Coppersmith's Method

Katherine Kosaian and Yong Kiam Tan

March 19, 2025

Abstract

We formalize *Coppersmith's method*, an algorithm for finding small (in magnitude) roots of univariate integer polynomials mod M. Coppersmith's method has important applications in cryptography and is used in various attacks on the RSA algorithm for public-key cryptography. We also formalize a related (more lightweight) result with slightly weaker bounds; we split out the generic mathematical results underlying both this lightweight result and Coppersmith's method into a dedicated locale, which could be used to prove other "Coppersmith-like" results. Our work builds on the existing formalization of the LenstraLenstraLovász (LLL) algorithm for lattice basis reduction [2], and our formalization adds a determinant bound on the length of the short vector produced by the LLL algorithm.

There are many resources on Coppersmith's method that we found useful in the course of our development. Some that we recommend include Chapter 19 of a textbook by Galbraith [3], Section 17.3 of a textbook by Trappe and Washington [4], and the following set of lecture notes [1].

Contents

1 Preliminary results for LLL

In this file, we prove some additional results involving lattices and a bound for LLL.

theory *More-LLL*
imports
LLL-Basis-Reduction.LLL
begin

context *vec-module* **begin**

lemma *in-lattice-ofD*:
assumes x: $x \in$ *lattice-of a*
assumes a: *set a* $\subseteq$ *carrier-vec n*
obtains v **where**
$v \in$ *carrier-vec* (*length a*)
mat-of-cols n a $*_v$ *map-vec of-int* (v::*int vec*) $= x$
proof −
from *in-latticeE*[*OF x*]

```
  obtain c where
    c: x = sumlist (map (λ i. of-int (c i) ·v a ! i) [0 ..< length a]) .

  have x = lincomb-list (of-int ∘ c) a
    unfolding lincomb-list-def c by auto
  also have ... =
    mat-of-cols n a ∗v vec (length a) (of-int ∘ c)
    apply (subst lincomb-list-as-mat-mult)
    using a by auto
  also have ... =
    mat-of-cols n a ∗v map-vec of-int (vec (length a) c)
    by (auto simp add: map-vec)
  finally have x = mat-of-cols n a ∗v map-vec of-int (vec (length a) c) .
  thus ?thesis
    by (auto intro!: that)
qed

lemma exists-list-all2:
  assumes ∀ x. (x ∈ set xs ⟶ (∃ y. P x y))
  obtains ys where list-all2 P xs ys
  using assms
proof (induction xs arbitrary: thesis)
  case Nil
  then show ?case
    by auto
next
  case c: (Cons x xs)
  obtain y where y: P x y
    using c(3) by auto
  obtain ys where ys: list-all2 P xs ys
    using c
    by auto
  show ?case using c y ys by auto
qed

lemma subset-lattice-ofD:
  assumes xs: set xs ⊆ lattice-of a set xs ⊆ carrier-vec n
  assumes a: set a ⊆ carrier-vec n
  obtains vs where
    vs ∈ carrier-mat (length a) (length xs)
    mat-of-cols n a ∗ map-mat of-int vs = mat-of-cols n xs
proof −
  define P where P =
    (λx v. v ∈ carrier-vec (length a) ∧
    mat-of-cols n a ∗v map-vec of-int (v::int vec) = x)
  obtain ys where ys: list-all2 P xs ys
    apply (subst exists-list-all2[where P = P, where xs = xs])
    unfolding P-def
    apply (meson a in-lattice-ofD subsetD xs)
```

```
    by auto

  then have len: length xs = length ys
    using list-all2-lengthD by blast

  define vs where vs = mat-of-cols (length a) ys
  have 1: vs ∈ carrier-mat (length a) (length xs)
    unfolding vs-def
    by (metis len mat-of-cols-carrier(1))

  have i < length ys ⟹
    mat-of-cols n a *_v
    of-int-hom.vec-hom (col (mat-of-cols (length a) ys) i) = xs ! i for i
    by (metis P-def col-mat-of-cols list-all2-conv-all-nth ys)
  then have i < length ys ⟹
   col (mat-of-cols n a * of-int-hom.mat-hom (mat-of-cols (length a) ys)) i =
   col (mat-of-cols n xs) i for i
    apply (subst col-mat-of-cols)
    subgoal using len by auto
    subgoal using len xs(2) by fastforce
    apply (subst col-mult2)
    by auto
  then have 2: mat-of-cols n a * map-mat of-int vs = mat-of-cols n xs
    unfolding vs-def
    apply (intro mat-col-eqI)
    using len by auto

  show ?thesis
    apply (intro that)
    using 1 2 by auto
qed

lemma mk-coeff-list-nth:
  assumes i < length ls distinct ls
  shows mk-coeff ls f (ls ! i) = f i
  using assms
proof (induction ls arbitrary: f i)
  case Nil
  then show ?case
    by auto
next
  case (Cons l ls)
  then show ?case
    by (auto simp add: mk-coeff-Cons nth-equal-first-eq)
qed
```

This next lemma is trivial when the cols are not distinct.

```
lemma mat-mul-non-zero-col-lin-dep:
  assumes A: A ∈ carrier-mat n y distinct (cols A)
```

```
  assumes U: U ∈ carrier-mat y z
  assumes i: i < z col U i ≠ 0_v y
  assumes eqz: A * U = 0_m n z
  shows lin-dep (set (cols A))
proof −
  obtain j where j: j < y U $$ (j,i) ≠ 0
    using U i
    unfolding vec-eq-iff
    by auto

  have Aeq: A = mat-of-cols n (cols A)
    using A by auto
  have 0_v n = A *_v (col U i)
    using eqz
    unfolding mat-eq-iff vec-eq-iff
    using A U i by auto
  also have ... = mat-of-cols n (cols A) *_v vec (length (cols A)) (λia. U $$ (ia, i))
    unfolding col-def Aeq[symmetric] using A U by auto
  also have ... =
    lincomb-list (λia. U $$ (ia, i)) (cols A)
    apply (subst lincomb-list-as-mat-mult[symmetric])
     apply (metis assms(1) carrier-matD(1) carrier-vecD cols-dim subsetD)
    by auto
  also have ... =
    lincomb (mk-coeff (cols A) (λia. U $$ (ia, i))) (set (cols A))
    apply (subst lincomb-list-as-lincomb)
    using assms(1) cols-dim apply blast
    by auto
  finally have
    lc:lincomb (mk-coeff (cols A) (λia. U $$ (ia, i))) (set (cols A)) = 0_v n by auto

  have mc: mk-coeff (cols A) (λia. U $$ (ia, i)) (col A j) ≠ 0
    using mk-coeff-list-nth
    by (metis assms(1) assms(2) carrier-matD(2) cols-length cols-nth j)

  have col A j ∈ set (cols A)
    by (metis assms(1) carrier-matD(2) cols-length cols-nth j(1) nth-mem)
  thus ?thesis
    apply (intro lin-dep-crit [OF - - - - mc lc])
    using A j by auto
qed

lemma lin-indpt-mul-eq-ident:
  assumes a: distinct a lin-indpt (set a)
    set a ⊆ carrier-vec n length a = m
  assumes u: u ∈ carrier-mat m m
  assumes e: mat-of-cols n a = mat-of-cols n a * u
  shows u = 1_m m
proof (rule ccontr)
```

assume $u \neq 1_m\ m$
then obtain i **where** i: $i < m$
$col\ (u - 1_m\ m)\ i \neq 0_v\ m$
unfolding *vec-eq-iff mat-eq-iff*
using u **by** *auto*
have *1*:*mat-of-cols n a* ∈ *carrier-mat n m*
using *assms*(*4*) *mat-of-cols-carrier*(*1*) **by** *blast*
have *2*:*distinct* (*cols* (*mat-of-cols n a*))
by (*simp add*: *assms*(*1*) *assms*(*3*))

have *mat-of-cols n a* ∗ (*u* − 1_m *m*) = 0_m *n m*
by (*smt* (*verit*, *ccfv-SIG*) *assms*(*4*) *e mat-of-cols-carrier*(*1*) *minus-r-inv-mat mult-minus-distrib-mat one-carrier-mat right-mult-one-mat u*)
from *mat-mul-non-zero-col-lin-dep*[*OF 1 2 - i this*]
show *False*
using *a* **by** (*simp add*: *assms*(*3*) *minus-carrier-mat*)
qed

end

Set up a locale: vec_module where the ring has characteristic zero

locale *ring-char-0-vec-module* = *vec-module f-ty n* **for**
f-ty::*'a*:: {*comm-ring-1*, *ring-char-0*} *itself*
and *n*
begin

This next lemma shows that different basis a, b of the same lattice have the same Gram determinant.

lemma *lattice-of-eq-gram-det-eq*:
fixes *a b*::*'a vec list*
assumes *a*: *distinct a lin-indpt* (*set a*) *set a* ⊆ *carrier-vec n*
assumes *b*: *set b* ⊆ *carrier-vec n length a* = *length b*
assumes *lat-eq*: *lattice-of a* = *lattice-of b*
defines *A*: *A* ≡ *mat-of-cols n a*
defines *B*: *B* ≡ *mat-of-cols n b*
shows *det* (A^T ∗ *A*) = *det* (B^T ∗ *B*)
proof −
let *?m* = *length a*
have *set b* ⊆ *lattice-of a*
by (*simp add*: *lat-eq b basis-in-latticeI subsetI*)
from *subset-lattice-ofD*[*OF this*]
obtain *vs* **where**
vs: *vs* ∈ *carrier-mat* (*length a*) (*length b*)
mat-of-cols n a ∗ *map-mat of-int vs* = *mat-of-cols n b*
using *a b* **by** *blast*
then have *vsn*: *vs* ∈ *carrier-mat ?m ?m* **using** *b*(*2*) **by** *auto*
have *set a* ⊆ *lattice-of b*
using *a*(*3*) *lat-eq basis-in-latticeI* **by** *auto*
from *subset-lattice-ofD*[*OF this*]

obtain *us* **where**
us: *us* ∈ *carrier-mat* (*length b*) (*length a*)
mat-of-cols n b ∗ *map-mat of-int us* = *mat-of-cols n a*
using *a b* **by** *blast*
then have *usn*: *us* ∈ *carrier-mat ?m ?m* **using** *b*(*2*) **by** *auto*
have *mat-of-cols n a* =
(*mat-of-cols n a* ∗ *map-mat of-int vs*) ∗ *map-mat of-int us*
using *us vs* **by** *simp*
also have ... = *mat-of-cols n a* ∗ (*map-mat of-int vs* ∗ *map-mat of-int us*)
by (*auto intro*!: *assoc-mult-mat us vs*)
also have ... = *mat-of-cols n a* ∗ (*map-mat of-int* (*vs* ∗ *us*))
by (*metis of-int-hom.mat-hom-mult us*(*1*) *vs*(*1*))
finally have *mat-of-cols n a* = *mat-of-cols n a* ∗ (*map-mat of-int* (*vs* ∗ *us*)) **.**

from *lin-indpt-mul-eq-ident*[*OF a*(*1*−*3*) - - *this*]
have ((*map-mat of-int* (*vs* ∗ *us*))::′*a mat*) = (*1*$_m$ *?m*::′*a mat*)
using *vsn usn* **by** *auto*
then have *vs* ∗ *us* = *1*$_m$ *?m*
by (*simp add*: *of-int-hom.mat-hom-inj of-int-hom.mat-hom-one*)

then have *det vs* ∗ *det us* = *1*
using *det-mult*[*OF vsn usn*] **by** *simp*
then have (*det us*)^*2* = *1*
using *power2-eq-1-iff zmult-eq-1-iff* **by** *blast*
then have *us1*: (*det* (*map-mat of-int us*))^*2* = *1*
by (*metis of-int-hom.hom-det of-int-hom.hom-one of-int-power*)

have *Bcarr*:*B* ∈ *carrier-mat n ?m*
unfolding *B* **using** *b* **by** *auto*
then have *BtBcarr*:B^T ∗ *B* ∈ *carrier-mat ?m ?m* **by** *auto*
then have *uBtBcarr*: (*of-int-hom.mat-hom us*)^T ∗ (B^T ∗ *B*) ∈ *carrier-mat ?m ?m*
using *usn*
by *simp*

have *d*: *of-int-hom.mat-hom us* ∈ *carrier-mat* (*length a*) (*length a*)
using *b*(*2*) *us*(*1*) **by** *auto*
have *det* (A^T ∗ *A*) =
det ((*B* ∗ *map-mat of-int us*)^T ∗ (*B* ∗ *map-mat of-int us*))
using *us A B* **by** *auto*
also have ... = *det* ((*map-mat of-int us*)^T ∗ (B^T ∗ *B*) ∗ *map-mat of-int us*)
apply (*subst transpose-mult*[*OF Bcarr d*])
using *us b*
by (*smt* (*verit*, *ccfv-threshold*) *Bcarr assoc-mult-mat map-carrier-mat mult-carrier-mat transpose-carrier-mat*)
also have ... = *det* (*map-mat of-int us*)^*2* ∗ *det* (B^T ∗ *B*)
apply (*subst det-mult*[*OF uBtBcarr d*])
apply (*subst det-mult*[*OF* - *BtBcarr*])
using *usn* **by** (*auto simp add*: *det-transpose power2-eq-square*)

finally show *?thesis*
using *us1*
by (*metis l-one*)
qed

lemma *lattice-of-eq-gram-det-rows-eq*:
fixes *a b*::*'a vec list*
assumes *a*: *distinct a lin-indpt* (*set a*) *set a* ⊆ *carrier-vec n*
assumes *b*: *set b* ⊆ *carrier-vec n length a* = *length b*
assumes *lat-eq*: *lattice-of a* = *lattice-of b*
defines *A*: *A* ≡ *mat-of-rows n a*
defines *B*: *B* ≡ *mat-of-rows n b*
shows *det* (*A* ∗ A^T) = *det* (*B* ∗ B^T)
proof −
have *transpose-mat-a*: *A* = *transpose-mat* (*mat-of-cols n a*)
by (*simp add*: *transpose-mat-of-cols A*)
have *transpose-mat-b*: *B* = *transpose-mat* (*mat-of-cols n b*)
by (*simp add*: *transpose-mat-of-cols B*)
from *lattice-of-eq-gram-det-eq*[*OF assms*(*1*−*6*)]
have *det* ((*mat-of-cols n a*)^T ∗ *mat-of-cols n a*) =
det ((*mat-of-cols n b*)^T ∗ *mat-of-cols n b*) **by** *blast*
then show *?thesis*
using *transpose-mat-a transpose-mat-b*
by *auto*
qed

lemma *lattice-of-eq-sq-det-eq*:
fixes *a b*::*'a vec list*
assumes *a*: *distinct a lin-indpt* (*set a*) *set a* ⊆ *carrier-vec n length a* = *n*
assumes *b*: *set b* ⊆ *carrier-vec n length b* = *n*
assumes *lat-eq*: *lattice-of a* = *lattice-of b*
shows (*det* (*mat-of-cols n a*))^2 = (*det* (*mat-of-cols n b*))^2
proof −
from *lattice-of-eq-gram-det-eq*[*OF a*(*1*−*3*) *b*(*1*) - *lat-eq*]
have *det* ((*mat-of-cols n a*)^T ∗ (*mat-of-cols n a*)) =
det ((*mat-of-cols n b*)^T ∗ (*mat-of-cols n b*))
using *a b* **by** *auto*

thus *?thesis*
by (*metis assms*(*4*) *b*(*2*) *carrier-matI det-mult det-transpose index-transpose-mat*(*2*)
index-transpose-mat(*3*) *mat-of-cols-carrier*(*2*) *mat-of-cols-carrier*(*3*) *power2-eq-square*)
qed

lemma *lattice-of-eq-sq-det-rows-eq*:
fixes *a b*::*'a vec list*
assumes *a*: *distinct a lin-indpt* (*set a*) *set a* ⊆ *carrier-vec n length a* = *n*
assumes *b*: *set b* ⊆ *carrier-vec n length b* = *n*
assumes *lat-eq*: *lattice-of a* = *lattice-of b*
shows (*det* (*mat-of-rows n a*))^2 = (*det* (*mat-of-rows n b*))^2

proof −
have *transpose-mat-a*: *mat-of-rows n a* = *transpose-mat* (*mat-of-cols n a*)
by (*simp add*: *transpose-mat-of-cols*)
have *transpose-mat-b*: *mat-of-rows n b* = *transpose-mat* (*mat-of-cols n b*)
by (*simp add*: *transpose-mat-of-cols*)
have (*det* (*mat-of-cols n a*))^2 = (*det* (*mat-of-cols n b*))^2
using *lattice-of-eq-sq-det-eq assms* **by** *blast*
then show *?thesis*
using *transpose-mat-a transpose-mat-b det-transpose*
by (*metis assms*(*4*) *b*(*2*) *mat-of-rows-carrier*(*1*) *transpose-mat-of-rows*)
qed

end

context *LLL-with-assms*
begin

This next lemma bounds the size of the shortest vector by the determinant.

lemma *short-vector-det-bound*:
assumes *m*: *m* ≠ *0*
assumes *k*: *k* ≤ *m*
shows
(*rat-of-int* ‖*short-vector*‖^2) ^ *k* ≤
α ^ (*k* ∗ (*k*−*1*) *div* *2*) ∗ *rat-of-int* (*gs.Gramian-determinant reduce-basis k*)
proof −

from *reduce-basis*
have *rb*: *lattice-of reduce-basis* = *L*
reduced reduce-basis m
lin-indep reduce-basis
length reduce-basis = *m* **by** *auto*

have *sv0*: *short-vector* = *reduce-basis* ! *0*
using *rb m* **unfolding** *short-vector-def*
by (*metis hd-conv-nth list.size*(*3*))

have *map of-int-hom.vec-hom reduce-basis* ! *0* = *of-int-hom.vec-hom short-vector*
apply (*subst nth-map*)
using *m* **by** (*auto simp add*: *sv0 rb*)

then have *sve*: *rat-of-int* ‖ *short-vector* ‖^2 =
‖*gram-schmidt-fs.gso n* (*map of-int-hom.vec-hom reduce-basis*) *0*‖^2
by (*smt* (*verit*, *del-insts*) *LLL-invD*(*2*) *assms*(*1*) *cof-vec-space.lin-indpt-list-def gram-schmidt-fs-Rn.fs0-gso0 gram-schmidt-fs-Rn.intro not-gr0 rb*(*3*) *reduce-basis-inv sq-norm-of-int*)

from *reduce-basis-inv*
have *LLLi*: *LLL-invariant True m reduce-basis* **by** *auto*

then have *LLL-invariant-weak reduce-basis*
using *LLL-inv-imp-w* **by** *auto*

from *Gramian-determinant*[*OF this k*]
have *gd*: *rat-of-int* (*gs.Gramian-determinant reduce-basis k*) =
($\prod i{<}k.$
$\|$*gram-schmidt-fs.gso n* (*map of-int-hom.vec-hom reduce-basis*) *i*$\|^2$)
0 < *gs.Gramian-determinant reduce-basis k* **by** *auto*

have *weakly-reduced reduce-basis m*
using *LLL.LLL-invD*(*8*) *LLLi* **by** *blast*

then have ∗: (∀ *i*. *Suc i* < *m* ⟶
$\|$*gram-schmidt-fs.gso n* (*map of-int-hom.vec-hom reduce-basis*) *i*$\|^2$ ≤
α ∗ $\|$*gram-schmidt-fs.gso n* (*map of-int-hom.vec-hom reduce-basis*) (*Suc i*)$\|^2$)
using *gram-schmidt-fs.weakly-reduced-def* **by** *blast*

have ∗∗: *i* < *k* ⟹
$\|$*gram-schmidt-fs.gso n* (*map of-int-hom.vec-hom reduce-basis*) *0*$\|^2$ ≤
α^*i* ∗ $\|$*gram-schmidt-fs.gso n* (*map of-int-hom.vec-hom reduce-basis*) *i*$\|^2$ **for** *i*
proof (*induction i*)
case *0*
then show *?case*
by *auto*
next
case (*Suc i*)
from *Suc*
have *1*: $\|$*gram-schmidt-fs.gso n* (*map of-int-hom.vec-hom reduce-basis*) *0*$\|^2$
≤ *α* ^ *i* ∗ $\|$*gram-schmidt-fs.gso n* (*map of-int-hom.vec-hom reduce-basis*) *i*$\|^2$
by *auto*

have *Suc i* < *m*
using *Suc*(*2*) *k order-less-le-trans* **by** *blast*
then
have $\|$*gram-schmidt-fs.gso n* (*map of-int-hom.vec-hom reduce-basis*) *i*$\|^2$
≤ *α* ∗ $\|$*gram-schmidt-fs.gso n* (*map of-int-hom.vec-hom reduce-basis*) (*Suc i*)$\|^2$
by (*meson* ∗ *Suc*(*2*) *less-Suc-eq not-less-eq*)

then have *2*: *α* ^ *i* ∗$\|$*gram-schmidt-fs.gso n* (*map of-int-hom.vec-hom reduce-basis*) *i*$\|^2$
≤ *α* ^ (*i*+*1*) ∗ $\|$*gram-schmidt-fs.gso n* (*map of-int-hom.vec-hom reduce-basis*) (*Suc i*)$\|^2$
by (*simp add*: *α0*(*1*))

show *?case* **using** *1 2* **by** *auto*
qed

have (*rat-of-int* ‖ *short-vector* ‖^2) ^ *k* =
(∏ *i*<*k*. ‖*gram-schmidt-fs.gso n* (*map of-int-hom.vec-hom reduce-basis*) *0*‖^2)
using *prod-pow sve*
by *simp*

also have ... ≤ (∏ *i*<*k*. α^*i* ∗ ‖*gram-schmidt-fs.gso n* (*map of-int-hom.vec-hom reduce-basis*) *i*‖^2)
apply (*intro prod-mono*)
using ∗∗ **by** *auto*

also have ... = (∏ *i*<*k*. α^*i*) ∗ *rat-of-int* (*gs.Gramian-determinant reduce-basis k*)
unfolding *gd*
by *simp*

also have ... = α^(*k* ∗ (*k*−*1*) *div 2*) ∗ *rat-of-int* (*gs.Gramian-determinant reduce-basis k*)
unfolding *power-sum*[*symmetric*]
by (*metis Sum-Ico-nat atLeast0LessThan mult-eq-0-iff verit-minus-simplify*(*2*))

finally show *?thesis* **.**
qed

lemma *square-Gramian-determinant-eq-det-square*:
assumes *sq*:*n* = *m*
shows *gs.Gramian-determinant fs-init m* =
(*det* (*mat-of-rows m fs-init*))^*2*
unfolding *gs.Gramian-determinant-def*
apply (*subst gs.Gramian-matrix-alt-def*)
apply (*simp add*: *len*)
by (*metis LLL-invD*(*9*) *assms det-mult det-transpose len mat-of-cols-carrier*(*1*) *mat-of-rows-carrier*(*1*) *power2-eq-square reduce-basis-inv take-all transpose-mat-of-rows*)

end

end

2 Algorithm for Coppersmith's Method

In this file, we formalize the algorithm for Coppersmith's method. We follow the descriptions in Chapter 19 of "Mathematics of Public Key Cryptography" by Steven Galbraith and Section 17.3 of "Introduction to Cryptography with Coding Theory" by Trappe and Washington. We first formalize an algorithm for a "lightweight" version of Coppersmith's method, and then formalize Coppersmith's method itself. Correctness proofs for these algorithms are in "Towards_Coppersmith.thy" and "Coppersmith.thy".

theory *Coppersmith-Algorithm*

imports *LLL-Factorization.LLL-Factorization*
begin

This next definition forms the vector associated to a polynomial as in (19.1) of Galbraith.

definition *vec-associated-to-poly*:: (*'a*::{*ring*,*power*}) *poly* ⇒ *'a* ⇒ *'a vec* **where**
vec-associated-to-poly p X = *vec* (*degree p* + *1*) (λ*j*. (*coeff p j*) ∗ *X*^*j*)

abbreviation *vec-associated-to-int-poly*:: *int poly* ⇒ *int* ⇒ *int vec* **where**
vec-associated-to-int-poly ≡ *vec-associated-to-poly*
abbreviation *vec-associated-to-real-poly*:: *real poly* ⇒ *real* ⇒ *real vec* **where**
vec-associated-to-real-poly ≡ *vec-associated-to-poly*

definition *int-poly-associated-to-vec*:: *int vec* ⇒ *nat* ⇒ *int poly* **where**
int-poly-associated-to-vec v X = (
let newvec = *vec* (*dim-vec v*) (λ*j*. *floor* (((*v* $ *j*)::*real*)/(*X*^*j*))) *in*
∑ *i*<(*dim-vec v*). *monom* (*newvec* $ *i*) *i*
)

2.1 Lightweight method similar to Coppersmith

In this section, we start with a more lightweight version of Coppersmith which doesn't achieve the full bounds, but is built on similar ideas.

This next definition constructs the g_i's

definition *g-i-vec*:: *nat* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *int vec* **where**
g-i-vec M X i n = *vec n* (λ*j*. *if j* = *i then M*∗*X*^*i else 0*)

This next definition should be called with degree d = p - 1

definition *form-basis-helper*:: *int poly* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *int vec list* **where**
form-basis-helper p M X d = *map* (λ*i*. *g-i-vec M X i* (*degree p* + *1*)) [*0*..<*d* + *1*]

definition *form-basis*:: *int poly* => *nat* ⇒ *nat* ⇒ *nat* ⇒ *int vec list* **where**
form-basis p M X d = (*form-basis-helper p M X d*) @ [*vec-associated-to-int-poly p X*]

definition *first-vector*:: *int poly* ⇒ *nat* ⇒ *nat* ⇒ *int vec* **where**
first-vector p M X = (
let cs-basis = *form-basis p M X* (*degree p* − *1*) *in*
(*short-vector 2 cs-basis*)
)

definition *towards-coppersmith*:: *int poly* ⇒ *nat* ⇒ *nat* ⇒ *int poly* **where**
towards-coppersmith p M X = *int-poly-associated-to-vec* (*first-vector p M X*) *X*

2.2 Full Coppersmith

In this section, we develop the full Coppersmith's algorithm.

definition *vec-associated-to-int-poly-padded*:: *nat* ⇒ *int poly* ⇒ *nat* ⇒ *int vec* **where**
vec-associated-to-int-poly-padded n p X = *vec n* (λ*j*. (*coeff p j*)∗*X*^*j*)

definition *row-of-coppersmith-matrix*:: *int poly* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *int vec*
where *row-of-coppersmith-matrix p M X h i j* =
vec-associated-to-int-poly-padded ((*degree p*)∗*h*) (*smult* (((*M*^(*h*−1−*j*)))) (*p*^*j*∗(*monom* 1 *i*))) *X*

definition *form-basis-coppersmith-aux*:: *int poly* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *int vec list*
where *form-basis-coppersmith-aux p M X h j* = (*map* (λ*i*. *row-of-coppersmith-matrix p M X h i j*) [*0*..<*degree p*])

definition *form-basis-coppersmith*:: *int poly* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *int vec list*
where *form-basis-coppersmith p M X h* = *concat* (*map* (λ*j*. *form-basis-coppersmith-aux p M X h j*) [*0*..< (*h*::*nat*)])

definition *calculate-h-coppersmith-aux*:: *int poly* ⇒ *real* ⇒ *int* **where**
calculate-h-coppersmith-aux p e = (*let d* = *degree p in* ((*ceiling* (((*d*−*1*)/(*d*∗(*e*::*real*)) + *1*)/*d*))::*int*))

definition *calculate-h-coppersmith*:: *int poly* ⇒ *real* ⇒ *nat* **where**
calculate-h-coppersmith p e = (*nat* (*calculate-h-coppersmith-aux p e*))

Note that we pass 2 as a parameter to the LLL algorithm. Any bound $> 4/3$ would work.

definition *first-vector-coppersmith*:: *int poly* ⇒ *nat* ⇒ *nat* ⇒ *real* ⇒ *int vec* **where**
first-vector-coppersmith p M X epsilon = (
let cs-basis = *form-basis-coppersmith p M X* (*calculate-h-coppersmith p epsilon*)
in
short-vector 2 cs-basis)

definition *coppersmith*:: *int poly* ⇒ *nat* ⇒ *nat* ⇒ *real* ⇒ *int poly* **where**
coppersmith p M X epsilon =
int-poly-associated-to-vec (*first-vector-coppersmith p M X epsilon*) *X*

end

3 Howgrave-Graham's theorem

In this file, we prove a result due to Howgrave-Graham on small-enough roots of polynomials mod M (see Theorem 19.1.2 in "Mathematics of Public Key Cryptography" by Galbraith).

theory *Howgrave-Graham*

imports *Coppersmith-Algorithm*

HOL−Analysis.L2-Norm
LLL-Basis-Reduction.Norms
begin

abbreviation *euclidean-norm-int-vec*::*int vec* ⇒ *real*
where *euclidean-norm-int-vec v* ≡ *sqrt* (*sq-norm-vec v*)

abbreviation *euclidean-norm-real-vec*::*real vec* ⇒ *real*
where *euclidean-norm-real-vec v* ≡ *sqrt* (*sq-norm-vec v*)

lemma *euclidean-norm-int-vec-eq*:
shows *euclidean-norm-int-vec v* = *sqrt* (∑ *i*<(*dim-vec v*). (*v*$*i*)^2)
unfolding *sq-norm-vec-def sum-list-sum-nth*
by (*auto simp add*: *list-of-vec-index lessThan-atLeast0 power2-eq-square*)

lemma *euclidean-norm-real-vec-eq*:
shows *sqrt* (*sq-norm-vec v*) = *sqrt* (∑ *i*<(*dim-vec v*). (*v*$*i*)^2)
unfolding *sq-norm-vec-def sum-list-sum-nth*
by (*auto simp add*: *list-of-vec-index lessThan-atLeast0 power2-eq-square*)

lemma *euclidean-norm-gteq0*:
shows *euclidean-norm-real-vec* (*a*::*real vec*) ≥ *0*
euclidean-norm-int-vec (*c*::*int vec*) ≥ *0*
by (*auto simp add*: *sq-norm-vec-ge-0*)

lemma *dim-vec-vec-associated-to-poly*[*simp*]:
shows *dim-vec* (*vec-associated-to-poly F X*) = *degree F* + *1*
unfolding *vec-associated-to-poly-def* **by** *auto*

lemma *Cauchy-Schwarz-sum*:
fixes *n*:: *nat*
fixes *x*:: *nat* ⇒ *real*
shows (∑ *i*≤*n*. *x i*) ≤ *sqrt* ((*n*+*1*) ∗ (∑ *i*≤*n*. (*x i*)^2))
proof −
have (∑ *i*≤*n*. *x i*) ≤ (∑ *i*≤*n*. |*x i*| ∗ |*1*|)
by (*auto intro*!: *sum-mono*)
also have ... ≤ *L2-set x* {..*n*} ∗ *L2-set* (λ-. *1*) {..*n*}
using *L2-set-mult-ineq*
by *fastforce*
also have ... = *sqrt* (∑ *i*≤*n*. (*x i*)^2) ∗ *sqrt* (∑ *i*≤*n*. *1*^2)
unfolding *L2-set-def* **by** *auto*
also have ... = *sqrt* ((*n*+*1*) ∗ (∑ *i*≤*n*. (*x i*)^2))
using *real-sqrt-mult* **by** *auto*
finally show *?thesis* **.**
qed

lemma *abs-mult-sum*:
fixes *f g*:: *nat* ⇒ *real*
fixes *n*:: *nat*

shows $abs(\sum i{\leq}n.\ (f\ i){*}(g\ i))$
$\leq (\sum i{\leq}n.\ (abs\ (f\ i)){*}(abs\ (g\ i)))$
proof (*induct n*)
case *0*
have $|f\ 0 * g\ 0| \leq |f\ 0| * |g\ 0|$
by (*simp add*: *abs-mult*)
then show *?case* **using** *0*
by *auto*
case (*Suc n*)
then have $|\sum i{\leq}Suc\ n.\ f\ i * g\ i| \leq |\sum i{\leq}n.\ f\ i * g\ i| + |f\ (Suc\ n) * g\ (Suc\ n)|$
by (*simp add*: *abs-triangle-ineq*)
then have $|\sum i{\leq}Suc\ n.\ f\ i * g\ i| \leq (\sum i{\leq}n.\ |f\ i| * |g\ i|) + |f\ (Suc\ n) * g\ (Suc\ n)|$
using *Suc* **by** *auto*
then show *?case*
using *abs-mult*[*of f* (*Suc n*) *g* (*Suc n*)]
by (*metis* (*mono-tags*, *lifting*) *sum.atMost-Suc*)
qed

lemma *sum-helper*:
fixes *g h*:: *nat* ⇒ *real*
assumes $\forall\ i \leq n.\ f\ i \geq 0$
assumes $\forall\ i \leq n.\ g\ i \leq h\ i$
shows $(\sum i{\leq}n.\ (f\ i){*}\ (g\ i)) \leq (\sum i{\leq}n.\ (f\ i){*}\ (h\ i))$
using *assms*
proof (*induct n*)
case *0*
have $0 \leq f\ 0 \Longrightarrow g\ 0 \leq h\ 0 \Longrightarrow f\ 0 * g\ 0 \leq f\ 0 * h\ 0$
by (*simp add*: *mult-left-mono*)
then show *?case* **using** *0*
by *auto*
next
case (*Suc n*)
have $0 \leq f\ (Suc\ n) \Longrightarrow g\ (Suc\ n) \leq h\ (Suc\ n) \Longrightarrow f\ (Suc\ n) * g\ (Suc\ n) \leq f\ (Suc\ n) * h\ (Suc\ n)$
by (*simp add*: *mult-left-mono*)
then have *Suc-h*: $f\ (Suc\ n) * g\ (Suc\ n) \leq f\ (Suc\ n) * h\ (Suc\ n)$
using *Suc* **by** *auto*
have *ind-h*: $(\sum i{\leq}n.\ f\ i * g\ i) \leq (\sum i{\leq}n.\ f\ i * h\ i)$
using *Suc* **by** *auto*
show *?case* **using** *ind-h Suc-h*
by *auto*
qed

theorem *Howgrave-Graham*:
fixes *F* :: *real poly*
fixes *M X* :: *nat*
fixes *x0 k* :: *int*
assumes *M-gt*: $M > 0$

assumes *root-mod-M*: *poly F* (*real-of-int x0*) = *k* ∗ *M*
assumes *root-bound*: *abs x0* ≤ *X*
assumes *norm-bound*: *sqrt* (‖*vec-associated-to-poly F X*‖²) < *M* / *sqrt* (*degree F* + *1*)
shows *poly F x0* = *0*
proof −
let *?d* = *degree F*
let *?bF* = *vec-associated-to-poly F X*

have *h2*: *i*≤*?d* ⟹ *real-of-int*(|*x0*| ^ *i*) ≤ *real* (*X* ^ *i*) **for** *i*
using *root-bound*
by (*simp add*: *linordered-semidom-class.power-mono*)

have *abs* (*poly F x0*) =
abs (∑ *i*≤*?d*. (*coeff F i*)∗ *x0*^*i*)
using *poly-altdef*
by (*smt* (*verit*) *of-int-power sum.cong*)
also have ... ≤ (∑ *n*≤*?d*. |*poly.coeff F n* ∗ *real-of-int* (*x0* ^ *n*)|)
by *blast*
also have ... = (∑ *n*≤*?d*. |*poly.coeff F n*| ∗ *real-of-int* (|*x0*| ^ *n*))
by (*simp add*: *abs-mult power-abs*)
also have ... ≤ (∑ *n*≤*?d*. |*poly.coeff F n*| ∗ (*X*^*n*))
using *h2*
by (*auto intro*:*sum-mono simp add*: *mult-left-mono*)
also have ... ≤ *sqrt* ((*?d*+*1*)∗(∑ *n*≤*?d*. (|*poly.coeff F n*| ∗ (*X*^*n*))^*2*))
using *Cauchy-Schwarz-sum* **by** *blast*
also have ... = *sqrt* ((*?d*+*1*)∗(∑ *n*≤*?d*. (*poly.coeff F n* ∗ (*X*^*n*))^*2*))
by (*simp add*: *power-mult-distrib*)
also have ... = *sqrt* ((*?d*+*1*)∗(∑ *i*≤*?d*. (*?bF* $ *i*)^*2*))
unfolding *vec-associated-to-poly-def* **by** *auto*
also have ... = *sqrt* ((*?d*+*1*)∗(∑ *i*<*?d*+*1*. (*?bF* $ *i*)^*2*))
using *Suc-eq-plus1 lessThan-Suc-atMost* **by** *presburger*
also have ... = *sqrt* (*?d*+*1*) ∗ *sqrt* (∑ *i*<*?d*+*1*. (*?bF* $ *i*)^*2*)
using *real-sqrt-mult* **by** *blast*
also have ... = *sqrt* (*?d*+*1*) ∗ *sqrt* (*sq-norm-vec ?bF*)
unfolding *euclidean-norm-real-vec-eq*
by *force*
also have ... < *M*
using *norm-bound*
by (*smt* (*verit*, *ccfv-SIG*) *Groups.mult-ac*(*2*) *add-is-0 eq-numeral-extra*(*2*) *mult-imp-div-pos-le of-nat-le-0-iff real-sqrt-gt-zero*)
finally have *abs* (*poly F x0*) < *M* **.**
then have −*M* < *poly F x0* ∧ *poly F x0* < *M*
by *linarith*
thus *?thesis*
using *root-mod-M M-gt*
by (*smt* (*verit*, *del-insts*) *aux-abs-int mult.commute mult-eq-0-iff of-int-hom.hom-0-iff of-int-less-iff of-int-of-nat-eq*)
qed

abbreviation *int-poly-to-real-poly*:: *int poly* ⇒ *real poly*
where *int-poly-to-real-poly F* ≡ *map-poly real-of-int F*

lemma *int-poly-to-real-poly-same-norm*:
fixes *X* :: *nat*
shows *euclidean-norm-int-vec* (*vec-associated-to-int-poly F X*) =
euclidean-norm-real-vec (*vec-associated-to-real-poly* (*int-poly-to-real-poly F*) *X*)
proof −
let *?d* = *dim-vec* (*vec-associated-to-int-poly F X*)
have *same-dim*: *dim-vec* (*vec-associated-to-int-poly F X*) =
dim-vec (*vec-associated-to-real-poly* (*int-poly-to-real-poly F*) *X*)
by *simp*
have ⋀*i*. *i* < *?d* ⟹ (*vec-associated-to-real-poly* (*int-poly-to-real-poly F*) *X* $ *i*) = (*real-of-int* (*vec-associated-to-int-poly F X* $ *i*))
unfolding *vec-associated-to-poly-def* **by** *auto*
then have (∑ *x*<*?d*. (*real-of-int* (*vec-associated-to-int-poly F X* $ *x*))^2) =
(∑ *i*<*?d*. (*vec-associated-to-real-poly* (*int-poly-to-real-poly F*) *X* $ *i*)^2)
by *fastforce*
then show *?thesis*
unfolding *euclidean-norm-int-vec-eq euclidean-norm-real-vec-eq*
using *same-dim*
by *auto*
qed

Now we restate the result over int polys.

lemma *Howgrave-Graham-int-poly*:
fixes *F*:: *int poly*
fixes *M X*:: *nat*
fixes *x0*:: *int*
assumes *M-gt*: *M* > *0*
assumes *root-mod-M*: *poly F x0 mod M* = *0*
assumes *root-bound*: *abs x0* ≤ *X*
assumes *norm-bound*: *sqrt* (*sq-norm-vec* (*vec-associated-to-int-poly F X*)) < *M* / *sqrt* (*degree F* + *1*)
shows *poly F x0* = *0*
proof −
let *?rF* = *int-poly-to-real-poly F*
from *root-mod-M* **have** ‹*M dvd poly F x0*›
by *presburger*
then obtain *k* **where** ‹*poly F x0* = *int M* ∗ *k*› **..**
then have *poly ?rF* (*real-of-int x0*) = *0*
apply (*intro Howgrave-Graham*[*OF M-gt - root-bound*])
using *assms int-poly-to-real-poly-same-norm*[*of F X*] **by** *auto*
thus *?thesis* **by** *auto*
qed

end

4 Coppersmith Generic

In this file, we develop the generic argument behind Coppersmith's method.

theory *Coppersmith-Generic*

imports *Coppersmith-Algorithm*
Howgrave-Graham
More-LLL
begin

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

4.1 Some matrix properties

This definition should only be used for lists of vectors that correspond to square matrices

definition *vec-list-to-square-mat*:: *'a vec list* ⇒ *'a mat*
where *vec-list-to-square-mat L* = *mat-of-rows* (*length L*) *L*

This lemma is similar to upper_triangular_imp_distinct, from the "Jordan_Normal_Form.Matrix" library by Thiemann and Yamada.

lemma *lower-triangular-imp-distinct*:
assumes *A*: *A* ∈ *carrier-mat n n*
and *tri*: ⋀*i j*. *i* < *j* ⟹ *j* < *n* ⟹ *A* $$ (*i*,*j*) = *0*
and *diag*: *0* ∉ *set* (*diag-mat A*)
shows *distinct* (*rows A*)
proof−
{fix *i* **and** *j*
assume *eq*: *rows A* ! *i* = *rows A* ! *j* **and** *ij*: *i* < *j* **and** *jn*: *j* < *n*
from *tri A ij jn* **have** *rows A* ! *i* $ *j* = *0*
by *simp*
with *eq* **have** *rows A* ! *j* $ *j* = *0* **by** *auto*
with *diag ij jn A* **have** *False* **by** (*auto simp*: *diag-mat-def*)
}
with *A* **show** *?thesis* **by** (*force simp*: *distinct-conv-nth nat-neq-iff*)
qed

lemma *lower-triangular-imp-det-eq-0-iff*:
fixes *A* :: *'a* :: *idom mat*
assumes *A* ∈ *carrier-mat n n* **and** ⋀*i j*. *i* < *j* ⟹ *j* < *n* ⟹ *A* $$ (*i*,*j*) = *0*
shows *det A* = *0* ⟷ *0* ∈ *set* (*diag-mat A*)
using *assms det-lower-triangular*
by (*metis semidom-class.prod-list-zero-iff*)

This next lemma is similar to upper_triangular_imp_lin_indpt_rows, from the "Jordan_Normal_Form.VS_Connect" library by Thiemann and Yamada.

lemma (**in** *idom-vec*) *lower-triangular-imp-lin-indpt-rows*:

assumes *A*: $A \in$ *carrier-mat n n*
and *lower-tri*: $\bigwedge i\ j.\ i < j \Longrightarrow j < n \Longrightarrow A\ \$\$\ (i,j) = 0$
and *diag*: $0 \notin$ *set* (*diag-mat A*)
shows *lin-indpt* (*set* (*rows A*))
using *det-not-0-imp-lin-indpt-rows lower-triangular-imp-det-eq-0-iff assms*
by *auto*

lemma *g-i-vec-ith-element*:
assumes *degree p* $\geq$ *1*
assumes *i* $<$ *degree p*
assumes *M* $>$ *0*
assumes *X* $>$ *0*
shows (*g-i-vec M X i* (*degree p* + *1*)) $ *i* = *M*X^i*
unfolding *g-i-vec-def*
using *assms*(*2*) *assms*(*3*) *assms*(*4*) **by** *auto*

lemma *ith-row-form-basis-helper*:
assumes *i* $\leq$ *d*
shows ((*form-basis-helper p M X d*)!*i*) = *g-i-vec M X i* (*degree p* + *1*)
using *assms* **unfolding** *form-basis-helper-def*
by (*metis add-cancel-right-left add-cancel-right-right le-imp-less-Suc less-diff-conv nth-map-upt semiring-norm*(*174*))

lemma *no-zeros-on-diagonal-helper*:
assumes *degree p* $\geq$ *1*
assumes *i* $<$ *degree p*
assumes *M* $>$ *0*
assumes *X* $>$ *0*
shows ((*form-basis-helper p M X* (*degree p* − *1*))!*i*)$*i* = *M*X^i*
using *ith-row-form-basis-helper assms g-i-vec-ith-element*
by (*metis Suc-eq-plus1 less-diff-conv2 not-less-eq verit-comp-simplify*(*3*))

4.2 Casting lemmas

lemma *casted-distinct-is-distinct*:
fixes *vecs*:: *int vec list*
assumes *distinct-vecs*: *distinct vecs*
shows *distinct* ((*map of-int-vec vecs*)::*rat vec list*)
proof −
let *?map-list* = (*map of-int-vec vecs*)::*rat vec list*
have *same-map*: *?map-list* = *map* (*map-vec rat-of-int*) *vecs*
by *simp*
have *eq-iff*: $\bigwedge a\ b$::*int vec*. *a* = *b* $\longleftrightarrow$ *map-vec rat-of-int a* = *map-vec rat-of-int b*
using *of-int-hom.vec-hom-inj* **by** *blast*
then have *inj* (*map-vec rat-of-int*) **unfolding** *inj-def*
by *simp*
then show *?thesis* **using** *distinct-map*[*of map-vec rat-of-int vecs*] *distinct-vecs same-map inj-on-def local.eq-iff*
by *blast*

qed

Copy pasted from VS_Connect, which only has it for field coefficients

lemma (**in** *vec-module*) *finsum-dim*:
shows $f \in A \to$ *carrier-vec* $n \Longrightarrow$
dim-vec (*finsum* V f A) = n
by (*smt* (*verit*) *M.add.finprod-closed carrier-vecD*)

lemma (**in** *vec-module*) *finsum-index*:
assumes *i*: $i < n$
and *f*: $f \in X \to$ *carrier-vec* n
and *X*: $X \subseteq$ *carrier-vec* n
shows *finsum* V f X \$ i = *sum* ($\lambda x.$ f x \$ i) X
using X f
proof (*induct X rule*: *infinite-finite-induct*)
case *empty*
then show *?case* **using** *i* **by** *simp* **next**
case (*insert x X*)
then have *Xf*: *finite X*
and *xX*: $x \notin X$
and *x*: $x \in$ *carrier-vec* n
and *X*: $X \subseteq$ *carrier-vec* n
and *fx*: f $x \in$ *carrier-vec* n
and *f*: $f \in X \to$ *carrier-vec* n **by** *auto*
have *i2*: $i <$ *dim-vec* (*finsum* V f X)
using *i finsum-closed*[*OF f*] **by** *auto*
have *ix*: $i <$ *dim-vec* x **using** x i **by** *auto*
show *?case*
unfolding *finsum-insert*[*OF Xf xX f fx*]
unfolding *sum.insert*[*OF Xf xX*]
unfolding *index-add-vec*(*1*)[*OF i2*]
using *insert lincomb-def*
by *auto*
qed (*insert i*, *auto*)

lemma *is-int-rat-mul-of-int*:
assumes *snd* (*quotient-of y*) *dvd x*
shows *is-int-rat* (*of-int x* $*$ *y*)
proof −
have *y*: *y* $*$ *of-int* (*snd* (*quotient-of y*)) = *of-int* (*fst* (*quotient-of y*))
by (*smt* (*verit*) *nonzero-eq-divide-eq of-int-eq-0-iff prod.collapse quotient-of-div quotient-of-nonzero*(*2*))
obtain *k* **where** *x* = *k* $*$ *snd* (*quotient-of y*)
by (*smt* (*verit*) *assms dvd-div-mult-self*)
thus *?thesis*
by (*smt* (*verit*) *Groups.mult-ac*(*2*) *Groups.mult-ac*(*3*) *Ints-of-int y is-int-rat of-int-mult*)
qed

lemma *casting-lin-comb-helper-set*:
assumes *vs*: $vs \subseteq$ *carrier-vec n*
assumes *ldi*: *module.lin-dep class-ring* (*module-vec TYPE*(*rat*) *n*) (*of-int-vec* ‘ *vs*)
shows *module.lin-dep class-ring* (*module-vec TYPE*(*int*) *n*) *vs*
proof −
interpret *vmint*: *vec-module TYPE*(*int*) *n* **.**
interpret *vmrat*: *vec-module TYPE*(*rat*) *n* **.**

have *mrat*:*module class-ring vmrat.V*
using *vec-module* **by** *blast*
have *mint*:*module class-ring vmint.V*
using *vmint.module-axioms* **by** *auto*

from *ldi*
obtain *A a va* **where** *av*: *finite A A* $\subseteq$ *of-int-vec* ‘ *vs*
$a \in A \to$ *carrier class-ring*
vmrat.lincomb a A $= \mathbf{0}_{vmrat.V}$
$va \in A$ *a va* $\neq \mathbf{0}_{class\text{-}ring}$
unfolding *module.lin-dep-def*[*OF mrat*]
by *blast*

obtain *B vb* **where** *B*: *finite B B* $\subseteq$ *vs A* = *of-int-vec* ‘ *B*
vb $\in$ *B* **and** *va*: *va* = *of-int-vec vb*
by (*metis* (*no-types*, *opaque-lifting*) *av*(*1*) *av*(*2*) *av*(*5*) *finite-subset-image image-iff*)

have *Bcarr*: *B* $\subseteq$ *carrier-vec n*
using *B*(*2*) *vs*
by *auto*
then have *Acarr*: *A* $\subseteq$ *carrier-vec n*
unfolding *B*
by *auto*
have *0*: *finite B*
using *B*(*1*) *assms*(*1*) *infinite-super* **by** *blast*

maximum denominator in any coefficient

define *md* **where** (*md*::*rat*) = *of-int* (*Prod* ((λ*r*. *snd* (*quotient-of r*)) ‘ (*image a A*)))

have *mdpos*: *md* > *0* **unfolding** *md-def*
by (*metis* (*no-types*, *lifting*) *imageE of-int-pos prod-pos quotient-of-denom-pos*′)

have *inj-of-int-vec*:*inj-on* (*of-int-vec*:: *int vec* ⇒ *rat vec*) *B*
unfolding *inj-on-def vec-eq-iff*
by (*auto simp add*:)

have ∗:(λ*v*. *md* ∗ *a v* ·$_v$ *v*) $\in$ *of-int-vec* ‘ *B* → *carrier-vec n*

using *Acarr B*(*3*) **by** *blast*

have $v \in A \Longrightarrow$ *is-int-rat* (*md* $*$ *a v*) **for** *v*
unfolding *md-def*
apply (*intro is-int-rat-mul-of-int*)
by (*metis av*(*1*) *dvd-prodI finite-imageI imageI*)

define *b* **where** *b* = (λ*v*::*int vec*. *int-of-rat* (*md* $*$ *a* (*of-int-vec v*)))

have *bvb*: *b vb* = *int-of-rat* (*md* $*$ *a va*)
unfolding *b-def va*[*symmetric*] **by** *auto*

have *a va* ≠ *0* **using** *av*(*6*)
by *auto*
then have *fst* (*quotient-of* (*a va*)) ≠ *0*
by (*metis Is-Rat-To-Rat.int-of-rat-0 Is-Rat-To-Rat.int-of-rat-def*)
then have *1*: *b vb* ≠ *0*
using *bvb mdpos*
by (*simp add*: ‹*a va* ≠ *0*›)

have [*simp*]: *dim-vec* ($\bigoplus_{vmint.V} v \in B$. *b v* $\cdot_v$ *v*) = *n*
apply (*subst vmint.finsum-dim*)
using *Bcarr* **by** *auto*
have [*simp*]: *dim-vec* (*finsum vmrat.V* (*of-int-vec* ∘ (λ*v*. *b v* $\cdot_v$ *v*)) *B*) = *n*
apply (*subst vmrat.finsum-dim*)
using *Bcarr* **by** *auto*

have *all-B-in-carrier-n*: $\bigwedge x.\ x \in B \Longrightarrow x \in$ *carrier-vec n*
by (*metis B*(*2*) *basic-trans-rules*(*23*) *smult-carrier-vec vmint.summands-valid vs*)
then have *B-prop2*: $\bigwedge i.\ i \in B \Longrightarrow$
rat-of-int (*int-of-rat* (*md* $*$ *a* (*of-int-vec i*))) $\cdot_v$
of-int-vec i =
md $*$ *a* (*of-int-vec i*) $\cdot_v$ *of-int-vec i*
using *B*(*3*) ‹$\bigwedge v.\ v \in A \Longrightarrow$ *is-int-rat* (*md* $*$ *a v*)› **by** *force*

have $\bigwedge i.\ i < n \Longrightarrow$
$B \subseteq$ *carrier-vec n* $\Longrightarrow$
($\sum x \in B$. *rat-of-int* ((*b x* $\cdot_v$ *x*) $ *i*)) = ($\sum x \in B$. *of-int-vec* (*b x* $\cdot_v$ *x*) $ *i*)
by (*auto intro*!: *sum.cong*)
then have $\bigwedge i.\ i < n \Longrightarrow$
rat-of-int (($\bigoplus_{vmint.V} v \in B$. *b v* $\cdot_v$ *v*) $ *i*) =
($\sum x \in B$. (*of-int-vec* ∘ (λ*v*. *b v* $\cdot_v$ *v*)) *x* $ *i*)
apply (*subst vmint.finsum-index*)
using *Bcarr* **by** *auto*[*4*]
then have *finsum-prop*: $\bigwedge i.\ i < n \Longrightarrow$
rat-of-int (($\bigoplus_{vmint.V} v \in B$. *b v* $\cdot_v$ *v*) $ *i*) =
finsum vmrat.V (*of-int-vec* ∘ (λ*v*. *b v* $\cdot_v$ *v*)) *B* $ *i*
apply (*subst vmrat.finsum-index*) **using** *Bcarr* **by** *auto*

```
  have of-int-vec (vmint.lincomb b B) =
    of-int-vec (finsum vmint.V (λv. b v ·_v v) B)
    unfolding vmint.lincomb-def by auto
  also have ... = finsum vmrat.V (of-int-vec ∘ (λv. b v ·_v v)) B
    using finsum-prop vec-eq-iff by auto

  also have ... =
    finsum vmrat.V
      (λv. (md * a (of-int-vec v)) ·_v of-int-vec v) B
    unfolding b-def o-def of-int-hom.vec-hom-smult
    using all-B-in-carrier-n B-prop2
    by (auto intro!: vmrat.M.finsum-cong2)
  also have ... =
    finsum vmrat.V (λv. (md * a v) ·_v v) A
    unfolding B(3)
    apply (subst vmrat.finsum-reindex[OF * inj-of-int-vec])
    by (auto simp add: smult-smult-assoc)
  also have ... =
    vmrat.lincomb (λv. md * (a v)) A
    by (metis vmrat.lincomb-def)
  also have ... = md ·_v vmrat.lincomb a A
    using Acarr
    by (metis vmrat.lincomb-smult)
  also have ... = 0_{vmrat.V}
    by (smt (verit) av(4) module-vec-simps(2) vmrat.smult-r-null)
  finally have of-int-vec (vmint.lincomb b B) = 0_{vmrat.V} .
  then have 2: vmint.lincomb b B = 0_v n
    by (smt (verit) module-vec-simps(2) of-int-hom.vec-hom-zero-iff)

  show ?thesis
    unfolding vmint.lin-dep-def
    using B 0 1 2
    by blast
qed

lemma casting-lin-comb-helper:
  assumes dim-vecs: ⋀v. v ∈ set vecs ⟹ dim-vec v = len-vec
  assumes module.lin-indpt class-ring (module-vec TYPE(int) (len-vec)) (set vecs)
  shows ¬ (module.lin-dep class-ring (module-vec TYPE(rat) (len-vec))
     (set ((map of-int-vec vecs)::rat vec list)))
  using assms(2)
  apply (rule contrapos-nn)
  using dim-vecs apply (auto intro!: casting-lin-comb-helper-set)[1]
  by (smt (verit) carrier-vec-dim-vec)

lemma casting-lin-comb-helper-set-2:
  assumes vs: vs ⊆ carrier-vec n
  assumes ldi: module.lin-dep class-ring (module-vec TYPE(int) n) vs
  shows module.lin-dep class-ring (module-vec TYPE(rat) n)
```

```
    (of-int-vec ‘ vs)
proof −
  interpret vmint: vec-module TYPE(int) n .
  interpret vmrat: vec-module TYPE(rat) n .

  have mrat:module class-ring vmrat.V
    using vec-module by blast
  have mint:module class-ring vmint.V
    using vmint.module-axioms by auto

  from ldi
  obtain A a va where av: finite A A ⊆ vs
    a ∈ A → carrier class-ring
    vmint.lincomb a A = 0_{vmint.V}
    va ∈ A a va ≠ 0_{class-ring}
    unfolding module.lin-dep-def[OF mint]
    by blast

  define B where (B :: rat vec set) = of-int-vec ‘ A
  define b where (b::rat vec ⇒ rat) = (λv::rat vec. of-int (a (map-vec floor v)))
  define vb where (vb::rat vec) = of-int-vec va

  have [simp]: dim-vec (⨁_{vmint.V} v∈A. a v ·_v v) = n
     apply (subst vmint.finsum-dim)
    using av
    using assms(1) by auto
  have [simp]: dim-vec
     (⨁_{vmrat.V} v∈of-int-vec ‘
                  A. rat-of-int
                      (a (map-vec floor v)) ·_v v) = n
    apply (subst vmrat.finsum-dim)
    using av(2) vs by auto

  have B1: finite B
    by (simp add: B-def av(1))
  have B2: B ⊆ of-int-vec ‘ vs
    unfolding B-def
    using av by blast
  have B3-aux-helper: ⋀i. i < n ⟹ A ⊆ carrier-vec n ⟹ (⋀i x. i < n ⟹
        x ∈ A ⟹
        (rat-of-int (a (map-vec floor (of-int-vec x))) ·_v of-int-vec x) $ i =
        rat-of-int ((a x ·_v x) $ i))
    by (smt (verit, del-insts) carrier-vecD eq-vecI floor-of-int index-map-vec(1) in-
dex-map-vec(2) index-smult-vec(1) of-int-mult smult-carrier-vec vmint.summands-valid)
  have B3-aux: ⋀i. i < n ⟹
       (⨁_{vmrat.V} v∈of-int-vec ‘ A. rat-of-int (a (map-vec floor v)) ·_v v) $ i =
       rat-of-int ((⨁_{vmint.V} v∈A. a v ·_v v) $ i)
    apply (subst vmrat.finsum-index)
    apply (metis)
```

subgoal
using *assms*(*1*) *av*(*2*) **by** *force*
apply (*smt* (*verit*, *ccfv-threshold*) *B2 B-def basic-trans-rules*(*31*) *image-iff map-carrier-vec subsetI vs*)
apply (*subst vmint.finsum-index*)
subgoal by *auto*
subgoal
using *av*(*2*) *vs* **by** *auto*
subgoal
using *assms*(*1*) *av*(*2*) **by** *auto*
apply (*subst sum.reindex*)
subgoal
by (*simp add*: *inj-on-def of-int-hom.vec-hom-inj*)
using *B3-aux-helper*
by (*smt* (*verit*, *del-insts*) ‹⋀*i*. *i* < *n* ⟹ *A* ⊆ *carrier-vec n*› *o-apply of-int-sum sum.cong*) **have** *B3*: *vmrat.lincomb b B* = *of-int-vec* (*vmint.lincomb a A*)
unfolding *b-def B-def*
vmrat.lincomb-def vmint.lincomb-def
apply (*subst vec-eq-iff*)
using *B3-aux* **by** *auto*

have *B4*: *vb* ∈ *B*
unfolding *vb-def B-def*
by (*simp add*: *av*(*5*))

have *B5*: *b vb* ≠ *0*
unfolding *vb-def b-def*
using *av*(*6*)
apply *clarsimp*
by (*metis eq-vecI floor-of-int index-map-vec*(*1*) *index-map-vec*(*2*))

show *?thesis*
unfolding *vmrat.lin-dep-def*
using *B1 B2 B3 B4 B5*
by (*metis av*(*4*) *module-vec-simps*(*2*) *of-int-hom.vec-hom-zero*)
qed

lemma *casting-lin-comb-helper-2*:
assumes *dim-vecs*: ⋀*v*. *v* ∈ *set vecs* ⟹ *dim-vec v* = *len-vec*
assumes ¬ (*module.lin-dep class-ring* (*module-vec TYPE*(*rat*) (*len-vec*)) (*set* ((*map of-int-vec vecs*)::*rat vec list*)))
shows *module.lin-indpt class-ring* (*module-vec TYPE*(*int*) (*len-vec*)) (*set vecs*)
using *assms*(*2*)
apply (*rule contrapos-nn*)
using *dim-vecs* **apply** (*auto intro*!: *casting-lin-comb-helper-set-2*)[*1*]
by (*smt* (*verit*) *carrier-vec-dim-vec*)

4.3 Properties of associated polynomial

lemma *f-representation*:
shows $f = (\sum i{<}degree\ f.\ monom\ (poly.coeff\ f\ i)\ i) +$
$monom\ (lead\text{-}coeff\ f)\ (degree\ f)$
by (*metis* (*full-types*) *lessThan-Suc-atMost poly-as-sum-of-monoms sum.lessThan-Suc*)

lemma *vec-associated-to-int-poly-inverse*:
assumes $X > 0$
fixes f:: *int poly*
shows $f = int\text{-}poly\text{-}associated\text{-}to\text{-}vec\ (vec\text{-}associated\text{-}to\text{-}int\text{-}poly\ f\ X)\ X$
using *assms f-representation*
unfolding *int-poly-associated-to-vec-def vec-associated-to-poly-def*
by *auto*

lemma *int-poly-associated-to-vec-degree-helper-le*:
shows $degree\ (\sum i{\leq}n.\ monom\ (f\ i)\ i) \leq n$
apply (*intro degree-sum-le*)
apply *clarsimp*
using *degree-monom-le*
using *ge-trans* **by** *blast*

lemma *int-poly-associated-to-vec-degree-helper-lt*:
assumes $n > 0$
shows $degree\ (\sum i{<}n.\ monom\ (f\ i)\ i) < n$
apply (*cases n*)
using *assms*
by (*auto simp add*: *lessThan-Suc-atMost less-Suc-eq-le int-poly-associated-to-vec-degree-helper-le*)

lemma *int-poly-associated-to-vec-degree*:
fixes v:: *int vec*
assumes $dim\text{-}vec\ v > 0$
shows $degree\ (int\text{-}poly\text{-}associated\text{-}to\text{-}vec\ v\ X) < dim\text{-}vec\ v$
unfolding *int-poly-associated-to-vec-def*
using *assms* **by** (*auto intro*!: *int-poly-associated-to-vec-degree-helper-lt*)

lemma *degree-associated-poly*:
shows $degree\ (int\text{-}poly\text{-}associated\text{-}to\text{-}vec\ v\ X) \leq dim\text{-}vec\ v$
unfolding *int-poly-associated-to-vec-def*
apply(*cases* $dim\text{-}vec\ v = 0$)
subgoal by *auto*
by (*metis bot-nat-0.not-eq-extremum less-imp-le-nat int-poly-associated-to-vec-def*
int-poly-associated-to-vec-degree)

lemma *degree-associated-poly-lt*:
assumes $X > 0$
assumes $dim\text{-}vec\ v \geq 1$
shows $degree\ (int\text{-}poly\text{-}associated\text{-}to\text{-}vec\ v\ X) < dim\text{-}vec\ v$
using *assms*
by (*simp add*: *int-poly-associated-to-vec-degree*)

lemma *degree-of-monom-sum-list*:
fixes *ell*:: *int list*
fixes *j*:: *nat*
assumes *ell* $\neq$ []
shows *degree*
$(\sum i<$*length ell*. *monom*
(*ell* ! *i*) *i*) < *length ell*
using *assms*
proof (*induct length ell arbitrary*: *ell*)
case *0*
then show *?case* **by** *auto*
next
case (*Suc x*)
then obtain *ell1 a* **where** *ell-prop*: *ell* = *ell1*@[*a*]
by (*metis length-Suc-conv-rev*)
then have *len-ell1*: *x* = *length ell1*
using *Suc.hyps*(*2*) **by** *auto*
have $\bigwedge i$. *i* < *length ell1* $\Longrightarrow$ (*ell1* @ [*a*]) ! *i* = *ell1* ! *i*
by (*simp add*: *append-Cons-nth-left*)
then have $(\sum i<$*length ell1*. *monom* ((*ell1* @ [*a*]) ! *i*) *i*) =
$(\sum i<$*length ell1*. *monom* (*ell1* ! *i*) *i*)
by *auto*
then have *sum-is*: $(\sum i<$*length ell*. *monom* (*ell* ! *i*) *i*) =
$(\sum i<$*length ell1*. *monom* (*ell1* ! *i*) *i*) + *monom a* (*length ell* − *1*)
using *ell-prop* **by** *auto*
{assume ∗: *ell1* = []
then have *length ell* = *1*
using *ell-prop* **by** *auto*
then have *degree* $(\sum i<$*length ell*. *monom* (*ell* ! *i*) *i*)
< *length ell* **using** ∗ *sum-is*
by *auto*
} moreover {assume ∗: *ell1* $\neq$ []
have *degree* $(\sum i<$*length ell1*. *monom* (*ell1* ! *i*) *i*) < *length ell1*
using *Suc len-ell1*
using ∗ **by** *blast*
then have *lt1*: *degree* $(\sum i<$*length ell1*. *monom* (*ell1* ! *i*) *i*) < *length ell*
using *ell-prop* **by** *auto*
have *lt2*: *degree* (*monom a* (*length ell* − *1*)) < *length ell*

by (*metis Suc*(*2*) *degree-0 degree-monom-eq diff-Suc-1 lessI less-Suc-eq-0-disj monom-eq-0-iff*)
then have *degree* $(\sum i<$*length ell*. *monom* (*ell* ! *i*) *i*)
< *length ell*
using *sum-is lt1*
by (*simp add*: *degree-add-less*)
}
ultimately show *?case*
by *blast*

```
qed

lemma coeff-of-monom-sum-list:
  fixes ell:: int list
  fixes j:: nat
  assumes j < length ell
  shows coeff
        (∑ i<length ell. monom
            (ell ! i) i) j = ell ! j
  using assms
proof (induct length ell arbitrary: ell)
  case 0
  then show ?case by auto
next
  case (Suc x)
  {assume *: x = 0
    obtain a where ell = [a]
      using Suc *
      by (metis length-0-conv length-Suc-conv)
    have j = 0
      using Suc * by auto
    then have poly.coeff (∑ i<1. monom (ell ! i) i) j =
       ell ! j
      by auto
  } moreover {assume *: x > 0
    then have length ell > 1
      using Suc
      by auto
    then obtain ell1 a where ell = ell1@[a]
      by (metis Suc.hyps(2) length-nth-simps(1) nat.simps(3) rev-cases)
    then have poly.coeff (∑ i<length ell. monom (ell ! i) i) j =
       ell ! j
      using Suc degree-of-monom-sum-list
      by (simp add: coeff-sum-monom lessThan-Suc-atMost not-less-eq)
  }
  ultimately show ?case
    using Suc.hyps(2)
    by (metis One-nat-def bot-nat-0.not-eq-extremum)
qed

lemma coeff-of-monom-sum:
  fixes a:: int vec
  assumes i < dim-vec a
  shows coeff
        (∑ i<dim-vec a. monom
            (a $ i) i) i = a $ i
  using coeff-of-monom-sum-list
 by (metis (no-types, lifting)  Matrix.length-list-of-vec assms list-of-vec-index sum.cong)
```

This next lemma required showing that every vector in the lattice has

its ith component divisible by X^n (a property of the basis).

lemma *access-entry-in-int-poly-associated-to-vec*:
fixes *x i*:: *nat*
fixes *v*:: *int vec*
assumes *i* < *dim-vec v*
shows (*coeff* (*int-poly-associated-to-vec v X*) *i*) = ⌊*real-of-int* (*v* $ *i*) / *real* (*X* ^ *i*)⌋
proof −
let *?newvec* = *vec* (*dim-vec v*) (λ*j*. ⌊*real-of-int* (*v* $ *j*) / *real* (*X* ^ *j*)⌋)
have ⋀*i k*. *k* ≠ *0* ⟹ *degree* (*monom k i*) = *i*
by (*simp add*: *degree-monom-eq*)
then have *helper*: ⋀(*j*::*nat*). ⋀ (*v*::*int vec*). *j* < *dim-vec v* ⟹ *poly.coeff* (∑ *i*<*dim-vec v*. *monom* (*v* $ *i*) *i*) *j* = (*v* $ *j*)
by (*smt* (*z3*) *coeff-of-monom-sum dim-vec not-less-zero*)
have *poly.coeff* (∑ *i*<*dim-vec v*. *monom* (*?newvec* $ *i*) *i*) *i* =
⌊*real-of-int* (*v* $ *i*) / *real* (*X* ^ *i*)⌋
using *helper*[*of i ?newvec*] *assms dim-vec index-vec sum.cong*
unfolding *int-poly-associated-to-vec-def*
by *force*
then show *?thesis* **unfolding** *int-poly-associated-to-vec-def* **by** *auto*
qed

lemma *dim-vec-associated-to-int-poly-lteq*:
fixes *v*:: *int vec*
assumes *dim-vec v* ≥ *1*
assumes *X* > *0*
shows *dim-vec* (*vec-associated-to-int-poly* (*int-poly-associated-to-vec v X*) *X*) ≤ *dim-vec v*
proof −
have ⋀*i k*. *k* ≠ *0* ⟹ *degree* (*monom k i*) = *i*
by (*simp add*: *degree-monom-eq*)
then have *helper*: ⋀(*j*::*nat*). ⋀ (*v*::*int vec*). *j* < *dim-vec v* ⟹ *poly.coeff* (∑ *i*<*dim-vec v*. *monom* (*v* $ *i*) *i*) *j* = (*v* $ *j*)
by (*smt* (*z3*) *coeff-of-monom-sum dim-vec not-less-zero*)
then have *degree* (*int-poly-associated-to-vec v X*) < *dim-vec v*
unfolding *int-poly-associated-to-vec-def* **using** *assms degree-associated-poly-lt*
by (*metis* (*no-types*) *assms int-poly-associated-to-vec-def*)
then show *?thesis*
unfolding *vec-associated-to-poly-def*
by *simp*
qed

lemma *dim-vec-associated-to-int-poly-lt-imp-zeros*:
fixes *v*:: *int vec*
fixes *i*:: *nat*
assumes *entries-div*: ⋀*j*. *j* < *dim-vec v* ⟹ (∃ *k*. (*X*^*j*)∗*k* = (*v* $ *j*))
assumes *X-gt*: *X* > *0*
assumes *dim-vec* (*vec-associated-to-int-poly* (*int-poly-associated-to-vec v X*) *X*) < *dim-vec v*

assumes *i-gt*: $i \geq$ *dim-vec* (*vec-associated-to-int-poly* (*int-poly-associated-to-vec* v X) X)
assumes *i-lt*: $i <$ *dim-vec* v
shows $v \$ i = 0$
proof −
let *?associated-p* = (*int-poly-associated-to-vec* v X)
let *?associated-v* = *vec-associated-to-int-poly* (*int-poly-associated-to-vec* v X) X
have $\bigwedge i\ k.\ k \neq 0 \Longrightarrow$ *degree* (*monom* k i) = i
by (*simp add*: *degree-monom-eq*)
then have *helper1*: $\bigwedge$(*j*::*nat*). $\bigwedge$ (*v*::*int vec*). $j <$ *dim-vec* $v \Longrightarrow$ *poly.coeff* ($\sum i$<*dim-vec v*. *monom* ($v \$ i$) i) $j = (v \$ j)$
by (*smt* (*z3*) *coeff-of-monom-sum dim-vec not-less-zero*)
have *helper2*: *poly.coeff ?associated-p i* = 0
using *i-gt*
by (*simp add*: *coeff-eq-0*)
have *coeff-is*: *poly.coeff ?associated-p i* = *floor* ((($v \$ i$)::*real*)/($X$^$i$))
unfolding *int-poly-associated-to-vec-def* **using** *i-lt helper1*
using *access-entry-in-int-poly-associated-to-vec int-poly-associated-to-vec-def* **by** *force*
then have $v\$i \neq 0 \Longrightarrow$ *floor* ((($v \$ i$)::*real*)/($X$^$i$)) ≠ 0
using *entries-div i-lt X-gt*
by (*smt* (*verit*) *divide-cancel-right divide-divide-eq-left divide-eq-0-iff mult-divide-mult-cancel-left of-int-eq-0-iff of-int-floor-cancel of-int-mult of-int-of-nat-eq of-nat-0-eq-iff of-nat-eq-of-nat-power-cancel-iff*)
then show *?thesis* **using** *coeff-is helper2* **by** *metis*
qed

lemma *int-poly-associated-to-notNil-vec-same-norm-upto*:
assumes *dim-vec* $v \geq 1$
assumes *X-gt*: $X > 0$
assumes *entries-div*: $\bigwedge j.\ j <$ *dim-vec* $v \Longrightarrow (\exists k.\ (X\hat{}j)*k = (v \$ j))$
assumes w = (*vec-associated-to-int-poly* (*int-poly-associated-to-vec* v X) X)
shows ($\sum i$<(*dim-vec v*). ($v\$i$)^2) = ($\sum i$<(*dim-vec w*). ($v\$i$)^2)
proof −
{assume ∗: *dim-vec w* = *dim-vec v*
then have ($\sum i$<(*dim-vec v*). ($v\$i$)^2) = ($\sum i$<(*dim-vec w*). ($v\$i$)^2)
by *auto*
} moreover {assume ∗:*dim-vec w* < *dim-vec v*
then have *dim-v-at-least-1*: *dim-vec v* ≥ 1
by *auto*
have $\bigwedge i$. *dim-vec* (*vec-associated-to-int-poly* (*int-poly-associated-to-vec* v X) X) ≤ $i \Longrightarrow$
$i <$ *dim-vec* $v \Longrightarrow v \$ i = 0$
using ∗ *dim-vec-associated-to-int-poly-lt-imp-zeros*[*OF entries-div X-gt*] *assms*
by *blast*
then have *i-zero-prop*: $\bigwedge i.\ i \in \{$*dim-vec w*..<(*dim-vec v*)$\} \Longrightarrow (v\$i)\hat{}2 = 0$
by (*metis assms*(*4*) *atLeastLessThan-iff zero-power2*)
then have ($\sum i$<(*dim-vec v*). ($v\$i$)^2) = ($\sum i$<(*dim-vec w*). ($v\$i$)^2)
proof −
have *finset1*: *finite* {*x*. *x* < *dim-vec w*}

```
    by auto
  have finset2: finite ({x. (dim-vec w) ≤ x} ∩ {x. x < dim-vec v})
    by blast
  have empty-inter: {x. x < dim-vec w} ∩ ({x. (dim-vec w) ≤ x} ∩ {x. x <
dim-vec v}) = {}
    by auto
  have {x. x < dim-vec v} = {x. x < dim-vec w} ∪ ({x. (dim-vec w) ≤ x} ∩
{x. x < dim-vec v})
    using *
    by auto
 then have (∑ i<(dim-vec v). (v$i)^2) = (∑ i<(dim-vec w). (v$i)^2) + (∑ i
∈ {dim-vec w..<(dim-vec v)}. (v$i)^2)
  using * empty-inter unfolding atLeastLessThan-def atLeast-def lessThan-def

    using sum-Un-nat[OF finset1 finset2]
    by (metis (no-types, lifting) finset1 finset2 sum.union-disjoint)
  then show ?thesis
  using assms i-zero-prop
  by auto
 qed
 }
 ultimately show ?thesis
  using dim-vec-associated-to-int-poly-lteq assms
  by (metis One-nat-def le-neq-implies-less)
 qed

lemma int-poly-associated-to-notNil-vec-same-entries:
 fixes i:: nat
 fixes v w:: int vec
 assumes i-lt: i < dim-vec w
 assumes dim-vec v ≥ 1
 assumes X-gt: X > 0
 assumes entries-div: ⋀j. j < dim-vec v ⟹ (∃ k. (X^j)*k = (v $ j))
 assumes w-is: w = (vec-associated-to-int-poly (int-poly-associated-to-vec v X) X)
 shows ((v $ i) = (w $ i))
proof −
 let ?p = int-poly-associated-to-vec v X
 have w$i = (coeff ?p i)*X^i
  using i-lt w-is unfolding int-poly-associated-to-vec-def
  using index-vec vec-associated-to-poly-def
  by (smt (verit, del-insts) dim-vec-vec-associated-to-poly semiring-1-class.of-nat-power)
 then show ?thesis
  using assms unfolding int-poly-associated-to-vec-def
  by (smt (verit) access-entry-in-int-poly-associated-to-vec assms(5) bot-nat-0.not-eq-extremum
dim-vec-associated-to-int-poly-lteq floor-divide-of-int-eq mult-of-nat-commute nat-SN.compat
nonzero-mult-div-cancel-left of-int-of-nat-eq of-nat-eq-iff semiring-1-class.of-nat-power
semiring-1-class.of-nat-simps(1) semiring-1-no-zero-divisors-class.power-not-zero vec-associated-to-int-poly-int
qed
```

lemma *int-poly-associated-to-nil-vec-same-norm*:
assumes $X > 0$
assumes *dim-vec* $v = 0$
shows *euclidean-norm-int-vec* v = *euclidean-norm-int-vec* (*vec-associated-to-int-poly* (*int-poly-associated-to-vec* v X) X)
proof −
have *v-is-nil*: v = *vNil*
using *assms*
by *simp*
then have *h1*: *euclidean-norm-int-vec* v= 0
unfolding *euclidean-norm-int-vec-eq* **by** *auto*
have *int-poly-associated-to-vec* v X = 0
using *v-is-nil* **unfolding** *int-poly-associated-to-vec-def* **by** *simp*
then have *vec-associated-to-int-poly* (*int-poly-associated-to-vec* v X) X = *vec-of-list* [0]
unfolding *vec-associated-to-poly-def* **by** *auto*
then have *h2*: *euclidean-norm-int-vec* (*vec-associated-to-int-poly* (*int-poly-associated-to-vec* v X) X) = 0
unfolding *euclidean-norm-int-vec-eq* **by** *auto*
show *?thesis*
using *h1 h2* **by** *auto*
qed

lemma *int-poly-associated-to-notNil-vec-same-norm*:
assumes *X-gt*: $X > 0$
assumes *dim-vec* $v > 0$
assumes $\bigwedge j.\ j <$ *dim-vec* $v \Longrightarrow (\exists k.\ (X\hat{}j)*k = (v\ \$\ j))$
shows *euclidean-norm-int-vec* v = *euclidean-norm-int-vec* (*vec-associated-to-int-poly* (*int-poly-associated-to-vec* v X) X)
proof −
let *?w* = *vec-associated-to-int-poly* (*int-poly-associated-to-vec* v X) X
have *h1*: $(\sum i<$(*dim-vec* v)$.\ (v\$i)\hat{}2) = (\sum i<$(*dim-vec ?w*)$.\ (v\$i)\hat{}2)$
using *assms int-poly-associated-to-notNil-vec-same-norm-upto*
by *simp*
then have *h2*: $\bigwedge i.\ i <$ *dim-vec ?w* $\Longrightarrow ((v\ \$\ i) = (?w\ \$\ i))$
using *int-poly-associated-to-notNil-vec-same-entries assms*
by (*metis One-nat-def Suc-leI*)
then have $(\sum i<$(*dim-vec* v)$.\ (v\$i)\hat{}2) = (\sum i<$(*dim-vec ?w*)$.\ (?w\$i)\hat{}2)$ **using** *h1 h2*
by *auto*
then show *?thesis*
using *assms euclidean-norm-int-vec-eq*
by *presburger*
qed

lemma *int-poly-associated-to-vec-same-norm*:
assumes $X > 0$
assumes $\bigwedge j.\ j <$ *dim-vec* $v \Longrightarrow (\exists k.\ (X\hat{}j)*k = (v\ \$\ j))$
shows *euclidean-norm-int-vec* v = *euclidean-norm-int-vec* (*vec-associated-to-int-poly*

(*int-poly-associated-to-vec v X*) *X*)
using *assms int-poly-associated-to-nil-vec-same-norm int-poly-associated-to-notNil-vec-same-norm*
by (*metis bot-nat-0.not-eq-extremum*)

lemma *int-poly-associated-to-vec-sum*:
assumes *dim-vec a = dim-vec b*
assumes $\bigwedge i.\ i < dim\text{-}vec\ a \Longrightarrow$
a $ i mod X ^ i = 0
assumes $\bigwedge i.\ i < dim\text{-}vec\ b \Longrightarrow$
b $ i mod X ^ i = 0
assumes *X > 0*
shows *int-poly-associated-to-vec (a+b) X = int-poly-associated-to-vec a X + int-poly-associated-to-vec b X*
proof −
let *?newvec-sum = vec (dim-vec (a + b)) (λj. ⌊real-of-int ((a + b) $ j) / real (X ^ j)⌋)*
let *?newvec-a = vec (dim-vec a) (λj. ⌊real-of-int (a $ j) / real (X ^ j)⌋)*
let *?newvec-b = vec (dim-vec b) (λj. ⌊real-of-int (b $ j) / real (X ^ j)⌋)*
have *same-dim*: *dim-vec (a + b) = dim-vec a*
using *assms(1)*
by *auto*
have $\bigwedge j.\ j < dim\text{-}vec\ a \Longrightarrow$ *⌊real-of-int ((a + b) $ j) / real (X ^ j)⌋ = ⌊real-of-int (a $ j) / real (X ^ j)⌋ + ⌊real-of-int (b $ j) / real (X ^ j)⌋*
proof − **fix** *j*
assume *j-lt*: *j < dim-vec a*
then obtain *k1 k2*::*int* **where** *k1k2-prop*: *a $ j = k1∗X^j b $ j = k2∗X^j*
using *assms(2)[of j] assms(3)[of j] assms(1)* **by** (*auto elim*!: *dvdE*) *blast*
have *roi*: *⌊real-of-int (a $ j) / real (X ^ j)⌋ = k1 ⌊real-of-int (b $ j) / real (X ^ j)⌋ = k2*
using *assms(4) k1k2-prop(1)* **apply** *simp*
using *assms(4) k1k2-prop(2)* **by** *simp*
have *(a + b) $ j = k1∗X^j + k2∗X^j*
using *k1k2-prop assms(1) j-lt* **by** *auto*
then show *⌊real-of-int ((a + b) $ j) / real (X ^ j)⌋ = ⌊real-of-int (a $ j) / real (X ^ j)⌋ + ⌊real-of-int (b $ j) / real (X ^ j)⌋*
using *k1k2-prop roi assms(4)*
using *div-mult-self3 floor-divide-real-eq-div floor-of-int of-int-of-nat-eq of-nat-0-eq-iff of-nat-0-le-iff power-eq-0-iff zero-less-iff-neq-zero* **by** *auto*
qed
then have $\bigwedge j.\ j < dim\text{-}vec\ a \Longrightarrow$ *monom (?newvec-sum $ j) j = monom (?newvec-a $ j) j + monom (?newvec-b $ j) j*
by (*simp add*: *add-monom assms(1)*)
then have $(\sum i{<}dim\text{-}vec\ a.\ monom\ (?newvec\text{-}sum\ \$\ i)\ i) = (\sum i{<}dim\text{-}vec\ a.\ monom\ (?newvec\text{-}a\ \$\ i)\ i) + (\sum i{<}dim\text{-}vec\ a.\ monom\ (?newvec\text{-}b\ \$\ i)\ i)$
by *auto*
then show *?thesis*
unfolding *int-poly-associated-to-vec-def* **using** *assms(1) same-dim* **by** *algebra*
qed

lemma *int-poly-associated-to-vec-constant-mult*:
fixes *c*:: *nat* ⇒ *int*
fixes *v*:: *int vec*
assumes *X* > *0*
assumes *div-by-X*: ⋀*i*. *i* < *dim-vec v* ⟹
v $ *i mod X* ^ *i* = *0*
shows *int-poly-associated-to-vec* (*c i* ·$_v$ *v*) *X* =
smult (*c i*) (*int-poly-associated-to-vec v X*)
proof −
let *?newvec* = *vec* (*dim-vec v*) (λ*j*. ⌊*real-of-int* (*v* $ *j*) / *real* (*X* ^ *j*)⌋)
let *?newvec-mult* = *vec* (*dim-vec* (*c i* ·$_v$ *v*)) (λ*j*. ⌊*real-of-int* ((*c i* ·$_v$ *v*) $ *j*) / *real* (*X* ^ *j*)⌋)
have *same-dim*: *dim-vec ?newvec* = *dim-vec ?newvec-mult*
by *auto*
then have (*c i*) ∗ *?newvec* $ *j* = (*?newvec-mult* $ *j*) **if** *j-prop*: *j* < *dim-vec v* **for** *j*
using *div-by-X* [*of j*] ‹*0* < *X*› *that*
by (*simp only*: *floor-divide-of-int-eq flip*: *Power.of-nat-power of-int-of-nat-eq* [**where** *?'a* = *real*])
auto
then have *smult* (*c i*) (*monom* (*?newvec* $ *j*) *j*) = *monom* (*?newvec-mult* $ *j*) *j* **if** *j-prop*: *j* < *dim-vec v* **for** *j*
by (*simp add*: *smult-monom j-prop*)
then have (∑*j*<*dim-vec v*. *monom* (*?newvec-mult* $ *j*) *j*) = (∑*j*<*dim-vec v*. *smult* (*c i*) (*monom* (*?newvec* $ *j*) *j*))
by *auto*
then have (∑*j*<*dim-vec v*. *monom* (*?newvec-mult* $ *j*) *j*) = *smult* (*c i*) (∑*j*<*dim-vec v*. *monom* (*?newvec* $ *j*) *j*)
by (*simp add*: *smult-sum2*)
then show *?thesis*
unfolding *int-poly-associated-to-vec-def* **by** *auto*
qed

4.4 Generic Coppersmith assumptions locale

The generic properties required are stored in this locale.

locale *coppersmith-assms* = *LLL-with-assms* +
fixes *x0* :: *int*
fixes *M X* :: *nat*
fixes *F* :: *int poly*
assumes *M*: *M* > *0*
assumes *X*: *X* > *0*
assumes *n*: *n* > *0*
assumes *x0*: *poly F x0 mod M* = *0* |*x0*| ≤ *int X*
assumes *lfs*: *length fs-init* ≠ *0*
assumes *Xpoly*: ⋀*v j*.
v ∈ *set fs-init* ⟹ *j* < *n* ⟹
v $ *j mod int X* ^ *j* = *0*
assumes *rt*: ⋀*v*.

```
    v ∈ set fs-init ⟹
    poly (int-poly-associated-to-vec v X) x0 mod M = 0
begin

lemma sumlist-mod:
  assumes ⋀v. v ∈ set xs ⟹ dim-vec v = n
  assumes m0: ⋀v j.
    v ∈ set xs ⟹ j < n ⟹
    v $ j mod int X ^ j = 0
  assumes i < n
  shows M.sumlist xs $ i mod X ^ i = 0
proof −
  from sumlist-nth
  have M.sumlist xs $ i =
    (∑ j = 0..<length xs. xs ! j $ i)
    by (metis assms(1,3))

  moreover have ... mod int (X ^ i) =
    (∑ j = 0..<length xs.
      xs ! j $ i  mod int X ^ i) mod
    int X ^ i
    apply (subst mod-sum-eq[symmetric])
    by (simp add: mod-mult-left-eq)
  moreover have ... = 0
    using m0
    by (simp add: assms(3))
  ultimately show ?thesis
    by auto
qed

lemma poly-associated-to-vec-sumlist:
  assumes ⋀v. v ∈ set xs ⟹ dim-vec v = n
  assumes ⋀v j. v ∈ set xs ⟹
      j < dim-vec v ⟹ v $ j mod X ^ j = 0
  shows int-poly-associated-to-vec (sumlist xs) X =
    (∑ i < length xs. int-poly-associated-to-vec (xs ! i) X)
  using assms
proof (induction xs)
  case Nil
  then show ?case
    unfolding int-poly-associated-to-vec-def
    by auto
next
  case ih: (Cons x xs)
  have int-poly-associated-to-vec (M.sumlist (x # xs)) X =
    int-poly-associated-to-vec (x + M.sumlist xs) X
    by auto
  also have ... =  int-poly-associated-to-vec x X +
     int-poly-associated-to-vec (M.sumlist xs) X
```

```
    apply (intro int-poly-associated-to-vec-sum[OF - - - X])
    using ih.prems(1) dim-sumlist apply clarsimp
    using ih.prems(2) apply simp
    using dim-sumlist ih(2) ih.prems(2) sumlist-mod by auto
  also have ... =  int-poly-associated-to-vec x X +
     (∑ i<length xs. int-poly-associated-to-vec (xs ! i) X)
    apply (subst ih(1))
    using ih by auto
  also have ... =
   (∑ i ∈ {0..<Suc (length xs)}.
    int-poly-associated-to-vec ((x # xs) ! i) X)
    apply (subst  sum.atLeast0-lessThan-Suc-shift)
    using atLeast-upt set-upt
    by auto
  finally show ?case
    using atLeast0LessThan by fastforce
qed

lemma short-vector-inherit-props:
  shows ⋀j. j < n ⟹ short-vector $ j mod X ^ j = 0
    poly (int-poly-associated-to-vec short-vector X) x0 mod M = 0
proof −
  have ∗: set fs-init ⊆ carrier-vec n
    by (simp add: fs-init)
  have svin: short-vector ∈ lattice-of fs-init
    using L-def lfs len short-vector(2) short-vector-impl by auto
  from in-latticeE[OF this]
  obtain c where
    c: short-vector =
        M.sumlist
         (map (λi. of-int (c i) ·v fs-init ! i)
           [0..<length fs-init]) .

  {
    fix i j
    assume i < m j < n
    then have fs-init ! i $ j mod int X ^ j = 0
      using Xpoly len nth-mem by blast

    moreover have (c i ·v fs-init ! i) $ j = c i ∗ fs-init ! i $ j
      apply (subst index-smult-vec)
      apply (metis LLL-inv-wD(4) LLL-inv-wI L-def ‹i < m› ‹j < n› carrier-vecD
fs-init len lin-dep)
      by auto
    ultimately have (c i ·v fs-init ! i) $ j mod int X ^ j = 0
      by force
  }
  then have m0: i < m ⟹ j < n ⟹ (c i ·v fs-init ! i) $ j mod int X ^ j = 0
for i j
```

```
    by auto

  {
    fix j
    assume j: j < n
    show short-vector $ j mod int (X ^ j) = 0
      unfolding c
      apply (intro sumlist-mod[OF - - j])
      subgoal apply clarsimp
     by (metis LLL-invD(7) LLL-inv-wD(4) LLL-inv-wI LLL-with-assms.LLL-inv-initial-state
LLL-with-assms-axioms carrier-vecD fs-init len lin-dep)
      using m0 len by auto
  }
  have 1: (∑ i<m.
        int-poly-associated-to-vec (c i ·v fs-init ! i) X) =
    (∑ i<m.
        smult (c i) (int-poly-associated-to-vec (fs-init ! i) X))
    apply (intro sum.cong)
     apply clarsimp
    apply (subst int-poly-associated-to-vec-constant-mult[OF X])
    using len nth-mem
    apply (auto intro!: Xpoly)[2]
    by (metis LLL-inv-wD(4) LLL-inv-wI L-def carrier-vecD fs-init lin-dep)

  then have 2: poly ... x0 =
    (∑ i<m. c i * poly (int-poly-associated-to-vec (fs-init ! i) X) x0)
    apply (subst poly-hom.hom-sum)
    by auto
  then have ... mod M =
    (∑ i<m. (c i * poly (int-poly-associated-to-vec (fs-init ! i) X) x0) mod M)  mod
M
    by (auto simp add: mod-sum-eq)
  also have ... =
    (∑ i<m. (0::int)) mod M
    apply (intro arg-cong[where f = λi::int. i mod M])
    apply (intro sum.cong)
    using len nth-mem
    using rt by (auto simp add: pull-mods(10))
  also have ... = 0 by auto
  finally show poly
     (int-poly-associated-to-vec local.short-vector X)
     x0 mod M = 0
    unfolding c
    apply (subst poly-associated-to-vec-sumlist)
    subgoal
      using LLL-inv-wD(4) LLL-inv-wI L-def fs-init len lin-dep by auto
    subgoal
      using m0 apply clarsimp
        by (metis LLL.LLL-inv-wD(4) LLL-inv-wI L-def carrier-vecD fs-init in-
```

```
dex-smult-vec(1) len lin-dep)
    using 1 2 len by auto
qed

lemma root-poly-short-vector:
  assumes bnd: real-of-int ‖short-vector‖² < M^2 / n
  shows poly (int-poly-associated-to-vec short-vector X) x0 = 0
proof −
  have dimsv: dim-vec short-vector = n
    using lfs carrier-dim-vec len short-vector(1) short-vector-impl by blast

  have ∗: ⋀j. j < dim-vec short-vector ⟹
        ∃ k. int X ^ j ∗ k = short-vector $ j
    unfolding dimsv
    apply (drule  short-vector-inherit-props(1))
    by auto

  have db: 1 +
     real (degree (int-poly-associated-to-vec short-vector X)) ≤ n
    by (smt (verit, best) dimsv n nat-less-real-le int-poly-associated-to-vec-degree)

  have sqrt (real-of-int ‖short-vector‖²) <
    sqrt (real M^2 / (real n))
    using bnd unfolding real-sqrt-le-iff by auto

  then have sqrt (real-of-int ‖short-vector‖²) <
    M / sqrt (real n)
    by (simp add: real-sqrt-divide)

  also have ... ≤
    real M / sqrt
      (real (degree(int-poly-associated-to-vec short-vector X) + 1))
    using db
    by (clarsimp simp add: field-simps M )

  finally have ∗∗:
    sqrt (real-of-int ‖short-vector‖²) < real M / sqrt
      (real
        (degree
          (int-poly-associated-to-vec short-vector X) + 1)) by auto

  show ?thesis
     apply (intro Howgrave-Graham-int-poly[OF M short-vector-inherit-props(2)
x0(2)])
    apply (subst int-poly-associated-to-vec-same-norm[OF X, symmetric, OF ∗])
    using ∗∗
    by auto
qed
```

lemma *bnd-raw-imp-short-vec-bound*:
assumes *bnd-raw*:
(*real-of-rat* α) ^ (*m* * (*m* − *1*) *div 2*) *
real-of-int (*gs.Gramian-determinant fs-init m*) *
(*real n*) ^ *m* <
M ^ (*2* * *m*)
shows *real-of-int* $\|$*short-vector*$\|^2$ < *M*^*2* / *n*
proof −

have *LLLinv*: *LLL-invariant True m reduce-basis*
using *reduce-basis-inv* **by** *auto*

have *i1*: *distinct reduce-basis*
by (*metis* (*no-types*, *lifting*) *LLL-invD*(*1*) *LLL-invD*(*2*) *LLL-invD*(*6*) *LLLinv distinct-conv-nth gs.lin-indpt-list-def nth-map*)

from *LLL-invD*[*OF LLLinv*]
have *gs.lin-indpt-list*
(*map of-int-vec reduce-basis*) **by** *blast*

then have *module.lin-indpt class-ring*
(*module-vec TYPE*(*rat*) *n*)
(*set* (*map of-int-vec reduce-basis*))
unfolding *gs.lin-indpt-list-def*
by *blast*

then have *i2*: *module.lin-indpt class-ring*
(*module-vec TYPE*(*int*) *n*)
(*set reduce-basis*)
using *casting-lin-comb-helper-2*
by (*metis* ‹*set local.reduce-basis* ⊆ *carrier-vec n*› *carrier-vecD subset-code*(*1*))

have *i3*: *set reduce-basis* ⊆ *carrier-vec n*
using *LLL.LLL-invD*(*3*) *LLLinv* **by** *blast*
have *i4*:*set fs-init* ⊆ *carrier-vec n*
using *fs-init* **by** *blast*
have *i5*:*length reduce-basis* = *length fs-init*
by (*simp add*: ‹*length local.reduce-basis* = *m*› *len*)

have *i6*: *vec-module.lattice-of n reduce-basis* =
vec-module.lattice-of n fs-init
using *LLL-invD*(*7*) *LLLinv L-def* **by** *blast*

from *ring-char-0-vec-module.lattice-of-eq-gram-det-rows-eq*[*OF i1 i2 i3 i4 i5 i6*]
have *det* (*mat-of-rows n reduce-basis* *
(*mat-of-rows n reduce-basis*)^T) =
det (*mat-of-rows n fs-init* * (*mat-of-rows n fs-init*)^T) **.**

```
  have deteq: gs.Gramian-determinant reduce-basis m = gs.Gramian-determinant
fs-init m
    unfolding gs.Gramian-determinant-def
    apply (subst  gs.Gramian-matrix-alt-def)
    using ‹length local.reduce-basis = m› apply linarith
    apply (subst  gs.Gramian-matrix-alt-def)
    using len apply blast
   by (metis ‹det (mat-of-rows n local.reduce-basis * (mat-of-rows n local.reduce-basis)^T)
= det (mat-of-rows n fs-init * (mat-of-rows n fs-init)^T)› ‹length local.reduce-basis
= m› ‹m ≤ m› i5 take-all)

  have bnd:
  (real-of-rat α) ^ (m * (m - 1) div 2) *
      real-of-int (gs.Gramian-determinant fs-init m) <
  (M^2 / (real n)) ^ m
  proof -
    have m-gt: m > 0
      using len lfs by blast
    from bnd-raw
    have real-of-rat α ^ (m * (m - 1) div 2) *
    real-of-int (gs.Gramian-determinant fs-init m)
    < real (M ^ (2 * m)) / (real n) ^ m
      using m-gt n by (auto simp add: field-simps)
    also have ... = (M^2 / (real n)) ^ m
      unfolding power-mult
      by (auto simp add: field-simps)
    finally show ?thesis .
  qed

  have rat-of-int ‖short-vector‖^2 ^ m
      ≤ α ^ (m * (m - 1) div 2) *
      rat-of-int (gs.Gramian-determinant reduce-basis m)
   using LLL.LLL-invD(6) LLL-inv-initial-state LLL-with-assms.short-vector-det-bound
LLL-with-assms-axioms lfs by blast

  have
    rat-of-int ‖short-vector‖^2 ^ m
      ≤ α ^ (m * (m - 1) div 2) *
      rat-of-int (gs.Gramian-determinant reduce-basis m)
   using LLL.LLL-invD(6) LLL-inv-initial-state LLL-with-assms.short-vector-det-bound
LLL-with-assms-axioms lfs by blast
  then have (real-of-int ‖short-vector‖^2) ^ m <
    (M^2 / (real n)) ^ m
    using bnd
      by (smt (verit, best) deteq of-rat-hom.hom-power of-rat-less-eq of-rat-mult
of-rat-of-int-eq)

  thus ?thesis
    using power-mono-iff n
```

by (*smt* (*verit*, *best*) *LLL-inv-initial-state LLL-invariant-def M Num.of-nat-simps*(*1*) *lfs length-0-conv length-greater-0-conv linorder-not-le nat-le-real-less nat-zero-less-power-iff zero-compare-simps*(*7*))
qed

end

end

5 Proof of Lightweight Algorithm

We start by proving the "lightweight algorithm" as a stepping stone to the full algorithm (proved in Coppersmith.thy).

theory *Towards-Coppersmith*

imports *Coppersmith-Generic*
begin

5.1 Basic properties

lemma *dim-form-basis-helper*:
shows *length* (*form-basis-helper p M X d*) = *d* + *1*
unfolding *form-basis-helper-def*
by *auto*

lemma *dim-form-basis*:
shows *length* (*form-basis p M X d*) = *d* + *2*
using *dim-form-basis-helper form-basis-def* **by** *simp*

5.2 Matrix properties

5.2.1 Basic properties and preliminaries

lemma *dim-row-basis-mat*:
assumes *degree p* ≥ *1*
shows *dim-row* (*vec-list-to-square-mat* (*form-basis p M X* (*degree p* − *1*))) = *degree p* + *1*
proof −
have *length* (*form-basis p M X* (*degree p* − *1*)) = *degree p* + *1*
using *assms dim-form-basis*[*of p M X degree p* − *1*]
by *auto*
then show *?thesis*
unfolding *vec-list-to-square-mat-def mat-of-rows-list-def* **by** *auto*
qed

lemma *dim-col-basis-mat*:
assumes *degree p* ≥ *1*
shows*dim-col* (*vec-list-to-square-mat* (*form-basis p M X* (*degree p* − *1*))) = (*degree p* + *1*)

proof −
have *length* (*form-basis p M X* (*degree p* − *1*)) = *degree p* + *1*
using *assms dim-form-basis*[*of p M X degree p* − *1*]
by *auto*
then show *?thesis* **unfolding** *vec-list-to-square-mat-def*
by *auto*
qed

lemma *matrix-carrier*:
assumes *degree p* ≥ *1*
assumes *i* < *degree p* + *1*
shows *vec-list-to-square-mat* (*form-basis p M X* (*degree p* − *1*)) ∈ *carrier-mat* (*degree p* + *1*) (*degree p* + *1*)
using *dim-row-basis-mat*[*of p M X*] *dim-col-basis-mat*[*of p M X*] *assms*
by *auto*

lemma *vector-sum-monom*:
fixes *v*:: *int vec*
fixes *d i*:: *nat*
assumes *d* = *dim-vec v*
assumes *d* > *0*
assumes *i-lt*: *i* < *d*
assumes *zero-monom*: ⋀*j*::*nat*. *j* < *d* ⟹ *j* ≠ *i* ⟹ *monom* (*v* $ *j*) *j* = *0*
shows (∑ *i*< *d*. *monom* (*v* $ *i*) *i*) = *monom* (*v* $ *i*) *i*
proof −
have (∑ *i*< *d*. *monom* (*v* $ *i*) *i*) = (∑ *i*∈{*0*..<*d*}. *monom* (*v* $ *i*) *i*)
using *atLeast-upt set-upt* **by** *presburger*
then have *sum-split*: (∑ *i*< *d*. *monom* (*v* $ *i*) *i*) = (∑ *i*∈{*0*..<*d*}−{*i*}. *monom* (*v* $ *i*) *i*) + *monom* (*v* $ *i*) *i*
using *i-lt*
by (*metis* (*no-types*, *lifting*) *Groups.add-ac*(*2*) *atLeast0LessThan finite-lessThan lessThan-iff sum.remove*)
have ⋀*j*::*nat*. *j* ∈ {*0*..<*d*}−{*i*} ⟹ *monom* (*v* $ *j*) *j* = *0*
using *zero-monom i-lt* **by** *simp*
then have (∑ *i*∈{*0*..<*d*}−{*i*}. *monom* (*v* $ *i*) *i*) = (∑ *i*∈{*0*..<*d*}−{*i*}. *0*)
by (*simp add*: *sum.neutral*)
then show *?thesis* **using** *sum-split* **by** *simp*
qed

lemma *int-poly-associated-to-g-i-vec*:
assumes *X-gt*: *X* > *0*
assumes *M-gt*: *M* > *0*
assumes *i-lt*: *i* ≤ (*degree p*)
shows *int-poly-associated-to-vec* (*g-i-vec M X i* (*degree p* + *1*)) *X* = *monom M i*
proof −
let *?gi* = *g-i-vec M X i* (*degree p* + *1*)
let *?d* = *degree p*
let *?newvec* = *vec* (*dim-vec* (*g-i-vec M X i* (*degree p* + *1*)))
(λ*j*. ⌊*real-of-int* (*g-i-vec M X i* (*degree p* + *1*) $ *j*) / *real* (*X* ^ *j*)⌋)

have *dim-vec-gi*: *dim-vec ?gi = degree p + 1*
unfolding *g-i-vec-def* **by** *simp*
then have *dim-vec-newvec*: *dim-vec ?newvec = ?d + 1*
by *simp*
then have *newvec-j*: $\bigwedge j.\ j < ?d+1 \Longrightarrow j \neq i \Longrightarrow ?newvec\ \$\ j = 0$
unfolding *g-i-vec-def* **by** *simp*
then have *monom-zero*: $\bigwedge j.\ j < ?d+1 \Longrightarrow j \neq i \Longrightarrow monom\ (?newvec\ \$\ j)\ j = 0$
by *blast*
have *poly-monom-sum*: $(\sum i{<}\ ?d+1.$
$monom\ (?newvec\ \$\ i)\ i) = monom\ (?newvec\ \$\ i)\ i$
using *vector-sum-monom*[*of ?d+1 ?newvec i*] *dim-vec-newvec monom-zero assms*(*3*)
by *simp*
have *newvec-i*: *?newvec $ i = M*
using *X-gt M-gt i-lt dim-vec-newvec* **unfolding** *g-i-vec-def* **by** *simp*
then have *monom-i*: *monom M i = monom (?newvec $ i) i*
by *auto*
show *?thesis*
using *monom-i poly-monom-sum dim-vec-newvec*
unfolding *int-poly-associated-to-vec-def* **by** *auto*
qed

lemma *ith-row-form-basis*:
shows $i \leq d \Longrightarrow ((form\text{-}basis\ p\ M\ X\ d)!i) = (form\text{-}basis\text{-}helper\ p\ M\ X\ d)!i$
$((form\text{-}basis\ p\ M\ X\ d)!(d+1)) = vec\text{-}associated\text{-}to\text{-}int\text{-}poly\ p\ X$
using *dim-form-basis-helper*[*of p M X d*] *form-basis-def*
apply (*simp add*: *append-Cons-nth-left*)
using *dim-form-basis-helper*[*of p M X d*] *form-basis-def*
by (*simp add*: *append-Cons-nth*(*1*) *append-Cons-nth-left*)

lemma *set-form-basis*:
shows $x \in set\ (form\text{-}basis\ p\ M\ X\ (degree\ p - 1)) \Longrightarrow x = vec\text{-}associated\text{-}to\text{-}int\text{-}poly\ p\ X \vee$
$x \in set\ (form\text{-}basis\text{-}helper\ p\ M\ X\ (degree\ p - 1))$
using *ith-row-form-basis ith-row-form-basis-helper dim-form-basis form-basis-def*
by *simp*

lemma *dim-vector-in-basis*:
fixes *i*:: *nat*
assumes *i < d + 2*
shows *dim-vec ((form-basis p M X d) ! i) = (degree p+1)*
proof −
let *?vec = (form-basis p M X d) ! i*
have *eo*: *List.member (form-basis-helper p M X d) ?vec ∨ ?vec = vec-associated-to-int-poly p X*
using *assms*
by (*metis List.member-def add-Suc-right dim-form-basis-helper form-basis-def*

less-Suc-eq nat-1-add-1 nth-append nth-append-length nth-mem plus-1-eq-Suc)
have *len-helper*: *length* (*form-basis-helper p M X d*) = *d* + *1*
using *dim-form-basis form-basis-def* **by** *simp*
have *h1*: ⋀*x*. *List.member* (*form-basis-helper p M X d*) *x* ⟹ *dim-vec x* = *degree p* + *1*
proof −
fix *x*
assume *List.member* (*form-basis-helper p M X d*) *x*
then have ∃ *k*. *x* = *g-i-vec M X k* (*degree p* + *1*)
using *ith-row-form-basis-helper*
by (*metis List.member-def dim-form-basis-helper in-set-conv-nth le-simps*(*2*) *semiring-norm*(*174*))
then show *dim-vec x* = *degree p* + *1*
using *assms* **unfolding** *g-i-vec-def* **by** *auto*
qed
have *h2*: *dim-vec* (*vec-associated-to-int-poly p X*) = *degree p* + *1*
using *assms dim-vec-vec-associated-to-poly*
by *auto*
show *?thesis*
using *h1 h2 eo* **by** *auto*
qed

5.2.2 Properties of matrix associated to input

lemma *matrix-row-form-basis-carrier*:
assumes *degree p* ≥ *1*
assumes *i* < *degree p* + *1*
shows *form-basis p M X* (*degree p* − *1*) ! *i* ∈ *carrier-vec* (*degree p* + *1*)
proof −
show *?thesis*
using *assms dim-vector-in-basis*[*of i degree p* − *1 p M X*]
unfolding *carrier-vec-def* **by** *auto*
qed

lemma *matrix-row-form-basis*:
assumes *degree p* ≥ *1*
assumes *i-lt*: *i* < *degree p* + *1*
shows *row* (*vec-list-to-square-mat* (*form-basis p M X* (*degree p* − *1*))) *i* = (*form-basis p M X* (*degree p* − *1*)) ! *i*
using *dim-form-basis*[*of p M X degree p* − *1*]
by (*metis add-Suc-right arith-special*(*3*) *assms*(*1*) *assms*(*2*) *le-add-diff-inverse mat-of-rows-row matrix-row-form-basis-carrier plus-1-eq-Suc semiring-norm*(*174*) *vec-list-to-square-mat-def*)

lemma *matrix-diagonal-element*:
assumes *degree p* ≥ *1*
assumes *i* < *degree p*
shows *vec-list-to-square-mat*
(*form-basis p M X* (*degree p* − *1*)) $$

$(i, i) = ((form\text{-}basis\ p\ M\ X\ (degree\ p - 1))\ !\ i)\$i$
using *assms*
by (*metis dim-col-basis-mat dim-row-basis-mat index-row*(*1*) *less-Suc-eq matrix-row-form-basis semiring-norm*(*174*))

lemma *no-zeros-on-diagonal*:
assumes *degree p* $\geq$ *1*
assumes *M* $>$ *0*
assumes *X* $>$ *0*
shows *0* $\notin$ *set* (*diag-mat* (*vec-list-to-square-mat* (*form-basis p M X* (*degree p* $-$ *1*))))
proof $-$
have *eval-elt*: $\bigwedge i.\ i < degree\ p + 1 \Longrightarrow vec\text{-}list\text{-}to\text{-}square\text{-}mat\ (form\text{-}basis\ p\ M\ X\ (degree\ p - 1))\ \$\$\ (i, i) = ((form\text{-}basis\ p\ M\ X\ (degree\ p - 1))!i)\$i$
proof $-$
fix *i*
assume *i* $<$ *degree p* + *1*
then show *vec-list-to-square-mat* (*form-basis p M X* (*degree p* $-$ *1*)) $\$\$$ (*i*, *i*) $=$ ((*form-basis p M X* (*degree p* $-$ *1*))!*i*)$*i*
using *assms dim-form-basis dim-row-basis-mat dim-vector-in-basis index-row*(*1*)
by (*metis dim-col-basis-mat matrix-row-form-basis*)
qed
have $\bigwedge i.\ i < degree\ p + 1 \Longrightarrow ((form\text{-}basis\ p\ M\ X\ (degree\ p - 1))!i)\$i \neq 0$
proof $-$
fix *i*
assume *i-lt*: *i* $<$ *degree p* + *1*
{assume $*$: *i* $<$ *degree p*
then have ((*form-basis p M X* (*degree p* $-$ *1*))!*i*) $=$ (*form-basis-helper p M X* (*degree p* $-$ *1*))!*i*
by (*metis add-2-eq-Suc*′ *add-diff-cancel-left*′ *assms*(*1*) *dim-form-basis form-basis-def length-append-singleton nth-append ordered-cancel-comm-monoid-diff-class.add-diff-inverse plus-1-eq-Suc*)
then have ((*form-basis p M X* (*degree p* $-$ *1*))!*i*)$*i* $\neq$ *0*
using *no-zeros-on-diagonal-helper assms* $*$
by (*metis bot-nat-0.not-eq-extremum mult-eq-0-iff of-nat-eq-0-iff power-eq-0-iff*)
}
moreover {assume $*$: *i* $=$ *degree p*
then have *eq*: ((*form-basis p M X* (*degree p* $-$ *1*))!*i*) $=$ *vec-associated-to-int-poly p X*
by (*metis assms*(*1*) *dim-form-basis-helper form-basis-def nth-append-length ordered-cancel-comm-monoid-diff-class.diff-add*)
have *coeff p* (*degree p*) $\neq$ *0*
using *assms*(*1*) **by** *force*
then have ((*form-basis p M X* (*degree p* $-$ *1*))!*i*)$*i* $\neq$ *0*
using $*$ *eq assms* **unfolding** *vec-associated-to-poly-def*
by *auto*
}

```
    ultimately show ((form-basis p M X (degree p − 1))!i)$i ≠ 0
      using i-lt
      by linarith
  qed
  then have imp-false: (∃ i < degree p + 1.
    (vec-list-to-square-mat (form-basis p M X (degree p − 1)) $$ (i, i))
     = 0) ⟹ False
    by (metis eval-elt)
  then have 0 ∈ set (map (λi. vec-list-to-square-mat
                        (form-basis p M X (degree p − 1)) $$
                       (i, i)) [0..< degree p + 1]) ⟹ False
  proof −
    assume 0 ∈ set (map (λi. vec-list-to-square-mat
                        (form-basis p M X (degree p − 1)) $$
                       (i, i)) [0..< degree p + 1])
    then have ∃(i::nat) <  degree p + 1. vec-list-to-square-mat
                        (form-basis p M X (degree p − 1)) $$
                       (i, i) = 0
      using less-SucI less-Suc-eq
      by auto
    then show False using imp-false
      by auto
  qed
  then show ?thesis
    unfolding diag-mat-def
    using dim-row-basis-mat assms
    by auto
qed

lemma form-basis-helper-is-lower-triangular:
  fixes i j:: nat
  assumes i < j
  assumes j < (degree p + 1)
  shows ((form-basis-helper p M X (degree p − 1))!i)$j = 0
proof −
  have (form-basis-helper p M X (degree p − 1))!i = g-i-vec M X i (degree p + 1)
    using ith-row-form-basis-helper assms
   by (smt (verit, best) Suc-diff-Suc diff-is-0-eq le-add-diff-inverse nat-SN.gt-trans
nat-le-linear nat-less-le plus-1-eq-Suc semiring-norm(174))
  then show ?thesis unfolding g-i-vec-def using assms
    by force
qed

lemma form-basis-is-lower-triangular:
  fixes i j::nat
  assumes i-lt: i < j
  assumes j-lt: j < (degree p + 1)
  shows (vec-list-to-square-mat (form-basis p M X (degree p − 1))) $$ (i,j) = 0
proof −
```

let *?M = vec-list-to-square-mat (form-basis p M X (degree p − 1))*
have *M-elt*: *?M $$ (i, j) = ((form-basis p M X (degree p − 1))!i)$j*
by (*smt* (*verit*) *Suc-eq-plus1 Suc-pred assms*(*1*) *assms*(*2*) *dim-col-basis-mat dim-row-basis-mat index-row*(*1*) *le-add1 less-Suc0 matrix-row-form-basis nat-SN.gt-trans nat-neq-iff plus-1-eq-Suc*)
{assume ∗: *i < degree p*
then have (*form-basis p M X (degree p − 1))!i = (form-basis-helper p M X (degree p − 1))!i*
by (*simp add*: *form-basis-def dim-form-basis-helper nth-append*)
then have ((*form-basis p M X (degree p − 1))!i)$j = ((form-basis-helper p M X (degree p − 1))!i)$j*
by *auto*
then have ((*form-basis p M X (degree p − 1))!i)$j = 0*
using *form-basis-helper-is-lower-triangular*[*OF i-lt j-lt*] *assms* ∗
by *auto*
} moreover {
assume ∗: *i = degree p*
have (*form-basis p M X (degree p − 1))!i = vec-associated-to-int-poly p X*
using ∗ *assms*(*1*) *assms*(*2*) **by** *linarith*
then have ((*form-basis p M X (degree p − 1))!i)$j = 0*
using *assms*
by (*metis* ∗ *Suc-eq-plus1 not-less-eq*)
}
ultimately have ((*form-basis p M X (degree p − 1))!i)$j = 0*
using *assms*(*1*) *assms*(*2*) **by** *linarith*
then show *?thesis* **using** *M-elt*
by *auto*
qed

lemma *form-basis-distinct*:
assumes *degree p ≥ 1*
assumes *M > 0*
assumes *X > 0*
shows *distinct (form-basis p M X (degree p − 1))*
proof −
let *?M = vec-list-to-square-mat (form-basis p M X (degree p − 1))*
let *?dim = degree p + 1*
have *rows-eq*: *rows ?M = form-basis p M X (degree p − 1)*
by (*smt* (*verit*, *best*) *assms*(*1*) *dim-row-basis-mat in-set-conv-nth mat-of-rows-carrier*(*2*) *matrix-row-form-basis-carrier rows-mat-of-rows subset-code*(*1*) *vec-list-to-square-mat-def*)
have *in-set*:*?M ∈ carrier-mat ?dim ?dim*
using *dim-row-basis-mat*[*of p M X*] *dim-col-basis-mat*[*of p M X*] *assms*
unfolding *carrier-mat-def* **by** *auto*
show *?thesis*
using *lower-triangular-imp-distinct*[*OF in-set*] *form-basis-is-lower-triangular*

no-zeros-on-diagonal assms rows-eq
by *auto*
qed

lemma *det-of-matrix*:
fixes *M X*:: *nat*
assumes *M* > *0*
assumes *X* > *0*
assumes *degree p* ≥ *1*
assumes *d-is*: *d* = *degree p*
assumes *n-is*: *n* = (∑ *i*≤*d*. *i*)
assumes *monic-poly*: *coeff p d* = *1*
shows *det* (*vec-list-to-square-mat* (*form-basis p M X* (*degree p* − *1*))) =
M^(*degree p*)∗*X*^*n*
proof −
let *?A* = *vec-list-to-square-mat* (*form-basis p M X* (*degree p* − *1*))
have *in-set*:*?A* ∈ *carrier-mat* (*d*+*1*) (*d*+*1*)
using *d-is dim-row-basis-mat*[*of p M X*] *dim-col-basis-mat*[*of p M X*] *assms*
unfolding *carrier-mat-def*
by *auto*
then have *det ?A* = *prod-list* (*diag-mat ?A*)
using *det-lower-triangular no-zeros-on-diagonal-helper form-basis-is-lower-triangular assms*
by *metis*
have *all-elems-diag*: ⋀*i*. *i* < *d* + *1* ⟹ (*?A* $$ (*i*, *i*)) = ((*form-basis p M X* (*degree p* − *1*)) ! *i*)$*i*
by (*metis assms*(*3*) *assms*(*4*) *carrier-matD*(*2*) *in-set list-of-vec-index list-of-vec-row-nth matrix-row-form-basis*)
have *last-elem*: ((*form-basis p M X* (*d* − *1*)) ! (*d*))$(*d*) = ((*vec-associated-to-int-poly p X*) $ (*d*))
using *dim-form-basis-helper*[*of p M X d*−*1*] *ith-row-form-basis*(*2*)[*of p M X d*−*1*] *d-is*
by (*metis Nat.le-imp-diff-is-add assms*(*3*))
have (∏ *i*<*d*+*1*. (((*form-basis p M X* (*d* − *1*)) ! *i*)$*i*)) = (∏ *i*<*d*. (((*form-basis p M X* (*d* − *1*)) ! *i*)$*i*))∗(((*form-basis p M X* (*d* − *1*)) ! *d*)$*d*)
by *auto*
then have *prod-is*: (∏ *i*<*d*+*1*. (((*form-basis p M X* (*d* − *1*)) ! *i*)$*i*)) = (∏ *i*<*d*. (((*form-basis p M X* (*d* − *1*)) ! *i*)$*i*))∗((*vec-associated-to-int-poly p X*) $ (*d*))
using *last-elem* **by** *auto*
have *prod-list* (*diag-mat ?A*) = (∏ *i*<*d*+*1*. (*?A* $$ (*i*, *i*)))
unfolding *diag-mat-def* **using** *dim-row-basis-mat*[*of p M X*] *assms*(*3*) *d-is*
by (*metis* (*no-types*, *lifting*) *atLeast-upt distinct-upt prod.distinct-set-conv-list*)
then have *eq1*: *prod-list* (*diag-mat ?A*) = (∏ *i*<*d*+*1*. (((*form-basis p M X* (*d* − *1*)) ! *i*)$*i*))
using *all-elems-diag d-is* **by** *auto*
have *sub-prod-is*: (∏ *i*<*d*. ((((*form-basis p M X* (*d*−*1*)))! *i*)$*i*)) = (∏ *i*<*d*. ((((*form-basis-helper p M X* (*d*−*1*)))! *i*)$*i*))
using *ith-row-form-basis*(*1*) **by** *auto*
have *prod-list* (*diag-mat ?A*) = (∏ *i*<*d*. ((((*form-basis p M X* (*d*−*1*)))! *i*)$*i*)) ∗

((*vec-associated-to-int-poly p X*) $ (*d*))
using *HOL.trans*[*OF eq1 prod-is*] **by** *auto*
then have *prod-list-is*: *prod-list* (*diag-mat ?A*) = (∏ *i*<*d*. ((((*form-basis-helper p M X* (*d*−*1*)))! *i*)$*i*)) ∗ ((*vec-associated-to-int-poly p X*) $ (*d*))
using *sub-prod-is* **by** *auto*
have *prod-two*: (*vec-associated-to-int-poly p X*) $ (*d*) = *X*^*d*
unfolding *vec-associated-to-poly-def*
using *d-is monic-poly* **by** *auto*
have *same-prod*: (∏ *i*<*d*. ((*form-basis-helper p M X* (*d*−*1*))! *i*)$*i*) = (∏ *i*<*d*. (*g-i-vec M X i* (*degree p* + *1*))$*i*)
using *ith-row-form-basis-helper* **by** *auto*
have ⋀*k*. *k* ≤ *degree p* ⟹ (∏ *i*<*k*. (*g-i-vec M X i* (*degree p* + *1*))$*i*) = *M*^*k*∗*X*^(∑ *i*<*k*. *i*)
proof −
fix *k*
assume *k* ≤ *degree p*
then show (∏ *i*<*k*. (*g-i-vec M X i* (*degree p* + *1*))$*i*) = *M*^*k*∗*X*^(∑ *i*<*k*. *i*)
proof (*induct k*)
case *0*
then show *?case* **unfolding** *g-i-vec-def* **by** *auto*
next
case (*Suc d*)
have (∏ *i*<*Suc d*. *g-i-vec M X i* (*degree p* + *1*) $ *i*) = (∏ *i*<*d*. *g-i-vec M X i* (*degree p* + *1*) $ *i*)∗*g-i-vec M X* (*d*) (*degree p* + *1*) $ (*d*)
by *auto*
then have *h1*: (∏ *i*<*Suc d*. *g-i-vec M X i* (*degree p* + *1*) $ *i*) = (*M* ^ *d* ∗ *X* ^ ∑ {..<*d*})∗*g-i-vec M X* (*d*) (*degree p* + *1*) $ (*d*)
using *Suc* **by** *auto*
have *h2*: *g-i-vec M X* (*d*) (*degree p* + *1*) $ (*d*) = *M*∗*X*^*d*
using *Suc*(*2*) **unfolding** *g-i-vec-def* **by** *auto*
show *?case*
using *h1 h2*
by (*simp add*: *mult.left-commute power-add*)
qed
qed
then have (∏ *i*<*d*. (*g-i-vec M X i* (*degree p* + *1*))$*i*) = *M*^*d*∗*X*^(∑ *i*<*d*. *i*)
using *d-is* **by** *auto*
then have *prod-one*: (∏ *i*<*d*. ((*form-basis-helper p M X* (*d*−*1*))! *i*)$*i*) = *M*^*d*∗*X*^(∑ *i*<*d*. *i*)
using *same-prod*
by *auto*
have *n-is-alt*: *n* = (∑ *i*<*d*. *i*) + *d*
using *n-is*
by (*metis lessThan-Suc-atMost sum.lessThan-Suc*)
have *prod-list* (*diag-mat ?A*) = *M*^*d*∗*X*^(∑ *i*<*d*. *i*)∗*X*^*d*
using *prod-one prod-two prod-list-is* **by** *auto*
then have *prod-list* (*diag-mat ?A*) = *M*^*d*∗*X*^((∑ *i*<*d*. *i*) + *d*)
by (*simp add*: *power-add*)
then have *det-prod*: *prod-list* (*diag-mat ?A*) = *M*^(*degree p*)∗*X*^*n*

using *n-is d-is n-is-alt* **by** *auto*
have *det ?A = prod-list* (*diag-mat ?A*)
using *prod-list-is form-basis-is-lower-triangular det-lower-triangular*
using ‹*det* (*vec-list-to-square-mat* (*form-basis p M X* (*degree p − 1*))) = *prod-list* (*diag-mat* (*vec-list-to-square-mat* (*form-basis p M X* (*degree p − 1*))))› **by** *force*
then show *?thesis*
using *det-prod* **by** *auto*
qed

5.2.3 Properties of casted matrix

lemma *dim-row-basis-of-int-mat*:
assumes *degree p ≥ 1*
shows *dim-row* (*vec-list-to-square-mat* (*map of-int-vec* (*form-basis p M X* (*degree p − 1*)))) = *degree p + 1*
proof −
have *length* (*form-basis p M X* (*degree p − 1*)) = *degree p + 1*
using *assms dim-form-basis*[*of p M X degree p − 1*]
by *auto*
then show *?thesis*
by (*simp add*: *vec-list-to-square-mat-def*)
qed

lemma *dim-col-basis-of-int-mat*:
assumes *degree p ≥ 1*
shows*dim-col* (*vec-list-to-square-mat* (*map of-int-vec* (*form-basis p M X* (*degree p − 1*)))) = (*degree p + 1*)
proof −
have *length* (*form-basis p M X* (*degree p − 1*)) = *degree p + 1*
using *assms dim-form-basis*[*of p M X degree p − 1*]
by *auto*
then show *?thesis* **unfolding** *vec-list-to-square-mat-def*
by *auto*
qed

lemma *of-int-matrix-row-form-basis-carrier*:
assumes *degree p ≥ 1*
assumes *i < degree p + 1*
shows (*map of-int-vec* (*form-basis p M X* (*degree p − 1*))) ! *i* ∈ *carrier-vec* (*degree p + 1*)
using *matrix-row-form-basis-carrier assms*
by (*metis* (*no-types*, *lifting*) *Suc-eq-plus1 add-Suc-right arith-special*(*3*) *dim-form-basis map-carrier-vec nth-map ordered-cancel-comm-monoid-diff-class.diff-add*)

lemma *of-int-matrix-carrier*:
assumes *degree p ≥ 1*
assumes *i < degree p + 1*
shows *vec-list-to-square-mat* (*map of-int-vec* (*form-basis p M X* (*degree p − 1*))) ∈ *carrier-mat* (*degree p + 1*) (*degree p + 1*)

using *dim-row-basis-of-int-mat*[*of p M X*] *dim-col-basis-of-int-mat*[*of p M X*] *assms*
by *auto*

lemma *of-int-matrix-row-form-basis*:
assumes *degree p* ≥ *1*
assumes *i-lt*: *i* < *degree p* + *1*
shows *row* (*vec-list-to-square-mat* (*map of-int-vec* (*form-basis p M X* (*degree p* − *1*)))) *i* = (*map of-int-vec* (*form-basis p M X* (*degree p* − *1*))) ! *i*
proof −
let *?vs* = *form-basis p M X* (*degree p* − *1*)
have *cv*: *map of-int-vec* (*form-basis p M X* (*degree p* − *1*)) ! *i*
∈ *carrier-vec* (*degree p* + *1*) **using** *matrix-row-form-basis-carrier*[*of p i M X*]
using *of-int-matrix-row-form-basis-carrier*[*of p i M X*] *assms* **by** *auto*
have *i-lt-len*: *i* < *length* (*map of-int-vec* (*form-basis p M X* (*degree p* − *1*)))
using *i-lt dim-form-basis*[*of p M X degree p* − *1*]
by *auto*
show *?thesis*
unfolding *vec-list-to-square-mat-def* **using** *mat-of-rows-row*[*OF i-lt-len cv*]
dim-form-basis[*of p M X degree p* − *1*]
using *assms*(*1*) **by** *auto*
qed

lemma *of-int-matrix-row-form-basis-var*:
assumes *degree p* ≥ *1*
assumes *i-lt*: *i* < *degree p* + *1*
shows *row* (*vec-list-to-square-mat* (*map of-int-vec* (*form-basis p M X* (*degree p* − *1*)))) *i* = *of-int-vec* ((*form-basis p M X* (*degree p* − *1*)) ! *i*)
using *of-int-matrix-row-form-basis assms*
by (*metis add-2-eq-Suc' dim-form-basis le-add-diff-inverse nth-map plus-1-eq-Suc semiring-norm*(*174*))

lemma *of-int-mat-element*:
fixes *i j*::*nat*
assumes *degree p* ≥ *1*
assumes *i* < (*degree p* + *1*)
assumes *j* < (*degree p* + *1*)
shows (*vec-list-to-square-mat* (*map of-int-vec* (*form-basis p M X* (*degree p* − *1*)))) $$ (*i*,*j*) =
of-int ((*vec-list-to-square-mat* (*form-basis p M X* (*degree p* − *1*))) $$ (*i*,*j*))
proof −
let *?M1* = *vec-list-to-square-mat* (*form-basis p M X* (*degree p* − *1*))
let *?M2* = *vec-list-to-square-mat* (*map of-int-vec* (*form-basis p M X* (*degree p* − *1*)))
have *?M1* $$ (*i*, *j*) = (*row ?M1 i*) $ *j*
using *assms*
by (*metis dim-col-basis-mat dim-row-basis-mat index-row*(*1*))
then have *row1*: *?M1* $$ (*i*, *j*) = ((*form-basis p M X* (*degree p* − *1*)) ! *i*)$*j*
using *assms*(*1*) *assms*(*2*) *matrix-row-form-basis* **by** *presburger*

have *?M2 $$ (i, j) = (row ?M2 i) $ j*
using *assms dim-col-basis-of-int-mat dim-row-basis-of-int-mat*
by (*metis index-row(1)*)
then have *row2*: *?M2 $$ (i, j) = (of-int-vec ((form-basis p M X (degree p − 1)) ! i)) $j*
using *assms of-int-matrix-row-form-basis-var*
by *metis*
show *?thesis*
using *row1 row2*
by (*metis assms(1) assms(2) assms(3) dim-col-basis-mat index-map-vec(1) index-row(2) matrix-row-form-basis*)
qed

lemma *of-int-mat-is-lower-triangular*:
fixes *i j*::*nat*
assumes *degree p ≥ 1*
assumes *i < j*
assumes *j < (degree p + 1)*
shows (*vec-list-to-square-mat* (*map of-int-vec* (*form-basis p M X* (*degree p − 1*)))) $$ (*i,j*) = *0*
using *assms of-int-mat-element form-basis-is-lower-triangular*
by (*smt* (*verit*, *best*) *less-imp-le-nat nat-SN.compat of-int-code*(*2*))

lemma *of-int-mat-no-zeros-on-diagonal*:
assumes *p-gt*: *degree p ≥ 1*
assumes *M > 0*
assumes *X > 0*
shows *0 ∉ set* (*diag-mat* (*vec-list-to-square-mat* ((*map of-int-vec* (*form-basis p M X* (*degree p − 1*)))::*rat vec list*)))
proof −
let *?M1 = vec-list-to-square-mat* (*form-basis p M X* (*degree p − 1*))
let *?M2 = vec-list-to-square-mat* ((*map of-int-vec* (*form-basis p M X* (*degree p − 1*)))::*rat vec list*)
let *?dim = degree p + 1*
have *all*: *⋀i. i < ?dim ⟹ ?M2 $$ (i, i) = 0 ⟹ False*
proof −
fix *i*
assume *i-lt*: *i < degree p + 1*
assume *M2-elt*: *vec-list-to-square-mat*
((*map of-int-vec* (*form-basis p M X* (*degree p − 1*)))::*rat vec list*) $$
(*i, i*) = *0*
obtain *k*::*int* **where** *k-prop*: *?M1 $$ (i, i) = k*
using *i-lt matrix-carrier*[*of p i*]
by *auto*
then have *k-inset*: *k ∈ set* (*map* (*λi. ?M1 $$ (i, i)*) [*0*..<*?dim*])
using *i-lt nth-map-upt*[*of i ?dim 0 λi. ?M1 $$ (i, i)*] *dim-row-basis-mat*[*of p M X*]
by (*metis* (*no-types*, *lifting*) *add.left-neutral diff-zero in-set-conv-nth length-map length-upt*)

```
    have same-elt: ?M2 $$ (i, i) = rat-of-int (?M1 $$ (i, i))
      using of-int-mat-element[OF p-gt i-lt i-lt] i-lt
      by auto
    then have (0::rat) = rat-of-int k
      using M2-elt k-prop
      by auto
    then have k = 0
      by auto
    then have 0 ∈ set (diag-mat ?M1)
      using k-inset unfolding diag-mat-def
      using dim-row-basis-mat[of p M X] assms
      by auto
    then show False
      using no-zeros-on-diagonal assms
      by blast
  qed
  then have 0 ∈ set (map (λi. ?M2 $$(i, i)) [0..< ?dim]) ⟹ False
  proof −
    assume 0 ∈ set (map (λi. ?M2 $$(i, i)) [0..< ?dim])
    then have ∃ i ∈ set [0..<degree p + 1]. 0 = ?M2$$(i, i)
      using all atLeastLessThan-iff imageE list.set-map set-upt by force
    then have ∃ i < degree p + 1. 0 = ?M2$$(i, i)
      using atLeast-upt by blast
    then show False using all
      by fastforce
  qed
  then have 0 ∈ set (diag-mat (vec-list-to-square-mat ((map of-int-vec (form-basis
p M X (degree p − 1)))::rat vec list))) ⟹ False
    using dim-row-basis-of-int-mat assms
    unfolding diag-mat-def
    by (smt (verit, best))
  then show ?thesis by auto
qed

lemma of-int-mat-form-basis-distinct:
  assumes degree p ≥ 1
  assumes M > 0
  assumes X > 0
  shows distinct ((map of-int-vec (form-basis p M X (degree p − 1)))::rat vec list)

proof −
  let ?M = vec-list-to-square-mat ((map of-int-vec (form-basis p M X (degree p −
1)))::rat vec list)
  let ?dim = degree p + 1
  have rows-eq: rows ?M = ((map of-int-vec (form-basis p M X (degree p − 1)))::rat
vec list)
    unfolding vec-list-to-square-mat-def
    apply (intro rows-mat-of-rows)
    apply clarsimp
```

by (*smt* (*verit*, *ccfv-SIG*) *One-nat-def add.commute add-Suc-right assms*(*1*) *dim-form-basis in-set-conv-nth matrix-row-form-basis-carrier one-add-one ordered-cancel-comm-monoid-diff-cla plus-1-eq-Suc*)
have *in-set*:*?M* ∈ *carrier-mat ?dim ?dim*
using *dim-row-basis-of-int-mat*[*of p M X*] *dim-col-basis-of-int-mat*[*of p M X*] *assms*
unfolding *carrier-mat-def* **by** *auto*
show *?thesis*
using *lower-triangular-imp-distinct*[*OF in-set*] *of-int-mat-is-lower-triangular of-int-mat-no-zeros-on-diagonal assms rows-eq*
by *metis*
qed

lemma *of-int-form-basis-lin-ind*:
assumes *M* > *0*
assumes *X* > *0*
assumes *degree p* ≥ *1*
shows ¬ *module.lin-dep class-ring* (*module-vec TYPE*(*rat*) (*degree p* + *1*))
(*set* ((*map of-int-vec* (*form-basis p M X* (*degree p* − *1*)))::*rat vec list*))
proof −
let *?M* = *vec-list-to-square-mat* ((*map of-int-vec* (*form-basis p M X* (*degree p* − *1*)))::*rat vec list*)
have *rows-M*: (*rows*
(*vec-list-to-square-mat*
(*map of-int-vec* (*form-basis p M X* (*degree p* − *1*))))::*rat vec list*)
= (*map of-int-vec* (*form-basis p M X* (*degree p* − *1*))::*rat vec list*)
unfolding *vec-list-to-square-mat-def*
apply (*intro rows-mat-of-rows*)
apply *clarsimp*
apply (*subst dim-form-basis*)
by (*smt* (*verit*) *One-nat-def add.commute add-Suc-right assms*(*3*) *dim-form-basis in-set-conv-nth matrix-row-form-basis-carrier nat-1-add-1 ordered-cancel-comm-monoid-diff-class.diff-add plus-1-eq-Suc*)
have *no-zeros-on-diagonal*: *0* ∉ *set* (*diag-mat ?M*)
using *of-int-mat-no-zeros-on-diagonal assms*
by *auto*
have *lower-triangular*: (⋀*i j*. *i* < *j* ⟹ *j* < (*degree p* + *1*) ⟹ *?M* $$ (*i*, *j*) = *0*)
using *of-int-mat-is-lower-triangular assms* **by** *auto*
have *form-basis-distinct*: *distinct* ((*map of-int-vec* (*form-basis p M X* (*degree p* − *1*))::*rat vec list*))
using *of-int-mat-form-basis-distinct assms* **by** *auto*
have *matrix-carrier*: *?M* ∈ *carrier-mat* (*degree p* + *1*) (*degree p* + *1*)
using *of-int-matrix-carrier assms* **by** *auto*
have ¬ *module.lin-dep class-ring* (*module-vec TYPE*(*rat*) (*degree p* + *1*)) (*set* (*rows ?M*))
using *idom-vec.lower-triangular-imp-lin-indpt-rows*[*OF matrix-carrier lower-triangular*] *form-basis-distinct no-zeros-on-diagonal*
by *blast*

then show *?thesis* **using** *rows-M*
by *auto*
qed

5.3 Top-level proof

lemma *towards-coppersmith*:
fixes *p f*:: *int poly*
fixes *M X*:: *nat*
fixes *x0*:: *int*
assumes *zero-mod-M*: *poly p x0 mod M = 0*
assumes *d-is*: *d = degree p*
assumes *d-gt*: *d > 0*
assumes *monic-poly*: *coeff p d = 1*
assumes *X-gt*: *X > 0*
assumes *X-lt*: *X < 1/(sqrt 2)∗1/(root d (d+1))∗(root (d∗(d+1)) M)^2*
assumes *M-gt*: *M > 0*
assumes *x0-le*: *abs x0 ≤ X*
assumes *f-is*: *f = towards-coppersmith p M X*
shows *poly f x0 = 0*
proof −
let *?cs-basis = form-basis p M X (degree p − 1)*

have *li1-1*: *vec-associated-to-int-poly p (int X) ∈ carrier-vec (Suc d)*
by (*metis assms*(*2*) *carrier-vec-dim-vec dim-vec-vec-associated-to-poly semiring-norm*(*174*))
have *li1-2*: ⋀*xa*. *xa ∈ set* (*form-basis-helper p M X* (*degree p − Suc 0*)) ⟹ *xa ∈ carrier-vec* (*Suc d*)
by (*metis* (*no-types*, *lifting*) *assms*(*2*) *carrier-vec-dim-vec dim-form-basis-helper dim-vec g-i-vec-def in-set-conv-nth ith-row-form-basis-helper le-simps*(*2*) *semiring-norm*(*174*))
have *li1*: *set* (*map of-int-vec ?cs-basis*) ⊆ *carrier-vec* (*d + 1*)
unfolding *form-basis-def* **using** *li1-1 li1-2*
by *auto*

have *distinct ?cs-basis*
using *d-is d-gt X-gt M-gt form-basis-distinct* **by** *force*
then have *li2*: *distinct* ((*map of-int-vec ?cs-basis*)::*rat vec list*)
using *casted-distinct-is-distinct* **by** *blast*

have *li3*: *module.lin-indpt class-ring*
(*module-vec TYPE*(*rat*) (*d + 1*))
(*set* (*map of-int-vec*
(*form-basis p M X*
(*degree p − 1*))))
unfolding *form-basis-def*
using *M-gt d-is X-gt d-gt form-basis-def of-int-form-basis-lin-ind* **by** *auto*

have *2*: ⋀*a*. *a ∈ set ?cs-basis* ⟹ *poly* (*int-poly-associated-to-vec a X*) *x0 mod M = 0*

proof −
fix *a*
assume *a* ∈ *set ?cs-basis*
then obtain *i* **where** *a-prop*: *i* < *d* + *1* ∧ *a* = *?cs-basis* ! *i*
by (*metis* (*no-types*, *lifting*) *Suc-leI add-Suc-right arith-special*(*3*) *d-is d-gt dim-form-basis in-set-conv-nth numeral-nat*(*7*) *ordered-cancel-comm-monoid-diff-class.diff-add semiring-norm*(*174*))
{assume ∗: *i* = *d*
then have *poly* (*int-poly-associated-to-vec a X*) *x0 mod M* = *0*
using *zero-mod-M*
using *ith-row-form-basis a-prop* ∗
by (*metis One-nat-def Suc-leI vec-associated-to-int-poly-inverse X-gt d-is d-gt ordered-cancel-comm-monoid-diff-class.diff-add*)
} moreover {assume ∗: *i* < *d*
then have *a-is*:*a* = *g-i-vec M X i* (*d* + *1*)
using *ith-row-form-basis d-is a-prop ith-row-form-basis-helper*
by *simp*
have *i-lteq*: *i* ≤ *degree p*
using *d-is* ∗ **by** *auto*
have *int-poly-associated-to-vec a X* = *monom* (*int M*) *i*
using *d-is a-is int-poly-associated-to-g-i-vec*[*OF X-gt M-gt i-lteq*]
by *blast*
then have *poly* (*int-poly-associated-to-vec a X*) *x0* = *x0*^*i* ∗ (*int M*)
by (*simp add*: *poly-monom*)
then have *poly* (*int-poly-associated-to-vec a X*) *x0 mod M* = *0*
by *auto*
}
ultimately show *poly* (*int-poly-associated-to-vec a X*) *x0 mod M* = *0*
using *a-prop*
by (*metis Suc-eq-plus1 Suc-leI linorder-neqE-nat not-less*)
qed
let *?vec* = *vec-associated-to-int-poly p X*
have *vaip-div*:⋀*j*. *j* < *dim-vec ?vec* ⟹ *?vec* $ *j mod X* ^ *j* = *0*
by (*simp add*: *vec-associated-to-poly-def*)
have *form-basis-helper-div*:(⋀*v j*. *v* ∈ *set* (*form-basis-helper p M X* (*degree p* − *1*)) ⟹
j < *dim-vec v* ⟹ *v* $ *j mod X* ^ *j* = *0*)
proof −
fix *v j*
assume *v-inset*: *v* ∈ *set* (*form-basis-helper p M X* (*degree p* − *1*))
assume *j-lt*: *j* < *dim-vec v*
obtain *i* **where** *i* < *degree p* ∧ *v* = (*form-basis-helper p M X* (*degree p* − *1*)) ! *i*
using *v-inset dim-form-basis-helper*
by (*metis* (*no-types*, *lifting*) *add.right-neutral add-Suc-right add-diff-inverse-nat assms*(*2*) *d-gt in-set-conv-nth less-Suc-eq not-less-zero plus-1-eq-Suc*)
then have *v-is*: *v* = *g-i-vec M X i* (*degree p* + *1*)
using *ith-row-form-basis-helper*[*of i degree p* − *1 p M X*]
by (*metis Suc-pred*′ *assms*(*2*) *d-gt le-simps*(*2*))

{assume $*$: $j = i$
then have $v\$j = int\ M * int\ X \hat{} i$
using *v-is j-lt* **unfolding** *g-i-vec-def*
by *fastforce*
then have $v\ \$\ j\ mod\ X \hat{} j = 0$
using $*$ **by** *auto*
} moreover {assume $*$: $j \neq i$
then have $v\$j = 0$
using *v-is j-lt* **unfolding** *g-i-vec-def*
by *auto*
then have $v\ \$\ j\ mod\ X \hat{} j = 0$
by *auto*
}
ultimately show $v\ \$\ j\ mod\ X \hat{} j = 0$
by *blast*
qed

then have *1*: $(\bigwedge v\ j.\ v \in set\ ?cs\text{-}basis \Longrightarrow j < d + 1 \Longrightarrow v\ \$\ j\ mod\ X \hat{} j = 0)$
using *vaip-div form-basis-helper-div ith-row-form-basis set-form-basis*
by (*metis assms*(*2*) *dim-form-basis dim-vector-in-basis in-set-conv-nth*)

have $det\ (vec\text{-}list\text{-}to\text{-}square\text{-}mat\ (?cs\text{-}basis)) = int\ (M \hat{} degree\ p * X \hat{} \sum\ \{..d\})$
apply (*intro det-of-matrix*[*OF M-gt X-gt - d-is - monic-poly*])
using *d-is d-gt* **by** *auto*

then have *d*: $det\ (mat\text{-}of\text{-}rows\ (d + 1)\ ?cs\text{-}basis) = M\hat{}d * X\hat{}(d * (d+1)\ div\ 2)$
unfolding *vec-list-to-square-mat-def*
dim-form-basis
by (*metis Suc-pred' add-2-eq-Suc' d-is d-gt atLeast0AtMost gauss-sum-nat semiring-norm*(*174*))

have *d1*: $(sqrt\ 2)\hat{}(d*\ (d+1)) = 2 \hat{} (d * (d + 1)\ div\ 2)$
apply (*subst sqrt-even-pow2*[*symmetric*])
using *real-sqrt-power* **by** *auto*

have *d2*:$root\ d\ (real\ (d + 1)) \hat{} (d * (d + 1)) = (d + 1) \hat{} (d + 1)$
by (*metis d-gt of-nat-0-le-iff power-mult real-root-pow-pos2 semiring-1-class.of-nat-power*)

have $sqrt\ 2 * root\ d\ (d+1) * X < (root\ (d*(d+1))\ M)\hat{}2$
using *X-lt d-gt*
by (*auto simp add*: *field-simps*)

then have $(sqrt\ 2 * root\ d\ (d+1) * X) \hat{} (d * (d+1)) < M \hat{} 2$
by (*smt* (*verit*, *ccfv-SIG*) *Num.of-nat-simps*(*4*) *d-gt linordered-semiring-strict-class.mult-pos-pos mult-sign-intros*(*1*) *nat-less-real-le of-nat-0-le-iff real-root-ge-0-iff real-root-less-iff real-root-pos-unique real-root-power real-sqrt-ge-0-iff semiring-1-class.of-nat-power*)

moreover have (*sqrt 2 * root d (d+1) * X) ^ (d * (d+1)) =*
(sqrt 2) ^ (d (d+1)) * X ^ (d * (d+1)) **
*(root d (d+1)) ^ (d * (d+1))*
by (*auto simp add*: *field-simps*)

ultimately have (*2 ^ (d * (d + 1) div 2) **
*real (X ^ (d * (d + 1) div 2)) ^ 2 **
real ((d + 1) ^ (d + 1)))
< M ^ 2
unfolding *d1 d2*
by (*smt* (*verit*) *Num.of-nat-simps*(*2*) *Num.of-nat-simps*(*4*) *Num.of-nat-simps*(*5*) *d1 d2 dvd-div-mult-self even-add even-mult-iff more-arith-simps*(*6*) *of-nat-le-iff one-add-one power-mult semiring-1-class.of-nat-power*)
then have
*(2 ^ (d * (d + 1) div 2) **
*(X ^ (d * (d + 1) div 2)) ^ 2 **
((d + 1) ^ (d + 1)))
< M ^ 2
by (*smt* (*verit*) *Num.of-nat-simps*(*2*) *Num.of-nat-simps*(*4*) *Num.of-nat-simps*(*5*) *nat-1-add-1 of-nat-power-less-of-nat-cancel-iff semiring-1-class.of-nat-power*)

then have
*(2 ^ (d * (d + 1) div 2) **
*(X ^ (d * (d + 1) div 2)) ^ 2 **
*((d + 1) ^ (d + 1))) * M ^ (2 * d)*
*< M ^ 2 * M ^ (2 * d)*
by (*simp add*: *M-gt*)

then have *h3*: *2 ^ (d * (d + 1) div 2) **
*(M ^ d * X ^ (d * (d + 1) div 2))^2 **
(d + 1) ^ (d + 1)
*< (M ^ (2 * (d + 1)))*
by (*auto simp add*: *algebra-simps monoid-mult-class.power-mult power2-eq-square*)
then have *Suc d ^ d * ((M ^ d)^2 * (2 ^ ((d + d * d) div 2) * (X ^ ((d + d * d) div 2))^2)) +*
*d * (Suc d ^ d * ((M ^ d)^2 * (2 ^ ((d + d * d) div 2) * (X ^ ((d + d * d) div 2))^2)))*
*< M * (M * M ^ (d * 2)) ⟹*
*(1 + real d) ^ d **
*((real M ^ d)^2 * (2 ^ ((d + d * d) div 2) * (real X ^ ((d + d * d) div 2))^2)) +*
*real d **
*((1 + real d) ^ d **
*((real M ^ d)^2 * (2 ^ ((d + d * d) div 2) * (real X ^ ((d + d * d) div 2))^2)))*
*< real M * (real M * real M ^ (d * 2))*
proof −
assume *a1*: *Suc d ^ d * ((M ^ d)^2 * (2 ^ ((d + d * d) div 2) * (X ^ ((d + d * d) div 2))^2)) + d * (Suc d ^ d * ((M ^ d)^2 * (2 ^ ((d + d * d) div 2) * (X ^ ((d + d * d) div 2))^2))) < M * (M * M ^ (d * 2))*

```
    have ∀ n na. real n < real na ∨ ¬ n < na
      by simp
    then have (1 + real d) ^ d * real ((M ^ d)^2 * (2 ^((d + d * d) div 2) * (X
^((d + d * d) div 2))^2)) + real d * ((1 + real d) ^ d * real ((M ^ d)^2 * (2 ^((d
+ d * d) div 2) * (X ^((d + d * d) div 2))^2))) < real (M * (M * M ^(d * 2)))
     using a1 by (smt (z3) Num.of-nat-simps(2) Num.of-nat-simps(4) Num.of-nat-simps(5)
plus-1-eq-Suc semiring-1-class.of-nat-power)
    then show (1 + real d) ^ d * ((real M ^ d)^2 * (2 ^((d + d * d) div 2) * (real
X ^((d + d * d) div 2))^2)) + real d * ((1 + real d) ^ d * ((real M ^ d)^2 * (2 ^
((d + d * d) div 2) * (real X ^((d + d * d) div 2))^2))) < real M * (real M * real
M ^(d * 2))
      by simp
  qed
  then have 3: real-of-rat 2 ^
    ((d + 1) * (d + 1 - 1) div 2) *
    real-of-int
     ((det (mat-of-rows (d + 1)
            (form-basis p M X
              (degree p -
               1))))^2) *
    (real (d + 1)) ^(d + 1)
    < real (M ^(2 * (d + 1)))
    using h3
    unfolding d
    by (auto simp add: field-simps)

  interpret lll: LLL-with-assms
    d+1 d+1 ?cs-basis 2
    apply unfold-locales
    subgoal by auto
    subgoal
      unfolding cof-vec-space.lin-indpt-list-def
      using li1 li2 li3 by auto
    using assms(2) d-gt dim-form-basis by auto

  interpret cs: coppersmith-assms
    d+1 d+1 ?cs-basis
    2 x0 M X p
    apply unfold-locales
    subgoal using M-gt by auto
    subgoal using X-gt by auto
    subgoal by auto
    subgoal using zero-mod-M by auto
    subgoal using x0-le by auto
    subgoal
      using assms(2) d-gt dim-form-basis by auto
    subgoal using 1 by auto
    using 2 by auto
```

```
  have *: lll.short-vector = short-vector 2 ?cs-basis
    using lll.short-vector-impl by presburger

  from cs.root-poly-short-vector
  show ?thesis
   using cs.bnd-raw-imp-short-vec-bound 3 lll.square-Gramian-determinant-eq-det-square
    unfolding f-is towards-coppersmith-def first-vector-def Let-def *
    by auto
qed

lemma towards-coppersmith-pretty:
  fixes p f:: int poly
  fixes M X:: nat
  fixes x0:: int
  defines d ≡ degree p
  defines f ≡ towards-coppersmith p M X
  assumes monic-poly: monic p
  assumes d > 0 and M > 0 and X > 0
  assumes zero-mod-M: poly p x0 mod M = 0
  assumes X-lt: X < 1/(sqrt 2)*1/(root d (d+1))*(root (d*(d+1)) M)^2
  assumes x0-le: abs x0 ≤ X
  shows poly f x0 = 0
  using assms towards-coppersmith
  by presburger

end
```

6 Proof of Coppersmith's Method

In this file, we prove the full version of Coppersmith's method.

```
theory Coppersmith
imports Coppersmith-Generic
begin
```

6.1 Preliminaries and setup

```
lemma calculate-h-coppersmith-aux-gteq1:
  fixes e::real
  assumes degree p > 1
  assumes e > 0
  shows calculate-h-coppersmith-aux p e ≥ 1
proof −
  let ?x = degree p
  have h1: real ?x − 1 > 0
    using assms by simp
  have h2: (real ?x * e) > 0
    using assms by simp
  have (real ?x − 1)/(real ?x*e) > 0
```

```
    using h1 h2 by simp
  then have ((real ?x − 1) / (real ?x ∗ e) + 1) / real ?x > 0
    using assms
    by (smt (verit) h1 zero-less-divide-iff)
  then show ?thesis  unfolding calculate-h-coppersmith-aux-def
    by (smt (verit) zero-less-ceiling)
qed

lemma calculate-h-coppersmith-aux-gt1:
  assumes deg-gt: degree p > 1
  assumes e > 0
  assumes e-lt2: e < 1/(real (degree p))
  shows calculate-h-coppersmith-aux p e > 1
proof −
  let ?x = degree p
  have xe: real ?x ∗ e < 1
    using e-lt2 deg-gt
    by (simp add: less-divide-eq mult-of-nat-commute)
  then have div-gt: (real ?x − 1) / (real ?x ∗ e) > real ?x − 1
   by (smt (verit) assms(1) assms(2) div-by-1 divide-strict-left-mono less-imp-of-nat-less
linordered-semiring-strict-class.mult-pos-pos of-nat-1)
  have ((real ?x − 1) / (real ?x ∗ e) + 1) > real ?x
    using div-gt
    by linarith
  then show ?thesis
  unfolding calculate-h-coppersmith-aux-def
 by (smt (verit, ccfv-SIG) assms(2) e-lt2 less-divide-eq-1-pos one-less-ceiling zero-less-divide-iff)
qed
```

6.1.1 Dimension properties of matrix

```
lemma dim-vector-vec-associated-to-int-poly-padded:
  shows dim-vec (vec-associated-to-int-poly-padded n p X) = n
  unfolding vec-associated-to-int-poly-padded-def by simp

lemma dim-vector-row-of-coppersmith-matrix:
  shows dim-vec (row-of-coppersmith-matrix p M X h i j) = (degree p)∗h
 unfolding row-of-coppersmith-matrix-def using dim-vector-vec-associated-to-int-poly-padded
  by blast

lemma dim-vector-form-basis-coppersmith-aux:
  fixes i:: nat
  assumes i < (degree p)
  shows dim-vec ((form-basis-coppersmith-aux p M X h j) ! i) = (degree p)∗h
proof −
  have (map (λi. row-of-coppersmith-matrix p M X h i j) [0..<degree p]) ! i
       = row-of-coppersmith-matrix p M X h i j
    using assms by simp
  then show ?thesis
```

unfolding *form-basis-coppersmith-aux-def*
using *dim-vector-row-of-coppersmith-matrix*[*of p M X h - j*] **by** *presburger*
qed

lemma *length-form-basis-coppersmith-aux*:
shows *length* (*form-basis-coppersmith-aux p M X h j*) = *degree p*
unfolding *form-basis-coppersmith-aux-def* **by** *simp*

lemma *length-form-basis-coppersmith*:
fixes *h*:: *nat*
assumes *h-gt*: *h* > *0*
shows *length* (*form-basis-coppersmith p M X h*) = (*degree p*)∗*h*
proof −
let *?form-basis* = *form-basis-coppersmith p M X h*
have *length* (*concat* (*map* (*form-basis-coppersmith-aux p M X h*) [*0*..<*h*])) =
sum-list (*map length* (*map* (*form-basis-coppersmith-aux p M X h*) [*0*..<*h*]))
using *length-concat*[*of map* (*form-basis-coppersmith-aux p M X h*) [*0*..<*h*]]
by *blast*
then have *length* (*concat* (*map* (*form-basis-coppersmith-aux p M X h*) [*0*..<*h*]))
=
sum-list (*map* (λ*j*. *length* (*form-basis-coppersmith-aux p M X h j*)) [*0*..<*h*])
by (*smt* (*verit*, *ccfv-SIG*) *length-map nth-equalityI nth-map*)
then have *length* (*concat* (*map* (*form-basis-coppersmith-aux p M X h*) [*0*..<*h*]))
=
sum-list (*map* (λ*j*. *degree p*) [*0*..<*h*])
using *length-form-basis-coppersmith-aux* **by** *presburger*
then have *len-is*: *length* (*concat* (*map* (*form-basis-coppersmith-aux p M X h*)
[*0*..<*h*])) =
(∑ *j*←[*0*..<*h*]. *degree p*) **by** *blast*
have (∑ *j*←[*0*..<*h*]. *degree p*) = *h*∗(*degree p*)
using *h-gt* **proof** (*induct h*)
case *0*
then show *?case* **by** *blast*
next
case (*Suc h*)
{assume ∗: *h* = *0*
then have (∑ *j*←[*0*..<*Suc h*]. *degree p*) = *Suc h* ∗ *degree p*
by *simp*
} moreover {assume ∗: *h* > *0*
then have (∑ *j*←[*0*..< *h*]. *degree p*) = *h* ∗ *degree p*
using *Suc* **by** *blast*
then have (∑ *j*←[*0*..<*Suc h*]. *degree p*) = *Suc h* ∗ *degree p*
by *simp*
}
ultimately show *?case* **by** *fast*
qed
then show *?thesis* **using** *len-is*
by (*simp add*: *form-basis-coppersmith-def*)
qed

lemma *concat-property-helper*:
assumes $j < h$
shows *concat* (*map* ($\lambda i.$ *f i*) [*0*..<*h*]) = *concat* (*map* ($\lambda i.$ *f i*) [*0*..<*j*]) @ *concat* (*map* ($\lambda i.$ *f i*) [*j*..<*h*])
using *assms*
proof (*induct j*)
case *0*
then show *?case* **by** *simp*
next
case (*Suc j*)
then show *?case*
by (*metis concat-append less-nat-zero-code map-append not-less-eq upt-append*)
qed

lemma *concat-equal-lists-length*:
fixes *f*:: *nat* ⇒ *int vec list*
fixes *i j d*:: *nat*
assumes *len-is*: $\bigwedge i.\ i < h \Longrightarrow$ *length* (*f i*) = *d*
shows *length* (*concat* (*map* ($\lambda i.$ *f i*) [*0*..<*h*])) = *d*∗*h*
using *assms*
proof (*induct h*)
case *0*
then show *?case* **by** *auto*
next
case (*Suc h*)
then have *ind-h*: *length* (*concat* (*map f* [*0*..<*h*])) = *d* ∗ *h*
by *simp*
have *concat-is*: *concat* (*map f* [*0*..<*Suc h*]) = *concat* (*map f* [*0*..<*h*]) @ *concat* (*map f* [*h*..<*Suc h*])
using *concat-property-helper* **by** *blast*
have *concat* (*map f* [*h*..<*Suc h*]) = *f h*
by *auto*
then have *length* (*concat* (*map f* [*h*..<*Suc h*])) = *d*
using *Suc.prems* **by** *force*
then show *?case* **using** *ind-h concat-is*
by *simp*
qed

lemma *concat-property*:
fixes *f*:: *nat* ⇒ *int vec list*
fixes *i j d*:: *nat*
assumes *h* > *0*
assumes *d-gt*: *d* > *0*
assumes *r-lt*: *r* < *d*∗*h*
assumes *len-is*: $\bigwedge i.\ i < h \Longrightarrow$ *length* (*f i*) = *d*
assumes *j-eq*: *j* = *nat* ⌊*real r* / *real d*⌋
assumes *i-eq*: *i* = *r* − *d* ∗ *j*
shows (*concat* (*map* ($\lambda i.$ *f i*) [*0*..<*h*])) ! *r* = (*f j*) ! *i*

using *assms nth-append*
proof −
have *j-lt-h*: $j < h$
using *r-lt j-eq*
by (*simp add*: *floor-divide-of-nat-eq less-mult-imp-div-less mult.commute*)
then have *concat-is*: *concat* (*map* (λ*i*. *f i*) [*0*..<*h*]) = *concat* (*map* (λ*i*. *f i*) [*0*..<*j*]) @ *concat* (*map* (λ*i*. *f i*) [*j*..<*h*])
using *concat-property-helper assms*
by *blast*
have *len-is*: *length* (*concat* (*map* (λ*i*. *f i*) [*0*..<*j*])) = *d*∗*j*
using *len-is*
by (*meson* ‹*j* < *h*› *concat-equal-lists-length nat-SN.gt-trans*)
have $r \geq d*j$
using *i-eq*
by (*simp add*: *assms*(*5*) *floor-divide-of-nat-eq*)
then have (*concat* (*map* (λ*i*. *f i*) [*0*..<*j*]) @ *concat* (*map* (λ*i*. *f i*) [*j*..<*h*])) ! *r* = *concat* (*map* (λ*i*. *f i*) [*j*..<*h*]) ! (*r* − *d*∗*j*)
using *nth-append len-is*
by (*metis leD*)
then have *concat-r-1*: (*concat* (*map* (λ*i*. *f i*) [*0*..<*h*])) ! *r* = *concat* (*map* (λ*i*. *f i*) [*j*..<*h*]) ! *i*
using *concat-is assms*(*6*) **by** *presburger*
have *concat-is-2*: *concat* (*map* (λ*i*. *f i*) [*j*..<*h*]) = (*f j*) @ *concat* (*map* (λ*i*. *f i*) [(*j*+*1*)..<*h*])
by (*metis Suc-eq-plus1* ‹*j* < *h*› *concat.simps*(*2*) *list.map*(*2*) *upt-conv-Cons*)
have *r* < *d*∗(*nat* ⌊*real r* / *real d*⌋) + *d*
using *d-gt*
by (*simp add*: *add.commute dividend-less-times-div floor-divide-of-nat-eq*)
then have *i* < *d*
using *i-eq j-eq*
using ‹*d* ∗ *j* ≤ *r*› **by** *presburger*
then have *concat* (*map* (λ*i*. *f i*) [*j*..<*h*]) ! *i* = (*f j*) ! *i*
using *concat-is-2 len-is j-lt-h nth-append*
by (*metis assms*(*4*))
then show *?thesis*
using *concat-r-1* **by** *metis*
qed

lemma *row-of*:
assumes *r-lt*: *r* < (*degree p*)∗*h*
defines *d* ≡ *degree p*
defines *j* ≡ *nat* ⌊*real r* / *real d*⌋
defines *i* ≡ *r* − *d* ∗ *j*
shows (*form-basis-coppersmith p M X h*) ! *r* =
((*form-basis-coppersmith-aux p M X h j*) ! *i*)
using *r-lt j-def i-def concat-property length-form-basis-coppersmith-aux*
using *assms*(*2*) *form-basis-coppersmith-def*
by (*smt* (*verit*, *best*) *less-imp-of-nat-less mult-0-right mult-cancel1 neq0-conv*)

lemma *dim-vector-form-basis-coppersmith*:
fixes *i*:: *nat*
assumes $i < (degree\ p)*h$
shows *dim-vec* ((*form-basis-coppersmith p M X h*) ! *i*) = $(degree\ p)*h$
proof −
let *?j* = (*nat* ⌊*real i* / *real* (*degree p*)⌋)
have (*form-basis-coppersmith p M X h*) ! *i* = (*form-basis-coppersmith-aux p M X h ?j*) ! ($i - (degree\ p) * ?j$)
using *dim-vector-form-basis-coppersmith-aux*
using *assms row-of* **by** *presburger*
then show *?thesis*
using *dim-vector-form-basis-coppersmith-aux*
by (*metis assms bot-nat-0.not-eq-extremum floor-divide-of-nat-eq less-nat-zero-code mod-less-divisor modulo-nat-def mult.commute mult-0-right nat-int*)
qed

6.1.2 Equivalent Coppersmith matrix

definition *form-coppersmith-matrix*:: *int poly* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *int mat*
where *form-coppersmith-matrix p M X h* = *mat* ($(degree\ p)*h$) ($(degree\ p)*h$)
($\lambda(r, c)$. (*let d* = *degree p*; *j* = *nat* (*floor* (r/d)); *i* = ($r - d*j$) *in* ($M\hat{}(h-1-j)$)*(*coeff* ($p\hat{}j*$ (*monom 1 i*)) *c*)*$X\hat{}c$))

lemma *matrix-match*:
assumes *r-lt*: $r < (degree\ p)*h$
assumes *c-lt*: $c < (degree\ p)*h$
assumes $h > 0$
shows (*vec-list-to-square-mat* (*form-basis-coppersmith p M X h*))$$(*r*, *c*) = (*form-coppersmith-matrix p M X h*)$$(*r*, *c*)
proof −
let *?d* = *degree p*
let *?j* = *nat* ⌊*real r* / *real ?d*⌋
let *?i* = *r* − *?d* ∗ *?j*

have *i-lt*: *?i* < *?d*
by (*metis* (*no-types*) *Groups.mult-ac*(2) *assms*(2) *bot-nat-0.not-eq-extremum floor-divide-of-nat-eq less-nat-zero-code mod-less-divisor modulo-nat-def mult-0-right nat-int*)
have *i-gt*: *?i* ≥ *0*
by *blast*
have *?j* ≥ *0*
by *auto*
have (*form-basis-coppersmith p M X h*) ! *r* =
((*form-basis-coppersmith-aux p M X h ?j*) ! *?i*)
using *length-form-basis-coppersmith-aux length-form-basis-coppersmith r-lt row-of*
by *presburger*
then have *eq*: *vec-list-to-square-mat* (*form-basis-coppersmith p M X h*) $$ (*r*, *c*) =

((*form-basis-coppersmith-aux p M X h ?j*) ! *?i*) $ *c*
using *c-lt r-lt* **unfolding** *vec-list-to-square-mat-def*
by (*metis assms*(*3*) *length-form-basis-coppersmith mat-of-rows-index*)
have *form-basis-elt*: ((*form-basis-coppersmith-aux p M X h ?j*) ! *?i*) $ *c* = (*row-of-coppersmith-matrix p M X h ?i ?j*) $ *c*
unfolding *form-basis-coppersmith-aux-def*
using *i-lt i-gt*
by *simp*
have (*row-of-coppersmith-matrix p M X h ?i ?j*) $ *c* = *int M* ^ (*h* − *1* − *?j*) ∗ *poly.coeff* (*p* ^ *?j* ∗ *monom 1 ?i*) *c* ∗ *int X* ^ *c*
unfolding *row-of-coppersmith-matrix-def vec-associated-to-int-poly-padded-def*

using *coeff-smult assms*(*2*) *index-vec semiring-1-class.of-nat-power* **by** *auto*
then have *eq1*: ((*form-basis-coppersmith-aux p M X h ?j*) ! *?i*) $ *c* = *int M* ^ (*h* − *1* − *?j*) ∗ *poly.coeff* (*p* ^ *?j* ∗ *monom 1 ?i*) *c* ∗ *int X* ^ *c*
using *form-basis-elt* **by** *argo*
have *eq2*: (*form-coppersmith-matrix p M X h*)$$(*r*, *c*) = *int M* ^ (*h* − *1* − *?j*) ∗ *poly.coeff* (*p* ^ *?j* ∗ *monom 1 ?i*) *c* ∗ *int X* ^ *c*
by (*metis* (*mono-tags*, *lifting*) *assms*(*1*) *assms*(*2*) *form-coppersmith-matrix-def index-mat*(*1*) *prod.simps*(*2*))
then show *?thesis* **using** *eq1 eq2 eq* **by** *auto*
qed

6.1.3 Lower triangular

lemma *form-coppersmith-matrix-is-lower-triangular*:
fixes *r c*::*nat*
assumes *h* > *0*
assumes *r-lt*: *r* < *c*
assumes *c-lt*: *c* < (*degree p*)∗*h*
shows (*form-coppersmith-matrix p M X h*)$$(*r*, *c*) = *0*
proof −
let *?d* = *degree p*
let *?j* = *nat* ⌊*real r* / *real ?d*⌋
let *?i* = *r* − *?d* ∗ *?j*
have *i-lt*: *?i* < *?d*
using *Groups.mult-ac*(*2*) *assms*(*2*) *bot-nat-0.not-eq-extremum floor-divide-of-nat-eq less-nat-zero-code mod-less-divisor modulo-nat-def mult-0-right nat-int*
by (*metis assms*(*3*))
have *i-gt*: *?i* ≥ *0*
by *blast*
have *?j* ≥ *0*
by *auto*
have *entry-prop*: (*form-coppersmith-matrix p M X h*)$$(*r*, *c*) = (*M*^(*h*−*1*−*?j*))∗(*coeff* (*p*^*?j*∗ (*monom 1 ?i*)) *c*)∗*X*^*c*
unfolding *form-coppersmith-matrix-def*
by (*metis* (*no-types*, *lifting*) *assms*(*2*) *assms*(*3*) *index-mat*(*1*) *less-imp-le-nat nat-SN.compat2 prod.simps*(*2*) *semiring-1-class.of-nat-power*)
have *degree* (*p*^*?j*∗ (*monom 1 ?i*)) = *?d*∗*?j* + *?i*

by (*smt* (*z3*) *degree-power-eq degree-0 degree-monom-eq degree-mult-eq i-gt i-lt monom-hom.hom-0-iff nat-SN.compat2 rel-simps*(*70*) *semiring-1-no-zero-divisors-class.power-not-zero*)
then have *coeff* (*p*^*?j*∗ (*monom 1 ?i*)) *c* = *0*
using *r-lt*
by (*metis* (*no-types*, *lifting*) *floor-divide-of-nat-eq leD le-add-diff-inverse le-degree nat-int thd*)
then show *?thesis* **using** *entry-prop* **by** *simp*
qed

lemma *form-coppersmith-basis-is-lower-triangular*:
fixes *i j*::*nat*
assumes *h* > *0*
assumes *i-lt*: *i* < *j*
assumes *j-lt*: *j* < (*degree p*)∗*h*
shows (*vec-list-to-square-mat* (*form-basis-coppersmith p M X h*)) $$ (*i*,*j*) = *0*
using *matrix-match assms*
by (*metis* (*no-types*, *lifting*) *form-coppersmith-matrix-is-lower-triangular nat-SN.gt-trans*)

6.1.4 Distinct elements

lemma *coppersmith-matrix-carrier-mat*:
assumes *h* > *0*
shows *vec-list-to-square-mat* (*form-basis-coppersmith p M X h*) ∈ *carrier-mat* ((*degree p*)∗*h*) ((*degree p*)∗*h*)
using *length-form-basis-coppersmith dim-vector-form-basis-coppersmith*
by (*simp add*: *assms mat-of-rows-def vec-list-to-square-mat-def*)

lemma *no-zeros-on-diagonal-coppersmith*:
assumes *degree p* ≥ *1*
assumes *M-gt*: *M* > *0*
assumes *X-gt*: *X* > *0*
assumes *h-gt*: *h* > *0*
shows *0* ∉ *set* (*diag-mat* (*vec-list-to-square-mat* (*form-basis-coppersmith p M X h*)))
proof −
let *?d* = *degree p*
have *row-dim*: *dim-row* (*vec-list-to-square-mat* (*form-basis-coppersmith p M X h*)) = *?d*∗*h*
using *coppersmith-matrix-carrier-mat assms* **by** *blast*
have ⋀ *i*::*nat*. *i* < *?d*∗*h* ⟹ (*vec-list-to-square-mat* (*form-basis-coppersmith p M X h*))$$(*i*, *i*) ≠ *0*
proof −
fix *i*::*nat*
assume ∗: *i* < *?d*∗*h*

let *?d* = *degree p*
let *?j* = *nat* ⌊*real i* / *real ?d*⌋
let *?i* = *i* − *?d* ∗ *?j*
have *coeff-is*: (*form-coppersmith-matrix p M X h*)$$(*i*, *i*) = (*M*^(*h*−*1*−*?j*))∗(*coeff*

```
(p^?j* (monom 1 ?i)) i)*X^i
      unfolding form-coppersmith-matrix-def
    by (metis (mono-tags, lifting) * index-mat(1) prod.simps(2) semiring-1-class.of-nat-power)
    have degree (p^?j* (monom 1 ?i)) = ?i + ?j*?d
      by (smt (verit, best) degree-power-eq assms(1) degree-0 degree-monom-eq de-
gree-mult-eq less-numeral-extra(3) monom-hom.hom-0-iff mult.commute of-nat-0-less-iff
of-nat-1 of-nat-le-iff semiring-1-no-zero-divisors-class.power-not-zero)
    also have ... = i
     by (metis add.commute floor-divide-of-nat-eq le-add-diff-inverse mult.commute
nat-int times-div-less-eq-dividend)
    finally have coeff (p^?j* (monom 1 ?i)) i ≠ 0
    by (smt (verit, del-insts) assms(1) degree-0 leading-coeff-neq-0 monom-hom.hom-0-iff
mult-eq-0-iff not-one-le-zero semiring-1-no-zero-divisors-class.power-not-zero)
    then have (form-coppersmith-matrix p M X h)$$(i, i) ≠ 0
      using coeff-is M-gt X-gt
      by (metis mult-eq-0-iff nat-0 nat-int semiring-norm(137) zero-less-power)
     then show (vec-list-to-square-mat (form-basis-coppersmith p M X h))$$(i, i)
≠ 0
      using matrix-match * h-gt by fastforce
  qed
  then show ?thesis
    unfolding diag-mat-def using row-dim
    by auto
qed

lemma form-basis-coppersmith-distinct:
  fixes M X::nat
  assumes 1 ≤ degree p
  assumes p-neq: p ≠ 0
  assumes M-gt: M > 0
  assumes X-gt: X > 0
  assumes h-gt: h > 0
  shows distinct (form-basis-coppersmith p M X h)
proof −
  let ?M = vec-list-to-square-mat (form-basis-coppersmith p M X h)
  let ?dim = degree p * h
  have carrier-mat: ?M ∈ carrier-mat ?dim ?dim
    using coppersmith-matrix-carrier-mat[OF h-gt] by auto
  have lower-triangular: (⋀i j. i < j ⟹
          j < degree p * h ⟹
          vec-list-to-square-mat (form-basis-coppersmith p M X h) $$ (i, j) =
          0)
    using form-coppersmith-basis-is-lower-triangular[OF h-gt]
    by blast
  have zero-not-in: 0 ∉ set (diag-mat
              (vec-list-to-square-mat
                (form-basis-coppersmith p M X
                  h)))
    using no-zeros-on-diagonal-coppersmith[OF assms(1) M-gt X-gt h-gt]
```

by *blast*
have *rows* (*vec-list-to-square-mat* (*form-basis-coppersmith p M X h*)) = *form-basis-coppersmith p M X h*
unfolding *vec-list-to-square-mat-def* **using** *carrier-mat*
by (*metis carrier-vec-dim-vec dim-vector-form-basis-coppersmith h-gt in-set-conv-nth length-form-basis-coppersmith rows-mat-of-rows subset-code*(*1*))
then show *?thesis*
using *lower-triangular-imp-distinct*[*OF carrier-mat lower-triangular zero-not-in*]

by *simp*
qed

lemma *matrix-row-form-basis-coppersmith*:
assumes *degree p* ≥ *1*
assumes *i-lt*: *i* < (*degree p*)∗*h*
assumes *h* > *0*
shows *row* (*vec-list-to-square-mat* (*form-basis-coppersmith p M X h*)) *i* = (*form-basis-coppersmith p M X h*) ! *i*
using *length-form-basis-coppersmith*[*of h p M X*]
by (*metis assms*(*2*) *assms*(*3*) *carrier-vec-dim-vec dim-vector-form-basis-coppersmith mat-of-rows-row vec-list-to-square-mat-def*)

6.1.5 Linear independence properties

lemma *form-basis-coppersmith-lin-ind*:
assumes *M* > *0*
assumes *X* > *0*
assumes *degree p* ≥ *1*
assumes *h* > *0*
shows ¬ *module.lin-dep class-ring* (*module-vec TYPE*(*int*) ((*degree p*)∗*h*)) (*set* (*form-basis-coppersmith p M X h*))
proof −
let *?M* = *vec-list-to-square-mat* (*form-basis-coppersmith p M X h*)
have *no-zeros-on-diagonal*: *0* ∉ *set* (*diag-mat ?M*)
using *no-zeros-on-diagonal-coppersmith assms*
by *presburger*
have *form-basis-distinct*: *distinct* (*form-basis-coppersmith p M X* ((*degree p*)∗*h*))
using *form-basis-coppersmith-distinct assms* **by** *fastforce*
have *matrix-carrier*: *?M* ∈ *carrier-mat* ((*degree p*)∗*h*) ((*degree p*)∗*h*)
using *assms*(*4*) *coppersmith-matrix-carrier-mat* **by** *blast*
have *lower-triangular*: (⋀*i j*. *i* < *j* ⟹ *j* < ((*degree p*)∗*h*) ⟹ *?M* $$ (*i*, *j*) = *0*)
using *form-coppersmith-basis-is-lower-triangular assms*
by *blast*
have ¬ *module.lin-dep class-ring* (*module-vec TYPE*(*int*) ((*degree p*)∗*h*)) (*set* (*rows ?M*))
using *idom-vec.lower-triangular-imp-lin-indpt-rows*[*OF matrix-carrier lower-triangular*] *no-zeros-on-diagonal form-basis-distinct*
by *blast*

then show *?thesis*
using *Suc-eq-plus1 assms(3) length-rows matrix-row-form-basis-coppersmith nth-equalityI nth-rows plus-1-eq-Suc*
by (*metis assms(4) carrier-matD(1) length-form-basis-coppersmith local.matrix-carrier*)
qed

6.1.6 Divisible by X property

lemma *row-of-cs-matrix-div-by-Xpow*:
fixes *M X i j h*::*nat*
assumes *i-lt*: *i* < *degree p*
assumes *j-lt*: *j* < *h*
assumes *j-lt*: *ja* < *h*∗(*degree p*)
shows (*row-of-coppersmith-matrix p M X h i j*) $ *ja mod* (*X* ^ *ja*) = *0*
proof −
let *?v* = *row-of-coppersmith-matrix p M X h i j*
have *ja* < *dim-vec ?v*
unfolding *row-of-coppersmith-matrix-def vec-associated-to-int-poly-padded-def*
using *j-lt*
by (*simp add*: *mult.commute*)
then show *?thesis*
unfolding *row-of-coppersmith-matrix-def vec-associated-to-int-poly-padded-def*
by *simp*
qed

lemma *form-basis-coppersmith-div-by-Xpow*:
fixes *M X*::*nat*
fixes *a*:: *int vec*
assumes *j-lt*: *ja* < *h*∗(*degree p*)
assumes *a-inset*: *a* ∈ *set* (*form-basis-coppersmith p M X h*)
shows (*a* $ *ja*) *mod* (*X* ^ *ja*) = *0*
proof −
have ∃ *i j*. *i* < *degree p* ∧ *j* < *h* ∧ *a* = *row-of-coppersmith-matrix p M X h i j*
using *a-inset* **unfolding** *form-basis-coppersmith-def form-basis-coppersmith-aux-def*
by *fastforce*
then show *?thesis*
using *assms row-of-cs-matrix-div-by-Xpow*
by *blast*
qed

6.1.7 Zero mod M property

lemma *row-of-cs-matrix-zero-mod-M*:
fixes *M X i j h*::*nat*
assumes *p-neq*: *p* ≠ *0*
assumes *M-gt*: *M* > *0*
assumes *X-gt*: *X* > *0*
assumes *h-gt1*: *h* > *1*
assumes *p-zero-mod-M*: *poly p x0 mod M* = *0*
assumes *deg-gt*: *degree p* > *1*

assumes *i-lt*: $i < degree\ p$
assumes *j-lt*: $j < h$
shows *poly* (*int-poly-associated-to-vec* (*row-of-coppersmith-matrix p M X h i j*) *X*) *x0 mod M* ^(*h*−*1*) = *0*
proof −
have *ipos*: $0 < 1/(real\ (degree\ p))$
using *deg-gt* **by** *auto*

have *hj*: $h - 1 = ((h - 1) - j) + j$
using *h-gt1 j-lt*
by *auto*
let *?dim-vec* = $(degree\ p * h)$
let *?og-vec* = *vec ?dim-vec* (λ*ja*. *poly.coeff* (*smult* (*int* (*M* ^ (*h* − *1* − *j*))) (*p* ^ *j* ∗ *monom 1 i*)) *ja* ∗ *int* (*X* ^ *ja*))::*int vec*
let *?new-vec* = *vec ?dim-vec* (λ*ja*. *poly.coeff* (*smult* (*int* (*M* ^ (*h* − *1* − *j*))) (*p* ^ *j* ∗ *monom 1 i*)) *ja*)

have *lhs-eq*: ⌊*real-of-int* (*?og-vec* $ *ja*) / *real* (*X* ^ *ja*)⌋ = *floor* (((*?og-vec* $ *ja*)::*real*)/(*X*^*ja*)) **if** *ja* < *?dim-vec* **for** *ja*::*nat*
by *simp*
have *rhs-eq*: *poly.coeff* (*smult* (*int* (*M* ^ (*h* − *1* − *j*))) (*p* ^ *j* ∗ *monom 1 i*)) *ja* = *?new-vec* $ *ja*
if ∗: *ja* < *?dim-vec* **for** *ja*::*nat*
using ∗ **by** *auto*
have *floor-div*: *floor* (((*?og-vec* $ *ja*)::*real*)/(*X*^*ja*)) = *poly.coeff* (*smult* (*int* (*M* ^ (*h* − *1* − *j*))) (*p* ^ *j* ∗ *monom 1 i*)) *ja*
if *ja* < *?dim-vec* **for** *ja*::*nat*
by (*metis* (*no-types*, *lifting*) *X-gt floor-divide-of-int-eq index-vec nat-neq-iff nonzero-mult-div-cancel-right of-int-of-nat-eq of-nat-eq-0-iff semiring-1-no-zero-divisors-class.power-not-zero that*)
then have *floor-div-match*: ⌊*real-of-int* (*?og-vec* $ *ja*) / *real* (*X* ^ *ja*)⌋ = *?new-vec* $ *ja* **if** ∗: *ja* < *?dim-vec* **for** *ja*::*nat*
using *lhs-eq rhs-eq* ∗ **by** *presburger*
let *?og-vec-div* = *vec* $(degree\ p * h)$
(λ*ja*. ⌊*real-of-int* (*?og-vec* $ *ja*) / *real* (*X* ^ *ja*)⌋)
have *same-vec*: *?og-vec-div* = *?new-vec* **using** *floor-div-match* **by** *force*

have *int-poly-associated-to-vec ?og-vec X* = $(\sum i{<}degree\ p * h.$
monom (*?og-vec-div* $ *i*) *i*)
unfolding *int-poly-associated-to-vec-def* **by** *auto*
then have *int-poly-associated-to-vec ?og-vec X* = $(\sum i{<}$*?dim-vec. monom* (*?new-vec* $ *i*) *i*)
using *same-vec* **by** *presburger*

then have *poly-of-interest-is*: *int-poly-associated-to-vec* (*row-of-coppersmith-matrix p M X h i j*) *X* =
$(\sum i{<}$*?dim-vec. monom* (*?new-vec* $ *i*) *i*)
unfolding *row-of-coppersmith-matrix-def vec-associated-to-int-poly-padded-def*
by *auto*

have *same-coeffs-lhs*: *poly.coeff* ($\sum i$<*?dim-vec*. *monom* (*?new-vec* $ *i*) *i*) *ia* = *?new-vec* $ *ia*
if ∗:*ia* < *?dim-vec* **for** *ia*
by (*metis* (*no-types*, *lifting*) ∗ ‹*int-poly-associated-to-vec* (*vec* (*degree p* ∗ *h*) (λ*ja*. *poly.coeff* (*smult* (*int* (*M* ^(*h* − 1 − *j*))) (*p* ^*j* ∗ *monom* 1 *i*)) *ja* ∗ *int* (*X* ^*ja*))) *X* = ($\sum$ *ia*<*degree p* ∗ *h*. *monom* (*vec* (*degree p* ∗ *h*) (*poly.coeff* (*smult* (*int* (*M* ^(*h* − 1 − *j*))) (*p* ^*j* ∗ *monom* 1 *i*))) $ *ia*) *ia*)› *access-entry-in-int-poly-associated-to-vec dim-vec floor-div-match*)
have *?new-vec* $ *ia* = *poly.coeff* (*smult* (*int* (*M* ^(*h* − 1 − *j*))) (*p* ^*j* ∗ *monom* 1 *i*)) *ia*
if ∗:*ia* < *?dim-vec* **for** *ia*
using ∗ **by** *auto*
then have *same-coeffs*: *poly.coeff* ($\sum i$<*?dim-vec*. *monom* (*?new-vec* $ *i*) *i*) *ia* =
poly.coeff (*smult* (*int* (*M* ^(*h* − 1 − *j*))) (*p* ^*j* ∗ *monom* 1 *i*)) *ia*
if ∗:*ia* < *?dim-vec* **for** *ia*
using *same-coeffs-lhs* ∗
by *presburger*
have *all-poly-eq*: ⋀*p1 p2*::*int poly*. ⋀*d*. *degree p1* < *d* ⟹ *degree p2* < *d* ⟹ (⋀*i*. *i* < *d* ⟹ *coeff p1 i* = *coeff p2 i*) ⟹ *p1* = *p2*
by (*metis coeff-eq-0 linorder-neqE-nat nat-SN.gt-trans poly-eqI*)
have *deg-lt-1*: *degree* (*smult* (*int* (*M* ^(*h* − 1 − *j*))) (*p* ^*j* ∗ *monom* 1 *i*)) < *?dim-vec*
using *i-lt ipos*
by (*smt* (*verit*, *ccfv-threshold*) *Euclidean-Rings.div-eq-0-iff M-gt Nat.diff-cancel Polynomial.degree-power-eq add.right-neutral add-diff-cancel-left' add-diff-inverse-nat degree-monom-eq degree-mult-eq degree-smult-eq diff-add-inverse div-mult2-eq gr-implies-not0 j-lt j-lt less-nat-zero-code monom-eq-0-iff monom-hom.hom-0-iff msrevs*(1) *mult-eq-0-iff mult-is-0 of-int-of-nat-eq of-nat-0-less-iff of-nat-0-less-iff p-neq semiring-1-no-zero-divisors-class.power-not-zero zero-less-divide-1-iff zero-less-power*)
have *poly-coeff-lt*: *degree* (*monom* (*?new-vec* $ *i*) *i*) < *?dim-vec*
if ∗: *i* < *?dim-vec* **for** *i*
using ∗
by (*metis bot-nat-0.not-eq-extremum degree-0 degree-monom-eq less-nat-zero-code monom-hom.hom-0-iff*)
have *poly-degree-helper*:⋀ *n*. ⋀*f*::*nat* ⇒ *int poly*. (⋀*i*. *i* < *d* ⟹ *degree* (*f i*) < *n*) ⟹ *degree* ($\sum i$ < *d*. (*f i*)) < *n* **if** ∗: *d* > *0* **for** *d*
using ∗
proof (*induct d*)
case *0*
then show *?case* **by** *blast*
next
case (*Suc d*)
{assume ∗: *d* = *0*
then have *degree* (*sum f* {..<*Suc d*}) < *n*
using *Suc.prems* **by** *auto*
} moreover {assume ∗: *d* > *0*
then have *degree* (*sum f* {..<*d*}) < *n*

```
      using Suc by simp
    then have degree (sum f {..<Suc d}) < n
      using Suc
      by (simp add: degree-add-less)
  }
  ultimately show ?case by blast
 qed
 have deg-lt-2: degree (∑ i<?dim-vec. monom (?new-vec $ i) i) < ?dim-vec
  using deg-gt h-gt1 poly-degree-helper poly-coeff-lt
  by force
 then have poly-of-interest-is2: (∑ i<?dim-vec. monom (?new-vec $ i) i) =
(smult (int (M ^ (h - 1 - j))) (p ^ j * monom 1 i))
  using all-poly-eq[OF deg-lt-1 deg-lt-2] same-coeffs
  by (metis (no-types, lifting))
 have poly.coeff (smult (int (M ^ (h - 1 - j))) (p ^ j * monom 1 i)) ja =
  int (M ^ (h - 1 - j))*(poly.coeff (p ^ j * monom 1 i)) ja if *: ja < ?dim-vec
for ja::nat
  by auto
 have poly-is: poly (p ^ j * monom 1 i) x0 = (poly p x0)^j*(poly (monom 1 i)
x0)
  by auto
 then have poly-is-2: poly (smult (int (M ^ (h - 1 - j))) (p ^ j * monom 1 i))
x0
  = int (M ^ (h - 1 - j))*poly (p ^ j * monom 1 i) x0
  by simp
 from p-zero-mod-M have ‹int M dvd poly p x0›
  by presburger
 then obtain k where ‹poly p x0 = int M * k› ..
 then have poly (p ^ j * monom 1 i) x0 = M^j*(k)^j*(poly (monom 1 i) x0)
  using poly-is
  by (simp add: power-mult-distrib)
 then have j-gt-0-exk: ∃k::int. poly (p ^ j * monom 1 i) x0 = M^j*k if j > 0
  using that by auto
 {assume *: j > 0
  then have ∃k::int. poly (p ^ j * monom 1 i) x0 = M^j*k
   using j-gt-0-exk[OF *] by auto
  then obtain k::int where k-prop: poly (p ^ j * monom 1 i) x0 = M^j*k
   by blast
  have poly (smult (int (M ^ (h - 1 - j))) (p ^ j * monom 1 i)) x0 =
  (int (M ^ (h - 1 - j))) * M^j*k
   using k-prop by auto
  also have ... = (int (M^(h-1)))*k
   using hj
   by (simp add: power-add)
  finally have ∃k::int. poly (smult (int (M ^ (h - 1 - j))) (p ^ j * monom 1
i)) x0 = (M^(h-1))*k
  by auto
 } moreover {assume *: j = 0
 then have int (M ^ (h - 1 - j)) = int (M ^ (h -1))
```

by *auto*
then have $\exists k$::*int*. *poly* (*smult* (*int* ($M \hat{} (h - 1 - j)$)) ($p \hat{} j * monom\ 1\ i$)) *x0* = ($M\hat{}(h-1)$)$*k$
using *poly-is-2* **by** *auto*
}
ultimately have $\exists k$::*int*. *poly* (*smult* (*int* ($M \hat{} (h - 1 - j)$)) ($p \hat{} j * monom$ *1 i*)) *x0* = ($M\hat{}(h-1)$)$*k$
by *fast*
then show *?thesis*
using *poly-of-interest-is poly-of-interest-is2* **by** *auto*
qed

lemma *form-basis-coppersmith-zero-mod-M*:
fixes *M X*::*nat*
assumes *p-neq*: $p \neq 0$
assumes *M-gt*: $M > 0$
assumes *X-gt*: $X > 0$
assumes *h-gt*: $h > 1$
assumes *p-zero-mod-M*: *poly p x0 mod M* = *0*
assumes *a-inset*: $a \in$ *set* (*form-basis-coppersmith p M X h*)
assumes *deg-gt*: *degree p* > *1*
shows *poly* (*int-poly-associated-to-vec a X*) *x0 mod* $M \hat{} (h - 1)$ = *0*
proof −
have $\exists i\ j.\ i <$ *degree* $p \wedge j < h \wedge a =$ *row-of-coppersmith-matrix p M X h i j*
using *a-inset* **unfolding** *form-basis-coppersmith-def form-basis-coppersmith-aux-def*
by *fastforce*
then show *?thesis*
using *assms row-of-cs-matrix-zero-mod-M*[*OF p-neq M-gt X-gt h-gt p-zero-mod-M deg-gt*]
by *auto*
qed

6.1.8 Determinant of matrix

lemma *determinant-bound-arithmetic-helper*:
fixes *k*:: *nat*
shows ($\prod$ *j*<(*w*+*1*). *k*$\hat{}$*j*) = *sqrt* (*k*$\hat{}$(*w*∗(*w*+*1*)))
proof (*induct w*)
case *0*
then show *?case*
by *simp*
next
case (*Suc w*)
have *prod* (($\hat{}$) *k*) {..<*Suc w* + *1*} = *k*$\hat{}$(*w*+*1*)∗(*prod* (($\hat{}$) *k*) {..< *w* + *1*})
using *Groups.mult-ac*(*2*) **by** *auto*
also have ... = *k*$\hat{}$(*w*+*1*)∗*sqrt* (*k* $\hat{}$ (*w* ∗ (*w* + *1*)))
by (*metis Suc of-nat-mult*)
also have ... = *sqrt* ((*k*$\hat{}$(*w*+*1*))$\hat{}$*2*)∗*sqrt* (*k* $\hat{}$ (*w* ∗ (*w* + *1*)))
by (*metis of-nat-0-le-iff pos2 real-root-pos2 semiring-1-class.of-nat-power sqrt-def*)

```
  also have ... = sqrt ((k^(2*w+2)))*sqrt (k ^ (w * (w + 1)))
    by (metis (no-types, opaque-lifting) Suc-eq-plus1 add.commute mult-Suc-right
power-even-eq)
  also have ... = sqrt (k^((2*w+2)+(w * (w + 1))))
    by (metis Num.of-nat-simps(5) power-add real-sqrt-mult)
  also have ... = sqrt (k^((2*w+2+w^2 + w)))
    by (simp add: power2-eq-square)
  also have ... = sqrt (k^((w^2 + 3*w+2)))
    by (smt (z3) eval-nat-numeral(3) more-arith-simps(11) mult-Suc power-add
power-commuting-commutes)
  finally have prod ((^) k) {..<Suc w + 1} = sqrt (k^((w+1)*(w+2)))
   by (smt (verit, del-insts) Groups.mult-ac(2) Num.of-nat-simps(5) Rings.ring-distribs(2)
‹prod ((^) k) {..<Suc w + 1} = k ^ (w + 1) * prod ((^) k) {..<w + 1}› ‹real
(k ^ (w + 1) * prod ((^) k) {..<w + 1}) = real (k ^ (w + 1)) * sqrt (real (k ^
(w * (w + 1))))› ‹real (k ^ (w + 1)) * sqrt (real (k ^ (w * (w + 1)))) = sqrt
(real ((k ^ (w + 1))^2)) * sqrt (real (k ^ (w * (w + 1))))› power-add power-even-eq
real-sqrt-mult)
  then show ?case by auto
qed

lemma det-of-form-coppersmith-matrix:
  fixes M X:: nat
  assumes M > 0
  assumes X > 0
  assumes d-is: d = degree p
  assumes monic-poly: coeff p d = 1
  assumes h-gt: h > 1
  assumes deg-gt: degree p > 1
  shows det (form-coppersmith-matrix p M X h) =
    (root 2 (M^((h-1)*d*h))) *  (root 2 (X^((d*h-1)*d*h)))
proof -
  let ?A = form-coppersmith-matrix p M X h
  have matrix-dims:?A ∈ carrier-mat (d*h) (d*h)
    unfolding form-coppersmith-matrix-def using d-is by auto
  then have det-eq: det ?A = prod-list (diag-mat ?A)
   using assms det-lower-triangular no-zeros-on-diagonal-helper form-coppersmith-matrix-is-lower-triangular
    by (smt (verit, del-insts) nat-SN.default-gt-zero nat-SN.gt-trans)
  have j = nat (floor (k/d)) ⟹ degree (p^j* (monom (1::int) (k - d*j))) = k if
k-lt: k < d*h for k j
  proof -
    assume j-eq: j = nat (floor (k/d))
    have deg-lhs: degree (p^j) = d*j
      using d-is
    by (metis Missing-Polynomial.degree-power-eq deg-gt degree-0 not-one-less-zero)

    have k - d*j ≥ 0
      using j-eq by blast
    then have deg-rhs: degree (monom (1::int) (k - d*j)) = k - d*j
      by (simp add: degree-monom-eq)
```

show *degree* (*p*^*j** (*monom* (*1*::*int*) (*k* − *d***j*))) = *k*
using *deg-lhs deg-rhs*
by (*simp add*: *assms*(*3*) *degree-mult-eq floor-divide-of-nat-eq j-eq*)
qed
then have *leading-coeff-1*: *j* = *nat* (*floor* (*k*/*d*)) ⟹ *coeff* (*p*^*j** (*monom* (*1*::*int*) (*k* − *d***j*))) *k* = *1* **if** *k-lt*: *k* < *d***h* **for** *k j*
using *monic-poly*
by (*metis lead-coeff-monom assms*(*3*) *k-lt monic-mult monic-power*)
have *all-elems-diag*: ⋀*k*. *k* < *d***h* ⟹ (*?A* $$ (*k*, *k*)) = (*let j* = *nat* (*floor* (*k*/*d*)); *i* = (*k* − *d***j*) *in* (*M*^(*h*−*1*−*j*))*(*coeff* (*p*^*j** (*monom* (*1*::*int*) *i*)) *k*)*X*^*k*)
unfolding *form-coppersmith-matrix-def* **using** *matrix-dims d-is*
by (*metis* (*no-types*, *lifting*) *index-mat*(*1*) *old.prod.case semiring-1-class.of-nat-power*)
then have *diag-rewrite*: ⋀*k*. *k* < *d***h* ⟹ (*?A* $$ (*k*, *k*)) = (*M*^(*h*−*1*−*nat* (*floor* (*k*/*d*))))*X*^*k*
using *leading-coeff-1*
by *simp*
have *prod-list* (*diag-mat ?A*) = (∏ *k*<*d***h*. *?A*$$(*k*, *k*))
by (*metis atLeast0LessThan carrier-matD*(*1*) *matrix-dims prod-list-diag-prod*)
then have *prod-list-is*: *prod-list* (*diag-mat ?A*) = (∏ *k*<*d***h*. (*M*^(*h*−*1*−*nat* (*floor* (*k*/*d*))))*X*^*k*)
using *diag-rewrite matrix-dims* **by** *simp*
have *prod-split*: (∏ *k*<*d***h*. (*M*^(*h*−*1*−*nat* (*floor* (*k*/*d*))))*X*^*k*) = (∏ *k*<*d***h*. (*M*^(*h*−*1*−*nat* (*floor* (*k*/*d*)))))* (∏ *k*<*d***h*. *X*^*k*)
by (*metis* (*no-types*) *prod.distrib*)

have *lhs-prod*: (∏ *k*<*d***h*. (*M*^(*h*−*1*−*nat* (*floor* (*k*/*d*))))) = (*root 2* (*M*^((*h*−*1*)**d***h*)))
using *deg-gt d-is h-gt*
proof (*induct d***h arbitrary*: *h rule*: *less-induct*)
case *less*
have *prod-split1*: ⋀*f*. *real* (∏ *k*<*d* * *2*. *f d*) =
real (∏ *k*<*d*. *f d*) * *real* (∏ *k*∈{*d*..<*2***d*}. *f d*)
using *deg-gt*
by (*smt* (*verit*, *ccfv-SIG*) *atLeast0LessThan le-add-same-cancel1 mult-2 mult-2-right of-nat-mult prod.atLeastLessThan-concat zero-order*(*1*))
{assume *: *h* = *2*
have *prod-is*: *real* (∏ *k*<*d* * *2*. *M* ^ (*2* − *1* − *nat* ⌊*real k* / *real d*⌋)) =
real (∏ *k*<*d*. *M* ^ (*2* − *1* − *nat* ⌊*real k* / *real d*⌋)) * *real* (∏ *k*∈{*d*..<*2***d*}. *M* ^ (*2* − *1* − *nat* ⌊*real k* / *real d*⌋))
using *prod-split1*
by (*smt* (*verit*) *le-add2 lessThan-atLeast0 mult.commute mult-2-right of-nat-mult prod.atLeastLessThan-concat zero-order*(*1*))
have ⋀*k*. *k*<*d* ⟹ *nat* ⌊*real k* / *real d*⌋ = *0*
by *auto*
then have *h1*: *real* (∏ *k*<*d*. *M* ^ (*2* − *1* − *nat* ⌊*real k* / *real d*⌋)) = (*M* ^ (*d***(*2* − *1*)))
by *simp*

have ⋀*k*. *k*∈{*d*..<*2***d*} ⟹ *nat* ⌊*real k* / *real d*⌋ = *1*

by (*simp add*: *floor-divide-of-nat-eq nat-div-eq-Suc-0-iff*)
then have *h2*: *real* ($\prod k{\in}\{d..<2{*}d\}$. $M \hat{} (2 - 1 - nat \lfloor real\ k\ /\ real\ d \rfloor)$) = *real* $M \hat{} (d{*}(2 - 1 - 1))$
by *auto*
have *real* ($\prod k{<}d * 2$. $M \hat{} (2 - 1 - nat \lfloor real\ k\ /\ real\ d \rfloor)$) = *real* $M \hat{} (d{*}(2 - 1 - 1)) * real\ (M \hat{} (d{*}(2 - 1)))$
using *h1 h2 prod-is* **by** *simp*
also have ... =
root 2 (*real* $(M \hat{} ((2 - 1) * d * 2)))$
using *Groups.mult-ac*(3) *Suc-1 Suc-eq-plus1 add-diff-cancel-left' of-nat-0-le-iff pos2 power-0 power-mult real-root-pos2 semiring-1-class.of-nat-power verit-minus-simplify*(1) *verit-prod-simplify*(2)
by (*smt* (*verit*, *ccfv-threshold*) *h1 h2 of-nat-mult prod-is*)
finally have *real* ($\prod k{<}d * h$. $M \hat{} (h - 1 - nat \lfloor real\ k\ /\ real\ d \rfloor)$) = *root 2* (*real* $(M \hat{} ((h - 1) * d * h)))$
using $*$ **by** *auto*
} moreover {assume $*$: $h > 2$
have *prod-split*: *real* ($\prod k{<}d * h$. $M \hat{} (h - 1 - nat \lfloor real\ k\ /\ real\ d \rfloor)$) = *real* ($\prod k{<}d{*}(h{-}1)$. $M \hat{} (h - 1 - nat \lfloor real\ k\ /\ real\ d \rfloor)$) * *real* ($\prod k{\in}\{d{*}(h{-}1)..<d * h\}$. $M \hat{} (h - 1 - nat \lfloor real\ k\ /\ real\ d \rfloor)$)
by (*smt* (*verit*) *Num.of-nat-simps*(5) *atLeast0LessThan diff-le-self mult.commute mult-less-cancel1 prod.atLeastLessThan-concat verit-comp-simplify1*(3) *zero-order*(1))
have *ind-dec*: $(d * (h - 1)) < d * h$
using *less.prems* **by** *simp*
have *ind-still*: $1 < h - 1$ **using** $*$ **by** *auto*
have *h1-aux*: *real* ($\prod k{<}d * (h{-}1)$. $M \hat{} ((h{-}1) - 1 - nat \lfloor real\ k\ /\ real\ d \rfloor)$) = *root 2* (*real* $(M \hat{} (((h{-}1) - 1) * d * (h{-}1))))$
using *less.hyps*[*OF ind-dec less.prems*(1) *less.prems*(2) *ind-still*]
by *blast*
have $2{*}d{*}(h{-}1){+}((h{-}1) - 1) * d * (h{-}1) = d * (h{-}1){*}((h{-}1){-}1{+}2)$
by *simp*
then have *arith-helper*: $2{*}d{*}(h{-}1){+}((h{-}1) - 1) * d * (h{-}1) = d * (h{-}1){*}(h)$
by (*metis Suc-1 Suc-diff-Suc Suc-eq-plus1 add-Suc-right diff-diff-left le-add-diff-inverse less-imp-le-nat local.less*(4) *plus-1-eq-Suc*)
have *gteq*: $(h{-}1) - 1 - nat \lfloor real\ k\ /\ real\ d \rfloor \geq 0$ **if** *k-is*: $k{<}d * (h{-}1)$ **for** k
proof −
have $k/d <{d * (h{-}1)}/d$
using *k-is less.prems*(1−2)
by (*metis divide-strict-right-mono ind-dec less-imp-of-nat-less nat-mult-less-cancel-disj of-nat-0-less-iff*)
then show *?thesis* **by** *blast*
qed
have *sum-pow*: $b \geq 0 \Longrightarrow c \geq 0 \Longrightarrow M \hat{} (b + c) = M\hat{}b * M\hat{}c$ **for** $b\ c$
using *power-add* **by** *blast*
then have $M \hat{} ((h{-}1){-}\ nat \lfloor real\ k\ /\ real\ d \rfloor) = M{*}\ M \hat{} ((h{-}1) - 1 - nat \lfloor real\ k\ /\ real\ d \rfloor)$ **if** *k-is*: $k{<}d * (h{-}1)$ **for** k
proof −

have $(h-1)-$ *nat* $\lfloor$*real k / real d*$\rfloor$ = 1 + $(h-1)-$ 1 $-$*nat* $\lfloor$*real k / real d*$\rfloor$
using *gteq* **by** *auto*
then have $M \hat{} ((h-1)-$ *nat* $\lfloor$*real k / real d*$\rfloor) = M \hat{} (1 + (h-1)-$ 1 $-$*nat* $\lfloor$*real k / real d*$\rfloor)$
by *argo*
then show *?thesis* **using** *gteq sum-pow*[*of* 1 $(h - 1) - 1 -$ *nat* $\lfloor$*real k / real d*$\rfloor$]
by (*metis* (*no-types*, *opaque-lifting*) *Groups.mult-ac*(2) *diff-commute div-le-dividend floor-divide-of-nat-eq k-is le-add-diff-inverse less-irrefl-nat less-mult-imp-div-less nat-int nonzero-mult-div-cancel-right power-one-right verit-prod-simplify*(1) *zero-less-diff zero-order*(1))
qed
then have *real* $(\prod k{<}d * (h-1).\ M \hat{} ((h-1)-$ *nat* $\lfloor$*real k / real d*$\rfloor)) =$
real $(\prod k{<}d * (h-1).\ M{*}\ M \hat{} ((h-1) - 1 -$ *nat* $\lfloor$*real k / real d*$\rfloor))$
by *simp*
moreover have ... = *real* $(\prod k{<}d * (h-1).\ M){*}$*real* $(\prod k{<}d * (h-1).\ M \hat{} ((h-1) - 1 -$ *nat* $\lfloor$*real k / real d*$\rfloor))$
by *auto*
moreover have ... =
$M \hat{} (d{*}(h-1)){*}$*root 2* (*real* $(M \hat{} (((h-1) - 1) * d * (h-1))))$
by (*metis card-lessThan h1-aux prod-constant*)
moreover have ... =
root 2 $M \hat{} (2{*}d{*}(h-1)){*}$*root 2* (*real* $(M \hat{} (((h-1) - 1) * d * (h-1))))$
by (*simp add*: *power-mult*)
moreover have ... = *root 2* (*real* $M \hat{} (2{*}d{*}(h-1)){*}(M \hat{} (((h-1) - 1) * d * (h-1))))$
using *pos2 real-root-mult real-root-power* **by** *presburger*
moreover have ... = *root 2* (*real* $(M \hat{} (\ 2{*}d{*}(h-1){+}((h-1) - 1) * d * (h-1))))$
by (*simp add*: *power-add*)
moreover have ... = *root 2* (*real* $(M \hat{} ((h - 1) * d * h)))$
using *arith-helper*
by (*metis Groups.mult-ac*(2))
ultimately have *h1*: *real* $(\prod k{<}d * (h-1).\ M \hat{} ((h-1)-$ *nat* $\lfloor$*real k / real d*$\rfloor)) =$ *root 2* (*real* $(M \hat{} ((h - 1) * d * h)))$
by *argo*
have *nat* $\lfloor$*real k / real d*$\rfloor = h-1$ **if** *k-in*: $k{\in}\{d{*}(h-1)..{<}d * h\}$ **for** *k*
proof $-$
have $k \geq d{*}(h-1)$
using *k-in* **by** *auto*
then have *real k / d* $\geq$ *real* $(d{*}(h-1))/d$
using *less.prems*(1$-$2)
by (*smt* (*verit*, *del-insts*) *frac-le ind-dec linordered-nonzero-semiring-class.of-nat-mono mult-less-cancel1 of-nat-0-le-iff of-nat-0-less-iff*)
then have *b1*: *real k / real d* $\geq h - 1$
using *ind-dec* **by** *auto*
have $k < d{*}h$
using *k-in* **by** *auto*
then have *real k / d* $<$ *real* $(d{*}h)/d$

```
    using less.prems(1-2)
   by (metis divide-strict-right-mono ind-dec less-imp-of-nat-less mult-less-cancel1
of-nat-0-less-iff)
    then have b2: real k / real d < h
     using less.prems
     by simp
    show ?thesis using b1 b2
     by (simp add: floor-eq4)
  qed
  then have h2: real (∏k∈{d*(h-1)..<d * h}. M ^ (h - 1 - nat ⌊real k /
real d⌋)) = 1
    by simp
  then have real (∏k<d * h. M ^ (h - 1 - nat ⌊real k / real d⌋)) =
    real (∏k<d * (h - 1). M ^ (h - 1 - nat ⌊real k / real d⌋))
    using prod-split
    by (smt (verit, del-insts) div-by-1 nonzero-mult-div-cancel-right)
  then have real (∏k<d * h. M ^ (h - 1 - nat ⌊real k / real d⌋)) =
     root 2 (real (M ^ ((h - 1) * d * h)))
    using h1
    by presburger
 }
 ultimately show ?case using less.prems(3)
  by linarith
qed

have dh-gt-1: d * h > 1
 using deg-gt h-gt
 by (metis One-nat-def assms(3) less-1-mult)
let ?k = d*h - 1
have rhs-prod: (∏k<d*h. X^k) = root 2 (X^((d*h-1)*d*h))
 using dh-gt-1 determinant-bound-arithmetic-helper[of X ?k]
 by (metis (no-types, lifting) Groups.add-ac(2) le-add-diff-inverse less-imp-le-nat
more-arith-simps(11) sqrt-def)
have (∏k<d*h. (M^(h-1-nat (floor (k/d))))*X^k) = (root 2 (M^((h-1)*d*h)))
* (root 2 (X^((d*h-1)*d*h)))
 using prod-split lhs-prod rhs-prod
 by simp
then show ?thesis
 using prod-list-is det-eq
 by linarith
qed

lemma det-of-matrix:
 fixes M X:: nat
 assumes M > 0
 assumes X > 0
 assumes d > 1
 assumes d-is: d = degree p
 assumes monic-poly: coeff p d = 1
```

assumes *h-gt*: $h > 1$
shows $det\ (vec\text{-}list\text{-}to\text{-}square\text{-}mat\ (form\text{-}basis\text{-}coppersmith\ p\ M\ X\ h)) =$
$(root\ 2\ (M\hat{}((h-1)*d*h))) *\ (root\ 2\ (X\hat{}((d*h-1)*d*h)))$
proof −
have *dims-1*: $vec\text{-}list\text{-}to\text{-}square\text{-}mat\ (form\text{-}basis\text{-}coppersmith\ p\ M\ X\ h) \in carrier\text{-}mat\ (d*h)\ (d*h)$
using *h-gt dim-vector-form-basis-coppersmith length-form-basis-coppersmith*
using *assms*(*4*) *coppersmith-matrix-carrier-mat* **by** *simp*
have *dims-2*: $form\text{-}coppersmith\text{-}matrix\ p\ M\ X\ h \in carrier\text{-}mat\ (d*h)\ (d*h)$
unfolding *form-coppersmith-matrix-def* **using** *d-is* **by** *auto*
have $\bigwedge r\ c.\ r < degree\ p * h \Longrightarrow$
$c < degree\ p * h \Longrightarrow$
$vec\text{-}list\text{-}to\text{-}square\text{-}mat\ (form\text{-}basis\text{-}coppersmith\ p\ M\ X\ h)\ \$\$\ (r,\ c) =$
$form\text{-}coppersmith\text{-}matrix\ p\ M\ X\ h\ \$\$\ (r,\ c)$
using *h-gt matrix-match*[*of - p h - M X*] **by** *blast*
then have $vec\text{-}list\text{-}to\text{-}square\text{-}mat\ (form\text{-}basis\text{-}coppersmith\ p\ M\ X\ h) = form\text{-}coppersmith\text{-}matrix\ p\ M\ X\ h$
using *dims-1 dims-2 d-is* **by** *auto*
then show *?thesis*
using *det-of-form-coppersmith-matrix assms*
by *metis*
qed

6.2 Top-level proof

definition $root\text{-}bound :: nat \Rightarrow nat \Rightarrow real \Rightarrow real$
where $root\text{-}bound\ M\ d\ eps = 1/2*(M\ powr\ (1/d - eps))$

6.2.1 Arithmetic

lemma *epsilon-bounded-below*:
assumes $d > 0$
assumes $eps > 0$
assumes $d*h-1 > 0$
assumes $d = degree\ p$
assumes $h = calculate\text{-}h\text{-}coppersmith\ p\ eps$
shows $eps \geq (d-1)/(d*(d*h-1))$
proof −
have $h \geq ((d-1)/(d*(eps)) + 1)/d$
using *assms* **unfolding** *calculate-h-coppersmith-def calculate-h-coppersmith-aux-def*
by (*smt* (*verit*, *ccfv-SIG*) *Num.of-nat-simps*(*2*) *One-nat-def divide-less-cancel nat-less-real-le of-nat-0-less-iff of-nat-add of-nat-ceiling of-nat-diff of-nat-le-iff of-nat-less-0-iff plus-1-eq-Suc*)
then have $d*h \geq ((d-1)/(d*(eps)) + 1)$
using *assms*
by (*metis Groups.mult-ac*(*2*) *Num.of-nat-simps*(*5*) *less-divide-eq of-nat-0-less-iff verit-comp-simplify1*(*3*))
then have $d*h - 1 \geq (d-1)/(d*(eps))$
using *assms*
by *linarith*

```
  then have (d*h-1)*d*(eps) ≥ d-1
    using assms(1-2)
    by (simp add: divide-le-eq)
  then show ?thesis
    using assms
    by (metis Groups.mult-ac(2) divide-le-eq mult-sign-intros(5) of-nat-0-less-iff)
qed

lemma z-arith:
  assumes x: (x::real) ≥ 0
  shows x / (1 + x) ≤ ln (1 + x)
proof -
  define g where g = (λz::real.  ln (1 + z) - z / 1 + z)
  define g' where g' = (λz::real. 1 / (1 + z))
  have gz: g 0 ≥ 0
    unfolding g-def by auto
  have 1: z ∈ {0..x} ⟹ (g has-real-derivative (g' z)) (at z) for z
    unfolding g-def g'-def
    by (auto intro!: derivative-eq-intros DERIV-powr)
  have 2: g' z ≥ 0 if z:z ∈ {0..x} for z
    unfolding g'-def
    using z by auto
  have g 0 ≤ g x
    by (intro deriv-nonneg-imp-mono[OF 1 2 x])
  thus ?thesis using gz unfolding g-def
    by (smt (verit, ccfv-SIG) assms divide-minus-left ln-diff-le)
qed

lemma powr-divide-both:
  assumes (a::real) ≥ 0 x > 0 b powr y ≥ 1
  assumes le: a powr x ≥ b powr y
  shows a ≥ b powr (y / x)
proof -
  have 0 ≤ 1/x using assms by auto
  from powr-mono-both[OF this - assms(3) le, of 1 / x]
  show ?thesis
    using assms by (auto simp add:powr-powr)
qed

lemma coppersmith-arithmetic-convergence-1:
  fixes y:: real
  assumes y: y ≥ 1 / 0.18
  shows 0 ≤ 1.414 - (1 + y) powr (1 / y)
proof -
  define g where g = (λz::real.  1.414 - (1 + z) powr (1 / z))

  define g' where g' = (λz::real. -((1 + z) powr (1 / z) *
      (1 / (z * (1 + z)) - ln (1 + z) / (z * z))))
```

```
  have a:(1414 / 10 ^ 3) powr 50 ≥ ((1 + 1 / (18 / 10^2))::real) powr 9
    by (auto simp add: field-simps)
  have (1414 / 10 ^ 3) ≥ ((1 + 1 / (18 / 10^2))::real) powr (9 / 50)
    apply(intro powr-divide-both[OF - - - a])
    by auto
  then have gz: g (1 / 0.18) ≥ 0
    unfolding g-def by auto

  have 1: z ∈ {1 / (18 / 10^2)..y} ⟹ (g has-real-derivative (g' z)) (at z) for z
    unfolding g-def g'-def
    by (auto intro!: derivative-eq-intros DERIV-powr)
  have 2: g' z ≥ 0 if z:z ∈ {1 / (18 / 10^2)..y} for z
  proof −
    have zpos: z > 0 using z by auto
    have 1: (1 + z) powr (1 / z) ≥ 0
      by simp
    have z / (1 + z) ≤ ln (1 + z) using z-arith zpos by auto
    then have 2: (1 / (z * (1 + z)) − ln (1 + z) / (z * z)) ≤ 0
      apply (simp add: field-simps zpos)
       by (metis (no-types, opaque-lifting) distrib-left ge-refl more-arith-simps(6)
nonzero-divide-mult-cancel-left not-less times-divide-eq-right zpos)
    have (1 + z) powr (1 / z) * (1 / (z * (1 + z)) − ln (1 + z) / (z * z)) ≤ 0
      using 1 2 mult-nonneg-nonpos
      by blast
    then show ?thesis
      unfolding g'-def
      by auto
  qed
  have g (1 / 0.18) ≤ g y
    by (intro deriv-nonneg-imp-mono[OF 1 2 y])
  thus ?thesis using gz unfolding g-def by auto
qed

lemma coppersmith-arithmetic-convergence:
  fixes x:: real
  assumes x: 0 < x x ≤ 0.18
  shows (1 + (1/x)) powr (x) ≤ sqrt 2
proof −
  have 1 / x ≥ 1 / 0.18
    using x by (auto simp add: le-divide-eq-numeral(1))
  from coppersmith-arithmetic-convergence-1[OF this]
  have 1: (1 + 1 / x) powr (1 / (1 / x)) ≤ 1.414
    using x by auto

  have 707 powr 2 ≤ (sqrt 2 * 500) powr 2
    by (auto simp add: field-simps)
  from powr-divide-both[OF - - - this]
  have 707 ≤ sqrt 2 * 500
    by auto
```

thus *?thesis*
using *assms 1* **by** *auto*
qed

6.2.2 Main results

lemma *coppersmith-finds-small-roots*:
fixes *p f*:: *int poly*
fixes *M X*:: *nat*
fixes *x0*:: *int*
fixes *eps*::*real*
assumes *zero-mod-M*: *poly p x0 mod M = 0*
assumes *d-is*: *d = degree p*
assumes *d-gt*: *d > 1*
assumes *monic-poly*: *coeff p d = 1*
assumes *X-lt*: *X < root-bound M d eps*
assumes *M-gt*: *M > 0*
assumes *X-gt-zero*: *X > 0*
assumes *x0-le*: *abs x0 ≤ X*
assumes *eps-le*: *eps ≤ 0.18∗(d−1)/d*
assumes *eps-lt*: *eps < 1/(real (degree p))*
assumes *eps-gt*: *eps > 0*
assumes *f-is*: *f = coppersmith p M X eps*
shows *poly f x0 = 0*
proof −
define *h* **where** *h = calculate-h-coppersmith p eps*
let *?cs-basis = form-basis-coppersmith p M X h*

have *h-gt-0*: *h > 0*
unfolding *h-def*
using *calculate-h-coppersmith-aux-gteq1 assms* **unfolding** *calculate-h-coppersmith-def*

by *fastforce*

then have *dh-gt-0*: *d∗h > 0*
unfolding *h-def*
using *d-gt* **by** *auto*

have *h-gt*: *h > 1*
unfolding *h-def*
using *One-nat-def eps-gt d-is d-gt calculate-h-coppersmith-aux-gt1 calculate-h-coppersmith-def eps-lt one-less-nat-eq*
by *presburger*

have *dim-form-basis*: *length ?cs-basis = d∗h*
using *length-form-basis-coppersmith*[*of h p M X*] *d-is d-gt h-gt*
using *h-def* **by** *blast*

have *li1*: *set* (*map of-int-vec ?cs-basis*) ⊆ *carrier-vec* (*d* ∗ *h*)
apply *clarsimp*
by (*metis assms*(*2*) *carrier-vec-dim-vec dim-vector-form-basis-coppersmith in-set-conv-nth local.dim-form-basis*)

have *dist*: *distinct ?cs-basis*
using *h-def*
by (*smt* (*verit*, *ccfv-SIG*) *X-gt-zero assms*(*2*) *assms*(*3*) *assms*(*4*) *assms*(*6*) *form-basis-coppersmith-distinct h-gt leading-coeff-0-iff less-imp-le-nat nat-SN.gt-trans zero-less-one-class.zero-less-one*)
then have *li2*: *distinct* ((*map of-int-vec ?cs-basis*)::*rat vec list*)
using *casted-distinct-is-distinct* **by** *blast*

have *form-basis-lin-ind*: ¬ *module.lin-dep class-ring* (*module-vec TYPE*(*int*) (*d*∗*h*))
(*set* (*form-basis-coppersmith p M X h*))
using *assms form-basis-coppersmith-lin-ind h-gt-0*
by *simp*

then have *li3*:*module.lin-indpt class-ring*
(*module-vec TYPE*(*rat*) (*d* ∗ *h*))
(*set* (*map of-int-vec* (*?cs-basis*)))
using *h-def*
by (*metis dist assms*(*2*) *casting-lin-comb-helper dim-vector-form-basis-coppersmith distinct-Ex1 local.dim-form-basis*)

from *form-basis-coppersmith-zero-mod-M*[*OF - M-gt X-gt-zero h-gt zero-mod-M*]
have *2*: ⋀*a*. *a* ∈ *set ?cs-basis* ⟹ *poly* (*int-poly-associated-to-vec a X*) *x0 mod M* ^ (*h* − *1*) = *0*
using *h-def*
by (*metis d-is d-gt degree-0 semiring-norm*(*136*))

have *pmod*: *poly* (*p* ^ (*h* − *1*)) *x0 mod* (*M* ^ (*h* − *1*)) = *0*
proof −
from *zero-mod-M*
obtain *k* **where** *poly p x0* = *k* ∗*int* *M*
by *fastforce*
then have (*poly p x0*) ^ (*h* − *1*) = (*k* ∗*int* *M*)^(*h*−*1*) **by** *auto*
moreover have ... = (*k*^(*h*−*1*) ∗ (*M*^(*h*−*1*)))
by (*simp add*: *power-mult-distrib*)
ultimately show *?thesis*
by *auto*
qed

have *1*: (⋀*v j*. *v* ∈ *set ?cs-basis* ⟹
j < *d* ∗ *h* ⟹ *v* $ *j mod X* ^ *j* = *0*)
using *form-basis-coppersmith-div-by-Xpow h-def*
by (*metis Groups.mult-ac*(*2*) *assms*(*2*))

have *real-of-int*

```
    (det (vec-list-to-square-mat ?cs-basis)) =
     root 2 ((M ^ ((h − 1) ∗ d ∗ h))) ∗
     root 2 (real (X ^ ((d ∗ h − 1) ∗ d ∗ h))) (is ?lhs = ?rhs)
     apply (subst det-of-matrix[OF M-gt X-gt-zero d-gt d-is monic-poly h-gt])
     by auto
   then have (?lhs)^2 = (?rhs)^2
     by auto
   then have drw: (det (mat-of-rows (d ∗ h) ?cs-basis))^2 =
     M ^ ((h − 1) ∗ d ∗ h) ∗ X ^ ((d ∗ h − 1) ∗ d ∗ h)
     apply (clarsimp simp add: field-simps)
    by (smt (verit, del-insts) local.dim-form-basis of-int-eq-iff of-int-mult of-int-of-nat-eq
of-int-power vec-list-to-square-mat-def)

   let ?cdh = root (d∗h − 1) (d∗h)∗(root 2 2)

   let ?x = 1/real(d∗h−1)
   have x-inv: 1/?x = d∗h −1
     by simp
   have eps∗d ≤ (0.18∗(d−1)/d)∗d
     using d-gt eps-le
     by (metis le-simps(1) of-nat-0-le-iff of-nat-1 of-nat-diff times-left-mono)
   then have 9 + eps ∗ real d ∗ 50 ≤ real d ∗ 9
     using d-gt by force
   then have 1/((d−1)/(d∗eps) − 1 + 1) ≤ 0.18
     using d-gt by (simp add: field-simps)
   then have helper-ineq:1/(d∗((d−1)/(d∗eps) + 1)/d − 1) ≤ 0.18
     using d-gt by auto
   have h = ⌈((real d − 1) / (real d ∗ eps) + 1) / real d⌉
     unfolding h-def calculate-h-coppersmith-def calculate-h-coppersmith-aux-def
     using d-is
    by (metis calculate-h-coppersmith-aux-def calculate-h-coppersmith-def h-def h-gt-0
nat-eq-iff nat-neq-iff)
   then have h-gteq-helper: h ≥ ((d−1)/(d∗eps) + 1)/d
    by (metis (no-types, opaque-lifting) Num.of-nat-simps(2) d-gt le-simps(1) nat-eq-iff2
of-nat-0-le-iff of-nat-diff real-nat-ceiling-ge)
   then have d∗h ≥ d∗((d−1)/(d∗eps) + 1)/d
     using d-gt h-gt
     by (simp add: Groups.mult-ac(2) divide-le-eq)
   then have h-gteq-helper2: d∗h − 1 ≥ d∗((d−1)/(d∗eps) + 1)/d −1
     by linarith

   have inequality-helper:a / b > a if ∗: a > 0 ∧ b > 0 ∧ b < 1 for a b :: real
     using ∗
     using frac-less2 by fastforce
   have d∗eps < 1
     using eps-lt d-is d-gt
     by (auto simp add: field-simps)
   then have (d−1)/(d∗eps) > d−1
     using d-gt eps-gt inequality-helper[of d−1 d∗eps] by auto
```

then have *gteq*: $d*((d-1)/(d*eps) + 1)/d - 1 > 0$
using *d-gt* **by** *simp*
have $1/(d*h - 1) \leq 0.18$
using *h-gteq-helper2 gteq helper-ineq*
by (*smt* (*verit*, *ccfv-SIG*) *frac-le*)
then have $?x \leq 0.18$ **by** *blast*
then have $(1 + (1/?x))$ *powr* $(?x) \leq$ *sqrt 2*
using *coppersmith-arithmetic-convergence*
by (*smt* (*verit*, *best*) *gteq h-gteq-helper2 zero-compare-simps*(7))
then have $(d*h)$ *powr* $(1/(d*h - 1)::real) \leq$ *sqrt 2*
using *x-inv*
apply (*clarsimp simp add*: *field-simps*)
by (*metis Num.of-nat-simps*(3) *Num.of-nat-simps*(5) *Suc-pred' dh-gt-0 numeral-nat*(7))
then have *root* $(d*h - 1)$ $(d*h) \leq$ *sqrt 2*
using *gteq h-gteq-helper2 root-powr-inverse* **by** *force*
then have *root* $(d*h - 1)$ $(d*h)*(root\ 2\ 2) \leq sqrt\ 2\ *(root\ 2\ 2)$
by *simp*
then have $?cdh \leq 2$
using *sqrt-def* **by** *auto*
then have *root* $(d*h - 1)$ $(d*h)*(root\ 2\ 2)*\ X \leq 2*X$
using *X-gt-zero* **by** *simp*
then have
root $(d*h - 1)$ $(d*h)*(root\ 2\ 2)*\ X <$ *M powr* $(1/d - eps)$
using *X-lt* **unfolding** *root-bound-def* **by** *simp*
then have *arith*: $(root\ (d*h - 1)\ (d*h)*(root\ 2\ 2)*\ X)\ \hat{}\ (d * h * (d*h - 1))$
$< (M\ powr\ (1/d - eps))\ \hat{}\ (d * h * (d*h - 1))$
by (*smt* (*verit*, *del-insts*) *d-gt dh-gt-0 h-gt less-1-mult power-strict-mono mult-nonneg-nonneg nat-0-less-mult-iff of-nat-0-le-iff real-root-ge-zero zero-less-diff*)
have $2\ \hat{}\ (d * h *\ (d * h - 1)\ div\ 2) *$
$X\ \hat{}\ ((d * h - 1) * d * h)\ *\ (d * h)\ \hat{}\ (d * h)$
$<\ (M\ \hat{}\ ((h - 1)\ *\ (d * h)))$
proof −
have *root* $(d*h - 1)$ $(d*h)\ \hat{}\ (d * h * (d*h - 1))$
$=$ *(root* $(d*h - 1)$ $(d*h)\ \hat{}\ (d*h - 1))\ \hat{}\ (d * h)$
using *power-mult*
by (*metis Groups.mult-ac*(2))
also have $... = (d * h)\ \hat{}\ (d * h)$
apply (*subst real-root-pow-pos2*)
using *gteq h-gteq-helper2* **apply** *linarith*
by *auto*
finally have $*$: *root* $(d*h - 1)$ $(d*h)\ \hat{}\ (d * h * (d*h - 1)) = (d * h)\ \hat{}\ (d * h)$ **.**

have $**$: *root 2 2* $\hat{}\ (d * h * (d*h - 1))$
$=\ 2\ \hat{}\ (d * h * (d*h - 1)\ div\ 2)$
by (*smt* (*verit*, *del-insts*) *Suc-pred' dh-gt-0 even-Suc even-mult-iff real-sqrt-power-even sqrt-def*)

have *eps-bounded-below*:$eps \geq (d-1)/(d*(d*h-1))$
using *epsilon-bounded-below h-gt d-gt d-is*
by (*metis eps-gt bot-nat-0.not-eq-extremum div-by-0 h-def less-eq-real-def nat-0-less-mult-iff semiring-1-class.of-nat-0*)

then have $(1/d - eps) * (d*h - 1) \leq h - 1$
using *d-gt* **apply** (*simp add*: *field-simps*)
by (*smt* (*verit*, *ccfv-threshold*) *One-nat-def diff-mult-distrib2 h-gt le-eq-less-or-eq linordered-semiring-strict-class.mult-pos-pos nat-SN.gt-trans nat-mult-1-right nonzero-eq-divide-eq of-nat-0-less-iff of-nat-1 of-nat-diff of-nat-mult pos-divide-less-eq zero-less-diff*)
then have $(d * h) * ((1/d - eps) * (d*h - 1)) \leq (d * h) * (h - 1)$
by (*simp add*: *mult-left-mono*)
then have $(1/d - eps) * (d * h * (d*h - 1)) \leq ((h - 1) * (d * h))$
by (*auto simp add*: *field-simps*)
then have $M\ powr\ ((1/d - eps) * (d * h * (d*h - 1))) \leq M\ powr\ ((h - 1) * (d * h))$
using *M-gt powr-mono* **by** *auto*
then have ***: $(M\ powr\ (1/d - eps)) \hat{}\ (d * h * (d*h - 1)) \leq (M \hat{}\ ((h - 1) * (d * h)))$
by (*metis M-gt of-nat-0-less-iff of-nat-power-eq-of-nat-cancel-iff powr-gt-zero powr-powr powr-realpow*)
show *?thesis*
using *arith* * ** ***
apply (*simp add*: *power-mult-distrib*)
by (*smt* (*verit*) *Groups.mult-ac*(2) *Num.of-nat-simps*(2) *Num.of-nat-simps*(4) *Num.of-nat-simps*(5) *more-arith-simps*(11) *of-nat-power-less-of-nat-cancel-iff one-add-one semiring-1-class.of-nat-power*)
qed
then have $2 \hat{}\ (d * h * (d * h - 1)\ div\ 2) *$
$X \hat{}\ ((d * h - 1) * d * h) *$
$(d * h) \hat{}\ (d * h) *$
$M \hat{}\ ((h - 1) * d * h)$
$< (M \hat{}\ ((h - 1) * (d * h))) *$
$M \hat{}\ ((h - 1) * d * h)$
using *M-gt*
by (*clarsimp simp add*: *field-simps*)
then have $2 \hat{}\ (d * h * (d * h - 1)\ div\ 2) *$
$M \hat{}\ ((h - 1) * d * h) *$
$X \hat{}\ ((d * h - 1) * d * h) *$
$(d * h) \hat{}\ (d * h)$
$< (M \hat{}\ (h - 1)) \hat{}\ (2 * (d * h))$
by (*smt* (*verit*, *del-insts*) *Groups.mult-ac*(2) *Groups.mult-ac*(3) *numeral-nat*(7) *power2-eq-square power-mult power-mult-distrib*)
then have *3*: $real\text{-}of\text{-}rat\ 2 \hat{}\ (d * h * (d * h - 1)\ div\ 2) *$
$real\text{-}of\text{-}int\ ((det\ (mat\text{-}of\text{-}rows\ (d * h)\ ?cs\text{-}basis))^2) *$
$real\ (d * h) \hat{}\ (d * h)$
$< ((M \hat{}\ (h - 1)) \hat{}\ (2 * (d * h)))$
unfolding *drw*
by (*smt* (*verit*, *del-insts*) *Groups.mult-ac*(2) *Groups.mult-ac*(3) *Num.of-nat-simps*(5)

more-arith-simps(*11*) *of-int-of-nat-eq of-nat-numeral of-nat-power-less-of-nat-cancel-iff of-rat-of-int-eq power-mult semiring-1-class.of-nat-power*)

interpret *lll*: *LLL-with-assms*
d ∗ *h d* ∗ *h ?cs-basis 2*
apply *unfold-locales*
subgoal by *auto*
subgoal
unfolding *cof-vec-space.lin-indpt-list-def*
using *li1 li2 li3* **by** *auto*
using *dim-form-basis* **by** *auto*

interpret *lll*: *coppersmith-assms*
d ∗ *h d* ∗ *h ?cs-basis*
2 x0 M^(*h* − *1*) *X p* ^ (*h* − *1*)
apply *unfold-locales*
subgoal using *M-gt* **by** *auto*
subgoal using *X-gt-zero* **by** *auto*
subgoal using *dh-gt-0* **by** *auto*
subgoal using *pmod* **by** *auto*
subgoal using *x0-le* **by** *auto*
subgoal
by (*metis d-gt bot-nat-0.extremum-strict h-gt local.dim-form-basis mult-eq-0-iff*)
subgoal using *1* **by** *auto*
using *2* **by** *auto*

have ∗: *lll.short-vector* = *short-vector 2 ?cs-basis*
using *lll.short-vector-impl* **by** *presburger*

from *lll.root-poly-short-vector*
show *?thesis*
using *lll.bnd-raw-imp-short-vec-bound 3 lll.square-Gramian-determinant-eq-det-square*
unfolding *f-is coppersmith-def first-vector-coppersmith-def Let-def* ∗
using *h-def* **by** *fastforce*
qed

theorem *coppersmith-finds-small-roots-pretty*:
fixes *p f*:: *int poly*
fixes *M X*:: *nat*
fixes *x0*:: *int*
fixes *eps*:: *real*
defines *d* ≡ *degree p*
defines *f* ≡ *coppersmith p M X eps*
assumes *monic-poly*: *monic p*
assumes *d* > *1* **and** *M* > *0* **and** *X* > *0*
assumes *zero-mod-M*: *poly p x0 mod M* = *0*
assumes *X-lt*: *X* < *root-bound M d eps*
assumes *x0-le*: *abs x0* ≤ *X*

assumes *eps-le*: $eps \leq 0.18 * (d-1)/d$
assumes *eps-lt*: $eps < 1 \ / \ d$
assumes *eps-gt0*: $eps > 0$
shows *poly f x0 = 0*
using *assms coppersmith-finds-small-roots*
by *presburger*

end

7 Examples of Coppersmith's Method

In this file, we provide some examples of Coppersmith's method, with correctness proofs (when applicable).

theory *Coppersmith-Examples*

imports *Coppersmith*
Towards-Coppersmith
begin

7.1 Example of lightweight method

Following Example 19.1.6 in Galbraith. This example produces [:444, 1, -20, - 2:], which corresponds to $-2x^3 - 20x^2 + x + 444$, which has 4 as a root. Computing $4^3 + 10 * 4^2 + 5000 * 4 - 222$ produces 20002, which is 0 mod 10001. Note that here, we cannot use our top-level correctness result for the lightweight method to prove correctness. This is because the conditions of this top-level result are not satisfied; however, the method succeeds despite not fulfilling the conditions of this result, because the LLL algorithm can be better than the bound in the result. See Example 19.1.6 in "Mathematics of Public Key Cryptography" by Galbraith for more discussion of this.

value *towards-coppersmith* [:-222, *5000*, *10*, *1*:] *10001 10*

7.2 Examples of Coppersmith's method

Following Exercise 19.1.12 in Galbraith.
This next example produces 15955575444164700778296 - 86948462676890416832x + 50262448764961319x^2+ 334700479564525x^3 - 611446097378x^4 - 577363178x^5 + 1008850x^6 + 8592x^7, which has 267 as a root, which is a root of the original polynomial mod 2^9.

value *coppersmith* [:*227*, *46976195*, *2^25−2883584*, *1*:] ((*2^20 + 7*)∗(*2^21 + 17*)) (*2^9*) *0.089*

We now prove that this example satisfies the conditions of our top-level correctness theorem for Coppersmith's method.

lemma *coppersmith-finds-small-roots-example1*:

```
  fixes p f:: int poly
  fixes M X:: nat
  fixes x0:: int
  fixes k:: nat
  defines p ≡ [:227, 46976195, 2^25−2883584, 1:]
  defines d ≡ degree p
  defines M ≡ ((2^20 + 7)*(2^21 + 17))
  defines X ≡ 2^9
  defines f ≡ coppersmith p M X 0.089
  assumes x0-le: |x0| ≤ X
  assumes zero-mod-M: poly p x0 mod M = 0
  shows poly f x0 = 0
proof −
  have d-eq-3: d = 3
    unfolding d-def p-def by auto
  have 1 / (3::real) − 89 / 10 ^ 3 = 733/3000
    by auto
  have all-prop: ⋀ A B :: nat. real A < B ⟹ real A powr n < B powr n if n >
0 for n
    by (simp add: powr-less-mono2 that)
  have real ((2 ^ 20*2 ^ 21)) < real ((2 ^ 20 + 7) * (2 ^ 21 + 17))
    by simp
  then have lt:  real ((2 ^ 20*2 ^ 21)) powr (733/3000) < real ((2 ^ 20 + 7) *
(2 ^ 21 + 17)) powr (733/3000)
    using all-prop[of (733/3000) (2 ^ 20*2 ^ 21) (2 ^ 20 + 7) * (2 ^ 21 + 17)]
    by argo
  have ⋀ b:: nat. (real 2^b) powr c = 2 powr (b*c) for c::real
    by (smt (verit) Num.of-nat-simps(4) Suc-1 of-nat-1 plus-1-eq-Suc powr-powr
powr-realpow)
  then have (2::real) powr (41 * (733 / 3000)) = (2 ^ 41) powr (733 / 3000)
     by (metis numeral-powr-numeral-real powr-powr)
 then have lt1: (((2::nat) ^ 20*2 ^ 21)) powr (733/3000) = 2 powr (41*733/3000)
    by simp
  have 41*733/3000 > real 10
    by simp
  then have 2 powr 10 < real 2 powr (41*733/3000)
    by (metis less-num-simps(2) numeral-code(1) numeral-less-iff of-nat-numeral
powr-less-mono)
  then have 2^10 < real ((2 ^ 20 + 7) * (2 ^ 21 + 17)) powr (1 / (3::real) −
89 / 10 ^ 3)
    using lt lt1 by auto
  then have root-bound: real X < root-bound M (degree p) 0.089
    unfolding M-def X-def root-bound-def using d-eq-3 unfolding d-def
    by simp
  show ?thesis
 using coppersmith-finds-small-roots-pretty[OF - - - - zero-mod-M root-bound x0-le]
  unfolding p-def M-def f-def X-def by auto
qed
```

In this next example, we are trying to find small roots less than 3 of

$x^3+1000x^2+25+1$ mod 4059. Running "coppersmith" on this example yields [:41858, - 28457, 6100, 376, 150, - 31, - 91, - 28, - 17:], which corresponds to $-17x^8 - 28x^7 - 91x^6 - 31x^5 + 150x^4 + 376x^3 + 6100x^2 - 28457x + 41858$, which has 2 as a root. Plugging in $2^3 + 1000 * 2^2 + 25 * 2 + 1$ indeed yields 4059, which is 0 mod 4059.

value *form-basis-coppersmith* [:*1*, *25*, *1000*, *1*:] *4059 3* (*calculate-h-coppersmith* [:*1*, *25*, *1000*, *1*:] *0.10*)
value *reduce-basis 2* (*form-basis-coppersmith* [:*1*, *25*, *1000*, *1*:] *4059 3* (*calculate-h-coppersmith* [:*1*, *25*, *1000*, *1*:] *0.10*))
value *coppersmith* [:*1*, *25*, *1000*, *1*:] *4059 3 0.10*

We now prove that this example satisifes the conditions of our top-level correctness theorem for Coppersmith's method.

lemma *coppersmith-finds-small-roots-example2*:
fixes *p f*:: *int poly*
fixes *M X*:: *nat*
fixes *x0*:: *int*
fixes *k*:: *nat*
defines *p* ≡ [:*1*, *25*, *1000*, *1*:]
defines *d* ≡ *degree p*
defines *M* ≡ *4059*
defines *X* ≡ *3*
defines *f* ≡ *coppersmith p M X 0.10*
assumes *x0-le*: |*x0*| ≤ *X*
assumes *zero-mod-M*: *poly p x0 mod M* = *0*
shows *poly f x0* = *0*
proof −
have *d-eq-3*: *d* = *3*
unfolding *d-def p-def*
by *simp*
have ⋀ *a b*:: *nat*. (*real 4059 powr c*) ^ *b* = *4059 powr* (*b*∗*c*) **for** *c*::*real*
using *Suc-1 plus-1-eq-Suc powr-powr powr-realpow*
by (*smt* (*verit*) *Groups.mult-ac*(*2*) *Num.of-nat-simps*(*2*) *numeral-Bit0 numeral-Bit1 numeral-One of-nat-numeral powr-gt-zero*)
then have *simplify-power*:(*real 4059 powr* (*7*/*30*))^*30* = *4059 powr 7*
by *auto*
have *gt*: (*4059*::*real*)/(*6*^*4*) > *3*
by *auto*
have (*6*::*real*)^*28* = *6*^(*4*∗*7*)
by *auto*
then have *h1*: (*6*::*real*)^*28* = (*6*^*4*)^*7*
by (*metis power-mult*)
have *all-div*: ⋀*a b c*::*real*. *b* > *0* ⟹ *a* < *c*/*b* ⟹ *a*∗*b* < *c*
by (*simp add*: *pos-less-divide-eq*)
have *h2*: *real 6*^*30* = *6*^*28*∗*6*^*2*
by *simp*
have ((*4059*)^*7*::*real*)/((*6*^*4*)^*7*) = (*4059*/*6*^*4*)^*7*
by (*metis power-divide*)

```
  then have ((4059)^7::real)/((6^4)^7) > 3^7
    using gt by simp
  then have real 6^2 < (4059)^7/((6^4)^7)
    by simp
  then have real 6^2*((6^4)^7) < (4059)^7
    using all-div[of (6^4)^7 6^2 (4059)^7]
    by simp
  then have real 6^30 < 4059 ^ 7
    using h1 h2 by simp
  then have real 6^30 < (real 4059 powr (7/30))^30
    using simplify-power by fastforce
  then have real 6 < real 4059 powr (7/30)
     by (smt (verit) of-nat-0-less-iff powr-gt-zero powr-less-mono2 powr-realpow
semiring-norm(118))
  then have root-bound: real X < root-bound M (degree p) 0.10
    unfolding M-def X-def root-bound-def using d-eq-3 unfolding d-def
    by auto
  show ?thesis
 using coppersmith-finds-small-roots-pretty[OF - - - - zero-mod-M root-bound x0-le]
  unfolding p-def M-def f-def X-def by auto
qed


end
```

References

[1] J. Alperin-Sheriff and C. Peikert. Lattices in cryptography: Lecture 4, Coppersmith, cryptanalysis. Georgia Tech lecture notes, Fall 2023. Online lecture notes available at https://web.eecs.umich.edu/~cpeikert/lic13/lec04.pdf.

[2] R. Bottesch, J. Divasón, M. W. Haslbeck, S. J. C. Joosten, R. Thiemann, and A. Yamada. A verified LLL algorithm. *Archive of Formal Proofs*, February 2018. https://isa-afp.org/entries/LLL_Basis_Reduction.html, Formal proof development.

[3] S. D. Galbraith. *Mathematics of Public Key Cryptography*. Cambridge University Press, 2012.

[4] W. Trappe and L. C. Washington. *Introduction to Cryptography with Coding Theory (2nd Edition)*. Prentice-Hall, Inc., USA, 2005.