TorDiv — A Maple Package on Toric Geometry and Geometric Invariant Theory

Florian Berchtold

Jürgen Hausen

Robert Vollmert

Marcel Widmann

Contents

CHAPTER 1

About TorDiv

1. General information

1.1. Aims of the project. The TorDiv package is an *open* software project, where open means open to contributions from further authors. The present aim is to provide an easy-to-use working environment for mathematicians active in Toric Geometry. A further intention is to offer a platform for the exchange and the discussion of advanced examples.

The TorDiv package provides, besides basic functions on lattices and convex polyhedra, several functions on Toric Geometry and, as there is a strong relation, on Geometric Invariant Theory of torus actions. At present, the TorDiv package mainly focusses on questions around divisors on toric varieties. For example, many results of [**3**] are implemented.

1.2. System and software requirements. As a Maple package, TorDiv can be run on any platform supporting Maple, version 8 or higher. The TorDiv package requires, however, the convex package by M. Franz [**8**]. The files of this package as well as the necessesary information for the installation are freely available on Franz's personal web page.

2. Getting started

2.1. How to get TorDiv? The TorDiv package is free software, distributed under the GNU General Public License. The files can be downloaded via the links available on the personal web pages of the authors (use for example *Google* to obtain the actual URL's).

2.2. Installing and starting. There are basically two ways to install the TorDiv package on your computer:

- Put the files TorDiv.ind and TorDiv.lib into the library directory (lib) of your Maple system.
- Create an extra directory for the files TorDiv.ind and TorDiv.lib and put them in.

For using the TorDiv package in a Maple session, proceed as follows. If you chose the second way of installation, say with an installation directory of full path name PATH, first tell Maple about this directory:

```
> libname := libname, "PATH":
```

Then, in both cases, the TorDiv package then is loaded, as any other Maple package, by means of the with command (first the convex package has to be load):

```
> with(TorDiv);
```

TorDiv, Version 1.3, (C) 2007 by F. Berchtold, J. Hausen, R. Vollmert, M. Widmann. This package is distributed under the GNU General Public License. It requires the convex package by M. Franz.

[*admitsgoodquot*, *amplecone*, *base*, *basedim*, *bunch*, *bunch2fan*, *bunchcones*, *bunchprojection*, *canonclass*, *cartierdiv*, *chamber*, *cocoref* , *coefficients*, *cols2matrix*, *completeseq*, *covcoll*, *coxconstr*, *cutpyramid*, *dualface*, *dualseq*, *effectivecone*, *fan2bunch*, *freebentcube*, *freecovering*, *gitfan*, *gitlimit*, *hirzebruch*, *ihermitecolops*, *ihermiterowops*, *imagefan*, *intcontains*, *intersectlattices*, *intersectrelint*, *intimage*, *intiscontained*, *intkernel*, *intpreimage*, *intprojection*, *intsection*, *isdivisorial*, *isfano*, *isgorenstein*, *isprimitive*, *isprojectedface*, *isprojective*, *isquasiprojective*, *isremovableray*, *isstandard*, *issurjective*, *istwocomplete*, *kajiwaraconstr*, *latticebasis*, *matrix2cols*, *movingcone*, *pairwiseintersectrelint*, *picardgroup*, *poldiv*, *ppdivisor*, *primitivespan*, *quotientfan*, *relintcontains*, *relintiscontained*, *removeray*, *ppfan*, *samplecone*, *semistablepoints*, *slice*, *tailcone*, *torusdim*, *twocompletion*, *weight2bunch*]

CHAPTER 2

Mathematical background

1. Lattices

1.1. Basic notions. By a *lattice* we mean a free finitely generated $\mathbb{Z}$-module. A sublattice N' of a lattice N is called *primitive* if there is a direct sum decomposition $N = N' \oplus N''$. A vector $v \in N$ is primitive if and only if the sublattice $\mathbb{Z}v \subset N$ is primitive.

1.2. Hermite and Smith normal forms. The Hermite and the Smith normal form of an integral matrix are standard instruments for many computational purposes, see for example [**6**, Section 2.4]. Recall the definitions of these normal forms:

DEFINITION 2.1. A matrix $A = (a_{ij}) \in \mathbb{Z}^{m\times n}$ is in Hermite normal form with respect to row operations (HNFR), if there are indices $1 \leq j_1 < j_2 < \cdots < j_r \leq m$ such that A fulfils the following properties for all i:

- the entries a_{ij_i} are positive, i.e. $a_{ij_i} \geq 1$,
- A is in row echelon form, i.e. $a_{kl} = 0$, if $k > r$ or $l < j_k$,
- the entries a_{ij_i} are the maximal elements in the respective column, i.e. $0 \leq a_{kj_i} < a_{ij_i}$ for $k < i$.

Note that the r in the above definition is just the rank of the matrix A.

DEFINITION 2.2. A matrix A is in Hermite normal form with respect to column operations (HNFC), if the transposed matrix $A^\top$ is in Hermite normal form with respect to row operations.

The crucial observation is that any matrix can be brought into HFNR (HNFC) by invertible row operations (column operations):

THEOREM 2.3. *Let $A \in \mathbb{Z}^{m\times n}$ be given. Then there exist invertible matrices $U \in \mathbb{Z}^{m\times m}$ and $V \in \mathbb{Z}^{n\times n}$ such that UA is in HNFR and AV is in HNFC.*

By applying both row and column operations one may obtain the Smith normal form:

DEFINITION 2.4. A matrix $A = (a_{ij}) \in \mathbb{Z}^{m\times n}$ is in Smith normal form, if, for $r := \operatorname{rank} A$,we have $a_{(i-1)(i-1)} | a_{ii}$ for $2 \leq i \leq r$ and $a_{kl} = 0$ holds for $k \neq l$ or $k > r$.

THEOREM 2.5. *Let $A \in \mathbb{Z}^{m\times n}$ be given. Then there exist invertible matrices $U \in \mathbb{Z}^{m\times m}$ and $V \in \mathbb{Z}^{n\times n}$ such that UAV is in Smith normal form.*

1.3. Lattice reduction. Given a sublattice of $\mathbb{Z}^n$, it is often desirable to find explicitly a basis consisting of "short" vectors. There is a variety of so-called lattice reduction algorithms tackling this problem. As we shall use it, we mention here the LLL-algorithm due to A.K. Lenstra, H.W. Lenstra and L. Lovász, see for example [**6**, Section 2.6].

2. Convex polyhedra

2.1. Cones and fans. We fix here the basic terminology and provide the basic concepts on convex polyhedral cones and fans. For a little more background, we refer the reader to the textbooks [**9**] and [**13**].

The rational vector space associated to a lattice N is $N_{\mathbb{Q}} := \mathbb{Q} \otimes_{\mathbb{Z}} N$. If $P\colon F \to N$ is a lattice homomorphism, then we denote the induced linear map $F_{\mathbb{Q}} \to N_{\mathbb{Q}}$ of rational vector spaces again by P.

By a *cone* in a lattice N we always mean a polyhedral (not necessarily strictly) convex cone in the associated rational vector space $N_{\mathbb{Q}}$. Let N be a lattice, and let $M := \mathrm{Hom}(N, \mathbb{Z})$ denote the dual lattice of N. The *orthogonal space* and the *dual cone* of a cone σ in N are

$$\sigma^{\perp} := \{u \in M_{\mathbb{Q}};\ u|_{\sigma} = 0\}, \qquad \sigma^{\vee} := \{u \in M_{\mathbb{Q}};\ u|_{\sigma} \geq 0\}.$$

The *relative interior* of a cone σ is denoted by σ°. If σ_0 is a *face* of σ, then we write $\sigma_0 \preceq \sigma$. The *dimension* of a cone σ is the dimension of the linear space $\mathrm{lin}(\sigma)$ generated by σ. The set of the k-dimensional faces of σ is denoted by $\sigma^{(k)}$, and the one-dimensional faces of a strictly convex cone are called its *rays*.

The *primitive generators* of a strictly convex cone σ in a lattice N are the primitive lattice vectors of its rays. A strictly convex cone in N is called *simplicial* if its primitive generators are linearly independent, and it is called *regular* if its primitive generators can be complemented to a lattice basis of N.

A *fan* in a lattice N is a finite collection Δ of strictly convex cones in N such that for each $\sigma \in \Delta$ also all $\sigma_0 \preceq \sigma$ belong to Δ and for any two $\sigma_i \in \Delta$ we have $\sigma_1 \cap \sigma_2 \preceq \sigma_i$. A *map of fans* Δ_i in lattices N_i is a lattice homomorphism $F\colon N_1 \to N_2$ such that for every $\sigma_1 \in \Delta_1$ there is a $\sigma_2 \in \Delta_2$ containing $F(\sigma_1)$.

Note that the compatibility condition $\sigma_1 \cap \sigma_2 \preceq \sigma_i$ in the above definition is equivalent to the existence of a *separating linear form* for the cones σ_1 and σ_2, i.e., a linear form u on N such that

$$u_{|\sigma_1} \geq 0, \qquad u_{|\sigma_2} \leq 0, \qquad u^{\perp} \cap \sigma_i = \sigma_1 \cap \sigma_2.$$

Replacing "strictly convex" with "convex" in the definition of a fan, we obtain the more general notion of a *quasifan*. For a quasifan Δ in N, we denote by $|\Delta|$ its *support*, that is the union of all its cones. Moreover, $\Delta^{\max}$ is the set of maximal cones of a quasifan Δ, and $\Delta^{(k)}$ is the set of the k-dimensional cones of Δ.

Every quasifan Δ in a lattice N can be made in a canonical way to a fan: For the maximal cones of Δ, there is a maximal common linear subspace $V \subset N_{\mathbb{Q}}$. Set $N_{\Delta} := V \cap N$ and $N_0 := N/N_{\Delta}$. Then the images of the cones of Δ under the projection $N \to N_0$ form a fan Δ_0 in N_0, the *quotient fan* of Δ.

An important object is the *normal quasifan* Δ_B associated to a polyhedron $B \subset M_{\mathbb{Q}}$: its cones correspond to the faces $F \preceq B$. More explicitely, this correspondence is given by

$$\mathrm{faces}(B) \to \Delta_B, \qquad F \mapsto \sigma_F := \{v \in N_{\mathbb{Q}};\ v(u) \leq v(u') \text{ for all } (u, u') \in F \times B\}.$$

Note that Δ_B is a fan if and only if B is of full dimension. Moreover, the support of Δ_B admits a nice description. Writing $B = B_0 + \sigma$ with a polytope B_0 and a cone σ, we have $|\Delta_B| = \sigma^\vee$.

In this setting, the following notions are common: a fan is called *polytopal* if it is the normal fan of a polytope, and it is called *quasipolytopal* if it is a subfan of some polytopal fan.

2.2. Bunches of cones. The concept of a bunch of cones was introduced in [**3**]. These bunches of cones are strongly related to fans; the connection is a certain Gale duality. We provide here the basic definitions and constructions, working in a slightly more special setup as in [**3**].

Let $(E \xrightarrow{Q} K, \gamma)$ be a *projected cone*, i.e., $Q\colon E \to K$ is a surjection of lattices and $\gamma \subset E_{\mathbb{Q}}$ is a regular polyhedral convex cone of full dimension. A *projected face* is the image $Q(\gamma_0)$ of a face $\gamma_0 \preceq \gamma$. A *bunch* in $(E \xrightarrow{Q} K, \gamma)$ is a nonempty collection Θ of projected faces with the following property: a projected face σ belongs to Θ if and only if $\emptyset \neq \sigma^\circ \cap \tau^\circ \neq \tau^\circ$ holds for all $\tau \in \Theta$ with $\tau \neq \sigma$.

We explain now the relation to fans. Bunches of cones correspond to what we will call maximal projectable fans, and to each of the latter ones we may associate a quotient fan. We briefly summarize these constructions and the basic statements. For the details, we refer to [**3**].

To any projected cone $(E \xrightarrow{Q} K, \gamma)$, we associate a dual projected cone as follows. Let M denote the kernel of $Q\colon E \to K$. Then we have two exact sequences of lattice homomorphisms

$$0 \longrightarrow M \longrightarrow E \xrightarrow{Q} K \longrightarrow 0,$$

$$0 \longleftarrow N \xleftarrow[P]{} F \longleftarrow L \longleftarrow 0,$$

where the second sequence arises from the first one by applying $\mathrm{Hom}(?, \mathbb{Z})$. The dual cone $\delta := \gamma^\vee$ is again strictly convex, regular and of full dimension. The *dual projected cone* of $(E \xrightarrow{Q} K, \gamma)$ is $(F \xrightarrow{P} N, \delta)$.

In the sequel, fix a projected cone $(E \xrightarrow{Q} K, \gamma)$, and denote the associated dual projected cone by $(F \xrightarrow{P} N, \delta)$. Recall that there is the order reversing face correspondence, see for example [**13**, Appendix A]:

$$\mathrm{faces}(\gamma) \to \mathrm{faces}(\delta), \qquad \gamma_0 \mapsto \gamma_0^* := \gamma_0^\perp \cap \delta.$$

The following lemma is a crucial observation on corresponding faces in projected cones. Let $L := \ker(P)$. By an *L-invariant* linear form we mean here an element $u \in E = \mathrm{Hom}(F, \mathbb{Z})$ with $L \subset u^\perp$.

Lemma 2.6 (Invariant Separation Lemma). *Let $\gamma_1, \gamma_2 \preceq \gamma$, and set $\delta_i := \gamma_i^*$., and let $L := \ker(P)$. Then the following statements are equivalent:*

(i) *There is an L-invariant separating linear form for δ_1 and δ_2.*
(ii) *For the relative interiors $Q(\gamma_i)^\circ$ we have $Q(\gamma_1)^\circ \cap Q(\gamma_2)^\circ \neq \emptyset$.*

Consider any bunch Θ in $(E \xrightarrow{Q} K, \gamma)$. We shall shift Θ into the dual projected cone. Firstly, we pass to the *covering collection* $\mathrm{cov}(\Theta)$ of the bunch Θ; this is the set of all faces $\gamma_0 \preceq \gamma$ that are minimal with respect to the property that $Q(\gamma_0)$ contains a cone of Θ. Then we set

$$\widehat{\Delta} := \{\delta_0;\ \delta_0 \preceq \gamma_0^* \text{ for some } \gamma_0 \in \mathrm{cov}(\Theta)\}.$$

By Lemma 2.6, $\widehat{\Delta}$ is a *maximal projectable fan* in $(F \xrightarrow{P} N, \delta)$, i.e., it consists of faces of $\delta \subset F_{\mathbb{Q}}$, any two maximal cones $\delta_1, \delta_2 \in \widehat{\Delta}$ admit an L-invariant separating linear form, and a face $\delta_0 \preceq \delta$ belongs to $\widehat{\Delta}$ provided it can be separated from all maximal cones of $\widehat{\Delta}$ by L-invariant linear forms.

PROPOSITION 2.7. *The map $\Theta \mapsto \widehat{\Delta}$ sets up a one to one correspondence between bunches in $(E \xrightarrow{Q} K, \gamma)$ and maximal projectable fans in $(F \xrightarrow{P} N, \delta)$.*

Finally, the *Gale transform* of the bunch Θ, is the quotient fan Δ_0 of the quasifan Δ in N generated by the images $P(\delta_0)$, where $\delta_0 \in \widehat{\Delta}^{\max}$. In general, different bunches may have the same Gale transform. But, as we shall see later, this procedure becomes reversible when restricted to a suitable subclass of bunches.

2.3. Secondary fans. Consider a lattice N and vectors $v_1, \ldots, v_r \in N$; for simplicity, we assume that these vectors generate N. Secondary fans provide a solution to the following problem: What are the possible quasipolytopal fans in N having their rays among the $\mathbb{Q}_{\geq 0} v_i$?

We present here an approach to secondary fans that starts in the setting of bunches of cones. Let $F := \mathbb{Z}^r$, and consider the projected cone $(F \xrightarrow{P} N, \delta)$, where P maps the i-th canonical base vector of F to v_i, and δ is the positive orthant in $F_{\mathbb{Q}}$.

Let $(E \xrightarrow{Q} K, \gamma)$ be the dual projected cone, and let $e_1, \ldots e_r$ denote the primitive generators of γ. For a collection of faces $\gamma_1, \ldots, \gamma_l \preceq \gamma$, consider the intersection of their images:

$$\tau(\gamma_1, \ldots, \gamma_l) \quad := \quad \bigcap_{i=1}^{l} Q(\gamma_i).$$

We say that $\tau(\gamma_1, \ldots, \gamma_l)$ is a *final cone* of $(E \xrightarrow{Q} K, \gamma)$ if for every face $\gamma_0 \preceq \gamma$ we have either $\tau(\gamma_1, \ldots, \gamma_l) \subset Q(\gamma_0)$ or $\tau(\gamma_1, \ldots, \gamma_l) \cap Q(\gamma_0)$ is of smaller dimension than $\tau(\gamma_1, \ldots, \gamma_l)$.

PROPOSITION 2.8. *The final cones of $(E \xrightarrow{Q} K, \gamma)$ form a fan Σ in K with $|\Sigma| = Q(\gamma)$. The fan Σ is the coarsest common refinement of all possible triangulations spanned by the vectors $Q(e_1), \ldots, Q(e_r)$.*

The fan Σ of final cones of $(E \xrightarrow{Q} K, \gamma)$ is usually called the *secondary fan* of the vector configuration $v_1, \ldots, v_r \in N_{\mathbb{Q}}$. As mentioned at the beginning of this section, its meaning is the following:

PROPOSITION 2.9. *There is a well defined injection from the secondary fan Σ of the vector configuration $v_1, \ldots, v_r \in N_{\mathbb{Q}}$ into the set of bunches in $(E \xrightarrow{Q} K, \gamma)$:*

$$\begin{array}{rccl} \mathfrak{B}\colon & \Sigma & \to & \text{bunches}(E \xrightarrow{Q} K, \gamma) \\ & \sigma & \mapsto & \{Q(\gamma_0);\ \gamma_0 \preceq \gamma \text{ minimal with } \sigma^\circ \subset Q(\gamma_0)^\circ\} \end{array}$$

The image of this map consists precisely of the bunches having a corresponding maximal projectable fan with a quasipolytopal quotient fan.

For a bunch Θ in $(E \xrightarrow{Q} K, \gamma)$, let $\widehat{\Delta}(\Theta)$ be the corresponding maximal projectable fan in $(F \xrightarrow{P} N, \delta)$, and let $\Delta(\Theta)$ denote the associated quotient fan.

PROPOSITION 2.10. *For $\sigma \in \Sigma$ choose a point $w \in \sigma^\circ$ and a point $u \in Q^{-1}(w)$. Then $\Delta(\mathfrak{B}(\sigma))$ is the quotient fan of the normal quasifan of the fibre polyhedron $(Q^{-1}(w) - u) \cap M_{\mathbb{Q}}$.*

3. Toric Geometry

3.1. Toric varieties and fans. We provide the basic definitions and constructions of the theory of toric varieties. We work over an algebraically closed field $\mathbb{K}$ of characteristic zero, i.e., by a variety we mean a reduced integral scheme over $\mathbb{K}$, and, similarly, morphisms are supposed to be defined over $\mathbb{K}$.

A *toric variety* is a normal algebraic variety X containing an algebraic torus T_X as an open subset such that the group structure of T_X extends to a regular action on X. A *toric morphism* is a regular map $X \to Y$ of toric varieties that restricts to a group homomorphism $T_X \to T_Y$.

The most important feature of the category of toric varieties is that it admits a complete description in terms of fans. We briefly indicate how this works. Let Δ be a fan in a lattice N, and let $M := \mathrm{Hom}(N, \mathbb{Z})$ be the dual lattice of N. For every cone $\sigma \in \Delta$ one defines an affine toric variety:

$$X_\sigma \quad := \quad \mathrm{Spec}(\mathbb{K}[\sigma^\vee \cap M]).$$

For any two such X_{σ_i}, one has canonical open embeddings of $X_{\sigma_1 \cap \sigma_2}$ into X_{σ_i}. Patching together all X_σ along these open embeddings gives a toric variety X_Δ. The assignment $\Delta \mapsto X_\Delta$ is functorial, and it turns out to be a (covariant) equivalence of categories.

This equivalence of categories provides a very concrete combinatorial language for the geometry of toric varieties. For example, a toric variety X_Δ is

- *smooth* if and only if all cones of Δ are regular,
- $\mathbb{Q}$-*factorial* if and only if all cones of Δ are simplicial,
- *complete* if and only if Δ is so, i.e., $|\Delta|$ is the whole space,
- *projective* if and only if Δ is the normal fan of a polytope.

For the proofs of these basic observations and many further results, we refer the reader to the standard textbooks [**9**] and [**13**].

3.2. Toric varieties and bunches. As an alternative to the language of fans, one may use bunches of cones to describe toric varieties. This is based on the correspondence presented in Section 2.2, and it works particularly nice in the following situation: let Δ be a fan in a lattice N, set $\mathrm{R} := \Delta^{(1)}$, and assume that

- the primitive vectors v_ϱ, $\varrho \in \mathrm{R}$, generate N as a lattice,
- the fan Δ cannot be enlarged without adding new rays.

Geometrically, the first property means that the toric variety X associated to Δ is *nondegenerate*, i.e., it admits no splitting $X \cong X' \times \mathbb{K}^*$, and that the divisor class group $\mathrm{Cl}(X)$ is free. Note that any toric variety is a finite quotient of one with a free divisor class group.

The second property means that X is *2-complete*, i.e., it does not occur as a proper invariant open subset of a toric variety X' such that $X' \setminus X$ is of codimension at least two. For example, every complete and every affine toric variety is 2-complete.

Let us indicate how to describe X by means of a bunch of cones Θ. The key is Cox's Construction [**7**]: set $F := \mathbb{Z}^{\mathrm{R}}$, and consider the map linear map $P\colon F \to N$

sending the canonical base vector e_ϱ to v_ϱ. For $\sigma \in \Delta^{\max}$, set

$$\widehat{\sigma} \quad := \quad \operatorname{cone}(e_\varrho;\ \varrho \in \sigma^{(1)}).$$

Then we obtain a projected cone $(F \xrightarrow{P} N, \delta)$, where $\delta := \operatorname{cone}(e_\varrho;\ \varrho \in \mathrm{R})$, and a fan $\widehat{\Delta}$ having the $\widehat{\sigma}$, where $\sigma \in \Delta^{\max}$, as its maximal cones. Here are the characteristic properties:

(i) $\widehat{\Delta}$ is a maximal projectable fan in $(F \xrightarrow{P} N, \delta)$ containing all rays of δ,
(ii) the map $P\colon F \to N$ induces bijections $\widehat{\Delta}^{\max} \to \Delta^{\max}$ and $\widehat{\Delta}^{(1)} \to \Delta^{(1)}$,
(iii) the images $P(e_\varrho)$ of the generators of δ are primitive vectors in N.

Let Θ be the bunch of cones in the dual projected cone $(E \xrightarrow{Q} K, \gamma)$ corresponding to $\widehat{\Delta}$. The above properties (i) to (iii) translate into the defining properties of a *standard bunch* for Θ: for every facet $\gamma_0 \preceq \gamma$, we have $Q(\gamma_0 \cap E) = K$ and $Q(\gamma_0)^\circ \supset \tau^\circ$ for some $\tau \in \Theta$.

The assignment $\Delta \mapsto \Theta$ turns out to be inverse to the Gale transformation restricted to standard bunches. Thus, we obtained an equivalence of categories between standard bunches and nondegenerate 2-complete toric varieties with a free divisor class group:

Given a standard bunch Θ, consider the corresponding maximal projectable fan $\widehat{\Delta}(\Theta)$ according to Proposition 2.7, and then define X_Θ to be the toric variety of the quotient fan $\Delta(\Theta)$, which is associated to $\widehat{\Delta}(\Theta)$.

3.3. Toric geometry in terms of bunches. Let Θ be a standard bunch in a projected cone $(E \xrightarrow{Q} K, \gamma)$. We present some results of [**3**] that allow to read off geometric properties of the associated toric variety $X := X_\Theta$ directly from Θ. First observations concern the singularities:

- The variety X is $\mathbb{Q}$-factorial if and only if every cone $\tau \in \Theta$ is of full dimension.
- The variety X is smooth if and only if $K = Q(\operatorname{lin}(\gamma_0) \cap E)$ holds for all γ_0 in $\operatorname{cov}(\Theta)$.

The power of the language of bunches lies in the description of geometric phenomena around divisors. The reason for this is the following (well known) observation:

PROPOSITION 2.11. *The divisor class group of X satisfies* $\operatorname{Cl}(X) \cong K$.

This is due to the fact that E is the group of invariant Weil divisors of X_Θ and $M := \ker(Q)$ is the group of invariant principal divisors. The Picard group of X can be expressed in terms of the covering collection:

THEOREM 2.12. *In the divisor class group* $\operatorname{Cl}(X) \cong K$*, the Picard group of X is given by*

$$\operatorname{Pic}(X) \quad = \quad \bigcap_{\gamma_0 \in \operatorname{cov}(\Theta)} Q(\operatorname{lin}(\gamma_0) \cap E).$$

Moreover, one has natural descriptions of important cones of divisors: the effective cone, the moving cone, the semiample cone and the ample cone:

THEOREM 2.13. *In the divisor class group* $\mathrm{Cl}(X) \cong K$*, we have the following descriptions:*

$$\mathrm{Eff}(X) \;=\; Q(\gamma), \qquad\qquad \mathrm{Mov}(X) \;=\; \bigcap_{\gamma_0 \text{ facet of } \gamma} Q(\gamma_0),$$

$$\mathrm{Sample}(X) \;=\; \bigcap_{\tau\in\Theta} \tau, \qquad\qquad \mathrm{Ample}(X) \;=\; \bigcap_{\tau\in\Theta} \tau^\circ.$$

The canonical divisor class of a toric variety is minus the sum of the classes of the invariant prime divisors. Thus, as a consequence of the above results, we can decide if X is Gorenstein or Fano:

THEOREM 2.14. *Let* $e_1, \ldots, e_r$ *be the primitive generators of* γ*. The toric variety* X *associated to* Θ *is*

(i) *Gorenstein if and only if we have*

$$\sum Q(e_i) \quad\in \bigcap_{\gamma_0\in\mathrm{cov}(\Theta)} Q(\mathrm{lin}(\gamma_0)\cap E).$$

(ii) *Fano if and only if we have*

$$\sum Q(e_i) \quad\in \bigcap_{\gamma_0\in\mathrm{cov}(\Theta)} Q(\mathrm{lin}(\gamma_0)\cap E) \;\cap\; \bigcap_{\tau\in\Theta} \tau^\circ.$$

4. Geometric Invariant Theory

4.1. Good quotients. Let the reductive group G act on a variety X by means of a morphism $G \times X \to X$. A *good quotient* for this action is a G-invariant affine morphism $p\colon X \to Y$ such that the canonical map $\mathcal{O}_Y \to p_*(\mathcal{O}_X)^G$ is an isomorphism. If it exists, then the good quotient space is usually denoted by $X/\!\!/G$.

In general, a G-variety X need not admit a good quotient $X \to X/\!\!/G$, but there frequently exist many invariant open subsets $U \subset X$ with good quotient $U \to U/\!\!/G$. We will call these sets for short the *good G-sets*. It is one of the central tasks of Geometric Invariant Theory (GIT) to describe all the good G-sets, see [**4**, Section 7.2].

We briefly present D. Mumfords construction [**12**] of good G-sets with quasiprojective quotient spaces. Let L be a line bundle over X. A *G-linearization* of L is a G-action on the total space such that the projection $L \to X$ is equivariant and the induced maps $L_x \to L_{gx}$ of the fibres are linear. Note that for any two G-linearized line bundles, their tensor product comes with a natural G-linearization.

To any G-linearized line bundle L over X, one associates a set of *semistable points* $X^{ss}(L)$. Consider the representation of G on the space $\Gamma(X, L)$ of global sections given by

$$(g\cdot f)(x) \quad:=\quad g\cdot(f(g^{-1}\cdot x)).$$

Then one can talk about the invariant sections $\Gamma(X, L)^G$, and one defines the set of semistable points associated to L to be

$$X^{ss}(L) \quad:=\quad \{x \in X;\ f(x) \neq 0 \text{ for some } f \in \Gamma(X, L^{\otimes n})^G, \text{ where } n > 0\}.$$

This set $X^{ss}(L)$ of semistable points turns out to be a good G-set and the quotient space $X^{ss}(L)/\!\!/G$ is a quasiprojective variety. Moreover, if X is smooth,

then every good G-set with a quasiprojective quotient space is saturated (w.r. to the quotient map) in the set of semistable points of a G-linearized line bundle.

Note that Mumford's construction is by far not the whole story: there exist many examples of good G-sets with a non quasiprojective quotient space, even if one starts with a quasiprojective or an affine variety X. Examples in the setting of torus actions are the Cox Constructions of non quasiprojective toric varieties.

4.2. Subtorus actions. Let a torus T act on a variety X. Following [**4**, Section 7.2] and [**5**], we say that a good T-set $U \subset X$ is *maximal*, if there is no good T-set $U' \subset X$ such that U is a proper subset of U' and U is saturated w.r. to the quotient map $U' \to U'/\!\!/G$.

Now assume that X is a toric variety, and that T is a subtorus of the big torus T_X. In the setting, the maximal good T-sets can be characterized in terms of fans. This relies on the following observation due to Święcicka, see [**15**, Proposition 2.5]: If $U \subset X$ is a T-maximal subset, then U is invariant under T_X.

Now, let X be the toric variety arising from a fan Δ in a lattice N, let $T \subset T_X$ be the subtorus corresponding to a primitive sublattice $L \subset N$. Then the T-maximal subsets $U \subset X$ correspond to certain subfans of Δ. The characterization of these fans is standard, see e.g. [**10**, Proposition 1.3]:

PROPOSITION 2.15. *Let $U \subset X$ be the open T_X-invariant subset defined by a subfan Σ of Δ. Then U is a maximal good T-set if and only if it satisfies the following two properties:*

(i) *any two maximal cones of Σ can be separated by an L-invariant linear form,*
(ii) *every $\sigma \in \Delta$ that can be separated by L-invariant linear forms from the maximal cones of Σ belongs to Σ.*

Note that these two properties occur as well in the definition of a maximal projectable fan in a projected cone. In the setting of Proposition 2.15, they ensure that the images of the maximal cones of Σ in N/L generate a quasifan. The associated quotient fan eventually describes the (again toric) quotient variety $U/\!\!/T$.

Finally, we consider the special case $X = \mathbb{K}^n$, that means that X is the affine toric variety arising from the positive orthant δ in $F := \mathbb{Z}^n$. We shall describe the maximal good T-sets in terms of bunches of cones. For this, let $L \subset F$ be the sublattice corresponding to $T \subset T_X$.

Setting $N := F/L$, we obtain a projected cone $(F \xrightarrow{P} N, \delta)$. Moreover, we have the dual projected cone $(E \xrightarrow{Q} K, \gamma)$. Recall that to any bunch Θ in $(E \xrightarrow{Q} K, \gamma)$, we associated a maximal projectable fan $\widehat{\Delta}(\Theta)$ in $(F \xrightarrow{P} N, \delta)$.

THEOREM 2.16. *The assignment $\Theta \mapsto X_{\widehat{\Delta}(\Theta)}$ defines a one-to-one correspondence between the bunches in $(E \xrightarrow{Q} K, \gamma)$ and the maximal good T-sets in X.*

A *geometric quotient* for an action of a reductive group G on a variety X is a good quotient that separates orbits. Geometric quotients are denoted by $X \to X/G$. In the setting of Theorem 2.16, the geometric quotients are described by the following:

PROPOSITION 2.17. *Let Θ be a bunch in $(E \xrightarrow{Q} K, \gamma)$. The open toric subvariety $X_{\widehat{\Delta}(\Theta)} \subset X$ admits a geometric quotient by the action of T if and only if $\dim(\tau) = \dim(K)$ holds for every $\tau \in \Theta$.*

4.3. Mumford quotients of $\mathbb{K}^n$. Consider the affine toric variety $X = \mathbb{K}^n$, and the action of subtorus $T \subset T_X$ on X. We discuss Mumford's construction in this setting. Note that X is factorial, and hence all line bundles on X are trivial. The possible T-linearizations of the trivial bundle are given by characters $\chi^w \colon T \to \mathbb{K}^*$ and are of the form

$$t \cdot (x, z) \quad = \quad (t \cdot x, \chi^w(t) z).$$

We describe now the associated set of semistable points in terms of fans. The affine toric variety X arises from the positive orthant δ in $F := \mathbb{Z}^n$. The subtorus $T \subset T_X$ corresponds to a sublattice $L \subset F$, and $K := \mathrm{Hom}(L, \mathbb{Z})$ is isomorphic to the character space of T via $w \mapsto \chi^w$.

Now, given $w \in K$, we denote by $X(w) \subset X$ the set of semistable points associated to the above linearization of the trivial bundle. As usual, we denote by $Q \colon E \to K$ the dual map of the inclusion $L \to F$. Then $X(w)$ is nonempty if and only if $w \in Q(\gamma)$, and in the latter case, it is the open toric subvariety of X corresponding to the fan

$$\Sigma(w) \quad = \quad \{\sigma \leq \delta;\ w \in Q(\sigma^*)\}.$$

The map $w \mapsto X(w)$ on $Q(\gamma)$ is described in terms of the secondary fan Δ subdividing the cone $Q(\gamma)$: we have $X(w) \subset X(w')$ if and only if $w \in \tau$ and $w' \in \tau'$ with $\tau' \preceq \tau$. In particular, $w \mapsto X(w)$ is constant on the relative interiors of the cones of the secondary fan. For that reason, the secondary fan is as well called the *GIT-fan*.

The various quotients $X(w)/\!\!/T$ form a directed system: Whenever we have $X(w) \subset X(w')$, there is a (unique) induced map $X(w)/\!\!/T \to X(w')/\!\!/T$, and these maps are compatible with respect to composition. In the direct limit of this system there is a unique irreducible component Y_0 rationally dominanting all the quotients $X(w)/\!\!/T$.

The normalization of Y_0 is a toric variety. Its fan Δ_0 is obtained as follows: Consider the projection $P \colon F \to N$ with $N := F/L$. Then Δ_0 is the coarsest common refinement of all the cones $P(\delta_0)$, where $\delta_0 \preceq \delta$.

CHAPTER 3

The `TorDiv` functions

1. Functions on lattices

1.1. Introductory remarks. The package `TorDiv` provides several functions performing standard computations in lattices, such as determinig the kernel of a linear map, intersecting sublattices etc.. Most of these functions are standard applications of the Hermite and Smith normal forms of integral matrices; for computing, we use implementations `ihermite` and `ismith` of Maple.

We deal with the lattices $\mathbb{Z}^m$. An element $v \in \mathbb{Z}^m$ always is represented by its coordinates with respect to the canonical basis; thus v is represented by a list of integers. Similarly, a linear map $p\colon \mathbb{Z}^m \to \mathbb{Z}^n$ is always represented by its matrix with respect to the canonical basis. A matrix can be entered as a listlist of integers representing the column vectors, or via the `matrix` command of Maple.

1.2. `cols2matrix.` Given a set of integral m-vectors, this function provides a matrix having the given vectors as columns.

1.2.1. *Usage.*

Input: A listlist of integers representing a set of vectors in $\mathbb{Z}^m$

Output: an integral $(m \times l)$-matrix.

1.2.2. *Example.*

```
> L := [[1,2,3],[4,5,6]]:
> cols2matrix(L);
```

$$\begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

1.3. `completeseq.` Given a linear injection $\imath\colon \mathbb{Z}^l \to \mathbb{Z}^m$ with a primitive image [a linear surjection $p\colon \mathbb{Z}^m \to \mathbb{Z}^n$], the function provides a completed exact sequence of integral linear maps

$$0 \longrightarrow \mathbb{Z}^l \xrightarrow{\imath} \mathbb{Z}^m \xrightarrow{p} \mathbb{Z}^n \longrightarrow 0.$$

All occuring linear maps are represented by matrices with respect to canonical bases. It is specified via an additional argument `inj` or `surj` wether the input map should be the injection $\imath$ or the surjection p of the resulting sequence.

1.3.1. *Usage.*

Input: An integral $(l \times m)$-matrix and a specification `inj`
[an integral $(m \times n)$-matrix and a specification `surj`].

Output: A pair of integral matrices.

1.3.2. *Example.*

```
> A := linalg[matrix]([[1,-1],[1,0],[1,1],[1,2]]);
```

$$A := \begin{bmatrix} 1 & -1 \\ 1 & 0 \\ 1 & 1 \\ 1 & 2 \end{bmatrix}$$

```
> completeseq(A,inj);
```

$$\left[\left[\begin{bmatrix} 1 & -1 \\ 1 & 0 \\ 1 & 1 \\ 1 & 2 \end{bmatrix}, \begin{bmatrix} 1 & -2 & 1 & 0 \\ 1 & -1 & -1 & 1 \end{bmatrix}\right]\right]$$

```
> B := linalg[transpose](A);
```

$$B := \begin{bmatrix} 1 & 1 & 1 & 1 \\ -1 & 0 & 1 & 2 \end{bmatrix}$$

```
> completeseq(B,surj);
```

$$\left[\left[\begin{bmatrix} 1 & 1 \\ -2 & -1 \\ 1 & -1 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 & 1 & 1 \\ -1 & 0 & 1 & 2 \end{bmatrix}\right]\right]$$

1.4. `dualseq`. Given a sequence of integral linear maps, this function computes the dualized sequence. Sequences are represented by pairs $[A, B]$ of integral matrices such that BA can be formed.

1.4.1. *Usage.*

Input: A pair of integral matrices.

Output: A pair of integral matrices.

1.4.2. *Example.*

```
> A := linalg[matrix]([[1,-1],[1,0],[1,1],[1,2]]);
```

$$A := \begin{bmatrix} 1 & -1 \\ 1 & 0 \\ 1 & 1 \\ 1 & 2 \end{bmatrix}$$

```
> B := linalg[matrix]([[1,-2,1,0],[1,-1,-1,1]]);
```

$$B := \begin{bmatrix} 1 & -2 & 1 & 0 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

```
> dualseq([A,B]);
```

$$\left[\left[\begin{bmatrix} 1 & 1 \\ -2 & -1 \\ 1 & -1 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 & 1 & 1 \\ -1 & 0 & 1 & 2 \end{bmatrix}\right]\right]$$

1.5. `ihermitecolops`. Given an integral matrix A, this function computes a pair $[H, U]$, where H is in Hermite normal form with respect to invertible integer column operations, and U is a unimodular matrix satisfying $H = AU$.

1.5.1. *Usage.*

Input: An integral matrix.

Output: A pair of integral matrices.

1.5.2. *Example.*

```
> A := linalg[matrix]([[1,2,3,4],[5,6,7,8]]);
```

$$A := \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{bmatrix}$$

```
> ihermitecolops(A);
```

$$\left[\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 4 & 0 & 0 \end{bmatrix}, \begin{bmatrix} -1 & 2 & 1 & 2 \\ 1 & -1 & -2 & -3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \right]$$

1.6. `ihermiterowops`. Given an integral matrix A, this function computes a pair $[H, U]$, where H is in Hermite normal form with respect to invertible integer row operations, and U is a unimodular matrix satisfying $H = UA$.

1.6.1. *Usage.*

Input: An integral matrix.

Output: A pair of integral matrices.

1.6.2. *Example.*

```
> A := linalg[matrix]([[1,2],[3,4],[5,6],[7,8]]);
```

$$A := \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 7 & 8 \end{bmatrix}$$

```
> ihermiterowops(A);
```

$$\left[\begin{bmatrix} 1 & 0 \\ 0 & 2 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} -2 & 1 & 0 & 0 \\ 3 & -1 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 2 & -3 & 0 & 1 \end{bmatrix} \right]$$

1.7. `intcontains`. This function checks whether or not a given sublattice $L \subset \mathbb{Z}^m$ contains a given vector $v \in \mathbb{Z}^m$.

1.7.1. *Usage.*

Input: a listlist of integers [a matrix], and a list of integers.

Output: a boolean.

1.7.2. *Example.*

```
> A := [[2,0,0],[0,2,0]]:
> v := [1,1,0]:
> intcontains(A,v);
```

false

1.8. `intersectlattices`. Given two sublattices of $\mathbb{Z}^m$, this function determines a basis for the intersection of these sublattices. Depending on which of the possible options `list`, `matrix` is chosen, the output is a listlist of integers a matrix (the default is a listlist). If the possible option `red` is chosen, then the output forms a LLL-reduced system.

1.8.1. *Usage.*

Input:	Two listlists of integers, representing systems of generators of sublattices of $\mathbb{Z}^m$ [two integral $(m \times l_i)$-matrices].
Option(s):	`list`, `matrix`, `red`.
Output:	A listlist of integers representing the vectors of a basis of the intersection sublattice [an integral $(n \times m)$-matrix the columns of which form a basis of the intersection sublattice].

1.8.2. *Example.*

```
> A := [[1,2,3],[4,5,6]]:
> B := [[6,7,8],[9,10,11]]:
> intersectlattices(A,B);
```

$$[[3,\ 0,\ -3],\ [0,\ 3,\ 6]]$$

1.9. `intimage`. Given a linear map $p\colon \mathbb{Z}^m \to \mathbb{Z}^n$, and vectors $v_1, \ldots, v_r \in \mathbb{Z}^m$, this function computes an integer basis for the image of the sublattice generated by the v_i under the map p. Depending on which of the possible options `list`, `matrix` is chosen, the output is a listlist of integers or a matrix (the default is a listlist). If the possible option `red` is chosen, then the basis for the kernel will be LLL-reduced.

1.9.1. *Usage.*

Input:	A listlist of integers, representing the images $p(e_i) \in \mathbb{Z}^n$ of the canonical base vectors $e_i \in \mathbb{Z}^n$ [an integral $(n \times m)$-matrix, representing $p\colon \mathbb{Z}^m \to \mathbb{Z}^n$ with respect to the canonical bases], a listlist of integers, representing the vectors generating the sublattice
Option(s):	`list`, `matrix`, `red`.
Output:	A listlist of integers, representing the vectors of the image [an integral $(m \times l)$-matrix, the columns of which form a basis of the image].

1.9.2. *Example.*

```
> Q := matrix([[1,2,3],[3,2,1]]);
```

$$Q := \begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{bmatrix}$$

```
> L := [[1,2,3],[4,5,6]]:
> intimage(Q,L);
```

$$[[2,\ 22],\ [0,\ 36]]$$

1.10. `intiscontained`. This function checks whether or not a given vector $v \in \mathbb{Z}^m$ is contained in a given sublattice $L \subset \mathbb{Z}^m$.

1.10.1. *Usage.*

Input: a list of integers, and a listlist of integers [a matrix].

Output: a boolean.

1.10.2. *Example.*

```
> A := [[2,0,0],[0,2,0]]:
> v := [1,1,0]:
> intiscontained(v,A);
```

false

1.11. `intkernel`. Given a linear map $p\colon \mathbb{Z}^m \to \mathbb{Z}^n$, this function computes an integer basis for the kernel of p (consisting of the zero vector in case of an injection). Depending on which of the possible options `list`, `matrix` is chosen, the output is a listlist of integers or a matrix (the default is a listlist). If the possible option `red` is chosen, then the basis for the kernel will be LLL-reduced.

1.11.1. *Usage.*

Input: A listlist of integers, representing the images $p(e_i) \in \mathbb{Z}^n$ of the canonical base vectors $e_i \in \mathbb{Z}^n$
[an integral $(n \times m)$-matrix, representing $p\colon \mathbb{Z}^m \to \mathbb{Z}^n$ with respect to the canonical bases].

Option(s): `list`, `matrix`, `red`.

Output: A listlist of integers, representing the vectors of a basis for the kernel of p
[an integral $(m \times l)$-matrix, the columns of which form a basis of the kernel of p].

1.11.2. *Example.*

```
> LL := [[1,2,3],[4,5,6],[7,8,9],[10,11,12]]:
> intkernel(LL);
```

$$[[1, -2, 1, 0], [2, -3, 0, 1]]$$

```
> intkernel(LL,matrix,red);
```

$$\begin{bmatrix} 1 & 1 \\ -2 & -1 \\ 1 & -1 \\ 0 & 1 \end{bmatrix}$$

1.12. `intpreimage`. Given a linear map $p\colon \mathbb{Z}^m \to \mathbb{Z}^n$ and a sublattice $L \subset \mathbb{Z}^n$, this function determines a basis for the preimage $p^{-1}(L)$. Depending on which of the possible options `list`, `matrix` is chosen, the output is a listlist of integers a matrix (the default is a listlist). If the possible option `red` is chosen, then the output forms a LLL-reduced system.

1.12.1. *Usage.*

Input: A pair (P, B), where P is a listlist of integers [a matrix] representing the linear map $p\colon \mathbb{Z}^m \to \mathbb{Z}^n$ and B is a listlists of integers [a matrix] representing a system of generators for the sublattice $L \subset \mathbb{Z}^n$.

Option(s): `list`, `matrix`, `red`.

Output: A listlist of integers representing the vectors of a basis of the preimage $p^{-1}(L) \subset \mathbb{Z}^m$.
[an integral $(m \times l)$-matrix the columns of which form a basis of the preimage $p^{-1}(L) \subset \mathbb{Z}^m$].

1.12.2. *Example.*

```
> A := linalg[matrix]([[2,6,10],[4,8,12]]);
```

$$A := \begin{bmatrix} 2 & 6 & 10 \\ 4 & 8 & 12 \end{bmatrix}$$

```
> B := linalg[matrix]([[1],[0]]);
```

$$B := \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

```
> P := intpreimage(A,B,matrix,red);
```

$$P := \begin{bmatrix} 1 & -2 \\ 1 & 1 \\ -1 & 0 \end{bmatrix}$$

1.13. `intprojection`. Given a set of vectors in $\mathbb{Z}^m$, this function provides a linear map $p\colon \mathbb{Z}^m \to \mathbb{Z}^n$ having the primitive span of these vectors its kernel. Depending on which of the possible options `list`, `matrix` is chosen, the output is a listlist of integers or a matrix (the default is a listlist). If the possible option `red` is chosen, then the rows of the matrix of p with respect to the canonical bases form an LLL-reduced system.

1.13.1. *Usage.*

Input: A listlist of integers representing vectors in $\mathbb{Z}^m$
[an integral $(m \times l)$-matrix].

Option(s): `list`, `matrix`, `red`.

Output: A listlist of integers representing the images $p(e_i) \in \mathbb{Z}^n$ of the canonical base vectors $e_i \in \mathbb{Z}^m$
[an integral $(n \times m)$-matrix representing the map $p\colon \mathbb{Z}^m \to \mathbb{Z}^n$ with respect to the canonical bases].

1.13.2. *Example.*

```
> LL := [[1,2,3,4],[5,6,7,8]]:
> intprojection(LL);
```

$$[[1, 2], [-2, -3], [1, 0], [0, 1]]$$

```
> intprojection(LL,matrix,red);
```

$$\begin{bmatrix} 1 & -2 & 1 & 0 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

1.14. `intsection`. Given a linear surjection $p\colon \mathbb{Z}^m \to \mathbb{Z}^n$, this function computes an integer section $s\colon \mathbb{Z}^n \to \mathbb{Z}^m$ of p. Depending on which of the possible options `list`, `matrix` is chosen, the output is a listlist of integers or a matrix (the default is a listlist).

1.14.1. *Usage.*

Input: A listlist of integers, representing the images $p(e_i) \in \mathbb{Z}^n$ of the canonical base vectors $e_i \in \mathbb{Z}^n$
[an integral $(m \times n)$-matrix, representing $p\colon \mathbb{Z}^m \to \mathbb{Z}^n$ with respect to the canonical bases].

Option(s): `list`, `matrix`.

Output: A listlist of integers, representing the images $s(e_i)$ of the canonical base vectors $\mathbb{Z}^n$
[an integral $(n \times m)$-matrix, representing $s\colon \mathbb{Z}^n \to \mathbb{Z}^m$].

1.14.2. *Example.*

```
> LL := [[1,2],[3,4],[4,5],[6,7]]:
> intsection(LL);
```

$$[[-2,\ 1,\ 0,\ 0],\ [3,\ -3,\ 0,\ 1]]$$

1.15. `isprimitive`. Given a set of integral m-vectors, this function tests if these vectors generate a primitive sublattice in $\mathbb{Z}^m$, i.e., a direct summand of $\mathbb{Z}^m$.

1.15.1. *Usage.*

Input: A listlist of integers representing a set of vectors of $\mathbb{Z}^m$
[an integral $(m \times l)$-matrix].

Output: A boolean.

1.15.2. *Example.*

```
> A := [[1,2,3],[4,5,6]]:
> isprimitive(A);
```

false

1.16. `issurjective`. This function test, if a given a linear map $p\colon \mathbb{Z}^m \to \mathbb{Z}^n$ is surjective.

1.16.1. *Usage.*

Input: A listlist of integers the images $p(e_i) \in \mathbb{Z}^n$ of the canonical base vectors $e_i \in \mathbb{Z}^m$
[an integral $(n \times m)$-matrix representing p with respect to the canonical bases].

Output: A boolean.

1.16.2. *Example.*

```
> LL := [[1,2],[3,4],[5,6]]:
> issurjective(LL);
```

false

1.17. `latticebasis`. Given a set of integral m-vectors, this function provides a basis for the sublattice generated by these vectors. Depending on which of the possible options `list`, `matrix` is chosen, the output is a listlist of integers or a matrix (the default is a matrix). If the possible option `red` is chosen, then the output forms a LLL-reduced system.

1.17.1. *Usage.*

Input:	A listlist of integers representing a set of vectors in $\mathbb{Z}^m$ [an integral $(m \times l)$-matrix].
Option(s):	`list`, `matrix`, `red`.
Output:	A listlist of integers [an integral $(m \times l')$-matrix].

1.17.2. *Examples.*

```
> LL := [[1,2,3],[4,5,6]]:
> latticebasis(LL);
```

$$[[1, 2, 3], [0, 3, 6]]$$

1.18. `matrix2cols`. Given a matrix, this function provides a list of the columns of the given matrix. Note that using the built-in maple procedure `convert` provides the rows of the respective matrix.

1.18.1. *Usage.*

Input:	An integral $(m \times l)$-matrix
Output:	A listlist of integers.

1.18.2. *Example.*

```
> A := matrix([[1,2,3],[4,5,6]]);
```

$$A := \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

```
> matrix2cols(A);
```

$$[[1, 4], [2, 5], [3, 6]]$$

1.19. `primitivespan`. Given a set of integral m-vectors, this function provides a basis for the minimal primitive sublattice containing these vectors. Depending on which of the possible options `list`, `matrix` is chosen, the output is a listlist of integers or a matrix (the default is a matrix). If the possible option `red` is chosen, then the output forms a LLL-reduced system.

1.19.1. *Usage.*

Input:	A listlist of integers representing a set of vectors in $\mathbb{Z}^m$ [an integral $(m \times l)$-matrix].
Option(s):	`list`, `matrix`, `red`.
Output:	A listlist of integers [an integral $(m \times l')$-matrix].

1.19.2. *Example.*

```
> LL := [[1,2,3],[4,5,6]]:
> primitivespan(LL);
```

$$[[1, 2, 3], [1, 1, 1]]$$

2. Functions on convex polyhedra

2.1. Introductory remarks. The `TorDiv` package provides functions for calculations with bunches of cones and also some general functions on conves polyhedra complementing the `convex` package. A bunch of cones is defined via an integral liner map $\mathbb{Z}^m \to \mathbb{Z}^k$ and a list of cones in $\mathbb{Z}^k$. For the ambient projected cone $(E \xrightarrow{Q} K, \gamma)$, we always take $E = \mathbb{Z}^m$ and $\gamma = \text{cone}(e_1, \ldots, e_m)$.

2.2. `bunch`. Given a linear map of lattices and a list of cones in the image lattice, this function checks if these data form a bunch in a projected cone and then declares a bunch of cones. To avoid the check, one may use the option `nocheck`. In order to retrieve explicitly the projection and the list of cone, use the functions `bunchprojection` and `bunchcones`.

2.2.1. *Usage.*

Input:	A matrix and a list of cones.
Option(s):	`nocheck`.
Output:	A bunch of cones: `BUNCH(m,k,l)`, where `m` is the dimension of the space projected from, `k` is the dimension of the weight space and `l` is the number of cones in the bunch.

2.2.2. *Example.*

```
> Q := linalg[matrix]([[1,3,5,7,0],[2,4,6,8,1]]);
```

$$Q := \begin{bmatrix} 1 & 3 & 5 & 7 & 0 \\ 2 & 4 & 6 & 8 & 1 \end{bmatrix}$$

```
> C := poshull([3,4],[5,6]):
> CL := [C]:
> B := bunch(Q,CL);
```

$$B := \text{BUNCH}(5,\ 2,\ 1)$$

2.3. `bunchcones`. Given a bunch of cones, this function provides the associated list of cones.

2.3.1. *Usage.*

Input:	A bunch of cones.
Output:	A list of cones.

2.3.2. *Example.*

```
> Q := linalg[matrix]([[1,3,5,7,0],[2,4,6,8,1]]);
```

$$Q := \begin{bmatrix} 1 & 3 & 5 & 7 & 0 \\ 2 & 4 & 6 & 8 & 1 \end{bmatrix}$$

```
> C := poshull([3,4],[5,6]):
> CL := [C]:
> B := bunch(Q,CL):
> bunchcones(B);
```

$$[\text{CONE}(2,\ 2,\ 0,\ 2,\ 2)]$$

2.4. `bunchprojection`. Given a bunch of cones, this function provides the projection matrix associated to the bunch.

2.4.1. *Usage.*

Input: A bunch of cones.

Output: A list of cones.

2.4.2. *Example.*

```
> Q := linalg[matrix]([[1,3,5,7,0],[2,4,6,8,1]]);
```

$$Q := \begin{bmatrix} 1 & 3 & 5 & 7 & 0 \\ 2 & 4 & 6 & 8 & 1 \end{bmatrix}$$

```
> C := poshull([3,4],[5,6]):
> CL := [C]:
> B := bunch(Q,CL):
> bunchprojection(B);
```

$$\begin{bmatrix} 1 & 3 & 5 & 7 & 0 \\ 2 & 4 & 6 & 8 & 1 \end{bmatrix}$$

2.5. `bunch2fan`. Given a standard bunch of cones, this function computes the corresponding fan.

As an optional second argument, one can fix an *associated pair* for the computation; this is a pair $[P, Q]$ of integral matrices describing the usual exact sequences, mutually dual to each other:

$$0 \longrightarrow \mathbb{Z}^l \longrightarrow \mathbb{Z}^m \xrightarrow{P} \mathbb{Z}^n \longrightarrow 0,$$

$$0 \longleftarrow \mathbb{Z}^l \xleftarrow[Q]{} \mathbb{Z}^m \longleftarrow \mathbb{Z}^n \longleftarrow 0,$$

where $Q\colon \mathbb{Z}^m \to \mathbb{Z}^l$ is the bunch projection. Similarly, if an unassigned variable is entered as second argument, then the procedure assigns to this variable the associated pair used in the computation.

2.5.1. *Usage.*

Input: a standard bunch; optionally: an associated pair [an unassigned variable].

Output: a fan. [if the second argument is an unassigned variable, it is set to an associated pair]

2.5.2. *Example.*

```
> Q := linalg[matrix]([[1,1,1]]):
> C := poshull([1]):
> B := bunch(Q,[C]);
```

$$B := \mathrm{BUNCH}(3,\ 1,\ 1)$$

```
> bunch2fan(B);
```

$$\mathrm{FAN}(2,\ 2,\ 0,\ 3,\ [0,\ 3])$$

2.6. `cocoref`. Given a set L of vectors in $\mathbb{Z}^m$, this function computes a fan: the coarsest common refinement of all possible triangulations of the set L.

2.6.1. *Usage.*

Input: a listlist of integers [a matrix].

Output: a fan.

2.6.2. *Example.*

```
> L := [[1,0,0],[0,1,0],[1,0,1],[0,1,1]]:
> F :=cocoref(L);
```

$$F := \text{FAN}(3,\ 3,\ 0,\ 5,\ [0,\ 0,\ 4])$$

```
> rays(F);
```

$$\{[1,\ 1,\ 1],\ [1,\ 0,\ 0],\ [0,\ 1,\ 0],\ [1,\ 0,\ 1],\ [0,\ 1,\ 1]\}$$

2.7. `covcoll`. Given a bunch of cones, this function determines the corresponding covering collection.

2.7.1. *Usage.*

Input: A bunch

Output: A list of cones.

2.7.2. *Example.*

```
> Q := linalg[matrix]([[1,1,1]]);
```

$$Q := \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$$

```
> C := poshull([1]):
> B := bunch(Q,[C]):
> covcoll(B);
```

[CONE(3, 1, 0, 1, 1), CONE(3, 1, 0, 1, 1), CONE(3, 1, 0, 1, 1)]

2.8. `dualface`. Given a cone σ and a face $\tau \preceq \sigma$ [a list $\tau_1, \ldots, \tau_r$ of faces $\tau_i \preceq \sigma$], this function returns a pair a pair $[\sigma^\vee, \tau^*]$, where $\sigma^\vee$ is the dual cone and $\tau^* \preceq \sigma^\vee$ is the face corresponding to $\tau \preceq \sigma$ [a pair $[\sigma^\vee, [\tau_1^*, \ldots, \tau_r^*]]$, where $\sigma^\vee$ is the dual cone and $\tau_i^* \preceq \sigma^\vee$ are the faces corresponding to $\tau_i \preceq \sigma$].

2.8.1. *Usage.*

Input: a pair of cones [a cone and a list of cones].

Output: a pair of cones [a cone and a list of cones].

2.8.2. *Example.*

```
> C := posorthant(3):
> F1 := poshull([1,0,0]):
> dualface(C,F1);
```

[CONE(3, 3, 0, 3, 3), CONE(3, 2, 0, 2, 2)]

```
> F2 := poshull([0,1,0],[0,0,1]):
> dualface(C,[F1,F2]);
```

[CONE(3, 3, 0, 3, 3), [CONE(3, 2, 0, 2, 2), CONE(3, 1, 0, 1, 1)]]

2.9. `fan2bunch`. Given a *standard fan* Δ in $\mathbb{Z}^n$, i.e., the primitive vectors of the rays of Δ generate $\mathbb{Z}^n$ and Δ cannot be enlarged without adding new rays, this function computes the correponding bunch of cones. Some checks for bunch properties are performed by default; they can be surpressed by the option `nocheck`.

As an optional second argument, one can fix an *associated pair* for the computation; this is a pair $[P, Q]$ of integral matrices describing the usual exact sequences, mutually dual to each other:

$$0 \longrightarrow \mathbb{Z}^l \longrightarrow \mathbb{Z}^m \xrightarrow{P} \mathbb{Z}^n \longrightarrow 0,$$

$$0 \longleftarrow \mathbb{Z}^l \xleftarrow[Q]{} \mathbb{Z}^m \longleftarrow \mathbb{Z}^n \longleftarrow 0,$$

where P sets up a one-to-one correspondence between the canonical basis vectors of $\mathbb{Z}^m$ and the primitive generators of the rays of Δ. Similarly, if an unassigned variable is entered as second argument, then the procedure assigns to this variable the associated pair used in the computation.

2.9.1. *Usage.*

Input:	a fan; optionally: an associated pair [an unassigned variable].
Option(s):	`nocheck`.
Output:	a bunch [if the additional argument is an unassigned variable it is set to an associated pair].

2.9.2. *Example.*

```
> F := projspace(2);
```

$$F := \mathrm{FAN}(2,\ 2,\ 0,\ 3,\ [0,\ 3])$$

```
> B := fan2bunch(F);
```

$$B := \mathrm{BUNCH}(3,\ 1,\ 1)$$

2.10. `intersectrelint`. Given two cones, this functions checks whether their relative interiors have a nonempty intersection.

2.10.1. *Usage.*

Input:	A pair of cones.
Output:	A boolean.

2.10.2. *Example.*

```
> C1 := poshull([1,0],[0,1]):
> C2 := poshull([1,1],[0,-1]):
> intersectrelint(C1,C2);
```

true

2.11. `isremovableray`. Given a fan and a vector generating a ray of this fan, the function checks if replacing the star of this ray with the convex cone generated by its support gives again a fan.

2.11.1. *Usage.*

Input:	a pair [fan, vector].
Output:	A boolean.

2.11.2. *Example.*

```
> F := fan(poshull([1, 0], [1, 1]), poshull([1, 1], [0, 1]));
```

$$F := FAN\,(2, 0, [0, 2])$$

```
> isremovableray(F, [1, 1]);
```

true

2.12. `isstandard`. Given a bunch of cones, this function checks whether or not it is a standard bunch in the associated projected cone.

2.12.1. *Usage.*

Input: a bunch.

Output: a boolean.

2.12.2. *Example.*

```
> Q := cols2matrix([[1,0],[1,0],[0,1]]);
```

$$Q := \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

```
> C := poshull([1,0],[0,1]):
> CL := [C]:
> B := bunch(Q,CL):
> isstandard(B);
```

false

2.13. `relintcontains`. Given two cones, this function checks whether or not the relative interior of the first one contains the relative interior of the second one.

2.13.1. *Usage.*

Input: a pair of cones.

Output: a boolean.

2.13.2. *Example.*

```
> C1 := poshull([1,0],[0,1]):
> C2 := poshull([1,0]):
> relintcontains(C1,C2);
```

false

2.14. `relintiscontained`. Given two cones, this function checks whether or not the relative interior of the first one is contained in the relative interior of the second one.

2.14.1. *Usage.*

Input: a pair of cones.

Output: a boolean.

2.14.2. *Example.*

```
> C1 := poshull([1,0]):
> C2 := poshull([1,0],[0,1]):
> relintiscontained(C1,C2);
```

false

2.15. `removeray`. Given a fan and a vector generating a ray of this fan, the function computes the fan obtained by replacing the star of this ray with the convex cone generated by its support; it ends with an error if the ray is not removable.

2.15.1. *Usage.*

Input: a pair [fan, vector].

Output: a fan.

2.15.2. *Example.*

```
> F := fan(poshull([1, 0], [1, 1]), poshull([1, 1], [0, 1]));
```

$$F := FAN\,(2, 0, [0, 2])$$

```
> removeray(F, [1, 1]);
```

$$FAN\,(2, 0, [0, 1])$$

3. Functions on Toric Geometry

3.1. Introductory remarks. The `TorDiv` package provides several functions on toric varieties. In most cases, the toric varieties must be nondegenerate, twocomplete, and have a free divisor class group. Such a toric variety can be encoded either by a standard bunch or by a *standard fan*, i.e., a fan that cannot be enlarged without adding new rays, and the primitive vectors of the rays of which generate the ambient lattice.

Most of the functions of the `TorDiv` package can handle both types of input, bunches and fans. However, the calculations are mainly based on bunches of cones. Hence, entering a bunch makes the computation in general much faster, because then internal conversion routines are avoided.

In many functions one, can fix, by entering an additional argument, an *associated pair* $[P, Q]$ of integral matrices for the computation. Recall that, to a standard fan Δ in $\mathbb{Z}^n$ one associates an exact sequence of integral linear maps

$$0 \longrightarrow \mathbb{Z}^l \xrightarrow{I} \mathbb{Z}^m \xrightarrow{P} \mathbb{Z}^n \longrightarrow 0,$$

where $P\colon \mathbb{Z}^m \to \mathbb{Z}^n$ induces a one-to-one correspondence between the canonical basis vectors of $\mathbb{Z}^m$ and the primitive lattice vectors of the rays of Δ. Analogously, to a standard bunch of cones, one associates an exact sequence of integral linear maps completing the bunch projection $Q\colon \mathbb{Z}^m \to \mathbb{Z}^l$:

$$0 \longleftarrow \mathbb{Z}^l \xleftarrow{Q} \mathbb{Z}^m \xleftarrow{J} \mathbb{Z}^n \longleftarrow 0.$$

By construction, these two sequences are dual to each other, and they are determined by the pair $[P, Q]$. We say that $[P, Q]$ is an *associated pair* to the fan or, respectively, the bunch of cones under consideration.

Note, that fixing an associated pair means fixing a numbering of the invariant divisors of the corresponding toric variety, and fixing a choice of coordinates for the divisor class group.

3.2. `amplecone`. Given a standard bunch [a standard fan], this function computes the closure of the cone of ample divisor classes of the corresponding toric variety. In case of a fan as argument, an associated pair can optionally be prescribed. If the ample cone is empty, i.e., the variety is not quasiprojective, then the function returns an error.

3.2.1. *Usage.*

Input: a standard bunch [a standard fan; optionally: an associated pair].
Output: a cone.

3.2.2. *Example.*

```
> F := projspace(2);
```

$$F := \text{FAN}(2,\ 2,\ 0,\ 3,\ [0,\ 3])$$

```
> amplecone(F);
```

$$\text{CONE}(1,\ 1,\ 0,\ 1,\ 1)$$

3.3. `canonclass`. Given a standard bunch [a standard fan], this function computes the canonical divisor class of the corresponding toric variety. In case of a fan as argument, an associated pair can optionally be prescribed.

3.3.1. *Usage.*

Input: a standard bunch [a standard fan; optionally: an associated pair].
Output: a list of integers.

3.3.2. *Example.*

```
> F := projspace(2);
```

$$F := \text{FAN}(2,\ 2,\ 0,\ 3,\ [0,\ 3])$$

```
> canonclass(F);
```

$$[-3]$$

3.4. `cartierdiv`. Given a standard bunch [a standard fan], this function computes the group of invariant Cartier divisors of the corresponding toric variety. In case of a fan as argument, an associated pair can optionally be prescribed.

3.4.1. *Usage.*

Input: a standard bunch [a standard fan; optionally: an associated pair].
Output: a listlist of integers.

3.4.2. *Example.*

```
> F := projspace(2);
```

$$F := \text{FAN}(2,\ 2,\ 0,\ 3,\ [0,\ 3])$$

```
> cartierdiv(F);
```

$$[[1,\ 0,\ 0],\ [0,\ 1,\ 0],\ [0,\ 0,\ 1]]$$

3.5. `coxconstr`. Given a nondegenerate fan Δ in a lattice $\mathbb{Z}^n$, this function computes a Cox construction. By default the output is a projection matrix $P\colon \mathbb{Z}^m \to \mathbb{Z}^n$, and a list of cones. The option `fan` transforms the latter list into a fan. Moreover, for a standard fan Δ an associated pair can optionally be prescribed.

3.5.1. *Usage.*

Input: a fan; optionally: an associated pair.
Option: `fan`.
Output: a matrix, and a list of cones [a fan].

3.5.2. *Example.*

```
> F := projspace(2);
```

$$F := \text{FAN}(2,\ 2,\ 0,\ 3,\ [0,\ 3])$$

```
> coxconstr(F,fan);
```

$$[\begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \end{bmatrix}, \text{FAN}(3,\ 3,\ 0,\ 3,\ [0,\ 3,\ 0])]$$

3.6. `effectivecone`. Given a standard bunch [a standard fan], this function computes the cone of effective divisor classes of the associated toric variety. In case of a fan as argument, an associated pair can optionally be prescribed.

3.6.1. *Usage.*

Input: a standard bunch [a standard fan; optionally: an associated pair].

Output: a cone.

3.6.2. *Example.*

```
> F := projspace(2);
```

$$F := \text{FAN}(2,\ 2,\ 0,\ 3,\ [0,\ 3])$$

```
> effectivecone(F);
```

$$\text{CONE}(1,\ 1,\ 0,\ 1,\ 1)$$

3.7. `freecovering`. Given a nondegenerate fan Δ this function computes a standard fan Δ' such that there is a finite toric morphism $X' \to X$ of the corresponding toric varieties. Note that X' has a free divisor class group.

3.7.1. *Usage.*

Input: A fan.

Output: A fan.

3.7.2. *Example.*

```
> F := fan(poshull([1,0],[1,2]));
```

$$F := \text{FAN}(2,\ 0,\ 1,\ [0,\ 1])$$

```
> CF := freecovering(F);
```

$$CF := \text{FAN}(2,\ 0,\ 1,\ [0,\ 1])$$

```
> rays(CF);
```

$$\{[1,\ 1],\ [1,\ 0]\}$$

3.8. `isdivisorial`. Given a nondegenerate fan, this function checks, whether or not the corresponding toric variety X is divisorial, i.e., is covered by affine open subsets of the form $X \setminus D$ with an effective Cartier divisor D.

3.8.1. *Usage.*

Input: a fan.

Output: a boolean.

3.8.2. *Example.*

```
> F1 := projspace(2);
```

$$F := \text{FAN}(2,\ 0,\ 1,\ [0,\ 3])$$

```
> isdivisorial(F1);
```

true

3.9. `isfano`. Given a standard bunch [a standard fan], this function checks whether or not the corresponding toric variety is a Fano variety, i.e., the anticanonical divisor is Cartier and ample.

3.9.1. *Usage.*

Input: a standard bunch [a standard fan].

Output: a boolean.

3.9.2. *Example.*

```
> F := hirzebruch(1);
```

$$F := \text{FAN}(2,\ 2,\ 0,\ 4,\ [0,\ 4])$$

```
> isfano(F);
```

true

```
> F := hirzebruch(2);
```

$$F := \text{FAN}(2,\ 2,\ 0,\ 4,\ [0,\ 4])$$

```
> isfano(F);
```

false

3.10. `isgorenstein`. Given a standard bunch [a standard fan], this function checks whether or not the corresponding toric variety is a Gorenstein variety, i.e., the anticanonical divisor is Cartier.

3.10.1. *Usage.*

Input: a standard bunch [a standard fan].

Output: a boolean.

3.10.2. *Example.*

```
> F := projspace(2);
```

$$F := \text{FAN}(2,\ 2,\ 0,\ 3,\ [0,\ 3])$$

```
> isgorenstein(F);
```

true

3.11. `isquasiprojective`. Given a fan, this function tests whether or not the corresponding toric variety is quasiprojective.

3.11.1. *Usage.*

Input: a fan.

Output: a boolean.

3.11.2. *Example.*

```
> F := projspace(3);
```

$$F := \text{FAN}(3,\ 3,\ 0,\ 4,\ [0,\ 0,\ 4])$$

```
> isquasiprojective(F);
```

true

```
> F := freebentcube();
```

$$F := \text{FAN}(3,\ 3,\ 0,\ 8,\ [0,\ 0,\ 6])$$

```
> isquasiprojective(F);
```

false

3.12. `isprojective`. Given a fan, this function tests whether or not the corresponding toric variety is quasiprojective.

3.12.1. *Usage.*

Input: a fan.

Output: a boolean.

3.12.2. *Example.*

```
> F := projspace(3);
```

$$F := \text{FAN}(3,\ 3,\ 0,\ 4,\ [0,\ 0,\ 4])$$

```
> isprojective(F);
```

true

```
> F := freebentcube();
```

$$F := \text{FAN}(3,\ 3,\ 0,\ 8,\ [0,\ 0,\ 6])$$

```
> isprojective(F);
```

false

3.13. `istwocomplete`. Given a nondegenerate fan, this function checks whether or not the corresponding toric variety X is 2-complete, i.e. does not admit open toric embeddings $X \subset X'$ with $X' \setminus X$ nonempty of codimension at least two.

3.13.1. *Usage.*

Input: A fan.

Output: A boolean.

3.13.2. *Example.*

```
> C1 := posorthant(2):
> C2 := poshull([-1,-1]):
> F := fan(C1,C2);
```

$$F := \text{FAN}(2,\ 2,\ 0,\ 3,\ [1,\ 1])$$

```
> istwocomplete(F);
```

false

3.14. `kajiwaraconstr.` Given a nondegenerate fan Δ in a lattice $\mathbb{Z}^n$, corresponding to a divisorial toric variety, this function computes Kajiwara's quotient presentation, see [**11**]. By default the output is a projection matrix $P\colon \mathbb{Z}^m \to \mathbb{Z}^n$, a cone and a list of faces of that cone. The option `fan` transforms the list of faces into a fan. If the input fan does not correspond to a divisorial toric variety, the function produces an error.

3.14.1. *Usage.*

Input: a fan.
Options: `fan`.
Output: A triple: matrix, cone, list of faces [fan].

3.14.2. *Example.*

```
> F1:=wprojspace(1,1,2);
```

$$F1 := \mathrm{FAN}(2,\ 0,\ 1,\ [0,\ 3])$$

```
> kajiwaraconstr(F1);
```

$$[\begin{bmatrix} 0 & 0 & -1 \\ 1 & -1 & 1 \end{bmatrix}, \mathrm{CONE}(3,\ 3,\ 0,\ 3,\ 3),$$
$$[\mathrm{CONE}(3,\ 2,\ 0,\ 2,\ 2), \mathrm{CONE}(3,\ 2,\ 0,\ 2,\ 2), \mathrm{CONE}(3,\ 2,\ 0,\ 2,\ 2)]]$$

3.15. `movingcone.` Given a standard bunch [a standard fan], this function computes the cone of moving divisor classes of the associated toric variety, i.e. divisors which contain no prime divisors in its base locus. In case of a fan as argument, an associated pair can optionally be prescribed.

3.15.1. *Usage.*

Input: a standard bunch [a standard fan; optionally: an associated pair].
Output: a cone.

3.15.2. *Example.*

```
> F := projspace(2);
```

$$F := \mathrm{FAN}(2,\ 2,\ 0,\ 3,\ [0,\ 3])$$

```
> movingcone(F);
```

$$\mathrm{CONE}(1,\ 1,\ 0,\ 1,\ 1)$$

3.16. `picardgroup.` Given a standard bunch [a standard fan], this function computes the Picard group of the corresponding toric variety. In case of a fan as argument, an associated pair can optionally be prescribed.

3.16.1. *Usage.*

Input: a standard bunch [a standard fan; optionally: an associated pair].
Output: a listlist of integers.

3.16.2. *Example.*

```
> F := projspace(2);
```

$$F := \mathrm{FAN}(2,\ 2,\ 0,\ 3,\ [0,\ 3])$$

```
> picardgroup(F);
```

$$[[1]]$$

3.17. `samplecone`. Given a standard bunch [a standard fan], this function computes the cone of semiample divisor classes of the corresponding toric variety, i.e. divisor classes with empty stable base locus. In case of a fan as argument, an associated pair can optionally be prescribed. An associated pair can be fixed via an optional argument.

3.17.1. *Usage.*

Input: a standard bunch [a standard fan; optionally: an associated pair].
Output: a cone.

3.17.2. *Example.*

```
> F := projspace(2);
```

$$F := \mathrm{FAN}(2,\ 2,\ 0,\ 3,\ [0,\ 3])$$

```
> samplecone(F);
```

$$\mathrm{CONE}(1,\ 1,\ 0,\ 1,\ 1)$$

3.18. `twocompletion`. Given a nondegenerate fan Δ, this procedure computes a fan Δ' such that the the toric variety X associated to Δ is an open toric subset of the toric variety X' associated to Δ', the complement $X' \setminus X$ is codimension at least two in X' and X' is 2-complete. Note that in general X' is not unique.

3.18.1. *Usage.*

Input: a fan.
Output: a fan.

3.18.2. *Example.*

```
> C1 := poshull([1,0]):
> C2 := poshull([-1,-1]):
> C3 := poshull([0,1]):
> F := fan(C1,C2,C3);
```

$$F := \mathrm{FAN}(2,\ 2,\ 0,\ 3,\ [3,\ 0])$$

```
> twocompletion(F);
```

$$\mathrm{FAN}(2,\ 2,\ 0,\ 3,\ [0,\ 3])$$

4. Functions on Geometric Invariant Theory

4.1. Introductory remarks. The `TorDiv` package provides several functions on Geometric Invariant Theory for effective diagonal torus actions on $\mathbb{K}^m$.

Usually, an effective diagonal action of a torus $T = (\mathbb{K}^*)^l$ is encoded by the list of weights $w_1, \ldots, w_m \in \mathbb{Z}^l$ of the variables $T_1, \ldots, T_m$. Alternatively, the action can be encoded by a matrix Q having the weights $w_1, \ldots, w_m$ as columns. Effectivity merely means that the weights $w_1, \ldots, w_m$ generate $\mathbb{Z}^l$ as a lattice.

In cases where it can be of importance for the computations, one can as well enter a torus action by providing an *associated pair* $[P, Q]$ of integral matrices. In such a pair, the columns of Q are precisely the weights $w_1, \ldots, w_m$, and P provides an exact sequence:

$$0 \longrightarrow \mathbb{Z}^l \xrightarrow{Q^\top} \mathbb{Z}^m \xrightarrow{P} \mathbb{Z}^n \longrightarrow 0.$$

4.2. `admitsgoodquot`. Given a diagonal torus action on $\mathbb{K}^m$, and a fan describing an open toric subset $U \subset \mathbb{K}^m$, this function checks, whether or not U admits a good quotient with by the given torus action.

4.2.1. *Usage.*

Input: a fan and a listlist of integers [a matrix];
Output: a boolean.

4.2.2. *Example.*

```
> Q := [[-1],[1]];
```

$$Q := [[-1], [1]]$$

```
> F := fan(poshull([1,0]), poshull([0,1]));
```

$$F := \text{FAN}(2, 2, 0, 2, [2, 0])$$

```
> admitsgoodquot(F,Q);
```

$$\mathit{false}$$

4.3. `chamber`. Given a diagonal torus action on $\mathbb{K}^n$ and a weight of the torus, this function computes the corresponding GIT chamber.

4.3.1. *Usage.*

Input: a list of integers, and a listlist of integers [a matrix].
Output: a cone.

4.3.2. *Example.*

```
> Q := [[1,0,0],[0,1,0],[1,0,1],[0,1,1]];
```

$$Q := [[1, 0, 0], [0, 1, 0], [1, 0, 1], [0, 1, 1]]$$

```
> w := [2,2,1];
```

$$w := [2, 2, 1]$$

```
> chamber(w,Q);
```

$$\text{CONE}(3, 3, 0, 3, 3)$$

4.4. `gitfan`. Given a diagonal torus action on $\mathbb{K}^m$, this function computes the associated GIT fan.

4.4.1. *Usage.*

Input: a listlist of integers [a matrix].
Output: a fan.

4.4.2. *Example.*

```
> QL := [[1],[-1]];
```

$$QL := [[1], [-1]]$$

```
> F := gitfan(QL);
```

$$F := \text{FAN}(1, 1, 0, 2, [2])$$

```
> rays(F);
```

$$\{[1], [-1]\}$$

```
> QL := [[1],[1]];
```

$$QL := [[1], [1]]$$

```
> F := gitfan(QL);
```

$$F := \text{FAN}(1,\ 1,\ 0,\ 1,\ [1])$$

```
> rays(F);
```

$$\{[1]\}$$

4.5. `gitlimit`. Given a diagonal torus action on $\mathbb{K}^m$, this function computes the fan of the normalization of the limit of all Mumford quotients. The action can be entered as a list of weights [weight matrix] or as an associated pair for the action.

4.5.1. *Usage.*

Input: a listlist of integers [a matrix, an associated pair].
Output: a fan.

4.5.2. *Example.*

```
> QL := [[1],[-1]];
```

$$QL := [[1],\ [-1]]$$

```
> F := gitlimit(QL);
```

$$F := \text{FAN}(1,\ 1,\ 0,\ 1,\ [1])$$

```
> rays(F);
```

$$\{[1]\}$$

```
> QL := [[1],[1]];
```

$$QL := [[1],\ [1]]$$

```
> F := gitlimit(QL);
```

$$F := \text{FAN}(1,\ 1,\ 0,\ 2,\ [2])$$

```
> rays(F);
```

$$\{[1],\ [-1]\}$$

4.6. `quotientfan`. Given a diagonal torus action on $\mathbb{K}^m$, and a fan describing an open toric subset of $\mathbb{K}^m$, this function computes the fan of the quotient variety of the subset by the given action. The function checks whether this quotient exists, unless the optional argument `nocheck` is given. Instead of the weight matrix, an associated pair for the action can be given.

4.6.1. *Usage.*

Input: a fan, and a listlist of integers [a matrix, an associated pair].
Option(s): `nocheck`.
Output: a fan.

4.6.2. *Example.*

```
> Q := [[1],[1],[1]];
```

$$Q := [[1],\ [1],\ [1]]$$

```
> w:=[1];
```

$$w := [1]$$

```
> F := semistablepoints(w,Q);
```

$$F := \text{FAN}(3,\ 3,\ 0,\ 3,\ [0,\ 3,\ 0])$$

```
> QF := quotientfan(F,Q);
```

$$QF := \text{FAN}(2,\ 2,\ 0,\ 3,\ [0,\ 3])$$

4.7. `semistablepoints`. Given a diagonal torus action on $\mathbb{K}^m$, and a weight for this action, this function computes the fan of the open toric subvariety of $\mathbb{K}^m$ consisting of the semistable points defined by the given weight. Instead of the weight matrix, an associated pair for the action can be given.

4.7.1. *Usage.*

Input: a listlist of integers [a matrix, a pair of matrices];
Output: a fan.

4.7.2. *Example.*

```
> Q := [[1],[1],[1]];
```

$$Q := [[1],\ [1],\ [1]]$$

```
> w:=[1];
```

$$w := [1]$$

```
> F := semistablepoints(w,Q);
```

$$F := \text{FAN}(3,\ 3,\ 0,\ 3,\ [0,\ 3,\ 0])$$

4.8. `weight2bunch`. Given a diagonal torus action and a weight w of the torus, this function computes the bunch corresponding to the GIT chamber of w.

4.8.1. *Usage.*

Input: a list of integers, and a listlist of integers [a matrix].
Output: a bunch.

4.8.2. *Example.*

```
> Q := [[1,0,0],[0,1,0],[1,0,1],[0,1,1]];
```

$$Q := [[1,\ 0,\ 0],\ [0,\ 1,\ 0],\ [1,\ 0,\ 1],\ [0,\ 1,\ 1]]$$

```
> w := [2,2,1];
```

$$w := [2,\ 2,\ 1]$$

```
> B := weight2bunch(w,Q);
```

$$B := \text{BUNCH}(4,\ 3,\ 2)$$

```
> bunchcones(B);
```

$$[\text{CONE}(3,\ 3,\ 0,\ 3,\ 3),\ \text{CONE}(3,\ 3,\ 0,\ 3,\ 3)]$$

5. Functions on Polyhedral Divisors

5.1. Introductory remarks. `TorDiv` provides several function to calculate polyhedral divisors for subtorus actions on toric varieties; for the theoretical background, we refer to [**1**].

5.2. `base`. Procedure to extract the fan of the base variety from the type POLDIV (polyhedral divisor).

Input: a polyhedral divisor.
Output: output: a fan.

5.2.1. *Example.*

```
> P := ppdivisor([[1],[1]])[2];
```

$$P := POLDIV(1,1)$$

```
> base(P);
```

$$FAN(1,0,[2])$$

5.3. `basedim`. Procedure to extract the dimension of the base variety from the type POLDIV (polyhedral divisor).

Input: a polyhedral divisor.
Output: output: an integer.

5.3.1. *Example.*

```
> P := ppdivisor([[1],[1]])[2];
```

$$P := POLDIV(1,1)$$

```
> basedim(P);
```

$$1$$

5.4. `coefficients`. Procedure to extract the list of polyhedral coefficients from the type POLDIV (polyhedral divisor).

Input: a polyhedral divisor.
Output: output: a list of pairs (vector, polyhedron).

5.4.1. *Example.*

```
> P := ppdivisor([[1],[1]])[2];
```

$$P := POLDIV(1,1)$$

```
> coefficients(P);
```

$$[[[1], POLYHEDRON(1,1,0,[1,1],[1])], [[-1], POLYHEDRON(1,1,0,[1,1],[1])]]$$

5.5. `imagefan`. Computes the coarsest common refinement of the images of all faces of a cone σ under a projection Q of lattices. The cone σ is assumed to be of full dimension but not necessarily pointed.

5.5.1. *Usage.*

Input: a cone and a listlist of integers [a matrix];
Output: a fan.

5.5.2. *Example.*

```
> sigma := poshull([1,0,0],[0,1,0],[0,1,1],[1,0,1]);
```

$$\sigma := CONE(3,3,0,4,4)$$

```
> Q := [[1,0],[1,1],[0,2]];
```

$$Q := [[1,0],[1,1],[0,2]]$$

```
> imagefan(sigma, Q);
```

$$FAN(2,0,[0,3])$$

5.6. `ppdivisor`. Computes the pp-divisor of diagonal torus action on $\mathbb{C}^n$.

Input: a listlist of integers [a matrix, an associated pair];
Output: a fan and a polyhedral divisor.

5.6.1. *Example.*

```
> Q := [[1],[1]];
```

$$Q := [[1],[1]]$$

```
> ppdivisor(Q);
```

$$[FAN\,(1,0,[2])\,,POLDIV\,(1,1)]$$

5.7. `ppfan`. Computes the fan of pp-divisors for a given semiprojective toric variety X and subtorus action. If X is affine, the result is a minimal pp-divisor for X.

Input: a fan [a cone] and a listlist of integers [a matrix, an associated pair];
Output: a fan and a a list of polyhedral divisors.

5.7.1. *Example.*

```
> X := projspace(2);
```

$$X := FAN\,(2,0,[0,3])$$

```
> Q := [[1],[0]];
```

$$Q := [[1],[0]]$$

```
> ppfan(X,Q);
```

$$[FAN\,(1,0,[2])\,,[POLDIV\,(1,1)\,,POLDIV\,(1,1)\,,POLDIV\,(1,1)]]$$

5.8. `slice`. Computes the slice of a fan of polyhedral divisors with respect to a list of weighted divisors. If the second argument is a list of pairs (ray, weight), then missing rays are assigned the weight 0.

Input: a list of polyhedral divisors and a list of pairs (vector, integer) [a function vector $\mapsto$ integer];
Output: output: a list of polyhedra.

5.8.1. *Example.*

```
> F := ppfan(projspace(2), [[1],[0]])[2];
```

$$F := [POLDIV\,(1,1)\,,POLDIV\,(1,1)\,,POLDIV\,(1,1)]$$

```
> slice(F, [[[1],1]]);
```

$$[POLYTOPE\,(1,0,1,1)\,,POLYTOPE\,(1,0,1,1)\,,POLYTOPE\,(1,0,1,1)]$$

```
> slice(F, x -> 0);
```

$$[POLYTOPE\,(1,0,1,1)\,,POLYTOPE\,(1,0,1,1)\,,POLYTOPE\,(1,0,1,1)]$$

5.9. `tailcone`. Procedure to extract the tail cone from the type POLDIV (polyhedral divisor).

Input: a polyhedral divisor.
Output: output: a cone.

5.9.1. *Example.*

```
> P := ppdivisor([[1],[1]])[2];
```

$$P := POLDIV\,(1,1)$$

```
> tailcone(P);
```

$$CONE\,(1,1,0,1,1)$$

5.10. `torusdim`. Procedure to extract the dimension of the acting torus from the type POLDIV (polyhedral divisor).

Input: a polyhedral divisor.

Output: output: an integer.

5.10.1. *Example.*

```
> P := ppdivisor([[1],[1]])[2];
```

$$P := POLDIV\,(1, 1)$$

```
> torusdim(P);
```

$$1$$

6. Examples

6.1. Introductory remarks. The `TorDiv` package provides a couple of standard examples of complete toric varieties, including nonprojective ones. Further examples of projective toric varietes can be obtained via the `normalfan` function and the various examples of polytopes implemented in the `convex` package.

6.2. `cutpyramid`. A $\mathbb{Q}$-factorial complete nonprojective toric variety of dimension three.

6.2.1. *Usage.*

Output: a fan.

6.2.2. *Example.*

```
> F := cutpyramid();
```

$$F := \text{FAN}(3,\ 0,\ 1,\ [0,\ 0,\ 8])$$

6.3. `freebentcube`. A complete toric variety of dimension three having trivial Picard group.

6.3.1. *Usage.*

Output: a fan.

6.3.2. *Example.*

```
> F := freebentcube();
```

$$F := \text{FAN}(3,\ 0,\ 1,\ [0,\ 0,\ 6])$$

6.4. `hirzebruch`. The a-th Hirzebruch Surface.

6.4.1. *Usage.*

Input: an integer

Output: a fan.

6.4.2. *Example.*

```
> F := hirzebruch(2);
```

$$F := \text{FAN}(2,\ 0,\ 1,\ [0,\ 4])$$

```
> rays(F);
```

$$\{[1,\ 0],\ [0,\ 1],\ [0,\ -1],\ [-1,\ 2]\}$$

Bibliography

[1] K.. Altmann, J. Hausen: Polyhedral divisors and algebraic torus actions. Math. Ann. 334, 557–607 (2006)

[2] F. Berchtold, J. Hausen: Homogeneous coordinates for algebraic varieties. J. Algebra 266, No. 2, 636–670 (2003)

[3] F. Berchtold, J. Hausen: Bunches of cones in the divisor class group — A new combinatorial language for toric varieties. Int. Math. Res. Not. 2004, 261–302 (2004)

[4] A. Białynicki-Birula: Algebraic Quotients. In: R.V. Gamkrelidze, V.L. Popov (Eds.), Encyclopedia of Mathematical Sciences, Vol. 131., 1–82 (2002)

[5] A. Białynicki-Birula, J. Święcicka: A recipe for finding open subsets of vector spaces with a good quotient. Colloq. Math. 77, No. 1 (1998), 97–114

[6] H. Cohen: A course in computational algebraic number theory. Springer GTM 138, 2nd corrected printing. Berlin, Heidelberg: Springer Verlag (1995)

[7] D. Cox: The homogeneous coordinate ring of a toric variety. J. Alg. Geom. 4 (1995), 17–50

[8] M. Franz: `Convex`, a software package for maple,
`http://www-fourier.ujf-grenoble.fr/ franz/convex/`, 2004

[9] W. Fulton: Introduction to toric varieties. The William H. Roever Lectures in Geometry, Princeton University Press (1993)

[10] J. Hausen: Producing good quotients by embedding into toric varieties. Sém. et Congr. 6 (2002), 193–212

[11] T. Kajiwara: The functor of a toric variety with enough effective Cartier divisors. Tôhoku Math. J. 50, 139–157 (1998)

[12] D. Mumford, J. Fogarty, F. Kirwan: Geometric invariant theory. 3rd enl. ed.. Ergebnisse der Mathematik und ihrer Grenzgebiete. Berlin: Springer-Verlag. (1993)

[13] T. Oda: Convex Bodies and Algebraic Geometry. Ergebnisse der Mathematik und ihrer Grenzgebiete, Band 15. Springer Verlag (1988)

[14] T. Oda, H.S. Park: Linear Gale transforms and Gelfand-Kapranov-Zelevinskij Decompositions. Tôhoku Math. J., 43 (1991), 375–399

[15] J. Święcicka: Quotients of toric varieties by actions of subtori. Colloq. Math. 82, No. 1 (1999), 105–116