

# A verified LLL algorithm\*

Ralph Bottesch      Jose Divasón      Maximilian Haslbeck  
Sebastiaan Joosten      René Thiemann      Akihisa Yamada

March 8, 2026

## Abstract

The Lenstra–Lenstra–Lovász basis reduction algorithm, also known as LLL algorithm, is an algorithm to find a basis with short, nearly orthogonal vectors of an integer lattice. Thereby, it can also be seen as an approximation to solve the shortest vector problem (SVP), which is an NP-hard problem, where the approximation quality solely depends on the dimension of the lattice, but not the lattice itself. The algorithm also possesses many applications in diverse fields of computer science, from cryptanalysis to number theory, but it is specially well-known since it was used to implement the first polynomial-time algorithm to factor polynomials. In this work we present the first mechanized soundness proof of the LLL algorithm to compute short vectors in lattices. The formalization follows a textbook by von zur Gathen and Gerhard [2].

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Missing lemmas</b>	<b>3</b>
<b>3</b>	<b>Auxiliary Lemmas and Definitions for Immutable Arrays</b>	<b>11</b>
<b>4</b>	<b>Norms</b>	<b>12</b>
4.1	L-∞ Norms . . . . .	13
4.2	Square Norms . . . . .	14
4.2.1	Square norms for vectors . . . . .	14
4.2.2	Square norm for polynomials . . . . .	15
4.3	Relating Norms . . . . .	15
<b>5</b>	<b>Optimized Code for Integer-Rational Operations</b>	<b>22</b>

---

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

<b>6</b>	<b>Representing Computation Costs as Pairs of Results and Costs</b>	<b>23</b>
<b>7</b>	<b>List representation</b>	<b>24</b>
<b>8</b>	<b>Gram-Schmidt</b>	<b>25</b>
8.1	Explicit Bounds for Size of Numbers that Occur During GSO Algorithm . . . . .	40
8.2	Gram-Schmidt Implementation for Integer Vectors . . . . .	44
8.3	Lemmas Summarizing All Bounds During GSO Computation	51
<b>9</b>	<b>The LLL Algorithm</b>	<b>52</b>
9.1	Core Definitions, Invariants, and Theorems for Basic Version	53
9.2	Basic LLL implementation based on previous results . . . . .	60
9.3	Integer LLL Implementation which Stores Multiples of the $\mu$ -Values . . . . .	62
9.3.1	Updates of the integer values for Swap, Add, etc. . . . .	63
9.3.2	Implementation of LLL via Integer Operations and Arrays . . . . .	64
9.4	Bound on Number of Arithmetic Operations for Integer Implementation . . . . .	72
9.5	Explicit Bounds for Size of Numbers that Occur During LLL Algorithm . . . . .	79
9.5.1	<i>LLL-bound-invariant</i> is maintained during execution of <i>reduce-basis</i> . . . . .	81
9.5.2	Bound extracted from <i>LLL-bound-invariant</i> . . . . .	83
<b>10</b>	<b>Certification of External LLL Invocations</b>	<b>86</b>
10.1	Checking Results of External LLL Solvers . . . . .	86
10.2	A Haskell Interface to the FPLLL-Solver . . . . .	90

## 1 Introduction

The LLL basis reduction algorithm by Lenstra, Lenstra and Lovász [3] is a remarkable algorithm with numerous applications in diverse fields. For instance, it can be used for finding the minimal polynomial of an algebraic number given to a good enough approximation, for finding integer relations, for integer programming and even for breaking knapsack based cryptographic protocols. Its most famous application is a polynomial-time algorithm to factor integer polynomials. Moreover, the LLL algorithm is used as part of the best known polynomial factorization algorithm that is used in today's computer algebra systems.

In this work we implement it in Isabelle/HOL and fully formalize the correctness of the implementation. The algorithm is parametric by some

$\alpha > \frac{4}{3}$ , and given  $fs$  a list of  $m$ -linearly independent vectors  $fs_0, \dots, fs_{m-1} \in \mathbb{Z}^n$ , it computes a short vector whose norm is at most  $\alpha^{\frac{m-1}{2}}$  larger than the norm of any nonzero vector in the lattice generated by the vectors of the list  $fs$ . The soundness theorem follows.

**Theorem 1 (Soundness of LLL algorithm)**

```

lemma short_vector :
assumes  $\alpha \geq 4/3$ 
and lin_indpt_list (RAT  $fs$ )
and short_vector  $\alpha$   $fs = v$ 
and length  $fs = m$ 
and  $m \neq 0$ 
shows  $v \in \text{lattice\_of } fs - \{0_v\}$ 
and  $h \in \text{lattice\_of } fs - \{0_v\} \longrightarrow \|v\|^2 \leq \alpha^{m-1} \cdot \|h\|^2$ 

```

To this end, we have performed the following tasks:

- We firstly have to improve some AFP entries, as well as generalize several concepts from the standard library.
- We have to develop a library about norms of vectors and their properties.
- We formalize the Gram–Schmidt orthogonalization procedure, which is a crucial sub-routine of the LLL algorithm. Indeed, we already formalized this procedure in Isabelle as a function *gram\_schmidt* when proving the existence of Jordan normal forms [4]. Unfortunately, lemma *gram\_schmidt* does not suffice for verifying the LLL algorithm and we have had to extend such a formalization.
- We prove the termination of the algorithm and its soundness.
- We prove polynomial runtime complexity by showing that there is a polynomial bound on the required number of arithmetic operations. Moreover, we formally prove that the representation size of the numbers that occur during the execution stays polynomial.

To our knowledge, this is the first formalization of the LLL algorithm in any theorem prover.

## 2 Missing lemmas

This theory contains many results that are important but not specific for our development. They could be moved to the standard library and some other AFP entries.

**theory** *Missing-Lemmas*

**imports**

*Berlekamp-Zassenhaus.Sublist-Iteration*  
*Berlekamp-Zassenhaus.Square-Free-Int-To-Square-Free-GFp*  
*Algebraic-Numbers.Resultant*  
*Jordan-Normal-Form.Conjugate*  
*Jordan-Normal-Form.Missing-VectorSpace*  
*Jordan-Normal-Form.VS-Connect*  
*Berlekamp-Zassenhaus.Finite-Field-Factorization-Record-Based*  
*Berlekamp-Zassenhaus.Berlekamp-Hensel*

**begin**

**hide-const**(**open**) *module.smult up-ring.monom up-ring.coeff*

**lemma** *log-prod*: **assumes**  $0 < a$   $a \neq 1$   $\wedge x. x \in X \implies 0 < f x$   
**shows**  $\log a$  (prod f X) = sum (log a o f) X  
*<proof>*

**subclass** (**in** *ordered-idom*) *zero-less-one* *<proof>*  
**hide-fact** *Missing-Ring.zero-less-one*

**instance** *real* :: *ordered-semiring-strict* *<proof>*  
**instance** *real* :: *linordered-idom* *<proof>*

**lemma** *upt-minus-eq-append*:  $i \leq j \implies i \leq j - k \implies [i..<j] = [i..<j-k] @ [j-k..<j]$   
*<proof>*

**lemma** *list-trisect*:  $x < \text{length } lst \implies [0..<\text{length } lst] = [0..<x] @ x \# [Suc x..<\text{length } lst]$   
*<proof>*

**lemma** *id-imp-bij-betw*:  
**assumes**  $f: f : A \rightarrow A$   
**and**  $ff: \bigwedge a. a \in A \implies f (f a) = a$   
**shows** *bij-betw* f A A  
*<proof>*

**lemma** *range-subsetI*:  
**assumes**  $\bigwedge x. f x = g (h x)$  **shows**  $\text{range } f \subseteq \text{range } g$   
*<proof>*

**lemma** *aux-abs-int*: **fixes**  $c :: \text{int}$   
**assumes**  $c \neq 0$   
**shows**  $|x| \leq |x * c|$   
*<proof>*

**lemma** *mod-0-abs-less-imp-0*:  
**fixes**  $a::int$   
**assumes**  $a1: [a = 0] \text{ (mod } m)$   
**and**  $a2: abs(a) < m$   
**shows**  $a = 0$   
 $\langle proof \rangle$

**lemma** *sum-list-zero*:  
**assumes**  $set\ xs \subseteq \{0\}$  **shows**  $sum\text{-list}\ xs = 0$   
 $\langle proof \rangle$

**lemma** *max-idem [simp]*:  $max\ a\ a = a$   
 $\langle proof \rangle$

**lemma** *hom-max*:  
**assumes**  $a \leq b \longleftrightarrow f\ a \leq f\ b$   
**shows**  $f\ (max\ a\ b) = max\ (f\ a)\ (f\ b)$   $\langle proof \rangle$

**lemma** *le-max-self*:  
**fixes**  $a\ b :: 'a :: preorder$   
**assumes**  $a \leq b \vee b \leq a$  **shows**  $a \leq max\ a\ b$  **and**  $b \leq max\ a\ b$   
 $\langle proof \rangle$

**lemma** *le-max*:  
**fixes**  $a\ b :: 'a :: preorder$   
**assumes**  $c \leq a \vee c \leq b$  **and**  $a \leq b \vee b \leq a$  **shows**  $c \leq max\ a\ b$   
 $\langle proof \rangle$

**fun** *max-list* **where**  
 $max\text{-list}\ [] = (THE\ x.\ False)$   
 $| max\text{-list}\ [x] = x$   
 $| max\text{-list}\ (x \# y \# xs) = max\ x\ (max\text{-list}\ (y \# xs))$

**declare** *max-list.simps(1)* [simp del]  
**declare** *max-list.simps(2-3)*[code]

**lemma** *max-list-Cons*:  $max\text{-list}\ (x\#\ xs) = (if\ xs = []\ then\ x\ else\ max\ x\ (max\text{-list}\ xs))$   
 $\langle proof \rangle$

**lemma** *max-list-mem*:  $xs \neq [] \implies max\text{-list}\ xs \in set\ xs$   
 $\langle proof \rangle$

**lemma** *mem-set-imp-le-max-list*:  
**fixes**  $xs :: 'a :: preorder\ list$   
**assumes**  $\bigwedge a\ b. a \in set\ xs \implies b \in set\ xs \implies a \leq b \vee b \leq a$

**and**  $a \in \text{set } xs$   
**shows**  $a \leq \text{max-list } xs$   
 <proof>

**lemma** *le-max-list*:

**fixes**  $xs :: 'a :: \text{preorder list}$   
**assumes**  $\text{ord}: \bigwedge a b. a \in \text{set } xs \implies b \in \text{set } xs \implies a \leq b \vee b \leq a$   
**and**  $ab: a \leq b$   
**and**  $b: b \in \text{set } xs$   
**shows**  $a \leq \text{max-list } xs$   
 <proof>

**lemma** *max-list-le*:

**fixes**  $xs :: 'a :: \text{preorder list}$   
**assumes**  $a: \bigwedge x. x \in \text{set } xs \implies x \leq a$   
**and**  $xs: xs \neq []$   
**shows**  $\text{max-list } xs \leq a$   
 <proof>

**lemma** *max-list-as-Greatest*:

**assumes**  $\bigwedge x y. x \in \text{set } xs \implies y \in \text{set } xs \implies x \leq y \vee y \leq x$   
**shows**  $\text{max-list } xs = (\text{GREATEST } a. a \in \text{set } xs)$   
 <proof>

**lemma** *hom-max-list-commute*:

**assumes**  $xs \neq []$   
**and**  $\bigwedge x y. x \in \text{set } xs \implies y \in \text{set } xs \implies h (\text{max } x y) = \text{max } (h x) (h y)$   
**shows**  $h (\text{max-list } xs) = \text{max-list } (\text{map } h xs)$   
 <proof>

**primrec** *rev-upt* ::  $\text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat list} \Rightarrow (\langle 1[->..] \rangle)$  **where**

*rev-upt-0*:  $[0>..j] = [] \mid$

*rev-upt-Suc*:  $[(\text{Suc } i)>..j] = (\text{if } i \geq j \text{ then } i \# [i>..j] \text{ else } [])$

**lemma** *rev-upt-rec*:  $[i>..j] = (\text{if } i > j \text{ then } [i>..\text{Suc } j] @ [j] \text{ else } [])$

<proof>

**definition** *rev-upt-aux* ::  $\text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat list} \Rightarrow \text{nat list}$  **where**

*rev-upt-aux*  $i j js = [i>..j] @ js$

**lemma** *upt-aux-rec* [code]:

*rev-upt-aux*  $i j js = (\text{if } j \geq i \text{ then } js \text{ else } \text{rev-upt-aux } i (\text{Suc } j) (j\#js))$

<proof>

**lemma** *rev-upt-code*[code]:  $[i>..j] = \text{rev-upt-aux } i j []$

<proof>

**lemma** *upt-rev-upt*:  
 $rev [j>..i] = [i..<j]$   
 ⟨proof⟩

**lemma** *rev-upt-upt*:  
 $rev [i..<j] = [j>..i]$   
 ⟨proof⟩

**lemma** *length-rev-upt [simp]*:  $length [i>..j] = i - j$   
 ⟨proof⟩

**lemma** *nth-rev-upt [simp]*:  $j + k < i \implies [i>..j] ! k = i - 1 - k$   
 ⟨proof⟩

**lemma** *nth-map-rev-upt*:  
 assumes  $i: i < m - n$   
 shows  $(map f [m>..n]) ! i = f (m - 1 - i)$   
 ⟨proof⟩

**lemma** *coeff-mult-monom*:  
 $coeff (p * monom a d) i = (if d \leq i then a * coeff p (i - d) else 0)$   
 ⟨proof⟩

**lemma** *vec-of-poly-0 [simp]*:  $vec-of-poly 0 = 0_v 1$  ⟨proof⟩

**lemma** *vec-index-vec-of-poly [simp]*:  $i \leq degree p \implies vec-of-poly p \$ i = coeff p (degree p - i)$   
 ⟨proof⟩

**lemma** *poly-of-vec-vec*:  $poly-of-vec (vec n f) = Poly (rev (map f [0..<n]))$   
 ⟨proof⟩

**lemma** *sum-list-map-dropWhile0*:  
 assumes  $f0: f 0 = 0$   
 shows  $sum-list (map f (dropWhile ((=) 0) xs)) = sum-list (map f xs)$   
 ⟨proof⟩

**lemma** *coeffs-poly-of-vec*:  
 $coeffs (poly-of-vec v) = rev (dropWhile ((=) 0) (list-of-vec v))$   
 ⟨proof⟩

**lemma** *poly-of-vec-vCons*:  
 $poly-of-vec (vCons a v) = monom a (dim-vec v) + poly-of-vec v$  (is ?l = ?r)  
 ⟨proof⟩

**lemma** *poly-of-vec-as-Poly*:  $poly-of-vec v = Poly (rev (list-of-vec v))$

*<proof>*

**lemma** *poly-of-vec-add*:

**assumes** *dim-vec a = dim-vec b*

**shows** *poly-of-vec (a + b) = poly-of-vec a + poly-of-vec b*

*<proof>*

**lemma** (**in** *vec-module*) *poly-of-vec-finsum*:

**assumes** *f ∈ X → carrier-vec n*

**shows** *poly-of-vec (finsum V f X) = (∑ i∈X. poly-of-vec (f i))*

*<proof>*

**definition** *vec-of-poly-n p n =*

*vec n (λi. if i < n - degree p - 1 then 0 else coeff p (n - i - 1))*

**lemma** *vec-of-poly-as*: *vec-of-poly-n p (Suc (degree p)) = vec-of-poly p*

*<proof>*

**lemma** *vec-of-poly-n-0 [simp]*: *vec-of-poly-n p 0 = vNil*

*<proof>*

**lemma** *vec-dim-vec-of-poly-n [simp]*:

*dim-vec (vec-of-poly-n p n) = n*

*vec-of-poly-n p n ∈ carrier-vec n*

*<proof>*

**lemma** *dim-vec-of-poly [simp]*: *dim-vec (vec-of-poly f) = degree f + 1*

*<proof>*

**lemma** *vec-index-of-poly-n*:

**assumes** *i < n*

**shows** *vec-of-poly-n p n \$ i =*

*(if i < n - Suc (degree p) then 0 else coeff p (n - i - 1))*

*<proof>*

**lemma** *vec-of-poly-n-pCons [simp]*:

**shows** *vec-of-poly-n (pCons a p) (Suc n) = vec-of-poly-n p n @<sub>v</sub> vec-of-list [a]*

(**is** *?l = ?r*)

*<proof>*

**lemma** *vec-of-poly-pCons*:

**shows** *vec-of-poly (pCons a p) =*

*(if p = 0 then vec-of-list [a] else vec-of-poly p @<sub>v</sub> vec-of-list [a])*

*<proof>*

**lemma** *list-of-vec-of-poly [simp]*:

*list-of-vec (vec-of-poly p) = (if p = 0 then [0] else rev (coeffs p))*

*<proof>*

**lemma** *poly-of-vec-of-poly-n*:

**assumes** *p*: *degree p < n*

**shows** *poly-of-vec (vec-of-poly-n p n) = p*

*<proof>*

**lemma** *vec-of-poly-n0[simp]*: *vec-of-poly-n 0 n = 0<sub>v</sub> n*

*<proof>*

**lemma** *vec-of-poly-n-add*: *vec-of-poly-n (a + b) n = vec-of-poly-n a n + vec-of-poly-n b n*

*<proof>*

**lemma** *vec-of-poly-n-poly-of-vec*:

**assumes** *n*: *dim-vec g = n*

**shows** *vec-of-poly-n (poly-of-vec g) n = g*

*<proof>*

**lemma** *poly-of-vec-scalar-mult*:

**assumes** *degree b < n*

**shows** *poly-of-vec (a ·<sub>v</sub> (vec-of-poly-n b n)) = smult a b*

*<proof>*

**definition** *vec-of-poly-rev-shifted where*

*vec-of-poly-rev-shifted p n s j ≡*

*vec n (λi. if i ≤ j ∧ j ≤ s + i then coeff p (s + i - j) else 0)*

**lemma** *vec-of-poly-rev-shifted-dim[simp]*: *dim-vec (vec-of-poly-rev-shifted p n s j) = n*

*<proof>*

**lemma** *col-sylvester-sub*:

**assumes** *j*: *j < m + n*

**shows** *col (sylvester-mat-sub m n p q) j =*

*vec-of-poly-rev-shifted p n m j @<sub>v</sub> vec-of-poly-rev-shifted q m n j (is ?l = ?r)*

*<proof>*

**lemma** *vec-of-poly-rev-shifted-scalar-prod*:

**fixes** *p v*

**defines** *q ≡ poly-of-vec v*

**assumes** *m*: *degree p ≤ m* **and** *n*: *dim-vec v = n*

**assumes** *j*: *j < m + n*

**shows** *vec-of-poly-rev-shifted p n m (n + m - Suc j) · v = coeff (p \* q) j (is ?l = ?r)*

*<proof>*

**lemma** *sylvester-sub-poly*:

**fixes**  $p\ q :: 'a :: \text{comm-semiring-0 poly}$   
**assumes**  $m: \text{degree } p \leq m$   
**assumes**  $n: \text{degree } q \leq n$   
**assumes**  $v: v \in \text{carrier-vec } (m+n)$   
**shows**  $\text{poly-of-vec } ((\text{sylvester-mat-sub } m\ n\ p\ q)^T *_{\mathbf{v}} v) =$   
 $\text{poly-of-vec } (\text{vec-first } v\ n) * p + \text{poly-of-vec } (\text{vec-last } v\ m) * q$  (**is**  $?l = ?r$ )  
 $\langle \text{proof} \rangle$

**lemma** *normalize-field* [simp]:  $\text{normalize } (a :: 'a :: \{\text{field, semiring-gcd}\}) = (\text{if } a = 0 \text{ then } 0 \text{ else } 1)$   
 $\langle \text{proof} \rangle$

**lemma** *content-field* [simp]:  $\text{content } (p :: 'a :: \{\text{field, semiring-gcd}\} \text{ poly}) = (\text{if } p = 0 \text{ then } 0 \text{ else } 1)$   
 $\langle \text{proof} \rangle$

**lemma** *primitive-part-field* [simp]:  $\text{primitive-part } (p :: 'a :: \{\text{field, semiring-gcd}\} \text{ poly}) = p$   
 $\langle \text{proof} \rangle$

**lemma** *primitive-part-dvd*:  $\text{primitive-part } a \text{ dvd } a$   
 $\langle \text{proof} \rangle$

**lemma** *degree-abs* [simp]:  
 $\text{degree } |p| = \text{degree } p$   $\langle \text{proof} \rangle$

**lemma** *degree-gcd1*:  
**assumes**  $a\text{-not0}: a \neq 0$   
**shows**  $\text{degree } (\text{gcd } a\ b) \leq \text{degree } a$   
 $\langle \text{proof} \rangle$

**lemma** *primitive-part-neg* [simp]:  
**fixes**  $a :: 'a :: \{\text{factorial-ring-gcd, factorial-semiring-multiplicative}\} \text{ poly}$   
**shows**  $\text{primitive-part } (-a) = - \text{primitive-part } a$   
 $\langle \text{proof} \rangle$

**lemma** *content-uminus*[simp]:  
**fixes**  $f :: \text{int poly}$   
**shows**  $\text{content } (-f) = \text{content } f$   
 $\langle \text{proof} \rangle$

**lemma** *pseudo-mod-monic*:  
**fixes**  $f\ g :: 'a :: \{\text{comm-ring-1, semiring-1-no-zero-divisors}\} \text{ poly}$   
**defines**  $r \equiv \text{pseudo-mod } f\ g$   
**assumes**  $\text{monic-g}: \text{monic } g$

**shows**  $\exists q. f = g * q + r \wedge r = 0 \vee \text{degree } r < \text{degree } g$   
 ⟨proof⟩

**lemma** *monic-imp-div-mod-int-poly-degree*:  
**fixes**  $p :: 'a::\{\text{comm-ring-1}, \text{semiring-1-no-zero-divisors}\}$  *poly*  
**assumes**  $m$ : *monic*  $u$   
**shows**  $\exists q r. p = q * u + r \wedge (r = 0 \vee \text{degree } r < \text{degree } u)$   
 ⟨proof⟩

**corollary** *monic-imp-div-mod-int-poly-degree2*:  
**fixes**  $p :: 'a::\{\text{comm-ring-1}, \text{semiring-1-no-zero-divisors}\}$  *poly*  
**assumes**  $m$ : *monic*  $u$  **and** *deg-u*:  $\text{degree } u > 0$   
**shows**  $\exists q r. p = q * u + r \wedge (\text{degree } r < \text{degree } u)$   
 ⟨proof⟩

**lemma** (*in zero-hom*) *hom-upper-triangular*:  
 $A \in \text{carrier-mat } n \ n \implies \text{upper-triangular } A \implies \text{upper-triangular } (\text{map-mat } \text{hom } A)$   
 ⟨proof⟩

end

### 3 Auxiliary Lemmas and Definitions for Immutable Arrays

We define some definitions on immutable arrays, and modify the simplification rules so that IArrays will mainly operate pointwise, and not as lists. To be more precise, IArray.of-fun will become the main constructor.

**theory** *More-IArray*  
**imports** *HOL-Library.IArray*  
**begin**

**definition** *iarray-update* ::  $'a \text{ iarray} \Rightarrow \text{nat} \Rightarrow 'a \Rightarrow 'a \text{ iarray}$  **where**  
 $\text{iarray-update } a \ i \ x = \text{IArray.of-fun } (\lambda j. \text{if } j = i \text{ then } x \text{ else } a \ !! \ j) \ (\text{IArray.length } a)$

**lemma** *iarray-cong*:  $n = m \implies (\bigwedge i. i < m \implies f \ i = g \ i) \implies \text{IArray.of-fun } f \ n = \text{IArray.of-fun } g \ m$   
 ⟨proof⟩

**lemma** *iarray-cong'*:  $(\bigwedge i. i < n \implies f \ i = g \ i) \implies \text{IArray.of-fun } f \ n = \text{IArray.of-fun } g \ n$   
 ⟨proof⟩

```

lemma iarray-update-length[simp]: IArray.length (iarray-update a i x) = IArray.length
a
  <proof>

lemma iarray-length-of-fun[simp]: IArray.length (IArray.of-fun f n) = n <proof>

lemma iarray-update-of-fun[simp]: iarray-update (IArray.of-fun f n) i x = IAr-
ray.of-fun (f (i := x)) n
  <proof>

fun iarray-append where iarray-append (IArray xs) x = IArray (xs @ [x])

lemma iarray-append-code[code]: iarray-append xs x = IArray (IArray.list-of xs @
[x])
  <proof>

lemma iarray-append-of-fun[simp]: iarray-append (IArray.of-fun f n) x = IAr-
ray.of-fun (f (n := x)) (Suc n)
  <proof>

declare iarray-append.simps[simp del]

lemma iarray-of-fun-sub[simp]: i < n  $\implies$  IArray.of-fun f n !! i = f i
  <proof>

lemma IArray-of-fun-conv: IArray xs = IArray.of-fun ( $\lambda i. xs ! i$ ) (length xs)
  <proof>

declare IArray.of-fun-def[simp del]
declare IArray.sub-def[simp del]

lemmas iarray-simps = iarray-update-of-fun iarray-append-of-fun IArray-of-fun-conv
iarray-of-fun-sub

end

```

## 4 Norms

In this theory we provide the basic definitions and properties of several norms of vectors and polynomials.

```

theory Norms
  imports HOL-Computational-Algebra.Polynomial
  Jordan-Normal-Form.Conjugate
  Algebraic-Numbers.Resultant
  Missing-Lemmas
begin

```

## 4.1 L- $\infty$ Norms

**consts** *linf-norm* :: 'a  $\Rightarrow$  'b ( $\langle \|(-)\|_\infty \rangle$ )

**definition** *linf-norm-vec* **where** *linf-norm-vec*  $v \equiv \text{max-list } (\text{map } \text{abs } (\text{list-of-vec } v)) @ [0]$

**ad hoc-overloading** *linf-norm*  $\equiv$  *linf-norm-vec*

**definition** *linf-norm-poly* **where** *linf-norm-poly*  $f \equiv \text{max-list } (\text{map } \text{abs } (\text{coeffs } f)) @ [0]$

**ad hoc-overloading** *linf-norm*  $\equiv$  *linf-norm-poly*

**lemma** *linf-norm-vec*:  $\| \text{vec } n \ f \|_\infty = \text{max-list } (\text{map } (\text{abs } \circ f) [0..<n]) @ [0]$   
 $\langle \text{proof} \rangle$

**lemma** *linf-norm-vec-vCons[simp]*:  $\| \text{vCons } a \ v \|_\infty = \text{max } |a| \ \|v\|_\infty$   
 $\langle \text{proof} \rangle$

**lemma** *linf-norm-vec-0 [simp]*:  $\| \text{vec } 0 \ f \|_\infty = 0$   $\langle \text{proof} \rangle$

**lemma** *linf-norm-zero-vec [simp]*:  $\| 0_v \ n :: 'a :: \text{ordered-ab-group-add-abs } \text{vec} \|_\infty = 0$   
 $\langle \text{proof} \rangle$

**lemma** *linf-norm-vec-ge-0 [intro!]*:  
**fixes**  $v :: 'a :: \text{ordered-ab-group-add-abs } \text{vec}$   
**shows**  $\|v\|_\infty \geq 0$   
 $\langle \text{proof} \rangle$

**lemma** *linf-norm-vec-eq-0 [simp]*:  
**fixes**  $v :: 'a :: \text{ordered-ab-group-add-abs } \text{vec}$   
**assumes**  $v \in \text{carrier-vec } n$   
**shows**  $\|v\|_\infty = 0 \longleftrightarrow v = 0_v \ n$   
 $\langle \text{proof} \rangle$

**lemma** *linf-norm-vec-greater-0 [simp]*:  
**fixes**  $v :: 'a :: \text{ordered-ab-group-add-abs } \text{vec}$   
**assumes**  $v \in \text{carrier-vec } n$   
**shows**  $\|v\|_\infty > 0 \longleftrightarrow v \neq 0_v \ n$   
 $\langle \text{proof} \rangle$

**lemma** *linf-norm-poly-0 [simp]*:  $\| 0 :: - \ \text{poly} \|_\infty = 0$   
 $\langle \text{proof} \rangle$

**lemma** *linf-norm-pCons [simp]*:  
**fixes**  $p :: 'a :: \text{ordered-ab-group-add-abs } \text{poly}$   
**shows**  $\| \text{pCons } a \ p \|_\infty = \text{max } |a| \ \|p\|_\infty$   
 $\langle \text{proof} \rangle$

**lemma** *linf-norm-poly-ge-0 [intro!]*:

**fixes**  $f :: 'a :: \text{ordered-ab-group-add-abs poly}$   
**shows**  $\|f\|_\infty \geq 0$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{linf-norm-poly-eq-0}$  [simp]:  
**fixes**  $f :: 'a :: \text{ordered-ab-group-add-abs poly}$   
**shows**  $\|f\|_\infty = 0 \longleftrightarrow f = 0$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{linf-norm-poly-greater-0}$  [simp]:  
**fixes**  $f :: 'a :: \text{ordered-ab-group-add-abs poly}$   
**shows**  $\|f\|_\infty > 0 \longleftrightarrow f \neq 0$   
 $\langle \text{proof} \rangle$

## 4.2 Square Norms

**consts**  $\text{sq-norm} :: 'a \Rightarrow 'b (\langle \|(-)\|^2 \rangle)$

**abbreviation**  $\text{sq-norm-conjugate } x \equiv x * \text{conjugate } x$   
**adhoc-overloading**  $\text{sq-norm} \equiv \text{sq-norm-conjugate}$

### 4.2.1 Square norms for vectors

We prefer `sum_list` over `sum` because it is not essentially dependent on commutativity, and easier for proving.

**definition**  $\text{sq-norm-vec } v \equiv \sum x \leftarrow \text{list-of-vec } v. \|x\|^2$   
**adhoc-overloading**  $\text{sq-norm} \equiv \text{sq-norm-vec}$

**lemma**  $\text{sq-norm-vec-vCons}$ [simp]:  $\|v\text{Cons } a \ v\|^2 = \|a\|^2 + \|v\|^2$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{sq-norm-vec-0}$ [simp]:  $\|\text{vec } 0 \ f\|^2 = 0$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{sq-norm-vec-as-cscalar-prod}$ :  
**fixes**  $v :: 'a :: \text{conjugatable-ring vec}$   
**shows**  $\|v\|^2 = v \cdot c \ v$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{sq-norm-zero-vec}$ [simp]:  $\|0_v \ n :: 'a :: \text{conjugatable-ring vec}\|^2 = 0$   
 $\langle \text{proof} \rangle$

**lemmas**  $\text{sq-norm-vec-ge-0}$  [intro!] =  $\text{conjugate-square-ge-0-vec}$ [folded  $\text{sq-norm-vec-as-cscalar-prod}$ ]

**lemmas**  $\text{sq-norm-vec-eq-0}$  [simp] =  $\text{conjugate-square-eq-0-vec}$ [folded  $\text{sq-norm-vec-as-cscalar-prod}$ ]

**lemmas**  $\text{sq-norm-vec-greater-0}$  [simp] =  $\text{conjugate-square-greater-0-vec}$ [folded  $\text{sq-norm-vec-as-cscalar-prod}$ ]

## 4.2.2 Square norm for polynomials

**definition** *sq-norm-poly* where *sq-norm-poly*  $p \equiv \sum a \leftarrow \text{coeffs } p. \|a\|^2$

**adhoc-overloading** *sq-norm*  $\equiv$  *sq-norm-poly*

**lemma** *sq-norm-poly-0* [*simp*]:  $\|0::\text{-poly}\|^2 = 0$   
 ⟨*proof*⟩

**lemma** *sq-norm-poly-pCons* [*simp*]:  
**fixes**  $a :: 'a :: \text{conjugatable-ring}$   
**shows**  $\|p\text{Cons } a \ p\|^2 = \|a\|^2 + \|p\|^2$   
 ⟨*proof*⟩

**lemma** *sq-norm-poly-ge-0* [*intro!*]:  
**fixes**  $p :: 'a :: \text{conjugatable-ordered-ring poly}$   
**shows**  $\|p\|^2 \geq 0$   
 ⟨*proof*⟩

**lemma** *sq-norm-poly-eq-0* [*simp*]:  
**fixes**  $p :: 'a :: \{\text{conjugatable-ordered-ring, ring-no-zero-divisors}\} \text{ poly}$   
**shows**  $\|p\|^2 = 0 \longleftrightarrow p = 0$   
 ⟨*proof*⟩

**lemma** *sq-norm-poly-pos* [*simp*]:  
**fixes**  $p :: 'a :: \{\text{conjugatable-ordered-ring, ring-no-zero-divisors}\} \text{ poly}$   
**shows**  $\|p\|^2 > 0 \longleftrightarrow p \neq 0$   
 ⟨*proof*⟩

**lemma** *sq-norm-vec-of-poly* [*simp*]:  
**fixes**  $p :: 'a :: \text{conjugatable-ring poly}$   
**shows**  $\|\text{vec-of-poly } p\|^2 = \|p\|^2$   
 ⟨*proof*⟩

**lemma** *sq-norm-poly-of-vec* [*simp*]:  
**fixes**  $v :: 'a :: \text{conjugatable-ring vec}$   
**shows**  $\|\text{poly-of-vec } v\|^2 = \|v\|^2$   
 ⟨*proof*⟩

## 4.3 Relating Norms

A class where ordering around 0 is linear.

**abbreviation** (in *ordered-semiring*) *is-real* where *is-real*  $a \equiv a < 0 \vee a = 0 \vee 0 < a$

**class** *semiring-real-line* = *ordered-semiring-strict* + *ordered-semiring-0* +  
**assumes** *add-pos-neg-is-real*:  $a > 0 \implies b < 0 \implies \text{is-real } (a + b)$   
**and** *mult-neg-neg*:  $a < 0 \implies b < 0 \implies 0 < a * b$   
**and** *pos-pos-linear*:  $0 < a \implies 0 < b \implies a < b \vee a = b \vee b < a$

**and** *neg-neg-linear*:  $a < 0 \implies b < 0 \implies a < b \vee a = b \vee b < a$   
**begin**

**lemma** *add-neg-pos-is-real*:  $a < 0 \implies b > 0 \implies \text{is-real } (a + b)$   
 ⟨*proof*⟩

**lemma** *nonneg-linorder-cases* [*consumes 2, case-names less eq greater*]:  
**assumes**  $0 \leq a$  **and**  $0 \leq b$   
**and**  $a < b \implies \text{thesis } a = b \implies \text{thesis } b < a \implies \text{thesis}$   
**shows** *thesis*  
 ⟨*proof*⟩

**lemma** *nonpos-linorder-cases* [*consumes 2, case-names less eq greater*]:  
**assumes**  $a \leq 0$   $b \leq 0$   
**and**  $a < b \implies \text{thesis } a = b \implies \text{thesis } b < a \implies \text{thesis}$   
**shows** *thesis*  
 ⟨*proof*⟩

**lemma** *real-linear*:  
**assumes** *is-real*  $a$  **and** *is-real*  $b$  **shows**  $a < b \vee a = b \vee b < a$   
 ⟨*proof*⟩

**lemma** *real-linorder-cases* [*consumes 2, case-names less eq greater*]:  
**assumes** *real*: *is-real*  $a$  *is-real*  $b$   
**and** *cases*:  $a < b \implies \text{thesis } a = b \implies \text{thesis } b < a \implies \text{thesis}$   
**shows** *thesis*  
 ⟨*proof*⟩

**lemma**  
**assumes** *a*: *is-real*  $a$  **and** *b*: *is-real*  $b$   
**shows** *real-add-le-cancel-left-pos*:  $c + a \leq c + b \iff a \leq b$   
**and** *real-add-less-cancel-left-pos*:  $c + a < c + b \iff a < b$   
**and** *real-add-le-cancel-right-pos*:  $a + c \leq b + c \iff a \leq b$   
**and** *real-add-less-cancel-right-pos*:  $a + c < b + c \iff a < b$   
 ⟨*proof*⟩

**lemma**  
**assumes** *a*: *is-real*  $a$  **and** *b*: *is-real*  $b$  **and** *c*:  $0 < c$   
**shows** *real-mult-le-cancel-left-pos*:  $c * a \leq c * b \iff a \leq b$   
**and** *real-mult-less-cancel-left-pos*:  $c * a < c * b \iff a < b$   
**and** *real-mult-le-cancel-right-pos*:  $a * c \leq b * c \iff a \leq b$   
**and** *real-mult-less-cancel-right-pos*:  $a * c < b * c \iff a < b$   
 ⟨*proof*⟩

**lemma**  
**assumes** *a*: *is-real*  $a$  **and** *b*: *is-real*  $b$   
**shows** *not-le-real*:  $\neg a \geq b \iff a < b$   
**and** *not-less-real*:  $\neg a > b \iff a \leq b$   
 ⟨*proof*⟩

**lemma** *real-mult-eq-0-iff*:  
**assumes** *a: is-real a and b: is-real b*  
**shows**  $a * b = 0 \longleftrightarrow a = 0 \vee b = 0$   
*<proof>*

**end**

**lemma** *real-pos-mult-max*:  
**fixes** *a b c :: 'a :: semiring-real-line*  
**assumes** *c: c > 0 and a: is-real a and b: is-real b*  
**shows**  $c * \max a b = \max (c * a) (c * b)$   
*<proof>*

**class** *ring-abs-real-line* = *ordered-ring-abs* + *semiring-real-line*

**class** *semiring-1-real-line* = *semiring-real-line* + *monoid-mult* + *zero-less-one*  
**begin**

**subclass** *ordered-semiring-1* *<proof>*

**lemma** *power-both-mono*:  $1 \leq a \implies m \leq n \implies a \leq b \implies a^m \leq b^n$   
*<proof>*

**lemma** *power-pos*:  
**assumes** *a0: 0 < a* **shows**  $0 < a^n$   
*<proof>*

**lemma** *power-neg*:  
**assumes** *a0: a < 0* **shows**  $\text{odd } n \implies a^n < 0$  **and**  $\text{even } n \implies a^n > 0$   
*<proof>*

**lemma** *power-ge-0-iff*:  
**assumes** *a: is-real a*  
**shows**  $0 \leq a^n \longleftrightarrow 0 \leq a \vee \text{even } n$   
*<proof>*

**lemma** *nonneg-power-less*:  
**assumes**  $0 \leq a$  **and**  $0 \leq b$  **shows**  $a^n < b^n \longleftrightarrow n > 0 \wedge a < b$   
*<proof>*

**lemma** *power-strict-mono*:  
**shows**  $a < b \implies 0 \leq a \implies 0 < n \implies a^n < b^n$   
*<proof>*

**lemma** *nonneg-power-le*:  
**assumes**  $0 \leq a$  **and**  $0 \leq b$  **shows**  $a^n \leq b^n \longleftrightarrow n = 0 \vee a \leq b$   
*<proof>*

```

end

subclass (in linordered-idom) semiring-1-real-line
  ⟨proof⟩

class ring-1-abs-real-line = ring-abs-real-line + semiring-1-real-line
begin

subclass ring-1 ⟨proof⟩

lemma abs-cases:
  assumes  $a = 0 \implies thesis$  and  $|a| > 0 \implies thesis$  shows  $thesis$ 
  ⟨proof⟩

lemma abs-linorder-cases[case-names less eq greater]:
  assumes  $|a| < |b| \implies thesis$  and  $|a| = |b| \implies thesis$  and  $|b| < |a| \implies thesis$ 
  shows  $thesis$ 
  ⟨proof⟩

lemma [simp]:
  shows not-le-abs-abs:  $\neg |a| \geq |b| \longleftrightarrow |a| < |b|$ 
    and not-less-abs-abs:  $\neg |a| > |b| \longleftrightarrow |a| \leq |b|$ 
  ⟨proof⟩

lemma abs-power-less [simp]:  $|a|^{\hat{n}} < |b|^{\hat{n}} \longleftrightarrow n > 0 \wedge |a| < |b|$ 
  ⟨proof⟩

lemma abs-power-le [simp]:  $|a|^{\hat{n}} \leq |b|^{\hat{n}} \longleftrightarrow n = 0 \vee |a| \leq |b|$ 
  ⟨proof⟩

lemma abs-power-pos [simp]:  $|a|^{\hat{n}} > 0 \longleftrightarrow a \neq 0 \vee n = 0$ 
  ⟨proof⟩

lemma abs-power-nonneg [intro!]:  $|a|^{\hat{n}} \geq 0$  ⟨proof⟩

lemma abs-power-eq-0 [simp]:  $|a|^{\hat{n}} = 0 \longleftrightarrow a = 0 \wedge n \neq 0$ 
  ⟨proof⟩

end

instance nat :: semiring-1-real-line ⟨proof⟩
instance int :: ring-1-abs-real-line ⟨proof⟩

lemma vec-index-vec-of-list [simp]:  $vec\text{-of-list } xs \$ i = xs ! i$ 
  ⟨proof⟩

lemma vec-of-list-append:  $vec\text{-of-list } (xs @ ys) = vec\text{-of-list } xs @_v vec\text{-of-list } ys$ 
  ⟨proof⟩

```

**lemma** *linf-norm-vec-of-list*:

$\|vec\text{-of-list } xs\|_\infty = \text{max-list } (\text{map } \text{abs } xs \text{ @ } [0])$

*<proof>*

**lemma** *linf-norm-vec-as-Greatest*:

**fixes**  $v :: 'a :: \text{ring-1-abs-real-line } \text{vec}$

**shows**  $\|v\|_\infty = (\text{GREATEST } a. a \in \text{abs ' set } (\text{list-of-vec } v) \cup \{0\})$

*<proof>*

**lemma** *vec-of-poly-pCons*:

**assumes**  $f \neq 0$

**shows**  $\text{vec-of-poly } (\text{pCons } a \ f) = \text{vec-of-poly } f \text{ @}_v \text{vec-of-list } [a]$

*<proof>*

**lemma** *vec-of-poly-as-vec-of-list*:

**assumes**  $f \neq 0$

**shows**  $\text{vec-of-poly } f = \text{vec-of-list } (\text{rev } (\text{coeffs } f))$

*<proof>*

**lemma** *linf-norm-vec-of-poly [simp]*:

**fixes**  $f :: 'a :: \text{ring-1-abs-real-line } \text{poly}$

**shows**  $\|\text{vec-of-poly } f\|_\infty = \|f\|_\infty$

*<proof>*

**lemma** *linf-norm-poly-as-Greatest*:

**fixes**  $f :: 'a :: \text{ring-1-abs-real-line } \text{poly}$

**shows**  $\|f\|_\infty = (\text{GREATEST } a. a \in \text{abs ' set } (\text{coeffs } f) \cup \{0\})$

*<proof>*

**lemma** *vec-index-le-linf-norm*:

**fixes**  $v :: 'a :: \text{ring-1-abs-real-line } \text{vec}$

**assumes**  $i < \text{dim-vec } v$

**shows**  $|v\$i| \leq \|v\|_\infty$

*<proof>*

**lemma** *coeff-le-linf-norm*:

**fixes**  $f :: 'a :: \text{ring-1-abs-real-line } \text{poly}$

**shows**  $|\text{coeff } f \ i| \leq \|f\|_\infty$

*<proof>*

**class** *conjugatable-ring-1-abs-real-line* = *conjugatable-ring* + *ring-1-abs-real-line* + *power* +

**assumes** *sq-norm-as-sq-abs [simp]*:  $\|a\|^2 = |a|^2$

**begin**

**subclass** *conjugatable-ordered-ring* *<proof>*

**end**

**instance** *int* :: *conjugatable-ring-1-abs-real-line*

*<proof>*

**instance** *rat* :: *conjugatable-ring-1-abs-real-line*  
⟨*proof*⟩

**instance** *real* :: *conjugatable-ring-1-abs-real-line*  
⟨*proof*⟩

**instance** *complex* :: *semiring-1-real-line*  
⟨*proof*⟩

Due to the assumption  $?a \leq |?a|$  from *Groups.thy*, *complex* cannot be *ring-1-abs-real-line*!

**instance** *complex* :: *ordered-ab-group-add-abs* ⟨*proof*⟩

**lemma** *sq-norm-as-sq-abs* [*simp*]: (*sq-norm* :: 'a :: *conjugatable-ring-1-abs-real-line*  
 $\Rightarrow$  'a) = *power2*  $\circ$  *abs*  
⟨*proof*⟩

**lemma** *sq-norm-vec-le-linf-norm*:  
**fixes** *v* :: 'a :: {*conjugatable-ring-1-abs-real-line*} *vec*  
**assumes** *v*  $\in$  *carrier-vec n*  
**shows**  $\|v\|^2 \leq \text{of-nat } n * \|v\|_\infty^2$   
⟨*proof*⟩

**lemma** *sq-norm-poly-le-linf-norm*:  
**fixes** *p* :: 'a :: {*conjugatable-ring-1-abs-real-line*} *poly*  
**shows**  $\|p\|^2 \leq \text{of-nat } (\text{degree } p + 1) * \|p\|_\infty^2$   
⟨*proof*⟩

**lemma** *coeff-le-sq-norm*:  
**fixes** *f* :: 'a :: {*conjugatable-ring-1-abs-real-line*} *poly*  
**shows**  $|\text{coeff } f \ i|^2 \leq \|f\|^2$   
⟨*proof*⟩

**lemma** *max-norm-witness*:  
**fixes** *f* :: 'a :: *ordered-ring-abs poly*  
**shows**  $\exists i. \|f\|_\infty = |\text{coeff } f \ i|$   
⟨*proof*⟩

**lemma** *max-norm-le-sq-norm*:  
**fixes** *f* :: 'a :: *conjugatable-ring-1-abs-real-line poly*  
**shows**  $\|f\|_\infty^2 \leq \|f\|^2$   
⟨*proof*⟩

**lemma** (in *conjugatable-ring*) *conjugate-minus*: *conjugate* (*x* - *y*) = *conjugate x*  
- *conjugate y*  
⟨*proof*⟩

**lemma** *conjugate-1* [simp]:  $(\text{conjugate } 1 :: 'a :: \{\text{conjugatable-ring, ring-1}\}) = 1$   
 ⟨proof⟩

**lemma** *conjugate-of-int* [simp]:  
 $(\text{conjugate } (\text{of-int } x) :: 'a :: \{\text{conjugatable-ring, ring-1}\}) = \text{of-int } x$   
 ⟨proof⟩

**lemma** *sq-norm-of-int*:  $\|\text{map-vec } \text{of-int } v :: 'a :: \{\text{conjugatable-ring, ring-1}\} \text{ vec}\|^2$   
 $= \text{of-int } \|v\|^2$   
 ⟨proof⟩

**definition** *norm1*  $p = \text{sum-list } (\text{map } \text{abs } (\text{coeffs } p))$

**lemma** *norm1-ge-0*:  $\text{norm1 } (f :: 'a :: \{\text{abs, ordered-semiring-0, ordered-ab-group-add-abs}\} \text{ poly})$   
 $\geq 0$   
 ⟨proof⟩

**lemma** *norm2-norm1-main-equality*: **fixes**  $f :: \text{nat} \Rightarrow 'a :: \text{linordered-idom}$   
**shows**  $(\sum i = 0..<n. |f\ i|^2) = (\sum i = 0..<n. f\ i * f\ i)$   
 $+ (\sum i = 0..<n. \sum j = 0..<n. \text{if } i = j \text{ then } 0 \text{ else } |f\ i| * |f\ j|)$   
 ⟨proof⟩

**lemma** *norm2-norm1-main-inequality*: **fixes**  $f :: \text{nat} \Rightarrow 'a :: \text{linordered-idom}$   
**shows**  $(\sum i = 0..<n. f\ i * f\ i) \leq (\sum i = 0..<n. |f\ i|^2)$   
 ⟨proof⟩

**lemma** *norm2-le-norm1-int*:  $\|f :: \text{int poly}\|^2 \leq (\text{norm1 } f)^{\wedge} 2$   
 ⟨proof⟩

**lemma** *sq-norm-smult-vec*:  $\text{sq-norm } ((c :: 'a :: \{\text{conjugatable-ring, comm-semiring-0}\})$   
 $\cdot_v v) = (c * \text{conjugate } c) * \text{sq-norm } v$   
 ⟨proof⟩

**lemma** *vec-le-sq-norm*:  
**fixes**  $v :: 'a :: \text{conjugatable-ring-1-abs-real-line } \text{vec}$   
**assumes**  $v \in \text{carrier-vec } n\ i < n$   
**shows**  $|v\ \$\ i|^2 \leq \|v\|^2$   
 ⟨proof⟩

**class** *trivial-conjugatable* =  
 conjugate +  
**assumes** *conjugate-id* [simp]:  $\text{conjugate } x = x$

**class** *trivial-conjugatable-ordered-field* =  
 conjugatable-ordered-field + trivial-conjugatable

**class** *trivial-conjugatable-linordered-field* =  
 trivial-conjugatable-ordered-field + linordered-field

```

begin
subclass conjugatable-ring-1-abs-real-line
  ⟨proof⟩
end

instance rat :: trivial-conjugatable-linordered-field
  ⟨proof⟩

instance real :: trivial-conjugatable-linordered-field
  ⟨proof⟩

lemma scalar-prod-ge-0: (x :: 'a :: linordered-idom vec) • x ≥ 0
  ⟨proof⟩

lemma cscalar-prod-is-scalar-prod[simp]: (x :: 'a :: trivial-conjugatable-ordered-field
vec) • c y = x • y
  ⟨proof⟩

lemma scalar-prod-Cauchy:
  fixes u v :: 'a :: {trivial-conjugatable-linordered-field} Matrix.vec
  assumes u ∈ carrier-vec n v ∈ carrier-vec n
  shows (u • v)2 ≤ ||u||2 * ||v||2
  ⟨proof⟩

end

```

## 5 Optimized Code for Integer-Rational Operations

```

theory Int-Rat-Operations
imports
  Sqrt-Babylonian.Sqrt-Babylonian-Auxiliary
  Norms
begin

definition int-times-rat :: int ⇒ rat ⇒ rat where int-times-rat i x = of-int i * x

declare int-times-rat-def[simp]

lemma int-times-rat-code[code abstract]: quotient-of (int-times-rat i x) =
  (case quotient-of x of (n,d) ⇒ Rat.normalize (i * n, d))
  ⟨proof⟩

definition square-rat :: rat ⇒ rat where [simp]: square-rat x = x * x

lemma quotient-of-square: assumes quotient-of x = (a,b)
  shows quotient-of (x * x) = (a * a, b * b)
  ⟨proof⟩

```

**lemma** *square-rat-code*[code abstract]: *quotient-of (square-rat x) = (case quotient-of x of (n,d) ⇒ (n \* n, d \* d))* ⟨proof⟩

**definition** *scalar-prod-int-rat* :: *int vec ⇒ rat vec ⇒ rat* (**infix** ⟨*·i*⟩ 70) **where**  
*x ·i y = (y · map-vec rat-of-int x)*

**lemma** *scalar-prod-int-rat-code*[code]: *v ·i w = (∑ i = 0..*dim-vec v*. int-times-rat (v \$ i) (w \$ i))*  
 ⟨proof⟩

**lemma** *scalar-prod-int-rat*[simp]: *dim-vec x = dim-vec y ⇒ x ·i y = map-vec of-int x · y*  
 ⟨proof⟩

**definition** *sq-norm-vec-rat* :: *rat vec ⇒ rat* **where** [simp]: *sq-norm-vec-rat x = sq-norm-vec x*

**lemma** *sq-norm-vec-rat-code*[code]: *sq-norm-vec-rat x = (∑ x ← list-of-vec x. square-rat x)*  
 ⟨proof⟩

end

## 6 Representing Computation Costs as Pairs of Results and Costs

**theory** *Cost*  
**imports** *Main*  
**begin**

**type-synonym** *'a cost = 'a × nat*

**definition** *cost* :: *'a cost ⇒ nat* **where** *cost = snd*  
**definition** *result* :: *'a cost ⇒ 'a* **where** *result = fst*

**lemma** *cost-simps*: *cost (a,c) = c result (a,c) = a*  
 ⟨proof⟩

**lemma** *result-costD*: **assumes** *result f-c = f*  
*cost f-c ≤ b*  
*f-c = (a,c)*  
**shows** *a = f c ≤ b* ⟨proof⟩

**lemma** *result-costD'*: **assumes** *result f-c = f ∧ cost f-c ≤ b*  
*f-c = (a,c)*  
**shows** *a = f c ≤ b* ⟨proof⟩

end

## 7 List representation

**theory** *List-Representation*  
  **imports** *Main*  
**begin**

**lemma** *rev-take-Suc*: **assumes**  $j: j < \text{length } xs$   
  **shows**  $\text{rev } (\text{take } (\text{Suc } j) \text{ } xs) = xs ! j \# \text{rev } (\text{take } j \text{ } xs)$   
   $\langle \text{proof} \rangle$

**type-synonym**  $'a \text{ list-repr} = 'a \text{ list} \times 'a \text{ list}$

**definition** *list-repr* ::  $\text{nat} \Rightarrow 'a \text{ list-repr} \Rightarrow 'a \text{ list} \Rightarrow \text{bool}$  **where**  
   $\text{list-repr } i \text{ } ba \text{ } xs = (i \leq \text{length } xs \wedge \text{fst } ba = \text{rev } (\text{take } i \text{ } xs) \wedge \text{snd } ba = \text{drop } i \text{ } xs)$

**definition** *of-list-repr* ::  $'a \text{ list-repr} \Rightarrow 'a \text{ list}$  **where**  
   $\text{of-list-repr } ba = (\text{rev } (\text{fst } ba) \text{ } @ \text{snd } ba)$

**lemma** *of-list-repr*:  $\text{list-repr } i \text{ } ba \text{ } xs \Longrightarrow \text{of-list-repr } ba = xs$   
   $\langle \text{proof} \rangle$

**definition** *get-nth-i* ::  $'a \text{ list-repr} \Rightarrow 'a$  **where**  
   $\text{get-nth-i } ba = \text{hd } (\text{snd } ba)$

**definition** *get-nth-im1* ::  $'a \text{ list-repr} \Rightarrow 'a$  **where**  
   $\text{get-nth-im1 } ba = \text{hd } (\text{fst } ba)$

**lemma** *get-nth-i*:  $\text{list-repr } i \text{ } ba \text{ } xs \Longrightarrow i < \text{length } xs \Longrightarrow \text{get-nth-i } ba = xs ! i$   
   $\langle \text{proof} \rangle$

**lemma** *get-nth-im1*:  $\text{list-repr } i \text{ } ba \text{ } xs \Longrightarrow i \neq 0 \Longrightarrow \text{get-nth-im1 } ba = xs ! (i - 1)$   
   $\langle \text{proof} \rangle$

**definition** *update-i* ::  $'a \text{ list-repr} \Rightarrow 'a \Rightarrow 'a \text{ list-repr}$  **where**  
   $\text{update-i } ba \text{ } x = (\text{fst } ba, x \# \text{tl } (\text{snd } ba))$

**lemma** *Cons-tl-drop-update*:  $i < \text{length } xs \Longrightarrow x \# \text{tl } (\text{drop } i \text{ } xs) = \text{drop } i \text{ } (xs[i := x])$   
   $\langle \text{proof} \rangle$

**lemma** *update-i*:  $\text{list-repr } i \text{ } ba \text{ } xs \Longrightarrow i < \text{length } xs \Longrightarrow \text{list-repr } i \text{ } (\text{update-i } ba \text{ } x) \text{ } (xs [i := x])$   
   $\langle \text{proof} \rangle$

**definition** *update-im1* ::  $'a \text{ list-repr} \Rightarrow 'a \Rightarrow 'a \text{ list-repr}$  **where**

$update-im1\ ba\ x = (x \# tl\ (fst\ ba),\ snd\ ba)$

**lemma** *update-im1*:  $list-repr\ i\ ba\ xs \implies i \neq 0 \implies list-repr\ i\ (update-im1\ ba\ x)$   
 $(xs\ [i - 1 := x])$   
*<proof>*

**lemma** *tl-drop-Suc*:  $tl\ (drop\ i\ xs) = drop\ (Suc\ i)\ xs$   
*<proof>*

**definition** *inc-i* ::  $'a\ list-repr \Rightarrow 'a\ list-repr$  **where**  
 $inc-i\ ba = (case\ ba\ of\ (b,a) \Rightarrow (hd\ a \# b,\ tl\ a))$

**lemma** *inc-i*:  $list-repr\ i\ ba\ xs \implies i < length\ xs \implies list-repr\ (Suc\ i)\ (inc-i\ ba)\ xs$   
*<proof>*

**definition** *dec-i* ::  $'a\ list-repr \Rightarrow 'a\ list-repr$  **where**  
 $dec-i\ ba = (case\ ba\ of\ (b,a) \Rightarrow (tl\ b,\ hd\ b \# a))$

**lemma** *dec-i*:  $list-repr\ i\ ba\ xs \implies i \neq 0 \implies list-repr\ (i - 1)\ (dec-i\ ba)\ xs$   
*<proof>*

**lemma** *dec-i-Suc*:  $list-repr\ (Suc\ i)\ ba\ xs \implies list-repr\ i\ (dec-i\ ba)\ xs$   
*<proof>*

**end**

## 8 Gram-Schmidt

**theory** *Gram-Schmidt-2*

**imports**

*Jordan-Normal-Form.Gram-Schmidt*

*Jordan-Normal-Form.Show-Matrix*

*Jordan-Normal-Form.Matrix-Impl*

*Norms*

*Int-Rat-Operations*

**begin**

**unbundle** *no m-inv-syntax*

**lemma** *rev-unsimp*:  $rev\ xs\ @\ (r \# rs) = rev\ (r\#\ xs)\ @\ rs$  *<proof>*

**lemma** *corthogonal-is-orthogonal[simp]*:

*corthogonal* (*xs* :: 'a :: trivial-conjugatable-ordered-field *vec list*) = *orthogonal xs*  
 ⟨*proof*⟩

**context** *cof-vec-space*  
**begin**

**definition** *lin-indpt-list* :: 'a *vec list* ⇒ *bool* **where**  
*lin-indpt-list fs* = (*set fs* ⊆ *carrier-vec n* ∧ *distinct fs* ∧ *lin-indpt (set fs)*)

**definition** *basis-list* :: 'a *vec list* ⇒ *bool* **where**  
*basis-list fs* = (*set fs* ⊆ *carrier-vec n* ∧ *length fs* = *n* ∧ *carrier-vec n* ⊆ *span (set fs)*)

**lemma** *upper-triangular-imp-lin-indpt-list*:  
**assumes** *A*: *A* ∈ *carrier-mat n n*  
**and** *tri*: *upper-triangular A*  
**and** *diag*:  $0 \notin \text{set } (\text{diag-mat } A)$   
**shows** *lin-indpt-list (rows A)*  
 ⟨*proof*⟩

**lemma** *basis-list-basis*: **assumes** *basis-list fs*  
**shows** *distinct fs lin-indpt (set fs) basis (set fs)*  
 ⟨*proof*⟩

**lemma** *basis-list-imp-lin-indpt-list*: **assumes** *basis-list fs* **shows** *lin-indpt-list fs*  
 ⟨*proof*⟩

**lemma** *basis-det-nonzero*:  
**assumes** *db*:*basis (set G)* **and** *len*:*length G* = *n*  
**shows** *det (mat-of-rows n G)* ≠ 0  
 ⟨*proof*⟩

**lemma** *lin-indpt-list-add-vec*: **assumes**  
*i*: *j* < *length us* *i* < *length us* *i* ≠ *j*  
**and** *indep*: *lin-indpt-list us*  
**shows** *lin-indpt-list (us [i := us ! i + c ·<sub>v</sub> us ! j]) (is lin-indpt-list ?V)*  
 ⟨*proof*⟩

**lemma** *scalar-prod-lincomb-orthogonal*: **assumes** *ortho*: *orthogonal gs* **and** *gs*: *set gs* ⊆ *carrier-vec n*  
**shows** *k* ≤ *length gs* ⇒ *sumlist (map (λ i. g i ·<sub>v</sub> gs ! i) [0 ..< k]) · sumlist (map (λ i. h i ·<sub>v</sub> gs ! i) [0 ..< k])*  
 = *sum-list (map (λ i. g i \* h i \* (gs ! i · gs ! i)) [0 ..< k])*  
 ⟨*proof*⟩  
**end**

**locale** *gram-schmidt* = *cof-vec-space n f-ty*

**for**  $n :: \text{nat}$  **and**  $f\text{-ty} :: 'a :: \{\text{trivial-conjugatable-linordered-field}\}$  *itself*  
**begin**

**definition** *Gramian-matrix* **where**

*Gramian-matrix*  $G\ k = (\text{let } M = \text{mat } k\ n\ (\lambda\ (i,j). (G\ !\ i)\ \$\ j)) \text{ in } M * M^T)$

**lemma** *Gramian-matrix-alt-def*:  $k \leq \text{length } G \implies$

*Gramian-matrix*  $G\ k = (\text{let } M = \text{mat-of-rows } n\ (\text{take } k\ G)) \text{ in } M * M^T)$   
 $\langle \text{proof} \rangle$

**definition** *Gramian-determinant* **where**

*Gramian-determinant*  $G\ k = \text{det } (\text{Gramian-matrix } G\ k)$

**lemma** *Gramian-determinant-0* [simp]: *Gramian-determinant*  $G\ 0 = 1$

$\langle \text{proof} \rangle$

**lemma** *orthogonal-imp-lin-indpt-list*:

**assumes** *ortho*: *orthogonal*  $gs$  **and**  $gs$ :  $\text{set } gs \subseteq \text{carrier-vec } n$   
**shows** *lin-indpt-list*  $gs$

$\langle \text{proof} \rangle$

**lemma** *orthocompl-span*:

**assumes**  $\bigwedge x. x \in S \implies v \cdot x = 0$   $S \subseteq \text{carrier-vec } n$  **and** [intro]:  $v \in \text{carrier-vec } n$

**and**  $y \in \text{span } S$   
**shows**  $v \cdot y = 0$

$\langle \text{proof} \rangle$

**lemma** *orthogonal-sumlist*:

**assumes** *ortho*:  $\bigwedge x. x \in \text{set } S \implies v \cdot x = 0$  **and**  $S$ :  $\text{set } S \subseteq \text{carrier-vec } n$  **and**  
 $v$ :  $v \in \text{carrier-vec } n$

**shows**  $v \cdot \text{sumlist } S = 0$

$\langle \text{proof} \rangle$

**lemma** *oc-projection-alt-def*:

**assumes**  $\text{carr}:(W::'a\ \text{vec}\ \text{set}) \subseteq \text{carrier-vec } n$   $x \in \text{carrier-vec } n$

**and**  $\text{alt1}: y1 \in W$   $x - y1 \in \text{orthogonal-complement } W$

**and**  $\text{alt2}: y2 \in W$   $x - y2 \in \text{orthogonal-complement } W$

**shows**  $y1 = y2$

$\langle \text{proof} \rangle$

**definition**

*is-oc-projection*  $w\ S\ v = (w \in \text{carrier-vec } n \wedge v - w \in \text{span } S \wedge (\forall u. u \in S \longrightarrow w \cdot u = 0))$

**lemma** *is-oc-projection-sq-norm*: **assumes** *is-oc-projection*  $w\ S\ v$

**and**  $S$ :  $S \subseteq \text{carrier-vec } n$

**and**  $v$ :  $v \in \text{carrier-vec } n$

**shows**  $\text{sq-norm } w \leq \text{sq-norm } v$

*<proof>*

**definition** *oc-projection* **where**

*oc-projection*  $S$   $fi \equiv (SOME\ v.\ is\text{-}oc\text{-}projection\ v\ S\ fi)$

**lemma** *inv-in-span*:

**assumes** *incarr*[*intro*]:  $U \subseteq carrier\text{-}vec\ n$  **and** *insp*:  $a \in span\ U$

**shows**  $-a \in span\ U$

*<proof>*

**lemma** *non-span-det-zero*:

**assumes** *len*:  $length\ G = n$

**and** *nonb*:  $\neg (carrier\text{-}vec\ n \subseteq span\ (set\ G))$

**and** *carr*:  $set\ G \subseteq carrier\text{-}vec\ n$

**shows**  $det\ (mat\text{-}of\text{-}rows\ n\ G) = 0$  *<proof>*

**lemma** *span-basis-det-zero-iff*:

**assumes**  $length\ G = n$   $set\ G \subseteq carrier\text{-}vec\ n$

**shows**  $carrier\text{-}vec\ n \subseteq span\ (set\ G) \longleftrightarrow det\ (mat\text{-}of\text{-}rows\ n\ G) \neq 0$  (**is** ?*q1*)

$carrier\text{-}vec\ n \subseteq span\ (set\ G) \longleftrightarrow basis\ (set\ G)$  (**is** ?*q2*)

$det\ (mat\text{-}of\text{-}rows\ n\ G) \neq 0 \longleftrightarrow basis\ (set\ G)$  (**is** ?*q3*)

*<proof>*

**lemma** *lin-indpt-list-nonzero*:

**assumes** *lin-indpt-list*  $G$

**shows**  $0_v\ n \notin set\ G$

*<proof>*

**lemma** *is-oc-projection-eq*:

**assumes** *ispr*: *is-oc-projection*  $a\ S\ v$  *is-oc-projection*  $b\ S\ v$

**and** *carr*:  $S \subseteq carrier\text{-}vec\ n$   $v \in carrier\text{-}vec\ n$

**shows**  $a = b$

*<proof>*

**fun** *adjuster-wit* ::  $'a\ list \Rightarrow 'a\ vec \Rightarrow 'a\ vec\ list \Rightarrow 'a\ list \times 'a\ vec$

**where** *adjuster-wit*  $wits\ w\ [] = (wits,\ 0_v\ n)$

| *adjuster-wit*  $wits\ w\ (u\#\ us) = (let\ a = (w \cdot u) / sq\text{-}norm\ u\ in$

$case\ adjuster\text{-}wit\ (a\ \#\ wits)\ w\ us\ of\ (wit,\ v)$

$\Rightarrow (wit,\ -a \cdot_v\ u + v)$ )

**fun** *sub2-wit* **where**

*sub2-wit*  $us\ [] = ([], [])$

| *sub2-wit*  $us\ (w\ \#\ ws) =$

$(case\ adjuster\text{-}wit\ []\ w\ us\ of\ (wit,\ aw) \Rightarrow let\ u = aw + w\ in$

$case\ sub2\text{-}wit\ (u\ \#\ us)\ ws\ of\ (wits,\ vvs) \Rightarrow (wit\ \#\ wits,\ u\ \#\ vvs))$

**definition** *main* ::  $'a\ vec\ list \Rightarrow 'a\ list\ list \times 'a\ vec\ list$  **where**

```

    main us = sub2-wit [] us
end

locale gram-schmidt-fs =
  fixes n :: nat and fs :: 'a :: {trivial-conjugatable-linordered-field} vec list
begin

sublocale gram-schmidt n TYPE('a) ⟨proof⟩

fun gso and μ where
  gso i = fs ! i + sumlist (map (λ j. - μ i j ·v gso j) [0 ..< i])
| μ i j = (if j < i then (fs ! i · gso j) / sq-norm (gso j) else if i = j then 1 else 0)

declare gso.simps[simp del]
declare μ.simps[simp del]

lemma gso-carrier'[intro]:
  assumes ∧ i. i ≤ j ⇒ fs ! i ∈ carrier-vec n
  shows gso j ∈ carrier-vec n
  ⟨proof⟩

lemma adjuster-wit: assumes res: adjuster-wit wits w us = (wits', a)
  and w: w ∈ carrier-vec n
  and us: ∧ i. i ≤ j ⇒ fs ! i ∈ carrier-vec n
  and us-gs: us = map gso (rev [0 ..< j])
  and wits: wits = map (μ i) [j ..< i]
  and j: j ≤ n j ≤ i
  and wi: w = fs ! i
  shows adjuster n w us = a ∧ a ∈ carrier-vec n ∧ wits' = map (μ i) [0 ..< i] ∧
    (a = sumlist (map (λ j. - μ i j ·v gso j) [0..<j]))
  ⟨proof⟩

lemma sub2-wit:
  assumes set us ⊆ carrier-vec n set ws ⊆ carrier-vec n length us + length ws =
  m
  and ws = map (λ i. fs ! i) [i ..< m]
  and us = map gso (rev [0 ..< i])
  and us: ∧ j. j < m ⇒ fs ! j ∈ carrier-vec n
  and mn: m ≤ n
  shows sub2-wit us ws = (wits, vvs) ⇒ gram-schmidt-sub2 n us ws = vvs
  ∧ vvs = map gso [i ..< m] ∧ wits = map (λ i. map (μ i) [0..<i]) [i ..< m]
  ⟨proof⟩

lemma partial-connect: fixes vs
  assumes length fs = m k ≤ m m ≤ n set us ⊆ carrier-vec n snd (main us) = vs

  us = take k fs set fs ⊆ carrier-vec n

```

**shows**  $gram\text{-}schmidt\ n\ us = vs$   
 $vs = map\ gso\ [0..<k]$   
 $\langle proof \rangle$

**lemma** *adjuster-wit-small*:  
 $(adjuster\text{-}wit\ v\ a\ xs) = (x1, x2)$   
 $\longleftrightarrow (fst\ (adjuster\text{-}wit\ v\ a\ xs) = x1 \wedge x2 = adjuster\ n\ a\ xs)$   
 $\langle proof \rangle$

**lemma** *sub2*:  $rev\ xs\ @\ snd\ (sub2\text{-}wit\ xs\ us) = rev\ (gram\text{-}schmidt\text{-}sub\ n\ xs\ us)$   
 $\langle proof \rangle$

**lemma** *gso-connect*:  $snd\ (main\ us) = gram\text{-}schmidt\ n\ us\ \langle proof \rangle$

**definition** *weakly-reduced* ::  $'a \Rightarrow nat \Rightarrow bool$

**where** *weakly-reduced*  $\alpha\ k = (\forall\ i.\ Suc\ i < k \longrightarrow$   
 $sq\text{-}norm\ (gso\ i) \leq \alpha * sq\text{-}norm\ (gso\ (Suc\ i)))$

**definition** *reduced* ::  $'a \Rightarrow nat \Rightarrow bool$

**where** *reduced*  $\alpha\ k = (weakly\text{-}reduced\ \alpha\ k \wedge$   
 $(\forall\ i\ j.\ i < k \longrightarrow j < i \longrightarrow abs\ (\mu\ i\ j) \leq 1/2))$

**end**

**locale** *gram-schmidt-fs-Rn* = *gram-schmidt-fs* +  
**assumes** *fs-carrier*:  $set\ fs \subseteq carrier\text{-}vec\ n$   
**begin**

**abbreviation** (*input*)  $m$  **where**  $m \equiv length\ fs$

**definition**  $M$  **where**  $M\ k = mat\ k\ k\ (\lambda\ (i,j).\ \mu\ i\ j)$

**lemma** *f-carrier[simp]*:  $i < m \Longrightarrow fs\ !\ i \in carrier\text{-}vec\ n$   
 $\langle proof \rangle$

**lemma** *gso-carrier[simp]*:  $i < m \Longrightarrow gso\ i \in carrier\text{-}vec\ n$   
 $\langle proof \rangle$

**lemma** *gso-dim[simp]*:  $i < m \Longrightarrow dim\text{-}vec\ (gso\ i) = n\ \langle proof \rangle$

**lemma** *f-dim[simp]*:  $i < m \Longrightarrow dim\text{-}vec\ (fs\ !\ i) = n\ \langle proof \rangle$

**lemma** *fs0-gso0*:  $0 < m \Longrightarrow fs\ !\ 0 = gso\ 0$   
 $\langle proof \rangle$

**lemma** *fs-by-gso-def* :

**assumes**  $i: i < m$   
**shows**  $fs ! i = gso\ i + M.sumlist\ (map\ (\lambda ja.\ \mu\ i\ ja\ \cdot_v\ gso\ ja)\ [0..<i])\ (\mathbf{is}\ - = - + ?sum)$   
 $\langle proof \rangle$

**lemma** *main-connect*:  
**assumes**  $m \leq n$   
**shows**  $gram-schmidt\ n\ fs = map\ gso\ [0..<m]$   
 $\langle proof \rangle$

**lemma** *reduced-gso-E*:  $weakly-reduced\ \alpha\ k \implies k \leq m \implies Suc\ i < k \implies sq-norm\ (gso\ i) \leq \alpha * sq-norm\ (gso\ (Suc\ i))$   
 $\langle proof \rangle$

**abbreviation** (*input*)  $FF$  **where**  $FF \equiv mat-of-rows\ n\ fs$   
**abbreviation** (*input*)  $Fs$  **where**  $Fs \equiv mat-of-rows\ n\ (map\ gso\ [0..<m])$

**lemma** *FF-dim[simp]*:  $dim-row\ FF = m\ dim-col\ FF = n\ FF \in carrier-mat\ m\ n$   
 $\langle proof \rangle$

**lemma** *Fs-dim[simp]*:  $dim-row\ Fs = m\ dim-col\ Fs = n\ Fs \in carrier-mat\ m\ n$   
 $\langle proof \rangle$

**lemma** *M-dim[simp]*:  $dim-row\ (M\ m) = m\ dim-col\ (M\ m) = m\ (M\ m) \in carrier-mat\ m\ m$   
 $\langle proof \rangle$

**lemma** *FF-index[simp]*:  $i < m \implies j < n \implies FF\ \$\$ (i,j) = fs ! i \$ j$   
 $\langle proof \rangle$

**lemma** *M-index[simp]*:  $i < m \implies j < m \implies (M\ m)\ \$\$ (i,j) = \mu\ i\ j$   
 $\langle proof \rangle$

**lemma** *matrix-equality*:  $FF = (M\ m) * Fs$   
 $\langle proof \rangle$

**lemma** *fi-is-sum-of-mu-gso*: **assumes**  $i: i < m$   
**shows**  $fs ! i = sumlist\ (map\ (\lambda j.\ \mu\ i\ j\ \cdot_v\ gso\ j)\ [0 ..< Suc\ i])$   
 $\langle proof \rangle$

**lemma** *gi-is-fi-minus-sum-mu-gso*:  
**assumes**  $i: i < m$   
**shows**  $gso\ i = fs ! i - sumlist\ (map\ (\lambda j.\ \mu\ i\ j\ \cdot_v\ gso\ j)\ [0 ..< i])\ (\mathbf{is}\ - = - - ?sum)$   
 $\langle proof \rangle$

**lemma** *det*: **assumes**  $m: m = n$  **shows**  $\det FF = \det Fs$   
 ⟨*proof*⟩  
**end**

**locale** *gram-schmidt-fs-lin-indpt* = *gram-schmidt-fs-Rn* +  
**assumes** *lin-indpt*: *lin-indpt* (*set fs*) **and** *dist*: *distinct fs*  
**begin**

**lemmas** *loc-assms* = *lin-indpt dist*

**lemma** *mn*:  
**shows**  $m \leq n$   
 ⟨*proof*⟩

**lemma**  
**shows** *span-gso*:  $\text{span } (\text{gso } \{0..<m\}) = \text{span } (\text{set } fs)$   
**and** *orthogonal-gso*:  $\text{orthogonal } (\text{map } \text{gso } [0..<m])$   
**and** *dist-gso*:  $\text{distinct } (\text{map } \text{gso } [0..<m])$   
 ⟨*proof*⟩

**lemma** *gso-inj[intro]*:  
**assumes**  $i < m$   
**shows** *inj-on gso*  $\{0..<i\}$   
 ⟨*proof*⟩

**lemma** *partial-span*:  
**assumes**  $i: i \leq m$   
**shows**  $\text{span } (\text{gso } \{0..<i\}) = \text{span } (\text{set } (\text{take } i \text{ } fs))$   
 ⟨*proof*⟩

**lemma** *partial-span'*:  
**assumes**  $i: i \leq m$   
**shows**  $\text{span } (\text{gso } \{0..<i\}) = \text{span } ((\lambda j. fs ! j) \{0..<i\})$   
 ⟨*proof*⟩

**lemma** *orthogonal*:  
**assumes**  $i < m \ j < m \ i \neq j$   
**shows**  $\text{gso } i \cdot \text{gso } j = 0$   
 ⟨*proof*⟩

**lemma** *same-base*:  
**shows**  $\text{span } (\text{set } fs) = \text{span } (\text{gso } \{0..<m\})$   
 ⟨*proof*⟩

**lemma** *sq-norm-gso-le-f*:

**assumes**  $i: i < m$   
**shows**  $sq\text{-norm } (gso\ i) \leq sq\text{-norm } (fs\ !\ i)$   
 $\langle proof \rangle$

**lemma** *oc-projection-exist*:  
**assumes**  $i: i < m$   
**shows**  $fs\ !\ i - gso\ i \in span\ (gso\ \{0..<i\})$   
 $\langle proof \rangle$

**lemma** *oc-projection-unique*:  
**assumes**  $i < m$   
 $v \in carrier\text{-vec } n$   
 $\bigwedge x. x \in gso\ \{0..<i\} \implies v \cdot x = 0$   
 $fs\ !\ i - v \in span\ (gso\ \{0..<i\})$   
**shows**  $v = gso\ i$   
 $\langle proof \rangle$

**lemma** *gso-oc-projection*:  
**assumes**  $i < m$   
**shows**  $gso\ i = oc\text{-projection } (gso\ \{0..<i\})\ (fs\ !\ i)$   
 $\langle proof \rangle$

**lemma** *gso-oc-projection-span*:  
**assumes**  $i < m$   
**shows**  $gso\ i = oc\text{-projection } (span\ (gso\ \{0..<i\}))\ (fs\ !\ i)$   
**and**  $is\text{-oc-projection } (gso\ i)\ (span\ (gso\ \{0..<i\}))\ (fs\ !\ i)$   
 $\langle proof \rangle$

**lemma** *gso-is-oc-projection*:  
**assumes**  $i < m$   
**shows**  $is\text{-oc-projection } (gso\ i)\ (set\ (take\ i\ fs))\ (fs\ !\ i)$   
 $\langle proof \rangle$

**lemma** *fi-scalar-prod-gso*:  
**assumes**  $i: i < m$  **and**  $j: j < m$   
**shows**  $fs\ !\ i \cdot gso\ j = \mu\ i\ j * \|gso\ j\|^2$   
 $\langle proof \rangle$

**lemma** *gso-scalar-zero*:  
**assumes**  $k < m$   $i < k$   
**shows**  $(gso\ k) \cdot (fs\ !\ i) = 0$   
 $\langle proof \rangle$

**lemma** *scalar-prod-lincomb-gso*:  
**assumes**  $k: k \leq m$   
**shows**  $sumlist\ (map\ (\lambda\ i. g\ i \cdot_v\ gso\ i)\ [0\ ..<k]) \cdot sumlist\ (map\ (\lambda\ i. h\ i \cdot_v\ gso\ i)\ [0\ ..<k])$   
 $= sum\text{-list } (map\ (\lambda\ i. g\ i * h\ i * (gso\ i \cdot gso\ i))\ [0\ ..<k])$

*<proof>*

**lemma** *gso-times-self-is-norm:*

**assumes**  $j < m$

**shows**  $fs ! j \cdot gso j = sq\text{-norm } (gso j)$

*<proof>*

**lemma** *gram-schmidt-short-vector:*

**assumes** *in-L*:  $h \in \text{lattice-of } fs - \{0_v\ n\}$

**shows**  $\exists i < m. \|h\|^2 \geq \|gso\ i\|^2$

*<proof>*

**lemma** *weakly-reduced-imp-short-vector:*

**assumes** *weakly-reduced*  $\alpha\ m$

**and** *in-L*:  $h \in \text{lattice-of } fs - \{0_v\ n\}$  **and**  $\alpha\text{-pos}:\alpha \geq 1$

**shows**  $fs \neq [] \wedge sq\text{-norm } (fs ! 0) \leq \alpha^{m-1} * sq\text{-norm } h$

*<proof>*

**lemma** *sq-norm-pos:*

**assumes**  $j < m$

**shows**  $sq\text{-norm } (gso j) > 0$

*<proof>*

**lemma** *Gramian-determinant:*

**assumes**  $k: k \leq m$

**shows** *Gramian-determinant*  $fs\ k = (\prod_{j < k}. sq\text{-norm } (gso j))$

*Gramian-determinant*  $fs\ k > 0$

*<proof>*

**lemma** *Gramian-determinant-div:*

**assumes**  $l < m$

**shows** *Gramian-determinant*  $fs\ (Suc\ l) / \text{Gramian-determinant } fs\ l = \|gso\ l\|^2$

*<proof>*

**end**

**lemma** (*in gram-schmidt-fs-Rn*) *Gramian-determinant-Ints:*

**assumes**  $k \leq m \wedge i\ j. i < n \implies j < m \implies fs ! j\ \$\ i \in \mathbf{Z}$

**shows** *Gramian-determinant*  $fs\ k \in \mathbf{Z}$

*<proof>*

**locale** *gram-schmidt-fs-int = gram-schmidt-fs-lin-indpt +*

**assumes** *fs-int*:  $\wedge i\ j. i < n \implies j < m \implies fs ! j\ \$\ i \in \mathbf{Z}$

**begin**

**lemma** *Gramian-determinant-ge1*:  
**assumes**  $k \leq m$   
**shows**  $1 \leq \text{Gramian-determinant } fs \ k$   
*<proof>*

**lemma** *mu-bound-Gramian-determinant*:  
**assumes**  $l < k \ k < m$   
**shows**  $(\mu \ k \ l)^2 \leq \text{Gramian-determinant } fs \ l * \|fs \ ! \ k\|^2$   
*<proof>*

**end**

**context** *gram-schmidt*  
**begin**

**lemma** *gso-cong*:  
**fixes**  $f1 \ f2 :: 'a \ \text{vec list}$   
**assumes**  $\bigwedge i. i \leq x \implies f1 \ ! \ i = f2 \ ! \ i$   
**shows**  $\text{gram-schmidt-fs.gso } n \ f1 \ x = \text{gram-schmidt-fs.gso } n \ f2 \ x$   
*<proof>*

**lemma** *mu-cong*:  
**fixes**  $f1 \ f2 :: 'a \ \text{vec list}$   
**assumes**  $\bigwedge k. j < i \implies k \leq j \implies f1 \ ! \ k = f2 \ ! \ k$   
**and**  $j < i \implies f1 \ ! \ i = f2 \ ! \ i$   
**shows**  $\text{gram-schmidt-fs.mu } n \ f1 \ i \ j = \text{gram-schmidt-fs.mu } n \ f2 \ i \ j$   
*<proof>*

**end**

**lemma** *prod-list-le-mono*: **fixes**  $us :: 'a :: \{\text{linordered-nonzero-semiring, ordered-ring}\}$   
*list*  
**assumes**  $\text{length } us = \text{length } vs$   
**and**  $\bigwedge i. i < \text{length } vs \implies 0 \leq us \ ! \ i \wedge us \ ! \ i \leq vs \ ! \ i$   
**shows**  $0 \leq \text{prod-list } us \wedge \text{prod-list } us \leq \text{prod-list } vs$   
*<proof>*

**lemma** *lattice-of-of-int*: **assumes**  $G: \text{set } F \subseteq \text{carrier-vec } n$   
**and**  $f \in \text{vec-module.lattice-of } n \ F$   
**shows**  $\text{map-vec rat-of-int } f \in \text{vec-module.lattice-of } n \ (\text{map } (\text{map-vec of-int}) \ F)$   
**(is**  $?f \in \text{vec-module.lattice-of } - \ ?F)$   
*<proof>*

**lemma** *Hadamard's-inequality*:  
**fixes**  $A::\text{real mat}$   
**assumes**  $A: A \in \text{carrier-mat } n \ n$   
**shows**  $\text{abs } (\text{det } A) \leq \text{sqrt } (\text{prod-list } (\text{map } \text{sq-norm } (\text{rows } A)))$

*<proof>*

**definition** *gram-schmidt-wit* = *gram-schmidt.main*

**declare** *gram-schmidt.adjuster-wit.simps*[code]

**declare** *gram-schmidt.sub2-wit.simps*[code]

**declare** *gram-schmidt.main-def*[code]

**definition** *gram-schmidt-int* :: *nat*  $\Rightarrow$  *int vec list*  $\Rightarrow$  *rat list list*  $\times$  *rat vec list*  
**where**

*gram-schmidt-int* *n us* = *gram-schmidt-wit* *n* (*map* (*map-vec of-int*) *us*)

**lemma** *snd-gram-schmidt-int* : *snd* (*gram-schmidt-int* *n us*) = *gram-schmidt* *n*  
(*map* (*map-vec of-int*) *us*)

*<proof>*

Faster implementation for rational vectors which also avoid recomputations of square-norms

**fun** *adjuster-triv* :: *nat*  $\Rightarrow$  *rat vec*  $\Rightarrow$  (*rat vec*  $\times$  *rat*) *list*  $\Rightarrow$  *rat vec*

**where** *adjuster-triv* *n w* [] =  $0_v$  *n*

| *adjuster-triv* *n w* ((*u,nu*)#*us*) =  $-(w \cdot u) / nu \cdot_v u + \text{adjuster-triv } n w \text{ us}$

**fun** *gram-schmidt-sub-triv*

**where** *gram-schmidt-sub-triv* *n us* [] = *us*

| *gram-schmidt-sub-triv* *n us* (*w* # *ws*) = (*let* *u* = *adjuster-triv* *n w us* + *w* *in*  
*gram-schmidt-sub-triv* *n* ((*u, sq-norm-vec-rat u*) # *us*) *ws*)

**definition** *gram-schmidt-triv* :: *nat*  $\Rightarrow$  *rat vec list*  $\Rightarrow$  (*rat vec*  $\times$  *rat*) *list*

**where** *gram-schmidt-triv* *n ws* = *rev* (*gram-schmidt-sub-triv* *n* [] *ws*)

**lemma** *adjuster-triv*: *adjuster-triv* *n w* (*map* ( $\lambda x. (x, \text{sq-norm } x)$ ) *us*) = *adjuster*  
*n w us*

*<proof>*

**lemma** *gram-schmidt-sub-triv*: *gram-schmidt-sub-triv* *n* ((*map* ( $\lambda x. (x, \text{sq-norm } x)$ )  
*us*)) *ws* =

*map* ( $\lambda x. (x, \text{sq-norm } x)$ ) (*gram-schmidt-sub* *n us ws*)

*<proof>*

**lemma** *gram-schmidt-triv[simp]*: *gram-schmidt-triv* *n ws* = *map* ( $\lambda x. (x, \text{sq-norm } x)$ )  
(*gram-schmidt* *n ws*)

*<proof>*

**context** *gram-schmidt*

**begin**

**fun** *mus-adjuster* :: '*a* *vec*  $\Rightarrow$  ('*a* *vec*  $\times$  '*a*) *list*  $\Rightarrow$  '*a* *list*  $\Rightarrow$  '*a* *vec*  $\Rightarrow$  '*a* *list*  $\times$  '*a*  
*vec*

**where**

```
mus-adjuster f []      mus g' = (mus, g') |
mus-adjuster f ((g, ng)#n-gs) mus g' = (let a = (f · g) / ng in
                                          mus-adjuster f n-gs (a # mus) (-a ·v g + g'))
```

**fun norms-mus' where**

```
norms-mus' []      n-gs mus = (map snd n-gs, mus) |
norms-mus' (f # fs) n-gs mus =
  (let (mus-row, g') = mus-adjuster f n-gs [] (0v n);
      g = g' + f in
  norms-mus' fs ((g, sq-norm-vec g) # n-gs) (mus-row#mus))
```

**lemma adjuster-wit-carrier-vec:**

```
assumes f ∈ carrier-vec n set gs ⊆ carrier-vec n
shows snd (adjuster-wit mus f gs) ∈ carrier-vec n
⟨proof⟩
```

**lemma adjuster-wit'':**

```
assumes adjuster-wit mus-acc f gs = (mus, g') n-gs = map (λx. (x, sq-norm-vec
x)) gs
f ∈ carrier-vec n acc ∈ carrier-vec n set gs ⊆ carrier-vec n
shows mus-adjuster f n-gs mus-acc acc = (mus, acc + g')
⟨proof⟩
```

**lemma adjuster-wit':**

```
assumes n-gs = map (λx. (x, sq-norm-vec x)) gs f ∈ carrier-vec n set gs ⊆
carrier-vec n
shows mus-adjuster f n-gs mus-acc (0v n) = adjuster-wit mus-acc f gs
⟨proof⟩
```

**lemma sub2-wit-norms-mus':**

```
assumes n-gs' = map (λv. (v, sq-norm-vec v)) gs'
sub2-wit gs' fs = (mus, gs) set fs ⊆ carrier-vec n set gs' ⊆ carrier-vec n
shows norms-mus' fs n-gs' mus-acc = (map sq-norm-vec (rev gs @ gs'), rev mus
@ mus-acc)
⟨proof⟩
```

**lemma sub2-wit-gram-schmidt-sub-triv'':**

```
assumes sub2-wit [] fs = (mus, gs) set fs ⊆ carrier-vec n
shows norms-mus' fs [] [] = (map sq-norm-vec (rev gs), rev mus)
⟨proof⟩
```

**definition norms-mus where**

```
norms-mus fs = (let (n-gs, mus) = norms-mus' fs [] [] in (rev n-gs, rev mus))
```

**lemma sub2-wit-gram-schmidt-norm-mus:**

```
assumes sub2-wit [] fs = (mus, gs) set fs ⊆ carrier-vec n
shows norms-mus fs = (map sq-norm-vec gs, mus)
⟨proof⟩
```

**lemma** (in *gram-schmidt-fs-Rn*) *norms-mus*: **assumes**  $set\ fs \subseteq carrier\ vec\ n$   $length\ fs \leq n$   
**shows**  $norms\ mus\ fs = (map\ (\lambda j. \|gso\ j\|^2)\ [0..<length\ fs],\ map\ (\lambda i. map\ (\mu\ i)\ [0..<i])\ [0..<length\ fs])$   
 <proof>

**end**

**fun** *mus-adjuster-rat* ::  $rat\ vec \Rightarrow (rat\ vec \times rat)\ list \Rightarrow rat\ list \Rightarrow rat\ vec \Rightarrow rat\ list \times rat\ vec$   
**where**  
 $mus\ adjuster\ rat\ f\ [] = (mus\ g' = (mus,\ g') \mid$   
 $mus\ adjuster\ rat\ f\ ((g,\ ng)\#n\ gs) = (let\ a = (f \cdot g) / ng\ in$   
 $mus\ adjuster\ rat\ f\ n\ gs\ (a\ \# \ mus)\ (-a \cdot_v\ g + g'))$

**fun** *norms-mus-rat'* **where**  
 $norms\ mus\ rat'\ n\ [] = n\ gs\ mus = (map\ snd\ n\ gs,\ mus) \mid$   
 $norms\ mus\ rat'\ n\ (f\ \# fs)\ n\ gs\ mus =$   
 $(let\ (mus\ row,\ g') = mus\ adjuster\ rat\ f\ n\ gs\ []\ (0_v\ n);$   
 $g = g' + f\ in$   
 $norms\ mus\ rat'\ n\ fs\ ((g,\ sq\ norm\ vec\ g)\ \# n\ gs)\ (mus\ row\ \# mus))$

**definition** *norms-mus-rat* **where**  
 $norms\ mus\ rat\ n\ fs = (let\ (n\ gs,\ mus) = norms\ mus\ rat'\ n\ fs\ []\ []\ in\ (rev\ n\ gs,\ rev\ mus))$

**lemma** *norms-mus-rat-norms-mus*:  
 $norms\ mus\ rat\ n\ fs = gram\ schmidt.\ norms\ mus\ n\ fs$   
 <proof>

**lemma** *of-int-dvd*:  
 $b\ dvd\ a\ \mathbf{if}\ of\ int\ a / (of\ int\ b :: 'a :: field\ char\ 0) \in \mathbb{Z}\ b \neq 0$   
 <proof>

**lemma** *denom-dvd-ints*:  
**fixes**  $i::int$   
**assumes**  $quotient\ of\ r = (z,\ n)\ of\ int\ i * r \in \mathbb{Z}$   
**shows**  $n\ dvd\ i$   
 <proof>

**lemma** *quotient-of-bounds*:  
**assumes**  $quotient\ of\ r = (n,\ d)\ rat\ of\ int\ i * r \in \mathbb{Z}\ 0 < i\ |r| \leq b$   
**shows**  $of\ int\ |n| \leq of\ int\ i * b\ d \leq i$   
 <proof>

**context** *gram-schmidt-fs-Rn*  
**begin**

**lemma** *ex-κ*:

**assumes**  $i < \text{length } fs \ l \leq i$

**shows**  $\exists \kappa. \text{sumlist } (\text{map } (\lambda j. - \mu \ i \ j \cdot_v \ gso \ j) \ [0..<l]) =$   
 $\text{sumlist } (\text{map } (\lambda j. \kappa \ j \cdot_v \ fs \ ! \ j) \ [0..<l])$  (**is**  $\exists \kappa. ?Prop \ l \ i \ \kappa$ )

*<proof>*

**definition** *κ-SOME-def*:

$\kappa = (\text{SOME } \kappa. \forall i \ l. i < \text{length } fs \longrightarrow l \leq i \longrightarrow$   
 $\text{sumlist } (\text{map } (\lambda j. - \mu \ i \ j \cdot_v \ gso \ j) \ [0..<l]) =$   
 $\text{sumlist } (\text{map } (\lambda j. \kappa \ i \ l \ j \cdot_v \ fs \ ! \ j) \ [0..<l]))$

**lemma** *κ-def*:

**assumes**  $i < \text{length } fs \ l \leq i$

**shows**  $\text{sumlist } (\text{map } (\lambda j. - \mu \ i \ j \cdot_v \ gso \ j) \ [0..<l]) =$   
 $\text{sumlist } (\text{map } (\lambda j. \kappa \ i \ l \ j \cdot_v \ fs \ ! \ j) \ [0..<l])$

*<proof>*

**lemma** (**in** *gram-schmidt-fs-lin-indpt*) *fs-i-sumlist-κ*:

**assumes**  $i < m \ l \leq i \ j < l$

**shows**  $(fs \ ! \ i + \text{sumlist } (\text{map } (\lambda j. \kappa \ i \ l \ j \cdot_v \ fs \ ! \ j) \ [0..<l])) \cdot fs \ ! \ j = 0$

*<proof>*

**end**

**lemma** *Ints-sum*:

**assumes**  $\bigwedge a. a \in A \implies f \ a \in \mathbb{Z}$

**shows**  $\text{sum } f \ A \in \mathbb{Z}$

*<proof>*

**lemma** *Ints-prod*:

**assumes**  $\bigwedge a. a \in A \implies f \ a \in \mathbb{Z}$

**shows**  $\text{prod } f \ A \in \mathbb{Z}$

*<proof>*

**lemma** *Ints-scalar-prod*:

$v \in \text{carrier-vec } n \implies w \in \text{carrier-vec } n$

$\implies (\bigwedge i. i < n \implies v \ \$ \ i \in \mathbb{Z}) \implies (\bigwedge i. i < n \implies w \ \$ \ i \in \mathbb{Z}) \implies v \cdot w \in \mathbb{Z}$

*<proof>*

**lemma** *Ints-det*: **assumes**  $\bigwedge i \ j. i < \text{dim-row } A \implies j < \text{dim-col } A$

$\implies A \ \$\$ \ (i,j) \in \mathbb{Z}$

**shows**  $\text{det } A \in \mathbb{Z}$

*<proof>*

**lemma** (in *gram-schmidt-fs-Rn*) *Gramian-matrix-alt-alt-def*:  
**assumes**  $k \leq m$   
**shows** *Gramian-matrix*  $fs\ k = mat\ k\ k\ (\lambda(i,j). fs\ !\ i \cdot fs\ !\ j)$   
*<proof>*

**lemma** (in *gram-schmidt-fs-int*) *fs-scalar-Ints*:  
**assumes**  $i < m\ j < m$   
**shows**  $fs\ !\ i \cdot fs\ !\ j \in \mathbb{Z}$   
*<proof>*

**abbreviation** (in *gram-schmidt-fs-lin-indpt*) *d* **where**  $d \equiv Gramian-determinant\ fs$

**lemma** (in *gram-schmidt-fs-lin-indpt*) *fs-i-fs-j-sum- $\kappa$*  :  
**assumes**  $i < m\ l \leq i\ j < l$   
**shows**  $-(fs\ !\ i \cdot fs\ !\ j) = (\sum\ t = 0..<l. fs\ !\ t \cdot fs\ !\ j * \kappa\ i\ l\ t)$   
*<proof>*

**lemma** (in *gram-schmidt-fs-lin-indpt*) *Gramian-matrix-times- $\kappa$*  :  
**assumes**  $i < m\ l \leq i$   
**shows** *Gramian-matrix*  $fs\ l * _v\ (vec\ l\ (\lambda t. \kappa\ i\ l\ t)) = (vec\ l\ (\lambda j. -(fs\ !\ i \cdot fs\ !\ j)))$   
*<proof>*

**lemma** (in *gram-schmidt-fs-int*) *d- $\kappa$ -Ints* :  
**assumes**  $i < m\ l \leq i\ t < l$   
**shows**  $d\ l * \kappa\ i\ l\ t \in \mathbb{Z}$   
*<proof>*

**lemma** (in *gram-schmidt-fs-int*) *d-gso-Ints*:  
**assumes**  $i < n\ k < m$   
**shows**  $(d\ k \cdot _v\ (gso\ k))\ \$\ i \in \mathbb{Z}$   
*<proof>*

**lemma** (in *gram-schmidt-fs-int*) *d-mu-Ints*:  
**assumes**  $l \leq k\ k < m$   
**shows**  $d\ (Suc\ l) * \mu\ k\ l \in \mathbb{Z}$   
*<proof>*

**lemma** *max-list-Max*:  $ls \neq [] \implies max-list\ ls = Max\ (set\ ls)$   
*<proof>*

## 8.1 Explicit Bounds for Size of Numbers that Occur During GSO Algorithm

**context** *gram-schmidt-fs-lin-indpt*  
**begin**

**definition**  $N = \text{Max}(\text{sq-norm } ' \text{ set } fs)$

**lemma**  $N\text{-ge-0}$ :  
 **assumes**  $0 < m$   
 **shows**  $0 \leq N$   
  $\langle \text{proof} \rangle$

**lemma**  $N\text{-fs}$ :  
 **assumes**  $i < m$   
 **shows**  $\|fs ! i\|^2 \leq N$   
  $\langle \text{proof} \rangle$

**lemma**  $N\text{-gso}$ :  
 **assumes**  $i < m$   
 **shows**  $\|gso\ i\|^2 \leq N$   
  $\langle \text{proof} \rangle$

**lemma**  $N\text{-d}$ :  
 **assumes**  $i \leq m$   
 **shows**  $\text{Gramian-determinant } fs\ i \leq N \wedge i$   
  $\langle \text{proof} \rangle$

**end**

**lemma**  $\text{ex-MAXIMUM}$ : **assumes**  $\text{finite } A\ A \neq \{\}$   
 **shows**  $\exists a \in A. \text{Max}(f ' A) = f\ a$   
  $\langle \text{proof} \rangle$

**context**  $\text{gram-schmidt-fs-int}$   
**begin**

**lemma**  $\text{fs-int}'$ :  $k < n \implies f \in \text{set } fs \implies f\ \$\ k \in \mathbb{Z}$   
  $\langle \text{proof} \rangle$

**lemma**  
 **assumes**  $i < m$   
 **shows**  $\text{fs-sq-norm-Ints}: \|fs ! i\|^2 \in \mathbb{Z}$  **and**  $\text{fs-sq-norm-ge-1}: 1 \leq \|fs ! i\|^2$   
  $\langle \text{proof} \rangle$

**lemma**  
 **assumes**  $\text{set } fs \neq \{\}$   
 **shows**  $N\text{-Ints}: N \in \mathbb{Z}$  **and**  $N\text{-1}: 1 \leq N$   
  $\langle \text{proof} \rangle$

**lemma**  $N\text{-mu}$ :

**assumes**  $i < m \ j \leq i$   
**shows**  $(\mu \ i \ j)^2 \leq N \wedge (Suc \ j)$   
 $\langle proof \rangle$

**end**

**lemma** *vec-hom-Ints*:

**assumes**  $i < n \ xs \in carrier\text{-}vec \ n$   
**shows**  $of\text{-}int\text{-}hom.vec\text{-}hom \ xs \ \$ \ i \in \mathbb{Z}$   
 $\langle proof \rangle$

**lemma** *division-to-div*:  $(of\text{-}int \ x \ :: \ 'a \ :: \ floor\text{-}ceiling) = of\text{-}int \ y \ / \ of\text{-}int \ z \implies x = y \ div \ z$   
 $\langle proof \rangle$

**lemma** *exact-division*: **assumes**  $of\text{-}int \ x \ / \ (of\text{-}int \ y \ :: \ 'a \ :: \ floor\text{-}ceiling) \in \mathbb{Z}$   
**shows**  $of\text{-}int \ (x \ div \ y) = of\text{-}int \ x \ / \ (of\text{-}int \ y \ :: \ 'a)$   
 $\langle proof \rangle$

**lemma** *int-via-rat-eqI*:  $rat\text{-}of\text{-}int \ x = rat\text{-}of\text{-}int \ y \implies x = y \ \langle proof \rangle$

**locale** *fs-int* =

**fixes**  
 $n \ :: \ nat \ \mathbf{and}$   
 $fs\text{-}init \ :: \ int \ vec \ list$

**begin**

**sublocale** *vec-module*  $TYPE(int) \ n \ \langle proof \rangle$

**abbreviation** *RAT* **where**  $RAT \equiv map \ (map\text{-}vec \ rat\text{-}of\text{-}int)$

**abbreviation** *(input) m* **where**  $m \equiv length \ fs\text{-}init$

**sublocale** *gs*:  $gram\text{-}schmidt\text{-}fs \ n \ RAT \ fs\text{-}init \ \langle proof \rangle$

**definition**  $d \ :: \ int \ vec \ list \Rightarrow nat \Rightarrow int$  **where**  $d \ fs \ k = gs.Gramian\text{-}determinant \ fs \ k$

**definition**  $D \ :: \ int \ vec \ list \Rightarrow nat$  **where**  $D \ fs = nat \ (\prod \ i < length \ fs. \ d \ fs \ i)$

**lemma** *of-int-Gramian-determinant*:

**assumes**  $k \leq length \ F \ \wedge \ i. \ i < length \ F \implies dim\text{-}vec \ (F \ ! \ i) = n$   
**shows**  $gs.Gramian\text{-}determinant \ (map \ of\text{-}int\text{-}hom.vec\text{-}hom \ F) \ k = of\text{-}int \ (gs.Gramian\text{-}determinant \ F \ k)$   
 $\langle proof \rangle$

**end**

**locale** *fs-int-indpt* =  $fs\text{-}int \ n \ fs$  **for**  $n \ fs$  +  
**assumes** *lin-indep*:  $gs.lin\text{-}indpt\text{-}list \ (RAT \ fs)$

**begin**

**sublocale** *gs*: *gram-schmidt-fs-lin-indpt n RAT fs*  
⟨*proof*⟩

**sublocale** *gs*: *gram-schmidt-fs-int n RAT fs*  
⟨*proof*⟩

**lemma** *f-carrier[dest]*:  $i < m \implies fs ! i \in carrier\text{-}vec\ n$   
**and** *fs-carrier [simp]*:  $set\ fs \subseteq carrier\text{-}vec\ n$   
⟨*proof*⟩

**lemma** *Gramian-determinant*:

**assumes**  $k: k \leq m$

**shows**  $of\text{-}int\ (gs.\text{Gramian-determinant}\ fs\ k) = (\prod_{j < k}. sq\text{-}norm\ (gs.gso\ j))$  (**is**  
*?g1*)

$gs.\text{Gramian-determinant}\ fs\ k > 0$  (**is** *?g2*)

⟨*proof*⟩

**lemma** *fs-int-d-pos [intro]*:

**assumes**  $k: k \leq m$

**shows**  $d\ fs\ k > 0$

⟨*proof*⟩

**lemma** *fs-int-d-Suc*:

**assumes**  $k: k < m$

**shows**  $of\text{-}int\ (d\ fs\ (Suc\ k)) = sq\text{-}norm\ (gs.gso\ k) * of\text{-}int\ (d\ fs\ k)$

⟨*proof*⟩

**lemma** *fs-int-D-pos*:

**shows**  $D\ fs > 0$

⟨*proof*⟩

**definition**  $d\mu\ i\ j = int\text{-}of\text{-}rat\ (of\text{-}int\ (d\ fs\ (Suc\ j)) * gs.\mu\ i\ j)$

**lemma** *fs-int-mu-d-Z*:

**assumes**  $j: j \leq ii$  **and**  $ii: ii < m$

**shows**  $of\text{-}int\ (d\ fs\ (Suc\ j)) * gs.\mu\ ii\ j \in \mathbb{Z}$

⟨*proof*⟩

**lemma** *fs-int-mu-d-Z-m-m*:

**assumes**  $j: j < m$  **and**  $ii: ii < m$

**shows**  $of\text{-}int\ (d\ fs\ (Suc\ j)) * gs.\mu\ ii\ j \in \mathbb{Z}$

⟨*proof*⟩

**lemma** *sq-norm-fs-via-sum-mu-gso*: **assumes**  $i: i < m$

**shows**  $of\text{-}int\ \|fs ! i\|^2 = (\sum_{j \leftarrow [0..<Suc\ i]}. (gs.\mu\ i\ j)^2 * \|gs.gso\ j\|^2)$

⟨*proof*⟩

```

lemma  $d\mu$ : assumes  $j < m \ \dot{i} < m$ 
  shows  $of\text{-}int \ (d\mu \ \dot{i} \ j) = of\text{-}int \ (d \ fs \ (Suc \ j)) * gs.\mu \ \dot{i} \ j$ 
   $\langle proof \rangle$ 

```

```

end

```

```

end

```

## 8.2 Gram-Schmidt Implementation for Integer Vectors

This theory implements the Gram-Schmidt algorithm on integer vectors using purely integer arithmetic. The formalization is based on [1].

```

theory Gram-Schmidt-Int

```

```

  imports

```

```

    Gram-Schmidt-2

```

```

    More-IArray

```

```

begin

```

```

context fixes

```

```

   $fs :: int \ vec \ iarray$  and  $m :: nat$ 

```

```

begin

```

```

fun sigma-array where

```

```

  sigma-array  $dmus \ dmusi \ dmusj \ dll \ l = (if \ l = 0 \ then \ dmusi \ !! \ l * \ dmusj \ !! \ l$ 

```

```

     $else \ let \ l1 = l - 1; \ dll1 = dmus \ !! \ l1 \ !! \ l1 \ in$ 

```

```

     $(dll * \ sigma\text{-}array \ dmus \ dmusi \ dmusj \ dll1 \ l1 + \ dmusi \ !! \ l * \ dmusj \ !! \ l) \ div$ 
     $dll1)$ 

```

```

declare sigma-array.simps[simp del]

```

```

partial-function(tailrec) dmu-array-row-main where

```

```

  [code]: dmu-array-row-main  $fi \ i \ dmus \ j = (if \ j = i \ then \ dmus$ 

```

```

     $else \ let \ sj = Suc \ j;$ 

```

```

     $dmus\text{-}i = dmus \ !! \ i;$ 

```

```

     $djj = dmus \ !! \ j \ !! \ j;$ 

```

```

     $dmu\text{-}ij = djj * (fi \cdot fs \ !! \ sj) - \ sigma\text{-}array \ dmus \ dmus\text{-}i \ (dmus \ !! \ sj) \ djj \ j;$ 

```

```

     $dmus' = iarray\text{-}update \ dmus \ i \ (iarray\text{-}append \ dmus\text{-}i \ dmu\text{-}ij)$ 

```

```

     $in \ dmu\text{-}array\text{-}row\text{-}main \ fi \ i \ dmus' \ sj)$ 

```

```

definition dmu-array-row where

```

```

  dmu-array-row  $dmus \ i = (let \ fi = fs \ !! \ i \ in$ 

```

```

     $dmu\text{-}array\text{-}row\text{-}main \ fi \ i \ (iarray\text{-}append \ dmus \ (IArray \ [fi \cdot fs \ !! \ 0])) \ 0)$ 

```

```

partial-function (tailrec) dmu-array where

```

```

  [code]: dmu-array  $dmus \ i = (if \ i = m \ then \ dmus \ else$ 

```

```

     $let \ dmus' = dmu\text{-}array\text{-}row \ dmus \ i$ 

```

```

     $in \ dmu\text{-}array \ dmus' \ (Suc \ i))$ 

```

```

end

```

**definition**  $d\mu\text{-impl} :: \text{int vec list} \Rightarrow \text{int iarray iarray}$  **where**  
 $d\mu\text{-impl fs} = \text{dmu-array (IArray fs) (length fs) (IArray [])} 0$

**definition** (**in** *gram-schmidt*)  $\beta$  **where**  $\beta fs l = \text{Gramian-determinant fs (Suc l)}$   
 $/ \text{Gramian-determinant fs l}$

**context** *gram-schmidt-fs-lin-indpt*  
**begin**

**lemma** *Gramian-beta*:  
**assumes**  $i < m$   
**shows**  $\beta fs i = \|fs ! i\|^2 - (\sum j = 0..<i. (\mu i j)^2 * \beta fs j)$   
 $\langle \text{proof} \rangle$

**lemma** *gso-norm-beta*:  
**assumes**  $j < m$   
**shows**  $\beta fs j = \text{sq-norm (gso j)}$   
 $\langle \text{proof} \rangle$

**lemma** *mu-Gramian-beta-def*:  
**assumes**  $j < i < m$   
**shows**  $\mu i j = (fs ! i \cdot fs ! j - (\sum k = 0..<j. \mu j k * \mu i k * \beta fs k)) / \beta fs j$   
 $\langle \text{proof} \rangle$

**end**

**lemma** (**in** *gram-schmidt*) *Gramian-matrix-alt-alt-alt-def*:  
**assumes**  $k \leq \text{length fs}$   $\text{set fs} \subseteq \text{carrier-vec } n$   
**shows**  $\text{Gramian-matrix fs } k = \text{mat } k k (\lambda(i,j). fs ! i \cdot fs ! j)$   
 $\langle \text{proof} \rangle$

**lemma** (**in** *gram-schmidt-fs-Rn*) *Gramian-determinant-1 [simp]*:  
**assumes**  $0 < \text{length fs}$   
**shows**  $\text{Gramian-determinant fs (Suc 0)} = \|fs ! 0\|^2$   
 $\langle \text{proof} \rangle$

**context** *gram-schmidt-fs-lin-indpt*  
**begin**

**definition**  $\mu'$  **where**  $\mu' i j \equiv d (Suc j) * \mu i j$

**fun**  $\sigma$  **where**  
 $\sigma 0 i j = 0$   
 $|\ \sigma (Suc l) i j = (d (Suc l) * \sigma l i j + \mu' i l * \mu' j l) / d l$

**lemma** *d-Suc*:  $d (Suc\ i) = \mu'\ i\ i$  *<proof>*

**lemma** *d-0*:  $d\ 0 = 1$  *<proof>*

**lemma**  $\sigma$ : **assumes**  $l j$ :  $l \leq m$

**shows**  $\sigma\ l\ i\ j = d\ l * (\sum_{k < l} \mu\ i\ k * \mu\ j\ k * \beta\ fs\ k)$   
*<proof>*

**lemma**  $\mu'$ : **assumes**  $j$ :  $j \leq i$  **and**  $i$ :  $i < m$

**shows**  $\mu'\ i\ j = d\ j * (fs!\ i \cdot fs!\ j) - \sigma\ j\ i\ j$   
*<proof>*

**lemma**  $\sigma$ -via- $\mu'$ :  $\sigma (Suc\ l)\ i\ j =$

(if  $l = 0$  then  $\mu'\ i\ 0 * \mu'\ j\ 0$  else  $(\mu'\ l\ l * \sigma\ l\ i\ j + \mu'\ i\ l * \mu'\ j\ l) / \mu'\ (l - 1)\ (l - 1)$ )  
*<proof>*

**lemma**  $\mu'$ -via- $\sigma$ : **assumes**  $j$ :  $j \leq i$  **and**  $i$ :  $i < m$

**shows**  $\mu'\ i\ j =$   
(if  $j = 0$  then  $fs!\ i \cdot fs!\ j$  else  $\mu'\ (j - 1)\ (j - 1) * (fs!\ i \cdot fs!\ j) - \sigma\ j\ i\ j$ )  
*<proof>*

**lemma** *fs-i-sumlist- $\kappa$* :

**assumes**  $i < m\ l \leq i\ j < l$

**shows**  $(fs!\ i + sumlist (map (\lambda j. \kappa\ i\ l\ j \cdot_v\ fs!\ j) [0..<l])) \cdot fs!\ j = 0$   
*<proof>*

**end**

**context** *gram-schmidt-fs-int*

**begin**

**lemma**  $\beta$ -pos :  $i < m \implies \beta\ fs\ i > 0$

*<proof>*

**lemma**  $\beta$ -zero :  $i < m \implies \beta\ fs\ i \neq 0$

*<proof>*

**lemma**  $\sigma$ -integer:

**assumes**  $l$ :  $l \leq j$  **and**  $j$ :  $j \leq i$  **and**  $i$ :  $i < m$

**shows**  $\sigma\ l\ i\ j \in \mathbb{Z}$

*<proof>*

**end**

**context** *fs-int-indpt*

**begin**

**fun**  $\sigma s$  **and**  $\mu'$  **where**

$\sigma s \ 0 \ i \ j = \mu' \ i \ 0 * \mu' \ j \ 0$   
|  $\sigma s \ (Suc \ l) \ i \ j = (\mu' \ (Suc \ l) \ (Suc \ l) * \sigma s \ l \ i \ j + \mu' \ i \ (Suc \ l) * \mu' \ j \ (Suc \ l)) \ \text{div} \ \mu' \ l \ l$   
|  $\mu' \ i \ j = (\text{if } j = 0 \ \text{then } fs \ ! \ i \cdot fs \ ! \ j \ \text{else } \mu' \ (j - 1) \ (j - 1) * (fs \ ! \ i \cdot fs \ ! \ j)) - \sigma s \ (j - 1) \ i \ j$

**declare**  $\mu'.simps[simp \ del]$

**lemma**  $\sigma s\text{-}\mu'$ :  $l < j \implies j \leq i \implies i < m \implies \text{of-int } (\sigma s \ l \ i \ j) = \text{gs.}\sigma \ (Suc \ l) \ i \ j$   
 $i < m \implies j \leq i \implies \text{of-int } (\mu' \ i \ j) = \text{gs.}\mu' \ i \ j$   
{proof}

**lemma**  $\mu'$ : **assumes**  $i < m \ j \leq i$   
**shows**  $\mu' \ i \ j = d\mu \ i \ j$   
 $j = i \implies \mu' \ i \ j = d \ fs \ (Suc \ i)$   
{proof}

**lemma**  $\sigma s$ -array: **assumes**  $mm: mm \leq m$  **and**  $j: j < mm$   
**shows**  $l \leq j \implies \sigma s\text{-array} \ (IArray.\text{of-fun } (\lambda i. IArray.\text{of-fun } (\mu' \ i) \ (\text{if } i = mm \ \text{then } Suc \ j \ \text{else } Suc \ i))) \ (Suc \ mm)$   
 $(IArray.\text{of-fun } (\mu' \ mm) \ (Suc \ j)) \ (IArray.\text{of-fun } (\mu' \ (Suc \ j)) \ (\text{if } Suc \ j = mm \ \text{then } Suc \ j \ \text{else } Suc \ (Suc \ j))) \ (\mu' \ l \ l) \ l =$   
 $\sigma s \ l \ mm \ (Suc \ j)$   
{proof}

**lemma**  $d\mu$ -array-row-main: **assumes**  $mm: mm \leq m$  **shows**  
 $j \leq mm \implies d\mu\text{-array-row-main} \ (IArray \ fs) \ (IArray \ fs \ !! \ mm) \ mm$   
 $(IArray.\text{of-fun } (\lambda i. IArray.\text{of-fun } (\mu' \ i) \ (\text{if } i = mm \ \text{then } Suc \ j \ \text{else } Suc \ i))) \ (Suc \ mm)$   
 $j = IArray.\text{of-fun } (\lambda i. IArray.\text{of-fun } (\mu' \ i) \ (Suc \ i)) \ (Suc \ mm)$   
{proof}

**lemma**  $d\mu$ -array-row: **assumes**  $mm: mm \leq m$  **shows**  
 $d\mu\text{-array-row} \ (IArray \ fs) \ (IArray.\text{of-fun } (\lambda i. IArray.\text{of-fun } (\mu' \ i) \ (Suc \ i)) \ mm)$   
 $mm =$   
 $IArray.\text{of-fun } (\lambda i. IArray.\text{of-fun } (\mu' \ i) \ (Suc \ i)) \ (Suc \ mm)$   
{proof}

**lemma**  $d\mu$ -array: **assumes**  $mm \leq m$   
**shows**  $d\mu\text{-array} \ (IArray \ fs) \ m \ (IArray.\text{of-fun } (\lambda i. IArray.\text{of-fun } (\lambda j. \mu' \ i \ j) \ (Suc \ i)) \ mm) \ mm$   
 $= IArray.\text{of-fun } (\lambda i. IArray.\text{of-fun } (\lambda j. \mu' \ i \ j) \ (Suc \ i)) \ m$   
{proof}

**lemma**  $d\mu$ -impl:  $d\mu\text{-impl} \ fs = IArray.\text{of-fun } (\lambda i. IArray.\text{of-fun } (\lambda j. d\mu \ i \ j) \ (Suc$

*i*) *m*  
⟨*proof*⟩

**end**

**context** *gram-schmidt-fs-int*  
**begin**

**lemma** *N-μ'*:  
  **assumes**  $i < m \ j \leq i$   
  **shows**  $(\mu' \ i \ j)^2 \leq N \wedge (3 * \text{Suc } j)$   
⟨*proof*⟩

**lemma** *N-σ*:  
  **assumes**  $i < m \ j \leq i \ l \leq j$   
  **shows**  $|\sigma \ l \ i \ j| \leq \text{of-nat } l * N \wedge (2 * l + 2)$   
⟨*proof*⟩

**lemma** *leq-squared*:  $(z::\text{int}) \leq z^2$   
⟨*proof*⟩

**lemma** *abs-leq-squared*:  $|z::\text{int}| \leq z^2$   
⟨*proof*⟩

**end**

**context** *gram-schmidt-fs-int*  
**begin**

**definition** *gso'* **where**  $\text{gso}' \ i = d \ i \cdot_v (\text{gso } i)$

**fun** *a* **where**  
   $a \ i \ 0 = \text{fs } ! \ i \ |$   
   $a \ i \ (\text{Suc } l) = (1 / d \ l) \cdot_v ((d \ (\text{Suc } l) \cdot_v (a \ i \ l)) - (\mu' \ i \ l) \cdot_v \text{gso}' \ l)$

**lemma** *gso'-carrier-vec*:  
  **assumes**  $i < m$   
  **shows**  $\text{gso}' \ i \in \text{carrier-vec } n$   
⟨*proof*⟩

**lemma** *a-carrier-vec*:  
  **assumes**  $l \leq i \ i < m$   
  **shows**  $a \ i \ l \in \text{carrier-vec } n$   
⟨*proof*⟩

**lemma** *a-l*:  
  **assumes**  $l \leq i \ i < m$   
  **shows**  $a \ i \ l = d \ l \cdot_v (\text{fs } ! \ i + M.\text{sumlist } (\text{map } (\lambda j. - \mu \ i \ j \cdot_v \text{gso } j) [0..<l]))$   
⟨*proof*⟩

**lemma** *a-l'*:  
**assumes**  $i < m$   
**shows**  $a\ i\ i = gso'\ i$   
 $\langle proof \rangle$

**lemma**  
**assumes**  $i < m\ l' \leq i$   
**shows**  $a\ i\ l' = (case\ l'\ of$   
 $\ 0 \Rightarrow fs\ !\ i\ |$   
 $\ Suc\ l \Rightarrow (1 / d\ l) \cdot_v (d\ (Suc\ l) \cdot_v (a\ i\ l) - (\mu'\ i\ l) \cdot_v a\ l\ l))$   
 $\langle proof \rangle$

**lemma** *a-Ints*:  
**assumes**  $i < m\ l \leq i\ k < n$   
**shows**  $a\ i\ l\ \$\ k \in \mathbf{Z}$   
 $\langle proof \rangle$

**lemma** *a-alt-def*:  
**assumes**  $l < length\ fs$   
**shows**  $a\ i\ (Suc\ l) = (let\ v = \mu'\ l\ l \cdot_v (a\ i\ l) - (\mu'\ i\ l) \cdot_v a\ l\ l\ in$   
 $\ (if\ l = 0\ then\ v\ else\ (1 / \mu'\ (l - 1)\ (l - 1)) \cdot_v v))$   
 $\langle proof \rangle$

**end**

**context** *fs-int-indpt*  
**begin**

**fun** *gso-int* ::  $nat \Rightarrow nat \Rightarrow int\ vec$  **where**  
 $gso-int\ i\ 0 = fs\ !\ i\ |$   
 $gso-int\ i\ (Suc\ l) = (let\ v = \mu'\ l\ l \cdot_v (gso-int\ i\ l) - \mu'\ i\ l \cdot_v gso-int\ l\ l\ in$   
 $\ (if\ l = 0\ then\ v\ else\ map-vec\ (\lambda k. k\ div\ \mu'\ (l - 1)\ (l - 1))\ v))$

**lemma** *gso-int-carrier-vec*:  
**assumes**  $i < length\ fs\ l \leq i$   
**shows**  $gso-int\ i\ l \in carrier-vec\ n$   
 $\langle proof \rangle$

**lemma** *gso-int*:  
**assumes**  $i < length\ fs\ l \leq i$   
**shows**  $of-int-hom.vec-hom\ (gso-int\ i\ l) = gs.a\ i\ l$   
 $\langle proof \rangle$

**function** *gso-int-tail'* ::  $nat \Rightarrow nat \Rightarrow int\ vec \Rightarrow int\ vec$  **where**  
 $gso-int-tail'\ i\ l\ acc = (if\ l \geq i\ then\ acc$   
 $\ else\ (let\ v = \mu'\ l\ l \cdot_v acc - \mu'\ i\ l \cdot_v gso-int\ l\ l;$

```

      acc' = (map-vec (λk. k div μ' (l - 1) (l - 1)) v)
      in gso-int-tail' i (l + 1) acc')
    ⟨proof⟩
termination
    ⟨proof⟩

fun gso-int-tail :: nat ⇒ int vec where
  gso-int-tail i = (if i = 0 then fs ! 0 else
    let acc = μ' 0 0 ·v fs ! i - μ' i 0 ·v fs ! 0 in
    gso-int-tail' i 1 acc)

lemma gso-int-tail':
  assumes acc = gso-int i l 0 < i 0 < l l ≤ i
  shows gso-int-tail' i l acc = gso-int i i
  ⟨proof⟩

lemma gso-int-tail: gso-int-tail i = gso-int i i
  ⟨proof⟩

end

locale gso-array
begin

function while :: nat ⇒ nat ⇒ int vec iarray ⇒ int iarray iarray ⇒ int vec ⇒
int vec where
  while i l gsa dmusa acc = (if l ≥ i then acc
    else (let v = dmusa !! l !! l ·v acc - dmusa !! i !! l ·v gsa !! l;
      acc' = (map-vec (λk. k div dmusa !! (l - 1) !! (l - 1)) v)
      in while i (l + 1) gsa dmusa acc'))
  ⟨proof⟩
termination
  ⟨proof⟩

declare while.simps[simp del]

definition gso' where
  gso' i fsa gsa dmusa = (if i = 0 then fsa !! 0 else
    let acc = dmusa !! 0 !! 0 ·v fsa !! i - dmusa !! i !! 0 ·v fsa !! 0 in
    while i 1 gsa dmusa acc)

function gsos' where
  gsos' i n dmusa fsa gsa = (if i ≥ n then gsa else
    gsos' (i + 1) n dmusa fsa (iarray-append gsa (gso' i fsa gsa dmusa)))
  ⟨proof⟩
termination
  ⟨proof⟩

declare gsos'.simps[simp del]

```

**definition** *gso'-array* **where**

*gso'-array* *dmusa fs* = *gsos' 0 (length fs) dmusa (IArray fs) (IArray [])*

**definition** *gso-array* **where**

*gso-array fs* = (*let dmusa = dμ-impl fs; gsa = gso'-array dmusa fs*  
*in IArray.of-fun (λi. (if i = 0 then 1 else inverse (rat-of-int (dmusa*  
*!! (i - 1) !! (i - 1))))*  
*·<sub>v</sub> of-int-hom.vec-hom (gsa !! i) (length fs)*)

**end**

**declare** *gso-array.gso-array-def*[*code*]

**declare** *gso-array.gso'-array-def*[*code*]

**declare** *gso-array.gsos'.simps*[*code*]

**declare** *gso-array.gso'-def*[*code*]

**declare** *gso-array.while.simps*[*code*]

**lemma** *map-vec-id*[*simp*]: *map-vec id = id*  
{*proof*}

**context** *fs-int-indpt*

**begin**

**lemma** *gso-array.gso'-array (dμ-impl fs) fs = IArray (map (λk. gso-int k k) [0..<length fs])*  
{*proof*}

**end**

### 8.3 Lemmas Summarizing All Bounds During GSO Computation

**context** *gram-schmidt-fs-int*

**begin**

**lemma** *combined-size-bound-integer*:

**assumes** *x*: *x* ∈ {*fs ! i \$ j | i j. i < m ∧ j < n*}

∪ {*μ' i j | i j. j ≤ i ∧ i < m*}

∪ {*σ l i j | i j l. i < m ∧ j ≤ i ∧ l ≤ j*}

(**is** *x* ∈ ?*fs* ∪ ?*μ'* ∪ ?*σ*)

**and** *m*: *m* ≠ 0

**shows** *|x|* ≤ *of-nat m \* N ^ (3 \* Suc m)*

{*proof*}

**end**

**context** *fs-int-indpt*  
**begin**

**lemma** *combined-size-bound-rat-log*:

**assumes**  $x: x \in \{gs.\mu' i j \mid i j. j \leq i \wedge i < m\}$   
 $\cup \{gs.\sigma l i j \mid i j l. i < m \wedge j \leq i \wedge l \leq j\}$   
**(is**  $x \in ?\mu' \cup ?\sigma$ )

**and**  $m: m \neq 0 \ x \neq 0$

**shows**  $\log 2 \mid \text{real-of-rat } x \mid \leq \log 2 \ m + (3 + 3 * m) * \log 2 \ (\text{real-of-rat } gs.N)$   
*<proof>*

**lemma** *combined-size-bound-integer-log*:

**assumes**  $x: x \in \{\mu' i j \mid i j. j \leq i \wedge i < m\}$   
 $\cup \{\sigma s l i j \mid i j l. i < m \wedge j \leq i \wedge l < j\}$   
**(is**  $x \in ?\mu' \cup ?\sigma$ )

**and**  $m: m \neq 0 \ x \neq 0$

**shows**  $\log 2 \mid \text{real-of-int } x \mid \leq \log 2 \ m + (3 + 3 * m) * \log 2 \ (\text{real-of-rat } gs.N)$   
*<proof>*

**end**  
**end**

## 9 The LLL Algorithm

Soundness of the LLL algorithm is proven in four steps. In the basic version, we do recompute the Gram-Schmidt orthogonal (GSO) basis in every step. This basic version will have a full functional soundness proof, i.e., termination and the property that the returned basis is reduced. Then in LLL-Number-Bounds we will strengthen the invariant and prove that all intermediate numbers stay polynomial in size. Moreover, in LLL-Impl we will refine the basic version, so that the GSO does not need to be recomputed in every step. Finally, in LLL-Complexity, we develop an cost-annotated version of the refined algorithm and prove a polynomial upper bound on the number of arithmetic operations.

This theory provides a basic implementation and a soundness proof of the LLL algorithm to compute a "short" vector in a lattice.

**theory** *LLL*

**imports**

*Gram-Schmidt-2*

*Missing-Lemmas*

*Jordan-Normal-Form.Determinant*

*Abstract-Rewriting.SN-Order-Carrier*

**begin**

## 9.1 Core Definitions, Invariants, and Theorems for Basic Version

**locale** *LLL* =

**fixes**  $n :: \text{nat}$   
**and**  $m :: \text{nat}$   
**and**  $fs\text{-init} :: \text{int vec list}$   
**and**  $\alpha :: \text{rat}$

**begin**

**sublocale** *vec-module* *TYPE(int)*  $n$   $\langle \text{proof} \rangle$

**abbreviation** *RAT* **where**  $RAT \equiv \text{map } (\text{map-vec rat-of-int})$

**abbreviation** *SRAT* **where**  $SRAT\ xs \equiv \text{set } (RAT\ xs)$

**abbreviation** *Rn* **where**  $Rn \equiv \text{carrier-vec } n :: \text{rat vec set}$

**sublocale** *gs*: *gram-schmidt-fs*  $n$  *RAT* *fs-init*  $\langle \text{proof} \rangle$

**abbreviation** *lin-indep* **where**  $\text{lin-indep } fs \equiv \text{gs.lin-indpt-list } (RAT\ fs)$

**abbreviation** *gso* **where**  $\text{gso } fs \equiv \text{gram-schmidt-fs.gso } n (RAT\ fs)$

**abbreviation**  $\mu$  **where**  $\mu\ fs \equiv \text{gram-schmidt-fs.}\mu\ n (RAT\ fs)$

**abbreviation** *reduced* **where**  $\text{reduced } fs \equiv \text{gram-schmidt-fs.reduced } n (RAT\ fs)\ \alpha$

**abbreviation** *weakly-reduced* **where**  $\text{weakly-reduced } fs \equiv \text{gram-schmidt-fs.weakly-reduced } n (RAT\ fs)\ \alpha$

lattice of initial basis

**definition**  $L = \text{lattice-of } fs\text{-init}$

maximum squared norm of initial basis

**definition**  $N = \text{max-list } (\text{map } (\text{nat } \circ \text{sq-norm})\ fs\text{-init})$

maximum absolute value in initial basis

**definition**  $M = \text{Max } (\{\text{abs } (fs\text{-init } !\ i\ \$\ j) \mid i\ j. i < m \wedge j < n\} \cup \{0\})$

This is the core invariant which enables to prove functional correctness.

**definition**  $\mu\text{-small } fs\ i = (\forall\ j < i. \text{abs } (\mu\ fs\ i\ j) \leq 1/2)$

**definition** *LLL-invariant-weak*  $:: \text{int vec list} \Rightarrow \text{bool}$  **where**

$\text{LLL-invariant-weak } fs = (\text{gs.lin-indpt-list } (RAT\ fs) \wedge$   
 $\text{lattice-of } fs = L \wedge$   
 $\text{length } fs = m)$

**lemma** *LLL-inv-wD*: **assumes** *LLL-invariant-weak fs*

**shows**

*lin-indep fs*

*length (RAT fs) = m*

*set fs ⊆ carrier-vec n*

$\bigwedge i. i < m \implies fs ! i \in carrier-vec n$

$\bigwedge i. i < m \implies gso fs i \in carrier-vec n$

*length fs = m*

*lattice-of fs = L*

*<proof>*

**lemma** *LLL-inv-wI*: **assumes**

*set fs ⊆ carrier-vec n*

*length fs = m*

*lattice-of fs = L*

*lin-indep fs*

**shows** *LLL-invariant-weak fs*

*<proof>*

**definition** *LLL-invariant* :: *bool*  $\implies$  *nat*  $\implies$  *int vec list*  $\implies$  *bool* **where**

*LLL-invariant upw i fs = (*  
  *gs.lin-indpt-list (RAT fs) ∧*  
  *lattice-of fs = L ∧*  
  *reduced fs i ∧*  
  *i ≤ m ∧*  
  *length fs = m ∧*  
  *(upw ∨ μ-small fs i)*  
*)*

**lemma** *LLL-inv-imp-w*: *LLL-invariant upw i fs*  $\implies$  *LLL-invariant-weak fs*

*<proof>*

**lemma** *LLL-invD*: **assumes** *LLL-invariant upw i fs*

**shows**

*lin-indep fs*

*length (RAT fs) = m*

*set fs ⊆ carrier-vec n*

$\bigwedge i. i < m \implies fs ! i \in carrier-vec n$

$\bigwedge i. i < m \implies gso fs i \in carrier-vec n$

*length fs = m*

*lattice-of fs = L*

*weakly-reduced fs i*

*i ≤ m*

*reduced fs i*

*upw ∨ μ-small fs i*

*<proof>*

**lemma** *LLL-invI*: **assumes**

*set fs ⊆ carrier-vec n*

$length\ fs = m$   
 $lattice-of\ fs = L$   
 $i \leq m$   
 $lin-indep\ fs$   
 $reduced\ fs\ i$   
 $upw \vee \mu\text{-small}\ fs\ i$   
**shows**  $LLL\text{-invariant}\ upw\ i\ fs$   
 $\langle proof \rangle$

**end**

**locale**  $fs\text{-int}' =$   
**fixes**  $n\ m\ fs\text{-init}\ fs$   
**assumes**  $LLL\text{-inv}: LLL.LLL\text{-invariant-weak}\ n\ m\ fs\text{-init}\ fs$

**sublocale**  $fs\text{-int}' \subseteq fs\text{-int-ndpt}$   
 $\langle proof \rangle$

**context**  $LLL$   
**begin**

**lemma**  $gso\text{-cong}$ : **assumes**  $\bigwedge i. i \leq x \implies f1\ !\ i = f2\ !\ i$   
 $x < length\ f1\ x < length\ f2$   
**shows**  $gso\ f1\ x = gso\ f2\ x$   
 $\langle proof \rangle$

**lemma**  $\mu\text{-cong}$ : **assumes**  $\bigwedge k. j < i \implies k \leq j \implies f1\ !\ k = f2\ !\ k$   
**and**  $i: i < length\ f1\ i < length\ f2$   
**and**  $j < i \implies f1\ !\ i = f2\ !\ i$   
**shows**  $\mu\ f1\ i\ j = \mu\ f2\ i\ j$   
 $\langle proof \rangle$

**definition**  $reduction\ \mathbf{where}\ reduction = (4 + \alpha) / (4 * \alpha)$

**definition**  $d :: int\ vec\ list \Rightarrow nat \Rightarrow int\ \mathbf{where}\ d\ fs\ k = gs.Gramian\text{-determinant}\ fs\ k$

**definition**  $D :: int\ vec\ list \Rightarrow nat\ \mathbf{where}\ D\ fs = nat\ (\prod\ i < m. d\ fs\ i)$

**definition**  $d\mu\ gs\ i\ j = int\text{-of-rat}\ (of\text{-int}\ (d\ gs\ (Suc\ j)) * \mu\ gs\ i\ j)$

**definition**  $logD :: int\ vec\ list \Rightarrow nat$   
**where**  $logD\ fs = (if\ \alpha = 4/3\ then\ (D\ fs)\ else\ nat\ (floor\ (log\ (1 / of\text{-rat}\ reduction)\ (D\ fs))))$

**definition**  $LLL\text{-measure} :: nat \Rightarrow int\ vec\ list \Rightarrow nat\ \mathbf{where}$   
 $LLL\text{-measure}\ i\ fs = (2 * logD\ fs + m - i)$

**context**  
**fixes**  $fs$   
**assumes**  $Linv$ : *LLL-invariant-weak*  $fs$   
**begin**

**interpretation**  $fs$ :  $fs\text{-int}'\ n\ m\ fs\text{-init}\ fs$   
 $\langle proof \rangle$

**lemma** *Gramian-determinant*:

**assumes**  $k$ :  $k \leq m$   
**shows**  $of\text{-int}\ (gs.\text{Gramian-determinant}\ fs\ k) = (\prod_{j < k}. sq\text{-norm}\ (gso\ fs\ j))$  (**is**  $?g1$ )  
 $gs.\text{Gramian-determinant}\ fs\ k > 0$  (**is**  $?g2$ )  
 $\langle proof \rangle$

**lemma** *LLL-d-pos* [*intro*]: **assumes**  $k$ :  $k \leq m$

**shows**  $d\ fs\ k > 0$   
 $\langle proof \rangle$

**lemma** *LLL-d-Suc*: **assumes**  $k$ :  $k < m$

**shows**  $of\text{-int}\ (d\ fs\ (Suc\ k)) = sq\text{-norm}\ (gso\ fs\ k) * of\text{-int}\ (d\ fs\ k)$   
 $\langle proof \rangle$

**lemma** *LLL-D-pos*:

**shows**  $D\ fs > 0$   
 $\langle proof \rangle$

**end**

Condition when we can increase the value of  $i$

**lemma** *increase-i*:

**assumes**  $Linv$ : *LLL-invariant upw*  $i\ fs$   
**assumes**  $i$ :  $i < m$   
**and**  $upw$ :  $upw \implies i = 0$   
**and**  $red\text{-}i$ :  $i \neq 0 \implies sq\text{-norm}\ (gso\ fs\ (i - 1)) \leq \alpha * sq\text{-norm}\ (gso\ fs\ i)$   
**shows** *LLL-invariant*  $True\ (Suc\ i)\ fs$  *LLL-measure*  $i\ fs > \text{LLL-measure}\ (Suc\ i)\ fs$   
 $\langle proof \rangle$

Standard addition step which makes  $\mu_{i,j}$  small

**definition**  $\mu\text{-small-row}\ i\ fs\ j = (\forall j'. j \leq j' \longrightarrow j' < i \longrightarrow abs\ (\mu\ fs\ i\ j') \leq inverse\ 2)$

**lemma** *basis-reduction-add-row-main*: **assumes**  $Linv$ : *LLL-invariant-weak*  $fs$

**and**  $i$ :  $i < m$  **and**  $j$ :  $j < i$   
**and**  $fs'$ :  $fs' = fs[\ i := fs\ !\ i - c \cdot_v\ fs\ !\ j]$   
**shows** *LLL-invariant-weak*  $fs'$   
 $LLL\text{-invariant}\ True\ i\ fs \implies LLL\text{-invariant}\ True\ i\ fs'$   
 $c = round\ (\mu\ fs\ i\ j) \implies \mu\text{-small-row}\ i\ fs\ (Suc\ j) \implies \mu\text{-small-row}\ i\ fs'\ j$   
 $c = round\ (\mu\ fs\ i\ j) \implies abs\ (\mu\ fs'\ i\ j) \leq 1/2$

*LLL-measure*  $i$   $fs'$  = *LLL-measure*  $i$   $fs$

$\wedge$   $i. i < m \implies gso\ fs' i = gso\ fs i$

$\wedge$   $i' j'. i' < m \implies j' < m \implies$   
 $\mu\ fs' i' j' = (\text{if } i' = i \wedge j' \leq j \text{ then } \mu\ fs\ i\ j' - \text{of-int } c * \mu\ fs\ j\ j' \text{ else } \mu\ fs\ i' j')$

$\wedge$   $ii. ii \leq m \implies d\ fs' ii = d\ fs ii$   
(*proof*)

Addition step which can be skipped since  $\mu$ -value is already small

**lemma** *basis-reduction-add-row-main-0*: **assumes** *Lin*: *LLL-invariant-weak*  $fs$   
**and**  $i: i < m$  **and**  $j: j < i$   
**and**  $0: \text{round } (\mu\ fs\ i\ j) = 0$   
**and** *mu-small*:  $\mu\text{-small-row } i\ fs\ (Suc\ j)$   
**shows**  $\mu\text{-small-row } i\ fs\ j$  (**is** ?*g1*)  
(*proof*)

**lemma** *mu-small-row-refl*:  $\mu\text{-small-row } i\ fs\ i$   
(*proof*)

**lemma** *basis-reduction-add-row-done*: **assumes** *Lin*: *LLL-invariant* *True*  $i\ fs$   
**and**  $i: i < m$   
**and** *mu-small*:  $\mu\text{-small-row } i\ fs\ 0$   
**shows** *LLL-invariant* *False*  $i\ fs$   
(*proof*)

**lemma** *d-swap-unchanged*: **assumes** *len*:  $\text{length } F1 = m$   
**and**  $i0: i \neq 0$  **and**  $i: i < m$  **and**  $ki: k \neq i$  **and**  $km: k \leq m$   
**and** *swap*:  $F2 = F1[i := F1 ! (i - 1), i - 1 := F1 ! i]$   
**shows**  $d\ F1\ k = d\ F2\ k$   
(*proof*)

**definition** *base* **where**  $base = \text{real-of-rat } ((4 * \alpha) / (4 + \alpha))$

**definition** *g-bound* ::  $\text{int vec list} \Rightarrow \text{bool}$  **where**  
 $g\text{-bound } fs = (\forall i < m. \text{sq-norm } (gso\ fs\ i) \leq \text{of-nat } N)$

**end**

**locale** *LLL-with-assms* = *LLL* +  
**assumes**  $\alpha: \alpha \geq 4/3$   
**and** *lin-dep*: *lin-indep*  $fs\text{-init}$   
**and** *len*:  $\text{length } fs\text{-init} = m$   
**begin**  
**lemma**  $\alpha0: \alpha > 0\ \alpha \neq 0$   
(*proof*)

**lemma** *fs-init*:  $set\ fs\text{-}init \subseteq carrier\text{-}vec\ n$   
 ⟨proof⟩

**lemma** *reduction*:  $0 < reduction\ reduction \leq 1$   
 $\alpha > 4/3 \implies reduction < 1$   
 $\alpha = 4/3 \implies reduction = 1$   
 ⟨proof⟩

**lemma** *base*:  $\alpha > 4/3 \implies base > 1$  ⟨proof⟩

**lemma** *basis-reduction-swap-main*: **assumes** *Linvw*: *LLL-invariant-weak fs*  
**and** *small*: *LLL-invariant False i fs*  $\vee$  *abs*  $(\mu\ fs\ i\ (i - 1)) \leq 1/2$   
**and** *i*:  $i < m$   
**and** *i0*:  $i \neq 0$   
**and** *norm-ineq*:  $sq\text{-}norm\ (gso\ fs\ (i - 1)) > \alpha * sq\text{-}norm\ (gso\ fs\ i)$   
**and** *fs'-def*:  $fs' = fs[i := fs\ !\ (i - 1), i - 1 := fs\ !\ i]$   
**shows** *LLL-invariant-weak fs'*  
**and** *LLL-invariant False i fs*  $\implies$  *LLL-invariant False (i - 1) fs'*  
**and** *LLL-measure i fs*  $>$  *LLL-measure (i - 1) fs'*

**and**  $\bigwedge k. k < m \implies gso\ fs'\ k = (if\ k = i - 1\ then$   
 $gso\ fs\ i + \mu\ fs\ i\ (i - 1) \cdot_v\ gso\ fs\ (i - 1)$   
*else if*  $k = i\ then$   
 $gso\ fs\ (i - 1) - (RAT\ fs\ !\ (i - 1) \cdot gso\ fs'\ (i - 1) / sq\text{-}norm\ (gso\ fs'\ (i$   
 $- 1))) \cdot_v\ gso\ fs'\ (i - 1)$   
*else*  $gso\ fs\ k)$  (**is**  $\bigwedge k. - \implies - = ?newg\ k)$

**and**  $\bigwedge k. k < m \implies sq\text{-}norm\ (gso\ fs'\ k) = (if\ k = i - 1\ then$   
 $sq\text{-}norm\ (gso\ fs\ i) + (\mu\ fs\ i\ (i - 1) * \mu\ fs\ i\ (i - 1)) * sq\text{-}norm\ (gso\ fs\ (i$   
 $- 1))$   
*else if*  $k = i\ then$   
 $sq\text{-}norm\ (gso\ fs\ i) * sq\text{-}norm\ (gso\ fs\ (i - 1)) / sq\text{-}norm\ (gso\ fs'\ (i - 1))$   
*else*  $sq\text{-}norm\ (gso\ fs\ k)$ ) (**is**  $\bigwedge k. - \implies - = ?new\text{-}norm\ k)$

**and**  $\bigwedge ii\ j. ii < m \implies j < ii \implies \mu\ fs'\ ii\ j = ($   
*if*  $ii = i - 1\ then$   
 $\mu\ fs\ i\ j$   
*else if*  $ii = i\ then$   
*if*  $j = i - 1\ then$   
 $\mu\ fs\ i\ (i - 1) * sq\text{-}norm\ (gso\ fs\ (i - 1)) / sq\text{-}norm\ (gso\ fs'\ (i - 1))$   
*else*  
 $\mu\ fs\ (i - 1)\ j$   
*else if*  $ii > i \wedge j = i\ then$   
 $\mu\ fs\ ii\ (i - 1) - \mu\ fs\ i\ (i - 1) * \mu\ fs\ ii\ i$   
*else if*  $ii > i \wedge j = i - 1\ then$   
 $\mu\ fs\ ii\ (i - 1) * \mu\ fs'\ i\ (i - 1) + \mu\ fs\ ii\ i * sq\text{-}norm\ (gso\ fs\ i) / sq\text{-}norm$   
 $(gso\ fs'\ (i - 1))$   
*else*  $\mu\ fs\ ii\ j)$  (**is**  $\bigwedge ii\ j. - \implies - \implies - = ?new\text{-}\mu\ ii\ j)$

**and**  $\bigwedge ii. ii \leq m \implies \text{of-int } (d \text{ fs}' ii) = (\text{if } ii = i \text{ then}$   
 $\text{sq-norm } (gso \text{ fs}' (i - 1)) / \text{sq-norm } (gso \text{ fs } (i - 1)) * \text{of-int } (d \text{ fs } i)$   
 $\text{else of-int } (d \text{ fs } ii))$   
 ⟨proof⟩

**lemma** *LLL-inv-initial-state*: *LLL-invariant True 0 fs-init*  
 ⟨proof⟩

**lemma** *LLL-inv-m-imp-reduced*: **assumes** *LLL-invariant True m fs*  
**shows** *reduced fs m*  
 ⟨proof⟩

**lemma** *basis-reduction-short-vector*: **assumes** *LLL-inv: LLL-invariant True m fs*  
**and** *v: v = hd fs*  
**and** *m0: m ≠ 0*  
**shows** *v ∈ carrier-vec n*  
 $v \in L - \{0_v n\}$   
 $h \in L - \{0_v n\} \implies \text{rat-of-int } (\text{sq-norm } v) \leq \alpha \wedge (m - 1) * \text{rat-of-int } (\text{sq-norm } h)$   
 $v \neq 0_v j$   
 ⟨proof⟩

**lemma** *LLL-mu-d-Z*: **assumes** *inv: LLL-invariant-weak fs*  
**and** *j: j ≤ ii and ii: ii < m*  
**shows**  $\text{of-int } (d \text{ fs } (\text{Suc } j)) * \mu \text{ fs } ii j \in \mathbf{Z}$   
 ⟨proof⟩

**context fixes** *fs*  
**assumes** *Liniv: LLL-invariant-weak fs and gbnd: g-bound fs*  
**begin**

**interpretation** *gs1: gram-schmidt-fs-lin-indpt n RAT fs*  
 ⟨proof⟩

**lemma** *LLL-inv-N-pos*: **assumes** *m: m ≠ 0*  
**shows**  $N > 0$   
 ⟨proof⟩

**lemma** *d-approx-main*: **assumes** *i: ii ≤ m m ≠ 0*  
**shows**  $\text{rat-of-int } (d \text{ fs } ii) \leq \text{rat-of-nat } (N \wedge ii)$   
 ⟨proof⟩

**lemma** *d-approx*: **assumes** *i: ii < m*  
**shows**  $\text{rat-of-int } (d \text{ fs } ii) \leq \text{rat-of-nat } (N \wedge ii)$   
 ⟨proof⟩

**lemma** *d-bound*: **assumes**  $i: ii < m$   
**shows**  $d\ fs\ ii \leq N^{ii}$   
 $\langle proof \rangle$

**lemma** *D-approx*:  $D\ fs \leq N^{(m * m)}$   
 $\langle proof \rangle$

**lemma** *LLL-measure-approx*: **assumes**  $\alpha > 4/3\ m \neq 0$   
**shows**  $LLL\text{-measure}\ i\ fs \leq m + 2 * m * m * \log\ base\ N$   
 $\langle proof \rangle$   
**end**

**lemma** *g-bound-fs-init*:  $g\text{-bound}\ fs\text{-init}$   
 $\langle proof \rangle$

**lemma** *LLL-measure-approx-fs-init*:  
 $LLL\text{-invariant}\ upw\ i\ fs\text{-init} \implies 4/3 < \alpha \implies m \neq 0 \implies$   
 $real\ (LLL\text{-measure}\ i\ fs\text{-init}) \leq real\ m + real\ (2 * m * m) * \log\ base\ (real\ N)$   
 $\langle proof \rangle$

**lemma** *N-le-MMn*: **assumes**  $m0: m \neq 0$   
**shows**  $N \leq nat\ M * nat\ M * n$   
 $\langle proof \rangle$

## 9.2 Basic LLL implementation based on previous results

We now assemble a basic implementation of the LLL algorithm, where only the lattice basis is updated, and where the GSO and the  $\mu$ -values are always computed from scratch. This enables a simple soundness proof and permits to separate an efficient implementation from the soundness reasoning.

**fun** *basis-reduction-add-rows-loop* **where**  
 $basis\text{-reduction-add-rows-loop}\ i\ fs\ 0 = fs$   
 $| basis\text{-reduction-add-rows-loop}\ i\ fs\ (Suc\ j) = (\$   
 $\quad let\ c = round\ (\mu\ fs\ i\ j);$   
 $\quad fs' = (if\ c = 0\ then\ fs\ else\ fs[\ i := fs!\ i - c \cdot_v\ fs!\ j]);$   
 $\quad in\ basis\text{-reduction-add-rows-loop}\ i\ fs'\ j)$

**definition** *basis-reduction-add-rows* **where**  
 $basis\text{-reduction-add-rows}\ upw\ i\ fs =$   
 $(if\ upw\ then\ basis\text{-reduction-add-rows-loop}\ i\ fs\ i\ else\ fs)$

**definition** *basis-reduction-swap* **where**  
 $basis\text{-reduction-swap}\ i\ fs = (False, i - 1, fs[i := fs!(i - 1), i - 1 := fs!i])$

**definition** *basis-reduction-step* **where**

*basis-reduction-step upw i fs* = (if  $i = 0$  then (*True*, *Suc i*, *fs*)  
else let  
  *fs'* = *basis-reduction-add-rows upw i fs*  
  in if  $\text{sq-norm } (\text{gso } \text{fs}' (i - 1)) \leq \alpha * \text{sq-norm } (\text{gso } \text{fs}' i)$  then  
    (*True*, *Suc i*, *fs'*)  
  else *basis-reduction-swap i fs'*)

**function** *basis-reduction-main* **where**

*basis-reduction-main (upw,i,fs)* = (if  $i < m \wedge \text{LLL-invariant upw i fs}$   
then *basis-reduction-main (basis-reduction-step upw i fs)* else  
*fs*)  
(*proof*)

**definition** *reduce-basis* = *basis-reduction-main (True, 0, fs-init)*

**definition** *short-vector* = *hd reduce-basis*

Soundness of this implementation is easily proven

**lemma** *basis-reduction-add-rows-loop*: **assumes**

*inv*: *LLL-invariant True i fs*  
**and** *mu-small*:  $\mu\text{-small-row } i \text{ fs } j$   
**and** *res*: *basis-reduction-add-rows-loop i fs j = fs'*  
**and**  $i: i < m$   
**and**  $j: j \leq i$

**shows** *LLL-invariant False i fs' LLL-measure i fs' = LLL-measure i fs*  
(*proof*)

**lemma** *basis-reduction-add-rows*: **assumes**

*inv*: *LLL-invariant upw i fs*  
**and** *res*: *basis-reduction-add-rows upw i fs = fs'*  
**and**  $i: i < m$

**shows** *LLL-invariant False i fs' LLL-measure i fs' = LLL-measure i fs*  
(*proof*)

**lemma** *basis-reduction-swap*: **assumes**

*inv*: *LLL-invariant False i fs*  
**and** *res*: *basis-reduction-swap i fs = (upw',i',fs')*  
**and** *cond*:  $\text{sq-norm } (\text{gso } \text{fs } (i - 1)) > \alpha * \text{sq-norm } (\text{gso } \text{fs } i)$   
**and**  $i: i < m \wedge i \neq 0$

**shows** *LLL-invariant upw' i' fs' (is ?g1)*  
*LLL-measure i' fs' < LLL-measure i fs (is ?g2)*  
(*proof*)

**lemma** *basis-reduction-step*: **assumes**

*inv*: *LLL-invariant upw i fs*  
**and** *res*: *basis-reduction-step upw i fs = (upw',i',fs')*  
**and**  $i: i < m$

**shows** *LLL-invariant upw' i' fs' LLL-measure i' fs' < LLL-measure i fs*

```

⟨proof⟩

termination ⟨proof⟩

declare basis-reduction-main.simps[simp del]

lemma basis-reduction-main: assumes LLL-invariant upw i fs
  and res: basis-reduction-main (upw,i,fs) = fs'
shows LLL-invariant True m fs'
  ⟨proof⟩

lemma reduce-basis-inv: assumes res: reduce-basis = fs
  shows LLL-invariant True m fs
  ⟨proof⟩

lemma reduce-basis: assumes res: reduce-basis = fs
  shows lattice-of fs = L
  reduced fs m
  lin-indep fs
  length fs = m
  ⟨proof⟩

lemma short-vector: assumes res: short-vector = v
  and m0: m ≠ 0
shows v ∈ carrier-vec n
  v ∈ L - {0_v n}
  h ∈ L - {0_v n} ⇒ rat-of-int (sq-norm v) ≤ α ^ (m - 1) * rat-of-int (sq-norm h)
  v ≠ 0_v j
  ⟨proof⟩
end

end

```

### 9.3 Integer LLL Implementation which Stores Multiples of the $\mu$ -Values

In this part we aim to update the integer values  $d(j+1) * \mu_{i,j}$  as well as the Gramian determinants  $d_i$ .

```

theory LLL-Impl
  imports
    LLL
    List-Representation
    Gram-Schmidt-Int
begin

```

### 9.3.1 Updates of the integer values for Swap, Add, etc.

We provide equations how to implement the LLL-algorithm by storing the integer values  $d(j+1) * \mu_{i,j}$  and all  $d i$  in addition to the vectors in  $f$ . Moreover, we show how to check condition like the one on norms via the integer values.

**definition** *round-num-denom* :: *int*  $\Rightarrow$  *int*  $\Rightarrow$  *int* **where**  
*round-num-denom*  $n d = ((2 * n + d) \text{ div } (2 * d))$

**lemma** *round-num-denom*: *round-num-denom*  $num\ denom =$   
*round (of-int num / rat-of-int denom)*  
*<proof>*

**context** *fs-int-indpt*

**begin**

**lemma** *round-num-denom-d $\mu$ -d*:

**assumes**  $j: j \leq i$  **and**  $i: i < m$

**shows** *round-num-denom*  $(d\mu\ i\ j) (d\ fs\ (Suc\ j)) = \text{round } (gs.\mu\ i\ j)$   
*<proof>*

**lemma** *d-sq-norm-comparison*:

**assumes** *quot*: *quotient-of*  $\alpha = (num, denom)$

**and**  $i: i < m$

**and**  $i0: i \neq 0$

**shows**  $(d\ fs\ i * d\ fs\ i * denom \leq num * d\ fs\ (i - 1) * d\ fs\ (Suc\ i))$   
 $= (sq\ norm\ (gs.gso\ (i - 1)) \leq \alpha * sq\ norm\ (gs.gso\ i))$

*<proof>*

**end**

**context** *LLL*

**begin**

**lemma** *d-d $\mu$ -add-row*: **assumes** *Lin*: *LLL-invariant-weak fs*

**and**  $i: i < m$  **and**  $j: j < i$

**and**  $fs'$ :  $fs' = fs[ i := fs ! i - c \cdot_v fs ! j ]$

**shows**

$\bigwedge ii. ii \leq m \implies d\ fs' ii = d\ fs ii$

$\bigwedge i' j'. i' < m \implies j' < i' \implies$

$d\mu\ fs' i' j' = ($

*if*  $i' = i \wedge j' < j$

*then*  $d\mu\ fs\ i' j' - c * d\mu\ fs\ j j'$

*else if*  $i' = i \wedge j' = j$

*then*  $d\mu\ fs\ i' j' - c * d\ fs\ (Suc\ j)$

*else*  $d\mu\ fs\ i' j'$ )

(**is**  $\bigwedge i' j'. - \implies - \implies - = ?new\mu\ i' j'$ )

*<proof>*

**end**

**context** *LLL-with-assms*

**begin**

**lemma** *d-dμ-swap*: **assumes** *invw*: *LLL-invariant-weak fs*

**and** *small*: *LLL-invariant False k fs*  $\vee$  *abs* ( $\mu$  *fs* *k* (*k* - 1))  $\leq$  1/2

**and** *k*: *k* < *m*

**and** *k0*: *k*  $\neq$  0

**and** *norm-ineq*: *sq-norm* (*gso fs* (*k* - 1)) >  $\alpha$  \* *sq-norm* (*gso fs* *k*)

**and** *fs'-def*: *fs'* = *fs*[*k* := *fs* ! (*k* - 1), *k* - 1 := *fs* ! *k*]

**shows**

$\bigwedge$  *i*. *i*  $\leq$  *m*  $\implies$

*d fs'* *i* = (

if *i* = *k* then

(*d fs* (*Suc k*) \* *d fs* (*k* - 1) +  $d\mu$  *fs* *k* (*k* - 1) \*  $d\mu$  *fs* *k* (*k* - 1)) *div d fs*

*k*

else *d fs* *i*)

**and**

$\bigwedge$  *i j*. *i* < *m*  $\implies$  *j* < *i*  $\implies$

$d\mu$  *fs'* *i j* = (

if *i* = *k* - 1 then

$d\mu$  *fs* *k j*

else if *i* = *k*  $\wedge$  *j*  $\neq$  *k* - 1 then

$d\mu$  *fs* (*k* - 1) *j*

else if *i* > *k*  $\wedge$  *j* = *k* then

(*d fs* (*Suc k*) \*  $d\mu$  *fs* *i* (*k* - 1) -  $d\mu$  *fs* *k* (*k* - 1) \*  $d\mu$  *fs* *i j*) *div d fs* *k*

else if *i* > *k*  $\wedge$  *j* = *k* - 1 then

( $d\mu$  *fs* *k* (*k* - 1) \*  $d\mu$  *fs* *i j* +  $d\mu$  *fs* *i k* \* *d fs* (*k* - 1)) *div d fs* *k*

else  $d\mu$  *fs* *i j*)

(**is**  $\bigwedge$  *i j*. -  $\implies$  -  $\implies$  - = ?*new-mu* *i j*)

*<proof>*

**end**

### 9.3.2 Implementation of LLL via Integer Operations and Arrays

**hide-fact** (**open**) *Word.inc-i*

**type-synonym** *LLL-dmu-d-state* = *int vec list-repr*  $\times$  *int iarray iarray*  $\times$  *int iarray*

**fun** *fi-state* :: *LLL-dmu-d-state*  $\Rightarrow$  *int vec* **where**

*fi-state* (*f*, *mu*, *d*) = *get-nth-i* *f*

**fun** *fim1-state* :: *LLL-dmu-d-state*  $\Rightarrow$  *int vec* **where**

*fim1-state* (*f*, *mu*, *d*) = *get-nth-im1* *f*

```

fun d-state :: LLL-dmu-d-state  $\Rightarrow$  nat  $\Rightarrow$  int where
  d-state (f,mu,d) i = d !! i

fun fs-state :: LLL-dmu-d-state  $\Rightarrow$  int vec list where
  fs-state (f,mu,d) = of-list-repr f

fun upd-fi-mu-state :: LLL-dmu-d-state  $\Rightarrow$  nat  $\Rightarrow$  int vec  $\Rightarrow$  int iarray  $\Rightarrow$  LLL-dmu-d-state
where
  upd-fi-mu-state (f,mu,d) i fi mu-i = (update-i f fi, iarray-update mu i mu-i,d)

fun small-fs-state :: LLL-dmu-d-state  $\Rightarrow$  int vec list where
  small-fs-state (f,-) = fst f

fun dmu-ij-state :: LLL-dmu-d-state  $\Rightarrow$  nat  $\Rightarrow$  nat  $\Rightarrow$  int where
  dmu-ij-state (f,mu,-) i j = mu !! i !! j

fun inc-state :: LLL-dmu-d-state  $\Rightarrow$  LLL-dmu-d-state where
  inc-state (f,mu,d) = (inc-i f, mu, d)

fun basis-reduction-add-rows-loop where
  basis-reduction-add-rows-loop n state i j [] = state
| basis-reduction-add-rows-loop n state i sj (fj # fjs) = (
  let fi = fi-state state;
      dsj = d-state state sj;
      j = sj - 1;
      c = round-num-denom (dmu-ij-state state i j) dsj;
      state' = (if c = 0 then state else upd-fi-mu-state state i (vec n ( $\lambda$  i. fi $ i
- c * fj $ i))
      (IArray.of-fun ( $\lambda$  jj. let mu = dmu-ij-state state i jj in
      if jj < j then mu - c * dmu-ij-state state j jj else
      if jj = j then mu - dsj * c else mu) i))
  in basis-reduction-add-rows-loop n state' i j fjs)

```

More efficient code which breaks abstraction of state.

```

lemma basis-reduction-add-rows-loop-code:
  basis-reduction-add-rows-loop n state i sj (fj # fjs) = (
  case state of ((f1,f2),mus,ds)  $\Rightarrow$ 
  let fi = hd f2;
      j = sj - 1;
      dsj = ds !! sj;
      mui = mus !! i;
      c = round-num-denom (mui !! j) dsj
  in (if c = 0 then
  basis-reduction-add-rows-loop n state i j fjs
  else
  let muj = mus !! j in
  basis-reduction-add-rows-loop n
  ((f1, vec n ( $\lambda$  i. fi $ i - c * fj $ i) # tl f2), iarray-update mus i
  (IArray.of-fun ( $\lambda$  jj. let mu = mui !! jj in

```

```

      if jj < j then mu - c * muj !! jj else
      if jj = j then mu - dsj * c else mu) i),
      ds) i j fjs))
⟨proof⟩

lemmas basis-reduction-add-rows-loop-code-equations =
  basis-reduction-add-rows-loop.simps(1) basis-reduction-add-rows-loop-code

declare basis-reduction-add-rows-loop-code-equations[code]

definition basis-reduction-add-rows where
  basis-reduction-add-rows n upw i state =
    (if upw
     then basis-reduction-add-rows-loop n state i i (small-fs-state state)
     else state)

context
  fixes α :: rat and n m :: nat and fs-init :: int vec list
begin

definition swap-mu :: int iarray iarray ⇒ nat ⇒ int ⇒ int ⇒ int ⇒ int ⇒ int ⇒ int
iarray iarray where
  swap-mu dmu i dmu-i-im1 dim1 di dsi = (let im1 = i - 1 in
    IArray.of-fun (λ ii. if ii < im1 then dmu !! ii else
      if ii > i then let dmu-ii = dmu !! ii in
        IArray.of-fun (λ j. let dmu-ii-j = dmu-ii !! j in
          if j = i then (dsi * dmu-ii !! im1 - dmu-i-im1 * dmu-ii-j) div di
          else if j = im1 then (dmu-i-im1 * dmu-ii-j + dmu-ii !! i * dim1) div di
          else dmu-ii-j) ii else
      if ii = i then let mu-im1 = dmu !! im1 in
        IArray.of-fun (λ j. if j = im1 then dmu-i-im1 else mu-im1 !! j) ii
      else IArray.of-fun (λ j. dmu !! i !! j) ii) — ii = i - 1
    m)

definition basis-reduction-swap where
  basis-reduction-swap i state = (let
    di = d-state state i;
    dsi = d-state state (Suc i);
    dim1 = d-state state (i - 1);
    fi = fi-state state;
    fim1 = fim1-state state;
    dmu-i-im1 = dmu-ij-state state i (i - 1);
    fi' = fim1;
    fim1' = fi
  in (case state of (f, dmus, djs) ⇒
    (False, i - 1,
     (dec-i (update-im1 (update-i f fi') fim1'),
      swap-mu dmus i dmu-i-im1 dim1 di dsi,

```

$iarray\text{-}update\ djs\ i\ (((dsi * dim1 + dm\mu\text{-}i\text{-}im1 * dm\mu\text{-}i\text{-}im1)\ div\ di))))$

More efficient code which breaks abstraction of state.

**lemma** *basis-reduction-swap-code*[code]:

```

basis-reduction-swap i ((f1,f2), dmus, ds) = (let
  di = ds !! i;
  dsi = ds !! (Suc i);
  im1 = i - 1;
  dim1 = ds !! im1;
  fi = hd f2;
  fim1 = hd f1;
  dmu-i-im1 = dmus !! i !! im1;
  fi' = fim1;
  fim1' = fi
in (False, im1,
  ((tl f1,fim1' # fi' # tl f2),
  swap-mu dmus i dmu-i-im1 dim1 di dsi,
  iarray-update ds i (((dsi * dim1 + dmu-i-im1 * dmu-i-im1) div di))))
⟨proof⟩

```

**definition** *basis-reduction-step* **where**

```

basis-reduction-step upw i state = (if i = 0 then (True, Suc i, inc-state state)
  else let
    state' = basis-reduction-add-rows n upw i state;
    di = d-state state' i;
    dsi = d-state state' (Suc i);
    dim1 = d-state state' (i - 1);
    (num,denom) = quotient-of  $\alpha$ 
  in if  $di * di * denom \leq num * dim1 * dsi$  then
    (True, Suc i, inc-state state')
    else basis-reduction-swap i state')

```

**partial-function** (*tailrec*) *basis-reduction-main* **where**

```

[code]: basis-reduction-main upw i state = (if i < m
  then case basis-reduction-step upw i state of (upw',i',state') ⇒
    basis-reduction-main upw' i' state' else
  state)

```

**definition** *initial-state* = (let

```

  dmus = dμ-impl fs-init;
  ds = IArray.of-fun ( $\lambda\ i.$  if  $i = 0$  then 1 else let  $i1 = i - 1$  in dmus !!  $i1$  !!  $i1$ )
(Suc m);
  dmus' = IArray.of-fun ( $\lambda\ i.$  let row-i = dmus !!  $i$  in
    IArray.of-fun ( $\lambda\ j.$  row-i !!  $j$ )  $i$ ) m
in (([], fs-init), dmus', ds) :: LLL-dmu-d-state)

```

**end**

**definition** *basis-reduction*  $\alpha$  *n* *fs* = (let *m* = *length* *fs* in

*basis-reduction-main*  $\alpha$   $n$   $m$  *True*  $0$  (*initial-state*  $m$   $fs$ )

**definition** *reduce-basis*  $\alpha$   $fs = (case$   $fs$  *of*  $Nil \Rightarrow fs \mid Cons$   $f - \Rightarrow fs$ -*state* (*basis-reduction*  $\alpha$  (*dim-vec*  $f$ )  $fs$ ))

**definition** *short-vector*  $\alpha$   $fs = hd$  (*reduce-basis*  $\alpha$   $fs$ )

**lemma** *map-rev-Suc*:  $map$   $f$  (*rev*  $[0..<Suc$   $j]$ )  $= f$   $j$   $\#$   $map$   $f$  (*rev*  $[0..<j]$ )  $\langle proof \rangle$

**context** *LLL*

**begin**

**definition** *mu-repr*  $:: int$  *iarray* *iarray*  $\Rightarrow int$  *vec list*  $\Rightarrow bool$  **where**  
 $mu$ -*repr*  $mu$   $fs = (mu = IArray.of$ -*fun*  $(\lambda i. IArray.of$ -*fun* ( $d\mu$   $fs$   $i$ )  $i$ )  $m$ )

**definition** *d-repr*  $:: int$  *iarray*  $\Rightarrow int$  *vec list*  $\Rightarrow bool$  **where**  
 $d$ -*repr*  $ds$   $fs = (ds = IArray.of$ -*fun* ( $d$   $fs$ ) (*Suc*  $m$ ))

**fun** *LLL-impl-inv*  $:: LLL$ -*dmu-d-state*  $\Rightarrow nat$   $\Rightarrow int$  *vec list*  $\Rightarrow bool$  **where**  
 $LLL$ -*impl-inv*  $(f, mu, ds)$   $i$   $fs = (list$ -*repr*  $i$   $f$  ( $map$   $(\lambda j. fs$   $!$   $j)$   $[0..<m]$ )  
 $\wedge d$ -*repr*  $ds$   $fs$   
 $\wedge mu$ -*repr*  $mu$   $fs$ )

**context** *fixes*  $state$   $i$   $fs$  *upw*  $f$   $mu$   $ds$   
**assumes** *impl*:  $LLL$ -*impl-inv*  $state$   $i$   $fs$   
**and** *inv*:  $LLL$ -*invariant* *upw*  $i$   $fs$   
**and** *state*:  $state = (f, mu, ds)$

**begin**

**lemma** *to-list-repr*:  $list$ -*repr*  $i$   $f$  ( $map$   $(!$ )  $fs$ )  $[0..<m]$   
 $\langle proof \rangle$

**lemma** *to-mu-repr*:  $mu$ -*repr*  $mu$   $fs$   $\langle proof \rangle$

**lemma** *to-d-repr*:  $d$ -*repr*  $ds$   $fs$   $\langle proof \rangle$

**lemma** *dmu-ij-state*: **assumes**  $j: j < ii$   
**and**  $ii: ii < m$   
**shows**  $dmu$ -*ij-state*  $state$   $ii$   $j = d\mu$   $fs$   $ii$   $j$   
 $\langle proof \rangle$

**lemma** *fi-state*:  $i < m \Longrightarrow fi$ -*state*  $state = fs$   $!$   $i$   
 $\langle proof \rangle$

**lemma** *fim1-state*:  $i < m \Longrightarrow i \neq 0 \Longrightarrow fim1$ -*state*  $state = fs$   $!$   $(i - 1)$   
 $\langle proof \rangle$

**lemma** *d-state*:  $ii \leq m \Longrightarrow d$ -*state*  $state$   $ii = d$   $fs$   $ii$   
 $\langle proof \rangle$

**lemma** *fs-state*:  $length$   $fs = m \Longrightarrow fs$ -*state*  $state = fs$

*<proof>*

**lemma** *LLL-state-inc-state*: **assumes**  $i: i < m$   
**shows** *LLL-impl-inv* (*inc-state state*) (*Suc i*) *fs*  
  *fs-state* (*inc-state state*) = *fs-state state*  
*<proof>*  
**end**  
**end**

**context** *LLL-with-assms*  
**begin**

**lemma** *basis-reduction-add-rows-loop-impl*: **assumes**  
  *impl*: *LLL-impl-inv state i fs*  
  **and** *inv*: *LLL-invariant True i fs*  
  **and** *mu-small*:  $\mu$ -*small-row i fs j*  
  **and** *res*: *LLL-Impl.basis-reduction-add-rows-loop n state i j*  
    (*map* (!) *fs*) (*rev* [0 ..< j])) = *state'*  
    (**is** *LLL-Impl.basis-reduction-add-rows-loop n state i j* (?*mapf fs j*) = -)  
  **and**  $j: j \leq i$   
  **and**  $i: i < m$   
  **and**  $fs'$ :  $fs' = fs\text{-state } state'$   
**shows**  
  *LLL-impl-inv state' i fs'*  
  *basis-reduction-add-rows-loop i fs j = fs'*  
*<proof>*

**lemma** *basis-reduction-add-rows-loop*: **assumes**  
  *impl*: *LLL-impl-inv state i fs*  
  **and** *inv*: *LLL-invariant True i fs*  
  **and** *mu-small*:  $\mu$ -*small-row i fs j*  
  **and** *res*: *LLL-Impl.basis-reduction-add-rows-loop n state i j*  
    (*map* (!) *fs*) (*rev* [0 ..< j])) = *state'*  
    (**is** *LLL-Impl.basis-reduction-add-rows-loop n state i j* (?*mapf fs j*) = -)  
  **and**  $j: j \leq i$   
  **and**  $i: i < m$   
  **and**  $fs'$ :  $fs' = fs\text{-state } state'$   
**shows**  
  *LLL-impl-inv state' i fs'*  
  *LLL-invariant False i fs'*  
  *LLL-measure i fs' = LLL-measure i fs*  
  *basis-reduction-add-rows-loop i fs j = fs'*  
*<proof>*

**lemma** *basis-reduction-add-rows-impl*: **assumes**  
  *impl*: *LLL-impl-inv state i fs*  
  **and** *inv*: *LLL-invariant upw i fs*  
  **and** *res*: *LLL-Impl.basis-reduction-add-rows n upw i state = state'*  
  **and**  $i: i < m$

**and**  $fs'$ :  $fs' = fs\text{-state } state'$   
**shows**  
 $LLL\text{-impl-inv } state' i fs'$   
 $basis\text{-reduction-add-rows } upw i fs = fs'$   
 ⟨proof⟩

**lemma** *basis-reduction-add-rows*: **assumes**  
 $impl$ :  $LLL\text{-impl-inv } state i fs$   
**and**  $inv$ :  $LLL\text{-invariant } upw i fs$   
**and**  $res$ :  $LLL\text{-Impl.basis-reduction-add-rows } n upw i state = state'$   
**and**  $i$ :  $i < m$   
**and**  $fs'$ :  $fs' = fs\text{-state } state'$   
**shows**  
 $LLL\text{-impl-inv } state' i fs'$   
 $LLL\text{-invariant } False i fs'$   
 $LLL\text{-measure } i fs' = LLL\text{-measure } i fs$   
 $basis\text{-reduction-add-rows } upw i fs = fs'$   
 ⟨proof⟩

**lemma** *basis-reduction-swap-impl*: **assumes**  
 $impl$ :  $LLL\text{-impl-inv } state i fs$   
**and**  $inv$ :  $LLL\text{-invariant } False i fs$   
**and**  $res$ :  $LLL\text{-Impl.basis-reduction-swap } m i state = (upw', i', state')$   
**and**  $cond$ :  $sq\text{-norm } (gso fs (i - 1)) > \alpha * sq\text{-norm } (gso fs i)$   
**and**  $i$ :  $i < m$  **and**  $i0$ :  $i \neq 0$   
**and**  $fs'$ :  $fs' = fs\text{-state } state'$   
**shows**  
 $LLL\text{-impl-inv } state' i' fs'$  (**is** ?g1)  
 $basis\text{-reduction-swap } i fs = (upw', i', fs')$  (**is** ?g2)  
 ⟨proof⟩

**lemma** *basis-reduction-swap*: **assumes**  
 $impl$ :  $LLL\text{-impl-inv } state i fs$   
**and**  $inv$ :  $LLL\text{-invariant } False i fs$   
**and**  $res$ :  $LLL\text{-Impl.basis-reduction-swap } m i state = (upw', i', state')$   
**and**  $cond$ :  $sq\text{-norm } (gso fs (i - 1)) > \alpha * sq\text{-norm } (gso fs i)$   
**and**  $i$ :  $i < m$  **and**  $i0$ :  $i \neq 0$   
**and**  $fs'$ :  $fs' = fs\text{-state } state'$   
**shows**  
 $LLL\text{-impl-inv } state' i' fs'$   
 $LLL\text{-invariant } upw' i' fs'$   
 $LLL\text{-measure } i' fs' < LLL\text{-measure } i fs$   
 $basis\text{-reduction-swap } i fs = (upw', i', fs')$   
 ⟨proof⟩

**lemma** *basis-reduction-step-impl*: **assumes**  
 $impl$ :  $LLL\text{-impl-inv } state i fs$   
**and**  $inv$ :  $LLL\text{-invariant } upw i fs$   
**and**  $res$ :  $LLL\text{-Impl.basis-reduction-step } \alpha n m upw i state = (upw', i', state')$

**and**  $i: i < m$   
**and**  $fs': fs' = fs\text{-state } state'$   
**shows**  
 $LLL\text{-impl-inv } state' i' fs'$   
 $basis\text{-reduction-step } upw i fs = (upw', i', fs')$   
 $\langle proof \rangle$

**lemma** *basis-reduction-step*: **assumes**  
 $impl: LLL\text{-impl-inv } state i fs$   
**and**  $inv: LLL\text{-invariant } upw i fs$   
**and**  $res: LLL\text{-Impl.basis-reduction-step } \alpha n m upw i state = (upw', i', state')$   
**and**  $i: i < m$   
**and**  $fs': fs' = fs\text{-state } state'$   
**shows**  
 $LLL\text{-impl-inv } state' i' fs'$   
 $LLL\text{-invariant } upw' i' fs'$   
 $LLL\text{-measure } i' fs' < LLL\text{-measure } i fs$   
 $basis\text{-reduction-step } upw i fs = (upw', i', fs')$   
 $\langle proof \rangle$

**lemma** *basis-reduction-main-impl*: **assumes**  
 $impl: LLL\text{-impl-inv } state i fs$   
**and**  $inv: LLL\text{-invariant } upw i fs$   
**and**  $res: LLL\text{-Impl.basis-reduction-main } \alpha n m upw i state = state'$   
**and**  $fs': fs' = fs\text{-state } state'$   
**shows**  $LLL\text{-impl-inv } state' m fs'$   
 $basis\text{-reduction-main } (upw, i, fs) = fs'$   
 $\langle proof \rangle$

**lemma** *basis-reduction-main*: **assumes**  
 $impl: LLL\text{-impl-inv } state i fs$   
**and**  $inv: LLL\text{-invariant } upw i fs$   
**and**  $res: LLL\text{-Impl.basis-reduction-main } \alpha n m upw i state = state'$   
**and**  $fs': fs' = fs\text{-state } state'$   
**shows**  
 $LLL\text{-invariant } True m fs'$   
 $LLL\text{-impl-inv } state' m fs'$   
 $basis\text{-reduction-main } (upw, i, fs) = fs'$   
 $\langle proof \rangle$

**lemma** *initial-state*:  $LLL\text{-impl-inv } (initial\text{-state } m fs\text{-init}) 0 fs\text{-init } (\mathbf{is } ?g1)$   
 $fs\text{-state } (initial\text{-state } m fs\text{-init}) = fs\text{-init } (\mathbf{is } ?g2)$   
 $\langle proof \rangle$

**lemma** *basis-reduction*: **assumes**  $res: basis\text{-reduction } \alpha n fs\text{-init} = state$   
**and**  $fs: fs = fs\text{-state } state$   
**shows**  $LLL\text{-invariant } True m fs$   
 $LLL\text{-impl-inv } state m fs$   
 $basis\text{-reduction-main } (True, 0, fs\text{-init}) = fs$

*<proof>*

**lemma** *reduce-basis-impl*: *LLL-Impl.reduce-basis*  $\alpha$  *fs-init* = *reduce-basis*  
*<proof>*

**lemma** *reduce-basis*: **assumes** *LLL-Impl.reduce-basis*  $\alpha$  *fs-init* = *fs*  
**shows** *lattice-of fs* = *L*  
*reduced fs m*  
*lin-indep fs*  
*length fs* = *m*  
*LLL-invariant True m fs*  
*<proof>*

**lemma** *short-vector-impl*: *LLL-Impl.short-vector*  $\alpha$  *fs-init* = *short-vector*  
*<proof>*

**lemma** *short-vector*: **assumes** *res*: *LLL-Impl.short-vector*  $\alpha$  *fs-init* = *v*  
**and** *m0*: *m*  $\neq$  0  
**shows**  
*v*  $\in$  *carrier-vec n*  
*v*  $\in$  *L* - {*0<sub>v</sub> n*}  
*h*  $\in$  *L* - {*0<sub>v</sub> n*}  $\implies$  *rat-of-int (sq-norm v)*  $\leq$   $\alpha$   $^{\wedge}$  (*m* - 1) \* *rat-of-int (sq-norm h)*  
*v*  $\neq$  *0<sub>v</sub> j*  
*<proof>*

**end**  
**end**

## 9.4 Bound on Number of Arithmetic Operations for Integer Implementation

In this section we define a version of the LLL algorithm which explicitly returns the costs of running the algorithm. Its soundness is mainly proven by stating that projecting away yields the original result.

The cost model counts the number of arithmetic operations that occur in vector-addition, scalar-products, and scalar multiplication and we prove a polynomial bound on this number.

**theory** *LLL-Complexity*  
**imports**  
*LLL-Impl*  
*Cost*  
*HOL-Library.Discrete-Functions*  
**begin**

**definition** *round-num-denom-cost* :: *int*  $\Rightarrow$  *int*  $\Rightarrow$  *int cost* **where**  
*round-num-denom-cost n d* =  $((2 * n + d) \text{ div } (2 * d), 4)$  — 4 arith. operations

**lemma** *round-num-denom-cost*:

**shows** *result* (*round-num-denom-cost*  $n$   $d$ ) = *round-num-denom*  $n$   $d$   
*cost* (*round-num-denom-cost*  $n$   $d$ )  $\leq 4$   
 ⟨*proof*⟩

**context** *LLL-with-assms*  
**begin**

**context**

**assumes**  $\alpha$ -gt:  $\alpha > 4/3$  **and**  $m0$ :  $m \neq 0$   
**begin**

**fun** *basis-reduction-add-rows-loop-cost* **where**

*basis-reduction-add-rows-loop-cost*  $state$   $i$   $j$  [] = (*state*, 0)  
 | *basis-reduction-add-rows-loop-cost*  $state$   $i$   $sj$  ( $fj$  #  $fjs$ ) = (  
   *let*  $fi$  = *fi-state*  $state$ ;  
    $dsj$  = *d-state*  $state$   $sj$ ;  
    $j$  =  $sj - 1$ ;  
   ( $c, cost1$ ) = *round-num-denom-cost* (*dmu-ij-state*  $state$   $i$   $j$ )  $dsj$ ;  
    $state'$  = (*if*  $c = 0$  *then*  $state$  *else* *upd-fi-mu-state*  $state$   $i$  (*vec*  $n$  ( $\lambda$   $i$ .  $fi$  \$  $i$   
 -  $c * fj$  \$  $i$ )) — 2n arith. operations  
   (*IArray.of-fun* ( $\lambda$   $jj$ . *let*  $mu$  = *dmu-ij-state*  $state$   $i$   $jj$  *in* — 3  $sj$  arith.  
 operations  
     *if*  $jj < j$  *then*  $mu - c * dmu-ij-state$   $state$   $j$   $jj$  *else*  
     *if*  $jj = j$  *then*  $mu - dsj * c$  *else*  $mu$ )  $i$ );  
    $local-cost$  =  $2 * n + 3 * sj$ ;  
   ( $res, cost2$ ) = *basis-reduction-add-rows-loop-cost*  $state'$   $i$   $j$   $fjs$   
   *in* ( $res, cost1 + local-cost + cost2$ ))

**lemma** *basis-reduction-add-rows-loop-cost*: **assumes**  $length$   $fs = j$

**shows** *result* (*basis-reduction-add-rows-loop-cost*  $state$   $i$   $j$   $fs$ ) = *LLL-Impl.basis-reduction-add-rows-loop*  
*n*  $state$   $i$   $j$   $fs$   
*cost* (*basis-reduction-add-rows-loop-cost*  $state$   $i$   $j$   $fs$ )  $\leq sum$  ( $\lambda$   $j$ . ( $2 * n + 4 +$   
 $3 * (Suc\ j)$ )) { $0..<j$ }  
 ⟨*proof*⟩

**definition** *basis-reduction-add-rows-cost* **where**

*basis-reduction-add-rows-cost*  $upw$   $i$   $state$  =  
 (*if*  $upw$  *then* *basis-reduction-add-rows-loop-cost*  $state$   $i$   $i$  (*small-fs-state*  $state$ )  
*else* ( $state, 0$ ))

**lemma** *basis-reduction-add-rows-cost*: **assumes** *impl*: *LLL-impl-inv*  $state$   $i$   $fs$  **and**  
*inv*: *LLL-invariant*  $upw$   $i$   $fs$

**shows** *result* (*basis-reduction-add-rows-cost*  $upw$   $i$   $state$ ) = *LLL-Impl.basis-reduction-add-rows*  
*n*  $upw$   $i$   $state$   
*cost* (*basis-reduction-add-rows-cost*  $upw$   $i$   $state$ )  $\leq (2 * n + 2 * i + 7) * i$   
 ⟨*proof*⟩

**definition** *swap-mu-cost* :: int iarray iarray ⇒ nat ⇒ int ⇒ int ⇒ int ⇒ int ⇒ int ⇒ int iarray iarray cost **where**

*swap-mu-cost* *dmu i dmu-i-im1 dim1 di dsi* = (let *im1* = *i* - 1;  
*res* = IArray.of-fun (λ *ii*. if *ii* < *im1* then *dmu* !! *ii* else  
if *ii* > *i* then let *dmu-ii* = *dmu* !! *ii* in  
IArray.of-fun (λ *j*. let *dmu-ii-j* = *dmu-ii* !! *j* in — 8 arith. operations  
for whole line  
if *j* = *i* then (*dsi* \* *dmu-ii* !! *im1* - *dmu-i-im1* \* *dmu-ii-j*) div *di* —  
4 arith. operations for this entry  
else if *j* = *im1* then (*dmu-i-im1* \* *dmu-ii-j* + *dmu-ii* !! *i* \* *dim1*) div  
*di* — 4 arith. operations for this entry  
else *dmu-ii-j*) *ii* else  
if *ii* = *i* then let *mu-im1* = *dmu* !! *im1* in  
IArray.of-fun (λ *j*. if *j* = *im1* then *dmu-i-im1* else *mu-im1* !! *j*) *ii*  
else IArray.of-fun (λ *j*. *dmu* !! *i* !! *j*) *ii*) — *ii* = *i* - 1  
*m*; — in total, there are *m* - (*i*+1) many lines that require arithmetic  
operations: *i* + 1, ..., *m* - 1  
*cost* = 8 \* (*m* - Suc *i*)  
in (*res*,*cost*))

**lemma** *swap-mu-cost*:

*result* (*swap-mu-cost* *dmu i dmu-i-im1 dim1 di dsi*) = *swap-mu* *m dmu i dmu-i-im1 dim1 di dsi*  
*cost* (*swap-mu-cost* *dmu i dmu-i-im1 dim1 di dsi*) ≤ 8 \* (*m* - Suc *i*)  
⟨*proof*⟩

**definition** *basis-reduction-swap-cost* **where**

*basis-reduction-swap-cost* *i state* = (let  
*di* = *d-state* *state i*;  
*dsi* = *d-state* *state* (Suc *i*);  
*dim1* = *d-state* *state* (*i* - 1);  
*fi* = *fi-state* *state*;  
*fm1* = *fm1-state* *state*;  
*dmu-i-im1* = *dmu-ij-state* *state i* (*i* - 1);  
*fi'* = *fm1*;  
*fm1'* = *fi*;  
*di'* = (*dsi* \* *dim1* + *dmu-i-im1* \* *dmu-i-im1*) div *di*; — 4 arith. operations  
*local-cost* = 4  
in (case *state* of (*f*,*dmus*,*djs*) ⇒  
case *swap-mu-cost* *dmus i dmu-i-im1 dim1 di dsi* of  
(*swap-res*, *swap-cost*) ⇒  
let *res* = (False, *i* - 1,  
(dec-*i* (*update-im1* (*update-i f fi'*) *fm1'*),  
*swap-res*,  
*iarray-update djs i di'*));  
*cost* = *local-cost* + *swap-cost*  
in (*res*, *cost*)))

**lemma** *basis-reduction-swap-cost*:

*result* (*basis-reduction-swap-cost* *i* *state*) = *LLL-Impl.basis-reduction-swap* *m* *i* *state*  
*cost* (*basis-reduction-swap-cost* *i* *state*)  $\leq 8 * (m - \text{Suc } i) + 4$   
 ⟨*proof*⟩

**definition** *basis-reduction-step-cost* **where**

*basis-reduction-step-cost* *upw* *i* *state* = (if *i* = 0 then ((*True*, *Suc* *i*, *inc-state* *state*), 0)  
 else let  
   (*state'*, *cost-add*) = *basis-reduction-add-rows-cost* *upw* *i* *state*;  
   *di* = *d-state* *state'* *i*;  
   *dsi* = *d-state* *state'* (*Suc* *i*);  
   *dim1* = *d-state* *state'* (*i* - 1);  
   (*num*, *denom*) = *quotient-of*  $\alpha$ ;  
   *cond* = (*di* \* *di* \* *denom*  $\leq$  *num* \* *dim1* \* *dsi*); — 5 arith. operations  
   *local-cost* = 5  
 in if *cond* then  
   ((*True*, *Suc* *i*, *inc-state* *state'*), *local-cost* + *cost-add*)  
   else case *basis-reduction-swap-cost* *i* *state'* of (*res*, *cost-swap*)  $\Rightarrow$  (*res*, *local-cost* + *cost-swap* + *cost-add*))

**definition** *body-cost* =  $2 + (8 + 2 * n + 2 * m) * m$

**lemma** *basis-reduction-step-cost*: **assumes**

*impl*: *LLL-impl-inv* *state* *i* *fs*  
**and** *inv*: *LLL-invariant* *upw* *i* *fs*  
**and** *i*: *i* < *m*  
**shows** *result* (*basis-reduction-step-cost* *upw* *i* *state*) = *LLL-Impl.basis-reduction-step*  $\alpha$  *n* *m* *upw* *i* *state* (**is** ?*g1*)  
*cost* (*basis-reduction-step-cost* *upw* *i* *state*)  $\leq$  *body-cost* (**is** ?*g2*)  
 ⟨*proof*⟩

**partial-function** (*tailrec*) *basis-reduction-main-cost* **where**

*basis-reduction-main-cost* *upw* *i* *state* *c* = (if *i* < *m*  
 then let ((*upw'*, *i'*, *state'*), *c-step*) = *basis-reduction-step-cost* *upw* *i* *state*  
   in *basis-reduction-main-cost* *upw'* *i'* *state'* (*c* + *c-step*)  
 else (*state*, *c*))

**definition** *num-loops* =  $m + 2 * m * m * \text{nat}(\text{ceiling}(\log \text{base}(\text{real } N)))$

**lemma** *basis-reduction-main-cost*: **assumes** *impl*: *LLL-impl-inv* *state* *i* (*fs-state* *state*)

**and** *inv*: *LLL-invariant* *upw* *i* (*fs-state* *state*)  
**and** *state*: *state* = *initial-state* *m* *fs-init*  
**and** *i*: *i* = 0  
**shows** *result* (*basis-reduction-main-cost* *upw* *i* *state* *c*) = *LLL-Impl.basis-reduction-main*  $\alpha$  *n* *m* *upw* *i* *state* (**is** ?*g1*)  
*cost* (*basis-reduction-main-cost* *upw* *i* *state* *c*)  $\leq c + \text{body-cost} * \text{num-loops}$  (**is**

?g2)  
 ⟨proof⟩

**context fixes**

*fs* :: int vec iarray

**begin**

**fun** *sigma-array-cost* **where**

*sigma-array-cost* *dmus dmusi dmusj dll l* = (if *l* = 0 then (*dmusi* !! *l* \* *dmusj* !! *l*, 1)

else let *l1* = *l* - 1; *dll1* = *dmus* !! *l1* !! *l1*;

(*sig*, *cost-rec*) = *sigma-array-cost* *dmus dmusi dmusj dll1 l1*;

*res* = (*dll* \* *sig* + *dmusi* !! *l* \* *dmusj* !! *l*) div *dll1*; — 4 arith. operations

*local-cost* = (4 :: nat)

in

(*res*, *local-cost* + *cost-rec*))

**declare** *sigma-array-cost.simps*[*simp del*]

**lemma** *sigma-array-cost*:

result (*sigma-array-cost* *dmus dmusi dmusj dll l*) = *sigma-array* *dmus dmusi dmusj dll l*

cost (*sigma-array-cost* *dmus dmusi dmusj dll l*) ≤ 4 \* *l* + 1

⟨proof⟩

**function** *dmu-array-row-main-cost* **where**

*dmu-array-row-main-cost* *fi i dmus j* = (if *j* ≥ *i* then (*dmus*, 0)

else let *sj* = *Suc* *j*;

*dmus-i* = *dmus* !! *i*;

*djj* = *dmus* !! *j* !! *j*;

(*sigma*, *cost-sigma*) = *sigma-array-cost* *dmus dmus-i* (*dmus* !! *sj*) *djj j*;

*dmu-ij* = *djj* \* (*fi* · *fs* !! *sj*) - *sigma*; — 2*n* + 2 arith. operations

*dmus'* = *iarray-update* *dmus i* (*iarray-append* *dmus-i* *dmu-ij*);

(*res*, *cost-rec*) = *dmu-array-row-main-cost* *fi i dmus' sj*;

*local-cost* = 2 \* *n* + 2

in (*res*, *cost-rec* + *cost-sigma* + *local-cost*))

⟨proof⟩

**termination** ⟨proof⟩

**declare** *dmu-array-row-main-cost.simps*[*simp del*]

**lemma** *dmu-array-row-main-cost*: **assumes** *j* ≤ *i*

**shows** result (*dmu-array-row-main-cost* *fi i dmus j*) = *dmu-array-row-main* *fs fi i dmus j*

cost (*dmu-array-row-main-cost* *fi i dmus j*) ≤ (∑ *jj* ∈ {*j* ..< *i*}. 2 \* *n* + 2 + 4 \* *jj* + 1)

⟨proof⟩

**definition** *dmu-array-row-cost* **where**

*dmu-array-row-cost* *dmus* *i* = (let *fi* = *fs* !! *i*;  
*sp* = *fi* · *fs* !! 0 — 2*n* arith. operations;  
*local-cost* = 2 \* *n*;  
(*res*, *main-cost*) = *dmu-array-row-main-cost* *fi* *i* (*iarray-append* *dmus* (*IArray*  
[*sp*])) 0 in  
(*res*, *local-cost* + *main-cost*))

**lemma** *dmu-array-row-cost*:

*result* (*dmu-array-row-cost* *dmus* *i*) = *dmu-array-row* *fs* *dmus* *i*  
*cost* (*dmu-array-row-cost* *dmus* *i*) ≤ 2 \* *n* + (2 \* *n* + 1 + 2 \* *i*) \* *i*  
⟨*proof*⟩

**function** *dmu-array-cost* **where**

*dmu-array-cost* *dmus* *i* = (if *i* ≥ *m* then (*dmus*, 0) else  
let (*dmus'*, *cost-row*) = *dmu-array-row-cost* *dmus* *i*;  
(*res*, *cost-rec*) = *dmu-array-cost* *dmus'* (*Suc* *i*)  
in (*res*, *cost-row* + *cost-rec*)  
⟨*proof*⟩

**termination** ⟨*proof*⟩

**declare** *dmu-array-cost.simps*[*simp del*]

**lemma** *dmu-array-cost*: **assumes** *i* ≤ *m*

**shows** *result* (*dmu-array-cost* *dmus* *i*) = *dmu-array* *fs* *m* *dmus* *i*  
*cost* (*dmu-array-cost* *dmus* *i*) ≤ (∑ *ii* ∈ {*i* ..< *m*}. 2 \* *n* + (2 \* *n* + 1 + 2 \*  
*ii*) \* *ii*)  
⟨*proof*⟩  
**end**

**definition** *dμ-impl-cost* :: *int* *vec* *list* ⇒ *int* *iarray* *iarray* *cost* **where**

*dμ-impl-cost* *fs* = *dmu-array-cost* (*IArray* *fs*) (*IArray* []) 0

**lemma** *dμ-impl-cost*: *result* (*dμ-impl-cost* *fs-init*) = *dμ-impl* *fs-init*

*cost* (*dμ-impl-cost* *fs-init*) ≤ *m* \* (*m* \* (*m* + *n* + 2) + 2 \* *n* + 1)  
⟨*proof*⟩

**definition** *initial-gso-cost* = *m* \* (*m* \* (*m* + *n* + 2) + 2 \* *n* + 1)

**definition** *initial-state-cost* *fs* = (let

(*dmus*, *cost*) = *dμ-impl-cost* *fs*;  
*ds* = *IArray.of-fun* (λ *i*. if *i* = 0 then 1 else let *i1* = *i* - 1 in *dmus* !! *i1* !! *i1*)  
(*Suc* *m*);  
*dmus'* = *IArray.of-fun* (λ *i*. let *row-i* = *dmus* !! *i* in  
*IArray.of-fun* (λ *j*. *row-i* !! *j*) *i*) *m*  
in ((([], *fs*), *dmus'*, *ds*), *cost*) :: *LLL-dmu-d-state* *cost*)

**definition** *basis-reduction-cost* :: -  $\Rightarrow$  LLL-dmu-d-state cost **where**  
*basis-reduction-cost* fs = (  
 case *initial-state-cost* fs of (state1, c1)  $\Rightarrow$   
 case *basis-reduction-main-cost* True 0 state1 0 of (state2, c2)  $\Rightarrow$   
 (state2, c1 + c2))

**definition** *reduce-basis-cost* :: -  $\Rightarrow$  int vec list cost **where**  
*reduce-basis-cost* fs = (case fs of Nil  $\Rightarrow$  (fs, 0) | Cons f -  $\Rightarrow$   
 case *basis-reduction-cost* fs of (state,c)  $\Rightarrow$   
 (fs-state state, c))

**lemma** *initial-state-cost*: result (*initial-state-cost* fs-init) = *initial-state* m fs-init  
 (is ?g1)  
 cost (*initial-state-cost* fs-init)  $\leq$  *initial-gso-cost* (is ?g2)  
 <proof>

**lemma** *basis-reduction-cost*:  
 result (*basis-reduction-cost* fs-init) = *basis-reduction*  $\alpha$  n fs-init (is ?g1)  
 cost (*basis-reduction-cost* fs-init)  $\leq$  *initial-gso-cost* + *body-cost* \* *num-loops* (is  
 ?g2)  
 <proof>

The lemma for the LLL algorithm with explicit cost annotations *reduce-basis-cost* shows that the termination measure indeed gives rise to an explicit cost bound. Moreover, the computed result is the same as in the non-cost counting *local.reduce-basis*.

**lemma** *reduce-basis-cost*:  
 result (*reduce-basis-cost* fs-init) = LLL-Impl.*reduce-basis*  $\alpha$  fs-init (is ?g1)  
 cost (*reduce-basis-cost* fs-init)  $\leq$  *initial-gso-cost* + *body-cost* \* *num-loops* (is ?g2)  
 <proof>

**lemma** *mn*:  $m \leq n$   
 <proof>

Theorem with expanded costs:  $O(n \cdot m^3 \cdot \log(\maxnorm F))$  arithmetic operations

**lemma** *reduce-basis-cost-expanded*:  
**assumes**  $Lg \geq \text{nat } \lceil \log(\text{of-rat } (4 * \alpha / (4 + \alpha))) N \rceil$   
**shows** cost (*reduce-basis-cost* fs-init)  
 $\leq 4 * Lg * m * m * m * n$   
 $+ 4 * Lg * m * m * m * m$   
 $+ 16 * Lg * m * m * m$   
 $+ 4 * Lg * m * m$   
 $+ 3 * m * m * m$   
 $+ 3 * m * m * n$   
 $+ 10 * m * m$   
 $+ 2 * n * m$   
 $+ 3 * m$   
 (is ?cost  $\leq$  ?exp Lg)

*<proof>*

**lemma** *reduce-basis-cost-0*: **assumes**  $m = 0$   
**shows**  $\text{cost } (\text{reduce-basis-cost } fs\text{-init}) = 0$   
*<proof>*

**lemma** *reduce-basis-cost-N*:  
**assumes**  $Lg \geq \text{nat } \lceil \log (\text{of-rat } (4 * \alpha / (4 + \alpha))) N \rceil$   
**and**  $0: Lg > 0$   
**shows**  $\text{cost } (\text{reduce-basis-cost } fs\text{-init}) \leq 49 * m ^ 3 * n * Lg$   
*<proof>*

**lemma** *reduce-basis-cost-M*:  
**assumes**  $Lg \geq \text{nat } \lceil \log (\text{of-rat } (4 * \alpha / (4 + \alpha))) (M * n) \rceil$   
**and**  $0: Lg > 0$   
**shows**  $\text{cost } (\text{reduce-basis-cost } fs\text{-init}) \leq 98 * m ^ 3 * n * Lg$   
*<proof>*

**end**  
**end**  
**end**

## 9.5 Explicit Bounds for Size of Numbers that Occur During LLL Algorithm

The LLL invariant does not contain bounds on the number that occur during the execution. We here strengthen the invariant so that it enforces bounds on the norms of the  $f_i$  and  $g_i$  and we prove that the stronger invariant is maintained throughout the execution of the LLL algorithm.

Based on the stronger invariant we prove bounds on the absolute values of the  $\mu_{i,j}$ , and on the absolute values of the numbers in the vectors  $f_i$  and  $g_i$ . Moreover, we further show that also the denominators in all of these numbers doesn't grow to much. Finally, we prove that each number (i.e., numerator or denominator) during the execution can be represented with at most  $\mathcal{O}(m \cdot \log(M \cdot n))$  bits, where  $m$  is the number of input vectors,  $n$  is the dimension of the input vectors, and  $M$  is the maximum absolute value of all numbers in the input vectors. Hence, each arithmetic operation in the LLL algorithm can be performed in polynomial time.

**theory** *LLL-Number-Bounds*  
**imports** *LLL*  
*Gram-Schmidt-Int*  
**begin**

**context** *LLL*

**begin**

The bounds for the  $f_i$  distinguishes whether we are inside or outside the inner for-loop.

**definition**  $f\text{-bound} :: \text{bool} \Rightarrow \text{nat} \Rightarrow \text{int vec list} \Rightarrow \text{bool}$  **where**

$f\text{-bound outside } ii \text{ fs} = (\forall i < m. \text{sq-norm } (fs ! i) \leq (\text{if } i \neq ii \vee \text{outside then int } (N * m) \text{ else } \text{int } (4 \wedge (m - 1) * N \wedge m * m * m)))$

**definition**  $g\text{-bnd} :: \text{rat} \Rightarrow \text{int vec list} \Rightarrow \text{bool}$  **where**

$g\text{-bnd } B \text{ fs} = (\forall i < m. \text{sq-norm } (gso \text{ fs } i) \leq B)$

**definition**  $\mu\text{-bound-row fs bnd } i = (\forall j \leq i. (\mu \text{ fs } i j) \wedge 2 \leq \text{bnd})$

**abbreviation**  $\mu\text{-bound-row-inner fs } i j \equiv \mu\text{-bound-row fs } (4 \wedge (m - 1 - j) * \text{of-nat } (N \wedge (m - 1) * m)) \text{ } i$

**definition**  $LLL\text{-bound-invariant outside upw } i \text{ fs} =$

$(LLL\text{-invariant upw } i \text{ fs} \wedge f\text{-bound outside } i \text{ fs} \wedge g\text{-bound fs})$

**lemma**  $\text{bound-invD}$ : **assumes**  $LLL\text{-bound-invariant outside upw } i \text{ fs}$

**shows**  $LLL\text{-invariant upw } i \text{ fs } f\text{-bound outside } i \text{ fs } g\text{-bound fs}$

$\langle \text{proof} \rangle$

**lemma**  $\text{bound-invI}$ : **assumes**  $LLL\text{-invariant upw } i \text{ fs } f\text{-bound outside } i \text{ fs } g\text{-bound fs}$

**shows**  $LLL\text{-bound-invariant outside upw } i \text{ fs}$

$\langle \text{proof} \rangle$

**lemma**  $\mu\text{-bound-rowI}$ : **assumes**  $\bigwedge j. j \leq i \implies (\mu \text{ fs } i j) \wedge 2 \leq \text{bnd}$

**shows**  $\mu\text{-bound-row fs bnd } i$

$\langle \text{proof} \rangle$

**lemma**  $\mu\text{-bound-rowD}$ : **assumes**  $\mu\text{-bound-row fs bnd } i \text{ } j \leq i$

**shows**  $(\mu \text{ fs } i j) \wedge 2 \leq \text{bnd}$

$\langle \text{proof} \rangle$

**lemma**  $\mu\text{-bound-row-1}$ : **assumes**  $\mu\text{-bound-row fs bnd } i$

**shows**  $\text{bnd} \geq 1$

$\langle \text{proof} \rangle$

**lemma**  $\text{reduced-}\mu\text{-bound-row}$ : **assumes**  $\text{red: reduced fs } i$

**and**  $ii: ii < i$

**shows**  $\mu\text{-bound-row fs } 1 \text{ } ii$

$\langle \text{proof} \rangle$

**lemma**  $f\text{-bound-True-arbitrary}$ : **assumes**  $f\text{-bound True } ii \text{ fs}$

**shows**  $f\text{-bound outside } j \text{ fs}$

$\langle \text{proof} \rangle$

**context** fixes  $fs :: \text{int vec list}$   
**assumes**  $\text{lin-indep}: \text{lin-indep } fs$   
**and**  $\text{len}: \text{length } fs = m$   
**begin**

**interpretation**  $fs: \text{fs-int-indpt } n \text{ fs}$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{sq-norm-fs-mu-g-bound}$ : **assumes**  $i: i < m$   
**and**  $\text{mu-bound}: \mu\text{-bound-row } fs \text{ bnd } i$   
**and**  $\text{g-bound}: \text{g-bound } fs$   
**shows**  $\text{of-int } \|fs ! i\|^2 \leq \text{of-nat } (\text{Suc } i * N) * \text{bnd}$   
 $\langle \text{proof} \rangle$   
**end**

**lemma**  $\text{increase-i-bound}$ : **assumes**  $\text{LLL}: \text{LLL-bound-invariant } \text{True } \text{upw } i \text{ fs}$   
**and**  $i: i < m$   
**and**  $\text{upw}: \text{upw} \implies i = 0$   
**and**  $\text{red-i}: i \neq 0 \implies \text{sq-norm } (\text{gso } fs \ (i - 1)) \leq \alpha * \text{sq-norm } (\text{gso } fs \ i)$   
**shows**  $\text{LLL-bound-invariant } \text{True } \text{True } (\text{Suc } i) \text{ fs}$   
 $\langle \text{proof} \rangle$

Addition step preserves  $\text{LLL-bound-invariant } \text{False}$

**lemma**  $\text{basis-reduction-add-row-main-bound}$ : **assumes**  $\text{Linv}: \text{LLL-bound-invariant } \text{False } \text{True } i \text{ fs}$   
**and**  $i: i < m$  **and**  $j: j < i$   
**and**  $c: c = \text{round } (\mu \text{ fs } i \ j)$   
**and**  $fs'$ :  $fs' = fs[ \ i := fs ! i - c \cdot_v \ fs ! j ]$   
**and**  $\text{mu-small}: \mu\text{-small-row } i \text{ fs } (\text{Suc } j)$   
**and**  $\text{mu-bnd}: \mu\text{-bound-row-inner } fs \ i \ (\text{Suc } j)$   
**shows**  $\text{LLL-bound-invariant } \text{False } \text{True } i \text{ fs}'$   
 $\mu\text{-bound-row-inner } fs' \ i \ j$   
 $\langle \text{proof} \rangle$   
**end**

**context**  $\text{LLL-with-assms}$   
**begin**

### 9.5.1 $\text{LLL-bound-invariant}$ is maintained during execution of $\text{reduce-basis}$

**lemma**  $\text{basis-reduction-add-rows-enter-bound}$ : **assumes**  $\text{binv}: \text{LLL-bound-invariant } \text{True } \text{True } i \text{ fs}$   
**and**  $i: i < m$   
**shows**  $\text{LLL-bound-invariant } \text{False } \text{True } i \text{ fs}$   
 $\mu\text{-bound-row-inner } fs \ i \ i$   
 $\langle \text{proof} \rangle$

**lemma** *basis-basis-reduction-add-rows-loop-leave*:  
**assumes** *binv*: *LLL-bound-invariant False True i fs*  
**and** *mu-small*:  $\mu$ -small-row *i fs 0*  
**and** *mu-bnd*:  $\mu$ -bound-row-inner *fs i 0*  
**and** *i*:  $i < m$   
**shows** *LLL-bound-invariant True False i fs*  
*<proof>*

**lemma** *basis-reduction-add-rows-loop-bound*: **assumes**  
*binv*: *LLL-bound-invariant False True i fs*  
**and** *mu-small*:  $\mu$ -small-row *i fs j*  
**and** *mu-bnd*:  $\mu$ -bound-row-inner *fs i j*  
**and** *res*: *basis-reduction-add-rows-loop i fs j = fs'*  
**and** *i*:  $i < m$   
**and** *j*:  $j \leq i$   
**shows** *LLL-bound-invariant True False i fs'*  
*<proof>*

**lemma** *basis-reduction-add-rows-bound*: **assumes**  
*binv*: *LLL-bound-invariant True upw i fs*  
**and** *res*: *basis-reduction-add-rows upw i fs = fs'*  
**and** *i*:  $i < m$   
**shows** *LLL-bound-invariant True False i fs'*  
*<proof>*

**lemma** *g-bnd-swap*:  
**assumes** *i*:  $i < m$   $i \neq 0$   
**and** *Linv*: *LLL-invariant-weak fs*  
**and** *mu-F1-i*:  $|\mu fs i (i-1)| \leq 1 / 2$   
**and** *cond*:  $sq\text{-norm } (gso fs (i-1)) > \alpha * sq\text{-norm } (gso fs i)$   
**and** *fs'-def*:  $fs' = fs[i := fs ! (i-1), i-1 := fs ! i]$   
**and** *g-bnd*: *g-bnd B fs*  
**shows** *g-bnd B fs'*  
*<proof>*

**lemma** *basis-reduction-swap-bound*: **assumes**  
*binv*: *LLL-bound-invariant True False i fs*  
**and** *res*: *basis-reduction-swap i fs = (upw',i',fs')*  
**and** *cond*:  $sq\text{-norm } (gso fs (i-1)) > \alpha * sq\text{-norm } (gso fs i)$   
**and** *i*:  $i < m$   $i \neq 0$   
**shows** *LLL-bound-invariant True upw' i' fs'*  
*<proof>*

**lemma** *basis-reduction-step-bound*: **assumes**  
*binv*: *LLL-bound-invariant True upw i fs*  
**and** *res*: *basis-reduction-step upw i fs = (upw',i',fs')*

**and**  $i: i < m$   
**shows**  $LLL\text{-bound-invariant True upw' i' fs'}$   
 $\langle proof \rangle$

**lemma**  $basis\text{-reduction-main-bound}$ : **assumes**  $LLL\text{-bound-invariant True upw i fs}$

**and**  $res: basis\text{-reduction-main (upw,i,fs)} = fs'$   
**shows**  $LLL\text{-bound-invariant True True m fs'}$   
 $\langle proof \rangle$

**lemma**  $LLL\text{-inv-initial-state-bound}$ :  $LLL\text{-bound-invariant True True 0 fs-init}$   
 $\langle proof \rangle$

**lemma**  $reduce\text{-basis-bound}$ : **assumes**  $res: reduce\text{-basis} = fs$   
**shows**  $LLL\text{-bound-invariant True True m fs}$   
 $\langle proof \rangle$

### 9.5.2 Bound extracted from $LLL\text{-bound-invariant}$ .

**fun**  $f\text{-bnd} :: bool \Rightarrow nat$  **where**  
 $f\text{-bnd False} = 2 \wedge (m - 1) * N \wedge m * m$   
 $| f\text{-bnd True} = N * m$

**lemma**  $f\text{-bnd-mono}$ :  $f\text{-bnd outside} \leq f\text{-bnd False}$   
 $\langle proof \rangle$

**lemma**  $aux\text{-bnd-mono}$ :  $N * m \leq (4 \wedge (m - 1) * N \wedge m * m * m)$   
 $\langle proof \rangle$

**context**  $fixes outside upw k fs$   
**assumes**  $binv: LLL\text{-bound-invariant outside upw k fs}$   
**begin**

**lemma**  $LLL\text{-f-bnd}$ :  
**assumes**  $i: i < m$  **and**  $j: j < n$   
**shows**  $|fs ! i \$ j| \leq f\text{-bnd outside}$   
 $\langle proof \rangle$

**lemma**  $LLL\text{-gso-bound}$ :  
**assumes**  $i: i < m$  **and**  $j: j < n$   
**and**  $quot: quotient\text{-of (gso fs i \$ j)} = (num, denom)$   
**shows**  $|num| \leq N \wedge m$   
**and**  $|denom| \leq N \wedge m$   
 $\langle proof \rangle$

**lemma**  $LLL\text{-f-bound}$ :  
**assumes**  $i: i < m$  **and**  $j: j < n$   
**shows**  $|fs ! i \$ j| \leq N \wedge m * 2 \wedge (m - 1) * m$   
 $\langle proof \rangle$

**lemma** *LLL-d-bound*:

**assumes**  $i: i \leq m$

**shows**  $\text{abs } (d \text{ fs } i) \leq N^{\wedge i} \wedge \text{abs } (d \text{ fs } i) \leq N^{\wedge m}$

*<proof>*

**lemma** *LLL-mu-abs-bound*:

**assumes**  $i: i < m$

**and**  $j: j < i$

**shows**  $|\mu \text{ fs } i \ j| \leq \text{rat-of-nat } (N^{\wedge (m-1)} * 2^{\wedge (m-1)} * m)$

*<proof>*

**lemma** *LLL-dmu-bound*:

**assumes**  $i: i < m$  **and**  $j: j < i$

**shows**  $\text{abs } (d\mu \text{ fs } i \ j) \leq N^{\wedge (2 * (m-1))} * 2^{\wedge (m-1)} * m$

*<proof>*

**lemma** *LLL-mu-num-denom-bound*:

**assumes**  $i: i < m$

**and** *quot*: *quotient-of*  $(\mu \text{ fs } i \ j) = (\text{num}, \text{denom})$

**shows**  $|\text{num}| \leq N^{\wedge (2 * m)} * 2^{\wedge m} * m$

**and**  $|\text{denom}| \leq N^{\wedge m}$

*<proof>*

Now we have bounds on each number  $(f_i)_j$ ,  $(g_i)_j$ , and  $\mu_{i,j}$ , i.e., for rational numbers bounds on the numerators and denominators.

**lemma** *logN-le-2log-Mn*: **assumes**  $m: m \neq 0$   $n: n \neq 0$  **and**  $N: N > 0$

**shows**  $\log 2 \ N \leq 2 * \log 2 \ (M * n)$

*<proof>*

We now prove a combined size-bound for all of these numbers. The bounds clearly indicate that the size of the numbers grows at most polynomial, namely the sizes are roughly bounded by  $\mathcal{O}(m \cdot \log(M \cdot n))$  where  $m$  is the number of vectors,  $n$  is the dimension of the vectors, and  $M$  is the maximum absolute value that occurs in the input to the LLL algorithm.

**lemma** *combined-size-bound*: **fixes**  $\text{number} :: \text{int}$

**assumes**  $i: i < m$  **and**  $j: j < n$

**and**  $x: x \in \{\text{of-int } (f \ ! \ i \ \$ \ j), \text{gso } f \ i \ \$ \ j, \mu \text{ fs } i \ j\}$

**and** *quot*: *quotient-of*  $x = (\text{num}, \text{denom})$

**and**  $\text{number}: \text{number} \in \{\text{num}, \text{denom}\}$

**and**  $\text{number0}: \text{number} \neq 0$

**shows**  $\log 2 \ |\text{number}| \leq 2 * m * \log 2 \ N \quad + \ m + \log 2 \ m$

$\log 2 \ |\text{number}| \leq 4 * m * \log 2 \ (M * n) + m + \log 2 \ m$

*<proof>*

And a combined size bound for an integer implementation which stores values  $f_i$ ,  $d_{j+1}\mu_{ij}$  and  $d_i$ .

**interpretation** *fs: fs-int-indpt n fs-init*  
 ⟨proof⟩

**lemma** *fs-gs-N-N'*: **assumes**  $m \neq 0$   
**shows**  $fs.gs.N = of-nat\ N$   
 ⟨proof⟩

**lemma** *fs-gs-N-N*:  $m \neq 0 \implies real-of-rat\ fs.gs.N = real\ N$   
 ⟨proof⟩

**lemma** *combined-size-bound-gso-integer*:  
**assumes**  $x \in$   
 $\{fs.\mu' i j \mid i j. j \leq i \wedge i < m\} \cup$   
 $\{fs.\sigma s l i j \mid i j l. i < m \wedge j \leq i \wedge l < j\}$   
**and**  $m: m \neq 0$  **and**  $x \neq 0$   $n \neq 0$   
**shows**  $\log 2 \mid real-of-int\ x \mid \leq (6 + 6 * m) * \log 2 (M * n) + \log 2\ m + m$   
 ⟨proof⟩

**lemma** *combined-size-bound-integer'*:  
**assumes**  $x: x \in \{fs ! i \$ j \mid i j. i < m \wedge j < n\}$   
 $\cup \{d\mu fs i j \mid i j. j < i \wedge i < m\}$   
 $\cup \{d fs i \mid i. i \leq m\}$   
**(is**  $x \in ?fs \cup ?d\mu \cup ?d$   
**and**  $m: m \neq 0$  **and**  $n: n \neq 0$   
**shows**  $abs\ x \leq N \wedge (2 * m) * 2 \wedge m * m$   
 $x \neq 0 \implies \log 2 \mid x \mid \leq 2 * m * \log 2\ N + m + \log 2\ m$  **(is**  $- \implies ?l1 \leq ?b1$ )  
 $x \neq 0 \implies \log 2 \mid x \mid \leq 4 * m * \log 2 (M * n) + m + \log 2\ m$  **(is**  $- \implies - \leq ?b2$ )  
 ⟨proof⟩

**lemma** *combined-size-bound-integer*:  
**assumes**  $x: x \in$   
 $\{fs ! i \$ j \mid i j. i < m \wedge j < n\}$   
 $\cup \{d\mu fs i j \mid i j. j < i \wedge i < m\}$   
 $\cup \{d fs i \mid i. i \leq m\}$   
 $\cup \{fs.\mu' i j \mid i j. j \leq i \wedge i < m\}$   
 $\cup \{fs.\sigma s l i j \mid i j l. i < m \wedge j \leq i \wedge l < j\}$   
**(is**  $?x \in ?s1 \cup ?s2 \cup ?s3 \cup ?g1 \cup ?g2$   
**and**  $m: m \neq 0$  **and**  $n: n \neq 0$  **and**  $x \neq 0$  **and**  $0 < M$   
**shows**  $\log 2 \mid x \mid \leq (6 + 6 * m) * \log 2 (M * n) + \log 2\ m + m$   
 ⟨proof⟩

**end**  
**end**  
**end**

## 10 Certification of External LLL Invocations

Instead of using a fully verified algorithm, we also provide a technique to invoke an external LLL solver. In order to check its result, we not only need the reduced basis, but also the matrices which translate between the input basis and the reduced basis. Then we can easily check whether the resulting lattices are indeed identical and just have to start the verified algorithm on the already reduced basis. This invocation will then usually just require one computation of Gram–Schmidt in order to check that the basis is already reduced. Alternatively, one could also throw an error message in case the basis is not reduced.

### 10.1 Checking Results of External LLL Solvers

**theory** *LLL-Certification*

**imports**

*LLL-Impl*

*Jordan-Normal-Form.Show-Matrix*

**begin**

**definition** *gauss-jordan-integer-inverse*  $n A B I = (\text{case gauss-jordan } A B \text{ of } (C, D) \Rightarrow C = I \wedge \text{list-all is-int-rat } (\text{concat } (\text{mat-to-list } D)))$

**definition** *integer-equivalent*  $n fs gs = (\text{let } fs' = \text{map-mat rat-of-int } (\text{mat-of-cols } n fs); gs' = \text{map-mat rat-of-int } (\text{mat-of-cols } n gs); I = 1_m n \text{ in gauss-jordan-integer-inverse } n fs' gs' I \wedge \text{gauss-jordan-integer-inverse } n gs' fs' I)$

**context** *vec-module*

**begin**

**lemma** *mat-mult-sub-lattice*: **assumes**  $fs$ : set  $fs \subseteq \text{carrier-vec } n$   
**and**  $gs$ : set  $gs \subseteq \text{carrier-vec } n$   
**and**  $A$ :  $A \in \text{carrier-mat } (\text{length } fs) (\text{length } gs)$   
**and**  $prod$ :  $\text{mat-of-rows } n fs = \text{map-mat of-int } A * \text{mat-of-rows } n gs$   
**shows**  $\text{lattice-of } fs \subseteq \text{lattice-of } gs$   
 $\langle \text{proof} \rangle$   
**end**

**context** *LLL-with-assms*

**begin**

**lemma** *mult-left-identity*:

**defines**  $B \equiv (\text{map-mat rat-of-int } (\text{mat-of-rows } n fs\text{-init}))$

**assumes**  $P\text{-carrier}[simp]$ :  $P \in \text{carrier-mat } m m$

**and**  $PB: P * B = B$   
**shows**  $P = 1_m \ m$   
 ⟨*proof*⟩

This is the key lemma. It permits to change from the initial basis *fs-init* to an arbitrary *gs* that has been computed by some external tool. Here, two change-of-basis matrices *U1* and *U2* are required to certify the change via the conditions *prod1* and *prod2*.

**lemma** *LLL-change-basis*: **assumes**  $gs: \text{set } gs \subseteq \text{carrier-vec } n$   
**and**  $len': \text{length } gs = m$   
**and**  $U1: U1 \in \text{carrier-mat } m \ m$   
**and**  $U2: U2 \in \text{carrier-mat } m \ m$   
**and**  $prod1: \text{mat-of-rows } n \ \text{fs-init} = U1 * \text{mat-of-rows } n \ gs$   
**and**  $prod2: \text{mat-of-rows } n \ gs = U2 * \text{mat-of-rows } n \ \text{fs-init}$   
**shows**  $\text{lattice-of } gs = \text{lattice-of } \text{fs-init} \ \text{LLL-with-assms } n \ m \ gs \ \alpha$   
 ⟨*proof*⟩

**lemma** *gauss-jordan-integer-inverse*: **fixes**  $fs \ gs :: \text{int vec list}$   
**assumes**  $gs: \text{set } gs \subseteq \text{carrier-vec } n$   
**and**  $len-gs: \text{length } gs = n$   
**and**  $fs: \text{set } fs \subseteq \text{carrier-vec } n$   
**and**  $len-fs: \text{length } fs = n$   
**and**  $gauss: \text{gauss-jordan-integer-inverse } n \ (\text{map-mat } \text{rat-of-int } (\text{mat-of-cols } n \ fs))$   
 $(\text{map-mat } \text{rat-of-int } (\text{mat-of-cols } n \ gs)) \ (1_m \ n) \ (\text{is } \text{gauss-jordan-integer-inverse}$   
 $- \ ?fs \ ?gs \ -)$   
**shows**  $\exists U. U \in \text{carrier-mat } n \ n \wedge \text{mat-of-rows } n \ gs = U * \text{mat-of-rows } n \ fs$   
 ⟨*proof*⟩

**lemma** *LLL-change-basis-mat-inverse*: **assumes**  $gs: \text{set } gs \subseteq \text{carrier-vec } n$   
**and**  $len': \text{length } gs = n$   
**and**  $m = n$   
**and**  $eq: \text{integer-equivalent } n \ \text{fs-init } gs$   
**shows**  $\text{lattice-of } gs = \text{lattice-of } \text{fs-init} \ \text{LLL-with-assms } n \ m \ gs \ \alpha$   
 ⟨*proof*⟩

**end**

External solvers must deliver a reduced basis and optionally two matrices to convert between the input and the reduced basis. These two matrices are mandatory if the input matrix is not a square matrix.

**consts** *external-lll-solver*  $:: \text{integer} \times \text{integer} \Rightarrow \text{integer list list} \Rightarrow$   
 $\text{integer list list} \times (\text{integer list list} \times \text{integer list list}) \text{option}$

**definition** *reduce-basis-external*  $:: \text{rat} \Rightarrow \text{int vec list} \Rightarrow \text{int vec list}$  **where**  
 $\text{reduce-basis-external } \alpha \ fs = (\text{case } fs \ \text{of } \text{Nil} \Rightarrow [] \mid \text{Cons } f \ - \Rightarrow (\text{let}$   
 $\text{rb} = \text{reduce-basis } \alpha;$

```

    fsi = map (map integer-of-int o list-of-vec) fs;
    n = dim-vec f;
    m = length fs in
  case external-lll-solver (map-prod integer-of-int integer-of-int (quotient-of  $\alpha$ )) fsi
of
  (gsi, co)  $\Rightarrow$ 
    let gs = (map (vec-of-list o map int-of-integer) gsi) in
    if  $\neg$  (length gs = m  $\wedge$  ( $\forall$  gi  $\in$  set gs. dim-vec gi = n)) then
      Code.abort (STR "error in external LLL invocation: dimensions of reduced
basis do not fit  $\boxed{\leftarrow}$  input to external solver: "
+ String.implode (show fs) + STR " $\boxed{\leftarrow \leftarrow}$ ") ( $\lambda$  -. rb fs)
    else
      case co of Some (u1i, u2i)  $\Rightarrow$  (let
        u1 = mat-of-rows-list m (map (map int-of-integer) u1i);
        u2 = mat-of-rows-list m (map (map int-of-integer) u2i);
        gs = (map (vec-of-list o map int-of-integer) gsi);
        Fs = mat-of-rows n fs;
        Gs = mat-of-rows n gs in
        if (dim-row u1 = m  $\wedge$  dim-col u1 = m  $\wedge$  dim-row u2 = m  $\wedge$  dim-col u2
= m
           $\wedge$  Fs = u1 * Gs  $\wedge$  Gs = u2 * Fs)
        then rb gs
        else Code.abort (STR "error in external lll invocation  $\boxed{\leftarrow}$  f,g,u1,u2 are as
follows  $\boxed{\leftarrow}$ "
+ String.implode (show Fs) + STR " $\boxed{\leftarrow \leftarrow}$ "
+ String.implode (show Gs) + STR " $\boxed{\leftarrow \leftarrow}$ "
+ String.implode (show u1) + STR " $\boxed{\leftarrow \leftarrow}$ "
+ String.implode (show u2) + STR " $\boxed{\leftarrow \leftarrow}$ "
) ( $\lambda$  -. rb fs))
      | None  $\Rightarrow$  (if (n = m  $\wedge$  integer-equivalent n fs gs) then
        rb gs
        else Code.abort (STR "error in external LLL invocation:  $\boxed{\leftarrow}$ " +
(if n = m then STR "reduced matrix does not span same lattice" else
STR "no certificate only allowed for square matrices")) ( $\lambda$  -. rb fs))
    ))

```

**definition** *short-vector-external* :: rat  $\Rightarrow$  int vec list  $\Rightarrow$  int vec **where**  
*short-vector-external*  $\alpha$  fs = (hd (reduce-basis-external  $\alpha$  fs))

**context** *LLL-with-assms*  
**begin**

**lemma** *reduce-basis-external*: **assumes** res: reduce-basis-external  $\alpha$  fs-init = fs  
**shows** reduced fs m LLL-invariant True m fs

*<proof>*

**lemma** *short-vector-external*: **assumes** res: short-vector-external  $\alpha$  fs-init = v  
**and** m0: m  $\neq$  0

```

shows  $v \in \text{carrier-vec } n$ 
   $v \in L - \{0_v \ n\}$ 
   $h \in L - \{0_v \ n\} \implies \text{rat-of-int } (\text{sq-norm } v) \leq \alpha \wedge (m - 1) * \text{rat-of-int } (\text{sq-norm } h)$ 
   $v \neq 0_v \ j$ 
  <proof>
end

```

Unspecified constant to easily enable/disable external lll solver in generated code

```

consts enable-external-lll-solver :: bool

```

```

definition short-vector-hybrid :: rat  $\Rightarrow$  int vec list  $\Rightarrow$  int vec where
  short-vector-hybrid = (if enable-external-lll-solver then short-vector-external else short-vector)

```

```

definition reduce-basis-hybrid :: rat  $\Rightarrow$  int vec list  $\Rightarrow$  int vec list where
  reduce-basis-hybrid = (if enable-external-lll-solver then reduce-basis-external else reduce-basis)

```

```

context LLL-with-assms

```

```

begin

```

```

lemma short-vector-hybrid: assumes res: short-vector-hybrid  $\alpha$  fs-init = v
  and m0:  $m \neq 0$ 

```

```

shows  $v \in \text{carrier-vec } n$ 
   $v \in L - \{0_v \ n\}$ 
   $h \in L - \{0_v \ n\} \implies \text{rat-of-int } (\text{sq-norm } v) \leq \alpha \wedge (m - 1) * \text{rat-of-int } (\text{sq-norm } h)$ 
   $v \neq 0_v \ j$ 
  <proof>

```

```

lemma reduce-basis-hybrid: assumes res: reduce-basis-hybrid  $\alpha$  fs-init = fs
  shows reduced fs m LLL-invariant True m fs
  <proof>
end

```

```

lemma lll-oracle-default-code[code]:

```

```

  external-lll-solver x = Code.abort (STR "no implementation of external-lll-solver specified") ( $\lambda$  -. external-lll-solver x)
  <proof>

```

By default, external solvers are disabled. For enabling an external solver, load it via a separate theory like `FPLLL_Solver.thy`

```

overloading enable-external-lll-solver  $\equiv$  enable-external-lll-solver

```

```

begin

```

```

  definition enable-external-lll-solver where enable-external-lll-solver = False

```

**end**

**definition** *short-vector-test-hybrid* *xs* =  
 (let *ys* = map (vec-of-list o map int-of-integer) *xs*  
 in integer-of-int (sq-norm (short-vector-hybrid (3/2) *ys*)))

**end**

## 10.2 A Haskell Interface to the FLLL-Solver

**theory** *FLLL-Solver*  
 **imports** *LLL-Certification*  
**begin**

We define *external-lll-solver* via an invocation of the `fpdll` solver. For `eta` we use the default value of `fpdll`, and `delta` is chosen so that the required precision of `alpha` will be guaranteed. We use the command-line option `-bv` in order to get the witnesses that are required for certification.

Warning: Since we only define a Haskell binding for FPLL, the target languages do no longer evaluate to the same results on *short-vector-hybrid!*

**code-printing**

```
code-module FPLL-Solver  $\rightarrow$  (Haskell)  
 $\langle$ module FPLL-Solver where {  
  
  import System.Process (proc,createProcess,waitForProcess,CreateProcess(..),StdStream(..));  
  import System.IO.Unsafe (unsafePerformIO);  
  import System.IO (stderr,hPutStrLn,hPutStr,hClose);  
  import Data.ByteString.Lazy (hPut,hGetContents,intercalate,ByteString);  
  import Data.ByteString.Lazy.Char8 (pack,unpack,uncons,cons);  
  import GHC.IO.Exception (ExitCode(ExitSuccess));  
  import Data.Char (isNumber, isSpace);  
  import GHC.IO.Handle (hSetBinaryMode,hSetBuffering,BufferMode(BlockBuffering));  
  import Control.Exception;  
  import Data.IORef;  
  
  fpdll-command :: String;  
  fpdll-command = fpdll;  
  
  default-eta :: Double;  
  default-eta = 0.51;  
  
  alpha-to-delta :: (Integer,Integer)  $\rightarrow$  Double;  
  alpha-to-delta (num,denom) = (fromIntegral denom / fromIntegral num) +  
    (default-eta * default-eta);  
  
  showrow :: [Integer]  $\rightarrow$  ByteString;
```

```

showrow rowA = (pack [] 'mappend' intercalate (pack ' ') (map (pack . show) rowA)
'mappend' (pack []));
showmat :: [[Integer]] -> ByteString;
showmat matA = (pack [] 'mappend' intercalate (pack '\n ') (map showrow matA)
'mappend' (pack []));

data Mode = Simple | Certificate;

flags :: Mode -> String;
flags Simple = b;
flags Certificate = bv;

getMode xs = (let m = length xs in if m == 0 then Certificate
else if m == length (head xs) then Simple else Certificate);

fpLLL-solver :: (Integer,Integer) -> [[Integer]] -> ([[Integer]], Maybe ([[Integer]],[[Integer]]));
fpLLL-solver alpha in-mat = unsafePerformIO $ catchE $ do {
  (Just f-in,Just f-out,Just f-err,f-pid) <- createProcess (proc fpLLL-command [-e,
show default-eta, -d, show (alpha-to-delta alpha), -of, flags mode]){std-in = CreatePipe,
std-err = CreatePipe, std-out = CreatePipe};
  hSetBinaryMode f-in True;
  hSetBinaryMode f-out True;
  hSetBinaryMode f-err True;
  hSetBuffering f-out (BlockBuffering Nothing);
  hPut f-in (showmat in-mat);
  res <- hGetContents f-out;
  hClose f-in;
  parseRes res}
where {
  mode = getMode in-mat;
  catchE m = catch m def;
  def :: SomeException -> IO ([[Integer]], Maybe ([[Integer]], [[Integer]]));
  def - = seq sendError $ default-answer;
  unconsIO a = case uncons a of{
    Just b -> return b;
    - -> abort Unexpected end of file / input};
  parseMat ('[,as)
  = do {
    (h0,rem0) <- parseSpaces ==<< unconsIO as;
    (rows,(h1,rem1)) <- parseRows (h0,rem0);
    case seq rows h1 of{
      [] -> return (rows,rem1);
      - -> abort$ Expecting closing '[' while parsing a matrix.\n}
    } :: IO ([[Integer]], ByteString);
  parseMat - = abort Expecting opening '[' while parsing a matrix;
  parseRows ('[,rem0)
  = do {
    (nums,(h2,rem2))<-parseNums ==<< parseSpaces ==<< unconsIO rem0;
    case seq nums h2 of

```

```

    ]' -> do { (h4,rem4) <- parseSpaces =<< unconsIO rem2;
              (rows,rem5) <- parseRows (h4,rem4);
              return (nums:rows,rem5) }
  - -> abort$ Expecting closing '[' while parsing a row\n
  } :: IO ([[Integer]],(Char, ByteString));
parseRows r = return ([],r);
parseNums (a,rem0) =
  (if isNumber a || a == '-' then do {
    (n,(h1,rem1)) <- parseNum =<< unconsIO rem0;
    rem2 <- parseSpaces (h1,rem1);
    num <- return (read (a:n));
    (nums,rem3) <- seq (num==num)$ parseNums rem2;
    return (seq nums $ num:nums,rem3) }
  else if isSpace a then do {
    rem1 <- parseSpaces (a,rem0);
    parseNums rem1 }
  else return ([],(a, rem0))) :: IO ([Integer], (Char, ByteString));
parseNum (a,rem0) =
  if isNumber a then do {
    (num,rem1) <- parseNum =<< unconsIO rem0;
    return (a:num,rem1)
  }
  else return (mempty,(a,rem0));
parseSpaces (a,as) = if isSpace a then case uncons as of { Nothing -> return
(a,mempty); Just v -> parseSpaces v } else return (a,as);
parseRes :: ByteString -> IO ([[Integer]], Maybe ([[Integer]], [[Integer]]));
parseRes res = if res == mempty
  then default-answer
  else do {
    rem0' <- parseSpaces =<< unconsIO res;
    (m1,rem1) <- parseMat rem0';
    -- putStrLn Parsed a matrix;
    case mode of
      Simple -> return (m1, Nothing);
    - -> do {
      rem1' <- parseSpaces =<< unconsIO rem1;
      (m2,rem2) <- seq m1$ parseMat rem1';
      -- putStrLn Parsed a matrix;
      rem2' <- parseSpaces =<< unconsIO rem2;
      (m3,rem3) <- seq m2$ parseMat rem2';
      seq m3$ return ();
      -- putStrLn Parsed a matrix;
      if rem3 /= mempty
        then do { (-,rem2') <- parseSpaces =<< unconsIO rem3;
                  if rem2' /= mempty
                    then abort Unexpected output after parsing three matrices.
                    else return (m1, Just (m2,m3)) }
        else return (m1,Just (m2,m3))
    }
  }

```

```

    };
    fail-to-execute = seq sendError default-answer;

    default-answer = -- not small enough, but it'll be accepted
    return (in-mat, case mode of Simple -> Nothing; - -> Just (id-ofsize (length
in-mat),id-ofsize (length in-mat)));
    abort str = error$ Runtime exception in parsing fpLLL output:\n++str;
    };

```

*sendError* :: (); -- bad trick using unsafeIO to make this error only appear once. I believe this is OK since the error is non-critical and the 'only appear once' is non-critical too.

```

sendError = unsafePerformIO $ do {
  hPutStrLn stderr ---- WARNING ----;
  hPutStrLn stderr Failed to run fpLLL.;
  hPutStrLn stderr To remove this warning, either;;
  hPutStrLn stderr - install fpLLL and ensure it is in your path.;
  hPutStrLn stderr - create an executable fpLLL that always returns successfully
without generating output.;
  hPutStrLn stderr Installing fpLLL correctly helps to reduce time spent verifying your
certificate.;
  hPutStrLn stderr ---- END OF WARNING ----
};

```

```

id-ofsize :: Int -> [[Integer]];
id-ofsize n = [[if i == j then 1 else 0 | j <- [0..n-1]] | i <- [0..n-1]];
}

```

**code-reserved** (Haskell) *FPLLL-Solver fpLLL-solver*

#### **code-printing**

```

constant external-lll-solver -> (Haskell) FPLLL'-Solver.fpLLL'-solver
| constant enable-external-lll-solver -> (Haskell) True

```

Note that since we only enabled the external LLL solver for Haskell, the result of *short-vector-hybrid* will usually differ when executed in Haskell in comparison to any of the other target languages. For instance, consider the invocation of:

```

value (code) short-vector-test-hybrid [[1,4903,4902], [0,39023,0], [0,0,39023]]

```

The above value-command evaluates the expression in Eval/SML to 77714 (by computing a short vector solely by the verified *short-vector* algorithm, whereas the generated Haskell-code via the external LLL solver yields 60414!

**end**

## References

- [1] Ú. Erlingsson, E. Kaltofen, and D. R. Musser. Generic Gram-Schmidt orthogonalization by exact division. In *Proceedings of the 1996 International Symposium on Symbolic and Algebraic Computation, ISSAC '96, Zurich, Switzerland, July 24-26, 1996*, pages 275–282. ACM, 1996.
- [2] J. v. z. Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, New York, NY, USA, 2nd edition, 2003.
- [3] A. K. Lenstra, H. W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:515–534, 1982.
- [4] R. Thiemann and A. Yamada. Formalizing Jordan normal forms in Isabelle/HOL. In *CPP 2016*, pages 88–99. ACM, 2016.