# THE FAUGÈRE, GIANNI, LAZARD, AND MORA ALGORITHM FOR GRÖBNER BASIS TERM ORDER CONVERSION

MAHMOUD FAKHRY, ROBERT AVERY

Abstract. In this paper, we provide theoretical details and applications of the Faugère, Gianni, Lazard, and Mora algorithm for converting the term ordering of a Gröbner basis of a zero-dimensional ideal. We highlight the importance and usefulness of term order conversion in practice, with examples from cryptography and solving systems of polynomial equations. Utilizing results from elimination theory, we prove that the FGLM algorithm terminates and analyze its computational complexity. Step-by-step examples and pseudocode are provided to illustrate the algorithm. We also compare FGLM to alternative approaches and discuss future research directions in Gröbner basis term order conversion.

## 1. Introduction

Gröbner bases are a fundamental tool in computational algebra with applications ranging from solving systems of polynomial equations to cryptography and robotics path planning [3][4]. However, the term ordering used to compute a Gröbner basis can greatly impact the computation time. Certain orderings like degree reverse lexicographic (degrevlex) are much more efficient than others like lexicographic (lex) [2].

Therefore, being able to convert a Gröbner basis from one ordering to another is very useful in practice. A typical workflow is to first compute a Gröbner basis using a more efficient ordering like degrevlex, and then convert it to an ordering more suitable for the application, like lex. The FGLM algorithm, named after its creators Faugère, Gianni, Lazard, and Mora, does exactly this for zero-dimensional ideals [1].

In this paper, we provide a detailed treatment of the FGLM algorithm. Our specific contributions are:

- A clear exposition of the mathematical background and theory underlying FGLM, including key results from elimination theory.
- A precise description of the FGLM algorithm, with pseudocode and analysis of its computational complexity.
- Rigorous proof of the termination of FGLM on zero-dimensional ideals.
- Illustrative examples showing the application of FGLM to term order conversion.
- Discussion of alternative approaches and suggestions for future research.

The rest of the paper is organized as follows. In Section 2, we provide the necessary mathematical background required. Section 3 describes the FGLM algorithm in detail. Section 4 analyzes its computational complexity and proves termination. Examples are given in Section 5 and the Appendix. In Section 6, we conclude and discuss related work and future directions. Finally, code for the FGLM algorithm is found in the Appendix.

## 2. Background

In this section, we review some necessary mathematical background. Readers familiar with the theory of Gröbner bases may wish to skip ahead to Section 3.

**Definition 1.** *Let $k$ be a field and $R = k[x_1, \ldots, x_n]$ a polynomial ring over $k$. A* term ordering $>$ *on $R$ is a total ordering on the monomials $x^\alpha = x_1^{\alpha_1} \cdots x_n^{\alpha_n}$ that satisfies:*

(1) $1 < x^\alpha$ *for all* $\alpha \neq 0$.
(2) *If* $x^\alpha > x^\beta$, *then* $x^{\alpha+\gamma} > x^{\beta+\gamma}$ *for all* $\gamma$.

Some common term orderings are:

- Lexicographic order (lex): $x^\alpha >_{\text{lex}} x^\beta$ iff the leftmost nonzero entry of $\alpha - \beta$ is positive.
- Degree lexicographic order (deglex): $x^\alpha >_{\text{deglex}} x^\beta$ iff $|\alpha| > |\beta|$, or $|\alpha| = |\beta|$ and $x^\alpha >_{\text{lex}} x^\beta$.
- Degree reverse lexicographic order (degrevlex): $x^\alpha >_{\text{degrevlex}} x^\beta$ iff $|\alpha| > |\beta|$, or $|\alpha| = |\beta|$ and the rightmost nonzero entry of $\alpha - \beta$ is negative.

**Definition 2.** *Fix a term order $>$. Given a nonzero polynomial $f \in R$, the* leading term *of $f$, denoted* $\operatorname{lt}(f)$, *is the greatest monomial in $f$ with respect to $>$. The* leading coefficient $\operatorname{lc}(f)$ *is the coefficient of* $\operatorname{lt}(f)$. *The* leading monomial *is defined as* $\operatorname{lm}(f) = \operatorname{lc}(f) \operatorname{lt}(f)$.

**Definition 3.** *An ideal $I \subseteq R$ is* zero-dimensional *if the affine variety $V(I) \subseteq k^n$ is finite.*

**Definition 4.** *Fix a term order $>$. A finite subset $G = \{g_1, \ldots, g_t\}$ of an ideal $I$ is a* Gröbner basis *for $I$ with respect to $>$ if* $\langle \operatorname{lt}(g_1), \ldots, \operatorname{lt}(g_t) \rangle = \langle \operatorname{lt}(I) \rangle$.

In other words, the leading terms of a Gröbner basis generate the ideal of leading terms of the entire ideal. Gröbner bases have many important properties, such as:

- Every ideal $I \subseteq R$ has a Gröbner basis (with respect to any term order).
- The remainder of any $f \in R$ on division by a Gröbner basis is unique.
- $G$ is a Gröbner basis for $I$ iff the remainder of $S(g_i, g_j)$ on division by $G$ is 0 for all $i \neq j$, where $S(g_i, g_j)$ is the S-polynomial of $g_i$ and $g_j$. (Buchberger's criterion)

The third property gives a way to compute Gröbner bases, by starting with a generating set for $I$ and adding nonzero remainders of S-polynomials until the criterion is satisfied. This is essentially Buchberger's algorithm. More efficient variants like F4 and F5 are typically used in practice.

The following result on zero-dimensional ideals will be key:

**Theorem 1.** [2, Thm 2.3.2] *Let $I \subseteq k[x_1, \ldots, x_n]$ be an ideal and let $G$ be a Gröbner basis for $I$ with respect to any term order. Then $I$ is zero-dimensional iff $G$ contains a nonzero univariate polynomial in each $x_i$.*

Finally, we recall an important result from elimination theory:

**Theorem 2.** [2, Thm 2.3.4] *Let $I \subseteq k[x_1, \ldots, x_n]$ be an ideal and $G$ a Gröbner basis for $I$ with respect to lex order where $x_1 > \ldots > x_n$. Then, for every $0 \leq \ell \leq n$, the set*

$$G_\ell = G \cap k[x_{\ell+1}, \ldots, x_n]$$

*is a Gröbner basis of the $\ell$-th elimination ideal $I_\ell = I \cap k[x_{\ell+1}, \ldots, x_n]$.*

This shows that a lex Gröbner basis can be split into Gröbner bases for certain elimination ideals, which will be useful for proving termination of FGLM.

## 3. The FGLM Algorithm

We now describe the FGLM algorithm in detail, following the exposition in [1]. Pseudocode is provided in Algorithm 1.

The key idea is to incrementally build up a lex Gröbner basis $G_{>_{\text{lex}}}$ and the associated set of *standard monomials* $\mathcal{N}_{>_{\text{lex}}}$ (monomials not divisible by any leading term of $G_{>_{\text{lex}}}$) by considering each monomial $x^i$ in lex order. At each step, we reduce $x^i$ modulo the input basis $G_>$ with respect to the original term order $>$. If the result is a linear combination of monomials in $\mathcal{N}_{>_{\text{lex}}}$, we can construct a new polynomial in $I$ with leading term $x^i$ (Line 12). This polynomial is added to $G_{>_{\text{lex}}}$. If not, then $x^i$ remains a standard monomial and is added to $\mathcal{N}_{>_{\text{lex}}}$.

**Algorithm 1** FGLM($G_>, n$)

**Require:** Gröbner basis $G_>$ for zero-dimensional ideal $I \subseteq k[x_1, \ldots, x_n]$ wrt term order $>$
**Ensure:** Gröbner basis $G_{>_{\text{lex}}}$ for $I$ wrt lex order

1: $G_{>_{\text{lex}}} \leftarrow \emptyset$
2: $\mathcal{N}_{>_{\text{lex}}} \leftarrow \{1\}$
3: **for all** $i = (0, \ldots, 0)$ **to** $(d_1, \ldots, d_n)$ **do**
4: $\quad p \leftarrow \text{NormalForm}(x^i, G_>)$ ▷ Compute normal form of $x^i$ wrt $G_>$
5: $\quad c_1, \ldots, c_s \leftarrow$ coefficients of $p$ wrt $\mathcal{N}_{>_{\text{lex}}}$
6: $\quad$ **if** $c_1 = \cdots = c_s = 0$ **then** ▷ $x^i$ is a new standard monomial
7: $\quad\quad \mathcal{N}_{>_{\text{lex}}} \leftarrow \mathcal{N}_{>_{\text{lex}}} \cup \{x^i\}$
8: $\quad$ **else** ▷ New polynomial with leading term $x^i$ found
9: $\quad\quad g \leftarrow x^i - p$
10: $\quad\quad G_{>_{\text{lex}}} \leftarrow G_{>_{\text{lex}}} \cup \{g\}$
11: $\quad\quad$ **if** $\text{lm}(g) = x_1^k$ for some $k$ **then return** $G_{>_{\text{lex}}}$ ▷ Elimination test
12: $\quad$ **end if**
13: **end for**
14: **return** $G_{>_{\text{lex}}}$

There are two key optimizations in Algorithm 1 compared to the basic approach:

(1) We exploit the block structure of lex order, only considering monomials up to a certain bound (Line 3). For a zero-dimensional ideal, this bound is given by the vector of degrees $(d_1, \ldots, d_n)$ of a lex Gröbner basis.
(2) The elimination test on Line 12 allows us to terminate early once a polynomial in $G_{>_{\text{lex}}}$ with leading monomial a pure power of $x_1$ is found. This follows from the elimination theorem above.

## 4. Complexity Analysis & Termination

In this section, we analyze the computational complexity of Algorithm 1 and prove that it terminates on any zero-dimensional ideal.

### 4.1. **Complexity.** The main steps in each iteration are:

- Computing the normal form of a monomial modulo $G_>$ (Line 4). This is a standard operation that depends on the term order and the size and number of polynomials in $G_>$.

- Linear algebra to test for linear dependence on the standard monomials (Line 5). This takes $O(|\mathcal{N}_{>_{\text{lex}}}|^2)$ operations.
- Potentially reducing by the new polynomial if it is added to $G_{>_{\text{lex}}}$ (not shown). This takes $O(|\mathcal{N}_{>_{\text{lex}}}|)$ operations.

The size of $\mathcal{N}_{>_{\text{lex}}}$ is bounded by the number of standard monomials of the initial ideal $I$, which is the degree of the affine variety $V(I) \subseteq \overline{k}^n$. If we let $D = \deg V(I)$, then the overall complexity is $O(nD^3)$, where the factor of $n$ comes from the number of monomials considered [3].

In terms of memory usage, we need to store the expanding set $\mathcal{N}_{>_{\text{lex}}}$ and the lex Gröbner basis $G_{>_{\text{lex}}}$ being constructed. The size of $\mathcal{N}_{>_{\text{lex}}}$ is again bounded by $D$, and $|G_{>_{\text{lex}}}| \leq D$ as well since each polynomial rules out at least one standard monomial. So the total memory required is $O(nD)$.

### 4.2. **Termination.**

**Theorem 3.** *Algorithm 1 terminates on any Gröbner basis $G_>$ of a zero-dimensional ideal $I \subseteq k[x_1, \ldots, x_n]$.*

*Proof.* Since $I$ is zero-dimensional, by the Finiteness Theorem [2, Sec 2.5], $I$ has finitely many standard monomials. Moreover, each iteration of FGLM adds either a new standard monomial or a new polynomial to $G_{>_{\text{lex}}}$. The set of standard monomials can never shrink, so this process must terminate.

In fact, we can bound the number of iterations by $1 + \sum_{i=1}^{n} d_i$, where $(d_1, \ldots, d_n)$ is the degree vector of a lex Gröbner basis for $I$. This is because once we have a polynomial with leading monomial $x_1^{d_1}$, FGLM will terminate by the elimination test (Line 12), invoking the Elimination Theorem. Since lex is an elimination order, a polynomial with leading monomial $x_1^{d_1}$ implies that $\langle x_1^{d_1} \rangle \subseteq I \cap k[x_1]$, so the first elimination ideal is generated by a pure power of $x_1$. By the Elimination Theorem, this means we have a lex Gröbner basis for $I$. □

## 5. Examples

We now illustrate the application of FGLM to some concrete examples. All computations are done in Sage. Note that two detailed worked solutions for computations can be found in the Appendix alongside the full code implementation of the FGLM algorithm.

**Example 1.** *Let $I = \langle y^3 + x^2, x^2y + x^2, x^3 - x^2, z^4 - x^2 - y \rangle \subseteq \mathbb{Q}[x, y, z]$. We first compute a degrevlex Gröbner basis:*

```
sage: R.<x,y,z> = PolynomialRing(QQ, order='degrevlex')
sage: I = ideal(y^3+x^2, x^2*y+x^2, x^3-x^2, z^4-x^2-y)
sage: I.groebner_basis()
[z^4 - x^2 - y, x^3 - x^2, x^2*y + x^2, y^3 + x^2]
```

*Now we convert to lex using FGLM:*

```
sage: S.<x,y,z> = PolynomialRing(QQ, order='lex')
sage: J = I.change_ring(S)
sage: J.transformed_basis('fglm')
[x^2 + y^3, x*y^3 - y^3, y^4 + y^3, z^4 + y^3 - y]
```

*Note how the resulting lex basis has a polynomial with leading term $x^2$, allowing the algorithm to terminate early.*

**Example 2.** *Let* $I = \langle y^3, x^2, x^3, z^3 - y \rangle \subseteq \mathbb{Q}[x, y, z]$. *A deglex Gröbner basis is:*

```
sage: R.<x,y,z> = PolynomialRing(QQ, order='deglex')
sage: I = ideal(y^3, x^2, x^3, z^3-y)
sage: I.groebner_basis()
[y^3, z^3 - y, x^2]
```

*Converting to lex:*

```
sage: S.<x,y,z> = PolynomialRing(QQ, order='lex')
sage: J = I.change_ring(S)
sage: J.transformed_basis('fglm')
[x^2, y^3, z^3 - y]
```

*Here FGLM does not terminate early, as there is no polynomial with leading term a pure power of* $x$. *The degree bound is reached instead.*

## 6. Conclusion & Future Work

In this paper we presented a detailed treatment of the FGLM algorithm for converting Gröbner bases of zero-dimensional ideals to different term orders, with a focus on the conversion to lex order. We provided the mathematical background, a complexity analysis, and a proof of termination. The effectiveness of the algorithm was demonstrated on some examples.

There are several directions for future research:

- Extending FGLM to positive-dimensional ideals. The main obstacle is that such ideals may have infinitely many standard monomials. Faugère proposed a possible extension of the FGLM algorithm [1].
- Improving the linear algebra. The complexity of FGLM (as typically implemented using row reduction) is dominated by the linear algebra operations to test for linear dependence on the standard monomials. Sparse linear algebra techniques or fast dense methods like Wiedemann's algorithm could potentially be used [3].
- Exploiting symmetry. If the ideal has symmetries induced by the action of a finite group, this can be used to speed up FGLM by identifying standard monomials that are equivalent under the group action [3].

In conclusion, FGLM is a fundamental algorithm in computational commutative algebra with many applications.

## References


[1] T. Mora. *Solving Polynomial Equation Systems II: Macaulay's paradigm and Gröbner technology*. Cambridge University Press, Cambridge, 2005.

[2] D. R. Cox, J. Little, and Donal O'shea. *Ideals, varieties, and algorithms: An introduction to computational algebraic geometry and commutative algebra*. New York: Springer, 2007.

[3] F. Winkler. *Polynomial Algorithms in Computer Algebra*. Springer Science & Business Media, 2012.

[4] W. W. Adams and Philippe Loustaunau. *An Introduction to Gröbner Bases*. Providence, R.I.: American Mathematical Society, 1994.

## Appendix A. Detailed Worked Examples

The curious reader should note that the two examples in Appendix A correspond directly to the code that is provided in Appendix B. The reader is encouraged to follow along by running the code in Appendix B in tandem to best understand the content.

**Example 1**

Let $x > y$, and let $I$ be an ideal of the polynomial ring $\mathbb{Q}[x, y]$ such that $I = \langle y^3 + x^2, x^2y + x^2 \rangle$. For degree reverse lexicographical term ordering, it is given that the Gröbner basis of $I$, denoted as $G_{\text{old}}$, is the set of polynomials $\{x^4 - x^2, x^2y + x^2, y^3 + x^2\}$.

Additionally, we begin with a set $N = \{1\}$ which will be used in the algorithm to contain linearly independent polynomials (which are monomials reduced by $G_{\text{old}}$). Similarly, we will define a set $O = \{1\}$ which will contain all monomials processed at each iteration of the algorithm. Finally, we initialize a set $G_{\text{new}}$ which will be the desired Gröbner basis upon termination.

To begin, we start with the least variable, which in this case is $y$ since we have $x > y$. (In the general case, given $x_1 > \cdots > x_n$, we iterate through the algorithm in the order $x_n, \ldots, x_1$.) Next, we reduce $y^1$ by $G_{\text{old}}$ in order to obtain the remainder of $y^1$ upon division by $G_{\text{old}}$.

$$\overline{y^1}^{G_{\text{old}}} = y$$

First, we add $y$ to $O$ such that $O := O \cup \{y\}$. Next, we check whether $\overline{y^1}^{G_{\text{old}}}$ is linearly independent to all polynomials in $N$. To do so, we add $\overline{y^1}^{G_{\text{old}}}$ to the set $N$ such that $N := N \cup \{\overline{y^1}^{G_{\text{old}}}\}$ and construct a coordinate matrix where each column is the coordinate vector of each polynomial in $N$ and proceed to check whether the nullity of the matrix is 0. Using a basis $\{1, y\}$ for the coordinate vectors, in reduced row echelon form, this matrix becomes

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

In this case, the polynomials are linearly independent, so we continue the algorithm by increasing the exponent on $y$ by 1 and repeating as follows:

$$\overline{y^2}^{G_{\text{old}}} = y^2$$

$O := O \cup \{y^2\}$, so $O = \{1, y, y^2\}$, and $N := N \cup \{\overline{y^2}^{G_{\text{old}}}\}$, so $N = \{1, \overline{y^1}^{G_{\text{old}}}, \overline{y^2}^{G_{\text{old}}}\}$.

$$\overline{y^3}^{G_{\text{old}}} = -x^2$$

$O := O \cup \{y^3\}$, so $O = \{1, y, y^2, y^3\}$, and $N := N \cup \{\overline{y^3}^{G_{\text{old}}}\}$, so $N = \{1, \overline{y^1}^{G_{\text{old}}}, \overline{y^2}^{G_{\text{old}}}, \overline{y^3}^{G_{\text{old}}}\} = \{1, y, y^2, -x^2\}$. Now, we notice that $O \neq N$.

$$\overline{y^4}^{G_{\text{old}}} = x^2$$

$O := O \cup \{y^4\}$, so $O = \{1, y, y^2, y^3, y^4\}$, and $N := N \cup \{\overline{y^4}^{G_{\text{old}}}\}$, so $N = \{1, \overline{y^1}^{G_{\text{old}}}, \overline{y^2}^{G_{\text{old}}}, \overline{y^3}^{G_{\text{old}}}, \overline{y^4}^{G_{\text{old}}}\} = \{1, y, y^2, -x^2, x^2\}$.

Now, since $N$ is a linearly dependent set of polynomials with the addition of $\overline{y^4}^{G_{\text{old}}}$, we remove $\overline{y^4}^{G_{\text{old}}}$ from $N$ with $N := N \setminus \{\overline{y^4}^{G_{\text{old}}}\}$. Next, a basis for the nullspace of the coordinate matrix can be obtained: $[0, 0, 0, 1, 1]$. These values correspond to the coefficients of the polynomials in $N$ that sum to 0 and can be written as the following sum where each $c_i$ corresponds to the $i^{\text{th}}$ entry in the basis vector (indexing from 0).

$$\overline{y^4}^{G_{\text{old}}} + \sum_{i=0}^{3} c_i \overline{y^i}^{G_{\text{old}}} = 0$$

From this fact, we know that $y^4 + \sum_{i=0}^{3} c_i y^i \in I$, and therefore add this polynomial into $G_{\text{new}}$. Notice that the sum that produces this polynomial is equivalently obtained by creating a vector whose entries are the polynomials in $O$ and taking the dot product of this vector with a basis vector in the nullspace of the coordinate matrix. Next, we continue with $x$ instead of $y$.

$$\overline{x^1}^{G_{\text{old}}} = x$$
$$\overline{x^2}^{G_{\text{old}}} = x^2$$

We find that $\overline{y^3}^{G_{\text{old}}} + \overline{x^2}^{G_{\text{old}}} = 0$ since a basis for the nullspace of the coordinate matrix is $[0, 0, 0, 1, 0, 1]$, so now $G_{\text{new}} = \{y^4 + y^3, x^2 + y^3\}$, and the algorithm terminates since no variables remain.

**Example 2:**

Let $z > y > x$, and let $I$ be an ideal of the polynomial ring $\mathbb{Q}[x, y, z]$ such that $I = \langle 5z + y^3, x^2y + x^2 + y + y^3, y^3 \rangle$. For degree reverse lexicographical term ordering, it is given that the Gröbner basis of $I$, denoted as $G_{\text{old}}$, is the set of polynomials $\{x^4 - x^2 - y, yx^2 + x^2 + y, y^2 - x^2 - y, z\}$.

We will omit unnecessary steps but will continue using the same sets as before. We begin with the smallest variable which is $x$.

$$\overline{x^1}^{G_{\text{old}}} = x$$
$$\overline{x^2}^{G_{\text{old}}} = x^2$$
$$\overline{x^3}^{G_{\text{old}}} = x^3$$
$$\overline{x^4}^{G_{\text{old}}} = x^2 + y$$
$$\overline{x^5}^{G_{\text{old}}} = x^3 + yx$$
$$\overline{x^6}^{G_{\text{old}}} = 0$$

Now, we have our current coordinate matrix in reduced row echelon form for $N = \{1, x, x^2, x^3, x^2 + y, x^3 + yx, 0\}$.

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

We find that a basis for the nullspace is $[0, 0, 0, 0, 0, 0, 1]$. So, currently $G_{\text{new}} = \{x^6\}$, and after $N := N \setminus \{\overline{x^6}^{G_{\text{old}}}\}$, we have $N = \{1, x, x^2, x^3, x^2 + y, x^3 + yx\}$.

$$\overline{y^1}^{G_{\text{old}}} = y$$

Now, we have our current coordinate matrix in reduced row echelon form for $N = \{1, x, x^2, x^3, x^2 + y, x^3 + yx, y\}$.

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

We find that a basis for the nullspace is $[0, 0, 1, 0, -1, 0, 1]$. So, currently $G_{\text{new}} = \{x^6, -x^4 + x^2 + y\}$, and after $N := N \setminus \{\overline{y^1}^{G_{\text{old}}}\}$, we have $N = \{1, x, x^2, x^3, x^2 + y, x^3 + yx\}$.

$$\overline{z^1}^{G_{\text{old}}} = 0$$

Now, we have our current coordinate matrix in reduced row echelon form for $N = \{1, x, x^2, x^3, x^2 + y, x^3 + yx, 0\}$.

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

We find that a basis for the nullspace is $[0, 0, 0, 0, 0, 0, 1]$. So, currently $G_{\text{new}} = \{x^6, -x^4 + x^2 + y, z\}$, and after $N := N \setminus \{\overline{z^1}^{G_{\text{old}}}\}$, we have $N = \{1, x, x^2, x^3, x^2 + y, x^3 + yx\}$.

Since all variables in our ring have been computed, the algorithm terminates and returns $G_{\text{new}} = \{x^6, -x^4 + x^2 + y, z\}$.

## Appendix B. FGLM Algorithm Implementation

**Example 1**

```
from sage.matrix.constructor import matrix

def nullspace_of_polynomialsR(S):
    # Get the monomials appearing in all polynomials
    all_monomials = set()
    for poly in S:
        all_monomials.update(poly.monomials())

    # Create coordinate vectors for each polynomial
    coord_vectors = []
    for polynomial in S:
        coeffs = [polynomial.monomial_coefficient(monomial) for monomial
        in all_monomials] coord_vectors.append(vector(coeffs))

    # Matrix with coordinate vectors as columns
    A = matrix(coord_vectors).transpose()

    # Print original matrix
    print("Original Matrix:")
    print(A)

    # Rank of the matrix
    rank = A.rank()

    print("Matrix in Reduced Row Echelon form:")
    print(A.rref())

    # Nullspace of the matrix
    nullspace = A.right_kernel()

    # Check if polynomials are linearly independent
    lin_indp = 1 if rank == len(S) else 0

    return lin_indp, rank, nullspace

# Example usage
R.<x,y> = PolynomialRing(ZZ, ['x', 'y'], order='degrevlex')
f1 = x^2
f2 = x^1
f3 = x^3
f4 = 4*(x^5 + y)
S = {f1, f2, f3, f4}
```

```
S1 = {f1, f2, f3}

lin_indp, rank, nullspace = nullspace_of_polynomialsR(S)
print("Rank of the matrix:", rank, "Lin indp:",lin_indp)
print("Basis for the nullspace:")
print(nullspace.basis())

lin_indp, rank, nullspace = nullspace_of_polynomialsR(S1)
print("Rank of the matrix:", rank, "Lin indp:",lin_indp)
print("Basis for the nullspace:")
print(nullspace.basis())

from sage.matrix.constructor import matrix

def fglm(S, G1):
    G2 = list()
    independent_polynomials = list()
    independent_polynomials.append(R.gen(1)^0)
    check_polys = independent_polynomials.copy()

    original_polys = list()
    original_polys.append(R.gen(1)^0)
    # Iterate through each variable
    for i in [1, 0]:

        exp = 1
        while True:
            # Reduce the monomial by G1
            original = R.gen(i)^exp
            original_polys.append(original)
            print("original polys:",original_polys)

            reduced = (original).reduce(G1)

            print("i",i,"#1",independent_polynomials)

            check_polys.append(reduced)

            print("i",i,"#1",independent_polynomials)

            # Check if the reduced monomial is a linear
            combination of elements in independent_polynomials
            lin_indp, _, nullspace =
            nullspace_of_polynomialsR(check_polys)
```

```
            print("reduced:",reduced,"linindp",lin_indp)
            if lin_indp:
                independent_polynomials.append(reduced)

                exp += 1 # inc exp
            else:
                # Extract coefficients from a vector in the nullspace

                basis_matrix = nullspace.basis_matrix()
                coefficients = basis_matrix[0]

                print("coefficients",basis_matrix)
                print("current polys:",independent_polynomials)

                dot_product = coefficients * vector(original_polys)
                G2.append(dot_product)

                original_polys.pop()
                check_polys.pop()

                break

    return G2

# Example input
R.<x,y> = PolynomialRing(QQ,2,order='degrevlex')
f1 = y^3+x^2
f2 = x^2*y+x^2

S = [f1, f2]

I = ideal(f1, f2)*R

# Compute the Groebner basis G1 in degrevlex order
G1 = I.groebner_basis()

# New lex order Groebner basis
G2 = fglm(S, G1)

print("G1 (degrevlex order):", G1)
print("G2 (lex order):", G2)
```

**Example 2**

```
from sage.matrix.constructor import matrix

def fglm(S, G1):
    G2 = list()
    independent_polynomials = list()
    independent_polynomials.append(R.gen(1)^0)
    check_polys = independent_polynomials.copy()

    original_polys = list()
    original_polys.append(R.gen(1)^0)
    # Iterate through each variable

    for i in [2, 1, 0]:

        exp = 1
        while True:
            # Reduce the monomial by G1
            original = R.gen(i)^exp
            original_polys.append(original)
            print("original polys:",original_polys)

            reduced = (original).reduce(G1)

            print("i",i,"#1",independent_polynomials)

            check_polys.append(reduced)

            print("i",i,"#1",independent_polynomials)

            # Check if the reduced monomial is a
            linear combination of elements in independent_polynomials
            lin_indp, _, nullspace = nullspace_of_polynomialsR(check_polys)
            print("reduced:",reduced,"linindp",lin_indp)
            if lin_indp:
                independent_polynomials.append(reduced)

                exp += 1 # inc exp
            else:
                # Extract coefficients from a vector in the nullspace

                basis_matrix = nullspace.basis_matrix()
                coefficients = basis_matrix[0]

                print("coefficients",basis_matrix)
```

```
                print("current polys:",independent_polynomials)

                dot_product = coefficients * vector(original_polys)
                G2.append(dot_product)

                print("G2 is", G2)

                original_polys.pop()
                check_polys.pop()

                break

    return G2

R.<z,y,x> = PolynomialRing(QQ,3,order='degrevlex')
f1 = y^3
f2 = x^2*y + x^2 + y + y^3
f3 = z*5 + y^3

S = [f1, f2, f3]

I = ideal(f1, f2, f3)*R

# Compute the Groebner basis G1 in degrevlex order
G1 = I.groebner_basis()
print("Original Groebner basis", G1)

# New lex order Groebner basis
G2 = fglm(S, G1)

print("G1 (degrevlex order):", G1)
print("G2 (lex order):", G2)
```