# The Hidden Number Problem

Sage Binder, Eric Ren, and Katherine Kosaian

June 26, 2025

**Abstract**

In this entry, we formalize the Hidden Number Problem (HNP), originally introduced by Boneh and Venkatesan in 1996 [2]. Intuitively, the HNP involves demonstrating the existence of an algorithm (the "adversary") which can compute (with high probability) a hidden number $\alpha$ given access to a bit-leaking oracle. Originally developed to establish the security of Diffie–Hellman key exchange, the HNP has since been used not only for protocol security but also in cryptographic attacks, including notable ones on DSA and ECDSA.

Additionally, the HNP makes use of an instance of Babai's nearest plane algorithm [1], which solves the approximate closest vector problem. Thus, building on the LLL algorithm [5] (which has already been formalized [4, 3, 6]), we formalize Babai's algorithm, which itself is of independent interest. Our formalizations of Babai's algorithm and the HNP adversary are executable, setting up potential future work, e.g. in developing formally verified instances of cryptographic attacks.

Note, our formalization of Babai's algorithm here is an updated version of a previous AFP entry of ours. The updates include a tighter error bound, which is required for our HNP proof.

## Contents

**theory** *Babai-Algorithm-Updated*
**imports**
*LLL-Basis-Reduction.LLL-Impl*
*HOL.Archimedean-Field*
*HOL−Analysis.Inner-Product*

**begin**

**fun** *calculate-c*:: *rat vec* ⇒ *rat vec list* ⇒ *nat* => *int* **where**
*calculate-c s L1 n* = *round* ((*s* ∙ (*L1*!((*dim-vec s*) − *n*))) / (*sq-norm-vec* (*L1*!((*dim-vec s*) − *n*))))

**fun** *update-s*:: *rat vec* ⇒ *rat vec list* ⇒ *rat vec list* ⇒ *nat* ⇒ *rat vec* **where**
*update-s sn M Mt n* = ((*rat-of-int* (*calculate-c sn Mt n*)) ·$_v$ *M*!((*dim-vec sn*) − *n*))

**fun** *babai-help*:: *rat vec* ⇒ *rat vec list* ⇒ *rat vec list* ⇒ *nat* ⇒ *rat vec* **where**
*babai-help s M Mt 0* = *s* |
*babai-help s M Mt* (*Suc n*) = (*let B* = (*babai-help s M Mt n*) *in B* − (*update-s B M Mt* (*Suc n*)))

This assumes an LLL-reduced input and outputs a short vector of the form $v + t$, with $v \in L$

**fun** *babai-of-LLL* :: *rat vec* ⇒ *rat vec list* ⇒ *rat vec* **where**
*babai-of-LLL s M* = *babai-help s M* (*gram-schmidt* (*dim-vec s*) *M*) (*dim-vec s*)

This begins with a non-reduced basis and outputs a vector $v$ in $L$ which is close to $t$

**fun** *full-babai* :: *int vec list* ⇒ *rat vec* ⇒ *rat* ⇒ *int vec*
**where** *full-babai fs target* α =
*map-vec int-of-rat*
(*babai-of-LLL*
(*uminus target*)
(*LLL.RAT* (*LLL-Impl.reduce-basis* α *fs*))
+ *target*)

**end**
**theory** *Babai-Correctness-Updated*
**imports** *Babai-Algorithm-Updated*
*LLL-Basis-Reduction.LLL-Impl*
*Jordan-Normal-Form.DL-Rank*
*BenOr-Kozen-Reif.More-Matrix*
**begin**

This theory contains the proof of correctness of the algorithm. The main theorem is "theorem babai-correct", under the locale "babai-with-assms". To use the theorem, one needs to show that lattice, the vectors in the lattice basis, and the target vector all have the same dimension, that the lattice basis vectors are linearly independent, and that the lattice basis is LLL-weakly-reduced for some parameter $\alpha \geq 4/3$.

# 1 Locale setup for Babai

**locale** *babai* =
**fixes** *M* :: *int vec list*
**fixes** *t* :: *rat vec*
**fixes** α :: *rat*
**assumes** *length-M*: *length M* = *dim-vec t*
**begin**

**abbreviation** *n* **where** *n* ≡ *length M*
**sublocale** *LLL n n M* α **.**

**abbreviation** *coset*::*rat vec set* **where** *coset*≡{(*map-vec rat-of-int x*)−*t*|*x*. *x*∈*L*}
**abbreviation** *Mt* **where** *Mt* ≡ *gram-schmidt n* (*RAT M*)

**definition** *s* :: *nat* ⇒ *rat vec* **where**
*s i* = *babai-help* (*uminus t*) (*RAT M*) *Mt i*

**definition** *closest-distance-sq*:: *real* **where**
*closest-distance-sq* = *Inf* {*real-of-rat* (*sq-norm x*::*rat*) |*x*. *x* ∈ *coset*}
**end**

Locale setup with additional assumptions required for main theorem. Contains an arbitrary extra constant epsilon which appears in the final bound in this locale, giving a theorem quantified over all epsilon > 1. The next locale, "babai-with-assms," uses this theorem to prove a theorem without epsilon.

**locale** *babai-with-assms-epsilon* = *babai* +
**fixes** *mat-M mat-M-inv*:: *rat mat*
**fixes** *epsilon*::*real*
**assumes** *basis*: *lin-indep M*
**defines** *mat-M* ≡ *mat-of-cols n* (*RAT M*)
**defines** *mat-M-inv* ≡
(*if* (*invertible-mat mat-M*) *then SOME B*. (*inverts-mat B mat-M*) ∧ (*inverts-mat mat-M B*) *else* (0$_m$ *n n*))
**assumes** *inv*:*invertible-mat mat-M*
**assumes** *reduced*:*weakly-reduced M n*
**assumes** *non-trivial*:*0*<*n*
**assumes** *alpha*:*α* ≥ *4*/*3*
**assumes** *epsilon*:*epsilon* > *1*
**begin**

**lemma** *dim-vecs-in-M*:
**shows** ∀ *v* ∈ *set M*. *dim-vec v* = *length M*
**using** *basis* **unfolding** *gs.lin-indpt-list-def* **by** *force*

**lemma** *inv1*:*mat-M* ∗ *mat-M-inv* = 1$_m$ *n*
**proof**−
**have** *dim-m*:*dim-row mat-M* = *n* **using** *dim-vecs-in-M* **unfolding** *mat-M-def* **by** *fastforce*
**then have** *inverts-mat mat-M mat-M-inv* **using** *inv*
**unfolding** *mat-M-inv-def*
**by** (*smt* (*verit*, *ccfv-SIG*) *invertible-mat-def some-eq-imp*)
**then show** *?thesis* **using** *dim-m* **unfolding** *inverts-mat-def* **by** *argo*
**qed**

**lemma** *inv2*:*mat-M-inv* ∗ *mat-M* = 1$_m$ *n*
**proof**−
**have** *dim-m*:*dim-col mat-M* = *n* **unfolding** *mat-M-def* **by** *fastforce*
**have** *inverts-mat mat-M-inv mat-M* **using** *inv*
**unfolding** *mat-M-inv-def*
**by** (*smt* (*verit*, *ccfv-SIG*) *invertible-mat-def some-eq-imp*)
**then have** *inv*:*mat-M-inv* ∗ *mat-M* = 1$_m$ (*dim-row mat-M-inv*)
**unfolding** *inverts-mat-def* **by** *blast*
**then have** *dim-n*:*dim-col* (1$_m$ (*dim-row mat-M-inv*)) = *n*
**using** *dim-m index-mult-mat*(*3*)[*of mat-M-inv mat-M*] **by** *fastforce*

**have** (*dim-row mat-M-inv*)= *n*
**proof**(*rule ccontr*)
**assume** (*dim-row mat-M-inv*)≠ *n*
**then have** *dim-col* (*1*$_m$ (*dim-row mat-M-inv*)) ≠ *n*
**by** *auto*
**then show** *False* **using** *dim-n* **by** *blast*
**qed**
**then show** *?thesis* **using** *inv* **by** *argo*
**qed**

**sublocale** *rats*: *vec-module TYPE*(*rat*) *n***.**

**lemma** *M-dim*: *dim-row mat-M* = *n dim-col mat-M* = *n*
**apply** (*metis index-mult-mat*(*2*) *index-one-mat*(*2*) *inv1*)
**by** (*metis index-mult-mat*(*3*) *index-one-mat*(*3*) *inv2*)

**lemma** *M-inv-dim*: *dim-row mat-M-inv* = *n dim-col mat-M-inv* = *n*
**apply** (*metis M-dim*(*1*) *index-mult-mat*(*2*) *inv1 inv2*)
**by** (*metis index-mult-mat*(*3*) *index-one-mat*(*3*) *inv1*)

**lemma** *babai-to-help*:
**shows** *s n* = *babai-of-LLL* (*uminus t*) (*RAT M*)
**using** *babai-def babai.s-def babai-of-LLL.simps babai-axioms* **by** *force*

## 2 Coordinates

This section sets up the use of the lattice basis and its GS orthogonalization as coordinate systems and some properties of that coordinate system. The important lemma here is coord-invariance, which shows that after step i of the algorithm, all coordinates (in both systems) after n-i are invariant.

**definition** *lattice-coord* :: *rat vec* ⇒ *rat vec*
**where** *lattice-coord a* = *mat-M-inv* ∗$_v$ *a*

**lemma** *dim-preserve-lattice-coord*:
**fixes** *v*::*rat vec*
**assumes** *dim-vec v*=*n*
**shows** *dim-vec* (*lattice-coord v*) = *n* **unfolding** *lattice-coord-def mat-M-inv-def*
**using** *M-inv-dim*
**by** (*simp add*: *mat-M-inv-def*)
**lemma** *vec-to-col*:
**assumes** *i* < *n*
**shows** (*RAT M*)!*i* = *col mat-M i*
**unfolding** *mat-M-def*
**by** (*metis babai-with-assms-epsilon-axioms babai-with-assms-epsilon-axioms-def babai-with-assms-epsilon-def M-dim*(*2*)
*assms cols-mat-of-cols cols-nth gs.lin-indpt-list-def mat-M-def*)

**lemma** *unit*:
**assumes** $i < n$
**shows** *lattice-coord* $((RAT\ M)!i) =$ *unit-vec* $n\ i$
**using** *assms inv2* **unfolding** *lattice-coord-def*
**by** (*metis M-dim*(*1*) *M-dim*(*2*) *M-inv-dim*(*2*) *carrier-matI col-mult2 col-one vec-to-col*)

**lemma** *linear*:
**fixes** *i*::*nat*
**fixes** *v1*::*rat vec*
**and** *v2*:: *rat vec*
**and** *q*:: *rat*
**assumes** *dim-vec v1* $= n$
**assumes** *dim-2*:*dim-vec v2* $= n$
**assumes** $0 \leq i$
**assumes** *dim-i*:$i<n$
**shows** (*lattice-coord* (*v1*+($q \cdot_v$*v2*)))$$i$ = (*lattice-coord v1*)$$i$ + $q*$((*lattice-coord v2*)$$i$)
**using** *assms*
**proof**(−)
**have** *linear-vec*:(*lattice-coord* (*v1*+($q \cdot_v$*v2*))) = (*lattice-coord v1*) + $q \cdot_v$((*lattice-coord v2*))
**unfolding** *lattice-coord-def*
**by** (*metis* (*mono-tags*, *opaque-lifting*) *M-inv-dim*(*2*) *assms*(*1*) *assms*(*2*) *carrier-mat-triv*
*carrier-vec-dim-vec mult-add-distrib-mat-vec mult-mat-vec smult-carrier-vec*)
**then have** *2*: (*lattice-coord* (*v1*+($q \cdot_v$*v2*)))$$i$= ((*lattice-coord v1*) + $q \cdot_v$((*lattice-coord v2*)))$$i$ **by** *auto*
**also have** *dim-v2*: *dim-vec* (*lattice-coord v2*) $= n$ **using** *dim-preserve-lattice-coord dim-2* **by** *blast*
**then have** *i-in-range*: $i<$*dim-vec* ($q \cdot_v$(*lattice-coord v2*)) **using** *dim-v2 dim-i* **by** *simp*
**also have** *3*:((*lattice-coord v1*) + $q \cdot_v$((*lattice-coord v2*)))$$i$=(*lattice-coord v1*)$$i$+
($q \cdot_v$(*lattice-coord v2*))$$i$ **using** *i-in-range* **by** *simp*
**also have** *4*: ($q \cdot_v$(*lattice-coord v2*))$$i$=$q*$(*lattice-coord v2*)$$i$ **using** *i-in-range* **by** *simp*
**thus** *?thesis* **unfolding** *vec-def* **using** *linear-vec 2 3 4* **by** *simp*
**qed**

**lemma** *sub-s*:
**fixes** *i*::*nat*
**assumes** $0 \leq i$
**assumes** $i<n$
**shows** *s* (*Suc i*) = (*s i*) −
( (*rat-of-int* (*calculate-c* (*s i*) *Mt* (*Suc i*) ) ) $\cdot_v$ (*RAT M*)!( (*dim-vec* (*s i*)) −(*Suc i*)))
**using** *assms babai-help.simps*[*of* −*t RAT M Mt*] **unfolding** *s-def*
**by** (*metis update-s.simps*)

**lemma** *M-locale-1*:
**shows** *gram-schmidt-fs-Rn n* (*RAT M*)
**by** (*smt* (*verit*) *M-dim*(*1*) *M-dim*(*2*) *carrier-dim-vec dim-col gram-schmidt-fs-Rn.intro in-set-conv-nth*
*mat-M-def mat-of-cols-carrier*(*3*) *subset-code*(*1*) *vec-to-col*)

**lemma** *M-locale-2*:
**shows** *gram-schmidt-fs-lin-indpt n* (*RAT M*)
**using** *basis M-locale-1 gram-schmidt-fs-lin-indpt.intro*[*of n* (*RAT M*)] **unfolding** *gs.lin-indpt-list-def*
**using** *gram-schmidt-fs-lin-indpt-axioms.intro* **by** *blast*

**lemma** *more-dim*: *length* (*RAT M*) = *n*
**by** *simp*

**lemma** *Mt-gso-connect*:
**fixes** *j*::*nat*
**assumes** *j*<*n*
**shows** *Mt*!*j* = *gs.gso j*
**proof**(−)
**have** *Mt* = *map gs.gso*[*0*..<*n*]
**using** *M-locale-1 gram-schmidt-fs-Rn.main-connect*[*of n* (*RAT M*)]
**by** *fastforce*
**then show** *?thesis*
**using** *assms*
**by** *simp*
**qed**

**lemma** *access-index-M-dim*:
**assumes** *0* ≤ *i*
**assumes** *i* < *n*
**shows** *dim-vec* (*map of-int-hom.vec-hom M* ! *i*) = *n*
**using** *assms dim-vecs-in-M*
**by** *auto*

**lemma** *s-dim*:
**fixes** *i*::*nat*
**assumes** *i*≤ *n*
**shows** *dim-vec* (*s i*) = *n* ∧ (*s i*) ∈*carrier-vec n*
**using** *assms*
**proof**(*induct i*)
**case** *0*
**have** *unfold1*:*s 0* = *babai-help* (*uminus t*) (*RAT M*) *Mt 0* **unfolding** *s-def* **by** *simp*
**also have** *unfold2*:*babai-help* (*uminus t*) (*RAT M*) *Mt 0* = *uminus t* **unfolding** *babai-help.simps* **by** *simp*
**also have** *unfold3*:*s 0* = *uminus t* **using** *unfold1 unfold2* **by** *simp*

**also have** *dim-eq*:*dim-vec* (*s 0*) = *dim-vec* (*uminus t*) **using** *unfold3* **by** *simp*
**moreover have** *dim-minus*:*dim-vec* (*uminus t*) = *n* **by** (*metis index-uminus-vec*(*2*) *length-M*)
**then have** *dim-vec* (*s 0*) = *n*
**using** *dim-eq dim-minus*
**by** *simp*
**then have** (*s 0*) ∈ *carrier-vec n*
**using** *carrier-vecI*[*of* (*s 0*) *n*]
**by** *simp*
**then show** *?case*
**by** *simp*
**next**
**case** (*Suc i*)
**then have** *leq*: *i*≤*n* **by** *linarith*
**have** *sub*:*s* (*Suc i*) = (*s i*) − ( (*rat-of-int* (*calculate-c* (*s i*) *Mt* (*Suc i*) ) ) ·$_v$ (*RAT M*)!( (*dim-vec* (*s i*)) −(*Suc i*)))
**using** *sub-s Suc*
**by** *auto*
**moreover have** *prev-s-dim*:(*s i*)∈*carrier-vec n*
**using** *Suc*
**by** *simp*
**moreover have** *dim-vec* (*s i*)=*n*
**using** *Suc*
**by** *simp*
**then have** *0*≤(*dim-vec* (*s i*)) −(*Suc i*)∧ (*dim-vec* (*s i*)) −(*Suc i*)<*n*
**using** *Suc*
**by** *linarith*
**then have** *dim-m*:(*dim-vec* ((*RAT M*)!( (*dim-vec* (*s i*)) −(*Suc i*)))) = *n*
**using** *access-index-M-dim*[*of* (*dim-vec* (*s i*)) −(*Suc i*)]
**by** *simp*
**then have** *dim-qm*:*dim-vec* ( (*rat-of-int* (*calculate-c* (*s i*) *Mt* (*Suc i*) ) ) ·$_v$ (*RAT M*)!((*dim-vec* (*s i*)) −(*Suc i*))) = *n*
**by** *simp*
**then have** *final-dim*:*dim-vec* ((*s i*) − ((*rat-of-int* (*calculate-c* (*s i*) *Mt* (*Suc i*) ) ) ·$_v$ (*RAT M*)!( (*dim-vec* (*s i*)) −(*Suc i*)))) = *n*
**using** *index-minus-vec*(*2*) *prev-s-dim dim-qm*
**by** *metis*
**show** *?case*
**using** *final-dim sub carrier-vecI*[*of s i n*]
**by** (*metis carrier-vec-dim-vec*)
**qed**

**lemma** *dim-vecs-in-Mt*:
**fixes** *i*::*nat*
**assumes** *i*<*n*
**shows** *dim-vec* (*Mt*!*i*) = *n*
**using** *Mt-gso-connect*[*of i*] *M-locale-1 assms gram-schmidt-fs-Rn.gso-dim*
**by** *fastforce*

**lemma** *upper-tri*:
**fixes** *i*::*nat*
**and** *j*::*nat*
**assumes** *j*>*i*
**assumes** *j*<*n*
**shows** ((*RAT M*)!*i*)· (*Mt*!*j*) =*0*
**proof**(−)
**have** (*gs.gso j*)· (*RAT M*)!*i*= *0*
**using** *gram-schmidt-fs-lin-indpt.gso-scalar-zero*[*of n* (*RAT M*) *j i*]
*Mt-gso-connect*[*of j*]
*assms*
*M-locale-2*
*more-dim*
**by** *presburger*
**then have** (*Mt*!*j*)· ((*RAT M*)!*i*)= *0*
**using** *Mt-gso-connect*[*of j*] *assms*
**by** *simp*
**then show** *?thesis*
**using** *comm-scalar-prod*[*of* (*Mt*!*j*) *n* ((*RAT M*)!*i*)]
*carrier-vecI*[*of* (*Mt*!*j*) *n*]
*carrier-vecI*[*of* ((*RAT M*)!*i*) *n*]
*access-index-M-dim*[*of i*]
*dim-vecs-in-Mt*[*of j*]
*assms*
**by** *auto*
**qed**
**lemma** *one-diag*:
**fixes** *i*::*nat*
**assumes** *0*≤*i*
**assumes** *i*<*n*
**shows** ((*RAT M*)!*i*)· (*Mt*!*i*)=*sq-norm* (*Mt*!*i*)
**proof**(−)
**have** *mu*:((*RAT M*)!*i*)·(*Mt*!*i*) = (*gs.μ i i*)∗*sq-norm* (*Mt*!*i*)
**using** *gram-schmidt-fs-lin-indpt.fi-scalar-prod-gso*[*of n* (*RAT M*) *i i*]
*M-locale-2*
*assms*
*more-dim*
*Mt-gso-connect*
**by** *presburger*
**moreover have** *gs.μ i i*=*1*
**by** (*meson gs.μ.elims order-less-imp-not-eq2*)
**then show** *?thesis*
**using** *mu*
**by** *fastforce*
**qed**

**lemma** *coord-invariance*:
**fixes** *j*::*nat*

```
  fixes k::nat
  fixes i::nat
  assumes k≤j
  assumes j+i≤n
  assumes k>0
  shows (lattice-coord (s (j+i)))$(n−k) = (lattice-coord (s j))$(n−k)
    ∧ (s (j+i)) · Mt!(n−k)=(s j) · Mt!(n−k)
  using assms
proof(induct i)
  case 0
  show ?case by simp
next
  case (Suc i)
  have j+ (Suc i) = Suc (j+i) by simp
  then have 1:s (Suc (j+i)) =s (j + (Suc i)) by simp
  then have sub:s (Suc (j+i) ) =
    (s (j+i)) −( (rat-of-int (calculate-c (s (j+i)) Mt (Suc (j+i)) ) )
       ·v (RAT M)!( (dim-vec (s (j+i))) −(Suc (j+i)) ) )
    using sub-s[of j+i ] Suc(3) by linarith
  then have dim1: dim-vec (s (j + i)) = n
    using s-dim[of j+i] using Suc(3) by auto
  then have dim2: dim-vec
     (map of-int-hom.vec-hom M !
      (dim-vec (s (j + i)) − Suc (j + i))) = n
    using access-index-M-dim[of n − Suc (j + i)] Suc(3)
    by auto
  have k-in-range:0≤(n−k) ∧(n−k)<n using Suc(2) Suc(3) Suc(4)
    by simp
  have index-in-range:0≤(dim-vec (s (j+i))) −(Suc (j+i))∧(dim-vec (s (j+i)))
−(Suc (j+i))<n
    using Suc(3) s-dim[of j+i]
    by simp
  moreover have carriers: s (j+i) ∈ carrier-vec n∧
                    map of-int-hom.vec-hom M ! (dim-vec (s (j + i)) − Suc (j +
i))∈carrier-vec n
    using dim1 dim2
          carrier-vecI[of s (j + i) n]
          carrier-vecI[of map of-int-hom.vec-hom M ! (dim-vec (s (j + i)) − Suc (j
+ i)) n]
    by fast

  let ?sSuc = (s (Suc (j+i)))
  let ?si = (s (j+i))
  let ?c = (rat-of-int (calculate-c (s (j+i)) Mt (Suc (j+i)) ) )
  let ?ind = (dim-vec (s (j+i))) −(Suc (j+i))

  have ?si − ?c·v (RAT M)!?ind = ?si + (−?c)·v (RAT M)!?ind
    using minus-add-uminus-vec[of ?si n ?c·v (RAT M)!?ind]
          carriers
```

**by** *fastforce*
**then have** (*lattice-coord* (*?si* − *?c*·$_v$ (*RAT M*)!*?ind*))$(*n*−*k*) =
(*lattice-coord*(*?si*))$(*n*−*k*) + (−*?c*)∗ (*lattice-coord*((*RAT M*)!*?ind*))$(*n*−*k*)
**using** *linear*[*of ?si* (*RAT M*)!*?ind n*−*k* −*?c*] *dim1 dim2 k-in-range*
**by** *metis*
**then have** *lin-lattice-coord*:(*lattice-coord* (*?sSuc*))$(*n*−*k*) =
(*lattice-coord*(*?si*))$(*n*−*k*) − *?c*∗ (*lattice-coord*((*RAT M*)!*?ind*))$(*n*−*k*)
**using** *sub*
**by** *algebra*
**have** *neq*:*Suc* (*j*+*i*)≠*k* **using** *Suc*(*3*) *Suc*(*2*) **by** *auto*
**moreover have** ((*dim-vec* (*s* (*j*+*i*))) −(*Suc* (*j*+*i*)))≠ (*n*−*k*)
**using** *s-dim*[*of j*+*i*] *neq Suc*(*3*)
**by** (*metis Suc*(*2*) ‹*j* + *Suc i* = *Suc* (*j* + *i*)› *diff-0-eq-0 diff-cancel2*
*diff-commute diff-diff-cancel diff-diff-eq diff-is-0-eq dim1*)
**moreover have** (*lattice-coord* ((*RAT M*)!( (*dim-vec* (*s* (*j*+*i*))) −(*Suc* (*j*+*i*))) )
)$(*n*−*k*)=
(*unit-vec n* ( (*dim-vec* (*s* (*j*+*i*))) −(*Suc* (*j*+*i*))))$(*n*−*k*)
**using** *unit*[*of dim-vec* (*s* (*j*+*i*)) −(*Suc* (*j*+*i*))] *index-in-range* **by** *presburger*
**then have** *zero*:(*lattice-coord* ((*RAT M*)!( (*dim-vec* (*s* (*j*+*i*))) −(*Suc* (*j*+*i*))) )
)$(*n*−*k*) = *0*
**unfolding** *unit-vec-def*
**using** *neq calculation*(*3*) *k-in-range* **by** *fastforce*
**then have** (*lattice-coord* (*s* (*Suc* (*j*+*i*))) )$(*n*−*k*) = ( (*lattice-coord* (*s* (*j*+*i*)))$(*n*−*k*))
−
(*rat-of-int* (*calculate-c* (*s* (*j*+*i*)) *Mt* (*Suc* (*j*+*i*)) ) )
∗ *0*
**using** *zero lin-lattice-coord* **by** *presburger*
**then have** *conclusion1*:(*lattice-coord* (*s* (*Suc* (*j*+*i*))) )$(*n*−*k*) = ( (*lattice-coord*
(*s* (*j*+*i*)))$(*n*−*k*))
**by** *simp*
**have** *init-sub*:(*s* (*Suc* (*j*+*i*)))· *Mt*!(*n*−*k*) = ((*s* (*j*+*i*)) −
((*rat-of-int* (*calculate-c* (*s* (*j*+*i*)) *Mt* (*Suc* (*j*+*i*)) ) ) ·$_v$ (*RAT M*)!( (*dim-vec* (*s*
(*j*+*i*))) −(*Suc* (*j*+*i*)) ) ))
· (*Mt*!(*n*−*k*))
**using** *sub*
**by** *simp*
**moreover have** *carrier-prod*:( (*rat-of-int* (*calculate-c* (*s* (*j*+*i*)) *Mt* (*Suc* (*j*+*i*))
) )
·$_v$ (*RAT M*)!( (*dim-vec* (*s* (*j*+*i*))) −(*Suc* (*j*+*i*)) ) )∈*carrier-vec n*
**using** *smult-carrier-vec*[*of* (*rat-of-int* (*calculate-c* (*s* (*j*+*i*)) *Mt* (*Suc* (*j*+*i*)) ) )
(*RAT M*)!( (*dim-vec* (*s* (*j*+*i*))) −(*Suc* (*j*+*i*)) ) *n*] *carrier-vecI dim2* **by**
*blast*
**moreover have** *l*:((*s* (*j*+*i*)) −
( (*rat-of-int* (*calculate-c* (*s* (*j*+*i*)) *Mt* (*Suc* (*j*+*i*)) ) ) ·$_v$ (*RAT M*)!( (*dim-vec* (*s*
(*j*+*i*))) −(*Suc* (*j*+*i*)) ) ))
· (*Mt*!(*n*−*k*)) = (*s* (*j*+*i*))· (*Mt*!(*n*−*k*)) − ( (*rat-of-int* (*calculate-c* (*s* (*j*+*i*))
*Mt* (*Suc* (*j*+*i*)) ) )
·$_v$ (*RAT M*)!( (*dim-vec* (*s* (*j*+*i*))) −(*Suc* (*j*+*i*)) ) )· (*Mt*!(*n*−*k*))
**using** *s-dim*[*of j*+*i*]

*assms*(2)
*access-index-M-dim*
*dim-vecs-in-Mt*
*carrier-vecI*[*of Mt*!(*n*−*k*) *n*]
*carrier-vecI*[*of* (*RAT M*)!((*dim-vec* (*s* (*j*+*i*))) −(*Suc* (*j*+*i*))) *n*]
*add-scalar-prod-distrib*[*of*
(*s* (*j*+*i*))
*n*
(*rat-of-int* (*calculate-c* (*s* (*j*+*i*)) *Mt* (*Suc* (*j*+*i*)) ) ) ·$_v$ (*RAT M*)!( (*dim-vec* (*s* (*j*+*i*))) −(*Suc* (*j*+*i*)) )
(*Mt*!(*n*−*k*))]
**using** *calculation*(5) *carriers k-in-range minus-scalar-prod-distrib* **by** *blast*

**moreover then have** *lin-scalar-prod*:((*s* (*j*+*i*)) −
( (*rat-of-int* (*calculate-c* (*s* (*j*+*i*)) *Mt* (*Suc* (*j*+*i*)) ) ) ·$_v$ (*RAT M*)!( (*dim-vec* (*s* (*j*+*i*))) −(*Suc* (*j*+*i*)) ) ))
· (*Mt*!(*n*−*k*)) = (*s* (*j*+*i*))· (*Mt*!(*n*−*k*)) − (*rat-of-int* (*calculate-c* (*s* (*j*+*i*)) *Mt* (*Suc* (*j*+*i*)) ) )
∗ ((*RAT M*)!( (*dim-vec* (*s* (*j*+*i*))) −(*Suc* (*j*+*i*)) ) · (*Mt*!(*n*−*k*)))
**by** (*metis dim2 dim-vecs-in-Mt k-in-range scalar-prod-smult-left*)
**moreover have** *step-past-index*:(*dim-vec* (*s* (*j*+*i*))) −(*Suc* (*j*+*i*))<*n*−*k*
**using** *s-dim*[*of j*+*i*] *Suc*(3) *Suc*(2)
**by** (*simp add*: *calculation*(3) *diff-le-mono2 dim1 le-SucI nat-less-le trans-le-add1*)
**moreover have** ( (*RAT M*)!( (*dim-vec* (*s* (*j*+*i*))) −(*Suc* (*j*+*i*)) ) · (*Mt*!(*n*−*k*) ) ) = *0*
**using** *step-past-index upper-tri*[*of* (*dim-vec* (*s* (*j*+*i*))) −(*Suc* (*j*+*i*)) *n*−*k*] *Suc*(4)
**by** *simp*
**then have** (*s* (*Suc* (*j*+*i*)))· *Mt*!(*n*−*k*) = (*s* (*j*+*i*))· *Mt*!(*n*−*k*) −
( (*rat-of-int* (*calculate-c* (*s* (*j*+*i*)) *Mt* (*Suc* (*j*+*i*)) ) ) ∗*0*)
**using** *lin-scalar-prod init-sub*
**by** *algebra*
**then have** *conclusion2*:(*s* (*Suc* (*j*+*i*)))· *Mt*!(*n*−*k*) = (*s* (*j*+*i*))· *Mt*!(*n*−*k*) **by** *auto*
**show** *?case*
**by** (*metis Suc*(2) *Suc*(3) *Suc*(4) *Suc.hyps Suc-leD* ‹*j* + *Suc i* = *Suc* (*j* + *i*)› *conclusion1 conclusion2*)
**qed**

**lemma** *small-orth-coord*:
**fixes** *i*::*nat*
**assumes** *1*≤*i*
**assumes** *i*≤*n*
**shows** *abs* ( (*s i*) · *Mt*!(*n*−*i*)) ≤ (*sq-norm* (*Mt*!(*n*−*i*)))∗(*1*/*2*)
**proof**(−)
**have** *minus-plus*:*Suc* (*i*−*1*) = *i* **using** *assms*(1) **by** *auto*
**then have** *init-sub*:*s i* = (*s* (*i*−*1*))−( (*rat-of-int* (*calculate-c* (*s* (*i*−*1*)) *Mt i* ) ) ·$_v$ (*RAT M*)!( (*dim-vec* (*s* (*i*−*1*))) −*i*))
**using** *sub-s*[*of i*−*1*]

**by** (*metis* (*full-types*) *Suc-le-eq assms*(2) *less-eq-nat.simps*(1))
**then have** *scalar-distrib*:(*s i*) ⋅ *Mt*!(*n*−*i*) = (*s* (*i*−1)) ⋅ *Mt*!(*n*−*i*)−(( (*rat-of-int* (*calculate-c* (*s* (*i*−1)) *Mt i* ) )
$\cdot_v$ (*RAT M*)!( (*dim-vec* (*s* (*i*−1))) −*i*))⋅*Mt*!(*n*−*i*))
**using** *add-scalar-prod-distrib*[*of* (*s* (*i*−1)) *n* ( (*rat-of-int* (*calculate-c* (*s* (*i*−1)) *Mt i* ) )
$\cdot_v$ (*RAT M*)!( (*dim-vec* (*s* (*i*−1))) −*i*)) *Mt*!(*n*−*i*)]
*s-dim*[*of i*−1]
*carrier-vecI*[*of Mt*!(*n*−*i*)]
*carrier-vecI*[*of* (*RAT M*)!( (*dim-vec* (*s* (*i*−1))) −*i*)]
*access-index-M-dim*[*of* ( (*dim-vec* (*s* (*i*−1))) −*i*)]
*dim-vecs-in-Mt*[*of n*−*i*]
*init-sub*
*minus-scalar-prod-distrib*[*of* (*s* (*i*−1)) *n* ( (*rat-of-int* (*calculate-c* (*s* (*i*−1)) *Mt i* ) )
$\cdot_v$ (*RAT M*)!( (*dim-vec* (*s* (*i*−1))) −*i*)) *Mt*!(*n*−*i*)]
**by** (*metis Suc-leD assms*(2) *diff-Suc-less gs.smult-closed le0 minus-plus non-trivial*)
**also have** *scalar-commute*:(*s* (*i*−1)) ⋅ *Mt*!(*n*−*i*)−(( (*rat-of-int* (*calculate-c* (*s* (*i*−1)) *Mt i* ) )
$\cdot_v$ (*RAT M*)!( (*dim-vec* (*s* (*i*−1))) −*i*))⋅*Mt*!(*n*−*i*)) =
(*s* (*i*−1)) ⋅ *Mt*!(*n*−*i*)−( (*rat-of-int* (*calculate-c* (*s* (*i*−1)) *Mt i* ) )
∗ (((*RAT M*)!( (*dim-vec* (*s* (*i*−1))) −*i*)) ⋅*Mt*!(*n*−*i*) ))
**using** *scalar-prod-smult-left*
*carrier-vecI*[*of Mt*!(*n*−*i*)]
*carrier-vecI*[*of* (*RAT M*)!( (*dim-vec* (*s* (*i*−1))) −*i*)]
*access-index-M-dim*
*dim-vecs-in-Mt*
**by** (*smt* (*verit*) *Suc-le-D assms*(2) *diff-less index-minus-vec*(2) *index-smult-vec*(2)
*init-sub minus-plus s-dim zero-less-Suc*)
**moreover have** *index-in-range*: 0≤*n*−*i* ∧ *n*−*i*<*n*
**using** *assms*(1) *assms*(2)
**by** *simp*
**moreover have** *sq-norm-eq*:((*RAT M*)!( (*dim-vec* (*s* (*i*−1))) −*i*)) ⋅*Mt*!(*n*−*i*) = *sq-norm* (*Mt*!(*n*−*i*))
**using** *one-diag*[*of n*−*i*]
*s-dim*[*of i*−1]
*index-in-range*
*assms*(1)
*assms*(2)
*less-imp-diff-less*
**by** *simp*
**then have** (*s i*) ⋅ *Mt*!(*n*−*i*) = (*s* (*i*−1)) ⋅ *Mt*!(*n*−*i*)−
( (*rat-of-int* (*calculate-c* (*s* (*i*−1)) *Mt i* ) ) ∗ *sq-norm* (*Mt*!(*n*−*i*)))
**using** *scalar-distrib scalar-commute sq-norm-eq* **by** *argo*
**then have** *final-sub*:*abs*((*s i*) ⋅ *Mt*!(*n*−*i*)) = *abs*(( (*rat-of-int* (*calculate-c* (*s* (*i*−1)) *Mt i* ) )
∗ *sq-norm* (*Mt*!(*n*−*i*))) − (*s* (*i*−1)) ⋅

```
Mt!(n−i))
    using abs-minus-commute by simp
  then have round-small:abs(rat-of-int (calculate-c (s (i−1)) Mt i )−
                        (((s (i−1)) · (Mt!( (dim-vec (s (i−1))) − i ) ) )
                        / (sq-norm-vec (Mt!( (dim-vec (s (i−1))) − i ) ) ) ))≤1/2
    by (metis calculate-c.simps of-int-round-abs-le)
  moreover have pos:0≤ sq-norm (Mt!(n−i))
    by (simp add: sq-norm-vec-ge-0)
  then have (sq-norm (Mt!(n−i)))*abs((rat-of-int (calculate-c (s (i−1)) Mt i )−
            (((s (i−1)) · (Mt!( (dim-vec (s (i−1))) − i ) ) ) /
            (sq-norm-vec (Mt!( (dim-vec (s (i−1))) − i ) ) ) )))
            ≤(sq-norm (Mt!(n−i)))*(1/2)
    using pos round-small mult-left-mono by blast
  then have 2:abs((sq-norm (Mt!(n−i)))*(rat-of-int (calculate-c (s (i−1)) Mt i
)−
              (((s (i−1)) · (Mt!( (dim-vec (s (i−1))) − i ) ) ) /
                 (sq-norm-vec (Mt!( (dim-vec (s (i−1))) − i ) ) ) ) )≤(sq-norm
(Mt!(n−i)))*(1/2)
    using pos by (smt (verit) abs-mult abs-of-nonneg)
  have i≤n
    using assms(2) by simp
  then have abs(
            (sq-norm (Mt!(n−i)))*(rat-of-int (calculate-c (s (i−1)) Mt i ))−
          (sq-norm (Mt!(n−i)))*( ((s (i−1)) · (Mt!( (dim-vec (s (i−1))) − i ) ) )/
           (sq-norm (Mt!(n−i))) )
           )≤(sq-norm (Mt!(n−i)))*(1/2)
    using 2
        s-dim[of i]
    by (smt (verit) Rings.ring-distribs(4) Suc-leD minus-plus s-dim)
  then have 1:abs(
            (sq-norm (Mt!(n−i)))*(rat-of-int (calculate-c (s (i−1)) Mt i ))−
            ((s (i−1)) · (Mt!( (dim-vec (s (i−1))) − i ) ) )*
           ( (sq-norm (Mt!(n−i)))/(sq-norm (Mt!(n−i))) )
           )≤(sq-norm (Mt!(n−i)))*(1/2)
    using assms(2) s-dim
    by (smt (z3) gs.cring-simprules(14) times-divide-eq-right)
  moreover have nonzero:sq-norm (Mt!(n−i))≠0
    using Mt-gso-connect[of n−i] assms
   by (metis M-locale-2 gram-schmidt-fs-lin-indpt.sq-norm-pos index-in-range length-map
rel-simps(70))
  moreover have cancel:(sq-norm (Mt!(n−i)))/(sq-norm (Mt!(n−i)))=1
    using nonzero
    by auto
  moreover have dim-match:dim-vec (s (i−1)) = n
    using s-dim[of i−1] assms(2)
    by linarith
  then have final-ineq:abs(
            (sq-norm (Mt!(n−i)))*(rat-of-int (calculate-c (s (i−1)) Mt i ))−
            ((s (i−1)) · (Mt!( (dim-vec (s (i−1))) − i ) ) )
```

```
        )≤(sq-norm (Mt!(n−i)))∗(1/2)
    using 1 cancel
    by (smt (verit) gs.r-one)
  then have rearrange-final-ineq: abs( (rat-of-int (calculate-c (s (i−1)) Mt i ))
          ∗ (sq-norm (Mt!(n−i))) −  ((s (i−1)) · (Mt!( n − i ) )  ))≤(sq-norm
(Mt!(n−i)))∗(1/2)
    using dim-match
    by algebra
  show ?thesis
    using final-sub rearrange-final-ineq
    by argo
qed
lemma lattice-carrier: L⊆ carrier-vec n
proof−
  have x∈carrier-vec n if x-def:x∈L for x
  proof−
    obtain f where f-def:x = sumlist (map (λi. (f i)·v M!i ) [0..<n])
      using x-def unfolding L-def lattice-of-def by fast
    have (f i)·v M!i∈carrier-vec n if 0≤i∧i<n for i
      using access-index-M-dim[of i]
      by (metis carrier-vec-dim-vec map-carrier-vec nth-map smult-closed that)
    then have set (map (λi. (f i)·v M!i ) [0..<n]) ⊆ carrier-vec n by auto
    then have sumlist (map (λi. (f i)·v M!i ) [0..<n]) ∈ carrier-vec n by simp
    then show x∈carrier-vec n using f-def by fast
  qed
  then show ?thesis by fast
qed
```

# 3 Lattice Lemmas

```
lemma lattice-sum-close:
  fixes u::int vec and v::int vec
  assumes u∈L v∈L
  shows u+v∈L
proof −
  let ?mM = mat-of-cols n M
  have 1:?mM ∈carrier-mat n n using dim-vecs-in-M by fastforce
  have set-M: set M ⊆ carrier-vec n
    using dim-vecs-in-M carrier-vecI by blast
  have as-mat-mult:lattice-of M = {y∈carrier-vec n. ∃ x∈carrier-vec n. ?mM ∗v x
= y}
     using lattice-of-as-mat-mult[OF set-M] by blast
  then obtain u1 where u1-def:u = ?mM ∗v u1 ∧ u1∈carrier-vec n using assms
unfolding L-def by auto
  obtain v1 where v1-def:v = ?mM ∗v v1 ∧ v1∈carrier-vec n
    using assms as-mat-mult unfolding L-def by auto
  have u1+v1∈carrier-vec n using u1-def v1-def by blast
  moreover have ?mM∗v (u1+v1) = u+v
    using u1-def v1-def 1 mult-add-distrib-mat-vec[of ?mM n n u1 v1]
```

```
    by metis
  moreover have u+v∈carrier-vec n using assms lattice-carrier by blast
  ultimately show u+v∈L
    using as-mat-mult unfolding L-def
    by blast
qed


lemma lattice-smult-close:
  fixes u::int vec and q::int
  assumes u∈L
  shows q·v u∈L

proof−
  let ?mM = mat-of-cols n M
  have 1:?mM ∈carrier-mat n n using dim-vecs-in-M by fastforce
  have set-M: set M ⊆ carrier-vec n
    using dim-vecs-in-M carrier-vecI by blast
  have as-mat-mult:lattice-of M = {y∈carrier-vec n. ∃ x∈carrier-vec n. ?mM ∗v x
= y}
    using lattice-of-as-mat-mult[OF set-M] by blast
  then obtain v::int vec where v-def:u = ?mM ∗v v∧v∈carrier-vec n
    using assms unfolding L-def by auto
  then have q·v v ∈carrier-vec n by blast
  moreover then have q·v u= ?mM ∗v (q·v v) using 1 v-def by fastforce
  ultimately show q·v u ∈ L
   by (metis (mono-tags, lifting) L-def as-mat-mult assms mem-Collect-eq smult-closed)
qed

lemma smult-vec-zero:
  fixes v ::'a::ring vec
  shows 0 ·v v = 0v (dim-vec v)
  unfolding smult-vec-def vec-eq-iff
  by (auto)

lemma coset-s:
  fixes i::nat
  assumes i≤n
  shows s i ∈coset
  using assms
proof(induct i)
  case 0
  have s 0 = −t unfolding s-def by simp
  moreover have carrier-mt:−t∈carrier-vec n using length-M carrier-vecI[of t n]
by fastforce
  ultimately have pzero:s 0 = of-int-hom.vec-hom (0v n) −t by fastforce
  let ?zero = λ j. 0
  have 0<length M using non-trivial by fast
  then have M!0 ∈ set M by force
```

**then have** $M!0 \in L$ **using** *basis-in-latticeI*[*of M M*!*0*] *dim-vecs-in-M carrier-vecI L-def*
**by** *blast*
**then have** $0_v\ n \in L$
**using** *lattice-smult-close*[*of M*!*0 0*] *smult-vec-zero*[*of M*!*0*] *access-index-M-dim*[*of 0*] *non-trivial*
**unfolding** *L-def*
**by** *fastforce*
**then show** *?case* **using** *pzero* **by** *blast*
**next**
**case** (*Suc i*)
**let** *?q* = (*rat-of-int* (*calculate-c* (*s i*) *Mt* (*Suc i*) ) )
**let** *?ind* = ( (*dim-vec* (*s i*)) −(*Suc i*))
**have** *sub*:*s* (*Suc i*) = (*s i*) − (*?q* ·$_v$ (*RAT M*)!*?ind*)
**using** *sub-s*[*of i*] *Suc.prems* **by** *linarith*
**have** *s i* ∈*coset* **using** *Suc* **by** *auto*
**then obtain** *x* **where** *x-def*:*x*∈*L* ∧ (*s i*) = *of-int-hom.vec-hom x*−*t* **by** *blast*
**have** ( *?q* ·$_v$ (*RAT M*)!*?ind*)∈ *of-int-hom.vec-hom*‘ *L*
**proof**−
**have** *dim-vec* (*s i*) = *n* **using** *s-dim*[*of i*] *Suc.prems* **by** *fastforce*
**then have** *in-range*:*?ind*<*n*∧ *0*≤*?ind* **using** *Suc.prems* **by** *simp*
**then have** *com-hom*:(*RAT M*)!(*?ind*) = *of-int-hom.vec-hom* (*M*!*?ind*) **by** *auto*
**have** *M*!*?ind*∈*set M* **using** *in-range* **by** *simp*
**then have** *mil*:*M*!*?ind* ∈ *L* **using** *basis-in-latticeI*[*of M M*!*?ind*] *dim-vecs-in-M carrier-vecI L-def*
**by** *blast*
**moreover have** *?q*·$_v$(*of-int-hom.vec-hom* (*M*!*?ind*)) =
*of-int-hom.vec-hom* ((*calculate-c* (*s i*) *Mt* (*Suc i*) ) ·$_v$ *M*!*?ind*)
**by** *fastforce*
**moreover have** (*calculate-c* (*s i*) *Mt* (*Suc i*) ) ·$_v$ *M*!*?ind*∈*L*
**using** *lattice-smult-close*[*of M*!*?ind* (*calculate-c* (*s i*) *Mt* (*Suc i*) )] *mil* **by** *simp*
**ultimately show** ( *?q* ·$_v$ (*RAT M*)!*?ind*) ∈ *of-int-hom.vec-hom*‘ *L*
**using** *com-hom*
**by** *force*
**qed**
**then obtain** *y* **where** *y-def*:( *?q* ·$_v$ (*RAT M*)!*?ind*) = *of-int-hom.vec-hom y*∧ *y*∈*L* **by** *blast*
**have** *carrier-x*: *x*∈*carrier-vec n* **using** *lattice-carrier x-def* **by** *blast*
**have** *carrier-y*: *y*∈*carrier-vec n* **using** *lattice-carrier y-def* **by** *blast*
**then have** *carrier-my*: −*y*∈*carrier-vec n* **by** *simp*
**then have** *1*:−( *?q* ·$_v$ (*RAT M*)!*?ind*) = *of-int-hom.vec-hom* (−*y*) **using** *y-def* **by** *fastforce*
**then have** *s* (*Suc i*) = *of-int-hom.vec-hom x*−*t*+ *of-int-hom.vec-hom* (−*y*)
**using** *sub x-def y-def 1* **by** *fastforce*
**then have** *s* (*Suc i*) = *of-int-hom.vec-hom x* + *of-int-hom.vec-hom* (−*y*) − *t*
**using** *lattice-carrier x-def y-def length-M*
**by** *fastforce*
**moreover have** *of-int-hom.vec-hom x* + *of-int-hom.vec-hom* (−*y*) = *of-int-hom.vec-hom*

$(x+ -y)$
  **using** *carrier-my carrier-x* **by** *fastforce*
 **ultimately have** *2*:*s* (*Suc i*) = *of-int-hom.vec-hom* (*x*+ −*y*) −*t*
  **by** *metis*
 **have** −*y* = −*1* ·$_v$ *y* **by** *auto*
 **then have** −*y*∈*L* **using** *lattice-smult-close y-def* **by** *simp*
 **then have** *x*+−*y*∈*L* **using** *lattice-sum-close x-def* **by** *simp*
 **then show** *?case* **using** *2* **by** *fast*
**qed**

**lemma** *subtract-coset-into-lattice*:
 **fixes** *v*::*rat vec*
 **fixes** *w*::*rat vec*
 **assumes** *v*∈*coset*
 **assumes** *w*∈*coset*
 **shows** (*v*−*w*)∈*of-int-hom.vec-hom*‘ *L*
**proof**−
 **obtain** *l1* **where** *l1-def*:*v*=*l1* −*t*∧ *l1*∈*of-int-hom.vec-hom*‘ *L* **using** *assms*(*1*) **by** *blast*
 **obtain** *l2* **where** *l2-def*:*w* = *l2*−*t*∧ *l2*∈*of-int-hom.vec-hom*‘ *L* **using** *assms*(*2*) **by** *blast*
 **have** *carrier-l1*:*l1* ∈ *carrier-vec n* **using** *lattice-carrier l1-def* **by** *force*
 **have** *carrier-l2*:*l2* ∈ *carrier-vec n* **using** *lattice-carrier l2-def* **by** *force*
 **obtain** *l1p* **where** *l1p-def*:*l1* = *of-int-hom.vec-hom l1p* ∧ *l1p*∈*L* **using** *l1-def* **by** *fast*
 **obtain** *l2p* **where** *l2p-def*:*l2* = *of-int-hom.vec-hom l2p* ∧ *l2p*∈*L* **using** *l2-def* **by** *fast*
 **have** −*l2p* = −*1*·$_v$ *l2p* **using** *carrier-l2* **by** *fastforce*
 **then have** *ml2p*:−*l2p*∈ *L* **using** *lattice-smult-close*[*of l2p* −*1*] *l2p-def* **by** *presburger*
 **then have** *of-int-hom.vec-hom* (−*l2p*)∈ *of-int-hom.vec-hom*‘ *L* **by** *simp*
 **moreover have** *of-int-hom.vec-hom* (−*l2p*) = −*l2* **using** *l2p-def* **by** *fastforce*
 **then have** *l1*−*l2* = *of-int-hom.vec-hom* (*l1p* − *l2p*) **using** *l1p-def l2p-def carrier-l1 carrier-l2* **by** *auto*
 **moreover have** *l1p*−*l2p*∈*L* **using** *lattice-sum-close*[*of l1p* −*l2p*]
   *l1p-def l2p-def ml2p carrier-l1 carrier-l2*
  **by** (*simp add*: *minus-add-uminus-vec*)
 **ultimately have** *l1*−*l2*∈ *of-int-hom.vec-hom*‘ *L* **by** *fast*
**moreover have** *v*−*w* = *l1*−*l2* **using** *l1-def l2-def length-M carrier-vecI carrier-l1 carrier-l2* **by** *force*
 **ultimately show** *?thesis* **by** *simp*
**qed**
**lemma** *t-in-coset*:
 **shows** *uminus t* ∈ *coset*
 **using** *coset-s*[*of 0*] *babai-help.simps* **unfolding** *s-def* **by** *simp*

# 4 Lemmas on closest distance

**lemma** *closest-distance-sq-pos*: *closest-distance-sq*≥*0*

**proof**−
**have** $\forall N \in \{real\text{-}of\text{-}rat\ (sq\text{-}norm\ x::rat)\ |x.\ x \in coset\}.\ 0 \leq N$
**using** *sq-norm-vec-ge-0* **by** *auto*
**moreover have** $\{real\text{-}of\text{-}rat\ (sq\text{-}norm\ x::rat)\ |x.\ x \in coset\} \neq \{\}$ **using** *t-in-coset*
**by** *blast*
**ultimately have** $0 \leq Inf\ \{real\text{-}of\text{-}rat\ (sq\text{-}norm\ x::rat)\ |x.\ x \in coset\}$
**by** (*meson cInf-greatest*)
**then show** *?thesis* **unfolding** *closest-distance-sq-def* **by** *blast*
**qed**

**definition** *witness*:: *rat vec*⇒*rat* ⇒ *bool*
**where** *witness v eps-closest* = (*sq-norm v* ≤ *eps-closest* ∧ *v*∈*coset*∧*dim-vec v* = *n*)

**definition** *close-condition*::*rat* ⇒ *bool*
**where** *close-condition eps-closest* ≡
(*if closest-distance-sq* = *0 then* 0≤ *real-of-rat eps-closest*
*else real-of-rat* (*eps-closest*)>*closest-distance-sq*)
∧ (*real-of-rat* (*eps-closest*)≤*epsilon*∗*closest-distance-sq*)

**lemma** *close-rat*:
**obtains** *eps-closest*::*rat*
**where** *close-condition eps-closest*
**proof**(*cases closest-distance-sq* = *0*)
**case** *t*:*True*
**then have** *epsilon*∗*closest-distance-sq* = *real-of-rat* (*0*::*rat*) **by** *simp*
**then have** *real-of-rat* (*0*::*rat*)≤ *epsilon*∗*closest-distance-sq*∧*closest-distance-sq*
≤(*real-of-rat* (*0*::*rat*))
**using** *t* **by** *force*
**then show** *?thesis*
**using** *that t* **unfolding** *close-condition-def* **by** *metis*
**next**
**case** *f*:*False*
**then have** *0*<*closest-distance-sq*
**using** *closest-distance-sq-pos* **by** *linarith*
**moreover have** (*1*::*real*)<*epsilon* **using** *epsilon* **by** *simp*
**ultimately have** *closest-distance-sq*<*epsilon*∗*closest-distance-sq* **by** *simp*
**then show** *?thesis*
**using** *Rats-dense-in-real*[*of closest-distance-sq epsilon*∗*closest-distance-sq*] *that*
**unfolding** *close-condition-def*
**by** (*metis Rats-cases less-eq-real-def*)
**qed**

**definition** *eps-closest*::*rat*
**where** *eps-closest* = (*if* ∃ *r. close-condition r then SOME r. close-condition r*
*else 0*)

**lemma** *eps-closest-lemma*: *close-condition eps-closest*
**using** *close-rat* **unfolding** *eps-closest-def* **by** (*metis* (*full-types*))

**lemma** *rational-tri-ineq*:
**fixes** *v*::*rat vec*
**fixes** *w*::*rat vec*
**assumes** *dim-vec v = dim-vec w*
**shows** (*sq-norm* (*v*+*w*))≤ *4*∗(*Max* {(*sq-norm v*), (*sq-norm w*)})
**proof**−
**let** *?d = dim-vec w*
**let** *?M = Max* {(*sq-norm v*), (*sq-norm w*)}
**have** *carr-v*:*v*∈*carrier-vec ?d* **using** *assms carrier-vecI*[*of v ?d*] **by** *fastforce*
**have** *carr-w*:*w*∈*carrier-vec ?d* **using** *carrier-vecI*[*of w ?d*] **by** *fastforce*
**have** *carr-vw*:*v*+*w*∈*carrier-vec ?d* **using** *carr-v carr-w add-carrier-vec* **by** *blast*
**have** *sq-norm* (*v*+*w*) = (*v*+*w*)•(*v*+*w*)
**by** (*simp add*: *sq-norm-vec-as-cscalar-prod*)
**also have** (*v*+*w*)•(*v*+*w*) = *v*•(*v*+*w*)+*w*•(*v*+*w*)
**using** *add-scalar-prod-distrib*[*of v ?d w v*+*w*]
*carr-v carr-w carr-vw* **by** *blast*
**also have** *v*•(*v*+*w*)+*w*•(*v*+*w*) = *v*•*v*+*v*•*w*+*w*•*v*+*w*•*w*
**using** *scalar-prod-add-distrib*[*of v ?d v w*]
*scalar-prod-add-distrib*[*of w ?d v w*]
*carr-v carr-w carr-vw* **by** *algebra*
**also have** *v*•*w*=*w*•*v*
**using** *carr-v carr-w comm-scalar-prod* **by** *blast*
**also have** *v*•*v* = *sq-norm v*
**using** *sq-norm-vec-as-cscalar-prod*[*of v*] **by** *force*
**also have** *w*•*w* = *sq-norm w*
**using** *sq-norm-vec-as-cscalar-prod*[*of w*] **by** *force*
**finally have** *sq-norm* (*v*+*w*) = *sq-norm v* + *sq-norm w* + *2*∗(*w*•*v*) **by** *force*
**also have** *b1*:*sq-norm v*≤*?M* **by** *force*
**also have** *b2*:*sq-norm w*≤*?M* **by** *force*
**also have** *2*∗(*w*•*v*)≤*2*∗(*Max* {(*sq-norm v*), (*sq-norm w*)})
**proof**−
**have** (*w*•*v*)^*2*≤ (*sq-norm v*) ∗ (*sq-norm w*)
**using** *scalar-prod-Cauchy*[*of w ?d v*] *carr-w carr-v* **by** *algebra*
**also have** (*sq-norm v*) ∗ (*sq-norm w*)≤*?M*∗*?M*
**using** *b1 b2 sq-norm-vec-ge-0*[*of w*] *sq-norm-vec-ge-0*[*of v*]
*mult-mono*[*of sq-norm v ?M sq-norm w ?M*] **by** *linarith*
**also have** *?M*∗*?M = ?M*^*2*
**using** *power2-eq-square*[*of ?M*] **by** *presburger*
**finally have** (*w*•*v*)^*2*≤*?M*^*2* **by** *blast*
**also have** (*w*•*v*)^*2*=*abs*(*w*•*v*)^*2* **by** *force*
**finally have** *abs*(*w*•*v*)^*2*≤*?M*^*2* **by** *presburger*
**moreover have** *0*≤*abs*(*w*•*v*) **by** *fastforce*
**moreover have** *0*≤*?M*
**using** *sq-norm-vec-ge-0*[*of w*] *sq-norm-vec-ge-0*[*of v*] **by** *fastforce*
**ultimately have** *abs*(*w*•*v*)≤*?M*
**using** *power2-le-imp-le* **by** *blast*
**also have** (*w*•*v*)≤*abs*(*w*•*v*) **by** *force*
**finally show** *?thesis* **by** *linarith*

```
  qed
  finally show ?thesis by auto
qed

lemma witness-exists:
  shows ∃ v. witness v eps-closest
proof(cases closest-distance-sq = 0)
  case t:True
  have eps-closest = 0
    using eps-closest-lemma t
    unfolding witness-def unfolding close-condition-def
    by auto
  then have equiv:?thesis = (∃ v. v∈coset∧ (dim-vec v = n) ∧ (sq-norm v) ≤ 0)
    unfolding witness-def eps-closest-def by auto
  show ?thesis
  proof(rule ccontr)
    assume contra:¬?thesis
    have {real-of-rat (sq-norm x::rat) |x. x ∈ coset}≠{} using t-in-coset by fast
   then have limit-point:∃ v::rat vec. real-of-rat (sq-norm v) < (eps::real) ∧ v∈coset
if 0<eps for eps
      using t cInf-lessD[of {real-of-rat (sq-norm x::rat) |x. x ∈ coset} eps] that
      unfolding closest-distance-sq-def by auto
    moreover have 0<real-of-rat ((sq-norm ((RAT M)!0) )/ (4∗α^(n−1)))
    proof−
      have 0<1/(4∗α^(n−1)) using non-trivial alpha by force
      moreover have 0< (sq-norm ((RAT M)!0))
        using gram-schmidt-fs-lin-indpt.sq-norm-pos[of n RAT M 0]
              gram-schmidt-fs-lin-indpt.sq-norm-gso-le-f[of n RAT M 0]
              M-locale-2 non-trivial
        by fastforce
      ultimately show ?thesis by auto
    qed
    ultimately obtain v::rat vec where v-def:real-of-rat (sq-norm v)
                          < real-of-rat ((sq-norm ((RAT M)!0) )/ (4∗α^(n−1)))∧
v∈coset
      by presburger
    then have dim-vec v = n
      using length-M by force
    then have 0< real-of-rat (sq-norm v)
      using equiv contra v-def by auto
     then obtain w::rat vec where w-def:real-of-rat (sq-norm w) < real-of-rat
(sq-norm v)∧w∈coset
      using limit-point by fast
   then have small-w:real-of-rat (sq-norm w)<real-of-rat ((sq-norm ((RAT M)!0)
)/ (4∗α^(n−1)))
      using v-def by argo
    have lat:w−v∈ of-int-hom.vec-hom‘ L using subtract-coset-into-lattice[of w v]
      using v-def w-def by force
    then obtain l where l-def:l∈L∧w−v=of-int-hom.vec-hom l by blast
```

**then have** *of-int-hom.vec-hom l* ∈ *gs.lattice-of* (*RAT M*)
**using** *lattice-of-of-int*[*of M n l*] *dim-vecs-in-M carrier-vecI L-def* **by** *blast*
**then have** *lat-hom*:*w*−*v* ∈ *gs.lattice-of* (*RAT M*) **using** *l-def* **by** *simp*
**have** *sq-norm v* ≠ *sq-norm w* **using** *w-def* **by** *auto*
**then have** *neq*:*w*≠*v* **by** *meson*
**have** *c1*:*w*∈*carrier-vec n* **using** *length-M w-def lattice-carrier carrier-dim-vec* **by** *fastforce*
**moreover have** *c2*:*v*∈*carrier-vec n* **using** *length-M v-def lattice-carrier carrier-dim-vec* **by** *fastforce*
**ultimately have** *c3*:*w*−*v*∈*carrier-vec n* **by** *simp*
**have** *neqzero*:*w*−*v*≠$0_v$ *n*
**proof**(*rule ccontr*)
**assume** *c*:¬ *?thesis*
**have** *w*−*v*=$0_v$ *n* **using** *c* **by** *blast*
**then have** *w*=*v*+ $0_v$ *n* **using** *c1 c2 c3*
**by** (*smt* (*verit*, *ccfv-SIG*) *gs.M.add.r-inv-ex minus-add-minus-vec minus-cancel-vec minus-zero-vec right-zero-vec*)
**then show** *False* **using** *c2 neq* **by** *simp*
**qed**
**then have** *w*−*v* ∈ *gs.lattice-of* (*RAT M*) − {$0_v$ *n*} **using** *lat-hom* **by** *blast*
**moreover have** *α*^(*n*−1) ∗ (*sq-norm* (*w*−*v*)) < (*sq-norm* ((*RAT M*)!0) )
**proof**−
**have** *w*−*v* = *w*+ (−*v*) **by** *fastforce*
**then have** *sq-norm* (*w*−*v*) = *sq-norm* (*w*+(−*v*)) **by** *simp*
**also have** *sq-norm* (*w*+(−*v*)) ≤ 4∗*Max*({*sq-norm w*, *sq-norm* (−*v*)})
**using** *rational-tri-ineq*[*of w* −*v*] *c1 c2* **by** *fastforce*
**also have** *sq-norm* (−*v*) = *sq-norm v*
**proof**−
**have** −*v* = (−1)·$_v$ *v* **by** *fastforce*
**then have** *sq-norm* (−*v*) = ((−1)·$_v$ *v*)•((−1)·$_v$ *v*) **using** *sq-norm-vec-as-cscalar-prod*[*of* −*v*] **by** *force*
**then have** *sq-norm* (−*v*) = (−1)∗(−1)∗(*v*•*v*) **using** *c1 c2* **by** *simp*
**then show** *?thesis* **using** *sq-norm-vec-as-cscalar-prod*[*of v*] **by** *simp*
**qed**
**also have** *Max*({*sq-norm w*, *sq-norm* (*v*)})<((*sq-norm* ((*RAT M*)!0) )/ (4∗*α*^(*n*−1)))
**using** *v-def small-w of-rat-less* **by** *auto*
**finally have** *sq-norm* (*w*−*v*)<4∗((*sq-norm* ((*RAT M*)!0) )/ (4∗*α*^(*n*−1))) **by** *linarith*
**then have** *sq-norm* (*w*−*v*)<(*sq-norm* ((*RAT M*)!0) )/ (*α*^(*n*−1)) **by** *linarith*
**moreover have** *p*:0<*α*^(*n*−1) **using** *alpha* **by** *fastforce*
**ultimately show** *?thesis* **using** *p*
**by** (*metis gs.cring-simprules*(14) *pos-less-divide-eq*)
**qed**
**ultimately show** *False*
**using** *gram-schmidt-fs-lin-indpt.weakly-reduced-imp-short-vector*[*of n* (*RAT M*) *α w*−*v*]
*M-locale-2 reduced alpha*
**unfolding** *gs.reduced-def L-def* **by** *force*

**qed**
**next**
**case** *False*
**then have** *closest-distance-sq* < *real-of-rat eps-closest*
**using** *eps-closest-lemma* **unfolding** *eps-closest-def close-condition-def*
**by** *presburger*
**moreover have** {*real-of-rat* (*sq-norm x*::*rat*) |*x*. *x* ∈ *coset*}≠{} **using** *t-in-coset*
**by** *fast*
**ultimately obtain** *l* **where** *l*∈{*real-of-rat* (*sq-norm x*::*rat*) |*x*. *x* ∈ *coset*}∧ *l*<
*real-of-rat eps-closest*
**using** *closest-distance-sq-pos*
**unfolding** *closest-distance-sq-def*
**by** (*meson cInf-lessD*)
**moreover then obtain** *v*::*rat vec* **where** *l* = *real-of-rat* (*sq-norm v*) ∧ *v*∈*coset*
**by** *blast*
**ultimately show** *?thesis* **unfolding** *witness-def lattice-carrier*
**by** (*smt* (*verit*) *length-M index-minus-vec*(*2*) *mem-Collect-eq of-rat-less-eq*)
**qed**

# 5 More linear algebra lemmas

**lemma** *carrier-Ms*:
**shows** *mat-M* ∈*carrier-mat n n mat-M-inv* ∈*carrier-mat n n*
**using** *M-dim M-inv-dim*
**apply** *blast*
**by** (*simp add*: *M-inv-dim*(*1*) *M-inv-dim*(*2*) *carrier-matI*)

**lemma** *carrier-L*:
**fixes** *v*::*rat vec*
**assumes** *dim-vec v* = *n*
**shows** *lattice-coord v*∈*carrier-vec n*
**unfolding** *lattice-coord-def*
**using** *mult-mat-vec-carrier*[*of mat-M-inv n n v*]
*carrier-Ms*
*carrier-vecI*[*of v*]
*assms*(*1*)
**by** *fast*

**lemma** *sumlist-index-commute*:
**fixes** *Lst*::*rat vec list*
**fixes** *i*::*nat*
**assumes** *set Lst*⊆*carrier-vec n*
**assumes** *i*<*n*
**shows** (*gs.sumlist Lst*)$*i* = *sum-list* (*map* (λ*j*. (*Lst*!*j*)$*i*) [*0*..<(*length Lst*)])
**using** *assms*
**proof**(*induct Lst*)
**case** *Nil*
**have** *gs.sumlist Nil* = $0_v$ *n* **using** *assms* **unfolding** *gs.sumlist-def* **by** *auto*
**then have** *lhs*:(*gs.sumlist Nil*)$*i* = *0* **using** *assms*(*2*) **by** *auto*

**have** $[0..<(length\ Nil)] = Nil$ **by** *simp*
**then have** $(map\ (\lambda j.\ (Nil!j)\$i)\ [0..<(length\ Nil)]) = Nil$ **by** *blast*
**then have** $sum\text{-}list\ (map\ (\lambda j.\ (Nil!j)\$i)\ [0..<(length\ Nil)]) = 0$ **by** *simp*
**then show** *?case* **using** *lhs* **by** *simp*
**next**
**case** $(Cons\ a\ Lst)$
**let** $?CaLst = Cons\ a\ Lst$
**have** $set\ Lst \subseteq carrier\text{-}vec\ n$ **using** *Cons.prems* **by** *auto*
**then have** *carr*:$gs.sumlist\ Lst \in carrier\text{-}vec\ n$ **using** *assms gs.sumlist-carrier*[*of Lst* ]
**by** *blast*
**have** $gs.sumlist\ (Cons\ a\ Lst) = a + gs.sumlist\ Lst$ **by** *simp*
**then have** *lhs*:$(gs.sumlist\ ?CaLst)\$i = a\$i + (gs.sumlist\ Lst)\$i$ **using** *assms carr* **by** *simp*
**have** $sum\text{-}list\ (map\ (\lambda j.\ (?CaLst!j)\$i)\ [0..<(length\ ?CaLst)]) = sum\text{-}list\ (map\ (\lambda l.\ l\$i)\ ?CaLst)$
**by** (*smt* (*verit*) *length-map map-eq-conv' map-nth nth-map*)
**moreover have** $sum\text{-}list\ (map\ (\lambda l.\ l\$i)\ ?CaLst) = a\$i + sum\text{-}list\ (map\ (\lambda l.\ l\$i)\ Lst)$ **by** *simp*
**moreover have** $sum\text{-}list\ (map\ (\lambda l.\ l\$i)\ Lst) = sum\text{-}list\ (map\ (\lambda j.\ (Lst!j)\$i)\ [0..<(length\ Lst)])$
**by** (*smt* (*verit*) *length-map map-eq-conv' map-nth nth-map*)
**moreover have** $sum\text{-}list\ (map\ (\lambda j.\ (Lst!j)\$i)\ [0..<(length\ Lst)]) = (gs.sumlist\ Lst)\$i$
**using** *Cons.prems Cons.hyps* **by** *simp*
**ultimately show** *?case* **using** *lhs*
**by** *argo*
**qed**

**lemma** *mat-mul-to-sum-list*:
**fixes** *A*::*rat mat*
**fixes** *v*::*rat vec*
**assumes** $dim\text{-}vec\ v = dim\text{-}col\ A$
**assumes** $dim\text{-}row\ A = n$
**shows** $A *_v v = gs.sumlist\ (map\ (\lambda j.\ v\$j\ \cdot_v\ (col\ A\ j))\ [0\ ..<\ dim\text{-}col\ A])$
**proof**−
**have** *carrier*:$set\ (map\ (\lambda j.\ v\ \$\ j \cdot_v col\ A\ j)\ [0..<dim\text{-}col\ A]) \subseteq Rn$
**by** (*smt* (*verit*) *assms*(*2*) *carrier-dim-vec dim-col ex-map-conv index-smult-vec*(*2*) *subset-code*(*1*))
**have** $(A *_v v)\$i = gs.sumlist\ (map\ (\lambda j.\ v\$j\ \cdot_v\ (col\ A\ j))\ [0\ ..<\ dim\text{-}col\ A])\$i$ **if** *small*:$i<dim\text{-}row\ A$ **for** *i*
**proof**−
**let** $?rAi = row\ A\ i$

**have** *1*:$(A *_v v)\$i = ?rAi \cdot v$ **using** *small* **by** *simp*
**have** *2*:$?rAi \cdot v = sum\text{-}list\ (map\ (\lambda j.\ (?rAi\$j)*(v\$j))\ [0..<dim\text{-}col\ A])$
**using** *assms sum-set-upt-conv-sum-list-nat* **unfolding** *scalar-prod-def* **by** *auto*
**have** $?rAi\$j*(v\$j) = (v\$j\ \cdot_v\ (col\ A\ j))\$i$ **if** *jsmall*:$j<dim\text{-}col\ A$ **for** *j*

**unfolding** *row-def col-def* **using** *small jsmall*
**by** *force*
**then have** (*map* (λ*j*. (*?rAi*$*j*)∗(*v*$*j*)) [*0*..<*dim-col A*]) = (*map* (λ*j*. (*v*$*j* ·$_v$ (*col A j*))$*i*) [*0*..<*dim-col A*])
**by** *fastforce*
**then have** (*A*∗$_v$*v*)$*i* = *sum-list* (*map* (λ*j*. (*v*$*j* ·$_v$ (*col A j*))$*i*) [*0*..<*dim-col A*])
**using** *1 2* **by** *algebra*
**then show** *?thesis* **using** *sumlist-index-commute*[*of map* (λ*j*. *v*$*j* ·$_v$ (*col A j*)) [*0* ..< *dim-col A*] *i*]
*small assms*(*2*) *carrier*
**by** (*smt* (*verit*) *gs.sumlist-vec-index length-map map-equality-iff nth-map subset-code*(*1*))
**qed**
**moreover have** *dim-vec* (*A*∗$_v$*v*) = *dim-row A* **by** *fastforce*
**moreover have** *dim-vec* (*gs.sumlist* (*map* (λ*j*. *v*$*j* ·$_v$ (*col A j*)) [*0* ..< *dim-col A*])) = *n*
**using** *carrier* **by** *auto*
**ultimately show** *?thesis* **using** *assms*
**by** *auto*
**qed**

**lemma** *recover-from-lattice-coord*:
**fixes** *v*::*rat vec*
**assumes** *dim-vec v* = *n*
**shows** *v* = *gs.sumlist* (*map* (λ*i*. (*lattice-coord v*)$*i* ·$_v$ (*RAT M*)!*i*) [*0* ..< *n*])
**proof** −
**have** (*mat-M* ∗ *mat-M-inv*)∗$_v$ *v*= *mat-M*∗$_v$(*lattice-coord v*)
**unfolding** *lattice-coord-def*
**using** *assms*(*1*) *carrier-Ms carrier-vecI*[*of v*]
*assoc-mult-mat-vec*[*of mat-M n n mat-M-inv n v*]
**by** *presburger*
**then have** (*1*$_m$ *n*)∗$_v$*v*=*mat-M*∗$_v$(*lattice-coord v*)
**using** *inv1*
**by** *simp*
**then have** *v* = *mat-M*∗$_v$(*lattice-coord v*)
**by** (*metis assms carrier-vec-dim-vec one-mult-mat-vec*)
**then have** *pre*:*v* = *gs.sumlist* (*map* (λ*i*. (*lattice-coord v*)$*i* ·$_v$ *col mat-M i*) [*0* ..< *dim-col mat-M*])
**using** *mat-mul-to-sum-list*[*of lattice-coord v mat-M*]
*M-dim*
*assms*
*dim-preserve-lattice-coord*
**by** *simp*
**moreover have** *col mat-M i* = (*RAT M*)!*i* **if** *i*<*n* **for** *i*
**using** *vec-to-col*
**by** (*simp add*: *that*)
**ultimately have** (*map* (λ*i*. (*lattice-coord v*)$*i* ·$_v$ *col mat-M i*) [*0* ..< *dim-col mat-M*]) =

```
                (map (λi. (lattice-coord  v)$i  ·v (RAT M)!i) [0 ..< n]) using
M-dim
    by simp
  then show v = gs.sumlist (map (λi. (lattice-coord  v)$i  ·v (RAT M)!i) [0 ..<
n])
    using pre by presburger
qed

lemma sumlist-linear-coord:
  fixes Lst::int vec list
  assumes ⋀i. i<length Lst ⟹dim-vec (Lst!i) = n
   shows lattice-coord (map-vec rat-of-int (sumlist Lst)) = gs.sumlist (map lat-
tice-coord (RAT Lst))
using assms
proof(induct Lst)
  case Nil
  have rhs:gs.sumlist(map lattice-coord (RAT Nil)) = 0v n by fastforce
  have map-vec rat-of-int (sumlist Nil) = 0v n by auto
  then have lattice-coord (map-vec rat-of-int (sumlist Nil)) = 0v n
    unfolding lattice-coord-def using M-inv-dim
   by (metis carrier-Ms(2) gs.M.add.r-cancel-one' gs.M.zero-closed mult-add-distrib-mat-vec
mult-mat-vec-carrier)
  then show ?case using rhs by simp
next
  case (Cons a Lst)
  let ?CaLst = Cons a Lst
  let ?ra = of-int-hom.vec-hom a
  have dim:i∈set ?CaLst ⟹ dim-vec i = n for i using Cons.prems
    by (metis in-set-conv-nth)
  then have i-lt: (i < length Lst ⟹ dim-vec (Lst ! i) = n) for i
    using Cons.prems carrier-dim-vec by auto
  have carrier:set ?CaLst⊆ carrier-vec n using Cons.prems
    using carrier-vecI dim by fast
  then have carrier-sumCaLst: (sumlist ?CaLst)∈carrier-vec n by force
  have carrier-a: a ∈ carrier-vec n using carrier by force
  have carrier-Lst:set Lst ⊆ carrier-vec n using carrier by simp
  have lhs:lattice-coord (map-vec rat-of-int (sumlist ?CaLst)) = (lattice-coord ?ra)
+ gs.sumlist (map lattice-coord (RAT Lst))
  proof−
    have carrier-sumLst: sumlist Lst∈carrier-vec n using carrier-Lst by force
    have sumlist ?CaLst = a + sumlist Lst by force
    then have (map-vec rat-of-int (sumlist ?CaLst)) = ?ra + (map-vec rat-of-int
(sumlist Lst))
      using carrier-a carrier-sumLst carrier-sumCaLst by auto
    then have lattice-coord (map-vec rat-of-int (sumlist ?CaLst))
            = lattice-coord(?ra) + lattice-coord(map-vec rat-of-int (sumlist Lst))
      unfolding lattice-coord-def
      using carrier-sumCaLst carrier-a carrier-sumLst
      by (metis carrier-Ms(2) map-carrier-vec mult-add-distrib-mat-vec)
```

```
    then show ?thesis using i-lt Cons.hyps
      by algebra
  qed
  moreover have rhs:gs.sumlist (map lattice-coord (RAT ?CaLst)) =
                  (lattice-coord ?ra) + gs.sumlist (map lattice-coord (RAT Lst))
    by fastforce
  ultimately show ?case by argo
qed


lemma integral-sum:
  fixes l::nat
  assumes ⋀j1. j1 < l ⟹
    map f [0..<l] ! j1  ∈ ℤ
  shows sum-list
     (map f [0..<l]) ∈ ℤ
  using assms
proof(induct l)
  case 0
  have (map f [0..<0]) = Nil by auto
  then have sum-list (map f [0..<0]) = 0 by simp
  then show ?case by simp
next
  case (Suc l)
  have nontriv:Suc l>0 by simp
  have break:sum-list (map f [0..<(Suc l)]) = sum-list (map f [0..<l]) + (f l) by
fastforce
  have l<Suc l by simp
  then have [0..<(Suc l)]!l = l
    by (metis nth-upt plus-nat.add-0)
  moreover then have f ([0..<(Suc l)] ! l) = (map f [0..<(Suc l)]) ! l
   by (metis One-nat-def Suc-diff-Suc diff-Suc-1 local.nontriv nat-SN.default-gt-zero

        nth-map-upt nth-upt plus-1-eq-Suc real-add-less-cancel-right-pos)
  ultimately have z:f l ∈ℤ using Suc.prems by fastforce
  have ⋀j1. j1 < l ⟹
    map f [0..<l] ! j1  ∈ ℤ
    by (metis Suc.prems diff-Suc-1' diff-Suc-Suc less-SucI nth-map-upt)
  then have sum-list (map f [0..<l])∈ℤ using Suc by blast
  then show ?case using z break by force
qed


lemma int-coord:
  fixes i::nat
  assumes 0≤i
  assumes i<n
  fixes v::int vec
  assumes v∈L
```

**assumes** *dim-vec v = n*
**shows** (*lattice-coord* (*map-vec rat-of-int v*))$*i*∈ℤ
**proof** −
**obtain** *w* **where** *w-def*:*v* = *sumlist* (*map* (λ *i*. *of-int* (*w i*) ·$_v$ *M* ! *i*) [*0* ..< *length M*])
**using** *L-def assms*(*3*) *vec-module.lattice-of-def*
**by** *blast*
**let** *?Lst* = (*map* (λ *i*. *of-int* (*w i*) ·$_v$ *M* ! *i*) [*0* ..< *length M*])
**have** *dims-j*:*dim-vec* (*?Lst*!*j*) = *n* **if** *j-lt*:*j*<*length ?Lst* **for** *j*
**using** *access-index-M-dim carrier-vecI j-lt* **by** *force*
**let** *?recover* = (*map lattice-coord* (*RAT ?Lst*))
**have** *1*:*lattice-coord* (*map-vec rat-of-int v*) = *gs.sumlist ?recover*
**using** *sumlist-linear-coord*[*of ?Lst*]
*w-def*
*dims-j*
**by** *blast*
**have** *int-recover*:⋀*j*. *j*<*n*⟹(*?recover*!*j*)$*i* ∈ℤ∧ (*dim-vec* (*?recover*!*j*)) = *n*
**proof** −
**fix** *j*::*nat*
**assume** *small*:*j*<*n*
**have** *?recover*!*j* = *lattice-coord* ((*RAT ?Lst*)!*j*)
**using** *List.nth-map*[*of j* (*RAT ?Lst*) *lattice-coord*]
*small*
**by** *simp*
**then have** *?recover*!*j* = *lattice-coord* (*of-int-hom.vec-hom* (*?Lst*!*j*))
**using** *List.nth-map*[*of j ?Lst of-int-hom.vec-hom*]
*small*
**by** *simp*
**then have** *?recover*!*j* = *lattice-coord* (*of-int-hom.vec-hom* (*of-int* (*w j*) ·$_v$ *M* ! *j*))
**using** *List.nth-map*[*of j* [*0* ..< *length M*] (λ *i*. *of-int* (*w i*) ·$_v$ *M* ! *i*)]
*small*
**by** *simp*
**then have** *commuted-maps*:*?recover*!*j* = *mat-M-inv* ∗$_v$ (*of-int-hom.vec-hom* (*of-int* (*w j*) ·$_v$ *M* ! *j*))
**unfolding** *lattice-coord-def*
**by** *simp*
**then have** *?recover*!*j* = *mat-M-inv* ∗$_v$((*of-int* (*of-int* (*w j*))) ·$_v$ *of-int-hom.vec-hom* (*M* ! *j*))
**using** *of-int-hom.vec-hom-smult*[*of of-int* (*w j*) *M* ! *j*]
**by** *metis*
**then have** *?recover*!*j* = (*of-int* (*of-int* (*w j*))) ·$_v$ (*mat-M-inv* ∗$_v$ *of-int-hom.vec-hom* (*M* ! *j*))
**using** *mult-mat-vec*[*of mat-M-inv n n of-int-hom.vec-hom* (*M* ! *j*) (*of-int* (*of-int* (*w j*)))]
*carrier-Ms*
*access-index-M-dim*[*of j*]
*carrier-vecI*[*of of-int-hom.vec-hom* (*M* ! *j*) *n*]
**by** (*simp add*: *small*)

```
  then have ?recover!j = (of-int (of-int (w j))) ·v (lattice-coord (of-int-hom.vec-hom
(M ! j)))
    unfolding lattice-coord-def
    by simp
  then have recover-unit:?recover!j = (of-int (of-int (w j))) ·v (unit-vec n j)
    using unit[of j]
          small
    by simp
  then have (?recover!j)$i=((of-int (of-int (w j))) ·v (unit-vec n j))$i
    by simp
  then have (?recover!j)$i = (of-int (of-int (w j))) * (unit-vec n j)$i
    by (simp add: assms(2))
  then have (?recover!j)$i = (of-int (of-int (w j))) * (if i=j then 1 else 0)
    using small assms(2)
    by simp
  moreover have (if i=j then 1 else 0) ∈ℤ
    by simp
  moreover have (of-int (of-int (w j)))∈ℤ
    by simp
  moreover have dim-vec (?recover!j) = n
  using recover-unit
        smult-closed[of (unit-vec n j) (of-int (of-int (w j)))]
        unit-vec-carrier[of n j]
  by force
  ultimately show (?recover!j)$i ∈ℤ ∧ dim-vec (?recover!j) = n
    by simp
 qed
 then have ∀ v∈set ?recover. dim-vec v = n
   by auto
 then have set ?recover⊆carrier-vec n
   using carrier-vecI
   by blast
 then have (gs.sumlist ?recover)$i = sum-list (map (λj. (?recover!j)$i) [0..<(length
?recover)])
   using sumlist-index-commute[of ?recover i] assms
   by blast
 moreover have length ?recover = n
   by auto
 ultimately have (gs.sumlist ?recover)$i = sum-list (map (λj. (?recover!j)$i)
[0..<n])
   by simp
 moreover have ⋀j. j<n ⟹ (map (λj. (?recover!j)$i) [0..<n])!j ∈ℤ
 proof−
   fix j::nat
   assume jsmall:j<n
   have (map (λj. (?recover!j)$i) [0..<n])!j =  (λj. (?recover!j)$i) j
     using List.nth-map[of j [0..<n] (λj. (?recover!j)$i)]
           jsmall
     by simp
```

```
    then have (map (λj. (?recover!j)$i) [0..<n])!j =(?recover!j)$i
      by simp
    then show (map (λj. (?recover!j)$i) [0..<n])!j∈ℤ
      using int-recover[of j] jsmall
      by simp
  qed
  ultimately have (gs.sumlist ?recover)$i∈ℤ
    using integral-sum[of n (λj. map lattice-coord
          (map of-int-hom.vec-hom (map (λi. of-int (w i) ·v M ! i) [0..<n])) !
         j $
         i)]
    by argo
  then show ?thesis
    using 1
    by simp
qed

lemma int-coord-for-rat:
  fixes i::nat
  assumes 0≤i
  assumes i<n
  fixes v::rat vec
  assumes v∈of-int-hom.vec-hom‘ L
  assumes dim-vec v = n
  shows (lattice-coord  v)$i∈ℤ
proof−
  let ?hom = of-int-hom.vec-hom
  obtain vint where v = ?hom vint∧ vint∈L using assms(3) by blast
  moreover then have (lattice-coord (?hom vint))$i∈ℤ using int-coord assms by
simp
  ultimately show ?thesis by simp
qed
```

# 6 Coord-Invariance

This section shows that the algorithm output matches true closest (or near-closest) vector in some trailing coordinates.

```
definition I where
  I =  (if ({i∈{0..<n}. ((sq-norm (Mt!i)::rat))≤4∗eps-closest}::nat set) ≠ {}
    then Max ({i∈{0..<n}. ((sq-norm (Mt!i)::rat))≤4∗eps-closest}::nat set) else
−1)

lemma I-geq:
  shows I≥−1
  unfolding I-def
  by simp

lemma I-leq:
```

**shows** $I<n$
**unfolding** *I-def*
**by** *force*

**lemma** *index-geq-I-big*:
**fixes** $i$::*nat*
**assumes** $i>I$
**assumes** $i<n$
**shows** $((sq\text{-}norm\ (Mt!i)::rat))>4*eps\text{-}closest$
**proof**(*rule ccontr*)
**assume** $\neg$ *?thesis*
**then have** $((sq\text{-}norm\ (Mt!i)::rat))\leq 4*eps\text{-}closest$ **by** *linarith*
**then have** *i-def*:$i\in(\{i\in\{0..<n\}.\ ((sq\text{-}norm\ (Mt!i)::rat))\leq 4*eps\text{-}closest\}::nat\ set)$
**using** *assms* **by** *fastforce*
**then have** $(\{i\in\{0..<n\}.\ ((sq\text{-}norm\ (Mt!i)::rat))\leq 4*eps\text{-}closest\}::nat\ set)\neq\{\}$ **by** *fast*
**moreover then have** $I= Max\ (\{i\in\{0..<n\}.\ ((sq\text{-}norm\ (Mt!i)::rat))\leq 4*eps\text{-}closest\}::nat\ set)$ **unfolding** *I-def* **by** *presburger*
**moreover have** *finite* $(\{i\in\{0..<n\}.\ ((sq\text{-}norm\ (Mt!i)::rat))\leq 4*eps\text{-}closest\}::nat\ set)$
**by** *simp*
**ultimately show** *False* **using** *assms i-def eq-Max-iff* **by** *auto*
**qed**

**lemma** *scalar-prod-gs-from-lattice-coord*:
**fixes** $i$::*nat*
**fixes** $v$::*rat vec*
**assumes** *dim-vec* $v = n$
**assumes** $i<n$
**shows** $v\cdot Mt!i=sum\text{-}list\ (map\ (\lambda k.\ (lattice\text{-}coord\ v)\$k\ *\ (((RAT\ M)!k)\cdot Mt!i))\ [i..<n])$
**proof**(−)
**let** *?lc* = *lattice-coord v*
**let** *?recover* = $((map\ (\lambda j.\ ?lc\$j\ \cdot_v\ (RAT\ M)!j)\ [0\ ..<\ n]))$
**let** *?gsv* = $Mt!i$
**have** $v$ = *gs.sumlist ?recover*
**using** *recover-from-lattice-coord*[*of v*] *assms*
**by** *blast*
**then have** *split-ip*: $v\ \cdot\ ?gsv = \ (gs.sumlist\ (map\ (\lambda j.\ ?lc\$j\ \cdot_v\ (RAT\ M)!j)\ [0\ ..<\ n]))\cdot\ ?gsv$
**by** *simp*
**have** $\bigwedge u.\ u\in set\ ?recover\Longrightarrow u\in carrier\text{-}vec\ n$
**proof**(−)
**fix** $u$::*rat vec*
**assume** *u-init*:$u\in$ *set ?recover*
**then have** *index-small*:*find-index ?recover u* < *length ?recover*
**by** (*meson find-index-leq-length*)
**then have** *carrier-v-ind-M*:$(RAT\ M)!(find\text{-}index\ ?recover\ u)\in carrier\text{-}vec\ n$

```
    using carrier-vecI[of (RAT M)!(find-index ?recover u) n]
        access-index-M-dim
    by (smt (z3) M-locale-1 gram-schmidt-fs-Rn.f-carrier length-map map-nth)
  then have u=?recover!(find-index ?recover u)
    using u-init
    by (simp add: find-index-in-set)
  then have u=(λj. ?lc$j ·v (RAT M)!j) (find-index ?recover u)
    using u-init
        List.nth-map[of find-index ?recover u [0..<n] (λj. ?lc$j ·v (RAT M)!j)]
        index-small
    by auto
  then have u = ?lc$(find-index ?recover u)·v (RAT M)!(find-index ?recover u)
    by simp
  then show u∈carrier-vec n
    using carrier-v-ind-M
          smult-carrier-vec[of ?lc$(find-index ?recover u) (RAT M)!(find-index
?recover u) n]
    by presburger
 qed
 then have result-sumlist-L:v • ?gsv =  sum-list (map (λw. w • ?gsv) ?recover)
  using split-ip
      gs.scalar-prod-left-sum-distrib[of ?recover ?gsv]
  by (metis (no-types, lifting) assms(2) carrier-dim-vec dim-vecs-in-Mt)
 let ?L=(map (λw. w • ?gsv) ?recover)
 have 2:⋀k. k<n⟹?L!k = ?lc$k ∗ ((RAT M)!k• ?gsv)
 proof(−)
  fix k::nat
  assume k-bound:k<n
  then have ?L!k= (λw. w • ?gsv) (?recover!k)
    by force
  then have ?L!k = ?recover!k • ?gsv
    by simp
  then have ?L!k = ((λj. (?lc$j ·v (RAT M)!j)) k) • ?gsv
    using List.nth-map[of k [0..<n] (λj. (?lc$j ·v (RAT M)!j))] k-bound
    by simp
  then have ?L!k = (?lc$k ·v(RAT M)!k)• ?gsv
    by simp
  then show ?L!k =?lc$k ∗ ((RAT M)!k• ?gsv)
    using smult-scalar-prod-distrib[of (RAT M)!k n ?gsv ?L!k]
        access-index-M-dim
        dim-vecs-in-Mt[of i]
        carrier-vecI[of ?gsv n]
        k-bound
        assms
    by force
 qed
 moreover have length ?L = n
  by fastforce
 ultimately have 1:?L = (map (λk. ?lc$k ∗ ((RAT M)!k• ?gsv)) [0..<n])
```

```
    by auto
  moreover then have filt:⋀k. k<i⟹ (λk. ?lc$k * ((RAT M)!k• ?gsv)) k =0
  proof(−)
    fix k::nat
    assume tri:k<i
    then have (?gsv •(RAT M)!k) = 0
      using gram-schmidt-fs-lin-indpt.gso-scalar-zero[of n (RAT M) i k]
            M-locale-2
            Mt-gso-connect[of i]
            assms(2)
            more-dim
      by presburger
    then have ((RAT M)!k)•?gsv = 0
      using comm-scalar-prod[of ((RAT M)!k) n ?gsv ]
            access-index-M-dim[of k]
            tri
            assms(2)
            dim-vecs-in-Mt[of i]
            carrier-vecI[of ?gsv] carrier-vecI[of ((RAT M)!k)]
      by fastforce
    then have ?lc$k * ((RAT M)!k• ?gsv) = 0
      by simp
    then show (λk. ?lc$k * ((RAT M)!k• ?gsv)) k =0
      by blast
  qed
  ultimately have sum-list ?L = sum-list (map (λk. ?lc$k * ((RAT M)!k• ?gsv))
(filter (λk. i≤k) [0..<n] ) )
    using sum-list-map-filter[of [0..<n] (λk. i≤k) (λk. ?lc$k * ((RAT M)!k• ?gsv))
]
    by (metis (no-types, lifting) le-eq-less-or-eq nat-neq-iff)
  moreover have (filter (λk. i≤k) [0..<n] ) = [i..<n]
    using assms(2) bot-nat-0.extremum filter-upt
    by presburger
  ultimately have sum-list ?L = sum-list (map (λk. ?lc$k * ((RAT M)!k• ?gsv))
[i..<n])
    by presburger
  then show ?thesis
    using result-sumlist-L
    by simp
qed

lemma correct-coord-help:
  fixes i::nat
  assumes i<(int n)−I
  assumes witness v (eps-closest)
  assumes 0<i
  shows (lattice-coord (s i))$(n−i)=(lattice-coord v)$(n−i)
        ∧ ( (s i) • Mt!(n−i) = v • Mt!(n−i) )
  using assms
```

**proof**(*induct i rule*: *less-induct*)
**case** (*less i*)
**let** *?lcs* = (*lattice-coord* (*s i*))
**let** *?lcIs* = λ*i*. *lattice-coord* (*s i*)$(*n*−*i*)
**let** *?lcv* = *lattice-coord v*
**let** *?gsv* = *Mt*!(*n*−(*i*))
**have** *leq*:(*int n*)−*I*≤*n*+*1*
**using** *I-geq*
**by** *simp*
**moreover have** *nonbase*:*0*<*i*
**using** *less* **by** *blast*
**then have** *1*:*i*≤*n*
**using** *leq less*
**by** *linarith*
**moreover have** *nms*:*n*−(*i*)<*n*
**using** *1 nonbase* **by** *linarith*
**ultimately have** *s-ip*:(*s* (*i*)) ∙ *?gsv* = *sum-list* (*map* (λ*j*. *?lcs*$*j* ∗((*RAT M*)!*j*∙ *?gsv*)) [*n*−(*i*)..<*n*])
**using** *scalar-prod-gs-from-lattice-coord*[*of s* (*i*) *n*−(*i*)]
*s-dim*[*of i*] **by** *force*
**have** *dim-v*:*dim-vec v* = *n*
**using** *assms*(*2*)
**unfolding** *witness-def*
**by** *blast*
**then have** *v-ip*:*v* ∙ *?gsv* = *sum-list* (*map* (λ*j*. *?lcv*$*j* ∗((*RAT M*)!*j*∙ *?gsv*)) [*n*−(*i*)..<*n*])
**unfolding** *witness-def*
**using** *scalar-prod-gs-from-lattice-coord*[*of v n*−*i*]
*nms assms*(*2*)
*carrier-vecI*[*of v n*]
**by** *satx*
**have** [*n*−*i*..<*n*]≠[] **using** *nms* **by** *auto*
**then have** *split-indices*:[*n*−(*i*)..<*n*] = (*n*−*i*) # [*n*−(*i*)+*1*..<*n*]
**by** (*simp add*: *upt-eq-Cons-conv*)
**then have** *split-s-list*:(*map* (λ*j*. *?lcs*$*j* ∗((*RAT M*)!*j*∙ *?gsv*)) [*n*−(*i*)..<*n*]) =
((λ*j*. *?lcs*$*j* ∗((*RAT M*)!*j*∙ *?gsv*)) (*n*−(*i*)))#(*map* (λ*j*. *?lcs*$*j* ∗((*RAT M*)!*j*∙ *?gsv*)) [*n*−(*i*)+*1*..<*n*])
**by** *simp*
**then have** *split-s-ip-pre*:(*s* (*i*)) ∙ *?gsv* = ((λ*j*. *?lcs*$*j* ∗((*RAT M*)!*j*∙ *?gsv*)) (*n*−(*i*)))
+ *sum-list* (*map* (λ*j*. *?lcs*$*j* ∗((*RAT M*)!*j*∙ *?gsv*)) [*n*−(*i*)+*1*..<*n*])
**using** *s-ip*
**by** *force*
**then have** *split-s-ip*: (*s* (*i*)) ∙ *?gsv* = ((λ*j*. *?lcs*$*j* ∗((*RAT M*)!*j*∙ *?gsv*)) (*n*−(*i*)))
+ *sum-list* (*map* (λ*j*. *?lcs*$*j* ∗((*RAT M*)!*j*∙ *?gsv*)) [*n*−*i*+*1*..<*n*])
**by** *presburger*
**have** *split-v-list*:(*map* (λ*j*. *?lcv*$*j* ∗((*RAT M*)!*j*∙ *?gsv*)) [*n*−(*i*)..<*n*]) =

((λj. ?lcv$j ∗((RAT M)!j· ?gsv)) (n−(i)))#(map (λj. ?lcv$j ∗((RAT M)!j· ?gsv)) [n−(i)+1..<n])
**using** *split-indices* **by** *simp*
**then have** *split-v-ip-pre*:*v* · *?gsv* = ((λj. *?lcv*$j ∗((RAT M)!j· *?gsv*)) (n−(i)))
+ *sum-list* (*map* (λj. *?lcv*$j ∗((RAT M)!j· *?gsv*)) [n−(i)+1..<n])
**using** *v-ip*
**by** *force*
**then have** *split-v-ip*:*v* · *?gsv* = ((λj. *?lcv*$j ∗((RAT M)!j· *?gsv*)) (n−(i)))
+ *sum-list* (*map* (λj. *?lcv*$j ∗((RAT M)!j· *?gsv*)) [n−i+1..<n])
**by** *presburger*
**have** *use-coord-inv*: (λj. *?lcs*$j ∗((RAT M)!j· *?gsv*)) k = (λj. *?lcv*$j ∗((RAT M)!j· *?gsv*)) k **if** *k-bound*: k<n ∧ k≥n−i+1 **for** k
**proof** −
**have** *nmssmall*:n−k<i
**using** *k-bound* **by** *linarith*
**then have** *arith*:(n−k)+(i − (n−k)) = i
**using** *k-bound 1* **by** *linarith*
**have** *2*:0<n−k
**using** *k-bound* **by** *linarith*
**moreover have** *3*:(n−k)+(i − (n−k))≤n
**using** *1 arith* **by** *linarith*
**moreover have** *4*:n−k≤n−k **by** *auto*
**ultimately have** *5*:*lattice-coord* (*s* (n−k + (i − (n−k)))) $ (n−(n−k)) = *lattice-coord* (*s* (n−k)) $ (n−(n−k))
**using** *coord-invariance*[*of* n−k n−k (i)−(n−k)] **by** *blast*
**also have** *cancel*:n−(n−k) =k
**using** *k-bound 2* **by** *auto*
**then have** *?lcs*$k = *?lcIs* (n−k)
**using** *arith 5* **by** *presburger*
**moreover have** *int* (n−k)<*int* n −*I*
**using** *assms nmssmall less* **by** *linarith*
**ultimately have** *?lcs*$k = *?lcv*$(n−(n−k))
**using** *less*(*1*)[*of* n−k] *nmssmall assms*(*2*) *2* **by** *argo*
**then have** *?lcs*$k = *?lcv*$k
**using** *cancel* **by** *presburger*
**then have** *?lcs*$k ∗((RAT M)!k· *?gsv*) = *?lcv*$k ∗((RAT M)!k· *?gsv*)
**by** *simp*
**then show** (λj. *?lcs*$j ∗((RAT M)!j· *?gsv*)) k = (λj. *?lcv*$j ∗((RAT M)!j· *?gsv*)) k
**by** *simp*
**qed**
**then have** (*map* (λj. *?lcs*$j ∗((RAT M)!j· *?gsv*)) [n−i+1..<n])
= (*map* (λj. *?lcv*$j ∗((RAT M)!j· *?gsv*)) [n−i+1..<n])
**by** *simp*
**then have** *sum-list* (*map* (λj. *?lcs*$j ∗((RAT M)!j· *?gsv*)) [n−i+1..<n])
= *sum-list* (*map* (λj. *?lcv*$j ∗((RAT M)!j· *?gsv*)) [n−i+1..<n])
**by** *presburger*
**then have** (*s i*) · *?gsv* =
((λj. *?lcs*$j ∗((RAT M)!j· *?gsv*)) (n−i)) +

```
        sum-list (map (λj. ?lcv$j *((RAT M)!j· ?gsv)) [n−i+1..<n])
    using split-s-ip by argo
  then have (s i) · ?gsv − v · ?gsv =
          ((λj. ?lcs$j *((RAT M)!j· ?gsv)) (n−i))−
          ((λj. ?lcv$j *((RAT M)!j· ?gsv)) (n−i))
    using split-v-ip by linarith
  then have (s i) · ?gsv − v · ?gsv = ((?lcs$(n−i) − ?lcv$(n−i)) * ((RAT
M)!(n−i)· ?gsv))
    by algebra
  then have case-2-from-case-1:(s i) · ?gsv − v · ?gsv = ((?lcs$(n−i) −
?lcv$(n−i)) * (sq-norm ?gsv))
    using one-diag[of n− i] 1 nms
    by fastforce
  then have abs ((s i) · ?gsv − v · ?gsv) = abs(?lcs$(n−i) − ?lcv$(n−i)) *
abs(sq-norm ?gsv)
    using abs-mult by auto
  then have a:abs ((s i) · ?gsv − v · ?gsv) = abs(?lcs$(n−i) − ?lcv$(n−i)) *
(sq-norm ?gsv)
    by (metis abs-of-nonneg sq-norm-vec-ge-0)
  have lattice-coord-equal:?lcs$(n−i) − ?lcv$(n−i)= 0
  proof(rule ccontr)
    assume ¬(?lcs$(n−i) − ?lcv$(n−i)= 0)
    then have contra:?lcs$(n−i) − ?lcv$(n−i) ≠ 0 by simp
    have ?lcs$(n−i) − ?lcv$(n−i) = (?lcs−?lcv)$(n−i)
      using index-minus-vec(1)[of n−i ?lcv ?lcs]
            dim-preserve-lattice-coord[of v]
            assms(2) nms
      unfolding witness-def by argo
    moreover have ?lcs−?lcv = lattice-coord((s i)−v)
      using mult-minus-distrib-mat-vec
      unfolding lattice-coord-def
      by (metis 1 carrier-Ms(2) carrier-vecI dim-v s-dim)
   ultimately have use-linear:?lcs$(n−i) − ?lcv$(n−i) = (lattice-coord((s i)−v))$(n−i)
      by presburger
    have (s i)−v∈ of-int-hom.vec-hom‘ L
      using subtract-coset-into-lattice[of s i v]
            coset-s[of i]
            1 assms(2)
      unfolding witness-def
      by linarith
    then have use-int-coord:(lattice-coord( ((s i)−v)) )$(n−i)∈ℤ
      using int-coord-for-rat[of n−i ((s i)−v)] 1 nms
      by (simp add: dim-v)
    then have abs((lattice-coord( ((s i)−v)) )$(n−i))>0
      using contra use-linear
      by linarith
    then have abs((lattice-coord( ((s i)−v)) )$(n−i))≥1
      using use-int-coord
      by (simp add: Ints-nonzero-abs-ge1 contra use-linear)
```

**then have** *abs(?lcs$(n−i) − ?lcv$(n−i))≥1*
**using** *use-linear* **by** *presburger*
**then have** *abs(?lcs$(n−i) − ?lcv$(n−i))∗(sq-norm ?gsv)≥sq-norm ?gsv*
**using** *sq-norm-vec-ge-0[of ?gsv] mult-left-mono[of 1 abs(?lcs$(n−i) − ?lcv$(n−i))*
*sq-norm ?gsv]* **by** *algebra*
**then have** *big1:abs ((s i) · ?gsv − v · ?gsv)≥sq-norm ?gsv*
**using** *a* **by** *argo*
**then have** *tri-ineq:abs(v · ?gsv)≥ abs(abs ((s i) · ?gsv − v · ?gsv) −abs((s i)*
*· ?gsv))*
**using** *cancel-ab-semigroup-add-class.diff-right-commute*
*cancel-comm-monoid-add-class.diff-cancel diff-zero* **by** *linarith*
**then have** *smallhalf:abs((s i) · ?gsv)≤(1/2)∗(sq-norm ?gsv)*
**using** *small-orth-coord[of i] nonbase 1*
**by** *fastforce*
**then have** *abs((s i) · ?gsv − v · ?gsv) −abs((s i) · ?gsv)≥ sq-norm ?gsv −*
*(1/2)∗(sq-norm ?gsv)*
**using** *big1* **by** *linarith*
**then have** *big2:abs((s i) · ?gsv − v · ?gsv) −abs((s i) · ?gsv)≥ (1/2)∗(sq-norm*
*?gsv)*
**by** *linarith*
**then have** *abs((s i) · ?gsv − v · ?gsv) −abs((s i) · ?gsv)≥0*
**using** *sq-norm-vec-ge-0[of ?gsv]* **by** *linarith*
**then have** *abs(abs ((s i) · ?gsv − v · ?gsv) −abs((s i) · ?gsv))*
*= abs((s i) · ?gsv − v · ?gsv) −abs((s i) · ?gsv)*
**by** *fastforce*
**then have** *abs(v · ?gsv)≥(1/2)∗(sq-norm ?gsv)*
**using** *big2*
**by** *linarith*
**moreover have** *(1/2)∗(sq-norm ?gsv)≥0*
**using** *sq-norm-vec-ge-0[of ?gsv]* **by** *simp*
**moreover have** *abs(v · ?gsv)≥0* **by** *simp*
**ultimately have** *abs(v · ?gsv)^2≥((1/2)∗(sq-norm ?gsv))^2*
**using** *nonneg-power-le* **by** *blast*
**moreover have** *(sq-norm v) ∗ (sq-norm ?gsv)≥abs(v · ?gsv)^2*
**using** *scalar-prod-Cauchy[of v n ?gsv]*
*carrier-vecI[of v n] assms(2)*
*carrier-vecI[of ?gsv] dim-vecs-in-Mt[of n−i] nms*
**unfolding** *witness-def*
**by** *fastforce*
**ultimately have** *sq-norm v ∗ sq-norm ?gsv ≥ ((1/2)∗(sq-norm ?gsv))^2*
**by** *order*
**then have** *sq-norm v ∗ sq-norm ?gsv ≥ (1/2)^2 ∗ (sq-norm ?gsv)^2*
**by** *(metis gs.nat-pow-distrib)*
**then have** *sq-norm v ∗ sq-norm ?gsv ≥ 1/4 ∗ (sq-norm ?gsv)^2*
**by** *(smt (z3) numeral-Bit0-eq-double one-power2 power2-eq-square times-divide-times-eq)*
**moreover have** *sq-norm ?gsv > 0*
**using** *gram-schmidt-fs-lin-indpt.sq-norm-pos[of n RAT M n−i]*
*M-locale-2 M-locale-1 gram-schmidt-fs-Rn.main-connect[of n (RAT M)]*
*nms* **by** *force*

```
    ultimately have big:sq-norm v ≥ 1/4 * sq-norm ?gsv
      by (simp add: power2-eq-square)
    have n−i>I
      using less by linarith
    then have big-again:sq-norm ?gsv > 4∗eps-closest
      using index-geq-I-big[of n−i] nms by simp
    then have sq-norm v> 1/4 ∗4∗eps-closest
      using big by fastforce
    then have sq-norm v > eps-closest by auto
    then show False
      using assms(2)
      unfolding witness-def
      by linarith
  qed
  then have piece1: lattice-coord (s i) $ (n − i) = lattice-coord v $ (n − i)
    using lattice-coord-equal by simp
  have (s i) · ?gsv − v · ?gsv = 0
    using lattice-coord-equal case-2-from-case-1
    by algebra
  then show ?case using piece1 by simp
qed

lemma correct-coord:
  fixes v::rat vec
  fixes k::nat
  assumes witness v eps-closest
  assumes I<k
  assumes k<n
  shows (s n) · Mt!(k) = v · Mt!(k)
proof −
  have (s n) · Mt!(k) = (s (n−k)) · Mt!(k)
    using coord-invariance[of n−k n−k k] assms
    by force
  moreover have (s (n−k)) · Mt!(k) = v · Mt!(k)
    using correct-coord-help[of n−k v] assms
    by simp
  ultimately show ?thesis by simp
qed
```

# 7 Bound on distance to target, with epsilon factor.

This section culminates in a bound (which includes the epsilon factor) on the output's distance to the target vector.

```
lemma sq-norm-from-Mt:
  fixes v::rat vec
  assumes v-carr:v∈carrier-vec n
  shows sq-norm v = sum-list (map (λi. (v•Mt!i)^2/(sq-norm (Mt!i))) [0..<n])
proof−
```

**let** *?Mt-inv-list* = *map* (λ*i*. (*1*/*sq-norm*(*Mt*!*i*))·$_v$ (*Mt*!*i*)) [*0*..<*n*]
**have** *nonsing*:*?Mt-inv-list*!*i* ∈ *carrier-vec n* **if** *i*:*0*≤*i*∧*i*<*n* **for** *i*
**proof**−
**have** *0*< *sq-norm*(*Mt*!*i*)
**using** *gram-schmidt-fs-lin-indpt.sq-norm-pos*[*of n RAT M i*]
*M-locale-1 gram-schmidt-fs-Rn.main-connect*[*of n* (*RAT M*)] *i*
**by** (*simp add*: *M-locale-2*)
**then have** *0*<*1*/*sq-norm*(*Mt*!*i*) **by** *fastforce*
**then have** (*1*/*sq-norm*(*Mt*!*i*))·$_v$ (*Mt*!*i*)∈*carrier-vec n*
**using** *carrier-vecI*[*of* (*Mt*!*i*)] *dim-vecs-in-Mt*[*of i*] *i* **by** *blast*
**moreover have** *?Mt-inv-list*!*i* = (*1*/*sq-norm*(*Mt*!*i*))·$_v$ (*Mt*!*i*)
**using** *i* **by** *simp*
**ultimately show** *?thesis* **by** *argo*
**qed**
**let** *?Mt-inv-mat* = *mat-of-rows n ?Mt-inv-list*
**have** *carrier-mat-inv*:*?Mt-inv-mat*∈*carrier-mat n n* **by** *fastforce*
**let** *?vMt* = *?Mt-inv-mat* ∗$_v$ *v*
**have** *?vMt*$*i* = ((*1*/*sq-norm*(*Mt*!*i*))·$_v$ (*Mt*!*i*))·*v* **if** *i*:*0*≤*i*∧*i*<*n* **for** *i*
**using** *i nonsing*[*of i*] **by** *auto*
**have** *dim-vMt*:*dim-vec ?vMt* = *n*
**using** *carrier-mat-inv v-carr* **by** *auto*
**let** *?Mt-mat* = *mat-of-cols n Mt*
**have** *l*:*length Mt* = *n*
**using** *gs.gram-schmidt-result*[*of RAT M Mt*] *basis dim-vecs-in-M*
**unfolding** *gs.lin-indpt-list-def*
**by** *fastforce*
**then have** *carrier-mat-Mt*:*?Mt-mat*∈*carrier-mat n n*
**using** *dim-vecs-in-Mt carrier-vecI* **by** *auto*
**then have** *to-sumlist*:*?Mt-mat*∗$_v$*?vMt* = *gs.sumlist* (*map* (λ*j*. *?vMt*$*j* ·$_v$ (*col ?Mt-mat j*)) [*0* ..< *n*])
**using** *mat-mul-to-sum-list*[*of ?vMt ?Mt-mat*] *dim-vMt*
**by** *fastforce*
**have** *?vMt*$*i* ·$_v$ (*col ?Mt-mat i*) = (*1*/*sq-norm*(*Mt*!*i*))∗ ((*Mt*!*i*)·*v*) ·$_v$ *Mt*!*i* **if** *i*:*0*≤*i*∧*i*<*n* **for** *i*
**using** *i l dim-vecs-in-Mt v-carr carrier-vecI* **by** *fastforce*
**then have** (*map* (λ*j*. *?vMt*$*j* ·$_v$ (*col ?Mt-mat j*)) [*0* ..< *n*])
= (*map* (λ*j*. (*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)·*v*) ·$_v$ *Mt*!*j*) [*0* ..< *n*])
**by** *simp*
**then have** *1*:*gs.sumlist* (*map* (λ*j*. *?vMt*$*j* ·$_v$ (*col ?Mt-mat j*)) [*0* ..< *n*])
=*gs.sumlist* (*map* (λ*j*. (*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)·*v*) ·$_v$ *Mt*!*j*) [*0* ..< *n*])
**by** *presburger*
**then have** *2*:*?Mt-mat*∗$_v$*?vMt* = *gs.sumlist* (*map* (λ*j*. (*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)·*v*) ·$_v$ *Mt*!*j*) [*0* ..< *n*])
**using** *to-sumlist* **by** *argo*
**have** *?Mt-mat* ∗$_v$ *?vMt* = (*?Mt-mat* ∗ *?Mt-inv-mat*)∗$_v$ *v*
**using** *carrier-mat-Mt carrier-mat-inv v-carr* **by** *auto*
**have** (*?Mt-inv-mat*∗*?Mt-mat*)$$(*i*,*j*) = (*1*$_m$ *n*)$$(*i*,*j*)
**if** *sensible-indices*:*0*≤*i* ∧ *i*<*n* ∧ *0*≤*j* ∧ *j*<*n* **for** *i j*

**proof**−
**have** (*?Mt-inv-mat*∗*?Mt-mat*)$$(*i*,*j*) = (*row ?Mt-inv-mat i*)·(*col ?Mt-mat j*)
**using** *sensible-indices carrier-mat-Mt carrier-mat-inv* **by** *auto*
**then have** (*?Mt-inv-mat*∗*?Mt-mat*)$$(*i*,*j*) = *?Mt-inv-list*!*i*·*Mt*!*j*
**using** *sensible-indices carrier-mat-Mt carrier-mat-inv nonsing*
**by** *auto*
**then have** (*?Mt-inv-mat*∗*?Mt-mat*)$$(*i*,*j*) = ((*1* / *sq-norm*(*Mt*!*i*))·$_v$ (*Mt*!*i*))·*Mt*!*j*
**using** *sensible-indices* **by** *simp*
**then have** (*?Mt-inv-mat*∗*?Mt-mat*)$$(*i*,*j*) = (*1* / *sq-norm*(*Mt*!*i*)) ∗((*Mt*!*i*)·(*Mt*!*j*))
**using** *dim-vecs-in-Mt*[*of i*] *dim-vecs-in-Mt*[*of j*] *sensible-indices* **by** *auto*
**moreover have** (*1* / *sq-norm*(*Mt*!*i*)) ∗((*Mt*!*i*)·(*Mt*!*j*)) = (*if i*=*j then 1 else 0*)
**proof**(*cases i*=*j*)
**case** *diag*:*True*
**have** *nonzero*:*0*< *sq-norm*(*Mt*!*i*)
**using** *gram-schmidt-fs-lin-indpt.sq-norm-pos*[*of n RAT M i*]
*M-locale-1 gram-schmidt-fs-Rn.main-connect*[*of n* (*RAT M*)] *sensible-indices*
**by** (*simp add*: *M-locale-2*)
**have** (*1* / *sq-norm*(*Mt*!*i*)) ∗((*Mt*!*i*)·(*Mt*!*j*)) = (*1* / *sq-norm*(*Mt*!*i*)) ∗ *sq-norm*(*Mt*!*i*)
**using** *sensible-indices diag sq-norm-vec-as-cscalar-prod*[*of Mt*!*i*] **by** *auto*
**then have** (*1* / *sq-norm*(*Mt*!*i*)) ∗((*Mt*!*i*)·(*Mt*!*j*)) = *1*
**using** *nonzero* **by** *auto*
**then show** *?thesis* **using** *diag* **by** *argo*
**next**
**case** *off*:*False*
**have** *nonzero*:*0*< *sq-norm*(*Mt*!*i*)
**using** *gram-schmidt-fs-lin-indpt.sq-norm-pos*[*of n RAT M i*]
*M-locale-1 gram-schmidt-fs-Rn.main-connect*[*of n* (*RAT M*)] *sensible-indices*
**by** (*simp add*: *M-locale-2*)
**then have** *0*<*1* / *sq-norm*(*Mt*!*i*) **by** *simp*
**moreover have** ((*Mt*!*i*)·(*Mt*!*j*)) = *0*
**using** *gram-schmidt-fs-lin-indpt.orthogonal*[*of n* (*RAT*) *M i j*] *off sensible-indices*
*M-locale-1 M-locale-2 gram-schmidt-fs-Rn.main-connect*
**by** *force*
**ultimately show** *?thesis* **using** *off* **by** *algebra*
**qed**
**moreover then have** (*1* / *sq-norm*(*Mt*!*i*)) ∗((*Mt*!*i*)·(*Mt*!*j*)) = (*1*$_m$ *n*)$$(*i*,*j*)
**using** *sensible-indices* **unfolding** *one-mat-def* **by** *simp*
**ultimately show** *?thesis* **by** *presburger*
**qed**
**then have** *inv-Mt*:(*?Mt-inv-mat*∗*?Mt-mat*) = *1*$_m$ *n*
**using** *carrier-mat-inv carrier-mat-Mt*
**by** *fastforce*
**then have** *?Mt-mat* ∗ *?Mt-inv-mat* = *1*$_m$ *n*
**using** *mat-mult-left-right-inverse*[*of ?Mt-inv-mat n ?Mt-mat*] *carrier-mat-inv carrier-mat-Mt*
**by** *argo*
**then have** *3*:(*?Mt-mat* ∗ *?Mt-inv-mat*)∗$_v$ *v* = *v*
**using** *v-carr* **by** *simp*

**then have** *4*:$v$ = *gs.sumlist* (*map* (λ*j*. (*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)⋅*v*) ⋅$_v$ *Mt*!*j*) [*0* ..< *n*])
**using** *v-carr carrier-mat-inv carrier-mat-Mt 1 2* **by** *auto*
**have** (*map* (λ*j*. (*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)⋅*v*) ⋅$_v$ *Mt*!*j*) [*0* ..< *n*])
= (*map* (λ*j*. (*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)⋅*v*) ⋅$_v$ *gs.gso j*) [*0* ..< *n*])
**using** *M-locale-1 gram-schmidt-fs-Rn.main-connect*[*of n RAT M*]
**by** *auto*
**then have** *gs.sumlist* (*map* (λ*j*. (*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)⋅*v*) ⋅$_v$ *Mt*!*j*) [*0* ..< *n*])
= *gs.sumlist* (*map* (λ*j*. (*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)⋅*v*) ⋅$_v$ *gs.gso j*) [*0* ..< *n*])
**by** *argo*
**then have** *v* = *gs.sumlist* (*map* (λ*j*. (*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)⋅*v*) ⋅$_v$ *gs.gso j*) [*0* ..< *n*])
**using** *4* **by** *argo*
**then have** *v*⋅*v* = *gs.sumlist* (*map* (λ*j*. (*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)⋅*v*) ⋅$_v$ *gs.gso j*) [*0* ..< *n*])⋅
*gs.sumlist* (*map* (λ*j*. (*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)⋅*v*) ⋅$_v$ *gs.gso j*) [*0* ..< *n*])
**by** *simp*
**then have** *a*:*v*⋅*v* =
*sum-list*(*map* (λ*j*. (*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)⋅*v*)∗(*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)⋅*v*)∗(*gs.gso j* ⋅ *gs.gso j*)) [*0*..<*n*])
**using** *gram-schmidt-fs-lin-indpt.scalar-prod-lincomb-gso*[
*of n RAT M n* (λ*j*. (*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)⋅*v*)) (λ*j*. (*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)⋅*v*))]
*M-locale-2*
*M-locale-1 gram-schmidt-fs-Rn.main-connect*[*of n RAT M*] **by** *force*
**have** (*map* (λ*j*. (*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)⋅*v*)∗(*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)⋅*v*)∗(*gs.gso j* ⋅ *gs.gso j*)) [*0*..<*n*])
= (*map* (λ*j*. (*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)⋅*v*)∗(*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)⋅*v*)∗(*Mt*!*j* ⋅ *Mt*!*j*)) [*0*..<*n*])
**using** *M-locale-1 gram-schmidt-fs-Rn.main-connect*[*of n RAT M*]
**by** *auto*
**then have** *b*:*sum-list* (*map* (λ*j*. (*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)⋅*v*)∗(*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)⋅*v*)∗(*gs.gso j* ⋅ *gs.gso j*)) [*0*..<*n*])
=*sum-list* (*map* (λ*j*. (*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)⋅*v*)∗(*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)⋅*v*)∗(*Mt*!*j* ⋅ *Mt*!*j*)) [*0*..<*n*])
**by** *argo*
**have** (*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)⋅*v*)∗(*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)⋅*v*)∗(*Mt*!*j* ⋅ *Mt*!*j*) =
(*v*⋅(*Mt*!*j*))^2/(*sq-norm* (*Mt*!*j*)) **if** *sensible-indices*:*0*≤*j*∧*j*<*n* **for** *j*
**proof**−
**have** *nonzero*:*0*< *sq-norm*(*Mt*!*j*)
**using** *gram-schmidt-fs-lin-indpt.sq-norm-pos*[*of n RAT M j*]
*M-locale-1 gram-schmidt-fs-Rn.main-connect*[*of n* (*RAT M*)] *sensible-indices*
**by** (*simp add*: *M-locale-2*)
**moreover have** (*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)⋅*v*)∗(*1*/*sq-norm*(*Mt*!*j*))∗ ((*Mt*!*j*)⋅*v*)∗(*Mt*!*j*

$\cdot$ $Mt!j$)
= $(1/sq\text{-}norm(Mt!j))*((Mt!j)\cdot v)*(1/sq\text{-}norm(Mt!j))*((Mt!j)\cdot v)*sq\text{-}norm$
$(Mt!j)$
**using** *sq-norm-vec-as-cscalar-prod*[*of* $Mt!j$] **by** *force*
**moreover have** $(1/sq\text{-}norm(Mt!j))*((Mt!j)\cdot v)*(1/sq\text{-}norm(Mt!j))*((Mt!j)\cdot v)*$
$sq\text{-}norm\ (Mt!j)$
$= ((Mt!j)\cdot v)\hat{}2 * (1/sq\text{-}norm(Mt!j))\hat{}2 *sq\text{-}norm\ (Mt!j)$
**by** (*simp add*: *power2-eq-square*)
**moreover have** $((Mt!j)\cdot v)\hat{}2 * (1/sq\text{-}norm(Mt!j))\hat{}2 *sq\text{-}norm\ (Mt!j) =$
$((Mt!j)\cdot v)\hat{}2/(sq\text{-}norm(Mt!j))$
**using** *nonzero*
**by** (*simp add*: *divide-divide-eq-left' power2-eq-square*)
**moreover have** $(Mt!j)\cdot v = v\cdot(Mt!j)$ **using** *v-carr dim-vecs-in-Mt sensible-indices*
**by** (*metis carrier-vecI comm-scalar-prod*)
**ultimately show** *?thesis* **by** *argo*
**qed**
**then have** $(map\ (\lambda j.\ (1/sq\text{-}norm(Mt!j))*((Mt!j)\cdot v)*(1/sq\text{-}norm(Mt!j))*((Mt!j)\cdot v)*(Mt!j$
$\cdot\ Mt!j))\ [0..<n])$
$= (map\ (\lambda j.\ (v\cdot(Mt!j))\hat{}2/(sq\text{-}norm(Mt!j)))\ [0..<n])$ **by** *force*
**then have** *c*:*sum-list* $(map\ (\lambda j.\ (1/sq\text{-}norm(Mt!j))*((Mt!j)\cdot v)*(1/sq\text{-}norm(Mt!j))*$
$((Mt!j)\cdot v)*(Mt!j\ \cdot\ Mt!j))\ [0..<n])$
$= sum\text{-}list\ (map\ (\lambda j.\ (v\cdot(Mt!j))\hat{}2/(sq\text{-}norm(Mt!j)))\ [0..<n])$ **by** *argo*
**then have** $v\cdot v = sum\text{-}list\ (map\ (\lambda j.\ (v\cdot(Mt!j))\hat{}2/(sq\text{-}norm(Mt!j)))\ [0..<n])$
**using** *a b c* **by** *argo*
**moreover have** $v\cdot v = v\cdot cv$ **by** *force*
**ultimately show** *?thesis* **using** *sq-norm-vec-as-cscalar-prod*[*of v*] *v-carr* **by** *argo*
**qed**

**lemma** *basis-decay*:
**fixes** *i*::*nat*
**fixes** *j*::*nat*
**assumes** $i<n$
**assumes** $i+j<n$
**shows** $sq\text{-}norm\ (Mt!i)\leq \alpha\hat{}j*sq\text{-}norm(Mt!(i+j))$
**using** *assms*
**proof**(*induct j*)
**case** *0*
**have** $\alpha\hat{}0 = 1$ **by** *simp*
**moreover have** $sq\text{-}norm\ (Mt!i) = sq\text{-}norm(Mt!(i+0))$ **by** *simp*
**moreover have** $0\leq sq\text{-}norm(Mt!i)$
**using** *gram-schmidt-fs-lin-indpt.sq-norm-pos*[*of n RAT M i*]
*M-locale-2 M-locale-1 gram-schmidt-fs-Rn.main-connect*[*of n* (*RAT M*)]
*assms* **by** *force*
**moreover have** $(0::rat)\leq(1::rat)$ **by** *force*
**ultimately show** *?case* **by** *simp*
**next**
**case** (*Suc j*)
**have** $(1::rat)\leq\alpha$ **using** *alpha* **by** *fastforce*
**moreover have** $n\geq 0$ **by** *simp*

**ultimately have** $(1::rat) \leq \alpha \hat{} j$ **by** *simp*
**moreover have** *sq-norm* $(Mt!(i+j)) \leq \alpha * (sq\text{-}norm\ (Mt!(i+Suc\ j)))$
**using** *reduced M-locale-1 gram-schmidt-fs-Rn.main-connect*[*of n* $(RAT\ M)$] *Suc.prems*
**unfolding** *gs.reduced-def gs.weakly-reduced-def*
**by** *force*
**moreover have** $0 \leq$ *sq-norm* $(Mt!(i+j))$
**using** *gram-schmidt-fs-lin-indpt.sq-norm-pos*[*of n RAT M i+j*]
*M-locale-2 M-locale-1 gram-schmidt-fs-Rn.main-connect*[*of n* $(RAT\ M)$]
*Suc.prems* **by** *force*
**ultimately have** $\alpha \hat{} j * sq\text{-}norm\ (Mt!(i+j)) \leq \alpha \hat{} j * \alpha * (sq\text{-}norm\ (Mt!(i+Suc\ j)))$
**by** *simp*
**moreover have** $sq\text{-}norm(Mt!i) \leq \alpha \hat{} j *$ *sq-norm* $(Mt!(i+j))$
**using** *Suc* **by** *linarith*
**ultimately have** $sq\text{-}norm(Mt!i) \leq \alpha \hat{} j * \alpha * (sq\text{-}norm\ (Mt!(i+Suc\ j)))$ **by** *order*
**moreover have** $\alpha \hat{} j * \alpha = \alpha \hat{} (Suc\ j)$ **by** *simp*
**ultimately show** *?case* **by** *argo*
**qed**

**lemma** *basis-decay-cor*:
**fixes** $i::nat$
**fixes** $j::nat$
**assumes** $i<n$
**assumes** $j<n$
**assumes** $i \leq j$
**shows** *sq-norm* $(Mt!i) \leq \alpha \hat{} n * sq\text{-}norm(Mt!j)$
**proof**−
**have** *1*:*sq-norm* $(Mt!i) \leq \alpha \hat{} (j-i) * sq\text{-}norm(Mt!j)$
**using** *basis-decay*[*of i j−i*] *assms*
**by** *simp*
**have** $\alpha \hat{} (j-i) \leq \alpha \hat{} n$ **using** *assms* **using** *alpha* **by** *force*
**then have** $\alpha \hat{} (j-i) * sq\text{-}norm(Mt!j) \leq \alpha \hat{} n * sq\text{-}norm(Mt!j)$
**using** *mult-right-mono* **by** *blast*
**then show** *?thesis* **using** *1* **by** *order*
**qed**

**theorem** *babai-correct*:
**shows** *real-of-rat* $((sq\text{-}norm\ (s\ n))::rat) \leq$ $(real\text{-}of\text{-}rat\ ((rat\text{-}of\text{-}int\ n) * \alpha \hat{} n) * epsilon * closest\text{-}distance\text{-}sq) \wedge s\ n \in coset$
**proof**−
**let** *?s* $= s\ n$
**let** *?component* $= (\lambda i.\ (?s \cdot Mt!i) \hat{} 2 / (sq\text{-}norm\ (Mt!i)))$
**obtain** $v$ **where** *wit-v*:*witness v* $(eps\text{-}closest)$
**using** *witness-exists* **by** *force*
**have** *split-norm*:*sq-norm ?s* $=$ *sum-list* $(map\ ?component\ [0..<n])$
**using** *s-dim*[*of n*] *sq-norm-from-Mt*[*of ?s*] **by** *fast*
**have** $I+1 \in \mathbb{N}$ **using** *I-geq*
**using** *Nats-0 Nats-1 Nats-add R.add.l-inv-ex R.add.r-inv-ex add-diff-cancel-right'*

*cring-simprules(21) rangeI range-abs-Nats verit-la-disequality verit-minus-simplify(3)*

*zabs-def zle-add1-eq-le* **by** *auto*
**then obtain** *Inat* **where** *Inat-def*:*int Inat* = *I*+*1*
**using** *Nats-cases* **by** *metis*
**then have** *Inat-small*:*Inat*≤*n* **using** *I-leq* **by** *fastforce*
**then have** [*0*..<*n*] = [*0*..<*Inat*] @ [*Inat*..<*n*]
**by** (*metis bot-nat-0.extremum-uniqueI le-Suc-ex nat-le-linear upt-add-eq-append*)
**then have** *split-norm-sum*:*sq-norm ?s* = *sum-list* (*map ?component* [*0*..<*Inat*])
+ *sum-list* (*map ?component* [*Inat*..<*n*])
**using** *split-norm* **by** *force*

**have** *?component i* ≤ *eps-closest* **if** *i*:*Inat*≤*i*∧*i*<*n* **for** *i*
**proof**−
**have** *ge0*:*sq-norm* (*Mt*!*i*) > *0*
**using** *gram-schmidt-fs-lin-indpt.sq-norm-pos*[*of n RAT M i*]
*M-locale-2 M-locale-1 gram-schmidt-fs-Rn.main-connect*[*of n* (*RAT M*)]
*i* **by** *force*
**then have** *?component i* = (*v*• *Mt*!*i*)^*2* / (*sq-norm* (*Mt*!*i*))
**using** *ge0 correct-coord*[*of v i*] *wit-v Inat-def i*
**by** *auto*
**also have** (*v*•*Mt*!*i*)^*2*≤ (*sq-norm v*)∗*sq-norm* (*Mt*!*i*)
**using** *scalar-prod-Cauchy*[*of v n Mt*!*i*]
*dim-vecs-in-Mt*[*of i*] *carrier-vecI*[*of v*] *carrier-vecI*[*of Mt*!*i*] *wit-v*
*i*
**unfolding** *witness-def*
**by** *algebra*
**also have** *sq-norm v* ≤ *eps-closest*
**using** *wit-v* **unfolding** *witness-def* **by** *fast*
**finally show** *?thesis* **using** *ge0*
**by** (*simp add*: *divide-right-mono*)
**qed**
**then have** ⋀*x*. *x*∈*set* [*Inat*..<*n*] ⟹ *?component x* ≤ (λ*i*. *eps-closest*) *x* **by** *simp*
**then have** *sum-list* (*map ?component* [*Inat*..<*n*])≤ *sum-list* (*map* (λ*i*. *eps-closest*)
[*Inat*..<*n*])
**using** *sum-list-mono*[*of* [*Inat*..<*n*] *?component* (λ*i*. *eps-closest*)] **by** *argo*
**then have** *right-sum*:*sum-list* (*map ?component* [*Inat*..<*n*])≤(*rat-of-nat* (*n*−*Inat*))∗*eps-closest*
**using** *sum-list-triv*[*of eps-closest* [*Inat*..<*n*] ] **by** *force*
**have** (*1*::*rat*) ≤α **using** *alpha* **by** *fastforce*
**moreover have** *n*≥*0* **by** *simp*
**ultimately have** (*1*::*rat*)≤α^*n* **by** *simp*
**moreover have** *0*≤(*rat-of-nat* (*n*−*Inat*))∗*eps-closest*
**proof**−
**have** *0*≤(*rat-of-nat* (*n*−*Inat*)) **using** *Inat-small* **by** *fast*
**moreover have** *0*≤*eps-closest*
**proof**(*cases closest-distance-sq* = *0*)
**case** *t*:*True*

**then show** *?thesis* **using** *eps-closest-lemma closest-distance-sq-pos* **unfolding** *close-condition-def*
**by** *auto*
**next**
**case** *f*:*False*
**then show** *?thesis* **using** *eps-closest-lemma closest-distance-sq-pos* **unfolding** *close-condition-def*
**by** (*smt* (*verit*, *del-insts*) *zero-le-of-rat-iff*)
**qed**
**ultimately show** *?thesis* **by** *blast*
**qed**
**ultimately have** (*rat-of-nat* (*n*−*Inat*))∗*eps-closest* ≤ (*rat-of-nat* (*n*−*Inat*))∗*eps-closest* ∗ *α*^*n*
**using** *mult-left-mono*[*of 1 α*^*n* (*rat-of-nat* (*n*−*Inat*))∗*eps-closest*] **by** *linarith*
**then have** *sum-list* (*map ?component* [*Inat*..<*n*])≤(*rat-of-nat* (*n*−*Inat*))∗*eps-closest*∗*α*^*n* **using** *right-sum* **by** *order*
**then have** *right-sum-alpha*:*sum-list* (*map ?component* [*Inat*..<*n*])≤(*rat-of-nat* (*n*−*Inat*))∗*α*^*n*∗*eps-closest*
**by** *algebra*
**have** *sum-list* (*map ?component* [0..<*Inat*]) + *sum-list* (*map ?component* [*Inat*..<*n*])≤ (*rat-of-int n*)∗*α*^*n*∗*eps-closest*
**proof**(*cases Inat* = 0)
**case** *Inat*:*True*
**then have** *sum-list* (*map ?component* [0..<*Inat*]) = 0 **by** *auto*
**then have** *sum-list* (*map ?component* [0..<*Inat*]) + *sum-list* (*map ?component* [*Inat*..<*n*])≤(*rat-of-int* (*n*−*Inat*))∗*α*^*n* ∗ *eps-closest*
**using** *right-sum-alpha* **by** *simp*
**also have** *n*−*Inat* = *n* **using** *Inat* **by** *simp*
**finally show** *?thesis* **by** *linarith*
**next**
**case** *False*
**then have** *non-zero*:*Inat*>0 **by** *blast*
**then have** *I-not-min*:*I*≥0 **using** *Inat-def* **by** *simp*
**then have** *non-empty*:*I* = *Max* ({*i*∈{0..<*n*}. ((*sq-norm* (*Mt*!*i*)::*rat*))≤4∗*eps-closest*}::*nat set*)
**unfolding** *I-def* **by** *presburger*
**then have** *max*:*Inat*−1= *Max*({*i*∈{0..<*n*}. ((*sq-norm* (*Mt*!*i*)::*rat*))≤4∗*eps-closest*}::*nat set*)
**using** *Inat-def* **by** *linarith*
**then have** *Inat* −1 ∈ ({*i*∈{0..<*n*}. ((*sq-norm* (*Mt*!*i*)::*rat*))≤4∗*eps-closest*}::*nat set*)
**proof**−
**have** *finite* ({*i*∈{0..<*n*}. ((*sq-norm* (*Mt*!*i*)::*rat*))≤4∗*eps-closest*}::*nat set*)
**by** *simp*
**moreover have** ({*i*∈{0..<*n*}. ((*sq-norm* (*Mt*!*i*)::*rat*))≤4∗*eps-closest*}::*nat set*)≠{}
**using** *I-not-min* **unfolding** *I-def* **by** *presburger*
**ultimately show** *Inat* −1 ∈ ({*i*∈{0..<*n*}. ((*sq-norm* (*Mt*!*i*)::*rat*))≤4∗*eps-closest*}::*nat set*)

**using** *max eq-Max-iff* **by** *blast*
**qed**
**then have** *2*:(*sq-norm* (*Mt*!(*Inat*−*1*))::*rat*)≤*4*∗*eps-closest* **by** *blast*
**have** (*1*::*rat*) ≤α **using** *alpha* **by** *fastforce*
**moreover have** *n*≥*0* **by** *simp*
**ultimately have** (*1*::*rat*)≤α^*n* **by** *simp*
**then have** ((*1*/*4*)::*rat*)≤*1*/*4* ∗ α^*n* **by** *auto*
**then have** (*0*::*rat*)<*1*/*4*∗α^*n* **by** *linarith*
**moreover have** *0*<(*sq-norm* (*Mt*!(*Inat*−*1*))::*rat*)
**using** *gram-schmidt-fs-lin-indpt.sq-norm-pos*[*of n RAT M Inat*−*1*]
*M-locale-2 M-locale-1 gram-schmidt-fs-Rn.main-connect*[*of n* (*RAT M*)]
*non-zero Inat-small* **by** *force*
**ultimately have** *bound*:*1*/*4* ∗α^*n*∗ (*sq-norm* (*Mt*!(*Inat*−*1*)))≤ ((*1*/*4* ∗ α^*n*)∗ *4*∗*eps-closest*)
**using** *2* **by** *auto*
**have** *?component i* ≤ α^*n* ∗*eps-closest* **if** *list1*:*i*<*Inat* **for** *i*
**proof**−
**have** *1*:*0*<*n*−*i* **using** *list1 Inat-small* **by** *simp*
**then have** *?s*•*Mt*!*i* = (*s* (*n*−*i*))•*Mt*!*i*
**using** *coord-invariance*[*of n*−*i n*−*i i*] **by** *fastforce*
**then have** *abs*(*?s*•*Mt*!*i*)≤ (*1*/*2*)∗(*sq-norm* (*Mt*!*i*))
**using** *small-orth-coord*[*of n*−*i*] *1* **by** *force*
**then have** (*?s*•*Mt*!*i*)^*2* ≤ ((*1*/*2*)∗(*sq-norm* (*Mt*!*i*)))^*2*
**by** (*meson abs-ge-self abs-le-square-iff ge-trans*)
**moreover have** *ge0*:*sq-norm* (*Mt*!*i*) > *0*
**using** *gram-schmidt-fs-lin-indpt.sq-norm-pos*[*of n RAT M i*]
*M-locale-2 M-locale-1 gram-schmidt-fs-Rn.main-connect*[*of n* (*RAT M*)]
*list1 Inat-small* **by** *force*
**ultimately have** *?component i* ≤((*1*/*2*)∗(*sq-norm* (*Mt*!*i*)))^*2* / (*sq-norm* (*Mt*!*i*))
**using** *divide-right-mono* **by** *auto*
**also have** ((*1*/*2*)∗(*sq-norm* (*Mt*!*i*)))^*2*/ (*sq-norm* (*Mt*!*i*)) = *1*/*4* ∗ (*sq-norm* (*Mt*!*i*))^*2*/ (*sq-norm* (*Mt*!*i*))
**by** (*metis* (*no-types*, *lifting*) *gs.cring-simprules*(*12*) *numeral-Bit0-eq-double power2-eq-square times-divide-eq-left times-divide-times-eq*)
**also have** *1*/*4* ∗ (*sq-norm* (*Mt*!*i*))^*2*/ (*sq-norm* (*Mt*!*i*)) = *1*/*4* ∗ (*sq-norm* (*Mt*!*i*))
**using** *ge0* **by** (*simp add*: *power2-eq-square*)
**also have** *1*/*4*∗*sq-norm* (*Mt*!*i*) ≤ *1*/*4*∗α^*n* ∗ (*sq-norm* (*Mt*!(*Inat*−*1*)))
**using** *basis-decay-cor*[*of i Inat*−*1*] *list1 Inat-small mult-left-mono*[
*of sq-norm* (*Mt*!*i*) α^*n* ∗ (*sq-norm* (*Mt*!(*Inat*−*1*))) *1*/*4*]
**by** *linarith*
**finally have** *?component i* ≤ *1*/*4* ∗ α^*n* ∗ *4* ∗*eps-closest*
**using** *bound* **by** *linarith*
**also have** *1*/*4* ∗ α^*n* ∗ *4* ∗ *eps-closest*=α^*n* ∗ *eps-closest* **by** *force*
**finally show** *?thesis* **by** *blast*
**qed**
**then have** *sum-list* (*map ?component* [*0*..<*Inat*])≤ *sum-list* (*map* (λ*i*. α^*n* ∗ *eps-closest*)[*0*..<*Inat*])

**using** *sum-list-mono*[*of* [*0..<Inat*] *?component* (λ*i*. α^*n* ∗ *eps-closest*)] **by** *fastforce*
**then have** *sum-list* (*map ?component* [*0..<Inat*])≤ (*rat-of-int Inat*)∗α^*n* ∗ *eps-closest*
**using** *sum-list-triv*[*of* α^*n* ∗ *eps-closest* [*0..<Inat*]] **by** *auto*
**then have** (*sum-list* (*map ?component* [*0..<Inat*])) + *sum-list* (*map ?component* [*Inat..<n*])
≤ (*rat-of-int Inat*)∗α^*n* ∗ *eps-closest*+(*rat-of-int* (*n*−*Inat*))∗α^*n* ∗ *eps-closest*
**using** *right-sum-alpha* **by** *linarith*
**then have** (*sum-list* (*map ?component* [*0..<Inat*])) + *sum-list* (*map ?component* [*Inat..<n*])
≤ ((*rat-of-int Inat*)+(*rat-of-int* (*n*−*Inat*)))∗α^*n* ∗ *eps-closest*
**using** *gs.cring-simprules*(*13*) **by** *auto*
**then show** *?thesis*
**by** (*metis* (*no-types*, *lifting*) *Inat-small add-diff-inverse-nat diff-is-0-eq′ less-nat-zero-code*
*of-int-of-nat-eq of-nat-add zero-less-diff*)
**qed**
**then have** *sq-norm ?s* ≤ (*rat-of-int n*)∗α^*n* ∗ *eps-closest*
**using** *split-norm-sum* **by** *argo*
**then have** *real-of-rat* (*sq-norm ?s*) ≤ *real-of-rat* ((*rat-of-int n*)∗α^*n* ∗ *eps-closest*)
**by** (*simp add*: *of-rat-less-eq*)
**then have** *real-of-rat* (*sq-norm ?s*) ≤ *real-of-rat* ((*rat-of-int n*)∗α^*n*) ∗ (*real-of-rat eps-closest*)
**using** *of-rat-mult* **by** *metis*
**moreover have** *real-of-rat eps-closest* ≤ *epsilon* ∗ *closest-distance-sq*
**using** *eps-closest-lemma closest-distance-sq-pos* **unfolding** *close-condition-def* **by** *blast*
**ultimately show** *?thesis*
**using** *coset-s alpha*
**using** *mult-left-mono*[*of real-of-rat eps-closest epsilon* ∗ *closest-distance-sq real-of-rat* (*rat-of-int* (*int n*) ∗ α ^ *n*)]
**by** *simp*
**qed**

**end**

In this section we remove the arbitrary constant epsilon and clean up the assumptions and results.

**locale** *babai-with-assms* = *babai* +
**fixes** *mat-M* :: *rat mat*
**assumes** *basis*: *lin-indep M*
**defines** *mat-M* ≡ *mat-of-cols n* (*RAT M*)
**assumes** *reduced*:*weakly-reduced M n*
**assumes** *non-trivial*:*0<n*
**assumes** *alpha*:α ≥ *4*/*3*
**begin**

**sublocale** *vec-space TYPE*(*rat*) *n* **.**

**definition** *mat-M-inv* ≡
(*if* (*invertible-mat mat-M*) *then SOME B*. (*inverts-mat B mat-M*) ∧ (*inverts-mat mat-M B*) *else* (*0*$_m$ *n n*))

**lemma** *carrier-M*:
**shows** *mat-M* ∈ *carrier-mat n n*
**using** *basis* **unfolding** *gs.lin-indpt-list-def mat-M-def* **by** *auto*

**lemma** *mat-M-inv-is-inv*:
**shows** *invertible-mat mat-M*
**proof**−
**have** *vec-space.rank n mat-M* = *n*
**using** *vec-space.lin-indpt-full-rank*[*of mat-M n n*] *carrier-M basis mat-M-def*
**unfolding** *cof-vec-space.lin-indpt-list-def*
**by** *auto*
**then have** *det mat-M* ≠ *0* **using** *vec-space.det-rank-iff*[*of mat-M n*] *carrier-M* **by** *fastforce*
**then show** *invertible-mat mat-M* **using** *invertible-det*[*of mat-M n*] *carrier-M* **by** *fastforce*
**qed**

**abbreviation** *babai-out* **where** *babai-out* ≡ *babai-of-LLL* (−*t*) (*RAT M*)

**lemma** *babai-with-assms-epsilon-connect*:
**shows** *babai-with-assms-epsilon M t* α *2*
**using** *mat-M-inv-is-inv* **unfolding** *babai-with-assms-epsilon-def babai-with-assms-epsilon-axioms-def*
**using** *babai-axioms alpha basis mat-M-def non-trivial reduced* **by** *simp*

This shows that the output, which is of the form v-t, with v in L, is short

**lemma** *babai-correct*:
**shows** *real-of-rat* (*sq-norm* (*babai-out*)) ≤ (*real-of-rat* ((*rat-of-int n*)∗α^*n*) ∗ *closest-distance-sq*) ∧ (*babai-out*) ∈ *coset*
**proof**−
**have** ∗:*real-of-rat* (*sq-norm* (*babai-out*)) ≤ (*real-of-rat* ((*rat-of-int n*)∗α^*n*) ∗ *epsilon* ∗ *closest-distance-sq*) **if** *eps*:*1* < *epsilon* **for** *epsilon*
**proof**−
**have** *babai-with-assms-epsilon M t* α *epsilon*
**using** *eps mat-M-inv-is-inv* **unfolding** *babai-with-assms-epsilon-def babai-with-assms-epsilon-axioms-def*
**using** *babai-axioms alpha basis mat-M-def non-trivial reduced* **by** *blast*
**then show** *?thesis* **using** *babai-with-assms-epsilon.babai-correct*[*of M t* α *epsilon*] *babai-with-assms-epsilon.babai-to-help*[*of M t* α *epsilon*]
*s-def* **by** *simp*
**qed**
**have** *real-of-rat* (*sq-norm* (*babai-out*)) ≤ (*real-of-rat* ((*rat-of-int n*)∗α^*n*) ∗ *closest-distance-sq*)
**proof**(*rule ccontr*)
**assume** ¬*?thesis*

```
  then have not:(real-of-rat ((rat-of-int n)*α^n) * closest-distance-sq) < real-of-rat
(sq-norm (babai-of-LLL (−t) (RAT M))) by auto
    then obtain epsilon where (real-of-rat ((rat-of-int n)*α^n) * epsilon * clos-
est-distance-sq) < real-of-rat (sq-norm (babai-of-LLL (−t) (RAT M))) ∧ 1 < ep-
silon
    proof(cases closest-distance-sq = 0)
      case True
        then have (real-of-rat ((rat-of-int n)*α^n) * 2 * closest-distance-sq) =
(real-of-rat ((rat-of-int n)*α^n) * closest-distance-sq) by auto
      moreover have 1 < (2::real) by simp
      ultimately show ?thesis using that[of 2] not by presburger
    next
      case f:False
        then have pos:0<( real-of-rat ((rat-of-int n)*α^n) * closest-distance-sq)
using babai-with-assms-epsilon.closest-distance-sq-pos[of M t α 11/10]
      using mat-M-inv-is-inv unfolding babai-with-assms-epsilon-def babai-with-assms-epsilon-axioms-def
        using babai-axioms alpha basis mat-M-def non-trivial reduced by simp
    obtain delta where delta:(real-of-rat ((rat-of-int n)*α^n) * closest-distance-sq)
< delta ∧ delta < real-of-rat (sq-norm (babai-of-LLL (−t) (RAT M)))
        using dense not by blast
      define epsilon where e:epsilon = delta / (real-of-rat ((rat-of-int n)*α^n) *
closest-distance-sq)
      then have 1 < epsilon
        using pos delta divide-strict-right-mono
       divide-eq-1-iff[of (real-of-rat (rat-of-int (int n) * α ^ n) * closest-distance-sq)
(real-of-rat (rat-of-int (int n) * α ^ n) * closest-distance-sq)]
        by auto
    moreover have epsilon * (real-of-rat ((rat-of-int n)*α^n) * closest-distance-sq)
= delta using e pos
        by (metis nonzero-divide-eq-eq rel-simps(70))
      ultimately show ?thesis using that[of epsilon] not delta by argo
    qed
    then show False using *[of epsilon] by linarith
  qed
  then show ?thesis
     using babai-with-assms-epsilon.babai-correct babai-with-assms-epsilon-connect
babai-with-assms-epsilon.babai-to-help[of M t α 2]
       s-def by force
qed
```

This shifts the output, which is of the form $v - t$, with $v \in L$, back to $v$.

```
abbreviation vL where vL ≡  map-vec int-of-rat ( babai-out + t)

lemma shifted-babai-correct:
  shows vL ∈ L
  ∧ real-of-rat (sq-norm (map-vec rat-of-int (vL) − t)) ≤
   real-of-rat ((rat-of-int n)*α^n)*closest-distance-sq
proof−
  let ?vC = (babai-of-LLL (−t) (RAT M))
```

**have** *t-carr*: $t \in$ *carrier-vec n* **using** *babai-axioms* **unfolding** *babai-def* **by** *fast-force*
**have** *?vC* $\in$ *coset* **using** *babai-correct* **by** *blast*
**then obtain** *v* **where** *v*:*?vC* = *of-int-hom.vec-hom v* − *t* ∧ *v* ∈ *L* **by** *blast*
**then have** *?vC* + *t* = *of-int-hom.vec-hom v* + $0_v$ *n* **using** *t-carr* **by** *force*
**then have** *2*:*?vC* + *t* = *of-int-hom.vec-hom v* **using** *babai-with-assms-epsilon.lattice-carrier v babai-with-assms-epsilon-connect* **by** *force*
**then have** *1*:*map-vec int-of-rat* (*?vC* + *t*) = *map-vec int-of-rat* (*of-int-hom.vec-hom v*) **by** *fastforce*
**have** (*map-vec int-of-rat* (*of-int-hom.vec-hom v*))$*i* = *v*$*i* **if** *i*:*i*<*n* **for** *i*
**using** *i babai-with-assms-epsilon.lattice-carrier v babai-with-assms-epsilon-connect* **by** *fastforce*
**then have** *map-vec int-of-rat* (*of-int-hom.vec-hom v*) = *v* **using** *babai-with-assms-epsilon.lattice-carrier v babai-with-assms-epsilon-connect* **by** *auto*
**then have** *part1*:*map-vec int-of-rat* (*?vC* + *t*) ∈ *L* **using** *1 v*
**by** *auto*
**have** *map-vec rat-of-int* (*map-vec int-of-rat* (*?vC* + *t*)) = (*?vC* + *t*) **using** *1 2* **by** *fastforce*
**then have** *map-vec rat-of-int* (*map-vec int-of-rat* (*?vC* + *t*)) − *t* = *?vC*
**using** *t-carr part1 babai-with-assms-epsilon.lattice-carrier v babai-with-assms-epsilon-connect*
**by** *auto*
**then show** *?thesis* **using** *babai-correct part1*
**by** *presburger*
**qed**

**end**

Here we prove correctness of the full algorithm, which starts with a non-reduced basis and outputs a vector in L close to t.

**locale** *full-babai-with-assms* =
**fixes** *target*::*rat vec*
**fixes** *n*::*nat*
**fixes** *fs-init*::*int vec list*
**fixes** *α*::*rat*
**assumes** *non-triv*:*0*<*n*
**assumes** *t-dim*:*dim-vec target* = *n*
**assumes** *LLL-assms*:*LLL.LLL-with-assms n n fs-init α*
**begin**

**sublocale** *vec-space TYPE*(*rat*) *n* **.**

**abbreviation** *B*::*real* **where** *B* ≡ (*real-of-rat* (*α*^*n*) ∗ (*n*)) ∗ *babai.closest-distance-sq fs-init target*
**abbreviation** *out* **where** *out* ≡ (*full-babai fs-init target α*)

**lemma** *LLL-output-babai-assms*:
**shows** *babai-with-assms* (*map of-int-hom.vec-hom* (*LLL-Impl.reduce-basis α fs-init*)) *target α*
**proof**−

**define** *fs* **where** *fs* = (*LLL-Impl.reduce-basis* $\alpha$ *fs-init*)
**have** *1*:*LLL.reduced n* $\alpha$ *fs n* $\wedge$ *LLL.lin-indep n fs* $\wedge$ *length*(*fs*) = *n*
**using** *LLL-with-assms.reduce-basis*[*of n n fs-init* $\alpha$ *fs*] *LLL-assms* **unfolding** *fs-def* **by** *blast*
**then have** *2*:*LLL.weakly-reduced n* $\alpha$ *fs n* **unfolding** *gram-schmidt-fs.reduced-def* **by** *simp*
**moreover have** *babai* (*map of-int-hom.vec-hom* (*reduce-basis* $\alpha$ *fs-init*)) *target*
**using** *1* **unfolding** *babai-def fs-def t-dim* **by** *fastforce*
**moreover have** *gram-schmidt-fs.weakly-reduced n*
(*map of-int-hom.vec-hom* (*map of-int-hom.vec-hom fs*)) $\alpha$
*n*
**using** *2* **by** *force*
**moreover have** *4 / 3* $\leq$ $\alpha$ **using** *LLL-assms* **unfolding** *LLL-with-assms-def* **by** *simp*
**ultimately show** *?thesis*
**using** *LLL-assms 1 t-dim non-triv*
**unfolding** *babai-with-assms-def babai-with-assms-axioms-def fs-def* **by** *auto*
**qed**

**lemma** *full-babai-correct*:
**shows** *real-of-rat* (*sq-norm* ((*map-vec rat-of-int out*) − *target*))
≤ *real-of-rat* ($\alpha$) ^ *n* ∗ *real n* ∗ *babai.closest-distance-sq fs-init target* $\wedge$
*out* ∈ *vec-module.lattice-of n fs-init*
**proof**−
**define** *fs* **where** *fs* = (*LLL-Impl.reduce-basis* $\alpha$ *fs-init*)
**have** *bab*:*babai-with-assms fs target* $\alpha$
**using** *LLL-output-babai-assms* **unfolding** *fs-def* **by** *simp*
**then have** *l*:*length fs* = *n*
**unfolding** *babai-with-assms-def babai-with-assms-axioms-def babai-def*
**using** *t-dim* **by** *simp*
**have** *out* = *map-vec int-of-rat*
(*babai-of-LLL*
(− *target*)
(*LLL.RAT fs*)
+ *target*)
**unfolding** *fs-def* **using** *full-babai.simps*[*of fs-init target* $\alpha$] **by** *argo*
**then have** *out* = *map-vec int-of-rat*
(*babai-of-LLL* (*uminus target*) (*LLL.RAT fs*)
+ *target*)
**using** *babai-with-assms.babai-with-assms-epsilon-connect*
*babai-with-assms-epsilon.babai-to-help*[*of fs target*] *bab* **by** *metis*
**then have** *out* ∈ *vec-module.lattice-of n fs* $\wedge$
*real-of-rat*
$\|$*of-int-hom.vec-hom out* − *target*$\|^2$
≤ *real-of-rat* (*rat-of-int* (*int n*) ∗ $\alpha$ ^ *n*) ∗
*babai.closest-distance-sq fs target*
**using** *babai-with-assms.shifted-babai-correct*[*of fs target* $\alpha$] *bab l*
**unfolding** *LLL.L-def*

**by** *algebra*
**moreover have** *lattices*:*vec-module.lattice-of n fs* = *vec-module.lattice-of n fs-init*
**using** *LLL-with-assms.reduce-basis*(*1*)[*of n n fs-init α fs*] *LLL-assms l* **unfolding** *fs-def LLL.L-def*
**by** *argo*
**moreover have** *babai.closest-distance-sq fs target* = *babai.closest-distance-sq fs-init target*
**proof**−
**have** *lf*:*length fs-init* = *n*
**using** *LLL-assms* **unfolding** *LLL-with-assms-def* **by** *blast*
**then have** *babai fs target* ∧ *babai fs-init target*
**using** *t-dim l LLL-assms* **unfolding** *babai-def* **by** *argo*
**then show** *?thesis*
**using** *babai.closest-distance-sq-def*[*of fs target*]
*babai.closest-distance-sq-def*[*of fs-init target*]
*lattices l lf*
**unfolding** *LLL.L-def*
**by** *presburger*
**qed**
**moreover have** *real-of-rat* (*rat-of-int* (*int n*) ∗ *α* ^ *n*) = *real-of-rat* (*α*) ^ *n* ∗ *real n*
**by** (*simp add*: *of-rat-divide of-rat-mult of-rat-power*)
**ultimately show** *?thesis* **by** *force*
**qed**
**end**

Here we prove correctness of the full algorithm, starting with a target vector of TYPE(int).

**lemma** *full-babai-correct-int-target*:
**assumes** *full-babai-with-assms* (*map-vec rat-of-int target*) *n fs-init α*
**shows** *real-of-int* (*sq-norm* ( (*full-babai fs-init* (*map-vec rat-of-int target*) *α*) − *target*))
≤ (*real-of-rat*(*α*)^*n* ∗ (*n*)) ∗ *babai.closest-distance-sq fs-init* (*map-vec rat-of-int target*) ∧
(*full-babai fs-init* (*map-vec rat-of-int target*) *α*) ∈ *vec-module.lattice-of n fs-init*
**proof**−
**let** *?t* = *map-vec rat-of-int target*
**have** *real-of-rat* (*sq-norm* ( *of-int-hom.vec-hom* (*full-babai fs-init ?t α*) − *?t*))
≤ (*real-of-rat* (*α*)^*n* ∗ (*n*))∗ *babai.closest-distance-sq fs-init ?t* ∧
(*full-babai fs-init ?t α*) ∈ *vec-module.lattice-of n fs-init*
**using** *full-babai-with-assms.full-babai-correct*[*of ?t n fs-init*] *assms* **by** *blast*
**moreover have** *real-of-rat* (*sq-norm* ( *of-int-hom.vec-hom* (*full-babai fs-init ?t α*) − *?t*))
= *real-of-int* (*sq-norm* ( (*full-babai fs-init ?t α*) − *target*))
**proof**−
**have** *of-int-hom.vec-hom* (*full-babai fs-init ?t α*) − *?t* = *map-vec rat-of-int* ((*full-babai fs-init ?t α*) − *target*)
**by** *fastforce*

```
   moreover have sq-norm (map-vec rat-of-int ((full-babai fs-init ?t α) − target))
= rat-of-int (sq-norm ((full-babai fs-init ?t α) − target))
     using sq-norm-of-int[of ((full-babai fs-init ?t α) − target)] by blast
    ultimately have real-of-rat (sq-norm ( of-int-hom.vec-hom (full-babai fs-init
?t α) − ?t)) = real-of-rat (rat-of-int (sq-norm ((full-babai fs-init ?t α) − target)))

     by force
    also have real-of-rat (rat-of-int (sq-norm ((full-babai fs-init ?t α) − target)))
= real-of-int (sq-norm ((full-babai fs-init ?t α) − target)) by simp
    finally show ?thesis .
  qed
  ultimately show ?thesis by auto
qed
```

The literature typically states the main result using $2^n$ as the approximation constant, rather than $\alpha^n * n$. Here we show that $\alpha^n * n$ is stronger for $\alpha = 4/3$.

```
lemma clean-bound:
  fixes n::nat
  shows (4/3)^n * n ≤ 2 ^ n
proof(induct n)
  case 0
  then show ?case by simp
next
  case (Suc n)
  let ?SN = Suc n
  have ?SN=1∨?SN=2∨2<?SN by fastforce
  then show ?case
  proof(elim disjE)
    {assume 1:?SN = 1
     then have real-of-rat ((rat-of-int ?SN)*(4/3)^?SN) = real-of-rat ((rat-of-int
1)*4/3)
       by auto
     then show ?thesis using 1 by simp}
  next
    {assume 2:?SN=2
     then have real-of-rat ((rat-of-int ?SN)*(4/3)^?SN) = real-of-rat ((rat-of-int
2)*(4/3)^2)
       by (metis int-ops(3))
     then show ?thesis
       using 2 by auto}
  next
    {assume ind:?SN>2
     then have n>0 by simp
     then have 1:?SN = n*(?SN/n) by auto
     moreover have 2:((4::rat)/3)^?SN = (4/3)^n*(4/3) by auto
    ultimately have 3:real-of-rat ((rat-of-int ?SN)*(4/3)^?SN) = (n*(?SN/n))
* (real-of-rat ((4/3)^n*(4/3)))
       by (metis of-int-of-nat-eq of-rat-mult of-rat-of-nat-eq)
```

```
    also have (n*(?SN/n)) * real-of-rat ((4/3)^n*(4/3)) = real-of-rat ((rat-of-int
n) * (4/3)^n) * ((?SN/n) *(real-of-rat (4/3)))
     by (simp add: ‹real (Suc n) = real n * (real (Suc n) / real n)› mult-of-int-commute
of-rat-divide of-rat-mult)
    finally have real-of-rat ((rat-of-int ?SN)*(4/3)^?SN) = real-of-rat ((rat-of-int
n) * (4/3)^n) * ((?SN/n) *(real-of-rat (4/3)))
      by presburger
    then have real-of-rat ((rat-of-int ?SN)*(4/3)^?SN) = real-of-rat ((rat-of-int
n) * (4/3)^n) * ((?SN/n) *(real-of-rat (4/3)))
      by argo
    moreover have ((?SN/n) *(real-of-rat (4/3)))≤2
    proof−
      have N-big:2≤n using ind
        by force
      then have 4≤2*n by fastforce
      then have 4*n+4≤6*n by fastforce
      then have 4/3*(Suc n)≤2*n by auto
      moreover have 0<1/n using N-big by simp
      ultimately have (4/3*?SN)* (1/n)≤ 2*n*(1/n)
        using N-big mult-right-mono[of (4/3*?SN) 2*n (1/n)] by linarith
      then have (4/3*?SN)/n≤ 2*n/n by argo
      then have 4/3*(?SN / n)≤ 2*(n/n) by linarith
      then have 4/3*(?SN/n)≤ 2 using N-big by auto
      moreover have 4/3 = real-of-rat (4/3) using of-rat-divide
        by (metis of-rat-numeral-eq)
      ultimately have (real-of-rat (4/3))*(?SN/n)≤ 2 by algebra
      then show ?thesis by argo
    qed
    moreover have
      0≤real-of-rat (rat-of-int (int n) * (4/3) ^ n) by force
    moreover have 0≤(?SN/n)*(real-of-rat (4/3)) by simp
     moreover have (4 / 3) ^ n * real n * (real (Suc n) / real n * real-of-rat
(4 / 3)) = real-of-rat (rat-of-int (int n) * (4 / 3) ^ n) * (real (Suc n) / real n *
real-of-rat (4 / 3))
      by (simp add: of-rat-divide of-rat-mult of-rat-power)
    ultimately have real-of-rat ((rat-of-int ?SN)*(4/3)^?SN) ≤2^n * 2
      using Suc mult-mono[of
                          (4 / 3) ^ n * real n
                          2^n
                          ((?SN/n) *(real-of-rat (4/3)))
                          2] by force
    then show ?thesis
    by (metis 1 3 mult-of-nat-commute of-rat-divide of-rat-numeral-eq of-rat-power
power-Suc2)
  }
 qed
qed

end
```

**theory** *Misc-PMF*
**imports**
*HOL−Probability.Probability*
*LLL-Basis-Reduction.LLL-Impl*

**begin**

**definition** *replicate-spmf* :: *nat* ⇒ *'b pmf* ⇒ *'b list spmf* **where**
*replicate-spmf m p* = *spmf-of-pmf* (*replicate-pmf m p*)

The preceding *replicate-spmf* definition is copied from *CRYSTALS−Kyber-Security*. We do this to avoid loading the entire library. In fact, we do not need *replicate-spmf* at all for the HNP. However, for each *replicate-pmf* result we prove here, we also prove a corresponding *replicate-spmf* result. The *replicate-spmf* results are here for completeness, but not needed for the HNP.

# 8 SPMF and PMF helper lemmas

**lemma** *spmf-eq-element*: *spmf* (*p* ⤜ (λ*x*. *return-spmf* (*x* = *t*))) *True* = *spmf p t*
**proof**−
**have** ∗: *map-option* (λ*a*. *a* = *t*) −‘ {*Some True*} = {*Some t*} **by** *fastforce*
**have** (*p* ⤜ (λ*x*. *return-spmf* (*x* = *t*))) = *map-spmf* (λ*a*. *a* = *t*) *p*
**by** (*simp add*: *map-spmf-conv-bind-spmf*)
**also have** *spmf* ... *True* = *spmf p t* **by** (*simp add*: *pmf-def* ∗)
**finally show** *?thesis* **.**
**qed**

**lemma** *pmf-true-false*:
**fixes** *p* :: *'a pmf*
**fixes** *P Q* :: *'a* ⇒ *bool*
**defines** *a* ≡ *p* ⤜ (λ*x*. *return-pmf* (*P x*))
**defines** *b* ≡ *p* ⤜ (λ*x*. *return-pmf* (¬ *P x*))
**shows** *pmf a True* = *pmf b False*
**proof**−
**have** *a* = *map-pmf P p* **by** (*simp add*: *a-def map-pmf-def*)
**moreover have** *b* = *map-pmf* (λ*x*. ¬ *P x*) *p* **by** (*simp add*: *b-def map-pmf-def*)
**moreover have** *P* −‘ {*True*} = (λ*x*. ¬ *P x*) −‘ {*False*} **by** *blast*
**ultimately show** *?thesis* **by** (*simp add*: *pmf-def*)
**qed**

**lemma** *spmf-true-false*:
**fixes** *p* :: *'a spmf*
**fixes** *P Q* :: *'a* ⇒ *bool*
**defines** *a* ≡ *p* ⤜ (λ*x*. *return-spmf* (*P x*))
**defines** *b* ≡ *p* ⤜ (λ*x*. *return-spmf* (¬ *P x*))
**shows** *spmf a True* = *spmf b False*
**proof**−

**have** $a = map\text{-}spmf\ P\ p$ **by** (*simp add*: *a-def map-spmf-conv-bind-spmf*)
**moreover have** $b = map\text{-}spmf\ (\lambda x.\ \neg\ P\ x)\ p$ **by** (*simp add*: *b-def map-spmf-conv-bind-spmf*)
**moreover have** $map\text{-}option\ P\ -`\ \{Some\ True\} = map\text{-}option\ (\lambda x.\ \neg\ P\ x)\ -`\ \{Some\ False\}$ **by** *blast*
**ultimately show** *?thesis* **by** (*simp add*: *pmf-def*)
**qed**

**lemma** *pmf-subset*:
**fixes** $p :: 'a\ pmf$
**fixes** $P\ Q :: 'a \Rightarrow bool$
**assumes** $\forall x \in set\text{-}pmf\ p.\ P\ x \longrightarrow Q\ x$
**shows** $pmf\ (p \gg= (\lambda x.\ return\text{-}pmf\ (P\ x)))\ True \leq pmf\ (p \gg= (\lambda x.\ return\text{-}pmf\ (Q\ x)))\ True$
**proof**−
**have** $p \gg= (\lambda x.\ return\text{-}pmf\ (P\ x)) = map\text{-}pmf\ P\ p$ **by** (*simp add*: *map-pmf-def*)
**moreover have** $p \gg= (\lambda x.\ return\text{-}pmf\ (Q\ x)) = map\text{-}pmf\ Q\ p$ **by** (*simp add*: *map-pmf-def*)
**moreover have** $pmf\ (map\text{-}pmf\ P\ p)\ True \leq pmf\ (map\text{-}pmf\ Q\ p)\ True$
**proof**−
**have** $pmf\ (map\text{-}pmf\ P\ p)\ True = prob\text{-}space.prob\ p\ ((P\ -`\ \{True\}) \cap set\text{-}pmf\ p)$
(**is** - = *prob-space.prob p ?lhs*)
**by** (*simp add*: *measure-Int-set-pmf pmf-def*)
**moreover have** $pmf\ (map\text{-}pmf\ Q\ p)\ True = prob\text{-}space.prob\ p\ ((Q\ -`\ \{True\}) \cap set\text{-}pmf\ p)$
(**is** - = *prob-space.prob p ?rhs*)
**by** (*simp add*: *measure-Int-set-pmf pmf-def*)
**moreover have** *?lhs* $\subseteq$ *?rhs* **using** *assms in-set-spmf* **by** *fast*
**ultimately show** *?thesis* **by** (*simp add*: *measure-pmf.finite-measure-mono*)
**qed**
**ultimately show** *?thesis* **by** *presburger*
**qed**

**lemma** *pmf-subset'*:
**fixes** $p :: 'a\ pmf$
**fixes** $P\ Q :: 'a \Rightarrow bool$
**assumes** $\forall x \in set\text{-}pmf\ p.\ \neg\ P\ x \longrightarrow \neg\ Q\ x$
**shows** $pmf\ (p \gg= (\lambda x.\ return\text{-}pmf\ (P\ x)))\ False \leq pmf\ (p \gg= (\lambda x.\ return\text{-}pmf\ (Q\ x)))\ False$
**using** *pmf-subset*[*OF assms*(*1*)] *pmf-true-false*[*of p* $\lambda x.\ \neg\ P\ x$] *pmf-true-false*[*of p* $\lambda x.\ \neg\ Q\ x$]
**by** *algebra*

**lemma** *spmf-subset*:
**fixes** $p :: 'a\ spmf$
**fixes** $P\ Q :: 'a \Rightarrow bool$
**assumes** $\forall x \in set\text{-}spmf\ p.\ P\ x \longrightarrow Q\ x$
**shows** $spmf\ (p \gg= (\lambda x.\ return\text{-}spmf\ (P\ x)))\ True \leq spmf\ (p \gg= (\lambda x.\ return\text{-}spmf\ (Q\ x)))\ True$

**proof** −
**have** $p \gg\!= (\lambda x.\ return\text{-}spmf\ (P\ x)) = map\text{-}spmf\ P\ p$ **by** (*simp add*: *map-spmf-conv-bind-spmf*)
**moreover have** $p \gg\!= (\lambda x.\ return\text{-}spmf\ (Q\ x)) = map\text{-}spmf\ Q\ p$ **by** (*simp add*: *map-spmf-conv-bind-spmf*)
**moreover have** $spmf\ (map\text{-}spmf\ P\ p)\ True \leq spmf\ (map\text{-}spmf\ Q\ p)\ True$
**proof** −
**have** $spmf\ (map\text{-}spmf\ P\ p)\ True = prob\text{-}space.prob\ p\ ((map\text{-}option\ P -` \{Some\ True\}) \cap set\text{-}pmf\ p)$
(**is** - = *prob-space.prob p ?lhs*)
**by** (*simp add*: *measure-Int-set-pmf pmf-def*)
**moreover have** $spmf\ (map\text{-}spmf\ Q\ p)\ True = prob\text{-}space.prob\ p\ ((map\text{-}option\ Q -` \{Some\ True\}) \cap set\text{-}pmf\ p)$
(**is** - = *prob-space.prob p ?rhs*)
**by** (*simp add*: *measure-Int-set-pmf pmf-def*)
**moreover have** *?lhs* $\subseteq$ *?rhs* **using** *assms in-set-spmf* **by** *fast*
**ultimately show** *?thesis* **by** (*simp add*: *measure-pmf.finite-measure-mono*)
**qed**
**ultimately show** *?thesis* **by** *presburger*
**qed**

**lemma** *spmf-subset'*:
**fixes** $p :: 'a\ spmf$
**fixes** $P\ Q :: 'a \Rightarrow bool$
**assumes** $\forall x \in set\text{-}spmf\ p.\ \neg\ P\ x \longrightarrow \neg\ Q\ x$
**shows** $spmf\ (p \gg\!= (\lambda x.\ return\text{-}spmf\ (P\ x)))\ False \leq spmf\ (p \gg\!= (\lambda x.\ return\text{-}spmf\ (Q\ x)))\ False$
**using** *spmf-subset*[*OF assms*(*1*)] *spmf-true-false*[*of p* $\lambda x.\ \neg\ P\ x$] *spmf-true-false*[*of p* $\lambda x.\ \neg\ Q\ x$]
**by** *algebra*

**lemma** *pmf-in-set*:
**fixes** $A :: 'a\ set$
**fixes** $p :: 'a\ pmf$
**shows** $pmf\ (p \gg\!= (\lambda x.\ return\text{-}pmf\ (x \in A)))\ True = prob\text{-}space.prob\ p\ A$
**proof** −
**have** $p \gg\!= (\lambda x.\ return\text{-}pmf\ (x \in A)) = map\text{-}pmf\ (\lambda x.\ x \in A)\ p$ **by** (*simp add*: *map-pmf-def*)
**also have** $pmf\ \ldots\ True = prob\text{-}space.prob\ p\ A$
**proof** −
**have** $(\lambda x.\ x \in A) -` \{True\} = A$ **by** *blast*
**thus** *?thesis* **by** (*simp add*: *pmf-def*)
**qed**
**finally show** *?thesis* **.**
**qed**

**lemma** *pmf-of-prop*:
**fixes** $P :: 'a \Rightarrow bool$
**fixes** $p :: 'a\ pmf$
**shows** $pmf\ (p \gg\!= (\lambda x.\ return\text{-}pmf\ (P\ x)))\ True = prob\text{-}space.prob\ p\ \{x \in set\text{-}pmf$

*p. P x*}
**using** *pmf-in-set*[*of p* {*x* ∈ *set-pmf p. P x*}]
**by** (*smt* (*verit*, *best*) *bind-pmf-cong mem-Collect-eq*)

**lemma** *spmf-in-set*:
**fixes** *A* :: *'a set*
**fixes** *p* :: *'a spmf*
**shows** *spmf* (*p* ⤜ (λ*x*. *return-spmf* (*x* ∈ *A*))) *True* = *prob-space.prob p* (*Some‘A*)
**proof**−
**have** *p* ⤜ (λ*x*. *return-spmf* (*x* ∈ *A*)) = *map-spmf* (λ*x*. *x* ∈ *A*) *p*
**by** (*simp add*: *map-spmf-conv-bind-spmf*)
**also have** *spmf* ... *True* = *prob-space.prob p* (*Some‘A*)
**proof**−
**have** *map-option* (λ*x*. *x* ∈ *A*) −‘ {*Some True*} = *Some‘A* **by** *blast*
**thus** *?thesis* **by** (*simp add*: *pmf-def*)
**qed**
**finally show** *?thesis* **.**
**qed**

**lemma** *spmf-of-prop*:
**fixes** *P* :: *'a* ⇒ *bool*
**fixes** *p* :: *'a spmf*
**shows** *spmf* (*p* ⤜ (λ*x*. *return-spmf* (*P x*))) *True* = *prob-space.prob p* (*Some‘*{*x* ∈ *set-spmf p. P x*})
**using** *spmf-in-set*[*of p* {*x* ∈ *set-spmf p. P x*}]
**by** (*smt* (*verit*) *bind-spmf-cong mem-Collect-eq*)

**lemma** *replicate-pmf-events-helper*:
**fixes** *p* :: *'a pmf*
**fixes** *n P*
**defines** *lhs* ≡ *pmf* (*pair-pmf p* (*replicate-pmf n p*) ⤜ (λ(*x*, *xs*). *return-pmf* (*x* # *xs*)) ⤜ (λ*xs*. *return-pmf* (*P xs*))) *True*
**defines** *rhs* ≡ *pmf* (*pair-pmf p* (*replicate-pmf n p*) ⤜ (λ(*x*, *xs*). *return-pmf* (*P* (*x* # *xs*)))) *True*
**shows** *lhs* = *rhs*
**unfolding** *lhs-def rhs-def pmf-def* **apply** *simp*
**by** (*smt* (*verit*, *ccfv-SIG*) *bind-assoc-pmf bind-pmf-cong bind-return-pmf case-prod-beta′*)

**lemma** *replicate-pmf-events*:
**fixes** *p* :: *'a pmf*
**fixes** *n* :: *nat*
**fixes** *E* :: *nat* ⇒ *'a* ⇒ *bool*
**assumes** ∀ *i* < *n*. *pmf* (*p* ⤜ (λ*x*. *return-pmf* (*E i x*))) *True* = *A i*
**shows** *pmf* (*replicate-pmf n p* ⤜ (λ*xs*. *return-pmf* (∀ *i* < *n*. *E i* (*xs*!*i*)))) *True* = (∏ *i*<*n*. *A i*)
**using** *assms*
**proof**(*induct n arbitrary*: *E A*)
**case** *0*
**thus** *?case* **by** *simp*

```
next
  case (Suc n)
  define ps where ps ≡ replicate-pmf (Suc n) p
  define ps' where ps' ≡ replicate-pmf n p
  define E' where E' ≡ λn. E (Suc n)
  define A' where A' ≡ (λi. pmf (p ⤜ (λx. return-pmf (E' i x))) True)
   have unfold: replicate-pmf (Suc n) p = do {x ← p; xs ← replicate-pmf n p;
return-pmf (x#xs)}
    unfolding replicate-pmf-def by force

  let ?rpmf0 = do {x ← p; xs ← ps'; return-pmf (x#xs)}
  let ?rpmf0' = do {(x, xs) ← pair-pmf p ps'; return-pmf (x#xs)}
  let ?rpmf1' = pair-pmf p ps'
  have 0: ?rpmf0 = ?rpmf0'
   by (smt (verit, best) bind-assoc-pmf bind-pmf-cong bind-return-pmf internal-case-prod-conv
internal-case-prod-def pair-pmf-def)
  have ∗: ∀ xs ∈ set-pmf ?rpmf0.
    (∀ i < Suc n. E i (xs!i)) ⟷ (E 0 (hd xs) ∧ (∀ i < n. E' i ((tl xs)!i)))
  proof
    fix xs assume xs ∈ set-pmf ?rpmf0
     hence len: length xs = Suc n unfolding ps'-def using set-replicate-pmf by
fastforce
    show (∀ i < Suc n. E i (xs ! i)) ⟷ (E 0 (hd xs) ∧ (∀ i<n. E' i (tl xs ! i)))
    proof
      assume ∗: ∀ i < Suc n. E i (xs ! i)
      hence E 0 (hd xs)
        by (metis len Nat.nat.distinct(1) hd-conv-nth length-0-conv zero-less-Suc)
      moreover have (∀ i<n. E' i (tl xs ! i))
      proof safe
        fix i assume ∗∗: i < n
        hence Suc i < Suc n by blast
        hence E (Suc i) (xs ! (Suc i)) using ∗ by presburger
        thus E' i (tl xs ! i) unfolding E'-def by (simp add: ∗∗ nth-tl len)
      qed
      ultimately show E 0 (hd xs) ∧ (∀ i<n. E' i (tl xs ! i)) by blast
    next
      assume ∗: E 0 (hd xs) ∧ (∀ i<n. E' i (tl xs ! i))
      show ∀ i < Suc n. E i (xs ! i)
      proof safe
        fix i assume ∗∗: i < Suc n
        show E i (xs ! i)
        proof(cases i = 0)
          case True
             then show ?thesis by (metis ∗ Nat.nat.distinct(1) hd-conv-nth len
length-0-conv)
        next
          case False
          hence E' (i − 1) = E i unfolding E'-def by simp
          moreover have i − 1 < n using ∗∗ False by linarith
```

```
        ultimately show ?thesis
          by (metis * ** False diff-Suc-1 len length-tl less-Suc-eq-0-disj nth-tl)
      qed
    qed
  qed
qed

have pmf (ps ⤜ (λxs. return-pmf (∀ i < Suc n. E i (xs!i)))) True
    = pmf (?rpmf0 ⤜ (λxs. return-pmf (∀ i < Suc n. E i (xs!i)))) True
  by (simp add: replicate-spmf-def ps-def ps'-def)
also have ... = pmf (?rpmf0 ⤜ (λxs. return-pmf (E 0 (hd xs) ∧ (∀ i < n. E' i ((tl xs)!i))))) True
  unfolding pmf-def by (smt (verit, ccfv-SIG) * bind-pmf-cong)
also have ... = pmf (?rpmf0' ⤜ (λxs. return-pmf (E 0 (hd xs) ∧ (∀ i < n. E' i ((tl xs)!i))))) True
  using 0 by presburger
also have ... = pmf (?rpmf1' ⤜ (λ(x,xs). return-pmf (E 0 (hd (x#xs)) ∧ (∀ i < n. E' i ((tl (x#xs))!i))))) True
  using replicate-pmf-events-helper[of p n λxs. (E 0 (hd xs) ∧ (∀ i < n. E' i ((tl xs)!i))), folded ps'-def] .
also have ... = pmf (?rpmf1' ⤜ (λ(x,xs). return-pmf (E 0 x ∧ (∀ i < n. E' i (xs!i))))) True
  by simp
also have ... = pmf (?rpmf1' ⤜ (λx. return-pmf (E 0 (fst x) ∧ (∀ i<n. E' i (snd x ! i))))) True
proof−
  have (λ(x, xs). return-pmf (E 0 x ∧ (∀ i<n. E' i (xs ! i)))) = (λx. return-pmf (E 0 (fst x) ∧ (∀ i<n. E' i (snd x ! i))))
    by force
  thus ?thesis by presburger
qed
also have ... = (measure (measure-pmf ?rpmf1')
    {x ∈ set-pmf ?rpmf1'. E 0 (fst x) ∧ (∀ i<n. E' i (snd x ! i))})
  using pmf-of-prop[of ?rpmf1' λx-xs. E 0 (fst x-xs) ∧ (∀ i<n. E' i ((snd x-xs) ! i))] .
also have ... = measure (measure-pmf ?rpmf1')
    {(x,xs) ∈ set-pmf ?rpmf1'. E 0 x ∧ (∀ i < n. E' i (xs!i))}
proof−
  have {x ∈ set-pmf ?rpmf1'. E 0 (fst x) ∧ (∀ i<n. E' i (snd x ! i))}
      ⊆ {(x,xs) ∈ set-pmf ?rpmf1'. E 0 x ∧ (∀ i < n. E' i (xs!i))}
    by force
  moreover have {(x,xs) ∈ set-pmf ?rpmf1'. E 0 x ∧ (∀ i < n. E' i (xs!i))}
      ⊆ {x ∈ set-pmf ?rpmf1'. E 0 (fst x) ∧ (∀ i<n. E' i (snd x ! i))}
    by force
  ultimately show ?thesis by force
qed
also have ... = measure (measure-pmf p) {x ∈ set-pmf p. E 0 x}
    * measure (measure-pmf ps') {xs ∈ set-pmf ps'. ∀ i < n. E' i (xs!i)}
proof−
```

```
    let ?A = {x ∈ set-pmf p. E 0 x}
    let ?B = {xs ∈ set-pmf ps'. ∀ i < n. E' i (xs!i)}
    have 1: countable ?A by simp
    have 2: countable ?B by simp
   have {(x,xs) ∈ set-pmf ?rpmf1'. E 0 x ∧ (∀ i < n. E' i (xs!i))} = ?A × ?B by
force
    thus ?thesis using measure-pmf-prob-product[OF 1 2, of p ps'] by presburger
  qed
  also have ... = A 0 * (∏ i<n. A' i)
  proof−
    have 1: ∀ i<n. pmf (p ⤜ (λx. return-pmf (E' i x))) True = A' i
      unfolding A'-def by blast
    have measure (measure-pmf p) {x ∈ set-pmf p. E 0 x} = A 0
      using Suc.prems pmf-of-prop[of p E 0] by force
     moreover have measure (measure-pmf ps') {xs ∈ set-pmf ps'. ∀ i < n. E' i
(xs!i)} = (∏ i < n. A' i)
      using Suc.hyps[OF 1, folded ps'-def]
       using pmf-of-prop[of replicate-pmf n p λxs. (∀ i < n. E' i (xs ! i)), folded
ps'-def]
      by presburger
    ultimately show ?thesis by presburger
  qed
  also have ... = A 0 * (∏ i=1..<Suc n. A i)
  proof−
    have (∏ i=1..<Suc n. A i) = (∏ i=1..<Suc n. A' (i − 1))
      unfolding A'-def E'-def using Suc.prems by force
    also have ... = (∏ i=0..<n. A' i)
     by (metis (mono-tags, lifting) Groups-Big.comm-monoid-mult-class.prod.cong
Set-Interval.comm-monoid-mult-class.prod.atLeast-Suc-lessThan-Suc-shift diff-Suc-1
numeral-nat(7) o-apply)
    finally show ?thesis using atLeast0LessThan by presburger
  qed
  also have ... = (∏ i<Suc n. A i)
   by (simp add: prod.atLeast1-atMost-eq prod.atMost-shift atLeastLessThanSuc-atLeastAtMost
lessThan-Suc-atMost)
  finally show ?case unfolding ps-def by auto
qed

lemma replicate-pmf-same-event:
  fixes p :: 'a pmf
  fixes n :: nat
  fixes E :: 'a ⇒ bool
  assumes pmf (p ⤜ (λx. return-pmf (E x))) True = A
  shows pmf (replicate-pmf n p ⤜ (λxs. return-pmf (∀ i < n. E (xs!i)))) True =
A^n
  using replicate-pmf-events[of n p λi::nat. E λi::nat. A] assms by force

lemma replicate-pmf-same-event-leq:
  fixes p :: 'a pmf
```

**fixes** $n :: nat$
**fixes** $E :: 'a \Rightarrow bool$
**assumes** $pmf\ (p \gg\!= (\lambda x.\ return\text{-}pmf\ (E\ x)))\ True \leq A$
**shows** $pmf\ (replicate\text{-}pmf\ n\ p \gg\!= (\lambda xs.\ return\text{-}pmf\ (\forall\, i < n.\ E\ (xs!i))))\ True \leq A \widehat{} n$
**by** (*metis replicate-pmf-same-event assms pmf-nonneg pow-mono*)

**lemma** *replicate-spmf-events*:
**fixes** $p :: 'a\ spmf$
**fixes** $n :: nat$
**fixes** $E :: nat \Rightarrow 'a\ option \Rightarrow bool$
**assumes** $\forall\, i < n.\ (spmf\ (p \gg\!= (\lambda x.\ return\text{-}spmf\ (E\ i\ x)))\ True = A\ i)$
**shows** $spmf\ (replicate\text{-}spmf\ n\ p \gg\!= (\lambda xs.\ return\text{-}spmf\ (\forall\, i < n.\ E\ i\ (xs!i))))\ True = (\prod i{<}n.\ A\ i)$
**proof**−
**have** $*$: $\forall\, i < n.\ pmf\ (p \gg\!= (\lambda x.\ return\text{-}pmf\ (E\ i\ x)))\ True = spmf\ (p \gg\!= (\lambda x.\ return\text{-}spmf\ (E\ i\ x)))\ True$
**by** (*metis* (*no-types*, *lifting*) *bind-pmf-cong spmf-of-pmf-bind spmf-of-pmf-return-pmf spmf-spmf-of-pmf*)
**have** $spmf\ (replicate\text{-}spmf\ n\ p \gg\!= (\lambda xs.\ return\text{-}spmf\ (\forall\, i < n.\ E\ i\ (xs!i))))\ True = pmf\ (replicate\text{-}pmf\ n\ p \gg\!= (\lambda xs.\ return\text{-}pmf\ (\forall\, i < n.\ E\ i\ (xs!i))))\ True$
**by** (*metis* (*no-types*, *lifting*) *bind-spmf-cong bind-spmf-of-pmf replicate-spmf-def spmf-of-pmf-bind spmf-of-pmf-return-pmf spmf-spmf-of-pmf*)
**also have** $... = (\prod i{<}n.\ A\ i)$
**using** *replicate-pmf-events*[*OF* $*$] *assms*(*1*) **by** *force*
**finally show** *?thesis* **.**
**qed**

**lemma** *replicate-spmf-same-event*:
**fixes** $p :: 'a\ spmf$
**fixes** $n :: nat$
**fixes** $E :: 'a\ option \Rightarrow bool$
**assumes** $spmf\ (p \gg\!= (\lambda x.\ return\text{-}spmf\ (E\ x)))\ True = A$
**shows** $spmf\ (replicate\text{-}spmf\ n\ p \gg\!= (\lambda xs.\ return\text{-}spmf\ (\forall\, i < n.\ E\ (xs!i))))\ True = A \widehat{} n$
**using** *replicate-spmf-events*[*of n p* $\lambda i{::}nat.\ E\ \lambda i{::}nat.\ A$] *assms* **by** *force*

**lemma** *replicate-pmf-indep'*:
**fixes** $p :: 'a\ pmf$
**fixes** $n :: nat$
**fixes** $E :: nat \Rightarrow 'a \Rightarrow bool$
**defines** $rp \equiv replicate\text{-}pmf\ n\ p$
**assumes** $\forall\, i < n.\ pmf\ (p \gg\!= (\lambda x.\ return\text{-}pmf\ (E\ i\ x)))\ True = A\ i$
**assumes** $I \subseteq \{..{<}n\}$
**assumes** $\forall\, i < n.\ i \notin I \longrightarrow A\ i = 1$
**shows** $pmf\ (replicate\text{-}pmf\ n\ p \gg\!= (\lambda xs.\ return\text{-}pmf\ (\forall\, i < n.\ E\ i\ (xs!i))))\ True = (\prod i{\in}I.\ A\ i)$
**proof**−
**have** $(\prod i{<}n.\ A\ i) = (\prod i{\in}I.\ A\ i)$

**proof**−
**have** $(\prod i{<}n.\ A\ i) = (\prod i{\in}\{..{<}n\}.\ A\ i)$ **by** *blast*
**moreover have** ... $= (\prod i{\in}\{..{<}n\}{-}I.\ A\ i) * (\prod i{\in}I \cap \{..{<}n\}.\ A\ i)$
**by** (*simp add*: *Groups-Big.comm-monoid-mult-class.prod.subset-diff Int-absorb2 assms*(*3*))
**moreover have** ... $= (\prod i{\in}\{..{<}n\}{-}I.\ 1) * (\prod i{\in}I \cap \{..{<}n\}.\ A\ i)$ **using** *assms*(*4*) **by** *simp*
**moreover have** ... $= (\prod i{\in}I.\ A\ i)$
**proof**−
**have** $(\prod i{\in}\{..{<}n\}{-}I.\ 1) = 1$
**using** *Groups-Big.comm-monoid-mult-class.prod.neutral-const* **by** *blast*
**moreover have** $I \cap \{..{<}n\} = I$ **using** *assms*(*3*) **by** *blast*
**ultimately show** *?thesis* **by** *force*
**qed**
**ultimately show** $(\prod i{<}n.\ A\ i) = (\prod i{\in}I.\ A\ i)$ **by** *presburger*
**qed**
**thus** *?thesis* **using** *replicate-pmf-events*[*OF assms*(*2*)] **by** *presburger*
**qed**

**lemma** *replicate-pmf-indep''*:
**fixes** $p :: 'a\ pmf$
**fixes** $n :: nat$
**fixes** $E :: 'a \Rightarrow bool$
**fixes** $i :: nat$
**defines** $rp \equiv replicate\text{-}pmf\ n\ p$
**defines** $A \equiv pmf\ (p \gg\!= (\lambda x.\ return\text{-}pmf\ (E\ x)))\ True$
**assumes** $i < n$
**shows** $pmf\ (replicate\text{-}pmf\ n\ p \gg\!= (\lambda xs.\ return\text{-}pmf\ (E\ (xs!i))))\ True = A$
**proof**−
**let** $?I = \{i\}$
**let** $?E = \lambda j\ x.\ if\ j = i\ then\ E\ x\ else\ True$
**let** $?A = \lambda j.\ if\ j = i\ then\ A\ else\ 1$

**have** *1*: $\forall\, i < n.\ (pmf\ (p \gg\!= (\lambda x.\ return\text{-}pmf\ (?E\ i\ x)))\ True = ?A\ i)$ **using** *assms*(*2*) **by** *simp*
**have** *2*: $?I \subseteq \{..{<}n\}$ **using** *assms*(*3*) **by** *blast*
**have** *3*: $\forall\, i < n.\ i \notin ?I \longrightarrow ?A\ i = 1$ **by** *auto*
**have** $*$: $(\lambda xs.\ return\text{-}pmf\ (E\ (xs\ !\ i)))$
$= (\lambda xs.\ return\text{-}pmf\ (\forall j{<}n.\ if\ j = i\ then\ E\ (xs\ !\ j)\ else\ True))$
**using** *assms*(*3*) **by** *presburger*
**have** $**$: $A = (\prod j{\in}\{i\}.\ if\ j = i\ then\ A\ else\ 1)$ **by** *fastforce*

**show** *?thesis* **using** *replicate-pmf-indep'*[*OF 1 2 3*, *folded* $** *$] **.**
**qed**

**lemma** *replicate-pmf-indep*:
**fixes** $p :: 'a\ pmf$
**fixes** $n :: nat$
**fixes** $E :: nat \Rightarrow 'a \Rightarrow bool$

**defines** *rp* ≡ *replicate-pmf n p*
**shows** *prob-space.indep-events rp* (λ*i*. {*xs* ∈ *rp*. *E i* (*xs*!*i*)}) {..<*n*}
**proof**−
**define** *A* **where** *A* ≡ (λ*i*. *pmf* (*p* ⤜ (λ*x*. *return-pmf* (*E i x*))) *True*)
**let** *?I* = {..<*n*}
**let** *?S* = λ*i*. {*xs* ∈ *rp*. *E i* (*xs*!*i*)}

**have** *0*: *prob-space rp* **by** *unfold-locales*
**have** *1*: *?S*‘*?I* ⊆ *prob-space.events rp* **by** *fastforce*
**have** *2*: (∀ *J*⊆*?I*.
*J* ≠ {}
⟶ *finite J*
⟶ *prob-space.prob rp* (⋂*j*∈*J*. *?S j*) = (∏*j*∈*J*. *prob-space.prob rp* (*?S j*)))
**proof** *clarify*
**fix** *J* **assume** ∗: *J* ⊆ *?I J* ≠ {} *finite J*

**define** *E*′ **where** *E*′ ≡ (λ*i x*. (*if i* ∈ *J then E i x else True*))
**define** *A*′ **where** *A*′ ≡ (λ*i*. (*if i* ∈ *J then A i else 1*))

**have** *E*′*A*′: ∀ *i*<*n*. *pmf* (*p* ⤜ (λ*x*. *return-pmf* (*E*′ *i x*))) *True* = *A*′ *i*
**unfolding** *A*′*-def E*′*-def A-def* **by** *force*
**have** *A*′*-notin-J*: ∀ *i*<*n*. *i* ∉ *J* ⟶ *A*′ *i* = *1* **unfolding** *A*′*-def* **by** *presburger*

**have** (⋂*j*∈*J*. *?S j*) = {*xs* ∈ *rp*. (∀ *j* ∈ *J*. *E j* (*xs*!*j*))} **using** ∗(*2*) **by** *blast*
**hence** *prob-space.prob rp* (⋂*j*∈*J*. *?S j*) = *prob-space.prob rp* {*xs* ∈ *rp*. (∀ *j* ∈ *J*. *E j* (*xs*!*j*))}
**by** *presburger*
**also have** ... = *pmf* (*rp* ⤜ (λ*xs*. *return-pmf* (∀ *j* ∈ *J*. *E*′ *j* (*xs*!*j*)))) *True*
**by** (*simp add*: *pmf-of-prop E*′*-def*)
**also have** ... = *pmf* (*rp* ⤜ (λ*xs*. *return-pmf* (∀ *j*<*n*. *E*′ *j* (*xs*!*j*)))) *True*
**proof**−
**have** (λ*xs*. *return-pmf* (∀ *j* ∈ *J*. *E*′ *j* (*xs*!*j*))) = (λ*xs*. *return-pmf* (∀ *j*<*n*. *E*′ *j* (*xs*!*j*)))
**unfolding** *E*′*-def* **using** ∗(*1*) **by** *fastforce*
**thus** *?thesis* **by** *presburger*
**qed**
**also have** ... = (∏*j*∈*J*. *A*′ *j*)
**using** *replicate-pmf-indep*′[*OF E*′*A*′ ∗(*1*) *A*′*-notin-J*, *folded rp-def*] **.**
**also have** ... = (∏*j*∈*J*. *prob-space.prob rp* (*?S j*))
**proof**−
**have** ∀ *j* ∈ *J*. *A*′ *j* = *prob-space.prob rp* (*?S j*)
**proof**
**fix** *j* **assume** ∗∗: *j* ∈ *J*
**hence** *A*′ *j* = *A j* **unfolding** *A*′*-def* **by** *presburger*
**also have** ... = *pmf* (*p* ⤜ (λ*x*. *return-pmf* (*E j x*))) *True*
**unfolding** *A-def* **using** ∗(*1*) ∗∗ **by** *auto*
**also have** ... = *pmf* (*rp* ⤜ (λ*xs*. *return-pmf* (*E j* (*xs* ! *j*)))) *True*
**using** *replicate-pmf-indep*″[*of j n p E j*, *folded rp-def*] ∗(*1*) ∗∗ **by** *fastforce*
**also have** ... = *prob-space.prob rp* (*?S j*) **by** (*simp add*: *pmf-of-prop*)

```
    finally show A' j = prob-space.prob rp (?S j) .
  qed
  thus ?thesis by force
 qed
 finally show prob-space.prob rp (⋂j∈J. ?S j) = (∏j∈J. prob-space.prob rp
(?S j)) .
qed
have ∗: ?S‘?I ⊆ prob-space.events rp ∧
   (∀ J⊆?I. J ≠ {}
     ⟶ finite J
     ⟶ prob-space.prob rp (⋂(?S ‘ J)) = (∏j∈J. prob-space.prob rp (?S j)))
  using 1 2 by blast

show ?thesis using prob-space.indep-events-def[OF 0] ∗ by blast
qed

lemma replicate-spmf-same-event-leq:
fixes p :: 'a spmf
fixes n :: nat
fixes E :: 'a option ⇒ bool
assumes spmf (p ⤜ (λx. return-spmf (E x))) True ≤ A
shows spmf (replicate-spmf n p ⤜ (λxs. return-spmf (∀ i < n. E (xs!i)))) True
≤ A^n
by (metis replicate-spmf-same-event assms pmf-nonneg pow-mono)

lemma pmf-of-finite-set-event:
fixes S :: 'a set
fixes p :: 'a pmf
fixes P :: 'a ⇒ bool
defines p ≡ pmf-of-set S
assumes S ≠ {}
assumes finite S
shows pmf (p ⤜ (λt. return-pmf (P t))) True = (card ({t ∈ S. P t})) / card S
proof−
 have ∗: set-pmf p = S
  using assms by auto
 have pmf (p ⤜ (λt. return-pmf (P t))) True = prob-space.prob p ({x ∈ set-pmf
p. P x})
  using pmf-of-prop[of p P] .
 also have ... = measure (measure-pmf (pmf-of-set S)) {x ∈ set-pmf p. P x}
  by (simp add: assms(1))
 also have ... = card ({x ∈ set-pmf p. P x}) / card (set-pmf p)
 proof−
  have S ∩ {x ∈ set-pmf p. P x} = {x ∈ set-pmf p. P x} using ∗ by blast
  thus ?thesis using measure-pmf-of-set[OF assms(2,3)] unfolding ∗ by pres-
burger
 qed
 also have ... = card ({t ∈ S. P t}) / card S using assms by auto
 finally show ?thesis .
```

**qed**

**lemma** *spmf-of-finite-set-event*:
**fixes** $S :: 'a\ set$
**fixes** $p :: 'a\ spmf$
**fixes** $P :: 'a \Rightarrow bool$
**defines** $p \equiv spmf\text{-}of\text{-}set\ S$
**assumes** *finite S*
**shows** $spmf\ (p \gg\!= (\lambda t.\ return\text{-}spmf\ (P\ t)))\ True = (card\ (\{t \in S.\ P\ t\}))\ /\ card\ S$
**proof**−
**have** $spmf\ (p \gg\!= (\lambda t.\ return\text{-}spmf\ (P\ t)))\ True = prob\text{-}space.prob\ p\ (Some`\{x \in set\text{-}spmf\ p.\ P\ x\})$
**using** *spmf-of-prop*[*of p P*] **.**
**also have** ... = $measure\ (measure\text{-}spmf\ (spmf\text{-}of\text{-}set\ S))\ \{x \in set\text{-}spmf\ p.\ P\ x\}$
**by** (*simp add*: *assms*(*1*) *measure-measure-spmf-conv-measure-pmf*)
**also have** ... = $card\ (\{x \in set\text{-}spmf\ p.\ P\ x\})\ /\ card\ (set\text{-}spmf\ p)$
**using** *measure-spmf-of-set*[*of S* $\{x \in set\text{-}spmf\ p.\ P\ x\}$]
**by** (*simp add*: *Collect-conj-eq assms*(*1*,*2*))
**also have** ... = $card\ (\{t \in S.\ P\ t\})\ /\ card\ S$ **using** *assms* **by** *simp*
**finally show** *?thesis* **.**
**qed**

**end**
**theory** *Hidden-Number-Problem*
**imports**
*HOL−Number-Theory.Number-Theory*
*Babai-Correctness-Updated*
*Misc-PMF*

**begin**

**hide-fact** *Finite-Cartesian-Product.mat-def*
**hide-const** *Finite-Cartesian-Product.mat*
**hide-const** *Finite-Cartesian-Product.row*
**hide-fact** *Finite-Cartesian-Product.row-def*
**hide-const** *Determinants.det*
**hide-fact** *Determinants.det-def*
**hide-type** *Finite-Cartesian-Product.vec*
**hide-const** *Finite-Cartesian-Product.vec*
**hide-fact** *Finite-Cartesian-Product.vec-def*
**hide-const** (**open**) *Finite-Cartesian-Product.transpose*
**hide-fact** (**open**) *Finite-Cartesian-Product.transpose-def*
**unbundle** *no inner-syntax*
**unbundle** *no vec-syntax*
**hide-const** (**open**) *Missing-List.span*
**hide-const** (**open**)
*dependent*
*independent*

*real-vector.representation*
*real-vector.subspace*
*span*
*real-vector.extend-basis*
*real-vector.dim*
**hide-const** (**open**) *orthogonal*
**no-notation** *fps-nth* (**infixl** $ *75*)

# 9 General helper lemmas

**lemma** *uminus-sq-norm*:
**fixes** *u v* :: *rat vec*
**assumes** *dim-vec u* = *dim-vec v*
**shows** *sq-norm* (*u* − *v*) = *sq-norm* (*v* − *u*)
**using** *assms*
**proof**−
**have** *u*−*v* = (−*1*) ·$_v$ (*v* − *u*) **using** *assms* **by** *auto*
**then show** *?thesis*
**using** *sq-norm-smult-vec*[*of* −*1 v*−*u*] **by** *auto*
**qed**

**lemma** *smult-sub-distrib-vec*:
**assumes** *v* ∈ *carrier-vec q w* ∈ *carrier-vec q*
**shows** (*a*::′*a*::*ring*) ·$_v$ (*v* − *w*) = *a* ·$_v$ *v* − *a* ·$_v$ *w*
**apply** (*rule eq-vecI*)
**unfolding** *smult-vec-def plus-vec-def*
**using** *assms right-diff-distrib*
**apply** *force*
**by** *auto*

**lemma** *set-list-subset*:
**fixes** *l* :: ′*a list*
**fixes** *S* :: ′*a set*
**assumes** ∀ *i* ∈ {*0*..<*length l*}. *l* ! *i* ∈ *S*
**shows** *set l* ⊆ *S*
**by** (*metis assms atLeastLessThan-iff in-set-conv-nth le0 subset-code*(*1*))

**lemma** *nat-subset-inf*:
**fixes** *A* :: *int set*
**assumes** *A* ⊆ **ℕ**
**assumes** *A* ≠ {}
**shows** *Inf A* ∈ *A*
**proof**−
**let** *?A*′ = *nat*‘*A*
**have** *?A*′ ≠ {} **using** *assms*(*2*) **by** *blast*
**hence** *Inf ?A*′ ∈ *?A*′ **using** *Inf-nat-def1* **by** *presburger*
**then obtain** *z* **where** *z*: *z* ∈ *A* ∧ *nat z* = *Inf ?A*′ **by** *fastforce*
**moreover have** *Inf A* = *z*

```
  proof−
    have ∗: z ≥ 0 using z assms(1) Nats-altdef2 by blast
    have ∀ z′ ∈ A. z ≤ z′
    proof
      fix z′ assume ∗∗: z′ ∈ A
      hence nat z′ ∈ ?A′ by blast
      hence nat z ≤ nat z′ using z wellorder-Inf-le1[of nat z′ ?A′] by argo
      moreover have z′ ≥ 0 using ∗∗ assms(1) Nats-altdef2 by fastforce
      ultimately show z ≤ z′ using ∗ by linarith
    qed
    thus ?thesis using z by (meson cInf-eq-minimum)
  qed
  ultimately show ?thesis by argo
qed

lemma cong-set-subseteq:
  fixes a b m :: 'a :: {unique-euclidean-ring,abs}
  defines S ≡ {|a + z ∗ m| | z. True}
  defines S′ ≡ {|b + z ∗ m| | z. True}
  assumes [a = b] (mod m)
  shows S ⊆ S′
proof
  fix x assume x ∈ S
  then obtain z where z: x = |a + z ∗ m| unfolding S-def by blast
  moreover obtain k where k: a = b + k ∗ m
    by (metis assms(3) cong-iff-lin cong-sym mult-commute-abs)
  ultimately have x = |b + k ∗ m + z ∗ m|
    by presburger
  also have ... = |b + (k + z) ∗ m|
    by (metis Groups.group-cancel.add1 distrib-right)
  also have ... ∈ S′ unfolding S′-def by blast
  finally show x ∈ S′ .
qed

lemma cong-set-eq:
  fixes a b m :: 'a :: {unique-euclidean-ring,abs}
  defines S ≡ {|a + z ∗ m| | z. True}
  defines S′ ≡ {|b + z ∗ m| | z. True}
  assumes [a = b] (mod m)
  shows S = S′
  apply auto[1]
   using cong-set-subseteq[of a b m] assms apply blast
  using cong-set-subseteq[of b a m] assms cong-sym by blast


context vec-module
begin
```

**lemma** *sumlist-distrib*:
**fixes** *ys* :: ('*a*::*comm-ring-1*) *vec list*
**assumes** ⋀*w*. *List.member ys w* ⟹ *dim-vec w* = *n*
**shows** *k* ·$_v$ *sumlist ys* = *sumlist* (*map* (λ*i*. *k* ·$_v$ *i*) *ys*)
**using** *assms*
**proof** (*induct ys*)
**case** *Nil*
**then show** *?case* **by** *auto*
**next**
**case** (*Cons a ys*)
**have** *sumlist-is*: *sumlist* (*a* # *ys*) = *a* + *sumlist ys*
**by** *simp*
**have** *dim-a*: *a* ∈ *carrier-vec n*
**using** *Cons*(*2*)
**by** (*metis carrier-vecI member-rec*(*1*))
**have** *dim-sumlist*: *sumlist ys* ∈ *carrier-vec n*
**using** *Cons*(*2*)
**by** (*metis List.member-def carrier-vec-dim-vec dim-sumlist member-rec*(*1*))
**have** *distrib*: *k* ·$_v$ (*a* + *sumlist ys*) = *k* ·$_v$ *a* + *k* ·$_v$ *sumlist ys*
**using** *smult-add-distrib-vec*[*OF dim-a dim-sumlist*]
**by** *auto*
**have** *ih*: (⋀*w*. *List.member ys w* ⟹ *dim-vec w* = *n*)
**using** *Cons*(*2*)
**by** (*meson member-rec*(*1*))
**show** *?case* **unfolding** *sumlist-is distrib*
**using** *Cons.hyps*[*OF ih*]
**by** *auto*
**qed**

**lemma** *smult-in-lattice-of*:
**fixes** *lattice-basis* :: ('*a*::*comm-ring-1*) *vec list*
**assumes** ⋀*w*. *List.member lattice-basis w* ⟹ *dim-vec w* = *n*
**fixes** *init-vec* :: ('*a*::*comm-ring-1*) *vec*
**fixes** *k* :: '*a*::*comm-ring-1*
**assumes** *init-vec* ∈ *lattice-of lattice-basis*
**shows** *k* ·$_v$ *init-vec* ∈ *lattice-of* ((*map* ((·$_v$) *k*) *lattice-basis*))
**proof** −
**have** *dims*: ⋀*w*. *List.member* (*map* (λ*i*. *of-int* (*c i*) ·$_v$ *lattice-basis* ! *i*)
[*0*..<*length lattice-basis*]) *w* ⟹ *dim-vec w* = *n* **for** *c*
**using** *assms*(*1*)
**by** (*smt* (*verit*) *List.member-def in-set-conv-nth index-smult-vec*(*2*) *length-map map-nth map-nth-eq-conv*)
**obtain** *c* **where** *init-vec* = *sumlist* (*map* (λ*i*. *of-int* (*c i*) ·$_v$ *lattice-basis* ! *i*)
[*0*..<*length lattice-basis*])
**using** *vec-module.in-latticeE*[*OF assms*(*2*)] **by** *blast*
**then have** *k* ·$_v$ *init-vec* = *k* ·$_v$ *sumlist* (*map* (λ*i*. *of-int* (*c i*) ·$_v$ *lattice-basis* ! *i*)
[*0*..<*length lattice-basis*])
**by** *blast*

**moreover have** ... = *sumlist* (*map* (λ*i*. *k* ·$_v$ *i*) ((*map* (λ*i*. *of-int* (*c i*) ·$_v$ *lattice-basis* ! *i*)
[*0*..<*length lattice-basis*])))
**using** *sumlist-distrib dims*
**by** *blast*
**moreover have** ... = *sumlist* (*map* (λ*i*. *k* ·$_v$ (*of-int* (*c i*) ·$_v$ *lattice-basis* ! *i*))
[*0*..<*length lattice-basis*])
**by** (*smt* (*verit*, *best*) *in-set-conv-nth length-map length-upt map-eq-conv map-upt-len-conv nth-map*)
**moreover have** *k-is*: ... = *sumlist* (*map* (λ*i*. *k* ·$_v$ (*of-int* (*c i*) ·$_v$ *lattice-basis* ! *i*))
[*0*..<*length lattice-basis*])
**by** *argo*
**ultimately have** *k-is*: *k* ·$_v$ *init-vec* = *sumlist* (*map* (λ*i*. (*of-int* (*c i*) ·$_v$ (*k* ·$_v$ *lattice-basis* ! *i*)))
[*0*..<*length lattice-basis*])
**by** (*simp add*: *mult-ac*(*2*) *smult-smult-assoc*)
**then have** *k* ·$_v$ *init-vec* =
*sumlist*
(*map* (λ*i*. *of-int* (*c i*) ·$_v$
*map* ((·$_v$) *k*) *lattice-basis* ! *i*)
[*0*..<*length lattice-basis*])
**by** (*smt* (*verit*, *del-insts*) *length-map map-nth map-nth-eq-conv*)
**then have** *mult-vec-is*: *k* ·$_v$ *init-vec* =
*sumlist*
(*map* (λ*i*. *of-int* (*c i*) ·$_v$
*map* ((·$_v$) *k*) *lattice-basis* ! *i*)
[*0*..<*length* (*map* ((·$_v$) *k*) *lattice-basis*)])
**by** *auto*
**then show** *?thesis*
**using** *assms in-latticeI*[*OF mult-vec-is*]
**by** *blast*
**qed**

**end**

**lemma** *filter-distinct-sorted*:
**fixes** *l* :: ('*a*::*linorder*) *list*
**fixes** *A* :: '*a set*
**defines** *P* ≡ (λ*x*. *x* ∈ *A*)
**defines** *n* ≡ *length l*
**assumes** *sorted l*
**assumes** *distinct l*
**assumes** *A* ⊆ *set l*
**shows** *filter P l* = *sorted-list-of-set A*
**using** *assms*
**proof**(*induct l arbitrary*: *n A P*)
**case** *Nil*
**thus** *?case* **by** *force*

**next**
**case** (*Cons a l'*)
**define** $A'$ **where** $A' \equiv A - \{a\}$
**define** $n'$ **where** $n' \equiv$ *length l'*
**define** $P'$ **where** $P' \equiv (\lambda x.\ x \in A')$
**define** $l$ **where** $l \equiv$ *Cons a l'*
**define** $P$ **where** $P \equiv (\lambda x.\ x \in A)$
**have** *1*: *sorted l'* **using** *Cons*(*2*) **by** *simp*
**have** *2*: *distinct l'* **using** *Cons*(*3*) **by** *simp*
**have** *3*: $A' \subseteq$ *set l'* **using** *A'-def Cons*(*4*) **by** *auto*
**have** *ih*: *filter P' l' = sorted-list-of-set A'* **using** *Cons.hyps*(*1*)[*OF 1 2 3*] **unfolding** *P'-def* **.**

**have** *?case* **if** $a \notin A$
**proof**−
**have** *filter P l = filter P' l'*
**unfolding** *l-def filter.simps*(*2*) *P-def P'-def A'-def* **using** *that* **by** *auto*
**moreover have** $A' = A$ **using** *that* **unfolding** *A'-def* **by** *blast*
**ultimately show** *?thesis* **unfolding** *l-def P-def* **using** *ih* **by** *argo*
**qed**
**moreover have** *?case* **if** $a \in A$
**proof**−
**have** $\forall x \in$ *set l'*. *P x = P' x* **using** *Cons*(*3*) **unfolding** *P-def P'-def A'-def* **by** *fastforce*
**hence** $*$: *filter P l = a # (filter P' l')*
**using** *that filter-cong*[*of l' l' P P'*]
**unfolding** *l-def filter.simps*(*2*) *P-def P'-def A'-def*
**by** *presburger*
**have** *1*: *finite A* **using** *Cons*(*4*) *finite-subset* **by** *blast*
**have** *2*: $A \neq \{\}$ **using** *that* **by** *blast*
**hence** *a*: *a = Min A*
**using** *sorted-Cons-Min*[*OF Cons.prems*(*1*)]
**by** (*metis 1 Cons*(*4*) *Lattices-Big.linorder-class.Min.boundedE List.list.simps*(*15*) *Min-antimono Min-eqI finite-set that*)
**show** *?thesis*
**using** *sorted-list-of-set-nonempty*[*OF 1 2*] *ih* $*$ **unfolding** *a A'-def P-def P'-def l-def* **by** *argo*
**qed**
**ultimately show** *?case* **by** *blast*
**qed**

**lemma** *filter-or*:
**fixes** *n a b*
**fixes** *l* :: *nat list*
**defines** $l \equiv [0..<n]$
**defines** $P \equiv (\lambda x.\ x = a \vee x = b)$
**assumes** $a < b$
**assumes** $b <$ *length l*

**shows** *filter P l* = [*a*, *b*]
**proof**−
**have** *1*: *sorted l* **using** *l-def sorted-upt* **by** *blast*
**have** *2*: *distinct l* **using** *distinct-upt l-def* **by** *blast*
**have** *3*: {*a*, *b*} ⊆ *set l* **using** *assms*(*3*,*4*) *l-def length-upt* **by** *auto*
**have** *P*: *P* = (λ*x*. *x* ∈ {*a*, *b*}) **unfolding** *P-def* **by** *simp*
**have** ∗: *sorted-list-of-set* {*a*, *b*} = [*a*, *b*]
**using** *sorted-list-of-set.idem-if-sorted-distinct*[*of* [*a*,*b*]] *assms*(*3*) **by** *force*
**show** *?thesis* **using** *filter-distinct-sorted*[*OF 1 2 3*] **unfolding** *P* ∗ **.**
**qed**

## 9.1 Casting lemmas

**lemma** *int-rat-real-casting-helper*:
**assumes** *a* = *rat-of-int b*
**shows** *real-of-rat a* = *real-of-int b*
**using** *assms* **by** *simp*

**lemma** *casting-expansion-aux*:
**shows** *int-of-rat* (*rat-of-int w1* + *rat-of-int w2*) = *w1* + *w2*
**proof** −
**have** *rat-of-int w1* + *rat-of-int w2* = *rat-of-int* (*w1*+*w2*)
**by** *auto*
**then show** *?thesis*
**using** *int-of-rat* **by** *algebra*
**qed**

**lemma** *casting-expansion*:
**assumes** *dim-vec w1* = *dim-vec w2*
**assumes** ∃ *w2-vec*. *w2* = *map-vec rat-of-int w2-vec*
**shows** *map-vec int-of-rat* ((*map-vec rat-of-int w1*) + *w2*) =
*map-vec int-of-rat* (*map-vec rat-of-int w1*) + (*map-vec int-of-rat w2*)
**proof** −
**have** *vec* (*dim-vec w1*)
(λ*i*. *int-of-rat*
((*vec* (*dim-vec w1*) (λ*i*. *rat-of-int* (*w1* $ *i*)) + *w2*) $ *i*)) $ *idx* =
(*vec* (*dim-vec w1*)
(λ*i*. *int-of-rat* (*vec* (*dim-vec w1*) (λ*i*. *rat-of-int* (*w1* $ *i*)) $ *i*)) +
*vec* (*dim-vec w2*) (λ*i*. *int-of-rat* (*w2* $ *i*))) $ *idx* **if** *idx-lt*: *idx* < *dim-vec w1* **for** *idx*
**proof** −
**have** *vec* (*dim-vec w1*)
(λ*i*. *int-of-rat*
((*vec* (*dim-vec w1*) (λ*i*. *rat-of-int* (*w1* $ *i*)) + *w2*) $ *i*)) $ *idx* =
*int-of-rat* ((*vec* (*dim-vec w1*) (λ*i*. *rat-of-int* (*w1* $ *i*)) + *w2*) $ *idx*)
**using** *idx-lt* **by** *simp*
**have** *vec* (*dim-vec w1*)

```
    (λi. int-of-rat
          ((vec (dim-vec w1) (λi. rat-of-int (w1 $ i)) + w2) $ i)) $
   idx = int-of-rat ((vec (dim-vec w1) (λi. rat-of-int (w1 $ i)) + w2) $ idx)
    using idx-lt by simp
   then have  vec (dim-vec w1)
    (λi. int-of-rat
          ((vec (dim-vec w1) (λi. rat-of-int (w1 $ i)) + w2) $ i)) $
   idx = int-of-rat (rat-of-int (w1 $ idx) + w2$idx)
    using assms idx-lt by auto
   then have vec (dim-vec w1)
    (λi. int-of-rat ((vec (dim-vec w1) (λi. rat-of-int (w1 $ i)) + w2) $ i)) $ idx
    = int-of-rat (rat-of-int (w1 $ idx)) + int-of-rat (w2$idx)
    using assms(1) assms(2) casting-expansion-aux idx-lt by force
   then show ?thesis using assms
    using idx-lt by fastforce
 qed
 then have vec (dim-vec w1)
    (λi. int-of-rat
          ((vec (dim-vec w1) (λi. rat-of-int (w1 $ i)) + w2) $ i)) =
   vec (dim-vec w1)
    (λi. int-of-rat (vec (dim-vec w1) (λi. rat-of-int (w1 $ i)) $ i)) +
   vec (dim-vec w2) (λi. int-of-rat (w2 $ i))
   using assms by auto
 then show ?thesis using assms
   unfolding map-vec-def
   by auto
qed

lemma casting-sum-aux:
 assumes dim-vec k1 = dim-vec k2
 shows (map-vec rat-of-int k1) + (map-vec rat-of-int k2) =
   map-vec rat-of-int (k1 + k2)
 using assms
proof (induct k1 arbitrary: k2)
 case vNil
 then show ?case by auto
next
 case (vCons h1 T1)
 then obtain h2 T2 where k2-is: k2 = vCons h2 T2
   by (metis Nat.nat.simps(3) dim-vec dim-vec-vCons vec-cases)
 then have map-vec rat-of-int T1 + map-vec rat-of-int T2 = map-vec rat-of-int
(T1 + T2)
   using vCons
   by auto
 then show ?case
   unfolding k2-is using vCons(2) k2-is by auto
qed

lemma casting-sum-lemma:
```

**assumes** $\bigwedge$*mem. List.member w mem* $\Longrightarrow$ *dim-vec mem* = *n*
**assumes** $\bigwedge$*mem. List.member w mem* $\Longrightarrow$
($\exists$ *k*::*int vec. map-vec rat-of-int k* = *mem*)
**shows** ($\exists$ *k*::*int vec.* (*abelian-monoid.sumlist* (*module-vec TYPE*(*rat*) *n*) *w*) = *map-vec rat-of-int k*)
**using** *assms*
**proof** (*induct w*)
**interpret** *vec-space TYPE*(*rat*) *n* .
**case** *Nil*
**then show** *?case* **by** (*metis Matrix.of-int-hom.vec-hom-zero sumlist-Nil*)
**next**
**interpret** *vec-space TYPE*(*rat*) *n* .
**case** (*Cons a w*)
**then obtain** *k1* **where** *k1-prop*: *a* = *map-vec rat-of-int k1*
**by** (*metis member-rec*(*1*))
**then obtain** *k2* **where** *sumlist w* = *map-vec rat-of-int k2*
**using** *Cons*
**by** (*meson member-rec*(*1*))
**then have** *sumlist-is*: *sumlist* (*a* # *w*) = (*map-vec rat-of-int k1*) + (*map-vec rat-of-int k2*)
**using** *k1-prop sumlist-Cons* **by** *presburger*
**then have** *sumlist* (*a* # *w*) = (*map-vec rat-of-int* (*k1* + *k2*))
**using** *Cons*(*2*) **using** *casting-sum-aux*
**by** (*metis List.member-def dim-sumlist index-add-vec*(*2*) *index-map-vec*(*2*) *k1-prop member-rec*(*1*))
**then show** *?case*
**by** *blast*
**qed**

**lemma** *casting-lattice-aux*:
**assumes** $\exists$ *k*::*int vec. map-vec rat-of-int k* = *v*
**assumes** *map-vec int-of-rat v* =
*map-vec int-of-rat* (*abelian-monoid.sumlist* (*module-vec TYPE*(*rat*) *n*) *w*)
**assumes** $\bigwedge$*mem. List.member w mem* $\Longrightarrow$
($\exists$ *k*::*int vec. map-vec rat-of-int k* = *mem*)
**assumes** $\bigwedge$*mem. List.member w mem* $\Longrightarrow$ *dim-vec mem* = *n*
**shows** *map-vec int-of-rat v* =
*abelian-monoid.sumlist* (*module-vec TYPE*(*int*) *n*)
(*map* (*map-vec int-of-rat*) *w*)
**proof** −
**interpret** *vec-space TYPE*(*rat*) *n* .
**interpret** *vec-int*: *vec-module TYPE*(*int*) *n* .
**obtain** *k* :: *int vec* **where** *k-prop*: *v* = *map-vec rat-of-int k*
**using** *assms* **by** *auto*
**then have** *map-k*: *map-vec int-of-rat* (*map-vec rat-of-int k*) =
*map-vec int-of-rat* (*sumlist w*)
**using** *assms*(*2*)
**by** *auto*

**have** *map-vec int-of-rat* (*sumlist w*) = *vec-int.M.sumlist* (*map* (*map-vec int-of-rat*) *w*)
**using** *assms*(*3*) *assms*(*4*)
**proof** (*induct w*)
**case** *Nil*
**then show** *?case*
**unfolding** *sumlist-def local.vec-int.M.sumlist-def*
**by** *auto*
**next**
**case** (*Cons a w*)
**have** *foldr* (+) (*map* (*map-vec int-of-rat*) (*a* # *w*)) ($0_v$ *n*) =
(*map-vec int-of-rat a*) + (*foldr* (+) (*map* (*map-vec int-of-rat*) *w*) ($0_v$ *n*))
**by** *simp*
**then have** *h1*: *foldr* (+) (*map* (*map-vec int-of-rat*) (*a* # *w*)) ($0_v$ *n*) =
(*map-vec int-of-rat a*) + *map-vec int-of-rat* (*sumlist w*)
**using** *Cons* **unfolding** *sumlist-def vec-int.M.sumlist-def*
**by** (*simp add*: *member-rec*(*1*))

**obtain** *k* :: *int vec* **where** *k-prop*: *map-vec rat-of-int k* = *a*
**using** *Cons*(*2*)
**by** (*meson member-rec*(*1*))

**then have** *dim-vec-k*: *dim-vec k* = *dim-vec a*
**by** *auto*
**then have** *dim-vec k* = *n*
**by** (*simp add*: *Cons*(*3*) *member-rec*(*1*))

**have** ∃ *w2-vec*. *foldr* (+) *w* ($0_v$ *n*) = *map-vec rat-of-int w2-vec*
**using** *Cons*(*2*) *casting-sum-lemma*
**by** (*metis Cons*(*3*) *member-rec*(*1*) *sumlist-def*)

**then have** *expand*: *map-vec int-of-rat* ((*map-vec rat-of-int k*) + *foldr* (+) *w* ($0_v$ *n*)) =
*map-vec int-of-rat* (*map-vec rat-of-int k*) + *map-vec int-of-rat* (*foldr* (+) *w* ($0_v$ *n*))
**using** *casting-expansion*[*of k foldr* (+) *w* ($0_v$ *n*)] *dim-vec-k*
**by** (*metis List.member-def* ‹*dim-vec k* = *n*› *dim-sumlist index-add-vec*(*2*) *local.Cons.prems*(*2*) *sumlist-Cons sumlist-def*)

**have** *map-vec int-of-rat* (*foldr* (+) (*a* # *w*) ($0_v$ *n*)) =
*map-vec int-of-rat* (*a* + *foldr* (+) *w* ($0_v$ *n*))
**by** *simp*
**then have** *map-vec int-of-rat* (*foldr* (+) (*a* # *w*) ($0_v$ *n*)) = *map-vec int-of-rat* ((*map-vec rat-of-int k*) + *foldr* (+) *w* ($0_v$ *n*))
**using** *k-prop* **by** *auto*

**then have** *map-vec int-of-rat* (*foldr* (+) (*a* # *w*) ($0_v$ *n*)) =
*map-vec int-of-rat* (*map-vec rat-of-int k*) + *map-vec int-of-rat* (*sumlist w*)

```
    using Cons(2) k-prop expand
    using sumlist-def by presburger
  then show ?case
    using h1 k-prop
    unfolding sumlist-def vec-int.M.sumlist-def
    by argo
 qed

 then have map-vec int-of-rat (map-vec rat-of-int k) =
  vec-int.M.sumlist
   (map (map-vec int-of-rat) w)
  using map-k by argo
 then show ?thesis unfolding k-prop by blast
qed

lemma in-lattice-casting:
 assumes ⋀mem. List.member qs mem ⟹ (∃ k::int vec. map-vec rat-of-int k =
mem)
 assumes (⋀mem. List.member qs mem ⟹ dim-vec mem = n)
 assumes v ∈ vec-module.lattice-of n qs
 shows ∃ k. map-vec rat-of-int k = v
proof −
 let ?sumlist = abelian-monoid.sumlist (module-vec TYPE(rat) n)
 obtain c where v-is: v = ?sumlist (map (λi. rat-of-int (c i) ·v qs ! i) [0..<length
qs])
  using assms(3) unfolding vec-module.lattice-of-def by blast
 let ?w = map (λi. rat-of-int (c i) ·v qs ! i) [0..<length qs]
 have dims: (⋀mem. List.member ?w mem ⟹ dim-vec mem = n)
  using assms(2)
   by (smt (verit, del-insts) List.member-def in-set-conv-nth index-smult-vec(2)
length-map map-nth map-nth-eq-conv)
 have mem-w: ∃ k. map-vec rat-of-int k = mem
  if mem-is: List.member (map (λi. rat-of-int (c i) ·v qs ! i) [0..<length qs]) mem
for mem
 proof −
  obtain i where i < length qs mem = rat-of-int (c i) ·v qs ! i
   using mem-is
    by (smt (verit) List.member-def add-0 in-set-conv-nth length-map map-nth
map-nth-eq-conv nth-upt)
  then show ?thesis using assms(1) mem-is
   by (metis Matrix.of-int-hom.vec-hom-smult in-set-conv-nth in-set-member)
 qed
 then obtain k where v = map-vec rat-of-int k
  unfolding v-is
  using casting-sum-lemma[of ?w, OF dims - ]
  by blast
 then show ?thesis
  by auto
qed
```

**lemma** *casting-lattice-lemma-aux2*:
**assumes** $\bigwedge$*mem. List.member qs mem* $\Longrightarrow$
(∃ *k*::*int vec. map-vec rat-of-int k* = *mem*)
**shows** (*map* (*map-vec int-of-rat*) (*map* (λ*i. rat-of-int* (*c i*) ·$_v$ *qs* ! *i*)
[*0*..<*length qs*])) =
(*map* (λ*i. of-int* (*c i*) ·$_v$ *map* (*map-vec int-of-rat*) *qs* ! *i*) [*0*..<*length qs*])
**proof** −
**have** ((*map-vec int-of-rat*) (*rat-of-int* (*c i*) ·$_v$ *qs* ! *i*)) =
*of-int* (*c i*) ·$_v$ (*map-vec int-of-rat*) (*qs* ! *i*) **if** *i-lt*: *i* < *length qs* **for** *i*
**proof** −
**obtain** *k* :: *int vec* **where** *qs-i-is*: *qs* ! *i* = *map-vec rat-of-int k*
**using** *assms i-lt*
**by** (*metis List.member-def in-set-conv-nth*)
**have** *map-vec rat-of-int* ((*c i*) ·$_v$ *k*) = (*rat-of-int* (*c i*) ·$_v$ *qs* ! *i*)
**using** *qs-i-is* **by** *force*
**then have** *lhs-is*: (*map-vec int-of-rat* (*rat-of-int* (*c i*) ·$_v$ *qs* ! *i*)) =
(*c i*) ·$_v$ *k*
**using** *qs-i-is*
**by** (*metis eq-vecI index-map-vec*(*1*) *index-map-vec*(*2*) *int-of-rat*(*1*))
**have** *map-vec int-of-rat* (*map-vec rat-of-int k*) = *k*
**by** *force*
**then have** *rhs-is*: *of-int* (*c i*) ·$_v$ *map-vec int-of-rat* (*qs* ! *i*) = (*c i*) ·$_v$ *k*
**unfolding** *qs-i-is* **by** *simp*
**then show** *?thesis* **using** *lhs-is rhs-is* **by** *argo*
**qed**
**then show** *?thesis* **by** *auto*
**qed**

**lemma** *casting-lattice-lemma*:
**fixes** *v* :: *rat vec*
**fixes** *qs* :: *rat vec list*
**assumes** ∃ *k*::*int vec. map-vec rat-of-int k* = *v*
**assumes** $\bigwedge$*mem. List.member qs mem* $\Longrightarrow$
(∃ *k*::*int vec. map-vec rat-of-int k* = *mem*)
**assumes** *dim-vecs*: $\bigwedge$*mem. List.member qs mem* $\Longrightarrow$ *dim-vec mem* = *n*
**assumes** *v* ∈ *vec-module.lattice-of n qs*
**shows** *map-vec int-of-rat v*
∈ *vec-module.lattice-of n* (*map* (*map-vec int-of-rat*) *qs*)
**proof** −
**let** *?sumlist* = *abelian-monoid.sumlist* (*module-vec TYPE*(*rat*) *n*)
**let** *?int-sumlist* = *abelian-monoid.sumlist* (*module-vec TYPE*(*int*) *n*)
**obtain** *c* :: *nat* ⇒ *int* **where** *v* = *?sumlist* (*map* (λ*i. rat-of-int* (*c i*) ·$_v$ *qs* ! *i*)
[*0*..<*length qs*])
**using** *assms* **unfolding** *vec-module.lattice-of-def* **by** *blast*
**then have** *map-is*: *map-vec int-of-rat v*
= *map-vec int-of-rat* (*?sumlist* (*map* (λ*i. rat-of-int* (*c i*) ·$_v$ *qs* ! *i*) [*0*..<*length qs*]))

**by** *blast*
**have** *length-qs*: *length qs* = *length* (*map* (*map-vec int-of-rat*) *qs*)
**by** *simp*
**then have** *map-vec-v*: *map-vec int-of-rat v*
= *map-vec int-of-rat*
(*?sumlist* (*map* (λ*i*. *rat-of-int* (*c i*) ·$_v$ *qs* ! *i*) [*0*..<*length* (*map* (*map-vec int-of-rat*) *qs*)]))
**using** *map-is* **by** *argo*

**let** *?w* = (*?sumlist* (*map* (λ*i*. *rat-of-int* (*c i*) ·$_v$ *qs* ! *i*) [*0*..<*length* (*map* (*map-vec int-of-rat*) *qs*)]))

**have** *dim-vecs*: (⋀*mem*. *List.member*
(*map* (λ*i*. *rat-of-int* (*c i*) ·$_v$ *qs* ! *i*)
[*0*..<*length* (*map* (*map-vec int-of-rat*) *qs*)])
*mem* ⟹
*dim-vec mem* = *n*)
**using** *dim-vecs*
**by** (*smt* (*verit*, *ccfv-threshold*) *List.member-def in-set-conv-nth index-smult-vec*(*2*) *length-map map-nth map-nth-eq-conv*)

**have** *casting-mem*: ∃*k*. *map-vec rat-of-int k* = *mem* **if** *mem-assm*: *List.member*
(*map* (λ*i*. *rat-of-int* (*c i*) ·$_v$ *qs* ! *i*)
[*0*..<*length* (*map* (*map-vec int-of-rat*) *qs*)]) *mem* **for** *mem*
**proof** −
**obtain** *i* **where** *i-prop*: *mem* = *rat-of-int* (*c i*) ·$_v$ *qs* ! *i*
*i* < *length* (*map* (*map-vec int-of-rat*) *qs*)
**using** *mem-assm*
**by** (*smt* (*verit*, *ccfv-SIG*) *List.member-def arith-simps*(*49*) *in-set-conv-nth length-map map-nth map-nth-eq-conv nth-upt*)
**obtain** *k1* **where** *map-vec rat-of-int k1* = *qs* ! *i*
**using** *i-prop*(*2*) *assms*(*2*)
**by** (*metis List.member-def* ‹*length qs* = *length* (*map* (*map-vec int-of-rat*) *qs*)› *in-set-conv-nth*)
**then show** *?thesis* **using** *i-prop*(*1*)
**by** (*metis Matrix.of-int-hom.vec-hom-smult*)
**qed**
**have** *map-vec-v*: *map-vec int-of-rat v* =
(*?int-sumlist*
(*map* (*map-vec int-of-rat*) (*map* (λ*i*. *rat-of-int* (*c i*) ·$_v$ *qs* ! *i*) [*0*..<*length* (*map* (*map-vec int-of-rat*) *qs*)])))
**using** *casting-lattice-aux*[*OF assms*(*1*) *map-vec-v casting-mem dim-vecs*]
**by** *blast*
**have** *map-vec int-of-rat v* = *?int-sumlist*
(*map* (λ*i*. *of-int* (*c i*) ·$_v$ *map* (*map-vec int-of-rat*) *qs* ! *i*)
[*0*..<*length* (*map* (*map-vec int-of-rat*) *qs*)])
**unfolding** *map-vec-v* **using** *casting-lattice-lemma-aux2 length-qs*
**by** (*metis* (*no-types*, *lifting*) *assms*(*2*))
**then show** *?thesis* **unfolding** *vec-module.lattice-of-def* **by** *blast*

**qed**

# 10 HNP adversary locale

**locale** *hnp-adversary* =
  **fixes** *d p* :: *nat*
**begin**

**type-synonym** *adversary* = (*nat* × *nat*) *list* ⇒ *nat*

**definition** *int-gen-basis* :: *nat list* ⇒ *int vec list* **where**
  *int-gen-basis ts* = *map* (λ*i*. (*p*^2 ·$_v$ (*unit-vec* (*d*+1) *i*))) [0..<*d*]
    @ [*vec-of-list* ((*map* (λ*x*. (*of-nat p*) ∗ *x*) *ts*) @ [1])]

**fun** *int-to-nat-residue* :: *int* ⇒ *nat* ⇒ *nat*
  **where** *int-to-nat-residue I modulus* = *nat* (*I mod modulus*)

**definition** *ts-from-pairs* :: (*nat* × *nat*) *list* ⇒ *nat list* **where**
  *ts-from-pairs pairs* = *map fst pairs*

**definition** *scaled-uvec-from-pairs* :: (*nat* × *nat*) *list* ⇒ *int vec* **where**
  *scaled-uvec-from-pairs pairs* = *vec-of-list* ((*map* (λ(*a*,*b*). *p*∗*b*) *pairs*) @ [0])

**fun** $\mathcal{A}$*-vec* :: (*nat* × *nat*) *list* ⇒ *int vec* **where**
  $\mathcal{A}$*-vec pairs* = (*full-babai*
    (*int-gen-basis* (*ts-from-pairs pairs*))
    (*map-vec rat-of-int* (*scaled-uvec-from-pairs pairs*))
    (4/3))

**fun** $\mathcal{A}$ :: *adversary* **where**
  $\mathcal{A}$ *pairs* = *int-to-nat-residue* (($\mathcal{A}$*-vec pairs*)$*d*) *p*

**end**

# 11 HNP locales

### 11.0.1 Arithmetic locale

**locale** *hnp-arith* =
  **fixes** *n* α *d p k* :: *nat*
  **assumes** *p*: *prime p*
  **assumes** α: α ∈ {1..<*p*}
  **assumes** *d*: *d* = 2 ∗ *ceiling* (*sqrt n*)
  **assumes** *n*: *n* = *ceiling* (*log* 2 *p*)
  **assumes** *k*: *k* = *ceiling* (*sqrt n*) + *ceiling* (*log* 2 *n*)
  **assumes** *n-big*: 961 < *n*
**begin**

**definition** μ :: *nat* **where**

```
  μ ≡ nat (ceiling (1/2 * (sqrt n)) + 3)

lemma μ: μ = (ceiling (1/2 * (sqrt n)) + 3)
proof−
  have ceiling (1/2 * (sqrt n)) + 3 ≥ 0
  proof−
    have sqrt n ≥ 0 by auto
    thus ?thesis by linarith
  qed
  thus ?thesis unfolding μ-def by presburger
qed

lemma int-p-prime: prime (int p) by (simp add: p)

lemma p-geq-2: p ≥ 2 using p prime-ge-2-nat by blast

lemma n-geq-1: n ≥ 1
proof−
  have log 2 p > 0
   by (smt (verit) p less-imp-of-nat-less of-nat-1 prime-gt-1-nat zero-less-log-cancel-iff)
  thus ?thesis using n by linarith
qed

lemma μ-le-k:
  shows μ + 1 ≤ k
proof−
  have 144 ≤ n using n-big by simp
  then have sqrt(144) ≤ sqrt n using numeral-le-real-of-nat-iff real-sqrt-le-iff by
blast
  then have 4 + sqrt (n) / 2 ≤ (sqrt n) − 2 by auto
  then have ceiling(sqrt (n) / 2) + 3 ≤ ceiling (sqrt n) − 1 by linarith
  moreover have 0 ≤ log 2 (n) using n-big by auto
  ultimately show ?thesis using μ k by linarith
qed

lemma k-plus-1-lt:
  shows k + 1 < log 2 p
proof −
  obtain r::real where r-eq: r = log 2 (real p) r ≤ n r > n−1
   by (smt (verit, best) n Multiseries-Expansion.intyness-simps(1) add-diff-inverse-nat
linorder-not-less n-geq-1 nat-ceiling-le-eq nat-int of-nat-less-iff of-nat-simps(2))

  then have p-eq: p = 2 powr r
    using n log-powr-cancel[of 2 r]
    by (simp add: p prime-gt-0-nat)

  then have p-gt: p > n
    using n n-big
     by (smt (verit, del-insts) One-nat-def Suc-pred r-eq(1,3) le-neq-implies-less
```

*le-simps(3) less-exp log2-of-power-le nat-le-real-less p prime-gt-0-nat*)

**have** *real n = ⌈log 2 (real p)⌉*
**using** *n* **by** *linarith*
**then have** *int k = ⌈sqrt ⌈log 2 (real p)⌉⌉ + ⌈log 2 ⌈log 2 (real p)⌉⌉*
**using** *k* **by** *auto*
**then have** *k-plus-1-eq*: *k + 1 = ⌈sqrt ⌈log 2 (real p)⌉⌉ + ⌈log 2 ⌈log 2 (real p)⌉⌉ + 1*
**by** *linarith*
**let** *?logp = ⌈log 2 (real p)⌉*
**have** *log-p-bound*: *?logp > 100*
**using** *p-gt n-big p-eq r-eq*
**by** *linarith*

**have** *geq*: *(log 2 (real p)) / 2 > (⌈log 2 (real p)⌉ − 1) / 2*
**using** *log-p-bound*
**by** (*smt* (*verit*) *divide-strict-right-mono less-ceiling-iff*)
**have** *arb*: *real-of-int ⌈sqrt (real-of-int a)⌉*
*< real-of-int (a − 1) / 2* **if** *a-gt*: *a > 100*
**for** *a*::*int*
**using** *a-gt*
**proof** −
**have** *eq*: *(a − 3) / 2 = (a − 1) / 2 − 1*
**using** *a-gt* **by** *auto*
**have** *10∗a < a∗a*
**using** *a-gt* **by** *simp*
**then have** *10∗a < a^2*
**by** (*simp add*: *power2-eq-square*)
**then have** *10∗a < a^2 + 9*
**by** *auto*
**then have** *4∗a < a^2 − 6∗a + 9*
**by** *simp*
**then have** *4∗a < (a − 3)^2*
**using** *a-gt power2-sum*[*of a −3*] **by** *simp*
**then have** *sqrt (4∗a) < sqrt ((a − 3)^2)*
**using** *real-sqrt-less-iff* **by** *presburger*
**then have** *sqrt (4∗a) < a − 3*
**using** *a-gt* **by** *simp*
**then have** *2∗(sqrt a) < a − 3*
**by** (*simp add*: *real-sqrt-mult*)
**then have** *sqrt a < (a − 3) / 2*
**by** *simp*
**then have** *sqrt a < (a − 1) / 2 − 1*
**using** *eq*
**by** *algebra*
**then have** *sqrt (real-of-int a) + 1*
*< real-of-int (a − 1) / 2*
**using** *a-gt* **by** *argo*
**then show** *?thesis*

```
    by linarith
  qed
  then have real-of-int ⌈sqrt (real-of-int ⌈log 2 (real p)⌉)⌉
    < (⌈log 2 (real p)⌉ − 1) / 2
    using arb[OF log-p-bound]
    by blast
  then have ineq1: ⌈sqrt (real-of-int ?logp)⌉ < (log 2 (real p))/2
    using geq by linarith

  have ineq2: ⌈log 2 (real-of-int ?logp)⌉ + 1 ≤ (log 2 (real p))/2
  proof −
    have  99 < log 2 (real p)
      using log-p-bound by simp
    then have p-gt: 2^99 < p
       by (metis Nat.bot-nat-0.extremum le-trans linorder-not-le log2-of-power-le
of-nat-numeral p-gt)
    obtain q::nat where q-prop: 2^q ≤ p ∧ 2^(q+1) > p
      using ex-power-ivl1 le-refl p prime-ge-1-nat by presburger
    then have q-geq: q ≥ 99 using p-gt
    by (smt (verit) less-log2-of-power log2-of-power-less nat-le-real-less numeral-plus-one
of-nat-add of-nat-numeral p prime-gt-0-nat)
    have leq-q: ⌈log 2 (real p)⌉ ≤ q+1
      using q-prop
     by (metis ceiling-mono ceiling-of-nat less-le log2-of-power-le p prime-gt-0-nat)

    have arith-help: 98 < q ⟹ 32 * q + 48 < 2 ^ q for q
    proof (induct q)
      case 0
      then show ?case by auto
    next
      case (Suc q)
      {assume ∗:  99 = Suc q
        then have ?case
          by simp
      } moreover {assume ∗:  99 < Suc q
        then have 32 * q + 48 < 2 ^ q
          using Suc by linarith
        then have ?case
          using Suc by auto
      }
      ultimately show ?case
        using Suc by linarith
    qed
    have (q+1)^2*16 < 2^q
      using q-geq
    proof (induct q)
      case 0
      then show ?case by auto
    next
```

**case** (*Suc q*)
**{ assume** ∗: *99 = Suc q*
**then have** *?case*
**by** *simp*
**}**
**moreover {**
**assume** ∗: *99 < Suc q*
**then have** *q-plus*: $(q + 1)^2 * 16 < 2 \hat{} q$
**using** *Suc* **by** *auto*
**have** $(Suc\ q + 1)^2 = q\hat{}2 + 4*q + 4$
**by** (*smt* (*verit*) *Groups.ab-semigroup-add-class.add.commute Groups.semigroup-add-class.add.assoc nat-1-add-1 numeral-Bit0-eq-double plus-1-eq-Suc power2-eq-square power2-sum*)
**then have** $(Suc\ q + 1)^2 = (q+1)\hat{}2 + 2*q + 3$
**by** (*metis* (*no-types*, *lifting*) *add-2-eq-Suc′ add-Suc-right more-arith-simps*(6) *mult-2 numeral-Bit1 numeral-One one-power2 plus-1-eq-Suc power2-sum semiring-norm*(163))
**then have** *suc-q*: $(Suc\ q + 1)^2 * 16 = 16*(q+1)\hat{}2 + 32*q + 3*16$
**by** *simp*
**have** $32*q + 48 < 2\hat{}q$
**using** ∗ *arith-help* **by** *auto*
**then have** $16*(q+1)\hat{}2 + 32*q + 3*16 < 2 * 2\hat{}q$
**using** *q-plus* **by** *linarith*
**then have** $(Suc\ q + 1)^2 * 16 < 2 * 2\hat{}q$
**using** *suc-q* **by** *argo*
**then have** *?case*
**by** *auto*
**}**
**ultimately show** *?case*
**using** *Suc* **by** *linarith*
**qed**
**then have** (*real-of-int* (*q*+*1*)) *powr* 2∗16 ≤ $2\hat{}q$
**proof** −
**have** *0 < q + 1*
**by** *simp*
**then have** *real-of-int* (*int* (*q + 1*)) *powr real-of-int* (*int 2*) ∗ *real-of-int* (*int 16*) ≤ *real-of-int* (*int 2*) $\hat{}$ *q*
**by** (*metis* (*no-types*) *Power.semiring-1-class.of-nat-power* ‹$(q + 1)^2 * 16 < 2 \hat{} q$› *less-le of-int-of-nat-eq of-nat-0-less-iff of-nat-less-numeral-power-cancel-iff of-nat-numeral of-nat-simps*(5) *powr-realpow*)
**then show** *?thesis*
**by** *simp*
**qed**
**then have** (*real-of-int* (*q*+*1*)) *powr* 2∗16 ≤ (*real p*)
**using** *q-prop*
**using** *numeral-power-le-of-nat-cancel-iff order-trans-rules*(23) **by** *blast*
**then have** (*real-of-int ?logp*) *powr* 2∗16 ≤ (*real p*)
**using** *leq-q*
**by** (*smt* (*verit*) *ceiling-le-iff p powr-less-mono2 prime-ge-1-nat real-of-nat-ge-one-iff zero-le-log-cancel-iff*)

```
    then have (2 powr (log 2 (real-of-int ?logp))) powr 2*16 ≤ (real p)
      using log-p-bound by fastforce
    then have 2 powr (log 2 (real-of-int ?logp)*2)*16 ≤ (real p)
      using powr-powr[of 2] by auto
    then have 2 powr (2*log 2 (real-of-int ?logp))*16 ≤ (real p)
      by argo
    then have 2 powr (2*log 2 (real-of-int ?logp) + 4) ≤ (real p)
      by (simp add: powr-add)
    then have 2*log 2 (real-of-int ?logp) + 4 ≤ (log 2 (real p))
      using le-log-iff p-gt by auto
    then show ?thesis using log-p-bound
    by (smt (verit, ccfv-threshold) ceiling-add-one field-sum-of-halves of-int-ceiling-le-add-one)
  qed

  have ⌈sqrt (real-of-int ⌈log 2 (real p)⌉)⌉ + ⌈log 2 (real-of-int ⌈log 2 (real p)⌉)⌉
+ 1
      < log 2 (real p)
    using ineq1 ineq2 by linarith
  then show ?thesis using k-plus-1-eq by linarith
qed

lemma p-k-le-p-mu:
  shows p/(2^k)≤p/(2^(μ + 1))
proof−
  have real(2) ^(μ + 1) ≤ 2^k using μ-le-k power-increasing[of μ + 1 k real 2]
by force
  then show ?thesis using divide-left-mono[of real(2) ^(μ + 1) 2^k p] by simp
qed

lemma k-geq-1: k ≥ 1 using n-geq-1 k μ-le-k by linarith

lemma final-ineq:
   (((4::real)/3) powr ((d + 1)/2) * (11/10) * (d + 1) * (p / (2 powr (real k −
1)))
    < p / (2 powr μ))
proof−
  let ?crn = ceiling (sqrt n)
  let ?chrn = ceiling (1/2 * sqrt n)
  have 31^2 < n using n-big
    by simp
  then have sn: 31 < sqrt n using real-less-rsqrt by auto
  have pos1: 0 ≤ (2 * 3 / 4 * 2 powr (− 1 / 2)) powr real-of-int ⌈sqrt (real n)⌉
by auto
  have pos2: 0 < 2 powr ?crn using sn by simp
  have pos3: 0 < (3/4) powr ?crn  by simp
  have (2::real) powr (− 1 / 2) = 1/(2 powr (1/2))
    using powr-minus[of 2::real 1/2]
    by (simp add: powr-minus-divide)
  then have powr-2-is: 2 powr (− 1 / 2) = 1 / (sqrt 2)
```

**by** (*metis powr-half-sqrt rel-simps*(27))
**have** (2∗ *sqrt* 2)^2 < 3^2
**by** *simp*
**then have** 2∗ *sqrt* 2 < 3
**using** *real-less-rsqrt* **by** *fastforce*
**then have** *gt-1*: 3/2 ∗ 1 / (*sqrt* 2) > 1
**by** *simp*
**have** *one-half*: (1/*sqrt* 2) *powr* 2 = 1/2
**by** (*smt* (*verit*, *best*) *eq-divide-eq powr-2-is powr-half-sqrt-powr powr-powr real-sqrt-divide real-sqrt-eq-iff real-sqrt-one*)
**have** *three-halves*: ((3::*real*)/2) ^ 2 = (9/4::*real*)
**using** *power-divide*[*of* 3::*real* 2 2] **by** *simp*
**have** ((3::*real*)/2 ∗ 1 / (*sqrt* 2)) *powr* 2 = (3/2) ^ 2 ∗ ((1/*sqrt* 2) *powr* 2)
**using** *powr-mult*[*of* 3/2 1 / (*sqrt* 2)]
**by** *auto*
**then have** *mult*: ((3::*real*)/2 ∗ 1 / (*sqrt* 2)) *powr* 2 = 9/8
**by** (*simp add*: *one-half three-halves*)
**have** (5::*real*) < (9^15/8^15)
**by** *auto*
**then have** (5::*real*) < (9/8) ^ 15
**by** (*metis power-divide*)
**then have** (5::*real*) < (9/8) *powr* 15
**by** *simp*
**then have** (5::*real*) < ((3/2 ∗ 1 / (*sqrt* 2)) *powr* 2) *powr* 15
**using** *mult* **by** *presburger*
**then have** (5::*real*) < (3/2 ∗ 1 / (*sqrt* 2)) *powr* 30
**by** *auto*
**then have** (5::*real*) < (3/2 ∗ 1 / (*sqrt* 2)) *powr* 31
**using** *gt-1*
**by** (*smt* (*verit*) *powr-less-cancel-iff*)
**then have** (5::*real*) < (2 ∗ 3/4 ∗ 2 *powr* (− 1/2) ) *powr* 31
**using** *powr-2-is* **by** *auto*
**moreover have** (31::*real*) < ?*crn* **using** *sn* **by** *simp*
**moreover have** 1 < ((2::*real*) ∗ 3/4 ∗ 2 *powr* (− 1/2) )
**using** *gt-1 powr-2-is* **by** *linarith*
**ultimately have** 1: (5::*real*) < (2 ∗ 3/4 ∗ 2 *powr* (− 1/2) ) *powr* ?*crn*
**using** *powr-less-mono*[*of* 31 ?*crn* ((2::*real*) ∗ 3/4 ∗ 2 *powr* (− 1/2) )] **by** *linarith*
**have** *mult-eq*: (32::*real*) ∗ (11/10) ∗ 3 / 5 = 1056/50
**by** *simp*
**have** ((4::*real*)/3) ∗ 1056^2 < 1550^2
**by** *auto*
**then have** ((4::*real*)/3) *powr* (1/2) ∗ 1056 < 1550
**by** (*metis divide-nonneg-nonneg powr-half-sqrt real-sqrt-less-mono real-sqrt-mult real-sqrt-pow2 real-sqrt-power zero-le-numeral*)
**then have** ((4::*real*)/3) *powr* (1/2) ∗ (32::*real*) ∗ (11/10) ∗ 3 / 5 < 31
**using** *mult-eq* **by** *linarith*
**then have** ((4::*real*)/3) *powr* (1/2) ∗ 32 ∗ (11/10) ∗ 3 / 5 < *sqrt n*
**using** *sn* **by** *simp*

**then have** $((4::real)/3)$ *powr* $(1/2) * 32 * (11/10) * 3 < sqrt\ n * 5$ **by** *linarith*
**then have** $((4::real)/3)$ *powr* $(1/2) * 32 * (11/10) * 3 < sqrt\ n * (2 * 3/4 * 2\ powr\ (-\ 1/2)\ )\ powr\ ?crn$
**using** *1 mult-strict-left-mono*[*of 5* $(2 * 3\ /\ 4 * 2\ powr\ (-\ 1\ /\ 2))\ powr$ *real-of-int* $\lceil sqrt\ (real\ n) \rceil\ sqrt\ n$] **by** *fastforce*
**then have** $((4::real)/3)$ *powr* $(1/2) * 32 * (11/10) * 3\ /\ sqrt\ n < (2 * 3/4 * 2\ powr\ (-\ 1/2)\ )\ powr\ ?crn$
**using** *sn divide-less-eq*[*of* $(4\ /\ 3)\ powr\ (1\ /\ 2) * 32 * (11\ /\ 10) * 3\ sqrt\ n\ (2 * 3\ /\ 4 * 2\ powr\ (-\ 1\ /\ 2))\ powr$ *real-of-int* $\lceil sqrt\ (real\ n) \rceil$] **by** *argo*
**then have** $((4::real)/3)$ *powr* $(1/2) * 32 * (11/10) * (3\ /\ sqrt\ n) < (2 * 3/4 * 2\ powr\ (-\ 1/2)\ )\ powr\ ?crn$ **by** *auto*
**moreover have** $3\ /\ sqrt\ n = 3 * sqrt\ n\ /\ n$
**proof**−
**have** $3\ /\ sqrt\ n = 3\ /\ (sqrt\ n) * (sqrt\ n\ /\ sqrt\ n)$ **using** *sn* **by** *force*
**also have** $\ldots = 3 * sqrt\ n\ /\ (sqrt\ n * sqrt\ n)$ **using** *sn* **by** *argo*
**finally show** *?thesis* **by** *auto*
**qed**
**ultimately have** $((4::real)/3)$ *powr* $(1/2) * 32 * (11/10) * (3 * sqrt\ n\ /\ n) < (2 * 3/4 * 2\ powr\ (-\ 1/2)\ )\ powr\ ?crn$ **by** *simp*
**moreover have** $(2 * 3/4 * 2\ powr\ (-\ 1/2))\ powr\ ?crn = (2\ powr\ ?crn) * ((3/4)\ powr\ ?crn) * ((2\ powr\ (-\ 1/2))\ powr\ ?crn)$
**by** (*metis times-divide-eq-right powr-mult*)
**ultimately have** $((4::real)/3)$ *powr* $(1/2) * 32 * (11/10) * (3 * sqrt\ n\ /\ n) < (2\ powr\ ?crn) * ((3/4)\ powr\ ?crn) * ((2\ powr\ (-\ 1/2))\ powr\ ?crn)$ **by** *linarith*
**then have** $((4::real)/3)$ *powr* $(1/2) * 32 * (11/10) * (3 * sqrt\ n\ /\ n) < ((2\ powr\ (-\ 1/2))\ powr\ ?crn) * (\ (3\ /4\ )\ powr\ ?crn) * (2\ powr\ ?crn)$ **by** *argo*
**then have** $((4::real)/3)$ *powr* $(1/2) * 32 * (11/10) * (3 * sqrt\ n\ /\ n)\ /\ (2\ powr\ ?crn) < ((2\ powr\ (-\ 1/2))\ powr\ ?crn) * (\ (3/4)\ powr\ ?crn)$
**using** *pos2 pos-divide-less-eq*[*of 2 powr real-of-int* $\lceil sqrt\ (real\ n) \rceil\ ((4::real)/3)\ powr\ (1/2) * 32 * (11/10) * (3 * sqrt\ n\ /\ n)\ ((2\ powr\ (-\ 1/2))\ powr\ ?crn) * (\ (3\ /4\ )\ powr\ ?crn)$] **by** *linarith*
**then have** $((4::real)/3)$ *powr* $(1/2) * 32 * (11/10) * (3 * sqrt\ n\ /\ n)\ /\ (2\ powr\ ?crn)\ /\ (\ (3/4)\ powr\ ?crn) < \ ((2\ powr\ (-\ 1/2))\ powr\ ?crn)$
**using** *pos3 pos-divide-less-eq*[*of* $((3/4)\ powr\ ?crn)\ ((4::real)/3)\ powr\ (1/2) * 32 * (11/10) * (3 * sqrt\ n\ /\ n)\ /\ (2\ powr\ ?crn)\ ((2\ powr\ (-\ 1/2))\ powr\ ?crn)$] **by** *linarith*
**then have** *main*: $(\ ((4::real)/3)\ powr\ (1/2)\ /\ (\ (3\ /4\ )\ powr\ ?crn)\ ) *$
$(16 * 11/10) *$
$(\ (2 * 3 * sqrt\ n\ /\ n)\ /\ (2\ powr\ ?crn)\ )$
$< ((2\ powr\ (-\ 1/2))\ powr\ ?crn)$ **by** *argo*

**have** *k-ineq*: $(d\ +\ 1)\ /\ (2\ powr\ (\ real\ k\ -\ 1\ )) \leq (\ (2 * 3 * sqrt\ n\ /\ n)\ /\ (2\ powr\ ?crn)\ )$
**proof**−
**have** $d \leq 3 * sqrt\ n$ **using** *sn d* **by** *linarith*
**have** $n = 2\ powr\ log\ 2\ n$
**using** *n-geq-1* **by** *auto*
**moreover have** $2\ powr\ log\ 2\ n \leq 2\ powr\ (ceiling\ (log\ 2\ n))$ **by** *fastforce*
**ultimately have** $2\ /\ (2\ powr\ (ceiling\ (log\ 2\ n))\ ) \leq 2\ /\ n$

**by** (*metis frac-le ge-refl powr-nonneg-iff verit-comp-simplify*(*7*) *verit-comp-simplify1*(*3*) *verit-eq-simplify*(*5*))
**moreover have** ( (*2 * 3 * sqrt n / n*) / (*2 powr ?crn*) ) = *3 * sqrt n * (2 / n / 2 powr ?crn*) **by** *argo*
**moreover have** *0 ≤ 3 * sqrt n* **using** *sn* **by** *linarith*
**ultimately have** *: *3 * sqrt n * (2 / (2 powr (ceiling (log 2 n))) / 2 powr ?crn) ≤ (2 * 3 * sqrt n / n) / (2 powr ?crn)*
**using** *mult-left-mono*[*of 2 / (2 powr (ceiling (log 2 n) ) ) 2 / n 3 * sqrt (real n)*]
*divide-right-mono*[*of 3 * sqrt n * (2 / (2 powr (ceiling (log 2 n)))) (2 * 3 * sqrt n / n) 2 powr ?crn*]
*pos2* **by** *simp*
**have** *2 / (2 powr (ceiling (log 2 n) ) ) = 2 powr ( 1 − ceiling (log 2 n))* **using** *powr-diff*[*of 2 1 ceiling (log 2 n)*] **by** *fastforce*
**then have** *(2 / (2 powr (ceiling (log 2 n))) / 2 powr ?crn) = 2 powr (1 − ceiling (log 2 n) − ?crn)*
**using** *powr-diff*[*of 2 1 − ceiling (log 2 n) ?crn*] **by** *fastforce*
**moreover have** *− (1 − ceiling (log 2 n) − ?crn) = ?crn + ceiling (log 2 n) − 1* **by** *fastforce*
**ultimately have** *(2 / (2 powr (ceiling (log 2 n))) / 2 powr ?crn) = 1 / (2 powr (?crn + ceiling (log 2 n) − 1))*
**by** (*smt* (*verit*) *of-int-minus powr-minus-divide*)
**moreover have** *real-of-int (?crn + ceiling (log 2 n) − 1) = real-of-int (?crn + ceiling (log 2 n)) − 1* **by** *linarith*
**ultimately have** *(2 / (2 powr (ceiling (log 2 n))) / 2 powr ?crn) = 1 / (2 powr (real-of-int (int k) − 1))*
**using** *k* **by** *presburger*
**then have** *3 * sqrt n / (2 powr (real k − 1)) ≤ (2 * 3 * sqrt n / n) / (2 powr ?crn)* **using** *** **by** *force*
**moreover have** *(d + 1) ≤ 3 * sqrt n*
**using** *d sn* **by** *linarith*
**moreover have** *0 < 2 powr (k − 1)* **by** *force*
**ultimately show** *?thesis*
**using** *divide-right-mono*[*of d + 1 3 * sqrt n 2 powr (real k − 1)*]
**by** *auto*
**qed**

**have** ( ((*4*::*real*)/*3*) *powr* (*1*/*2*) / ( (*3* /*4* ) *powr ?crn*) ) = (*4*/*3*) *powr* ((*d* + *1*) / *2*)
**proof**−
**have** *inv*: *1*/((*4*::*real*)/*3*) = (*3*::*real*) / *4*
**by** *force*
**have** (*4* / *3*) *powr real-of-int* (− ⌈*sqrt* (*real n*)⌉) = *1* / (*4* / *3*) *powr real-of-int* ⌈*sqrt* (*real n*)⌉
**using** *powr-minus-divide*[*of 4*/*3 ?crn*]
**by** *simp*
**then have** (*4* / *3*) *powr real-of-int* (− ⌈*sqrt* (*real n*)⌉) = *1 powr real-of-int* ⌈*sqrt* (*real n*)⌉ / (*4* / *3*) *powr real-of-int* ⌈*sqrt* (*real n*)⌉
**by** *simp*

```
    then have powr-minus-div: (4 / 3) powr real-of-int (− ⌈sqrt (real n)⌉) = (1
/ (4 / 3)) powr real-of-int ⌈sqrt (real n)⌉
      by (simp add: powr-divide)
    have (3 /4) powr ?crn = (4/3) powr (− ?crn)
      unfolding powr-minus-div inv
      by auto
    then have ((4::real)/3) powr (1/2) / ( (3 /4 ) powr ?crn) = ((4::real) / 3)
powr (1/2 − (− ?crn))
      using powr-diff[of 4/3 1/2 −?crn] by algebra
    moreover have 1/2 − (− ?crn) = (d + 1)/2 using d by force
    ultimately show ?thesis by presburger
  qed
  then have  (4/3) powr ((d + 1) / 2) *
            (16 * 11/10) *
            ( (2 * 3 * sqrt n / n) / (2 powr ?crn) )
          < ((2 powr (− 1/2)) powr ?crn) using main
    by presburger
  moreover have 0<(4/3) powr ((d + 1) / 2) *
            (16 * 11/10) by fastforce
  ultimately have (4/3) powr ((d + 1) / 2) * (16 * 11/10) * (d + 1) / (2 powr
( real k − 1 )) < (2 powr (− 1/2)) powr ?crn
    using mult-left-mono[of real (d + 1) / 2 powr (real k − 1) 2 * 3 * sqrt (real
n) / real n / 2 powr real-of-int ⌈sqrt (real n)⌉ (4 / 3) powr (real (d + 1) / 2) *
(16 * 11 / 10)]
        k-ineq by argo
  then have (4/3) powr ((d + 1) / 2)  * (11/10) * (d + 1) / (2 powr ( real k
− 1 )) < (2 powr (− 1/2)) powr ?crn / 16 by linarith
  also have ... = 2 powr (− 1/2 * ?crn) / 16
    by (simp add: powr-powr)
  also have ... = 2 powr (− 1/2 * ?crn − 4)
    using powr-diff[of 2 −1/2*?crn 4] by force
  also have ... ≤ 2 powr (−μ) using powr-mono[of − 1/2 * ?crn − 4 −μ 2::real]
μ by linarith
  also have ... = 1 / 2 powr (μ)
    by (simp add: powr-minus-divide)
  finally have (4/3) powr ((d + 1) / 2)  * (11/10) * (d + 1) / (2 powr ( real k
− 1 )) < 1 / 2 powr (μ)
    by meson
  then have (4/3) powr ((d + 1) / 2)  * (11/10) * (d + 1) / (2 powr ( real k
− 1 )) * p <  1 / 2 powr (μ) * p
    using p mult-strict-right-mono[of (4/3) powr ((d + 1) / 2)  * (11/10) * (d
+ 1) / (2 powr ( real k − 1 )) 1 / 2 powr (μ) p]
    by fastforce
  then show ?thesis
    by force
qed

end
```

## 11.1 Main HNP locale

**locale** *hnp* = *hnp-arith n α d p k* + *hnp-adversary d p* **for** *n α d p k* +
  **fixes** *msb-p* :: *nat* ⇒ *nat*
  **assumes** *msb-p-dist*: ⋀*x*. |*int x* − *int* (*msb-p x*)| < *p* / (2^*k*)
**begin**

## 11.2 Uniqueness lemma

**sublocale** *vec-space TYPE*(*rat*) *d* + *1* .

**lemma** *sumlist-index-commute*:
  **fixes** *Lst* :: *rat vec list*
  **fixes** *i* :: *nat*
  **assumes** *set Lst* ⊆ *carrier-vec* (*d* + *1*)
  **assumes** *i* < (*d* + *1*)
  **shows** (*sumlist Lst*)$*i* = *sum-list* (*map* (λ*j*. (*Lst*!*j*)$*i*) [*0*..<(*length Lst*)])
  **using** *assms vec-module.sumlist-vec-index*
  **by** (*smt* (*verit*, *ccfv-SIG*) *map-eq-conv map-upt-len-conv subset-code*(*1*))

**definition** *ts-pmf* **where** *ts-pmf* = *replicate-pmf d* (*pmf-of-set* {*1*..<*p*})

**lemma** *set-pmf-ts*: *set-pmf ts-pmf* = {*l*. *length l* = *d* ∧ (∀ *i* < *d*. *l*!*i* ∈ {*1*..<*p*})}
**proof**−
  **have** *set-pmf ts-pmf* = {*xs* ∈ *lists* (*set-pmf* (*pmf-of-set* {*1*..<*p*})). *length xs* = *d*}
    **unfolding** *ts-pmf-def* **using** *set-replicate-pmf*[*of d pmf-of-set* {*1*..<*p*}] .
  **also have** ... = {*l*. *length l* = *d* ∧ (∀ *i* < *d*. *l*!*i* ∈ {*1*..<*p*})}
    **apply** *safe*
    **apply**(*metis One-nat-def α empty-iff finite-atLeastLessThan nth-mem set-pmf-of-set*)
    **by** (*metis empty-iff finite-atLeastLessThan in-set-conv-nth set-pmf-of-set*)
  **finally show** *?thesis* .
**qed**

**definition** *ts-to-as* :: *nat list* ⇒ *nat list* **where**
  *ts-to-as ts* = (*map* (*msb-p* ∘ (λ*t*. (*α*∗*t*) *mod p*)) *ts*)

**definition** *ts-to-u* :: *nat list* ⇒ *rat vec* **where**
  *ts-to-u ts* = *vec-of-list* (*map of-nat* (*ts-to-as ts*) @ [*0*])

**lemma** *ts-to-u-alt*:
  *ts-to-u ts* = *vec-of-list* ((*map* (*of-nat* ∘ *msb-p* ∘ (λ*t*. (*α*∗*t*) *mod p*)) *ts*) @ [*0*])
  **unfolding** *ts-to-u-def ts-to-as-def* **by** (*metis map-map*)

**lemma** *u-carrier*: *length ts* = *d* ⟹ *ts-to-u ts* ∈ *carrier-vec* (*d* + *1*)
  **by** (*simp add*: *carrier-dim-vec ts-to-as-def ts-to-u-def*)

**lemma** *ts-to-u-carrier*:
  **fixes** *ts* :: *nat list*
  **shows** (*ts-to-u ts*) ∈ *carrier-vec* ((*length ts*) + *1*)

**proof** −
**have** *dim-vec* (*vec-of-list* (*map* (*rat-of-nat* ∘ *msb-p* ∘ (λ*t*. *α* ∗ *t mod p*)) *ts* @ [*0*]))

= (*length ts* + *1*)
**by** *simp*
**then show** *?thesis*
**unfolding** *ts-to-u-def ts-to-as-def carrier-vec-def* **by** *fastforce*
**qed**

### 11.2.1 Lattice construction and lemmas

**definition** *p-vecs* :: *rat vec list* **where**
*p-vecs* = *map* (λ*i*. *of-int-hom.vec-hom* ((*of-nat p*) ·$_v$ (*unit-vec* (*d*+*1*) *i*))) [*0*..<*d*]

**lemma** *length-p-vecs*: *length p-vecs* = *d* **unfolding** *p-vecs-def* **by** *auto*

**lemma** *p-vecs-carrier*: ∀ *v* ∈ *set p-vecs*. *dim-vec v* = *d* + *1* **unfolding** *p-vecs-def* **by** *force*

**lemma** *lincomb-of-p-vecs-last*: (*lincomb-list* (*of-int* ∘ *cs*) *p-vecs*)$*d* = *0* (**is** *?lhs* = *0*)
**proof**−
**let** *?xs* = (*map* (λ*i*. (*rat-of-int* ∘ *cs*) *i* ·$_v$ *p-vecs* ! *i*) [*0*..<*length p-vecs*])
**have** *dim*: ∀ *v* ∈ *set ?xs*. *dim-vec v* = *d* + *1* **using** *p-vecs-carrier* **by** *simp*
**have** ∗: ∀ *v* ∈ *set ?xs*. *v*$*d* = *0* **unfolding** *p-vecs-def* **by** *fastforce*
**have** *?lhs* = *sumlist* (*map* (λ*i*. (*rat-of-int* ∘ *cs*) *i* ·$_v$ *p-vecs* ! *i*) [*0*..<*length p-vecs*])$*d*
**unfolding** *lincomb-list-def* **by** *blast*
**also have** ... = (∑ *j* = *0*..<*length ?xs*. *?xs* ! *j* $ *d*)
**using** *sumlist-nth*[*OF dim*, *of d*] **by** *linarith*
**finally show** *?thesis* **using** ∗ **by** *force*
**qed**

**definition** *gen-basis* :: *nat list* ⇒ *rat vec list* **where**
*gen-basis ts* = *p-vecs* @ [*vec-of-list* ((*map of-nat ts*) @ [*1* / (*of-nat p*)])]

**lemma** *gen-basis-length*: *length* (*gen-basis ts*) = *d* + *1* **unfolding** *gen-basis-def p-vecs-def* **by** *force*

**lemma** *gen-basis-units*:
**assumes** *i* < *d*
**assumes** *j* < *d* + *1*
**shows** ((*gen-basis ts*)!*i*)$*j* = *of-nat* (*if i* = *j then p else 0*) (**is** (*?x*)$*j* = -)
**proof**−
**have** *?x* = *of-int-hom.vec-hom* ((*of-nat p*) ·$_v$ (*unit-vec* (*d*+*1*) *i*))
**proof**−
**have** *?x* = (*map* (λ*i*. *of-int-hom.vec-hom* ((*of-nat p*) ·$_v$ (*unit-vec* (*d*+*1*) *i*))) [*0*..<*d*])!*i*
**unfolding** *gen-basis-def p-vecs-def* **using** *assms*(*1*) **by** (*simp add*: *nth-append*)

```
  also have ... = of-int-hom.vec-hom ((of-nat p) ·v (unit-vec (d+1) i)) by (simp
add: assms(1))
  finally show ?thesis .
 qed
 thus ?thesis using assms by auto
qed

definition int-gen-lattice :: nat list ⇒ int vec set where
 int-gen-lattice ts = vec-module.lattice-of (d + 1) (int-gen-basis ts)

definition gen-lattice :: nat list ⇒ rat vec set where
 gen-lattice ts = vec-module.lattice-of (d + 1) (gen-basis ts)

definition close-vec :: nat list ⇒ rat vec ⇒ bool where
 close-vec ts v ⟷ (sq-norm ((ts-to-u ts) − v) < ((of-nat p) / 2^μ)^2)

definition good-vec :: rat vec ⇒ bool where
 good-vec v ⟷ dim-vec v = d + 1 ∧ (∃ β::int. [α = β] (mod p) ∧ of-rat (v$d)
= β/p)

definition good-lattice :: nat list ⇒ bool where
 good-lattice ts ⟷ (∀ v ∈ gen-lattice ts. close-vec ts v ⟶ good-vec v)

definition bad-lattice :: nat list ⇒ bool where
 bad-lattice ts ⟷ ¬ good-lattice ts

definition sampled-lattice-good :: bool pmf where
 sampled-lattice-good = do {
  ts ← ts-pmf;
  return-pmf (good-lattice ts)
 }

interpretation vec-int: vec-module TYPE(int) d + 1 .

lemma int-gen-basis-carrier:
 fixes ts :: nat list
 assumes length ts = d
 shows set (int-gen-basis ts) ⊆ carrier-vec (d + 1)
proof −
 have first-part: ⋀i. int (p^2) ·v unit-vec (d + 1) i ∈ carrier-vec (d + 1)
  unfolding unit-vec-def by simp
 have second-part: (vec-of-list (map ((∗) (int p)) (map int ts) @ [1])) ∈ carrier-vec
(d+1)
  by (simp add: assms carrier-dim-vec)
 show ?thesis
  unfolding int-gen-basis-def using first-part second-part
  by auto
qed
```

**lemma** *int-gen-lattice-carrier*:
**fixes** *ts* :: *nat list*
**assumes** *length ts* = *d*
**shows** *int-gen-lattice ts* ⊆ *carrier-vec* (*d* + *1*)
**proof** −
**have** *set* (*int-gen-basis ts*) ⊆ *carrier-vec* (*d* + *1*)
**using** *int-gen-basis-carrier*[*OF assms*(*1*)] **unfolding** *int-gen-basis-def*
**by** *blast*
**then show** *?thesis*
**unfolding** *int-gen-lattice-def*
**using** *lattice-of-as-mat-mult* **by** *blast*
**qed**

**lemma** *gen-basis-vecs-carrier*:
**fixes** *ts* :: *nat list*
**fixes** *i* :: *nat*
**assumes** *length ts* = *d*
**assumes** *i* ∈ {*0*..< *d* + *1*}
**shows** (*gen-basis ts*) ! *i* ∈ *carrier-vec* (*d* + *1*)
**proof** (*cases i*=*d*)
**case** *t*:*True*
**have** *length* (*gen-basis ts*) = *d* + *1* **unfolding** *gen-basis-def p-vecs-def* **by** *auto*
**then have** *1*: (*gen-basis ts*) ! *i* = *vec-of-list* ((*map of-nat ts*) @ [*1* / (*of-nat p*)])
**unfolding** *gen-basis-def* **using** *t* **by** (*simp add*: *append-Cons-nth-middle*)
**have** *length* ((*map of-nat ts*) @ [*1* / (*of-nat p*)]) = *d* + *1* **using** *assms* **by** *fastforce*
**then have** *vec-of-list* ((*map of-nat ts*) @ [*1* / (*of-nat p*)]) ∈ *carrier-vec* (*d* + *1*)
**by** (*metis vec-of-list-carrier*)
**then show** *?thesis* **using** *1* **by** *algebra*
**next**
**case** *False*
**then have** *less*: *i*<*d* **using** *assms* **by** *simp*
**have** *length* (*gen-basis ts*) = *d* + *1* **unfolding** *gen-basis-def p-vecs-def* **by** *auto*
**then have** (*gen-basis ts*) ! *i* = *of-int-hom.vec-hom* ((*of-nat p*) ·$_v$ (*unit-vec* (*d*+*1*) *i*))
**using** *less* **by** (*simp add*: *gen-basis-def nth-append p-vecs-def*)
**then show** *?thesis* **by** *fastforce*
**qed**

**lemma** *gen-basis-carrier*:
**fixes** *ts* :: *nat list*
**assumes** *length ts* = *d*
**shows** *set* (*gen-basis ts*) ⊆ *carrier-vec* (*d* + *1*)
**proof**−
**have** *length* (*gen-basis ts*) = *d* + *1* **unfolding** *gen-basis-def p-vecs-def* **by** *auto*
**then have** (*gen-basis ts*) ! *i* ∈ *carrier-vec* (*d* + *1*) **if** *in-range*: *i* ∈ {*0*..<(*length* (*gen-basis ts*))} **for** *i*
**using** *gen-basis-vecs-carrier*[*of ts i*] *in-range assms* **by** *argo*
**then show** *?thesis* **using** *set-list-subset*[*of gen-basis ts carrier-vec* (*d* + *1*)] **by**

*presburger*
**qed**

**lemma** *gen-lattice-carrier*:
**fixes** *ts* :: *nat list*
**assumes** *length ts* = *d*
**shows** *gen-lattice ts* ⊆ *carrier-vec* (*d* + *1*)
**proof** −
**have** *set-basis*: *set* (*gen-basis ts*) ⊆ *carrier-vec* (*d* + *1*)
**using** *gen-basis-carrier*[*of ts*] *assms* **unfolding** *gen-lattice-def*
**by** *auto*
**then have** *dim-vec v* = *d*+*1* **if** *v-in*: *v* ∈ *lattice-of* (*gen-basis ts*) **for** *v*
**proof** −
**obtain** *c* **where** *v-is*: *v* = *sumlist* (*map* (λ*i*. *rat-of-int* (*c i*) ·$_v$ *gen-basis ts* ! *i*) [*0*..<*length* (*gen-basis ts*)])
**using** *v-in* **unfolding** *lattice-of-def* **by** *blast*
**have** *all-x*: ∀ *x*∈*set* (*map* (λ*i*. *rat-of-int* (*c i*) ·$_v$ *gen-basis ts* ! *i*) [*0*..<*length* (*gen-basis ts*)]).
*dim-vec x* = *d* + *1*
**using** *set-basis* **unfolding** *carrier-vec-def*
**using** *assms gen-basis-length gen-basis-vecs-carrier* **by** *auto*
**then show** *?thesis*
**using** *v-is carrier-dim-vec dim-sumlist*[*OF all-x*]
**by** *argo*
**qed**
**then show** *?thesis* **unfolding** *gen-lattice-def* **using** *carrier-dim-vec* **by** *blast*
**qed**

**lemma** *sampled-lattice-good-map-pmf*: *sampled-lattice-good* = *map-pmf good-lattice ts-pmf*
**by** (*simp add*: *sampled-lattice-good-def map-pmf-def*)

**lemma** *coordinates-of-gen-lattice*:
**fixes** *ts* :: *nat list*
**fixes** *c* :: *nat* ⇒ *int*
**fixes** *i* :: *nat*
**assumes** *i* ≤ *length ts*
**assumes** *length ts* = *d*
**shows** (*sumlist* (*map* (λ*i*. *of-int* (*c i*) ·$_v$ ((*gen-basis ts*) ! *i*)) [*0* ..< *length* (*gen-basis ts*)]))$*i*
= (*if* (*i* = *d*) *then* (*rat-of-int* (*c d*) / *rat-of-nat p*) *else rat-of-int* ((*c d*) ∗ *ts*!*i* + (*c i*)∗*p*))
**proof**(*cases i*=*d*)
**case** *t*:*True*
**define** *Lst* **where** *Lst* = (*map* (λ*i*. *of-int* (*c i*) ·$_v$ ((*gen-basis ts*) ! *i*)) [*0* ..< *length* (*gen-basis ts*)])
**have** ⋀*x*. *x* ∈ *set Lst* ⟹ *x* ∈ *carrier-vec* (*d* + *1*)
**proof**−

**fix** $x$ **assume** $x \in set\ Lst$
**then obtain** $y$ **where** $y$: $Lst!y = x \wedge y \in \{0..<length(Lst)\}$
**by** (*meson atLeastLessThan-iff find-first-le nth-find-first zero-le*)
**then have** $x = of\text{-}int\ (c\ y) \cdot_v ((gen\text{-}basis\ ts)\ !\ y)$ **unfolding** *Lst-def* **by** *auto*
**moreover have** $length(Lst) = d + 1$ **unfolding** *Lst-def* **using** *gen-basis-length*[*of ts*] **by** *force*
**ultimately show** $x \in carrier\text{-}vec\ (d + 1)$ **using** *gen-basis-vecs-carrier*[*of ts*] *assms y* **by** *auto*
**qed**
**then have** $(sumlist\ Lst)\$i = sum\text{-}list\ (map\ (\lambda l.\ l\$i)\ Lst)$
**using** *sumlist-vec-index*[*of Lst i*] *gen-basis-carrier*[*of ts*] *assms* **by** *force*
**moreover have** $sum\text{-}list\ (map\ (\lambda l.\ l\$i)\ Lst)$
$= sum\text{-}list\ (map\ ($
$(\lambda l.\ l\$i) \circ (\lambda i.\ of\text{-}int\ (c\ i) \cdot_v ((gen\text{-}basis\ ts)\ !\ i))$
$)$
$[0\ ..<\ length\ (gen\text{-}basis\ ts)])$
**unfolding** *Lst-def* **by** *auto*
**moreover have** $\bigwedge j.\ j \in (set\ [0\ ..<\ length\ (gen\text{-}basis\ ts)])$
$\Longrightarrow \neg\ (j = i \vee j = d)$
$\Longrightarrow\ ((\lambda l.\ l\$i) \circ (\lambda i.\ of\text{-}int\ (c\ i) \cdot_v ((gen\text{-}basis\ ts)\ !\ i)))\ j = 0$
**proof**−
**fix** $j$ **assume** *j1*: $j \in (set\ [0\ ..<\ length\ (gen\text{-}basis\ ts)])$ **and** *j2*: $\neg\ (j = i \vee j = d)$
**have** *j3*: $j < d \wedge j \neq i$ **using** *j2 j1 gen-basis-length*[*of ts*] **by** *force*
**have** $((\lambda l.\ l\$i) \circ (\lambda i.\ of\text{-}int\ (c\ i) \cdot_v ((gen\text{-}basis\ ts)\ !\ i)))\ j =$
$(of\text{-}int\ (c\ j) \cdot_v ((gen\text{-}basis\ ts)\ !\ j))\$i$ **by** *simp*
**also have** $(of\text{-}int\ (c\ j) \cdot_v ((gen\text{-}basis\ ts)\ !\ j))\$i = (of\text{-}int\ (c\ j) * ((gen\text{-}basis\ ts)\ !\ j)\$i)$
**using** *gen-basis-vecs-carrier*[*of ts j*] *j1 assms gen-basis-length*[*of ts*] **by** *fastforce*
**also have** $((gen\text{-}basis\ ts)\ !\ j)\$i = 0$
**using** *gen-basis-units*[*of j i ts*] *j3 assms* **by** *fastforce*
**finally show** $((\lambda l.\ l\$i) \circ (\lambda i.\ of\text{-}int\ (c\ i) \cdot_v ((gen\text{-}basis\ ts)\ !\ i)))\ j = 0$ **by** *fastforce*
**qed**
**ultimately have** $(sumlist\ Lst)\$i$
$= sum\text{-}list\ (map$
$((\lambda l.\ l\$i) \circ (\lambda i.\ of\text{-}int\ (c\ i) \cdot_v ((gen\text{-}basis\ ts)\ !\ i)))$
$(filter\ (\lambda j.\ j{=}i \vee j = d)$
$[0\ ..<\ length\ (gen\text{-}basis\ ts)]))$
**using** *sum-list-map-filter*[*of* $[0..<length\ (gen\text{-}basis\ ts)]$ $(\lambda j.\ j{=}i \vee j = d)$
$(\lambda l.\ l\$i) \circ (\lambda i.\ of\text{-}int\ (c\ i) \cdot_v ((gen\text{-}basis\ ts)\ !\ i))$] **by** *argo*
**moreover have** $(filter\ (\lambda j.\ j{=}i \vee j = d)\ [0\ ..<\ length\ (gen\text{-}basis\ ts)]) = [d]$
**using** *t gen-basis-length*[*of ts*] **by** *fastforce*
**ultimately have** $(sumlist\ Lst)\$i = (of\text{-}int\ (c\ d) \cdot_v ((gen\text{-}basis\ ts)\ !\ d))\$d$ **using** *t* **by** *force*
**also have** $(of\text{-}int\ (c\ d) \cdot_v ((gen\text{-}basis\ ts)\ !\ d))\$d = (of\text{-}int\ (c\ d)) * (gen\text{-}basis\ ts)!d\$d$
**using** *gen-basis-vecs-carrier*[*of ts d*] *assms* **by** *simp*
**also have** $(gen\text{-}basis\ ts)!d\$d = 1\ /\ (rat\text{-}of\text{-}nat\ p)$

**proof**−
**have** (*gen-basis ts*)!*d* = *vec-of-list* ((*map of-nat ts*) @ [*1* / (*of-nat p*)])
**using** *gen-basis-length*[*of ts*] **unfolding** *gen-basis-def*
**by** (*simp add*: *append-Cons-nth-middle*)
**also have** (*vec-of-list* ((*map of-nat ts*) @ [*1* / (*of-nat p*)]))$*d* = ((*map of-nat ts*) @ [*1* / (*of-nat p*)])!*d*
**by** *auto*
**also have** ((*map of-nat ts*) @ [*1* / (*of-nat p*)])!*d* = *1* / (*of-nat p*)
**using** *assms append-Cons-nth-middle*[*of d* (*map of-nat ts*) *1* / (*of-nat p*) []]
**by** *force*
**finally show** *?thesis* **by** *fastforce*
**qed**
**finally show** *?thesis* **using** *t* **unfolding** *Lst-def* **by** *simp*
**next**
**case** *f*:*False*
**define** *Lst* **where** *Lst* = (*map* (λ*i*. *of-int* (*c i*) ·$_v$ ((*gen-basis ts*) ! *i*)) [*0* ..< *length* (*gen-basis ts*)])
**have** ⋀*x*. *x* ∈ *set Lst* ⟹ *x* ∈ *carrier-vec* (*d* + *1*)
**proof**−
**fix** *x* **assume** *x* ∈ *set Lst*
**then obtain** *y* **where** *y-def*: *x* = *Lst*!*y* ∧ *y*∈{*0*..<*length*(*Lst*)}
**by** (*metis atLeastLessThan-iff in-set-conv-nth zero-le*)
**then have** *x* = *of-int* (*c y*) ·$_v$ ((*gen-basis ts*) ! *y*)
**using** *Lst-def* **by** *force*
**moreover have** *y*∈{*0*..<*d*+*1*}
**using** *y-def gen-basis-length*[*of ts*] **unfolding** *Lst-def* **by** *simp*
**ultimately show** *x* ∈ *carrier-vec* (*d* + *1*)
**using** *gen-basis-vecs-carrier*[*of ts y*] *assms* **by** *fastforce*
**qed**
**then have** (*sumlist Lst*)$*i* = *sum-list* (*map* (λ*l*. *l*$*i*) *Lst*)
**using** *sumlist-vec-index*[*of Lst i*] *gen-basis-carrier*[*of ts*] *assms* **by** *force*
**moreover have** *sum-list* (*map* (λ*l*. *l*$*i*) *Lst*)
= *sum-list* (*map* (
(λ*l*. *l*$*i*) ∘ (λ*i*. *of-int* (*c i*) ·$_v$ ((*gen-basis ts*) ! *i*))
)
[*0* ..< *length* (*gen-basis ts*)])
**unfolding** *Lst-def* **by** *auto*
**moreover have** ⋀*j*. *j* ∈ (*set* [*0* ..< *length* (*gen-basis ts*)])
⟹ ¬ (*j* = *i* ∨ *j* = *d*)
⟹ ((λ*l*. *l*$*i*) ∘ (λ*i*. *of-int* (*c i*) ·$_v$ ((*gen-basis ts*) ! *i*))) *j* = *0*
**proof**−
**fix** *j* **assume** *j1*: *j* ∈ (*set* [*0* ..< *length* (*gen-basis ts*)]) **and** *j2*: ¬ (*j* = *i* ∨ *j* = *d*)
**have** *j3*: *j* < *d* ∧ *j* ≠ *i* **using** *j2 j1 gen-basis-length*[*of ts*] **by** *force*
**have** ((λ*l*. *l*$*i*) ∘ (λ*i*. *of-int* (*c i*) ·$_v$ ((*gen-basis ts*) ! *i*))) *j* =
(*of-int* (*c j*) ·$_v$ ((*gen-basis ts*) ! *j*))$*i* **by** *simp*
**also have** (*of-int* (*c j*) ·$_v$ ((*gen-basis ts*) ! *j*))$*i* = (*of-int* (*c j*) ∗ ((*gen-basis ts*) ! *j*)$*i*)
**using** *gen-basis-vecs-carrier*[*of ts j*] *j1 assms gen-basis-length*[*of ts*] **by** *fastforce*

```
    also have ((gen-basis ts) ! j)$i = 0
      using gen-basis-units[of j i ts] j3 assms by fastforce
     finally show ((λl. l$i) ∘ (λi. of-int (c i) ·v ((gen-basis ts) ! i))) j = 0 by
fastforce
  qed
  ultimately have (sumlist Lst)$i
                = sum-list (map
                        ((λl. l$i) ∘ (λi. of-int (c i) ·v ((gen-basis ts) ! i)))
                        (filter (λj. j=i ∨ j = d)
                            [0 ..< length (gen-basis ts)]))
    using sum-list-map-filter[of [0..<length (gen-basis ts)] (λj. j=i ∨ j = d)
                          (λl. l$i) ∘ (λi. of-int (c i) ·v ((gen-basis ts) ! i))] by argo
  moreover have (filter (λj. j = i ∨ j = d) [0 ..< length (gen-basis ts)]) = [i, d]
    using filter-or[of i d length (gen-basis ts)] assms gen-basis-length[of ts] f
    by fastforce
  ultimately have (sumlist Lst)$i =
                  ( ((λl. l$i) ∘ (λi. of-int (c i) ·v ((gen-basis ts) ! i))) i) +
                  ( ((λl. l$i) ∘ (λi. of-int (c i) ·v ((gen-basis ts) ! i))) d) by fastforce
  also have ( ((λl. l$i) ∘ (λi. of-int (c i) ·v ((gen-basis ts) ! i))) i) = (of-int (c i)
∗ ((gen-basis ts) ! i)$i)
    using gen-basis-vecs-carrier[of ts i] assms gen-basis-length[of ts] by force
  also have ((gen-basis ts) ! i)$i = rat-of-nat p
    using gen-basis-units[of i i ts] assms f by force
  also have ( ((λl. l$i) ∘ (λi. of-int (c i) ·v ((gen-basis ts) ! i))) d) = (of-int (c
d) ∗ ((gen-basis ts) ! d)$i)
    using gen-basis-vecs-carrier[of ts d] assms gen-basis-length[of ts] by force
  also have ((gen-basis ts) ! d)$i = rat-of-nat (ts!i)
  proof−
    have (gen-basis ts)!d = vec-of-list ((map of-nat ts) @ [1 / (of-nat p)])
      using gen-basis-length[of ts] unfolding gen-basis-def
      by (simp add: append-Cons-nth-middle)
     also have (vec-of-list ((map of-nat ts) @ [1 / (of-nat p)]))$i = ((map of-nat
ts) @ [1 / (of-nat p)])!i
      by auto
    also have ((map of-nat ts) @ [1 / (of-nat p)])!i = rat-of-nat (ts!i)
      using assms f append-Cons-nth-left[of i (map of-nat ts) 1 / (of-nat p) []]
      by force
    finally show ?thesis by fastforce
  qed
  finally have (sumlist Lst)$i = (of-int (c i)) ∗ (rat-of-nat p) + (rat-of-int (c d))
∗ (rat-of-nat (ts!i))
    by blast
  then show ?thesis using f unfolding Lst-def by force
qed

lemma gen-lattice-int-gen-lattice-vec:
  fixes scaled-v :: int vec
  fixes ts :: nat list
  assumes length ts = d
```

**assumes** (*scaled-v* ∈ *int-gen-lattice ts*)
**shows** ((*1*/(*of-nat p*)) ·$_v$ (*map-vec rat-of-int scaled-v*) ∈ (*gen-lattice ts*))
**proof**−
**let** *?iL* = *int-gen-lattice ts*
**let** *?ib* = *int-gen-basis ts*
**let** *?L* = *gen-lattice ts*
**let** *?b* = *gen-basis ts*
**have** *il*: *length ?ib* = *d* + *1*
**unfolding** *int-gen-basis-def* **by** *fastforce*
**obtain** *c* **where** *c*: *scaled-v* = *vec-int.lincomb-list* (*of-int* ∘ *c*) *?ib*
**using** *int-gen-basis-carrier*[*of ts*] *assms* **unfolding** *int-gen-lattice-def vec-int.lincomb-list-def vec-int.lattice-of-def* **by** *auto*
**let** *?v* = *lincomb-list* (*of-int* ∘ *c*) *?b*
**have** *carr-v*: *?v* ∈ *carrier-vec* (*d* + *1*)
**using** *lincomb-list-carrier gen-basis-carrier assms* **by** *blast*
**have** *carr-sv*: (*1*/(*of-nat p*)) ·$_v$ (*map-vec rat-of-int scaled-v*) ∈ *carrier-vec* (*d* + *1*)
**using** *assms int-gen-lattice-carrier*[*of ts*] **by** *auto*
**have** *carr-iv*: *scaled-v* ∈ *carrier-vec* (*d* + *1*)
**using** *int-gen-lattice-carrier*[*of ts*] *assms* **by** *fast*
**have** *?v* ∈ *?L*
**unfolding** *lincomb-list-def gen-lattice-def lattice-of-def* **by** *simp*
**moreover have** *?v* = (*1*/(*of-nat p*)) ·$_v$ (*map-vec rat-of-int scaled-v*)
**proof**−
**have** *?v*$*i* = ((*1*/(*of-nat p*)) ·$_v$ (*map-vec rat-of-int scaled-v*))$*i* **if** *in-range*: *i*∈{*0*..<(*d* + *1*)} **for** *i*
**proof**−
**let** *?Lst* = (*map* (λ*i*. (*of-int* ∘ *c*) *i* ·$_v$ *int-gen-basis ts* ! *i*) [*0*..<*length* (*int-gen-basis ts*)])
**have** *length* (*int-gen-basis ts*) = *d* + *1* **unfolding** *int-gen-basis-def* **by** *force*
**then have** *set ?Lst* ⊆ *carrier-vec* (*d* + *1*)
**using** *int-gen-basis-carrier*[*of ts*] *assms*(*1*) *gen-basis-vecs-carrier length-map map-eq-conv map-nth set-list-subset smult-closed*
**by** *fastforce*
**then have** *scaled-v* $ *i* = *sum-list* (*map* (λ*l*. *l*$*i*) *?Lst*)
**using** *vec-int.sumlist-vec-index*[*of ?Lst i*] *in-range c*
**unfolding** *vec-int.lincomb-list-def* **by** *auto*
**then have** ∗: *scaled-v* $ *i* = *sum-list* ((*map* ((λ*l*. *l*$*i*) ∘ (λ*j*. (*of-int* ∘ *c*) *j* ·$_v$ (*?ib* ! *j*) ) )) [*0*..<(*d* + *1*)])
**using** *il* **by** *simp*
**show** *?thesis*
**proof**(*cases i* = *d*)
**case** *t*:*True*
**have** [*0*..<(*d* + *1*)] = [*0*..<*d*] @ [*d*] **by** *simp*
**then have** *sv*: *scaled-v* $ *i* = *sum-list* ((*map* ((λ*l*. *l*$*i*) ∘ (λ*j*. (*of-int* ∘ *c*) *j* ·$_v$ (*?ib* ! *j*) ) )) [*0*..<(*d*)]) + ((λ*l*. *l*$*i*) ∘ (λ*j*. (*of-int* ∘ *c*) *j* ·$_v$ (*?ib* ! *j*) ) ) *d*
**using** ∗ **by** *fastforce*
**have** *x* = *0* **if** *x*: *x* ∈ *set* ((*map* ((λ*l*. *l*$*i*) ∘ (λ*j*. (*of-int* ∘ *c*) *j* ·$_v$ (*?ib* ! *j*) )) [*0*..<(*d*)]) **for** *x*

**proof**−
**obtain** $l$ **where** $l$: $x = ((of\text{-}int \circ c)\ l \cdot_v (?ib\ !\ l))\$i \wedge l \in \{0..<d\}$ **using** $x$ **by** *fastforce*
**then have** $(?ib\ !\ l) \in carrier\text{-}vec\ (d + 1)$ **using** *int-gen-basis-carrier*[*of ts*] *assms il* **by** *auto*
**then have** $((of\text{-}int \circ c)\ l \cdot_v (?ib\ !\ l))\$i = (of\text{-}int \circ c)\ l * (?ib\ !l)\$i$ **using** $t$ **by** *auto*
**moreover have** $(?ib\ !l)\$i = (int\ (p^2) \cdot_v unit\text{-}vec\ (d + 1)\ l)\$i$
**unfolding** *int-gen-basis-def* **using** *append-Cons-nth-left*[*of l map* $(\lambda i.\ int\ (p^2) \cdot_v unit\text{-}vec\ (d + 1)\ i)$ $[0..<d]$ *vec-of-list* $(map\ ((*)\ (int\ p))\ (map\ int\ ts)\ @\ [1])$ $[]$] $l$
**by** *force*
**moreover have** $(int\ (p^2) \cdot_v unit\text{-}vec\ (d + 1)\ l)\$i = 0$
**using** $t$ *in-range* $l$ **by** *fastforce*
**ultimately show** $x = 0$ **using** $l$ **by** *simp*
**qed**
**then have** $sum\text{-}list\ ((map\ ((\lambda l.\ l\$i) \circ (\lambda j.\ (of\text{-}int \circ c)\ j \cdot_v (?ib\ !\ j))\ ))\ [0..<(d)]) = 0$
**using** *sum-list-neutral* **by** *blast*
**moreover have** $((\lambda l.\ l\$i) \circ (\lambda j.\ (of\text{-}int \circ c)\ j \cdot_v (?ib\ !\ j))\ )\ d = c\ d$
**proof**−
**have** $c$: $(?ib\ !\ d) \in carrier\text{-}vec\ (d + 1)$
**using** *int-gen-basis-carrier*[*of ts*] *assms il* **by** *force*
**have** $((\lambda l.\ l\$i) \circ (\lambda j.\ (of\text{-}int \circ c)\ j \cdot_v (?ib\ !\ j))\ )\ d = (((of\text{-}int \circ c)\ d) \cdot_v (?ib\ !\ d))\$i$ **by** *simp*
**moreover have** $\ldots = ((of\text{-}int \circ c)\ d) * (?ib\ !\ d)\$i$ **using** $c\ t$ **by** *simp*
**moreover have** $((?ib\ !\ d))\$i = 1$
**unfolding** *int-gen-basis-def*
**using** *append-Cons-nth-middle*[*of d map* $(\lambda i.\ int\ (p^2) \cdot_v unit\text{-}vec\ (d + 1)\ i)$ $[0..<d]$ *vec-of-list* $(map\ ((*)\ (int\ p))\ (map\ int\ ts)\ @\ [1])$ $[]$]
$t$ *append-Cons-nth-middle*[*of i map* $((*)\ (int\ p))\ (map\ int\ ts)\ 1\ []$] *assms*(*1*) **by** *force*
**ultimately show** *?thesis* **by** *force*
**qed**
**ultimately have** *scaled-v* \$ $i = c\ d$ **using** *sv* **by** *presburger*
**then have** $((1/(of\text{-}nat\ p)) \cdot_v (map\text{-}vec\ rat\text{-}of\text{-}int\ scaled\text{-}v))\$i = rat\text{-}of\text{-}int\ (c\ d)\ /\ rat\text{-}of\text{-}nat\ p$
**using** *carr-iv* $t$ **by** *simp*
**then show** *?thesis* **using** *coordinates-of-gen-lattice*[*of d ts c*]
**unfolding** *lincomb-list-def* **using** *assms* $t$ **by** *force*
**next**
**case** $f$:*False*
**then have** $[0..<(d\ +\ 1)] = [0..<i]@[i]@[(i + 1)..<d]@[d]$
**using** *in-range upt-append* **by** *fastforce*
**then have** *0*: *scaled-v* \$ $i =$ $sum\text{-}list\ ((map\ ((\lambda l.\ l\$i) \circ (\lambda j.\ (of\text{-}int \circ c)\ j \cdot_v (?ib\ !\ j))\ ))\ [0..<i]) +$
$((\lambda l.\ l\$i) \circ (\lambda j.\ (of\text{-}int \circ c)\ j \cdot_v (?ib\ !\ j))\ )\ i +$
$sum\text{-}list\ ((map\ ((\lambda l.\ l\$i) \circ (\lambda j.\ (of\text{-}int \circ c)\ j \cdot_v (?ib\ !\ j))\ ))\ [(i + 1)..<(d)]) +$

```
                                ((λl. l$i) ∘ (λj. (of-int ∘ c) j ·v (?ib ! j) ) ) d
        using ∗ by force
      have 3: x = 0 if x: x ∈ set ((map ((λl. l$i) ∘ (λj. (of-int ∘ c) j ·v (?ib ! j)
) )) [(i+1)..<d]) for x
      proof−
       obtain l where l: x = ((of-int ∘ c) l ·v (?ib!l) )$i ∧ l∈{(i+1)..<d} using
x by auto
        then have ?ib!l ∈ carrier-vec (d + 1)
          using int-gen-basis-carrier[of ts] assms(1) il by fastforce
        then have ((of-int ∘ c) l ·v (?ib!l) )$i = (of-int ∘ c) l ∗ (?ib!l)$i
          using f in-range by fastforce
        moreover have ?ib!l$i = (int (p^2) ·v unit-vec (d + 1) l)$i
           unfolding int-gen-basis-def using append-Cons-nth-left[of l map (λi.
int (p^2) ·v unit-vec (d + 1) i) [0..<d] vec-of-list (map ((∗) (int p)) (map int ts) @
[1])]
          using l by simp
       moreover have (int (p^2) ·v unit-vec (d + 1) l)$i = 0 using l by fastforce
        ultimately show ?thesis using x l by presburger
      qed
      have x = 0 if x: x ∈ set ((map ((λl. l$i) ∘ (λj. (of-int ∘ c) j ·v (?ib ! j) )
)) [0..<i]) for x
      proof−
       obtain l where l: x = ((of-int ∘ c) l ·v (?ib!l) )$i ∧ l∈{0..<i} using x
by auto
        then have ?ib!l ∈ carrier-vec (d + 1)
          using int-gen-basis-carrier[of ts] assms(1) il in-range by fastforce
        then have ((of-int ∘ c) l ·v (?ib!l) )$i = (of-int ∘ c) l ∗ (?ib!l)$i
          using f in-range by fastforce
        moreover have ?ib!l$i = (int (p^2) ·v unit-vec (d + 1) l)$i
           unfolding int-gen-basis-def using append-Cons-nth-left[of l map (λi.
int (p^2) ·v unit-vec (d + 1) i) [0..<d] vec-of-list (map ((∗) (int p)) (map int ts) @
[1])]
          using l in-range by simp
         moreover have (int (p^2) ·v unit-vec (d + 1) l)$i = 0 using l in-range
by fastforce
        ultimately show ?thesis using x l by presburger
      qed
       then have sum-list ((map ((λl. l$i) ∘ (λj. (of-int ∘ c) j ·v (?ib ! j) ) ))
[0..<i]) = 0
        using sum-list-neutral by blast
      moreover have sum-list ((map ((λl. l$i) ∘ (λj. (of-int ∘ c) j ·v (?ib ! j) )
)) [(i+1)..<d]) = 0
        using sum-list-neutral 3 by blast
      moreover have ((λl. l$i) ∘ (λj. (of-int ∘ c) j ·v (?ib ! j) ) ) i = c i ∗ p^2
      proof−
        have ?ib!i ∈ carrier-vec (d + 1)
          using int-gen-basis-carrier[of ts] assms(1) il in-range by force
        then have ((λl. l$i) ∘ (λj. (of-int ∘ c) j ·v (?ib ! j) ) ) i = ((of-int ∘ c)
i) ∗ (?ib!i$i)
```

**using** *in-range* **by** *force*
**moreover have** *?ib*!*i* = (*int* (*p*$^2$) ·$_v$ *unit-vec* (*d* + *1*) *i*)
**unfolding** *int-gen-basis-def* **using** *append-Cons-nth-left*[*of i map* (λ*i*. *int* (*p*$^2$) ·$_v$ *unit-vec* (*d* + *1*) *i*) [*0*..<*d*] *vec-of-list* (*map* ((∗) (*int p*)) (*map int ts*) @ [*1*])]
*in-range f* **by** *fastforce*
**moreover have** (*int* (*p*$^2$) ·$_v$ *unit-vec* (*d* + *1*) *i*)$*i* = *p*^*2* **using** *in-range* **by** *force*
**ultimately show** *?thesis* **by** *simp*
**qed**
**moreover have** ((λ*l*. *l*$*i*) ∘ (λ*j*. (*of-int* ∘ *c*) *j* ·$_v$ (*?ib* ! *j*) ) ) *d* = *c d* ∗ *p* ∗ (*ts*!*i*)
**proof**−
**have** *?ib*!*d* ∈ *carrier-vec* (*d* + *1*)
**using** *int-gen-basis-carrier*[*of ts*] *assms*(*1*) *il* **by** *fastforce*
**then have** ((λ*l*. *l*$*i*) ∘ (λ*j*. (*of-int* ∘ *c*) *j* ·$_v$ (*?ib* ! *j*) ) ) *d* = ((*of-int* ∘ *c*) *d*) ∗ (*?ib*!*d*$*i*)
**using** *in-range* **by** *simp*
**moreover have** (*?ib*!*d*) = *vec-of-list* (*map* ((∗) (*int p*)) (*map int ts*) @ [*1*])
**unfolding** *int-gen-basis-def*
**using** *append-Cons-nth-middle*[*of d map* (λ*i*. *int* (*p*$^2$) ·$_v$ *unit-vec* (*d* + *1*) *i*) [*0*..<*d*] *vec-of-list* (*map* ((∗) (*int p*)) (*map int ts*) @ [*1*])]
**by** *simp*
**moreover have** *vec-of-list* (*map* ((∗) (*int p*)) (*map int ts*) @ [*1*]) $ *i* = *map* ((∗) (*int p*)) (*map int ts*) ! *i*
**using** *append-Cons-nth-left*[*of i map* ((∗) (*int p*)) (*map int ts*) *1* []] *in-range f*
*vec-index-vec-of-list*[*of* (*map* ((∗) (*int p*)) (*map int ts*) @ [*1*]) *i*] *assms* **by** *force*
**moreover have** *map* ((∗) (*int p*)) (*map int ts*) ! *i* = (*int p*) ∗ *ts*!*i* **using** *in-range f assms* **by** *auto*
**ultimately show** *?thesis* **by** *auto*
**qed**
**ultimately have** *scv*: *scaled-v* $ *i* = (*c i*) ∗ *p*^*2* + *c d* ∗ *p*∗(*ts*!*i*) **using** *0* **by** *linarith*
**have** *foil*: *rat-of-int* (*c i* ∗ *int* (*p*$^2$) + *c d* ∗ *int p* ∗ *int* (*ts* ! *i*)) =
*rat-of-nat p* ∗ *rat-of-int* (*c i* ∗ *int p* + *c d* ∗ *int* (*ts* ! *i*))
**by** (*smt* (*verit*, *del-insts*) *Power.semiring-1-class.of-nat-power Ring-Hom.of-int-hom.hom-mult int-distrib*(*1*) *more-arith-simps*(*11*) *mult-ac*(*2*) *of-int-of-nat-eq power2-eq-square*)
**have** ((*1*/(*of-nat p*))·$_v$(*map-vec rat-of-int scaled-v*))$*i* = *rat-of-int* ((*c i*) ∗ *p*^*2* + *c d* ∗ *p*∗(*ts*!*i*)) / (*rat-of-nat p*)
**using** *scv carr-iv f in-range* **by** *force*
**moreover have** ... = *rat-of-int* ((*c i*) ∗ *p* + (*c d*) ∗ (*ts*!*i*))
**using** *foil p* **by** *auto*
**ultimately show** *?thesis*
**using** *coordinates-of-gen-lattice*[*of i ts c*] *assms f in-range*
**unfolding** *lincomb-list-def*
**by** *simp*

```
    qed
  qed
  then show ?thesis using carr-v carr-sv by auto
 qed
 ultimately show (1/(of-nat p)) ·_v (map-vec rat-of-int scaled-v) ∈ ?L by argo
qed

definition t-vec :: nat list ⇒ rat vec where
 t-vec ts = vec-of-list ((map of-nat ts) @ [1 / (of-nat p)])

lemma t-vec-dim: dim-vec (t-vec ts) = length ts + 1
 unfolding t-vec-def by simp

lemma t-vec-last: length ts = d ⟹ (t-vec ts)$d = 1 / (of-nat p)
 unfolding t-vec-def
 by (simp add: append-Cons-nth(1))

definition z-vecs :: int vec set where
 z-vecs = {v. dim-vec v = d + 1 ∧ v$d = 0}

definition vec-class :: nat list ⇒ int ⇒ rat vec set where
 vec-class ts β = {(of-int β) ·_v (t-vec ts) + lincomb-list (of-int ∘ cs) p-vecs | cs ::
nat ⇒ int. True}

definition vec-class-mod-p :: nat list ⇒ int ⇒ rat vec set where
 vec-class-mod-p ts β = ⋃{vec-class ts β' | β'. [β = β'] (mod p)}

lemma vec-class-carrier:
 assumes length ts = d
 shows vec-class ts β ⊆ carrier-vec (d + 1)
proof
 fix x assume x ∈ vec-class ts β
 then obtain cs where x = (of-int β) ·_v (t-vec ts) + lincomb-list (of-int ∘ cs)
p-vecs
   unfolding vec-class-def by blast
 moreover have t-vec ts ∈ carrier-vec (d + 1)
   using t-vec-dim[of ts] unfolding assms(1) carrier-vec-def by blast
 moreover have lincomb-list (of-int ∘ cs) p-vecs ∈ carrier-vec (d + 1)
   by (metis p-vecs-carrier carrier-vec-dim-vec lincomb-list-carrier subsetI)
 ultimately show x ∈ carrier-vec (d + 1) by blast
qed

lemma vec-class-mod-p-carrier:
 assumes length ts = d
 shows vec-class-mod-p ts β ⊆ carrier-vec (d + 1)
 unfolding vec-class-mod-p-def using assms vec-class-carrier by blast

lemma vec-class-last:
 assumes length ts = d
```

**assumes** $v \in$ *vec-class ts* $\beta$
**shows** $v\$d =$ *rat-of-int* $\beta$ / *rat-of-int p*
**using** *assms*(2) **unfolding** *vec-class-def*
**proof**−
**obtain** *cs* **where** *cs*: $v =$ *rat-of-int* $\beta \cdot_v$ *t-vec ts* + *lincomb-list* (*rat-of-int* ∘ *cs*) *p-vecs*
**using** *assms*(2) **unfolding** *vec-class-def* **by** *blast*
**let** *?a* = *rat-of-int* $\beta \cdot_v$ *t-vec ts*
**let** *?b* = *lincomb-list* (*rat-of-int* ∘ *cs*) *p-vecs*
**have** *?a*$*d* = *rat-of-int* $\beta$ / *rat-of-int p*
**using** *t-vec-last*[*OF assms*(1)] **by** (*simp add*: *assms*(1) *t-vec-dim*)
**moreover have** *?b*$*d* = *0* **using** *lincomb-of-p-vecs-last* **by** *blast*
**ultimately show** *?thesis* **using** *cs vec-class-carrier*[*OF assms*(1), *of* $\beta$]
**by** (*smt* (*z3*) *One-nat-def add-Suc-right assms*(1,2) *carrier-vec-dim-vec index-add-vec*(1) *index-add-vec*(2) *lessI r-zero semiring-norm*(51) *subsetD t-vec-dim*)
**qed**

**lemma** *gen-lattice-int-gen-lattice-vec′*:
**fixes** *v* :: *rat vec*
**fixes** *ts* :: *nat list*
**assumes** *length ts* = *d*
**assumes** $v \in$ *gen-lattice ts*
**shows** *map-vec int-of-rat* ((*of-nat p*) $\cdot_v$ *v*) ∈ *int-gen-lattice ts*
*map-vec rat-of-int* (*map-vec int-of-rat* ((*of-nat p*) $\cdot_v$ *v*)) = (*rat-of-nat p*) $\cdot_v$ *v*
**proof** −
**have** *dims*: (⋀*w*. *List.member* (*gen-basis ts*) *w* ⟹ *dim-vec w* = *d* + *1*)
**by** (*meson List.member-def assms*(1) *carrier-dim-vec gen-basis-carrier subsetD*)
**have** *pv-in*: (*rat-of-nat p* $\cdot_v$ *v*) ∈ *lattice-of* (*map* (λ*x*. *rat-of-nat p* $\cdot_v$ *x*) (*gen-basis ts*))
**using** *assms*(2)
**unfolding** *gen-lattice-def*
**using** *smult-in-lattice-of*[*OF dims*, *of* (*gen-basis ts*) *v rat-of-nat p*]
**by** *blast*
**then have** *lattice-of-map*: *rat-of-nat p* $\cdot_v$ *v*
∈ *lattice-of*
(*map* ((·$_v$) (*rat-of-nat p*))
(*p-vecs* @
[*vec-of-list*
(*map rat-of-nat ts* @ [*1* / *rat-of-nat p*])]))
**unfolding** *gen-basis-def* **by** *blast*
**then have** *eq1*: *rat-of-nat p* $\cdot_v$ *v*
∈ *lattice-of*
((*map* ((·$_v$) (*rat-of-nat p*))
(*p-vecs*)) @
[((·$_v$) (*rat-of-nat p*) (*vec-of-list*
(*map rat-of-nat ts* @ [*1* / *rat-of-nat p*])))])
**by** *simp*
**have** *generic-aux*: ⋀*p*::*rat*. [(·$_v$) *p* (*vec-of-list ell*)] = [*vec-of-list* (*map* (λ*x*. *p*∗*x*)

```
ell)]
    for ell :: rat list
    by auto
  have generic: ⋀p elt::rat. [(·ᵥ) p (vec-of-list (ell @ [elt]))] =
      [ (vec-of-list ((map (λx. p*x) ell) @ [p* elt]))]
    for ell :: rat list
  proof −
    fix p elt :: rat
   have [(·ᵥ) p (vec-of-list (ell @ [elt]))] = [vec-of-list (map (λx. p*x) (ell @ [elt]))]
      using generic-aux by blast
    then show [(·ᵥ) p (vec-of-list (ell @ [elt]))] =
      [ (vec-of-list ((map (λx. p*x) ell) @ [p* elt]))]
      by simp
  qed
  have h1: (map ((*) (rat-of-nat p)) (map rat-of-nat ts)) = map (λx. rat-of-nat p
* rat-of-nat x) ts
    by simp
  have h2: rat-of-nat p * (1 / rat-of-nat p) = 1
    by (simp add: p prime-gt-0-nat)
  then have eq1: [((·ᵥ) (rat-of-nat p) (vec-of-list
          (map rat-of-nat ts @ [1 / rat-of-nat p])))] =
      [vec-of-list
          (map (λx. (rat-of-nat p) * rat-of-nat x) ts @ [1])]
    unfolding generic[of (rat-of-nat p) map rat-of-nat ts 1 / rat-of-nat p]
    using h1 h2 by argo

  let ?qs = p-vecs @ [vec-of-list (map rat-of-nat ts @ [1 / rat-of-nat p])]
  let ?qs2 = (map ((·ᵥ) (rat-of-nat p))
       (p-vecs)) @
        [vec-of-list
          (map (λx. (rat-of-nat p) * rat-of-nat x) ts @ [1])]

  have rat-of-nat-qs2: rat-of-nat p ·ᵥ v ∈ lattice-of ?qs2
    using eq1 gen-basis-def pv-in by force


  have mem1: ∃ k. map-vec rat-of-int k = mem if
      is-mem: List.member (map ((·ᵥ) (rat-of-nat p)) p-vecs) mem
    for mem
  proof −
    obtain i where i-prop: mem = ((·ᵥ) (rat-of-nat p))
     (map-vec rat-of-int (int p ·ᵥ unit-vec (d + 1) i))
      i < d
      using is-mem unfolding p-vecs-def
    by (smt (verit, ccfv-SIG) Groups.monoid-add-class.add.left-neutral List.List.list.set-map
List.member-def diff-zero imageE in-set-conv-nth length-map length-upt nth-map-upt)
    show ?thesis
      unfolding i-prop(1)
      by (metis Matrix.of-int-hom.vec-hom-smult of-int-of-nat-eq)
```

**qed**
**have** *mem2*: $\exists k.$ *map-vec rat-of-int k* = *mem* **if** *mem-ts*: *mem* = *vec-of-list* (*map* (λ*x*. *rat-of-nat p* ∗ *rat-of-nat x*) *ts* @ [*1*]) **for** *mem*
**proof** −
**let** *?k* = *vec-of-list* (*map* (λ*x*. (*int p*) ∗ *x*) *ts* @ [*1*])
**have** *map rat-of-int* (*map* (λ*x*. (*int p*) ∗ *x*) *ts* @ [*1*]) =
*map* (λ*x*. *rat-of-nat p* ∗ *rat-of-nat x*) *ts* @ [*1*]
**by** *auto*
**then have** *map-vec rat-of-int ?k* =
*vec-of-list* (*map* (λ*x*. *rat-of-nat p* ∗ *rat-of-nat x*) *ts* @ [*1*])
**by** (*metis vec-of-list-map*)
**then show** *?thesis* **unfolding** *mem-ts*
**by** *blast*
**qed**

**have** *mem-int*: (⋀*mem*. *List.member ?qs2 mem* ⟹ $\exists k.$ *map-vec rat-of-int k* = *mem*)
**using** *mem1 mem2*
**by** (*metis append-insert in-set-member insert-iff*)
**have** *mem-dims*: (⋀*mem*. *List.member ?qs2 mem* ⟹ *dim-vec mem* = *d* + *1*)
**using** *dims gen-basis-def*
**by** (*smt* (*verit*, *ccfv-threshold*) *List.List.list.set-map List.member-def append-insert eq1 imageE index-smult-vec*(*2*) *insert-iff*)
**then have** *casting-hyp*: $\exists k$::*int vec*. *map-vec rat-of-int k* = *rat-of-nat p* ·$_v$ *v*
**using** *assms*(*2*) **unfolding** *gen-lattice-def gen-basis-def*
**using** *in-lattice-casting*[*OF mem-int mem-dims rat-of-nat-qs2*]
**by** *blast*

**have** *rat-of-nat-is*: *rat-of-nat p* ·$_v$ *v*
∈ *lattice-of*
((*map* ((·$_v$) (*rat-of-nat p*))
(*p-vecs*)) @
[*vec-of-list*
(*map* (λ*x*. (*rat-of-nat p*) ∗ *rat-of-nat x*) *ts* @ [*1*])])
**using** *eq1 lattice-of-map* **by** *simp*
**have** *vec-is-in-lattice*: (*map-vec int-of-rat* (*rat-of-nat p* ·$_v$ *v*))
∈ *local.vec-int.lattice-of*
(*map* (*map-vec int-of-rat*) (*map* ((·$_v$) (*rat-of-nat p*)) *p-vecs* @
[*vec-of-list* (*map* (λ*x*. *rat-of-nat p* ∗ *rat-of-nat x*) *ts* @ [*1*])]))
**using** *casting-lattice-lemma*[*OF casting-hyp mem-int mem-dims rat-of-nat-is*]
**by** *blast*

**have** *big-map-is*: (*map* (*map-vec int-of-rat*)
(*map* ((·$_v$) (*rat-of-nat p*))
(*map* (λ*i*. *map-vec rat-of-int*
(*int p* ·$_v$ *vec* (*d* + *1*) (λ*j*. *if j* = *i then 1 else 0*)))
[*0*..<*d*]))) ! *i* =
*map-vec int-of-rat* ((·$_v$) (*rat-of-nat p*)) (*map-vec rat-of-int*
(*int p* ·$_v$ *vec* (*d* + *1*) (λ*j*. *if j* = *i then 1 else 0*)))) **if** *i-lt*: *i* < *d* **for** *i*

**using** *i-lt* **by** *auto*
**have** $(\cdot_v)$ (*rat-of-nat p*) (*map-vec rat-of-int*
(*int p* $\cdot_v$ *vec* (*d* + *1*) (λ*j*. *if j* = *i then 1 else 0*))) =
*map-vec rat-of-int* (($\cdot_v$) *p* (*int p* $\cdot_v$ *vec* (*d* + *1*) (λ*j*. *if j* = *i then 1 else 0*))) **if** *i-lt*: *i* < *d* **for** *i*
**by** (*simp add*: *Matrix.of-int-hom.vec-hom-smult*)
**then have** *map-vec int-of-rat* (($\cdot_v$) (*rat-of-nat p*) (*map-vec rat-of-int*
(*int p* $\cdot_v$ *vec* (*d* + *1*) (λ*j*. *if j* = *i then 1 else 0*)))) =
*map-vec int-of-rat* (*map-vec rat-of-int* (($\cdot_v$) *p* (*int p* $\cdot_v$ *vec* (*d* + *1*) (λ*j*. *if j* = *i then 1 else 0*)))) **if** *i-lt*: *i* < *d* **for** *i*
**by** *auto*
**then have** *map-vec int-of-rat* (($\cdot_v$) (*rat-of-nat p*) (*map-vec rat-of-int*
(*int p* $\cdot_v$ *vec* (*d* + *1*) (λ*j*. *if j* = *i then 1 else 0*)))) =
(($\cdot_v$) *p* (*int p* $\cdot_v$ *vec* (*d* + *1*) (λ*j*. *if j* = *i then 1 else 0*))) **if** *i-lt*: *i* < *d* **for** *i*
**by** (*metis* (*no-types*, *lifting*) *Matrix.of-int-hom.vec-hom-smult eq-vecI index-map-vec*(*1*) *index-map-vec*(*2*) *int-of-rat*(*1*) *of-int-of-nat-eq*)
**then have** *lhs-is*: (*map* (*map-vec int-of-rat*)
(*map* (($\cdot_v$) (*rat-of-nat p*))
(*map* (λ*i*. *map-vec rat-of-int*
(*int p* $\cdot_v$ *vec* (*d* + *1*) (λ*j*. *if j* = *i then 1 else 0*)))
[*0*..<*d*]))) ! *i* =
($\cdot_v$) *p* (*int p* $\cdot_v$ *vec* (*d* + *1*) (λ*j*. *if j* = *i then 1 else 0*)) **if** *i-lt*: *i* < *d* **for** *i*
**using** *i-lt* **by** *simp*

**have** *rhs-is*: (*map* (λ*i*. *int* ($p^2$) $\cdot_v$ *vec* (*d* + *1*) (λ*j*. *if j* = *i then 1 else 0*)) [*0*..<*d*]) ! *i* =
*int* ($p^2$) $\cdot_v$ *vec* (*d* + *1*) (λ*j*. *if j* = *i then 1 else 0*) **if** *i-lt*: *i* < *d* **for** *i*
**using** *i-lt* **by** *simp*

**have** *lhs-eq-rhs*: ($\cdot_v$) *p* (*int p* $\cdot_v$ *vec* (*d* + *1*) (λ*j*. *if j* = *i then 1 else 0*)) = *int* ($p^2$) $\cdot_v$ *vec* (*d* + *1*) (λ*j*. *if j* = *i then 1 else 0*) **if** *i-lt*: *i* < *d* **for** *i*
**by** (*simp add*: *local.vec-int.smult-assoc-simp power2-eq-square*)

**have** *first-part-eq*: *map* (*map-vec int-of-rat*) (*map* (($\cdot_v$) (*rat-of-nat p*)) *p-vecs*) =
(*map* (λ*i*. *int* ($p^2$) $\cdot_v$ *unit-vec* (*d* + *1*) *i*) [*0*..<*d*])
**unfolding** *p-vecs-def unit-vec-def*
**using** *lhs-is rhs-is lhs-eq-rhs* **by** *simp*
**have** *map-vec-is*: *map int-of-rat* (*map* (λ*x*. *rat-of-nat p* ∗ *rat-of-nat x*) *ts* @ [*1*])
= (*map* ((∗) (*int p*)) (*map int ts*) @ [*1*])
**apply** (*induct ts*)
**subgoal by** *simp* (*metis int-of-rat*(*1*) *of-int-1*)
**subgoal by** *simp* (*metis int-of-rat*(*1*) *of-int-of-nat-eq of-nat-simps*(*5*))
**done**
**then have** *last-elem-eq*: *map* (*map-vec int-of-rat*) [*vec-of-list*
(*map* (λ*x*. *rat-of-nat p* ∗ *rat-of-nat x*) *ts* @ [*1*])] =
[*vec-of-list* (*map* ((∗) (*int p*)) (*map int ts*) @ [*1*])]
**by** (*smt* (*verit*) *List.list.simps*(*8*) *List.list.simps*(*9*) *vec-of-list-map*)
**show** *int-gen-lattice*: *map-vec int-of-rat* ((*of-nat p*) $\cdot_v$ *v*) ∈ *int-gen-lattice ts*
**unfolding** *int-gen-lattice-def int-gen-basis-def gen-basis-def p-vecs-def*

```
    using vec-is-in-lattice last-elem-eq first-part-eq
    by auto
  have LHS-simp: (rat-of-int
          (vec (dim-vec v)
            (λi. int-of-rat
                  (vec (dim-vec v)
                    (λi. rat-of-nat p * v $ i) $
                   i)) $
           i)) =
     rat-of-int
          (int-of-rat
                (rat-of-nat p * v $ i)) if i-lt: i < dim-vec v for i
    using i-lt by simp
  have same-dims: dim-vec
                (vec (dim-vec
                (vec (dim-vec v)
                  (λi. rat-of-nat p * v $ i)))
           (λi. int-of-rat
                 (vec (dim-vec v)
                   (λi. rat-of-nat p * v $ i) $
                  i))) = dim-vec v
    by simp
  have rat-of-int
          (vec (dim-vec v)
            (λi. int-of-rat
                  (vec (dim-vec v)
                    (λi. rat-of-nat p * v $ i) $
                   i)) $
           i) = rat-of-nat p * v $ i if i-lt: i < dim-vec v
   for i
  proof −
    show ?thesis
      using i-lt unfolding LHS-simp[OF i-lt]
      by (metis casting-hyp index-map-vec(1) index-map-vec(2) index-smult-vec(1)
index-smult-vec(2) int-of-rat(1))
  qed
  then have same-vec: vec (dim-vec v) (λi. rat-of-int
          (vec (dim-vec
                 (vec (dim-vec v)
                   (λi. rat-of-nat p * v $ i)))
            (λi. int-of-rat
                  (vec (dim-vec v)
                    (λi. rat-of-nat p * v $ i) $
                   i)) $
           i)) = vec (dim-vec v) (λi. rat-of-nat p * v $ i) using same-dims
    by (simp add: Matrix.vec-eq-iff)
  show map-vec rat-of-int (map-vec int-of-rat ((of-nat p) ·v v)) = (of-nat p)·v v
    unfolding map-vec-def smult-vec-def
    using same-dims same-vec by argo
```

**qed**

**lemma** *gen-lattice-int-gen-lattice-closest*:
**fixes** *scaled-v* :: *int vec*
**fixes** *u* :: *rat vec*
**fixes** *ts* :: *nat list*
**assumes** *length ts* = *d*
**assumes** *dim-vec u* = *d* + *1*
**shows** *real*(*p*^*2*) ∗ *Inf*{*real-of-rat* (*sq-norm* (*x* − *u*))| *x*. *x*∈ *gen-lattice ts*}
= *babai.closest-distance-sq* (*int-gen-basis ts*) ((*rat-of-nat p*) ·$_v$ *u*)
**proof**−
**let** *?iL* = *int-gen-lattice ts*
**let** *?ib* = *int-gen-basis ts*
**let** *?L* = *gen-lattice ts*
**let** *?b* = *gen-basis ts*
**have** *il*: *length ?ib* = *d* + *1*
**unfolding** *int-gen-basis-def* **by** *fastforce*
**have** *su*: *dim-vec* ((*rat-of-nat p*)·$_v$ *u*) = *d* + *1*
**using** *assms*(*2*) **by** *force*
**let** *?lhs* = *real*(*p*^*2*) ∗ *Inf*{*real-of-rat* (*sq-norm* (*x* − *u*))| *x*. *x*∈ *gen-lattice ts*}
**let** *?rhs* = *babai.closest-distance-sq* (*int-gen-basis ts*) ((*rat-of-nat p*)·$_v$*u*)
**let** *?coset* = {(*map-vec rat-of-int x*) − ((*rat-of-nat p*)·$_v$*u*)| *x*. *x* ∈ *int-gen-lattice ts*}
**have** *bab*: *babai ?ib* ((*rat-of-nat p*)·$_v$*u*)
**using** *il su* **unfolding** *babai-def* **by** *argo*
**have** *?coset* ≠ {} **unfolding** *int-gen-lattice-def vec-int.lattice-of-def* **by** *blast*
**then have** ∗: {(*real-of-rat* (*sq-norm* (*x*)))| *x*. *x*∈ *?coset*} ≠ {} **by** *simp*
**have** *?L* ≠ {} **unfolding** *gen-lattice-def lattice-of-def* **by** *blast*
**then have** ∗∗: {(*real-of-rat* (*sq-norm* (*x* − *u*)))| *x*. *x* ∈ *?L*} ≠ {} **by** *blast*
**have** *rhs*: *?rhs* = *Inf* {(*real-of-rat* (*sq-norm* (*x*)))| *x*. *x*∈ *?coset*}
**using** *babai.closest-distance-sq-def*[*of ?ib rat-of-nat p* ·$_v$ *u*] *bab su il LLL.LLL.L-def*[*of d* + *1 ?ib*] **unfolding** *int-gen-lattice-def* **by** *presburger*
**have** ∀ *x*∈{*real-of-rat* (*sq-norm* (*x* − *u*))| *x*. *x*∈ *gen-lattice ts*}. *0* ≤ *x*
**using** *sq-norm-vec-ge-0* **by** *fastforce*
**then have** *bdd*: *bdd-below* {*real-of-rat* (*sq-norm* (*x* − *u*))| *x*. *x*∈ *gen-lattice ts*}
**by** *fast*
**have** ∀ *x*∈{(*real-of-rat* (*sq-norm* (*x*)))| *x*. *x*∈ *?coset*}. *0* ≤ *x*
**using** *sq-norm-vec-ge-0* **by** *fastforce*
**then have** *bdd2*: *bdd-below* {(*real-of-rat* (*sq-norm* (*x*)))| *x*. *x*∈ *?coset*} **by** *fast*
**have** *?lhs* ≤ *?rhs*
**proof**(*rule ccontr*)
**assume** ¬(*?lhs* ≤ *?rhs*)
**then have** *?lhs* > *?rhs* **by** *linarith*
**then obtain** *D* **where** *D*: *D* < *?lhs* ∧ *D* ∈ {(*real-of-rat* (*sq-norm* (*x*)))| *x*. *x*∈ *?coset*}
**using** *rhs cInf-lessD*[*of* {(*real-of-rat* (*sq-norm* (*x*)))| *x*. *x*∈ *?coset*} *?lhs*] ∗ **by** *auto*
**then obtain** *iv* **where** *iv*: *real-of-rat* (*sq-norm* ((*map-vec rat-of-int iv*) − (*rat-of-nat p* ·$_v$ *u*)) ) = *D* ∧ *iv* ∈ *?iL* **by** *blast*

**then have** *iv-carr*: *iv* ∈ *carrier-vec* (*d* + *1*)
**using** *int-gen-lattice-carrier assms* **by** *fast*
**let** *?v* = (*1*/(*of-nat p*))·$_v$ (*map-vec rat-of-int iv*)
**have** *?v* ∈ *?L* **using** *iv gen-lattice-int-gen-lattice-vec assms* **by** *presburger*
**then have** *real-of-rat* (*sq-norm* (*?v* − *u*)) ∈ {*real-of-rat* (*sq-norm* (*x* − *u*))| *x*. *x*∈ *gen-lattice ts*} **by** *fast*
**moreover have** *real-of-rat* (*sq-norm* (*?v* − *u*)) < *Inf*{*real-of-rat* (*sq-norm* (*x* − *u*))| *x*. *x*∈ *gen-lattice ts*}
**proof**−
**have** *h*: *0*<*real*(*p*^*2*) **using** *p* **by** *simp*
**have** *real-of-rat* (*sq-norm* ((*map-vec rat-of-int iv*) − (*rat-of-nat p* ·$_v$ *u*)) ) < *?lhs*
**using** *D iv* **by** *fast*
**then have** *real-of-rat* (*sq-norm* ((*map-vec rat-of-int iv*) − (*rat-of-nat p* ·$_v$ *u*)) ) < *Inf*{*real-of-rat* (*sq-norm* (*x* − *u*))| *x*. *x*∈ *gen-lattice ts*} ∗ *real*(*p*^*2*) **by** *argo*
**then have** *real-of-rat* (*sq-norm* ((*map-vec rat-of-int iv*) − (*rat-of-nat p* ·$_v$ *u*)) ) / (*of-nat p*^*2*) < *Inf*{*real-of-rat* (*sq-norm* (*x* − *u*))| *x*. *x*∈ *gen-lattice ts*}
**using** *h divide-less-eq*[*of real-of-rat* (*sq-norm* ((*map-vec rat-of-int iv*) − (*rat-of-nat p* ·$_v$ *u*)) ) *real* (*p*^*2*) *Inf*{*real-of-rat* (*sq-norm* (*x* − *u*))| *x*. *x*∈ *gen-lattice ts*}]
**by** *simp*
**moreover have** *real-of-rat* (*sq-norm* ((*map-vec rat-of-int iv*) − (*rat-of-nat p* ·$_v$ *u*)) ) / (*of-nat p*^*2*) = *real-of-rat* (*sq-norm* ((*map-vec rat-of-int iv*) − (*rat-of-nat p* ·$_v$ *u*)) / (*of-nat p*^*2*))
**by** (*simp add*: *Ring-Hom.of-rat-hom.hom-div Ring-Hom.of-rat-hom.hom-power*)
**moreover have** ... = *real-of-rat* ( *sq-norm-conjugate* (*1* / (*of-nat p*)) ∗ *sq-norm* ((*map-vec rat-of-int iv*) − (*rat-of-nat p* ·$_v$ *u*)) )
**using** *conjugatable-ring-1-abs-real-line-class.sq-norm-as-sq-abs*[*of 1* / (*of-nat p*)]
**by** (*simp add*: *power2-eq-square*)
**moreover have** *sq-norm-conjugate* (*1* / (*of-nat p*)) ∗ *sq-norm* ((*map-vec rat-of-int iv*) − (*rat-of-nat p* ·$_v$ *u*))
= *sq-norm* ( (*1* / (*of-nat p*)) ·$_v$ ((*map-vec rat-of-int iv*) − (*rat-of-nat p* ·$_v$ *u*)) )
**using** *sq-norm-smult-vec* **by** *metis*
**moreover have** (*1* / (*of-nat p*)) ·$_v$ ((*map-vec rat-of-int iv*) − (*rat-of-nat p* ·$_v$ *u*)) = *1* / (*of-nat p*) ·$_v$ (*map-vec rat-of-int iv*) − (*1* / (*of-nat p*)) ·$_v$(*rat-of-nat p* ·$_v$ *u*)
**using** *su iv-carr carrier-vecI*[*OF su*]
**using** *p smult-sub-distrib-vec*[*of - d+1 rat-of-nat p* ·$_v$ *u 1* / *rat-of-nat p*]
**by** *auto*
**moreover have** (*1* / (*of-nat p*)) ·$_v$(*rat-of-nat p* ·$_v$ *u*) = *u*
**using** *smult-smult-assoc*[*of* (*1* / (*of-nat p*)) *rat-of-nat p*] *p* **by** *auto*
**ultimately show** *?thesis* **by** *algebra*
**qed**
**ultimately show** *False*
**using** *cInf-lower*[*of real-of-rat* (*sq-norm* (*?v* − *u*)) {*real-of-rat* (*sq-norm* (*x* − *u*))| *x*. *x*∈ *gen-lattice ts*}] *bdd* **by** *linarith*
**qed**

**moreover have** *?rhs* $\leq$ *?lhs*
**proof**(*rule ccontr*)
**assume** ¬(*?rhs* $\leq$ *?lhs*)
**then have** *?rhs* > *?lhs* **by** *linarith*
**then have** *?rhs* / *p*^2 > *Inf* {(*real-of-rat* (*sq-norm* (*x* − *u*)))| *x*. *x* ∈ *?L*}
**using** *p*
**by** (*metis* (*no-types*, *lifting*) *less-divide-eq mult-of-nat-commute of-nat-0-less-iff power-pos prime-gt-0-nat*)
**then obtain** *D* **where** *D*: *D* < (*?rhs* / *p*^2) ∧ *D* ∈ {(*real-of-rat* (*sq-norm* (*x* − *u*)))| *x*. *x* ∈ *?L*}
**using** *cInf-lessD*[*of* {(*real-of-rat* (*sq-norm* (*x* − *u*)))| *x*. *x* ∈ *?L*} *?rhs* / *p*^2] ** **by** *meson*
**then obtain** *v* **where** *v*: (*real-of-rat* (*sq-norm* (*v* − *u*))) = *D* ∧ *v* ∈ *?L*
**by** *blast*
**let** *?iv* = *map-vec int-of-rat* ((*rat-of-nat p*) ·*v* *v*)
**have** *?iv* ∈ *?iL* **using** *gen-lattice-int-gen-lattice-vec*′ *assms*(*1*) *v* **by** *blast*
**then have** *real-of-rat* (*sq-norm* ((*map-vec rat-of-int ?iv*) − ((*rat-of-nat p*)·*v u*))) ∈ {(*real-of-rat* (*sq-norm* (*x*))) | *x*. *x*∈ *?coset*} **by** *fast*
**moreover have** *real-of-rat* (*sq-norm* ((*map-vec rat-of-int ?iv*) − ((*rat-of-nat p*)·*v u*))) < *?rhs*
**proof**−
**have** *dim-v*: *v* ∈ *carrier-vec* (*d* + *1*)
**using** *v assms*(*1*) *gen-lattice-carrier* **by** *blast*
**have** *0*<*real*(*p*^2) **using** *p* **by** *simp*
**then have** (*real-of-rat* (*sq-norm* (*v* − *u*))) ∗ *p*^2 < *?rhs*
**using** *less-divide-eq*[*of real-of-rat* (*sq-norm* (*v* − *u*)) *?rhs real*(*p*^2)] **using** *D v* **by** *algebra*
**moreover have** (*real-of-rat* (*sq-norm* (*v* − *u*))) ∗ *p*^2 = *real-of-rat* ( *rat-of-nat p*^2 ∗ *sq-norm* (*v* − *u*))
**by** (*metis Power.semiring-1-class.of-nat-power Ring-Hom.of-rat-hom.hom-mult cross3-simps*(*11*) *of-rat-of-nat-eq*)
**moreover have** *real-of-rat* ( *rat-of-nat p*^2 ∗ *sq-norm* (*v* − *u*)) = *real-of-rat* ((*sq-norm-conjugate* (*rat-of-nat p*)) ∗ *sq-norm* (*v* − *u*))
**using** *conjugatable-ring-1-abs-real-line-class.sq-norm-as-sq-abs*[*of rat-of-nat p*] *power2-eq-square*[*of rat-of-nat p*] **by** *simp*
**moreover have** (*sq-norm-conjugate* (*rat-of-nat p*)) ∗ *sq-norm* (*v* − *u*) = *sq-norm* ((*rat-of-nat p*) ·*v* (*v*−*u*) )
**using** *sq-norm-smult-vec* **by** *metis*
**moreover have** (*rat-of-nat p*) ·*v* (*v*−*u*) = (*rat-of-nat p*) ·*v* *v*− (*rat-of-nat p*) ·*v* *u*
**using** *smult-sub-distrib-vec*[*OF dim-v*, *of u* (*rat-of-nat p*)]
**using** *assms*(*2*)
**using** *carrier-vecI* **by** *blast*
**moreover have** (*rat-of-nat p*) ·*v* *v* = (*map-vec rat-of-int ?iv*)
**using** *gen-lattice-int-gen-lattice-vec*′ *assms v* **by** *simp*
**ultimately show** *?thesis* **by** *algebra*
**qed**
**ultimately show** *False*
**using** *rhs bdd2 cInf-lower*[*of real-of-rat* (*sq-norm* ((*map-vec rat-of-int ?iv*) −

((*rat-of-nat p*)$\cdot_v$*u*))) {(*real-of-rat* (*sq-norm* (*x*)))| *x*. *x*∈ *?coset*}] **by** *linarith*
**qed**
**ultimately show** *?lhs* = *?rhs* **by** *simp*
**qed**

**lemma** *close-vector-exists*:
**fixes** *ts* :: *nat list*
**assumes** *length ts* = *d*
**shows** ∃ *w*∈(*gen-lattice ts*). *sq-norm* ((*ts-to-u ts*) − *w*) ≤ (*of-nat* ((*d*+1) ∗ *p*^2))/2^(2∗*k*)
**proof**−
**let** *?u* = (*ts-to-u ts*)
**let** *?basis* = *gen-basis ts*
**let** *?L* = *gen-lattice ts*

**let** *?c* = λ*i*. (*if i* = *d* *then int-of-nat* α *else* (− *int-of-nat* (α ∗ *ts* ! *i div p*)))

**let** *?Lst* = (*map* (λ*i*. *of-int* (*?c i*) $\cdot_v$ (*?basis* ! *i*)) [*0* ..< *length ?basis*])
**let** *?w* = *sumlist ?Lst*
**have** *l*: *length ?basis* = *d* + *1* **unfolding** *gen-basis-def p-vecs-def* **by** *simp*
**then have** *l2*: *length ?Lst* = *d* + *1* **by** *auto*
**have** *in-lattice*: *?w* ∈ *?L* **unfolding** *gen-lattice-def lattice-of-def* **by** *fast*
**have** *dim-u*: *?u* ∈ *carrier-vec* (*d* + *1*) **using** *ts-to-u-carrier*[*of ts*] *assms* **by** *blast*
**have** *dim-w*: *?w* ∈ *carrier-vec* (*d* + *1*)
**using** *in-lattice gen-lattice-carrier*[*of ts*] *assms* **by** *blast*

**have** *k-small*: *k* + *1* < *log 2 p* **using** *k-plus-1-lt* **by** *linarith*

**have** *lst-i*: *?Lst* ! *i* = *rat-of-int* (*?c i*) $\cdot_v$ *?basis* ! *i*
**if** *in-range*: *i*∈{*0*..<*d*+*1*} **for** *i* **using** *in-range*
**by** (*smt* (*verit*, *ccfv-threshold*) *in-set-conv-nth l length-map map-nth nth-map set-upt*)
**then have** *?Lst* ! *i* ∈ *carrier-vec* (*d* + *1*) **if** *in-range*: *i*∈{*0*..<*d*+*1*} **for** *i*
**using** *gen-basis-vecs-carrier*[*of ts i*] *assms in-range l* **by** *simp*
**then have** *carr*: *set ?Lst* ⊆ *carrier-vec* (*d* + *1*)
**using** *set-list-subset*[*of ?Lst carrier-vec* (*d* + *1*)] *l2* **by** *presburger*
**have** *w-coord*: (*?w* $ *i* = *rat-of-int* (α ∗ (*ts* ! *i*) *mod p*)) **if** *in-range*: *i* ∈ {*0*..<*d*} **for** *i*
**proof**−
**have** *?w* $ *i* = *sum-list* (*map* (λ*j*. (*?Lst*!*j*)$*i*) [*0*..<(*length ?Lst*)])
**using** *sumlist-index-commute*[*of ?Lst i*] *in-range carr* **by** *fastforce*
**moreover have** ⋀*j*. *j*∈{*0*..<*d*+*1*} ⟹ ¬ (*j*=*i* ∨ *j* = *d*) ⟹ *?Lst*!*j*$*i* = *0*
**proof**−
**fix** *j*
**assume** *j-in-range*: *j*∈{*0*..<*d*+*1*}
**assume** ¬ (*j*=*i* ∨ *j* = *d*)
**then have** *j* ≠ *i* ∧ *j* ≠ *d* **using** *j-in-range* **by** *argo*
**then have** *off-diag*: *j* ≠ *i* ∧ *j* ∈ {*0*..<*d*} **using** *j-in-range* **by** *fastforce*

**have** *dim-basis*: *dim-vec* (*?basis* ! *j*) = (*d* + *1*)
**using** *gen-basis-vecs-carrier*[*of ts j*] *off-diag assms* **by** *simp*
**have** *?Lst*!*j*$*i* = (*rat-of-int* (*?c j*) ·$_v$ *?basis* ! *j*)$*i*
**using** *lst-i*[*of j*] *off-diag* **by** *simp*
**also have** (*rat-of-int* (*?c j*) ·$_v$ *?basis* ! *j*)$*i*
= *rat-of-int* (*?c j*) ∗ (*?basis* ! *j*)$*i*
**using** *dim-basis in-range* **by** *force*
**also have** (*?basis* ! *j*)$*i* = *0* **using** *gen-basis-units*[*of j i ts*] *off-diag in-range* **by** *simp*
**finally show** *?Lst*!*j*$*i* = *0* **by** *linarith*
**qed**
**moreover have** *set* [*0*..<(*length ?Lst*)] = {*0*..<*d*+*1*} **using** *l2* **by** *auto*
**ultimately have** *?w* $ *i* = *sum-list* (*map* (λ*j*. (*?Lst*!*j*)$*i*) (*filter* (λ*j*. *j*=*i* ∨ *j* = *d*) [*0*..<(*length ?Lst*)]))
**using** *sum-list-map-filter*[*of* [*0*..<(*length ?Lst*)] (λ*j*. *j*=*i* ∨ *j* = *d*) (λ*j*. (*?Lst*!*j*)$*i*)] *l2* **by** *algebra*
**also have** (*filter* (λ*j*. *j*=*i* ∨ *j* = *d*) [*0*..<(*length ?Lst*)]) = [*i*, *d*]
**using** *filter-or*[*of i d*]
**by** (*smt* (*verit*) *atLeastLessThan-iff diff-zero filter-or in-range l2 length-upt less-add-one*)
**finally have** *?w* $ *i* = *sum-list* (*map* (λ*j*. (*?Lst*!*j*)$*i*) [*i*, *d*]) **by** *force*
**then have** *?w* $ *i* = *?Lst*!*i*$*i* + *?Lst*!*d*$*i* **by** *simp*
**then have** *?w* $ *i* =
(*rat-of-int* (*?c i*) ·$_v$ *?basis* ! *i*)$*i* +
(*rat-of-int* (*?c d*) ·$_v$ *?basis* ! *d*)$*i*
**using** *lst-i*[*of i*] *lst-i*[*of d*] *in-range* **by** *force*
**also have** (*rat-of-int* (*?c i*) ·$_v$ *?basis* ! *i*)$*i* = *rat-of-int* (− *int-of-nat* (α ∗ *ts* ! *i div p*)) ∗ *rat-of-nat p*
**using** *in-range gen-basis-vecs-carrier*[*of ts i*] *assms gen-basis-units*[*of i i*] **by** *force*
**also have** (*rat-of-int* (*?c d*) ·$_v$ *?basis* ! *d*)$*i* = *rat-of-int* (*int-of-nat* α) ∗ *?basis*!*d*$*i*
**using** *in-range gen-basis-vecs-carrier*[*of ts d*] *assms* **by** *simp*
**also have** *?basis*!*d*$*i* = *vec-of-list* ((*map of-nat ts*) @ [*1* / (*of-nat p*)]) $ *i*
**unfolding** *gen-basis-def p-vecs-def* **using** *gen-basis-length*[*of ts*] **by** (*simp add*: *append-Cons-nth-middle*)
**also have** *vec-of-list* ((*map of-nat ts*) @ [*1* / (*of-nat p*)]) $ *i* = ((*map of-nat ts*) @ [*1* / (*of-nat p*)])!*i*
**by** *simp*
**also have** ((*map of-nat ts*) @ [*1* / (*of-nat p*)])!*i* = (*map of-nat ts*)!*i*
**using** *in-range assms append-Cons-nth-left*[*of i* (*map of-nat ts*) *1* / (*of-nat p*) []] **by** *auto*
**also have** (*map of-nat ts*)!*i* = *of-nat* (*ts*!*i*) **using** *in-range assms* **by** *auto*
**finally have** *?w* $ *i* = *rat-of-int* (− *int-of-nat* (α ∗ *ts* ! *i div p*)) ∗ *rat-of-nat p*
+ *rat-of-int* (*int-of-nat* α) ∗ (*of-nat* (*ts*!*i*)) **by** *blast*
**also have** *rat-of-int* (− *int-of-nat* (α ∗ *ts* ! *i div p*)) ∗ *rat-of-nat p*
+ *rat-of-int* (*int-of-nat* α) ∗ (*of-nat* (*ts*!*i*))
= *rat-of-int* (*int-of-nat* (α ∗ (*ts*!*i*)) − *int-of-nat* ((α ∗ *ts* ! *i div p*) ∗ *p*))
**using** *int-of-nat-def* **by** *auto*

**also have** *rat-of-int* (*int-of-nat* ($\alpha$ * (*ts*!*i*)) − *int-of-nat* (($\alpha$ * *ts* ! *i div p*) * *p*)) =
*rat-of-int* ($\alpha$ * (*ts*!*i*) *mod p*)
**by** (*simp add*: *int-of-nat-def int-ops*(9) *minus-div-mult-eq-mod of-nat-div*)
**finally show** (*?w* $ *i* = *rat-of-int* ($\alpha$ * (*ts* ! *i*) *mod p*)) **by** *linarith*
**qed**
**then have** *coords-close*: *abs*((*?u*−*?w*)$*i*)≤*rat-of-nat p*/ *rat-of-nat* 2^*k* **if** *in-range*: *i*∈{*0*..<*d*+*1*} **for** *i*
**proof**(*cases i* = *d*)
**case** *f*:*False*
**have** *i-upper*: *i*∈{*0*..<*d*} **using** *f in-range* **by** *auto*
**then have** *ree*: *i*<*length*(*ts* @ [*0*]) **using** *assms* **by** *fastforce*
**moreover have** *?u* ∈ *carrier-vec* (*d* + *1*) **using** *ts-to-u-carrier*[*of ts*] *assms* **by** *blast*
**moreover have** *?w* ∈ *carrier-vec* (*d* + *1*)
**using** *in-lattice gen-lattice-carrier*[*of ts*] *assms* **by** *blast*
**ultimately have** (*?u* − *?w*)$*i* = *?u*$*i* − *?w*$*i* **using** *in-range* **by** *fastforce*
**moreover have** *?u*$*i* = (*map* (*rat-of-nat* ∘ *msb-p* ∘ (λ*t*. $\alpha$ * *t mod p*)) *ts* @ [*0*]) !*i*
**unfolding** *ts-to-u-alt* **by** *fastforce*
**moreover have** (*map* (*rat-of-nat* ∘ *msb-p* ∘ (λ*t*. $\alpha$ * *t mod p*)) *ts* @ [*0*]) !*i*
= (*rat-of-nat* ∘ *msb-p* ∘ (λ*t*. $\alpha$ * *t mod p*)) (*ts*!*i*)
**using** *List.nth-map*[*of i* (*ts* @ [*0*]) (*rat-of-nat* ∘ *msb-p* ∘ (λ*t*. $\alpha$ * *t mod p*))]
*i-upper append-Cons-nth-left*[*of i map* (*rat-of-nat* ∘ *msb-p* ∘ (λ*t*. $\alpha$ * *t mod p*)) *ts 0* []]
*assms ree*
**by** *simp*
**moreover have** (*rat-of-nat* ∘ *msb-p* ∘ (λ*t*. $\alpha$ * *t mod p*)) (*ts*!*i*) = *rat-of-nat* (*msb-p* ($\alpha$ * *ts*!*i mod p*))
**by** *force*
**ultimately have** (*?u* − *?w*)$*i* = *rat-of-nat* (*msb-p* ($\alpha$ * *ts*!*i mod p*)) − *rat-of-nat* ($\alpha$ * *ts*!*i mod p*)
**using** *w-coord*[*of i*] *ree i-upper* **by** *linarith*
**then have** *rat-of-int*: (*?u* − *?w*)$*i* = *rat-of-int* ((*int* (*msb-p* ($\alpha$ * *ts*!*i mod p*)) − *int*($\alpha$ * *ts*!*i mod p*)))
**by** *linarith*
**have** *real-of-rat* ((*?u* − *?w*)$*i*) = *real-of-int* ((*int* (*msb-p* ($\alpha$ * *ts*!*i mod p*)) − *int*($\alpha$ * *ts*!*i mod p*)))
**using** *int-rat-real-casting-helper*[*OF rat-of-int*]
**by** *blast*
**then have** *abs*(*real-of-rat* ((*?u* − *?w*)$*i*)) = |*real-of-int* (*int* (*msb-p* ($\alpha$ * *ts*!*i mod p*)) − *int* ($\alpha$ * *ts*!*i mod p*))|
**by** *presburger*
**also have** ... = *real-of-int* |(*int* (*msb-p* ($\alpha$ * *ts*!*i mod p*)) − ($\alpha$ * *ts*!*i mod p*))|
**by** *linarith*
**finally have** *abs*(*real-of-rat* ((*?u* − *?w*)$*i*)) ≤ *real*(*p*)/*2*^(*k*)
**using** *msb-p-dist*[*of* ($\alpha$ * *ts*!*i mod p*)] **by** *linarith*
**moreover have** *real*(*p*)/*2*^(*k*) = *real-of-rat*((*rat-of-nat p*) / *2*^*k*)
**by** (*simp add*: *of-rat-divide of-rat-power*)

**moreover have** *abs(real-of-rat ((?u − ?w)$i)) = real-of-rat (abs ((?u − ?w)$i))* **by** *simp*
**ultimately have** *real-of-rat (abs ((?u − ?w)$i))≤real-of-rat((rat-of-nat p) / 2^k)* **by** *algebra*
**then have** *(abs ((?u − ?w)$i))≤(rat-of-nat p) / 2^k*
**using** *of-rat-less-eq* **by** *blast*
**then show** *?thesis* **by** *auto*
**next**
**case** *t:True*
**have** *?u ∈ carrier-vec (d + 1)* **using** *ts-to-u-carrier[of ts] assms* **by** *blast*
**moreover have** *?w ∈ carrier-vec (d + 1)*
**using** *in-lattice gen-lattice-carrier[of ts] assms* **by** *blast*
**ultimately have** *(?u − ?w)$i = ?u$i − ?w$i* **using** *in-range* **by** *fastforce*
**also have** *?u$i = (map (rat-of-nat ∘ msb-p ∘ (λt. α ∗ t mod p)) ts @ [0])!i*
**unfolding** *ts-to-u-alt* **using** *t assms* **by** *auto*
**also have** *(map (rat-of-nat ∘ msb-p ∘ (λt. α ∗ t mod p)) ts @ [0])!i = 0*
**unfolding** *ts-to-u-def*
**using** *append-Cons-nth-middle[of i (map (rat-of-nat ∘ msb-p ∘ (λt. α ∗ t mod p)) ts) 0 []*
*t assms*
**by** *fastforce*
**also have** *?w$i = (rat-of-int (int-of-nat α))/(rat-of-nat p)*
**using** *coordinates-of-gen-lattice[of i ts λi. (if i = d then int-of-nat α else − int-of-nat (α ∗ ts ! i div p))]*
*t assms* **by** *auto*
**finally have** *(?u−?w)$i = −(rat-of-int (int-of-nat α))/(rat-of-nat p)* **by** *linarith*
**moreover have** *rat-of-int (int-of-nat α) = rat-of-nat α*
**using** *int-of-nat-def* **by** *fastforce*
**moreover have** *α < p* **using** *α* **by** *auto*
**ultimately have** *abs((?u−?w)$i) ≤ 1* **by** *force*
**moreover have** *1 ≤ rat-of-nat p / rat-of-nat 2^k*
**proof**−
**have** *k < log 2 p* **using** *k-small* **by** *linarith*
**then have** *2 powr k < 2 powr (log 2 p)* **by** *force*
**then have** *2^k < p* **using** *powr-realpow[of 2 k] p prime-gt-0-nat[of p]* **by** *force*
**then show** *1 ≤ rat-of-nat p / rat-of-nat 2^k* **by** *force*
**qed**
**ultimately show** *?thesis* **by** *order*
**qed**
**define** *Lst* **where** *Lst = (list-of-vec (?u − ?w))*
**have** *Lst!i = (?u − ?w)$i* **if** *i∈{0..<d+1}* **for** *i*
**using** *dim-u dim-w l list-of-vec-index* **unfolding** *Lst-def* **by** *blast*
**then have** *abs-small*: *abs(Lst!i) ≤ rat-of-nat p / rat-of-nat 2^k* **if** *in-range*: *i∈{0..<d+1}* **for** *i*
**using** *in-range coords-close* **by** *presburger*
**then have** *small-coord*: *sq-norm (Lst!i) ≤ (rat-of-nat p / rat-of-nat 2^k)^2* **if** *in-range*: *i∈{0..<d+1}* **for** *i*
**proof**−
**have** *sq-norm (Lst!i) = (Lst!i) ^2*

**by** (*simp add*: *power2-eq-square*)
**moreover have** (*Lst*!*i*)^2 = *abs*(*Lst*!*i*)^2 **by** *auto*
**moreover have** *0*≤*abs*(*Lst*!*i*) **by** *simp*
**ultimately show** *?thesis* **using** *abs-small*[*of i*] *in-range*
**by** (*metis Power.linordered-semidom-class.power-mono*)
**qed**
**moreover have** *length*: *length Lst* = *d* + *1* **unfolding** *Lst-def* **using** *dim-u dim-w* **by** *auto*
**ultimately have** ⋀*x*. *x* ∈ *set Lst* ⟹ *sq-norm x* ≤ (*rat-of-nat p* / *rat-of-nat* 2^*k*)^2
**proof**−
**fix** *x* **assume** *x* ∈ *set Lst*
**then obtain** *y* **where** *x-def*: *x* = *Lst* ! *y* ∧ *y*∈ {*0*..<*length Lst*}
**by** (*metis atLeastLessThan-iff in-set-conv-nth le0*)
**then have** *sq-norm* (*Lst* ! *y*) ≤ (*rat-of-nat p* / *rat-of-nat* 2^*k*)^2
**using** *length small-coord*[*of y*] **by** *argo*
**then show** *sq-norm x* ≤ (*rat-of-nat p* / *rat-of-nat* 2^*k*)^2 **using** *x-def* **by** *blast*

**qed**
**moreover have** *sq-norm* (*?u* − *?w*) = *sum-list* (*map sq-norm* (*list-of-vec* (*?u* − *?w*)))
**using** *sq-norm-vec-def*[*of ?u* − *?w*] **by** *fast*
**ultimately have** *sq-norm* (*?u* − *?w*) ≤ *sum-list* (*map* (λ*x*. (*rat-of-nat p* / *rat-of-nat* 2^*k*)^2) (*list-of-vec* (*?u* − *?w*)))
**unfolding** *Lst-def* **using** *sum-list-mono*[*of* (*list-of-vec* (*?u* − *?w*)) *sq-norm* (λ*x*. (*rat-of-nat p* / *rat-of-nat* 2^*k*)^2)]
**by** *argo*
**also have** *sum-list* (*map* (λ*x*. (*rat-of-nat p* / *rat-of-nat* 2^*k*)^2) (*list-of-vec* (*?u* − *?w*)))
= *rat-of-nat* (*d* + *1*) ∗ (*rat-of-nat p* / *rat-of-nat* 2^*k*)^2
**using** *length sum-list-triv*[*of* (*rat-of-nat p* / *rat-of-nat* 2^*k*)^2 (*list-of-vec* (*?u* − *?w*))]
**unfolding** *Lst-def*
**by** *argo*
**finally have** *sq-norm* (*?u* − *?w*) ≤ *rat-of-nat* (*d* + *1*) ∗ (*rat-of-nat p* / *rat-of-nat* 2^*k*)^2
**by** *blast*
**also have** *rat-of-nat* (*d* + *1*) ∗ (*rat-of-nat p* / *rat-of-nat* 2^*k*)^2
= *rat-of-nat* (*d* + *1*) ∗ ((*rat-of-nat p*)^2 / (*rat-of-nat* 2^*k*)^2 )
**by** (*simp add*: *power-divide*)
**also have** *rat-of-nat* (*d* + *1*) ∗ ((*rat-of-nat p*)^2 / (*rat-of-nat* 2^*k*)^2 )
= (*rat-of-nat* (*d* + *1*) ∗ ((*rat-of-nat p*)^2)) / (*rat-of-nat* 2^*k*)^2
**by** *simp*
**also have** (*rat-of-nat* (*d* + *1*) ∗ ((*rat-of-nat p*)^2)) / (*rat-of-nat* 2^*k*)^2
= *rat-of-nat* ((*d* + *1*) ∗ *p*^2) / ((2^*k*)^2)
**by** (*metis Power.semiring-1-class.of-nat-power of-nat-numeral of-nat-simps*(*5*))
**also have** *rat-of-nat* ((*d* + *1*) ∗ *p*^2) / ((2^*k*)^2) = *rat-of-nat* ((*d* + *1*) ∗ *p*^2) / ((2^(*2*∗*k*)))
**by** (*simp add*: *power-even-eq*)

**finally show** *?thesis* **using** *in-lattice* **by** *force*
**qed**

**lemma** *gen-lattice-dim*:
**assumes** $ts \in$ *set-pmf ts-pmf*
**assumes** $v \in$ *gen-lattice ts*
**shows** *dim-vec* $v = d + 1$
**using** *assms set-pmf-ts carrier-dim-vec gen-lattice-carrier gen-basis-carrier*
**unfolding** *gen-lattice-def lattice-of-def*
**by** *fast*

**lemma** *vec-class-union*:
**fixes** *ts* :: *nat list*
**assumes** $ts \in$ *set-pmf ts-pmf*
**defines** $L \equiv$ *gen-lattice ts*
**shows** $L = \bigcup \{$*vec-class ts* $\beta \mid \beta.$ *True*$\}$
**proof** *safe*
**let** *?*$\mathcal{B}$ = *gen-basis ts*
**have** *length-*$\mathcal{B}$: *length ?*$\mathcal{B} = d + 1$ **using** *gen-basis-length* **.**
**have** *length-ts*: *length ts* $= d$ **using** *assms*(*1*) *set-pmf-ts* **by** *blast*
**fix** $x$ **assume** $*$: $x \in L$
**then obtain** *cs* **where** *cs*: $x =$ *sumlist* (*map* ($\lambda i.$ *of-int* (*cs i*) $\cdot_v$ *?*$\mathcal{B}$*!i*) [*0..<length ?*$\mathcal{B}$])
**unfolding** *L-def gen-lattice-def* **using** *in-latticeE* **by** *blast*
**define** *xs* **where** *xs* $\equiv$ (*map* ($\lambda i.$ *of-int* (*cs i*) $\cdot_v$ *?*$\mathcal{B}$*!i*) [*0..<length ?*$\mathcal{B}$])
**have** *x*: $x =$ *sumlist xs* **unfolding** *xs-def cs* **by** *blast*

**define** $\beta$ **where** $\beta \equiv$ *cs d*
**have** $x \in$ *vec-class ts* $\beta$
**proof**−
**let** *?I* = [*0..<length ?*$\mathcal{B} - 1$]
**define** *as* **where** *as* $\equiv$ (*map* ($\lambda i.$ *of-int* (*cs i*) $\cdot_v$ *?*$\mathcal{B}$*!i*) [*0..<length ?*$\mathcal{B} - 1$])
**define** *bs* **where** *bs* $\equiv$ (*map* ($\lambda i.$ *of-int* (*cs i*) $\cdot_v$ *?*$\mathcal{B}$*!i*) [*length ?*$\mathcal{B} - 1$])
**define** *a* **where** *a* $\equiv$ *sumlist as*
**define** *b* **where** *b* $\equiv$ *sumlist bs*
**have** $\forall\, i <$ *length ?*$\mathcal{B}$. *?*$\mathcal{B}$*!i* $\in$ *carrier-vec* $(d + 1)$
**using** *gen-basis-carrier length-ts nth-mem* **by** *blast*
**hence** *c-*$\mathcal{B}$*-dims*: $\forall\, i <$ *length ?*$\mathcal{B}$. *of-int* (*cs i*) $\cdot_v$ *?*$\mathcal{B}$*!i* $\in$ *carrier-vec* $(d + 1)$ **by** *blast*
**hence** *as-dims*: *set as* $\subseteq$ *carrier-vec* $(d + 1)$ **unfolding** *as-def* **by** *auto*
**have** *bs-dims*: *set bs* $\subseteq$ *carrier-vec* $(d + 1)$
**apply** (*simp add*: *bs-def*)
**using** *c-*$\mathcal{B}$*-dims* **by** (*simp add*: *length-*$\mathcal{B}$)

**have** *a-carr*: $a \in$ *carrier-vec* $(d + 1)$ **using** *a-def as-dims sumlist-carrier* **by** *presburger*
**moreover have** *b-carr*: $b \in$ *carrier-vec* $(d + 1)$ **using** *b-def bs-dims sumlist-carrier* **by** *blast*
**ultimately have** *ab-comm*: $a + b = b + a$ **using** *a-ac*(*2*) *a-carr b-carr* **by**

*presburger*

**have** $b = sumlist\ [(of\text{-}int\ (cs\ (length\ ?\mathcal{B} - 1)) \cdot_v\ ?\mathcal{B}!(length\ ?\mathcal{B} - 1))]$
**unfolding** *b-def bs-def* **by** *force*
**hence** *b*: $b = (of\text{-}int\ (cs\ (length\ ?\mathcal{B} - 1)) \cdot_v\ ?\mathcal{B}!(length\ ?\mathcal{B} - 1))$
**by** (*metis cV diff-add-inverse2 gen-basis-carrier length-*$\mathcal{B}$ *length-ts less-add-one local.M.add.l-cancel-one local.M.add.one-closed nth-mem smult-closed sum-simp sum-list-Cons sumlist-Nil*)

**have** $x = a + b$
**proof**−
**have** $xs = as\ @\ bs$ **by** (*simp add*: *length-*$\mathcal{B}$ *xs-def as-def bs-def*)
**thus** *?thesis* **using** *sumlist-append*[*OF as-dims bs-dims*] **unfolding** *x a-def b-def* **by** *blast*
**qed**
**hence** *1*: $x = b + a$ **using** *ab-comm* **by** *blast*
**have** *2*: $a = lincomb\text{-}list\ (rat\text{-}of\text{-}int \circ cs)\ p\text{-}vecs$ (**is** $a = ?rhs$)
**proof**−
**have** $\forall\, i < length\ p\text{-}vecs.\ p\text{-}vecs!i = ?\mathcal{B}!i$
**unfolding** *gen-basis-def* **using** *length-p-vecs* **by** (*simp add*: *append-Cons-nth-left*)
**hence** $\forall\, i < length\ p\text{-}vecs.\ of\text{-}int\ (cs\ i) \cdot_v\ p\text{-}vecs\ !\ i = of\text{-}int\ (cs\ i) \cdot_v\ ?\mathcal{B}!i$
**by** *presburger*
**hence** $(map\ (\lambda i.\ of\text{-}int\ (cs\ i) \cdot_v\ p\text{-}vecs\ !\ i)\ [0..<length\ p\text{-}vecs])$
$= (map\ (\lambda i.\ of\text{-}int\ (cs\ i) \cdot_v\ ?\mathcal{B}!i)\ [0..<length\ p\text{-}vecs])$
**by** *simp*
**moreover have** $?rhs = sumlist\ (map\ (\lambda i.\ of\text{-}int\ (cs\ i) \cdot_v\ p\text{-}vecs\ !\ i)\ [0..<length\ p\text{-}vecs])$
**using** *lincomb-list-def*[*of rat-of-int* $\circ$ *cs p-vecs*] **by** *simp*
**ultimately have** $?rhs = sumlist\ (map\ (\lambda i.\ of\text{-}int\ (cs\ i) \cdot_v\ ?\mathcal{B}!i)\ [0..<length\ p\text{-}vecs])$
**by** *argo*
**thus** *?thesis* **unfolding** *a-def as-def length-*$\mathcal{B}$ *length-p-vecs* **by** *force*
**qed**
**have** *3*: $b = rat\text{-}of\text{-}int\ \beta \cdot_v\ t\text{-}vec\ ts$
**proof**
**show** *dims*: $dim\text{-}vec\ b = dim\text{-}vec\ (rat\text{-}of\text{-}int\ \beta \cdot_v\ t\text{-}vec\ ts)$
**using** *b-carr length-ts t-vec-dim* **by** *auto*
**show** $\bigwedge i.\ i < dim\text{-}vec\ (rat\text{-}of\text{-}int\ \beta \cdot_v\ t\text{-}vec\ ts) \Longrightarrow b\ \$\ i = (rat\text{-}of\text{-}int\ \beta \cdot_v\ t\text{-}vec\ ts)\ \$\ i$
**proof**−
**fix** *i* **assume** ∗: $i < dim\text{-}vec\ (rat\text{-}of\text{-}int\ \beta \cdot_v\ t\text{-}vec\ ts)$
**show** $b\ \$\ i = (rat\text{-}of\text{-}int\ \beta \cdot_v\ t\text{-}vec\ ts)\ \$\ i$
**proof**(*cases* $i = d$)
**case** *True*
**hence** $b\$i = (of\text{-}int\ (cs\ (length\ ?\mathcal{B} - 1)) \cdot_v\ ?\mathcal{B}!(length\ ?\mathcal{B} - 1))\$i$ **unfolding** *b* **by** *blast*
**also have** $\ldots = (of\text{-}int\ (cs\ (length\ ?\mathcal{B} - 1)) * (?\mathcal{B}!(length\ ?\mathcal{B} - 1))\$i)$
**using** *dims* ∗ *b* **by** *auto*
**also have** $\ldots = (of\text{-}int\ (cs\ (length\ ?\mathcal{B} - 1)) * (last\ ?\mathcal{B})\$i)$

**by** (*metis length-$\mathcal{B}$ True last-conv-nth length-0-conv less-add-one not-less0*)
**also have** ... = (*of-int* (*cs* (*length ?$\mathcal{B}$* − *1*)) ∗ (*1* / *of-nat p*))
**using** *length-ts* **by** (*simp add*: *gen-basis-def True append-Cons-nth-middle*)
**also have** ... = *of-int* $\beta$ / *of-nat p* **by** (*simp add*: $\beta$*-def length-$\mathcal{B}$*)
**finally show** *?thesis*
**by** (*metis t-vec-def List.list.discI List.list.size*(*4*) *One-nat-def* $\beta$*-def b diff-add-inverse2 gen-basis-def last-conv-nth last-snoc length-0-conv length-$\mathcal{B}$ length-ts*)
**next**
**case** *False*
**hence** *i* < *d*
**by** (*metis* ∗ *Suc-eq-plus1 b-carr carrier-vecD dims less-Suc-eq*)
**hence** (*?$\mathcal{B}$*!*d*)$*i* = *of-nat* (*ts*!*i*)
**by** (*simp add*: *gen-basis-def append-Cons-nth-left append-Cons-nth-middle length-p-vecs length-ts*)
**thus** *?thesis*
**apply** (*simp add*: *b length-$\mathcal{B}$ t-vec-def* $\beta$*-def*)
**by** (*metis gen-basis-def length-p-vecs nth-append-length*)
**qed**
**qed**
**qed**
**have** *x* = (*rat-of-int* $\beta$ ·$_v$ *t-vec ts*) + (*lincomb-list* (*rat-of-int* ∘ *cs*) *p-vecs*)
**unfolding** *1 2 3* **by** *blast*
**thus** *?thesis* **unfolding** *vec-class-def* **by** *blast*
**qed**
**thus** *x* ∈ ⋃{*vec-class ts* $\beta$ | $\beta$. *True*} **by** *blast*
**next**
**fix** *x* $\beta$
**assume** *x* ∈ *vec-class ts* $\beta$
**then obtain** *cs* **where** *cs*: *x* = *of-int* $\beta$ ·$_v$ *t-vec ts* + *lincomb-list* (*of-int* ∘ *cs*) *p-vecs*
**unfolding** *vec-class-def* **by** *blast*
**define** *cs′* **where** *cs′* = (λ*x*. *if x* < *d then cs x else* $\beta$)

**have** *x* = *sumlist* (*map* (λ*i*. *rat-of-int* (*cs′ i*) ·$_v$ *gen-basis ts* ! *i*) [*0*..<*length* (*gen-basis ts*)])
(**is** *x* = *?sum*)
**proof**−
**let** *?f′* = λ*i*. *rat-of-int* (*cs′ i*) ·$_v$ *gen-basis ts* ! *i*
**let** *?f* = λ*i*. *rat-of-int* (*cs i*) ·$_v$ *p-vecs* ! *i*
**let** *?I* = [*0*..<*length* (*gen-basis ts*)]
**let** *?I1* = [*0*..<*length* (*gen-basis ts*) − *1*]
**let** *?I2* = [*length* (*gen-basis ts*) − *1*]

**have** *length-ts*: *length ts* = *d* **using** *assms*(*1*) *set-pmf-ts* **by** *blast*
**have** *I*: *?I* = *?I1* @ *?I2* **using** *gen-basis-length* **by** *auto*
**have** *I-dims*: *set* (*map ?f′ ?I*) ⊆ *carrier-vec* (*d* + *1*)
**using** *gen-basis-carrier*[*OF length-ts*] *gen-basis-length*[*of ts*] **by** *fastforce*
**have** *I1-dims*: *set* (*map ?f′ ?I1*) ⊆ *carrier-vec* (*d* + *1*) **using** *I I-dims* **by** *simp*
**have** *I2-dims*: *set* (*map ?f′ ?I2*) ⊆ *carrier-vec* (*d* + *1*) **using** *I I-dims* **by** *simp*

```
    have sum1-dim: sumlist (map ?f' ?I1) ∈ carrier-vec (d + 1)
      using I1-dims sumlist-carrier by blast
    have sum2-dim: sumlist (map ?f' ?I2) ∈ carrier-vec (d + 1)
      using I2-dims sumlist-carrier by blast

    from I have map ?f' ?I = (map ?f' ?I1) @ (map ?f' ?I2) by auto
    hence sumlist (map ?f' ?I) = sumlist (map ?f' ?I1) + sumlist (map ?f' ?I2)
      using sumlist-append[of map ?f' ?I1 map ?f' ?I2] I1-dims I2-dims by argo
    hence sumlist (map ?f' ?I) = sumlist (map ?f' ?I2) + sumlist (map ?f' ?I1)
      using sum1-dim sum2-dim a-ac(2) by presburger
    moreover have sumlist (map ?f' ?I1) = lincomb-list (of-int ∘ cs) p-vecs
    proof−
      have ∀ i < d. gen-basis ts ! i = p-vecs ! i
        by (simp add: nth-append length-p-vecs gen-basis-def)
      moreover have ∀ i < d. cs i = cs' i using cs'-def by presburger
      ultimately have ∀ i < d. ?f' i = ?f i by presburger
      hence map ?f' [0..<d] = map ?f [0..<d] by fastforce
      hence sumlist (map ?f' [0..<d]) = sumlist (map ?f [0..<d]) by argo
     thus ?thesis unfolding lincomb-list-def length-p-vecs gen-basis-length by auto
    qed
    moreover have sumlist (map ?f' ?I2) = of-int β ·v t-vec ts
    proof−
      have sumlist (map ?f' ?I2) = ?f' (length (gen-basis ts) − 1)
        unfolding sumlist-def
      using Suc-eq-plus1 I assms(1) diff-add-inverse2 diff-diff-left diff-zero gen-basis-length
gen-basis-vecs-carrier length-upt lessI mem-Collect-eq nth-append-length nth-mem
right-zero-vec set-pmf-ts set-upt
        by auto
     also have ... = of-int (cs' d) ·v gen-basis ts ! d by (simp add: gen-basis-length)
      also have ... = of-int β ·v gen-basis ts ! d unfolding cs'-def by auto
      also have ... = of-int β ·v t-vec ts
       by (simp add: append-Cons-nth-middle length-p-vecs t-vec-def gen-basis-def)
      finally show ?thesis .
    qed
    ultimately show ?thesis using cs by argo
  qed
  thus x ∈ L unfolding L-def vec-class-def gen-lattice-def lattice-of-def by blast
qed

lemma vec-class-mod-p-union:
  fixes ts :: nat list
  assumes ts ∈ set-pmf ts-pmf
  defines L ≡ gen-lattice ts
  shows L = ⋃{vec-class-mod-p ts β | β. β ∈ {0..<p::int}} (is - = ?rhs)
proof
  show L ⊆ ?rhs
  proof
    fix x assume x ∈ L
```

**then obtain** $\beta$ **where** $\beta$: $x \in$ *vec-class ts* $\beta$
**using** *vec-class-union*[*OF assms*(*1*)] **unfolding** *L-def* **by** *blast*
**define** $\beta_p$ **where** $\beta_p \equiv \beta$ *mod p*
**hence** $\beta_p \in \{0..<p::int\}$
**by** (*metis Nat.bot-nat-0.not-eq-extremum mod-ident-iff mod-mod-trivial not-prime-0 of-nat-0-less-iff p*)
**moreover have** $x \in$ *vec-class-mod-p ts* $\beta_p$
**unfolding** *vec-class-mod-p-def* $\beta_p$*-def*
**by** (*smt* (*verit*, *best*) *UnionI* $\beta$ *cong-mod-right cong-refl mem-Collect-eq*)
**ultimately show** *?rhs* **by** *blast*
**qed**
**show** *?rhs* $\subseteq L$ **using** *vec-class-union*[*OF assms*(*1*)] **unfolding** *L-def vec-class-mod-p-def* **by** *blast*
**qed**

### 11.2.2 dist-p definition and lemmas

**definition** *dist-p* :: *int* $\Rightarrow$ *int* $\Rightarrow$ *int* **where**
*dist-p i j* = *Inf* {*abs* (*i* − *j* + *z* ∗ (*of-nat p*)) | *z*. *True*}

**lemma** *dist-p-well-defined*:
**fixes** *i* :: *int*
**fixes** *j* :: *int*
**shows** *dist-p i j* ∈ {*abs* (*i* − *j* + *z* ∗ (*of-nat p*)) | *z*. *True*} (**is** - ∈ *?S*)
**proof**−
**have** *?S* ⊆ **N** **by** (*smt* (*verit*, *del-insts*) *mem-Collect-eq range-abs-Nats range-eqI subsetI*)
**moreover have** *?S* ≠ {} **by** *blast*
**ultimately show** *?thesis* **using** *nat-subset-inf* **unfolding** *dist-p-def* **by** *algebra*
**qed**

**lemma** *dist-p-set-bdd-below*:
**fixes** *i j* :: *int*
**shows** ∀ *x* ∈ {*abs* (*i* − *j* + *z* ∗ (*of-nat p*)) | *z*. *True*}. *0* ≤ *x*
**by** *force*

**lemma** *dist-p-le*:
**fixes** *i j* :: *int*
**shows** ∀ *x* ∈ {*abs* (*i* − *j* + *z* ∗ (*of-nat p*)) | *z*. *True*}. *dist-p i j* ≤ *x* (**is** ∀ *x* ∈ *?S*. -)
**proof**
**fix** *x* **assume** ∗: *x* ∈ *?S*
**have** *bdd-below ?S* **using** *dist-p-set-bdd-below* **by** *fast*
**thus** *dist-p i j* ≤ *x*
**unfolding** *dist-p-def* **using** *dist-p-well-defined*[*of i j*] *cInf-lower*[*of x ?S*] ∗ **by** *fast*
**qed**

**lemma** *dist-p-equiv*:

**fixes** $i :: int$
**fixes** $j :: int$
**shows** $[dist\text{-}p\ i\ j = i - j]\ (mod\ p) \vee [dist\text{-}p\ i\ j = j - i]\ (mod\ p)$
**proof**−
**let** $?S = \{abs\ (i - j + z * (of\text{-}nat\ p)) \mid z.\ True\}$
**have** $[x = i - j]\ (mod\ p) \vee [x = j - i]\ (mod\ p)$ **if** *x-in*: $x \in ?S$ **for** $x$
**proof**−
**obtain** $z$ **where** $x = abs\ (i - j + z * (of\text{-}nat\ p))$ **using** *x-in* **by** *blast*
**then have** $x = i - j + z * (of\text{-}nat\ p) \vee x = -\ (i - j + z * (of\text{-}nat\ p))$ **by** *linarith*
**then show** *?thesis*
**proof**(*rule disjE*)
**assume** *left*: $x = i - j + z * (of\text{-}nat\ p)$
**then have** $x - (i - j) = z * (of\text{-}nat\ p)$ **by** *force*
**also have** $[z * (of\text{-}nat\ p) = 0]\ (mod\ p)$ **using** *cong-mult-self-right*[*of z of-nat p*] **by** *blast*
**finally have** $[x - (i - j) = 0]\ (mod\ p)$ **by** *blast*
**then have** $[x = i - j]\ (mod\ p)$ **using** *cong-diff-iff-cong-0*[*of x i − j int p*] **by** *presburger*
**then show** *?thesis* **by** *blast*
**next**
**assume** *right*: $x = -\ (i - j + z * (of\text{-}nat\ p))$
**then have** $x = j - i - z * (of\text{-}nat\ p)$ **by** *linarith*
**then have** $x - (j - i) = -\ z * (of\text{-}nat\ p)$ **by** *force*
**also have** $[-\ z * (of\text{-}nat\ p) = 0]\ (mod\ p)$ **using** *cong-mult-self-right*[*of −z of-nat p*] **by** *blast*
**finally have** $[x - (j - i) = 0]\ (mod\ p)$ **by** *blast*
**then have** $[x = j - i]\ (mod\ p)$ **using** *cong-diff-iff-cong-0*[*of x j − i int p*] **by** *presburger*
**then show** *?thesis* **by** *blast*
**qed**
**qed**
**then show** *?thesis* **using** *dist-p-well-defined*[*of i j*] **by** *presburger*
**qed**

**lemma** *dist-p-equiv′*:
**fixes** $i :: int$
**fixes** $j :: int$
**shows** $dist\text{-}p\ i\ j = min\ ((i - j)\ mod\ p)\ ((j - i)\ mod\ p)$
**proof**−
**let** $?S = \{abs\ (i - j + z * (of\text{-}nat\ p)) \mid z.\ True\}$
**have** $[dist\text{-}p\ i\ j = i - j]\ (mod\ p) \vee [dist\text{-}p\ i\ j = j - i]\ (mod\ p)$ **using** *dist-p-equiv*
.
**moreover have** $0 \leq dist\text{-}p\ i\ j$ **using** *dist-p-well-defined*[*of i j*] **by** *force*
**moreover have** $dist\text{-}p\ i\ j < p$
**proof**(*rule ccontr*)
**obtain** $k$ **where** $k$: $dist\text{-}p\ i\ j = |i - j + k * int\ p|$
**using** *dist-p-well-defined*[*of i j*] **by** *blast*
**hence** *k-min*: $(\forall\, k'.\ dist\text{-}p\ i\ j \leq |i - j + k' * int\ p|)$

```
  using dist-p-well-defined[of i j] by (metis (mono-tags, lifting) dist-p-le mem-Collect-eq)

  assume ¬ dist-p i j < int p
  hence *: dist-p i j ≥ p by linarith
  show False
  proof(cases i − j + k * int p ≥ 0)
    case True
    hence dist-p i j = i − j + k * p using k by fastforce
    moreover then have 0 ≤ i − j + k * p − p using * by simp
   moreover then have ... = |i − j + (k − 1) * p| by (simp add: left-diff-distrib')
    moreover have p > 0 using p prime-gt-0-nat by blast
    ultimately have dist-p i j > |i − j + (k − 1) * p| by fastforce
    moreover have dist-p i j ≤ |i − j + (k − 1) * p| using k-min by blast
    ultimately show False by fastforce
  next
    case False
    hence dist-p i j = − i + j − k * p using k by linarith
    moreover then have 0 ≤ − i + j − k * p − p using * by simp
    moreover then have ... = |i − j + (k + 1) * p|
      by (smt (verit, ccfv-SIG) left-diff-distrib' mult-cancel-right2)
    moreover have p > 0 using p prime-gt-0-nat by blast
    ultimately have dist-p i j > |i − j + (k + 1) * p| by fastforce
    moreover have dist-p i j ≤ |i − j + (k + 1) * p| using k-min by blast
    ultimately show False by fastforce
  qed
 qed
 ultimately have *: dist-p i j = (i − j) mod p ∨ dist-p i j = (j − i) mod p
  by (simp add: Cong.unique-euclidean-semiring-class.cong-def)

 have (i − j) mod p ∈ ?S
 proof−
  obtain k :: int where (i − j) mod p = (i − j) + k * p
   by (metis mod-eqE mod-mod-trivial mult-of-nat-commute)
  moreover have (i − j) mod p ≥ 0 using prime-gt-1-nat[OF p] by simp
  ultimately have (i − j) mod p = |i − j + k * p| by force
  thus ?thesis by auto
 qed
 moreover have (j − i) mod p ∈ ?S
 proof−
  obtain k :: int where (j − i) mod p = (j − i) + k * p
   by (metis mod-eqE mod-mod-trivial mult-of-nat-commute)
  hence (j − i) mod p = − ((i − j) − k * p) by simp
  moreover have (j − i) mod p ≥ 0 using prime-gt-1-nat[OF p] by simp
  ultimately have (j − i) mod p = |i − j − k * p| by force
  then obtain k' :: int where (j − i) mod p = |(i − j) + k' * p| using that[of
− k] by force
  thus ?thesis by blast
 qed
 ultimately have dist-p i j ≤ (i − j) mod p ∧ dist-p i j ≤ (j − i) mod p
```

```
    by (metis dist-p-le)
  thus ?thesis using * by linarith
qed

lemma dist-p-equiv'':
  fixes i :: int
  fixes j :: int
  shows dist-p i j = ((i - j) mod p) ∨ dist-p i j = ((j - i) mod p)
  using dist-p-equiv'[of i j] by linarith

lemma dist-p-instances-help:
  fixes i :: int
  fixes j :: int
  fixes dist :: int
  assumes coprime (i - j) p
  shows card {t ∈ {1..<p}. (dist-p (t*i) (t*j)) = dist} ≤ 2
proof-
  obtain ij-inv where ij-inv: [(i-j)*ij-inv = 1] (mod p)
    using assms(1) cong-solve-coprime-int[of (i - j) p] by auto
  have coprime (j - i) p
    by (meson assms(1) coprimeE coprimeI dvd-diff-commute)
  then obtain ji-inv where ji-inv: [(j - i)*ji-inv = 1] (mod p)
    using cong-solve-coprime-int[of (j - i) p] by auto
  have disj: [t = dist*ij-inv] (mod p) ∨ [t = dist*ji-inv] (mod p)
     if dist-eq: (dist-p (t*i) (t*j)) = dist
     for t::int
  proof-
    have [dist = (t*i - t*j) ] (mod p) ∨ [dist = t*j - t*i] (mod p)
      using dist-p-equiv[of t*i t*j] dist-eq by simp
    then show ?thesis
    proof(rule disjE)
      assume left: [dist = (t*i - t*j) ] (mod p)
      have t * (i - j) = t * i - t * j
        using int-distrib(4) by auto
      then have [t * (i - j) = t * i - t * j] (mod p) by simp
      then have [dist = t * (i - j)] (mod p)
        using cong-trans[of t * (i - j) t * i - t * j p dist] left cong-sym
        by blast
      then have [dist * ij-inv = t * (i - j) * ij-inv] (mod p)
        using cong-mult[of dist t * (i - j) p ij-inv] by force
      moreover have [t * (i - j) * ij-inv = t * ((i - j) * ij-inv)] (mod p)
        by (simp add: more-arith-simps(11))
      moreover have [t * ((i - j) * ij-inv) = t] (mod p)
        using ij-inv cong-mult[of t t p (i - j) * ij-inv 1] by auto
      ultimately show ?thesis using cong-trans cong-sym by meson
    next
      assume right: [dist = t*j - t*i] (mod p)
      have t * (j - i) = t * j - t * i
        using int-distrib(4) by auto
```

**then have** $[t * (j - i) = t * j - t * i]$ $(mod\ p)$ **by** *simp*
**then have** $[dist = t * (j - i)]$ $(mod\ p)$
**using** *cong-trans*[*of* $t * (j - i)$ $t * j - t * i$ *p dist*] *right cong-sym*
**by** *blast*
**then have** $[dist * ji\text{-}inv = t * (j - i) * ji\text{-}inv]$ $(mod\ p)$
**using** *cong-mult*[*of dist* $t * (j - i)$ *p ji-inv*] **by** *force*
**moreover have** $[t * (j - i) * ji\text{-}inv = t * ((j - i) * ji\text{-}inv)]$ $(mod\ p)$
**by** (*simp add*: *more-arith-simps*(*11*))
**moreover have** $[t * ((j - i) * ji\text{-}inv) = t]$ $(mod\ p)$
**using** *ji-inv cong-mult*[*of t t p* $(j - i) * ji\text{-}inv$ *1*] **by** *auto*
**ultimately show** *?thesis* **using** *cong-trans cong-sym* **by** *meson*
**qed**
**qed**
**define** *S1* **where** $S1 = (\{t \in \{1..<p\}.\ (dist\text{-}p\ (t*i)\ (t*j)) = dist\} \cap \{t \in \{1..<p\}.\ [t = dist*ij\text{-}inv]\ (mod\ p)\})$
**define** *S2* **where** $S2 = (\{t \in \{1..<p\}.\ (dist\text{-}p\ (t*i)\ (t*j)) = dist\} \cap \{t \in \{1..<p\}.\ \neg\ ([t = dist*ij\text{-}inv]\ (mod\ p))\})$
**have** $\{t \in \{1..<p\}.\ (dist\text{-}p\ (t*i)\ (t*j)) = dist\} = S1 \cup S2$
**unfolding** *S1-def S2-def* **by** *blast*
**moreover have** *card1*: $card(S1) \leq 1$
**proof**(*rule ccontr*)
**assume** $\neg$ $card\ S1 \leq 1$
**hence** $*$: $card\ S1 \geq 2$ **by** *simp*
**obtain** *t1 t2* **where** *ts*: $t1 \neq t2 \wedge t1 \in S1 \wedge t2 \in S1$
**proof**−
**obtain** *t1 S1'* **where** *t1*: $t1 \in S1 \wedge S1' = S1 - \{t1\}$ **using** $*$ **by** *fastforce*
**hence** $card\ S1' \geq 1$ **using** $*$ **by** *fastforce*
**then obtain** *t2* **where** $t2 \in S1'$ **by** *fastforce*
**hence** $t1 \neq t2 \wedge t1 \in S1 \wedge t2 \in S1$ **using** *t1* **by** *blast*
**thus** *?thesis* **using** *that* **by** *blast*
**qed**
**have** *1*: $[t1 = dist*ij\text{-}inv]$ $(mod\ p)$ **using** *ts* **unfolding** *S1-def* **by** *simp*
**have** *2*: $[t2 = dist*ij\text{-}inv]$ $(mod\ p)$ **using** *ts* **unfolding** *S1-def* **by** *simp*
**have** $t1<p$ **using** *ts* **unfolding** *S1-def* **by** *simp*
**moreover have** $0 \leq t1$ **using** *ts* **unfolding** *S1-def* **by** *simp*
**moreover have** $t2<p$ **using** *ts* **unfolding** *S1-def* **by** *simp*
**moreover have** $0 \leq t2$ **using** *ts* **unfolding** *S1-def* **by** *simp*
**moreover have** $[int\ t1 = int\ t2]$ $(mod\ p)$
**using** *1 2 cong-sym*[*of t2 dist*∗*ij-inv p*] *cong-trans*[*of t1 dist*∗*ij-inv p t2*]
**by** *blast*
**ultimately have** $int\ t1 = int\ t2$
**using** *cong-less-imp-eq-int*[*of int t1 p int t2*] **by** *auto*
**then have** $t1 = t2$ **by** *simp*
**then show** *False* **using** *ts* **by** *auto*
**qed**
**moreover have** $card(S2) \leq 1$
**proof**(*rule ccontr*)
**assume** $\neg$ $card\ S2 \leq 1$
**hence** $*$: $card\ S2 \geq 2$ **by** *simp*

**obtain** *t1 t2* **where** *ts*: $t1 \neq t2 \wedge t1 \in S2 \wedge t2 \in S2$
**proof**−
**obtain** *t1 S2'* **where** *t1*: $t1 \in S2 \wedge S2' = S2 - \{t1\}$ **using** $*$ **by** *fastforce*
**hence** *card S2'* $\geq 1$ **using** $*$ **by** *fastforce*
**then obtain** *t2* **where** $t2 \in S2'$ **by** *fastforce*
**hence** $t1 \neq t2 \wedge t1 \in S2 \wedge t2 \in S2$ **using** *t1* **by** *blast*
**thus** *?thesis* **using** *that* **by** *blast*
**qed**
**have** $\neg[t1 = dist{*}ij\text{-}inv]\ (mod\ p)$ **using** *ts* **unfolding** *S2-def* **by** *simp*
**then have** *1*: $[t1 = dist{*}ji\text{-}inv]\ (mod\ p)$ **using** *disj ts* **unfolding** *S2-def* **by** *blast*
**have** $\neg[t2 = dist{*}ij\text{-}inv]\ (mod\ p)$ **using** *ts* **unfolding** *S2-def* **by** *simp*
**then have** *2*: $[t2 = dist{*}ji\text{-}inv]\ (mod\ p)$ **using** *disj ts* **unfolding** *S2-def* **by** *blast*
**have** $t1 < p$ **using** *ts* **unfolding** *S2-def* **by** *simp*
**moreover have** $0 \leq t1$ **using** *ts* **unfolding** *S2-def* **by** *simp*
**moreover have** $t2 < p$ **using** *ts* **unfolding** *S2-def* **by** *simp*
**moreover have** $0 \leq t2$ **using** *ts* **unfolding** *S2-def* **by** *simp*
**moreover have** $[int\ t1 = int\ t2]\ (mod\ p)$
**using** *1 2 cong-sym*[*of t2 dist*∗*ji-inv p*] *cong-trans*[*of t1 dist*∗*ji-inv p t2*]
**by** *blast*
**ultimately have** *int t1* = *int t2*
**using** *cong-less-imp-eq-int*[*of int t1 p int t2*] **by** *auto*
**then have** *t1* = *t2* **by** *simp*
**then show** *False* **using** *ts* **by** *auto*
**qed**
**ultimately show** *?thesis* **using** *card-Un-le*[*of S1 S2*] **by** *simp*
**qed**

**lemma** *dist-p-nat*:
**fixes** *i* :: *int*
**fixes** *j* :: *int*
**shows** *dist-p i j* $\in \mathbb{N}$
**proof**−
**let** *?S* = $\{abs\ (i - j + z * (of\text{-}nat\ p)) \mid z.\ True\}$
**have** *?S* $\subseteq \mathbb{N}$ **by** (*smt* (*verit*, *del-insts*) *mem-Collect-eq range-abs-Nats range-eqI subsetI*)
**then show** *?thesis* **using** *dist-p-well-defined*[*of i j*] **by** *blast*
**qed**

**lemma** *dist-p-nat'*:
**shows** *nat* (*dist-p i j*) = *dist-p i j*
**by** (*metis dist-p-nat*[*of i j*] *Nats-induct int-eq-iff*)

**lemma** *dist-p-instances*:
**fixes** *i* :: *int*
**fixes** *j* :: *int*
**fixes** *bound* :: *rat*
**assumes** $i \neq j$

**assumes** *coprime* $(i - j)$ $p$
**assumes** $0 < bound$
**shows** *rat-of-nat* $(card\ \{t \in \{1..<p\}.\ rat\text{-}of\text{-}int\ (dist\text{-}p\ (t*i)\ (t*j)) \leq bound\}) \leq 2*bound$
**proof**−
**let** $?f = \lambda t.\ (dist\text{-}p\ (t*i)\ (t*j))$
**define** $S$ **where** $S = \{t \in \{1..<p\}.\ rat\text{-}of\text{-}int\ (?f\ t) \leq bound\}$
**have** $S = \{t \in \{1..<p\}.\ rat\text{-}of\text{-}int\ (?f\ t) \leq bound \wedge (?f\ t) \in \mathbb{N}\}$
$\cup\ \{t \in \{1..<p\}.\ rat\text{-}of\text{-}int\ (?f\ t) \leq bound \wedge (?f\ t) \notin \mathbb{N}\}$
**unfolding** *S-def* **by** *fastforce*
**moreover have** $\{t \in \{1..<p\}.\ rat\text{-}of\text{-}int\ (?f\ t) \leq bound \wedge (?f\ t) \notin \mathbb{N}\} = \{\}$
**using** *dist-p-nat* **by** *force*
**ultimately have** $S = \{t \in \{1..<p\}.\ rat\text{-}of\text{-}int\ (?f\ t) \leq bound \wedge (?f\ t) \in \mathbb{N}\}$
**by** *fast*
**moreover have** $(rat\text{-}of\text{-}int\ (?f\ t) \leq bound \wedge (?f\ t) \in \mathbb{N})$
$\longleftrightarrow ((?f\ t) \in \{1..<floor(bound)+1\})$ (**is** $?P = ?Q$) **if** $t \in \{1..<p\}$ **for** $t$
**proof**
**assume** $*$: $?P$
**hence** $?f\ t < (floor\ bound) + 1$ **by** *linarith*
**moreover have** $1 \leq ?f\ t$
**proof**−
**have** $\forall x \in \{abs\ ((t*i) - (t*j) + z * (of\text{-}nat\ p)) \mid z.\ True\}.\ 1 \leq x$
**proof**
**fix** $x$ **assume** $x \in \{abs\ ((t*i) - (t*j) + z * (of\text{-}nat\ p)) \mid z.\ True\}$
**then obtain** $z$ **where** $z$: $x = abs\ ((t*i) - (t*j) + z * (of\text{-}nat\ p))$ **by** *blast*
**moreover have** $t \neq 0$ **using** *that* **by** *force*
**ultimately have** $x$: $x = abs(t * (i - j) + z * p)$ **using** *int-distrib*(4) **by** *presburger*
**have** $x \neq 0$
**proof**−
**have** *coprime t p*
**unfolding** *coprime-def'*
**proof** *clarify*
**fix** $r$ **assume** $*$: $r\ dvd\ t\ r\ dvd\ p$
**have** *is-unit r* $\vee$ $r = p$
**by** (*metis* $*(2)$ *One-nat-def dvd-1-iff-1 p prime-nat-iff*)
**moreover have** $r \neq p$ **using** $*(1)$ *that* **by** *auto*
**ultimately show** *is-unit r* **by** *blast*
**qed**
**moreover have** *coprime* $(i - j)$ $p$ **using** *assms*(2) **.**
**ultimately have** *coprime* $(t * (i - j))$ $p$ **using** $p$ **by** *auto*
**hence** $\neg\ (p\ dvd\ (t * (i - j)))$
**by** (*metis coprime-def' One-nat-def Suc-0-not-prime-nat abs-of-nat dvd-refl int-ops*(2) *nat-int p zdvd1-eq*)
**hence** $t * (i - j) \neq - z * p$ **by** *fastforce*
**hence** $t * (i - j) + z * p \neq 0$ **by** *fastforce*
**thus** *?thesis* **unfolding** $x$ **by** *simp*
**qed**
**moreover have** $x \geq 0$ **using** $z$ **by** *force*

```
      ultimately show 1 ≤ x by linarith
    qed
     thus ?thesis using dist-p-well-defined[of t∗i t∗j] unfolding dist-p-def by
blast
  qed
  ultimately show ?Q by force
 next
  assume ?Q
  thus ?P by (simp add: dist-p-nat le-floor-iff)
 qed
 ultimately have S = {t ∈ {1..<p}. ((?f t) ∈ {1..<floor(bound)+1})} by blast
 moreover have {1..<floor(bound)+1} = ⋃{{i}| i. i∈{1..<floor(bound)+1}}
by blast
 ultimately have S = ⋃{{t ∈ {1..<p}. (?f t) ∈ {dist}} |dist. dist∈{1..<floor(bound)+1}}
  by blast
 then have card S ≤ sum card {{t ∈ {1..<p}. (?f t) ∈ {dist}} |dist. dist∈{1..<floor(bound)+1}}
  using card-Union-le-sum-card[of {{t ∈ {1..<p}. (?f t) ∈ {dist}} |dist. dist∈{1..<floor(bound)+1}}]

  by blast
 moreover have card s ≤ 2 if s-def: s ∈ {{t ∈ {1..<p}. (?f t) ∈ {dist}} |dist.
dist∈{1..<floor(bound)+1}} for s
 proof−
  obtain dist where s = {t ∈ {1..<p}. (?f t) ∈ {dist}}
   using s-def by blast
  then show ?thesis using dist-p-instances-help[of i j dist] assms(2) by auto
 qed
 ultimately have card S ≤ sum (λs. 2) {{t ∈ {1..<p}. (?f t) ∈ {dist}} | dist.
dist ∈ {1..<floor(bound)+1}}
  using sum-mono[of {{t ∈ {1..<p}. (?f t) ∈ {dist}} |dist. dist∈{1..<floor(bound)+1}}
card (λs. 2)]
  by force
 moreover have sum (λs. 2) {{t ∈ {1..<p}. (?f t) ∈ {dist}} | dist. dist ∈
{1..<floor(bound)+1}}
     = 2 ∗ card{{t ∈ {1..<p}. (?f t) ∈ {dist}} | dist. dist ∈ {1..<floor(bound)+1}}
by force
 moreover have card{{t ∈ {1..<p}. (?f t) ∈ {dist}} | dist. dist ∈ {1..<floor(bound)+1}}
≤ card{1..<floor(bound)+1}
 proof−
  have finite{1..<floor(bound)+1} by blast
  moreover have (λdist. {t ∈ {1..<p}. (?f t) ∈ {dist}}) ‘ {1..<floor(bound)+1}

       = {{t ∈ {1..<p}. (?f t) ∈ {dist}} | dist. dist ∈ {1..<floor(bound)+1}}
    by blast
  ultimately show ?thesis
    using card-image-le[of {1..<floor(bound)+1} λdist. {t ∈ {1..<p}. (?f t) ∈
{dist}}]
    by algebra
 qed
 moreover have rat-of-nat (card{1..<floor(bound)+1}) ≤ bound
```

```
    using assms(3) by force
  ultimately show ?thesis
    using S-def mult-2 of-nat-simps(4) by auto
qed

lemma dist-p-smallest:
  fixes β ts i
  assumes ts ∈ set-pmf ts-pmf
  assumes v ∈ vec-class-mod-p ts β
  assumes i < d
  defines u ≡ ts-to-u ts
  shows (of-int (dist-p (int (ts ! i) * β) (int (ts-to-as ts ! i))))² ≤ ((u − v)$i)²
    (is (of-int ?d)² ≤ -)
proof−
  have len-ts: length ts = d using assms(1) by (simp add: set-pmf-ts)
 have v-carrier: v ∈ carrier-vec (d + 1) using assms(2) len-ts vec-class-mod-p-carrier
by blast
  have u-carrier: u ∈ carrier-vec (d + 1) using len-ts u-carrier u-def by blast
  obtain β' where β': v ∈ vec-class ts β' [β = β'] (mod p)
    using assms(2) unfolding vec-class-mod-p-def by blast
  let ?a = int (ts ! i) * β
  let ?a' = int (ts ! i) * β'
  let ?b = int (ts-to-as ts ! i)
  let ?S = {abs (?a − ?b + z * (of-nat p)) | z. True}
  let ?S' = {abs (?a' − ?b + z * (of-nat p)) | z. True}
  obtain cs where cs: v = sumlist (map (λi. rat-of-int (cs i) ·_v gen-basis ts ! i)
[0..<length (gen-basis ts)])
  proof−
    have v ∈ gen-lattice ts
      by (smt (verit, ccfv-SIG) vec-class-mod-p-union assms(1,2) mem-Collect-eq
mem-simps(9) vec-class-mod-p-def vec-class-union)
    thus ?thesis unfolding gen-lattice-def lattice-of-def using that by fast
  qed
  obtain z :: int where z: |(u − v)$i| = of-int |?a' − ?b + z * p|
  proof−
    have v$i = of-int (cs d * int (ts ! i) + cs i * int p)
      using coordinates-of-gen-lattice[of i ts cs] assms(3) len-ts unfolding cs by
auto
    moreover have u$i = of-int ?b
      using len-ts assms(3) by (simp add: append-Cons-nth-left ts-to-as-def u-def
ts-to-u-def)
    ultimately have (u − v)$i = of-int (?b − cs d * int (ts ! i) − cs i * int p)
      using v-carrier u-carrier assms(3) by simp
    moreover have cs d = β'
    proof−
      have v$d = rat-of-int (cs d) / rat-of-nat p
        using coordinates-of-gen-lattice[of d ts cs] len-ts unfolding cs by fastforce
      hence rat-of-int (cs d) = v$d * (rat-of-nat p) using p by simp
      moreover have (v$d) = rat-of-int β' / rat-of-nat p
```

```
      using vec-class-last[OF len-ts β'(1)] by simp
    ultimately show ?thesis using p by fastforce
  qed
  ultimately have (u − v)$i = of-int (?b − β' ∗ int (ts ! i) − cs i ∗ int p) by
blast
  hence −(u − v)$i = of-int (β' ∗ int (ts ! i) − ?b + cs i ∗ int p) by auto
  moreover have |−(u − v)$i| = |(u − v)$i| by fastforce
  ultimately have |(u − v)$i| = |of-int (β' ∗ int (ts ! i) − ?b + cs i ∗ int p)|
by presburger
  moreover have |rat-of-int (β' ∗ int (ts ! i) − int (ts-to-as ts ! i) + cs i ∗ int
p)|
      = rat-of-int |β' ∗ int (ts ! i) − int (ts-to-as ts ! i) + cs i ∗ int p|
    by linarith
  moreover have β' ∗ int (ts ! i) = int (ts ! i) ∗ β' by simp
  ultimately show ?thesis using that[of cs i] by argo
 qed
 let ?c = |?a' − ?b + z ∗ p|
 have ?c ∈ ?S' by blast
 moreover have ?d = Inf ?S by (rule dist-p-def)
 moreover have ?S = ?S'
 proof−
  have a-a': [?a − ?b = ?a' − ?b] (mod int p)
    using β'(2) cong-scalar-left cong-diff cong-refl by blast
  show ?thesis using cong-set-eq[OF a-a'] .
 qed
 ultimately have ?d ≤ ?c using dist-p-le by blast
 hence ?d^2 ≤ ?c^2
  by (meson dist-p-set-bdd-below dist-p-well-defined Power.linordered-semidom-class.power-mono)
 moreover have |(u − v)$i|^2 = ((u − v)$i)^2 by auto
 ultimately show ?thesis
  by (metis (mono-tags, lifting) of-int-le-of-int-power-cancel-iff of-int-power z)
qed

lemma dist-p-diff-helper:
 fixes a b b'
 assumes b ≥ b'
 shows |dist-p a b − dist-p a b'| ≤ b − b'
proof−
 let ?d = dist-p a b
 let ?d' = dist-p a b'
 obtain k where d: ?d = |a − b + k ∗ int p| using dist-p-well-defined[of a b]
by blast
 obtain k' where d': ?d' = |a − b' + k' ∗ int p| using dist-p-well-defined[of a
b'] by blast
 show ?thesis
 proof(cases ?d' ≤ ?d)
  case True
  have ∀ z :: int. 0 ≤ |a − b + z ∗ (of-nat p)| by fastforce
  hence bdd-below {|a − b + z ∗ (of-nat p)| | z. True} by fast
```

**hence** *?d* $\leq |a - b + k' * int\ p|$
**unfolding** *dist-p-def*
**using** *cInf-lower*[*of* $|a - b + k' * int\ p|$ $\{|a - b + z * (of\text{-}nat\ p)| \mid z.\ True\}$]
**by** *blast*
**also have** $\ldots = |a - b' + b' - b + k' * int\ p|$ **by** *auto*
**also have** $\ldots \leq$ *?d'* $+ |b' - b|$ **using** *d'* **by** *auto*
**finally show** *?thesis* **using** *True assms* **by** *simp*
**next**
**case** *False*
**have** $\forall z :: int.\ 0 \leq |a - b' + z * (of\text{-}nat\ p)|$ **by** *fastforce*
**hence** *bdd-below* $\{|a - b' + z * (of\text{-}nat\ p)| \mid z.\ True\}$ **by** *fast*
**hence** *?d'* $\leq |a - b' + k * int\ p|$
**unfolding** *dist-p-def*
**using** *cInf-lower*[*of* $|a - b' + k * int\ p|$ $\{|a - b' + z * (of\text{-}nat\ p)| \mid z.\ True\}$]
**by** *fast*
**also have** $\ldots = |a - b + b - b' + k * int\ p|$ **by** *auto*
**also have** $\ldots \leq$ *?d* $+ |b - b'|$ **using** *d* **by** *simp*
**finally show** *?thesis* **using** *False assms* **by** *simp*
**qed**
**qed**

**lemma** *dist-p-diff*:
**fixes** *a b b'*
**shows** $|$*dist-p a b* $-$ *dist-p a b'*$| \leq |b - b'|$
**apply**(*cases* $b \geq b'$)
**using** *dist-p-diff-helper*[*of b' b a*] **apply** *linarith*
**using** *dist-p-diff-helper*[*of b b' a*] **by** *linarith*

### 11.2.3 Uniquness lemma argument

**definition** *good-beta* **where**
*good-beta ts* $\beta \longleftrightarrow (\forall v \in$ *vec-class-mod-p ts* $\beta.$ *close-vec ts v* $\longrightarrow$ *good-vec v*)

**definition** *bad-beta* **where**
*bad-beta ts* $\beta \longleftrightarrow \neg$ *good-beta ts* $\beta$

**definition** *some-bad-beta* **where**
*some-bad-beta ts* $\longleftrightarrow (\exists \beta \in \{0..<p::int\}.$ *bad-beta ts* $\beta)$

**definition** *bad-beta-union* **where**
*bad-beta-union* $= (\bigcup \beta \in \{0..<p::int\}.\ \{ts.$ *bad-beta ts* $\beta\})$

**lemma** *reduction-1*:
**assumes** *ts* $\in$ *set-pmf ts-pmf*
**assumes** $\neg$ *good-lattice ts*
**shows** *some-bad-beta ts*
**using** *assms vec-class-mod-p-union set-pmf-ts*
**unfolding** *some-bad-beta-def bad-beta-def good-beta-def good-lattice-def*
**by** *blast*

**lemma** *reduction-1-pmf*:
*pmf sampled-lattice-good False ≤ pmf (ts-pmf ⤜ (λts. return-pmf (some-bad-beta ts))) True*
**proof**−
**have** *pmf sampled-lattice-good False = pmf (ts-pmf ⤜ (λts. return-pmf (¬ good-lattice ts))) True*
**unfolding** *sampled-lattice-good-def pmf-true-false* **by** *presburger*
**also have** *... ≤ pmf (ts-pmf ⤜ (λts. return-pmf (some-bad-beta ts))) True*
**using** *reduction-1 pmf-subset*[*of ts-pmf λts. ¬ good-lattice ts λts. some-bad-beta ts*]
**by** *blast*
**finally show** *?thesis* **.**
**qed**

**lemma** *reduction-2*: *{ts. some-bad-beta ts} ⊆ bad-beta-union*
**unfolding** *some-bad-beta-def bad-beta-union-def* **by** *blast*

**lemma** *reduction-2-pmf*:
*pmf (ts-pmf ⤜ (λts. return-pmf (ts ∈ {ts. some-bad-beta ts}))) True*
*≤ pmf (ts-pmf ⤜ (λts. return-pmf (ts ∈ bad-beta-union))) True*
**using** *reduction-2*
**using** *pmf-subset*[*of ts-pmf λts. ts ∈ {ts. some-bad-beta ts} λts. ts ∈ bad-beta-union*]
**by** *blast*

**lemma** *bad-beta-union-bound*:
*pmf (ts-pmf ⤜ (λts. return-pmf (ts ∈ bad-beta-union))) True*
*≤ (∑β ∈ {0..<p::int}. pmf (ts-pmf ⤜ (λts. return-pmf (ts ∈ {ts. bad-beta ts β})))) True*)
(**is** *?lhs ≤ ?rhs*)
**proof**−
**let** *?I = {0..<p::int}*
**let** *?B = λβ. {ts. bad-beta ts β}*
**let** *?M = ts-pmf*
**have** *bad-beta-union = (⋃β ∈ ?I. ?B β)*
**unfolding** *bad-beta-union-def* **by** *blast*
**hence** ∗: *?lhs = measure ts-pmf (⋃β ∈ ?I. ?B β)* **using** *pmf-in-set* **by** *metis*

**have** *1*: *finite ?I* **by** *auto*
**have** *2*: *?B ‘ ?I ⊆ sets ?M* **using** *sets-measure-pmf* **by** *blast*
**have** *?lhs ≤ (∑i∈?I. measure ?M (?B i))*
**unfolding** ∗ **using** *measure-pmf.finite-measure-subadditive-finite*[*OF 1 2*] **.**
**also have** *... = ?rhs*
**proof**−
**have** *⋀β. measure ?M (?B β)*
*= pmf (ts-pmf ⤜ (λts. return-pmf (ts ∈ {ts. bad-beta ts β}))) True*
**by** (*smt* (*verit*, *ccfv-SIG*) *bind-pmf-cong measure-pmf-single mem-Collect-eq pmf-in-set*)
**thus** *?thesis* **by** *presburger*

```
  qed
  finally show ?thesis .
qed

lemma fixed-beta-close-vec-dist-p:
  fixes β ts
  assumes ts ∈ set-pmf ts-pmf
  assumes v ∈ vec-class-mod-p ts β
  assumes close-vec ts v
  defines u ≡ ts-to-u ts
  shows ∀ i < d. real-of-int (dist-p (int (ts ! i) * β) (int (ts-to-as ts ! i))) < real
p / 2 ^ μ
proof clarify
  fix i assume *: i < d
  have (of-int (dist-p ((ts ! i) * β) ((ts-to-as ts ! i))))² ≤ ((u - v)$i)²
    using dist-p-smallest[OF assms(1,2) *] unfolding u-def .
  moreover have of-rat ... < (p / 2 ^ μ)²
  proof−
    have u - v ∈ carrier-vec (d + 1)
    proof−
      have u ∈ carrier-vec (d + 1)
       unfolding u-def using u-carrier assms(1) set-pmf-ts vec-class-mod-p-carrier
by blast
      moreover have v ∈ carrier-vec (d + 1)
        using assms(1,2) set-pmf-ts vec-class-mod-p-carrier by fastforce
      ultimately show ?thesis by auto
    qed
    hence ((ts-to-u ts - v) $ i)² < (rat-of-nat p / 2 ^ μ)²
      using assms(3) vec-le-sq-norm[of u - v d + 1 i] *
      unfolding close-vec-def u-def sq-norm-vec-def
      by fastforce
    thus ?thesis
      unfolding u-def
     by (metis (mono-tags, opaque-lifting) Ring-Hom.of-rat-hom.hom-div of-nat-numeral
of-rat-less of-rat-of-nat-eq of-rat-power)
  qed
  ultimately show (of-int (dist-p ((ts ! i) * β) ((ts-to-as ts ! i)))) < p / 2 ^ μ
   by (smt (verit, ccfv-SIG) Power.linordered-semidom-class.power-mono divide-nonneg-nonneg
of-nat-0-le-iff of-rat-less-eq of-rat-of-int-eq of-rat-power zero-less-power)
qed

lemma fixed-beta-bad-prob-arith-helper:
  defines T-lwr-bnd ≡ (rat-of-nat (p - 1) - 2 * (2 * (rat-of-nat p))/(2^μ))
  shows 1 - 5/(2^μ) ≤ real-of-rat T-lwr-bnd / (p - 1)
proof−
  have 5 ≤ p
  proof−
    have 2 powr (log 2 p) = p by (simp add: p prime-gt-0-nat)
    moreover have n - 1 ≤ log 2 p using n n-big by linarith
```

**ultimately have** $2\hat{}(n-1) \leq p$ **by** (*meson log2-of-power-less not-le p prime-gt-0-nat*)
**moreover have** $32 = (2::nat)\hat{}5$ **by** *simp*
**moreover have** $\ldots \leq 2\hat{}(n-1)$
**proof**−
**have** $5 \leq n-1$ **using** *n-big* **by** *auto*
**thus** *?thesis* **by** (*meson pow-mono-exp rel-simps*(25))
**qed**
**ultimately show** *?thesis* **by** *linarith*
**qed**
**hence** $4*p \leq 5*(p-1)$ **by** *fastforce*
**hence** $(4*p) / (p-1) \leq 5$
**by** (*metis* (*mono-tags*, *opaque-lifting*) *Multiseries-Expansion.intyness-simps*(6) *divide-le-eq not-le of-nat-0-le-iff of-nat-le-iff of-nat-simps*(5))
**hence** $((4*p) / (p-1)) * (1 / 2\hat{}\mu) \leq 5 / (2\hat{}\mu)$ **using** *divide-right-mono* **by** *fastforce*
**hence** $(4*p) / ((p-1) * 2\hat{}\mu) \leq 5 / (2\hat{}\mu)$ **by** *auto*
**hence** $1 - 5 / (2\hat{}\mu) \leq 1 - (4*p) / ((p-1) * 2\hat{}\mu)$ **by** *argo*
**also have** $\ldots = ((p-1) * (1 / (p-1))) - ((4*p) / (2\hat{}\mu)) * (1 / (p-1))$
**using** ‹$5 \leq p$› **by** *auto*
**also have** $\ldots = ((p-1) - ((4*p) / (2\hat{}\mu))) / (p-1)$ **by** *argo*
**also have** $\ldots =$ *real-of-rat T-lwr-bnd* $/ (p-1)$
**by** (*simp add*: *T-lwr-bnd-def Ring-Hom.of-rat-hom.hom-div Ring-Hom.of-rat-hom.hom-mult Ring-Hom.of-rat-hom.hom-power of-rat-diff*)
**finally show** *?thesis* **.**
**qed**

**lemma** *dist-p-helper*:
**fixes** *ts i*
**assumes** *ts*: $ts \in$ *set-pmf ts-pmf*
**assumes** *i*: $i < d$
**assumes** *dist*: *dist-p* (*int* (*ts*!*i*) $*\ \beta$) ((*ts-to-as ts*)!*i*) $< p/(2\hat{}\mu)$
**shows** *dist-p* ((*ts*!*i*) $*\ \beta$) ((*ts*!*i*) $*\ \alpha$) $\leq (2*p)/(2\hat{}\mu)$
**proof**−
**let** *?x* $= \alpha *$ (*ts* ! *i*) *mod p*
**let** *?*$a_i$ $=$ *msb-p ?x*
**let** *?*$ts_i$-$\beta$ $=$ (*ts* ! *i*) $*\ \beta$
**let** *?*$ts_i$-$\alpha$ $=$ (*ts* ! *i*) $*\ \alpha$
**have** *ts-to-as ts* ! *i* $=$ *?*$a_i$ **unfolding** *ts-to-as-def* **using** *i ts set-pmf-ts* **by** *force*
**hence** $*$: *dist-p ?*$ts_i$-$\beta$ *?*$a_i$ $<$ *real p* $/$ $2\ \hat{}\ \mu$ **using** *ts i dist* **by** *metis*

**have** *1*: $1 < p$ **using** *p prime-gt-1-nat* **by** *blast*
**have** *2*: *real* $(k+1) <$ *log 2* (*real p*) **using** *k-plus-1-lt* **by** *blast*
**have** $|int\ ?x - ?a_i| \leq$ *real p* $/$ $2\ \hat{}\ k$ **using** *msb-p-dist*[*of ?x*] **by** *argo*
**also have** $\ldots \leq$ *real p* $/$ $2\ \hat{}\ \mu$
**proof**−
**have** $(2::real)\ \hat{}\ k \geq 2\ \hat{}\ \mu$ **using** $\mu$-*le-k* **by** *auto*
**thus** *?thesis*
**by** (*metis frac-le less-eq-real-def of-nat-0-le-iff zero-less-numeral zero-less-power*)
**qed**

```
  finally have |int ?x − ?a_i| ≤ real p / 2 ^ μ .

  hence dist-p ?ts_i-β ?x − dist-p ?ts_i-β (int (msb-p ?x)) ≤ real p / 2 ^ μ
    using dist-p-diff[of ?ts_i-β ?x ?a_i] by linarith
  hence dist-p ?ts_i-β ?x ≤ real p / 2 ^ μ + dist-p ?ts_i-β ?a_i by linarith
  also have ... ≤ real p / 2 ^ μ + real p / 2 ^ μ using ∗ by argo
  also have ... = real (2 ∗ p) / 2 ^ μ by simp
  finally have dist-p ?ts_i-β ?x ≤ real (2 ∗ p) / 2 ^ μ .
  moreover have dist-p ?ts_i-β (ts ! i ∗ α) ≤ dist-p ?ts_i-β ?x (is ?lhs ≤ ?rhs)
  proof−
    let ?a = ?ts_i-β
    let ?b = ts ! i ∗ α
    let ?b' = ?x
    let ?S = {|?a − ?b + z ∗ int p| | z. True}
    have ?rhs ∈ ?S
    proof−
     obtain k where k: ?rhs = |?a − ?b' + k ∗ int p| using dist-p-well-defined[of
?a ?b'] by fast
      have [?b = ?b'] (mod int p)
        by (simp add: Groups.ab-semigroup-mult-class.mult.commute of-nat-mod)
     then obtain k' where ?b' = ?b + k' ∗ int p by (metis cong-iff-lin cross3-simps(11))
      hence ?rhs = |?a − (?b + k' ∗ int p) + k ∗ int p| using k by presburger
      also have ... = |?a − ?b − k' ∗ int p + k ∗ int p| by linarith
      also have ... = |?a − ?b + (− k' + k) ∗ int p|
        by (smt (verit, ccfv-SIG) Rings.ring-class.ring-distribs(2))
      also have ... ∈ ?S by blast
      finally show ?thesis .
    qed
    thus ?thesis unfolding dist-p-def using dist-p-le dist-p-def by auto
  qed
  ultimately show real-of-int (dist-p (map int ts ! i ∗ β) (int (ts ! i ∗ α))) ≤ real
(2 ∗ p) / 2 ^ μ
    using i ts set-pmf-ts by fastforce
qed

lemma prob-A-helper:
  fixes β :: int
  defines [simp]: t-pmf ≡ pmf-of-set {1..<p}
  defines [simp]: A-cond ≡ λt. (2∗p)/(2^μ) < dist-p (β ∗ t) (α ∗ t)
  defines [simp]: A-pmf ≡ t-pmf ⤜ (λt. return-pmf (A-cond t))
  defines [simp]: A ≡ pmf A-pmf True
  assumes β: β ∈ {0..<p::int}
  assumes False: ¬ (β = α mod p)
  shows (1 − A) ≤ (5 / (2^μ))
proof−
  let ?S = {1..<p}
  let ?T = {x ∈ ?S. A-cond x}
  let ?T-lwr-bnd = (rat-of-nat (p − 1) − 2 ∗ (2 ∗ (rat-of-nat p))/(2^μ))
  have card-S: card ?S = p − 1 by simp
```

```
  have fin-S: finite ?S and S-nempty: ?S ≠ {} using p-geq-2 by simp-all
  have A = card ?T / card ?S
    using pmf-of-finite-set-event[OF S-nempty fin-S, of A-cond] by simp
  moreover have real-of-rat ?T-lwr-bnd ≤ real-of-nat (card ?T)
  proof−
    let ?bound = (2 ∗ (rat-of-nat p))/(2^μ)
    let ?good = λt. rat-of-int (dist-p (t ∗ β) (t ∗ int α)) ≤ ?bound
    let ?bad = λt. ¬ ?good t
    let ?good-ts = {t ∈ ?S. ?good t}
    let ?bad-ts = {t ∈ ?S. ?bad t}
    have good-bad-card: card ?bad-ts = card ?S − card ?good-ts
    proof−
      have ?S = ?good-ts ∪ ?bad-ts by fastforce
      moreover have ?good-ts ∩ ?bad-ts = {} by blast
      ultimately have card ?good-ts + card ?bad-ts = card ?S
        by (smt (verit, ccfv-threshold) card-Un-disjoint fin-S finite-Un)
      thus ?thesis by linarith
    qed

    have 1: β ≠ int α using False β by force
    have 2: coprime (β − α) p
    proof−
      have 1: ¬ p dvd (β − α)
        by (metis assms(5,6) int-ops(9) int-p-prime mod-eq-dvd-iff mod-ident-iff
prime-gt-0-int)
      show ?thesis using prime-imp-power-coprime[OF int-p-prime 1, of 1] by
simp
    qed
    have 3: 0 < 2 ∗ rat-of-nat p / 2 ^ μ using p by (simp add: prime-gt-0-nat)
    have rat-of-nat (card ?good-ts) ≤ 2 ∗ ?bound
      by (rule dist-p-instances[OF 1 2 3])
   hence rat-of-nat (p − 1) − 2 ∗ ?bound ≤ rat-of-nat ((p − 1) − card ?good-ts)
by linarith
  also have ... = rat-of-nat (card ?bad-ts) using good-bad-card card-S by fastforce
    also have ... ≤ rat-of-nat (card ?T)
    proof−
      have ?bad-ts ⊆ ?T
      proof
        fix t assume t ∈ ?bad-ts
        hence ∗: t ∈ ?S ∧ ?bad t by blast
       hence 2 ∗ rat-of-nat p / 2^μ < rat-of-int (dist-p (int t ∗ β) (int t ∗ int α))
by linarith
        hence A-cond t
        proof(subst A-cond-def)
          assume ∗: 2 ∗ rat-of-nat p / 2 ^ μ < rat-of-int (dist-p (int t ∗ β) (int t ∗
int α))
         hence real-of-rat (2 ∗ rat-of-nat p / 2 ^ μ) < real-of-rat (rat-of-int (dist-p
(int t ∗ β) (int t ∗ int α)))
            using of-rat-less by blast
```

**moreover have** *real-of-rat* $(2 * \textit{rat-of-nat}\ p\ /\ 2\ \hat{}\ \mu) = \textit{real}\ (2 * p)\ /\ 2\ \hat{}\ \mu$
**by** (*metis* (*no-types*, *opaque-lifting*) *Ring-Hom.of-rat-hom.hom-div Ring-Hom.of-rat-hom.hom-power of-nat-numeral of-nat-simps*(5) *of-rat-of-nat-eq*)
**moreover have** *real-of-rat* (*rat-of-int* (*dist-p* (*int t* $*$ $\beta$) (*int t* $*$ *int* $\alpha$)))
$=$ *real-of-int* (*dist-p* ($\beta$ $*$ *int t*) (*int* ($\alpha$ $*$ *t*)))
**by** (*simp add*: *mult-ac*(2))
**ultimately show** *real* $(2 * p)\ /\ 2\ \hat{}\ \mu <$ *real-of-int* (*dist-p* ($\beta$ $*$ *int t*) (*int* ($\alpha$ $*$ *t*)))
**by** *algebra*
**qed**
**thus** $t \in ?T$ **unfolding** *A-cond-def* **using** $*$ **by** *blast*
**qed**
**moreover have** *finite* $?T$ **by** *simp*
**ultimately show** *?thesis* **by** (*meson card-mono of-nat-mono*)
**qed**
**finally show** *?thesis*
**by** (*metis* (*lifting*) *of-rat-less-eq of-rat-of-nat-eq times-divide-eq-right*)
**qed**
**ultimately have** *real-of-rat ?T-lwr-bnd* / (*card* $?S$) $\leq A$ **using** *divide-right-mono* **by** *blast*
**moreover have** *card* $?S = p - 1$ **by** *fastforce*
**ultimately have** *real-of-rat ?T-lwr-bnd* / $(p - 1) \leq A$ **by** *simp*
**moreover have** $1 - 5\ /\ (2\hat{}\mu) \leq$ *real-of-rat ?T-lwr-bnd* / $(p - 1)$
**by** (*rule fixed-beta-bad-prob-arith-helper*)
**ultimately show** *?thesis* **by** *argo*
**qed**

**lemma** *prob-A-helper'*:
**fixes** $\beta$ :: *int*
**defines** [*simp*]: *t-pmf* $\equiv$ *pmf-of-set* $\{1..<p\}$
**defines** [*simp*]: *A-cond* $\equiv \lambda t.\ (2*p)/(2\hat{}\mu) <$ *dist-p* ($\beta$ $*$ *t*) ($\alpha$ $*$ *t*)
**defines** [*simp*]: *A-pmf* $\equiv$ *t-pmf* $\gg\!=$ ($\lambda t.$ *return-pmf* (*A-cond t*))
**defines** [*simp*]: *A* $\equiv$ *pmf A-pmf True*
**assumes** $\beta$: $\beta \in \{0..<p::int\}$
**assumes** *False*: $\neg$ ($\beta = \alpha$ *mod p*)
**shows** $(1 - A)\hat{}d \leq (5\ /\ (2\hat{}\mu))\hat{}d$
**using** *prob-A-helper*[*OF* $\beta$ *False*] **apply** *simp*
**by** (*meson Power.linordered-semidom-class.power-mono diff-ge-0-iff-ge pmf-le-1*)

**lemma** *fixed-beta-bad-prob*:
**fixes** $\beta$
**assumes** $\beta \in \{0..<p::int\}$
**defines** *M* $\equiv$ *ts-pmf* $\gg\!=$ ($\lambda ts.$ *return-pmf* (*ts* $\in \{ts.$ *bad-beta ts* $\beta\}$))
**shows** $\beta = \alpha$ *mod p* $\Longrightarrow$ *pmf M True* $= 0$
$\neg$ ($\beta = \alpha$ *mod p*) $\Longrightarrow$ *pmf M True* $\leq (5/(2\hat{}\mu))\hat{}d$
**proof**−
**assume** *True*: $\beta = \alpha$ *mod p*

```
  have ∀ ts ∈ set-pmf ts-pmf. ¬ bad-beta ts β
  proof
    fix ts assume ts: ts ∈ set-pmf ts-pmf
    hence length-ts: length ts = d by (simp add: set-pmf-ts)

    show ¬ bad-beta ts β
      unfolding bad-beta-def good-beta-def
    proof clarify
      fix v assume v: v ∈ vec-class-mod-p ts β close-vec ts v
     hence dim-v: dim-vec v = d + 1 using vec-class-mod-p-carrier[OF length-ts,
of β] by auto
      have (∃ β::int. [α = β] (mod p) ∧ real-of-rat (v $ d) = β / p)
      proof−
        from v(1) obtain β where β: [α = β] (mod int p) ∧ v ∈ vec-class ts β
          unfolding vec-class-mod-p-def using True
            by (smt (verit, ccfv-threshold) UnionE mem-Collect-eq mod-mod-trivial
of-nat-mod
              unique-euclidean-semiring-class.cong-def)
       then obtain cs where cs: v = (of-int β) ·v (t-vec ts) + lincomb-list (of-int
∘ cs) p-vecs
          unfolding vec-class-def by blast
          hence v $ d = ((of-int β) ·v (t-vec ts))$d + (lincomb-list (of-int ∘ cs)
p-vecs)$d
            (is - = ?a + ?b)
          using dim-v by simp
          hence real-of-rat (v$d) = real-of-rat ?a + real-of-rat ?b by (simp add:
of-rat-add)
        moreover have real-of-rat ?a = β / p
        proof−
          have (t-vec ts)$d = 1 / (rat-of-nat p)
            unfolding t-vec-def using length-ts t-vec-def t-vec-last by presburger
          hence real-of-rat ((t-vec ts)$d) = 1 / p by (simp add: of-rat-divide)
          moreover have ?a = (of-int β) ∗ ((t-vec ts)$d)
            using length-ts t-vec-dim by auto
          ultimately show ?thesis by (simp add: of-rat-mult)
        qed
         moreover have real-of-rat ?b = 0 using lincomb-of-p-vecs-last[of cs] by
blast
         ultimately have [α = β] (mod p) ∧ real-of-rat (v $ d) = β / p using β
by linarith
        thus ?thesis by blast
      qed
      moreover have dim-vec v = d + 1
        using vec-class-mod-p-carrier[OF length-ts, of β] v by force
      ultimately show good-vec v unfolding good-vec-def by blast
    qed
  qed
  thus pmf M True = 0
    apply (simp add: M-def pmf-def)
```

**by** (*smt* (*verit*, *ccfv-SIG*) *Probability-Mass-Function.set-pmf.rep-eq bind-eq-return-pmf bind-return-pmf mem-Collect-eq return-pmf-inj*)
**next**
**assume** *False*: ¬ (*β* = *α mod p*)
**let** *?a* = *λts i*. (*ts-to-as ts*)!*i*
**let** *?u* = *λts*. *ts-to-u ts*
**let** *?E1* = *λts*. ∀ *i* < *d*. *dist-p* (*int* (*ts*!*i*) ∗ *β*) (*?a ts i*) < *p*/(2^*μ*)
**let** *?E2* = *λts*. ∀ *i* < *d*. *dist-p* ((*ts*!*i*) ∗ *β*) ((*ts*!*i*) ∗ *α*) ≤ (2∗*p*)/(2^*μ*)
**let** *?E1-pmf* = *ts-pmf* ⤜ (*λts*. *return-pmf* (*?E1 ts*))
**let** *?E2-pmf* = *ts-pmf* ⤜ (*λts*. *return-pmf* (*?E2 ts*))
**let** *?t-pmf* = *pmf-of-set* {*1*..<*p*}
**let** *?A-cond* = *λt*. (2∗*p*)/(2^*μ*) < *dist-p* (*β* ∗ *t*) (*α* ∗ *t*)
**let** *?A-pmf* = *?t-pmf* ⤜ (*λt*. *return-pmf* (*?A-cond t*))
**let** *?A* = *pmf ?A-pmf True*

**have** ∀ *ts* ∈ *set-pmf ts-pmf*. *bad-beta ts β* ⟶ *?E1 ts*
**using** *fixed-beta-close-vec-dist-p* **unfolding** *bad-beta-def good-beta-def* **by** *blast*

**moreover have** ∀ *ts* ∈ *set-pmf ts-pmf*. *?E1 ts* ⟶ *?E2 ts* **using** *dist-p-helper* **by** *blast*
**ultimately have** ∀ *ts* ∈ *set-pmf ts-pmf*. *ts* ∈ {*ts*. *bad-beta ts β*} ⟶ *?E2 ts* **by** *blast*
**hence** *pmf M True* ≤ *pmf* (*?E2-pmf*) *True*
**unfolding** *M-def* **using** *pmf-subset*[*of ts-pmf λts*. *ts* ∈ {*ts*. *bad-beta ts β*} *?E2*] **by** *blast*
**also have** ... ≤ (*1* − *?A*)^*d*
**proof**−
**let** *?E* = *λx*. *dist-p* (*int x* ∗ *β*) (*int x* ∗ *α*) ≤ (2∗*p*)/(2^*μ*)
**have** ∗: *pmf* (*pmf-of-set* {*1*..<*p*} ⤜ (*λx*. *return-pmf* (*?E x*))) *True* ≤ (*1* − *?A*)
**proof**−
**let** *?A′-pmf* = *?t-pmf* ⤜ (*λt*. *return-pmf* (¬ *?A-cond t*))
**let** *?p* = *pmf-of-set* {*1*..<*p*} ⤜ (*λx*. *return-pmf* (*?E x*))
**have** *1*: ∀ *x* ∈ *set-pmf ?t-pmf*. *?E x* ⟶ ¬ *?A-cond x* **by** (*simp add*: *mult-ac*(*2*))
**have** *pmf ?p True* ≤ *pmf ?A′-pmf True* **using** *pmf-subset*[*OF 1*] **by** *blast*
**also have** ... = *pmf ?A-pmf False* **using** *pmf-true-false*[*of ?t-pmf*] **by** *presburger*
**also have** ... = *1* − (*pmf ?A-pmf True*) **using** *pmf-True-conv-False* **by** *auto*
**finally show** *?thesis*
**by** (*smt* (*verit*, *del-insts*) *α empty-iff finite-atLeastLessThan pmf-of-set-def*)
**qed**
**show** *?thesis*
**using** *replicate-pmf-same-event-leq*[*OF* ∗, *of d*, *folded ts-pmf-def*]
**by** (*smt* (*verit*, *best*) *bind-pmf-cong mem-Collect-eq nth-map of-nat-simps*(*5*) *set-pmf-ts*)
**qed**
**also have** ... ≤ (*5* / (2^*μ*))^*d* **by** (*rule prob-A-helper′*[*OF assms*(*1*) *False*])
**finally show** *pmf M True* ≤ (*5*/(2^*μ*))^*d* **.**

**qed**

**lemma** *bad-beta-union-bound-pmf*:
*pmf* (*ts-pmf* ⤜ (λ*ts*. *return-pmf* (*ts* ∈ *bad-beta-union*))) *True* ≤ (*p* − *1*) ∗ (*5*/(*2*^μ))^*d*
**proof**−
**let** *?b* = (*5*/(*2*^μ))^*d*
**let** *?M′* = λβ. *ts-pmf* ⤜ (λ*ts*. *return-pmf* (*ts* ∈ {*ts*. *bad-beta ts* β}))
**have** (∑ β ∈ {*0*..<*p*::*int*}. *pmf* (*?M′* β) *True*) ≤ (*p* − *1*) ∗ *?b*
**proof**−
**let** *?x* = λβ. *pmf* (*?M′* β) *True*
**let** *?A* = {*0*..<*p*::*int*}
**have** α ∈ *?A* **using** α **by** *force*
**hence** (∑ β ∈ *?A*. *?x* β) = (∑ β ∈ *?A* − {α}. *?x* β) + *?x* α
**using** *sum.remove*[*of ?A* α *?x*] **by** *fastforce*
**also have** ... ≤ (∑ β ∈ *?A* − {α}. *?b*) + *?x* α
**proof**−
**have** ⋀β. β ∈ *?A* − {α} ⟹ *?x* β ≥ *0* **by** *auto*
**moreover have** ⋀β. β ∈ *?A* − {α} ⟹ *?x* β ≤ *?b* **using** *fixed-beta-bad-prob*(*2*)
**by** (*metis DiffE* ‹*int* α ∈ {*0*..<*int p*}› *atLeastLessThan-iff insertI1 linorder-not-le nat-mod-eq′ of-nat-le-iff*)
**ultimately have** (∑ β ∈ *?A* − {α}. *?x* β) ≤ (∑ β ∈ *?A* − {α}. *?b*) **by** (*meson sum-mono*)
**thus** *?thesis* **by** *argo*
**qed**
**also have** ... = (*p* − *1*) ∗ *?b* **using** *fixed-beta-bad-prob*(*1*)[*of* α] α **by** *simp*
**finally show** *?thesis* **.**
**qed**
**thus** *?thesis* **using** *bad-beta-union-bound* **by** *linarith*
**qed**

**lemma** *sampled-lattice-unlikely-bad*:
**shows** *pmf sampled-lattice-good False* ≤ (*p* − *1*) ∗ ((*5*/(*2*^μ))^*d*)
**using** *reduction-1-pmf reduction-2-pmf bad-beta-union-bound-pmf* **by** *force*

### 11.2.4 Main uniqueness lemma statement

**lemma** *sampled-lattice-likely-good*: *pmf sampled-lattice-good True* ≥ *1*/*2*
**proof**−
**have** (*p* − *1*) ∗ ((*5*/(*2*^μ))^*d*) ≤ *1*/*2*
**proof**−
**have** ∗: (*5*/(*2*^μ))^*d* ≤ ((*2 powr* (*2.5* ∗ (*of-nat d*)) / (*2*^(μ∗*d*)))::*real*)
**proof**−
**have** (*5*::*real*)^*d* ≤ ((*2*::*real*) *powr 2.5*)^*d*
**proof**−
**have** (*2*::*real*) *powr 2* = *4* **by** *fastforce*
**moreover have** (*2*::*real*) *powr 0.5* ≥ *1.25*
**proof**−
**have** (*1.25*::*real*) ∗ *1.25* = *1.5625* **by** *fastforce*

**moreover have** (*1.25*::*real*) * *1.25* = *1.25 powr 2* **by** (*simp add*: *power2-eq-square*)
**ultimately have** (*1.25*::*real*) *powr 2* ≤ *2* **by** *simp*
**hence** (*1.25*::*real*) *powr 2* ≤ (*sqrt 2*) *powr 2* **by** *fastforce*
**moreover have** *sqrt 2* = (*2*::*real*) *powr 0.5* **using** *powr-half-sqrt* **by** *simp*
**ultimately show** *?thesis* **by** *fastforce*
**qed**
**moreover have** (*2*::*real*) *powr 2.5* = (*2 powr 2*) * (*2 powr 0.5*)
**proof**−
**have** (*2*::*real*) *powr 2.5* = (*2*::*real*) *powr* (*2* + *0.5*) **by** *fastforce*
**thus** *?thesis* **by** (*metis powr-add*)
**qed**
**ultimately have** (*5*::*real*) ≤ *2 powr 2.5* **by** *auto*
**thus** *?thesis* **by** (*simp add*: *nonneg-power-le*)
**qed**
**also have** ... = (*2*::*real*) *powr* (*2.5* * *d*)
**by** (*simp add*: *Groups.ab-semigroup-mult-class.mult.commute powr-power*)
**finally have** (*5*::*real*)^*d* ≤ (*2*::*real*) *powr* (*2.5* * *d*) **.**
**hence** (*5*^*d*/*2*^(μ*d)) ≤ ((*2*::*real*) *powr* (*2.5* * *d*)) / *2*^(μ*d) **by** (*simp add*: *divide-right-mono*)
**moreover have** ((*5*::*real*)/(*2*^μ))^*d* = (*5*^*d*/*2*^(μ*d)) **by** (*simp add*: *power-divide power-mult*)
**ultimately show** *?thesis* **by** *presburger*
**qed**

**have** μ * *d* ≥ *n* + *6* * *ceiling* (*sqrt n*)
**proof**−
**have** μ * *d* = *6* * *ceiling* (*sqrt n*) + (⌈*1* / *2* * *sqrt* (*real n*)⌉ * *2* * ⌈*sqrt* (*real n*)⌉)
**by** (*simp add*: μ *d Ring-Hom.mult-hom.hom-add mult-ac*(*2*))
**moreover have** ⌈*1* / *2* * *sqrt* (*real n*)⌉ * *2* * ⌈*sqrt* (*real n*)⌉ ≥ *1* / *2* * *sqrt* (*real n*) * *2* * *sqrt* (*real n*)
**proof**−
**have** ⌈*1* / *2* * *sqrt* (*real n*)⌉ ≥ *1* / *2* * *sqrt* (*real n*) **by** *linarith*
**moreover have** ⌈*sqrt* (*real n*)⌉ ≥ *sqrt* (*real n*) **by** *linarith*
**ultimately show** *?thesis*
**by** (*smt* (*verit*, *best*) *Ring-Hom.mult-hom.hom-add divide-nonneg-nonneg mult-2-right mult-mono of-int-add of-int-mult of-nat-0-le-iff powr-ge-zero powr-half-sqrt*)
**qed**
**moreover have** *1* / *2* * *sqrt* (*real n*) * *2* * *sqrt* (*real n*) = *n* **by** *simp*
**ultimately show** *?thesis* **by** *linarith*
**qed**
**hence** *2 powr* (μ*d) ≥ *2 powr* (*n* + *6* * *ceiling* (*sqrt n*))
**by** (*smt* (*verit*) *of-int-le-iff of-int-of-nat-eq powr-mono*)
**hence** *2 powr* (*2.5**d) / (*2 powr* (μ*d)) ≤ *2 powr* (*2.5**d) / ((*2*::*real*) *powr* (*n* + *6* * *ceiling* (*sqrt n*)))
**by** (*metis frac-le less-eq-real-def powr-ge-zero powr-gt-zero*)
**moreover have** *2 powr* (*2.5**d) ≤ *2 powr* (*5* * *ceiling* (*sqrt n*))
**using** *d* **by** *fastforce*

**ultimately have** *2 powr (2.5∗d) / (2 powr (μ∗d))*
*≤ (2 powr (5 ∗ ceiling (sqrt n))) / ((2::real) powr (n + 6 ∗ ceiling (sqrt n)))*
**by** (*smt* (*verit*, *ccfv-SIG*) *frac-le powr-gt-zero*)
**also have** *... = ((2::real) powr (5 ∗ ceiling (sqrt n))) / ((2 powr n) ∗ (2 powr (6 ∗ ceiling (sqrt n))))*
**using** *powr-add* **by** *auto*
**also have** *... ≤ (1 / ((2::real) powr n)) ∗ (2 powr (5 ∗ ceiling (sqrt n))) / ((2 powr (6 ∗ ceiling (sqrt n))))*
**by** *argo*
**also have** *... = (1 / ((2::real) powr n)) ∗ (2 powr (5 ∗ ceiling (sqrt n))) / ((2 powr (ceiling (sqrt n))) ∗ (2 powr (5 ∗ ceiling (sqrt n))))*
**by** (*smt* (*verit*) *of-int-add powr-add*)
**also have** *... = (1 / ((2::real) powr n)) ∗ (1 / (2 powr (ceiling (sqrt n))))*
**by** *auto*
**finally have** ∗∗: *2 powr (2.5∗d) / (2 powr (μ∗d)) ≤ (1 / (2 powr n)) ∗ (1 / (2 powr (ceiling (sqrt n))))*
(**is** *?A ≤ -*) **.**
**moreover have** *((2::real) powr n) ∗ ... = (1 / (2 powr (ceiling (sqrt n))))* **by** *fastforce*
**ultimately have** *((2::real) powr n) ∗ ?A ≤ (1 / (2 powr (ceiling (sqrt n))))*
**by** (*metis mult-left-mono powr-ge-zero*)
**moreover have** *(p − 1) ∗ ?A ≤ ((2::real) powr n) ∗ ?A*
**proof**−
**have** *p − 1 ≤ ((2::real) powr n)*
**proof**−
**have** *p − 1 ≤ p* **by** *simp*
**also have** *... = 2 powr (log 2 p)* **by** (*simp add*: *p prime-gt-0-nat*)
**also have** *... ≤ (2::real) powr n*
**by** (*smt* (*verit*, *ccfv-SIG*) *n le-diff-iff le-diff-iff' le-numeral-extra*(*4*) *n-geq-1 nat-ceiling-le-eq nat-int powr-le-cancel-iff*)
**finally show** *?thesis* **by** *blast*
**qed**
**thus** *?thesis* **by** (*meson divide-nonneg-nonneg less-eq-real-def mult-mono powr-ge-zero*)
**qed**
**ultimately have** *(p − 1) ∗ ?A ≤ (1 / (2 powr (ceiling (sqrt n))))* **by** *linarith*
**moreover have** *(5/(2^μ))^d ≤ ?A* **by** (*smt* (*verit*, *best*) ∗ *powr-realpow*)
**moreover have** *(1 / (2 powr (ceiling (sqrt n)))) ≤ 1/2*
**proof**−
**have** *sqrt n ≥ 1* **using** *n-geq-1* **by** *simp*
**hence** *ceiling (sqrt n) ≥ 1* **by** *fastforce*
**hence** *2 powr (ceiling (sqrt n)) ≥ 2*
**by** (*metis of-int-1-le-iff powr-mono powr-one rel-simps*(*25*) *rel-simps*(*27*))
**thus** *?thesis* **by** *force*
**qed**
**ultimately show** *?thesis* **by** (*smt* (*verit*, *best*) *mult-left-mono of-nat-0-le-iff*)
**qed**
**hence** *1 − (p − 1) ∗ ((5/(2^μ))^d) ≥ 1/2* **by** *simp*

**moreover have** *pmf sampled-lattice-good True = 1 − pmf sampled-lattice-good False*
**using** *pmf-True-conv-False* **by** *blast*
**ultimately show** *?thesis* **using** *sampled-lattice-unlikely-bad* **by** *linarith*
**qed**

## 11.3 Main theorem

### 11.3.1 Oracle definition

**definition** $\mathcal{O}$ :: *nat* ⇒ *nat* **where**
$\mathcal{O}\ t$ = *msb-p* ((α ∗ *t*) *mod p*)

### 11.3.2 Apply Babai lemmas to adversary

We use Babai lemmas to show that the adversary wins when the *ts* generate a good lattice.

**lemma** *gen-basis-assms*:
**fixes** *ts* :: *nat list*
**assumes** *length ts* = *d*
**shows** *LLL.LLL-with-assms* (*d*+*1*) (*d*+*1*) (*int-gen-basis ts*) (*4*/*3*)
**proof**−
**have** *l*: *length* (*int-gen-basis ts*) = *d* + *1* **unfolding** *int-gen-basis-def* **by** *force*
**moreover have** *LLL.lin-indep* (*d* + *1*) (*int-gen-basis ts*)
**proof**−
**let** *?b* = *int-gen-basis ts*
**let** *?M* = (*mat-of-rows* (*d* + *1*) (*LLL.RAT ?b*))
**let** *?Mt* = *transpose-mat ?M*
**have** *carrier-M*: *?M* ∈ *carrier-mat* (*d* + *1*) (*d* + *1*) **using** *l* **by** *force*
**have** *diag*: *?Mt*$$(*i*,*i*) ≠ *0* **if** *i*: *i*∈{*0*..<*dim-row ?Mt*} **for** *i*
**proof**(*cases i*=*d*)
**case** *t*:*True*
**have** *carrier-vec*: (*LLL.RAT ?b*)!*i* ∈ *carrier-vec* (*d* + *1*)
**using** *int-gen-basis-carrier*[*of ts*] *assms l i* **by** *fastforce*
**have** *?Mt*$$(*i*,*i*) = *?M*$$(*i*,*i*)
**using** *index-transpose-mat carrier-M i* **by** *auto*
**also have** *?M*$$(*i*,*i*) = *row ?M i* $*i* **using** *carrier-M i* **by** *auto*
**also have** *row ?M i* $*i* = (*LLL.RAT ?b*)!*i*$*i*
**using** *mat-of-rows-row l i carrier-M carrier-vec* **by** *auto*
**finally have** *1*: *?Mt*$$(*i*,*i*) = (*LLL.RAT ?b*)!*d*$*d* **using** *t* **by** *simp*
**moreover have** (*LLL.RAT ?b*)!*d*$*d* = (*of-int-hom.vec-hom* (*?b*!*d*))$*d* **using** *nth-map l* **by** *auto*
**moreover have** (*of-int-hom.vec-hom* (*?b*!*d*))$*d* = *of-int-hom.vec-hom*(*vec-of-list* (*map* ((∗) (*int p*)) (*map int ts*) @ [*1*]))$*d*
**using** *append-Cons-nth-middle*[*of d* (*map* (λ*i*. (*p*^*2* ·$_v$ (*unit-vec* (*d*+*1*) *i*))) [*0*..<*d*]) *vec-of-list* (*map* ((∗) (*int p*)) (*map int ts*) @ [*1*]) []]
**unfolding** *int-gen-basis-def* **by** *auto*
**moreover have** *of-int-hom.vec-hom* (*vec-of-list* (*map* ((∗) (*int p*)) (*map int ts*) @ [*1*]))$*d* = *1*

**using** *append-Cons-nth-middle*[*of d map* ((∗) (*int p*)) (*map int ts*) *1* []] *assms*
**by** *fastforce*
**ultimately have** *?Mt*$$(*i*,*i*) = *1*
**by** *metis*
**then show** *?thesis*
**by** *simp*
**next**
**case** *f*:*False*
**have** *carrier-vec*: (*LLL.RAT ?b*)!*i* ∈ *carrier-vec* (*d* + *1*)
**using** *int-gen-basis-carrier*[*of ts*] *assms l i* **by** *fastforce*
**have** *?Mt*$$(*i*,*i*) = *?M*$$(*i*,*i*)
**using** *index-transpose-mat carrier-M i* **by** *auto*
**also have** *?M*$$(*i*,*i*) = *row ?M i* $*i* **using** *carrier-M i* **by** *auto*
**also have** *row ?M i* $*i* = (*LLL.RAT ?b*)!*i*$*i*
**using** *mat-of-rows-row l i carrier-M carrier-vec* **by** *auto*
**finally have** *1*: *?Mt*$$(*i*,*i*) = (*LLL.RAT ?b*)!*i*$*i* **by** *simp*
**moreover have** (*LLL.RAT ?b*)!*i*$*i* = (*of-int-hom.vec-hom* (*?b*!*i*))$*i* **using**
*nth-map l i* **by** *auto*
**moreover have** (*of-int-hom.vec-hom* (*?b*!*i*))$*i* = *of-int-hom.vec-hom*(*p*^*2* ·$_v$
(*unit-vec* (*d*+*1*) *i*))$*i*
**using** *append-Cons-nth-left*[*of i* (*map* (λ*i*. (*p*^*2* ·$_v$ (*unit-vec* (*d*+*1*) *i*)))
[*0*..<*d*]) *vec-of-list* (*map* ((∗) (*int p*)) (*map int ts*) @ [*1*]) []]
*i f l*
**unfolding** *int-gen-basis-def* **by** *auto*
**moreover have** *of-int-hom.vec-hom*(*p*^*2* ·$_v$ (*unit-vec* (*d*+*1*) *i*))$*i* = *rat-of-nat*
(*p*^*2* ∗ (*unit-vec* (*d*+*1*) *i*)$*i*) **using** *i*
**by** *simp*
**moreover have** (*unit-vec* (*d*+*1*) *i*)$*i* = *1* **using** *i* **by** *simp*
**ultimately have** *?Mt*$$(*i*,*i*) = *rat-of-nat* (*p*^*2* ∗ *1*) **by** *metis*
**then show** *?thesis* **using** *p*
**by** *auto*
**qed**
**have** *?Mt*$$(*i*,*j*) = *0* **if** *ij*: *i*∈{*0*..<*dim-row ?Mt*} ∧ *j*∈{*0*..<*i*} **for** *i j*
**proof**−
**have** *carrier-vec*: (*LLL.RAT ?b*)!*j* ∈ *carrier-vec* (*d* + *1*)
**using** *int-gen-basis-carrier*[*of ts*] *assms l ij* **by** *fastforce*
**have** *?Mt*$$(*i*,*j*) = *?M*$$(*j*,*i*)
**using** *index-transpose-mat carrier-M ij* **by** *auto*
**also have** *?M*$$(*j*,*i*) = *row ?M j* $*i* **using** *carrier-M ij* **by** *auto*
**also have** *row ?M j* $*i* = (*LLL.RAT ?b*)!*j*$*i*
**using** *mat-of-rows-row l ij carrier-M carrier-vec* **by** *auto*
**finally have** *1*: *?Mt*$$(*i*,*j*) = (*LLL.RAT ?b*)!*j*$*i* **by** *simp*
**have** *jd*: *j*<*d* **using** *ij carrier-M* **by** *simp*
**then have** *?b*!*j* = (*map* (λ*i*. (*p*^*2* ·$_v$ (*unit-vec* (*d*+*1*) *i*))) [*0*..<*d*])!*j*
**using** *append-Cons-nth-left*[*of j* (*map* (λ*i*. (*p*^*2* ·$_v$ (*unit-vec* (*d*+*1*) *i*)))
[*0*..<*d*]) *vec-of-list* (*map* ((∗) (*int p*)) (*map int ts*) @ [*1*]) []]
**unfolding** *int-gen-basis-def*
**by** *fastforce*
**then have** *?b*!*j* = *p*^*2* ·$_v$ (*unit-vec* (*d*+*1*) *j*) **using** *jd* **by** *force*

```
      then have (LLL.RAT ?b)!j = of-int-hom.vec-hom (p^2 ·v (unit-vec (d+1)
j))
        using nth-map[of j ?b of-int-hom.vec-hom] jd l
        by auto
      then have (LLL.RAT ?b)!j$i = rat-of-nat ((p^2 ·v (unit-vec (d+1) j))$i)
        using  ij by force
       also have rat-of-nat ((p^2 ·v (unit-vec (d+1) j))$i) = rat-of-nat (p^2 *
(unit-vec (d+1) j)$i)
        using ij by auto
      also have (unit-vec (d+1) j)$i = 0 unfolding unit-vec-def using ij
        by simp
      finally have (LLL.RAT ?b)!j$i = 0 by linarith
      then show ?thesis using 1
        by presburger
    qed
    then have upper-triangular ?Mt
      by auto
     moreover have carrier-Mt: ?Mt ∈ carrier-mat (d + 1) (d + 1) using car-
rier-M by simp
    moreover have 0 ∉ set (diag-mat ?Mt) using diag unfolding diag-mat-def
      by fastforce
    ultimately have det ?Mt ≠ 0 using upper-triangular-imp-det-eq-0-iff[of ?Mt]
by blast
    then have det-M: det ?M ≠ 0
      by (metis Determinant.det-transpose Matrix.transpose-transpose carrier-Mt)
   then have lin-indpt (set (Matrix.rows ?M)) using det-not-0-imp-lin-indpt-rows[of
?M] carrier-M
      by fast
    then have LI: lin-indpt (set (LLL.RAT ?b))
       using rows-mat-of-rows[of LLL.RAT ?b d + 1] int-gen-basis-carrier[of ts]
assms by fastforce
     have (LLL.RAT (int-gen-basis ts))!i ≠ (LLL.RAT (int-gen-basis ts))!j if ij:
i≠j ∧ i<(d + 1) ∧ j < (d + 1) for i j
    proof−
      have (LLL.RAT (int-gen-basis ts))!j ∈ carrier-vec (d + 1)
        using int-gen-basis-carrier[of ts] assms l ij by fastforce
      moreover have (LLL.RAT (int-gen-basis ts))!i ∈ carrier-vec (d + 1)
        using int-gen-basis-carrier[of ts] assms l ij by fastforce
      moreover have row ?M i ≠ row ?M j
        using Determinant.det-identical-rows[of ?M d + 1 i j] carrier-M using ij
det-M by fast
     ultimately show (LLL.RAT (int-gen-basis ts))!i ≠ (LLL.RAT (int-gen-basis
ts))!j
        using mat-of-rows-row[of i (LLL.RAT (int-gen-basis ts)) d + 1]
              mat-of-rows-row[of j (LLL.RAT (int-gen-basis ts)) d + 1]
              ij l
        by force
    qed
    then have distinct (LLL.RAT (int-gen-basis ts))
```

**using** *distinct-conv-nth*[*of LLL.RAT* (*int-gen-basis ts*)] *l* **by** *simp*
**moreover have** *set-in*: *set* (*LLL.RAT* (*int-gen-basis ts*)) ⊆ *carrier-vec* (*d* + *1*)
**using** *int-gen-basis-carrier*[*of ts*] *assms* **by** *auto*
**ultimately show** *?thesis* **unfolding** *Gram-Schmidt-2.cof-vec-space.lin-indpt-list-def* **using** *LI*
**by** *blast*
**qed**
**ultimately show** *?thesis* **unfolding** *LLL-with-assms-def* **by** *simp*
**qed**

**definition** *u-vec-is-msbs* :: (*nat*×*nat*) *list* ⇒ *bool* **where**
*u-vec-is-msbs pairs* = ((*of-nat p*) ·$_v$ *ts-to-u* (*ts-from-pairs pairs*) = *of-int-hom.vec-hom* (*scaled-uvec-from-pairs pairs*))

**lemma** *updated-full-Babai-correct-int-target*:
**assumes** *full-babai-with-assms* (*map-vec rat-of-int target*) (*d* + *1*) *fs-init* (*4*/*3*)
**shows** *real-of-int* (*sq-norm* ( (*full-babai fs-init* (*map-vec rat-of-int target*) (*4*/*3*)) − *target*))
≤ ((*4*/*3*)^(*d* + *1*) ∗ (*d* + *1*)) ∗ *11*/*10* ∗ *babai.closest-distance-sq fs-init* (*map-vec rat-of-int target*) ∧
(*full-babai fs-init* (*map-vec rat-of-int target*) (*4*/*3*)) ∈ *vec-module.lattice-of* (*d* + *1*) *fs-init*
**proof**−
**have** *1*: *0*≤*real-of-int* (*sq-norm* ( (*full-babai fs-init* (*map-vec rat-of-int target*) (*4*/*3*)) − *target*)) **by** *auto*
**have** *real-of-int* (*sq-norm* ( (*full-babai fs-init* (*map-vec rat-of-int target*) (*4*/*3*)) − *target*))
≤ (*real-of-rat* (*4*/*3*)^(*d* + *1*) ∗ (*d* + *1*)) ∗ *babai.closest-distance-sq fs-init* (*map-vec rat-of-int target*) ∧
(*full-babai fs-init* (*map-vec rat-of-int target*) (*4*/*3*)) ∈ *vec-module.lattice-of* (*d* + *1*) *fs-init*
**using** *full-babai-correct-int-target*[*of target d* + *1 fs-init 4*/*3*] *assms* **by** *blast*
**moreover then have** *2*: *0* ≤ (*real-of-rat* (*4*/*3*)^(*d* + *1*) ∗ (*d* + *1*)) ∗ *babai.closest-distance-sq fs-init* (*map-vec rat-of-int target*)
**using** *1* **by** *linarith*
**moreover have** (*real-of-rat* (*4*/*3*)^(*d* + *1*) ∗ (*d* + *1*)) ∗ *babai.closest-distance-sq fs-init* (*map-vec rat-of-int target*) =
((*4*/*3*)^(*d* + *1*) ∗ (*d* + *1*)) ∗ *babai.closest-distance-sq fs-init* (*map-vec rat-of-int target*)
**by** (*simp add*: *Ring-Hom.of-rat-hom.hom-div*)
**moreover then have** ((*4*/*3*)^(*d* + *1*) ∗ (*d* + *1*)) ∗ *babai.closest-distance-sq fs-init* (*map-vec rat-of-int target*) ≤
((*4*/*3*)^(*d* + *1*) ∗ (*d* + *1*)) ∗ *11*/*10* ∗ *babai.closest-distance-sq fs-init* (*map-vec rat-of-int target*)
**using** *mult-right-mono*[*of 1 11*/*10* ((*4*/*3*)^(*d* + *1*) ∗ (*d* + *1*)) ∗ *babai.closest-distance-sq fs-init* (*map-vec rat-of-int target*)] *2* **by** *argo*
**ultimately show** *?thesis* **by** *argo*
**qed**

**lemma** *ad-output-vec-close*:
**fixes** *pairs* :: (*nat*×*nat*) *list*
**assumes** *length pairs* = *d*
**assumes** *u-vec-is-msbs pairs*
**shows** *real-of-int*(*sq-norm*
((*A-vec pairs*)
− (*scaled-uvec-from-pairs pairs*))) ≤
(*4* / *3*) ^ (*d* + *1*) ∗ *real* (*d* + *1*) ∗ *11* / *10* ∗ *p*^*2* ∗ (*of-nat* ((*d*+*1*) ∗ *p*^*2*))/*2*^(*2*∗*k*)
∧ (*A-vec pairs*) ∈ *int-gen-lattice* (*ts-from-pairs pairs*)
**proof**−
**define** *ts* **where** *ts* = *ts-from-pairs pairs*
**define** *u-vec* **where** *u-vec* = *scaled-uvec-from-pairs pairs*
**define** *rat-u-vec* **where** *rat-u-vec* = *map-vec rat-of-int u-vec*
**define** *basis* **where** *basis* = *int-gen-basis ts*
**have** *t-length*: *length ts* = *d* **unfolding** *ts-def ts-from-pairs-def* **using** *assms* **by** *fastforce*
**have** *basis-length*: *length basis* = *d* + *1* **unfolding** *basis-def int-gen-basis-def* **by** *auto*
**have** *p-le*: *0*≤*real*(*p*^*2*) **by** *blast*
**have** *u-dim*: *dim-vec u-vec* = *d* + *1* **unfolding** *scaled-uvec-from-pairs-def u-vec-def* **using** *assms* **by** *force*
**then have** *rat-u-dim*: *dim-vec rat-u-vec* = *d* + *1* **unfolding** *rat-u-vec-def* **by** *fastforce*
**have** *length ts* = *d* **unfolding** *ts-from-pairs-def ts-def* **using** *assms* **by** *fastforce*
**then have** *LLL-assms*: *LLL.LLL-with-assms* (*d*+*1*) (*d*+*1*) *basis* (*4*/*3*)
**using** *gen-basis-assms*[*of ts*] **unfolding** *basis-def* **by** *argo*
**then have** *f*: *full-babai-with-assms rat-u-vec* (*d* + *1*) *basis* (*4*/*3*)
**using** *u-dim* **unfolding** *rat-u-vec-def full-babai-with-assms-def* **by** *force*
**then have** *1*: *real-of-int* (*sq-norm* ((*full-babai basis rat-u-vec* (*4*/*3*)) − *u-vec*))
≤ ((*4*::*real*)/*3*)^(*d*+*1*) ∗ (*d* + *1*) ∗ *11*/*10* ∗ *babai.closest-distance-sq basis rat-u-vec* ∧
(*full-babai basis rat-u-vec* (*4*/*3*)) ∈ *vec-module.lattice-of* (*d* + *1*) *basis*
**using** *updated-full-Babai-correct-int-target*[*of u-vec basis*] *u-dim*
**unfolding** *rat-u-vec-def* **by** *blast*
**have** *babai.closest-distance-sq basis rat-u-vec*
= *Inf* {*real-of-rat* (*sq-norm x* )| *x*. *x* ∈ {*of-int-hom.vec-hom x* − *rat-u-vec*| *x*. *x*∈*LLL.LLL.L* (*d* + *1*) *basis*}}
**using** *babai.closest-distance-sq-def*[*of basis rat-u-vec*] *babai-def*[*of basis rat-u-vec*]
*basis-length rat-u-dim* **by** *presburger*
**moreover have** *0*≤(*real*(*4*)/*3*)^(*d*+*1*) ∗ (*d* + *1*) ∗ *11*/*10* **by** *force*
**moreover have** *0*≤*babai.closest-distance-sq basis rat-u-vec*
**proof**−
**have** *b*: *babai-with-assms* (*LLL-Impl.reduce-basis* (*4*/*3*) *basis*) *rat-u-vec* (*4*/*3*)
**using** *full-babai-with-assms.LLL-output-babai-assms*[*of rat-u-vec d* + *1 basis*] *f* **by** *simp*
**then have** *b1*: *babai* (*LLL-Impl.reduce-basis* (*4*/*3*) *basis*) *rat-u-vec* **unfolding** *babai-with-assms-def* **by** *auto*

**have** *b2*: *babai basis rat-u-vec* **using** *basis-length rat-u-dim* **unfolding** *babai-def*
**by** *simp*
**have** *0* ≤ *babai.closest-distance-sq* (*LLL-Impl.reduce-basis* (*4/3*) *basis*) *rat-u-vec*
**using** *b babai-with-assms-epsilon.closest-distance-sq-pos*[*of* (*LLL-Impl.reduce-basis*
(*4/3*) *basis*) *rat-u-vec 4/3 2*]
*babai-with-assms.babai-with-assms-epsilon-connect*[*of* (*LLL-Impl.reduce-basis*
(*4/3*) *basis*) *rat-u-vec 4/3*] **by** *simp*
**moreover have** *LLL.L* (*d* + *1*) *basis* = *LLL.L* (*d* + *1*) (*LLL-Impl.reduce-basis*
(*4/3*) *basis*)
**using** *LLL-with-assms.reduce-basis*(*1*)[*of d* + *1 d* + *1 basis 4/3* (*LLL-Impl.reduce-basis*
(*4/3*) *basis*)] *LLL-assms*
**unfolding** *LLL.L-def* **by** *argo*
**moreover have** *length* (*LLL-Impl.reduce-basis* (*4/3*) *basis*) = *d* + *1*
**using** *LLL-with-assms.reduce-basis*(*4*)[*of d+1 d+1 basis 4/3*] *LLL-assms* **by**
*fast*
**ultimately show** *?thesis*
**using** *babai.closest-distance-sq-def basis-length b1 b2* **by** *algebra*
**qed**
**moreover have** *babai.closest-distance-sq basis rat-u-vec*
≤ *p^2* ∗ (*of-nat* ((*d+1*) ∗ *p^2*))/*2^*(*2*∗*k*)
**proof**−
**let** *?rat-basis* = *gen-basis ts*
**let** *?unscaled-u* = *ts-to-u ts*
**obtain** *w* **where** *w*: *w* ∈ *gen-lattice ts* ∧ *sq-norm* (*?unscaled-u* − *w*) ≤ (*of-nat*
((*d+1*) ∗ *p^2*))/*2^*(*2*∗*k*)
**using** *close-vector-exists*[*of ts*] *t-length*
**by** *blast*
**also have** ∗∗: *sq-norm* (*?unscaled-u* − *w*) = *sq-norm* (*w* − *?unscaled-u*)
**proof**−
**have** *?unscaled-u* ∈ *carrier-vec* (*d* + *1*) **using** *ts-to-u-carrier*[*of ts*] *t-length*
**by** *fastforce*
**moreover have** *carr-w*: *w*∈ *carrier-vec* (*d* + *1*) **using** *w gen-lattice-carrier*[*of*
*ts*] *t-length* **by** *blast*
**ultimately have** (*w* − *?unscaled-u*) = (−*1*)·*v* (*?unscaled-u* − *w*) **by** *fastforce*
**then have** *sq-norm-conjugate* (−*1*) ∗ *sq-norm* (*?unscaled-u* − *w*) = *sq-norm*
(*w* − *?unscaled-u*)
**using** *sq-norm-smult-vec* **by** *metis*
**then show** *?thesis* **by** *auto*
**qed**
**finally have** ∗: *real-of-rat* (*sq-norm* (*w* − *?unscaled-u*)) ≤ *real-of-rat* ( (*of-nat*
((*d+1*) ∗ *p^2*))/*2^*(*2*∗*k*) )
**using** *of-rat-less-eq* **by** *blast*
**have** ∀ *x*∈{*real-of-rat* (*sq-norm* (*y* − *?unscaled-u*))| *y*. *y*∈ *gen-lattice ts*}. *0*≤*x*
**using** *sq-norm-vec-ge-0* **by** *fastforce*
**then have** *bdd-below* {*real-of-rat* (*sq-norm* (*x* − *?unscaled-u*))| *x*. *x*∈ *gen-lattice*
*ts*} **by** *fast*
**then have** *Inf*{*real-of-rat* (*sq-norm* (*x* − *?unscaled-u*))| *x*. *x*∈ *gen-lattice ts*}
≤ *real-of-rat* ‖*w* − *ts-to-u ts*‖$^2$
**using** *w cInf-lower*[*of real-of-rat* (*sq-norm* (*w* − *?unscaled-u*)) {*real-of-rat*

(*sq-norm* (*x* − *?unscaled-u*))| *x*. *x*∈ *gen-lattice ts*}]
**by** *fast*
**then have** *1*: *Inf*{*real-of-rat* (*sq-norm* (*x* − *?unscaled-u*))| *x*. *x*∈ *gen-lattice ts*} ≤ *real-of-rat* ( (*of-nat* ((*d*+*1*) ∗ *p*^*2*))/*2*^(*2*∗*k*) ) **using** ∗
**by** *auto*
**have** *rat-u-vec* = ((*of-nat p*)·$_v$ (*ts-to-u ts*))
**using** *assms*(*2*) **unfolding** *u-vec-is-msbs-def rat-u-vec-def u-vec-def ts-def* **by** *auto*
**then have** *babai.closest-distance-sq basis rat-u-vec* = *real*(*p*^*2*) ∗ (*Inf*{*real-of-rat* (*sq-norm* (*x* − *?unscaled-u*))| *x*. *x*∈ *gen-lattice ts*})
**using** *gen-lattice-int-gen-lattice-closest*[*of ts ts-to-u ts*] *t-length ts-to-u-carrier*[*of ts*] **unfolding** *basis-def* **by** *auto*
**then have** *babai.closest-distance-sq basis rat-u-vec* ≤ *real*(*p*^*2*) ∗ *real-of-rat* ((*of-nat* ((*d*+*1*) ∗ *p*^*2*))/*2*^(*2*∗*k*) )
**using** *1 mult-left-mono*[*of Inf*{*real-of-rat* (*sq-norm* (*x* − *?unscaled-u*))| *x*. *x*∈ *gen-lattice ts*}
*real-of-rat* ((*of-nat* ((*d*+*1*) ∗ *p*^*2*))/*2*^(*2*∗*k*) ) *p*^*2*] *p-le* **by** *presburger*
**then show** *?thesis*
**by** (*metis of-nat-id of-nat-numeral of-nat-simps*(*5*) *of-rat-divide of-rat-of-nat-eq of-rat-power power2-eq-square times-divide-eq-right*)
**qed**
**ultimately have** (*4*/*3*)^(*d*+*1*) ∗ *real*(*d* + *1*) ∗ *11*/*10* ∗ *babai.closest-distance-sq basis rat-u-vec*
≤ (*4* / *3*) ^ (*d* + *1*) ∗ *real* (*d* + *1*) ∗ *11* / *10* ∗ *p*^*2* ∗ (*of-nat* ((*d*+*1*) ∗ *p*^*2*))/*2*^(*2*∗*k*)
**using** *mult-mono*[*of*
(*4* / *3*) ^ (*d* + *1*) ∗ *real* (*d* + *1*) ∗ *11* / *10*
(*4* / *3*) ^ (*d* + *1*) ∗ *real* (*d* + *1*) ∗ *11* / *10*
*babai.closest-distance-sq basis rat-u-vec*
*p*^*2* ∗ (*of-nat* ((*d*+*1*) ∗ *p*^*2*))/*2*^(*2*∗*k*)]
**by** *auto*
**then have** *real-of-int* (*sq-norm* (*full-babai basis rat-u-vec* (*4*/*3*) − *u-vec*)) ≤ (*4* / *3*) ^ (*d* + *1*) ∗ *real* (*d* + *1*) ∗ *11* / *10* ∗ *p*^*2* ∗ (*of-nat* ((*d*+*1*) ∗ *p*^*2*))/*2*^(*2*∗*k*)
**using** *1* **by** *linarith*
**then show** *?thesis* **using** *1* $\mathcal{A}$*-vec.simps*
**unfolding** *basis-def u-vec-def rat-u-vec-def ts-def int-gen-lattice-def*
**by** *presburger*
**qed**

**lemma** *ad-output-vec-class*:
**fixes** *pairs* :: (*nat* × *nat*) *list*
**assumes** *length pairs* = *d*
**assumes** *u-vec-is-msbs pairs*
**shows** *sq-norm* ( (*1*/(*rat-of-nat p*))·$_v$(*map-vec rat-of-int* ($\mathcal{A}$*-vec pairs*)) − (*ts-to-u* (*ts-from-pairs pairs*)) ) < ((*of-nat p*)/*2*^($\mu$))^*2*
∧ ( (*1*/(*rat-of-nat p*))·$_v$(*map-vec rat-of-int* ($\mathcal{A}$*-vec pairs*)) ∈ *vec-class-mod-p* (*ts-from-pairs pairs*) ($\mathcal{A}$ *pairs*))
**proof**−

**let** *?ts = ts-from-pairs pairs*
**have** *tl*: *length ?ts = d* **using** *assms(1)* **unfolding** *ts-from-pairs-def* **by** *simp*
**then have** *carrier-$\mathcal{A}$-vec*: *($\mathcal{A}$-vec pairs) ∈ carrier-vec (d + 1)*
**using** *ad-output-vec-close[of pairs] assms int-gen-lattice-carrier[of ts-from-pairs pairs]* **by** *fast*
**then have** *rat-carrier-ad*: *map-vec rat-of-int ($\mathcal{A}$-vec pairs) ∈ carrier-vec (d + 1)* **by** *fastforce*
**have** *carrier-scaled-u*: *scaled-uvec-from-pairs pairs ∈ carrier-vec (d + 1)*
**proof**−
**have** *length ((map (λ(a,b). p∗b) pairs) @ [0]) = d + 1*
**using** *assms* **by** *force*
**then show** *?thesis* **unfolding** *scaled-uvec-from-pairs-def*
**by** (*metis length-map vec-of-list-carrier*)
**qed**
**then have** *rat-carrier-scaled-u*: *map-vec rat-of-int (scaled-uvec-from-pairs pairs) ∈ carrier-vec (d + 1)* **by** *force*
**have** *close*: *real-of-int(sq-norm*
*(($\mathcal{A}$-vec pairs)*
*− (scaled-uvec-from-pairs pairs))) ≤*
*(4 / 3) ^ (d + 1) ∗ real (d + 1) ∗ 11 / 10 ∗ p^2 ∗ (of-nat ((d+1) ∗ p^2))/2^(2∗k)*
*∧ ($\mathcal{A}$-vec pairs) ∈ int-gen-lattice (ts-from-pairs pairs)*
**using** *ad-output-vec-close[of pairs] assms* **by** *simp*
**moreover have** *0 ≤ 1/(real-of-nat p^2)* **by** *auto*
**ultimately have** *1/(real-of-nat p^2) ∗ real-of-int(sq-norm (($\mathcal{A}$-vec pairs) − (scaled-uvec-from-pairs pairs)))*
*≤ 1/(real-of-nat p^2) ∗ (4 / 3) ^ (d + 1) ∗ real (d + 1) ∗ 11 / 10 ∗ p^2 ∗ (of-nat ((d+1) ∗ p^2))/2^(2∗k)*
**using** *mult-mono[of 1/(real-of-nat p^2) 1/(real-of-nat p^2) real-of-int(sq-norm (($\mathcal{A}$-vec pairs) − (scaled-uvec-from-pairs pairs))) (4 / 3) ^ (d + 1) ∗ real (d + 1) ∗ 11 / 10 ∗ p^2 ∗ (of-nat ((d+1) ∗ p^2))/2^(2∗k)]*
*sq-norm-vec-ge-0[of (($\mathcal{A}$-vec pairs) − (scaled-uvec-from-pairs pairs))]* **by** *force*
**moreover have** *1/(real-of-nat p^2) ∗ (4 / 3) ^ (d + 1) ∗ real (d + 1) ∗ 11 / 10 ∗ p^2 ∗ (of-nat ((d+1) ∗ p^2))/2^(2∗k)*
*= (4 / 3) ^ (d + 1) ∗ real (d + 1) ∗ 11 / 10 ∗ (of-nat ((d+1) ∗ p^2))/2^(2∗k)* **by** *simp*
**moreover have** *(4 / 3) ^ (d + 1) ∗ real (d + 1) ∗ 11 / 10 ∗ (of-nat ((d+1) ∗ p^2))/2^(2∗k) < ((of-nat p)/2^(μ))^2*
**proof**−
**have** *(((4::real)/3) powr ((d + 1)/2) ∗ (11/10) ∗ (d + 1) ∗ (p / (2 powr (real k − 1)))*
*< p / (2 powr μ))*
**using** *final-ineq* **by** *simp*
**moreover have** *0≤ (((4::real)/3) powr ((d + 1)/2) ∗ (11/10) ∗ (d + 1) ∗ (p / (2 powr (real k − 1))))*
**by** *fastforce*
**ultimately have** *(((4::real)/3) powr ((d + 1)/2) ∗ (11/10) ∗ (d + 1) ∗ (p / (2 powr (real k − 1))))^2*

```
    < (p / (2 powr μ))^2
      using Power.linordered-semidom-class.power-strict-mono pos2 by blast
    moreover have (((4::real)/3) powr ((d + 1)/2) * (11/10) * (d + 1) * (p /
(2 powr (real k − 1))))^2 =
          ( ((4::real)/3) powr ((d + 1)/2) )^2 * (11/10)^2 * (d + 1)^2 * ( (p /
(2 powr (real k − 1))) )^2
    by (metis (no-types, opaque-lifting) Power.semiring-1-class.of-nat-power power-mult-distrib)
    moreover have (((4::real)/3) powr ((d + 1)/2) )^2 = ((4::real)/3) ^ (d +
1)
    by (metis add-le-cancel-left divide-nonneg-nonneg le0 powr-ge-zero powr-half-sqrt-powr
powr-realpow' real-sqrt-pow2 rel-simps(45) semiring-norm(94))
    moreover have ( (p / (2 powr (real k − 1))) )^2 = p^2 / (2 ^ (2*k)) * 4
    proof−
      have k: 2 ≤ 2*k using k-geq-1 by auto
      have 2: (2::real) ≠ 0 by simp
      have ( (p / (2 powr (real k − 1))) )^2 = p^2 / (2 ^ (2*(k − 1)))
        using powr-realpow[of 2 k − 1] power-divide[of p 2 ^ (k − 1) 2]
      by (smt (verit) Multiseries-Expansion.intyness-1 Multiseries-Expansion.intyness-simps(3)
Multiseries-Expansion.intyness-simps(5) k-geq-1 le-trans of-nat-le-0-iff of-nat-le-iff
power-diff' power-divide power-even-eq powr-diff powr-realpow' semiring-norm(112)
semiring-norm(159))
      moreover have ((2::real) ^ (2*(k − 1))) = 2^(2*k−2)
        by auto
      moreover have (2::real)^(2*k−2) = 2^(2*k)/(2^2)
        using k power-diff'[of 2 2*k 2] 2 by meson
      moreover have  (2::real)^(2*k)/(2^2) =  2^(2*k)/(4)
        using power2-eq-square[of 2] by simp
      ultimately show ?thesis
        by (metis divide-divide-eq-right times-divide-eq-left)
    qed
     moreover have (p / (2 powr μ))^2 ≤ ((of-nat p)/2^μ)^2 by (simp add:
powr-realpow)
    ultimately have ((4::real)/3) ^ (d + 1) *  (11/10)^2 * (d + 1)^2 * (p^2 /
(2 ^ (2*k)) * 4) < ((of-nat p)/2^μ)^2
      by (smt (verit))
    moreover have ((11::real)/10)^2 = (11/10)*(11/10)
      using power2-eq-square by blast
    moreover have ((4::real)/3) ^ (d + 1) *  (11/10)*(11/10) * (d + 1)^2 *
(p^2 / (2 ^ (2*k)) * 4)
          = ((4::real)/3) ^ (d + 1) *  (11/10)*(11/10) * (d + 1)^2 * (p^2 /
(2 ^ (2*k))) * 4 by argo
    moreover have (d + 1)^2 * (p^2 / (2 ^ (2*k))) = (d + 1) * (of-nat ((d +
1) * p^2) / (2 ^ (2*k)))
       using power2-eq-square[of d + 1] mult-ac(1)[of d + 1 d + 1 (p^2 / (2 ^
(2*k)))]
    by (metis (no-types, lifting) more-arith-simps(11) of-nat-simps(5) times-divide-eq-right)
    ultimately have *: (4 * (11/10))* ( ((4::real)/3) ^ (d + 1) *  (11/10) * (d
+ 1) * (of-nat ((d + 1) * p^2) / (2 ^ (2*k)))) < ((of-nat p)/2^μ)^2
      using mult-ac by auto
```

**have** $1 < (4::real) * (11/10)$ **by** *simp*
**moreover have** $0 < (4::real) * (11/10)$ **by** *simp*
**moreover have** $0 \leq ( ((4::real)/3) \hat{} (d + 1) * (11/10) * (d + 1) * (of\text{-}nat ((d + 1) * p\hat{}2) / (2 \hat{} (2*k))))$ **by** *force*
**ultimately have** $1 * ( ((4::real)/3) \hat{} (d + 1) * (11/10) * (d + 1) * (of\text{-}nat ((d + 1) * p\hat{}2) / (2 \hat{} (2*k)))) \leq (4 * (11/10))* ( ((4::real)/3) \hat{} (d + 1) * (11/10) * (d + 1) * (of\text{-}nat ((d + 1) * p\hat{}2) / (2 \hat{} (2*k))))$
**using** *mult-mono*[*of* $1$ $(4::real) * (11/10)$ $( ((4::real)/3) \hat{} (d + 1) * (11/10) * (d + 1) * (of\text{-}nat ((d + 1) * p\hat{}2) / (2 \hat{} (2*k))))$ $( ((4::real)/3) \hat{} (d + 1) * (11/10) * (d + 1) * (of\text{-}nat ((d + 1) * p\hat{}2) / (2 \hat{} (2*k))))$]
**by** *argo*
**then show** *?thesis* **using** $*$ **by** *argo*
**qed**
**ultimately have** *rhs*: $1/(real\text{-}of\text{-}nat\ p\hat{}2) * real\text{-}of\text{-}int(sq\text{-}norm\ ((\mathcal{A}\text{-}vec\ pairs) - (scaled\text{-}uvec\text{-}from\text{-}pairs\ pairs))) < ((of\text{-}nat\ p)/2\hat{}(\mu))\hat{}2$ **by** *argo*
**have** $1/(real\text{-}of\text{-}nat\ p\hat{}2) * real\text{-}of\text{-}int(sq\text{-}norm\ ((\mathcal{A}\text{-}vec\ pairs) - (scaled\text{-}uvec\text{-}from\text{-}pairs\ pairs)))$
$= real\text{-}of\text{-}rat\ (1/(rat\text{-}of\text{-}nat\ p\hat{}2) * ((sq\text{-}norm\ (map\text{-}vec\ rat\text{-}of\text{-}int\ ((\mathcal{A}\text{-}vec\ pairs) - (scaled\text{-}uvec\text{-}from\text{-}pairs\ pairs)))\ ) ))$
**by** (*simp add*: *of-rat-divide of-rat-power sq-norm-of-int*)
**moreover have** $sq\text{-}norm\text{-}conjugate\ (1/(rat\text{-}of\text{-}nat\ p)) = (1/(rat\text{-}of\text{-}nat\ p\hat{}2))$
**using** *conjugatable-ring-1-abs-real-line-class.sq-norm-as-sq-abs*[*of* $1/(rat\text{-}of\text{-}nat\ p)$]
*power2-abs*[*of* $1$ / *rat-of-nat p*]
**by** (*simp add*: *power-divide*)
**ultimately have** $1/(real\text{-}of\text{-}nat\ p\hat{}2) * real\text{-}of\text{-}int(sq\text{-}norm\ ((\mathcal{A}\text{-}vec\ pairs) - (scaled\text{-}uvec\text{-}from\text{-}pairs\ pairs)))$
$= real\text{-}of\text{-}rat(\ (sq\text{-}norm\ ((1\ /\ rat\text{-}of\text{-}nat\ p) \cdot_v (map\text{-}vec\ rat\text{-}of\text{-}int\ ((\mathcal{A}\text{-}vec\ pairs) - (scaled\text{-}uvec\text{-}from\text{-}pairs\ pairs))))\ ) )$
**using** *sq-norm-smult-vec* **by** *metis*
**moreover have** $(real\ p\ /\ 2 \hat{} \mu)^2 = real\text{-}of\text{-}rat\ ((rat\text{-}of\text{-}nat\ p\ /\ 2 \hat{} \mu)^2)$
**by** (*simp add*: *of-rat-divide of-rat-power*)
**ultimately have** $(sq\text{-}norm\ ((1\ /\ rat\text{-}of\text{-}nat\ p) \cdot_v (map\text{-}vec\ rat\text{-}of\text{-}int\ ((\mathcal{A}\text{-}vec\ pairs) - (scaled\text{-}uvec\text{-}from\text{-}pairs\ pairs))))\ )$
$< ((of\text{-}nat\ p)/2\hat{}(\mu))\hat{}2$ **using** *rhs*
**by** (*simp add*: *of-rat-less*)
**moreover have** $(1\ /\ rat\text{-}of\text{-}nat\ p) \cdot_v (map\text{-}vec\ rat\text{-}of\text{-}int\ ((\mathcal{A}\text{-}vec\ pairs) - (scaled\text{-}uvec\text{-}from\text{-}pairs\ pairs)))$
$= (1\ /\ rat\text{-}of\text{-}nat\ p) \cdot_v ((map\text{-}vec\ rat\text{-}of\text{-}int\ (\mathcal{A}\text{-}vec\ pairs)) - (map\text{-}vec\ rat\text{-}of\text{-}int\ (scaled\text{-}uvec\text{-}from\text{-}pairs\ pairs)))$
**by** *fastforce*
**moreover have** $(1\ /\ rat\text{-}of\text{-}nat\ p) \cdot_v ((map\text{-}vec\ rat\text{-}of\text{-}int\ (\mathcal{A}\text{-}vec\ pairs)) - (map\text{-}vec\ rat\text{-}of\text{-}int\ (scaled\text{-}uvec\text{-}from\text{-}pairs\ pairs)))$
$= (1\ /\ rat\text{-}of\text{-}nat\ p) \cdot_v ((map\text{-}vec\ rat\text{-}of\text{-}int\ (\mathcal{A}\text{-}vec\ pairs)) + -(map\text{-}vec\ rat\text{-}of\text{-}int\ (scaled\text{-}uvec\text{-}from\text{-}pairs\ pairs)))$
**by** *force*
**moreover have** $(1\ /\ rat\text{-}of\text{-}nat\ p) \cdot_v ((map\text{-}vec\ rat\text{-}of\text{-}int\ (\mathcal{A}\text{-}vec\ pairs)) + -(map\text{-}vec\ rat\text{-}of\text{-}int\ (scaled\text{-}uvec\text{-}from\text{-}pairs\ pairs)))$
$= (1\ /\ rat\text{-}of\text{-}nat\ p) \cdot_v (map\text{-}vec\ rat\text{-}of\text{-}int\ (\mathcal{A}\text{-}vec\ pairs)) - (1\ /$

*rat-of-nat p*) $\cdot_v$ (*map-vec rat-of-int* (*scaled-uvec-from-pairs pairs*))
**using** *smult-add-distrib-vec*[*of* (*map-vec rat-of-int* (*A-vec pairs*)) *d* + *1* −(*map-vec rat-of-int* (*scaled-uvec-from-pairs pairs*))]
*rat-carrier-scaled-u rat-carrier-ad*
**by** (*metis calculation*(*3*) *smult-sub-distrib-vec*)
**moreover have** (*1* / *rat-of-nat p*) $\cdot_v$ (*map-vec rat-of-int* (*scaled-uvec-from-pairs pairs*)) = *ts-to-u* (*ts-from-pairs pairs*)
**proof**−
**have** (*1* / *rat-of-nat p*) $\cdot_v$ (*map-vec rat-of-int* (*scaled-uvec-from-pairs pairs*))
= (*1* / *rat-of-nat p*) $\cdot_v$ ( ((*of-nat p*) $\cdot_v$ (*ts-to-u* (*ts-from-pairs pairs*))))
**using** *assms*(*2*) **unfolding** *u-vec-is-msbs-def* **by** *argo*
**then show** *?thesis* **using** *tl smult-smult-assoc*[*of* (*1* / *rat-of-nat p*) (*of-nat p*)] *p*
**by** *auto*
**qed**
**ultimately have** *part1*: *sq-norm* ((*1* / *rat-of-nat p*) $\cdot_v$ (*map-vec rat-of-int* (*A-vec pairs*)) − *ts-to-u* (*ts-from-pairs pairs*))
< ((*of-nat p*)/*2*^(*μ*))^*2* **by** *argo*
**let** *?v* = (*1* / *rat-of-nat p*) $\cdot_v$ (*map-vec rat-of-int* (*A-vec pairs*))
**have** *?v* ∈ *gen-lattice ?ts* **using** *tl*
**using** *gen-lattice-int-gen-lattice-vec*[*of ?ts A-vec pairs*] *close* **by** *fastforce*
**then obtain** *c* **where** *c*: *?v* = (*sumlist* (*map* (λ*i*. *of-int* (*c i*) $\cdot_v$ ((*gen-basis ?ts*) ! *i*)) [*0* ..< *length* (*gen-basis ?ts*)]))
**unfolding** *gen-lattice-def lattice-of-def* **by** *blast*
**then have** *?v*$*d* = (*rat-of-int* (*c d*)) / *rat-of-nat p*
**using** *coordinates-of-gen-lattice*[*of d ?ts*] *tl* **by** *simp*
**moreover have** *?v*$*d* = (*1* / *rat-of-nat p*) ∗ (*rat-of-int* ((*A-vec pairs*)$*d*))
**using** *rat-carrier-ad carrier-A-vec*
*index-map-vec*(*1*)[*of d A-vec pairs rat-of-int*] *index-smult-vec*(*1*)[*of d map-vec rat-of-int* (*A-vec pairs*) *1* / (*rat-of-nat p*)]
*carrier-vecD*[*of - d* + *1*] **by** *force*
**ultimately have** *c d* = (*A-vec pairs*)$*d*
**using** *mult-cancel-right2 not-one-le-zero of-nat-eq-0-iff p prime-ge-1-nat times-divide-eq-left* **by** *auto*
**then have** [*int* (*A pairs*) = (*c d*)] (*mod p*)
**using** *A.simps*[*of pairs*] *int-to-nat-residue.simps*[*of A-vec pairs* $*d p*] *of-nat-0-less-iff p pos-mod-bound prime-gt-0-nat* **by** *auto*
**moreover have***?v* ∈ *vec-class ?ts* (*c d*)
**proof**−
**have** [*0* ..< *length* (*gen-basis ?ts*)] = [*0* ..< *d*] @ [*d*]
**using** *gen-basis-length*[*of ?ts*] *d* **by** *simp*
**moreover have** *set* (*map* (λ*i*. *of-int* (*c i*) $\cdot_v$ ((*gen-basis ?ts*) ! *i*)) [*0* ..< *d*]) ⊆ *carrier-vec* (*d* + *1*)
**using** *gen-basis-carrier*[*of ?ts*] *tl*
**proof**−
**have** *v* ∈ *carrier-vec* (*d* + *1*) **if** *v*: *v* ∈ *set* (*map* (λ*i*. *of-int* (*c i*) $\cdot_v$ ((*gen-basis ?ts*) ! *i*)) [*0* ..< *d*]) **for** *v*
**proof**−
**obtain** *i* **where** *i*: *v* = *of-int* (*c i*) $\cdot_v$ ((*gen-basis ?ts*) ! *i*) ∧ *i*∈{*0*..<*d*}

**using** *v* **by** *auto*
**then show** *v* ∈ *carrier-vec* (*d* + *1*) **using** *gen-basis-vecs-carrier*[*of ?ts i*] *tl* **by** *simp*
**qed**
**then show** *?thesis* **by** *fast*
**qed**
**moreover have** (*of-int* (*c d*) ·$_v$ ((*gen-basis ?ts*) ! *d*)) ∈ *carrier-vec* (*d* + *1*) **using** *gen-basis-vecs-carrier*[*of ?ts d*] *tl* **by** *simp*
**ultimately have** ∗: *?v* = (*sumlist* (*map* (λ*i*. *of-int* (*c i*) ·$_v$ ((*gen-basis ?ts*) ! *i*)) [*0* ..< *d*])) + (*of-int* (*c d*) ·$_v$ ((*gen-basis ?ts*) ! *d*))
**using** *gen-basis-carrier*[*of ?ts*] *tl c*
*sumlist-snoc*[*of* (*map* (λ*i*. *of-int* (*c i*) ·$_v$ ((*gen-basis ?ts*) ! *i*)) [*0* ..< *d*]) (*of-int* (*c d*) ·$_v$ ((*gen-basis ?ts*) ! *d*))]
**by** *fastforce*
**have** (*gen-basis ?ts*)!*i* = *p-vecs*!*i* **if** *i*: *i*∈{*0*..<*d*} **for** *i*
**unfolding** *gen-basis-def*
**using** *i length-p-vecs append-Cons-nth-left*[*of i p-vecs vec-of-list* (*map rat-of-nat* (*ts-from-pairs pairs*) @ [*1* / *rat-of-nat p*]) []]
**by** *force*
**then have** (*sumlist* (*map* (λ*i*. *of-int* (*c i*) ·$_v$ ((*gen-basis ?ts*) ! *i*)) [*0* ..< *d*])) = (*sumlist* (*map* (λ*i*. *of-int* (*c i*) ·$_v$ (*p-vecs* ! *i*)) [*0* ..< *d*]))
**by** (*smt* (*verit*) *List.List.list.map-cong atLeastLessThan-upt*)
**moreover have** (*sumlist* (*map* (λ*i*. *of-int* (*c i*) ·$_v$ (*p-vecs* ! *i*)) [*0* ..< *d*])) = (*sumlist* (*map* (λ*i*. ((*of-int* ∘ *c*) *i*) ·$_v$ (*p-vecs* ! *i*)) [*0* ..< *d*]))
**by** *auto*
**moreover have** (*sumlist* (*map* (λ*i*. ((*of-int* ∘ *c*) *i*) ·$_v$ (*p-vecs* ! *i*)) [*0* ..< *d*])) = *lincomb-list* (*of-int* ∘ *c*) *p-vecs*
**unfolding** *lincomb-list-def* **using** *length-p-vecs* **by** *presburger*
**moreover have** ((*gen-basis ?ts*) ! *d*) = *t-vec ?ts*
**unfolding** *gen-basis-def t-vec-def*
**using** *append-Cons-nth-middle*[*of d p-vecs*] *length-p-vecs* **by** *presburger*
**ultimately have** *?v* = *lincomb-list* (*of-int* ∘ *c*) *p-vecs* + (*of-int* (*c d*) ·$_v$ (*t-vec ?ts*))
**using** ∗ **by** *argo*
**moreover have** *lincomb-list* (*rat-of-int* ∘ *c*) *p-vecs* ∈ *carrier-vec* (*d* + *1*)
**using** *p-vecs-carrier carrier-vecI lincomb-list-carrier*[*of p-vecs of-int* ∘ *c*] **by** *fast*
**moreover have** *of-int* (*c d*) ·$_v$ (*t-vec ?ts*) ∈ *carrier-vec* (*d* + *1*)
**using** *t-vec-dim*[*of ?ts*] *carrier-vecI*[*of t-vec ?ts d* + *1*] *assms*(*1*) **unfolding** *ts-from-pairs-def* **by** *force*
**ultimately have** *?v* = (*of-int* (*c d*) ·$_v$ (*t-vec ?ts*)) + *lincomb-list* (*of-int* ∘ *c*) *p-vecs*
**by** *force*
**then show** *?thesis* **unfolding** *vec-class-def* **by** *blast*
**qed**
**ultimately have** *?v* ∈ *vec-class-mod-p ?ts* (*int* (*A pairs*))
**unfolding** *vec-class-mod-p-def* **by** *blast*
**then show** *?thesis* **using** *part1* **by** *blast*
**qed**

If we get a good lattice (which is likely), the adversary finds the hidden number

**lemma** *hnp-adversary-exists-helper*:
**assumes** $ts \in$ *set-pmf ts-pmf*
**defines** *ts-Ots* $\equiv$ *map* $(\lambda t.\ (t,\ \mathcal{O}\ t))$ *ts*
**assumes** *good-lattice ts*
**shows** $\alpha = \mathcal{A}$ *ts-Ots*
**proof**−
**let** *?A-vec* = $(1/($*rat-of-nat p*$))\cdot_v$(*map-vec rat-of-int* ($\mathcal{A}$*-vec ts-Ots*))
**let** *?vec1* = *rat-of-nat p* $\cdot_v$ *vec-of-list*
(*map rat-of-nat*
(*ts-to-as* (*map* $(\lambda(a,\ b).\ a)$ (*map* $(\lambda t.\ (t,\ \mathcal{O}\ t))$ *ts*))) @ [*0*])
**let** *?vec2* = *Matrix.of-int-hom.vec-hom* (*vec-of-list*
(*map int* (*map* $(\lambda(a,\ b).\ p * b)$ (*map* $(\lambda t.\ (t,\ \mathcal{O}\ t))$ *ts*) @ [*0*])))

**have** *dim-v1*: *dim-vec ?vec1* = *length ts* + *1*
**unfolding** *ts-to-as-def* **by** *auto*
**have** *dim-v2*: *dim-vec ?vec2* = *length ts* + *1*
**by** *auto*

**have** *l*: *length ts* = *d* **using** *assms*(*1*) *set-pmf-ts* **by** *blast*
**then have** *ts-l*: *length ts-Ots* = *d* **unfolding** *ts-Ots-def* **by** *simp*
**have** *ts-from-pairs*: *ts-from-pairs ts-Ots* = *ts*
**proof**−
**have** *map* $(\lambda(a,\ b).\ a)$ (*map* $(\lambda t.\ (t,\ \mathcal{O}\ t))$ *ts*) = *map* ( $(\lambda(a,\ b).\ a) \circ (\lambda t.\ (t,\ \mathcal{O}\ t))$ ) *ts* **by** *simp*
**moreover have** $(\lambda(a,\ b).\ a) \circ (\lambda t.\ (t,\ \mathcal{O}\ t)) = (\lambda t.\ t)$ **by** *fastforce*
**ultimately show** *?thesis* **unfolding** *ts-from-pairs-def ts-Ots-def fst-def* **by** *simp*
**qed**

**have** *?vec1*$*i* = *?vec2*$*i* **if** *in-range*: $i \in \{0..<(d + 1)\}$ **for** *i*
**proof**(−)
**have** *?vec1*$*i*
= (*rat-of-nat p*) ∗ (*vec-of-list* (*map rat-of-nat* (*ts-to-as* (*map* $(\lambda(a,\ b).\ a)$ (*map* $(\lambda t.\ (t,\ \mathcal{O}\ t))$ *ts*))) @ [*0*]) $ *i*)
**using** *in-range dim-v1 l* **by** *simp*
**then have** *vec1-help*: *?vec1*$*i* = (*rat-of-nat p*) ∗ ((*map rat-of-nat* (*ts-to-as* (*map* $(\lambda(a,\ b).\ a)$ (*map* $(\lambda t.\ (t,\ \mathcal{O}\ t))$ *ts*))) @ [*0*])!*i*) **by** *simp*
**have** *vec2-help*: *?vec2*$*i* = *rat-of-int* ((*map int* (*map* $(\lambda(a,\ b).\ p * b)$ (*map* $(\lambda t.\ (t,\ \mathcal{O}\ t))$ *ts*) @ [*0*]))!*i*)
**using** *in-range dim-v2 l* **by** *auto*
**have** *l1*: *length* (*map rat-of-nat* (*ts-to-as* (*map* $(\lambda(a,\ b).\ a)$ (*map* $(\lambda t.\ (t,\ \mathcal{O}\ t))$ *ts*)))) = *d* **using** *l* **unfolding** *ts-to-as-def* **by** *simp*
**have** *l2*: *length* (*map int* (*map* $(\lambda(a,\ b).\ p * b)$ (*map* $(\lambda t.\ (t,\ \mathcal{O}\ t))$ *ts*))) = *d* **using** *l* **by** *simp*
**show** *?thesis*
**proof**(*cases i* = *d*)
**case** *t*:*True*

**then have** (*map rat-of-nat* (*ts-to-as* (*map* (λ(*a*, *b*). *a*) (*map* (λ*t*. (*t*, 𝒪 *t*)) *ts*))) @ [*0*]) ! *i* = *0*
**using** *append-Cons-nth-middle*[*of i map rat-of-nat* (*ts-to-as* (*map* (λ(*a*, *b*). *a*) (*map* (λ*t*. (*t*, 𝒪 *t*)) *ts*))) *0* []] **using** *l1* **by** *blast*
**then have** *1*: *?vec1*$*i* = *0* **using** *vec1-help* **by** *auto*
**then show** *?thesis*
**using** *vec2-help l2 append-Cons-nth-middle*[*of i* (*map int* (*map* (λ(*a*, *b*). *p* ∗ *b*) (*map* (λ*t*. (*t*, 𝒪 *t*)) *ts*))) *0* []] *t*
**by** *fastforce*
**next**
**case** *f*:*False*
**then have** *?vec1*$*i* = (*rat-of-nat p*) ∗ (*rat-of-nat* ( (*ts-to-as* (*map* (λ(*a*, *b*). *a*) (*map* (λ*t*. (*t*, 𝒪 *t*)) *ts*))) ! *i*) )
**using** *vec1-help in-range l1 append-Cons-nth-left*[*of i map rat-of-nat* (*ts-to-as* (*map* (λ(*a*, *b*). *a*) (*map* (λ*t*. (*t*, 𝒪 *t*)) *ts*))) *0* []] **by** *simp*
**also have** ... = (*rat-of-nat p*) ∗ *rat-of-nat* ( (*msb-p* (α ∗ (*ts*!*i*) *mod p*) ) )
**using** *l f in-range ts-from-pairs* **unfolding** *ts-to-as-def ts-from-pairs-def ts-Ots-def* **by** *simp*
**also have** ... = (*rat-of-nat p*) ∗ *rat-of-nat* (𝒪 (*ts*!*i*)) **unfolding** 𝒪-*def* **by** *blast*
**also have** ... = *rat-of-int* (*map int* (*map* (λ(*a*, *b*). *p* ∗ *b*) (*map* (λ*t*. (*t*, 𝒪 *t*)) *ts*)) ! *i*)
**using** *l f in-range ts-from-pairs* **unfolding** *ts-to-as-def ts-from-pairs-def ts-Ots-def* **by** *simp*
**also have** ... = *rat-of-int* (*map int* (*map* (λ(*a*, *b*). *p* ∗ *b*) (*map* (λ*t*. (*t*, 𝒪 *t*)) *ts*) @ [*0*]) ! *i*)
**using** *l2 in-range append-Cons-nth-left*[*of i* (*map int* (*map* (λ(*a*, *b*). *p* ∗ *b*) (*map* (λ*t*. (*t*, 𝒪 *t*)) *ts*))) *0* []] *f* **by** *simp*
**finally show** *?thesis* **using** *vec2-help* **by** *linarith*
**qed**
**qed**
**then have** *vec1-is-vec2*: *?vec1* = *?vec2*
**using** *dim-v1 dim-v2 l* **by** *auto*
**then have** *u-vec-is-msbs ts-Ots*
**unfolding** *u-vec-is-msbs-def*
**unfolding** *ts-Ots-def*
**unfolding** *scaled-uvec-from-pairs-def ts-from-pairs-def fst-def ts-to-u-def*
**by** *argo*
**then have** ∗: ‖*?*𝒜*-vec* − *ts-to-u ts*‖² < (*rat-of-nat p* / *2* ^ μ)² ∧ *?*𝒜*-vec* ∈ *vec-class-mod-p ts* (*int* (𝒜 *ts-Ots*))
**unfolding** *close-vec-def*
**using** *ad-output-vec-class*[*of ts-Ots*] *ts-l ts-from-pairs*
**by** *blast*
**then have** *close*: *close-vec ts ?*𝒜*-vec* **unfolding** *close-vec-def* **using** *uminus-sq-norm*

**by** (*metis* (*no-types*, *lifting*) *int-gen-lattice-carrier ts-to-u-carrier* ‹*ts-from-pairs ts-Ots* = *ts*› ‹*u-vec-is-msbs ts-Ots*› *ad-output-vec-close assms*(*2*) *basic-trans-rules*(*31*) *carrier-dim-vec index-smult-vec*(*2*) *length-map map-carrier-vec ts-l*)
**obtain** *c* **where** *?*𝒜*-vec* ∈ *vec-class ts c*

**using** ∗ **unfolding** *vec-class-mod-p-def* **by** *fast*
**then have** *gl*: *?𝒜-vec* ∈ *gen-lattice ts*
**using** *vec-class-union*[*of ts*] *assms*(*1*) **by** *blast*
**then have** *dim*: *dim-vec* (*𝒜-vec ts-Ots*) = *d* + *1* **using** *gen-lattice-carrier*[*of ts*] *assms*(*1*) *l* **by** *fastforce*
**have** *good-vec ?𝒜-vec* **using** *assms*(*3*) *close gl* **unfolding** *good-lattice-def* **by** *blast*
**then obtain** *β* **where** *b*: [*int α* = *β*] (*mod int p*) ∧ *real-of-rat* (*?𝒜-vec* $ *d*) = *real-of-int β* / *real p* **unfolding** *good-vec-def* **by** *blast*
**then have** *p* ∗ *real-of-rat* (*?𝒜-vec* $ *d*) = *β* **using** *p* **by** *simp*
**then have** *real p* ∗ *real-of-rat* ( (*1*/(*rat-of-nat p*)) ∗ ((*map-vec rat-of-int* ((*𝒜-vec ts-Ots*)) $ *d*)) ) = *β*
**using** *index-smult-vec*(*1*)[*of d* (*𝒜-vec ts-Ots*)] *dim* **by** *force*
**then have** *real p* ∗ *real-of-rat* (*1*/(*rat-of-nat p*) ∗ (*rat-of-int* ((*𝒜-vec ts-Ots*) $ *d*))) = *β*
**using** *dim index-map-vec*(*1*)[*of d* (*𝒜-vec ts-Ots*) *rat-of-int*] **by** *simp*
**then have** *real p* ∗ *real-of-rat* (*1*/(*rat-of-nat p*)) ∗ *real-of-rat* (*rat-of-int* ((*𝒜-vec ts-Ots*) $ *d*)) = *β*
**by** (*metis more-arith-simps*(*11*) *of-rat-mult*)
**moreover have** *real p* ∗ *real-of-rat* (*1*/(*rat-of-nat p*)) = *1*
**by** (*metis* (*no-types*, *lifting*) *Ring-Hom.of-rat-hom.hom-div Ring-Hom.of-rat-hom.hom-one divide-cancel-right nonzero-mult-div-cancel-left not-prime-0 of-nat-eq-0-iff of-rat-of-nat-eq p*)
**ultimately have** (*𝒜-vec ts-Ots*) $ *d* = *β*
**by** *fastforce*
**then have** [*int* (*𝒜 ts-Ots*) = *β*] (*mod int p*) **using** *𝒜.simps*[*of ts-Ots*] *int-to-nat-residue.simps*[*of* (*𝒜-vec ts-Ots* $ *d*) *p*]
**by** (*metis* (*no-types*, *lifting*) *Cong.unique-euclidean-semiring-class.cong-def int-mod-eq int-nat-eq local.𝒜.elims local.int-to-nat-residue.simps m2pths*(*1*) *of-nat-0-less-iff p pos-mod-bound prime-gt-0-nat*)
**then have** [(*𝒜 ts-Ots*) = *α*] (*mod p*) **using** *b*
**by** (*meson cong-int-iff cong-sym cong-trans*)
**moreover have** *𝒜 ts-Ots* ∈ {*0*..<*p*}
**using** *𝒜.simps*[*of ts-Ots*] *int-to-nat-residue.simps*[*of* (*𝒜-vec ts-Ots* $ *d*) *p*]
**by** (*metis Cong.unique-euclidean-semiring-class.cong-def α* ‹*𝒜-vec ts-Ots* $ *d* = *β*› *atLeastLessThan-iff b int-ops*(*9*) *le0 local.int-to-nat-residue.simps mod-less nat-int*)
**ultimately show** *?thesis*
**by** (*metis α atLeastLessThan-iff cong-less-modulus-unique-nat*)
**qed**

### 11.3.3 Main theorem statement

The cryptographic game which the adversary should win with high probability.

**definition** *game* :: *adversary* ⇒ *bool pmf* **where**
*game 𝒜′* = *do* {
*ts* ← *replicate-pmf d* (*pmf-of-set* {*1*..<*p*});
*return-pmf* (*α* = *𝒜′* (*map* (λ*t*. (*t*, *𝒪 t*)) *ts*))

}

The adversary finds the hidden number with probability at least 1/2.

**theorem** *hnp-adversary-exists*: *pmf* (*game* $\mathcal{A}$) *True* $\geq$ *1/2*
**proof** −
**define** *game′* **where** *game′* ≡
*do* {
*ts* ← *replicate-pmf d* (*pmf-of-set* {*1*..<*p*});
*return-pmf* (*good-lattice ts*)
}

**have** *1*: ∀ *ts* ∈ *set-pmf ts-pmf*. *good-lattice ts* ⟶ (α = $\mathcal{A}$ (*map* (λ*t*. (*t*, $\mathcal{O}$ *t*)) *ts*))
**using** *hnp-adversary-exists-helper* **by** *simp*

**have** *1 / 2* ≤ *pmf game′ True*
**using** *sampled-lattice-likely-good*
**unfolding** *sampled-lattice-good-def game′-def ts-pmf-def* **.**
**also have** ... ≤ *pmf* (*game* $\mathcal{A}$) *True*
**unfolding** *game′-def game-def*
**using** *pmf-subset*[*OF 1*, *unfolded ts-pmf-def*]
**by** *presburger*
**finally show** *?thesis* **.**
**qed**

Alternative definition of *game*, written as a *map-pmf*

**definition** *game′* :: *adversary* ⇒ *bool pmf* **where**
*game′* $\mathcal{A}'$ = *map-pmf* ((λ*ts*. (α = $\mathcal{A}'$ (*map* (λ*t*. (*t*, $\mathcal{O}$ *t*)) *ts*)))) (*replicate-pmf d* (*pmf-of-set* {*1*..<*p*}))

**lemma** *game′* = *game*
**unfolding** *game′-def game-def map-pmf-def* **by** *argo*

**end**

# 12 Some MSB instantiations and lemmas

## 12.1 Bit-shift MSB

**definition** *msb* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ *nat* **where**
*msb k n x* ≡ (*x div 2*^(*n* − *k*)) ∗ *2*^(*n* − *k*)

**lemma** *msb-dist*:
**fixes** *k n* :: *nat*
**assumes** *k* < *n*
**assumes** *1* ≤ *k*
**shows** |*int* (*x*) − *int* (*msb k n x*)| < *2*^*n* / (*2*^*k*)
**proof** −
**define** *z* **where** *z* = *msb k n x*

```
  have k-small: k ∈{1..<n}
    by (meson assms atLeastLessThan-iff nat-ceiling-le-eq not-le)
  have abs(int(x) − int(z)) < real(2^(n − k))
  proof−
    have z = int x div 2 ^ (n − k) ∗ 2 ^ (n − k)
      unfolding z-def msb-def
      by (metis int-ops(7) int-ops(8) numeral-power-eq-of-nat-cancel-iff)
    then have int(x) − int(z) = int(x) mod 2^(n − k)
      using minus-div-mult-eq-mod[of int(x) 2^(n − k)] by presburger
    then show ?thesis by fastforce
  qed
  also have ... = 2^n / 2^k
    by (metis Suc-leI assms(1) ge-trans le-add2 numeral-power-eq-of-nat-cancel-iff
plus-1-eq-Suc power-diff semiring-norm(142))
  finally show ?thesis unfolding z-def .
qed

lemma (in hnp-arith) msb-kp1-dist-hnp:
  defines msb-k ≡ msb (k + 1) n
  shows |int (x) − int (msb-k x)| < p / (2^k)
proof−
  have 1: k + 1 < n using k-plus-1-lt n unfolding hnp-def by linarith
  have 2: 1 ≤ k + 1 using μ-le-k by linarith

  have |int (x) − int(msb-k x)| < 2 ^ n / 2 ^ (k + 1)
    using msb-dist[OF 1 2, of x, folded msb-k-def] .
  also have ... < p / (2^k)
  proof−
    have 2^(n − 1) < p
    proof−
      have 2 powr (floor (log 2 p)) ≤ 2 powr (log 2 p) by simp
      moreover have n − 1 ≤ floor (log 2 p) using n n-geq-1 by linarith
     ultimately have 2 powr (n − 1) ≤ 2 powr (log 2 p) by (simp add: le-floor-iff)
      hence 2^(n − 1) ≤ 2 powr (log 2 p) by (simp add: powr-realpow)
      thus ?thesis using p-geq-2 less-le-trans n n-geq-1 by fastforce
    qed
    hence 2^(n − 1) / 2^k < p / 2^k by (simp add: divide-strict-right-mono)
    thus ?thesis using le-imp-diff-is-add n-geq-1 by fastforce
  qed
  finally show ?thesis .
qed

lemma msb-kp1-valid-hnp:
  assumes hnp-arith n α d p k
  shows hnp n α d p k (msb (k + 1) n)
  using hnp-arith.msb-kp1-dist-hnp[OF assms(1)] assms(1)
  unfolding hnp-def hnp-arith-def hnp-axioms-def
  by presburger
```

## 12.2 MSB-p

**definition** *msb-p* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ *nat* **where**
*msb-p p k x* =
(*let t* = (*THE t*. (*t* − *1*) ∗ (*p* / *2*^*k*) ≤ *x* ∧ *x* < *t* ∗ *p* / *2*^*k*)
*in nat* (*floor* (*t* ∗ *p* / *2*^*k*)) − *1*)

**lemma** *msb-p-defined*:
**fixes** *p k* :: *nat*
**fixes** *x* :: *nat*
**assumes** *p* > *0*
**assumes** *k* > *0*
**shows** (∃!*t*. (*t* − *1*) ∗ (*p* / *2*^*k*) ≤ *x* ∧ *x* < *t* ∗ *p* / *2*^*k*)
**proof** *safe*
**show** ∃ *t*. (*real t* − *1*) ∗ (*real p* / *2* ^ *k*) ≤ *real x* ∧ *real x* < *real* (*t* ∗ *p*) / *2* ^ *k*
**proof**
**let** *?S* = {*t*. (*real t* − *1*) ∗ (*real p* / *2* ^ *k*) ≤ *real x*}
**define** *t* **where** *t* = *Max ?S*
**have** *?S* ≠ {}
**by** (*metis Multiseries-Expansion.intyness-1 arith-simps*(*62*) *empty-iff mem-Collect-eq of-nat-0-le-iff verit-minus-simplify*(*1*))
**moreover have** *finite ?S*
**proof**−
**obtain** *M* **where** *?S* ⊆ {*0*..<*M*}
**proof**−
**let** *?M* = *nat* (*ceiling* (*x* / (*p* / *2*^*k*) + *1*)) + *1*
**have** *?S* ⊆ {*0*..<*?M*}
**proof**
**fix** *t* **assume** *t* ∈ *?S*
**moreover have** *p* / *2*^*k* > *0* **by** (*simp add*: *assms*(*1*))
**ultimately have** *real t* − *1* ≤ *x* / (*p* / *2*^*k*) **using** *mult-imp-le-div-pos* **by** *blast*
**hence** *t* ≤ *x* / (*p* / *2*^*k*) + *1* **by** *argo*
**hence** *t* < *nat* (*ceiling* (*x* / (*p* / *2*^*k*) + *1*)) + *1* **by** *linarith*
**moreover have** *0* ≤ *t* **by** *simp*
**ultimately show** *t* ∈ {*0*..<*?M*} **by** *fastforce*
**qed**
**thus** *?thesis* **using** *that*[*of ?M*] **by** *blast*
**qed**
**moreover have** *finite* {*0*..<*M*} **by** *blast*
**ultimately show** *?thesis* **using** *rev-finite-subset*[*of* {*0*..<*M*} *?S*] **by** *fast*
**qed**
**ultimately have** *t* ∈ *?S* ∧ (∀ *t*′ ∈ *?S*. *t*′ ≤ *t*) **using** *Max-eq-iff t-def* **by** *blast*
**moreover then have** *t* + *1* ∉ *?S* **by** (*metis Suc-eq-plus1 not-less-eq-eq*)
**ultimately show** (*real t* − *1*) ∗ (*real p* / *2* ^ *k*) ≤ *real x* ∧ *real x* < *real* (*t* ∗ *p*) / *2* ^ *k*
**by** *force*
**qed**
**next**
**fix** *x t t*′

**assume** $t$: $(real\ t - 1) * (real\ p\ /\ 2\ \hat{}\ k) \leq real\ x\ real\ x < real\ (t * p)\ /\ 2\ \hat{}\ k$
**assume** $t'$: $(real\ t' - 1) * (real\ p\ /\ 2\ \hat{}\ k) \leq real\ x\ real\ x < real\ (t' * p)\ /\ 2\ \hat{}\ k$

**have** $*$: $p\ /\ 2\hat{}k > 0$ **by** $(simp\ add{:}\ assms(1))$

**have** $(real\ t' - 1) * (real\ p\ /\ 2\ \hat{}\ k) < real\ (t * p)\ /\ 2\ \hat{}\ k$
**using** $t(2)\ t'(1)$ **by** *linarith*
**also have** $\ldots = t * (p\ /\ 2\hat{}k)$ **by** *fastforce*
**finally have** $(real\ t' - 1) < t$
**by** $(meson * le\text{-}less\text{-}trans\ less\text{-}eq\text{-}real\text{-}def\ mult\text{-}imp\text{-}le\text{-}div\text{-}pos\ pos\text{-}divide\text{-}less\text{-}eq)$
**hence** $1{:}\ t' - 1 < t$ **using** $le\text{-}less\ t(2)$ **by** *fastforce*

**have** $(real\ t - 1) * (real\ p\ /\ 2\ \hat{}\ k) < real\ (t' * p)\ /\ 2\ \hat{}\ k$
**using** $t(1)\ t'(2)$ **by** *linarith*
**also have** $\ldots = t' * (p\ /\ 2\hat{}k)$ **by** *fastforce*
**finally have** $(real\ t - 1) < t'$
**by** $(meson * le\text{-}less\text{-}trans\ less\text{-}eq\text{-}real\text{-}def\ mult\text{-}imp\text{-}le\text{-}div\text{-}pos\ pos\text{-}divide\text{-}less\text{-}eq)$
**hence** $2{:}\ t - 1 < t'$ **using** $1$ **by** *linarith*

**show** $t = t'$ **using** $1\ 2$ **by** *linarith*
**qed**

**lemma** (**in** *hnp-arith*) *msb-p-dist-hnp*:
**defines** $msb\text{-}k \equiv msb\text{-}p\ p\ k$
**shows** $|int\ x - int\ (msb\text{-}k\ x)| < p\ /\ 2\hat{}k$
**proof**−
**let** $?P = \lambda t.\ (t - 1) * (p\ /\ 2\hat{}k) \leq x \wedge x < t * p\ /\ 2\hat{}k$
**define** $t$ **where** $t = (THE\ t.\ ?P\ t)$
**have** $1{:}\ p > 0$ **using** $p\text{-}geq\text{-}2$ **by** *simp*
**have** $2{:}\ k > 0$ **using** $\mu\text{-}le\text{-}k$ **by** *linarith*
**have** $*$: $(t - 1) * (p\ /\ 2\hat{}k) \leq x \wedge x < t * p\ /\ 2\hat{}k$
**using** $theI\text{-}unique[OF\ msb\text{-}p\text{-}defined[OF\ 1\ 2,\ of\ x],\ of\ t,\ folded\ t\text{-}def]\ of\text{-}nat\text{-}diff\text{-}real$
**by** *fastforce*
**moreover have** $(t - 1) * (p\ /\ 2\hat{}k) = (t * (p\ /\ 2\hat{}k)) - (p\ /\ 2\hat{}k)$
**by** $(metis\ (no\text{-}types,\ opaque\text{-}lifting)\ Nat.bot\text{-}nat\text{-}0.not\text{-}eq\text{-}extremum\ One\text{-}nat\text{-}def\ Suc\text{-}pred * diff\text{-}is\text{-}0\text{-}eq'\ divide\text{-}eq\text{-}0\text{-}iff\ left\text{-}diff\text{-}distrib\ more\text{-}arith\text{-}simps(5)\ mult\text{-}eq\text{-}0\text{-}iff\ nat\text{-}le\text{-}linear\ not\text{-}le\ numeral\text{-}One\ of\text{-}nat\text{-}diff\ of\text{-}nat\text{-}eq\text{-}0\text{-}iff\ of\text{-}nat\text{-}numeral)$
**moreover have** $t * p\ /\ 2\hat{}k - ((t * (p\ /\ 2\hat{}k)) - p\ /\ 2\hat{}k) = p\ /\ 2\hat{}k$ **by** *simp*
**ultimately have** $nat\ (floor\ (t * p\ /\ 2\hat{}k) - 1) - x < p\ /\ 2\hat{}k$ **by** *linarith*
**moreover have** $p\ /\ 2\hat{}k > 1$
**by** $(smt\ (verit,\ ccfv\text{-}threshold)\ 1\ k\text{-}plus\text{-}1\text{-}lt\ less\text{-}divide\text{-}eq\text{-}1\text{-}pos\ log\text{-}of\text{-}power\text{-}le\ of\text{-}nat\text{-}0\text{-}le\text{-}iff\ of\text{-}nat\text{-}add\ zero\text{-}less\text{-}power)$
**ultimately show** *?thesis*
**unfolding** *msb-k-def msb-p-def t-def*
**by** $(smt\ (verit,\ ccfv\text{-}SIG) * int\text{-}ops(6)\ le\text{-}nat\text{-}floor\ le\text{-}nat\text{-}iff\ nat\text{-}diff\text{-}distrib'\ nat\text{-}less\text{-}as\text{-}int\ nat\text{-}one\text{-}as\text{-}int\ of\text{-}int\text{-}1\text{-}less\text{-}iff\ of\text{-}nat\text{-}0\text{-}le\text{-}iff\ of\text{-}nat\text{-}less\text{-}of\text{-}int\text{-}iff\ t\text{-}def\ zero\text{-}le\text{-}floor\ zless\text{-}nat\text{-}eq\text{-}int\text{-}zless)$
**qed**

**lemma** *msb-p-valid-hnp*:
**assumes** *hnp-arith n α d p k*
**defines** *msb-k ≡ msb-p p k*
**shows** *hnp n α d p k msb-k*
**using** *hnp-arith.msb-p-dist-hnp*[*OF assms*(*1*)] *assms*(*1*)
**unfolding** *hnp-def msb-k-def hnp-arith-def hnp-axioms-def*
**by** *presburger*

**end**
**theory** *Ad-Codegen-Example*
**imports** *Hidden-Number-Problem*

**begin**

# 13 This theory demonstrates an example of the executable adversary.

*full-babai* does not need a global interpretation to be executed.

**value** *full-babai* [*vec-of-list* [*0*, *1*], *vec-of-list* [*2*, *3*]] (*vec-of-list* [*2.3*, *6.4*]) (*4*/*3*)

Let's define our $d$, $n$, $p$, $\alpha$, and $k$.

**abbreviation** *d ≡ 72*
**abbreviation** *n ≡ 1279*
**abbreviation** *p ≡* (*2*::*nat*)^*1279 − 1*
**abbreviation** *α ≡ p div 3*
**abbreviation** *k ≡ 47*

**value** *p*
**value** *α*

Since our adversary definition is inside a locale, we need a global interpretation.

**global-interpretation** *ad-interp*: *hnp-adversary d p*
**defines** *𝒜 = ad-interp.𝒜*
**and** *int-gen-basis = ad-interp.int-gen-basis*
**and** *int-to-nat-residue = ad-interp.int-to-nat-residue*
**and** *ts-from-pairs = ad-interp.ts-from-pairs*
**and** *scaled-uvec-from-pairs = ad-interp.scaled-uvec-from-pairs*
**and** *𝒜-vec = ad-interp.𝒜-vec*
**by** *unfold-locales*

For this example, we use our executable msb function.

**abbreviation** *𝒪 ≡ λt. msb k n* ((*α* ∗ *t*) *mod p*)

**definition** *inc-amt* :: *nat* **where** *inc-amt = p div 5*

**fun** *gen-ts* :: *nat* ⇒ *nat* ⇒ *nat list* **where**
  *gen-ts 0 t* = []
| *gen-ts* (*Suc i*) *t* = *t* # (*gen-ts i* ((*t* + *inc-amt*) *mod p*))

**definition** *gen-pairs* :: (*nat* × *nat*) *list* **where**
  *gen-pairs* = *map* (λ*t*. (*t*, $\mathcal{O}$ *t*)) (*gen-ts d 1*)

The *gen-pairs* function generates the data that the adversary receives. We prove that the adversary is likely to be successful when the *ts* which define this data are uniformly distributed. Here, we use the *gen-ts* function to generate an explicit list of *ts*.

**value** *length gen-pairs*

**value** *gen-pairs*

**value** *ad-interp*.$\mathcal{A}$ *gen-pairs*

**value** $\alpha$

**end**

# References


[1] L. Babai. On Lovász' lattice reduction and the nearest lattice point problem. *Comb.*, 6(1):1–13, 1986.

[2] D. Boneh and R. Venkatesan. Hardness of computing the most significant bits of secret keys in Diffie-Hellman and related schemes. In N. Koblitz, editor, *CRYPTO*, volume 1109 of *LNCS*, pages 129–142. Springer, 1996.

[3] R. Bottesch, M. W. Haslbeck, and R. Thiemann. A verified efficient implementation of the LLL basis reduction algorithm. In G. Barthe, G. Sutcliffe, and M. Veanes, editors, *LPAR*, volume 57 of *EPiC Series in Computing*, pages 164–180. EasyChair, 2018.

[4] J. Divasón, S. J. C. Joosten, R. Thiemann, and A. Yamada. A Verified Factorization Algorithm for Integer Polynomials with Polynomial Complexity. *Archive of Formal Proofs*, February 2018. https://isa-afp.org/entries/LLL_Factorization.html, Formal proof development.

[5] A. Lenstra, H. Lenstra, and L. László. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261, 12 1982.

[6] R. Thiemann, R. Bottesch, J. Divasón, M. W. Haslbeck, S. J. C. Joosten, and A. Yamada. Formalizing the LLL basis reduction algorithm and the LLL factorization algorithm in Isabelle/HOL. *J. Autom. Reason.*, 64(5):827–856, 2020.