*]: *sub-mset-sums xs* = *sum-mset* ‘ { *ys*. *ys* ⊆# *mset xs*
}
⟨*proof*⟩

end

7 Implementation and soundness of a modified version of Algorithm 16.22

Algorithm 16.22 is quite similar to the LLL factorization algorithm that was verified in the previous section. Its main difference is that it has an inner loop where each inner loop iteration has one invocation of the LLL basis reduction algorithm. Algorithm 16.22 of the textbook is therefore closer to the factorization algorithm as it is described by Lenstra, Lenstra, and Lovász [3], which also uses an inner loop.

The advantage of the inner loop is that it can find factors earlier, and then small lattices suffice where without the inner loop one invokes the basis reduction algorithm on a large lattice. The disadvantage of the inner loop is that if the input is irreducible, then one cannot find any factor early, so that all but the last iteration have been useless: only the last iteration will prove irreducibility.

We will describe the modifications w.r.t. the original Algorithm 16.22 of the textbook later in this theory.

theory *Factorization-Algorithm-16-22*
  imports
    LLL-Factorization
    Sub-Sums
begin

7.1 Previous lemmas obtained using local type definitions

context *poly-mod-prime-type*
begin

lemma *irreducible-m-dvdm-prod-list-connect*:
  assumes *irr*: *irreducible-m a*
  and *dvd*: *a dvdm* (*prod-list xs*)
shows ∃ *b* ∈ *set xs*. *a dvdm b*
⟨*proof*⟩

end

lemma (**in** *poly-mod-prime*) *irreducible-m-dvdm-prod-list*:
  assumes *irr*: *irreducible-m a*
  and *dvd*: *a dvdm* (*prod-list xs*)
  shows ∃ *b* ∈ *set xs*. *a dvdm b*
  ⟨*proof*⟩

7.2 The modified version of Algorithm 16.22

definition *B2-LLL* :: *int poly* ⇒ *int* **where**
  B2-LLL f = *2* ^ (*2* ∗ *degree f*) ∗ $\|f\|^2$

hide-const (**open**) *factors*
hide-const (**open**) *factors*
hide-const (**open**) *factor*
hide-const (**open**) *factor*

context
  fixes p :: *int* **and** l :: *nat*
begin

context
  fixes *gs* :: *int poly list*
    and f :: *int poly*
    and u :: *int poly*
    and *Degs* :: *nat set*
begin

This is the critical inner loop.

In the textbook there is a bug, namely that the filter is applied to g' and not to the primitive part of g'. (Problems occur if the content of g' is divisible by p.) We have fixed this problem in the obvious way.

However, there also is a second problem, namely it is only guaranteed that g' is divisible by u modulo p^l. However, for soundness we need to know that then also the primitive part of g' is divisible by u modulo p^l. This is not necessary true, e.g., if $g' = p^l$, then the primitive part is 1 which is not divisible by u modulo p^l. It is open, whether such a large g' can actually occur. Therefore, the current fix is to manually test whether the leading coefficient of g' is strictly smaller than p^l.

With these two modifications, Algorithm 16.22 will become sound as proven below.

definition *LLL-reconstruction-inner j* ≡
  let $j' = j - 1$ *in*
  — optimization: check whether degree j' is possible
  if $j' \notin$ *Degs then None else*
  — short vector computation
  let
    $ll = (let\ n = sqrt\text{-}int\text{-}ceiling\ (\|f\|^2\ \hat{}\ (2 * j') * 2\ \hat{}\ (5 * j' * j'));$
    $ll' = find\text{-}exponent\ p\ n\ in\ if\ ll' < l\ then\ ll'\ else\ l);$
  — optimization: dynamically adjust the modulus
    $pl = p\hat{}ll;$
    $g' = LLL\text{-}short\text{-}polynomial\ pl\ j\ u$
  — fix: forbid multiples of p^l as short vector, unclear whether this is really required
  in if abs (*lead-coeff* g') $\geq$ *pl then None else*
  let ppg = *primitive-part* g'
  in
  — slight deviation from textbook: we check divisibility instead of norm-inequality
  case div-int-poly f ppg of Some f' ⇒

— fix: consider modular factors of ppg and not of g'
Some (*filter* (λ*gi*. $\neg$ *poly-mod.dvdm p gi ppg*) *gs*, *lead-coeff f'*, *f'*, *ppg*)
| *None* $\Rightarrow$ *None*

function *LLL-reconstruction-inner-loop* **where**
LLL-reconstruction-inner-loop j =
(*if j* > *degree f then* ([],*1*,*1*,*f*)
else case LLL-reconstruction-inner j
of Some tuple $\Rightarrow$ *tuple*
| *None* $\Rightarrow$ *LLL-reconstruction-inner-loop* (*j*+*1*))
⟨*proof*⟩
termination ⟨*proof*⟩

end

partial-function (*tailrec*) *LLL-reconstruction''* **where** [*code*]:
LLL-reconstruction'' gs b f factors =
(*if gs* = [] *then factors*
else
let u = *choose-u gs*;
d = *degree u*;
gs' = *remove1 u gs*;
degs = *map degree gs'*;
Degs = ((+) *d*) ' *sub-mset-sums degs*;
(*gs'*, *b'*, *f'*, *factor*) = *LLL-reconstruction-inner-loop gs f u Degs* (*d*+*1*)
in LLL-reconstruction'' gs' b' f' (*factor*#*factors*)
)

definition *reconstruction-of-algorithm-16-22 gs f* ≡
let G = [];
b = *lead-coeff f*
in LLL-reconstruction'' gs b f G

end

definition *factorization-algorithm-16-22* :: *int poly* $\Rightarrow$ *int poly list* **where**
factorization-algorithm-16-22 f = (*let*
— find suitable prime
p = *suitable-prime-bz f*;
— compute finite field factorization
(-, *fs*) = *finite-field-factorization-int p f*;
— determine l and B
n = *degree f*;
— bound improved according to textbook, which uses $no = (n + 1) * (max - normf)^2$
no = $\|f\|^2$;
— possible improvement: $B = sqrt(2^{5*n*(n-1)} * no^{2*n-1}$, cf. *LLL-factorization*

B = *sqrt-int-ceiling* $(2 \mathbin{\hat{}} (5 * n * n) * no \mathbin{\hat{}} (2 * n))$;
l = *find-exponent* p B;
— perform hensel lifting to lift factorization to mod p^l
vs = *hensel-lifting* p l f *fs*
— reconstruct integer factors
in reconstruction-of-algorithm-16-22 p l *vs* f)

7.3 Soundness proof

7.3.1 Starting the proof

Key lemma to show that forbidding values of p^l or larger suffices to find correct factors.

lemma (**in** *poly-mod-prime*) *Mp-smult-p-removal*: *poly-mod.Mp* $(p * p \mathbin{\hat{}} k)$ (*smult* p f) = 0 $\Longrightarrow$ *poly-mod.Mp* $(p\hat{}k)$ f = 0
⟨*proof*⟩

lemma (**in** *poly-mod-prime*) *eq-m-smult-p-removal*: *poly-mod.eq-m* $(p * p \mathbin{\hat{}} k)$ (*smult* p f) (*smult* p g)
$\Longrightarrow$ *poly-mod.eq-m* $(p\hat{}k)$ f g ⟨*proof*⟩

lemma *content-le-lead-coeff*: *abs* (*content* (f :: *int poly*)) $\leq$ *abs* (*lead-coeff* f)
⟨*proof*⟩

lemma *poly-mod-dvd-drop-smult*: **assumes** *u*: *monic u* **and** *p*: *prime p* **and** *c*: $c \neq 0$ $|c| < p\hat{}l$
and *dvd*: *poly-mod.dvdm* $(p\hat{}l)$ u (*smult* c f)
shows *poly-mod.dvdm* p u f
⟨*proof*⟩

context
fixes p :: *int*
and F :: *int poly*
and N :: *nat*
and l :: *nat*
defines [*simp*]: $N \equiv$ *degree* F
assumes *p*: *prime p*
and *N0*: $N > 0$
and *bound-l*: $2 \mathbin{\hat{}} N^2 * \mathit{B2\text{-}LLL}\ F \mathbin{\hat{}} (2 * N) \leq (p\hat{}l)^2$
begin

private lemma *F0*: $F \neq 0$ ⟨*proof*⟩ **lemma** *p1*: $p > 1$ ⟨*proof*⟩

interpretation *p*: *poly-mod-prime p* ⟨*proof*⟩

interpretation *pl*: *poly-mod* $p\hat{}l$⟨*proof*⟩

lemma *B2-2*: $2 \leq \mathit{B2\text{-}LLL}\ F$
⟨*proof*⟩

lemma *l-gt-0*: *l* > *0*
⟨*proof*⟩

lemma *l0*: *l* ≠ *0* ⟨*proof*⟩

lemma *pl-not0*: *p* ^ *l* ≠ *0* ⟨*proof*⟩

interpretation *pl*: *poly-mod-2 p^l*
⟨*proof*⟩ **lemmas** *pl-dvdm-imp-p-dvdm* = *p.pl-dvdm-imp-p-dvdm*[*OF l0*]

lemma *p-Mp-pl-Mp*[*simp*]: *p.Mp* (*pl.Mp k*) = *p.Mp k*
⟨*proof*⟩

context
fixes *u* :: *int poly*
and *d* **and** *f* **and** *n*
and *gs* :: *int poly list*
and *Degs* :: *nat set*
defines [*simp*]: *d* ≡ *degree u*
assumes *d0*: *d* > *0*
and *u*: *monic u*
and *irred-u*: *p.irreducible-m u*
and *u-f*: *p.dvdm u f*
and *f-dvd-F*: *f dvd F*
and [*simp*]: *n* == *degree f*
and *f-gs*: *pl.unique-factorization-m f* (*lead-coeff f*, *mset gs*)
and *cop*: *coprime* (*lead-coeff f*) *p*
and *sf*: *p.square-free-m f*
and *sf-F*: *square-free f*
and *u-gs*: *u* ∈ *set gs*
and *norm-gs*: *map pl.Mp gs* = *gs*
and *Degs*: ⋀ *factor*. *factor dvd f* ⟹ *p.dvdm u factor* ⟹ *degree factor* ∈ *Degs*
begin
interpretation *pl*: *poly-mod-2 p^l* ⟨*proof*⟩ **lemma** *f0*: *f* ≠ *0* ⟨*proof*⟩ **lemma** *Mpf0*: *pl.Mp f* ≠ *0*
⟨*proof*⟩ **lemma** *pMpf0*: *p.Mp f* ≠ *0*
⟨*proof*⟩ **lemma** *dn*: *d* ≤ *n* ⟨*proof*⟩ **lemma** *n0*: *n* > *0* ⟨*proof*⟩ **lemma** *B2-0*[*intro*!]: *B2-LLL F* > *0* ⟨*proof*⟩ **lemma** *deg-u*: *degree u* > *0* ⟨*proof*⟩ **lemma** *n-le-N*: *n*≤*N*
⟨*proof*⟩

lemma *dvdm-power*: **assumes** *g dvd f*
shows *p.dvdm u g* ⟷ *pl.dvdm u g*
⟨*proof*⟩ **lemma** *uf*: *pl.dvdm u f* ⟨*proof*⟩

lemma *exists-reconstruction*: ∃ *h0*. *irreducible*$_d$ *h0* ∧ *p.dvdm u h0* ∧ *h0 dvd f*
⟨*proof*⟩

lemma *factor-dvd-f-0*: **assumes** *factor dvd f*
  shows *pl.Mp factor* $\neq$ *0*
⟨*proof*⟩

lemma *degree-factor-ge-degree-u*:
  assumes *u-dvdm-factor*: *p.dvdm u factor*
    and *factor-dvd*: *factor dvd f* **shows** *degree u* $\leq$ *degree factor*
⟨*proof*⟩

7.3.2 Inner loop

context
  fixes j' :: *nat*
  assumes *dj'*: $d \leq j'$
    and *j'n*: $j' < n$
    and *deg*: $\bigwedge$*factor*. *p.dvdm u factor* $\Longrightarrow$ *factor dvd f* $\Longrightarrow$ *degree factor* $\geq j'$
begin

private abbreviation (*input*) $j \equiv$ *Suc* j'

private lemma *jn*: $j \leq n$ ⟨*proof*⟩ **lemma** *factor-irreducible*$_d$*I*: **assumes** *hf*: *h dvd f*
  and *puh*: *p.dvdm u h*
  and *degh*: *degree h* > 0
  and *degh-j*: *degree h* $\leq j'$
shows *irreducible*$_d$ *h*
⟨*proof*⟩ **definition** *ll* = (*let n* = *sqrt-int-ceiling* ($\|f\|^2$ ^ ($2 * j'$) $*$ 2 ^ ($5 * j' * j'$));
    ll' = *find-exponent p n in if ll'* $<$ *l then ll' else l*)

lemma *ll*: *ll* $\leq$ *l* ⟨*proof*⟩

lemma *ll0*: *ll* $\neq$ *0* ⟨*proof*⟩

lemma *pll1*: *p*^*ll* $>$ *1* ⟨*proof*⟩

interpretation *pll*: *poly-mod-2 p*^*ll*
  ⟨*proof*⟩

lemma *pll0*: *p*^*ll* $\neq$ *0* ⟨*proof*⟩

lemma *dvdm-l-ll*: **assumes** *pl.dvdm a b*
  shows *pll.dvdm a b*
⟨*proof*⟩ **definition** $g \equiv$ *LLL-short-polynomial* (*p*^*ll*) *j u*

lemma *deg-g-j*: *degree g* $< j$
    and *g0*: $g \neq 0$
    and *ug* :*pll.dvdm u g*
    and *short-g*: $h \neq 0$ $\Longrightarrow$ *pll.dvdm u h* $\Longrightarrow$ *degree h* $\leq j'$ $\Longrightarrow$ $\|g\|^2 \leq 2$ ^ j' $*$

$\|h\|^2$
⟨*proof*⟩

lemma *LLL-reconstruction-inner-simps*: *LLL-reconstruction-inner p l gs f u Degs j*
= (*if* $j' \notin$ *Degs then None else if p* ^ *ll* $\leq$ |*lead-coeff g*| *then None*
else case div-int-poly f (*primitive-part g*) *of None* ⇒ *None*
| *Some* f' ⇒ *Some* ([*gi*←*gs* . ¬ *p.dvdm gi* (*primitive-part g*)], *lead-coeff* f', f', *primitive-part g*))
⟨*proof*⟩

lemma *LLL-reconstruction-inner-complete*:
assumes *ret*: *LLL-reconstruction-inner p l gs f u Degs j* = *None*
shows ⋀*factor*. *p.dvdm u factor* ⟹ *factor dvd f* ⟹ *degree factor* ≥ *j*
⟨*proof*⟩

lemma *LLL-reconstruction-inner-sound*:
assumes *ret*: *LLL-reconstruction-inner p l gs f u Degs j* = *Some* (gs',b',f',h)
shows $f = f' * h$ (**is** *?g1*)
and *irreducible*$_d$ *h* (**is** *?g2*)
and b' = *lead-coeff* f' (**is** *?g3*)
and *pl.unique-factorization-m* f' (*lead-coeff* f', *mset* gs') (**is** *?g4*)
and *p.dvdm u h* (**is** *?g5*)
and *degree h* = j' (**is** *?g6*)
and *length* gs' < *length gs* (**is** *?g7*)
and *set* gs' ⊆ *set gs* (**is** *?g8*)
and $gs' \neq$ [] (**is** *?g9*)
⟨*proof*⟩
end

interpretation *LLL d* ⟨*proof*⟩

lemma *LLL-reconstruction-inner-None-upt-j'*:
assumes *ij*: ∀ *i*∈{*d*+*1*..*j*}. *LLL-reconstruction-inner p l gs f u Degs i* = *None*
and *dj*: *d*<*j* **and** *j*≤*n*
shows ⋀*factor*. *p.dvdm u factor* ⟹ *factor dvd f* ⟹ *degree factor* ≥ *j*
⟨*proof*⟩

corollary *LLL-reconstruction-inner-None-upt-j*:
assumes *ij*: ∀ *i*∈{*d*+*1*..*j*}. *LLL-reconstruction-inner p l gs f u Degs i* = *None*
and *dj*: *d*≤*j* **and** *jn*: *j*≤*n*
shows ⋀*factor*. *p.dvdm u factor* ⟹ *factor dvd f* ⟹ *degree factor* ≥ *j*
⟨*proof*⟩

lemma *LLL-reconstruction-inner-all-None-imp-irreducible*:
assumes *i*: ∀ *i*∈{*d*+*1*..*n*}. *LLL-reconstruction-inner p l gs f u Degs i* = *None*
shows *irreducible*$_d$ *f*
⟨*proof*⟩

lemma *irreducible-imp-LLL-reconstruction-inner-all-None*:
  assumes *irr-f*: $irreducible_d\ f$
  shows $\forall\, i\in\{d+1..n\}.$ *LLL-reconstruction-inner p l gs f u Degs i = None*
⟨*proof*⟩

lemma *LLL-reconstruction-inner-all-None*:
  assumes *i*: $\forall\, i\in\{d+1..n\}.$ *LLL-reconstruction-inner p l gs f u Degs i = None*
  and *dj*: $d<j$
shows *LLL-reconstruction-inner-loop p l gs f u Degs j* = ([],*1*,*1*,*f*)
  ⟨*proof*⟩

corollary *irreducible-imp-LLL-reconstruction-inner-loop-f*:
  assumes *irr-f*: $irreducible_d\ f$ **and** *dj*: $d<j$
shows *LLL-reconstruction-inner-loop p l gs f u Degs j* = ([],*1*,*1*,*f*)
  ⟨*proof*⟩

lemma *exists-index-LLL-reconstruction-inner-Some*:
 assumes *inner-loop*: *LLL-reconstruction-inner-loop p l gs f u Degs j* = $(gs',b',f',factor)$
   and *i*: $\forall\, i\in\{d+1..<j\}.$ *LLL-reconstruction-inner p l gs f u Degs i = None*
   and *dj*: $d<j$ **and** *jn*: $j\le n$ **and** *f*: $\neg\ irreducible_d\ f$
  shows $\exists\, j'.\ j \le j' \wedge j'\le n \wedge d<j'$
   $\wedge$ (*LLL-reconstruction-inner p l gs f u Degs* j' = *Some* $(gs', b', f', factor)$)
   $\wedge$ ($\forall\, i\in\{d+1..<j'\}.$ *LLL-reconstruction-inner p l gs f u Degs i = None*)
  ⟨*proof*⟩

lemma *unique-factorization-m-1*: *pl.unique-factorization-m 1* (*1*, {#})
⟨*proof*⟩

lemma *LLL-reconstruction-inner-loop-j-le-n*:
  assumes *ret*: *LLL-reconstruction-inner-loop p l gs f u Degs j* = $(gs',b',f',factor)$
   and *ij*: $\forall\, i\in\{d+1..<j\}.$ *LLL-reconstruction-inner p l gs f u Degs i = None*
   and *n*: n = *degree f*
   and *jn*: $j \le n$
   and *dj*: $d < j$
  shows $f = f' * factor$ (**is** *?g1*)
   and $irreducible_d$ *factor* (**is** *?g2*)
   and b' = *lead-coeff* f' (**is** *?g3*)
   and *pl.unique-factorization-m* f' (b', *mset* gs') (**is** *?g4*)
   and *p.dvdm u factor* (**is** *?g5*)
   and $gs \neq [] \longrightarrow$ *length* gs' < *length gs* (**is** *?g6*)
   and *factor dvd f* (**is** *?g7*)
   and f' *dvd f* (**is** *?g8*)
   and *set* $gs' \subseteq$ *set gs* (**is** *?g9*)
   and $gs' = [] \longrightarrow f' = 1$ (**is** *?g10*)
  ⟨*proof*⟩

lemma *LLL-reconstruction-inner-loop-j-ge-n*:
  assumes *ret*: *LLL-reconstruction-inner-loop p l gs f u Degs j* = $(gs',b',f',factor)$

and *ij*: $\forall\, i{\in}\{d{+}1..n\}$. *LLL-reconstruction-inner p l gs f u Degs i* = *None*
and *dj*: $d < j$
and *jn*: $j{>}n$
shows $f = f' * factor$ (**is** *?g1*)
and $irreducible_d$ *factor* (**is** *?g2*)
and $b' = lead\text{-}coeff\ f'$ (**is** *?g3*)
and *pl.unique-factorization-m* f' (b', *mset* gs') (**is** *?g4*)
and *p.dvdm u factor* (**is** *?g5*)
and $gs \neq [] \longrightarrow length\ gs' < length\ gs$ (**is** *?g6*)
and *factor dvd f* (**is** *?g7*)
and f' *dvd f* (**is** *?g8*)
and *set* $gs' \subseteq$ *set gs* (**is** *?g9*)
and $f' = 1$ (**is** *?g10*)
⟨*proof*⟩

lemma *LLL-reconstruction-inner-loop*:
assumes *ret*: *LLL-reconstruction-inner-loop p l gs f u Degs j* = $(gs',b',f',factor)$
and *ij*: $\forall\, i{\in}\{d{+}1..{<}j\}$. *LLL-reconstruction-inner p l gs f u Degs i* = *None*
and *n*: $n = degree\ f$
and *dj*: $d < j$
shows $f = f' * factor$ (**is** *?g1*)
and $irreducible_d$ *factor* (**is** *?g2*)
and $b' = lead\text{-}coeff\ f'$ (**is** *?g3*)
and *pl.unique-factorization-m* f' (b', *mset* gs') (**is** *?g4*)
and *p.dvdm u factor* (**is** *?g5*)
and $gs \neq [] \longrightarrow length\ gs' < length\ gs$ (**is** *?g6*)
and *factor dvd f* (**is** *?g7*)
and f' *dvd f* (**is** *?g8*)
and *set* $gs' \subseteq$ *set gs* (**is** *?g9*)
and $gs' = [] \longrightarrow f' = 1$ (**is** *?g10*)
⟨*proof*⟩
end

7.3.3 Outer loop

lemma *LLL-reconstruction''*:
assumes *1*: *LLL-reconstruction''* $p\ l\ gs\ b\ f\ G = G'$
and *irreducible-G*: $\bigwedge factor.\ factor \in set\ G \Longrightarrow irreducible_d\ factor$
and *3*: $F = f * prod\text{-}list\ G$
and *4*: *pl.unique-factorization-m f* (*lead-coeff f*, *mset gs*)
and *5*: $gs \neq []$
and *6*: $\bigwedge\ gi.\ gi \in set\ gs \Longrightarrow pl.Mp\ gi = gi$
and *7*: $\bigwedge\ gi.\ gi \in set\ gs \Longrightarrow p.irreducible_d\text{-}m\ gi$
and *8*: *p.square-free-m f*
and *9*: *coprime* (*lead-coeff f*) *p*
and *sf-F*: *square-free F*
shows $(\forall\, g \in set\ G'.\ irreducible_d\ g) \wedge F = prod\text{-}list\ G'$
⟨*proof*⟩

context
  fixes *gs* :: *int poly list*
  assumes *gs-hen*: *berlekamp-hensel p l F* = *gs*
   and *cop*: *coprime* (*lead-coeff F*) *p*
   and *sf*: *poly-mod.square-free-m p F*
   and *sf-F*: *square-free F*
begin

lemma *gs-not-empty*: *gs* ≠ []
⟨*proof*⟩

lemma *reconstruction-of-algorithm-16-22*:
  assumes *1*: *reconstruction-of-algorithm-16-22 p l gs F* = *G*
  shows (∀ *g*∈*set G*. $irreducible_d$ *g*) ∧ *F* = *prod-list G*
⟨*proof*⟩
end
end

7.3.4 Final statement

lemma *factorization-algorithm-16-22*:
  assumes *res*: *factorization-algorithm-16-22 f* = *G*
  and *sff*: *square-free f*
  and *deg*: *degree f* > *0*
  shows (∀ *g*∈*set G*. $irreducible_d$ *g*) ∧ *f* = *prod-list G*
⟨*proof*⟩

lift-definition *increasing-lattices-LLL-factorization* :: *int-poly-factorization-algorithm*
  is *factorization-algorithm-16-22* ⟨*proof*⟩

thm *factorize-int-poly*[*of increasing-lattices-LLL-factorization*]

end

8 Mistakes in the textbook Modern Computer Algebra (2nd edition)

theory *Modern-Computer-Algebra-Problem*
  imports *Factorization-Algorithm-16-22*
begin

fun *max-degree-poly* :: *int poly* ⇒ *int poly* ⇒ *int poly*
  where *max-degree-poly a b* = (*if degree a* ≥ *degree b then a else b*)

fun *choose-u* :: *int poly list* ⇒ *int poly*
   where *choose-u* [] = *undefined*
 | *choose-u* [*gi*] = *gi*
 | *choose-u* (*gi* # *gj* # *gs*) = *max-degree-poly gi* (*choose-u* (*gj* # *gs*))

8.1 A real problem of Algorithm 16.22

Bogus example for Modern Computer Algebra (2nd edition), Algorithm 16.22, step 9: After having detected the factor [:*1*, *1*, *0*, *1*:], the remaining polynomial f^* will be 1, and the remaining list of modular factors will be empty.

lemma *let f* = [:*1*,*1*:] ∗ [:*1*,*1*,*0*,*1*:];
  p = *suitable-prime-bz f*;
  b = *lead-coeff f*;
  A = *linf-norm-poly f*; *n* = *degree f*; *B* = *sqrt-int-ceiling* (*n*+*1*) ∗ *2*^*n* ∗ *A*;
  Bnd = *2*^(*n*^*2 div 2*) ∗ *B*^(*2*∗*n*); *l* = *log-ceiling p Bnd*;
  (-, *fs*) = *finite-field-factorization-int p f*;
  gs = *hensel-lifting p l f fs*;
  u = *choose-u gs*;
  d = *degree u*;
  g-star = [:*2*,*2*,*0*,*2* :: *int* :];
  (*gs'*,*hs'*) = *List.partition* (λ*gi. poly-mod.dvdm p gi g-star*) *gs*;
  h-star = *smult b* (*prod-list hs'*);
  f-star = *primitive-part h-star*
 in (*hs'* = [] ∧ *f-star* = *1*) ⟨*proof*⟩

8.2 Another potential problem of Algorithm 16.22

Suppose that g^* is p^l. (It is is not yet clear whether lattices exists where this g^* is short enough). Then $pp(g^*) = 1$ is detected as *irreducible* factor and the algorithm stops.

definition *input-poly* = [: *1*,*0*,*0*,*0*,*1*,*1*,*0*,*0*,*1*,*0*,*1*,*0*,*1* :: *int* :]

For *input-poly* the factorization will result in a lattice where each initial basis element has a Euclidean norm of at least p^l (since the input polynomial u has a norm larger than p^l.) So, just from the norm of the basis one cannot infer that the lattice contains small vectors.

lemma *let f* = *input-poly*;
  p = *suitable-prime-bz f*;
  b = *lead-coeff f*;
  A = *linf-norm-poly f*; *n* = *degree f*; *B* = *sqrt-int-ceiling* (*n*+*1*) ∗ *2*^*n* ∗ *A*;
  Bnd = *2*^(*n*^*2 div 2*) ∗ *B*^(*2*∗*n*); *l* = *log-ceiling p Bnd*;
  (-, *fs*) = *finite-field-factorization-int p f*;
  gs = *hensel-lifting p l f fs*;
  u = *choose-u gs*;
  pl = *p*^*l*;
  pl2 = *pl div 2*;
  u' = *poly-mod.inv-Mp2 pl pl2* (*poly-mod.Mp pl* (*smult b u*))
 in sqrt-int-floor (*sq-norm u'*) > *pl* ⟨*proof*⟩

The following calculation will show that the norm of g^* is not that much shorter than p^l which is an indication that it is not obvious that in general p^l cannot be chosen as short polynomial.

definition *compute-norms* = (*let* *f* = *input-poly*;
  p = *suitable-prime-bz* *f*;
  b = *lead-coeff* *f*;
  A = *linf-norm-poly* *f*; *n* = *degree* *f*; *B* = *sqrt-int-ceiling* (*n*+*1*) ∗ *2*^*n* ∗ *A*;
  Bnd = *2*^(*n*^*2* *div* *2*) ∗ *B*^(*2*∗*n*); *l* = *log-ceiling* *p* *Bnd*;
  (-, *fs*) = *finite-field-factorization-int* *p* *f*;
  gs = *hensel-lifting* *p* *l* *f* *fs*;
  u = *choose-u* *gs*;
  pl = *p*^*l*;
  pl2 = *pl* *div* *2*;
  u′ = *poly-mod.inv-Mp2* *pl* *pl2* (*poly-mod.Mp* *pl* (*smult* *b* *u*));
  d = *degree* *u*;
  pl = *p*^*l*;
  L = *factorization-lattice* *u*′ *1* *pl*;
  g-star = *short-vector* *2* *L*
 in (
   ′′*p*^*l*:   ′′ @ *show* *pl* @ *shows-nl* [] @
   ′′*norm* *u*:   ′′ @ *show* (*sqrt-int-floor* (*sq-norm-poly* *u*′)) @ *shows-nl* [] @
   ′′*norm* *g-star*: ′′ @ *show* (*sqrt-int-floor* (*sq-norm-vec* *g-star*)) @ *shows-nl* [] @
shows-nl []
))

export-code *compute-norms* **in** *Haskell*

- p^l: $\approx 6.61056 \cdot 10^{122}$, namely 66105596879024859895191530803277103982840468296428121928464
- *norm u*: $\approx 6.67555 \cdot 10^{122}$, namely 66755505893812790838614155970749040661775649285326930
- *norm g-star*: $\approx 5.02568 \cdot 10^{110}$, namely 50256787188889378925810759939795033899734873138630

8.3 Verified wrong results

An equality in example 16.24 of the textbook which is not valid.

lemma *let* *g2* = [:−*984*,*1*:];
      g3 = [:−*72*,*1*:];
      g4 = [:−*6828*,*1*:];
      rhs = [:−*1728*,−*840*,−*420*,*6*:]
 in ¬ *poly-mod.eq-m* (*5*^*6*) (*smult* *6* (*g2*∗*g3*∗*g4*)) (*rhs*) ⟨*proof*⟩

end

References

[1] J. Divasón, S. J. C. Joosten, R. Thiemann, and A. Yamada. A formalization of the Berlekamp–Zassenhaus factorization algorithm. In *CPP 2017*, pages 17–29. ACM, 2017.

[2] J. v. z. Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, New York, NY, USA, 2nd edition, 2003.

[3] A. K. Lenstra, H. W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:515–534, 1982.