4-21-2022 11:15 AM

Algorithms for Regular Chains of Dimension One

Juan P. Gonzalez Trochez, *The University of Western Ontario*

Supervisor: Moreno Maza, Marc, *The University of Western Ontario*

A thesis submitted in partial fulfillment of the requirements for the Master of Science degree in Computer Science

Abstract

One of the core commands in the RegularChains library inside MAPLE is Triangularize. The underlying decomposes the solution set of a polynomial system into geometrically meaningful components represented by regular chains. This algorithm works by repeatedly calling a procedure, called Intersect, which computes the common zeros of a polynomial p and a regular chain T.

As the number of variables of p and T, as well as their degrees, increase, the call to the function Intersect(p, T) becomes more and more computationally expensive. It was observed in [16] that when the input polynomial system is zero-dimensional and T is one-dimensional then this cost can be substantially reduced. The method proposed by the authors is a probabilistic algorithm based on evaluation and interpolation techniques. This is the type of method which is typically challenging to implement in a high-level language like MAPLE's language, as a sharp control of computing resources (in particular memory) is needed. In this document, we report on a successful implementation of this algorithm in MAPLE as well as in the BPAS library.

On the other hand, multivariate Laurent series are a generalization of multivariate power series. They play an important roll when defining Puiseux series, and in consequence, they are required to implement Nowak's version of the famous Newton-Puiseux algorithm. In this document, we also report a first implementation of a Laurent series object inside MAPLE together with the challenges that we have encountered during its development.

Keywords: Regular chains, modular method, evaluation and interpolation, Laurent series, power series, subresultant chains

Lay Summary

Regular chains are triangular sets of polynomials used to solve systems of polynomial equations. In this work, we focused in one dimensional regular chains. That is, regular chains where the number of equations is one less than the number of variables. In particularly, we solve the intersection between a one dimensional regular chain and a polynomial by means of evaluation and interpolation, under certain hypothesis. A MAPLE and a C version of this modular algorithm is presented. In addition, we also report a first implementation of a Laurent series object inside MAPLE as a first step to achieve multivariate Puiseux series. Multivariate Laurent series are a generalization of multivariate power series; and Puiseux series are key when computing limit points of one dimensional regular chains.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

The theories of Gröbner bases and regular chains are two of the main tools on which algorithms for polynomial algebra rely. They were introduced respectively in [13] by Bruno Buchberger and in [20] by Michael Kalkbrener, a PhD student of Bruno Buchberger. These theories support a variety of algorithms covering problems like solving systems of polynomial (algebraic or differential) equations, and operating on ideals and varieties. See the landmark textbook [17] of David A. Cox, John Little and Donal O'Shea for an introduction to those problems.

In an attempt to highlight the specific features of each of these two theories, one can say that Gröbner bases are better suited for manipulating polynomial ideals, while regular chains are well adapted for operating on algebraic varieties and constructible sets. Algorithms based on Gröbner bases tend to either decompose polynomials into homogeneous components, or proceed by variable elimination. Meanwhile, the most successful algorithms based on regular chains use the paradigm of incremental solving, where solving a system of equations

$$f_1 = f_2 = \cdots = f_m = 0$$

is done by, for $i = 1, \ldots, m$, solving $f_i = 0$ against the solution set of

$$f_1 = f_2 = \cdots = f_{i-1} = 0.$$

In the realm of methods for solving polynomial systems, the principle of *incremental solving* plays a major role, yielding elegant algorithms and attractive complexity results. This principle allows one to control the properties and the size of the algebraic entities (curves, surfaces, etc.) that are produced at each computational step. See the works of Lazard [21], Lecerf [22] and Faugère [8].

Lazard proposed incremental solving for computing triangular decompositions by means of regular chains, a path which was extended and improved by Moreno Maza [26] and Chen [16]. In their work, the incremental solving process is based on a procedure, named `Intersect`, which computes the intersection of

1. a hyper-surface $V(f)$ and,
2. the quasi-component $W(T)$ of a regular chain T.

To rephrase the above statement in loose terms, the procedure `Intersect` computes the common solutions of a regular chain and a polynomial.

Returning to a more formal presentation, for a field $\mathbb{K}$, ordered variables $x_1 < \cdots < x_n$, a given multivariate polynomial $f \in \mathbb{K}[x_1, \ldots, x_n]$ and a regular chain $T \subset \mathbb{K}[x_1, \ldots, x_n]$, the

function call `Intersect`(f, T) returns regular chains $T_1, \ldots, T_e \subset \mathbb{K}[x_1, \ldots, x_n]$ such that we have:

$$V(f) \cap W(T) \subseteq W(T_1) \cup \cdots \cup W(T_e) \subseteq V(f) \cap \overline{W(T)}, \tag{1.1}$$

where $V(f)$ denotes the zero set of f, $W(T)$ denotes the common solutions of the polynomials of T and $\overline{W(T)}$ denotes the topological closure of $W(T)$ for the topology of Zariski. The reader may be surprised to see that $V(f) \cap W(T)$ is not given by an equality. Instead, `Intersect`(f, T) computes an "approximate decomposition" $W(T_1) \cup \cdots \cup W(T_e)$ which contains $V(f) \cap W(T)$ and is contained in $V(f) \cap \overline{W(T)}$. Since $V(f) \cap W(T)$ and $V(f) \cap \overline{W(T)}$ are topologically very close, this approximation is sharp. Moreover, it turns out that, when solving a system of polynomial equations, say

$$F = \begin{cases} f_1 &= 0 \\ f_2 &= 0 \\ \vdots & \vdots \\ f_m &= 0 \end{cases},$$

by repeated applications of the procedure `Intersect` with $f_1, f_2 \ldots, f_m$, regular chains $\{T_1, \ldots, T_v\} \subset \mathbb{K}[x_1, \ldots, x_n]$ are obtained such that we have:

$$V(F) = W(T_1) \cup \cdots \cup W(T_v).$$

This is essentially how the `Triangularize` command of the `RegularChains` library in MAPLE works.

Figure 1.1 illustrates how `Triangularize` solves an input polynomial system F incrementally, while Figure 1.2 shows a MAPLE session displaying `Triangularize`'s output for the same system F.

On Figure 1.1, all regular chains (the intermediate ones and those in the output) are organized by dimension. At the bottom, we see the regular chains in the output and note that they all have dimension zero, that is, each of them has finitely many solutions. Immediately above the zero-dimensional regular chains, all regular chains have dimension one; indeed each of them has 3 equations for 4 variables. In the above row, regular chains have dimension two; above them is the regular chain $\{y + w\}$, which has dimension three. One can see that, in solving the system F, the step from regular chains of dimension one to regular chains of dimension zero, has the dominant computational cost. This observation is very often true for systems with finitely many solutions. Figure 1.3 provides another illustration of this observation.

This observation motivates the first objective of this dissertation: controlling expression swell when computing `Intersect`(f, T) with a one-dimensional regular chain. Our contribution to this question is summarized in Section 1.1 and detailed in Chapter 2.

Not all polynomial systems are zero-dimensional, of course. Therefore, obtaining regular chains of positive dimension in the solution set of a polynomial system may happen, in particular, when the input system has fewer equations than variables. Consider then a polynomial set $F \subset \mathbb{K}[x_1, \ldots, x_n]$ and regular chains $T_1, \ldots, T_v \subset \mathbb{K}[x_1, \ldots, x_n]$ such that we have:

$$V(F) = W(T_1) \cup \cdots \cup W(T_v),$$

and at least one regular chain, say T, is of dimension 1. From the definition of a regular chain (see Section 2.1) and since variables are ordered as $x_1 < x_2 < \cdots < x_n$, one can assume that the

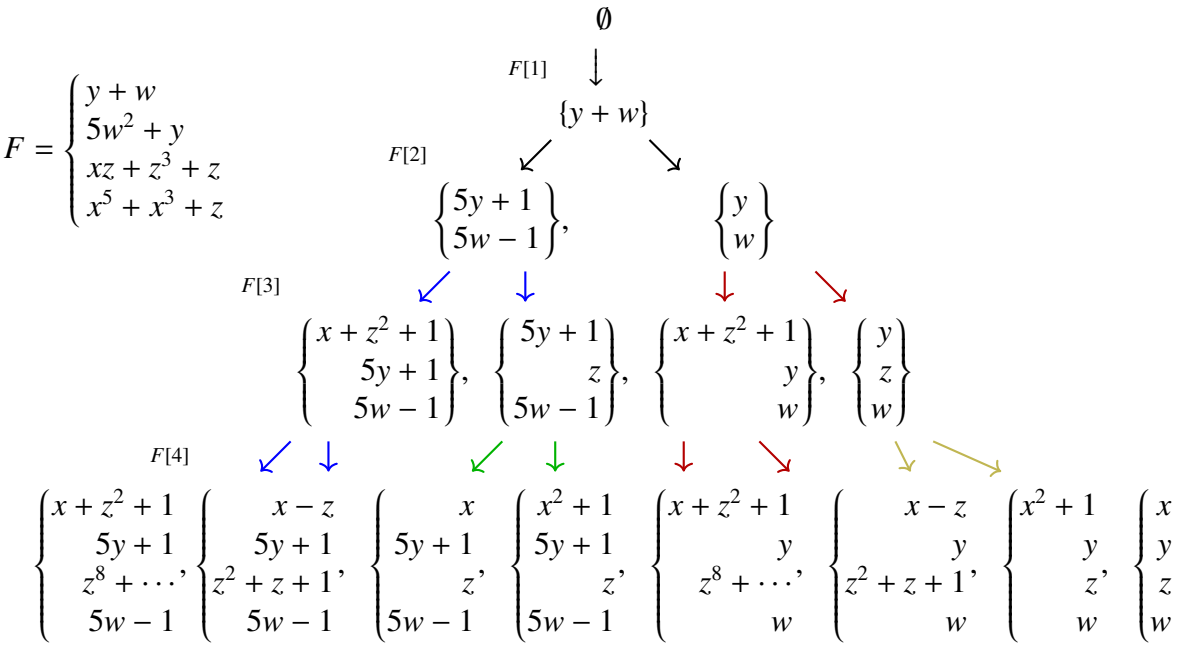

Figure 1.1: An illustration of incremental solving

```
> R := PolynomialRing([x, y, z, w]) : F := [y + w, 5*w^2 + y, x*z + z^3 + z, x^5 + x^3 + z];
                    F := [y + w, 5 w^2 + y, z^3 + z x + z, x^5 + x^3 + z]
> dec := Triangularize(F, R): map(Display, dec, R);
```

$$\left[\left\{\begin{array}{c} x+z^2+1=0 \\ 5y+1=0 \\ z^8-z^7+5z^6-4z^5+10z^4-6z^3+9z^2-3z+2=0 \\ 5w-1=0 \end{array}\right\}, \left\{\begin{array}{c} x+z^2+1=0 \\ y=0 \\ z^8-z^7+5z^6-4z^5+10z^4-6z^3+9z^2-3z+2=0 \\ w=0 \end{array}\right\}, \left\{\begin{array}{c} x-z=0 \\ 5y+1=0 \\ z^2+z+1=0 \\ 5w-1=0 \end{array}\right\}, \left\{\begin{array}{c} x-z=0 \\ y=0 \\ z^2+z+1=0 \\ w=0 \end{array}\right\}, \left\{\begin{array}{c} x^2+1=0 \\ 5y+1=0 \\ z=0 \\ 5w-1=0 \end{array}\right\}, \left\{\begin{array}{c} x^2+1=0 \\ y=0 \\ z=0 \\ w=0 \end{array}\right\}, \left\{\begin{array}{c} x=0 \\ 5y+1=0 \\ z=0 \\ 5w-1=0 \end{array}\right\}, \left\{\begin{array}{c} x=0 \\ y=0 \\ z=0 \\ w=0 \end{array}\right\}\right]$$

Figure 1.2: A MAPLE session with the `Triangularize` command

```
> R := PolynomialRing([x, y, z]) : F := [x^3 + y^2 - 3·y - 1, 5·y^4 - 3, -20·x + y - z];
                        F := [x^3 + y^2 - 3 y - 1, 5 y^4 - 3, -20 x + y - z]
> lrc := [Empty(R)] : lrc := [seq(op(Intersect(F[3], rc, R)), rc = lrc)] : Display(lrc, R);
                                  [20 x - y + z = 0]
> lrc := [seq(op(Intersect(F[2], rc, R)), rc = lrc)] : Display(lrc, R);
                                  [{ 20 x - y + z = 0 }]
                                  [{ 5 y^4 - 3 = 0    }]
> lrc := [seq(op(Intersect(F[1], rc, R)), rc = lrc)] : equ := Equations(lrc[1], R) :
> for p in equ do p = 0; end do;
                                  20 x - y + z = 0
(25 z^6 + 120000 z^5 - 1260000 z^4 - 960320000 z^3 + 13920000009 z^2 + 2879928000 z - 42239743964000) y
    - 15 z^7 - 20000 z^6 + 360000 z^5 - 60000 z^4 - 1760000015 z^3 + 2880252000 z^2 - 384216000 z
    - 12031424012000 = 0
125 z^12 + 4000000 z^9 - 225 z^8 - 14400000 z^7 + 38472000000 z^6 + 345628800000 z^5 - 1871999999865 z^4
    - 1280921614400000 z^3 + 15206376985920000 z^2 + 4331312644320000 z - 35859742679680000027
    = 0
```

Figure 1.3: An illustration of expression swell from dimensions 1 to 0

regular chain T looks like

$$T : \begin{cases} t_2(x_1, x_2) &= h_2(x_1)x_2^{d_2} + \cdots \\ t_3(x_1, x_2, x_3) &= h_3(x_1)x_3^{d_3} + \cdots \\ \vdots & \vdots \\ t_n(x_1, x_2, ..., x_n) &= h_n(x_1)x_n^{d_n} + \cdots , \end{cases} \tag{1.2}$$

where $t_2, t_3, \ldots, t_n$ have respective leading variables $x_2, x_3, \ldots, x_n$, with respective leading coefficient $h_2, h_3, \ldots, h_n$ and respective leading degrees $d_2, d_3, \ldots, d_n$. By definition, A point $z = (z_1, \ldots, z_n) \in \overline{\mathbb{K}}^n$, where $\overline{\mathbb{K}}$ is the algebraic closure of $\mathbb{K}$, belongs to the *quasi-component* $W(T)$ of T (that is, the solution set of T) if z belongs to $V(T)$ (that is, cancels $t_2, \ldots, t_n$) and z does not cancel any of $h_2, h_3, \ldots, h_n$. Therefore, the quasi-component $W(T)$ may not be closed in Zariski topology. Of course, the closure $\overline{W(T)}$ is necessarily contained in $V(F)$, since $V(F)$ is closed. Therefore, if $W(T)$ is not closed, there exist regular chains $T_{i_1}, T_{i_2}, \ldots$, with $2 \leq \cdots \leq i_1 < \cdots \leq i_2 \leq \cdots \leq v$ such that we have:

$$\overline{W(T)} \setminus W(T) \subseteq W(T_{i_1}) \cup W(T_{i_2}) \cup \cdots .$$

Because $\overline{W(T)} \setminus W(T)$ is necessarily zero-dimensional, the regular chains $T_{i_1}, T_{i_2}, \ldots$ are likely (but not necessarily) to be zero-dimensional too. Since, when solving a polynomial system, the computation of the zero-dimensional regular chains has often a dominant cost, it is natural to ask whether $\overline{W(T)} \setminus W(T)$ could be computed at some lower cost "directly" from $W(T)$ by means of analytical arguments.

A positive answer to this question was given in [3] by Alvandi, Chen and Moreno Maza. We summarize their construction. From now on, we assume that $\mathbb{K}$ is the field $\mathbb{C}$ of complex numbers. Let h be the product of the polynomials $h_2, h_3, \ldots, h_n$. Note that we have $h \in \mathbb{K}[x_1]$. Let $\zeta \in \mathbb{K}$ be a root of h. Since $W(T)$ is one-dimensional, the set T can be seen as a parametrization of a space curve C with x_1 as a parameter. It is natural to ask what are the limit points of C when x_1 approaches ζ. It turns out that the set of all the limit points obtained in this way (thus considering all roots ζ of h) is exactly $\overline{W(T)} \setminus W(T)$. As a byproduct, this explains why $\overline{W(T)} \setminus W(T)$

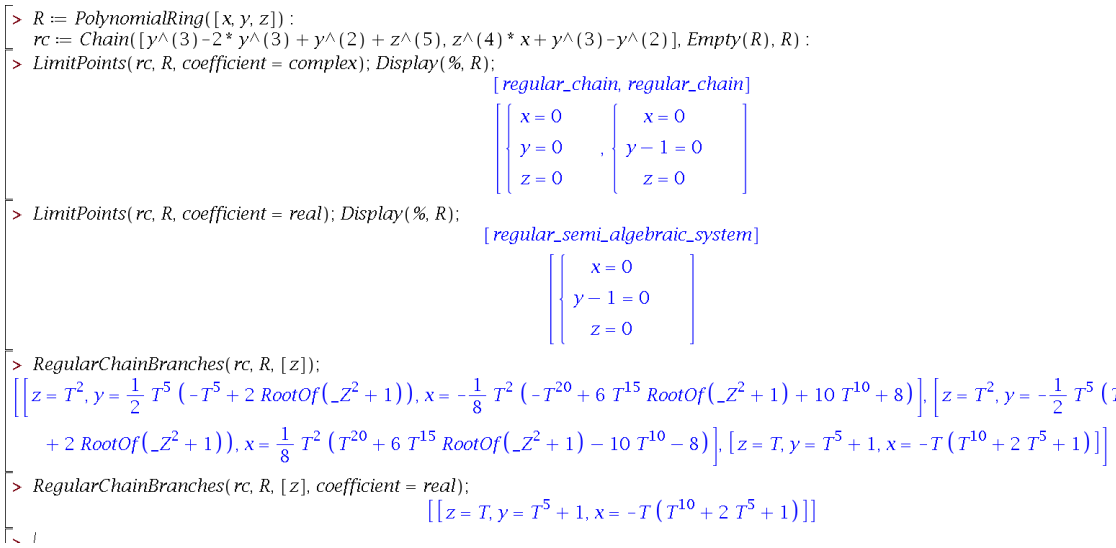

Figure 1.4: Illustration of MAPLE's `LimitPoints`

is a zero-dimensional algebraic set. Moreover, the limit points of C when x_1 approaches ζ can be computed as follows. For simplicity, assume $\zeta = 0$. Over the algebraically closed field $\mathbb{C}((x_1^*))$ of univariate Puiseux series in x_1, the univariate polynomial $t_2(x_1, x_2) \in \mathbb{C}((x_1^*))[x_2]$ has d_2 roots. Substituting each of those roots into $t_3(x_1, x_2, x_3)$ yields polynomials in $\mathbb{C}((x_1^*))[x_3]$, which in turn have d_3 roots, etc. Continuing in this manner, one can compute the solution set $\mathcal{V}(T)$ of T regarded as a zero-dimensional chain in $\mathbb{C}((x_1^*))[x_2, x_3, \ldots, x_n]$. In practice, the coordinates of those points of $\mathcal{V}(T)$ are computed up to a precision which is sufficient to separate those points from each other.

The authors of [3] have proved that the limit points of C when x_1 approaches ζ are obtained by evaluating at $x_1 = 0$ all the points $(s_2, \ldots, s_n)$ of $\mathcal{V}(T)$ so that each Puiseux series $s_2, \ldots, s_n$ has a non-negative order. The computation of limit points of one-dimensional regular chains was implemented in MAPLE by the authors of [3] as the command `LimitPoints` of the subpackage `AlgebraicGeometryTools` of the `RegularChains` library.

As a first example of limit point computation, consider the following regular chain $T \subseteq \mathbb{K}[x, y, z]$ for the variable ordering $z < y < x$:

$$T := \left\{ \begin{array}{l} z\,x - y^2 \\ y^5 - z^4 \end{array} \right. .$$

The set $\mathcal{V}(T)$ of the Puiseux series solutions of T consisting of the single point

$$(x = z^{3/5}, y = z^{4/5})$$

Clearly the Puiseux series $z^{3/5}$ and $z^{4/5}$ have non-negative orders and evaluating the above point at $z = 0$ yields $(0, 0)$. Thus we have:

$$\overline{W(T)} \setminus W(T) = \{(0, 0, 0)\}.$$

Consider now this other regular chain $T \subseteq \mathbb{K}[x, y, z]$ for the variable ordering $z < y < x$:

$$T := \left\{ \begin{array}{l} z\,x - y^2 = 0 \\ y^5 - z^2 = 0 \end{array} \right. .$$

The set $\mathcal{V}(T)$ of the Puiseux series solutions of T consisting of the single point

$$(x = z^{-1/5}, y = z^{2/5})$$

Since the Puiseux series $z^{-1/5}$ has a negative order, we have:

$$\overline{W(T)} \setminus W(T) = \emptyset.$$

Figure 1.4 illustrates how MAPLE computes the Puiseux series of a regular chain (with the command `RegularChainBranches`) and the corresponding limit points (with the command `LimiPoints`).

The above discussion shows that the computation of limit points of one-dimensional regular chains essentially boils down to solve univariate polynomials over a field of univariate Puiseux series. There is a number of algorithms for this latter task, in particular the so-called *Extended Hensel-Sasaki Construction* studied in [2] and implemented in MAPLE by Alvandi. The Extended Hensel-Sasaki Construction does make an explicit use of Puiseux series arithmetic and reduces computations to polynomial arithmetic. Other algorithms make explicit use of Puiseux series arithmetic and thus of Laurent series arithmetic. This is the case for the algorithm of Krzysztof Nowak [27]. Nowak's construction is a 2-way divide-and-conquer algorithm reducing the factorization of polynomials in $\mathbb{C}((x_1^*))[x_2]$ to Hensel lifting in $\mathbb{C}[[x_1]][x_2]$ by means Tschirnhaus transformation. Recent work with this type of algorithms, see [12] has lead to very successful implementation of Hensel lifting in $\mathbb{C}[[x_1, \ldots, x_{n-1}]][x_n]$ in both MAPLE and C/C++. This observation motivates the second objective of this dissertation: developing effective algorithms for multivariate Laurent series so as to factorize polynomials in $\mathbb{C}((x_1^*, \ldots, x_{n-1}^*))[x_n]$. Our contribution to this question is summarized in Section 1.2 and detailed in Chapter 3.

1.1 A modular method for the Intersect algorithm

To understand the challenges that the present document is addressing, let us describe loosely the algorithm underlying `Intersect`(f, T). Assume that f is a non-constant polynomial with main variable v and T contains a polynomial T_v with the same main variable v. Up to details that we shall ignore here, Intersect(f, T) proceeds as follows:

1. compute the subresultant chain of f and T_v w.r.t. v; this produces the resultant r of f and T_v w.r.t. v modulo the lower part of T (that is, the regular chain consisting of the polynomials in T with main variable less than v);
2. if $r = 0$ then replace T_v by $\gcd(f, T_v)$ (actually the GCD of f and T_v w.r.t. v modulo the lower part of T) in T, obtaining a new regular chain T' and returning $\{T'\}$;
3. if r is a non-zero constant, then $V(f) \cap W(T)$ is empty and the empty-list $\{\}$ is returned;
4. otherwise r is a non-constant polynomial, Intersect(r, T) is computed recursively, obtaining regular chains $T_1, \ldots, T_e$ and the union of Intersect$(f, T_1), \ldots,$ Intersect(f, T_e) is returned.

This process leads to computing iterated resultants which, in general, have "large extraneous" factors. By that, we mean factors that not only bring no useful information, but also create massive expression swell. The `Triangularize` algorithm manages, of course, to remove those extraneous factors, but this is an expensive task. Ideally, one would prefer not to compute those extraneous factors at all!

In case T is a one-dimensional regular chain (that is a regular chain with n variables and $n-1$ polynomials) it was proved by Chen and Moreno Maza in [16] that the iterated resultant involved in the computations of $\mathsf{Intersect}(f, T)$ can be computed efficiently as follows:

1. Let x_1 be the variable of T which is free, so that we can assume that T looks like

$$T : \begin{cases} t_2(x_1, x_2) &= h_2(x_1)x_2^{d_2} + \cdots \\ t_3(x_1, x_2, x_3) &= h_3(x_1)x_3^{d_3} + \cdots \\ \vdots \quad \vdots & \quad \vdots \\ t_n(x_1, x_2, ..., x_n) &= h_n(x_1)x_n^{d_n} + \cdots , \end{cases} \tag{1.3}$$

   where $t_2, t_3, \ldots, t_n$ have respective main variables $x_2, x_3, \ldots, x_n$, with respective initials $h_2, h_3, \ldots, h_n$ and respective main degrees $d_2, d_3, \ldots, d_n$;
2. Let D be the Bézout bound of the input system (assumed to be zero-dimensional) from which T was obtained;
3. Choose $m := (2D + 1)$ values $v_1, \ldots, v_m$ for x_1 so that for each $v \in \{v_1, \ldots, v_m\}$ we have $h_2(v) \neq 0, h_3(v) \neq 0, \ldots, h_n(v) \neq 0$ so that the specialized triangular set $T_{x_1=v}$ remains a regular chain;
4. For each $v \in \{v_1, \ldots, v_m\}$, let N_v be the reduced lexicographical Gröbner basis of the ideal generated by $T_{x_1=v}$ (note that N_v is a regular chain where each initial is equal to 1) and Let f_v be the polynomial f specialized at $x_1 = v$;
5. For each $v \in \{v_1, \ldots, v_m\}$ compute r_v the iterated resultant of f_v and N_v;
6. Using rational function interpolation, interpolate the points $(v_1, r_{v_1}), \ldots, (v_m, r_{v_m})$ which yields a univariate rational function R in x_1; and
7. Return the numerator of R.

We stress the fact that the above procedure only computes the iterated resultant $\mathrm{res}(f, T)$ (see Section 2.1 for a definition) of the polynomial f w.r.t. T. In other words, this procedure does not compute $\mathsf{Intersect}(f, T)$. We also note that Section 6 in [16] explains with details why the above procedure is a solution to controlling expression swell in the computations of iterated resultants. It follows that one can derive from that procedure an algorithmic solution to control expression swell when computing $\mathsf{Intersect}(f, T)$ by means of subresultant theory.

The goal of the present document is actually to enhance the above procedure so that it computes $\mathsf{Intersect}(f, T)$ while avoiding extraneous factors. There is, however, a major obstacle to overcome in order to implement this enhancement: while $\mathrm{res}(f, T)$ does not need to split the computations into different cases, splitting may be needed for computing $\mathsf{Intersect}(f, T)$. Moreover the *evaluation-interpolation* scheme of the above procedure makes this obstacle even harder to overcome. Indeed, given two different values v_1 and v_2 at which x_1 can be specialized, it is possible that $\mathsf{Intersect}(f_{x_1=v_1}, T_{x_1=v_1})$ splits computations differently than $\mathsf{Intersect}(f_{x_1=v_2}, T_{x_1=v_2})$ does. Thanks to the concept of *equiprojectable decomposition* introduced in [18], there is a solution to this challenge and we shall present our solution in a forthcoming paper.

In fact, towards that solution, we first need to handle the "non-splitting case". By that, we mean the case where none of $\mathsf{Intersect}(f_{x_1=v}, T_{x_1=v})$, for each $v \in \{v_1, \ldots, v_m\}$, needs to split. But here comes another challenge: before computing $\mathsf{Intersect}(f, T)$, we may not know in advance whether splitting will or will not be needed.

The present document addresses that second challenge and offers an algorithm computing

Intersect(f, T) by means of an evaluation-interpolation scheme. To keep the presentation simple, we explain this algorithm in the context of trivariate polynomials. We believe that the expert reader will understand how our results can be extended to polynomials with an arbitrary number of variables; we shall present this extension in the forthcoming paper mentioned above.

With this trivariate presentation, we have three polynomials $f, t, b \in \mathbb{K}[x, y, z]$ so that $T := \{t, b\}$ is a regular chain and we aim at computing Intersect$(f, \{t, b\})$. In Section 2.2, we exhibit conditions on f, t, b so that Intersect$(f, \{t, b\})$ returns a single zero-dimensional regular chain C; see Theorem 1.

These conditions can be understood as *genericity assumptions*. Those are slightly less restrictive than those of the *Shape Lemma* [9] and slightly more restrictive than equiprojectability [18]. The advantage of those conditions (namely Hypotheses 1, 2, 3 and 4) is that they can be tested during the execution of the proposed algorithm for computing Intersect$(f, \{t, b\})$. As a result, this algorithm, which is presented in Section 2.3, returns either the regular chain C of Theorem 1 or an error message indicating which genericity assumption does not hold. In that latter case, Intersect$(f, \{t, b\})$ can be computed by the general algorithm presented in [16]. Therefore, the proposed algorithm for computing Intersect$(f, \{t, b\})$ can be seen as an optimization of the general algorithm. And, indeed, the experimentation reported in Section 2.6 shows that this is the case in practice. More comments on this appear in Section 4.

The information presented in Chapter 2 has been accepted for publication in the MAPLE transactions journal (see [14]).

1.2 Algorithms for arithmetic operations on multivariate Laurent series

A Laurent series is a generalization of a power series in which negative degrees are allowed. In this document, we follow the ideas exposed by Monforte and Kauers in [25]. Our objective is to report on a first implementation of multivariate Laurent series inside of MAPLE. In order to accomplish this goal, we make use of the already existing `MultitivariatePowerSeries` package (see [5]).

To reach our goal, we first need to select an additive total order. Thus, we choose the well-known *graded reverse lexicographic order* or *grevlex order* for short. The grevlex order is said to be *compatible with a cone* $C \subseteq \mathbb{R}^p$ if $\mathbf{0} \preceq \mathbf{k}$ holds, for all $\mathbf{k} \in C \cap \mathbb{Z}^p$. It can be proved that a cone C is compatible with the grevlex order, if and only if C is generated by a set of non-negative grevlex rays.

After having selected a total order, the basic implementation strategy is to define any multivariate Laurent series f as the composition of a multivariate power series g and an affine transformation, given by a point $\mathbf{e}$ and a set of non-negative grevlex rays $\mathbf{R} = \{\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_m\}$, mapping the exponent vectors of g to those of f. The properties of that affine transformation guarantee that the set formed by the exponent vectors of the non-zero monomials (that is, the support) in our Laurent series f are contained inside of the cone, C, generated by $\mathbf{R}$. Monforte and Kauers have shown that the set of multivariate Laurent series with support contained inside a cone C, denoted by $\mathbb{K}_C[[\mathbf{x}]]$, form an integral domain. Furthermore, they have proved that

the sets

$$\mathbb{K}_{\leq}[[\mathbf{x}]] := \bigcup_{C\in\mathcal{C}} \mathbb{K}_C[[\mathbf{x}]] \quad \text{and} \quad \mathbb{K}_{\leq}((\mathbf{x})) := \bigcup_{\mathbf{e}\in\mathbb{Z}^p} \mathbf{x}^{\mathbf{e}}\mathbb{K}_{\leq}((\mathbf{x})),$$

are respectively a ring and a field, where $\mathcal{C}$ is the set of all cones $C \subseteq \mathbb{R}^p$ which are compatible with an additive order, and

$$\mathbf{x}^{\mathbf{e}}\mathbb{K}_{\leq}((\mathbf{x})) := \bigcup_{C\in\mathcal{C}} \mathbf{x}^{\mathbf{e}}\mathbb{K}_C[[\mathbf{x}]].$$

Laurent series with support contained inside the same cone can be added or multiplied easily. Thus, our first challenge is as follows: given two cones C_1 and C_2 compatible with the grevlex order, how can we add and multiply elements inside of $\mathbb{K}_{C_1}[[\mathbf{x}]]$ with elements in $\mathbb{K}_{C_2}[[\mathbf{x}]]$? The first step to solve this is to find a cone C, compatible with the grevlex order and containing the cones C_1 and C_2. The easiest choice would be to take C as $C_1 + C_2$. This, however, would imply that generating set of C would be equal to the union of the generating set of C_1 and the generating set of C_2. For computational efficiency reasons, we would like to avoid this inflation in the number of generators. Hence, we develop, in Chapter 3 Section 3.3, Algorithm 9. This algorithm takes two or three generating sets and returns a minimal generating set, such that its generated cone contains the cones generated by the input generating sets. By applying this algorithm before adding and multiplying, we guarantee that the generating set of the result is small.

We also observe that the inversion of multivariate Laurent series is hard to compute. Since, we are using multivariate power series as part of our construction, we need to take them into account. It is known that, every power series is invertible if and only it has a non-zero constant term. Thus, inverting Laurent series with an internal power series with a non-zero constant is not a challenge. Now, if a power series does not have a non-zero constant term, how do we invert a Laurent series associated with it? Following Monforte and Kauers ideas, we need to look for the minimum element inside the support of our Laurent series, and then factor it out the respective monomials in our Laurent series. This, however, causes a great computational problem. How do we find the minimum element of a possibly infinite set? As it is stated in Chapter 3 Section 3.3, this is not always possible. Furthermore, this depends greatly of the generating set.

If the generating set of a Laurent series contains at least a vector with the sum of its components equal to zero (a so-called zero-ray), we show that it is not always possible to to find the minimum element of the support. Thus, our solution for the multiplicative inverse presented in Chapter 3 is only a partial solution. Nevertheless, we have divided the problem into various cases for which we are able to guarantee that the inverse of a Laurent series has been found even when we have a zero-ray. For instance, if a Laurent series has an associate power series with a fraction of polynomials as its analytic expression, then standard calculations produce the inverse of that power series.

Chapter 2

A modular approach for the Intersect algorithm

In this chapter, we develop the necessary concepts to understand our modular approach for computing the intersection between a 1-dimensional regular chain and a polynomial. We explain first the basic concepts of the regular chains theory, and also we make an introduction of the Algorithm Intersect (Section 2.1). Following, we explain a non-modular method for solving our problem (Section 2.2). Next, we detail the modular method and the necessary conditions for our algorithm to work correctly (Section 2.3). After this, in Sections 2.4 and 2.5, we can find information about our MAPLE and C implementations of the modular algorithm. Lastly, Section 2.6 contains the collected data during our experimentation.

2.1 Preliminaries

This section is a short review of concepts from the theory of regular chains and triangular decompositions of polynomial systems. Details can be found in [16]. This document also relies on the theory of subresultants and we refer the unfamiliar reader to the concise preliminaries section of [6].

Throughout this chapter, let $\mathbb{K}$ be a perfect field, $\overline{\mathbb{K}}$ be its algebraic closure, and $\mathbb{K}[X]$ be the polynomial ring over $\mathbb{K}$ with n ordered variables $X = X_1 < \cdots < X_n$. Let $p \in \mathbb{K}[X]$. Assume that $p \notin \mathbb{K}$ holds. Denote by $\mathrm{mvar}(p)$, $\mathrm{init}(p)$, $\mathrm{mdeg}(p)$, and $\mathrm{tail}(p)$, respectively, the greatest variable appearing in p (called the *main variable* of p), the leading coefficient of p w.r.t. $\mathrm{mvar}(p)$ (called the *initial* of p), the degree of p w.r.t. $\mathrm{mvar}(p)$ (called the *main degree* of p) and the reductum of p w.r.t. $\mathrm{mvar}(p)$ (called the *tail* of p). For $F \subseteq \mathbb{K}[X]$, we denote by $\langle F \rangle$ and $V(F)$ the ideal generated by F in $\mathbb{K}[X]$ and the algebraic set of $\overline{\mathbb{K}}^n$ consisting of the common roots of the polynomials of F, respectively.

2.1.1 Triangular set

Let $T \subseteq \mathbb{K}[X]$ be a *triangular set*, that is, a set of non-constant polynomials with pairwise distinct main variables. Denote by $\mathrm{mvar}(T)$ the set of main variables of the polynomials in T. A variable $v \in X$ is called *algebraic* w.r.t. T if $v \in \mathrm{mvar}(T)$, otherwise it is said *free*

w.r.t. T. For $v \in \mathrm{mvar}(T)$, we denote by T_v and T_v^- (resp. T_v^+) the polynomial $f \in T$ with $\mathrm{mvar}(f) = v$ and the polynomials $f \in T$ with $\mathrm{mvar}(f) < v$ (resp. $\mathrm{mvar}(f) > v$). Let h_T be the product of the initials of the polynomials of T. We denote by $\mathrm{sat}(T)$ the *saturated ideal* of T: if $T = \emptyset$ holds, then $\mathrm{sat}(T)$ is defined as the trivial ideal $\langle 0 \rangle$, otherwise it is the ideal $\langle T \rangle : h_T^\infty$. The *quasi-component* $W(T)$ of T is defined as $V(T) \setminus V(h_T)$. For $f \in \mathbb{K}[X]$, we define $Z(f, T) := V(f) \cap W(T)$. The Zariski closure of $W(T)$ in $\overline{\mathbb{K}}^n$, denoted by $\overline{W(T)}$, is the intersection of all algebraic sets $V \subseteq \overline{\mathbb{K}}^n$ such that $W(T) \subseteq V$ holds; moreover we have $\overline{W(T)} = V(\mathrm{sat}(T))$. For $f \in \mathbb{K}[X]$, we denote by $\mathrm{res}(f, T)$ the *iterated resultant* of f w.r.t. T, that is: if $f \in \mathbb{K}$ or $T = \emptyset$ then f itself, else $\mathrm{res}(\mathrm{res}(f, T_v, v), T_v^-)$ if $v \in \mathrm{mvar}(T)$ and $v = \mathrm{mvar}(f)$ hold, or $\mathrm{res}(f, T_v^-)$ otherwise.

2.1.2 Regular chain

A triangular set $T \subseteq \mathbb{K}[X]$ is a *regular chain* if either T is empty, or letting v be the largest variable occurring in T, the set T_v^- is a regular chain, and the initial of T_v is regular (that is, neither zero nor zero divisor) modulo $\mathrm{sat}(T_v^-)$. Let $H \subseteq \mathbb{K}[X]$. The pair $[T, H]$ is a *regular system* if each polynomial in H is regular modulo $\mathrm{sat}(T)$. If H consists of a single polynomial h, then we also write $[T, h]$, for short, instead of $[T, H]$. The *dimension* of T, denoted by $\dim(T)$, is by definition the dimension of its saturated ideal and, as a property, equals $n - |T|$, where $|T|$ is the number of elements of T. If T has dimension zero, then T generates $\mathrm{sat}(T)$ and we have $V(T) = W(T)$.

A regular chain T, or a regular system $[T, H]$, is *square-free* if for all $t \in T$, the polynomial $\mathrm{der}(t)$ is regular w.r.t. $\mathrm{sat}(T)$, where $\mathrm{der}(t) = \frac{\partial t}{\partial v}$ and $v = \mathrm{mvar}(t)$. When the regular chain T is *square-free*, then the ideal $\mathrm{sat}(T)$ is radical.

The saturated ideal $\mathrm{sat}(T)$ of the regular chain T enjoys important properties, in particular the following, proved in [10]. Let $U_1, \ldots, U_d$ be all the free variables of T. Then $\mathrm{sat}(T)$ is unmixed of dimension d. Moreover, we have $\mathrm{sat}(T) \cap \mathbb{K}[U_1, \ldots, U_d] = \langle 0 \rangle$. Another property is the fact that a polynomial p belongs to $\mathrm{sat}(T)$ if and only if p reduces to 0 by pseudo-division w.r.t T, see [7]. Last but not least, a polynomial p is regular modulo $\mathrm{sat}(T)$ if and only if we have $\mathrm{res}(f, T) \neq 0$.

2.1.3 Normalized regular chain

The regular chain $T \subseteq \mathbb{K}[X]$ is said to be *normalized* if for every $v \in \mathrm{mvar}(T)$, none of the variables occurring in $\mathrm{init}(T_v)$ is algebraic w.r.t. T_v^-. Denote by d the dimension of T. Let Y and $U = U_1, \ldots, U_d$ stand respectively for $\mathrm{mvar}(T)$ and $X \setminus Y$. Then, T normalized means that for every $t \in T$ we have $\mathrm{init}(t) \in \mathbb{K}[U]$. It follows that if T is normalized, then T is a lexicographical Gröbner basis of the ideal that T generates in $(\mathbb{K}[U])[Y]$ (that is, over the field $(\mathbb{K}[U])$ of rational functions), and we denote by $\mathrm{nf}(p, T)$ the normal form of a polynomial $p \in (\mathbb{K}[U])[Y]$ w.r.t. T as a Gröbner basis. In particular, if T is normalized and has dimension zero, then for every $t \in T$ we have $\mathrm{init}(t) \in \mathbb{K}$.

2.1.4 Regular GCD

Let i be an integer with $1 \leq i \leq n$, let $T \subseteq \mathbb{K}[X]$ be a regular chain, let $p, t \in \mathbb{K}[X] \setminus \mathbb{K}$ be polynomials with the same main variable X_i, and $g \in \mathbb{K}$ or $g \in \mathbb{K}[X]$ with $\mathrm{mvar}(g) \leq X_i$. Assume that

1. $X_i > X_j$ holds for all $X_j \in \mathrm{mvar}(T)$; and
2. both $\mathrm{init}(p)$ and $\mathrm{init}(t)$ are regular w.r.t. $\mathrm{sat}(T)$.

Denote by $\mathcal{A}$ the total ring of fractions of the residue class ring $\mathbb{K}[X_1, \ldots, X_{i-1}]/\sqrt{\mathrm{sat}(T)}$. Note that $\mathcal{A}$ is isomorphic to a direct product of fields. We say that g is a *regular GCD* of p, t w.r.t. T whenever the following conditions hold:

(G_1) the leading coefficient of g in X_i is invertible in $\mathcal{A}$;
(G_2) g belongs to the ideal generated by p and t in $\mathcal{A}[X_i]$; and
(G_3) if $\deg(g, X_i) > 0$, then g divides both p and t in $\mathcal{A}[X_i]$, that is, both $\mathrm{prem}(p, g)$ and $\mathrm{prem}(t, g)$ belong to $\sqrt{\mathrm{sat}(T)}$.

When Conditions (G_1), (G_2), (G_3) and $\deg(g, X_i) > 0$ hold, we have:

(G_4) if $\mathrm{mdeg}(g) = \mathrm{mdeg}(t)$, then $\sqrt{\mathrm{sat}(T \cup t)} = \sqrt{\mathrm{sat}(T \cup g)}$ and $W(T \cup t) \subseteq Z(h_g, T \cup t) \cup W(T \cup g) \subseteq \overline{W(T \cup t)}$ both hold;
(G_5) if $\mathrm{mdeg}(g) < \mathrm{mdeg}(t)$, let $q = \mathrm{pquo}(t, g)$, then $T \cup q$ is a regular chain and the following two relations
  (a) $\sqrt{\mathrm{sat}(T \cup t)} = \sqrt{\mathrm{sat}(T \cup g)} \cap \sqrt{\mathrm{sat}(T \cup q)}$,
  (b) $W(T \cup t) \subseteq Z(h_g, T \cup t) \cup W(T \cup g) \cup W(T \cup q) \subseteq \overline{W(T \cup t)}$,
  hold;
(G_6) $W(T \cup g) \subseteq V(p)$; and
(G_7) $V(p) \cap W(T \cup t) \subseteq W(T \cup g) \cup V(p, h_g) \cap W(T \cup t) \subseteq V(p) \cap \overline{W(T \cup t)}$.

2.1.5 The algorithms Intersect and Regularize

Let $p \in \mathbb{K}[X]$ and let $T \subseteq \mathbb{K}[X]$ be a regular chain. The function call $\mathsf{Intersect}(p, T)$ computes regular chains $T_1, \ldots, T_e \subseteq \mathbb{K}[X]$ such that:

$$V(p) \cap W(T) \subseteq W(T_1) \cup \cdots \cup W(T_e) \subseteq V(p) \cap \overline{W(T)}. \tag{2.1}$$

The function call $\mathsf{Regularize}(p, T)$ computes a set of regular chains $\{T_1, \ldots, T_e\}$ such that:

1. for each $i = 1, \ldots, e$, either $p \in \mathrm{sat}(T_i)$ holds or p is regular w.r.t. $\mathrm{sat}(T_i)$;
2. we have $\overline{W(T)} = \overline{W(T_1)} \cup \cdots \cup \overline{W(T_e)}$, and $\mathrm{mvar}(T) = \mathrm{mvar}(T_i)$ holds for each $i = 1, \ldots, e$.

2.1.6 Triangular decomposition

Let $F \subseteq \mathbb{K}[X]$. Regular chains $T_1, \ldots, T_e$ of $\mathbb{K}[X]$ form a *triangular decomposition* of $V(F)$ in the sense of Kalkbrener (resp. Wu and Lazard) whenever we have $V(F) = \cup_{i=1}^{e} \overline{W(T_i)}$ (resp. $V(F) = \cup_{i=1}^{e} W(T_i)$). Hence, a triangular decomposition of $V(F)$ in the sense of Wu and Lazard is necessarily a triangular decomposition of $V(F)$ in the sense of Kalkbrener, while the converse is not true.

2.1.7 Specialization and border polynomial

Let $[T, H]$ be a regular system (resp. square-free regular system) of $\mathbb{K}[X]$. Recall that Y and $U = U_1, \ldots, U_d$ stand respectively for $\mathrm{mvar}(T)$ and $X \setminus Y$. Let $a = (a_1, \ldots, a_d)$ be a point of $\overline{\mathbb{K}}^d$. We say that $[T, H]$ *specializes well* at a if:

(*i*) for each $t \in T$ the polynomial $\mathrm{init}(t)$ is not zero modulo the ideal $\langle U_1 - a_1, \ldots, U_d - a_d \rangle$; and

(*ii*) the image of $[T, H]$ modulo $\langle U_1 - a_1, \ldots, U_d - a_d \rangle$ is a regular system (resp. square-free regular system).

Let $B_{T,H}$ be the primitive and square-free part of the product of all $\mathrm{res}(h, T)$ (resp. $\mathrm{res}(\mathrm{der}(t), T)$, $\mathrm{res}(h, T)$) for $h \in H \cup \{h_T\}$ and $t \in T$. We call $B_{T,H}$ the *border polynomial* of $[T, H]$. Proposition 1 follows from the specialization property of sub-resultants and states a fundamental property of border polynomials [23].

Proposition 1. *The system $[T, H]$ specializes well at $a \in \overline{\mathbb{K}}^d$ if and only if the border polynomial satisfies $B_{T,H}(a) \neq 0$.*

2.2 The non-modular method and its genericity assumptions

Let $\mathbb{K}$ be a field of characteristic zero or a prime field of *sufficiently large* characteristic, where that latter condition will be specified later. Let $f, t, b \in \mathbb{K}[x, y, z]$ be non-constant polynomials in the ordered variables $x > y > z$. Assume that $T := \{t, b\}$ is a regular chain with $\mathrm{mvar}(t) = x$ and $\mathrm{mvar}(b) = y$.

Our goal is to compute the intersection $V(f) \cap W(T)$ in the sense of the function call $\mathsf{Intersect}(f, T)$, as specified in Section 2.1. That is, we want to compute regular chains $T_1, \ldots, T_e \subseteq \mathbb{K}[x, y, z]$ such that we have:

$$V(f) \cap W(T) \subseteq W(T_1) \cup \cdots \cup W(T_e) \subseteq V(f) \cap \overline{W(T)}. \tag{2.2}$$

We shall show that under some assumptions, one can compute a single regular chain $C \subseteq \mathbb{K}[x, y, z]$ so that C is zero-dimensional and we have

$$V(f) \cap W(T) = W(C). \tag{2.3}$$

The first of those assumptions makes the problem truly trivariate as follows.

Hypothesis 1.

$$\mathrm{mvar}(f) = x. \tag{2.4}$$

Let $S(t, f, x)$ be the subresultant chain of t and f (resp. f and t) regarded as polynomials in $(\mathbb{K}[y, z])[x]$ if $\mathrm{mdeg}(t) \geq \mathrm{mdeg}(f)$ (resp. $\mathrm{mdeg}(t) < \mathrm{mdeg}(f)$). Let $S_0(t, f, x)$ and $S_1(t, f, x)$ be the subresultants of index 0 and 1 from $S(t, f, x)$. We let

$$r := S_0(t, f, x) \text{ and } \ell := S_1(t, f, x). \tag{2.5}$$

To make the problem generic, we assume the following:

Hypothesis 2.

$$r \notin \mathbb{K} \text{ and } \operatorname{mvar}(r) = y. \tag{2.6}$$

Let $S(r, b, y)$ be the subresultant chain of r and b (resp. b and r) regarded as polynomials in $(\mathbb{K}[z])[y]$ if $\operatorname{mdeg}(r) \geq \operatorname{mdeg}(b)$ (resp. $\operatorname{mdeg}(r) < \operatorname{mdeg}(b)$). Let $S_0(r, b, y)$ and $S_1(r, b, y)$ be the subresultants of index 0 and 1 from $S(r, b, y)$. We let

$$s := S_0(r, b, y) \text{ and } g := S_1(r, b, y). \tag{2.7}$$

We denote by $\overline{s}$ the squarefree part of s, that is, $s/\gcd(s, \operatorname{der}(s))$.

Continuing in making the problem generic, we assume the following:

Hypothesis 3. *The polynomial set $C := \{\overline{s}, g, \ell\}$ is a regular chain.*

Hypothesis 3 has a number of consequences which, essentially, rephrase the fact that C is a regular chain. Proposition 2 gathers those consequences. Building on that, Proposition 3 yields Equation (2.8) which plays a key role in our method for computing $\mathsf{Intersect}(f, T)$.

Proposition 2. *The polynomials $\overline{s}, g, \ell$ are non-constant with respective main variables z, y, x. Moreover, the initial h_g is invertible modulo $\overline{s}$ and the initial h_ℓ is invertible modulo the ideal $\langle \overline{s}, g \rangle$ generated by s and g in $\mathbb{K}[y, z]$.*

Proposition 3. *The polynomial g is a regular GCD of r and b modulo the regular chain $\{\overline{s}\}$. Moreover, we have:*

$$V(\overline{s}, r, b) = V(\overline{s}, g). \tag{2.8}$$

Proof. We first prove that g is a regular GCD of r and b modulo the regular chain $\{\overline{s}\}$. Since $\{g, \overline{s}\}$ is a regular chain, Property (G_1) of a regular GCD clearly holds. We prove (G_2). Subresultant theory tells us that there exist polynomials $u, v \in \mathbb{K}[y, z]$ so that we have:

$$ur + vb = g. \tag{2.9}$$

Let $\mathcal{A}$ be the total ring of fractions of the residue class ring $\mathbb{K}[z]/\langle \overline{s} \rangle$. Since $\overline{s}$ is squarefree, the ring $\mathcal{A}$ is actually a direct product of fields and Equation (2.9) tells us that g is the GCD (in the sense of a Euclidean domain) of r and b over each of those fields. Therefore, Property (G_2) holds. In particular, both r and b belong to the ideal generated by g in $\mathcal{A}[y]$. Thus, there exist polynomials $q_r, q_b \in \mathcal{A}[y]$ so that the following hold in $\mathcal{A}[y]$:

$$r = q_r g \quad \text{and} \quad b = q_b g. \tag{2.10}$$

Every polynomial $p \in \mathcal{A}[y]$ can be written as the fraction of a polynomial $n \in \mathbb{K}[y, z]$ over a polynomial $d \in \mathbb{K}[z]$ so that d is invertible modulo $\overline{s}$. Therefore, there exist polynomials in $\mathbb{K}[y, z]$, that we denote again q_r, q_b for convenience, so that the following hold in $\mathbb{K}[z, y]$:

$$r \equiv q_r g \bmod \overline{s} \quad \text{and} \quad b \equiv q_b g \bmod \overline{s}. \tag{2.11}$$

From the above, it is clear that g pseudo-divides (actually divides) both r and b modulo $\overline{s}$. Therefore, Property (G_3) holds and we have proved that g is a regular GCD of r and b modulo the regular chain $\{\overline{s}\}$. The second claim of this proposition follows from the first one and Lemma 1. □

Lemma 1. *Let $\hat{g} \in \mathbb{K}[y, z]$ be a non-constant polynomial with* $\mathrm{mvar}(\hat{g}) = y$*. Assume that $\hat{g}$ is a regular GCD of r and b modulo the regular chain $\{\overline{s}\}$. Then, we have:*

$$V(\overline{s}, r, b) = V(\overline{s}, \hat{g}). \tag{2.12}$$

Proof. It follows from Property (G_7) that we have:

$$V(r) \cap W(\{\overline{s}, b\}) \;\subseteq\; W(\{\overline{s}, \hat{g}\}) \;\cup\; V(r, h_{\hat{g}}) \cap W(\{\overline{s}, b\}) \subseteq V(r) \cap \overline{W(\{\overline{s}, b\})}, \tag{2.13}$$

where $h_{\hat{g}}$ is the initial of $\hat{g}$. Since $h_{\hat{g}}$ is invertible modulo $\overline{s}$ we observe that we have:

$$V(r, h_{\hat{g}}) \cap W(\{\overline{s}, b\}) = \emptyset. \tag{2.14}$$

In addition, since $V(\{\overline{s}, b\})$ and $V(\{\overline{s}, \hat{g}\})$ are both zero-dimensional we have:

$$V(\overline{s}, b) = W(\{\overline{s}, b\}) = \overline{W(\{\overline{s}, b\})} \;\text{ and }\; V(\overline{s}, \hat{g}) = W(\{\overline{s}, \hat{g}\}). \tag{2.15}$$

Therefore, Relation (2.13) simply becomes:

$$V(r, \overline{s}, b) \;\subseteq\; V(\overline{s}, \hat{g}) \;\subseteq\; V(r, \overline{s}, b). \tag{2.16}$$

This completes the proof. □

Our last genericity assumption is the following:

Hypothesis 4. *The initial of the polynomial t is invertible modulo the ideal* $\mathrm{sat}(\{\overline{s}, g\}) = \langle \overline{s}, g \rangle$.

Proposition 4 yields Equation (2.17) which is another key ingredient in our method for computing $\mathsf{Intersect}(f, T)$.

Proposition 4. *The polynomial ℓ is a regular GCD of f and t modulo the regular chain $\{\overline{s}, g\}$. Moreover, we have:*

$$V(f, \overline{s}, g, t) \;=\; V(\{\overline{s}, g, \ell\}). \tag{2.17}$$

Proof. We first observe that $\mathbb{K}[z, y]/\langle \overline{s}, g \rangle$ is a direct product of fields. Indeed, $\overline{s}$ is squarefree and $\deg(g, y) = 1$. Therefore, proving that ℓ is a regular GCD of f and t modulo the regular chain $\{\overline{s}, g\}$ is done similarly to the proof that g is a regular GCD of r and b modulo the regular chain $\{\overline{s}\}$ in the proof of Proposition 3. The second claim of this proposition follows from the first one and Lemma 2. □

Lemma 2. *Let $\hat{\ell} \in \mathbb{K}[x, y, z]$ be a non-constant polynomial with* $\mathrm{mvar}(\hat{\ell}) = x$*. Assume that $\hat{\ell}$ is a regular GCD of f and t modulo the regular chain $\{\overline{s}, g\}$. Then, we have:*

$$V(f, \overline{s}, g, t) \;=\; V(\{\overline{s}, g, \hat{\ell}\}. \tag{2.18}$$

Proof. It follows from Property (G_7) that we have:

$$V(f) \cap W(\{\overline{s}, g, t\}) \;\subseteq\; W(\{\overline{s}, g, \hat{\ell}\}) \;\cup\; V(f, h_{\hat{\ell}}) \cap W(\{\overline{s}, g, t\}) \subseteq V(f) \cap \overline{W(\{\overline{s}, g, t\})}, \tag{2.19}$$

where $h_{\hat{\ell}}$ is the initial of $\hat{\ell}$. Since $h_{\hat{\ell}}$ is invertible modulo $\langle \overline{s}, g \rangle$, we observe that we have:

$$V(f, h_{\hat{\ell}}) \cap W(\{\overline{s}, g, t\}) = \emptyset. \tag{2.20}$$

In addition, since $V(\{\overline{s}, b\})$ and $V(\{\overline{s}, b\})$ are both zero-dimensional we have:

$$V(\overline{s}, g, t) = W(\{\overline{s}, g, t\}) = \overline{W(\{\overline{s}, g, t\})} \text{ and } V(\overline{s}, g, \hat{\ell}) = W(\{\overline{s}, g, \hat{\ell}\}). \tag{2.21}$$

Therefore, Relation (2.19) simply becomes:

$$V(f, \overline{s}, g, t) \subseteq V(\overline{s}, g, \hat{\ell}) \subseteq V(f, \overline{s}, g, t). \tag{2.22}$$

This completes the proof. □

Theorem 1 tells us that, under our genericity assumptions, $\mathsf{Intersect}(f, \{t, b\})$ is simply given by the regular chain $C = \{\overline{s}, g, \ell\}$.

Theorem 1. *With our four Hypotheses 1, 2, 3 and 4, we have:*

$$V(f, t, b) = V(\overline{s}, g, \ell). \tag{2.23}$$

Proof. This follows immediately from Proposition 3, Proposition 4 and Lemma 3. □

Lemma 3. *Let $\hat{g} \in \mathbb{K}[y, z]$ be a non-constant polynomial with $\mathrm{mvar}(\hat{g}) = y$ and let $\hat{\ell} \in \mathbb{K}[x, y, z]$ be a non-constant polynomial with $\mathrm{mvar}(\hat{\ell}) = x$. Assume that $\hat{g}$ is a regular GCD of r and b modulo the regular chain $\{\overline{s}\}$. Assume also that $\hat{\ell}$ is a regular GCD of f and t modulo the regular chain $\{\overline{s}, \hat{g}\}$. Then, we have:*

$$V(f, t, b) = V(\overline{s}, \hat{g}, \hat{\ell}). \tag{2.24}$$

Proof. Since r belongs to the ideal generated by f and t in in $\mathbb{K}[x, y, z]$, we have:

$$V(f, t, b) = V(f, t, b, r). \tag{2.25}$$

Since s belongs to the ideal generated by r and b in in $\mathbb{K}[x, y, z]$ we have:

$$V(f, t, b) = V(f, t, b, r, s). \tag{2.26}$$

Since $\overline{s}$ is the squarefree part of s, we also have:

$$V(f, t, b) = V(f, t, b, r, \overline{s}). \tag{2.27}$$

With Lemma 1, we deduce:

$$V(f, t, b) = V(f, t, \hat{g}, \overline{s}). \tag{2.28}$$

Finally, with Lemma 2, we deduce

$$V(f, t, b) = V(\hat{\ell}, \hat{g}, \overline{s}). \tag{2.29}$$

□

2.3 The modular method

The key ideas of the proposed modular method are:

1. computing the subresultants $r = S_0(t, f, x)$, $\ell = S_1(t, f, x)$, $s = S_0(r, b, y)$, and $g = S_1(r, b, y)$ by evaluation and interpolation, specializing the variable z;
2. using for this evaluation-interpolation scheme, as a degree bound, the smallest possible multiple of the Bézout bound $D = \deg(f)\deg(t)\deg(b)$ of the (zero-dimensional) variety $V(f, t, b)$, and trying to avoid the traditional, and often more pessimistic, bounds from subresultant theory; and
3. verifying the genericity assumptions as we recover s, g, ℓ from this evaluation-interpolation scheme, and returning an error message, if one of those hypotheses is not met.

Since the proposed modular method aims at computing s, g, ℓ by specializing z, we need bounds for the degree in z of each of the polynomials s, g, ℓ. We start with ℓ. Consider the Sylvester matrix of t and f (resp. f and t) regarded as polynomials in $(\mathbb{K}[y, z])[x]$ if $\mathrm{mdeg}(t) \geq \mathrm{mdeg}(f)$ (resp. $\mathrm{mdeg}(t) < \mathrm{mdeg}(f)$). The degree of the resultant r satisfies:

$$\deg(r) \leq \deg(f, x)\deg(t, y) + \deg(t, x)\deg(f, y) \leq 2\deg(f)\deg(t), \tag{2.30}$$

where $\deg(r), \deg(f), \deg(t)$ are the total degrees of r, f, t respectively, and where $\deg(., x)$ and $\deg(., y)$ denote partial degrees w.r.t. x and y, respectively. Consider now the determinantal formulations of the subresultant $\ell = S_1(t, f, x)$ of index 1, using the notations of Section 2.2. Each of the two coefficients w.r.t. x of ℓ has a total degree (and thus in degree in z) bounded over by $2\deg(f)\deg(t)$.

Recall that s is the resultant of r and b (resp. b and r) regarded as polynomials in $(\mathbb{K}[z])[y]$ if $\mathrm{mdeg}(r) \geq \mathrm{mdeg}(b)$ (resp. $\mathrm{mdeg}(r) < \mathrm{mdeg}(b)$). From the classical proof of Bézout theorem (in the case of plane curves) based on the Sylvester resultant, see for instance [17], we have

$$\deg(s) \leq \deg(r)\deg(b), \tag{2.31}$$

which yields:

$$\deg(s) \leq 2D. \tag{2.32}$$

Of course, one can also use

$$\deg(s) \leq \min(\deg(r)\deg(b), \deg(r, y)\deg(b, z) + \deg(b, y)\deg(r, z)) \tag{2.33}$$

yielding an improved upper bound for $\deg(s)$.

It remains to find an upper bound for $\deg(g, z)$. Considerations based on determinantal formulations of the subresultant of index 1 and Relation (2.30) yields

$$\deg(g, z) \leq 4D. \tag{2.34}$$

However, in practice $\deg(g, z) \leq 2D$ is usually true. Indeed, in a given subresultant chain, the height of the coefficients in a subresultant generically increases as the index of the subresultant decreases. However, cancellations may happen making this observation false in non-generic situations, for instance, when the subresultant of index 0 (that is, the resultant) is constant.

Therefore, our modular method has two flavors. The first one is *probabilistic* and uses $2D$ as a bound for $\deg(s), \deg(g, z), \deg(\ell, z)$. The second one is *deterministic* and uses $2D$ as a bound for $\deg(s), \deg(\ell, z)$, while using $4D$ as a bound for $\deg(g, z)$.

When the probabilistic method is used and returns a regular chain $s, \hat{g}, \hat{\ell}$, one can check whether

1. $\hat{g}$ divides r and b modulo the regular chain $\{\overline{s}\}$, and
2. $\hat{\ell}$ divides f and t modulo the regular chain $\{\overline{s}, \hat{g}\}$,

both hold; if they do then $V(s, \hat{g}, \hat{\ell}) \subseteq V(f, t, b)$ holds. Moreover, if $\deg(\overline{s}) = D$ holds then we have $V(s, \hat{g}, \hat{\ell}) = V(f, t, b)$. We note that using a probabilistic approach for computing resultants in modular fashion is common, see for instance [24, 6]. In the sequel of the section, for simplicity of presentation, we focus on the probabilistic approach and we note that switching to the deterministic approach requires minor adjustments in our pseudo-code.

Our proposed modular method is stated in the form of pseudo-code with Algorithms 1 2, 3, 4, 5, 6 and 7.

Algorithm 7 is the top level procedure: it takes the polynomials $f, t, b \in \mathbb{K}[x > y > z]$ as input and:

1. returns a regular chain $\{\overline{s}, \hat{g}, \hat{\ell}\}$ so that $V(\overline{s}, \hat{g}, \hat{\ell}) = V(\overline{s}, g, \ell)$ holds, if Hypotheses 1, 2, 3 and 4 all hold; or
2. an ERROR message otherwise.

Algorithm 7 directly calls Algorithms 5 and 6 which, respectively:

1. collects modular images of the polynomials s, g, ℓ as well as modular images of other polynomials; and
2. combines those modular images in order to produce the polynomials $s, \hat{g}, \hat{\ell}$ as well as other polynomials which are used to decide whether Hypotheses 1, 2, 3 and 4 hold or not; moreover, if no ERROR message is returned we know that g and ℓ are, up to a unit in $\mathbb{K}[z]/\langle\overline{s}\rangle$, equal to $\hat{g}$ and $\hat{\ell}$ respectively.

Algorithm 5 relies on Algorithms 1, 2 and 4 for computing subresultants of univariate polynomials and subresultants of bivariate polynomials. Algorithm 3 is a helper function of Algorithm 5.

Notation 1. *In Algorithm 1, 3 and 5 we use the symbol + to denote the concatenation of lists.*

The basic idea of Algorithm 7 is to specialize, in our regular chain $T := \{t, b\}$, the variable z at sufficiently many good points $v \in \mathbb{Z}$, using the notions and results of Section 2.1.7. In other words:

1. we require that after specializing b to $b_v := b(y, v)$, its initial is non-zero; and
2. we require that after specializing t to $t_v := t(x, y, v)$, its initial is invertible modulo b_v.

Next, we need to check that $f_v := f(x, y, v)$, the specialization of f at $z = v$, has the same main variable and main degree as f. If any of the above conditions is not fulfilled, then we discard the point v and try a new one. This process continues until we have obtained $B := 2(\deg(f)\ \deg(t)\ \deg(b)) + 1$ good specializations of T and f.

After checking that v is a good specialization point, we normalize the regular chain $T_v := \{t_v, b_v\}$ to $\tilde{T}_v := \{\tilde{t}_v, \tilde{b}_v\}$, see Lines 11 and 12 in Algorithm 5. The following step (see Algorithm 4) consists of computing the subresultants

$$r_v := S_0(\tilde{t}_v, f_v, x) \text{ and } \ell_v := S_1(\tilde{t}_v, f_v, x) \tag{2.35}$$

Algorithm 1 SubresultantChain

Require: $a, b \in \mathbb{B}[y]$ with $m = \deg(a) \geq n = \deg(b)$ and $\mathbb{B}$ an integral domain; note that when this algorithm is called, the domain $\mathbb{B}$ is a prime field.

Ensure: the subresultant chain $(S_0, S_1, \ldots, S_{m-1}, S_m)$ if a and b, where S_i is the subresultant of index i.

$S := ()$ ▷ Initializing S to the empty list.
if $m > n$ **then** $S := (\mathrm{lc}(b)^{m-n-1} b) + S$ ▷ Inserting an element into S.
$s := \mathrm{lc}(b)^{m-n}$
$A := b$; $B := \mathrm{prem}(a, -b)$
$S := (B) + S$
$d := \deg A$
while true **do**
  if $B = 0$ **then return** $(0, \ldots, 0)_{d-1} + S$ ▷ Inserting $d - 1$ zeros into S.
  $e := \deg B$; $\delta := d - e$
  $C := \frac{\mathrm{lc}(B)^{\delta-1} B}{s^{\delta-1}}$ ▷ With $\mathbb{B}$ a prime field, no need to use Lazard's optimization.
  if $\delta > 1$ **then** $S := (C) + (0, \ldots, 0)_{\delta-2} + S$ ▷ Inserting $\delta - 2$ zeros into S.
  if $\deg C = 0$ **then return** S
  $B := \frac{\mathrm{prem}(A, -B)}{s^{\delta} \mathrm{lc}(A)}$ ▷ With $\mathbb{B}$ a prime field, no need to Ducos' optimization.
  $S := (B) + S$
  $A := C$
  $s := \mathrm{lc}(A)$
  $d := \deg A$

Algorithm 2 ModularSRCForBivariatePolynomials

Require: $a, b \in \mathbb{K}[x][y]$ with $m = \deg(a) \geq n = \deg(b)$, an upper bound D on the degree in x of the subresultants of a, b w.r.t. y.

Ensure: the subresultants S_0 and S_1 of index 0 and 1 for a, b .

$S_0 := (); S_1 := (); V := ()$ ▷ Initializing S_0, S_1, V to the empty list.
$v := 1$ ▷ The next value for specializing x.
$N := 0$ ▷ The number of specializations used so far.
while $N < D$ **do**
  Let A and B be the images of a and b in $\mathbb{K}[x, y]/\langle x - v \rangle$
  if $\deg(A, y) = \deg(a, y)$ and $\deg(B, y) = \deg(b, y)$ **then**
    $S :=$ SubresultantChain$(\mathbb{K}, A, B)$ ▷ Good specializations since $\mathrm{lc}(a, y) \neq 0$.
    ▷ and $\mathrm{lc}(b, y) \neq 0$ at $x = v$.
    $S_0 := S_0 + (S[0]), S_1 := S_1 + (S[0])$ ▷ Adding the images at $x = v$ of the subresultants.
    $V := V + (v)$ ▷ Adding v to the list of good specializations.
    $N := N + 1$
  $v := v + 1$ ▷ Get the next specialization.
$s_0 :=$ Interpolate(V, S_0) ▷ Computing the resultant S_0 of a, b w.r.t. y by interpolation.
$s_1 :=$ InterpolateCoefficients(V, S_1) ▷ Computing the coefficients w.r.t y of S_1.
return s_0, s_1

between $\tilde{t}_v$ and f_v (resp. f_v and $\tilde{t}_v$), if $\mathrm{mdeg}(\tilde{t}_v) \geq \mathrm{mdeg}(f_v)$ (resp. $\mathrm{mdeg}(f_v) > \mathrm{mdeg}(\tilde{t}_v)$).

Algorithm 3 GoodSpecializationPoint

Require: $f, t, b \in \mathbb{K}[x, y, z]$ satisfying the hypotheses of Section 2.2; in particular, we have $\mathrm{mvar}(f) = x$, $\mathrm{mvar}(t) = x$, $\mathrm{mvar}(b) = y$; we are also given a point $v \in \mathbb{K}$.
Ensure: TRUE, if $z = v$ is a good specialiazation, FALSE otherwise.
1: $h_f := \mathrm{init}(f)$, $h_t := \mathrm{init}(t)$, $h_b := \mathrm{init}(b)$
2: Let H_b be the image of h_b in $\mathbb{K}[x, y, z]/\langle z - v\rangle$ and H_f, H_t be the images of h_f, h_t in $\mathbb{K}[x, y, z]/\langle z - v, b\rangle$
3: **if** $H_b = 0$ or H_f is zero-divisor or H_t is zero-divisor **then**
4: **return** FALSE
5: **else**
6: **return** TRUE

Algorithm 4 SubresultantsOfIndexZeroAndOne

Require: $f_v, t_v, b_v \in \mathbb{K}[x, y]$ with $\mathrm{mvar}(f) = x$, $\mathrm{mvar}(t) = x$, $\mathrm{mvar}(b) = y$, a point $v \in \mathbb{K}$.
Ensure: $r_v, \ell_v, s_v, g_v, \lambda_v, \theta_v$ as in Equations 2.35, 2.36, 2.37.
1: **if** $\mathrm{mdeg}(t_v) \geq \mathrm{mdeg}(f_v)$ **then**
2: $r_v, \ell_v :=$ ModularSRCForBivariatePolynomials(t_v, f_v, x, y)
3: **else**
4: $r_v, \ell_v :=$ ModularSRCForBivariatePolynomials(f_v, t_v, x, y)
5: **if** $r_v \in \mathbb{K}$ **then**
6: **throw** ERROR
7: **if** $\mathrm{mdeg}(r_v) \geq \mathrm{mdeg}(b_v)$ **then**
8: $s_v, g_v :=$ SubresultantChain(r_v, b_v, y)
9: **else**
10: $s_v, g_v :=$ SubresultantChain(b_v, r_v, y)
11: $\lambda_v :=$ resultant$(\mathrm{init}(\ell_v), g_v)$
12: $\theta_v :=$ resultant$(\mathrm{init}(t_v), g_v)$
13: **return** $r_v, s_v, g_v, \ell_v, \lambda_v, \theta_v$

Then, we calculate the subresultants

$$s_v := S_0(r_v, \tilde{b}_v, y) \text{ and } g_v := S_1(r_v, \tilde{b}_v, y) \tag{2.36}$$

between r_v and $\tilde{b}_v$ (resp. $\tilde{b}_v$ and r_b), if $\mathrm{mdeg}(r_v) \geq \mathrm{mdeg}(\tilde{b}_v)$ (resp. $\mathrm{mdeg}(\tilde{b}_v) > \mathrm{mdeg}(r_v)$). Here, we additionally compute the resultants,

$$\lambda_v := \mathrm{res}(\mathrm{init}(l_v), g_v) \text{ and } \theta_v := \mathrm{res}(\mathrm{init}(t_v), g_v). \tag{2.37}$$

These extra computations are used to check our Hypotheses 3 and 4. Additionally, we also check that r_v is not constant.

Notation 2. *Let V be the collection of good specialization points v and let S, G, L, Λ, Θ be the collections of images $s_v, g_v, \ell_v, \lambda_v, \theta_v$, respectively, for $v \in V$.*

Note that, in Algorithm 5, if we encounter a polynomial r_v with degree less than the currently set value d_r, then we discard v. On the other hand, if the degree of r_v is greater than the current d_r, then we discard all the previously saved images and start again with $d_r := \deg r_v$.

Algorithm 5 CollectingImages

Require: $f, t, b \in \mathbb{K}[x, y, z]$ with $\mathrm{mvar}(f) = x$, $\mathrm{mvar}(t) = x$, $\mathrm{mvar}(b) = y$.
Ensure: $V, S, G, L, \Lambda, \Theta$ as in Notation 2.

$V := (); S := (); G := (); L := (); \Lambda := (); \Theta := ()$ ▷ Initializing S, G, L, Λ, Θ to the empty list.
$v := 1$ ▷ The next value for specializing x.
$B := 2(\deg(f)\ \deg(t)\ \deg(b)) + 1$ ▷ Number of images to be collected.
$N := 0$ ▷ Number of images collected so far.
$d_r := 0$ ▷ Candidate to degree of **r**.
while $N < B$ **do**
  if not GoodSpecializationPoint(f, t, b, v) **then**
    $v := v + 1$ ▷ Get the next specialization.
    next
  $f_v, t_v, b_v := f(x, y, z = v), t(x, y, z = v), b(y, z = v)$ ▷ Specializing f, t, b at $z = v$.
  $\tilde{b}_v := \mathrm{init}(b_v)^{-1} b_v$ ▷ We normalize $\{t_v, b_v\}$.
  $\tilde{t}_v := \mathrm{init}(t)_v^{-1} t_v \bmod \tilde{b}_v$ ▷ We normalize $\{t_v, b_v\}$.
  try
    $r_v, s_v, g_v, \ell_v, \lambda_v, \theta_v :=$ SubresultantsOfIndexZeroAndOne$(f_v, \tilde{t}_v, \tilde{b}_v, v)$
  catch Error
    throw Error
  end try
  if $\mathrm{mdeg}(r_v) < d_r$ **then** ▷ $z = v$ is an unlucky specialization.
    $v := v + 1$ ▷ Get the next specialization.
    next
  if $\mathrm{mdeg}(r_v) > d_r$ **then** ▷ All previous specializations were unlucky.
    $d_r := \mathrm{mdeg}(r_v); N := 0; V := (); S := (); G := (); L := (); \Lambda := (); \Theta := ()$ ▷ So, restart ▷ from scratch.
  $S, G, L := S + (s_v), G + (g_v), L + (\ell_v)$ ▷ Adding s_v, g_v, ℓ_v to their respective list.
  $\Lambda, \Theta := \Lambda + (\lambda_v), \Theta + (\theta_v)$ ▷ Adding λ_v, θ_v to their respective list.
  $V := V + (v)$ ▷ Adding v to the list of good specializations.
  $v := v + 1$ ▷ Get the next specialization.
  $N := N + 1$
if $d_r = 0$ **then**
  throw Error ▷ Checking whether Hypothesis 2 holds or not.
return V, S, G, L, Λ, Θ

This checking is needed since we do not know a priory the degree of r, and after specializing f and t to $z = v$, the degree of r_v could be different than $\deg r$.

After collecting enough images and points v, we interpolate all the coefficients of s_v, g_v, ℓ_v into $s, \hat{g}, \hat{\ell}$ using the `Interp modp1` function (see Algorithm 6). At this point, we apply rational function reconstruction to these interpolated values, and we write $s, \hat{g}, \hat{\ell}$ as a fraction between two polynomials. Finally, we take only their numerators and called them again $s, \hat{g}, \hat{\ell}$. We also interpolate our λ_v and θ_v into λ and θ, respectively.

Notation 3. *Let $s, \hat{g}, \hat{\ell}, \lambda, \theta$ be the polynomials obtained by interpolation using the respective value sets S, G, L, Λ, Θ, and the point set V.*

Note that we compute the square free part, $\overline{s}$, of s during Algorithm 6. At this point, it

Algorithm 6 Interpolate

Require: $V, S, G, L, \Lambda, \Theta$ as in Notation 2.
Ensure: $\overline{s}, \hat{g}, \hat{\ell}, \lambda, \theta$ as in Notation 3.

```
m:=1
for v ∈ V do
    m := (z − v)m                          ▷ "Product of the moduli".
s := Interpolate(V, S)
s := RationalFunctionReconstruction(s, m)   ▷ We apply rational function reconstruction to s.
s := Numerator(s)                           ▷ We get the numerator of s.
s̄ := SquareFreePart(s)                      ▷ We compute the squarefree part of s.
ĝ := InterpolateCoefficients(V, G)          ▷ We interpolate the coefficients of the polynomials in G.
ĝ := RatFuncRecCoeff(ĝ, m)                  ▷ We apply rational function reconstruction to the coefficients of ĝ.
ĝ := numer(ĝ)                               ▷ We write ĝ as a fraction and take its denominator.
ℓ̂ := InterpolateCoefficients(V, L)          ▷ We interpolate the coefficients of the polynomials in L.
ℓ̂ := RatFuncRecCoeff(ℓ̂, m)                  ▷ We apply rational function reconstruction to the coefficients of ℓ̂.
ℓ̂ := numer(ℓ̂)                               ▷ We write ℓ̂ as a fraction and take its denominator.
λ := Interpolate(V, Λ)
θ := Interpolate(V, Θ)
return s̄, ĝ, ℓ̂, λ, θ
```

Algorithm 7 IntersectBySpecialization

Require: $f, t, b \in \mathbb{K}[x > y > z]$ with $\mathrm{mvar}(f) = x$, $\mathrm{mvar}(t) = x$, $\mathrm{mvar}(b) = y$, assuming $\{t, b\}$ is a regular chain.
Ensure: $\{\overline{s}, \hat{g}, \hat{\ell}\}$ as defined in Section 2.2.

```
V, S, G, L, Λ, Θ := CollectingImages(f, t, b)
s̄, ĝ, ℓ̂, λ, θ := Interpolate(S, G, L, Λ, Θ)
if deg(gcd(init(g), s̄)) ≠ 0 or deg(gcd(λ, s̄)) ≠ 0 then
    throw ERROR                             ▷ Hypothesis 3 does not hold.
if deg(gcd(θ, s̄)) ≠ 0 then                  ▷ θ is Not Regular modulo s̄.
    throw ERROR                             ▷ Hypothesis 4 does not hold.
return {s̄, ĝ, ℓ̂}
```

remains to check that $\{\overline{s}, \hat{g}, \hat{\ell}\}$ is a regular chain and that the initial of t is regular modulo this regular chain. For this purpose, we can easily check whether $\mathrm{init}(\hat{g})$ is regular modulo $\overline{s}$ using gcd computations in one variable. Thus, we only need to check that $\mathrm{init}(\hat{\ell})$ and $\mathrm{init}(t)$ are regular modulo $\{\overline{s}, \hat{g}\}$. Here, we apply to λ and θ the same process as the one applied to $\mathrm{init}(\hat{g})$. Putting all of the above together leads us to Algorithm 7.

2.4 Implementation in MAPLE

We implemented the algorithms presented in Section 2.3, over a large prime field, using the release 2021 of the MAPLE language. This version is required to use the algorithm. A released version can be found in the github repository [15].

2.4.1 The `modp1` library

A key point of our implementation is that it only relies on well-optimized kernel functions of MAPLE. For instance, all of our subresultant chain computations are done using the `modp1` library. The `modp1` functions provide efficient arithmetic and other key operations for the domain $\mathbb{Z}_n[x]$ of univariate polynomials over the integers modulo n. To achieve high performance, `modp1` uses a special representation. If the integer n is small enough, arithmetic in $\mathbb{Z}_n$ is performed directly with machine-word arithmetic instead of using multi-precision integer arithmetic [1].

The most relevant `modp1` functions used in the development of the modular algorithm are the following ones:

1. **ConvertIn\ConvertOut:** convert to and from the `modp1` representation.
2. **Constant:** convert in a constant polynomial to the `modp1` representation for a given variable.
3. **Add\Multiply\Quo\Power\Rem:** basic operations modulo a prime.
4. **Prem:** pseudoremainder modulo a prime.
5. **Degree\Coeff:** degree of a `modp1` polynomial and the coefficient of a monomial of a given degree.
6. **Eval:** evaluation at a point.
7. **Interp:** univariate polynomial interpolation.

An important remark is that the `modp1` package does not provide a rational function reconstruction procedure. Thus, before applying the MAPLE function Ratrecon, we must convert out of the `modp1` representation to the MAPLE representation. This is done after using the Interp function.

2.4.2 Computation of subresultant chains

The modular method presented in Section 2.3 relies on the computations of subresultant chains of univariate polynomials as well as the computations of subresultant chains of bivariate polynomials.

For the case of subresultant chains of univariate polynomials, we follow (a variant of) **Algorithm 1** from [6]. Our code takes advantage of MAPLE's `modp1` library for univariate polynomial arithmetic. In particular, the input polynomials, as well as the returned subresultant chain are in the the `modp1` representation. It is also important to mention that since we are working over a Finite Field, we did not need the Ducos optimization when developing Algorithm 1.

For computing the subresultant chain of two bivariate polynomials, we use a modular approach, reconstructing only the subresultants S_0 and S_1. We specialize one of the variables and then compute the subresultants chain of our, now, univariate polynomials. As a consequence of this, our approach is not "speculative" (discussed below). We repeat this process at sufficiently many points, and subsequently use univariate interpolation to reconstruct the coefficients of S_0 and S_1. Most of these computations are performed with the `modp1` representation; avoiding conversions between the MAPLE and the `modp1` representations as much as possible. Our implementation was inspired by Michael Monagan's `modp1` library code for computing the resultant of bivariate polynomials.

As a first remark, we apply our `modp1` conversions during the computations of r_v and ℓ_v,

i.e., during the calculation of subresultant chains of polynomials in two variables. Our implementation of Algorithm 2 takes as input two polynomials in the MAPLE representation and converts them to the `modp1` representation. Then, our implementation of Algorithm 1 requires its input polynomials to be in the `modp1` representation, again avoiding conversions. Also, both Algorithms 2 and 1 output `modp1` polynomials.

Finally, it is important to mention that there is large room for improvement. Indeed, using a modular method opens the door to using *speculative algorithms* for computing subresultants; see [6]. Speculative algorithms are asymptotically fast algorithms that:

1. compute the subresultants of index 0 and 1 without computing the other subresultants, while
2. being able to resume the computations for obtaining the subresultants of higher index, if needed.

2.5 Implementation in C

A second version of the algorithms presented in Section 2.3, over a large prime field, using the C programming language was developed. In C, we have sharp control of computing resources, in particular memory. This brings a huge improvement to our BPAS implementation of Algorithm IntersectBySpecialization. To be precise, we make use of the Basic Polynomial Algebra Subprograms (BPAS) library (see [4]). BPAS provides support for arithmetic operations with polynomials on modern computer architectures, in particular hardware accelerators. In this version, we make use of *speculative algorithms* for computing subresultants (see [6]), which improves even more the performance of our BPAS implementation compare to our previous MAPLE version.

Instead of using Algorithms 1 and 2, we decide to use Half-GCD based versions of the subresultant algorithm for univariate and bivariate polynomials. Specifically, we make use of the already existing versions inside BPAS. The univariate algorithm is based in the Half-GCD algorithm (for details see [6]). Thus, we only compute the subresultants of index 0 and 1, and in case we need to compute subresultants of higher index, we are able to resume our computations. This greatly improves the speed of Algorithm 4. On the other hand, the bivariate version of the subresultant algorithm works by means of evaluation an interpolation in a similar way than our MAPLE implementation (Algorithm 2). The main difference with the MAPLE version is that the BPAS algorithm uses the Half-GCD univariate subresultant algorithm instead of computing the whole subresultant chain between the input polynomials. Thus, this algorithm also only computes subresultants of index 0 and 1.

For the interpolation part (Algorithm 6), we use the implementation for univariate Lagrange interpolation inside of BPAS. As it was shown in Figure 5.1 of [11], this Algorithm is faster than the linear equations method and also the MAPLE's CurveFitting:-PolynomialInterpolation.

All of these improvements, make the BPAS implementation of Algorithm IntersectBySpecialization much faster than its MAPLE counterpart as shown in the experimental results in Tables 2.2 and 2.3 of Section 2.6.

2.6 Benchmarking

Our experimentation was collected on a laptop running Linux Mint 19.3 with a Intel Core i7-9750H processors at 2.60 GHz, and a 8GB DDR3 memory.

2.6.1 A promising example

For the following experiments, we use a prime characteristic of 469762049. Although, our implementations supports any big enough characteristic. We can observe that the modular algorithm performs better with increasing degrees of the input polynomials. All the examples described in this section can be found in the github repository [15]. It is important to mention that these experiments were made using our probabilistic approach, and the results were verified using the non-modular and deterministic algorithms provided in the `RegularChains` package of MAPLE.

Example 1. *Consider the following promising example in characteristic* 469762049. *Let* $t = 4x^9 - 40x^5y^2z + 6x^3y^3z + 27xy^6 + 68xy^3z^2 - 11z^5$, $b = -33y^8z + 8y^5z^2 - 69y^4z^2 - 34z^6 - 58y^5 - 53yz^2$ *and* $f = -7x^3y^2z^4 - 50y^4z^5 - 70x^3y^5 + 19xy^5 - 5y^3z + 48x$. *If we compute the intersection between* f *and the regular chain* $T := \{t, b\}$, *using the* Intersect *command of* MAPLE *the time spent is approximately 298.017***s**. *On the other hand, if we use our* MAPLE *version of the modular algorithm the computations only take 30.187***s**.

The previous example shows how much we can improve by using modular methods in the computations of intersections. We can also use the MAPLE command CodeTools:-Profiling:-Profile to profile the time spent by each of our procedures. Table 2.1 presents the profiling of the most important statements executed when calling Algorithm 7 during the computation of Example 1. There, the first column represents the number of the statement in the procedure. The next three columns give the count of the number of calls of that statement, the total time spent executing the statement (in seconds), and the number of memory words allocated while executing the statement. The last column gives the name of the statement.

Here, it is important to notice that most of the time is spent computing the images $s_v, g_v, \ell_v, \lambda_v, \theta_v$ (PROC 38). Thus, by improving the performance of this procedure using `modp1`, we were able to enhance the overall performance of the algorithm. Also note, that since we are using the Interp function of the `modp1` package, the interpolations of polynomials are cheap (PROC 68). Unfortunately, the `modp1` package does not provide a rational function reconstruction procedure. Thus, we are working in the MAPLE representation (PROC 70).

2.6.2 Random test

The examples presented in Table 2.2 and 2.3 are given by randomly generated polynomials t, b, f of fixed degree in the variables $x > y > z$ (columns 2, 3 and 4). There, we compare the times taken by the Intersect algorithm (of `RegularChains`), our MAPLE version of Intersect by Specialization algorithm and our BPAS version of Intersect by Specialization algorithm

	Calls	Seconds	Words	Name
PROC	1	29.434	547420758	
⋮	⋮	⋮	⋮	⋮
30	1459	0.878	11276713	GoodSpecializationPoint
⋮	⋮	⋮	⋮	⋮
38	1459	23.292	417278789	SubresultantsOfIndexZeroAndOne
⋮	⋮	⋮	⋮	⋮
68	1	0.046	8979	Interpolate(V, S)
⋮	⋮	⋮	⋮	⋮
70	1	1.425	92824827	RationalFunctionReconstruction(s)
⋮	⋮	⋮	⋮	⋮
74	1	3.362	21229523	InterpolateCoefficients(V, L) RatFuncRecCoeff(ℓ)
75	1	0.081	1210070	InterpolateCoefficients(V, G) RatFuncRecCoeff(g)
⋮	⋮	⋮	⋮	⋮
83	1	0.031	8984	Interpolate(V, Λ)
⋮	⋮	⋮	⋮	⋮
87	1	0.030	8979	Interpolate(V, Θ)

Table 2.1: Profiling of Example 1

(columns 7, 8 and 9). We also include the bound $B := 2(\deg(f)\ \deg(t)\ \deg(b)) + 1$ of the input system as well as the number of iterations needed to reach that bound (columns 5 and 6). These numbers differ when a bad specialization point is discovered.

N	deg(t)	deg(b)	deg(f)	bound B	Num. Iterations	Intersect	Intersect by Specialization	Intersect by Specialization (BPAS)
1	5	4	5	201	201	**0.184s**	1.705**s**	0.0983**s**
2	5	4	4	161	161	**0.126s**	0.377**s**	0.0392**s**
3	5	4	5	201	201	**0.200s**	0.673**s**	0.0772**s**
4	5	4	5	201	201	**0.433s**	1.091**s**	0.1038**s**
5	8	8	8	1025	1025	24.687**s**	**13.763s**	2.6651**s**
6	8	8	8	1025	1025	43.324**s**	**18.497s**	4.1805**s**
7	8	8	8	1025	1025	43.557**s**	**16.778s**	3.4076**s**
8	8	8	8	1025	1025	**5.700s**	12.683**s**	2.4368**s**

Table 2.2: Examples 1

N	deg(t)	deg(b)	deg(f)	bound B	Num. Iterations	Intersect	Intersect by Specialization	Intersect by Specialization (BPAS)
9	8	8	8	1025	1025	**1.696s**	7.075**s**	0.9383**s**
10	7	6	7	589	589	13.110**s**	**11.313s**	1.3616**s**
11	8	7	8	897	897	17.246**s**	**16.084s**	2.1516**s**
12	8	7	8	897	897	20.584**s**	**17.331s**	2.8275**s**
13	9	9	9	1459	1459	301.062**s**	**27.999s**	7.6849**s**
14	8	8	8	1153	1153	63.850**s**	**23.085s**	4.6934**s**
15	8	7	8	897	897	15.580**s**	**15.870s**	2.2245**s**
16	8	7	8	897	897	**10.970s**	16.910**s**	2.3988**s**
17	8	8	8	1025	1025	24.418**s**	**12.920s**	2.7127**s**
18	9	8	9	1153	1153	70.321**s**	**24.952s**	4.6852**s**

Table 2.3: Examples 2

As stated before, we observe that when the degrees of the input system f, t, b grow larger, the performance of the modular method improves significantly. As a remark, we do not display any information about a version of the Intersect inside of BPAS in Tables 2.2 and 2.2 because software limitations. Unfortunately, the current version of BPAS does not count with a version of this algorithm for prime characteristic. Though, there is a version of the Intersect algorithm in characteristic zero. It would not be fair, however, to compare against it.

Chapter 3

Algorithms for multivariate Laurent series

The current chapter is about arithmetic operation between multivariate Laurent series. We first present the preliminary concepts needed to develop our theory (Section 3.1). Then, we show the most important aspects from the algebraic construction develop by Monforte and Kauers in [25] (Section 3.2). The following section handles the algorithmic part of this work (Section 3.3). This is the main section of this chapter, and here we can find our approach for coding the Laurent series object, addition and multiplication between these objects and also computing the multiplicative inverse of them. Finally, Section 3.4 illustrates how to use the basic commands of our Laurent series object.

3.1 Preliminaries

We start by recalling the notion of a cone. We do not state that notion in its full generality, but rather with additional constrains adapted to our context. Then, we explain the relationship between cones and total orders. We use these tools in our constructive presentation and implementation of fields of multivariate Laurent series.

3.1.1 Cones

A set $C \subseteq \mathbb{R}^p$ is a *cone* whenever $\mathbf{v} \in C$ and $c \geq 0$ hold, then we have $c\mathbf{v} \in C$; however, we require additional constrains leafing to the following definition.

Definition 1. *A set $C \subseteq \mathbb{R}^p$ is a cone when the following properties hold:*

1. *if $\boldsymbol{v} \in C$ and $c \geq 0$ hold, then we have $c \cdot \boldsymbol{v} \in C$.*
2. *C is finitely generated, i.e., there exist $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_m \in \mathbb{R}^p$ such that*

$$C = \{z_1\boldsymbol{r}_1 + \cdots + z_n\boldsymbol{r}_m \mid z_1, \ldots, z_m \geq 0\}.$$

   When the above holds, the set $\boldsymbol{R} := \{\boldsymbol{r}_1, \ldots, \boldsymbol{r}_m\}$ is called a generating set *of C (also denoted by $C(\boldsymbol{R})$ to emphasize that $\boldsymbol{R}$ is a generating set) and its members are called* rays *of the cone C.*
3. *C is rational, i.e., C is finitely generated and has a generating set $\{\boldsymbol{r}_1, \ldots, \boldsymbol{r}_m\} \subset \mathbb{Z}^p$.*

We also define the translation of the cone $C(\boldsymbol{R})$ to $\boldsymbol{e} \in \mathbb{R}^p$ by

$$C(\boldsymbol{e}, \boldsymbol{R}) := \boldsymbol{e} + C(\boldsymbol{R}) := \{\boldsymbol{e} + z_1\boldsymbol{r}_1 + \cdots + z_m\boldsymbol{r}_m \mid z_1, \ldots, z_m \geq 0\}.$$

We refer to $C(\boldsymbol{e}, \boldsymbol{R})$ simply as the cone C with vertex $\boldsymbol{e}$ and rays $\boldsymbol{R}$.

An another property of cones that we use through this document is the following: C is said to be *line-free* if for every $\mathbf{v} \in C \setminus \{\mathbf{0}\}$, we have $-\mathbf{v} \notin C$.

Remark 1 (see [25])**.** *With our definition, cones:*

1. *are closed in the Euclidean topology,*
2. *contain $\boldsymbol{0}$,*
3. *are either unbounded or equal to $\{\boldsymbol{0}\}$,*
4. *are convex, that is, for all $\boldsymbol{v}_1, \boldsymbol{v}_2 \in C$ and for all $c \in [0, 1]$ we have $c\boldsymbol{v}_1 + (1 - c)\boldsymbol{v}_2 \in C$ as well,*
5. *$\boldsymbol{v}_1, \boldsymbol{v}_2 \in C$ implies that $\boldsymbol{v}_1 + \boldsymbol{v}_2 \in C$,*
6. *when C and D are cones, then so is $C + D = \{\boldsymbol{v}_1 + \boldsymbol{v}_2 \mid \boldsymbol{v}_1 \in C, \boldsymbol{v}_2 \in D\}$.*

3.1.2 Total orders

Definition 2. *A* total *or* linear *order is a partial order in which any two elements are comparable. In other words, a total order is a binary relation $\preceq$ on a set S, which satisfies the following properties for all s_1, s_2 and s_3 in S*

1. *$s_1 \preceq s_1$;*
2. *$s_1 \preceq s_2$ and $s_2 \preceq s_3$, imply $s_1 \preceq s_3$;*
3. *$s_1 \preceq s_2$ and $s_2 \preceq s_1$, imply $s_1 = s_2$;*
4. *$s_1 \preceq s_2$ or $s_2 \preceq s_1$.*

A *monomial order* is a total order over the set of all monomials of a given polynomial ring with the additional property that for all monomials s_1, s_2 and s_3 we have:

$$s_1 \preceq s_2 \implies s_1 s_3 \preceq s_1 s_3.$$

We said that a total order $\preceq$ on $\mathbb{Z}^p$ is *additive* if for all $\mathbf{i}, \mathbf{j}, \mathbf{k} \in \mathbb{Z}^p$, we have:

$$\mathbf{i} \preceq \mathbf{j} \implies \mathbf{i} + \mathbf{k} \preceq \mathbf{j} + \mathbf{k}.$$

Definition 3. *An additive order $\preceq$ on $\mathbb{Z}^p$ is said to be* compatible with a cone $C \subseteq \mathbb{R}^p$ *if $\boldsymbol{0} \preceq \boldsymbol{k}$ holds, for all $\boldsymbol{k} \in C \cap \mathbb{Z}^p$.*

Lemma 4 (see [25])**.** *Let $C, D \subseteq \mathbb{R}^p$ be cones and let $\preceq$ be an additive order on $\mathbb{Z}^p$. Let $\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k\}$ be a set of generators of C.*

1. *If C is compatible with $\preceq$, then C is line-free.*
2. *C is compatible with $\preceq$, if and only if $\boldsymbol{0} \preceq \boldsymbol{v}_i$ for all i.*
3. *If C, D are compatible with $\preceq$, then $C + D$ is also compatible with $\preceq$.*

Lemma 5 (see [25])**.** *Let $\preceq$ be an additive order on $\mathbb{Z}^p$ and $C \subseteq \mathbb{R}^p$ be a cone. If $\preceq$ is compatible with C, then $\preceq$ is a well-founded order on $C \cap \mathbb{Z}^p$, i.e., every strictly decreasing sequence $\boldsymbol{v}_1 \succneqq \boldsymbol{v}_2 \succneqq \boldsymbol{v}_3 \succneqq \ldots$ of elements in $C \cap \mathbb{Z}^p$ terminates, or equivalently, every subset of $C \cap \mathbb{Z}^p$ contains a $\preceq$-minimal element.*

3.2 Construction

Let $\mathbb{K}$ be a field, $\mathbf{x} = x_1, \ldots, x_p$ and $\mathbf{u} = u_1, \ldots, u_m$ be indeterminates. We denote by $\mathbb{K}[[\mathbf{u}]]$ the set of formal power series with coefficients over the field $\mathbb{K}$, i.e., $g(\mathbf{u}) \in \mathbb{K}[[\mathbf{u}]]$ implies that

$$g(\mathbf{u}) = \sum_{\mathbf{k} \in \mathbb{N}^m} a_{\mathbf{k}} \mathbf{u}^{\mathbf{k}},$$

for some $a_{\mathbf{k}}$ in $\mathbb{K}$, and $\mathbf{u}^{\mathbf{k}}$ is a notation for $u_1^{k_1} \cdots u_p^{k_p}$ where $k_, \ldots, k_p$ are non-negative integers. The set $\mathbb{K}[[\mathbf{u}]]$ together with the usual addition and multiplication form a Unique Factorization Domain (UFD). Now, we consider all the formal infinite Laurent series of the form

$$f(\mathbf{x}) := \sum_{\mathbf{k}} a_{\mathbf{k}} \mathbf{x}^{\mathbf{k}},$$

where the sum runs over all $\mathbf{k} = (k_1, \ldots, k_p) \in \mathbb{Z}^p$, the $a_{\mathbf{k}}$ are elements of $\mathbb{K}$, and $\mathbf{x}^{\mathbf{k}} = x_1^{k_1} \cdots x_p^{k_p}$. These objects together with the natural addition and scalar multiplication form a vector space over $\mathbb{K}$. We define the support of the Laurent series f by

$$\operatorname{supp}(f(\mathbf{x})) := \{\mathbf{k} \in \mathbb{Z}^p \mid a_{\mathbf{k}} \neq 0\}.$$

Notation 4. *Let $C \subseteq \mathbb{R}^p$ be a line-free cone. Then, we denote by $\mathbb{K}_C[[\mathbf{x}]]$ the set of the Laurent series f over $\mathbb{K}$ and with variables $\mathbf{x}$ so that we have:*

$$\operatorname{supp}(f(\mathbf{x})) \subseteq C.$$

Theorem 2 (see [25])**.** *$\mathbb{K}_C[[\mathbf{x}]]$ together with the natural addition and multiplication is an integral domain.*

Theorem 3 (see [25])**.** *Let $C \subseteq \mathbb{R}^p$ be a line-free cone and $f(\mathbf{x}) = \sum_{\boldsymbol{k}} a_{\boldsymbol{k}} \mathbf{x}^{\boldsymbol{k}} \in \mathbb{K}_C[[\mathbf{x}]]$. Then, there exists $g(\mathbf{x}) \in \mathbb{K}_C[[\mathbf{x}]]$ with $f(\mathbf{x})g(\mathbf{x}) = 1$, if and only if $a_{\boldsymbol{0}} \neq 0$.*

Definition 4. *Let $\preceq$ be an additive order in $\mathbb{Z}^p$. Then, we define the sets*

$$\mathbb{K}_{\preceq}[[\mathbf{x}]] := \bigcup_{C \in \mathcal{C}} \mathbb{K}_C[[\mathbf{x}]] \quad \text{and} \quad \mathbb{K}_{\preceq}((\mathbf{x})) := \bigcup_{\boldsymbol{e} \in \mathbb{Z}^p} \mathbf{x}^{\boldsymbol{e}} \mathbb{K}_{\preceq}[[\mathbf{x}]],$$

where $\mathcal{C}$ is the set of all cones $C \subseteq \mathbb{R}^p$ which are compatible with $\preceq$, and

$$\mathbf{x}^{\boldsymbol{e}} \mathbb{K}_{\preceq}[[\mathbf{x}]] := \bigcup_{C \in \mathcal{C}} \mathbf{x}^{\boldsymbol{e}} \mathbb{K}_C[[\mathbf{x}]].$$

Theorem 4 (see [25])**.** *If $\preceq$ is an additive order on $\mathbb{Z}^p$, then $\mathbb{K}_{\preceq}[[\mathbf{x}]]$ is a ring and $\mathbb{K}_{\preceq}((\mathbf{x}))$ is a field.*

3.3 Algorithms

In order to develop a complete implementation of a Laurent series object in MAPLE, we need to do at least the following: Choose an additive total order in $\mathbb{Z}^p$; define a way to represent a Laurent series in MAPLE; implement addition and multiplication of Laurent series; and finally, implement the inversion of a Laurent series. Subsections 3.3.1, 3.3.2, 3.3.3, and 3.3.4 take care of all these tasks, respectively.

3.3.1 Graded reverse lexicographic order

The first decision to be made is the election of the monomial order. For our implementation, we choose the well-known *graded reverse lexicographic order* or *grevlex order* for short, and we denote it by $<_{glex}$ (see [28]). The grevlex order compares first the total degree, then uses a reverse lexicographic order as tie-breaker. But it reverses the outcome of the lexicographic comparison so that lexicographically larger monomials of the same degree are considered to be the grevlex smaller. We denote grevlex-greater, grevlex-less or equal, grevlex-greater or equal, and grevlex-equal, respectively by $>_{glex}, \leq_{glex}, \geq_{glex}, =_{glex}$.

Also, notice that the grevlex order can be defined as a sequence of the weight vectors $\mathbf{1}_p := (1, 1, 1, \ldots, 1), \mathbf{1}_{p-1} := (1, 1, \ldots, 1, 0), \ldots, \mathbf{1}_1 := (1, 0, \ldots, 0)$. Indeed, let $\mathbf{v} = (v_1, \ldots, v_p) \in \mathbb{R}^p$, then for $i = 1, \ldots, p$, define the i-weight of $\mathbf{v}$ as $w_i(\mathbf{v}) := \mathbf{v} \cdot \mathbf{1}_i$. We also define the weight of $\mathbf{v}$ as $|\mathbf{b}| := w_p(\mathbf{v})$. Thus, if we have $\mathbf{v}_1 = (v_1, \ldots, v_p), \mathbf{v}_2 = (v'_1, \ldots, v'_p) \in \mathbb{R}^p$, and we wish to know if $\mathbf{v}_1 >_{glex} \mathbf{v}_2$, $\mathbf{v}_1 <_{glex} \mathbf{v}_2$ or $\mathbf{v}_1 =_{glex} \mathbf{v}_2$, we can apply the following algorithm:

Algorithm 8 GrevLexComparison

Require: $\mathbf{v}_1 = (v_1, \ldots, v_p), \mathbf{v}_2 = (v'_1, \ldots, v'_p) \in \mathbb{R}^p$.
Ensure: $\mathbf{v}_1 <_{glex} \mathbf{v}_2, \mathbf{v}_1 >_{glex} \mathbf{v}_2$ or $\mathbf{v}_1 =_{glex} \mathbf{v}_2$.

```
for i from p to 1 do
    if w_i(v_1) < w_i(v_2) then                ▷ i-weight Comparison.
        return v_1 <_glex v_2
    else if w_i(v_1) > w_i(v_2) then           ▷ i-weight Comparison.
        return v_1 >_glex v_2
    else
        next                                   ▷ We check the next weight.
return v_1 =_glex v_2
```

Lemma 6. *Let $\boldsymbol{v}_1, \boldsymbol{v}_2$ elements of $\mathbb{Z}^p$, then*

$$w_i(\boldsymbol{v}_1 + \boldsymbol{v}_2) = w_i(\boldsymbol{v}_1) + w_i(\boldsymbol{v}_2),$$

for $i = 1, \ldots, p$.

Proof. Set $i \in \{1, \ldots, p\}$. Then

$$w_i(\mathbf{v}_1 + \mathbf{v}_2) = (\mathbf{v}_1 + \mathbf{v}_2) \cdot \mathbf{1}_i = \mathbf{v}_1 \cdot \mathbf{1}_i + \mathbf{v}_2 \cdot \mathbf{1}_i = w_i(\mathbf{v}_1) + w_i(\mathbf{v}_2).$$

□

Now, we establish that $\leq_{glex}$ is additive.

Proposition 5. *The grevlex order is additive on $\mathbb{Z}^p$.*

Proof. Let $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$ elements of $\mathbb{Z}^p$, with $\mathbf{v}_1 \leq_{glex} \mathbf{v}_2$. If $\mathbf{v}_1 =_{glex} \mathbf{v}_2$, then the result follows directly. Indeed, by Lemma

$$w_i(\mathbf{v}_1 + \mathbf{v}_3) = w_i(\mathbf{v}_1) + w_i(\mathbf{v}_3) = w_i(\mathbf{v}_2) + w_i(\mathbf{v}_3) = w_i(\mathbf{v}_2 + \mathbf{v}_3),$$

for $i = 1, \ldots, p$, which implies that $\mathbf{v}_1 + \mathbf{v}_3 =_{glex} \mathbf{v}_2 + \mathbf{v}_3$.
Now, assume that $\mathbf{v}_1 <_{glex} \mathbf{v}_2$. Then, there exist i in $\{1, \ldots, p\}$, such that

$$w_i(\mathbf{v}_1) < w_i(\mathbf{v}_2), \text{ and } w_j(\mathbf{v}_1) = w_j(\mathbf{v}_2) \text{ for } j = i+1, \ldots, p.$$

Hence, by the additivity of $<$ and $=$ in $\mathbb{Z}$, and Lemma 3.3.1, we have:

$$\begin{aligned} &w_i(\mathbf{v}_1) + w_i(\mathbf{v}_3) < w_i(\mathbf{v}_2) + w_i(\mathbf{v}_3), \text{ and} \\ &w_j(\mathbf{v}_1) + w_i(\mathbf{v}_3) = w_j(\mathbf{v}_2) + w_i(\mathbf{v}_3) \text{ for } j = i+1, \ldots, p. \end{aligned}$$

Thus,

$$\begin{aligned} &w_i(\mathbf{v}_1 + \mathbf{v}_3) < w_i(\mathbf{v}_2 + \mathbf{v}_3), \text{ and} \\ &w_j(\mathbf{v}_1 + \mathbf{v}_3) = w_j(\mathbf{v}_2 + \mathbf{v}_3) \text{ for } j = i+1, \ldots, p. \end{aligned}$$

Consequetly, we have: $\mathbf{v}_1 + \mathbf{v}_3 <_{glex} \mathbf{v}_2 + \mathbf{v}_3$. □

Example 2. *Set $\boldsymbol{v}_1 = (1, 0, -1)$, $\boldsymbol{v}_2 = (0, 0, 0)$, $\boldsymbol{v}_3 = (1, 1, -1)$, and $\boldsymbol{v}_4 = (2, -1, -1)$. Then,*

1. *Since the weight of $\boldsymbol{v}_3$ is 1, and the weight of $\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_4$ is 0, we observe that*

$$\boldsymbol{v}_3 >_{glex} \boldsymbol{v}_i, \text{ with } i = 1, 2, 4.$$

2. *Now, we can compute the 2-weight of $\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_4$, and we get $1, 0, 1$, respectively. Thus, $\boldsymbol{v}_2 <_{glex} \boldsymbol{v}_i$ with $i = 1, 4$. Finally, the 1-weights of $\boldsymbol{v}_1, \boldsymbol{v}_4$, are, respectively, $1, 2$. Hence, $\boldsymbol{v}_1 <_{glex} \boldsymbol{v}_4$. In summary,*

$$\boldsymbol{v}_2 <_{glex} \boldsymbol{v}_1 <_{glex} \boldsymbol{v}_4 <_{glex} \boldsymbol{v}_3.$$

Remark 2. *Notice that if $\boldsymbol{v} \in \mathbb{Z}^p$ is a grevlex non-negative vector with zero weight (that is, $w_p(\boldsymbol{v}) = 0$) then for any integer $s \geq 0$, the vector $s\boldsymbol{v}$ is also a grevlex non-negative vector with weight equal to 0. This property makes it challenging to work with a cone C, counting, within its generating set, at least one ray with weight zero. This observation will be illustrated with Example 4 of the following section.*

3.3.2 The Laurent series object

Our objective is to encode Laurent Series as objects (to be referred as LSO in the sequel of this paper) in MAPLE. In order to achieve this, we will make use of the existing `MultitivariatePowerSeries` package (see [5]).

Recall that $\mathbb{K}$ is a field and that we work with two groups of variables $\mathbf{x} = x_1, \ldots, x_p$ and $\mathbf{u} = u_1, \ldots, u_m$. To keep the presentation of this document as simple as possible, we shall assume that $m \geq p$ holds. However, our MAPLE implementation, within the `MultitivariatePowerSeries` package, does not make that assumption, and thus can handle the case where $m < p$ holds. By Theorem 4, to encode a Laurent series $f \in \mathbb{K}_{\leq}((\mathbf{x}))$, it is enough to find a set of grevlex non-negative rays $\mathbf{R} := \{\mathbf{r}_1, \ldots, \mathbf{r}_m\} \subset \mathbb{Z}^p$, and $\mathbf{e} \in \mathbb{Z}^p$ such that $f \in \mathbf{x}^{\mathbf{e}}\mathbb{K}_C[[\mathbf{x}]]$, where C the cone generated by $\mathbf{R}$. Thus, we need first to encode $\mathbb{K}_C[[\mathbf{x}]]$.

Now notice that the cone C can also represent a change of variables. Indeed, consider the ring of multivariate power series $\mathbb{K}[[\mathbf{u}]]$, where $\mathbf{u} := (u_1, \ldots, u_m)$ are ordered indeterminates.

Also, consider the change of variables $u_i := \mathbf{x}^{\mathbf{r}_i}$. Then, for every $g \in \mathbb{K}[[\mathbf{u}]]$, we observe that $g(u_1 = \mathbf{x}^{\mathbf{r}_1}, \ldots, u_m := \mathbf{x}^{\mathbf{r}_m})$ is an element of $\mathbb{K}_C[[\mathbf{x}]]$. Next, observe that the monomial $\mathbf{x}^{\mathbf{e}}$, multiplying our LSO, can simply be seen as an $\mathbf{e}$. Our idea is to encode a Laurent series object $f \in \mathbb{K}_C[[\mathbf{x}]]$ as a power series $g \in \mathbb{K}[[\mathbf{u}]]$ together with a set of rays $\mathbf{R} := \{\mathbf{r}_1, \ldots, \mathbf{r}_m\} \subset \mathbb{Z}^p$ and the point $\mathbf{e} \in \mathbb{Z}^p$. In summary,

Proposition 6. *Let $g \in \mathbb{K}[[\boldsymbol{u}]]$ be a power series, $\boldsymbol{e} \in \mathbb{Z}^p$ be a point, and $\boldsymbol{R} := \{\boldsymbol{r}_1, \ldots, \boldsymbol{r}_m\} \subset \mathbb{Z}^p$ be set of grevlex non-negative rays. Then,*

$$f = \mathbf{x}^{\boldsymbol{e}} g(\mathbf{x}^{\boldsymbol{r}_1}, \ldots, \mathbf{x}^{\boldsymbol{r}_m}),$$

is a Laurent series living in $\mathbf{x}^{\boldsymbol{e}}\mathbb{K}_C[[\mathbf{x}]]$, where C is the cone generated by $\boldsymbol{R}$.

Remember that the rays used to generate the cone C, of our object, must be grevlex non-negative. Hence, Lemma 4 guaranties that our cone will be line-free and compatible with $\geq_{glex}$. Thus, $\mathbb{K}_C[[\mathbf{x}]]$ is an integral domain.

Example 3. *Consider*

$$f := x^{-4}y^5 \sum_{i=0}^{\infty} x^{2i}y^{-i}.$$

If we want to encode f as a LSO, we can choose the following parameters:

$$\begin{aligned} \mathbf{x} &= [x, y], \\ \boldsymbol{u} &= [u, v], \\ g &= \mathsf{Inverse}(\mathsf{PowerSeries}(1 + uv)), \\ \boldsymbol{r} &= [[1, 0], [1, -1]], \\ \boldsymbol{e} &= [x = -4, y = 5], \\ C &= Cone(\boldsymbol{R}). \end{aligned}$$

Notice that $[1, 0] \geq_{glex} [0, 0]$ and $[1, -1] \geq_{glex} [0, 0]$, so the ring $\mathbb{K}_C[[\mathbf{x}]]$ is well defined. Also, the cone C generates a change of variable equal to $u = x$ and $v = xy^{-1}$.

We can use Maple*'s command* MultitivariatePowerSeries:-Truncate *to obtain the terms of the power series g up to a specific degree. As an example, if we type* Truncate$(g, 10)$*, we get*

$$g(u, v) = 1 - uv + u^2v^2 - u^3v^3 + u^4v^4 - u^5v^5.$$

Now, if we apply our change of variables , we see that

$$g(x, xy^{-1}) = 1 - x^2v^{-1} + u^4v^{-2} - u^6v^{-3} + u^8v^{-4} - u^{10}v^{-5}.$$

Thus, $x^{-4}y^5g(x, xy^{-1})$ will produce the first terms of our Laurent series f.

For simplicity, we define $\mathbf{x}^{\mathbf{R}} := \mathbf{x}^{\mathbf{r}_1}, \ldots, \mathbf{x}^{\mathbf{r}_m}$ and also $\overline{\mathbf{R}} = (\mathbf{r}_1^T, \ldots, \mathbf{r}_m^T)$, where $A \longmapsto A^T$ denotes matrix transposition.

Remark 3. *Let $g \in \mathbb{K}[[\boldsymbol{u}]]$ be a power series, and $\boldsymbol{R} := \{\boldsymbol{r}_1, \ldots, \boldsymbol{r}_m\} \subset \mathbb{Z}^p$ be set of grevlex non-negative rays. Then, the following identities follow immediately from the notations introduced above.*

1. $g(\mathbf{x}^{\boldsymbol{R}}) = g(\mathbf{x}^{\boldsymbol{r}_1}, \ldots, \mathbf{x}^{\boldsymbol{r}_m})$.
2. $\left(\mathbf{x}^{\boldsymbol{R}}\right)^{\boldsymbol{k}} = \mathbf{x}^{k_1\boldsymbol{r}_1} \cdots \mathbf{x}^{k_m\boldsymbol{r}_m} = \mathbf{x}^{k_1\boldsymbol{r}_1 + \cdots + k_m\boldsymbol{r}_m} = \mathbf{x}^{\overline{\boldsymbol{R}} \cdot \boldsymbol{k}^T}$ *for all* $\boldsymbol{k} = (k_1, \ldots, k_m) \in \mathbb{Z}^m$.
3. $\boldsymbol{u} = \mathbf{x}^{\boldsymbol{R}} \Leftrightarrow u_i = \mathbf{x}^{\boldsymbol{r}_i}$ *for all* $i = 1, \ldots, m$.

3.3.3 Addition and multiplication

Throughout this subsection, let $C_1 \subseteq \mathbb{Z}^p$ be a line-free cone given by a set of rays, $\mathbf{R}_1 := \{\mathbf{r}'_1, \ldots, \mathbf{r}'_m\} \subset \mathbb{Z}^p$, and $C_2 \subseteq \mathbb{Z}^p$ be a line-free cone given by a set of rays, $\mathbf{R}_2 := \{\mathbf{r}''_1, \ldots, \mathbf{r}''_m\} \subset \mathbb{Z}^p$, with $m \geq p$. Let also $\mathbf{e}_1, \mathbf{e}_2$ be two points in $\mathbb{Z}^p$. Now, if we want to add or multiply two Laurent series, $f_1 \in \mathbf{x}^{\mathbf{e}_1}\mathbb{K}_{C_1}[[\mathbf{x}]]$ and $f_2 \in \mathbf{x}^{\mathbf{e}_2}\mathbb{K}_{C_2}[[\mathbf{x}]]$, we must first compute a line-free cone C containing the union of C_1 and C_2. Suppose that

$$f_1 = \mathbf{x}^{\mathbf{e}_1} g_1(\mathbf{x}^{\mathbf{R}_1}) \text{ and } f_2 = \mathbf{x}^{\mathbf{e}_2} g_2(\mathbf{x}^{\mathbf{R}_2}),$$

with $g_1, g_2 \in \mathbb{K}[[\mathbf{u}]]$. Note that, in order to simplify the presentation, we are assuming that $\mathbf{R}_1$ and $\mathbf{R}_2$ have the same number of rays $m \geq p$. The general case is a direct consequence of the procedures presented in this document, and has already been implemented in MAPLE.

Next, note that we have:

$$f_1 f_2 = (\mathbf{x}^{\mathbf{e}_1}\mathbf{x}^{\mathbf{e}_2})\left(g_1(\mathbf{x}^{\mathbf{R}_1})g_2(\mathbf{x}^{\mathbf{R}_2})\right) = \mathbf{x}^{\mathbf{e}_1+\mathbf{e}_2}\left(g_1(\mathbf{x}^{\mathbf{R}_1})g_2(\mathbf{x}^{\mathbf{R}_2})\right).$$

Thus, we can set the point $\mathbf{e}$, of our new Laurent series, simply as $\mathbf{e}_1 + \mathbf{e}_2$. The selection of the rays $\mathbf{R}$ for the cone C is, however, not as easy. Since $g_1(\mathbf{x}^{\mathbf{R}_1})$ and $g_2(\mathbf{x}^{\mathbf{R}_2})$ are Laurent series, respectively living in $\mathbb{K}_{C_1}[[\mathbf{x}]]$ and $\mathbb{K}_{C_2}[[\mathbf{x}]]$, we must compute a line-free cone C that contains the union of C_1 and C_2. Then, we would deduce $g_1(\mathbf{x}^{\mathbf{R}_1})g_2(\mathbf{x}^{\mathbf{R}_2}) \in \mathbb{K}_C[[\mathbf{x}]]$ and $f_1 f_2 \in \mathbf{x}^{\mathbf{e}}\mathbb{K}_C[[\mathbf{x}]]$.

Note that, for the case of addition, we can do a similar trick. Let the point $\mathbf{e}$ be the grevlex-minimum between $\mathbf{e}_1$ and $\mathbf{e}_2$. Without lost of generality, assume $\mathbf{e} = \mathbf{e}_1$. Then, we can re-write $f_1 + f_2$ as

$$f_1 + f_2 = \mathbf{x}^{\mathbf{e}}\left(g_1(\mathbf{x}^{\mathbf{R}_1}) + \mathbf{x}^{\mathbf{e}_2-\mathbf{e}}g_2(\mathbf{x}^{\mathbf{R}_2})\right).$$

Observe that $\mathbf{e}_2 - \mathbf{e}$ is grevlex non-negative. Thus, searching for a cone C such that $f_1 + f_2 \in \mathbf{x}^{\mathbf{e}}\mathbb{K}_C[[\mathbf{x}]]$, is equivalent to look for a cone C such that

$$g_1(\mathbf{x}^{\mathbf{R}_1}) + \mathbf{x}^{\mathbf{e}_2-\mathbf{e}}g_2(\mathbf{x}^{\mathbf{R}_2}) \in \mathbb{K}_C[[\mathbf{x}]].$$

Hence, it is enough to find a cone C that contains the cone generated by the set of rays $\mathbf{R}_1 \cup \mathbf{R}_2 \cup \{\mathbf{e}_2 - \mathbf{e}\}$.

Note that by Lemma 4, we can simply choose the cone C to be $C_1 + C_2$ for the multiplication, and the cone generated by the rays $\mathbf{R}_1 \cup \mathbf{R}_2 \cup \{\mathbf{e}_2 - \mathbf{e}\}$, for the case of the addition. However, since we desire to keep the number of rays associated to a given Laurent series to a minimum (for computational efficiency purposes), we develop Algorithm 9, which takes a set of grevlex non-negative rays and an order $\mathbf{x} = (x_1, \ldots, x_p)$ as input, and returns a minimal set of p grevlex non-negative rays. The cone generated by this new set of rays contains the cones generated by the input rays. As a consequence, we will be able to write $\mathbf{R}_1$ and $\mathbf{R}_2$ in terms of $\mathbf{R}$.

In algorithm 9, LookForGreatestGrevlexLess takes a rational vector as input and returns an integer vector that is grevlex less or equal than the input and that has the same weight as it. Note that this algorithm is based in Algorithm 8. Observe also that algorithm 9 proceeds as follows:

1. while i varies between p and 1, we select all the rays in W with positive $i-$weight;
2. then, we make all these rays comparable by dividing them by their norm; after this, we select the grevlex-smallest one between them, say v;

Algorithm 9 MakeRaysCompatible

Require: $\mathbf{R}_1$, $\mathbf{R}_2$, $\mathbf{R}_3$ sets of grevlex-non negative rays, a field $\mathbb{K}$, ordered indeterminates $\mathbf{x} := (x_1, \ldots, x_p)$.
Ensure: A set of p rays $\mathbf{R}$ such that $C(\mathbf{R}) \supseteq C(\mathbf{R}_1) \cup C(\mathbf{R}_2) \cup C(\mathbf{R}_3)$.

1: $W := \mathbf{R}_1 \cup \mathbf{R}_2 \cup \mathbf{R}_3$
2: **for** i **from** p **to** 1 **do**
3: $S := \{\mathbf{w} \in W \mid \mathbf{w} \cdot \mathbf{1}_i > 0\}$ ▷ We get all the rays in W with positive i-weight.
4: **if** $|S| > 1$ **then** ▷ $|S|$ denotes the number of elements in S.
5: $S' := \{\mathbf{s}/(\mathbf{s} \cdot \mathbf{1}_i) \mid \mathbf{s} \in S\}$ ▷ We "normalize" the elements of S
6: ▷ to make them comparable.
7: $\mathbf{v} := \min_{glex}(S')$ ▷ We get the grevlex smallest element of S'.
8: $\mathbf{v}' :=$ LookForGreatestGrevlexLess($\mathbf{v}$) ▷ $\mathbf{v} \in \mathbb{Q}^p$, so we look for $\mathbf{v}' \in \mathbb{Z}^p$
9: ▷ such that $\mathbf{v} \geq_{glex} \mathbf{v}'$ and $|\mathbf{v}| = |\mathbf{v}'|$.
10: **else**
11: $\mathbf{v}' := S[1]$
12:
13: $\mathbf{R}[i] := \mathbf{v}'$ ▷ We save $\mathbf{v}'$.
14: **for** $\mathbf{w} \in W$ **do**
15: **if** $\mathbf{w} \in S$ **then**
16: $\mathbf{w} := \mathbf{w} - \frac{\mathbf{w} \cdot \mathbf{1}_i}{\mathbf{R}[i] \cdot \mathbf{1}_i} \cdot \mathbf{R}[i]$ ▷ We subtract a multiple of $\mathbf{R}[i]$
17: ▷ to achieve $\mathbf{w} \cdot \mathbf{1}_i = 0$.
18: **return R**

3. if v is not an integer vector, we replace v by an integer and grevlex non-negative vector v' such that $\mathbf{v} \geq_{glex} \mathbf{v}'$ and $|\mathbf{v}| = |\mathbf{v}'|$; then v is renamed $\mathbf{R}[i]$;
4. the last step of this iteration is to subtract the biggest possible multiple of $\mathbf{R}[i]$ from the rays selected in W with positive $i-$weight such that the different is grevlex non-negative.

By doing this last step, we guarantee that in the next iteration, $i = i + 1$, all the rays in W are grevlex less or equal to $\mathbf{R}[i]$. Thus, by following this process we obtain a list of p rays such that $\mathbf{R}[p] \geq_{glex} \mathbf{R}[p-1] \geq_{glex} \ldots \geq_{glex} \mathbf{R}[1]$. Also, every ray in $\mathbf{R}_1, \mathbf{R}_2$ and $\mathbf{R}_3$ can be written as a linear combination of the rays in $\mathbf{R}$.

Remark 4. *Observe that at Line 17, of Algorithm 9, the fraction* $\frac{\mathbf{w} \cdot \mathbf{1}_i}{\mathbf{R}[i] \cdot \mathbf{1}_i}$ *is always going to be integer. Indeed, if* $|S| > 1$*, then* $v \cdot \mathbf{1}_i = 1$*, and in consequence* $v' \cdot \mathbf{1}_i = 1$*. Thus,* $\mathbf{R}[i] \cdot \mathbf{1}_i = 1$ *as well. On the other hand, if* $|S| = 1$*, then* $\frac{\mathbf{w} \cdot \mathbf{1}_i}{\mathbf{R}[i] \cdot \mathbf{1}_i} = \frac{\mathbf{v} \cdot \mathbf{1}_i}{\mathbf{v} \cdot \mathbf{1}_i} = 1$.

Proposition 7. *Every vector in* $W = \mathbf{R}_1 \cup \mathbf{R}_2 \cup \mathbf{R}_3$ *is a non-negative integer linear combination of the vectors in* $\mathbf{R}$*, as computed in Algorithm 9.*

Proof. Let C_i be the integer cone generated by W and $\mathbf{R}$ at the beginning of each iteration of Algorithm 9, i.e.,

$$C_i := \left\{ \sum_{w \in W} a_w w + \sum_{r \in \mathbf{R}} b_r r \mid a_w \geq 0, b_r \geq 0 \in \mathbb{Z} \right\},$$

with $i \in \{p, p-1, \ldots, 1\}$. Observe that in the first iteration $\mathbf{R}$ is empty, and also that in the last iteration W is empty. Now, notice that $C_i \subseteq C_{i-1}$, by construction. Thus, we have that

the integer cone, C_p, generated by W is contained inside the integer cone, C_1, generated by $\mathbf{R}$. Finally we see that Remark 4 implies that in each iteration of Algorithm 9, we are subtracting from W and integer multiple of $\mathbf{R}_i$. Hence, the vectors of W are a non-negative integer linear combination of the vectors in $\mathbf{R}$. □

Algorithm 10 Multiply

Require: Laurent series $f_1(\mathbf{x}) = \mathbf{x}^{\mathbf{e}_1} g_1(\mathbf{x}^{\mathbf{R}_1})$, $f_2(\mathbf{x}) = \mathbf{x}^{\mathbf{e}_2} g_2(\mathbf{x}^{\mathbf{R}_2})$. Remember $\mathbf{R}_1 := \{\mathbf{r}'_1, \ldots, \mathbf{r}'_m\} \subset \mathbb{Z}^p$ and $\mathbf{R}_2 := \{\mathbf{r}''_1, \ldots, \mathbf{r}''_m\} \subset \mathbb{Z}^p$.
Ensure: $\mathbf{x}^{\mathbf{e}} g(\mathbf{x}^{\mathbf{R}})$ the product of f_1 and f_2.

$\mathbf{e} := \mathbf{e}_1 + \mathbf{e}_2$ ▷ We get the exponent of $\mathbf{x}^{\mathbf{e}_1}\mathbf{x}^{\mathbf{e}_2}$.
$\mathbf{R} :=$ MakeRaysCompatible$([\mathbf{R}_1, \mathbf{R}_2], \mathbf{x})$ ▷ We get the rays of a cone C such that
▷ $C_1 \cup C_2 \subseteq C$.
$\mathbf{u}' := \mathbf{x}^{\mathbf{R}}$ ▷ We compute the new change of variable.

for i **from** 1 **to** m **do** ▷ We see $g_1(\mathbf{u})$ and $g_2(\mathbf{u})$ as a function of $\mathbf{u}'$:
  Solve$(\mathbf{r}'_i = \overline{\mathbf{R}} \cdot \mathbf{k}_i^T)$ ▷ We compute $\mathbf{k}_i$ such that $\mathbf{r}'_i = \overline{\mathbf{R}} \cdot \mathbf{k}_i^T$.
  Solve$(\mathbf{r}''_i = \overline{\mathbf{R}} \cdot (\mathbf{k}'_i)^T)$ ▷ We compute $\mathbf{k}'_i$ such that $\mathbf{r}''_i = \overline{\mathbf{R}} \cdot (\mathbf{k}'_i)^T$.
$g'_1(\mathbf{u}') := g_1((\mathbf{u}')^{\mathbf{k}_1}, \ldots, (\mathbf{u}')^{\mathbf{k}_m})$ ▷ $g'_1(\mathbf{u}') \in \mathbb{K}[[\mathbf{u}']]$.
$g'_2(\mathbf{u}') := g_2((\mathbf{u}')^{\mathbf{k}'_1}, \ldots, (\mathbf{u}')^{\mathbf{k}'_m})$ ▷ $g'_2(\mathbf{u}') \in \mathbb{K}[[\mathbf{u}']]$.
$g := g'_1 g'_2$ ▷ We multiply Power Series in $\mathbb{K}[[\mathbf{u}']]$.
return $\mathbf{x}, \mathbf{u}', g, \mathbf{R}, \mathbf{e}$

Algorithm 11 Addition

Require: Laurent series $f_1(\mathbf{x}) = \mathbf{x}^{\mathbf{e}_1} g_1(\mathbf{x}^{\mathbf{R}_1})$, $f_2(\mathbf{x}) = \mathbf{x}^{\mathbf{e}_2} g_2(\mathbf{x}^{\mathbf{R}_2})$. Remember $\mathbf{R}_1 := \{\mathbf{r}'_1, \ldots, \mathbf{r}'_m\} \subset \mathbb{Z}^p$ and $\mathbf{R}_2 := \{\mathbf{r}''_1, \ldots, \mathbf{r}''_m\} \subset \mathbb{Z}^p$.
Ensure: $\mathbf{x}^{\mathbf{e}} g(\mathbf{x}^{\mathbf{R}})$ the addition between f_1 and f_2.

$\mathbf{e} := \min_{grevlex}(\mathbf{e}_1 + \mathbf{e}_2)$ ▷ We get the grevlex-smallest exponent.
if $\mathbf{e} \neq \mathbf{e}_1$ **then** ▷ We assume $\mathbf{e} = \mathbf{e}_1$.
  return Addition(f_2, f_1)
$\mathbf{R} :=$ MakeRaysCompatible$([\mathbf{R}_1, \mathbf{R}_2, \mathbf{e}_2 - \mathbf{e}], \mathbf{x})$ ▷ We get the rays of a cone C such that
▷ $C_1 \cup C_2 \cup \{\mathbf{e}_2 - \mathbf{e}\} \subseteq C$.
$\mathbf{u}' := \mathbf{x}^{\mathbf{R}}$ ▷ We compute the new change of variable.

for i **from** 1 **to** m **do** ▷ We see $g_1(\mathbf{u})$ and $g_2(\mathbf{u})$ as a function of $\mathbf{u}'$:
  Solve$(\mathbf{r}'_i = \overline{\mathbf{R}} \cdot \mathbf{k}_i^T)$ ▷ We compute $\mathbf{k}_i$ such that $\mathbf{r}'_i = \overline{\mathbf{R}} \cdot \mathbf{k}_i^T$.
  Solve$(\mathbf{r}''_i = \overline{\mathbf{R}} \cdot (\mathbf{k}'_i)^T)$ ▷ We compute $\mathbf{k}'_i$ such that $\mathbf{r}''_i = \overline{\mathbf{R}} \cdot (\mathbf{k}'_i)^T$.
Solve$(\mathbf{e}_2 - \mathbf{e} = \overline{\mathbf{R}} \cdot \mathbf{k}^T)$ ▷ We compute $\mathbf{k}$ such that $\mathbf{e}_2 - \mathbf{e} = \overline{\mathbf{R}} \cdot \mathbf{k}^T$.
$g'_1(\mathbf{u}') := g_1((\mathbf{u}')^{\mathbf{k}_1}, \ldots, (\mathbf{u}')^{\mathbf{k}_m})$ ▷ $g'_1(\mathbf{u}') \in \mathbb{K}[[\mathbf{u}']]$.
$g'_2(\mathbf{u}') := g_2((\mathbf{u}')^{\mathbf{k}'_1}, \ldots, (\mathbf{u}')^{\mathbf{k}'_m})$ ▷ $g'_2(\mathbf{u}') \in \mathbb{K}[[\mathbf{u}']]$.
$m(\mathbf{u}') = (\mathbf{u}')^{\mathbf{k}}$ ▷ $m(\mathbf{u}') \in \mathbb{K}[[\mathbf{u}']]$.
$g := g'_1 + m \cdot g'_2$ ▷ We do power series operations in $\mathbb{K}[[\mathbf{u}']]$.
return $\mathbf{x}, \mathbf{u}', g, \mathbf{R}, \mathbf{e}$

Now, if we want to compute the product of f_1 and f_2, then, we follow Algorithm 10. The addition is a little more complicate than the multiplication, see Algorithm 11 for details. It follows, however, similar ideas as Algorithm 10. In Algorithm 11, it is important to notice that $\mathbf{e}_2 \geq_{glex} \mathbf{e}$ implies that $\mathbf{e}_2 - \mathbf{e}$ is grevlex non-negative. So, when we apply MakeRaysCompatible($[\mathbf{R}_1, \mathbf{R}_2, \mathbf{e}_2 - \mathbf{e}], \mathbf{x}$), we are looking for a cone that contains the union of the cones associated to f_1 and f_2 and also the cone generated by the ray $\mathbf{e}_2 - \mathbf{e}$. Observe also that Proposition 7 guarantees that the vectors $\mathbf{k}_i$'s and $\mathbf{k}'_i$'s have non-negative components. In consequence, g'_1, g'_2 and m are going to be well-defined as multivariate power series. A similar argument can be used in Algorithm 10.

3.3.4 Inversion

From now and through this subsection, let $C \subseteq \mathbb{Z}^p$ be a line-free cone described by the set of rays, $\mathbf{R} := \{\mathbf{r}_1, \ldots, \mathbf{r}_m\} \subset \mathbb{Z}^p$, and let $\mathbf{e} \in \mathbb{Z}^p$ be a point. Now, let $f \in \mathbf{x}^{\mathbf{e}}\mathbb{K}_C[[\mathbf{x}]]$ be a non-zero Laurent series, with

$$f = \mathbf{x}^{\mathbf{e}} g(\mathbf{x}^{\mathbf{R}}),$$

where $g \in \mathbb{K}[[\mathbf{u}]]$ is a power series. We wish to compute the inverse of f. Note that the power series g is invertible if and only if it has a constant term (see [19] for details). Hence, f can be easily inverted

$$f^{-1} = \mathbf{x}^{-\mathbf{e}} g^{-1}(\mathbf{x}^{\mathbf{R}}).$$

On the other hand, if g does not have a constant term, then the power series g is not invertible. The Laurent series $g(\mathbf{x}^{\mathbf{R}})$, however, is invertible. Following the ideas of Monforte and Kauers, from [25], we would like to look for the grevlex-minimum element of the support of $g(\mathbf{x}^{\mathbf{R}})$, say $\bar{\mathbf{e}}$. Then we would like to factor $\mathbf{x}^{\bar{\mathbf{e}}}$ out of $g(\mathbf{x}^{\mathbf{R}})$. Note that this would imply that the support of $g(\mathbf{x}^{\mathbf{R}})/\mathbf{x}^{\bar{\mathbf{e}}}$ is still formed by grevlex non-negative elements. Here we find the first challenge of our implementation: the set $\mathrm{supp}(g(\mathbf{x}^{\mathbf{R}}))$ could be infinite and not increasingly ordered. To better describe the problem, first we need to explain the relationship between the support of g and the support of $g(\mathbf{x}^{\mathbf{R}})$. Secondly, we describe the main ways of defining a multivariate power series in Maple.

The minimum grevlex element of $\mathrm{supp}(g(\mathbf{x}^{\mathbf{R}}))$

Lemma 7. *Let $C \subseteq \mathbb{Z}^p$ be a line-free cone described by a set of grevlex non-negative rays $\boldsymbol{R} := \{\boldsymbol{r}_1, \ldots, \boldsymbol{r}_m\} \subset \mathbb{Z}^p$. Let $\boldsymbol{e} \in \mathbb{Z}^p$ be a point. Now, let $f \in \mathbf{x}^{\boldsymbol{e}}\mathbb{K}_C[[\mathbf{x}]]$ be a Laurent series, with*

$$f = \mathbf{x}^{\boldsymbol{e}} g(\mathbf{x}^{\boldsymbol{R}}),$$

where $g \in \mathbb{K}[[\boldsymbol{u}]]$. Then,

$$\mathrm{supp}(g(\mathbf{x}^{\boldsymbol{R}})) = \{(\boldsymbol{r}_1^T, \ldots, \boldsymbol{r}_m^T) \cdot \boldsymbol{k}^T \mid \boldsymbol{k} \in \mathrm{supp}(g)\} \subseteq \mathbb{Z}^p.$$

Proof. Indeed, suppose $g(\mathbf{u}) = \sum_{\mathbf{k}\in\mathbb{Z}^m} a_{\mathbf{k}}\mathbf{u}^{\mathbf{k}} = \sum_{\mathbf{k}\in\mathbb{Z}^m} a_{\mathbf{k}} u_1^{k_1} \dots u_m^{k_m}$, then,

$$\begin{aligned} g(\mathbf{x}^{\mathbf{R}}) = g(\mathbf{x}^{\mathbf{r}_1}, \dots, \mathbf{x}^{\mathbf{r}_m}) &= \sum_{\mathbf{k}\in\mathbb{Z}^p} a_{\mathbf{k}} (\mathbf{x}^{\mathbf{r}_1})^{k_1} \dots (\mathbf{x}^{\mathbf{r}_m})^{k_m} \\ &= \sum_{\mathbf{k}\in\mathbb{Z}^p} a_{\mathbf{k}} (\mathbf{x}^{k_1\mathbf{r}_1}) \dots (\mathbf{x}^{k_m\mathbf{r}_m}) \\ &= \sum_{\mathbf{k}\in\mathbb{Z}^p} a_{\mathbf{k}} \mathbf{x}^{k_1\mathbf{r}_1+\dots+k_m\mathbf{r}_m}. \end{aligned}$$

Then, we can re-write $k_1\mathbf{r}_1 + \dots + k_m\mathbf{r}_m$ as $(\mathbf{r}_1^T, \dots, \mathbf{r}_m^T) \cdot \mathbf{k}^T$ using matrix notation. Thus, by definition of support, the result follows. □

Observe that Lemma 7 implies that knowing the support of the power series g is equivalent to know the support of the Laurent series f. Therefore, looking for the grevlex-minimum element of supp(f) can be done using the supp(g). Thus, how we access the monomials in g is a key.

Multivariate power series in MAPLE are created in a lazy manner ([5]). This means that computations are done only when they are necessary and only once. Thanks to this paradigm, we compute the terms of a multivariate power series using its degree as a iterator, i.e., we truncate the series up to a specific degree, and then MAPLE computes all the terms in our power series up to that degree once and stores them. This means that, if we ask for those terms a second time, MAPLE needs not to re-compute them. Since we use power series as a foundation of our implementation of Laurent series, we must use the same paradigm of lazy evaluation for Laurent series. To be more precise, when a MAPLE end-user truncates a Laurent series f, then (1) the underlying power series g must be truncated, (2) the composition of $\mathbf{x}^{\mathbf{R}}$ with those terms must be computed, and (3) the whole thing is multiplied by $x^{\mathbf{e}}$. The above has a few implications with respect to the computation of the minimum element:

1. In general, we can not find the smallest monomial of a power series easily. For instance, imagine that in the internal power series all the elements up to degree $10^{10000000}$ are zero. Finding this first element would take a while, and we will probably give up before that happens. Also, if the power series is the trivial one (and we do not know it), then we would never find this minimum element.
2. Finding the minimum element of a power series does not guarantee that we can find the grevlex-minimum element of a Laurent series. To illustrate this, consider the following examples.

Example 4. *Let $g := u^3 + u^2 + u + v^{25} \in \mathbb{K}[[u, v]]$, $\boldsymbol{R} = \{(1, 0), (1, -1)\}$, $\boldsymbol{e} = (0, 0)$, and $\boldsymbol{x} = (x, y)$. Then, $u = x$ and $v = xy^{-1}$, and $f = x^3 + x^2 + x + (xy^{-1})^{25}$. If we look for the monomial in g with minimum degree, we get u. This term corresponds to x in the Laurent series f. However, the grevlex-minimum element of f is $(xy^{-1})^{25}$.*

Example 4 shows the first challenge of our implementation. Whenever we have a zero-ray, i.e., a ray with the sum of its components equal to zero, we can not guarantee that the grevlex-minimum element is going to be found. Let us further illustrate this problem with a second example.

Example 5. *Consider a power series* $g \in \mathbb{K}[[u, v]]$ *with support equal to*

$$\{(0,0),(1,1),(1,2),(1,4),(2,2),(2,3),(3,2),(3,3),(3,4),(4,0),(4,1),(4,2),(4,4),(5,2),\ldots\},$$

a random infinite set. Let $\boldsymbol{R} = \{(1,1),(1,-1)\}$. *Then, the support of* $g(xy, xy^{-1})$ *is going to be equal to*

$$\{(0,0),(2,0),(3,-1),(5,-3),(4,0),(5,-1),(5,1),(6,0),(7,-1),(4,4),(5,3),(6,2),(8,0),(7,3),\ldots\}.$$

Now, let us analyze the support of g.

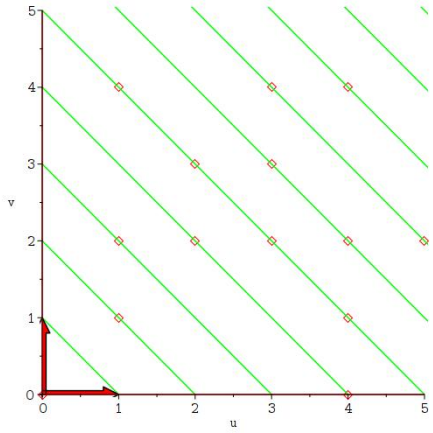

Figure 3.1: Support of g

In Figure 3.1, the red arrows represent the vectors $(1,0)$ *and* $(0,1)$. *We can think about these two vectors as the generating rays of the cone*

$$C_1 := \{(a,b) \in \mathbb{R}^2 \mid a \geq 0 \text{ and } b \geq 0\}.$$

We see that the $\operatorname{supp}(g)$ *is contained inside* C_1. *The green lines have equation* $u + v = d$, *with* $d = 0, 1, 2, \ldots$. *Thus, by intersecting* $\operatorname{supp}(g)$ *with* $u + v = d$, *we can get all the elements in the support of* g *with degree equal to* d. *We also observe that, if* d *varies from* 0 *to* ∞, *we get in an increasing and ordered way all the elements in* $\operatorname{supp}(g)$. *Ideally, we would like to have similar properties for* $\operatorname{supp}(g(xy, xy^{-1}))$. *We can analyze the support of* $g(xy, xy^{-1})$ *in the same way as follows.*

In Figure 3.2, the red arrows represent the rays $(1,1)$ *and* $(1,-1)$. *These rays generate the cone* C_2, *and by construction we know that the* $\operatorname{supp}(g(xy, xy^{-1}))$ *is contained inside* C_2. *The green lines have equation* $x + y = d$, *with* $d = 0, 1, 2, \ldots$.

By applying Lemma 7, we can get the elements in $\operatorname{supp}(g(xy, xy^{-1}))$. *Here we are just applying a linear transformation to the elements inside* $\operatorname{supp}(g)$. *However, as we observe in Figure 3.2, now the intersection between the line* $x + y = d$ *and* $\operatorname{supp}(g(xy, xy^{-1}))$ *does not generate a bounded region. This explains geometrically, why we can never be sure that we have found the minimum element of the support of a Laurent series* f, *whenever the cone used in the representation of* f *is generated by at lest one zero-ray.*

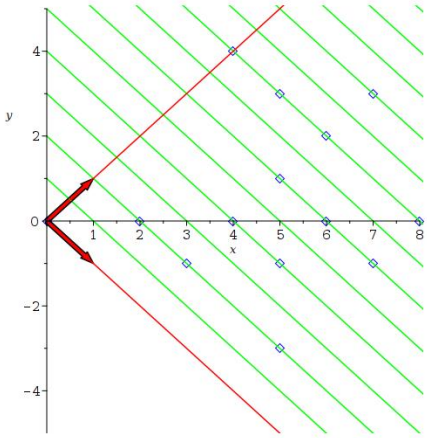

Figure 3.2: Support of $g(xy, xy^{-1})$

As just illustrated by Examples 4 and 5, having at least one zero-ray makes the process of determining the grevlex minimum element of a Laurent series not always possible. On the other hand, whenever we do not have a zero-ray, we are able to determine the minimum element of our Laurent series based in the element of our internal power series with smallest degree (as long as we can find it).

Proposition 8. *Let $g \in \mathbb{K}[[\boldsymbol{u}]]$ be a power series, $\boldsymbol{e} \in \mathbb{Z}^p$ be a point, and $\boldsymbol{R} := \{\boldsymbol{r}_1, \ldots, \boldsymbol{r}_m\} \subset \mathbb{Z}^p$ be a set of grevlex positive rays. Set*

$$f = \mathbf{x}^{\boldsymbol{e}} g(\mathbf{x}^{\boldsymbol{r}_1}, \ldots, \mathbf{x}^{\boldsymbol{r}_m}) \in \mathbf{x}^{\boldsymbol{e}} \mathbb{K}_C[[\mathbf{x}]],$$

with C the cone generated by $\boldsymbol{R}$. Then

$$\min \operatorname{supp}(g(\mathbf{x}^{\boldsymbol{R}})) = \min \left\{ \overline{\boldsymbol{R}} \cdot \boldsymbol{k}^T \mid \boldsymbol{k} \in \operatorname{supp}(g) \text{ with } \left|\overline{\boldsymbol{R}} \cdot \boldsymbol{k}^T\right| \leq \left|\overline{\boldsymbol{R}} \cdot \overline{\boldsymbol{k}}^T\right| \right\},$$

with $\overline{\boldsymbol{k}}$ the smallest element in $\operatorname{supp}(g)$ *and $\overline{\boldsymbol{R}} = (\boldsymbol{r}_1^T, \ldots, \boldsymbol{r}_m^T)$.*

Proof. Note that Lemma 7 implies that

$$\operatorname{supp}(g(\mathbf{x}^{\mathbf{R}})) = \left\{ \overline{\mathbf{R}} \cdot \mathbf{k}^T \mid \mathbf{k} \in \operatorname{supp}(g), \left|\overline{\mathbf{R}} \cdot \mathbf{k}^T\right| \leq \left|\overline{\mathbf{R}} \cdot \overline{\mathbf{k}}^T\right| \right\} \cup \left\{ \overline{\mathbf{R}} \cdot \mathbf{k}^T \mid \mathbf{k} \in \operatorname{supp}(g), \left|\overline{\mathbf{R}} \cdot \mathbf{k}^T\right| > \left|\overline{\mathbf{R}} \cdot \overline{\mathbf{k}}^T\right| \right\}.$$

Let $\mathbf{k}$ be an element of $\operatorname{supp}(g)$ such that

$$\left|\overline{\mathbf{R}} \cdot \mathbf{k}^T\right| > \left|\overline{\mathbf{R}} \cdot \overline{\mathbf{k}}^T\right|.$$

Then, by definition of the grevlex order

$$\overline{\mathbf{R}} \cdot \mathbf{k}^T >_{glex} \overline{\mathbf{R}} \cdot \overline{\mathbf{k}}^T.$$

Hence, the grevlex minimum element of $\operatorname{supp}(g(\mathbf{x}^{\mathbf{R}}))$ must be the grevlex minimum element of

$$\left\{ \overline{\mathbf{R}} \cdot \mathbf{k}^T \mid \mathbf{k} \in \operatorname{supp}(g), \left|\overline{\mathbf{R}} \cdot \mathbf{k}^T\right| \leq \left|\overline{\mathbf{R}} \cdot \overline{\mathbf{k}}^T\right| \right\}.$$

□

As a remark, notice that all of the above is only possible as long as we are able to find $\overline{\mathbf{k}}$, which as we mentioned before, it is not always feasible.

To handle our zero-ray case, we decided to implement an *internal bound*. We ask for $\mathbf{k}$ in the support of our internal power series object such that $\left|\overline{\mathbf{R}} \cdot \mathbf{k}^T\right|$ is less or equal to this bound. Then, we select between the images, $\overline{\mathbf{R}} \cdot \mathbf{k}^T$, of these $\mathbf{k}'s$ the one with smallest grevlex degree, say $\overline{\mathbf{e}} = \overline{\mathbf{R}} \cdot \mathbf{s}^T$, and we assume that this is the smallest element of the support of our Laurent series. If this is not the case, then an *Error message* will be raised whenever we discover that our assumption was false. The only way of knowing that we have made a mistake in the selection of our $\mathbf{x}^{\overline{\mathbf{e}}}$ is through our Truncate function. When the user asks for more terms of our inverse, MAPLE internally checks that the selection of $\mathbf{x}^{\overline{\mathbf{e}}}$ was not wrong.

Definition 5. *We refer to the just mentioned bound (when we have a cone generated by at least a zero-ray) or to the bound* $\left|\overline{\boldsymbol{R}} \cdot \overline{\boldsymbol{k}}^T\right|$ *(as in Proposition 8) as the inversion bound of our Laurent series* f.

Algorithm 12 LookForSmallestTerm

Require: Laurent series $f(\mathbf{x}) = \mathbf{x}^{\mathbf{e}} g(\mathbf{x}^{\mathbf{R}})$.
Ensure: $\overline{\mathbf{e}}$ the candidate to smallest term in $\text{supp}(g(\mathbf{x}^{\mathbf{R}}))$. B the internal bound of f.

1: **if** HasZeroRays(f) **then** ▷ We check if f has a **zero ray**.
2: $B :=$ InternalBound(f) ▷ We get the internal bound of f.
3: $\overline{\mathbf{e}} := \min_{glex}\{\mathbf{k} \in \text{supp}(g(\mathbf{x}^{\mathbf{R}}) \mid \left|\overline{\mathbf{R}} \cdot \mathbf{k}^T\right| \leq B)\}$ ▷ Smallest element in $\text{supp}(g(\mathbf{x}^{\mathbf{R}}))$
4: **else**
5: $\overline{\mathbf{k}} := \min \text{supp}(g(\mathbf{u}))$ ▷ Smallest element in $\text{supp}(g(\mathbf{u}))$
6: $B := \left|\overline{\mathbf{R}} \cdot \overline{\mathbf{k}}^T\right|$ ▷ We compute the internal bound of f.
7: $\overline{\mathbf{e}} := \min_{glex}\{\mathbf{k} \in \text{supp}(g(\mathbf{x}^{\mathbf{R}}) \mid \left|\overline{\mathbf{R}} \cdot \mathbf{k}^T\right| \leq B)\}$ ▷ Smallest element in $\text{supp}(g(\mathbf{x}^{\mathbf{R}}))$
8: **return** $\overline{\mathbf{e}}, B$

Now, suppose we have already made the selection of our $\overline{\mathbf{e}}$ as shown in Algorithm 12, and it is correct. Then, we can factor out $\mathbf{x}^{\overline{\mathbf{e}}}$ from $g(\mathbf{x}^{\mathbf{R}})$. Now, $\frac{g(\mathbf{x}^{\mathbf{R}})}{\mathbf{x}^{\overline{\mathbf{e}}}}$ has a constant element, and in consequence it is invertible. Hence,

$$f^{-1} = \mathbf{x}^{-\mathbf{e}}\mathbf{x}^{-\overline{\mathbf{e}}}\left(\frac{g(\mathbf{x}^{\mathbf{R}})}{\mathbf{x}^{\overline{\mathbf{e}}}}\right)^{-1} = \mathbf{x}^{-\mathbf{e}-\overline{\mathbf{e}}}\left(\frac{g(\mathbf{x}^{\mathbf{R}})}{\mathbf{x}^{\overline{\mathbf{e}}}}\right)^{-1}.$$

We can categorize multivariate power series in MAPLE into two types: power series with a *defined analytic expression* and power series with an undefined analytic expression (see [5]). For each power series g, created by the command PowerSeries as the image of a polynomial p (under the natural embedding from $\mathbb{C}[\mathbf{u}]$ to $\mathbb{C}[[\mathbf{u}]]$) the polynomial p is the analytic expression of g. We can define a power series by the sequence of its homogeneous parts. In this case, the user can optionally specify the sum of that series which is then set as analytic expression of the power series. As a nice property, we see that power series with a defined analytic expression are closed under addition, multiplication and inversion.

Multivariate power series in MAPLE with a defined rational analytic expression

We can use analytic expressions to define multivariate power series in MAPLE and this information is saved inside the MultivariatePowerSeriesObject. Note that multiplying or adding multivariate power series with a rational analytic expression generate a new power series with a rational analytic expression. Also, the analytic expression of the inverse of a multivariate power series (in case it exists), is a rational function. Then, any rational analytic expression can be seem as a rational function of the form $q(\mathbf{u}) = \frac{q_1(\mathbf{u})}{q_2(\mathbf{u})}$ by applying basic polynomial arithmetic. Notice that using the analytic expression associated to a power series g, we can easily invert the Laurent series $g(\mathbf{x}^{\mathbf{R}})$. Indeed,

Proposition 9. *Let $g \in \mathbb{K}[[\boldsymbol{u}]]$ be a power series, and $\boldsymbol{R} := \{\boldsymbol{r}_1, \ldots, \boldsymbol{r}_m\} \subset \mathbb{Z}^p$ be a set of grevlex non-negative rays. Assume that g has an associated analytic expression $q(\boldsymbol{u}) = \frac{q_1(\boldsymbol{u})}{q_2(\boldsymbol{u})}$, where $q_1(\boldsymbol{u})$ and $q_2(\boldsymbol{u})$ are polynomials in the variables $\boldsymbol{u}$. Then,*

1. *If q_1 has a constant term, then*

$$g(\mathbf{x}^{\boldsymbol{R}})^{-1} = q(\mathbf{x}^{\boldsymbol{R}})^{-1} = \frac{q_2(\mathbf{x}^{\boldsymbol{R}})}{q_1(\mathbf{x}^{\boldsymbol{R}})} \in \mathbb{K}_C[[\mathbf{x}]],$$

   with C the cone generated by $\boldsymbol{R}$.
2. *If q_1 does not have a constant term, then there exists a minimum set of rays $\boldsymbol{R}' := \{\boldsymbol{r}'_1, \ldots, \boldsymbol{r}'_p\} \subset \mathbb{Z}^p$, polynomials q'_1 and q'_2 such that*

$$g(\mathbf{x}^{\boldsymbol{R}})^{-1} = \mathbf{x}^{-\bar{e}} \frac{q'_2(\mathbf{x}^{\boldsymbol{R}'})}{q'_1(\mathbf{x}^{\boldsymbol{R}'})} \in \mathbf{x}^{-\bar{e}} \mathbb{K}_{C'}[[\mathbf{x}]],$$

   with C' the cone generated by $\boldsymbol{R}$' and $\bar{\boldsymbol{e}}$ the grevlex minimum element of $\operatorname{supp}(q_1(\mathbf{x}^{\boldsymbol{R}}))$.

Proof. Let C the cone generated by $\mathbf{R}$.

1. If q_1 has a constant term, them q_1 is invertible as a power series. Thus,

$$g(\mathbf{x}^{\mathbf{R}})^{-1} = q(\mathbf{x}^{\mathbf{R}})^{-1} = \frac{q_2(\mathbf{x}^{\mathbf{R}})}{q_1(\mathbf{x}^{\mathbf{R}})} \in \mathbb{K}_C[[\mathbf{x}]],$$

   is well defined.
2. Suppose that q_1 does not have a constant term. Now let $\bar{\mathbf{e}}$ be the grevlex minimum element of the finite set $\operatorname{supp}(q_1(\mathbf{x}^{\mathbf{R}}))$ and $\mathbf{s}$ be the respective element in $\operatorname{supp}(q_1(\mathbf{u}))$, such that $\bar{\mathbf{e}} = \overline{\mathbf{R}} \cdot \mathbf{s}^T$. Set $q''_1(\mathbf{u}) = \dfrac{q_1(\mathbf{u})}{\mathbf{u}^{\mathbf{s}}}$, and note that q''_1 is not necessarily a power series. Define the cone C' as the sum between the cone C, and the cone generated by the non-negative grevlex set of rays

$$\{\mathbf{k} - \bar{\mathbf{e}} \mid \mathbf{k} \in \operatorname{supp}(q_1(\mathbf{x}^{\mathbf{R}}))\} \cup \{\bar{\mathbf{e}}\}.$$

   Then, $q''_1(\mathbf{x}^{\mathbf{R}})$ and $q_2(\mathbf{x}^{\mathbf{R}})$ are elements in $\mathbb{K}_{C'}[[\mathbf{x}]]$, since $\operatorname{supp}(q''_1(\mathbf{x}^{\mathbf{R}})) \subseteq C'$ and $\operatorname{supp}(q_2(\mathbf{x}^{\mathbf{R}})) \subseteq C'$. Now, compute a set of non-negative grevlex generators of C', $\mathbf{R}' := \{\mathbf{r}'_1, \ldots, \mathbf{r}'_p\} \subset \mathbb{Z}^p$, as the output of Algorithm 9. Then, for $i \in \{1, \ldots, m\}$, there exists a $\mathbf{k}_i \in \mathbb{Z}^p$ such that $\mathbf{r}_i = \overline{\mathbf{R}'} \cdot \mathbf{k}_i^T$. Define

$$q_1'(\mathbf{v}) = q_1''(\mathbf{v}^{\mathbf{k}_1}, \ldots, \mathbf{v}^{\mathbf{k}_m}) \text{ and } q_2'(\mathbf{v}) = q_2(\mathbf{v}^{\mathbf{k}_1}, \ldots, \mathbf{v}^{\mathbf{k}_m}),$$

with $\mathbf{v} = v_1, \ldots, v_p$. Now observe

$$q_1'(\mathbf{x}^{\mathbf{R}'}) = \frac{q_1\left(\left(\mathbf{x}^{\mathbf{R}'}\right)^{\mathbf{k}_1}, \ldots, \left(\mathbf{x}^{\mathbf{R}'}\right)^{\mathbf{k}_m}\right)}{\left(\mathbf{x}^{\mathbf{R}'}\right)^{\mathbf{s}}} = \frac{q_1\left(\mathbf{x}^{\overline{\mathbf{R}'}\cdot\mathbf{k}_1^T}, \ldots, \mathbf{x}^{\overline{\mathbf{R}'}\cdot\mathbf{k}_m^T}\right)}{\mathbf{x}^{\overline{\mathbf{R}'}\cdot\mathbf{s}^T}} = \frac{q_1(\mathbf{x}^{\mathbf{R}})}{\mathbf{x}^{\bar{\mathbf{e}}}}.$$

Proposition 7 guarantees that $\mathbf{k}_1, \ldots, \mathbf{k}_p$ are non-negative integer vectors. Thus, q_1' and q_2' are well-defined multivariate power series. Notice also that $q_1'(\mathbf{v})$ is invertible as a power series, since it has a constant term. Thus,

$$g(\mathbf{x}^{\mathbf{R}})^{-1} = q(\mathbf{x}^{\mathbf{R}})^{-1} = \mathbf{x}^{-\bar{\mathbf{e}}}\frac{q_2'(\mathbf{x}^{\mathbf{R}'})}{q_1'(\mathbf{x}^{\mathbf{R}'})} \in \mathbf{x}^{-\bar{\mathbf{e}}}\mathbb{K}_{C'}[[\mathbf{x}]].$$

□

In Algorithm 13, we describe Proposition 9 in an algorithmic way.

Algorithm 13 InverseOfAnalyticExpression

Require: $q(\mathbf{u}) = \frac{q_1(\mathbf{u})}{q_2(\mathbf{u})}$, with $q_1(\mathbf{u}), q_2(\mathbf{u})$ polynomials. $\mathbf{x}^{\mathbf{R}}$ a change of variables with $\mathbf{R} := \{\mathbf{r}_1, \ldots, \mathbf{r}_m\} \subset \mathbb{Z}^p$.

Ensure: $q^{-1}(\mathbf{x}^{\mathbf{R}})$ the inverse of q.

1: **if** HasConstantTerm(q_1) **then**
2: **return** $\frac{q_2(\mathbf{x}^{\mathbf{R}})^{-1}}{q_1(\mathbf{x}^{\mathbf{R}})^{-1}}$ ▷ If q_1 has a constant term, then q is invertible.
3: **else**
4: $\bar{\mathbf{e}} := \min_{glex} \operatorname{supp}(q_1(\mathbf{x}^{\mathbf{R}}))$ ▷ We get the grevlex-smallest element in $\operatorname{supp}(q_1(\mathbf{x}^{\mathbf{R}}))$.
5: Solve($\bar{\mathbf{e}} = \overline{\mathbf{R}} \cdot \mathbf{s}^T$) ▷ We compute $\mathbf{s}$ such that $\bar{\mathbf{e}} = \overline{\mathbf{R}} \cdot \mathbf{s}$.
6: $q_1''(\mathbf{u}) := \frac{q_1(\mathbf{u})}{\mathbf{u}^{\mathbf{s}}}$ ▷ We define a new function. Note q_1'' could not be a power series.
7: $\mathbf{R}_1 := \{\mathbf{k} - \bar{\mathbf{e}} | \mathbf{k} \in \operatorname{supp}(q_1(\mathbf{x}^{\mathbf{R}}))\} \cup \{\bar{\mathbf{e}}\}$ ▷ A set of grevlex non-negative terms.
8: $\mathbf{R}' :=$ MakeRaysCompatible($\mathbf{R}, \mathbf{R}_1, \mathbb{K}, \mathbf{x}$) ▷ We get the rays for our new cone.
9: **for** i **from** 1 **to** m **do**
10: Solve($\mathbf{r}_i = \overline{\mathbf{R}'} \cdot \mathbf{k}_i^T$) ▷ We compute $\mathbf{k}_i$ such that $\mathbf{r}_i' = \overline{\mathbf{R}'} \cdot \mathbf{k}_i^T$.
11: $\mathbf{v} := v_1, \ldots, v_p$ ▷ New ordered variables for our power series.
12: $q_1'(\mathbf{v}) := q_1''(\mathbf{v}^{\mathbf{k}_1}, \ldots, \mathbf{v}^{\mathbf{k}_m})$ ▷ $q_1'(\mathbf{v}) \in \mathbb{K}[[\mathbf{v}]]$.
13: $q_2'(\mathbf{v}) := q_2(\mathbf{v}^{\mathbf{k}_1}, \ldots, \mathbf{v}^{\mathbf{k}_m})$ ▷ $q_2'(\mathbf{v}) \in \mathbb{K}[[\mathbf{v}]]$.
14: **return** $\mathbf{x}^{-\bar{\mathbf{e}}}\frac{(q_2')^{-1}(\mathbf{x}^{\mathbf{R}'})}{(q_1')^{-1}(\mathbf{x}^{\mathbf{R}'})}$

Multivariate power series in MAPLE with an undefined or non-rational analytic expression

Now, for the case of a power series with an undefined or non-rational analytic expression we can consider two types of Laurent series: Those with zero-rays and those without them. As

explain in Section 3.3.4, for both of our cases, we have a process to find a candidate for the minimum grevlex element, $\overline{\mathbf{e}}$, of $\operatorname{supp}(g(\mathbf{x}^{\mathbf{R}}))$. We can also compute $\mathbf{s}$, such that $\overline{\mathbf{e}} = \overline{\mathbf{R}} \cdot \mathbf{s}^T$. After this selection, the process for computing the inverse of f is almost the same in both of our cases (see Algorithm 14). We want to compute generating rays for a cone that contains the inverse. For that, we select all the elements in the $\operatorname{supp}(g(\mathbf{x}^{\mathbf{R}}))$ with weight less or equal to the inversion bound B of f, since we hope that the $\min \operatorname{supp}(g(\mathbf{x}^{\mathbf{R}}))$ is between them, and the rays in $\mathbf{R}$ that are grater than $\overline{\mathbf{e}}$. Then, we subtract $\overline{\mathbf{e}}$ from them.

$$S := \{\overline{\mathbf{R}} \cdot \mathbf{k} - \overline{\mathbf{e}} \mid |\overline{\mathbf{R}} \cdot \mathbf{k}| \leq B \text{ and } \mathbf{k} \in \operatorname{supp}(g)\} \cup \{\mathbf{r} - \overline{\mathbf{e}} \mid \mathbf{r} \in \mathbf{R} \text{ and } \mathbf{r} >_{glex} \overline{\mathbf{e}}\}.$$

Then, we get a new set of rays $\mathbf{R}'$ using our Algorithm 9 together with the sets $\mathbf{R}$ and S. Thus, the cone generated by $\mathbf{R}'$, will contain the cone $C(-\overline{\mathbf{e}}, \mathbf{R})$. After this, we compute a change of variable, $\mathbf{v}$, for the power series $g(\mathbf{u})/\mathbf{u}^{\mathbf{s}}$ with respect to $\mathbf{R}'$ and apply it. This new power series, $g'(\mathbf{v})$, has a constant term and in consequence, it is now invertible. Hence,

$$g(\mathbf{x}^{\mathbf{R}})^{-1} = \mathbf{x}^{-\overline{e}} g'(\mathbf{x}^{\mathbf{R}})^{-1}.$$

Algorithm 14 InverseOfUndefinedAnalyticExpression

Require: Laurent series $g(\mathbf{x}^{\mathbf{R}})$ with **undefined** or **non-rational** analytic expression and $\mathbf{R} := \{\mathbf{r}_1, \ldots, \mathbf{r}_m\} \subset \mathbb{Z}^p$.
Ensure: The inverse $g^{-1}(\mathbf{x}^{\mathbf{R}})$ of g.

if HasConstantTerm(g) **then**
  return $g^{-1}(\mathbf{x}^{\mathbf{R}})$ ▷ If g has a constant term, then g is invertible as a power series.
else
  $B, \overline{\mathbf{e}} :=$ LookForSmallestTerm($g(\mathbf{x}^{\mathbf{R}})$) ▷ Smallest term in $\operatorname{supp}(g(\mathbf{x}^{\mathbf{R}}))$.
  $S := \{\overline{\mathbf{R}} \cdot \mathbf{k} - \overline{\mathbf{e}} \mid |\overline{\mathbf{R}} \cdot \mathbf{k}| \leq B \text{ and } \mathbf{k} \in \operatorname{supp}(g)\} \cup \{\mathbf{r} - \overline{\mathbf{e}} \mid \mathbf{r} \in \mathbf{R} \text{ and } \mathbf{r} >_{glex} \overline{\mathbf{e}}\}$
  $\mathbf{R}' :=$ MakeRaysCompatible($\mathbf{R}, S, \mathbb{K}, \mathbf{x}$) ▷ We get the rays for our new cone.
  for i **from** 1 **to** m **do**
    Solve($\mathbf{r}_i = \overline{\mathbf{R}'} \cdot \mathbf{k}_i^T$) ▷ We compute $\mathbf{k}_i$ such that $\mathbf{r}'_i = \overline{\mathbf{R}'} \cdot \mathbf{k}_i^T$.
  $\mathbf{v} := v_1, \ldots, v_p$ ▷ New ordered variables for our power series.
  Solve($\overline{\mathbf{e}} = \overline{\mathbf{R}'} \cdot \mathbf{s}^T$) ▷ We compute $\mathbf{s}$ such that $\overline{\mathbf{e}} = \overline{\mathbf{R}'} \cdot \mathbf{s}^T$.
  $g''(\mathbf{u}) := g(\mathbf{u})/\mathbf{u}^{\mathbf{s}}$ ▷ We factor $\mathbf{u}^{\mathbf{s}}$ out from g.
  $g'(\mathbf{v}) := g''(\mathbf{v}^{\mathbf{k}_1}, \ldots, \mathbf{v}^{\mathbf{k}_m})$ ▷ $g'(\mathbf{v}) \in \mathbb{K}[[\mathbf{v}]]$.
  return $\mathbf{x}^{-\overline{e}}(g')^{-1}(\mathbf{x}^{\mathbf{R}'})$

For a complete implementation of the Inverse of a multivariate Laurent series see Algorithm 15. Here we put together all the ideas explained in Subsection 3.3.4.

Algorithm 15 Inverse

Require: Laurent series $f(\mathbf{x}) = \mathbf{x}^{\mathbf{e}} g(\mathbf{x}^{\mathbf{R}})$.
Ensure: The inverse f^{-1} of f.

if AnalyticExpression(f) = **Undefined** or **non-rational then**
  return $\mathbf{x}^{-\mathbf{e}}$InverseOfUndefinedAnalyticExpression($g(\mathbf{x}^{\mathbf{R}})$)
else
  $q :=$ AnalyticExpression(f) ▷ The analytic expression of f.
  return $\mathbf{x}^{-\mathbf{e}}$InverseOfAnalyticExpression($q, \mathbf{x}^{\mathbf{R}}$)

3.4 An overview of the Laurent series object in Maple

As mentioned in Section 3.3 a first implementation of the Laurent series object has been implemented inside the package `MultivariatePowerSeries` of Maple 2022. Currently, this object and its associate commands are hidden, though. Thus, if we want to access to this object, we must do it through the package `MultivariatePowerSeries`, i.e., like `MultivariatePowerSeries:-LaurentSeriesObject`. In order to make this process easier, we can use Maple `kernelops()` command, see Figure 3.3.

```
> with(MultivariatePowerSeries);
[Add, ApproximatelyEqual, ApproximatelyZero, Copy, Degree, Divide, EvaluateAtOrigin, Exponentiate, GeometricSeries,
  GetAnalyticExpression, GetCoefficient, HenselFactorize, HomogeneousPart, Inverse, IsUnit, MainVariable, Multiply, Negate,
  PowerSeries, Precision, SetDefaultDisplayStyle, SetDisplayStyle, Substitute, Subtract, SumOfAllMonomials, TaylorShift,
  Truncate, UnivariatePolynomialOverPowerSeries, UpdatePrecision, Variables, WeierstrassPreparation]

> kernelopts(opaquemodules = false):
  LaurentSeries := MultivariatePowerSeries:-LaurentSeriesObject:
  kernelopts(opaquemodules = true):
```

Figure 3.3: Laurent series object

As illustrated in Figure 3.4, there are two main ways of defining a Laurent series Object in Maple:

1. We can define an order, X, for the space $\mathbb{K}_C[[X]]$; another order, U, for the space $\mathbb{K}[[U]]$; a generating set R for the cone C; a list of equations, e, representing the exponents of the monomial multiplying our Laurent series; and finally our internal multivariate power series g.
2. We can define a list of equations representing the change of variables to be applied to our internal multivariate power series g; a list of equations, e, representing the exponents of the monomial multiplying our Laurent series; and finally our internal multivariate power series g.

```
> X := [x, y]: U := [u, v]:
  g1 := Inverse(PowerSeries(1 + u*v)):
  e := [x = -5, y = 3]:
  R := [[1, 0], [1, -1]]:
> f1 := LaurentSeries(g1, X, U, R, e);

f1 := [LaurentSeries of y^3/((x^2/y + 1) x^5) : y^3/x^5 - y^2/x^3 + y/x - x + x^3/y - x^5/y^2 + x^7/y^3 - x^9/y^4 + ...]

> LaurentSeries:-Truncate(f1, 8);

y^3 (x^8/y^4 - x^6/y^3 + x^4/y^2 - x^2/y + 1) / x^5

> g2 := PowerSeries(1/(1 + u)):
  mp := [u = x^(-1)*y^2]: e := [x = 3, y = -4]:
> f2 := LaurentSeries(g2, mp, e);

f2 := [LaurentSeries of x^3/((1 + y^2/x) y^4) : x^3/y^4 + ...]

> LaurentSeries:-Truncate(f2, 8);

x^3 (y^16/x^8 - y^14/x^7 + y^12/x^6 - y^10/x^5 + y^8/x^4 - y^6/x^3 + y^4/x^2 - y^2/x + 1) / y^4
```

Figure 3.4: Creation Laurent series

We can also add and multiply Laurent series objects, see Figures 3.6 and 3.5. Finally, as shown in Figure 3.7, we can compute the inverse of a Laurent series and check the result via a

multiplication. Here, we are only showing the main commands inside the Laurent series object, there are more worth mentioning, though:

1. `ApproximatelyZero`, `ApproximatelyEqual` to compare Laurent series;
2. `GrevLexGreater`, `Positive`, `Smallest` to do grevlex comparisons;
3. `ChangeOfVariables` to compute change of variable that is applied to the internal power series;
4. `Truncate` to truncate our Laurent series up to a certain degree making use of the internal power series.

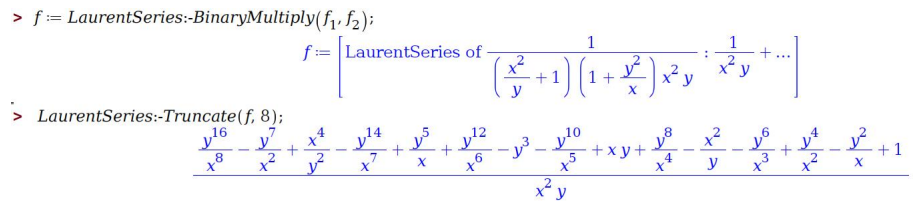

Figure 3.5: Multiplication of Laurent series

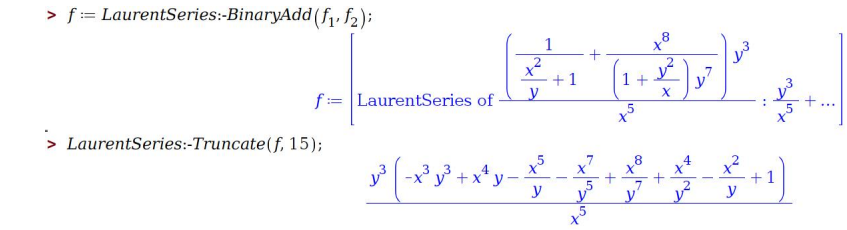

Figure 3.6: Addition of Laurent series

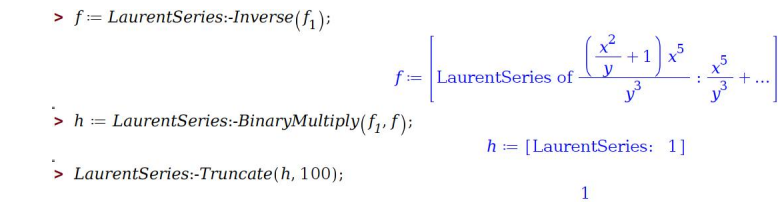

Figure 3.7: Inverse of a Laurent series

Chapter 4

Conclusions and future work

We have discussed an algorithm for computing Intersect(f, T) where T is a one-dimensional regular chain. Our presentation was specialized to the case of trivariate polynomials.

This algorithm permits the control of expression swell by means of an evaluation-interpolation scheme, assuming that the computations do not need to split. Genericity assumptions ensure this latter requirement and are tested by the algorithm. When those assumptions are not met, the algorithm fails, in which case the general (and non-modular) algorithm for Intersect(f, T) can take over.

When those assumptions are met, the modular algorithm relies on the computations of subresultants of index 0 and index 1, without requiring subresultants of higher indices. This strategy opens the door to using *speculative algorithms* for computing subresultants, an idea introduced and studied in [6]. Such algorithms are asymptotically fast algorithms that:

1. compute the subresultants of index 0 and 1, without computing the other subresultants, while
2. being able to resume the computations for obtaining the subresultants of higher index, if needed.

As we demonstrate in Section 2.5 and 2.6, using those speculative algorithms within our modular algorithm for computing Intersect(f, T)

1. further improve performance, while
2. allowing us to relax the genericity assumptions.

As mention in Section 2.5, our BPAS implementation of the Algorithm 6 is based in an improved version of Lagrange interpolation. Thus, a natural future direction of research would be to consider a Fast Fourier Transform (FFT) approach instead of the current Lagrange interpolation. Another natural direction of research would involve a parallelized version of Algorithm 5, which should be easy to implement inside BPAS. We will also like to relax our genericity assumptions as much as possible as well as to solve for n variable case.

On the other hand, we have successfully written a first implementation a Laurent series object inside MAPLE. With this object, we are able to define multivariate Laurent series, multiply them, add them and find their inverse (in most of the cases). There is, however, room for improvements. As we have shown with Example 5, whenever we have a cone C generated by at least a zero-ray, there exists the possibility of not been able of finding the inverse of a given Laurent series with support is contained in C. The natural question to ask in this moment is: Could it be possible to avoid this issue by choosing a different total order?

It is possible to define any monomial ordering by a sequence of weight vectors (for details see [17]). As we mention in Subsection 3.3.1, the grevlex order can be defined as a sequence of the weight vectors $\mathbf{1}_p := (1, 1, 1, \ldots, 1), \mathbf{1}_{p-1} := (1, 1, \ldots, 1, 0), \ldots, \mathbf{1}_1 := (1, 0, \ldots, 0)$. Thus, we could try to change $\mathbf{1}_p$ by an adequate weight vector in such way that problem illustrated in Example 5 is avoided. We could also try to find another way of handling cones generated by at least a zero-ray. The propose of using the ring $\mathbb{K}_C[[\mathbf{x}]]$ is to only have to deal with a finite number of monomials with a set degree d at the same time. This, however, exclude the monomials of degree zero, since in this case we could have an infinite amount of them. Thus, by effectively manipulating this cones, we could find better ways of operating Laurent series, and in particular inverting them.

A next possible direction of research is the implementation of Puiseux series. The famous Newton-Puiseux algorithm (and its extensions) essentially allows for the local study of curves (separating their branches about a point) via the manipulation of Laurent and Puiseux series. Thus, by extending our current implementation of a multivariate Laurent series to multivariate Puiseux series, we should be able to code at least one of the version of the Newton-Puiseux algorithm that only relies in Puiseux series arithmetic. For instance, the Nowak's construction [27].

Bibliography

[1] Maple modp1 online help. `https://www.maplesoft.com/support/help/maple/view.aspx?path=modp1`.

[2] Parisa Alvandi, Masoud Ataei, Mahsa Kazemi, and Marc Moreno Maza. On the extended Hensel construction and its application to the computation of real limit points. *J. Symb. Comput.*, 98:120–162, 2020.

[3] Parisa Alvandi, Changbo Chen, and Marc Moreno Maza. Computing the limit points of the quasi-component of a regular chain in dimension one. In Vladimir P. Gerdt, Wolfram Koepf, Ernst W. Mayr, and Evgenii V. Vorozhtsov, editors, *Computer Algebra in Scientific Computing - 15th International Workshop, CASC 2013, Berlin, Germany, September 9-13, 2013. Proceedings*, volume 8136 of *Lecture Notes in Computer Science*, pages 30–45. Springer, 2013.

[4] Mohammadali Asadi, Alexander Brandt, Changbo Chen, Svyatoslav Covanov, Farnam Mansouri, Davood Mohajerani, Robert H. C. Moir, Marc Moreno Maza, Delaram Talaashrafi, Linxiao Wang, Ning Xie, and Yuzhen Xie. Basic Polynomial Algebra Subprograms (BPAS), 2021. `www.bpaslib.org`.

[5] Mohammadali Asadi, Alexander Brandt, Mahsa Kazemi, Marc MorenoMaza, and Erik J. Postma. Multivariate power series in Maple. In Robert M. Corless, Jürgen Gerhard, and Ilias S. Kotsireas, editors, *Maple in Mathematics Education and Research*, pages 48–66, Cham, 2021. Springer International Publishing.

[6] Mohammadali Asadi, Alexander Brandt, and Marc Moreno Maza. Computational schemes for subresultant chains. In François Boulier, Matthew England, Timur M. Sadykov, and Evgenii V. Vorozhtsov, editors, *Computer Algebra in Scientific Computing - 23rd International Workshop, CASC 2021, Sochi, Russia, September 13-17, 2021, Proceedings*, volume 12865 of *Lecture Notes in Computer Science*, pages 21–41. Springer, 2021.

[7] Philippe Aubry, Daniel Lazard, and Marc Moreno Maza. On the theories of triangular sets. *J. Symb. Comput.*, 28(1-2):105–124, 1999.

[8] Magali Bardet, Jean-Charles Faugère, and Bruno Salvy. On the complexity of the F5 Gröbner basis algorithm. *J. Symb. Comput.*, 70:49–70, 2015.

[9] Eberhard Becker, Teo Mora, Maria Grazia Marinari, and Carlo Traverso. The shape of the shape lemma. In Malcolm A. H. MacCallum, editor, *Proceedings of the International Symposium on Symbolic and Algebraic Computation, ISSAC '94, Oxford, UK, July 20-22, 1994*, pages 129–133. ACM, 1994.

[10] François Boulier, François Lemaire, and Marc Moreno Maza. Well known theorems on triangular systems and the D5 principle. In *Proc. of* Transgressive Computing 2006, Granada, Spain, 2006.

[11] Alexander Brandt. High performance sparse multivariate polynomials: Fundamental data structures and algorithms. Master's thesis, Western University, 2018.

[12] Alexander Brandt and Marc Moreno Maza. On the complexity and parallel implementation of hensel's lemma and weierstrass preparation. In François Boulier, Matthew England, Timur M. Sadykov, and Evgenii V. Vorozhtsov, editors, *Computer Algebra in Scientific Computing - 23rd International Workshop, CASC 2021, Sochi, Russia, September 13-17, 2021, Proceedings*, volume 12865 of *Lecture Notes in Computer Science*, pages 78–99. Springer, 2021.

[13] Bruno Buchberger. *An algorithm for finding a basis for the residue class ring of a zero-dimensional polynomial ideal*. PhD thesis, Ph. D. thesis, University of Innsbruck, Austria, 1965.

[14] Matt Calder, Juan P. González Trochez, Marc Moreno Maza, and Erik Postma. A maple implementation of a modular algorithm for computing the common zeros of a polynomial and a regular chain maple. *Maple Transactions*, 2021.

[15] Matt Calder, Juan Pablo González Trochez, Marc Moreno Maza, and Erik Postma. Modular intersect. `https://github.com/JuanPabloGonzalezTrochez/ModularIntersect`, 2021.

[16] Changbo Chen and Marc Moreno Maza. Algorithms for computing triangular decomposition of polynomial systems. *J. Symb. Comput.*, 47(6):610–642, 2012.

[17] David A. Cox, John Little, and Donal O'Shea. *Ideals, varieties, and algorithms - an introduction to computational algebraic geometry and commutative algebra (2. ed.)*. Undergraduate texts in mathematics. Springer, 1997.

[18] Xavier Dahan, Marc Moreno Maza, Éric Schost, Wenyuan Wu, and Yuzhen Xie. Lifting techniques for triangular decompositions. In *ISSAC 2005, Beijing, China, 2005, Proceedings*, pages 108–115, 2005.

[19] Gerd Fischer. *Plane algebraic curves*, volume 15. American Mathematical Soc., 2001.

[20] Michael Kalkbrener. A generalized euclidean algorithm for computing triangular representations of algebraic varieties. *J. Symb. Comput.*, 15(2):143–167, 1993.

[21] Daniel Lazard. A new method for solving algebraic systems of positive dimension. *Discret. Appl. Math.*, 33(1-3):147–160, 1991.

[22] Grégoire Lecerf. Computing the equidimensional decomposition of an algebraic closed set by means of lifting fibers. *Journal of Complexity*, 19(4):564–596, 2003.

[23] Marc Moreno Maza, Bican Xia, and Rong Xiao. On solving parametric polynomial systems. *Math. Comput. Sci.*, 6(4):457–473, 2012.

[24] Michael B. Monagan. Probabilistic algorithms for computing resultants. In *ISSAC*, pages 245–252. ACM, 2005.

[25] Ainhoa Aparicio Monforte and Manuel Kauers. Formal Laurent series in several variables. *Expositiones Mathematicae*, 31(4):350–367, 2013.

[26] Marc Moreno Maza. On triangular decompositions of algebraic varieties. Technical Report TR 4/99, NAG Ltd, Oxford, UK, 1999. Presented at the MEGA-2000 Conference, Bath, England. http://www.csd.uwo.ca/~moreno.

[27] Krzysztof Jan Nowak. Some elementary proofs of Puiseuxs theorems. *Univ. Iagel. Acta Math*, 38:279–282, 2000.

[28] Wikipedia contributors. Monomial order — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Monomial_order&oldid=1004440098`, 2021.

Curriculum Vitae

Name:	Juan Gonzalez Trochez
Post-Secondary Education and Degrees:	University of Western Ontario London, ON 2020 - 2022 M.Sc. Computer Science 2019 - 2020 M.Sc. Mathematics Universidad del Norte Barranquilla, Colombia 2016 - 2018 Master in Mathematics 2011 - 2015 Mathematics
Honours and Awards:	Uninorte Caribe scholarship 2011-2015
Related Work Experience:	Teaching Assistant The University of Western Ontario 2019 - 2022 Lecturer Universidad del Norte 2018 - 2019 Teaching Assistant Universidad del Norte 2016 - 2017

Publications:

Matt Calder, Juan P. González Trochez, Marc Moreno Maza, and Erik Postma. A maple implementation of a modular algorithm for computing the common zeros of a polynomial and a regular chain maple. *Maple Transactions*, 2021