2024

Computational Methods for OI-Modules

Michael Morrow
University of Kentucky, michaelhmorrow98@gmail.com
Author ORCID Identifier:
https://orcid.org/0009-0007-9749-9010
Digital Object Identifier: https://doi.org/10.13023/etd.2024.189

STUDENT AGREEMENT:

I represent that my thesis or dissertation and abstract are my original work. Proper attribution has been given to all outside sources. I understand that I am solely responsible for obtaining any needed copyright permissions. I have obtained needed written permission statement(s) from the owner(s) of each third-party copyrighted matter to be included in my work, allowing electronic distribution (if such use is not permitted by the fair use doctrine) which will be submitted to UKnowledge as Additional File.

I hereby grant to The University of Kentucky and its agents the irrevocable, non-exclusive, and royalty-free license to archive and make accessible my work in whole or in part in all forms of media, now or hereafter known. I agree that the document mentioned above may be made available immediately for worldwide access unless an embargo applies.

I retain all other ownership rights to the copyright of my work. I also retain the right to use in future works (such as articles or books) all or part of my work. I understand that I am free to register the copyright to my work.

REVIEW, APPROVAL AND ACCEPTANCE

The document mentioned above has been reviewed and accepted by the student's advisor, on behalf of the advisory committee, and by the Director of Graduate Studies (DGS), on behalf of the program; we verify that this is the final, approved version of the student's thesis including all changes required by the advisory committee. The undersigned agree to abide by the statements above.

Michael Morrow, Student

Dr. Uwe Nagel, Major Professor

Dr. Benjamin Braun, Director of Graduate Studies

Computational Methods for OI-Modules

DISSERTATION

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the College of Arts and Sciences at the University of Kentucky

By
Michael H. Morrow
Lexington, Kentucky

Director: Dr. Uwe Nagel, Professor of Mathematics
Lexington, Kentucky
2024

https://orcid.org/0009-0007-9749-9010

ABSTRACT OF DISSERTATION

Computational Methods for OI-Modules

Computational commutative algebra has become an increasingly popular area of research. Central to the theory is the notion of a Gröbner basis, which may be thought of as a nonlinear generalization of Gaussian elimination. In 2019, Nagel and Römer introduced FI- and OI-modules over FI- and OI-algebras, which provide a framework for studying sequences of related modules defined over sequences of related polynomial rings. In particular, they laid the foundations of a theory of Gröbner bases for certain classes of OI-modules. In this dissertation we develop an OI-analog of Buchberger's algorithm in order to compute such Gröbner bases, as well as an OI-analog of Schreyer's theorem to compute their modules of syzygies. We also give an application of our results to the computation of free OI-resolutions, and showcase our Macaulay2 package "OIGroebnerBases.m2" which implements these constructions. Lastly, we show how our results can be tweaked to compute free FI-resolutions.

KEYWORDS: Gröbner bases, syzygies, OI-modules, free resolutions, algorithm

Michael H. Morrow

April 26, 2024

Computational Methods for OI-Modules

By
Michael H. Morrow

Dr. Uwe Nagel

Director of Dissertation

Dr. Benjamin Braun

Director of Graduate Studies

April 26, 2024

Date

To my family.

ACKNOWLEDGMENTS

I would like to thank first my advisor, Dr. Uwe Nagel, for his continued support over the last several years. Thank you for always making time to meet with me, for your invaluable feedback, and for all of our enlightening discussions.

Thank you to my family for pushing me forward and providing a safe place to fall back on when I needed it.

I would also like to thank the many friends I made along the way, and in particular those in our office 722POT. It has been a pleasure developing as a mathematician alongside you.

Thank you to my undergraduate math department at Central Washington University for preparing me for graduate school. Special thanks to Dr. Aaron Montgomery for working with me and giving me my first taste of mathematical research.

To Dale and Mary Jo Comstock, thank you for your support during my transition to graduate school.

Lastly, I would like to thank my doctoral committee members Uwe Nagel, Chris Manon, Dave Jensen, Bradley Plaster, and Solomon Harrar, for their guidance and support.

TABLE OF CONTENTS

Chapter 1 Introduction

Computing Gröbner bases and syzygies is a classical problem in commutative and computational algebra. Recall that if R is a noetherian polynomial ring over a field and F is a free module over R of finite rank, then a finite Gröbner basis of any finitely generated submodule M of F can be computed using Buchberger's algorithm. Furthermore, a theorem of Schreyer shows how to determine a finite Gröbner basis for the module of syzygies of M. These results can be used to give a constructive proof of Hilbert's Syzygy Theorem which asserts that M has a finite free resolution. We refer the reader to [**4**, Chapter 15] for more on this topic.

Gröbner bases in non-noetherian contexts have been studied in, e.g., [**1, 2, 8, 9**]. As an example, let $R = K[x_1, x_2, \ldots]$ be the polynomial ring in infinitely many variables over a field K. This ring is not noetherian since, for example, the ideal $I = \langle x_1, x_2, \ldots \rangle$ is not finitely generated. However, if we let the monoid $\mathrm{Inc}(\mathbb{N})$ of strictly increasing maps on $\mathbb{N}$ act on R by permuting indices, then I is invariant under the action of $\mathrm{Inc}(\mathbb{N})$ and is finitely generated *up to symmetry* by x_1, i.e. I is generated by the single $\mathrm{Inc}(\mathbb{N})$-orbit of x_1. In fact, the ring R is $\mathrm{Inc}(\mathbb{N})$-*noetherian*, meaning every $\mathrm{Inc}(\mathbb{N})$-invariant ideal of R is generated by finitely many $\mathrm{Inc}(\mathbb{N})$-orbits and, in particular, has a finite $\mathrm{Inc}(\mathbb{N})$-Gröbner basis (see, e.g., [**2**]). In [**13**], Nagel and Römer introduced OI-modules over OI-algebras and developed a theory of Gröbner bases in this setting which, as a consequence, gives a new proof that the ring R is $\mathrm{Inc}(\mathbb{N})$-noetherian.

The existence of Gröbner bases for OI-modules established in [**13**] is nonconstructive. In this dissertation we give an algorithm for computing such Gröbner bases in finite time. Furthermore, we use our Gröbner basis techniques to address the problem of computing the relations (or *syzygies*) among generators of an OI-module. By iterating this process, we obtain the first algorithmic procedure for computing *free resolutions* of OI-modules out to desired homological degree, which one can informally think of as a "Taylor polynomial approximation" of an OI-module. Finally, we show how to adapt our results to compute free FI-resolutions, where FI is a closely related category to OI. Our results are implemented in the package "OIGroebnerBases.m2" [**11**] for the software language *Macaulay2* [**7**]. A brief introduction to the package can be found in [**10**].

This dissertation is organized as follows. In Chapter 2 we review the necessary background on OI-modules over OI-algebras. In Chapter 3 we discuss our techniques for computing Gröbner bases and syzygies of OI-modules. In particular, we develop OI-analogs of Buchberger's criterion and algorithm in Section 3.1, and we prove an OI-analog of Schreyer's theorem in Section 3.2. In Chapter 4 we apply our results to the problem of computing free resolutions. Specifically, Section 4.1 is devoted to computing free OI-resolutions, and in Section 4.2 we show how to adapt our techniques to compute free FI-resolutions. Along the way, we provide examples using our package "OIGroebnerBases.m2" whose source code is included in Appendix A.

Chapter 2 Preliminaries

In this chapter we review the basic theory of OI-modules over OI-algebras and fix notation. We follow the approaches taken in [**12, 13**]. Let K be a noetherian commutative ring with identity.

Definition 2.0.1. Denote by OI the category of totally ordered finite sets and order-preserving injective maps.

The idea behind the category OI is that it will serve as a combinatorial device for parameterizing collections of related algebraic objects. To this end, we will often be working with functors out of OI, and to define such a functor, it suffices to define it on the corresponding skeleton. Hence, we adopt the following convention.

Convention 2.0.2. Regard OI as the category whose objects are intervals of the form $[n] = \{1, \ldots, n\}$ (thus, $[0] = \emptyset$) and whose morphisms are order-preserving injective maps $[m] \to [n]$.

If $\mathbf{A}$ is any functor out of OI, we often write $\mathbf{A}_n$ in place of $\mathbf{A}([n])$ and refer to $\mathbf{A}_n$ as the *width n component* of $\mathbf{A}$. If ε is any OI-morphism, we sometimes write ε_* instead of $\mathbf{A}(\varepsilon)$. We abuse notation and write $\mathrm{Hom}(m, n)$ for the set of OI-morphisms $\mathrm{Hom}_{\mathrm{OI}}([m], [n])$ from $[m]$ to $[n]$.

2.1 OI-Algebras

We first introduce the algebras over which our OI-modules will be defined.

Definition 2.1.1. An OI-*algebra* over K is a covariant functor from OI to the category K-Alg of commutative, associative, unital K-algebras.

Informally, an OI-algebra over K may be thought of as a sequence of related K-algebras. The following example illustrates this.

Example 2.1.2. Fix an integer $c \geq 0$ and define a functor $\mathbf{P} = \mathbf{P}^{\mathrm{OI},c} : \mathrm{OI} \to K\text{-Alg}$ as follows. For each $n \geq 0$ define

$$\mathbf{P}_n = K \begin{bmatrix} x_{1,1} & \cdots & x_{1,n} \\ \vdots & \ddots & \vdots \\ x_{c,1} & \cdots & x_{c,n} \end{bmatrix}$$

and for each $\varepsilon \in \mathrm{Hom}(m, n)$ define $\varepsilon_* : \mathbf{P}_m \to \mathbf{P}_n$ via $x_{i,j} \mapsto x_{i,\varepsilon(j)}$. Then $\mathbf{P}$ forms an OI-algebra over K. Futhermore, one may think of $\mathbf{P}$ as a sequence of polynomial rings $(\mathbf{P}_n)_{n \geq 0}$ over K whose numbers of variables increase linearly in n. For instance, if $c = 1$ we obtain the sequence

$$\mathbf{P}^{\mathrm{OI},1} \ : \ K, \ K[x_1], \ K[x_1, x_2], \ \ldots$$

and if $c = 0$ we obtain the *constant* OI-algebra

$$\mathbf{P}^{\mathrm{OI},0} \ : \ K, \ K, \ K, \ \ldots$$

that sends every interval $[n]$ to K.

We say an OI-algebra $\mathbf{A}$ is *graded* if each $\mathbf{A}_n$ is a graded K-algebra and each $\varepsilon_* : \mathbf{A}_m \to \mathbf{A}_n$ is a graded homomorphism of degree 0. For all $c \geq 0$, the OI-algebra $\mathbf{P} = \mathbf{P}^{\mathrm{OI},c}$ is an example of a graded OI-algebra by assigning each variable degree 1. We will always use this grading of $\mathbf{P}$.

2.2 OI-Modules

We now turn to OI-modules, our primary objects of interest.

Definition 2.2.1. An OI-*module* $\mathbf{M}$ over an OI-algebra $\mathbf{A}$ is a covariant functor from OI to the category K-Mod of modules over K such that

(i) each $\mathbf{M}_n$ is an $\mathbf{A}_n$-module, and

(ii) for each $a \in \mathbf{A}_m$ and $\varepsilon \in \mathrm{Hom}(m, n)$ we have a commuting diagram

$$\begin{tikzcd} \mathbf{M}_m \arrow[r, "\mathbf{M}(\varepsilon)"] \arrow[d, "a\cdot"'] & \mathbf{M}_n \arrow[d, "\mathbf{A}(\varepsilon)(a)\cdot"] \\ \mathbf{M}_m \arrow[r, "\mathbf{M}(\varepsilon)"] & \mathbf{M}_n \end{tikzcd}$$

where the vertical maps are multiplication by the indicated elements.

We sometimes call $\mathbf{M}$ an $\mathbf{A}$-*module*.

Intuitively, one may think of an $\mathbf{A}$-module $\mathbf{M}$ as a sequence $(\mathbf{M}_n)_{n\geq 0}$ of $\mathbf{A}_n$-modules together with K-linear maps $\varepsilon_* : \mathbf{M}_m \to \mathbf{M}_n$ induced by any $\varepsilon \in \mathrm{Hom}(m, n)$.

2.2.1 Basic Properties

In this section we collect some basic properties of OI-modules that follow readily from the definitions.

A *homomorphism* of $\mathbf{A}$-modules is a natural transformation $\varphi : \mathbf{M} \to \mathbf{N}$ such that each $\varphi_n : \mathbf{M}_n \to \mathbf{N}_n$ is an $\mathbf{A}_n$-module homomorphism. We sometimes call φ an $\mathbf{A}$-*linear map*.

Lemma and Definition 2.2.2 ([**13**]). *$\mathbf{A}$-modules together with $\mathbf{A}$-linear maps form an abelian category denoted* OI-Mod($\mathbf{A}$).

It follows therefore that all concepts in OI-Mod($\mathbf{A}$) such as subobject, quotient object, kernel, cokernel, injection and surjection are defined "width-wise" from the corresponding concepts in the category of K-modules (see [**16**, A.3.3]).

For example, if $\mathbf{M}$ and $\mathbf{N}$ are $\mathbf{A}$-modules, their *direct sum* is the $\mathbf{A}$-module $\mathbf{M} \oplus \mathbf{N}$ defined width-wise by

$$(\mathbf{M} \oplus \mathbf{N})_n = \mathbf{M}_n \oplus \mathbf{N}_n$$

for all $n \geq 0$. Similarly, if $\varphi : \mathbf{M} \to \mathbf{N}$ is an $\mathbf{A}$-linear map, then the *kernel* of φ is a submodule of $\mathbf{M}$ defined by $(\ker(\varphi))_n = \ker(\varphi_n)$ for all $n \geq 0$. The *image* of φ is a submodule of $\mathbf{N}$ defined in an analogous fashion.

Any OI-algebra $\mathbf{A}$ may be considered as an OI-module over itself. An *ideal* of $\mathbf{A}$ is a submodule of $\mathbf{A}$.

An *element* f of an OI-module $\mathbf{M}$, denoted $f \in \mathbf{M}$, is an element $f \in \mathbf{M}_n$ for some $n \geq 0$. We say f *has width* n, denoted $w(f) = n$. A *subset* S of $\mathbf{M}$, denoted $S \subseteq \mathbf{M}$, is a subset of the disjoint union $\coprod_{n \geq 0} \mathbf{M}_n$.

Let $S \subseteq \mathbf{M}$ be a subset. The submodule of $\mathbf{M}$ *generated by* S is the smallest submodule of $\mathbf{M}$ containing S, and is denoted $\langle S \rangle_{\mathbf{M}}$. If S is finite, we say $\langle S \rangle_{\mathbf{M}}$ is *finitely generated by* S.

It is desirable to have a width-wise description of the submodule generated by a set. This motivates the following construction.

Definition 2.2.3 ([**12**])**.** For any $m \geq 0$, the *m-orbit* of S is the set

$$\operatorname{Orb}(S, m) = \{\mathbf{M}(\varepsilon)(s) \ : \ s \in S \cap \mathbf{M}_\ell, \ \varepsilon \in \operatorname{Hom}(\ell, m)\} \subseteq \mathbf{M}_m.$$

In other words, $\operatorname{Orb}(S, m)$ is the collection of all images of elements of S under maps $\mathbf{M}(\varepsilon)$ that land in width m. With this notation, we have

$$(\langle S \rangle_{\mathbf{M}})_m = \langle \operatorname{Orb}(S, m) \rangle$$

for all $m \geq 0$, where the right-hand side is generated as an $\mathbf{A}_m$-submodule of $\mathbf{M}_m$. Note that if S is finite, each $\operatorname{Orb}(S, m)$ is also finite.

Example 2.2.4. Let $\mathbf{P} = \mathbf{P}^{\mathrm{OI},2}$ and let $f = x_{1,1}x_{2,2} - x_{1,2}x_{2,1} \in \mathbf{P}_2$. Let $\mathbf{I}$ be the ideal of $\mathbf{P}$ generated by f. We have

$$\operatorname{Orb}(f, n) = \{x_{1,i}x_{2,j} - x_{1,j}x_{2,i} \ : \ 1 \leq i < j \leq n\}$$

and hence $\mathbf{I}_n$ is the ideal of $\mathbf{P}_n$ that is generated by the 2-minors of a generic $2 \times n$ matrix.

Let now $\mathbf{J}$ be the ideal of $\mathbf{P}$ that is generated by $x_{1,1}x_{2,3} - x_{1,3}x_{2,1} \in \mathbf{P}_3$. Then a similar calculation shows that $\mathbf{J}_n$ is the ideal of $\mathbf{P}_n$ that is generated by the 2-minors of a generic $2 \times n$ matrix using columns i, j with $j \geq i + 2$.

The following lemma shows that "high-width" orbits of a finite set $S \subset \mathbf{M}$ are determined by $\operatorname{Orb}(S, w)$ where w is the largest width of an element in S.

Lemma 2.2.5. *Let $S \subset \mathbf{M}$ be a finite set and let $w = \max\{w(s) \ : \ s \in S\}$. Then for all $n \geq w$, one has* $\operatorname{Orb}(S, n) = \operatorname{Orb}(\operatorname{Orb}(S, w), n)$.

Proof. The inclusion $\mathrm{Orb}(\mathrm{Orb}(S,w),n) \subseteq \mathrm{Orb}(S,n)$ follows from the functoriality of $\mathbf{M}$. Now pick $f \in \mathrm{Orb}(S,n)$ and write $f = \mathbf{M}(\varepsilon)(g)$ for some $g \in S$ and some $\varepsilon \in \mathrm{Hom}(w(g),n)$. By repeated application of [**13**, Lemma 2.3], we have $\varepsilon = \pi \circ \rho$ for maps $\rho \in \mathrm{Hom}(w(g),w)$ and $\pi \in \mathrm{Hom}(w,n)$. We conclude by observing that $f = \mathbf{M}(\pi)(\mathbf{M}(\rho)(g))$ and $\mathbf{M}(\rho)(g) \in \mathrm{Orb}(S,w)$. □

An OI-module $\mathbf{M}$ over a graded OI-algebra $\mathbf{A}$ is called *graded* if each $\mathbf{M}_m$ is a graded $\mathbf{A}_m$-module and each $\mathbf{M}(\varepsilon) : \mathbf{M}_m \to \mathbf{M}_n$ is a graded homomorphism of degree 0. It is useful for bookkeeping purposes to alter the grading of an OI-module as follows. Given a graded OI-module $\mathbf{M}$ and an integer $d \in \mathbb{Z}$, the d^{th} *twist* (or *shift*) of $\mathbf{M}$ is defined to be the OI-module $\mathbf{M}(d)$ that is isomorphic to $\mathbf{M}$ as an OI-module, and whose grading is determined by

$$[\mathbf{M}(d)_n]_j = [\mathbf{M}_n]_{d+j}$$

for all $n \geq 0$ and $j \in \mathbb{Z}$. If $\mathbf{M}$ and $\mathbf{N}$ are graded $\mathbf{A}$-modules, an $\mathbf{A}$-linear map $\varphi : \mathbf{M} \to \mathbf{N}$ is called *graded* if each φ_n is graded of degree 0.

2.2.2 Free OI-Modules

We now introduce the *free* OI-modules, which form the most important class of OI-modules for our purposes.

Definition 2.2.6. Let $d \geq 0$ be an integer and let $\mathbf{A}$ be an OI-algebra. Define a functor $\mathbf{F}^{\mathrm{OI},d} : \mathrm{OI} \to K\text{-Mod}$ as follows. For all $n \geq 0$ define

$$\mathbf{F}^{\mathrm{OI},d}_n = \bigoplus_{\pi \in \mathrm{Hom}(d,n)} \mathbf{A}_n e_\pi \cong (\mathbf{A}_n)^{\binom{n}{d}}.$$

For each $\varepsilon \in \mathrm{Hom}(m,n)$, define $\mathbf{F}^{\mathrm{OI},d}(\varepsilon) \colon \mathbf{F}^{\mathrm{OI},d}_m \to \mathbf{F}^{\mathrm{OI},d}_n$ via $ae_\pi \mapsto \varepsilon_*(a)e_{\varepsilon\circ\pi}$. One verifies that $\mathbf{F}^{\mathrm{OI},d}$ forms an OI-module over $\mathbf{A}$. A *free* OI-module over $\mathbf{A}$ is an OI-module $\mathbf{F}$ that is isomorphic to a direct sum $\bigoplus_{\lambda\in\Lambda} \mathbf{F}^{\mathrm{OI},d_\lambda}$ for integers $d_\lambda \geq 0$. If $|\Lambda| = n < \infty$, then $\mathbf{F}$ is said to have *rank* n.

It follows as a special case of [**5**, Lemma 3.6] that the rank of a free OI-module is well-defined. Note also that $\mathbf{F}^{\mathrm{OI},0}$ is isomorphic to the OI-algebra $\mathbf{A}$ considered as an OI-module over itself.

Remark 2.2.7. Let $\mathbf{F} = \bigoplus_{i=1}^s \mathbf{F}^{\mathrm{OI},d_i}$ be a free OI-module over $\mathbf{A}$.

(i) For all $n \geq 0$ we have

$$\mathbf{F}_n = \bigoplus_{\substack{\pi \in \mathrm{Hom}(d_i,n)\\ 1\leq i\leq s}} \mathbf{A}_n e_{\pi,i} \cong (\mathbf{A}_n)^{\sum_{i=1}^s \binom{n}{d_i}}$$

where the second index on $e_{\pi,i}$ is for keeping track of which direct summand it lives in. Thus, we can think of $\mathbf{F}$ as a sequence of free modules whose ranks grow according to sums of binomial coefficients.

(ii) $\mathbf{F}$ is finitely generated as an $\mathbf{A}$-module by all $e_{\mathrm{id}_{[d_i]},i}$. We call these the *basis elements* of $\mathbf{F}$.

(iii) To define an $\mathbf{A}$-linear map $\varphi : \mathbf{F} \to \mathbf{N}$ where $\mathbf{N}$ is any $\mathbf{A}$-module, it suffices to specify images $\varphi(e_{\mathrm{id}_{[d_i]},i}) \in \mathbf{N}_{d_i}$ for each basis element $e_{\mathrm{id}_{[d_i]},i}$.

Example 2.2.8 ([**12**])**.** Let $\mathbf{P} = \mathbf{P}^{\mathrm{OI},1}$ so that $\mathbf{P}_n = K[x_1, \ldots, x_n]$ for all $n \geq 0$. Consider $\mathbf{F} = \mathbf{F}^{\mathrm{OI},1} \oplus \mathbf{F}^{\mathrm{OI},2}$. Since $\mathrm{Hom}(1, n)$ and $\mathrm{Hom}(2, n)$ contain $\binom{n}{1}$ and $\binom{n}{2}$ elements respectively, we have

$$\begin{aligned}
&\mathbf{F}_0 = 0,\\
&\mathbf{F}_1 = K[x_1],\\
&\mathbf{F}_2 = K[x_1, x_2]^2 \oplus K[x_1, x_2] \cong K[x_1, x_2]^3,\\
&\mathbf{F}_3 = K[x_1, x_2, x_3]^3 \oplus K[x_1, x_2, x_3]^3 \cong K[x_1, x_2, x_3]^6,
\end{aligned}$$

and in general, $\mathbf{F}_n \cong K[x_1, \ldots, x_n]^{\binom{n}{1}+\binom{n}{2}}$.

Example 2.2.9. Let $\mathbf{P} = \mathbf{P}^{\mathrm{OI},2}$ and $f = x_{1,1}x_{2,2} - x_{1,2}x_{2,1} \in \mathbf{P}_2$ be as in Example 2.2.4. Let $\mathbf{F} = \mathbf{F}^{\mathrm{OI},2}$ have basis $\{e_{\mathrm{id}_{[2]}}\}$ and define a map $\varphi : \mathbf{F} \to \mathbf{P}$ via $e_{\mathrm{id}_{[2]}} \mapsto f$. Then φ is a map of $\mathbf{P}$-modules whose image is the ideal of $\mathbf{P}$ generated by f.

2.3 Noetherian Algebras and Modules

In this section we recall a fundamental finiteness condition for OI-algebras and OI-modules (see [**13**, Section 4]).

Definition 2.3.1. Let $\mathbf{A}$ be an OI-algebra over K, and let $\mathbf{M}$ be an OI-module over $\mathbf{A}$. We say $\mathbf{M}$ is *noetherian* if every submodule of $\mathbf{M}$ is finitely generated. We say $\mathbf{A}$ is *noetherian* if it is noetherian as an OI-module over itself.

The following result provides equivalent conditions for an OI-module to be noetherian.

Proposition 2.3.2 ([**13**])**.** *Let $\mathbf{A}$ be an* OI*-algebra over K and let $\mathbf{M}$ be an $\mathbf{A}$-module. The following are equivalent:*

(i) $\mathbf{M}$ is noetherian.

(ii) Every ascending chain of submodules of $\mathbf{M}$

$$\mathbf{M}_1 \subseteq \mathbf{M}_2 \subseteq \cdots$$

stabilizes, i.e. there exists $n \geq 1$ such that $\mathbf{M}_i = \mathbf{M}_j$ for all $i, j \geq n$.

(iii) Every non-empty set of submodules of $\mathbf{M}$ has a maximal element.

We will primarily focus on the following noetherian OI-algebras and OI-modules in this dissertation.

Theorem 2.3.3. *Let $c \geq 0$.*

(i) The OI*-algebra* $\mathbf{P}^{\mathrm{OI},c}$ *from Example 2.1.2 is noetherian.*

(ii) Any finitely generated OI*-module over* $\mathbf{P}^{\mathrm{OI},c}$ *is noetherian.*

Proof. (i) and (ii) follow from [**13**, Theorem 6.15 and Corollary 6.16] using the fact that, in the notation of [**13**], we have $\mathbf{P}^{\mathrm{OI},c} = (\mathbf{X}^{\mathrm{OI},1})^{\otimes c}$ for $c > 0$ and $\mathbf{P}^{\mathrm{OI},0} = \mathbf{X}^{\mathrm{OI},0}$. □

It is an open question whether a finitely generated OI-module over an arbitrary noetherian OI-algebra is noetherian.

Remark 2.3.4. The OI-algebras $\mathbf{P}^{\mathrm{OI},c}$ for $c \geq 0$ are precisely the noetherian *polynomial* OI-algebras as shown in [**13**, Proposition 4.8].

Chapter 3 Computational Methods

The purpose of this chapter is to develop computational tools for working with OI-modules. We follow the exposition given in [**12**, Sections 3 and 4]. For the duration of this chapter, we assume that K is a field and consider OI-modules over $\mathbf{P} = \mathbf{P}^{\mathrm{OI},c}$ where $c > 0$ (see Example 2.1.2). The *constant coefficient* case of $c = 0$ has been studied in, for example, [**15**].

In [**13**], Nagel and Römer showed that any submodule of a finitely generated free $\mathbf{P}$-module admits a finite Gröbner basis. This result has been used in, e.g., [**14**] where it is shown that the equivariant Hilbert series of any finitely generated graded OI-module over $\mathbf{P}$ is rational. In Section 3.1, we show how to compute such Gröbner bases in finite time for any prescribed monomial order. We achieve this by proving OI-analogs of Buchberger's criterion and algorithm that hinge on a combinatorial result about factoring OI-morphisms (see Lemma 3.1.16).

In Section 3.2 we prove an OI-analog of Schreyer's theorem that describes how to compute the module of relations (or *syzygies*) of any submodule of a finitely generated free $\mathbf{P}$-module. We refer the reader to [**4**, Chapter 15] for a classical treatment of Buchberger's criterion and Schreyer's theorem.

3.1 Gröbner Bases

Gröbner bases are a powerful computational device that allow for algorithmic solutions to many problems. We begin this section with the following motivational question.

Problem 3.1.1. *Let $\mathbf{F}$ be a finitely generated free $\mathbf{P}$-module and let $\mathbf{M} = \langle g_1, \dots, g_s \rangle_{\mathbf{F}}$ for some $g_i \in \mathbf{F}$. Given an arbitrary $f \in \mathbf{F}$, how does one decide whether $f \in \mathbf{M}$?*

If $w(f) = n$, then $f \in \mathbf{M}$ if and only if there is an expression

$$f = \sum_{j=1}^{t} a_j \mathbf{F}(\varepsilon_j)(g_{i_j})$$

for some $a_j \in \mathbf{P}_n$, some $i_j \in [s]$ and some $\varepsilon_j \in \mathrm{Hom}(w(g_{i_j}), n)$. But how does one actually go about finding such an expression, or proving that such an expression does not exist? We will see that Gröbner bases provide an answer to this question.

3.1.1 Monomial Orders

Let $\mathbf{F} = \bigoplus_{i=1}^{s} \mathbf{F}^{\mathrm{OI},d_i}$ be a free $\mathbf{P}$-module with basis $\{e_{\mathrm{id}_{[d_i]},i} \ : \ i \in [s]\}$. A *monomial* in $\mathbf{F}$ is an element of the form $ae_{\pi,i}$ where a is a monomial in $\mathbf{P}$. We say the monomial $ae_{\pi,i}$ *involves the basis element* $e_{\mathrm{id}_{[d_i]},i}$. The collection of all monomials in $\mathbf{F}$ is denoted $\mathrm{Mon}(\mathbf{F})$. A *monomial submodule* of $\mathbf{F}$ is a submodule generated by monomials. Any nonzero element of $\mathbf{F}$ can be uniquely expressed as a K-linear combination of distinct monomials with nonzero coefficients.

We need a suitable way to order the monomials in $\mathbf{F}$.

Definition 3.1.2 (cf. [**12, 13, 14**]). A *monomial order on* $\mathbf{F}$ is a total order $<$ on $\operatorname{Mon}(\mathbf{F})$ such that if $\mu < \nu$ for any monomials $\mu, \nu \in \mathbf{F}_m$, we have

(i) $\mu < a\mu < a\nu$ for any monomial $1 \neq a \in \mathbf{P}_m$, and

(ii) $\mu < \mathbf{F}(\varepsilon)(\mu) < \mathbf{F}(\varepsilon)(\nu)$ for all $\varepsilon \in \operatorname{Hom}(m, n)$ with $m < n$.

To gain some intuition for this defintion, consider the following example.

Example 3.1.3 ([**12**]). Let $\mathbf{P} = \mathbf{P}^{\mathrm{OI},1}$ and let $\mathbf{F} = \mathbf{F}^{\mathrm{OI},0} \cong \mathbf{P}$. Take $x_1 e_\pi \in \mathbf{F}_5$ and $x_1 e_\rho \in \mathbf{F}_3$. Here, π maps the empty set to $[5]$ and ρ maps the empty set to $[3]$. If we choose any map $\varepsilon \in \operatorname{Hom}(3, 5)$ with $\varepsilon(1) = 1$, then Condition (ii) of Definition 3.1.2 implies

$$x_1 e_\rho < \mathbf{F}(\varepsilon)(x_1 e_\rho) = x_{\varepsilon(1)} e_{\varepsilon \circ \rho} = x_1 e_\pi.$$

Monomial orders exist. We will primarily use the *lexicographic order* throughout this dissertation, which is defined as follows.

Example 3.1.4. Order the monomials in each $\mathbf{P}_m$ lexicographically with $x_{i',j'} < x_{i,j}$ if either $i' < i$ or $i' = i$ and $j' < j$. Define $e_{\rho,j} < e_{\pi,i}$ if $i < j$. For a fixed i, identify a monomial $e_{\pi,i} \in \mathbf{F}_m$ with a vector $(m, \pi(1), \ldots, \pi(d_i)) \in \mathbb{N}^{d_i+1}$ and order such monomials using the lexicographic order on $\mathbb{N}^{d_i+1}$. Finally, for monomials $ae_{\pi,i}$ and $be_{\rho,j}$ in $\mathbf{F}$, define $be_{\rho,j} < ae_{\pi,i}$ if $e_{\rho,j} < e_{\pi,i}$ or $e_{\pi,i} = e_{\rho,j}$ and $b < a$ in $\mathbf{P}$. One checks that this gives indeed a monomial order on $\mathbf{F}$.

Following [**13**, Definition 6.1], a monomial $be_{\sigma,j}$ is said to be OI-*divisible* by a monomial $ae_{\pi,i}$ if there is an OI-morphism ε such that $\mathbf{F}(\varepsilon)(ae_{\pi,i}) = \varepsilon_*(a)e_{\varepsilon\circ\pi,i}$ divides $be_{\sigma,j}$. It was shown in [**13**] (see also [**14**, Proposition 5.3]) that OI-divisibility gives a well-partial-order on $\operatorname{Mon}(\mathbf{F})$. We claim that any monomial order $<$ on $\mathbf{F}$ refines the OI-divisibility partial order on $\operatorname{Mon}(\mathbf{F})$. To see this, suppose $\varepsilon_*(a)e_{\varepsilon\circ\pi,i}$ divides $be_{\sigma,j}$ so that $i = j$, $\varepsilon \circ \pi = \sigma$, and there is some monomial $g \in \mathbf{P}$ such that $g\varepsilon_*(a) = b$. The claim then follows by observing that

$$ae_{\pi,i} < \varepsilon_*(a)e_{\varepsilon\circ\pi,i} = \varepsilon_*(a)e_{\sigma,j} < g\varepsilon_*(a)e_{\sigma,j} = be_{\sigma,j}$$

where the first inequality follows from Condition (ii) of Definition 3.1.2 and the second from Condition (i).

Remark 3.1.5. Our discussion above yields a generalization of Dickson's lemma for our context: any monomial order $<$ on $\mathbf{F}$ is in fact a *well-order* on $\operatorname{Mon}(\mathbf{F})$, i.e. any nonempty subset of $\operatorname{Mon}(\mathbf{F})$ has a unique minimal element with respect to $<$.

Remark 3.1.6. Let $B \subset \mathbf{F}$ be a set of monomials and let $\mathbf{M} = \langle B \rangle_{\mathbf{F}}$ be the monomial submodule of $\mathbf{F}$ generated by B. Then a monomial $\mu \in \operatorname{Mon}(\mathbf{F})$ belongs to $\mathbf{M}$ if and only if μ is OI-divisible by an element of B.

With the language of monomial orders at our fingertips, we can now make the following definitions.

Definition 3.1.7. Fix a monomial order $<$ on $\mathbf{F}$. Let $f \in \mathbf{F}$ be nonzero and write $f = \sum c_i \mu_i$ for some nonzero $c_i \in K$ and distinct monomials $\mu_i \in \mathrm{Mon}(\mathbf{F})$.

- The *leading monomial* of f is $\mathrm{lm}(f) = \max_<\{\mu_i\}$.
- If $\mathrm{lm}(f) = \mu_i$ then the *leading coefficient* of f is $\mathrm{lc}(f) = c_i$.
- The *leading term* of f is $\mathrm{lt}(f) = \mathrm{lc}(f)\mathrm{lm}(f)$.
- For any subset $E \subseteq \mathbf{F}$ we define $\mathrm{lm}(E) = \{\mathrm{lm}(f) \ : \ f \in E \text{ is nonzero}\}$.

Example 3.1.8. Let $a \in \mathbf{P}$, let $f \in \mathbf{F}$, and suppose $a = \sum_{i=1}^{t} \lambda_i a_i$ and $f = \sum_{j=1}^{s} c_j f_j$ for nonzero $\lambda_i, c_j \in K$ and distinct monomials $a_i \in \mathbf{P}$ and $f_j \in \mathrm{Mon}(\mathbf{F})$. Possibly after reindexing, assume $f_1 > f_2 > \cdots > f_s$ and $a_1 f_1 > a_2 f_1 > \cdots > a_t f_1$. It follows that $a_1 f_1 > a_i f_j$ if $(i, j) \neq (1, 1)$ so that $\mathrm{lm}(af) = a_1 f_1 = a_1 \mathrm{lm}(f)$.

We conclude this section with some useful facts that follow from the properties of a monomial order on $\mathbf{F}$.

Remark 3.1.9. For all nonzero $f, g \in \mathbf{F}_m$ and $\varepsilon \in \mathrm{Hom}(m, n)$ one has

(i) $\mathrm{lm}(f + g) \leq \max(\mathrm{lm}(f), \mathrm{lm}(g))$,

(ii) $\mathrm{lm}(af) = a\mathrm{lm}(f)$ if $a \in \mathbf{P}_m$ is a monomial, and

(iii) $\mathbf{F}(\varepsilon)(\mathrm{lm}(f)) = \mathrm{lm}(\mathbf{F}(\varepsilon)(f))$.

3.1.2 Gröbner Bases and Division with Remainder

Gröbner bases for OI-modules are defined as in the classical situation.

Definition 3.1.10 ([12, 13, 14]). Fix a monomial order $<$ on $\mathbf{F}$ and let $\mathbf{M} \subseteq \mathbf{F}$ be a submodule. A subset $G \subseteq \mathbf{M}$ is called a *Gröbner basis* of $\mathbf{M}$ (with respect to $<$) if

$$\langle \mathrm{lm}(\mathbf{M}) \rangle_{\mathbf{F}} = \langle \mathrm{lm}(G) \rangle_{\mathbf{F}}.$$

Every submodule $\mathbf{M}$ of $\mathbf{F}$ has a Gröbner basis with respect to any monomial order by taking $G = \mathbf{M}$. We are interested in computing finite Gröbner bases, which are guaranteed to exist by [**13**].

For the remainder of this section we fix a monomial order $<$ on $\mathbf{F}$. Restricting $<$ to $\mathbf{F}_n$ for each n, one obtains a monomial order on the free $\mathbf{P}_n$-module $\mathbf{F}_n$ in the classical sense (see, for example, [**4**]) which we denote by $<_n$.

The following result compares Gröbner bases of OI-modules with Gröbner bases of their width n components.

Lemma 3.1.11. *Let* $\mathbf{M}$ *be a submodule of* $\mathbf{F}$*. For any subset* $G \subseteq \mathbf{M}$*, put* $G_n = \mathrm{Orb}(G, n)$ *for all* $n \geq 0$*. Then* G *is a Gröbner basis of* $\mathbf{M}$ *with respect to* $<$ *if and only if every* G_n *is a Gröbner basis for* $\mathbf{M}_n$ *with respect to* $<_n$*.*

Proof. Suppose that G is a Gröbner basis of $\mathbf{M}$ and let $h \in \mathbf{M}_n$. Then $\operatorname{lm}_{<_n}(h) = \operatorname{lm}(h) \in \langle \operatorname{lm}(G) \rangle_{\mathbf{F}}$ so that $\operatorname{lm}_{<_n}(h)$ is divisible by some $\mathbf{F}(\varepsilon)(\operatorname{lm}(g)) = \operatorname{lm}_{<_n}(\mathbf{F}(\varepsilon)(g))$ with $g \in G$ and $\varepsilon \in \operatorname{Hom}(w(g), n)$. As $\mathbf{F}(\varepsilon)(g)$ is in G_n, this shows that $\operatorname{lm}_{<_n}(\mathbf{M}_n) \subseteq \langle \operatorname{lm}_{<_n}(G_n) \rangle$. Thus, G_n is a Gröbner basis for $\mathbf{M}_n$.

The argument for the reverse implication is similar and left to the reader. $\square$

Combined with Lemma 2.2.5, the above result says that computing a finite Gröbner basis G for $\mathbf{M}$ is the same as "simultaneously" computing a finite Gröbner basis for each $\mathbf{M}_n$ with the additional property that the Gröbner basis for $\mathbf{M}_n$ is obtained "combinatorially" by the OI-action from the Gröbner basis of $\mathbf{M}_w$ for each $n \geq w$, where w is the largest width of an element in G.

The basis of our computational results is a division algorithm.

Definition 3.1.12. Let $f \in \mathbf{F}_n$ and $B \subseteq \mathbf{F}$. An element $r \in \mathbf{F}_n$ is called a *remainder of f modulo B* if $f = \sum a_i q_i + r$ for some $q_i \in \operatorname{Orb}(B, n)$ and $a_i \in \mathbf{P}_n$ such that

(i) either $r = 0$ or $\operatorname{lm}(r)$ is not OI-divisible by any element of $\operatorname{lm}(B)$,

(ii) $\operatorname{lm}(r) < \operatorname{lm}(f)$ provided $r, f \neq 0$ and $r \neq f$, and

(iii) $\operatorname{lm}(a_i q_i) \leq \operatorname{lm}(f)$ whenever $a_i q_i, f \neq 0$.

Every element $f \in \mathbf{F}$ has a remainder modulo B. If f has width n such a remainder can be computed by applying the classical division algorithm in $\mathbf{F}_n$ to f and the set $\operatorname{Orb}(B, n)$ with respect to the monomial order $<_n$ on $\mathbf{F}_n$ (see, e.g., [**4**, Ch. 15]).

Remark 3.1.13. Together with Example 3.1.8, Condition (iii) of Definition 3.1.12 implies that $\operatorname{lm}(u q_i) \leq \operatorname{lm}(f)$ for any monomial u in a_i.

We are now in a position to solve Problem 3.1.1.

Proposition 3.1.14. *Let G be a Gröbner basis for a submodule $\mathbf{M}$ of a free* OI-*module $\mathbf{F}$. Then an element $f \in \mathbf{F}$ is in $\mathbf{M}$ if and only if f has a remainder of zero modulo G.*

Proof. If f has a remainder of zero modulo G, then $f \in \mathbf{M}$ since G generates $\mathbf{M}$. Assume now that $f \in \mathbf{M}$ and let $r \neq 0$ be a remainder of f modulo G. Then $r \in \mathbf{M}$ so that $\operatorname{lm}(r) \in \langle \operatorname{lm}(\mathbf{M}) \rangle_{\mathbf{F}} = \langle \operatorname{lm}(G) \rangle_{\mathbf{F}}$ which implies that $\operatorname{lm}(r)$ is OI-divisible by an element of $\operatorname{lm}(G)$. This contradicts Condition (i) of Definition 3.1.12 and the result follows. $\square$

We conclude this section with a technical result that will be useful in the proof of Theorem 3.1.23.

Proposition 3.1.15. *Suppose $f \in \mathbf{F}_m$ has a remainder of zero modulo $B \subseteq \mathbf{F}$. Then for all $\varepsilon \in \operatorname{Hom}(m, n)$, $\mathbf{F}(\varepsilon)(f)$ also has a remainder of zero modulo B.*

Proof. Let $f = \sum a_i q_i$ with $a_i \in \mathbf{P}_m$, $q_i \in \operatorname{Orb}(B, m)$ and $\operatorname{lm}(a_i q_i) \leq \operatorname{lm}(f)$ for all i. Then $\mathbf{F}(\varepsilon)(f) = \sum \varepsilon_*(a_i)\mathbf{F}(\varepsilon)(q_i)$ where $\varepsilon_*(a_i) \in \mathbf{P}_n$ and $\mathbf{F}(\varepsilon)(q_i) \in \operatorname{Orb}(B, n)$. We need to check that $\operatorname{lm}(\varepsilon_*(a_i)\mathbf{F}(\varepsilon)(q_i)) \leq \operatorname{lm}(\mathbf{F}(\varepsilon)(f))$ for all i. Using Remark 3.1.9 (iii) and Definition 3.1.2 (ii), we have

$$\begin{aligned}\operatorname{lm}(\varepsilon_*(a_i)\mathbf{F}(\varepsilon)(q_i)) &= \mathbf{F}(\varepsilon)(\operatorname{lm}(a_i q_i)) \\ &\leq \mathbf{F}(\varepsilon)(\operatorname{lm}(f)) \\ &= \operatorname{lm}(\mathbf{F}(\varepsilon)(f)),\end{aligned}$$

and we are done. □

3.1.3 The OI-Factorization Lemma

We now introduce an important combinatorial result about factoring OI-morphisms that will play an essential role in our finiteness arguments. A similar argument has been used in the proof of [**8**, Proposition 3.4].

Lemma 3.1.16 (OI-Factorization Lemma)**.** *Consider any maps $\sigma \in \operatorname{Hom}(k_1, m)$ and $\tau \in \operatorname{Hom}(k_2, m)$ with $k_1, k_2, m \in \mathbb{Z}_{\geq 0}$ and $k_1, k_2 \leq m$. Set $\ell = |\operatorname{im}(\sigma) \cup \operatorname{im}(\tau)|$. Then there are maps $\bar{\sigma} \in \operatorname{Hom}(k_1, \ell)$, $\bar{\tau} \in \operatorname{Hom}(k_2, \ell)$ and $\rho \in \operatorname{Hom}(\ell, m)$ such that*

$$\sigma = \rho \circ \bar{\sigma} \quad \textit{and} \quad \tau = \rho \circ \bar{\tau}$$

as well as $\ell = |\operatorname{im}(\bar{\sigma}) \cup \operatorname{im}(\bar{\tau})|$.

Proof. Let $L = \operatorname{im}(\sigma) \cup \operatorname{im}(\tau)$ so that $\ell = |L|$. Define a map $\gamma \colon L \to [\ell]$ via $i \mapsto |L \cap [i]|$. Clearly γ is strictly increasing. Since L and $[\ell]$ are finite sets of the same cardinality, it follows that γ is a bijection. Now define maps

$$\begin{aligned}\bar{\sigma} \colon [k_1] \to [\ell] \quad &\text{by} \quad i \mapsto \gamma(\sigma(i)), \\ \bar{\tau} \colon [k_2] \to [\ell] \quad &\text{by} \quad i \mapsto \gamma(\tau(i)), \\ \text{and } \rho \colon [\ell] \to [m] \quad &\text{by} \quad i \mapsto \gamma^{-1}(i).\end{aligned}$$

Since γ, σ and τ are strictly increasing, so are $\bar{\sigma}$, $\bar{\tau}$ and ρ. We have $\sigma = \rho \circ \bar{\sigma}$ and $\tau = \rho \circ \bar{\tau}$ by construction. The last claim follows from the fact that ρ is injective. □

We provide an example below to demonstrate how the OI-Factorization Lemma works in practice.

Example 3.1.17. Let $\sigma \in \operatorname{Hom}(2, 6)$ and $\tau \in \operatorname{Hom}(3, 6)$ be given by

$$\sigma : \begin{cases} 1 \mapsto 3 \\ 2 \mapsto 6 \end{cases} \qquad \text{and} \quad \tau : \begin{cases} 1 \mapsto 2 \\ 2 \mapsto 5 \\ 3 \mapsto 6. \end{cases}$$

Then $\operatorname{im}(\sigma) \cup \operatorname{im}(\tau) = \{2, 3, 5, 6\}$ so we define $\rho \in \operatorname{Hom}(4, 6)$ via

$$\rho : \begin{cases} 1 \mapsto 2 \\ 2 \mapsto 3 \\ 3 \mapsto 5 \\ 4 \mapsto 6. \end{cases}$$

Finally, we define $\bar{\sigma} \in \operatorname{Hom}(2, 4)$ and $\bar{\tau} \in \operatorname{Hom}(3, 4)$ via

$$\bar{\sigma} : \begin{cases} 1 \mapsto 2 \\ 2 \mapsto 4 \end{cases} \qquad \text{and} \quad \bar{\tau} : \begin{cases} 1 \mapsto 1 \\ 2 \mapsto 3 \\ 3 \mapsto 4. \end{cases}$$

One verifies that $\sigma = \rho \circ \bar{\sigma}$ and $\tau = \rho \circ \bar{\tau}$.

3.1.4 S-Polynomials and Critical Pairs

The classical Buchberger's criterion determines when a set forms a Gröbner basis by analyzing so-called *S-polynomials* (see [**4**, Chapter 15]). We give the corresponding construction in the OI-setting below.

Definition 3.1.18. Let $ae_{\pi,i}$ and $be_{\rho,j}$ be monomials in $\mathbf{F}$. Define

$$\operatorname{lcm}(ae_{\pi,i}, be_{\rho,j}) = \begin{cases} \operatorname{lcm}(a, b)e_{\pi,i} & \text{if } e_{\pi,i} = e_{\rho,j} \\ 0 & \text{otherwise.} \end{cases}$$

Here, $\operatorname{lcm}(a, b)$ is the least common multiple of the monomials a and b in $\mathbf{P}$.

Definition 3.1.19. Let $f, g \in \mathbf{F}_m$ be nonzero. The *S-polynomial* of f and g is the combination

$$S(f, g) = \frac{\operatorname{lcm}(\operatorname{lm}(f), \operatorname{lm}(g))}{\operatorname{lt}(f)} f - \frac{\operatorname{lcm}(\operatorname{lm}(f), \operatorname{lm}(g))}{\operatorname{lt}(g)} g \in \mathbf{F}_m,$$

where the coefficients of f and g are elements in $\mathbf{P}_m$.

Remark 3.1.20. By design, S-polynomials produce cancellation of leading terms. Specifically, one has $\operatorname{lm}(S(f, g)) < \operatorname{lcm}(\operatorname{lm}(f), \operatorname{lm}(g))$ for any nonzero $f, g \in \mathbf{F}_m$.

S-polynomials are compatible with the OI-action in the following sense.

Lemma 3.1.21. *For any nonzero $f, g \in \mathbf{F}_m$ and any $\rho \in \operatorname{Hom}(m, n)$, one has*

$$\mathbf{F}(\rho)(S(f, g)) = S(\mathbf{F}(\rho)(f), \mathbf{F}(\rho)(g)).$$

Proof. It suffices to show that if $\mu, \nu \in \mathbf{F}_m$ are monomials, then

$$\mathbf{P}(\rho)\left(\frac{\mathrm{lcm}(\mu,\nu)}{\mu}\right) = \frac{\mathrm{lcm}(\mathbf{F}(\rho)(\mu), \mathbf{F}(\rho)(\nu))}{\mathbf{F}(\rho)(\mu)}. \tag{3.1.1}$$

Without loss of generality we may assume that $\mu = ae_{\pi,i}$ and $\nu = be_{\pi,i}$ for monomials $a, b \in \mathbf{P}_m$. For an arbitrary monomial $\omega \in \mathbf{P}$, let $[\omega]_{s,t} \in \mathbb{Z}_{\geq 0}$ denote the exponent of the variable $x_{s,t}$ in ω. Therefore, if ω lives in width m, we have

$$[\mathbf{P}(\rho)(\omega)]_{s,t} = \begin{cases} [\omega]_{s,t'} & \text{if } \rho(t') = t \\ 0 & \text{otherwise} \end{cases} \tag{3.1.2}$$

since ρ is injective. To show (3.1.1), it is enough to argue that

$$\mathbf{P}(\rho)\left(\frac{\mathrm{lcm}(a,b)}{a}\right) = \frac{\mathrm{lcm}(\mathbf{P}(\rho)(a), \mathbf{P}(\rho)(b))}{\mathbf{P}(\rho)(a)}. \tag{3.1.3}$$

Using (3.1.2), we see that

$$\begin{aligned}
\left[\mathbf{P}(\rho)\left(\frac{\mathrm{lcm}(a,b)}{a}\right)\right]_{s,t} &= \begin{cases} \left[\frac{\mathrm{lcm}(a,b)}{a}\right]_{s,t'} & \text{if } \rho(t') = t \\ 0 & \text{otherwise} \end{cases} \\
&= \begin{cases} \max([a]_{s,t'}, [b]_{s,t'}) - [a]_{s,t'} & \text{if } \rho(t') = t \\ 0 & \text{otherwise} \end{cases} \\
&= \max([\mathbf{P}(\rho)(a)]_{s,t}, [\mathbf{P}(\rho)(b)]_{s,t}) - [\mathbf{P}(\rho)(a)]_{s,t} \\
&= \left[\frac{\mathrm{lcm}(\mathbf{P}(\rho)(a), \mathbf{P}(\rho)(b))}{\mathbf{P}(\rho)(a)}\right]_{s,t}
\end{aligned}$$

which establishes (3.1.3) and hence also (3.1.1). $\square$

Central to our theory is the notion of a *critical pair*, which we define as follows.

Definition 3.1.22. Let $B \subseteq \mathbf{F}$. A tuple

$$(\mathbf{F}(\sigma)(b_i), \mathbf{F}(\tau)(b_j)) \in \mathrm{Orb}(B, m) \times \mathrm{Orb}(B, m)$$

is called a *critical pair of* B if the following properties hold:

(i) $b_i, b_j \neq 0$,

(ii) $\mathrm{lm}(b_i)$ and $\mathrm{lm}(b_j)$ involve the same basis element, and

(iii) $m = |\mathrm{im}(\sigma) \cup \mathrm{im}(\tau)|$.

The set of all critical pairs of B is denoted $\mathcal{C}(B)$.

A key property of critical pairs is the fact that if B is finite, then $\mathcal{C}(B)$ is also finite. Indeed, using the above notation, finiteness of B implies that there are only finitely many choices for $b_i, b_j \in B$, and for each such pair there are only finitely many possible m because

$$m = |\mathrm{im}(\sigma) \cup \mathrm{im}(\tau)| \leq |\mathrm{im}(\sigma)| + |\mathrm{im}(\tau)| = w(b_i) + w(b_j).$$

Finally, the sets $\mathrm{Hom}(w(b_i), m)$ and $\mathrm{Hom}(w(b_j), m)$ are finite.

3.1.5 The OI-Buchberger's Criterion and Algorithm

We are now ready to state and prove an OI-analog of Buchberger's criterion which detects when a subset of a free OI-module is a Gröbner basis.

Theorem 3.1.23 (OI-Buchberger's Criterion)**.** *Let $\mathbf{M} = \langle B \rangle_{\mathbf{F}}$ be the* OI*-module generated by a subset $B \subseteq \mathbf{F}$. Then B is a Gröbner basis of $\mathbf{M}$ if and only if $S(f,g)$ has a remainder of zero modulo B for every critical pair $(f,g) \in \mathcal{C}(B)$.*

Proof. Suppose first that B is a Gröbner basis for $\mathbf{M}$ and let $(f,g) \in \mathcal{C}(B)$. Since $S(f,g) \in \mathbf{M}$, it follows by Proposition 3.1.14 that $S(f,g)$ has a remainder of zero modulo B.

Conversely, assume each $S(f,g)$ with $(f,g) \in \mathcal{C}(B)$ has a remainder of zero modulo B. By Lemma 3.1.11 it suffices to show for each $n \in \mathbb{Z}_{\geq 0}$ that $B_n := \mathrm{Orb}(B, n)$ is a Gröbner basis of $\mathbf{M}_n$ with respect to the restricted monomial order $<_n$. Pick nonzero $f, g \in B_n$ and write $f = \mathbf{F}(\sigma)(b_1)$ and $g = \mathbf{F}(\tau)(b_2)$ for nonzero elements $b_1, b_2 \in B$ and maps $\sigma \in \mathrm{Hom}(w(b_1), n)$ and $\tau \in \mathrm{Hom}(w(b_2), n)$. Using the classical Buchberger's criterion (see, e.g., [**4**, Theorem 15.8]), we need to show that $S(f,g)$ has a remainder of zero modulo B_n.

Without loss of generality we may assume that $\mathrm{lm}(b_1)$ and $\mathrm{lm}(b_2)$ involve the same basis element, for otherwise $S(f,g)$ is zero. By the OI-Factorization Lemma, there are maps $\bar{\sigma} \in \mathrm{Hom}(w(b_1), \ell)$, $\bar{\tau} \in \mathrm{Hom}(w(b_2), \ell)$ and $\rho \in \mathrm{Hom}(\ell, n)$ with $\ell = |\mathrm{im}(\sigma) \cup \mathrm{im}(\tau)| = |\mathrm{im}(\bar{\sigma}) \cup \mathrm{im}(\bar{\tau})|$ such that $\sigma = \rho \circ \bar{\sigma}$ and $\tau = \rho \circ \bar{\tau}$. Put

$$\bar{f} = \mathbf{F}(\bar{\sigma})(b_1) \quad \text{and} \quad \bar{g} = \mathbf{F}(\bar{\tau})(b_2)$$

and note that $(\bar{f}, \bar{g}) \in \mathcal{C}(B)$. Functoriality of $\mathbf{F}$ gives $\mathbf{F}(\rho)(\bar{f}) = f$ and $\mathbf{F}(\rho)(\bar{g}) = g$. By Lemma 3.1.21 we get

$$\mathbf{F}(\rho)(S(\bar{f}, \bar{g})) = S(\mathbf{F}(\rho)(\bar{f}), \mathbf{F}(\rho)(\bar{g})) = S(f,g).$$

Since $S(\bar{f}, \bar{g})$ has a remainder of zero modulo B by assumption, Proposition 3.1.15 says that $S(f,g)$ also has a remainder of zero modulo B. So we can write $S(f,g) = \sum a_i q_i$ for some $a_i \in \mathbf{P}_n$ and some $q_i \in \mathrm{Orb}(B, n)$ such that $\mathrm{lm}(a_i q_i) \leq \mathrm{lm}(S(f,g))$ whenever $a_i q_i \neq 0$. As $<$ and $<_n$ agree on $\mathbf{F}_n$, this says that $S(f,g) \in \mathbf{M}_n$ has a remainder of zero modulo B_n with respect to the order $<_n$, completing the argument. $\square$

The OI-Buchberger's criterion leads naturally to an algorithm for computing Gröbner bases in finite time. The idea is analogous to the classical case: check if any S-polynomial has a nonzero remainder, and if so, append the remainder and repeat.

Algorithm 3.1.24 (OI-Buchberger's Algorithm).

Input: A finite subset $B \subset \mathbf{F}$
Output: A finite Gröbner basis G of $\langle B \rangle_{\mathbf{F}}$

set $G = B$
set $P = \mathcal{C}(G)$
while $P \neq \emptyset$ **do**
  choose $(f, g) \in P$
  set $P = P \setminus \{(f, g)\}$
  let r be a remainder of $S(f, g)$ modulo G
  if $r \neq 0$ **then**
    set $G = G \cup \{r\}$
    set $P = \mathcal{C}(G)$
  end if
end while
return G

Correctness and Termination of 3.1.24. Let $G_1, G_2, \ldots$ denote the values of G each time it gets updated (so in particular, $G_1 = B$). Suppose the algorithm terminates and returns G_t. Then $S(f, g)$ has a remainder of zero modulo G_t for every $(f, g) \in \mathcal{C}(G_t)$, so G_t forms a finite Gröbner basis for $\langle G_t \rangle_{\mathbf{F}}$ by the OI-Buchberger's criterion. For any t, if $(f, g) \in \mathcal{C}(G_t)$ and $S(f, g)$ has a remainder of $r \neq 0$ modulo G_t, then $r \in \langle G_t \rangle_{\mathbf{F}}$, and so $\langle G_t \rangle_{\mathbf{F}} = \langle G_{t+1} \rangle_{\mathbf{F}}$. It follows that $\langle G_t \rangle_{\mathbf{F}} = \langle B \rangle_{\mathbf{F}}$ and the algorithm is correct.

Suppose now that the algorithm does not terminate. Since each $\mathcal{C}(G_n)$ is finite we obtain an infinite chain $G_1 \subseteq G_2 \subseteq \cdots$. Let $n \in \mathbb{N}$ and let $r \neq 0$ be the remainder modulo G_n used to form G_{n+1}. By the definition of remainder, we have $\mathrm{lm}(r) \notin \langle \mathrm{lm}(G_n) \rangle_{\mathbf{F}}$, and hence $\langle \mathrm{lm}(G_n) \rangle_{\mathbf{F}} \subsetneq \langle \mathrm{lm}(G_{n+1}) \rangle_{\mathbf{F}}$. This yields a strictly increasing chain of submodules of $\mathbf{F}$, contradicting the fact that $\mathbf{F}$ is noetherian (see Proposition 2.3.2). Hence the algorithm terminates in finite time. ☐

The main result of this section follows easily now.

Theorem 3.1.25. *Every submodule of a finitely generated free* OI*-module* $\mathbf{F}$ *over* $\mathbf{P}$ *has a finite Gröbner basis with respect to any monomial order on* $\mathbf{F}$ *that can be computed in finitely many steps.*

Proof. By Theorem 2.3.3, any submodule of $\mathbf{F}$ is finitely generated. Thus, we conclude by applying Algorithm 3.1.24. ☐

We have implemented the OI-Buchberger's criterion in our *Macaulay2* [**7**] package "OIGroebnerBases.m2" [**11**]. We give an example of how this package works below. For more information about this package, see [**10**].

Example 3.1.26 ([**10, 12**]). Let $\mathbf{F} = \mathbf{F}^{\mathrm{OI},1} \oplus \mathbf{F}^{\mathrm{OI},1} \oplus \mathbf{F}^{\mathrm{OI},2}$ have basis $\{e_{\mathrm{id}_{[1]},1}, e_{\mathrm{id}_{[1]},2}, e_{\mathrm{id}_{[2]},3}\}$. Let $\mathbf{P} = \mathbf{P}^{\mathrm{OI},2}$ so that $\mathbf{P}$ has two rows of variables, and let

$$B = \{x_{1,1} e_{\mathrm{id}_{[1]},1} + x_{2,1} e_{\mathrm{id}_{[1]},2},\ x_{1,2} x_{1,1} e_{\pi,2} + x_{2,2} x_{2,1} e_{\mathrm{id}_{[2]},3}\}$$

where $\pi : [1] \to [2]$ is given by $1 \mapsto 2$. Thus, the first element of B has width 1 and the second element has width 2. Fix the *lex order* on $\mathbf{F}$ as described in Example 3.1.4. We compute a finite Gröbner basis for $\langle B \rangle_{\mathbf{F}}$ with our package `OIGroebnerBases` as follows.

First, we define our OI-algebra $\mathbf{P}$ with `makePolynomialOIAlgebra`. The user must specify the number of variable rows, the variable symbol, and the ground field K:

```
i1 : needsPackage "OIGroebnerBases";
i2 : P = makePolynomialOIAlgebra(2, x, QQ);
```

Now we define our free OI-module $\mathbf{F}$ with `makeFreeOIModule`. The user specifies the basis symbol, a list of basis element widths, and the underlying OI-algebra:

```
i3 : F = makeFreeOIModule(e, {1,1,2}, P);
```

To define our elements of B, we first need to call the `installGeneratorsInWidth` method in order to work with our basis symbol `e`. This method takes a free OI-module and a width as input:

```
i4 : installGeneratorsInWidth(F, 1);
i5 : installGeneratorsInWidth(F, 2);
```

We're ready to define the elements of B:

```
i6 : use F_1; b1 = x_(1,1)*e_(1,{1},1) + x_(2,1)*e_(1,{1},2);
i8 : use F_2; b2 = x_(1,2)*x_(1,1)*e_(2,{2},2) +
     x_(2,2)*x_(2,1)*e_(2,{1,2},3);
```

Here, for example, `e_(2,{2},2)` is the element $e_{\pi,2}$ as defined above. In general, an element $e_{\sigma,i} \in \mathbf{F}$ translates to an object in our package as follows. Suppose $\sigma \in \mathrm{Hom}(m, n)$, and write $\mathrm{im}(\sigma) = \{a_1, \ldots, a_m\}$ where each $a_j \in [n]$. Then $e_{\sigma,i}$ becomes `e_(n,{a₁,...,aₘ},i)` in our package.

Now let's compute a Gröbner basis with the method `oiGB`, which takes a list of elements as input:

```
i10 : oiGB {b1, b2}
o10 = {x    e        + x    e        , x    x    e        + x    x    e          ,
        1,1 1,{1},1    2,1 1,{1},2    1,2 1,1 2,{2},2    2,2 2,1 2,{1, 2},3
      ------------------------------------------------------------------------
      x    x    x    e           - x    x    x    e          }
       2,3 2,2 1,1 3,{2, 3},3     2,3 2,1 1,2 3,{1, 3},3
```

This tells us that a Gröbner basis for $\mathbf{M} = \langle B \rangle_{\mathbf{F}}$ with respect to the lex order is given by the following elements:

$$b_1 = x_{1,1}e_{\mathrm{id}_{[1]},1} + x_{2,1}e_{\mathrm{id}_{[1]},2} \in \mathbf{F}_1$$
$$b_2 = x_{1,2}x_{1,1}e_{\pi,2} + x_{2,2}x_{2,1}e_{\mathrm{id}_{[2]},3} \in \mathbf{F}_2$$
$$b_3 = x_{2,3}x_{2,2}x_{1,1}e_{\sigma_1,3} - x_{2,3}x_{2,1}x_{1,2}e_{\sigma_2,3} \in \mathbf{F}_3$$

where $\sigma_1 : [2] \to [3]$ is given by $1 \mapsto 2$ and $2 \mapsto 3$ and $\sigma_2 : [2] \to [3]$ is given by $1 \mapsto 1$ and $2 \mapsto 3$. It follows that, given any $n \geq 3$, a finite Gröbner basis for $\mathbf{M}_n$ is given by the images of b_1, b_2 and b_3 under any morphism $[1] \to [n]$, $[2] \to [n]$ and $[3] \to [n]$ respectively (see Lemma 3.1.11).

Given a submodule $\mathbf{M}$ of $\mathbf{F}$, the fact that $\mathbf{M}$ has a finite Gröbner basis G combined with Lemma 3.1.11 shows in particular that a Gröbner basis of $\mathbf{M}$ can be obtained as the union of Gröbner bases of $\mathbf{M}_n$ with respect to the order $<_n$ with $n \leq w$, where w is the maximum width of an element in G. This suggests an alternative way for computing a Gröbner basis of $\mathbf{M}$ by determining and comparing Gröbner bases of $\mathbf{M}_n$ with respect to the order $<_n$ for sufficiently large n. The problem is that a priori one does not know when to stop, i.e., what n is large enough. The following result gives a sufficient condition for the desired stabilization.

Proposition 3.1.27. *Let $\mathbf{M}$ be a submodule of a free $\mathbf{P}$-module $\mathbf{F}$ generated by a finite set $B \subset \mathbf{F}$. Let $<$ be any monomial order on $\mathbf{F}$ and let $W \geq 1$ denote the maximum width of an element in B. Then B is a Gröbner basis of $\mathbf{M}$ with respect to $<$ if and only if $\mathrm{Orb}(B, m)$ is a Gröbner basis for $\mathbf{M}_m$ with respect to $<_m$ for each integer $m \leq 2W$.*

Proof. The forward direction is immediate from Lemma 3.1.11. For the reverse, by the OI-Buchberger's criterion, it suffices to show for any critical pair $(f, g) \in \mathcal{C}(B)$ that $S(f, g)$ has a remainder of zero modulo B. Considering such a critical pair, we can write $(f, g) = (\mathbf{F}(\sigma)(b_1), \mathbf{F}(\tau)(b_2))$ for maps σ and τ and elements $b_1, b_2 \in B$ such that σ, τ land in width $m = |\mathrm{im}(\sigma) \cup \mathrm{im}(\tau)| \leq w(b_1) + w(b_2) \leq 2W$. Hence our hypothesis gives that $\mathrm{Orb}(B, m)$ is a Gröbner basis for $\mathbf{M}_m$, and so $S(f, g)$ has a remainder of zero modulo $\mathrm{Orb}(B, m)$ with respect to $<_m$. But since $<$ and $<_m$ agree on $\mathbf{M}_m$, this says that $S(f, g)$ has a remainder of zero modulo B with respect to $<$. □

3.2 Syzygies

In this section we concern ourselves with the computation of *syzygies*, i.e. relations among generators of an OI-module. Recall that in the classical setting, if M is a finitely generated submodule of a free module of finite rank over a noetherian polynomial ring, Schreyer's theorem [**4**, Theorem 15.10] computes a finite Gröbner basis for the module of syzygies of M. Using the theory developed in the previous section, we extend Schreyer's result to the setting of OI-modules.

Throughout this section, $\mathbf{F} = \bigoplus_{i=1}^{s} \mathbf{F}^{\mathrm{OI},d_i}$ always denotes a finitely generated free OI-module over $\mathbf{P} = \mathbf{P}^{\mathrm{OI},c}$ with some $c > 0$. We fix a monomial order $<$ on $\mathbf{F}$.

Definition 3.2.1. Let $B = \{b_1, \ldots, b_t\} \subset \mathbf{F}$ and let $\mathbf{G} = \bigoplus_{i=1}^{t} \mathbf{F}^{\mathrm{OI},w(b_i)}$ be a free OI-module with basis $\{\epsilon_{\mathrm{id}_{[w(b_i)]},i} \ : \ i \in [t]\}$. We define the *module of syzygies of B* as $\mathrm{Syz}(B) = \ker(\varphi)$, where φ denotes the surjective morphism

$$\varphi\colon \mathbf{G} = \bigoplus_{i=1}^{t} \langle \epsilon_{\mathrm{id}_{[w(b_i)]},i} \rangle_{\mathbf{G}} \to \langle B \rangle_{\mathbf{F}}, \text{ defined by } \epsilon_{\mathrm{id}_{[w(b_i)]},i} \mapsto b_i.$$

Our goal will be to compute a finite Gröbner basis for $\mathrm{Syz}(B)$.

3.2.1 Induced Monomial Orders

We first need a suitable monomial order on $\mathbf{G}$.

Definition 3.2.2. Let $B = \{b_1, \ldots, b_t\} \subset \mathbf{F}$.

(i) For any $i \in [t]$, order the set $\bigcup_{n \geq w(b_i)} \mathrm{Hom}(w(b_i), n)$ as follows: given $\pi \in \mathrm{Hom}(w(b_i), m)$ and $\rho \in \mathrm{Hom}(w(b_i), n)$, define $\pi < \rho$ if

$$(m, \pi(1), \ldots, \pi(w(b_i))) < (n, \rho(1), \ldots, \rho(w(b_i)))$$

in the lexicographic order on $\mathbb{N}^{w(b_i)+1}$.

(ii) Let $\prec_B$ be the total order on the set

$$\{(\pi, i) \ : \ i \in [t], \ \pi \in \mathrm{Hom}(w(b_i), n), \ n \geq w(b_i)\}$$

defined by $(\pi, i) \prec_B (\rho, j)$ if either $i < j$ or $i = j$ and $\pi < \rho$.

(iii) Define a total order $<_B$ on $\mathrm{Mon}(\mathbf{G})$ via $a\epsilon_{\pi,i} <_B b\epsilon_{\rho,j}$ if either $\mathrm{lm}(\varphi(a\epsilon_{\pi,i})) < \mathrm{lm}(\varphi(b\epsilon_{\rho,j}))$ or $\mathrm{lm}(\varphi(a\epsilon_{\pi,i})) = \mathrm{lm}(\varphi(b\epsilon_{\rho,j}))$ and $(\rho, j) \prec_B (\pi, i)$.

Remark 3.2.3. Note that for any monomial $a\epsilon_{\pi,i} \in \mathrm{Mon}(\mathbf{G})$, one has

$$\mathrm{lm}(\varphi(a\epsilon_{\pi,i})) = \mathrm{lm}(a\mathbf{F}(\pi)(b_i)) = a\mathbf{F}(\pi)(\mathrm{lm}(b_i)) = a\pi_*(\mathrm{lm}(b_i)).$$

Thus, Part (iii) can be more explicitly re-stated as $a\epsilon_{\pi,i} <_B b\epsilon_{\rho,j}$ if either $a\pi_*(\mathrm{lm}(b_i)) < b\rho_*(\mathrm{lm}(b_j))$ or $a\pi_*(\mathrm{lm}(b_i)) = b\rho_*(\mathrm{lm}(b_j))$ and $(\rho, j) \prec_B (\pi, i)$.

Together with the fact that $<$ is a monomial order on $\mathbf{F}$, the following observation implies that $<_B$ is a monomial order on $\mathbf{G}$.

Lemma 3.2.4. *Consider any maps $\pi \in \mathrm{Hom}(w(b_i), m)$ and $\rho \in \mathrm{Hom}(w(b_j), m)$ with $i, j \in [t]$ and $\sigma \in \mathrm{Hom}(m, n)$. Then $(\sigma \circ \pi, i) \prec_B (\sigma \circ \rho, j)$ if and only if $(\pi, i) \prec_B (\rho, j)$.*

Proof. This follows from the fact that σ is a strictly increasing map. $\square$

We call $<_B$ the *Schreyer order on $\mathbf{G}$ induced by $<$ and B.*

3.2.2 The OI-Schreyer's Theorem

We assume from now on that $B = \{b_1, \ldots, b_t\}$ is a Gröbner basis of $\langle B \rangle_{\mathbf{F}}$ and that each b_i is monic, possibly after multiplying by a suitable element of K. We need some further notation.

For any $i, j \in [t]$, $\sigma \in \operatorname{Hom}(w(b_i), m)$ and $\tau \in \operatorname{Hom}(w(b_j), m)$ such that $m \geq \max(w(b_i), w(b_j))$, the remainder of $S(\mathbf{F}(\sigma)(b_i), \mathbf{F}(\tau)(b_j))$ modulo B is zero since B is a Gröbner basis. Hence the division algorithm gives an expression

$$S(\mathbf{F}(\sigma)(b_i), \mathbf{F}(\tau)(b_j)) = \sum_{\ell} a_{i,j,\ell}^{\sigma,\tau} \mathbf{F}(\pi_{i,j,\ell}^{\sigma,\tau})(b_{k_{i,j,\ell}^{\sigma,\tau}})$$

with $a_{i,j,\ell}^{\sigma,\tau} \in \mathbf{P}_m$ and $\mathbf{F}(\pi_{i,j,\ell}^{\sigma,\tau})(b_{k_{i,j,\ell}^{\sigma,\tau}}) \in \operatorname{Orb}(B, m)$, where

$$\operatorname{lm}(u\mathbf{F}(\pi_{i,j,\ell}^{\sigma,\tau})(b_{k_{i,j,\ell}^{\sigma,\tau}})) \leq \operatorname{lm}(S(\mathbf{F}(\sigma)(b_i), \mathbf{F}(\tau)(b_j))) \tag{3.2.1}$$

for any monomial u in $a_{i,j,\ell}^{\sigma,\tau}$ (see Remark 3.1.13). Define

$$s_{i,j}^{\sigma,\tau} = m_{i,j}^{\sigma,\tau} \epsilon_{\sigma,i} - m_{j,i}^{\tau,\sigma} \epsilon_{\tau,j} - \sum_{\ell} a_{i,j,\ell}^{\sigma,\tau} \epsilon_{\pi_{i,j,\ell}^{\sigma,\tau}, k_{i,j,\ell}^{\sigma,\tau}} \in \mathbf{G}_m \tag{3.2.2}$$

where

$$m_{i,j}^{\sigma,\tau} = \frac{\operatorname{lcm}(\mathbf{F}(\sigma)(\operatorname{lm}(b_i)), \mathbf{F}(\tau)(\operatorname{lm}(b_j)))}{\mathbf{F}(\sigma)(\operatorname{lm}(b_i))} \in \mathbf{P}_m.$$

We need one more preparatory observation.

Lemma 3.2.5. *If $(\sigma, i) \prec_B (\tau, j)$ then $\operatorname{lm}_{<_B}(s_{i,j}^{\sigma,\tau}) = m_{i,j}^{\sigma,\tau} \epsilon_{\sigma,i}$.*

Proof. We have

$$\begin{aligned}
\operatorname{lm}(\varphi(m_{i,j}^{\sigma,\tau} \epsilon_{\sigma,i})) &= \frac{\operatorname{lcm}(\sigma_*(\operatorname{lm}(b_i)), \tau_*(\operatorname{lm}(b_j)))}{\sigma_*(\operatorname{lm}(b_i))} \sigma_*(\operatorname{lm}(b_i)) \\
&= \operatorname{lcm}(\sigma_*(\operatorname{lm}(b_i)), \tau_*(\operatorname{lm}(b_j))) \\
&= \frac{\operatorname{lcm}(\tau_*(\operatorname{lm}(b_j)), \sigma_*(\operatorname{lm}(b_i)))}{\tau_*(\operatorname{lm}(b_j))} \tau_*(\operatorname{lm}(b_j)) \\
&= \operatorname{lm}(\varphi(m_{j,i}^{\tau,\sigma} \epsilon_{\tau,j}))
\end{aligned}$$

and hence $(\sigma, i) \prec_B (\tau, j)$ implies $m_{j,i}^{\tau,\sigma} \epsilon_{\tau,j} <_B m_{i,j}^{\sigma,\tau} \epsilon_{\sigma,i}$ by the definition of $<_B$.

Now fix some ℓ and let u be the monomial in $a_{i,j,\ell}^{\sigma,\tau}$ such that

$$\operatorname{lm}_{<_B}(a_{i,j,\ell}^{\sigma,\tau} \epsilon_{\pi_{i,j,\ell}^{\sigma,\tau}, k_{i,j,\ell}^{\sigma,\tau}}) = u \epsilon_{\pi_{i,j,\ell}^{\sigma,\tau}, k_{i,j,\ell}^{\sigma,\tau}}.$$

Then we obtain

$$\begin{aligned}
\operatorname{lm}(u\mathbf{F}(\pi_{i,j,\ell}^{\sigma,\tau})(b_{k_{i,j,\ell}^{\sigma,\tau}})) &\leq \operatorname{lm}(S(\mathbf{F}(\sigma)(b_i), \mathbf{F}(\tau)(b_j))) && \text{(by (3.2.1))} \\
&< \operatorname{lcm}(\mathbf{F}(\sigma)(\operatorname{lm}(b_i)), \mathbf{F}(\tau)(\operatorname{lm}(b_j))) && \text{(by Remark 3.1.20)} \\
&= m_{i,j}^{\sigma,\tau} \mathbf{F}(\sigma)(\operatorname{lm}(b_i)).
\end{aligned}$$

By definition of $<_B$, this gives $u \epsilon_{\pi_{i,j,\ell}^{\sigma,\tau}, k_{i,j,\ell}^{\sigma,\tau}} <_B m_{i,j}^{\sigma,\tau} \epsilon_{\sigma,i}$ and the claim follows. ∎

We now state and prove the main result of this section.

Theorem 3.2.6 (OI-Schreyer's Theorem)**.** *The* $s_{i,j}^{\sigma,\tau}$ *with* $(\mathbf{F}(\sigma)(b_i), \mathbf{F}(\tau)(b_j)) \in \mathcal{C}(B)$ *and* $(\sigma, i) \prec_B (\tau, j)$ *form a finite Gröbner basis for* $\operatorname{Syz}(B)$ *with respect to* $<_B$*.*

Proof. First note that each $s_{i,j}^{\sigma,\tau}$ is indeed a syzygy, for

$$\varphi(s_{i,j}^{\sigma,\tau}) = S(\mathbf{F}(\sigma)(b_i), \mathbf{F}(\tau)(b_j)) - S(\mathbf{F}(\sigma)(b_i), \mathbf{F}(\tau)(b_j)) = 0.$$

Now let $h \in \operatorname{Syz}(B)$ be any nonzero syzygy. We need to show that $\operatorname{lm}(h)$ is OI-divisible by the leading monomial of one of the $s_{i,j}^{\sigma,\tau}$ in the claimed Gröbner basis of $\operatorname{Syz}(B)$.

To see this write $h = \sum_{\ell=1}^{n} c_\ell x^{p_\ell} \epsilon_{\pi_\ell, i_\ell}$ for some nonzero $c_\ell \in K$ and distinct $x^{p_\ell} \epsilon_{\pi_\ell, i_\ell} \in \operatorname{Mon}(\mathbf{G})$. Note that $n \geq 2$ since h is a nonzero syzygy. Possibly after reindexing, assume the monomials of h are ordered so that

$$x^{p_\ell} \epsilon_{\pi_\ell, i_\ell} >_B x^{p_{\ell'}} \epsilon_{\pi_{\ell'}, i_{\ell'}}$$

if $\ell < \ell'$. Then $\operatorname{lt}_{<_B}(h) = c_1 x^{p_1} \epsilon_{\pi_1, i_1}$ and, by definition of $<_B$, we get

$$x^{p_1} \mathbf{F}(\pi_1)(\operatorname{lm}(b_{i_1})) \geq \cdots \geq x^{p_n} \mathbf{F}(\pi_n)(\operatorname{lm}(b_{i_n})). \tag{3.2.3}$$

We claim that

$$x^{p_1} \mathbf{F}(\pi_1)(\operatorname{lm}(b_{i_1})) = x^{p_2} \mathbf{F}(\pi_2)(\operatorname{lm}(b_{i_2})). \tag{3.2.4}$$

Indeed, otherwise (3.2.3) becomes

$$x^{p_1} \mathbf{F}(\pi_1)(\operatorname{lm}(b_{i_1})) > x^{p_2} \mathbf{F}(\pi_2)(\operatorname{lm}(b_{i_2})) \geq \cdots \geq x^{p_n} \mathbf{F}(\pi_n)(\operatorname{lm}(b_{i_n})). \tag{3.2.5}$$

Since h is a syzygy we have $\sum_{\ell=1}^{n} c_\ell x^{p_\ell} \mathbf{F}(\pi_\ell)(b_{i_\ell}) = 0$ which, using (3.2.5), implies that $x^{p_1} \mathbf{F}(\pi_1)(\operatorname{lm}(b_{i_1}))$ can be written as a K-linear combination of strictly smaller monomials. This is impossible, and hence we have established (3.2.4).

Since $x^{p_1} \epsilon_{\pi_1, i_1} >_B x^{p_2} \epsilon_{\pi_2, i_2}$ we have

$$(\pi_1, i_1) \prec_B (\pi_2, i_2) \tag{3.2.6}$$

by (3.2.4) together with the definition of $<_B$.

Now set $\alpha = x^{p_1} \mathbf{F}(\pi_1)(\operatorname{lm}(b_{i_1}))$. By (3.2.4), α is divisible by $\mathbf{F}(\pi_2)(\operatorname{lm}(b_{i_2}))$, and so α is divisible by

$$\beta := \operatorname{lcm}(\mathbf{F}(\pi_1)(\operatorname{lm}(b_{i_1})), \mathbf{F}(\pi_2)(\operatorname{lm}(b_{i_2}))).$$

Let m be the width of h. By the OI-Factorization Lemma, one can find maps $\bar{\pi}_1 \in \operatorname{Hom}(w(b_{i_1}), \ell)$, $\bar{\pi}_2 \in \operatorname{Hom}(w(b_{i_2}), \ell)$ and $\rho \in \operatorname{Hom}(\ell, m)$ such that

$$\rho \circ \bar{\pi}_1 = \pi_1 \quad \text{and} \quad \rho \circ \bar{\pi}_2 = \pi_2$$

where $\ell = |\operatorname{im}(\bar{\pi}_1) \cup \operatorname{im}(\bar{\pi}_2)|$. By construction, $(\mathbf{F}(\bar{\pi}_1)(b_{i_1}), \mathbf{F}(\bar{\pi}_2)(b_{i_2}))$ is in $\mathcal{C}(B)$. Moreover, Relation (3.2.6) and Lemma 3.2.4 imply $(\bar{\pi}_1, i_1) \prec_B (\bar{\pi}_2, i_2)$. Putting everything together, we obtain

$$\begin{aligned}
\frac{\alpha}{\beta}\mathbf{G}(\rho)(\mathrm{lm}_{<_B}(s_{i_1,i_2}^{\bar\pi_1,\bar\pi_2})) &= \frac{\alpha}{\beta}\mathbf{G}(\rho)(m_{i_1,i_2}^{\bar\pi_1,\bar\pi_2}\epsilon_{\bar\pi_1,i_1}) && \text{(by Lemma 3.2.5)}\\
&= \frac{\alpha}{\beta}\frac{\mathrm{lcm}(\mathbf{F}(\pi_1)(\mathrm{lm}(b_{i_1})),\mathbf{F}(\pi_2)(\mathrm{lm}(b_{i_2})))}{\mathbf{F}(\pi_1)(\mathrm{lm}(b_{i_1}))}\epsilon_{\pi_1,i_1} && \text{(by (3.1.1))}\\
&= \frac{x^{p_1}\mathbf{F}(\pi_1)(\mathrm{lm}(b_{i_1}))}{\mathbf{F}(\pi_1)(\mathrm{lm}(b_{i_1}))}\epsilon_{\pi_1,i_1}\\
&= x^{p_1}\epsilon_{\pi_1,i_1}\\
&= \mathrm{lm}_{<_B}(h).
\end{aligned}$$

This shows that $\mathrm{lm}_{<_B}(h)$ is OI-divisible by $\mathrm{lm}_{<_B}(s_{i_1,i_2}^{\bar\pi_1,\bar\pi_2})$, which completes the proof. $\square$

In the graded case, the method can be modified to give homogenous syzygies.

Remark 3.2.7. If $\mathbf{P}$ and $\mathbf{F}$ are graded and the elements of B are homogenous, we use the morphism

$$\varphi\colon \mathbf{G} = \bigoplus_{i=1}^{t}\mathbf{F}^{\mathrm{OI},w(b_i)}(-\deg(b_i)) \to \langle B\rangle_{\mathbf{F}},\ \text{defined by } \epsilon_{\mathrm{id}_{[w(b_i)]},i}\mapsto b_i.$$

It is graded (of degree zero). One checks that as a consequence the syzygies defined in (3.2.2) are homogeneous.

If the generating set B consists of monomials, then one gets a more explicit Gröbner basis of $\mathrm{Syz}(B)$. The result is an OI-version of the description of the syzygy module of a monomial module in the classical situation (see, e.g., [**4**, Lemma 15.1]).

Corollary 3.2.8. *For any finite set of monomials $B\subset\mathbf{F}$, the elements $s_{i,j}^{\sigma,\tau}\in\mathbf{G}$ with $(\mathbf{F}(\sigma)(b_i),\mathbf{F}(\tau)(b_j))\in\mathcal{C}(B)$ and $(\sigma,i)\prec_B(\tau,j)$ form a finite Gröbner basis for* $\mathrm{Syz}(B)$ *with respect to $<_B$, where*

$$s_{i,j}^{\sigma,\tau} = \frac{\mathrm{lcm}(\mathbf{F}(\sigma)(b_i),\mathbf{F}(\tau)(b_j))}{\mathbf{F}(\sigma)(b_i)}\epsilon_{\sigma,i} - \frac{\mathrm{lcm}(\mathbf{F}(\sigma)(b_i),\mathbf{F}(\tau)(b_j))}{\mathbf{F}(\tau)(b_j)}\epsilon_{\tau,j}.$$

Proof. Since B consists of monomials, it is a Gröbner basis of $\langle B\rangle_{\mathbf{F}}$ with respect to any monomial order on $\mathbf{F}$. Thus, Theorem 3.2.6 is applicable. Moreover, each S-polynomial $S(\mathbf{F}(\sigma)(b_i),\mathbf{F}(\tau)(b_j))$ is zero, which implies the stated form of each $s_{i,j}^{\sigma,\tau}$ (see (3.2.2)). The result follows by Theorem 3.2.6. $\square$

We conclude this section with an example using our *Macaulay2* package "OIGroebnerBases.m2" [**11**].

Example 3.2.9 ([**10**])**.** Let $\mathbf{P} = \mathbf{P}^{\mathrm{OI},2}$ and let $\mathbf{F} = \mathbf{F}^{\mathrm{OI},1}\oplus\mathbf{F}^{\mathrm{OI},1}$ have basis $\{e_{\mathrm{id}_{[1]},1}, e_{\mathrm{id}_{[1]},2}\}$. Define

$$f = x_{1,2}x_{1,1}e_{\pi,1} + x_{2,2}x_{2,1}e_{\rho,2}\in\mathbf{F}_2$$

where $\pi : [1] \to [2]$ is given by $1 \mapsto 2$ and $\rho : [1] \to [2]$ is given by $1 \mapsto 1$. We will compute a Gröbner basis G for $\langle f \rangle_{\mathbf{F}}$, and then compute the syzygy module of G. Starting a new *Macaulay2* session, we run the following:

```
i1 : needsPackage "OIGroebnerBases";
i2 : P = makePolynomialOIAlgebra(2, x, QQ);
i3 : F = makeFreeOIModule(e, {1,1}, P);
i4 : installGeneratorsInWidth(F, 2);
i5 : use F_2; f = x_(1,2)*x_(1,1)*e_(2,{2},1)+x_(2,2)*x_(2,1)*e_(2,{1},2);
i7 : G = oiGB {f}
o7 = {x    x    e          + x    x    e          ,
       1,2 1,1 2,{2},1      2,2 2,1 2,{1},2
     --------------------------------------------
     x    x    x    e          - x    x    x    e          }
      2,3 2,2 1,1 3,{2},2       2,3 2,1 1,2 3,{1},2
```

Hence, $\langle f \rangle_{\mathbf{F}}$ has a Gröbner basis (with respect to the lex order)

$$G = \{x_{1,2}x_{1,1}e_{\pi,1} + x_{2,2}x_{2,1}e_{\rho,2},\ x_{2,3}x_{2,2}x_{1,1}e_{\sigma_1,2} - x_{2,3}x_{2,1}x_{1,2}e_{\sigma_2,2}\}$$

where $\sigma_1 : [1] \to [3]$ is given by $1 \mapsto 2$ and $\sigma_2 : [1] \to [3]$ is given by $1 \mapsto 1$. Define the free OI-module $\mathbf{G} = \mathbf{F}^{\mathrm{OI},2}(-2) \oplus \mathbf{F}^{\mathrm{OI},3}(-3)$ with basis $\{d_{\mathrm{id}_{[2]},1}, d_{\mathrm{id}_{[3]},2}\}$. The package assigns these degree shifts automatically. Putting $g = x_{2,3}x_{2,2}x_{1,1}e_{\sigma_1,2} - x_{2,3}x_{2,1}x_{1,2}e_{\sigma_2,2} \in \mathbf{G}_3$ so that $G = \{f, g\}$, we define the map $\varphi : \mathbf{G} \to \langle G \rangle_{\mathbf{F}}$ via $d_{\mathrm{id}_{[2]},1} \mapsto f$ and $d_{\mathrm{id}_{[3]},2} \mapsto g$. We can now compute a Gröbner basis D for $\ker(\varphi)$ (with respect to the induced order $<_G$) using the method `oiSyz`. The user inputs the Gröbner basis G and the basis symbol d:

```
i8 : D = oiSyz(G, d)
o8 = {x    d              - x    d              + 1d                 ,
       1,2 3,{1, 3},1      1,1 3,{2, 3},1        3,{1, 2, 3},2
     -----------------------------------------------------------
     x    d                 - x    d                 , x    d
      2,4 4,{1, 2, 3},2      2,3 4,{1, 2, 4},2        1,2 4,{1, 3, 4},2
     -----------------------------------------------------------
     - x    d                 - x    d                 }
        1,1 4,{2, 3, 4},2      1,3 4,{1, 2, 4},2
```

This says that $\mathrm{Syz}(G) = \ker(\varphi)$ has a Gröbner basis D given by the following elements of $\mathbf{G}$:

$$x_{1,2}d_{\pi_1,1} - x_{1,1}d_{\pi_2,1} - d_{\mathrm{id}_{[3]},2} \in \mathbf{G}_3$$

$$x_{2,4}d_{\pi_3,2} - x_{2,3}d_{\pi_4,2} \in \mathbf{G}_4$$

$$x_{1,2}d_{\pi_5,2} - x_{1,1}d_{\pi_6,2} - x_{1,3}d_{\pi_4,2} \in \mathbf{G}_4$$

where the π_i for $1 \le i \le 6$ are given as follows:

$$\begin{aligned}
\pi_1 &: [2] \to [3] \quad \text{via} \quad 1 \mapsto 1, 2 \mapsto 3 \\
\pi_2 &: [2] \to [3] \quad \text{via} \quad 1 \mapsto 2, 2 \mapsto 3 \\
\pi_3 &: [3] \to [4] \quad \text{via} \quad 1 \mapsto 1, 2 \mapsto 2, 3 \mapsto 3 \\
\pi_4 &: [3] \to [4] \quad \text{via} \quad 1 \mapsto 1, 2 \mapsto 2, 3 \mapsto 4 \\
\pi_5 &: [3] \to [4] \quad \text{via} \quad 1 \mapsto 1, 2 \mapsto 3, 3 \mapsto 4 \\
\pi_6 &: [3] \to [4] \quad \text{via} \quad 1 \mapsto 2, 2 \mapsto 3, 3 \mapsto 4.
\end{aligned}$$

We can verify that these syzygies are homogeneous using the `isHomogeneous` method:

```
i9 : apply(D, isHomogeneous)
o9 = {true, true, true}
```

Finally, we can compute their degrees with `degree`:

```
i10 : apply(D, degree)
o10 = {3, 4, 4}
```

Chapter 4 Application: Computing Free Resolutions

This chapter showcases an application of our theory to the problem of computing free resolutions. We continue to use the notation from the previous chapter. In Section 4.1 we discuss the computation of free OI-resolutions. This will amount to iterating the OI-Schreyer's theorem from Section 3.2. In Section 4.2 we discuss free FI-resolutions, where FI is a closely related category to OI. Both FI- and OI-resolutions have been studied in, for example, [**5, 6, 12, 13**].

4.1 Free OI-Resolutions

Let $\mathbf{A}$ be an OI-algebra. Recall that if $\mathbf{M}$ is an OI-module over $\mathbf{A}$, then a *free resolution* of $\mathbf{M}$ is an exact sequence

$$\mathbf{F}^\bullet : \quad \cdots \to \mathbf{F}^2 \to \mathbf{F}^1 \to \mathbf{F}^0 \to \mathbf{M} \to 0,$$

where each $\mathbf{F}^i$ is a free $\mathbf{A}$-module. If $\mathbf{M}$ is a finitely generated $\mathbf{P}$-module then $\mathbf{M}$ admits a free resolution in which every free module $\mathbf{F}^i$ is finitely generated (see [**13**, Theorem 7.1]). The argument in [**13**] is based on the finiteness results established in that paper and not constructive. Using the results of the previous chapter we obtain the first algorithm to compute, for any integer $p \geq 0$, the first p steps of a free resolution of a submodule $\mathbf{M}$ of a finitely generated free $\mathbf{P}$-module. Indeed, given a finite Gröbner basis B_0 of $\mathbf{M}$, define $\varphi_0 \colon \mathbf{F}^0 \to \mathbf{M}$ by mapping the basis elements of $\mathbf{F}^0$ onto the corresponding generators in B_0. Now let B_1 be a generating set of $\ker(\varphi_0)$ determined by Theorem 3.2.6 and repeat this process.

Procedure 4.1.1 (Method for Computing Free OI-Resolutions)**.** Let $\mathbf{M}$ be a submodule of a finitely generated free $\mathbf{P}$-module. Suppose $\mathbf{M}$ is finitely generated by a subset B. Compute a free resolution for $\mathbf{M}$ as follows:

(i) Determine a finite Gröbner basis B_0 for $\mathbf{M}$ using the OI-Buchberger's algorithm applied to the set B.

(ii) Apply the OI-Schreyer's theorem to B_0 to obtain a finite Gröbner basis B_1 of $\mathrm{Syz}(B_0)$.

(iii) Repeat (ii) as many times as desired to compute a finite Gröbner basis B_n of $\mathrm{Syz}(B_{n-1})$.

(iv) Form (the beginning of) the free resolution $\mathbf{F}^\bullet$ by defining $\mathbf{F}^i$ such that φ_i maps the basis elements of $\mathbf{F}^i$ onto the corresponding generators in B_i as in Definition 3.2.1.

Using Remark 3.2.7, the above procedure can be modified in the case that $\mathbf{M}$ is a graded OI-module to produce a graded free resolution of $\mathbf{M}$.

If $\mathbf{M}$ is a graded submodule of $\mathbf{F}$, then it is shown in [**5**, Theorem 3.10] that $\mathbf{M}$ has a graded free resolution $\mathbf{F}^\bullet$ such that for any other graded free resolution $\mathbf{G}^\bullet$ and for any homological degree d, the rank of $\mathbf{F}^d$ is at most the rank of $\mathbf{G}^d$. By [**5**, Theorem 3.10], the resolution $\mathbf{F}^\bullet$ is unique up to isomorphisms of graded free resolutions and called a *minimal* graded free resolution of $\mathbf{M}$. Our methods allow us to show that there is an algorithm that produces such a minimal free resolution.

As in the classical situation, a key is to characterize the maps occurring in a graded minimal free resolution.

Definition 4.1.2 ([**5**, Definition 3.1])**.** Let $\mathbf{F}$ and $\mathbf{G}$ be free $\mathbf{P}$-modules with bases $\{f_1, \ldots, f_s\}$ and $\{g_1, \ldots, g_t\}$ where each f_i lives in width u_i and each g_j lives in width v_j. A morphism $\varphi \colon \mathbf{F} \to \mathbf{G}$ is determined by the images of the f_i, i.e. by s expressions of the form

$$\varphi(f_i) = \sum_{\substack{1 \le j \le t \\ \varepsilon_{i,j} \in \operatorname{Hom}(v_j, u_i)}} a_{\varepsilon_{i,j}} \mathbf{G}(\varepsilon_{i,j})(g_j)$$

where each $a_{\varepsilon_{i,j}} \in \mathbf{P}_{u_i}$. We say φ is *minimal* if whenever any $\varepsilon_{i,j}$ is an identity map, the coefficient $a_{\varepsilon_{i,j}}$ is not a unit.

This concept allows one to check whether a graded free resolution is minimal. Indeed, [**5**, Theorem 3.10] gives the following characterization:

Theorem 4.1.3. *A graded free resolution $\mathbf{F}^\bullet$ of $\mathbf{M}$ is minimal if and only if each map between free modules in $\mathbf{F}^\bullet$ is minimal.*

We are ready to establish that the first steps of a graded minimal free resolution can be computed algorithmically.

Theorem 4.1.4. *If $\mathbf{M}$ is a graded submodule of $\mathbf{F}$ with a finite generating set B consisting of homogenous elements, then, for any integer $p \geq 0$, there is a finite algorithm that determines the first p steps*

$$\mathbf{F}^p \to \cdots \to \mathbf{F}^2 \to \mathbf{F}^1 \to \mathbf{F}^0 \to \mathbf{M} \to 0$$

in a graded minimal free resolution of $\mathbf{M}$.

Proof. Using the graded version of Procedure 4.1.1, we compute a graded exact sequence

$$\mathbf{F}^p \to \cdots \to \mathbf{F}^2 \to \mathbf{F}^1 \to \mathbf{F}^0 \to \mathbf{M} \to 0,$$

where each $\mathbf{F}^i$ is a finitely generated graded free $\mathbf{P}$-module. If one of the maps in this sequence is not minimal we prune the sequence to obtain an exact sequence such that two consecutive free modules are replaced by free modules of strictly smaller rank. Repeating this process as often as needed beginning with the right-most map one eventually obtains (the beginning) of a graded minimal free resolution of $\mathbf{M}$.

The pruning is described in detail in the proof of [**5**, Lemma 3.5]. □

Our *Macaulay2* script "OIGroebnerBases.m2" [**11**] implements Procedure 4.1.1 and the algorithm in Theorem 4.1.4.

Example 4.1.5 ([**10**])**.** Let $\mathbf{P} = \mathbf{P}^{\mathrm{OI},2}$ and let $\mathbf{F} = \mathbf{F}^{\mathrm{OI},1} \oplus \mathbf{F}^{\mathrm{OI},1}$ have basis $\{e_{\mathrm{id}_{[1]},1}, e_{\mathrm{id}_{[1]},2}\}$, so $\mathbf{F}$ has rank 2. Define

$$f = x_{1,2}x_{1,1}e_{\pi,1} + x_{2,2}x_{2,1}e_{\rho,2} \in \mathbf{F}_3$$

where $\pi : [1] \to [3]$ is given by $1 \mapsto 2$ and $\rho : [1] \to [3]$ is given by $1 \mapsto 1$. Since f is homogeneous, $\langle f \rangle_{\mathbf{F}}$ is a graded submodule, and we will compute the beginning of a graded minimal free resolution using `oiRes`. The user specifies a list of elements (who generate the module to be resolved) and a homological degree. In a new *Macaulay2* session, we run the following:

```
i1 : needsPackage "OIGroebnerBases";
i2 : P = makePolynomialOIAlgebra(2, x, QQ);
i3 : F = makeFreeOIModule(e, {1, 1}, P);
i4 : installGeneratorsInWidth(F, 3);
i5 : use F_3; f = x_(1,2)*x_(1,1)*e_(3,{2},1)+x_(2,2)*x_(2,1)*e_(3,{1},2);
i7 : ranks oiRes({f}, 5)
o7 = 0: rank 1
     1: rank 2
     2: rank 4
     3: rank 7
     4: rank 11
     5: rank 22
```

Note: if one computes out to homological degree n, then only the first $n - 1$ ranks are guaranteed to be minimal. Thus, we have the beginning of a minimal free resolution for $\mathbf{M} = \langle f \rangle_{\mathbf{F}}$:

$$\cdots \to \mathbf{F}^4 \to \mathbf{F}^3 \to \mathbf{F}^2 \to \mathbf{F}^1 \to \mathbf{F}^0 \to \mathbf{M} \to 0$$

where

$$\begin{aligned} \operatorname{rank}(\mathbf{F}^0) &= 1 \\ \operatorname{rank}(\mathbf{F}^1) &= 2 \\ \operatorname{rank}(\mathbf{F}^2) &= 4 \\ \operatorname{rank}(\mathbf{F}^3) &= 7 \\ \operatorname{rank}(\mathbf{F}^4) &= 11. \end{aligned}$$

Finally, we remark that any free resolution $\mathbf{F}^\bullet$ of $\mathbf{M}$ induces, for any integer $w \geq 0$, a free resolution $\mathbf{F}^\bullet_w$ of $\mathbf{M}_w$ by restricting each differential width-wise. Thus, if we think of $\mathbf{M}$ as a sequence $(\mathbf{M}_n)_{n \geq 0}$ of related modules, our techniques "simultaneously" compute a free resolution for each $\mathbf{P}_n$-module $\mathbf{M}_n$ in the sequence. However, even if the resolution $\mathbf{F}^\bullet$ was minimal, its width w component $\mathbf{F}^\bullet_w$ need not be a minimal resolution of $\mathbf{M}_n$ over $\mathbf{P}_n$. For more on this topic, we refer the reader to [**5**].

4.2 Free FI-Resolutions

In this section we show how to adapt our results to compute free FI-resolutions, where FI is the category of finite sets and injective functions. FI-modules over FI-algebras and their resolutions were introduced in [**13**]. By ignoring orders, most of the definitions and results in Chapter 2 have analogous counterparts in the FI-case. We highlight some of these below.

As in the OI-case, it will be notationally beneficial to work with the skeleton of FI. We thus adopt the following convention.

Convention 4.2.1. Regard FI as the category whose objects are intervals $[n]$ and whose morphisms are injective functions $[m] \to [n]$.

We will write $\mathrm{Hom}_{\mathrm{FI}}(m, n)$ for the set of FI-morphisms from $[m]$ to $[n]$. The FI-analog of $\mathbf{P}^{\mathrm{OI},c}$ is defined as follows.

Definition 4.2.2. Let $c \geq 0$ and define an FI-algebra $\mathbf{P} = \mathbf{P}^{\mathrm{FI},c} : \mathrm{FI} \to K\text{-Alg}$ as follows. For each $n \geq 0$ define

$$\mathbf{P}_n = K \begin{bmatrix} x_{1,1} & \cdots & x_{1,n} \\ \vdots & \ddots & \vdots \\ x_{c,1} & \cdots & x_{c,n} \end{bmatrix}$$

and for each $\varepsilon \in \mathrm{Hom}_{\mathrm{FI}}(m, n)$ define $\varepsilon_* : \mathbf{P}_m \to \mathbf{P}_n$ via $x_{i,j} \mapsto x_{i,\varepsilon(j)}$.

Note that $\mathbf{P}_n^{\mathrm{OI},c} = \mathbf{P}_n^{\mathrm{FI},c}$ for all $n \geq 0$. In fact, since OI may be regarded as a subcategory of FI, one obtains $\mathbf{P}^{\mathrm{OI},c}$ from $\mathbf{P}^{\mathrm{FI},c}$ by restricting the domain category to OI. This idea can be formalized for any FI-algebra: if $\mathbf{A}$ is an FI-algebra over K and $I : \mathrm{OI} \to \mathrm{FI}$ is the inclusion functor, then $\mathbf{A} \circ I$ is an OI-algebra over K and is denoted by $\mathbf{A}|_{\mathrm{OI}}$. In this notation, we have $\mathbf{P}^{\mathrm{OI},c} = \mathbf{P}^{\mathrm{FI},c}|_{\mathrm{OI}}$.

The free FI-modules are defined as follows.

Definition 4.2.3 ([**13**])**.** Let $d \in \mathbb{Z}_{\geq 0}$ and let $\mathbf{A}$ be an FI-algebra over K. Define an FI-module $\mathbf{F}^{\mathrm{FI},d} : \mathrm{FI} \to K\text{-Mod}$ as follows. For all $n \geq 0$ define

$$\mathbf{F}_n^{\mathrm{FI},d} = \bigoplus_{\pi \in \mathrm{Hom}_{\mathrm{FI}}(d,n)} \mathbf{A}_n e_\pi \cong (\mathbf{A}_n)^{\binom{n}{d} d!}.$$

For each $\varepsilon \in \mathrm{Hom}_{\mathrm{FI}}(m, n)$, define $\mathbf{F}^{\mathrm{FI},d}(\varepsilon) \colon \mathbf{F}_m^{\mathrm{FI},d} \to \mathbf{F}_n^{\mathrm{FI},d}$ via $ae_\pi \mapsto \varepsilon_*(a) e_{\varepsilon \circ \pi}$. A *free* FI-module over $\mathbf{A}$ is an FI-module $\mathbf{F}$ that is isomorphic to a direct sum $\bigoplus_{\lambda \in \Lambda} \mathbf{F}^{\mathrm{FI},d_\lambda}$ for integers $d_\lambda \geq 0$. If $|\Lambda| = n < \infty$, then $\mathbf{F}$ is said to have *rank* n.

As is the case with free OI-modules, the following properties of free FI-modules hold.

Remark 4.2.4. Let $\mathbf{F} = \bigoplus_{i=1}^s \mathbf{F}^{\mathrm{FI},d_i}$ be a free FI-module over an FI-algebra $\mathbf{A}$.

(i) For all $n \geq 0$ we have

$$\mathbf{F}_n = \bigoplus_{\substack{\pi \in \mathrm{Hom}_{\mathrm{FI}}(d_i,n) \\ 1 \leq i \leq s}} \mathbf{A}_n e_{\pi,i} \cong (\mathbf{A}_n)^{\sum_{i=1}^s \binom{n}{d_i} d_i!}$$

where the second index on $e_{\pi,i}$ is for keeping track of which direct summand it lives in.

(ii) $\mathbf{F}$ is finitely generated as an $\mathbf{A}$-module by all $e_{\mathrm{id}_{[d_i]},i}$. We call these the *basis elements* of $\mathbf{F}$.

(iii) To define an $\mathbf{A}$-linear map $\varphi : \mathbf{F} \to \mathbf{N}$ where $\mathbf{N}$ is any $\mathbf{A}$-module, it suffices to specify images $\varphi(e_{\mathrm{id}_{[d_i]},i}) \in \mathbf{N}_{d_i}$ for each basis element $e_{\mathrm{id}_{[d_i]},i}$.

If $\mathbf{M}$ is any FI-module over an FI-algebra $\mathbf{A}$, and $I : \mathrm{OI} \to \mathrm{FI}$ is the inclusion functor, then $\mathbf{M} \circ I$ is an OI-module over $\mathbf{A}|_{\mathrm{OI}}$ and is denoted by $\mathbf{M}|_{\mathrm{OI}}$. If $\varphi : \mathbf{M} \to \mathbf{N}$ is an FI-morphism of $\mathbf{A}$-modules, then φ restricts to an OI-morphism $\varphi|_{\mathrm{OI}} : \mathbf{M}|_{\mathrm{OI}} \to \mathbf{N}|_{\mathrm{OI}}$ of $\mathbf{A}|_{\mathrm{OI}}$-modules.

The following result relates finitely generated free FI-modules with finitely generated free OI-modules.

Proposition 4.2.5. *Fix integers $d_1, \ldots, d_s \geq 0$ and let $\mathbf{F} = \bigoplus_{i=1}^s \mathbf{F}^{\mathrm{FI},d_i}$ be a free* FI*-module over an* FI*-algebra $\mathbf{A}$. We have an isomorphism of* OI*-modules*

$$\bigoplus_{i=1}^s (\mathbf{F}^{\mathrm{OI},d_i})^{\oplus d_i!} \cong \mathbf{F}|_{\mathrm{OI}}$$

over $\mathbf{A}|_{\mathrm{OI}}$, and thus $\mathbf{F}|_{\mathrm{OI}}$ is a free OI*-module of rank $\sum_{i=1}^s d_i!$.*

Proof. The free OI-module $\bigoplus_{i=1}^s (\mathbf{F}^{\mathrm{OI},d_i})^{\oplus d_i!}$ has a basis given by

$$\{e_{\mathrm{id}_{[d_i]},i,j} \ : \ i \in [s], j \in [d_i!]\}.$$

For each $i \in [s]$, the set $\mathrm{Sym}(d_i) = \mathrm{Hom}_{\mathrm{FI}}(d_i, d_i)$ has $d_i!$ elements, which we denote by $\{\pi_{i,1}, \ldots, \pi_{i,d_i!}\}$. Let $\mathbf{F}$ have basis $\{e_{\mathrm{id}_{[d_i]},i} \ : \ i \in [s]\}$ and define the OI-morphism

$$\varphi : \bigoplus_{i=1}^s (\mathbf{F}^{\mathrm{OI},d_i})^{\oplus d_i!} \to \mathbf{F}|_{\mathrm{OI}} \quad \text{via} \quad e_{\mathrm{id}_{[d_i]},i,j} \mapsto e_{\pi_{i,j},i}.$$

Fix $n \geq 0$ and note that $(\bigoplus_{i=1}^s (\mathbf{F}^{\mathrm{OI},d_i})^{\oplus d_i!})_n$ has an $(\mathbf{A}|_{\mathrm{OI}})_n$-basis given by

$$A = \{e_{\sigma,i,j} \ : \ i \in [s], j \in [d_i!], \sigma \in \mathrm{Hom}_{\mathrm{OI}}(d_i, n)\}.$$

Furthermore, $(\mathbf{F}|_{\mathrm{OI}})_n$ has an $(\mathbf{A}|_{\mathrm{OI}})_n$-basis given by

$$B = \{e_{\tau,i} \ : \ i \in [s], \tau \in \mathrm{Hom}_{\mathrm{FI}}(d_i, n)\}.$$

It suffices to show that φ_n bijects A onto B. Note that $\varphi_n(e_{\sigma,i,j}) = e_{\sigma \circ \pi_{i,j},i}$ and $\sigma \circ \pi_{i,j}$ is an FI-morphism from $[d_i]$ to $[n]$, hence $\varphi_n(A) \subseteq B$. Now pick any $e_{\tau,i} \in B$. There is a unique $\sigma \in \mathrm{Hom}_{\mathrm{OI}}(d_i, n)$ with the same image as τ. Then $\tau = \sigma \circ \pi_{i,j}$ for some $\pi_{i,j} \in \mathrm{Sym}(d_i)$. Now $\varphi_n(e_{\sigma,i,j}) = e_{\sigma \circ \pi_{i,j},i} = e_{\tau,i}$ so that φ_n surjects A onto B. Since A and B both share the same finite cardinality of $\sum_{i=1}^s \binom{n}{d_i} d_i!$, it follows that φ_n bijects A onto B and we are done. $\square$

Remark 4.2.6. The isomorphism in Proposition 4.2.5 is not canonical. In particular, it depends on how we choose to order the sets $\mathrm{Sym}(d_i)$ for $i \in [s]$.

Given a submodule $\mathbf{M}$ of a finitely generated free FI-module $\mathbf{F}$ over $\mathbf{P}^{\mathrm{FI},c}$, we have an induced submodule $\mathbf{M}|_{\mathrm{OI}}$ of the finitely generated free OI-module $\mathbf{F}|_{\mathrm{OI}}$ over $\mathbf{P}^{\mathrm{OI},c}$. Thus, one could immediately apply the results of the previous chapter to compute a free OI-resolution of $\mathbf{M}|_{\mathrm{OI}}$. However, such a resolution will only be compatible with the OI-symmetries of $\mathbf{M}$. In order to resolve $\mathbf{M}$ in a way compatible with all its FI-symmetries, one desires a free FI-resolution

$$\mathbf{F}^\bullet : \quad \cdots \to \mathbf{F}^2 \to \mathbf{F}^1 \to \mathbf{F}^0 \to \mathbf{M} \to 0$$

where each $\mathbf{F}^i$ is a free FI-module over $\mathbf{P}^{\mathrm{FI},c}$. By [**13**, Theorem 7.1], such a free FI-resolution of $\mathbf{M}$ always exists, where each $\mathbf{F}^i$ is a finitely generated free $\mathbf{P}^{\mathrm{FI},c}$-module. As noted in the previous section, the existence argument given in [**13**] is nonconstructive. In what follows, we present the first algorithmic method for computing a free FI-resolution of $\mathbf{M}$ in which each free $\mathbf{P}^{\mathrm{FI},c}$-module is finitely generated.

We need some more preparatory observations.

Lemma 4.2.7. *Let $\varphi : \mathbf{M} \to \mathbf{N}$ be a map of* FI*-modules over an* FI*-algebra $\mathbf{A}$. Suppose $\ker(\varphi|_{\mathrm{OI}})$ is generated as an $\mathbf{A}|_{\mathrm{OI}}$-module by a subset $B \subseteq \mathbf{M}|_{\mathrm{OI}}$. Then $\ker(\varphi)$ is generated as an $\mathbf{A}$-module by $B \subseteq \mathbf{M}$.*

Proof. First note that $B \subseteq \ker(\varphi)$ since for all $b \in B$, we have $\varphi(b) = \varphi|_{\mathrm{OI}}(b) = 0$. Now pick any $f \in \ker(\varphi)$. Then $\varphi|_{\mathrm{OI}}(f) = \varphi(f) = 0$, so $f \in \ker(\varphi|_{\mathrm{OI}})$. Thus we have an expression $f = \sum a_i q_i$ for elements $a_i \in (\mathbf{A}|_{\mathrm{OI}})_{w(f)}$ and $q_i \in \mathrm{Orb}_{\mathrm{OI}}(B, w(f))$. The result follows by noticing that $(\mathbf{A}|_{\mathrm{OI}})_{w(f)} = \mathbf{A}_{w(f)}$ and $\mathrm{Orb}_{\mathrm{OI}}(B, w(f)) \subseteq \mathrm{Orb}_{\mathrm{FI}}(B, w(f))$. □

Lemma 4.2.8. *Let $\varphi : \mathbf{G} \to \mathbf{F}$ be a map of finitely generated free* OI*-modules over $\mathbf{P} = \mathbf{P}^{\mathrm{OI},c}$. Then a finite generating set for $\ker(\varphi)$ can be computed in finite time.*

Proof. Let $\mathbf{G}$ have basis $\{\epsilon_{\mathrm{id}_{[d_i]},i} \ : \ i \in [s]\}$ for some integers $d_1, \ldots, d_s \geq 0$. For each $i \in [s]$ define $b_i = \varphi(\epsilon_{\mathrm{id}_{[d_i]},i}) \in \mathbf{F}_{d_i}$. Fix a monomial order $<$ on $\mathbf{F}$ and let $B = \{b_i \ : \ i \in [s]\} \subset \mathbf{F}$. One checks in finite time whether B is a Gröbner basis of $\langle B \rangle_{\mathbf{F}}$ with respect to $<$ by using the OI-Buchberger's criterion. If B is a Gröbner basis then the result follows immediately by applying the OI-Schreyer's theorem to φ.

Suppose now that B is not a Gröbner basis. We will run the OI-Buchberger's algorithm on B. Set $B_0 = B$ and let $(\mathbf{F}(\sigma_1)(b_{i_1}), \mathbf{F}(\tau_1)(b_{j_1})) \in \mathcal{C}(B_0)$ be a critical pair in width $m_1 \geq 0$ whose S-polynomial has a nonzero remainder $r_1 \in \mathbf{F}_{m_1}$ modulo B_0. This gives an expression

$$S(\mathbf{F}(\sigma_1)(b_{i_1}), \mathbf{F}(\tau_1)(b_{j_1})) = \sum_\ell a_{1,\ell} q_{1,\ell} + r_1 \tag{4.2.1}$$

with $a_{1,\ell} \in \mathbf{P}_{m_1}$ and $q_{1,\ell} \in \mathrm{Orb}(B_0, m_1)$. Now set $B_1 = B_0 \cup \{r_1\}$ and repeat this process until the algorithm terminates to obtain sets $B_0 \subset \cdots \subset B_t$ (where B_t is a Gröbner basis w.r.t. $<$) and expressions

$$S(\mathbf{F}(\sigma_v)(b_{i_v}), \mathbf{F}(\tau_v)(b_{j_v})) = \sum_\ell a_{v,\ell} q_{v,\ell} + r_v \tag{4.2.2}$$

such that $a_{v,\ell} \in \mathbf{P}_{m_v}$ and $q_{v,\ell} \in \mathrm{Orb}(B_{v-1}, m_v)$ for all $v \in [t]$. Define now $G = \{b_1, \ldots, b_s, b_{s+1}, \ldots, b_{s+t}\}$ where $b_{s+v} = r_v$ for each $v \in [t]$. Let $d_{s+v} = m_v = w(b_{s+v})$ for all $v \in [t]$ and let $\widehat{\mathbf{G}}$ be the free $\mathbf{P}$-module with basis $\{\epsilon_{\mathrm{id}_{[d_i]},i} \ : \ i \in [s+t]\}$. Define the $\mathbf{P}$-linear map

$$\widehat{\varphi} : \widehat{\mathbf{G}} \to \mathbf{F} \quad \text{via} \quad \epsilon_{\mathrm{id}_{[d_i]},i} \mapsto b_i$$

and equip $\widehat{\mathbf{G}}$ with the Schreyer order $<_G$ induced by $<$ and G. By the OI-Schreyer's theorem, $\ker(\widehat{\varphi})$ is generated by syzygies of the form $s_{i,j}^{\sigma,\tau}$ (see (3.2.2)) such that $(\mathbf{F}(\sigma)(b_i), \mathbf{F}(\tau)(b_j)) \in \mathcal{C}(G)$. Call this generating set $\mathcal{S} \subset \ker(\widehat{\varphi})$. Using the expressions from (4.2.2), one obtains syzygies $s_{i_v,j_v}^{\sigma_v,\tau_v} \in \ker(\widehat{\varphi})$ with the following special form:

$$s_{i_v,j_v}^{\sigma_v,\tau_v} = \epsilon_{\mathrm{id}_{[d_{s+v}]},s+v} + \sum_n a_n \epsilon_{\pi_n,u_n} \tag{4.2.3}$$

where $a_n \in \mathbf{P}_{d_{s+v}}$ and $u_n < s+v$ for all n. Let $\mathbf{M}$ be the submodule of $\ker(\widehat{\varphi})$ generated by the $s_{i_v,j_v}^{\sigma_v,\tau_v}$ with $v \in [t]$. Passing to the quotient, we have an induced map $\psi : \widehat{\mathbf{G}}/\mathbf{M} \to \mathbf{F}$ given by $g + \mathbf{M}_n \mapsto \widehat{\varphi}(g)$. If $\pi : \widehat{\mathbf{G}} \to \widehat{\mathbf{G}}/\mathbf{M}$ denotes the canonical projection map, then $\pi(\mathcal{S})$ is a generating set for $\ker(\psi)$. Using (4.2.3), we have an isomorphism $\Phi : \mathbf{G} \to \widehat{\mathbf{G}}/\mathbf{M}$ given by $\epsilon_{\mathrm{id}_{[d_i]},i} \mapsto \pi(\epsilon_{\mathrm{id}_{[d_i]},i})$ for all $i \in [s]$. Since $\varphi = \psi \circ \Phi$, a finite generating set for $\ker(\varphi)$ is given by $\Phi^{-1}(\pi(\mathcal{S}))$ as desired. $\square$

We conclude this chapter by putting everything together to give an algorithm for computing free FI-resolutions.

Procedure 4.2.9 (Method for Computing Free FI-Resolutions)**.** Let $\mathbf{M}$ be a submodule of a finitely generated free FI-module $\mathbf{F}$ over $\mathbf{P} = \mathbf{P}^{\mathrm{FI},c}$. Suppose $\mathbf{M}$ is finitely generated by a subset B. For notational convenience, set $\mathbf{F}^{-1} = \mathbf{F}$ and set $B_0 = B$. Compute a free FI-resolution for $\mathbf{M}$ as follows:

(i) Define the map $\varphi_0 : \mathbf{F}^0 \to \mathbf{F}^{-1}$ by mapping the basis elements of $\mathbf{F}^0$ onto the corresponding generators in B_0.

(ii) Apply Proposition 4.2.5 to obtain a map $\varphi_0|_{\mathrm{OI}} : \mathbf{F}^0|_{\mathrm{OI}} \to \mathbf{F}^{-1}|_{\mathrm{OI}}$ of finitely generated free OI-modules over $\mathbf{P}^{\mathrm{OI},c}$.

(iii) Use Lemma 4.2.8 to compute a finite generating set B_1 of $\ker(\varphi_0|_{\mathrm{OI}})$. By Lemma 4.2.7, B_1 generates $\ker(\varphi_0)$ as an FI-module over $\mathbf{P}^{\mathrm{FI},c}$.

(iv) Repeat steps (i)-(iii) as many times as desired to compute a finite generating set B_n of $\ker(\varphi_n)$.

(v) Form (the beginning of) the free FI-resolution $\mathbf{F}^\bullet$ by defining $\mathbf{F}^i$ such that φ_i maps the basis elements of $\mathbf{F}^i$ onto the corresponding generators in B_i.

Chapter 5 Open Questions

Below is a collection of open questions related to our work.

Degree Bounds. Much work has been done in analyzing the complexity of Buchberger's algorithm. For example, a bound on the largest degree of an element in the reduced Gröbner basis of an ideal in a finite dimensional polynomial ring was given in [**3**] which depends on the degrees of the generators of the ideal and the dimension of the ring. Finding similar results for OI-modules is an interesting and open problem.

Question 1. *How can one bound the largest degree of an element in the reduced Gröbner basis of an* OI-*module?*

Gröbner Stabilization. Continuing the theme of analyzing Buchberger's algorithm, one can also ask for a bound on the largest width of an element in the reduced basis.

Question 2. *How can one bound the largest width of an element in the reduced Gröbner basis of an* OI-*module?*

Asymptotic Monomiality. OI-modules provide a framework for considering asymptotic questions. In particular, one can ask questions about the "long-term behavior" of sequences of related ideals such as the following.

Question 3. *Given an ideal* $\mathbf{I}$ *of* $\mathbf{P} = \mathbf{P}^{\mathrm{OI},c}$, *is there a finite algorithm to detect whether* $\mathbf{I}_n$ *is a monomial ideal for* $n \gg 0$*?*

Appendices

Appendix A: OIGroebnerBases.m2 Source Code

In this section we include the source code for our *Macaulay2* package "OIGroebner-Bases.m2" [**11**]. Line breaks are indicated with the special character "↪".

```
-- -*- coding: utf-8 -*-

-*
Copyright 2023 Michael Morrow

This program is free software; you can redistribute it and/or modify it under the
    ↪ terms of the GNU General Public License as published by the Free Software
    ↪ Foundation; either version 2 of the License, or (at your option) any later
    ↪ version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY
    ↪ WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
    ↪ FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
    ↪ details.

You should have received a copy of the GNU General Public License along with this
    ↪ program; if not, see <http://www.gnu.org/licenses/>
*-

newPackage("OIGroebnerBases",
    Headline => "OI-modules over Noetherian polynomial OI-algebras",
    Version => "1.0.0",
    Date => "September 6, 2023",
    Keywords => { "Commutative Algebra" },
    Authors => {
        { Name => "Michael Morrow", HomePage => "https://michaelmorrow.me", Email
            ↪ => "michaelhmorrow98@gmail.com" }
    },
    DebuggingMode => false,
    HomePage => "https://github.com/morrowmh/OIGroebnerBases"
)

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-- EXPORT AND PROTECT ----------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

export {
       "PolynomialOIAlgebra",
       "FreeOIModule", "ModuleInWidth", "VectorInWidth", "FreeOIModuleMap",
       "OIResolution",
```

```
        "ColUpRowUp", "ColUpRowDown", "ColDownRowUp", "ColDownRowDown",
        "RowUpColUp", "RowUpColDown", "RowDownColUp", "RowDownColDown",
        "makePolynomialOIAlgebra",
        "makeFreeOIModule", "installGeneratorsInWidth", "isZero", "getBasisElements", "getWidth", "getFreeOIModule", "getSchreyerMap", "getRank", "oiOrbit",
        "oiGB", "minimizeOIGB", "reduceOIGB", "isOIGB",
        "oiSyz",
        "describeFull", "ranks", "restrictedRanks", "oiRes", "isComplex",
        "VariableOrder",
        "DegreeShifts", "OIMonomialOrder",
        "TopNonminimal"
}

scan({
        targWidth, img,
        varRows, varSym, baseField, varOrder, algebras, maps,
        basisSym, genWidths, degShifts, polyOIAlg, monOrder, modules, basisKeys, wid, rawMod, freeOIMod, key, vec, oiMap, srcMod, targMod, genImages,
        quo, rem, divTuples,
        map0, idx0, im0, map1, idx1, im1
}, protect)

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-- BODY ------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

-- Should be of the form {targWidth => ZZ, img => List}
OIMap = new Type of HashTable

net OIMap := f -> "Source: [" | net(#f.img) | "] Target: [" | net f.targWidth | "]" || "Image: " | net f.img

-- Evaluate an OI-map f at an integer n
OIMap ZZ := (f, n) -> f.img#(n - 1)

-- Make a new OIMap
-- Args: n = ZZ, L = List
makeOIMap := (n, L) -> new OIMap from {targWidth => n, img => L}

-- Get the OI-maps between two widths
-- Args: m = ZZ, n = ZZ
getOIMaps := (m, n) -> (
    if n < m then return {};

    sets := subsets(1..n, m);
    for i to #sets - 1 list makeOIMap(n, sets#i)
)

-- Compose two OI-maps
OIMap OIMap := (f, g) -> makeOIMap(f.targWidth, for i from 1 to #g.img list f g i)
```

```
-- Should be of the form {varRows => ZZ, varSym => Symbol, baseField => Ring, varOrder => Symbol, algebras => MutableHashTable, maps => MutableHashTable}
PolynomialOIAlgebra = new Type of HashTable

toString PolynomialOIAlgebra := P -> "(" | toString P.varRows | ", " | toString P.varSym | ", " | toString P.baseField | ", " | toString P.varOrder | ")"

net PolynomialOIAlgebra := P -> "Number of variable rows: " | net P.varRows ||
    "Variable symbol: " | net P.varSym ||
    "Base field: " | net P.baseField ||
    "Variable order: " | net P.varOrder

makePolynomialOIAlgebra = method(TypicalValue => PolynomialOIAlgebra, Options => {VariableOrder => RowUpColUp})
makePolynomialOIAlgebra(ZZ, Symbol, Ring) := opts -> (c, x, K) -> (
    if c < 1 then error "expected at least one row of variables";

    v := opts.VariableOrder;
    if not member(v, {
        ColUpRowUp, ColUpRowDown, ColDownRowUp, ColDownRowDown,
        RowUpColUp, RowUpColDown, RowDownColUp, RowDownColDown
    }) then error "invalid variable order";

    new PolynomialOIAlgebra from {
            varRows => c,
            varSym => x,
            baseField => K,
            varOrder => v,
            algebras => new MutableHashTable,
            maps => new MutableHashTable}
)

-- Lookup table for linearFromRowCol
orderTable := new HashTable from {
    ColUpRowUp => (P, n, i, j) -> P.varRows * (n - j + 1) - i, -- x_(i',j') < x_(i,j) if j'<j or j'=j and i'<i
    ColUpRowDown => (P, n, i, j) -> P.varRows * (n - j) + i - 1, -- x_(i',j') < x_(i,j) if j'<j or j'=j and i'>i
    ColDownRowUp => (P, n, i, j) -> P.varRows * j - i,        -- x_(i',j') < x_(i,j) if j'>j or j'=j and i'<i
    ColDownRowDown => (P, n, i, j) -> P.varRows * (j - 1) + i - 1, -- x_(i',j') < x_(i,j) if j'>j or j'=j and i'>i
    RowUpColUp => (P, n, i, j) -> n * (P.varRows - i + 1) - j, -- x_(i',j') < x_(i,j) if i'<i or i'=i and j'<j
    RowUpColDown => (P, n, i, j) -> n * (P.varRows - i) + j - 1, -- x_(i',j') < x_(i,j) if i'<i or i'=i and j'>j
    RowDownColUp => (P, n, i, j) -> n * i - j,                 -- x_(i',j') < x_(i,j) if i'>i or i'=i and j'<j
    RowDownColDown => (P, n, i, j) -> n * (i - 1) + j - 1      -- x_(i',j') < x_(i,j) if i'>i or i'=i and j'>j
}

-- Linearize the variables based on P.varOrder
-- Args: P = PolynomialOIAlgebra, n = ZZ, i = ZZ, j = ZZ
```

```
linearFromRowCol := (P, n, i, j) -> (orderTable#(P.varOrder))(P, n, i, j)

-- Get the algebra of P in width n
-- Args: P = PolynomialOIAlgebra, n = ZZ
getAlgebraInWidth := (P, n) -> (
    -- Return the algebra if it already exists
    if P.algebras#?n then return P.algebras#n;

    -- Generate the variables
    local ret;
    variables := new MutableList;
    for j from 1 to n do
        for i from 1 to P.varRows do variables#(linearFromRowCol(P, n, i, j)) = P.
            ↪ varSym_(i, j);

    -- Make the algebra
    ret = P.baseField[toList variables, Degrees => {#variables:1}, MonomialOrder =>
        ↪  {Lex}];

    -- Store the algebra
    P.algebras#n = ret
)

PolynomialOIAlgebra _ ZZ := (P, n) -> getAlgebraInWidth(P, n)

-- Get the algebra map induced by an OI-map
-- Args: P = PolynomialOIAlgebra, f = OIMap
getInducedAlgebraMap := (P, f) -> (
    -- Return the map if it already exists
    if P.maps#?f then return P.maps#f;

    -- Generate the assignments
    m := #f.img;
    n := f.targWidth;
    src := P_m;
    targ := P_n;
    subs := flatten for j from 1 to m list
        for i from 1 to P.varRows list src_(linearFromRowCol(P, m, i, j)) => targ_(
            ↪ linearFromRowCol(P, n, i, f j)); -- Permute the second index

    -- Make the map
    ret := map(targ, src, subs);

    -- Store the map
    P.maps#f = ret
)

-- Should be of the form {basisSym => Symbol, genWidths => List, degShifts => List
    ↪ , polyOIAlg => PolynomialOIAlgebra, monOrder => Thing, modules =>
    ↪ MutableHashTable, maps => MutableHashTable, basisKeys => MutableHashTable}
FreeOIModule = new Type of HashTable

toString FreeOIModule := F -> "(" | toString F.basisSym | ", " | toString F.
    ↪ genWidths | ", " | toString F.degShifts | ")"
```

```
net FreeOIModule := F -> (
    monOrderNet := if F.monOrder === Lex then net Lex
    else if instance(F.monOrder, List) then "Schreyer"
    else error "invalid monomial order";

    "Basis symbol: " | net F.basisSym ||
    "Basis element widths: " | net F.genWidths ||
    "Degree shifts: " | net F.degShifts ||
    "Polynomial OI-algebra: " | toString F.polyOIAlg ||
    "Monomial order: " | monOrderNet
)

makeFreeOIModule = method(TypicalValue => FreeOIModule, Options => {DegreeShifts
    ↪ => null, OIMonomialOrder => Lex})
makeFreeOIModule(Symbol, List, PolynomialOIAlgebra) := opts -> (e, W, P) -> (
    shifts := if opts.DegreeShifts === null then toList(#W : 0)
    else if instance(opts.DegreeShifts, List) and #opts.DegreeShifts === #W then
        ↪ opts.DegreeShifts
    else error "invalid DegreeShifts option";

    -- Validate the monomial order
    if not opts.OIMonomialOrder === Lex and not (
        instance(opts.OIMonomialOrder, List) and
        W === apply(opts.OIMonomialOrder, getWidth) and
        #set apply(opts.OIMonomialOrder, getFreeOIModule) == 1) then error "invalid
            ↪  monomial order";

    new FreeOIModule from {
        basisSym => e,
        genWidths => W,
        degShifts => shifts,
        polyOIAlg => P,
        monOrder => opts.OIMonomialOrder,
        modules => new MutableHashTable,
        maps => new MutableHashTable,
        basisKeys => new MutableHashTable}
)

-- Get the rank of a FreeOIModule
getRank = method(TypicalValue => ZZ)
getRank FreeOIModule := F -> #F.genWidths

-- Check if a FreeOIModule is zero
isZero = method(TypicalValue => Boolean)
isZero FreeOIModule := F -> F.genWidths === {}

-- Should be of the form {wid => ZZ, rawMod => Module, freeOIMod => FreeOIModule}
ModuleInWidth = new Type of HashTable

net ModuleInWidth := M -> net M.rawMod | " in width " | net M.wid |
    if not set M.freeOIMod.degShifts === set {0} and not zero M.rawMod then ",
        ↪ degrees " | net flatten degrees M.rawMod else ""
```

```
-- Get the module of F in width n
-- Args: F = FreeOIModule, n = ZZ
getModuleInWidth := (F, n) -> (
    -- Return the module if it already exists
    if F.modules#?n then return F.modules#n;

    -- Generate the degrees
    alg := getAlgebraInWidth(F.polyOIAlg, n);
    degList := for i to #F.genWidths - 1 list binomial(n, F.genWidths#i) : F.
        ↪ degShifts#i;

    -- Generate and store the module
    F.modules#n = new ModuleInWidth of VectorInWidth from hashTable {
        wid => n,
        rawMod => alg^degList,
        freeOIMod => F
    }
)

FreeOIModule _ ZZ := (F, n) -> getModuleInWidth(F, n)

use ModuleInWidth := M -> ( use getAlgebraInWidth(M.freeOIMod.polyOIAlg, M.wid); M
    ↪ )

VectorInWidth = new Type of HashTable

-- Should be of the form {key => Sequence, vec => VectorInWidth}
KeyedVectorInWidth = new Type of HashTable

-- Get the width of a VectorInWidth
getWidth = method(TypicalValue => ZZ)
getWidth VectorInWidth := v -> (class v).wid

-- Get the FreeOIModule of a VectorInWidth
getFreeOIModule = method(TypicalValue => FreeOIModule)
getFreeOIModule VectorInWidth := v -> (class v).freeOIMod

-- Get the basis keys in a given width
-- Args: F = FreeOIModule, n = ZZ
getBasisKeys := (F, n) -> (
    -- Return the basis keys if they already exist
    if F.basisKeys#?n then return F.basisKeys#n;

    -- Store the basis keys
    F.basisKeys#n = flatten for i to #F.genWidths - 1 list
        for oiMap in getOIMaps(F.genWidths#i, n) list (oiMap, i + 1)
)

-- Make a VectorInWidth
-- Args: M = ModuleInWidth, A => List
makeVectorInWidth := (M, A) -> new M from new VectorInWidth from A

-- Make a VectorInWidth with a single basis key
-- Args: M = ModuleInWidth, key = Sequence, elt = RingElement
```

```
makeSingle := (M, key, elt) -> (
    A := for keyj in getBasisKeys(M.freeOIMod, M.wid) list keyj => if key === keyj
        ↪ then elt else 0_(class elt);
    makeVectorInWidth(M, A)
)

-- Make a KeyedVectorInWidth
-- Args: M = ModuleInWidth, key0 = Sequence, elt = RingElement
makeKeyedVectorInWidth := (M, key0, elt) -> new KeyedVectorInWidth from {key =>
    ↪ key0, vec => makeSingle(M, key0, elt)}

-- Make the zero VectorInWidth
-- Args: M = ModuleInWidth
makeZero := M -> (
    A := for key in getBasisKeys(M.freeOIMod, M.wid) list key => 0_(
        ↪ getAlgebraInWidth(M.freeOIMod.polyOIAlg, M.wid));
    makeVectorInWidth(M, A)
)

-- Install the generators in a given width
installGeneratorsInWidth = method();
installGeneratorsInWidth(FreeOIModule, ZZ) := (F, n) -> (
    M := getModuleInWidth(F, n);
    K := getBasisKeys(F, n);

    for key in K do F.basisSym_((key#0).targWidth, (key#0).img, key#1) <-
        ↪ makeSingle(M, key, 1_(getAlgebraInWidth(F.polyOIAlg, n)))
)

-- Check if a VectorInWidth is zero
isZero VectorInWidth := v -> (
    for val in values v do if not zero val then return false;
    true
)

-- Get the terms of a VectorInWidth
terms VectorInWidth := v -> flatten for key in keys v list
    for term in terms v#key list makeSingle(class v, key, term)

-- Get the keyed terms of a VectorInWidth
-- Args: v = VectorInWidth
keyedTerms := v -> flatten for key in keys v list
    for term in terms v#key list makeKeyedVectorInWidth(class v, key, term)

-- Get the keyed singles of a VectorInWidth
-- Args: v = VectorInWidth
keyedSingles := v -> flatten for key in keys v list
    if zero v#key then continue else makeKeyedVectorInWidth(class v, key, v#key)

-- Get the ith basis element of a FreeOIModule
-- Args: F = FreeOIModule, i = ZZ
-- Comment: expects 0 <= i <= #F.genWidths - 1
getBasisElement := (F, i) -> (
    n := F.genWidths#i;
```

```
    M := getModuleInWidth(F, n);
    key := (makeOIMap(n, toList(1..n)), i + 1);
    makeSingle(M, key, 1_(getAlgebraInWidth(F.polyOIAlg, n)))
)

-- Get the basis elements of a FreeOIModule
getBasisElements = method(TypicalValue => List)
getBasisElements FreeOIModule := F -> for i to #F.genWidths - 1 list
    ↪ getBasisElement(F, i)

-- Get the keyed lead term of a VectorInWidth
keyedLeadTerm := v -> (
    if isZero v then return new KeyedVectorInWidth from {key => null, vec => v};

    T := keyedTerms v;
    if #T === 1 then return T#0;

    largest := T#0;
    for term in T do if largest < term then largest = term;
    largest
)

-- Get the lead term of a VectorInWidth
leadTerm VectorInWidth := v -> (keyedLeadTerm v).vec

-- Get the keyed lead monomial of a VectorInWidth
keyedLeadMonomial := v -> (
    if isZero v then error "the zero element has no lead monomial";
    lt := keyedLeadTerm v;
    makeKeyedVectorInWidth(class v, lt.key, leadMonomial lt.vec#(lt.key))
)

-- Get the lead monomial of a VectorInWidth
leadMonomial VectorInWidth := v -> (keyedLeadMonomial v).vec

-- Get the lead coefficient of a VectorInWidth
leadCoefficient VectorInWidth := v -> (
    if isZero v then return 0_(getAlgebraInWidth((class v).freeOIMod.polyOIAlg, (
        ↪ class v).wid));
    lt := keyedLeadTerm v;
    leadCoefficient lt.vec#(lt.key)
)

-- Cache for storing KeyedVectorInWidth term comparisons
compCache = new MutableHashTable

-- Comparison method for KeyedVectorInWidth terms
KeyedVectorInWidth ? KeyedVectorInWidth := (v, w) -> (
    keyv := v.key;
    keyw := w.key;
    monv := leadMonomial(v.vec#keyv);
    monw := leadMonomial(w.vec#keyw);
    oiMapv := keyv#0;
    oiMapw := keyw#0;
```

```
    idxv := keyv#1;
    idxw := keyw#1;
    fmod := (class v.vec).freeOIMod;
    ord := fmod.monOrder;

    -- Return the comparison if it already exists
    if compCache#?(keyv, monv, keyw, monw, ord) then return compCache#(keyv, monv,
        ↪  keyw, monw, ord);

    -- Generate the comparison
    local ret;
    if v === w then ret = symbol ==
    else if ord === Lex then ( -- Lex order
        if not idxv === idxw then ( if idxv < idxw then ret = symbol > else ret =
            ↪ symbol < )
        else if not oiMapv.targWidth === oiMapw.targWidth then ret = oiMapv.
            ↪ targWidth ? oiMapw.targWidth
        else if not oiMapv.img === oiMapw.img then ret = oiMapv.img ? oiMapw.img
        else ret = monv ? monw
    )
    else if instance(ord, List) then ( -- Schreyer order
        fmodMap := new FreeOIModuleMap from {srcMod => fmod, targMod =>
            ↪ getFreeOIModule ord#0, genImages => ord};
        lmimgv := keyedLeadMonomial fmodMap v.vec;
        lmimgw := keyedLeadMonomial fmodMap w.vec;

        if not lmimgv === lmimgw then ret = lmimgv ? lmimgw
        else if not idxv === idxw then ( if idxv < idxw then ret = symbol > else
            ↪ ret = symbol < )
        else if not oiMapv.targWidth === oiMapw.targWidth then ( if oiMapv.
            ↪ targWidth < oiMapw.targWidth then ret = symbol > else ret = symbol <
            ↪  )
        else if not oiMapv.img === oiMapw.img then ( if oiMapv.img < oiMapw.img
            ↪ then ret = symbol > else ret = symbol < )
        else ret = symbol ==
    )
    else error "invalid monomial order";

    -- Store the comparison
    compCache#(keyv, monv, keyw, monw, ord) = ret
)

-- Addition method for VectorInWidth
VectorInWidth + VectorInWidth := (v, w) -> (
    if not class v === class w then error("cannot add " | net v | " and " | net w);
        ↪
    if isZero v then return w else if isZero w then return v;

    cls := class v;
    K := getBasisKeys(cls.freeOIMod, cls.wid);
    A := for key in K list key => v#key + w#key;

    makeVectorInWidth(cls, A)
)
```

```
-- Module multiplication method for VectorInWidth
RingElement * VectorInWidth := (r, v) -> (
    clsv := class v;
    fmod := clsv.freeOIMod;
    wid := clsv.wid;

    if not class r === getAlgebraInWidth(fmod.polyOIAlg, wid) then error("cannot
        ↪ multiply " | net r | " and " | net v);
    if isZero v then return v;

    K := getBasisKeys(fmod, wid);
    A := for key in K list key => r * v#key;

    makeVectorInWidth(clsv, A)
)

-- Number multiplication method for VectorInWidth
Number * VectorInWidth := (n, v) -> (
    cls := class v;
    n_(getAlgebraInWidth(cls.freeOIMod.polyOIAlg, cls.wid)) * v
)

-- Negative method for VectorInWidth
- VectorInWidth := v -> (-1) * v

-- Subtraction method for VectorInWidth
VectorInWidth - VectorInWidth := (v, w) -> v + -w

-- Get the degree of a VectorInWidth
degree VectorInWidth := v -> (
    if isZero v then return 0;

    lt := keyedLeadTerm v;
    elt := lt.vec#(lt.key);
    degElt := (degree elt)#0;

    basisIdx := lt.key#1;
    degBasisElt := -(class v).freeOIMod.degShifts#(basisIdx - 1);

    degElt + degBasisElt
)

-- Check if a VectorInWidth is homogeneous
isHomogeneous VectorInWidth := v -> (
    if isZero v then return true;

    #set apply(terms v, degree) === 1
)

-- Make a VectorInWidth monic
-- Args: v = VectorInWidth
-- Comment: assumes v is nonzero
makeMonic := v -> (1 / leadCoefficient v) * v
```

```
-- Display a VectorInWidth with terms in order
net VectorInWidth := v -> (
    if isZero v then return net 0;

    fmod := (class v).freeOIMod;
    sorted := reverse sort keyedTerms v;

    firstTerm := sorted#0;
    N := net firstTerm.vec#(firstTerm.key) | net fmod.basisSym_(toString (
        ↪ firstTerm.key#0).targWidth, toString (firstTerm.key#0).img, toString
        ↪ firstTerm.key#1);

    for i from 1 to #sorted - 1 do (
        term := sorted#i;
        elt := term.vec#(term.key);
        coeff := leadCoefficient elt;
        basisNet := net fmod.basisSym_(toString (term.key#0).targWidth, toString (
            ↪ term.key#0).img, toString term.key#1);

        N = N | if coeff > 0 then " + " | net elt | basisNet else " - " | net(-elt)
            ↪  | basisNet
    );

    N
)

-- Should be of the form {freeOIMod => FreeOIModule, oiMap => OIMap, img =>
    ↪ HashTable}
InducedModuleMap = new Type of HashTable

-- Get the module map induced by an OI-map
-- Args: F = FreeOIModule, f = OIMap
getInducedModuleMap := (F, f) -> (
    -- Return the map if it already exists
    if F.maps#?(F, f) then return F.maps#(F, f);

    -- Generate the basis element assignments
    m := #f.img;
    K := getBasisKeys(F, m);
    H := hashTable for key in K list key => (f key#0, key#1);

    -- Store the map
    F.maps#(F, f) = new InducedModuleMap from {freeOIMod => F, oiMap => f, img =>
        ↪ H}
)

-- Apply an InducedModuleMap to a VectorInWidth
-- Comment: expects v to belong to the domain of f
InducedModuleMap VectorInWidth := (f, v) -> (
    fmod := f.freeOIMod;
    targWidth := f.oiMap.targWidth;
    targMod := getModuleInWidth(fmod, targWidth);
```

```
    -- Handle the zero vector
    if isZero v then return makeZero targMod;

    algMap := getInducedAlgebraMap(fmod.polyOIAlg, f.oiMap);

    sum for single in keyedSingles v list makeSingle(targMod, f.img#(single.key),
        ↪ algMap single.vec#(single.key))
)

-- Should be of the form {srcMod => FreeOIModule, targMod => FreeOIModule,
    ↪ genImages => List}
FreeOIModuleMap = new Type of HashTable

describe FreeOIModuleMap := f -> "Source: " | toString f.srcMod | " Target: " |
    ↪ toString f.targMod || "Basis element images: " | net f.genImages

net FreeOIModuleMap := f -> "Source: " | toString f.srcMod | " Target: " |
    ↪ toString f.targMod

image FreeOIModuleMap := f -> f.genImages

-- Check if a FreeOIModuleMap is zero
isZero FreeOIModuleMap := f -> isZero f.srcMod or isZero f.targMod or set apply(f.
    ↪ genImages, isZero) === set {true}

-- Apply a FreeOIModuleMap to a VectorInWidth
-- Comment: expects v to belong to the domain of f
FreeOIModuleMap VectorInWidth := (f, v) -> (
    -- Handle the zero vector or zero map
    if isZero f or isZero v then return makeZero getModuleInWidth(f.targMod,
        ↪ getWidth v);

    sum for single in keyedSingles v list (
        elt := single.vec#(single.key);
        oiMap := single.key#0;
        basisIdx := single.key#1;
        modMap := getInducedModuleMap(f.targMod, oiMap);
        elt * modMap f.genImages#(basisIdx - 1)
    )
)

-- Check if a FreeOIModuleMap is a graded map
isHomogeneous FreeOIModuleMap := f -> (
    if isZero f then return true;

    for elt in f.genImages do if not isHomogeneous elt then return false;

    -- -f.srcMod.degShifts === apply(f.genImages, degree)
    for i to #f.genImages - 1 do if not (isZero(f.genImages#i) or -f.srcMod.
        ↪ degShifts#i === degree(f.genImages#i)) then return false;

    true
)
```

```
-- Compute the n-orbit of a List of VectorInWidth objects
oiOrbit = method(TypicalValue => List)
oiOrbit(List, ZZ) := (L, n) -> (
    if n < 0 then error "expected a nonnegative integer";

    unique flatten for elt in L list for oimap in getOIMaps(getWidth elt, n) list
        ↪ (getInducedModuleMap(getFreeOIModule elt, oimap)) elt
)

-- Get the Schreyer map of a FreeOIModule object, if it exists
getSchreyerMap = method(TypicalValue => FreeOIModuleMap)
getSchreyerMap FreeOIModule := F -> if not instance(F.monOrder, List) then error "
    ↪ invalid monomial order" else (
    new FreeOIModuleMap from {srcMod => F, targMod => getFreeOIModule(F.monOrder#0)
        ↪ , genImages => F.monOrder}
)

-- Division function for KeyedVectorInWidth terms
-- Args: v = KeyedVectorInWidth, w = KeyedVectorInWidth
-- Comment: tries to divide v by w and returns a HashTable of the form {quo =>
    ↪ RingElement, oiMap => OIMap}
termDiv := (v, w) -> (
    clsv := class v.vec;
    clsw := class w.vec;
    fmod := clsv.freeOIMod;

    if isZero v.vec then return hashTable {quo => 0_(getAlgebraInWidth(fmod.
        ↪ polyOIAlg, clsv.wid)), oiMap => null};

    widv := clsv.wid;
    widw := clsw.wid;
    keyv := v.key;
    keyw := w.key;

    if widv === widw then (
        if keyv === keyw and zero(v.vec#keyv % w.vec#keyw) then
            return hashTable {quo => v.vec#keyv // w.vec#keyw, oiMap => (getOIMaps(
                ↪ widw, widv))#0}
    )
    else for oiMap0 in getOIMaps(widw, widv) do (
        modMap := getInducedModuleMap(fmod, oiMap0);
        imgw := modMap w.vec;
        keyimgw := (oiMap0 w.key#0, w.key#1);

        if keyv === keyimgw and zero(v.vec#keyv % imgw#keyimgw) then
            return hashTable {quo => v.vec#keyv // imgw#keyimgw, oiMap => oiMap0}
    );

    return hashTable {quo => 0_(getAlgebraInWidth(fmod.polyOIAlg, clsv.wid)),
        ↪ oiMap => null}
)

-- Divide a VectorInWidth by a List of VectorInWidth objects
-- Args: v = VectorInWidth, L = List
```

```
-- Comment: returns a HashTable of the form {quo => VectorInWidth, rem =>
    ↪ VectorInWidth, divTuples => List}
-- Comment: expects L to consist of nonzero elements
polyDiv := (v, L) -> (
    if isZero v then return new HashTable from {quo => v, rem => v, divTuples =>
        ↪ {}};

    cls := class v;
    quo0 := makeZero cls;
    rem0 := v;

    done := false;
    divTuples0 := while not done list (
        divTuple := null;
        for i to #L - 1 do (
            elt := L#i;
            div := termDiv(keyedLeadTerm rem0, keyedLeadTerm elt);
            if zero div.quo then continue;

            modMap := getInducedModuleMap(cls.freeOIMod, div.oiMap);
            q := modMap elt;
            quo0 = quo0 + div.quo * q;
            rem0 = rem0 - div.quo * q;

            divTuple = (div, i);
            break
        );

        if divTuple === null then break;
        if isZero rem0 then done = true;

        divTuple
    );

    new HashTable from {quo => quo0, rem => rem0, divTuples => divTuples0}
)

-- Compute the normal form of a VectorInWidth modulo a List of VectorInWidth
    ↪ objects
-- Args: v = VectorInWidth, L = List
-- Comment: expects L to consist of nonzero elements
oiNormalForm := (v, L) -> (
    if isZero v then return v;

    cls := class v;
    rem := makeZero cls;

    while not isZero v do (
        divisionOccurred := false;

        for elt in L do (
            div := termDiv(keyedLeadTerm v, keyedLeadTerm elt);
            if zero div.quo then continue;
```

```
            modMap := getInducedModuleMap(cls.freeOIMod, div.oiMap);
            v = v - div.quo * modMap elt;

            divisionOccurred = true;
            break
        );

        if not divisionOccurred then (
            rem = rem + leadTerm v;
            v = v - leadTerm v
        )
    );

    rem
)

-- Compute the S-polynomial of two VectorInWidth objects
-- Args: v = VectorInWidth, w = VectorInWidth
-- Comment: expects class v === class w
SPolynomial := (v, w) -> (
    cls := class v;

    if isZero v or isZero w then return makeZero cls;

    ltv := keyedLeadTerm v;
    ltw := keyedLeadTerm w;
    ltvelt := ltv.vec#(ltv.key);
    ltwelt := ltw.vec#(ltw.key);
    lcmlmvw := lcm(leadMonomial ltvelt, leadMonomial ltwelt);

    (lcmlmvw // ltvelt) * v - (lcmlmvw // ltwelt) * w
)

-- Should be of the form {map0 => OIMap, vec0 => VectorInWidth, im0 =>
    ↪ VectorInWidth, map1 => OIMap, vec1 => VectorInWidth, im1 => VectorInWidth}
OIPair = new Type of HashTable

-- Comparison method for OIPair objects
OIPair ? OIPair := (p, q) -> getWidth p.im0 ? getWidth q.im0

-- Compute the critical pairs for a List of VectorInWidth objects
-- Args: L = List, V = Boolean
-- Comment: map0 and map1 are the OI-maps applied to vec0 and vec1 to make im0 and
    ↪  im1
oiPairs := (L, V) -> sort unique flatten flatten flatten flatten for fIdx to #L -
    ↪ 1 list (
    f := L#fIdx;
    ltf := keyedLeadTerm f;
    for gIdx from fIdx to #L - 1 list (
        g := L#gIdx;
        ltg := keyedLeadTerm g;
        clsf := class f;
        clsg := class g;
```

```
            if not ltf.key#1 === ltg.key#1 then continue; -- These will have lcm zero

            widf := clsf.wid;
            widg := clsg.wid;
            searchMin := max(widf, widg);
            searchMax := widf + widg;
            for i to searchMax - searchMin list (
                k := searchMax - i;
                oiMapsFromf := getOIMaps(widf, k);

                -- Given an OI-map from f, we construct the corresponding OI-maps from
                    ↪ g
                for oiMapFromf in oiMapsFromf list (
                    base := set(1..k) - set oiMapFromf.img; -- Get the starting set

                    -- Add back in the i-element subsets of oiMapFromf.img and make the
                        ↪ pairs
                    for subset in subsets(oiMapFromf.img, i) list (
                        oiMapFromg := makeOIMap(k, sort toList(base + set subset));

                        if not oiMapFromf ltf.key#0 === oiMapFromg ltg.key#0 then
                            ↪ continue; -- These will have lcm zero
                        if fIdx === gIdx and oiMapFromf === oiMapFromg then continue; --
                            ↪  These will yield trivial S-polynomials and syzygies

                        if V then print("Found suitable OI-maps " | net oiMapFromf | "
                            ↪ and " | net oiMapFromg);

                        modMapFromf := getInducedModuleMap(clsf.freeOIMod, oiMapFromf);
                        modMapFromg := getInducedModuleMap(clsg.freeOIMod, oiMapFromg);

                        new OIPair from {map0 => oiMapFromf, idx0 => fIdx, im0 =>
                            ↪ modMapFromf f, map1 => oiMapFromg, idx1 => gIdx, im1 =>
                            ↪ modMapFromg g}
                    )
                )
            )
        )
)

-- Cache for storing OI-Groebner bases
oiGBCache = new MutableHashTable

-- Compute an OI-Groebner basis for a List of VectorInWidth objects
oiGB = method(TypicalValue => List, Options => {Verbose => false, Strategy =>
    ↪ Minimize})
oiGB List := opts -> L -> (
    if not (opts.Strategy === FastNonminimal or opts.Strategy === Minimize or opts.
        ↪ Strategy === Reduce) then
        error "expected Strategy => FastNonminimal or Strategy => Minimize or
            ↪ Strategy => Reduce";

    if opts.Verbose then print "Computing OIGB...";
```

```
-- Return the GB if it already exists
if oiGBCache#?(L, opts.Strategy) then return oiGBCache#(L, opts.Strategy);

-- Throw out any repeated or zero elements
ret := unique for elt in L list if isZero elt then continue else elt;
if #ret === 0 then error "expected a nonempty list of nonzero elements";

encountered := new List;
totalAdded := 0;

-- Enter the main loop: terminates by an equivariant Noetherianity argument
while true do (
    oipairs := oiPairs(ret, opts.Verbose);

    remToAdd := null;
    for i to #oipairs - 1 do (
        s := SPolynomial((oipairs#i).im0, (oipairs#i).im1);
        if isZero s then continue;

        if member(s, encountered) then continue -- Skip S-Polynomials that have
            ↪  already appeared
        else encountered = append(encountered, s);

        if opts.Verbose then print("On critical pair " | toString(i + 1) | "
            ↪ out of " | toString(#oipairs));

        rem := (polyDiv(s, ret)).rem;
        if not isZero rem and not member(rem, ret) then (
            if opts.Verbose then (
                print("Found nonzero remainder: " | net rem);
                totalAdded = totalAdded + 1;
                print("Elements added total: " | net totalAdded);
            );

            remToAdd = rem;
            break
        )
    );

    if remToAdd === null then break;
    ret = append(ret, remToAdd)
);

-- Minimize the basis
if opts.Strategy === Minimize then (
    if opts.Verbose then print "\n----------------------------------------\n
        ↪ ----------------------------------------\n";
    ret = minimizeOIGB(ret, Verbose => opts.Verbose)
);

-- Reduce the basis
if opts.Strategy === Reduce then (
    if opts.Verbose then print "\n----------------------------------------\n
        ↪ ----------------------------------------\n";
```

```
        ret = reduceOIGB(ret, Verbose => opts.Verbose)
    );

    -- Store the GB
    oiGBCache#(L, opts.Strategy) = ret
)

-- Minimize an OI-Groebner basis in the sense of monic and lt(p) not in <lt(G - {p
    ↪ })> for all p in G
minimizeOIGB = method(TypicalValue => List, Options => {Verbose => false})
minimizeOIGB List := opts -> G -> (
    if opts.Verbose then print "Computing minimal OIGB...";

    -- Throw out any repeated or zero elements
    G = unique for elt in G list if isZero elt then continue else elt;
    if #G === 0 then error "expected a nonempty list of nonzero elements";

    nonRedundant := new List;
    currentBasis := unique apply(G, makeMonic); -- unique is used again because
        ↪ collisions may happen after makeMonic

    if #currentBasis === 1 then return currentBasis;

    while true do (
        redundantFound := false;

        for p in currentBasis do (
            if member(p, nonRedundant) then continue; -- Skip elements already
                ↪ verified to be nonredundant

            minusp := toList((set currentBasis) - set {p});
            ltp := keyedLeadTerm p;
            for elt in minusp do if not zero (termDiv(ltp, keyedLeadTerm elt)).quo
                ↪ then (
                if opts.Verbose then print("Found redundant element: " | net p);
                redundantFound = true;
                currentBasis = minusp;
                break
            );

            if redundantFound then break;
            nonRedundant = append(nonRedundant, p);
        );

        if not redundantFound then break
    );

    currentBasis
)

-- Remove a single element from a List
-- Args: L = List, i = ZZ
sdrop := (L, i) -> drop(L, {i, i})
```

```
-- Reduce an OI-Groebner basis in the sense of monic and no term of any element is OI-divisible by the lead term of any other
reduceOIGB = method(TypicalValue => List, Options => {Verbose => false})
reduceOIGB List := opts -> G -> (
    minG := minimizeOIGB(G, Verbose => opts.Verbose);

    if opts.Verbose then print "\n----------------------------------------\n----------------------------------------\n\nComputing reduced OIGB...";

    if #minG === 1 then return minG;

    -- Reduce the basis
    newG := new MutableList from minG;
    for i to #newG - 1 do (
        if opts.Verbose then print("Reducing element " | toString(i + 1) | " of " | toString(#newG));
        dropped := flatten sdrop(toList newG, i);
        red := oiNormalForm(newG#i, dropped);
        newG#i = if member(red, dropped) then {} else red
    );

    flatten toList newG
)

-- Check if a List is an OI-Groebner basis
isOIGB = method(TypicalValue => Boolean, Options => {Verbose => false})
isOIGB List := opts -> L -> (
    if opts.Verbose then print "Checking Buchberger's Criterion...";

    -- Throw out any repeated or zero elements
    L = unique for elt in L list if isZero elt then continue else elt;
    if #L === 0 then error "expected a nonempty list of nonzero elements";

    encountered := new List;
    oipairs := oiPairs(L, opts.Verbose);
    for i to #oipairs - 1 do (
        s := SPolynomial((oipairs#i).im0, (oipairs#i).im1);
        if isZero s then continue;

        if member(s, encountered) then continue -- Skip S-Polynomials that have already appeared
        else encountered = append(encountered, s);

        if opts.Verbose then print("On critical pair " | toString(i + 1) | " out of " | toString(#oipairs));

        rem := (polyDiv(s, L)).rem;
        if not isZero rem then (
            if opts.Verbose then print("Found nonzero remainder: " | net rem);
            return false
        )
    );

    true
```

```
)

-- Cache for storing Groebner bases computed with oiSyz
oiSyzCache = new MutableHashTable

-- Compute an OI-Groebner basis for the syzygy module of a List of VectorInWidth
    ↪ objects
oiSyz = method(TypicalValue => List, Options => {Verbose => false, Strategy =>
    ↪ Minimize})
oiSyz(List, Symbol) := opts -> (L, d) -> (
    if not (opts.Strategy === FastNonminimal or opts.Strategy === Minimize or opts.
        ↪ Strategy === Reduce) then
        error "expected Strategy => FastNonminimal or Strategy => Minimize or
            ↪ Strategy => Reduce";

    if opts.Verbose then print "Computing syzygies...";

    -- Return the GB if it already exists
    if oiSyzCache#?(L, d, opts.Strategy) then return oiSyzCache#(L, d, opts.
        ↪ Strategy);

    -- Throw out any repeated or zero elements
    L = unique for elt in L list if isZero elt then continue else elt;
    if #L === 0 then error "expected a nonempty list of nonzero elements";

    fmod := getFreeOIModule L#0;
    shifts := for elt in L list -degree elt;
    widths := for elt in L list getWidth elt;
    G := makeFreeOIModule(d, widths, fmod.polyOIAlg, DegreeShifts => shifts,
        ↪ OIMonomialOrder => L);

    oipairs := oiPairs(L, opts.Verbose);
    if opts.Verbose then print "Iterating through critical pairs...";
    i := 0;
    ret := for pair in oipairs list (
        if opts.Verbose then (
            print("On critical pair " | toString(i + 1) | " out of " | toString(#
                ↪ oipairs));
            print("Pair: (" | net pair.im0 | ", " | net pair.im1 | ")");
            i = i + 1
        );

        ltf := keyedLeadTerm pair.im0;
        ltg := keyedLeadTerm pair.im1;
        ltfelt := ltf.vec#(ltf.key);
        ltgelt := ltg.vec#(ltg.key);
        lcmlmfg := lcm(leadMonomial ltfelt, leadMonomial ltgelt);
        s := SPolynomial(pair.im0, pair.im1);
        M := getModuleInWidth(G, getWidth s);
        thingToSubtract := makeZero M;

        -- Calculate the stuff to subtract off
        if not isZero s then for tuple in (polyDiv(s, L)).divTuples do
```

```
            thingToSubtract = thingToSubtract + makeSingle(M, ((tuple#0).oiMap, 1 +
                ↪  tuple#1), (tuple#0).quo);

        -- Make the syzygy
        sing1 := makeSingle(M, (pair.map0, 1 + pair.idx0), lcmlmfg // ltfelt);
        sing2 := makeSingle(M, (pair.map1, 1 + pair.idx1), lcmlmfg // ltgelt);
        syzygy := sing1 - sing2 - thingToSubtract;

        if opts.Verbose then print("Generated syzygy: " | net syzygy);

        syzygy
    );

    -- Throw out any repeated or zero elements
    ret = unique for elt in ret list if isZero elt then continue else elt;

    -- Minimize the basis
    if #ret > 0 and opts.Strategy === Minimize then (
        if opts.Verbose then print "\n----------------------------------------\n
            ↪ ----------------------------------------\n";
        ret = minimizeOIGB(ret, Verbose => opts.Verbose)
    );

    -- Reduce the basis
    if #ret > 0 and opts.Strategy === Reduce then (
        if opts.Verbose then print "\n----------------------------------------\n
            ↪ ----------------------------------------\n";
        ret = reduceOIGB(ret, Verbose => opts.Verbose)
    );

    -- Store the GB
    oiSyzCache#(L, d, opts.Strategy) = ret
)

-- Cache for storing OI-resolutions
oiResCache = new MutableHashTable

-- Should be of the form {dd => List, modules => List}
OIResolution = new Type of HashTable

net OIResolution := C -> (
    N := "0: " | toString C.modules#0;
    for i from 1 to #C.modules - 1 do N = N || toString i | ": " | toString C.
        ↪ modules#i;
    N
)

describe OIResolution := C -> (
    N := "0: Module: " | net C.modules#0 || "Differential: " | net C.dd#0;
    for i from 1 to #C.modules - 1 do N = N || toString i | ": Module: " | net C.
        ↪ modules#i || "Differential: " | net C.dd#i;
    N
)
```

```
describeFull = method(TypicalValue => Net)
describeFull OIResolution := C -> (
    N := "0: Module: " | net C.modules#0 || "Differential: " | net C.dd#0;
    for i from 1 to #C.modules - 1 do N = N || toString i | ": Module: " | net C.
        ↪ modules#i || "Differential: " | describe C.dd#i;
    N
)

ranks = method(TypicalValue => Net)
ranks OIResolution := C -> (
    N := "0: rank " | toString getRank C.modules#0;
    for i from 1 to #C.modules - 1 do N = N || toString i | ": rank " | toString
        ↪ getRank C.modules#i;
    N
)

restrictedRanks = method(TypicalValue => Net)
restrictedRanks(OIResolution, ZZ) := (C, w) -> (
    N := "0: rank " | #degrees ((C_0)_w).rawMod;
    for i from 1 to #C.modules - 1 do N = N || toString i | ": rank " | #degrees
        ↪ ((C_i)_w).rawMod;
    N
)

OIResolution _ ZZ := (C, n) -> C.modules#n

-- Compute an OI-resolution of length n for the OI-module generated by L
oiRes = method(TypicalValue => OIResolution, Options => {Verbose => false,
    ↪ Strategy => Minimize, TopNonminimal => false})
oiRes(List, ZZ) := opts -> (L, n) -> (
    if not (opts.Verbose === true or opts.Verbose === false) then error "expected
        ↪ Verbose => true or Verbose => false";
    if not (opts.TopNonminimal === true or opts.TopNonminimal === false) then
        ↪ error "expected TopNonminimal => true or TopNonminimal => false";
    if not (opts.Strategy === FastNonminimal or opts.Strategy === Minimize or opts.
        ↪ Strategy === Reduce) then
        error "expected Strategy => FastNonminimal or Strategy => Minimize or
            ↪ Strategy => Reduce";

    if n < 0 then error "expected a nonnegative integer";

    if opts.Verbose then print "Computing OI-resolution";

    -- Return the resolution if it already exists
    if oiResCache#?(L, n, opts.Strategy, opts.TopNonminimal) then return
        ↪ oiResCache#(L, n, opts.Strategy, opts.TopNonminimal);

    strat := opts.Strategy;
    if n === 0 and opts.TopNonminimal then strat = FastNonminimal;
    oigb := oiGB(L, Verbose => opts.Verbose, Strategy => strat);
    currentGB := oigb;

    ddMut := new MutableList;
    modulesMut := new MutableList;
```

```
groundFreeOIMod := getFreeOIModule currentGB#0;
e := groundFreeOIMod.basisSym;
currentSymbol := getSymbol concatenate(e, "0");
count := 0;

if n > 0 then for i to n - 1 do (
       if opts.Verbose then print "\n----------------------------------------\
           ↪ n----------------------------------------\n";

       if i === n - 1 and opts.TopNonminimal then strat = FastNonminimal;
       syzGens := oiSyz(currentGB, currentSymbol, Verbose => opts.Verbose,
           ↪ Strategy => strat);

       if #syzGens === 0 then break;
       count = count + 1;

       targFreeOIMod := getFreeOIModule currentGB#0;
       srcFreeOIMod := getFreeOIModule syzGens#0;

       modulesMut#i = srcFreeOIMod;
       ddMut#i = new FreeOIModuleMap from {srcMod => srcFreeOIMod, targMod =>
           ↪ targFreeOIMod, genImages => currentGB};

       currentGB = syzGens;
       currentSymbol = getSymbol concatenate(e, toString count)
);

-- Append the last term in the sequence
shifts := for elt in currentGB list -degree elt;
widths := for elt in currentGB list getWidth elt;
modulesMut#count = makeFreeOIModule(currentSymbol, widths, groundFreeOIMod.
    ↪ polyOIAlg, DegreeShifts => shifts, OIMonomialOrder => currentGB);
ddMut#count = new FreeOIModuleMap from {srcMod => modulesMut#count, targMod =>
    ↪  if count === 0 then groundFreeOIMod else modulesMut#(count - 1),
    ↪ genImages => currentGB};

-- Cap the sequence with zeros
for i from count + 1 to n do (
    currentSymbol = getSymbol concatenate(e, toString i);
    modulesMut#i = makeFreeOIModule(currentSymbol, {}, groundFreeOIMod.
        ↪ polyOIAlg);
    ddMut#i = new FreeOIModuleMap from {srcMod => modulesMut#i, targMod =>
        ↪ modulesMut#(i - 1), genImages => {}}
);

-- Minimize the resolution
if #ddMut > 1 and not isZero ddMut#1 then (
    if opts.Verbose then print "\n----------------------------------------\n
        ↪ ----------------------------------------\n\nMinimizing resolution
        ↪ ...";

    done := false;
    while not done do (
        done = true;
```

```
-- Look for units on identity basis elements
unitFound := false;
local data;
for i from 1 to #ddMut - 1 do (
    ddMap := ddMut#i;
    if isZero ddMap then continue;

    srcFreeOIMod := ddMap.srcMod;
    targFreeOIMod := ddMap.targMod;
    for j to #ddMap.genImages - 1 do (
        if isZero ddMap.genImages#j then continue;

        for single in keyedSingles ddMap.genImages#j do if (single.key
            ↪ #0).img === toList(1..(single.key#0).targWidth) and
            ↪ isUnit single.vec#(single.key) then (
            unitFound = true;
            done = false;
            data = {i, j, single};
            if opts.Verbose then print("Unit found on term: " | net
                ↪ single.vec);
            break
        );

        if unitFound then break
    );

    if unitFound then break
);

-- Prune the sequence
if unitFound then (
    if opts.Verbose then print "Pruning...";

    unitSingle := data#2;
    targBasisPos := unitSingle.key#1 - 1;
    srcBasisPos := data#1;
    ddMap := ddMut#(data#0);
    srcFreeOIMod := ddMap.srcMod;
    targFreeOIMod := ddMap.targMod;

    -- Make the new free OI-modules
    newSrcWidths := sdrop(srcFreeOIMod.genWidths, srcBasisPos);
    newSrcShifts := sdrop(srcFreeOIMod.degShifts, srcBasisPos);
    newTargWidths := sdrop(targFreeOIMod.genWidths, targBasisPos);
    newTargShifts := sdrop(targFreeOIMod.degShifts, targBasisPos);
    newSrcFreeOIMod := makeFreeOIModule(srcFreeOIMod.basisSym,
        ↪ newSrcWidths, srcFreeOIMod.polyOIAlg, DegreeShifts =>
        ↪ newSrcShifts);
    newTargFreeOIMod := makeFreeOIModule(targFreeOIMod.basisSym,
        ↪ newTargWidths, targFreeOIMod.polyOIAlg, DegreeShifts =>
        ↪ newTargShifts);

    -- Compute the new differential
```

```
newGenImages := for i to #srcFreeOIMod.genWidths - 1 list (
    if i === srcBasisPos then continue;

    -- Calculate the stuff to subtract off
    thingToSubtract := makeZero getModuleInWidth(srcFreeOIMod,
        ↪ srcFreeOIMod.genWidths#i);
    for single in keyedSingles ddMap.genImages#i do (
        if not single.key#1 === targBasisPos + 1 then continue;

        modMap := getInducedModuleMap(srcFreeOIMod, single.key#0);
        basisElt := getBasisElement(srcFreeOIMod, srcBasisPos);
        thingToSubtract = thingToSubtract + single.vec#(single.key) *
            ↪  modMap basisElt
    );

    -- Calculate the new image
    basisElt := getBasisElement(srcFreeOIMod, i);
    newGenImage0 := ddMap(basisElt - lift(1 // unitSingle.vec#(
        ↪ unitSingle.key), srcFreeOIMod.polyOIAlg.baseField) *
        ↪ thingToSubtract);
    M := getModuleInWidth(newTargFreeOIMod, getWidth newGenImage0);
    newGenImage := makeZero M;
    for newSingle in keyedSingles newGenImage0 do (
        idx := newSingle.key#1;
        if idx > targBasisPos + 1 then idx = idx - 1; -- Relabel
        newGenImage = newGenImage + makeSingle(M, (newSingle.key#0,
            ↪ idx), newSingle.vec#(newSingle.key))
    );

    newGenImage
);

ddMut#(data#0) = new FreeOIModuleMap from {srcMod => newSrcFreeOIMod
    ↪ , targMod => newTargFreeOIMod, genImages => newGenImages};
modulesMut#(data#0) = newSrcFreeOIMod;
modulesMut#(data#0 - 1) = newTargFreeOIMod;

-- Adjust the map to the right
ddMap = ddMut#(data#0 - 1);
ddMut#(data#0 - 1) = new FreeOIModuleMap from {srcMod =>
    ↪ newTargFreeOIMod, targMod => ddMap.targMod, genImages =>
    ↪ sdrop(ddMap.genImages, targBasisPos)}; -- Restriction

-- Adjust the map to the left
if data#0 < #ddMut - 1 then (
    ddMap = ddMut#(data#0 + 1);
    newGenImages = new MutableList;

    for i to #ddMap.genImages - 1 do (
        M := getModuleInWidth(newSrcFreeOIMod, getWidth ddMap.
            ↪ genImages#i);
        newGenImage := makeZero M;
        for single in keyedSingles ddMap.genImages#i do (
            idx := single.key#1;
```

```
                    if idx === srcBasisPos + 1 then continue; -- Projection
                    if idx > srcBasisPos + 1 then idx = idx - 1; -- Relabel
                    newGenImage = newGenImage + makeSingle(M, (single.key#0,
                        ↪ idx), single.vec#(single.key))
                );

                newGenImages#i = newGenImage
            );

            ddMut#(data#0 + 1) = new FreeOIModuleMap from {srcMod => ddMap.
                ↪ srcMod, targMod => newSrcFreeOIMod, genImages => new List
                ↪  from newGenImages}
        )
    )
  )
);

-- Store the resolution
oiResCache#(L, n, opts.Strategy, opts.TopNonminimal) = new OIResolution from {
    ↪ dd => new List from ddMut, modules => new List from modulesMut}
)

-- Verify that an OIResolution is a complex
isComplex = method(TypicalValue => Boolean, Options => {Verbose => false})
isComplex OIResolution := opts -> C -> (
    if #C.dd < 2 then error "expected a sequence with at least two maps";

    -- Check if the maps compose to zero
    for i from 1 to #C.dd - 1 do (
        modMap0 := C.dd#(i - 1);
        modMap1 := C.dd#i;
        if isZero modMap0 or isZero modMap1 then continue;

        for basisElt in getBasisElements modMap1.srcMod do (
            result := modMap0 modMap1 basisElt;

            if opts.Verbose then print(net basisElt | " maps to " | net result);

            if not isZero result then (
                if opts.Verbose then print("Found nonzero image: " | net result);
                return false
            )
        )
    );

    true
)

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-- DOCUMENTATION ---------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```

```
beginDocumentation()

doc ///
    Key
        OIGroebnerBases
    Headline
        OI-modules over Noetherian polynomial OI-algebras
    Description
        Text
            {\em OIGroebnerBases} is a package for Gröbner bases, syzygies and free
                ↪  resolutions for submodules of free OI-modules over Noetherian
                ↪ polynomial OI-algebras. For an introduction to the theory of OI-
                ↪ modules, see [3].

            Given a Noetherian polynomial OI-algebra $\mathbf{P} := (\mathbf{X}^{\
                ↪ text{OI},1})^{\otimes c}$ for some integer $c > 0$, one can
                ↪ consider free OI-modules $\mathbf{F} := \bigoplus_{i=1}^s\mathbf
                ↪ {F}^{\text{OI}, d_i}$ over $\mathbf{P}$ for integers $d_i\geq 0$.
                ↪

            Gröbner bases for submodules of $\mathbf{F}$ were introduced in [3].
                ↪ Free resolutions and homological aspects of submodules have been
                ↪  studied in [2,3]. Using the methods of [1], Gröbner bases,
                ↪ syzygy modules, and free resolutions for submodules can be
                ↪ computed with @TO oiGB@, @TO oiSyz@ and @TO oiRes@ respectively.

            {\em References:}

            [1] M. Morrow and U. Nagel, {\it Computing Gröbner Bases and Free
                ↪ Resolutions of OI-Modules}, Preprint, arXiv:2303.06725, 2023.

            [2] N. Fieldsteel and U. Nagel, {\it Minimal and cellular free
                ↪ resolutions over polynomial OI-algebras}, Preprint, arXiv
                ↪ :2105.08603, 2021.

            [3] U. Nagel and T. Römer, {\it FI- and OI-modules with varying
                ↪ coefficients}, J. Algebra 535 (2019), 286-322.
    Subnodes
        :Polynomial OI-algebras
        PolynomialOIAlgebra
        makePolynomialOIAlgebra
        VariableOrder
        ColUpRowUp
        ColUpRowDown
        ColDownRowUp
        ColDownRowDown
        RowUpColUp
        RowUpColDown
        RowDownColUp
        RowDownColDown
        (net,PolynomialOIAlgebra)
        (toString,PolynomialOIAlgebra)
        (symbol _,PolynomialOIAlgebra,ZZ)
        :Free OI-modules
```

```
FreeOIModule
FreeOIModuleMap
ModuleInWidth
VectorInWidth
makeFreeOIModule
DegreeShifts
OIMonomialOrder
installGeneratorsInWidth
isZero
getBasisElements
getFreeOIModule
getSchreyerMap
getWidth
getRank
oiOrbit
(isZero,FreeOIModuleMap)
(isZero,VectorInWidth)
(symbol SPACE,FreeOIModuleMap,VectorInWidth)
(net,FreeOIModule)
(net,FreeOIModuleMap)
(net,ModuleInWidth)
(net,VectorInWidth)
(image,FreeOIModuleMap)
(toString,FreeOIModule)
(symbol _,FreeOIModule,ZZ)
(degree,VectorInWidth)
(use,ModuleInWidth)
(terms,VectorInWidth)
(leadTerm,VectorInWidth)
(leadMonomial,VectorInWidth)
(leadCoefficient,VectorInWidth)
(symbol *,Number,VectorInWidth)
(symbol +,VectorInWidth,VectorInWidth)
(symbol *,RingElement,VectorInWidth)
(symbol -,VectorInWidth)
(symbol -,VectorInWidth,VectorInWidth)
(describe,FreeOIModuleMap)
(isHomogeneous,FreeOIModuleMap)
(isHomogeneous,VectorInWidth)
:OI-Gröbner bases
oiGB
minimizeOIGB
reduceOIGB
isOIGB
:OI-syzygies
oiSyz
:OI-resolutions
OIResolution
oiRes
ranks
restrictedRanks
TopNonminimal
isComplex
describeFull
```

```
        (describe,OIResolution)
        (net,OIResolution)
        (symbol _,OIResolution,ZZ)
///

doc ///
    Key
        PolynomialOIAlgebra
    Headline
        the class of all Noetherian polynomial OI-algebras over a field
    Description
        Text
            This type implements OI-algebras of the form $(\mathbf{X}^{\text{OI
                ↪ },1})^{\otimes c}$ for some integer $c>0$. To make a @TT "
                ↪ PolynomialOIAlgebra"@ object, use @TO makePolynomialOIAlgebra@.
                ↪ Each @TT "PolynomialOIAlgebra"@ object is equipped with a
                ↪ variable order; see @TO VariableOrder@.
        Example
            P = makePolynomialOIAlgebra(2, x, QQ, VariableOrder => RowDownColUp)
///

doc ///
    Key
        makePolynomialOIAlgebra
        (makePolynomialOIAlgebra,ZZ,Symbol,Ring)
        [makePolynomialOIAlgebra,VariableOrder]
    Headline
        make a PolynomialOIAlgebra object
    Usage
        makePolynomialOIAlgebra(c,x,K)
    Inputs
        c:ZZ
        x:Symbol
        K:Ring
    Outputs
        :PolynomialOIAlgebra
    Description
        Text
            Makes a polynomial OI-algebra over the field @TT "K"@ with @TT "c"@
                ↪ rows of variables in the symbol @TT "x"@. The @TO VariableOrder@
                ↪  option is used to specify the ordering of the variables.
        Example
            P = makePolynomialOIAlgebra(1, x, QQ)
            Q = makePolynomialOIAlgebra(2, y, QQ, VariableOrder => RowDownColUp)
///

doc ///
    Key
        VariableOrder
    Headline
        variable ordering for polynomial OI-algebras
    Description
        Text
```

```
            Used as an optional argument in @TO makePolynomialOIAlgebra@ to specify
                ↪  an ordering on the variables of a polynomial OI-algebra.

            Permissible values:
        Code
            UL {
                {TT "VariableOrder => ", TT TO ColUpRowUp},
                {TT "VariableOrder => ", TT TO ColUpRowDown},
                {TT "VariableOrder => ", TT TO ColDownRowUp},
                {TT "VariableOrder => ", TT TO ColDownRowDown},
                {TT "VariableOrder => ", TT TO RowUpColUp},
                {TT "VariableOrder => ", TT TO RowUpColDown},
                {TT "VariableOrder => ", TT TO RowDownColUp},
                {TT "VariableOrder => ", TT TO RowDownColDown}
            }
        Example
            P = makePolynomialOIAlgebra(2, x, QQ, VariableOrder => RowDownColUp)
///

doc ///
    Key
        ColUpRowUp
    Headline
        column up row up variable order
    Description
        Text
            Orders the variables of a polynomial OI-algebra according to $x_{i',j'}
                ↪  < x_{i,j}$ if $j'<j$ or $j'=j$ and $i'<i$.
///

doc ///
    Key
        ColUpRowDown
    Headline
        column up row down variable order
    Description
        Text
            Orders the variables of a polynomial OI-algebra according to $x_{i',j'}
                ↪  < x_{i,j}$ if $j'<j$ or $j'=j$ and $i'>i$.
///

doc ///
    Key
        ColDownRowUp
    Headline
        column down row up variable order
    Description
        Text
            Orders the variables of a polynomial OI-algebra according to $x_{i',j'}
                ↪  < x_{i,j}$ if $j'>j$ or $j'=j$ and $i'<i$.
///

doc ///
    Key
```

```
        ColDownRowDown
    Headline
        column down row down variable order
    Description
        Text
            Orders the variables of a polynomial OI-algebra according to $x_{i',j'}
                ↪  < x_{i,j}$ if $j'>j$ or $j'=j$ and $i'>i$.
///

doc ///
    Key
        RowUpColUp
    Headline
        row up column up variable order
    Description
        Text
            Orders the variables of a polynomial OI-algebra according to $x_{i',j'}
                ↪  < x_{i,j}$ if $i'<i$ or $i'=i$ and $j'<j$.
///

doc ///
    Key
        RowUpColDown
    Headline
        row up column down variable order
    Description
        Text
            Orders the variables of a polynomial OI-algebra according to $x_{i',j'}
                ↪  < x_{i,j}$ if $i'<i$ or $i'=i$ and $j'>j$.
///

doc ///
    Key
        RowDownColUp
    Headline
        row up column up variable order
    Description
        Text
            Orders the variables of a polynomial OI-algebra according to $x_{i',j'}
                ↪  < x_{i,j}$ if $i'>i$ or $i'=i$ and $j'<j$.
///

doc ///
    Key
        RowDownColDown
    Headline
        row down column down variable order
    Description
        Text
            Orders the variables of a polynomial OI-algebra according to $x_{i',j'}
                ↪  < x_{i,j}$ if $i'>i$ or $i'=i$ and $j'>j$.
///

doc ///
```

```
    Key
        (net,PolynomialOIAlgebra)
    Headline
        display a polynomial OI-algebra
    Usage
        net P
    Inputs
        P:PolynomialOIAlgebra
    Outputs
        :Net
    Description
        Text
            Displays the base field, number of variable rows, variable symbol, and
                ↪ variable order of a @TO PolynomialOIAlgebra@ object.
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
            net P
///

doc ///
    Key
        (toString,PolynomialOIAlgebra)
    Headline
        display a polynomial OI-algebra in condensed form
    Usage
        toString P
    Inputs
        P:PolynomialOIAlgebra
    Outputs
        :String
    Description
        Text
            Displays the base field, number of variable rows, variable symbol, and
                ↪ variable order of a @TO PolynomialOIAlgebra@ object as a string.
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
            toString P
///

doc ///
    Key
        (symbol _,PolynomialOIAlgebra,ZZ)
    Headline
        get the $K$-algebra in a specified width of a polynomial OI-algebra
    Usage
        P_n
    Inputs
        P:PolynomialOIAlgebra
        n:ZZ
    Outputs
        :PolynomialRing
    Description
        Text
            Returns the $K$-algebra in width @TT "n"@ of a polynomial OI-algebra.
```

```
        Example
            P = makePolynomialOIAlgebra(2, y, QQ);
            P_4
///

doc ///
    Key
        FreeOIModule
    Headline
        the class of all free OI-modules over a polynomial OI-algebra
    Description
        Text
            This type implements free OI-modules over polynomial OI-algebras. To
                ↪ make a @TT "FreeOIModule"@ object, use @TO makeFreeOIModule@. To
                ↪  get the basis elements and rank of a free OI-module, use @TO
                ↪ getBasisElements@ and @TO getRank@ respectively. To install the
                ↪ generators of a component of a free OI-module in a specified
                ↪ width, use @TO installGeneratorsInWidth@.

            Each @TT "FreeOIModule"@ object comes equipped with either the @TO Lex@
                ↪  monomial order induced by the monomial order on its underlying
                ↪ polynomial OI-algebra, or the Schreyer monomial order induced by
                ↪  another free OI-module; see @TO makeFreeOIModule@ and @TO
                ↪ OIMonomialOrder@.
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
            F = makeFreeOIModule(e, {1,2}, P)
///

doc ///
    Key
        FreeOIModuleMap
    Headline
        the class of all maps between free OI-modules
    Description
        Text
            This type implements morphisms between free OI-modules. Given free OI-
                ↪ modules $\mathbf{F}$ and $\mathbf{G}$ over an OI-algebra $\
                ↪ mathbf{P}$, a $\mathbf{P}$-linear map $\varphi:\mathbf{F}\to\
                ↪ mathbf{G}$ is determined by the images of the basis elements of
                ↪ $\mathbf{F}$.

            To evaluate $\varphi$ on an element of $\mathbf{F}$, use @TO (symbol
                ↪ SPACE,FreeOIModuleMap,VectorInWidth)@.

            One obtains @TT "FreeOIModuleMap"@ objects through the use of @TO
                ↪ oiRes@, as in the below example.
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
            F = makeFreeOIModule(e, {1,2}, P);
            installGeneratorsInWidth(F, 3);
            b = x_(1,2)*x_(1,1)*e_(3,{2},1)+x_(2,2)*x_(2,1)*e_(3,{1,3},2);
            C = oiRes({b}, 2)
            phi = C.dd_1
```

```
            G = getBasisElements C_1
            phi G#0
            phi G#1
///

doc ///
    Key
        ModuleInWidth
    Headline
        the class of all modules that appear as a component of a free OI-module
    Description
        Text
            The width $n$ component of a free OI-module is implemented as a @TT "
                ↪ ModuleInWidth"@ object. To obtain a @TT "ModuleInWidth"@ object,
                ↪  one restricts a given free OI-module to a specified width using
                ↪  @TO (symbol _,FreeOIModule,ZZ)@, as seen in the example below.
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
            F = makeFreeOIModule(e, {1,2}, P);
            F_2
///

doc ///
    Key
        VectorInWidth
    Headline
        the class of all elements of a free OI-module
    Description
        Text
            An element of a free OI-module $\mathbf{F}$ is defined to be an element
                ↪  of $\mathbf{F}_n$ for some integer $n\geq0$. Such an element is
                ↪  implemented as a @TT "VectorInWidth"@ object. One typically
                ↪ makes @TT "VectorInWidth"@ objects by defining a @TO
                ↪ FreeOIModule@ object, calling @TO installGeneratorsInWidth@, and
                ↪  then manipulating the generators; see below for an example.
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
            F = makeFreeOIModule(e, {1,1}, P);
            installGeneratorsInWidth(F, 2);
            b = x_(1,1)*e_(2,{1},1)+x_(2,1)*e_(2,{1},2)
            instance(b, VectorInWidth)
///

doc ///
    Key
        makeFreeOIModule
        (makeFreeOIModule,Symbol,List,PolynomialOIAlgebra)
        [makeFreeOIModule,DegreeShifts]
        [makeFreeOIModule,OIMonomialOrder]
    Headline
        make a FreeOIModule object
    Usage
        makeFreeOIModule(e,L,P)
    Inputs
```

```
        e:Symbol
        L:List
        P:PolynomialOIAlgebra
    Outputs
        :FreeOIModule
    Description
        Text
            Makes a free OI-module of the form $\bigoplus_{i=1}^s\mathbf{F}^{\text{
                ↪ OI}, d_i}$ over the @TO PolynomialOIAlgebra@ object @TT "P"@
                ↪ with $L=\{d_1,\ldots,d_s\}$ and basis symbol @TT "e"@.

            The @TO DegreeShifts@ option is used to specify a shift of grading.
                ↪ This option must be set to either @TT "null"@ for no shifts, or
                ↪ a list of integers describing the desired shifts.

            The @TO OIMonomialOrder@ option must be set to either @TO Lex@, for the
                ↪  lexicographic order, or a list of elements of some free OI-
                ↪ module for the Schreyer order. See below for examples.

            @HEADER2 "Example 1: Lex"@
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
            F = makeFreeOIModule(e, {1,1,2}, P, DegreeShifts => {3,2,1})
        Text
            @HEADER2 "Example 2: Schreyer"@
        Example
            F = makeFreeOIModule(e, {1,1}, P);
            installGeneratorsInWidth(F, 2);
            b = x_(1,1)*e_(2,{1},1)+x_(2,1)*e_(2,{1},2);
            G = makeFreeOIModule(d, {2}, P, DegreeShifts => {-degree b},
                ↪ OIMonomialOrder => {b})
///

doc ///
    Key
        DegreeShifts
    Headline
        grading shifts for free OI-modules
    Description
        Text
            Used as an optional argument in @TO makeFreeOIModule@ to specify shifts
                ↪  of grading.

            Permissible values:
        Code
            UL {
                {TT "DegreeShifts => ", TT "null", " for no shift of grading"},
                {TT "DegreeShifts => ", TT TO List, " for a shift of grading
                    ↪ indicated by a list of integers"}
            }
        Text
            @HEADER2 "Example 1: no shifts"@
```

```
            The free OI-module $\mathbf{F}^{\text{OI},1}\oplus\mathbf{F}^{\text{OI
                ↪ },2}$ has its basis elements in degree zero.
        Example
            P = makePolynomialOIAlgebra(1, x, QQ);
            F = makeFreeOIModule(e, {1,2}, P)
            apply(getBasisElements F, degree)
        Text
            @HEADER2 "Example 2: nontrivial shifts"@

            The free OI-module $\mathbf{F}^{\text{OI},1}(-3)\oplus\mathbf{F}^{\text
                ↪ {OI},2}(-4)$ has its basis elements in degrees $3$ and $4$.
        Example
            F = makeFreeOIModule(e, {1,2}, P, DegreeShifts => {-3,-4})
            apply(getBasisElements F, degree)
///

doc ///
    Key
        OIMonomialOrder
    Headline
        monomial order option for free OI-modules
    Description
        Text
            Used as an optional argument in @TO makeFreeOIModule@ to specify the
                ↪ desired monomial order.

            Permissible values:
        Code
            UL {
                {TT "OIMonomialOrder => ", TT TO Lex, " for the lexicographic order
                    ↪ induced by the monomial order of the underlying polynomial
                    ↪ OI-algebra; see [1, Example 3.2]"},
                {TT "OIMonomialOrder => ", TT TO List, " for the Schreyer order
                    ↪ induced by a list of elements of a free OI-module; see [1,
                    ↪ Definition 4.2]"}
            }
        Text
            {\em References:}

            [1] M. Morrow and U. Nagel, {\it Computing Gröbner Bases and Free
                ↪ Resolutions of OI-Modules}, Preprint, arXiv:2303.06725, 2023.
///

doc ///
    Key
        installGeneratorsInWidth
        (installGeneratorsInWidth,FreeOIModule,ZZ)
    Headline
        install the generators for a component of a free OI-module in a specified
            ↪ width
    Usage
        installGeneratorsInWidth(F, n)
    Inputs
        F:FreeOIModule
```

```
        n:ZZ
    Description
        Text
            This method assigns the generators of the width @TT "n"@ component of
                ↪ @TT "F"@ to the appropriate @TO IndexedVariable@ corresponding
                ↪ to the basis symbol of @TT "F"@.
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
            F = makeFreeOIModule(e, {1,1}, P);
            installGeneratorsInWidth(F, 2);
            e_(2,{1},1)
            e_(2,{1},2)
            e_(2,{2},1)
            e_(2,{2},2)
            x_(1,1)*e_(2,{1},1)+x_(2,1)*e_(2,{1},2)-x_(1,2)*e_(2,{2},1)-x_(2,2)*e_
                ↪ (2,{2},2)
///

doc ///
    Key
        getBasisElements
        (getBasisElements,FreeOIModule)
    Headline
        get the basis elements of a free OI-module
    Usage
        getBasisElements F
    Inputs
        F:FreeOIModule
    Outputs
        :List
    Description
        Text
            Returns the basis elements of a free OI-module.
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
            F = makeFreeOIModule(e, {1,1}, P);
            getBasisElements F
///

doc ///
    Key
        getFreeOIModule
        (getFreeOIModule,VectorInWidth)
    Headline
        get the free OI-module of an element
    Usage
        getFreeOIModule f
    Inputs
        f:VectorInWidth
    Outputs
        :FreeOIModule
    Description
        Text
            Returns the free OI-module in which the element @TT "f"@ lives.
```

```
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
            F = makeFreeOIModule(e, {1,1,2}, P);
            installGeneratorsInWidth(F, 4);
            f = x_(2,4)*e_(4,{3},1)+x_(1,3)^2*e_(4,{2,4},3)
            getFreeOIModule f
///

doc ///
    Key
        getSchreyerMap
        (getSchreyerMap,FreeOIModule)
    Headline
        get the Schreyer map of a free OI-module if it exists
    Usage
        getSchreyerMap H
    Inputs
        H:FreeOIModule
    Outputs
        :FreeOIModuleMap
    Description
        Text
            Let $G'$ be a non-empty Gröbner basis for the syzygy module of a
                ↪ finitely generated submodule $\mathbf{M}$ of a free OI-module $\
                ↪ mathbf{F}$ computed using @TO oiSyz@. Let @TT "H"@ be the free
                ↪ OI-module obtained by applying @TO getFreeOIModule@ to any
                ↪ element of $G'$. This method returns the canonical surjective
                ↪ map from @TT "H"@ to $\mathbf{M}$.
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
            F = makeFreeOIModule(e, {1,1}, P);
            installGeneratorsInWidth(F, 2);
            b = x_(1,2)*x_(1,1)*e_(2,{2},1)+x_(2,2)*x_(2,1)*e_(2,{1},2);
            G = oiGB {b}
            G' = oiSyz(G, d)
            H = getFreeOIModule G'#0
            getSchreyerMap H
    Caveat
        If $G'$ is empty or if @TT "H"@ does not have a Schreyer order, this method
            ↪  will throw an error.
///

doc ///
    Key
        getWidth
        (getWidth,VectorInWidth)
    Headline
        get the width of an element of a free OI-module
    Usage
        getWidth f
    Inputs
        f:VectorInWidth
    Outputs
        :ZZ
```

```
    Description
        Text
            Returns the width of an element of a free OI-module.
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
            F = makeFreeOIModule(e, {1,1,2}, P);
            installGeneratorsInWidth(F, 4);
            f = x_(2,4)*e_(4,{3},1)+x_(1,3)^2*e_(4,{2,4},3)
            getWidth f
///

doc ///
    Key
        getRank
        (getRank,FreeOIModule)
    Headline
        get the rank of a free OI-module
    Usage
        getRank F
    Inputs
        F:FreeOIModule
    Outputs
        :ZZ
    Description
        Text
            Returns the rank of a free OI-module. Recall that the rank of a free OI-
                ↪ module $\bigoplus_{i=1}^s\mathbf{F}^{\text{OI}, d_i}$ is defined
                ↪  to be $s$.
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
            F = makeFreeOIModule(e, {1,1,2}, P);
            getRank F
///

doc ///
    Key
        oiOrbit
        (oiOrbit,List,ZZ)
    Headline
        compute the n-orbit of a list of elements of a free OI-module
    Usage
        oiOrbit(L, n)
    Inputs
        L:List
        n:ZZ
    Outputs
        :List
    Description
        Text
            Returns the $n$-orbit of the list @TT "L"@ of @TO VectorInWidth@
                ↪ objects.
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
            F = makeFreeOIModule(e, {1,1}, P);
```

```
            installGeneratorsInWidth(F, 2);
            b = x_(1,1)*e_(2,{1},1)+x_(2,1)*e_(2,{1},2)
            oiOrbit({b}, 4)
///

doc ///
    Key
        (symbol SPACE,FreeOIModuleMap,VectorInWidth)
    Headline
        apply a free OI-module map to an element
    Usage
        phi f
    Inputs
        phi:FreeOIModuleMap
        f:VectorInWidth
    Outputs
        :VectorInWidth
    Description
        Text
            Evaluates @TT "phi"@ at @TT "f"@.
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
            F = makeFreeOIModule(e, {1,2}, P);
            installGeneratorsInWidth(F, 3);
            b = x_(1,2)*x_(1,1)*e_(3,{2},1)+x_(2,2)*x_(2,1)*e_(3,{1,3},2);
            C = oiRes({b}, 2)
            phi = C.dd_1
            G = getBasisElements C_1
            phi G#0
            phi G#1
///

doc ///
    Key
        (net,FreeOIModule)
    Headline
        display a free OI-module
    Usage
        net F
    Inputs
        F:FreeOIModule
    Outputs
        :Net
    Description
        Text
            Displays the basis symbol, basis element widths, degree shifts,
                ↪ underlying polynomial OI-algebra, and monomial order of a free
                ↪ OI-module.
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
            F = makeFreeOIModule(e, {1,2}, P);
            net F
///
```

```
doc ///
    Key
        (net,FreeOIModuleMap)
    Headline
        display a free OI-module map source and target
    Usage
        net phi
    Inputs
        phi:FreeOIModuleMap
    Outputs
        :Net
    Description
        Text
            Displays the source module and target module a free OI-module map.
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
            F = makeFreeOIModule(e, {1,2}, P);
            installGeneratorsInWidth(F, 3);
            b = x_(1,2)*x_(1,1)*e_(3,{2},1)+x_(2,2)*x_(2,1)*e_(3,{1,3},2);
            C = oiRes({b}, 2);
            phi = C.dd_1;
            net phi
///

doc ///
    Key
        (image,FreeOIModuleMap)
    Headline
        get the basis element images of a free OI-module map
    Usage
        image phi
    Inputs
        phi:FreeOIModuleMap
    Outputs
        :List
    Description
        Text
            Returns a list containing the basis element images of a free OI-module
                ↪ map.
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
            F = makeFreeOIModule(e, {1,2}, P);
            installGeneratorsInWidth(F, 3);
            b = x_(1,2)*x_(1,1)*e_(3,{2},1)+x_(2,2)*x_(2,1)*e_(3,{1,3},2);
            C = oiRes({b}, 2);
            phi = C.dd_1;
            image phi
///

doc ///
    Key
        (describe,FreeOIModuleMap)
    Headline
        display a free OI-module map
```

```
    Usage
        describe phi
    Inputs
        phi:FreeOIModuleMap
    Outputs
        :Net
    Description
        Text
            Displays the source module, target module, and basis element images of
                ↪ a free OI-module map.
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
            F = makeFreeOIModule(e, {1,2}, P);
            installGeneratorsInWidth(F, 3);
            b = x_(1,2)*x_(1,1)*e_(3,{2},1)+x_(2,2)*x_(2,1)*e_(3,{1,3},2);
            C = oiRes({b}, 2);
            phi = C.dd_1;
            describe phi
///

doc ///
    Key
        (net,ModuleInWidth)
    Headline
        display a component of a free OI-module in a specified width
    Usage
        net M
    Inputs
        M:ModuleInWidth
    Outputs
        :Net
    Description
        Text
            Displays information about a widthwise component of a free OI-module.
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
            F = makeFreeOIModule(e, {1,2}, P);
            net F_5
///

doc ///
    Key
        (net,VectorInWidth)
    Headline
        display an element of a free OI-module
    Usage
        net f
    Inputs
        f:VectorInWidth
    Outputs
        :Net
    Description
        Text
```

```
            Displays an element of a free OI-module. Note: terms are displayed in
                ↪ descending order according to the monomial order of the free OI-
                ↪ module.
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
            F = makeFreeOIModule(e, {1,1}, P);
            installGeneratorsInWidth(F, 2);
            f=x_(1,1)*e_(2,{1},1)+x_(2,1)*e_(2,{1},2)-x_(1,2)*e_(2,{2},1)-x_(2,2)*
                ↪ e_(2,{2},2);
            net f
///

doc ///
    Key
        (toString,FreeOIModule)
    Headline
        display a free OI-module in condensed form
    Usage
        toString F
    Inputs
        F:FreeOIModule
    Outputs
        :String
    Description
        Text
            Displays the basis symbol, basis element widths, and degree shifts of a
                ↪  free OI-module as a string.
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
            F = makeFreeOIModule(e, {1,2}, P);
            toString F
///

doc ///
    Key
        (symbol _,FreeOIModule,ZZ)
    Headline
        get the width n component of a free OI-module
    Usage
        F _ n
    Inputs
        F:FreeOIModule
        n:ZZ
    Outputs
        :ModuleInWidth
    Description
        Text
            Returns the module in width @TT "n"@ of a free OI-module.
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
            F = makeFreeOIModule(e, {1,2}, P);
            F_5
///
```

```
doc ///
    Key
        isZero
        (isZero,FreeOIModule)
    Headline
        check if a free OI-module is zero
    Usage
        isZero F
    Inputs
        F:FreeOIModule
    Outputs
        :Boolean
    Description
        Text
            Checks if a free OI-module is the zero module.
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
            F = makeFreeOIModule(e, {1,2}, P);
            G = makeFreeOIModule(d, {}, P);
            isZero F
            isZero G
///

doc ///
    Key
        (isZero,FreeOIModuleMap)
    Headline
        check if a free OI-module map is zero
    Usage
        isZero phi
    Inputs
        phi:FreeOIModuleMap
    Outputs
        :Boolean
    Description
        Text
            Checks if a free OI-module map is the zero map.
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
            F = makeFreeOIModule(e, {2}, P);
            installGeneratorsInWidth(F, 2);
            b = e_(2,{1,2},1);
            C = oiRes({b}, 2)
            phi0 = C.dd_0
            phi1 = C.dd_1
            isZero phi0
            isZero phi1
///

doc ///
    Key
        (isZero,VectorInWidth)
    Headline
        check if an element of a free OI-module is zero
```

```
    Usage
        isZero f
    Inputs
        f:VectorInWidth
    Outputs
        :Boolean
    Description
        Text
            Checks if an element of a free OI-module is zero.
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
            F = makeFreeOIModule(e, {1,2}, P);
            installGeneratorsInWidth(F, 3);
            f = x_(1,2)*x_(1,1)*e_(3,{2},1)+x_(2,2)*x_(2,1)*e_(3,{1,3},2)
            isZero f
            isZero(f-f)
///

doc ///
    Key
        (degree,VectorInWidth)
    Headline
        get the degree of an element of a free OI-module
    Usage
        degree f
    Inputs
        f:VectorInWidth
    Outputs
        :ZZ
    Description
        Text
            Returns the degree of the lead term of an element of a free OI-module.
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
            F = makeFreeOIModule(e, {1,2}, P);
            installGeneratorsInWidth(F, 3);
            f = x_(1,2)*x_(1,1)*e_(3,{2},1)+x_(2,2)*x_(2,1)*e_(3,{1,3},2)
            degree f
///

doc ///
    Key
        (use,ModuleInWidth)
    Headline
        use a component of a free OI-module
    Usage
        use M
    Inputs
        M:ModuleInWidth
    Description
        Text
            This method invokes @TO use@ on the underlying polynomial ring of @TT "
                ↪ M"@. This is useful since distinct components of a free OI-
                ↪ module need not be modules over the same polynomial ring.
```

```
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
            F = makeFreeOIModule(e, {1,1,2}, P);
            installGeneratorsInWidth(F, 1);
            installGeneratorsInWidth(F, 2);
            use F_1; b1 = x_(1,1)*e_(1,{1},1)+x_(2,1)*e_(1,{1},2)
            use F_2; b2 = x_(1,2)*x_(1,1)*e_(2,{2},2)+x_(2,2)*x_(2,1)*e_(2,{1,2},3)
///

doc ///
    Key
        (terms,VectorInWidth)
    Headline
        get the terms of an element of a free OI-module
    Usage
        terms f
    Inputs
        f:VectorInWidth
    Outputs
        :List
    Description
        Text
            Returns the list of terms of an element of a free OI-module.
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
            F = makeFreeOIModule(e, {1,1}, P);
            installGeneratorsInWidth(F, 2);
            f=x_(1,1)*e_(2,{1},1)+x_(2,1)*e_(2,{1},2)-x_(1,2)*e_(2,{2},1)-x_(2,2)*
                ↪ e_(2,{2},2)
            terms f
    Caveat
        The list of terms need not be in order. To find the lead term of an element
            ↪ , see @TO (leadTerm,VectorInWidth)@.
///

doc ///
    Key
        (leadTerm,VectorInWidth)
    Headline
        get the lead term of an element of a free OI-module
    Usage
        leadTerm f
    Inputs
        f:VectorInWidth
    Outputs
        :VectorInWidth
    Description
        Text
            Returns the lead term of an element of a free OI-module according to
                ↪ the specified @TO OIMonomialOrder@.
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
            F = makeFreeOIModule(e, {1,1}, P);
            installGeneratorsInWidth(F, 2);
```

```
            f=x_(1,1)*e_(2,{1},1)+x_(2,1)*e_(2,{1},2)-x_(1,2)*e_(2,{2},1)-x_(2,2)*
                ↪ e_(2,{2},2)
            leadTerm f
///

doc ///
    Key
        (leadMonomial,VectorInWidth)
    Headline
        get the lead monomial of an element of a free OI-module
    Usage
        leadMonomial f
    Inputs
        f:VectorInWidth
    Outputs
        :VectorInWidth
    Description
        Text
            Returns the lead monomial of an element of a free OI-module according
                ↪ to the specified @TO OIMonomialOrder@.
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
            F = makeFreeOIModule(e, {1,1}, P);
            installGeneratorsInWidth(F, 2);
            f=x_(1,1)*e_(2,{1},1)+x_(2,1)*e_(2,{1},2)-x_(1,2)*e_(2,{2},1)-x_(2,2)*
                ↪ e_(2,{2},2)
            leadMonomial f
    Caveat
        Asking for the lead monomial of a zero element will cause an error.
///

doc ///
    Key
        (leadCoefficient,VectorInWidth)
    Headline
        get the lead coefficient of an element of a free OI-module
    Usage
        leadCoefficient f
    Inputs
        f:VectorInWidth
    Outputs
        :VectorInWidth
    Description
        Text
            Returns the lead coefficient of an element of a free OI-module
                ↪ according to the specified @TO OIMonomialOrder@.
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
            F = makeFreeOIModule(e, {1,1}, P);
            installGeneratorsInWidth(F, 2);
            f=x_(1,1)*e_(2,{1},1)+x_(2,1)*e_(2,{1},2)-5*x_(1,2)*e_(2,{2},1)-x_(2,2)
                ↪ *e_(2,{2},2)
            leadCoefficient f
///
```

```
doc ///
    Key
        (symbol *,Number,VectorInWidth)
    Headline
        multiply an element of a free OI-module by a number
    Usage
        n * f
    Inputs
        n:Number
        f:VectorInWidth
    Outputs
        :VectorInWidth
    Description
        Text
            Multiplies @TT "f"@ by @TT "n"@.
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
            F = makeFreeOIModule(e, {1,1}, P);
            installGeneratorsInWidth(F, 2);
            f=x_(1,1)*e_(2,{1},1)+x_(2,1)*e_(2,{1},2)-5*x_(1,2)*e_(2,{2},1)-x_(2,2)
                ↪ *e_(2,{2},2)
            22*f
///

doc ///
    Key
        (symbol +,VectorInWidth,VectorInWidth)
    Headline
        add two elements of a free OI-module
    Usage
        f + g
    Inputs
        f:VectorInWidth
        g:VectorInWidth
    Outputs
        :VectorInWidth
    Description
        Text
            Adds @TT "f"@ and @TT "g"@.
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
            F = makeFreeOIModule(e, {1,1}, P);
            installGeneratorsInWidth(F, 2);
            f=x_(1,1)*e_(2,{1},1)+x_(2,1)*e_(2,{1},2)-5*x_(1,2)*e_(2,{2},1)-x_(2,2)
                ↪ *e_(2,{2},2)
            g = 5*x_(1,2)*e_(2,{2},1)
            f + g
///

doc ///
    Key
        (symbol *,RingElement,VectorInWidth)
    Headline
```

```
        multiply an element of a free OI-module by a ring element
    Usage
        r * f
    Inputs
        r:RingElement
        f:VectorInWidth
    Outputs
        :VectorInWidth
    Description
        Text
            Multiplies @TT "f"@ by @TT "r"@ provided that @TT "r"@ belongs to the
                ↪ underlying polynomial ring of the component in which @TT "f"@
                ↪ lives.
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
            F = makeFreeOIModule(e, {1,1}, P);
            installGeneratorsInWidth(F, 2);
            f=x_(1,1)*e_(2,{1},1)+x_(2,1)*e_(2,{1},2)-x_(1,2)*e_(2,{2},1)-x_(2,2)*
                ↪ e_(2,{2},2)
            (x_(1,2)+x_(2,2)^2)*f
///

doc ///
    Key
        (symbol -,VectorInWidth)
    Headline
        multiply an element of a free OI-module by -1
    Usage
        - f
    Inputs
        f:VectorInWidth
    Outputs
        :VectorInWidth
    Description
        Text
            Flips the sign of an element of a free OI-module.
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
            F = makeFreeOIModule(e, {1,2}, P);
            installGeneratorsInWidth(F, 3);
            f = x_(1,2)*x_(1,1)*e_(3,{2},1)+x_(2,2)*x_(2,1)*e_(3,{1,3},2)
            -f
///

doc ///
    Key
        (symbol -,VectorInWidth,VectorInWidth)
    Headline
        subtract an element of a free OI-module from another
    Usage
        f - g
    Inputs
        f:VectorInWidth
        g:VectorInWidth
```

```
    Outputs
        :VectorInWidth
    Description
        Text
            Subtracts @TT "g"@ from @TT "f"@.
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
            F = makeFreeOIModule(e, {1,2}, P);
            installGeneratorsInWidth(F, 3);
            f = x_(1,2)*x_(1,1)*e_(3,{2},1)+x_(2,2)*x_(2,1)*e_(3,{1,3},2)
            g = x_(2,2)*x_(2,1)*e_(3,{1,3},2)+x_(2,1)*e_(3,{1,2},2)
            f - g
///

doc ///
    Key
        (isHomogeneous,VectorInWidth)
    Headline
        check if an element of a free OI-module is homogeneous
    Usage
        isHomogeneous f
    Inputs
        f:VectorInWidth
    Outputs
        :Boolean
    Description
        Text
            Checks if an element of a free OI-module is homogeneous, i.e., if every
                ↪  term has the same degree.
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
            F = makeFreeOIModule(e, {1,2}, P);
            installGeneratorsInWidth(F, 3);
            f = x_(1,2)*x_(1,1)*e_(3,{2},1)+x_(2,2)*x_(2,1)*e_(3,{1,3},2)
            g = x_(2,2)*x_(2,1)*e_(3,{1,3},2)+x_(2,1)*e_(3,{1,2},2)
            isHomogeneous f
            isHomogeneous g
///

doc ///
    Key
        (isHomogeneous,FreeOIModuleMap)
    Headline
        checks if a free OI-module map is homogeneous
    Usage
        isHomogeneous phi
    Inputs
        phi:FreeOIModuleMap
    Outputs
        :Boolean
    Description
        Text
            Checks if a map of free OI-modules is homogeneous, i.e., if it
                ↪ preserves degrees of elements.
```

```
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
            F = makeFreeOIModule(e, {1,2}, P);
            installGeneratorsInWidth(F, 3);
            b = x_(1,2)*x_(1,1)*e_(3,{2},1)+x_(2,2)*x_(2,1)*e_(3,{1,3},2);
            C = oiRes({b}, 2);
            phi = C.dd_1
            isHomogeneous phi
///

doc ///
    Key
        oiGB
        (oiGB,List)
        [oiGB,Verbose]
        [oiGB,Strategy]
    Headline
        compute a Gröbner basis for a submodule of a free OI-module
    Usage
        oiGB L
    Inputs
        L:List
    Outputs
        :List
    Description
        Text
            Given a list of elements @TT "L"@ belonging to a free OI-module $\
                ↪ mathbf{F}$, this method computes a Gröbner basis for the
                ↪ submodule generated by @TT "L"@ with respect to the monomial
                ↪ order of $\mathbf{F}$. The @TO Verbose@ option must be either
                ↪ @TT "true"@ or @TT "false"@, depending on whether one wants
                ↪ debug information printed.

            The @TO Strategy@ option has the following permissible values:
        Code
            UL {
                {TT "Strategy => ", TT TO FastNonminimal, " for no post-processing
                    ↪ of the basis"},
                {TT "Strategy => ", TT TO Minimize, " to minimize the basis after it
                    ↪  is computed; see ", TO minimizeOIGB},
                {TT "Strategy => ", TT TO Reduce, " to reduce the basis after it is
                    ↪ computed; see ", TO reduceOIGB}
            }
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
            F = makeFreeOIModule(e, {1,1,2}, P);
            installGeneratorsInWidth(F, 1);
            installGeneratorsInWidth(F, 2);
            use F_1; b1 = x_(1,1)*e_(1,{1},1)+x_(2,1)*e_(1,{1},2);
            use F_2; b2 = x_(1,2)*x_(1,1)*e_(2,{2},2)+x_(2,2)*x_(2,1)*e_(2,{1,2},3)
                ↪ ;
            time oiGB {b1, b2}
///
```

```
doc ///
    Key
        minimizeOIGB
        (minimizeOIGB,List)
        [minimizeOIGB,Verbose]
    Headline
        minimize an OI-Gröbner basis
    Usage
        minimizeOIGB G
    Inputs
        G:List
    Outputs
        :List
    Description
        Text
            This method minimizes @TT "G"@ with respect to the Gröbner property, i.
                ↪ e., removes any elements whose leading monomial is OI-divisible
                ↪ by the leading monomial of another element of @TT "G"@. The @TO
                ↪ Verbose@ option must be either @TT "true"@ or @TT "false"@,
                ↪ depending on whether one wants debug information printed.
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
            F = makeFreeOIModule(e, {1,1,2}, P);
            installGeneratorsInWidth(F, 1);
            installGeneratorsInWidth(F, 2);
            installGeneratorsInWidth(F, 3);
            use F_1; b1 = x_(1,1)*e_(1,{1},1)+x_(2,1)*e_(1,{1},2);
            use F_2; b2 = x_(1,2)*x_(1,1)*e_(2,{2},2)+x_(2,2)*x_(2,1)*e_(2,{1,2},3)
                ↪ ;
            time B = oiGB {b1, b2}
            use F_3; b3 = x_(2,3)*x_(2,2)*x_(1,1)*e_(3,{2,3},3)-x_(2,1)*x_(1,2)^2*
                ↪ e_(3,{1,3},3);
            C = append(B, b3) -- dump in a redundant element
            minimizeOIGB C -- an element gets removed
///

doc ///
    Key
        reduceOIGB
        (reduceOIGB,List)
        [reduceOIGB,Verbose]
    Headline
        reduce an OI-Gröbner basis
    Usage
        reduceOIGB G
    Inputs
        G:List
    Outputs
        :List
    Description
        Text
            This method reduces @TT "G"@ as a Gröbner basis, i.e., ensures that no
                ↪ monomial of any element of @TT "G"@ is OI-divisible by the
                ↪ leading monomial of any other element of @TT "G"@. The @TO
```

```
        ↪ Verbose@ option must be either @TT "true"@ or @TT "false"@,
        ↪ depending on whether one wants debug information printed.
    Example
        P = makePolynomialOIAlgebra(2, x, QQ);
        F = makeFreeOIModule(e, {1,1,2}, P);
        installGeneratorsInWidth(F, 1);
        installGeneratorsInWidth(F, 2);
        use F_1; b1 = x_(2,1)*e_(1,{1},2)+x_(1,1)*e_(1,{1},2);
        use F_2; b2 = x_(1,2)*x_(1,1)*e_(2,{2},1)+x_(2,2)*x_(1,2)*e_(2,{2},2);
        time B = oiGB({b1, b2}, Strategy => FastNonminimal)
        minimizeOIGB B -- does not change the basis
        reduceOIGB B -- changes the basis
///

doc ///
    Key
        isOIGB
        (isOIGB,List)
        [isOIGB,Verbose]
    Headline
        check if a list of elements of a free OI-module forms a Gröbner basis
    Usage
        isOIGB L
    Inputs
        L:List
    Outputs
        :Boolean
    Description
        Text
            Checks if a @TT "L"@ forms a Gröbner basis for the submodule it
            ↪ generates. The @TO Verbose@ option must be either @TT "true"@ or
            ↪  @TT "false"@, depending on whether one wants debug information
            ↪ printed.
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
            F = makeFreeOIModule(e, {1,1,2}, P);
            installGeneratorsInWidth(F, 1);
            installGeneratorsInWidth(F, 2);
            installGeneratorsInWidth(F, 3);
            use F_1; b1 = x_(1,1)*e_(1,{1},1)+x_(2,1)*e_(1,{1},2);
            use F_2; b2 = x_(1,2)*x_(1,1)*e_(2,{2},2)+x_(2,2)*x_(2,1)*e_(2,{1,2},3)
            ↪ ;
            isOIGB {b1, b2}
            time B = oiGB {b1, b2}
            isOIGB B
///

doc ///
    Key
        oiSyz
        (oiSyz,List,Symbol)
        [oiSyz,Verbose]
        [oiSyz,Strategy]
    Headline
```

```
        compute a Gröbner basis for the syzygy module of a submodule of a free OI-
            ↪ module
    Usage
        oiSyz(G, d)
    Inputs
        G:List
        d:Symbol
    Outputs
        :List
    Description
        Text
            Given a non-empty Gröbner basis @TT "G"@ for a submodule $\mathbf{M}$
                ↪ of a free OI-module $\mathbf{F}$, this method computes a Gröbner
                ↪  basis $G'$ for the syzygy module of $\mathbf{M}$ with respect
                ↪ to the Schreyer order induced by $G$; see @TO OIMonomialOrder@.

            The new Gröbner basis $G'$ lives in an appropriate free OI-module $\
                ↪ mathbf{G}$ with basis symbol @TT "d"@ whose basis elements are
                ↪ mapped onto the elements of $G$ by a canonical surjective map $\
                ↪ varphi:\mathbf{G}\to\mathbf{M}$ (see Definition 4.1 of [1]).
                ↪ Moreover, the degrees of the basis elements of $\mathbf{G}$ are
                ↪ automatically shifted to coincide with the degrees of the
                ↪ elements of $G$, so that $\varphi$ is homogeneous if $G$
                ↪ consists of homogeneous elements. If $G'$ is not empty, then one
                ↪  obtains $\mathbf{G}$ by applying @TO getFreeOIModule@ to any
                ↪ element of $G'$. One obtains $\varphi$ by using @TO
                ↪ getSchreyerMap@.

            The @TO Verbose@ option must be either @TT "true"@ or @TT "false"@,
                ↪ depending on whether one wants debug information printed.

            The @TO Strategy@ option has the following permissible values:
        Code
            UL {
                {TT "Strategy => ", TT TO FastNonminimal, " for no post-processing
                    ↪ of the basis"},
                {TT "Strategy => ", TT TO Minimize, " to minimize the basis after it
                    ↪  is computed; see ", TO minimizeOIGB},
                {TT "Strategy => ", TT TO Reduce, " to reduce the basis after it is
                    ↪ computed; see ", TO reduceOIGB}
            }
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
            F = makeFreeOIModule(e, {1,1}, P);
            installGeneratorsInWidth(F, 2);
            b = x_(1,2)*x_(1,1)*e_(2,{2},1)+x_(2,2)*x_(2,1)*e_(2,{1},2);
            G = oiGB {b}
            oiSyz(G, d)
        Text
            {\em References:}

            [1] M. Morrow and U. Nagel, {\it Computing Gröbner Bases and Free
                ↪ Resolutions of OI-Modules}, Preprint, arXiv:2303.06725, 2023.
///
```

```
doc ///
    Key
        OIResolution
    Headline
        the class of all resolutions of submodules of free OI-modules
    Description
        Text
            This type implements free resolutions of submodules of free OI-modules.
                ↪  To make an @TT "OIResolution"@ object, use @TO oiRes@. To
                ↪ verify that an OI-resolution is a complex, use @TO isComplex@.
                ↪ To get the $n$th differential in an OI-resolution @TT "C"@, use
                ↪ @TT "C.dd_n"@.
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
            F = makeFreeOIModule(e, {1,1}, P);
            installGeneratorsInWidth(F, 2);
            b = x_(1,2)*x_(1,1)*e_(2,{2},1)+x_(2,2)*x_(2,1)*e_(2,{1},2);
            time C = oiRes({b}, 1)
            C.dd_0
///

doc ///
    Key
        describeFull
        (describeFull, OIResolution)
    Headline
        describe an OI-resolution and the maps
    Usage
        describe C
    Inputs
        C:OIResolution
    Outputs
        :Net
    Description
        Text
            Displays the free OI-modules and describes the differentials of an OI-
                ↪ resolution.
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
            F = makeFreeOIModule(e, {1,1}, P);
            installGeneratorsInWidth(F, 2);
            b = x_(1,2)*x_(1,1)*e_(2,{2},1)+x_(2,2)*x_(2,1)*e_(2,{1},2);
            time C = oiRes({b}, 1);
            describeFull C
///

doc ///
    Key
        (describe,OIResolution)
    Headline
        describe an OI-resolution
    Usage
        describe C
```

```
    Inputs
        C:OIResolution
    Outputs
        :Net
    Description
        Text
            Displays the free OI-modules and differentials of an OI-resolution.
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
            F = makeFreeOIModule(e, {1,1}, P);
            installGeneratorsInWidth(F, 2);
            b = x_(1,2)*x_(1,1)*e_(2,{2},1)+x_(2,2)*x_(2,1)*e_(2,{1},2);
            time C = oiRes({b}, 1);
            describe C
///

doc ///
    Key
        (net,OIResolution)
    Headline
        display an OI-resolution
    Usage
        net C
    Inputs
        C:OIResolution
    Outputs
        :Net
    Description
        Text
            Displays the basis element widths and degree shifts of the free OI-
                ↪ modules in an OI-resolution.
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
            F = makeFreeOIModule(e, {1,1}, P);
            installGeneratorsInWidth(F, 2);
            b = x_(1,2)*x_(1,1)*e_(2,{2},1)+x_(2,2)*x_(2,1)*e_(2,{1},2);
            time C = oiRes({b}, 1);
            net C
///

doc ///
    Key
        (symbol _,OIResolution,ZZ)
    Headline
        get a module of an OI-resolution is specified homological degree
    Usage
        C _ n
    Inputs
        C:OIResolution
        n:ZZ
    Outputs
        :FreeOIModule
    Description
        Text
```

```
            Returns the free OI-module of $C$ in homological degree $n$.
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
            F = makeFreeOIModule(e, {1,1}, P);
            installGeneratorsInWidth(F, 2);
            b = x_(1,2)*x_(1,1)*e_(2,{2},1)+x_(2,2)*x_(2,1)*e_(2,{1},2);
            time C = oiRes({b}, 1);
            C_0
            C_1
///

doc ///
    Key
        oiRes
        (oiRes,List,ZZ)
        [oiRes,Verbose]
        [oiRes,Strategy]
        [oiRes,TopNonminimal]
    Headline
        compute an OI-resolution
    Usage
        oiRes(L, n)
    Inputs
        L:List
        n:ZZ
    Outputs
        :OIResolution
    Description
        Text
            Computes an OI-resolution of the submodule generated by @TT "L"@ out to
                ↪  homological degree @TT "n"@. If @TT "L"@ consists of
                ↪ homogeneous elements, then the resulting resolution will be
                ↪ graded and minimal out to homological degree $n-1$. The @TO
                ↪ Verbose@ option must be either @TT "true"@ or @TT "false"@,
                ↪ depending on whether one wants debug information printed.

            The @TO Strategy@ option has the following permissible values:
        Code
            UL {
                {TT "Strategy => ", TT TO FastNonminimal, " for no post-processing
                    ↪ of the Gröbner basis computed at each step"},
                {TT "Strategy => ", TT TO Minimize, " to minimize the Gröbner basis
                    ↪ after it is computed at each step; see ", TO minimizeOIGB},
                {TT "Strategy => ", TT TO Reduce, " to reduce the Gröbner basis
                    ↪ after it is computed at each step; see ", TO reduceOIGB}
            }
        Text
            The @TO TopNonminimal@ option must be either @TT "true"@ or @TT "false"
                ↪ @, depending on whether one wants the Gröbner basis in
                ↪ homological degree $n-1$ to be minimized. Therefore, use @TT "
                ↪ TopNonminimal => true"@ for no minimization of the basis in
                ↪ degree $n-1$.
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
```

```
        F = makeFreeOIModule(e, {1,1}, P);
        installGeneratorsInWidth(F, 2);
        b = x_(1,2)*x_(1,1)*e_(2,{2},1)+x_(2,2)*x_(2,1)*e_(2,{1},2);
        time oiRes({b}, 2, TopNonminimal => true)
///

doc ///
  Key
    ranks
    (ranks,OIResolution)
  Headline
    display the ranks of an OI-resolution
  Usage
    ranks C
  Inputs
    C:OIResolution
  Outputs
    :Net
  Description
    Text
      Given an OI-resolution $C$, this method displays the ranks of the
        ↪ modules in $C$.
    Example
      P = makePolynomialOIAlgebra(2, x, QQ);
      F = makeFreeOIModule(e, {1,1}, P);
      installGeneratorsInWidth(F, 2);
      b = x_(1,2)*x_(1,1)*e_(2,{2},1)+x_(2,2)*x_(2,1)*e_(2,{1},2);
      C = oiRes({b}, 2)
      ranks C
///

doc ///
  Key
    restrictedRanks
    (restrictedRanks,OIResolution,ZZ)
  Headline
    display the ranks of an OI-resolution restricted to a given width
  Usage
    restrictedRanks(C,w)
  Inputs
    C:OIResolution
    w:ZZ
  Outputs
    :Net
  Description
    Text
      Given an OI-resolution $C$ and a width $w$, this method displays the
        ↪ ranks of the modules in $C$ restricted to width $w$.
    Example
      P = makePolynomialOIAlgebra(2, x, QQ);
      F = makeFreeOIModule(e, {1,1}, P);
      installGeneratorsInWidth(F, 2);
      b = x_(1,2)*x_(1,1)*e_(2,{2},1)+x_(2,2)*x_(2,1)*e_(2,{1},2);
      C = oiRes({b}, 2)
```

```
            ranks C
            restrictedRanks(C,5)
///

doc ///
    Key
        TopNonminimal
    Headline
        option for minimizing the top Gröbner basis of an OI-resolution
    Description
        Text
            Used as an optional argument in the @TO oiRes@ method. The @TT "
                ↪ TopNonminimal"@ option must be either @TT "true"@ or @TT "false"
                ↪ @, depending on whether one wants the Gröbner basis in
                ↪ homological degree $n-1$ to be minimized. Therefore, use @TT "
                ↪ TopNonminimal => true"@ for no minimization of the basis in
                ↪ degree $n-1$.
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
            F = makeFreeOIModule(e, {1,1}, P);
            installGeneratorsInWidth(F, 2);
            b = x_(1,2)*x_(1,1)*e_(2,{2},1)+x_(2,2)*x_(2,1)*e_(2,{1},2);
            time oiRes({b}, 2, TopNonminimal => true)
///

doc ///
    Key
        isComplex
        (isComplex,OIResolution)
        [isComplex,Verbose]
    Headline
        verify that an OI-resolution is a complex
    Usage
        isComplex C
    Inputs
        C:OIResolution
    Outputs
        :Boolean
    Description
        Text
            This method verifies that an OI-resolution is indeed a complex. The @TO
                ↪  Verbose@ option must be either @TT "true"@ or @TT "false"@,
                ↪ depending on whether one wants debug information printed.
        Example
            P = makePolynomialOIAlgebra(2, x, QQ);
            F = makeFreeOIModule(e, {1,1}, P);
            installGeneratorsInWidth(F, 2);
            b = x_(1,2)*x_(1,1)*e_(2,{2},1)+x_(2,2)*x_(2,1)*e_(2,{1},2);
            time C = oiRes({b}, 2, TopNonminimal => true)
            isComplex C
///

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```

```
-- TESTS ------------------------------------------------------------------
---------------------------------------------------------------------------
---------------------------------------------------------------------------

-- TEST 0
TEST ///
P = makePolynomialOIAlgebra(2, x, QQ);
F = makeFreeOIModule(e, {1,1,2}, P);
installGeneratorsInWidth(F, 1);
installGeneratorsInWidth(F, 2);
use F_1; b1 = x_(1,1)*e_(1,{1},1)+x_(2,1)*e_(1,{1},2);
use F_2; b2 = x_(1,2)*x_(1,1)*e_(2,{2},2)+x_(2,2)*x_(2,1)*e_(2,{1,2},3);
B = oiGB {b1, b2};
assert(#B === 3);
assert isOIGB B
///

-- TEST 1
TEST ///
P = makePolynomialOIAlgebra(2, x, QQ);
F = makeFreeOIModule(e, {1,1}, P);
installGeneratorsInWidth(F, 2);
b = x_(1,2)*x_(1,1)*e_(2,{2},1)+x_(2,2)*x_(2,1)*e_(2,{1},2);
B = oiGB {b};
assert(#B === 2);
C = oiSyz(B, d);
assert(#C === 3);
assert isOIGB C
///

-- TEST 2
TEST ///
P = makePolynomialOIAlgebra(2, x, QQ);
F = makeFreeOIModule(e, {1,2}, P);
installGeneratorsInWidth(F, 3);
b = x_(1,2)*x_(1,1)*e_(3,{2},1)+x_(2,2)*x_(2,1)*e_(3,{1,3},2);
C = oiRes({b}, 2);
assert isComplex C;
assert(getRank C_0 === 1);
assert(getRank C_1 === 2);
assert(apply(getBasisElements C_0, getWidth) === {3});
assert(apply(getBasisElements C_1, getWidth) === {5, 5})
///

-- TEST 3
TEST ///
P = makePolynomialOIAlgebra(2, x, QQ);
F = makeFreeOIModule(e, {1,1}, P);
installGeneratorsInWidth(F, 2);
b = x_(1,2)*x_(1,1)*e_(2,{2},1)+x_(2,2)*x_(2,1)*e_(2,{1},2);
C = oiRes({b}, 3);
assert isComplex C;
assert isHomogeneous(C.dd_0);
assert isHomogeneous(C.dd_1);
```

```
assert isHomogeneous(C.dd_2);
assert isHomogeneous(C.dd_3);
assert(getRank C_0 === 1);
assert(getRank C_1 === 1);
assert(getRank C_2 === 2);
assert(apply(getBasisElements C_0, getWidth) === {2});
assert(apply(getBasisElements C_1, getWidth) === {4});
assert(apply(getBasisElements C_2, getWidth) === {5,5})
///

end

-- GB example 1: one linear and one quadratic
-- Comment: see https://arxiv.org/pdf/2303.06725.pdf example 3.20
restart
P = makePolynomialOIAlgebra(2, x, QQ);
F = makeFreeOIModule(e, {1,1,2}, P);
installGeneratorsInWidth(F, 1);
installGeneratorsInWidth(F, 2);
use F_1; b1 = x_(1,1)*e_(1,{1},1)+x_(2,1)*e_(1,{1},2);
use F_2; b2 = x_(1,2)*x_(1,1)*e_(2,{2},2)+x_(2,2)*x_(2,1)*e_(2,{1,2},3);
time B = oiGB({b1, b2}, Verbose => true)

-- Res example 1: single quadratic in width 3
-- Comment: see https://arxiv.org/pdf/2303.06725.pdf example 5.5 (i)
restart
P = makePolynomialOIAlgebra(2, x, QQ);
F = makeFreeOIModule(e, {1,2}, P);
installGeneratorsInWidth(F, 3);
b = x_(1,2)*x_(1,1)*e_(3,{2},1)+x_(2,2)*x_(2,1)*e_(3,{1,3},2);
time C = oiRes({b}, 2, Verbose => true)

-- Res example 2: single quadratic in width 2
-- Comment: see https://arxiv.org/pdf/2303.06725.pdf example 5.5 (ii)
restart
P = makePolynomialOIAlgebra(2, x, QQ);
F = makeFreeOIModule(e, {1,1}, P);
installGeneratorsInWidth(F, 2);
b = x_(1,2)*x_(1,1)*e_(2,{2},1)+x_(2,2)*x_(2,1)*e_(2,{1},2);
time C = oiRes({b}, 4, Verbose => true)

-- Res example 3: single quadratic in width 2
-- Comment: compare with res example 1
restart
P = makePolynomialOIAlgebra(2, x, QQ);
F = makeFreeOIModule(e, {1}, P);
installGeneratorsInWidth(F, 2);
b = x_(1,2)*x_(1,1)*e_(2,{2},1)+x_(2,2)*x_(2,1)*e_(2,{1},1);
time C = oiRes({b}, 5, Verbose => true) -- Takes my laptop 30 minutes (minimal
    ↪ ranks 1, 2, 5, 9, 14)

-- Res example 4: single quadratic in width 3
-- Comment: compare with res example 1
restart
```

```
P = makePolynomialOIAlgebra(2, x, QQ);
F = makeFreeOIModule(e, {1,1}, P);
installGeneratorsInWidth(F, 3);
b = x_(1,2)*x_(1,1)*e_(3,{2},1)+x_(2,2)*x_(2,1)*e_(3,{1},2);
time C = oiRes({b}, 5, Verbose => true)

-- OI-ideal example
-- Comment: 2x2 minors with a gap of at least 1
restart
P = makePolynomialOIAlgebra(2, x, QQ);
F = makeFreeOIModule(e, {0}, P);
installGeneratorsInWidth(F, 3);
b = (x_(1,1)*x_(2,3)-x_(2,1)*x_(1,3))*e_(3,{},1);
time C = oiRes({b}, 2, Verbose => true)

-- Res example 5: single quadratic in width 3
-- Comment: compare with res examples 1 and 4
restart
P = makePolynomialOIAlgebra(2, x, QQ);
F = makeFreeOIModule(e, {1,1}, P);
installGeneratorsInWidth(F, 3);
b = x_(1,2)*x_(1,1)*e_(3,{2},1)+x_(2,2)*x_(2,1)*e_(3,{1},2);
time C = oiRes({b}, 3, Verbose => true)
ranks C
```

Bibliography

[**1**] A. Brouwer and J. Draisma, *Equivariant Gröbner bases and the two-factor model.* Math. Comput. **80** (2011), 1123–1133.

[**2**] D. E. Cohen, *Closure relations, Buchberger's algorithm, and polynomials in infinitely many variables*, In *Computation theory and logic*, Lecture Notes in Comput. Sci. **270** (1987), 78–87.

[**3**] T. W. Dubé, *The structure of polynomial ideals and Gröbner bases*, SIAM J. Comput. **19** (1990), 750—773.

[**4**] D. Eisenbud, *Commutative algebra*, Graduate Texts in Mathematics **150**, Springer-Verlag, New York, 1995.

[**5**] N. Fieldsteel and U. Nagel, *Minimal and cellular free resolutions over polynomial OI-algebras*, Preprint, arXiv:2105.08603, 2021.

[**6**] N. Fieldsteel and U. Nagel, *Buchsbaum-Eisenbud complexes of OI-modules*, Preprint, arXiv:2206.12960, 2022.

[**7**] D. Grayson and M. Stillman, *Macaulay2, a software system for research in algebraic geometry*; available at http://www.math.uiuc.edu/Macaulay2/.

[**8**] C. J. Hillar, R. Krone and A Leykin, *Equivariant Gröbner bases*, In: The 50th Anniversary of Gröbner Bases, T. Hibi, ed. (Tokyo: Mathematical Society of Japan, 2018), Adv. Stud. Pure Math. **77**, 129–154.

[**9**] C. J. Hillar and S. Sullivant, *Finite Gröbner bases in infinite dimensional polynomial rings and applications*, Adv. Math. **229** (2012), 1–25.

[**10**] M. Morrow, *OIGroebnerBases.m2*, Macaulay2 package; available at https://github.com/morrowmh/OIGroebnerBases.

[**11**] M. Morrow, *The OIGroebnerBases Package for Macaulay2*, Preprint, 2023, arXiv:2310.04891.

[**12**] M. Morrow and U. Nagel, *Computing Gröbner Bases and Free Resolutions of OI-Modules*, Preprint, arXiv:2303.06725, 2023.

[**13**] U. Nagel and T. Römer, *FI- and OI-modules with varying coefficients*, J. Algebra **535** (2019), 286-322.

[**14**] U. Nagel, *Rationality of equivariant Hilbert series and asymptotic properties*, Trans. Amer. Math. Soc. **374** (2021), no. 10, 7313-7357.

[**15**] S. V. Sam and A. Snowden, *Gröbner methods for representations of combinatorial categories*, J. Amer. Math. Soc. **30** (2017), 159–203.

[**16**] C. A. Weibel, *An introduction to homological algebra*, Cambridge Studies in Advanced Mathematics **38**, Cambridge University Press, Cambridge, 1994.

Vita

Michael Morrow

Place of Birth:

- Edmonds, WA

Education:

- University of Kentucky, Lexington, KY
  M.S. in Mathematics, April 2021

- Central Washington University, Ellensburg, WA
  B.S. in Mathematics, June 2019
  cum laude

Professional Positions:

- Graduate Teaching Assistant, University of Kentucky, Fall 2019–Spring 2024

Honors:

- Summer Research Fellowship, University of Kentucky, Summer 2023

- Eaves Fellowship, University of Kentucky, Summer 2022

- Dale and Mary Jo Comstock Scholarship, Central Washington University, Spring 2019

Publications & Preprints:

- M. Morrow and U. Nagel, *Computing Gröbner Bases and Free Resolutions of OI-Modules*, submitted, 2023. Available at arXiv:2303.06725.

- M. Morrow, *The OIGroebnerBases Package for Macaulay2*, submitted, 2023. Available at arXiv:2310.04891.

Software:

- M. Morrow, *OIGroebnerBases.m2*, a Macaulay2 package, 2023.
  Available at https://github.com/morrowmh/OIGroebnerBases.