SINGULARITY THEORY AND ARF RINGS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

NİL ŞAHİN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN
MATHEMATICS

DECEMBER 2012

Approval of the thesis:

SINGULARITY THEORY AND ARF RINGS

submitted by **NİL ŞAHİN** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy in Mathematics Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen
Dean, Graduate School of **Natural and Applied Sciences** _______________

Prof. Dr. Mustafa Korkmaz
Head of Department, **Mathematics** _______________

Assoc. Prof. Dr. Ahmet İrfan Seven
Supervisor, **Mathematics Dept., METU** _______________

Assoc. Prof. Dr. Feza Arslan
Co-supervisor, **Mathematics Dept., Mimar Sinan Fine Arts University** _______________

Examining Committee Members:

Prof. Dr. Ali Sinan Sertöz
Mathematics Dept., Bilkent University _______________

Assoc. Prof. Dr. Ahmet İrfan Seven
Mathematics Dept., METU _______________

Prof. Dr. Yıldıray Ozan
Mathematics Dept., METU _______________

Prof. Dr. Mustafa Korkmaz
Mathematics Dept., METU _______________

Prof. Dr. Hurşit Önsiper
Mathematics Dept., METU _______________

Date: _______________

ABSTRACT

SINGULARITY THEORY AND ARF RINGS

Şahin, Nil
Ph.D, Department of Mathematics
Supervisor : Assoc. Prof. Dr. Ahmet İrfan Seven
Co-Supervisor : Assoc. Prof. Dr. Feza Arslan

December 2012, 131 pages

Arf closure of a local ring corresponding to a curve branch, which carries a lot of information about the branch is an important object of study. Though both Arf rings and Arf semigroups are being studied by many mathematicians, there is not an implementable fast algorithm for constructing the Arf closure. The main aim of this thesis is to give an easily implementable fast algorithm to compute the Arf Closure of a given local ring. The speed of the algorithm is a result of the fact that the algorithm avoids computing the semigroup of the local ring. Moreover, in doing this, we give a bound for the conductor of the semigroup of the Arf Closure without computing the Arf Closure by using the theory of plane branches. We also give an exposition of plane algebroid curves and present the SINGULAR library written by us to compute the invariants of plane algebroid curves.

Keywords: Arf Rings, Multiplicity Sequence, Singularity Theory, Arf Closure, Hilbert Series

ÖZ

TEKİLLİK TEORİSİ VE ARF HALKALARI

Şahin, Nil
Doktora, Matematik Bölümü
Tez Yöneticisi : Doç. Dr. Ahmet İrfan Seven
Ortak Tez Yöneticisi : Doç. Dr. Feza Arslan

Aralık 2012, 131 sayfa

Bir eğri koluna karşılık gelen lokal halkanın Arf kapanışı, o eğri kolu ile ilgili birçok bilgi taşıyan, önemli bir çalışma objesidir. Arf Halkaları ve yarı grupları birçok matematikçi tarafından çalışılmasına rağmen, Arf kapanışını inşa eden, hızlı bir algoritma bulunmamaktadır. Bu tezin asıl amacı, Arf kapanışını hesaplamak için kolay uygulanabilir bir algoritma vermektir. Algoritmanın hızı, lokal halkanın yarıgrubunu hesaplamadan çalışmasının bir sonucudur. Bunların yanında, Arf kapanışının yarı grubunun kondaktörüne, düzlem eğri kollarının teorisini kullanarak, Arf kapanışını hesaplamadan bir sınır veriyoruz. Düzlem eğri kollarının teorisini açıklayıp, bizim tarafımızdan yazılan ve düzlem eğri kollarının invaryantlarını hesaplayan SINGULAR kütüphanesini tanıtıyoruz.

Anahtar Kelimeler: Arf Halkaları, Çok katlılık Dizisi, Tekillik Teorisi, Arf Kapanışı, Hilbert Serileri

To my family

ACKNOWLEDGMENTS

It is a great pleasure for me to thank people who made it possible to write this thesis.

I would like to express my heartfelt thanks to Professor Feza Arslan not only for his guidance about my research and mathematics, but also for his generous help in life. Because of his patience, I had the encourage to ask any questions that I have in my mind. Besides his mathematical intelligence, his humanity have provided a good example for me. One simply could not wish for a better or friendlier supervisor. I would also like to thank Professor Ahmet İrfan Seven and all the members of the department of mathematics of the Middle East Technical University who helped me in my supervisor's absence.

The thesis jury who kindly accepted to report on my work are also acknowledged.

I am deeply grateful to Prof. Dr. Gerhard Pfister for introducing me the library ”space-curve.lib” and helping me to understand it in every step. I have been extremely lucky to have a supervisor like him in Germany who cared so much about my work and who responded my questions patiently.

My sincere thanks also goes to Prof. Dr. Dr.h.c. Gert-Martin Greuel for giving me the opportunity to work with the SINGULAR group and introducing me with Professor Gerhard Pfister.

Completing this thesis with my sanity would definitely be harder without the support of my friends Zeynep, Köksal, Sevtap, Nuray, Hakan, Canan, Dürdane, Nagehan, Arzu and Sidre. I would like to thank all of them for helping me to get through the hard times and for all the fun we had together during these past seven years. I would like to thank Hakan and Köksal in advance for sharing their office with me, providing me a nice working environment and making me smile all the time. I am also grateful to Canan for tutoring me for an exam in the last minute and for all her advices about my future.

I wish to thank Christian Eder and Martin Lee from the mathematics department of Technische Universitat Kaiserslautern for helping me both in SINGULAR and my life in Kaiserslautern.

They have been perfect hosts, office mates and friends. I also thank Petra Basell for her kind and warm friendship. It was very nice to know there were people that I can talk to when I needed help. Without them, it wouldn’t be possible for me to get used to living in Germany.

I especially thank my mom, dad and sister for their endless support, unconditional love and care. They have been my best friends and they have a big role in me completing this thesis. I love them so much.

Finally, I want to thank The Scientific and Technological Research Council of Turkey, TUBITAK for supporting me financially both in Turkey and Germany.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In this thesis, we focus on space curve singularities. The study of singularities goes back to 1918's to Enriques-Chisini [26]. Together with Zariski's studies, singularities of plane curves had been been fully worked out with the theory of characteristic exponents. Du Val in 1942, showed that there are some special numbers, which he calls the characters of the branch. If the modified Jacobian Algorithm is applied to these characters, the multiplicity sequence of the curve branch is obtained [23]. Contrary to the well-known plane case, in which characteristic exponents, multiplicity sequence and characters determine each other, it was not known how to obtain the characters in space case, until Cahit Arf developed his theory [1]. In 1946, Arf showed that the characters of a space branch could be obtained from the completion of the local ring corresponding to the branch by constructing its canonical closure, later known as Arf closure [1]. His method was fully algebraic, but the results were also answering some very important geometric questions, The geometrical aspects of Arf's results were made clear by Du Val in 1949 [24]. Since then, many algebraic geometers and algebraists have worked on Arf rings, Arf closure and the the invariants of the curve singularities [3], [11], [12], [20]. Arf semigroups and their applications in coding theory have been a recent area of interest [7], [9], [14], [22], [35]. In the meantime, Hamburger-Noether expansions developed by Ancochea has been a major tool in the computation of the invariants of the curve singularities [10], [13]. Castellanos has also presented the construction of Arf closure by using Hamburger-Noether matrices [16].

In spite of the fact that many mathematicians are interested in Arf rings and Arf closure, there is not a fast implementable algorithm for the computation of Arf closure in the literature. The only implemented algorithm is given by Arslan [2]. The algorithm uses Arf's method and starts with determining the semigroup of values and the conductor of the branch. Determining the semigroup of values and the conductor of the branch is a difficult problem, which has been studied by many mathematicians and different algorithms have been given [15], [28]. Noting that the special case of this problem is the famous Fobenius problem (or coin problem) makes it clear, why this problem is difficult, and it is unnecessary to mention that there is a vast literature on the Frobenius problem.

In this thesis, our main aim is to give a fast and easily implementable algorithm for the computation of the Arf closure, which does not need determining the semigroup of values and the conductor of the branch. Our second aim is to give codes in the computer algebra program SINGULAR both for the computation of the Arf closure and the invariants of a plane algebroid curve.

The thesis is organized as follows. In Chapter 2, we give the basic definitions and theory related with a space curve singularity or an algebroid curve. We also recall the necessary theory about the blowing up and desingularization process of the branches. We also explain the singularity theory and characters of curve branches as Du Val did in [23].

In Chapter 3, we focus on plane curve singularities. We consider both reducible and irreducible algebroid curves. This expository chapter contains the multiplicity sequence, characteristic exponents, semigroup of values, Puiseux pairs and resolution graphs of irreducible curve branches. The facts that are presented in this section for plane branches are used strongly in Chapter 6 to give a bound for the conductor of the value semigroup of the Arf closure of the local ring, which is crucial in our proposed fast algorithm. This chapter is also essential for understanding the importance of Arf theory. We also give the definitions of the contact numbers of branches of reducible algebroid curves and the resolution graph of reducible algebroid curves are defined.

Chapter 4 is devoted to the study of Hamburger-Noether expansions. We study the subject in two parts as 2-dimensional and n-dimensional case by using [10] and [13]. First, Hamburger-Noether expansions for plane branches and the method of determining the invariants from the Hamburger-Noether expansions are explained. Moreover, obtaining the intersection multiplicity and the contact numbers of two branches from their Hamburger-Noether expansions is presented. Then, Hamburger-Noether expansions for curve branches in n-dimensional case is explained and Hamburger-Noether matrices are defined in the general case. The theory in this chapter is used in Chapter 8, while writing the codes in SINGULAR.

In Chapter 5, we describe Arf theory [1]. Arf rings, Arf semigroups, the computation of the Arf closure and Arf characters are presented with many examples. The construction of Arf closure by using Hamburger-Noether matrices is also presented.

In Chapter 6, we present our ideas and main contributions. We propose a new fast algorithm for constructing the Arf Closure of a local ring without determining the semigroup of values and the conductor. After giving the theoretical framework, explaining why the algorithm works, we give a bound for the conductor of the Arf closure of a local ring. In doing this, we combine the results in Chapter 3 with our results presented in this chapter.

In Chapter 7, we state some results about the Hilbert series of Arf rings and present a conjecture of Arslan and Sertoz about the Hilbert series of Arf rings having the same Arf Closure. We also present a lot of examples obtained by the algorithm given in Chapter 6 that support the conjecture.

In Chapter 8, we describe the SINGULAR library ”curvepar.lib” written by us with Gerhard Pfister and Maryna Viazovska [18]. That library computes the resolution graph and the invariants of the branches of an algebroid plane curve. The library computes all the invariants for a reducible algebroid plane curve.

Finally, we conclude the thesis by some remarks and ideas about the future research.

CHAPTER 2

THEORETICAL BACKGROUND

2.1 Curve Branches

Throughout this thesis, our main objects of interest are algebroid curve branches, so we start with its definition.

Definition 2.1.1 *A space curve singularity or an algebroid curve is $C = \operatorname{Spec} R$, where $(R, \mathfrak{m}, k)$ is a local ring, complete for the $\mathfrak{m}$-adic topology, with Krull dimension 1 and having k as a coefficient field ($k \subset R$ and $k \cong R/\mathfrak{m}$). Moreover, if R is also a domain, then C is called an irreducible algebroid curve or an algebroid curve branch, [13].*

Definition 2.1.2 *Let $C = \operatorname{Spec} R$ be an algebroid curve. The embedding dimension of C is defined to be the* $\dim_k(\mathfrak{m}/\mathfrak{m}^2)$ *and it is denoted by* embdim(R). *The embedding dimension of R is also defined to be* $\text{embdim(R)} = \dim_k(\mathfrak{m}/\mathfrak{m}^2)$.

Theorem 2.1.3 *Let $C = \operatorname{Spec} R$ be an algebroid curve. Then the following holds.*

(i)[17, Cohen's Structure Theorem] There exists an ideal I in $k[[x_1, ..., x_n]]$ with $R \cong k[[x_1, ..., x_n]]/I$ and $n =$ embdim(R).

(ii) There exists prime ideals $\mathfrak{p}_1, ..., \mathfrak{p}_s \in k[[x_1, ..., x_n]]$ with $\mathfrak{p}_i \not\subset \mathfrak{p}_j$ for $i \neq j$ and $I = \bigcap_{i=1}^{s} \mathfrak{p}_i$. ($\mathfrak{p}_1, ..., \mathfrak{p}_s$ are minimal prime ideals.)

Thus, we will be working with local rings of the form $k[[x_1, ..., x_n]]/I$, and we can reformulate the algebroid curve branch definition as follows.

Definition 2.1.4 *The algebroid curve* $C = \operatorname{Spec} k[[x_1, ..., x_n]]/I$ *is called an algebroid curve branch (or an irreducible algebroid curve), if* I *is a prime ideal in* $k[[x_1, ..., x_n]]$.

Example 2.1.5 *Consider the nodal cubic curve defined as the zero set of the polynomial* $y^2 - x^2(1 + x)$. *Since we are interested in the behavior around the point* $(0, 0)$, *we work with the algebroid curve* $\operatorname{Spec} k[[x, y]]/\langle y^2 - x^2(1 + x)\rangle$. *Observing that*

$$\sqrt{1+x} = \pm\left(1 + \frac{1}{2}x - \frac{1}{8}x^2 + ...\right)$$

we have

$$y^2 - x^2(1+x) = \left(y - x - \frac{1}{2}x^2 + \frac{1}{8}x^3 - ...\right)\left(y + x + \frac{1}{2}x^2 - \frac{1}{8}x^3 + ...\right)$$

in the formal power series ring $k[[x, y]]$. *Thus, we have two branches* $\operatorname{Spec} k[[x, y]]/\mathfrak{p}_1$ *and* $\operatorname{Spec} k[[x, y]]/\mathfrak{p}_2$, *where* $\mathfrak{p}_1 = \langle y - x - \frac{1}{2}x^2 + \frac{1}{8}x^3 - ...\rangle$ *and* $\mathfrak{p}_2 = \langle y + x + \frac{1}{2}x^2 - \frac{1}{8}x^3 + ...\rangle$. *This example clarifies, why we work with the completion of the local ring at the singularity. (See Figure 2.1.)*

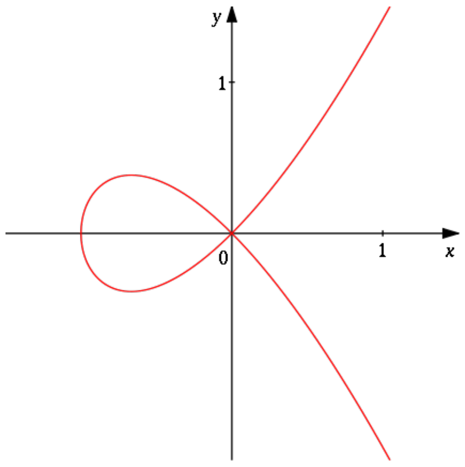


Figure 2.1: Nodal cubic

2.1.1 Parametrization

Let R be the local ring of an algebroid curve branch over an algebraically closed field k. Let us denote the integral closure of the local ring R in its ring of fractions by $\overline{R}$. Then, $\overline{R} \cong k[[t]]$, i.e. $\overline{R}$ is a discrete valuation ring [10]. As a consequence, there is a natural valuation v_R on the elements of $\overline{R}$ induced by this isomorphism. For an arbitrary element r of $\overline{R}$ there exists an element $f(t)$ of $k[[t]]$ such that $r = f(t)$ and $\upsilon_R(r) = \upsilon(f)$ where υ is the natural valuation on $k[[t]]$, which is given by the order at t of the associated series defined below.

Definition 2.1.6 *Let f be an element of the power series ring $k[[t]]$. The order of the power series f is defined to be the smallest power appearing in the series f and it is denoted by ord(f).*

Example 2.1.7 *Let $f(t) = t^4 + t^7 + t^{12}$, $g(t) = t^{21} + t^{12} + t^{31}$ and $h(t) = 1 + t^3 + t^{11}$ be power series in $\mathbb{R}[[t]]$. Then ord(f)$= 4$, ord(g)$= 12$ and ord(h)$= 0$.*

We can now introduce a parameterization of an algebroid curve.

Definition 2.1.8 *Let $C = \operatorname{Spec} R$ be an algebroid curve singularity and $\{x_1, ..., x_N\}$ be the system of generators for the maximal ideal $\mathfrak{m}$ of R, where $N = embdim(R)$. An algebraic parametrization of C corresponding to $\{x_1, ..., x_N\}$ is a continuous k-algebra homomorphism*

$$\mathcal{P} : k[[X_1, ..., X_N]] \to k[[t]]$$

where t is an indeterminate, $\operatorname{im}(\mathcal{P}) \not\subset \mathrm{k}$ and kernel of the map $\varsigma : k[[X_1, ..., X_N]] \to R$, which sends X_i to x_i is contained in the kernel of P (ς exists by Cohen's Theorem). In other words, an algebraic parametrization is the set of power series $\{x_1(t), ..., x_N(t)\}$, where each $x_i(t)$ is different than 0 and for each f in the kernel of ς, $f(x_1(t), ..., x_N(t)) = 0$.

Note that algebraic parameterizations are not unique for algebroid curves, so we answer the question next, how we distinguish between two parameterizations.

Definition 2.1.9 *Let $\mathcal{P}$ and $\mathcal{P}'$ be two parameterizations of the algebroid curve C. The parametrization $\mathcal{P}$ is said to be derived from $\mathcal{P}'$ if there exists a power series $f(T) \in k[[T]]$ of positive order such that $\mathcal{P}$ is given by $\{x_1(f(T)), .., x_N(f(T))\}$, where the parametrization $\mathcal{P}'$ is given by $\{x_1(t), ..., x_N(t)\}$. In this case, we say $\mathcal{P}' \prec \mathcal{P}$.*

Proposition 2.1.10 *The relation $\prec$ is a partial order on the class of algebraic parameterizations of C.*

Definition 2.1.11 *Two parameterizations $\mathcal{P}$ and $\mathcal{P}'$ is said to be equivalent, if $\mathcal{P}' \prec \mathcal{P}$ and $\mathcal{P} \prec \mathcal{P}'$ or equivalently, if they can be derived from each other.*

Definition 2.1.12 *A parametrization is said to be primitive, if it is minimal in the class of algebraic parameterizations of C up to the equivalence.*

Remark 2.1.13 *[10, 1.3.11] Let $\{x_1(t), .., x_N(t)\}$ be a primitive parametrization of C. Then any other parametrization in the same basis is obtained by a substitution of type*

$$f(T) = c_h T^h + c_{h+1} T^{h+1} + ..., \quad h > 0, \quad c_i \in k, \quad c_h \neq 0.$$

The new parametrization is primitive if and only if $h = 1$.

Proposition 2.1.14 *A parametrization given by $\{x_1(t), .., x_N(t)\}$ is primitive if and only if the greatest common divisor of the powers of the terms appearing in series $x_1(t), ..., x_N(t)$ is 1.*

Proof. ($\Rightarrow$:)

Let $\mathcal{P}$ be a primitive parametrization given by $\{x_1(t), .., x_N(t)\}$ and assume that the greatest common divisor of the powers of the terms of $x_1(t), ..., x_N(t)$ is $k \neq 1$, which implies that $x_i'(t) = x_i(t^{\frac{1}{k}})$ is again a power series for $i = 1, .., N$. Hence, $\mathcal{P}$ is derived from the parametrization given by the series $x_1'(T), ..., x_N'(T)$ by substituting $T = f(t) = t^k$, which contradicts the primitiveness of $\mathcal{P}$.

($\Leftarrow$:)

Let the greatest common divisor of the powers of the terms appearing in the series is 1 and assume that the parametrization is not primitive. Then there exists a power series $f(T) \in k[[T]]$ of positive order k, and a primitive parametrization $\mathcal{P}'$ such that $\mathcal{P}$ is derived from $\mathcal{P}'$ by substituting $f(T)$ and is given by $x_1(f(T)), ..., x_N(f(T)))$. This immediately implies that $k = 1$ (Otherwise greatest common divisor can not be 1). Then from Remark 2.1.13, $\mathcal{P}$ is also primitive, in other words $\mathcal{P}$ is equivalent to $\mathcal{P}'$. ∎

The next proposition can be considered as a reformulation of the definition of an algebroid branch.

Proposition 2.1.15 *[13, Proposition 1.3.1] For an algebroid curve $C = \operatorname{Spec} R$, and fixed set of generators for $\mathfrak{m}$, there is a bijection between the primitive parameterizations of C and the algebraic branches of C.*

We can redefine an *algebroid branch* as an equivalence class of primitive parameterizations for an algebroid curve.

2.2 Resolution of Singularities

Before giving the definition of the blowing up of an algebroid curve $C = \operatorname{Spec} R$ at its maximal ideal, we give the basic theory of resolution of singularities of a variety. The general idea of the resolution of singularities of a variety V is determining a smooth variety V' and a birational map $\phi : V' \to V$ (i.e. ϕ has an inverse defined on the complement of finitely many subvarieties of V). The existence of such maps is first proved by Newton in the case of plane curves over the fields of characteristic 0. Then Hironaka proved it for the varieties with arbitrary dimensions over the field of 0 characteristics in 1964. For the varieties of positive characteristic, resolution is known for the dimensions 1 (curves), 2 (surfaces) and 3 for the characteristic greater than five [19].

While there may be many nonsingular projective models for the varieties in higher dimensions, this is not the case for the curve singularities. Although there are several methods of proving the existence of resolution of curve singularities [31], they all construct the same nonsingular projective model. Among the ways of constructing the unique nonsingular projective model for the curves, we present the method of *blowing up*. We blow up the curve until, we obtain a smooth curve.

2.2.1 Blowing up

Blowing up of a variety $V \subset \mathbb{A}^n$ in a singular point P can be thought as just replacing the singular point P by a $\mathbb{P}^{n-1}$, which corresponds to the tangent directions of that point. To define the blowing up, we use Hartshorne's notation [27].

2.2.1.1 Blowing up of $\mathbb{A}^n$ at a point

We first explain the notion of blowing up of the point $O = (0, ..., 0)$ in $\mathbb{A}^n$. Let $x_1, x_2, ..., x_n$ be the affine coordinates in $\mathbb{A}^n$ and let $y_1, y_2, ..., y_n$ be the homogeneous coordinates in $\mathbb{P}^{n-1}$. In the product space $\mathbb{A}^n \times \mathbb{P}^{n-1}$, the closed subsets are the polynomials in x_i, y_j which are homogeneous in y_j.

The blow up of $\mathbb{A}^n$ at the point O is defined to be the closed subset of $\mathbb{A}^n \times \mathbb{P}^{n-1}$ defined by the equations $\{x_i y_j - x_j y_i\}$ for $1 \leq i, j \leq n$, and let us denote this by X.

Consider the natural morphism $\pi : X \to \mathbb{A}^n$ defined by the composition of the natural inclusion $\iota : X \hookrightarrow \mathbb{A}^n \times \mathbb{P}^{n-1}$ and the projection to the first factor from $\mathbb{A}^n \times \mathbb{P}^{n-1}$ to $\mathbb{A}^n$.

- For a point P of the affine n-space other than O, $\pi^{-1}(P)$ is a single point.
- $\pi^{-1}(O) = (0, ..., 0) \times [y_1 : ... : y_n]$ so $\pi^{-1}(O)$ is isomorphic to $\mathbb{P}^{n-1}$.
- ϕ gives an isomorphism from $X - \pi^{-1}(O)$ to $\mathbb{A}^n - O$.
- The points of $\pi^{-1}(O)$ are in one to one correspondence with the lines of $\mathbb{A}^n$ through O.

Definition 2.2.1 *The fibre $E = \pi^{-1}(O)$ isomorphic to $\mathbb{P}^{n-1}$ is called the exceptional divisor.*

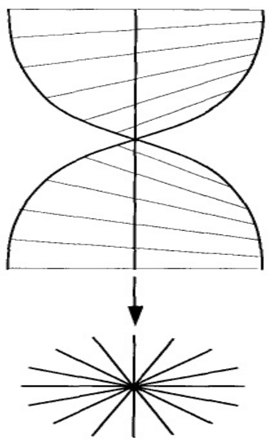


Figure 2.2: Blowing up of origin

2.2.1.2 Blowing up of a variety at a point

Definition 2.2.2 *For a variety V of $\mathbb{A}^n$ with singularity at the origin, blowing up of V at the origin is the Zariski closure of the inverse image $\pi^{-1}(V - O)$, where $\pi : X \to \mathbb{A}^n$ is the blowing up of $\mathbb{A}^n$ at O. The restriction of $\pi : X \to \mathbb{A}^n$ is also denoted by $\pi : \tilde{Y} \to V$*

Similar to the blowing up of $\mathbb{A}^n$, π induces an isomorphism from $\tilde{Y} - \pi^{-1}(O)$ to $Y - O$, that is, π is a birational morphism of Y on to $\tilde{Y}$.

Blowing up of a variety or $\mathbb{A}^n$ at a point $P \neq O$ can be done in the same way, after a linear change of coordinates sending P to O.

Example 2.2.3 *Let V be the nodal cubic curve given by the equation $y^2 = x^2(1 + x)$ in $\mathbb{A}^2$. It has a singularity at the origin $O = (0, 0)$. If we set u, v to be the homogeneous coordinates of $\mathbb{P}^1$, the blow up of $\mathbb{A}^2$ at the origin is given by $xv - yu$. Blowing up of the curve is given by this equation together with the equation of the curve. That is, if we look at the blown up curve in the chart $u \neq 0$, we have $y = xv$. Substituting this in the equation of the curve, we get $x^2(v^2 - 1 - x) = 0$, which says that the blow up has two components. One is $x = 0$, which corresponds to the exceptional divisor, and the other is $v^2 = 1 + x$. Observe that the two intersects at two points: $(0, 0) \times [1 : -1]$ and $(0, 0) \times [1 : 1]$, see Figure 2.3.*

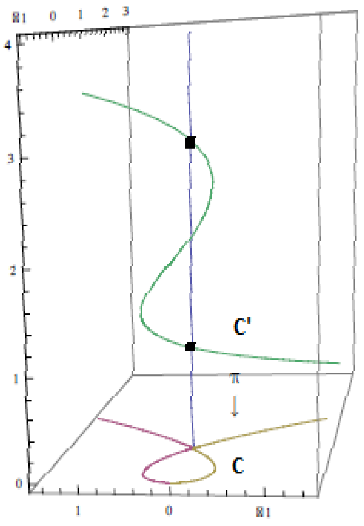


Figure 2.3: Blowing up of the nodal cubic

Example 2.2.4 *Let C be the cuspidal cubic with the defining equation $y^2 = x^3$. The blow up* *of C in the chart $u \neq 0$ is the smooth curve $v^2 = x$ and it intersects the exceptional divisor at the point* $(0,0) \times [1:0]$.

Observe that we got a nonsingular curve after blowing up the nodal cubic and the cuspidal cubic curve, but this is not the case for all the curves. That is, we may get a singular curve even after blowing up a singular curve.

Example 2.2.5 *Let C be the curve defined by the equation $y^2 = x^5$. First blow up gives the curve $v^2 = x^3$ in the first chart $u \neq 0$. The intersection of the blown up curve with the exceptional divisor is* $(0,0) \times [1:0]$ *and the blown up curve is singular there. Relabeling the blown up curve as $y^2 = x^3$ and blowing up again at the point* $(0,0)$, *we get a smooth curve $y^2 = x$.*

Observe that in the first example, we have started with a node, which has two local components around the point $(0,0)$ (branches), the blown up curve and the exceptional divisor had intersections at two different points. On the other hand, in the second and third examples, we had started with a branch, so that the exceptional divisor and the blown up curve had intersection at exactly one point, namely the singular point of the blown up curve. We can continue blowing up at that point.

Remark 2.2.6 *[38] The blow up of an algebroid branch has intersection with exceptional divisor at most one point.*

2.2.2 Blowing up of a branch

Let the algebroid curve branch $C = \operatorname{Spec} R$ be given by the parametrization

$$\{x_1 = \varphi_1(t),\ x_2 = \varphi_2(t),\ ...,\ x_n = \varphi_n(t)\}$$

with $\varphi_1(t)$ having the smallest order.

The blowing up of C at its maximal ideal is the algebroid branch $C_1 = \operatorname{Spec} R_1$ given by the parametrization

$$\{x_1 = \varphi_1(t),\ x_2 = \frac{\varphi_2(t)}{\varphi_1(t)},\ ...,\ x_n = \frac{\varphi_n(t)}{\varphi_1(t)}\}$$

Definition 2.2.7 *[13, Remark 1.9.1] The multiplicity of the algebroid curve branch with parametrization* $\{x_1 = \varphi_1(t),\ x_2 = \varphi_2(t),\ ...,\ x_n = \varphi_n(t)\}$ *is the minimum of the orders of the series* $\varphi_i(t)$ *for* $1 \leq i \leq n$.

Note that the multiplicity of an algebroid curve branch $C = \operatorname{Spec} R$ is also equal to the multiplicity of the local ring R and denoted by $e(R)$.

Example 2.2.8 *Let C be the algebroid curve branch given by parametrization* $\{x_1 = t^2, x_2 = t^3\}$*, and thus with multiplicity* 2*. Note that the local ring* $k[[t^2, t^3]]$ *is the completion of the localization at the singular point (0,0) of the coordinate ring of the cuspidal cubic. Then the blow up of C is given by the parametrization* $\{x_1 = t, x_2 = t^2\}$ *which has multiplicity 1.*

Remark 2.2.9 *[29] Let V be a singular variety over a field of characteristic zero. The singularity can be resolved by applying finitely many blowing ups to V.*

2.2.3 Singularity Theory

Let $C = C_0$ be a curve branch with singularity of multiplicity $m = m_0$ at the origin $O = O_0$. Blowing up C_0 at O_0, we get a curve branch C_1, the blowing up map $\pi_1 : C_1 \to C_0$ and the exceptional divisor $E_1 = {\pi_1}^{-1}(O_0)$ of the blow up. E_1 and C_1 meet at exactly one point, say O_1. Let m_1 be the multiplicity of O_1. Now, blowing up C_1 at O_1, we get a curve branch C_2, a map $\pi_2 : C_2 \to C_1$ and the exceptional divisor $E_2 = \pi_2^{-1}(O_1)$. E_2 and C_2 meet at a point O_2 with multiplicity m_2. Continuing k steps in this way inductively, we get

$$C_k \xrightarrow{\pi_k} C_{k-1} \xrightarrow{\pi_{k-1}} \cdots \xrightarrow{\pi_3} C_2 \xrightarrow{\pi_2} C_1 \xrightarrow{\pi_1} C_0$$

Definition 2.2.10 *The sequence of points* $(O_0, O_1, ..., O_k)$ *is called the infinitely near points of C . The sequence of natural numbers* $(m_0, m_1, ..., m_k)$ *is called the multiplicity sequence of the curve branch. The sum* $m_0 + m_1 + ... + m_i$ *is said to be the ith multiplicity sum or the multiplicity sum corresponding to the point* O_i.

Definition 2.2.11 *Two curve branches are said to be equiresoluble or simply equivalent iff they have the same multiplicity sequence.*

Our purpose is to classify curve branches up to their multiplicity sequences. In [23], Du Val showed that there are some 'special' numbers, which he called the characters of the curve. By applying the modified Jacobian algorithm to these numbers, we obtain the multiplicity sequence of the curve branch. We first give some necessary definitions to understand the geometric significance of these characters.

Definition 2.2.12 *The point O_i is said to be proximate to the point O_j iff O_i is in the proper transform of E_j under the morphism $\pi_i \circ \pi_{i-1} \circ \cdots \circ \pi_{j+1}$ for $i > j + 1$.*

Remark 2.2.13 *Let $(O_0, O_1, ..., O_k)$ be the infinitely near points of C. Then, O_i is proximate to O_{i-1} for any i.*

Proposition 2.2.14 *[23] Let $(O_0, O_1, ..., O_k)$ be the infinitely near points of C in n space. Then the following hold:*

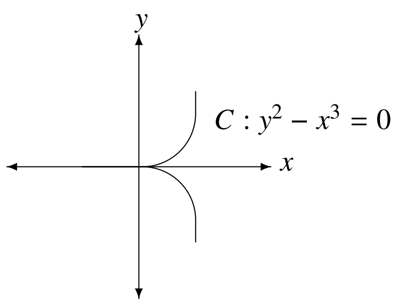
- *If O_i is proximate to O_j and O_s with $s < j < i$, then O_j is also proximate to O_s.*
- *The number of points that O_i is proximate to can not be greater than n.*
- *The multiplicity of C at O_i is equal to the sum of the multiplicities of the points which are proximate to O_i.*

Definition 2.2.15 *The restriction of a point O_i, namely $r(O_i)$ is defined to be the number of points to which O_i is proximate to.*

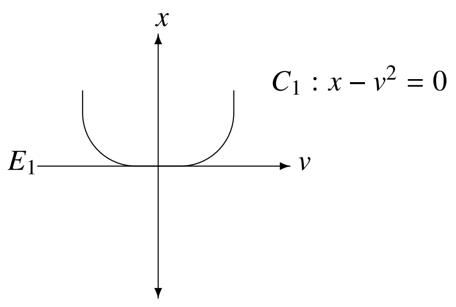
Definition 2.2.16 *A point O_i is said to be a leading point iff its restriction is less than the restriction of O_{i+1}.*

Definition 2.2.17 *The multiplicity sum corresponding to a leading point is called a character of the curve branch.*

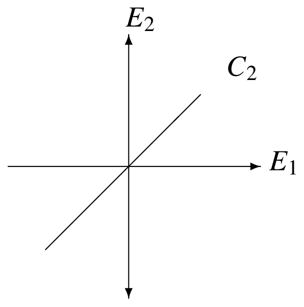
Example 2.2.18 *Consider the irreducible plane curve C defined by $y^2 - x^3 = 0$.*



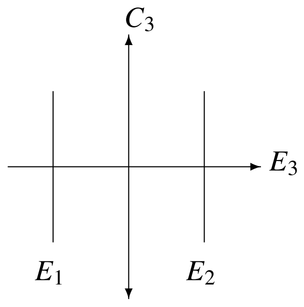
Blowing up C, we get the curve C_1 given by $y^2 - x = 0$ and the exceptional divisor E_1 given with $(0, 0) \times [1 : v]$.



Now we have a smooth curve but the intersection of C_1 and E_1 is not transversal so we continue blowing up till we reach a good resolution. We get the curve C_2 given by $y = x$ and the exceptional divisor $(0, 0) \times [u : 1]$.

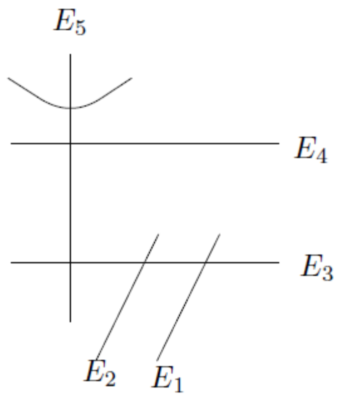


Again, intersection of C_2 is not transversal with the exceptional divisor, blowing up once more, we reach:



So we see that E_3 intersects E_1. This shows that O_1 is on E_3, that is, O_3 is proximate to O_1. We already know that O_2 is proximate O_1. Then $r(O_1) = 0$, $r(O_2) = 1$ and $r(O_3) = 2$ telling us that O_1 and O_2 are leading points. As the multiplicity sequence of the curve branch is $(2, 1, 1)$, the multiplicity sum corresponding to O_1 is 2 and the multiplicity sum corresponding to O_2 is $1 + 2 = 3$. So the characters of the curve branch are 2 and 3.

Example 2.2.19 *[21, Example 5.3.7] For the curve C with defining equation $y^4 - 2x^3y^2 - 4x^5y + x^6 - x^7 = 0$, we can see from [21] that the exceptional divisors look like:*



We already know the proximity relations $O_5 \to O_4$, $O_4 \to O_3$, $O_3 \to O_2$ and $O_2 \to O_1$ as every point is proximate to its predecessor. But from the resolution process, we see from the graph that E_3 intersects E_1 so O_3 is also proximate to O_1 and O_5 is also proximate to O_3 as E_5 intersects E_3. These all say that $r(O_1) = 0$, $r(O_2) = 1$, $r(O_3) = 2$, $r(O_4) = 1$ and $r(O_5) = 2$ and the leading points of the curve branch are O_1, O_2 and O_4. It can be shown that the multiplicity sequence of the curve branch is $(4, 2, 2, 1, 1)$ so the characters of the curve branch are, $4, 4 + 2 = 6, 4 + 2 + 2 + 1 = 9$.

Remark 2.2.20 *Observe that we can check the proximity relations in Example 2.2.18 and*

Example 2.2.19 by the help of Proposition 2.2.14. Indeed, for the Example 2.2.18, we know that the multiplicity sequence is $(2, 1, 1)$*. That is, we have the relations* $m_1 = m_2 + m_3$ *and* $m_2 = m_3$*. Then the last part of Proposition 2.2.14 tells us that* O_3 *and* O_2 *are proximate to* O_1 *and* O_3 *is proximate to* O_2*. In the same way, we can see that* $m_1 = m_2 + m_3$*,* $m_2 = m_3$*,* $m_3 = m_4 + m_5$ *and* $m_4 = m_5$ *for Example 2.2.19. Hence* O_2 *and* O_3 *are proximate to* O_1*,* O_3 *is proximate to* O_2*,* O_4 *and* O_5 *are proximate to* O_3 *and* O_5 *is proximate to* O_4 *which supports our previous results.*

Proposition 2.2.21 *If the characters of the curve branch are defined as above, the modified Jacobian algorithm applied to these characters gives the multiplicity sequence of the curve branch.*

Example 2.2.22 *For the curve branch defined by* $y^2 - x^3 = 0$*, we now know that the characters are* 2 *and* 3*. Applying the modified jacobian algorithm to the characters, we indeed obtain the the multiplicity sequence* $(2, 1, 1)$*. In the same way, for the curve branch defined by* $y^4 - 2x^3y^2 - 4x^5y + x^6 - x^7 = 0$*, the characters are* $4, 6, 9$ *and the modified jacobian algorithm applied to these numbers gives* $(4, 2, 2, 1, 1)$*, which is indeed the multiplicity sequence of the branch.*

In his paper [23], Du Val explains the modified Jacobian Algorithm and proved the previous statement. However, it was not known how to compute the characters of a space curve branch. In [1], Cahit Arf found the algebraic pattern behind this geometric problem and presented a way to calculate these characters, which will be explained in Chapter 5.

CHAPTER 3

PLANE CURVE SINGULARITIES

In this chapter we give a summary of the results obtained in the study of plane algebroid curves. To understand the significance of Arf's contribution, we must understand the plane algebroid curve case. In the first section, we deal with irreducible plane (algebroid) curves, namely plane (algebroid) curve branches. Certain numbers and graphs which can be obtained from the local ring of an algebroid curve branch are said to be invariants. We introduce the invariants of plane curve branches and the relations between them. The invariants that we are interested in are the semigroup of values, Puiseux pairs, characteristic exponents, multiplicity sequence and resolution graph of the curve branch. We also present the Puiseux expansions and Hamburger-Noether expansions. In the second section, we deal with reducible algebroid plane curves, plane curves with multiple branches. We give the definition of the contact number of the branches of curve and the definition of resolution graph for reducible plane algebroid curves. These two are the invariants of reducible plane curves together with the ones we mention for the plane branches. We introduce the relation between the contact number and the intersection multiplicity of the branches. We also explain the relation between the contact numbers together with the resolution graph of branches and resolution graph of reducible plane curve. For most of the definitions and theorems, we follow [21].

3.1 Plane Curve Branches

In this section, we work with irreducible algebroid plane curves. The word irreducible here means that the curve does not have multiple branches, it has only one branch. That is, if $f(x, y) = 0$ is the defining equation of an algebroid plane curve, f is irreducible in $k[[x, y]]$.

Let R be the local ring of the plane curve branch C and let $\overline{R}$ be the integral closure of R in its ring of fractions. As we have already mentioned in 2.1.1, $\overline{R} \cong k[[t]]$.

Definition 3.1.1 *The number $dim_k(\overline{R}/R)$ is called the δ invariant of the branch and it is denoted by $\delta(R)$.*

Definition 3.1.2 *The set $W(R) = \{ord(r) \mid r \in R\}$ consisting of the orders of the elements of R is called the semigroup of values or semigroup of orders of R.*

Theorem 3.1.3 *[21, Lemma 5.2.2] The δ invariant is equal to the number of gaps in the semigroup of values of R. That is, $\delta(R)$ is equal to the cardinality of the set $\mathbb{N} - W(R)$.*

Definition 3.1.4 *The number $min\{h \in W(R) \mid h + \mathbb{N} \subset W(R)\}$ is called the conductor of the semigroup of values of R.*

Theorem 3.1.5 *(Gorenstein)[21, Theorem 5.2.4] Let c be the conductor of the semigroup of values of R. Then $c = 2\delta(R)$.*

Remark 3.1.6 *Note that this is true for plane algebroid curves. In general, for a space curve branch, $c \leq 2\delta(R)$ [15]. For the curve branches with symmetric semigroup of values, $c = 2\delta(R)$.*

Remark 3.1.7 *[40, Theorem 2.1.1] Let C be a plane curve branch given by the primitive parametrization $x(t)$, $y(t) \in k[[t]]$. With a coordinate change and interchanging x and y if necessary, we can assume that the parametrization is of Puiseux type. That is $x(t) = t^n$ and $y(t) = \sum a_i t^i$ with $ord(y(t)) \geq n$ and $a_i \in k$.*

Let C be a plane curve branch given in Puiseux form $x(t) = t^n$ and $y(t) = \sum a_i t^i$ with $n \leq ord(y(t))$. Let β_0 be n and consider the powers of the terms appearing in the power series $y(t)$. Let β_1 be the smallest power that is not divisible by n and let e_1 be the greatest common divisor of β_0 and β_1. Then define inductively β_i to be the smallest power for which $gcd(\beta_0, \beta_1, ..., \beta_i) < gcd(\beta_0, \beta_1, ..., \beta_{i-1})$ and $e_i = gcd(\beta_0, \beta_1, ..., \beta_i)$. After finitely many steps, we get $e_q = 1$ for some q and we stop.

Definition 3.1.8 *The set* $\{\beta_0, \beta_1, ..., \beta_q\}$ *is called the characteristic exponents of the curve branch.*

Definition 3.1.9 *The number* q *is called the genus of* C.

Example 3.1.10 *Let* C *be a plane curve branch given with the parametrization* $x(t) = t^6$ *and* $y(t) = t^{12} + t^{15} + t^{18} + t^{20}$. *Then,*

$\beta_0 = 6,$

$\beta_1 = 15, \quad e_1 = gcd(6, 15) = 3$

$\beta_2 = 20, \quad e_2 = gcd(6, 15, 20) = 1$

So the characteristic exponents are $\{6,15,20\}$.

Remark 3.1.11 *Plane curve branches with only one characteristic exponent* $\{\beta_0 = 1\}$ *are smooth.*

Theorem 3.1.12 *[40, Theorem 3.5.5] Let* C *be a plane curve branch with local ring* R *and characteristic exponents* $\{\beta_0, \beta_1, ..., \beta_q\}$ *and let* R_1 *be the local ring of the blowing-up of* C. *Then the characteristic exponents of the blown up curve is,*

(i) $\{\beta_1, \beta_1 - \beta_0, ..., \beta_q - \beta_0\}$ *if* $\beta_1 > 2\beta_0$

(ii) $\{\beta_1 - \beta_0, \beta_0, \beta_2 - \beta_1 + \beta_0, ..., \beta_q - \beta_1 + \beta_0\}$ *if* $\beta_1 < 2\beta_0$ *and* $\beta_1 - \beta_0$ *does not divide* β_0

(iii) $\{\beta_1 - \beta_0, \beta_2 - \beta_1 + \beta_0, ..., \beta_q - \beta_1 + \beta_0\}$ *if* $\beta_1 < 2\beta_0$ *and* $\beta_1 - \beta_0$ *divides* β_0.

Example 3.1.13 *Consider the plane curve branch* C *in Example 3.1.10. According to the Theorem 3.1.12, the plane curve branch* C', *that is obtained by blowing up* C, *will have characteristic exponents* $(6, 9, 14)$ *as* $\beta_1 = 15 > 2\beta_0 = 12$ *for* C.
Indeed, by blowing up C, *we see that* C' *is parameterized by* $x = t^6$, *and* $y = t^6 + t^9 + t^{12} + t^{14}$ *by 2.2.2. Now it is easy to see from the definition of characteristic exponents that* C' *has characteristic exponents* $(6, 9, 14)$.

Before explaining the connection of the multiplicity sequence and the characteristic exponents, we need to introduce a notation. Let m and n be two natural numbers with $m > n$. If the Euclidian algorithm applied to m and n gives:

$$\begin{aligned} m &= q_0 n + r_1 \\ n &= q_1 r_1 + r_2 \\ .... & \quad ... \\ r_{s-1} &= q_s r_s \end{aligned}$$

The sequence $\underbrace{n, ..., n}_{q_0 \text{ times}}, \underbrace{r_1, ..., r_1}_{q_1 \text{ times}}, ..., \underbrace{r_s, ..., r_s}_{q_s \text{ times}}$ is denoted by $M(n, m)$.

The next lemma expressed in a different way can be found in [10, Lemma 3.3.6], but the proof given belongs to us.

Lemma 3.1.14 *Let n and m be two natural numbers, $e = gcd(n, m)$ and also let d_i be the divisors obtained by applying the Euclidian algorithm to n and m. In this case $\sum_i d_i = n+m-e$*

Proof. We prove the lemma by induction on k = the number of different divisors. Without loss of generalization, assume $n < m$.

Let $k = 2$.

$$\begin{array}{ll|l}
\begin{array}{l} n \\ n \\ \vdots \end{array} & \begin{array}{l} m \\ m-n \\ \vdots \end{array} & \left.\begin{array}{l} n \\ n \\ \vdots \end{array}\right\} k_1 \text{ times} \\
\begin{array}{l} n \\ (k_1+1)n - m \\ \vdots \\ n-(k_2-1)(m-k_1 n) \end{array} & \begin{array}{l} m-k_1 n \\ m-k_1 n \\ \vdots \\ m-k_1 n \end{array} & \left.\begin{array}{l} m-k_1 n \\ m-k_1 n \\ \vdots \\ m-k_1 n \end{array}\right\} k_2 \text{ times}
\end{array}$$

Here, $k_1 n < m < (k_1+1)n$, $n = k_2(m - k_1 n)$ and $n-(k_2-1)(m-k_1 n) = m - k_1 n = gcd(n, m)$. Then the sum of divisors is $k_1 n + k_2(m - k_1 n) = k_1 n + n = n + m - m + k_1 n = n + m - gcd(n, m)$. Now assume we have k different divisors and assume also the claim is true for k-1 different divisors . Then,

$$\begin{array}{ll|l}
\begin{array}{l} n \\ \vdots \\ n \end{array} & \begin{array}{l} m \\ \vdots \\ m-(k_1-1)n \end{array} & \left.\begin{array}{l} n \\ \vdots \\ n \end{array}\right\} k_1 \text{ times} \\
\begin{array}{l} n \\ \vdots \\ e \end{array} & \begin{array}{l} m-k_1 n \\ \vdots \\ e \end{array} & \left.\begin{array}{l} m-k_1 n \\ \vdots \\ e \end{array}\right\} k-1 \text{ different divisors}
\end{array}$$

Here, we have $k_1 n < m < (k_1 + 1)n$ and $e = gcd(n, m)$. Observing that $e = gcd(n, m - k_1 n)$ also, the sum of the divisors is the summation of $k_1 n$ and the sum of divisors in the Euclidian algorithm of n and $m - k_1 n$. From the induction assumption, the sum of divisors in the Euclidian algorithm of n and $m - k_1 n$ is $n + m - k_1 n - e$. Thus, the sum of the divisors obtained by applying the Euclidian algorithm to n and m is $k_1 n + n + m - k_1 n - d = n + m - e$, which completes the proof. ■

Now, as a consequence of Theorem 3.1.12, we can state the next theorem, which will be used in the latter chapters.

Theorem 3.1.15 *[4] Let C be a plane curve branch with characteristic exponents* $\{\beta_0, \beta_1, ..., \beta_q\}$. *Then* e_i *being* $gcd(\beta_0, ..., \beta_i)$ *and q being the smallest number such that* $e_q = 1$, *the multiplicity sequence of C is* $M(\beta_0, \beta_1), M(e_1, \beta_2 - \beta_1), ..., M(e_{q-1}, \beta_q - \beta_{q-1})$.

Example 3.1.16 *Let C be the plane curve branch with the parametrization* $x = t^6$ *and* $y = t^{12} + t^{15} + t^{18} + t^{20}$. *In Example 3.1.10, we have shown that the characteristic exponents are* $\{6, 15, 20\}$. *So the multiplicity sequence of the curve branch is* $M(6, 15)$, $M(gcd(6, 15), 20 - 15)$*, which gives* $6, 6, 3, 3, 3, 2, 1, 1$

The following lemma and Lemma 3.1.14 give us the opportunity to prove Corollary 3.1.18 which will also be used in the latter chapters.

Lemma 3.1.17 *Let* $a_1, a_2, \dots, a_k$ *be natural numbers s.t.* $gcd(a_1, \dots, a_k) = 1$. *Define* $b_1 = a_1$, $b_i = gcd(b_{i-1}, a_i) = gcd(a_1, \dots, a_i)$ $(1 < i \le k)$ *inductively. Then,*

$$gcd(b_{i-1}, a_i - a_{i-1}) = b_i$$

Proof. Since $b_i = gcd(b_{i-1}, a_i)$ and $b_{i-1} = gcd(b_{i-2}, a_{i-1})$, we have $b_{i-1} = b_i h$, $a_i = b_i r$, $a_{i-1} = b_{i-1} p$ and $b_{i-2} = b_{i-1} q$, where $gcd(h, r) = 1$ and $gcd(p, q) = 1$. Then $gcd(b_{i-1}, a_i - a_{i-1}) = gcd(b_i h, b_i(r - hp)) = b_i$ as $gcd(h, r) = 1$ ■

Corollary 3.1.18 *Let C be a curve branch with characteristic exponents* $\{\beta_0, \beta_1, ..., \beta_q\}$. *Then the sum of the multiplicities appearing in the multiplicity sequence of C is* $\beta_0 + \beta_q - 1$

Proof. By Lemma 3.1.14, the sum of the multiplicities is $\beta_0 + \beta_1 - gcd(\beta_0, \beta_1) + gcd(\beta_0, \beta_1) + \beta_2 - \beta_1 + gcd(gcd(\beta_0, \beta_1), \beta_2 - \beta_1) + ... + gcd(\beta_1, ..., \beta_{q-1}) + \beta_q - \beta_{q-1} - 1$ and by Lemma 3.1.17, this is equal to $\beta_0 + \beta_q - 1$. ■

Corollary 3.1.19 *[40, Theorem 3.5.6] Characteristic exponents of a plane curve branch C determine the multiplicity sequence of C and the multiplicity sequence determines the characteristic exponents.*

Let C be a plane curve branch with the characteristic exponents $\{\beta_0, \beta_1, ..., \beta_q\}$. Let n_1 and m_1 be natural numbers such that $\frac{\beta_1}{\beta_0} = \frac{n_1}{m_1}$ and $gcd(m_1, n_1) = 1$. Similarly for $i = 2, ..., q$, define the natural numbers n_i and m_i such that $\frac{\beta_i}{\beta_0} = \frac{n_i}{m_1 ... m_i}$ and $gcd(m_i, n_i) = 1$.

Definition 3.1.20 *The pairs $\{(m_1, n_1), ..., (m_q, n_q)\}$ defined above are called the Puiseux pairs of C.*

Remark 3.1.21 *[21, Remark 5.2.9] Puiseux pairs determine the characteristic exponents and characteristic exponents determine the Puiseux pairs.*

Remark 3.1.22 *Characteristic exponents are independent of the chosen parametrization of the curve branch.*

Next, we present another method for considering the resolution of the plane curve branch.

Definition 3.1.23 *Let C be a plane curve branch with a singularity at the origin and consider the resolution process of C:*

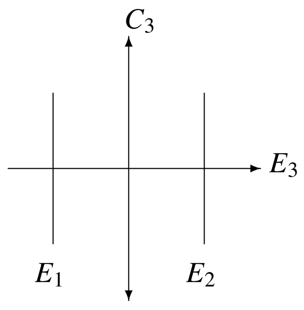
$$C_k \xrightarrow{\pi_k} C_{k-1} \xrightarrow{\pi_{k-1}} \cdots \xrightarrow{\pi_2} C_1 \xrightarrow{\pi_1} C$$

The resolution is said to be a good resolution, if C_{i-1} still has a singular point for $i \leq k$ or C_{i-1} is smooth but the intersection of C_{i-1} with the exceptional divisor E_{i-1} is not transversal. Furthermore C_k is smooth and has transversal intersection with the exceptional divisor.

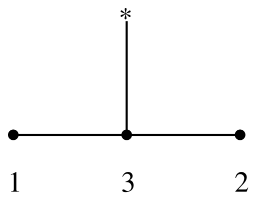
Remark 3.1.24 *The multiplicity sequence for a good resolution ends with a sequence of* 1*'s.*

Definition 3.1.25 *The weighted graph constructed by associating "$\bullet$" for each E_i and a "$*$" for C_k as vertices, and connecting the vertices if they intersect is called the resolution graph of C. The weight of E_i is i in the graph.*

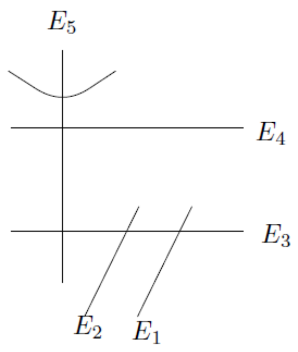
Example 3.1.26 *Consider the irreducible plane curve C defined by $y^2 - x^3 = 0$. We already know from Example 2.2.18 that the exceptional divisors look like:*



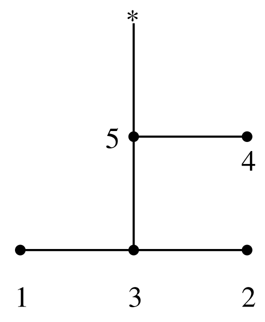
So, the resolution graph of C must be:



Example 3.1.27 *[21, Example 5.3.7] For the curve C with defining equation $y^4 - 2x^3y^2 - 4x^5y + x^6 - x^7 = 0$, the exceptional divisors look like:*



So the resolution graph looks like:



Theorem 3.1.28 *[21, Theorem 5.3.9] Let C be a plane curve branch. The following data for C determine each other:*

1. *the Puiseux pairs of C.*
2. *the characteristic exponents of C.*
3. *the minimal set of generators for the semigroup of C.*
4. *the multiplicity sequence of C.*
5. *the resolution graph*

We already know that $1, 2, 3, 4$ determine each other, so let us explain the relationship between the characteristic exponents and the resolution graph. For all the proofs and details, see [21].

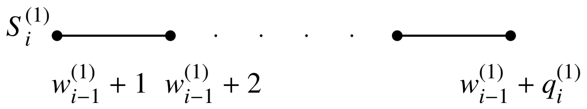
Let $(\beta_0, \beta_1, ..., \beta_q)$ be the characteristic exponents of a curve branch C and let e_i be the greatest common divisor of $\beta_0, \beta_1, ..., \beta_i$ for $0 \leq i \leq q$. The multiplicity sequence of C is $M(\beta_0, \beta_1)$, $M(e_1, \beta_2 - \beta_1), ..., M(e_{q-1}, \beta_q - \beta_{q-1})$ from Theorem 3.1.15. Here, each Euclidian algorithm forms a layer $S^{(i)}$ of the resolution graph and each different divisor in a Euclidian algorithm corresponds to a building block $S^{(i)}_j$ of $S^{(i)}$. To be more precise, consider $M(\beta_0, \beta_1)$. It forms the first layer $S^{(1)}$ of the resolution graph:

$$\begin{aligned} \beta_1 &= q^{(1)}_1 \beta_0 + r^{(1)}_1 \\ \beta_0 &= q^{(1)}_2 r^{(1)}_1 + r^{(1)}_2 \\ .... &\quad ... \\ r^{(1)}_{s_1-2} &= q^{(1)}_{s_1} e_1 \end{aligned}$$

Here, we have s_1 different divisors $(\beta_0, r_1^{(1)}, .., r_{s_1-2}^{(1)}, e_1)$, so the first layer contains s_1 building blocks $S_1^{(1)}, .., S_{s_1}^{(1)}$. For the first division, we have $q_1^{(1)}$ as the quotient, so the first building block $S_1^{(1)}$ is $\{1, 2, ..., q_1^{(1)}\}$.

$S_1^{(1)}$
1
2
$q_1^{(1)} - 1$
$q_1^{(1)}$

Similarly, since we have $q_i^{(1)}$ as a quotient in the ith division $r_{i-2}^{(1)} = q_i^{(1)} r_{i-1}^{(1)} + r_i^{(1)}$, $w_{i-1}^{(1)}$ denoting the largest weight on $S_{i-1}^{(1)}$, $S_i^{(1)}$ is:



Observing that $w_i^{(1)} = w_{i-1}^{(1)} + q_i^{(1)}$, we can start connecting the building blocks of $S^{(1)}$.

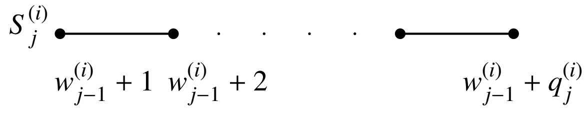
If $s_1 > 2$ and $j + 2 \leq s_1$, we connect the end point of $S_j^{(1)}$ with the start point of $S_{j+2}^{(1)}$. Otherwise, we connect the end point of $S_j^{(1)}$ with the end point of $S_{j+1}^{(1)}$.

$S_1^{(1)}, S_2^{(1)}, ..., S_{s_1}^{(1)}$ connected in this way forms the first layer $S^{(1)}$ of the resolution graph.

Similarly $S^{(i)}$ is formed by the Euclidian algorithm $M(e_{i+1}, \beta_{i+2} - \beta_{i+1})$ for $i = 2, ..., q$.

$$\begin{aligned}
\beta_{i+2} - \beta_{i+1} &= q_1^{(i)} e_{i+1} + r_1^{(i)} \\
e_{i+1} &= q_2^{(i)} r_1^{(i)} + r_2^{(i)} \\
.... &\ ... \\
r_{s_i-2}^{(i)} &= q_{s_i}^{(i)} e_{i+1}
\end{aligned}$$

$S^{(i)}$ will be formed by $S_1^{(i)}, S_2^{(i)}, ..., S_{s_i}^{(i)}$ by connecting the end point of $S_j^{(i)}$ with the start point of $S_{j+2}^{(i)}$, if $s_i > 2$ and $j + 2 \leq s_i$, and the end point of $S_j^{(i)}$ with the end point of $S_{j+1}^{(i)}$ otherwise. The smallest weight of $S_1^{(i)}$ will be the largest weight of the last building block of $S^{(i-1)}$ plus one. (Note that, the point with the smallest weight is said to be the start point and the point with the largest weight is said to be the end point of $S_j^{(i)}$. Here, $S_j^{(i)}$ is

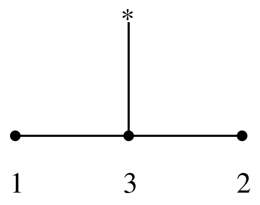


The resolution graph of C is formed from $S^{(1)}, ...S^{(q)}$ by connecting the end point of $S^{(i)}_{s_i}$ with the start point of $S^{(i+1)}_1$ if $q^{(i+1)}_1 \neq 0$. If $q^{(i+1)}_1 = 0$, the connection point depends on s_{i+1}. To be more precise, if $q^{(i+1)}_1 = 0$ and $s_{i+1} \geq 3$, the end point of $S^{(i)}_{s_i}$ is being connected with the start point of $S^{(i+1)}_3$, if $q^{(i+1)}_1 = 0$ and $s_{i+1} = 2$, then $S^{(i)}_{s_i}$ is being connected with the end point of $S^{(i+1)}_2$.

Example 3.1.29 *Let's find the resolution graph of the curve branch given with the equation $y^2 - x^3 = 0$. Since it can be parameterized as $x = t^2$, $y = t^3$, its characteristic exponents are $(2, 3)$. This implies that the resolution graph of C will consist of only one layer $S^{(1)}$, which is determined by the Euclidian algorithm $M(2, 3)$.*

$$\begin{aligned} 3 &= 1 \cdot 2 + 1 \\ 2 &= 2 \cdot 1 \end{aligned}$$

$S^{(1)}_1 = \{1\}$ *as* $q^{(1)}_1 = 1$ *and* $S^{(1)}_2 = \{2, 3\}$ *as* $q^{(1)}_2 = 2$ *and as* $s_1 = 2$, *we connect the end point of* $S^{(1)}_1$ *with the end point of* $S^{(1)}_2$. *So the resolution graph is*



We can check from Example 3.1.26, that our result is the same with that one.

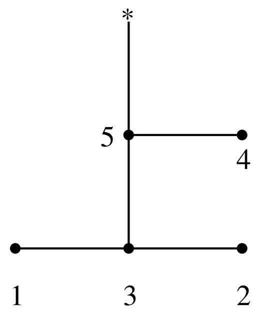
Example 3.1.30 *Let C be the curve in Example 3.1.27, defined by $x - 4y - 4xy - xy^2 - x^2y^2 = 0$. It can be shown that its characteristic exponents are $(4, 6, 7)$. So the resolution graph of C contains two layers, which are determined by $M(4, 6)$ and $M(2, 1)$ respectively. First, the Euclidian algorithm is*

$$\begin{aligned} 6 &= 1 \cdot 4 + 2 \\ 4 &= 2 \cdot 2 \end{aligned}$$

So $S_1^{(1)} = \{1\}$*,* $S_2^{(1)} = \{2,3\}$*. We connect the end point of* $S_1^{(1)}$ *with the end point of* $S_2^{(1)}$ *as* $s_1 = 2$*. The second Euclidian algorithm is*

$$\begin{aligned} 1 &= 0 \cdot 2 + 1 \\ 2 &= 2 \cdot 1 \end{aligned}$$

So $S_1^{(2)} = \emptyset$*,* $S_2^{(2)} = \{4,5\}$*. As* $q_1^{(2)} = 0$ *and* $s_2 = 2$*, we connect the end point of* $S_2^{(1)}$ *with the end point of* $S_1^{(2)}$*. So the resolution graph of C is*



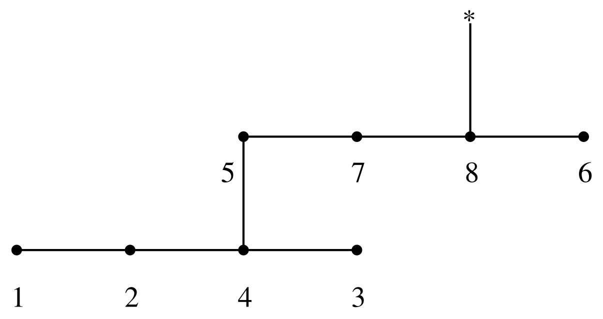
Example 3.1.31 *Let C be the curve in Example 3.1.10 with characteristic exponents* $(6,9,20)$*. Let's form the resolution graph of C. The resolution graph consists of two layers* $S^{(1)}$ *and* $S^{(2)}$*, which are determined by Euclidian algorithms* $M(6,15)$ *and* $M(3,5)$ *respectively. First Euclidian algorithm is*

$$\begin{aligned} 15 &= 2 \cdot 6 + 3 \\ 6 &= 3 \cdot 2 + 0 \end{aligned}$$

which tells us $S_1^{(1)} = \{1,2\}$ *and* $S_2^{(1)} = \{3,4\}$*. We connect the end point of* $S_1^{(1)}$ *with the end point of* $S_2^{(1)}$ *as* $s_1 = 2$*. The second Euclidian algorithm is*

$$\begin{aligned} 5 &= 1 \cdot 3 + 2 \\ 3 &= 1 \cdot 2 + 1 \\ 2 &= 2 \cdot 1 + 0 \end{aligned}$$

So the second layer contains $S_1^{(2)} = \{5\}$*,* $S_2^{(2)} = \{6\}$ *and* $S_3^{(2)} = \{7,8\}$*. The resolution graph of C is:*



Example 3.1.32 *Let C be the curve defined by the parametrization $x(t) = t^{28}$, $y(t) = t^{56} + t^{64} + t^{66} + t^{77}$. Let's find the resolution graph of C. The characteristic exponents of C are* $(28, 64, 66, 77)$ *so the resolution graph consists of three layers, which are determined by the Euclidian algorithms $M(28, 64)$, $M(4, 2)$ and $M(2, 11)$ respectively. The first Euclidian algorithm gives*

$$\begin{aligned} 64 &= 2 \cdot 28 + 8 \\ 28 &= 3 \cdot 8 + 4 \\ 8 &= 2 \cdot 4 + 0 \end{aligned}$$

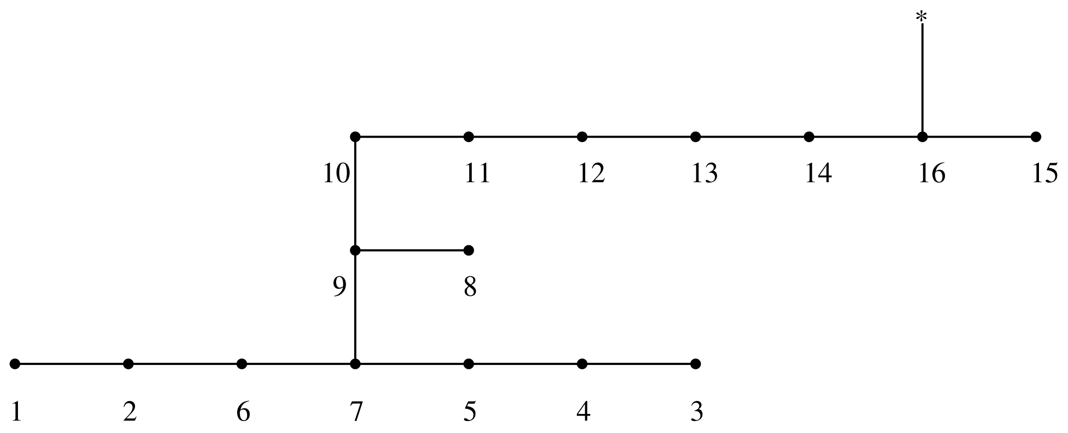
So the first layer $S^{(1)}$ is determined by $S^{(1)}_1 = \{1, 2\}$, $S^{(1)}_2 = \{3, 4, 5\}$ and $S^{(1)}_3 = \{6, 7\}$.
The second Euclidian algorithm gives

$$\begin{aligned} 2 &= 0 \cdot 4 + 2 \\ 4 &= 2 \cdot 2 + 0 \end{aligned}$$

So the second layer $S^{(2)}$ is determined by the building blocks $S^{(2)}_1 = \emptyset$ and $S^{(2)}_2 = \{8, 9\}$.
The last Euclidian algorithm gives

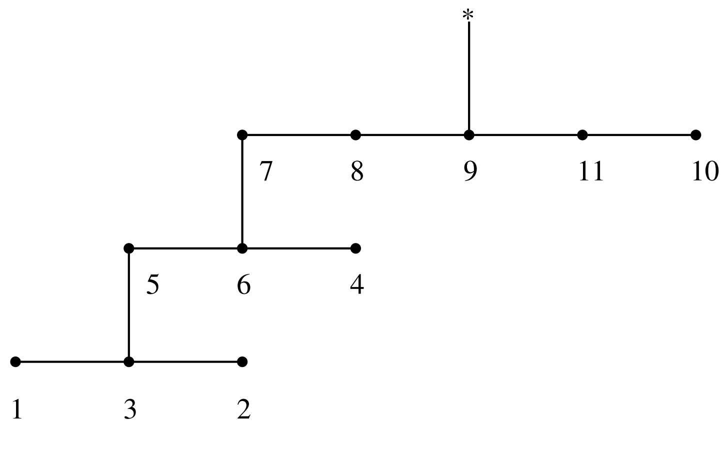
$$\begin{aligned} 11 &= 5 \cdot 2 + 1 \\ 2 &= 2 \cdot 1 + 0 \end{aligned}$$

The third layer is determined by $S^{(3)}_2 = \{10, 11, 12, 13, 14\}$ and $S^{(2)}_2 = \{15, 16\}$. As a result, the resolution graph is:



Now, we have understood how to obtain the resolution graph from the characteristic exponents. To obtain the characteristic exponents from the resolution graph, we simply start from the first layer of the resolution graph. We group the consecutive weights starting from the last and find the building blocks. Then starting from the last layer and the last building block, we write the Euclidian algorithms by using the $S_j^{(i)}$'s.

Example 3.1.33 *Let's find the characteristic exponents of the curve branch given by the resolution graph:*



There are three layers so there must be three Euclidian algorithms, four characteristic exponents. Starting from the first layer, we see that $S_1^{(1)} = \{1\}$, $S_2^{(1)} = \{2,3\}$. *Passing to the second layer, observe that the largest weight* 3 *of the first layer is not connected to the small-*

est weight 4 *of the second layer. This hints us that* $q_1^{(2)} = 0$ *so that* $S_1^{(2)} = \emptyset$. *Continuing on the second layer,* $S_2^{(2)} = \{4\}$ *and* $S_3^{(2)} = \{5, 6\}$. *Finally, for the last layer,* $S_2^{(3)} = \{7, 8, 9\}$ *and* $S_1^{(3)} = \{10, 11\}$. *Then, we construct the Euclidian algorithms going backwards. We know that the last greatest common divisor must be* 1 *in the last Euclidian algorithm. The cardinality* 2 *of the set* $S_1^{(3)}$ *says that the last step of the last Euclidian algorithm must be* $2 = 2 \cdot 1 + 0$ *(Quotient corresponds to the cardinality of* $S_1^{(3)}$*). This hints us that the remainder is* 1 *and the divisor is* 2 *in the previous division. As the cardinality of* $S_2^{(3)}$ *is* 3*, division must be* $7 = 3 \cdot 2 + 1$. *We pass to the second layer keeping in mind that the divisor must be* 2 *and the remainder must be* 0*, since we change layers. The cardinality of* $S_3^{(2)}$ *is* 2 *giving us the division* $4 = 2 \cdot 2 + 0$. *From this division, we keep* 4 *as the divisor,* 2 *as the remainder and we use the cardinality* 1 *as the quotient for the previous division, we get* $6 = 1 \cdot 4 + 2$. *As we have already mentioned,* $S_1^{(2)} = \emptyset$, $q_1^{(2)} = 0$ *and division must be* $4 = 0 \cdot 6 + 4$. *Continuing in this manner, it's not hard to see that the divisions corresponding to the first layer are:*

$$\begin{aligned} 18 &= 1 \cdot 12 + 6 \\ 12 &= 2 \cdot 6 + 0 \end{aligned}$$

To sum up, we read the first two characteristic exponents 12 *and* 18 *from the first line of the Euclidian algorithm corresponding to the first layer. Second euclidian algorithm*

$$\begin{aligned} 4 &= 0 \cdot 6 + 4 \\ 6 &= 1 \cdot 4 + 2 \\ 4 &= 2 \cdot 2 + 0 \end{aligned}$$

tells us that the third characteristic exponent is $18 + 4 = 22$. *Finally, last Euclidian algorithm*

$$\begin{aligned} 7 &= 3 \cdot 2 + 1 \\ 2 &= 2 \cdot 1 + 0 \end{aligned}$$

tells us that the fourth characteristic exponent is $22 + 7 = 29$. *That is, the characteristic exponents of the curve branch with the given resolution graph are* $(12, 18, 22, 29)$.

3.2 Reducible Plane Algebroid Curves

We first define the good resolution for reducible plane algebroid curves. Let C be a reducible plane algebroid curve, having the branches $C^{(1)}$, $C^{(2)}$,..., $C^{(r)}$, where $r \geq 2$. Consider the resolution process of C.

$$C_k \xrightarrow{\pi_k} C_{k-1} \xrightarrow{\pi_{k-1}} \cdots \xrightarrow{\pi_2} C_1 \xrightarrow{\pi_1} C$$

with strict transforms $C_i = (\pi_1 \circ \pi_2 \circ \cdots \circ \pi_i)^{-1}(C - \{0\})$ and exceptional divisors $E_i = (\pi_1 \circ \pi_2 \circ \cdots \circ \pi_i)^{-1}(0)$.

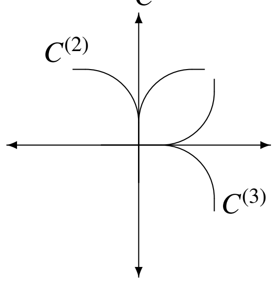
Definition 3.2.1 *[21, Definition 5.4.1] The resolution of C is said to be a good resolution, if all branches of C_k are smooth, they do not intersect with each other, they do intersect with only one component of E_k and they do intersect with this component transversally.*

Definition 3.2.2 *The contact number for the branches $C^{(i)}$ and $C^{(j)}$ is defined to be the smallest integer s for which $C_s^{(i)} \cap C_s^{(j)} = \emptyset$ where $C_s^{(i)}$ is the ith branch of C_s and $C_0^{(i)} := C^{(i)}$.*

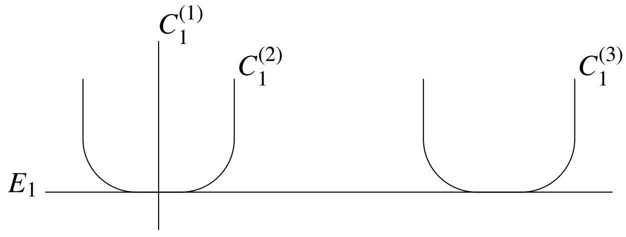
Given a reducible algebroid curve, we already know how to form the resolution graph for each of its components. Now, we understand how to do it for the reducible plane algebroid curve without losing the information on the branches.

Definition 3.2.3 *The weighted graph constructed by associating a " $\bullet$ " for each component of E_k and a " $*$ " for each component of C_k as vertices and connecting vertices if the corresponding components intersect is called the resolution graph of the reducible plane algebroid curve C.*

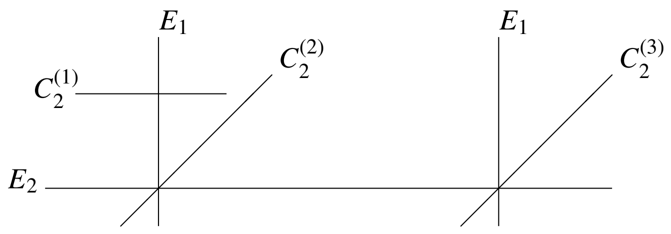
Example 3.2.4 *[8] Consider the reducible plane algebroid curve C defined by $x(x^2 - y^3)(x^3 - y^2) = 0$. C has three branches $C^{(1)}$, $C^{(2)}$ and $C^{(3)}$ defined by $x = 0$, $x^2 - y^3 = 0$ and $x^3 - y^2 = 0$ respectively.*



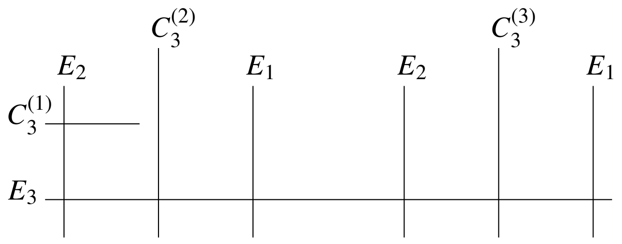
Blowing up C, we get the curve C_1:



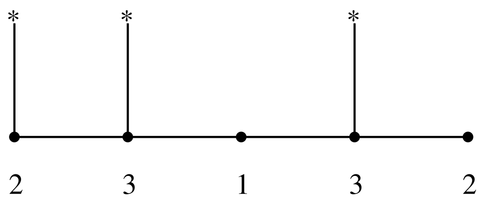
We see that $C_1^{(3)}$ and $C_1^{(1)}$ are separated in the first blowing up. Hence, contact of the branches $C^{(1)}$ and $C^{(3)}$ is one. Similarly, contact number of the branches $C^{(2)}$ and $C^{(3)}$ is one. On the other hand, observe that the branches $C^{(1)}$ and $C^{(2)}$ are still not separated. We blow up the curve again.



We see that the contact number of the first and second branches is two as they are separated in the second blowing up. But we need to continue blowing up as the intersection of the second and third branches are not transversal with the exceptional divisors.



Now, we have a good resolution and from this resolution, we conclude that the resolution graph of the reducible algebroid curve is:



Observe that the first star corresponds to the branch $C^{(1)}$, the second to $C^{(2)}$ and the third to $C^{(3)}$.

Theorem 3.2.5 *[21, Theorem 5.4.5] For a reducible curve C, the following data determine each other:*

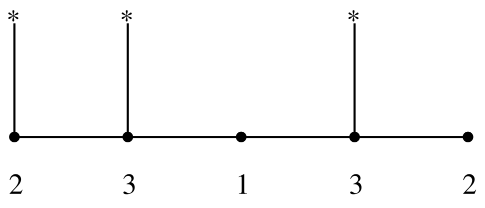
(i) resolution graph of C
(ii) resolution graph of the branches of C together with the contact numbers.

We explain how the procedure works while passing from the resolution graph of the curve to the resolution graph and the contact numbers of branches and vice versa.

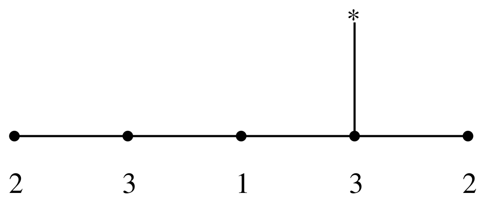
Definition 3.2.6 *A point on a resolution graph is called contractible, if it satisfies one of the following:*
(1) It is connected to a star and only to another point of lower weight.
(2) It is not connected to a star but connected to only points of lower weight.

Assume we have a reducible curve C and its resolution graph. Let $C^{(1)}$, $C^{(2)}$,...,$C^{(r)}$ be the branches of C. We want to find the resolution graph of $C^{(i)}$ and the contact number of $C^{(i)}$ with $C^{(j)}$ for $i \neq j$ and $1 \leq i, j \leq r$. For the resolution graph of C, we start by erasing all $*$'s, which do not belong to $C^{(i)}$. Then we start erasing points, if they are contractible. We stop when there are no contractible points.

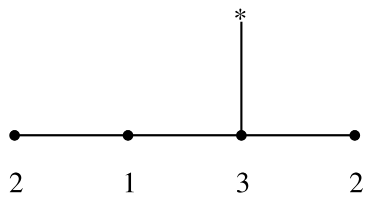
Example 3.2.7 *Let C be the curve given by equation $x(x^2-y^3)(x^3-y^2) = 0$ in Example 3.2.4. Let's try to find the resolution graph of the third branch $C^{(3)}$ given by $x^3 - y^2 = 0$. We know from Example 3.2.4 that the resolution graph of C is:*



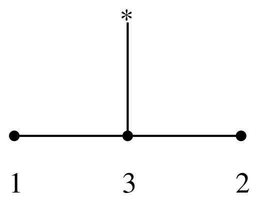
Erasing the first two stars corresponding to the first two branches, we get:



As 3 *on the left hand side is only connected to the points* 1 *and* 2 *of lower weights, we erase* 3 *and get:*



Now, the 2 *on the left hand side is only connected to the point* 1 *of lower weight. Hence, we erase it and get:*



Now, there is nothing else left to erase. Indeed, we can check from Example 3.1.29 that we have obtained the right resolution graph.

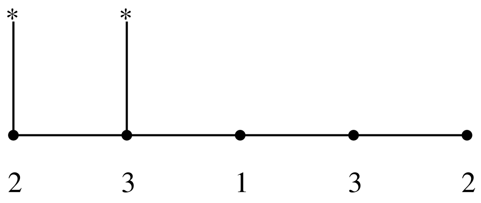
We can now explain the process of finding the contact numbers between the branches from the resolution graph of the reducible algebroid curve. To find the contact number of branches $C^{(i)}$ and $C^{(j)}$, we keep the stars corresponding to $C^{(i)}$ and $C^{(j)}$ and erase all the others. Then we remove the contractible points. We continue till there are no contractible points. Then we pick out the following points from the resolution graph we obtained after erasing stars and contractible points:

(1) The maximal number occurring as weight just once in the last resolution graph.

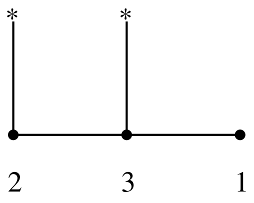
(2) The weights of the points connected to the stars in the last resolution graph.

Then the contact number is the smallest one among these three points.

Example 3.2.8 *Let us find the contact numbers between the branches of the curve C given by $x(x^2 - y^3)(x^3 - y^2) = 0$ from the resolution graph of C. We already know the resolution graph of C by Example 3.2.4. To find the contact number of $C^{(1)}$ and $C^{(2)}$, we start by erasing the star corresponding to $C^{(3)}$.*

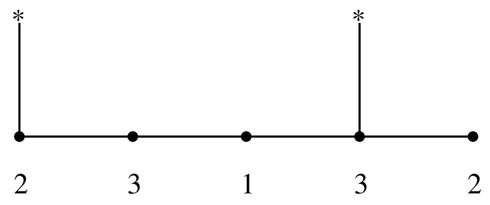


Now, 3 *is a contractible point. Erasing* 3*, it is not hard to see that* 2 *is also contractible, so we erase it too. Hence, we end up with:*

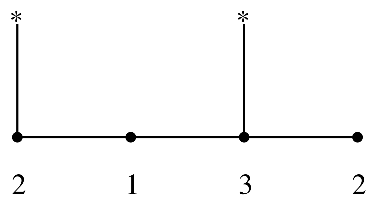


The numbers that we pick are $3,2,3$ *and our contact number is* 2*, which is the smallest of these numbers.*

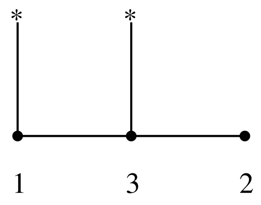
To find the contact number of the branches $C^{(1)}$ *and* $C^{(3)}$*, we erase the star corresponding to* $C^{(2)}$*.*



The 3 *on the left is contractible, so we erase it.*



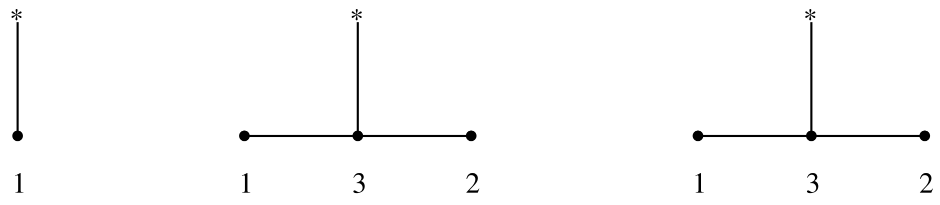
The 2 *on the left is contractible, as it is connected to a star and only to a point of lower weight. Hence, we erase it also.*



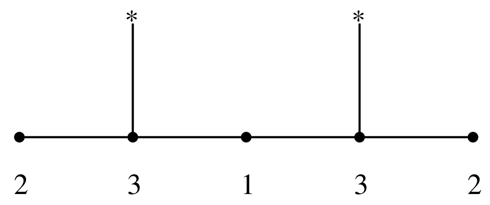
The numbers to be picked are $3,1,3$ *so the contact number of* $C^{(1)}$ *and* $C^{(3)}$ *is one. The contact number for the branches* $C^{(2)}$ *and* $C^{(3)}$ *can be found in a similar way.*

We now give another example, which shows how we can find the resolution graph of a reducible algebroid curve C, if we know the resolution graphs of the branches of C and their contact numbers.

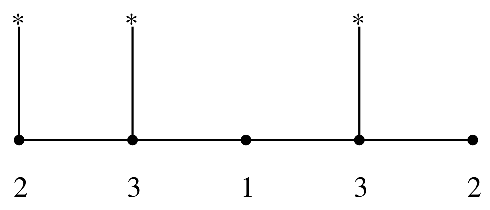
Example 3.2.9 *Let us again consider the curve* $x(x^2 - y^3)(x^3 - y^2) = 0$ *in Example 3.2.4. The resolution graphs of the branches are:*



Let us glue $C^{(2)}$ *with* $C^{(3)}$ *first. We know that the contact number is one for the chosen branches. In other words, they are separated in the first blowing up.*



Now, we glue the resolution graph of $C^{(1)}$ *to this resolution graph. For this, we look for the contact numbers of* $C^{(1)}$ *with the other branches. From Example 3.2.4, we know that the contact number of* $C^{(1)}$ *and* $C^{(2)}$ *is* 2*, and* $C^{(1)}$ *and* $C^{(3)}$ *is* 1*. We see that the maximum contact number is achieved with the branch* $C^{(2)}$*. Hence, we glue* $C^{(1)}$ *with* $C^{(2)}$*. As the contact number is* 2*, we get:*



For more complicated examples and the whole explanation of the process, see [21].

We can now state the next theorem summarizing the invariants of reducible plane algebroid curves.

Theorem 3.2.10 *[21, Theorem 5.4.7] For a reducible plane algebroid curve C, the following data determine each other:*

1. *Resolution graph of C*
2. *Resolution graphs of the branches of C and their contact numbers.*
3. *Multiplicity sequences of the branches of C and their contact numbers.*

The last theorem that we present in this section is the following one, which we use in our codes to compute the contact numbers of the branches.

Theorem 3.2.11 *[21, Theorem 5.4.8] Let C and C' be two irreducible plane algebroid curves with contact number k. If the multiplicity sequences of the branches are $(m_1, ..., m_r)$ and $(m_1^{'}, ..., m_s^{'})$ respectively, then taking $m_i = 1$ if $k > r$ and $m_i^{'} = 1$ if $k > s$, the intersection multiplicity of C and $C^{'}$ is:*

$$\sum_{i=1}^{k} m_i.m_i^{'}$$

Example 3.2.12 *Let us find the intersection multiplicity of the irreducible algebroid curves defined by $x = 0$ and $x^2 - y^3 = 0$. We know from Example 3.2.4 that the contact number of the branches is 2 and the multiplicity sequences are (1) and $(2, 1, 1)$ respectively. Then, from the previous theorem the intersection multiplicity of the branches is: $2 \cdot 1 + 1 \cdot 1 = 3$.*

CHAPTER 4

HAMBURGER NOETHER EXPANSIONS AND MATRICES

In general, we are interested in curve branches over an algebraically closed field of zero characteristic, but it has to be noted that some of the procedures we use are not valid in the positive characteristic case like the Puiseux expansions. However, Hamburger Noether expansions, which can be used instead of Puiseux expansions work in any characteristic. Moreover, we can actually define Hamburger Noether expansions not only for the plane curve branches. but also for the higher dimensional curve branches. We associate a matrix to a Hamburger Noether expansion and it is also possible to the read the invariants of the curve branch from that matrix. Hamburger Noether expansions can also be used to find the parametrization of a curve branch, if it is given in closed form. In this chapter, we explain how we get Hamburger Noether expansions and matrices from the parametrization of the curve branch and how we determine the invariants from the Hamburger Noether matrices. To understand the process better, we start with the plane case.

4.1 Hamburger Noether Expansions Of Plane Curve Branches

Let C be a plane curve branch given in parametric form $x(t)$ and $y(t)$. Without loss of generality, suppose that the order of $x(t)$ is smaller than the order of $y(t)$. We construct the Hamburger Noether expansion of C by doing successive divisions. Normally, Hamburger Noether expansions can be found for any parametrization regardless of whether they are primitive or not. However, for our future purposes, we explain the procedure only for the primitive parameterizations [10]. Let $(x(t), y(t))$ be a primitive parametrization. Divide y by x and let y_1 be the power series obtained from this division. If the constant term of y_1 is a_{01}, then set $y_1 =: y_1 - a_{01}$. If $ord(x) \leq ord(y_1)$, apply the same procedure by taking y_1 instead of y till you get a power series of order less than the order of x. At the end, we get an expression for y as $y = a_{01}x + a_{02}x^2 + ... + a_{0r}x^r + x^r z_1$ with $ord(z_1) < ord(x)$. If $ord(x) > ord(y_1)$, we have a similar expression for y as $y = a_{01}x + xz_1$ with $ord(z_1) < ord(x)$ ($z_1 = y_1$ in this case). Now in either case, if $ord(z_1) = 1$, we write x in terms of z_1 by dividing x by z_1 and the algorithm stops. Otherwise, we continue in the same manner by taking x as the dividend and z_1 as the divisor and we get an expression for x as $x = a_{11}z_1 + a_{12}{z_1}^2 + ... + a_{1r_1}{z_1}^{r_1} + {z_1}^{r_1}z_2$ with $ord(z_2) < ord(z_1)$. Again, if $ord(z_2) = 1$, we write z_1 in terms of z_2 by dividing z_1 by z_2 and the algorithm stops. Otherwise, we continue with z_1 and z_2. But as we have started with a primitive parametrization, after finitely many steps, we get $ord(z_s) = 1$ for some s. Expressing z_{s-1} in terms of z_s, and the algorithm stops ([10, Corollary 2.2.6]). Hence, we obtain a set of expressions for the parametrization system $\{x(t), y(t)\}$:

$$\begin{array}{rcl} y & = & a_{01}x + a_{02}x^2 + ... + a_{0r}x^r + x^r z_1 \\ x & = & a_{11}z_1 + a_{12}{z_1}^2 + ... + a_{1r_1}{z_1}^{r_1} + {z_1}^{r_1}z_2 \\ ... & .. & .... \\ z_{s-1} & = & a_{s1}z_s + a_{s2}z_s^2 + ... \end{array} \tag{4.1}$$

where the a_{ij}'s are from the coefficient field, z_j's are power series with descending orders, that is $ord(y) > ord(x) > ord(z_1) > ... > ord(z_s) = 1$.

Definition 4.1.1 *The expressions we have for $\{x(t), y(t)\}$ in 4.1 are called the Hamburger-Noether expansion for the curve branch defined by $\{x(t), y(t)\}$. For the sake of simplicity, we simply denote the Hamburger-Noether expansion given in 4.1 by*

$$z_{j-1} = \sum_{i=1}^{r_j} a_{ji}z_j^i + z_j^{r_j}z_{j+1}$$

with $z_{-1} = y$, $z_0 = x$ and $0 \leq j \leq r$.

Example 4.1.2 *Let C be the curve given by parametrization $x(t) = t^2$ and $y(t) = t^3$. Let us find the Hamburger-Noether expansion of C. We have $z_0 = x$ and $z_{-1} = y$ to start with.*

$$y_1 = \frac{y}{x} = \frac{t^3}{t^2} = t$$

The constant term of y_1 is 0, so $a_{01} = 0$ and $ord(x) = 2 > 1 = ord(y_1)$. Hence, we have the expression

$$z_{-1} = z_0 z_1$$

for z_{-1}, where $z_1 = t$. We write z_0 in terms of z_1 and the algorithm stops, as we have reached an order one element, z_1. As $z_0 = {z_1}^2$, the Hamburger-Noether expansion for the parametrization system $\{x(t), y(t)\}$ is:

$$z_{-1} = z_0 z_1$$

$$z_0 = {z_1}^2$$

We can now give more complicated examples.

Example 4.1.3 *Let C be the curve with the parametrization $x(t) = t^4 + t^7$ and $y(t) = t^{10} + 2t^{13} + t^{16}$. Let's find the Hamburger-Noether expansion of C. Dividing y by x, we get*

$$y_1 = \frac{t^{10} + 2t^{13} + t^{16}}{t^4 + t^7} = t^6 + t^9$$

$a_{01} = 0$ and as $ord(y_1) = 6 > 4 = ord(x)$, we continue dividing y_1 by x and get

$$y_2 = \frac{y_1}{x} = \frac{t^6 + t^9}{t^4 + t^7} = t^2$$

$a_{02} = 0$ and $ord(y_2) = 2 < 4 = ord(x)$. Hence, having the expression

$$z_{-1} = z_0^2 z_1$$

where $z_{-1} = y$, $z_0 = x$ and $z_1 = t^2$, we continue dividing z_0 by z_1.

$$y_1 = \frac{z_0}{z_1} = \frac{t^4 + t^7}{t^2} = t^2 + t^5$$

$a_{11} = 0$ and $ord(y_1) = 2 = ord(z_1)$, we continue dividing y_1 by z_1.

$$y_2 = \frac{y_1}{z_1} = \frac{t^2 + t^5}{t^2} = 1 + t^3$$

Hence, $a_{12} = 1$ and $y_2 := y_2 - a_{12} = t^3$. Thus, $ord(y_2) = 3 > ord(z_1)$, so we continue dividing y_2 by z_1.

$$y_3 = \frac{y_2}{z_1} = \frac{t^3}{t^2} = t$$

$a_{13} = 0$ and as $ord(y_3) = 1 < 2 = ord(z_1)$, we have

$$z_0 = {z_1}^2 + {z_1}^3 z_2$$

where $z_2 = t$. Now, we express z_1 in terms of z_2 and the algorithm stops, as $ord(z_2) = 1$. As $z_1 = {z_2}^2$, Hamburger-Noether expansion for the curve branch is:

$$\begin{aligned} z_{-1} &= z_0^2 z_1 \\ z_0 &= {z_1}^2 + {z_1}^3 z_2 \\ z_1 &= {z_2}^2 \end{aligned}$$

Example 4.1.4 *Let C be the plane curve branch with parametrization $x(t) = t^4 + t^7 + t^{10}$ and $y(t) = t^{10} + 2t^{13} + 3t^{16} + 2t^{19} + t^{22}$. Let us find the Hamburger-Noether expansion of C. Dividing y by x, we get*

$$y_1 = \frac{t^{10} + 2t^{13} + 3t^{16} + 2t^{19} + t^{22}}{t^4 + t^7 + t^{10}} = t^6 + t^9 + t^{12}$$

$a_{01} = 0$ *and as* $ord(y_1) = 6 > 4 = ord(x)$*, we continue dividing* y_1 *by x and get*

$$y_2 = \frac{y_1}{x} = \frac{t^6 + t^9 + t^{12}}{t^4 + t^7 + t^{10}} = t^2$$

$a_{02} = 0$ *and* $ord(y_2) = 2 < 4 = ord(x)$*. Hence, having the expression*

$$z_{-1} = z_0^2 z_1$$

where $z_{-1} = y$, $z_0 = x$ *and* $z_1 = t^2$*, we continue dividing* z_0 *by* z_1*.*

$$y_1 = \frac{z_0}{z_1} = \frac{t^4 + t^7 + t^{10}}{t^2} = t^2 + t^5 + t^8$$

$a_{11} = 0$ *and* $ord(y_1) = 2 = ord(z_1)$*, we continue dividing* y_1 *by* z_1*.*

$$y_2 = \frac{y_1}{z_1} = \frac{t^2 + t^5 + t^8}{t^2} = 1 + t^3 + t^6$$

Hence, $a_{12} = 1$ *and* $y_2 := y_2 - a_{12} = t^3 + t^6$*.* $ord(y_2) = 3 > ord(z_1)$*, so we continue dividing* y_2 *by* z_1*.*

$$y_3 = \frac{y_2}{z_1} = \frac{t^3 + t^6}{t^2} = t + t^4$$

$a_{13} = 0$ *and as* $ord(y_3) = 1 < 2 = ord(z_1)$*, we have*

$$z_0 = {z_1}^2 + {z_1}^3 z_2$$

for z_0 *where* $z_2 = t + t^4$*. Now, we write* z_1 *in terms of* z_2 *and the algorithm stops* $ord(z_2) = 1$*. As*

$$z_1 = {z_2}^2 + ...$$

Hamburger-Noether expansion for the curve branch is:

$$\begin{aligned} z_{-1} = &\ z_0^2 z_1 \\ z_0 = &\ {z_1}^2 + {z_1}^3 z_2 \\ z_1 = &\ {z_2}^2 + ... \end{aligned}$$

The following algorithm for finding the Hamburger-Noether expansion of the curve branch with parametrization $\{x(t), y(t)\}$ can be found in [32]:

Algorithm 1 PrimParam-HNE

```
Input: $(x(t), y(t))$ is a primitive parametrization with $ord(x(t)) < ord(y(t))$
Output: Hamburger Noether expansion $z_{j-1} = \sum_{i=1}^{h_j} a_{ji} z_j^i + z_j^{h_j} z_{j+1}$, $j = 1, ..., r$
 1: $z_{-1} \leftarrow y$,  $z_0 \leftarrow x$,  $s \leftarrow 0$
 2: $i \leftarrow 1$,  $y_0 \leftarrow z_{s-1}$
 3: Calculate $a_{si} \in k$ and $y_i \in k[[t]]$ with $\frac{y_{i-1}}{z_s} = a_{si} + y_i$ and $ord(y_i) > 0$
 4: if $a_{s_i} \neq 0$ and $ord(z_s) = 1$ then
 5:     STOP
 6: end if
 7: if $y_{i-1} = 0$ then
 8:     $a_{sj} \leftarrow 0$ for all $j > i$ and STOP
 9: end if
10: if $ord(y_i) \geq ord(z_s)$ then
11:     $i \leftarrow i + 1$ and go to 3
12: else
13:     $i \leftarrow h_s$
14: end if
15: $s \leftarrow s + 1$,  $z_s \leftarrow y_{h_{s-1}}$ and go back to 2.
```

Having understood the algorithm to find the Hamburger-Noether expansions, we can talk about the importance of these expansions. It is actually possible to read the invariants of the curve branch from its Hamburger-Noether expansion. Our aim now is to understand this process.

Proposition 4.1.5 *[10, Proposition 2.2.9] Let C be a curve branch with parametrization $\{x(t), y(t)\}$ and Hamburger-Noether expansion*

$$z_{j-1} = \sum_{i=1}^{r_j} a_{ji} z_j^i + z_j^{r_j} z_{j+1}$$

Let C_1 be the blow-up of C with parametrization $\{x, y_1\}$ where $y_1 = \frac{y}{x} - a_{01}$. Then the Hamburger-Noether expansion for C_1 is:

(1) If $r_0 > 1$,

$$y_1 = a_{02}x + ... + a_{0r_0}x^{r_0-1} + x^{r_0-1}z_1$$

$$z_{j-1} = \sum_{i=1}^{r_j} a_{ji}z_j^i + z_j^{r_j}z_{j+1}, \quad 1 \leq j \leq s$$

(2) If $r_0 = 1$,

$$z_{j-1} = \sum_{i=1}^{r_j} a_{ji}z_j^i + z_j^{r_j}z_{j+1}, \quad 1 \leq j \leq s$$

Example 4.1.6 *Let C be the curve branch given with $\{t^4 + t^7, t^{10} + 2t^{13} + t^{16}\}$. We know that the Hamburger-Noether expansion of C is*

$$\begin{aligned} z_{-1} &= z_0^2 z_1 \\ z_0 &= {z_1}^2 + {z_1}^3 z_2 \\ z_1 &= {z_2}^2 \end{aligned}$$

from Example 4.1.3. We also know from section 2.2.2 that blowing up of C is given by the parametrization $\{t^4 + t^7, t^6 + t^9\}$. According to Proposition 4.1.5, as $r_0 = 2 > 1$, Hamburger-Noether expansion of the blown up curve is

$$\begin{aligned} z_{-1} &= z_0 z_1 \\ z_0 &= {z_1}^2 + {z_1}^3 z_2 \\ z_1 &= {z_2}^2 \end{aligned}$$

Indeed, If we find the Hamburger-Noether expansion of C_1 from the algorithm: $z_{-1} = t^6 + t^9$, $z_0 = t^4 + t^7$, $y_1 = \frac{z_{-1}}{z_0} = t^2$, $ord(y_1) < ord(z_0)$ so $z_1 = t^2$ and we get:

$$z_{-1} = z_0 z_1$$

We divide z_0 by z_1, but as z_0 and z_1 are the same with the ones in Example 4.1.3, the rest of the algorithm should be the same and we get the expressions

$$\begin{aligned} z_0 &= {z_1}^2 + {z_1}^3 z_2 \\ z_1 &= {z_2}^2 \end{aligned}$$

Hence, the Hamburger-Noether expansion for C_1 is:

$$\begin{aligned} z_{-1} &= z_0 z_1 \\ z_0 &= {z_1}^2 + {z_1}^3 z_2 \\ z_1 &= {z_2}^2 \end{aligned}$$

as expected.

Proposition 4.1.7 *[10, Proposition 2.2.10] Let C be a curve branch with parametrization $\{x(t), y(t)\}$ and Hamburger-Noether expansion*

$$z_{j-1} = \sum_{i=1}^{r_j} a_{ji} z_j^i + z_j^{r_j} z_{j+1}$$

n_i being the order of the power series z_i for $0 \leq i \leq s$. Then the multiplicity sequence of C is

$$\underbrace{n_0, ..., n_0}_{r_0 \text{ times}}, \underbrace{n_1, ..., n_1}_{r_1 \text{ times}}, ..., \underbrace{n_s, ..., n_s}_{r_s \text{ times}}$$

Example 4.1.8 *For the curve branch C defined by $\{t^4 + t^7, t^{10} + 2t^{13} + t^{16}\}$ in Example 4.1.3,*

$$\begin{aligned} n_0 &= ord(z_0) = 4 \qquad h_0 = 2 \\ n_1 &= ord(z_1) = 2 \qquad h_1 = 3 \\ n_2 &= ord(z_2) = 1 \qquad h_2 = 2 \end{aligned}$$

Hence, from Proposition 4.1.7, the multiplicity sequence of C is:

$$4, 4, 2, 2, 2, 1, 1$$

Indeed, we know from section 2.2.2 that the blowing up of C gives C_1 with parametrization $(t^4 + t^7, t^6 + t^9)$. The multiplicity of the singular point of the branch C_1 is 4 and the blowing up of C_1 gives the curve C_2 with parametrization $(t^2, t^4 + t^7)$. The multiplicity is 2 and the blowing up of C_2 is C_3, defined by $(t^2, t^2 + t^5)$ with singular point of multiplicity 2. C_4, which is the blowing up of C_3, has the parametrization (t^2, t^3) and the multiplicity is 2. Finally, C_4 and C_5 have parameterizations (t, t^2), (t, t) and singular points have multiplicities 1, 1 respectively. Thus, C has multiplicity sequence 4, 4, 2, 2, 2, 1, 1 indeed.

Corollary 4.1.9 *[10, Corollary 2.2.11] Let C and C^* be two curve branches with Hamburger-Noether expansions*

$$z_{j-1} = \sum_{i=1}^{r_j} a_{ji} z_j^i + z_j^{r_j} z_{j+1}, \quad 0 \leq j \leq s$$

$$z^*_{j-1} = \sum_{i=1}^{r^*_j} a^*_{ji} z_j^{*i} + z_j^{*r^*_j} z^*_{j+1}, \quad 0 \leq j \leq s^*$$

respectively. Then C and C^* are equivalent (equiresoluble) iff $s = s^*$, $r_j = r^*_j$ and $n_j = n^*_j$ for $0 \leq j \leq s$.

Example 4.1.10 *The curve branches with parameterizations $(t^4 + t^7, t^{10} + 2t^{13} + t^{16})$ and $(t^4 + t^7 + t^{10}, t^{10} + 2t^{13} + 3t^{16} + 2t^{19} + t^{22})$ respectively are equivalent, as it can be checked from their Hamburger-Noether expansions obtained in Examples 4.1.3 and 4.1.4.*

4.1.1 Intersection Multiplicity, Contact numbers and Hamburger-Noether expansions

Definition 4.1.11 *Let C be a curve branch defined by $(x(t), y(t))$ and let D be another curve, not necessarily irreducible, defined by $g(x, y) \in k[[x, y]]$. The intersection multiplicity of C and D is the order of the power series $g(x(t), y(t))$.*

Proposition 4.1.12 *[10] Let C and C^* be two curve branches with Hamburger-Noether expansions*

$$z_{j-1} = \sum_{i=1}^{r_j} a_{ji} z_j^i + z_j^{r_j} z_{j+1}, \quad 0 \leq j \leq s$$

$$z^*_{j-1} = \sum_{i=1}^{r^*_j} a^*_{ji} z^{*i}_j + z^{*r^*_j}_j z^*_{j+1}, \quad 0 \leq j \leq s^*$$

respectively. Let k be the greatest integer for which $r_0 = r^*_0$, $r_1 = r^*_1$, ..., $r_{k-1} = r^*_{k-1}$ and $a_{jl} = a^*_{jl}$ for $j \leq k-1$ and $l \leq r_j$. Let i be the least index such that $a_{ki} \neq a^*_{ki}$ ($i \leq r_k + 1$, $i \leq r^*_k + 1$). The integer

$$N = r_0 + r_1 + ... + r_{k-1} + i$$

is equal to the contact number of C and C^*.

Example 4.1.13 *Let C and C^* be two curve branches with Hamburger-Noether expansions*

$$\begin{aligned} z_{-1} &= {z_0}^2 z_1 \\ z_0 &= {z_1}^2 + {z_1}^2 z_2 \\ z_1 &= {z_2}^3 \end{aligned}$$

and

$$\begin{aligned} z^*_{-1} &= {z^*_0}^2 z^*_1 \\ z^*_0 &= {z^*_1}^2 z^*_2 \\ z^*_1 &= {z^*_2}^3 \end{aligned}$$

respectively. Then $k = 1$ and $i = 2$ with the notation of the previous proposition. So the contact number of the branches C and C^* is

$$r_0 + i = 2 + 2 = 4$$

Remark 4.1.14 *Knowing the contact numbers and multiplicity sequences from the Hamburger-Noether expansions, the intersection multiplicity of the curve branches can be found by Theorem 3.2.11.*

Example 4.1.15 *For the curve branches in Example 4.1.13, it is not hard to show that the multiplicity sequences are* $(6,6,3,3,1,1,1)$ *and* $(7,7,3,3,1,1,1)$ *respectively. Hence, the intersection multiplicity of the curve branches is* $6\cdot 7+6\cdot 7+3\cdot 3+3\cdot 3 = 102$*. Indeed, it is not hard to show that* $(t^6+t^7, t^{15}+2t^{16}+t^{17})$ *is the parametrization of C and* $x^{17}-y^7=0$ *is the defining equation of* C^**. We can also check from Definition 4.1.11 that the intersection multiplicity is* 102 *as the order of the polynomial* $(t^6+t^7)^{17}-(t^{15}+2t^{16}+t^{17})^7$ *is* 102.

4.1.2 Characteristic Exponents and Hamburger-Noether expansions

Let C be a plane algebroid curve with Hamburger-Noether expansion

$$z_{j-1} = \sum_{i=1}^{r_j} a_{ji} z_j^i + z_j^{r_j} z_{j+1}, \quad 0 \le j \le s$$

Setting $n_j = ord(z_j)$, we know that the multiplicity sequence of C is $n_0...n_0, n_1...n_1, ..., n_s...n_s$. Looking at n_j's, we pick out the different indices $m_1,...,m_q$ if n_{m_i} divides n_{m_i-1}. We assume $m_0 = 0$.

Example 4.1.16 *If the multiplicity sequence is* $4,4,3,1,1,1$, $n_0=4$, $n_1=3$ *and* $n_2=1$*. Then* $m_0=0$ *and* $m_1=2$ *as* n_2 *divides* n_1.

Proposition 4.1.17 *q is exactly the genus of C.*

Definition 4.1.18 *Let C be a plane curve branch with Hamburger-Noether expansion*

$$z_{j-1} = \sum_{i=1}^{r_j} a_{ji} z_j^i + z_j^{r_j} z_{j+1}, \quad 0 \le j \le s$$

Then the characteristic exponents $\beta_0, \beta_1, ..., \beta_q$ *are:*

$$\begin{aligned} \beta_0 &= n_0 \\ \beta_{v+1} &= \sum_{j=0}^{m_v} r_j n_j + n_{m_v} + n_{m_v+1} - n_0, \quad 0 \le v \le q-1 \end{aligned}$$

Example 4.1.19 *For the curve branches in Example 4.1.13, the characteristic exponents are* $(5,15,16)$ *and* $(7,17)$ *respectively. For the curve branch C, the multiplicity sequence is (6,6,3,3,1,1,1). So,* $m_0=0$ *by definition and* $m_1=1$ *as* n_1 *divides* n_0 *and* $m_2=2$ *as* n_2

divides n_1. *Hence,*

$$\begin{aligned}\beta_0 &= n_0 && = 6\\ \beta_1 &= r_0n_0 + n_0 + n_1 - n_0 && = 2\cdot 6 + 3 = 15\\ \beta_2 &= \sum_{j=0}^{m_1} r_jn_j + n_1 + n_2 - n_0 && = 12 + 6 + 3 + 1 - 6 = 16\end{aligned}$$

For the curve branch C^**, the multiplicity sequence is* $(7,7,3,3,1,1,1)$. *Hence* $m_0 = 0$*,* $m_1 = 2$ *as* n_2 *divides* n_1. *Hence,*

$$\begin{aligned}\beta_0 &= n_0 && = 7\\ \beta_1 &= r_0n_0 + n_0 + n_1 - n_0 && = 2\cdot 7 + 3 = 17\end{aligned}$$

4.2 Hamburger-Noether Expansions For General Curve Branches

In this section, we generalize these expansions to the curves in higher dimensions. Let C be a curve branch given in primitive parametric form $(x_1(t), x_2(t), ..., x_n(t))$ such that $0 \leq ord(x_i)$ for $2 \leq i \leq n$. Choose the index i_0 with $ord(x_{i_0}(t)) = min\{x_1(t), ..., x_n(t)\}$. Then,

(i) If $ord(x_{i_0}) = 1$, we write

$$x_i(t) = \sum_{p\geq 1} a_{i_p} x_{i_0}^p(t)$$

for all $i \neq i_0$ and keep the information (i_0, x_{i_0}).

(ii) Else, we divide all x_i but x_{i_0} by x_{i_0}, subtract the constant terms and get the series

$$\begin{aligned} y_{i_0}(t) &= x_{i_0}(t) \\ y_i(t) &= \tfrac{x_i(t)}{x_{i_0}(t)} - a_{i1} \quad for\ i \neq i_0 \end{aligned}$$

We keep the information i_0, a_{i1} and start the same process with the parametrization $(y_1(t), ..., y_n(t))$. After applying (ii) a couple of times, we reach (i) eventually. Then the information we have kept throughout the process is called the Hamburger-Noether expansion for the branch C.

Algorithm 2 HNE for Twisted curves

Input: $(x_1(t), ..., x_n(t))$ is a primitive parametrization with $ord(x_1) \leq ord(x_i(t))$

Output: Hamburger Noether expansion $\overline{Z_{j-1}} = \sum_{i=1}^{h_j} A_{ji}{z_j}^i + {z_j}^{h_j} Z_{j+1}, \quad j = 0, ..., r$

Step 1. We set $\overline{Z_{-1}} = Y = \begin{bmatrix} x_2 \\ . \\ . \\ x_n \end{bmatrix}$ and $z_0 = x_1$

Step 2. We divide all the series x_i with x_1 and subtract the constant terms for $1 < i \leq n$ and continue dividing until one of the series have order less than the order of x_1.

Step 3. This step is divided into two cases:

• If one of the series obtained has order 1, then we have $(N - 1) \times 1$ matrices A_{0i} for $1 \leq i$ with entries from the ground field, and an expansion for Y as

$$\overline{Z_{-1}} = Y = A_{01}x_1 + A_{02}{x_1}^2 + ...$$

and the algorithm stops.

• If none of the series obtained has order 1, then we have $(N-1)\times 1$ matrices A_{0i} for $1 \leq i \leq r_0$ with entries from the ground field, an $(N - 1) \times 1$ matrix with entries from the power series ring and an expansion for Y as

$$\overline{Z_{-1}} = Y = A_{01}x_1 + A_{02}{x_1}^2 + ... + A_{0h}{x_1}^{r_0} + Z_1{x_1}^{r_0}$$

with at least one of the entries of Z_1 has order less than the order of x_1. In this case, we pick out an entry of Z_1 with smallest order and call it z_1. Then we continue doing the same process starting from step one with the parametrization system $(z_1, x, z_{13}, ..., z_{1n})$, where $z_{13}, ..., z_{1n}$ are the entries of Z_1 other than z_1.

We set $\overline{Z_0} = \begin{bmatrix} x \\ z_{13} \\ . \\ z_{1n} \end{bmatrix}$. We divide the entries of $\overline{Z_0}$ by z_1 and get an expansion for $\overline{Z_0}$. After finitely many steps, we obtain an order one power series and the algorithm stops.

Eventually, we obtain an expression of type:

$$\overline{Z_{j-1}} = \sum_{i=1}^{h_j} A_{ji} {z_j}^i + {z_j}^{h_j} Z_{j+1}, \quad j = 0, ..., r$$

Example 4.2.1 *Let C be the space curve given with the parametrization $x_1(t) = t^4+t^6, x_2(t) = t^6 + t^8, x_3(t) = t^9 + t^{11}$. We can now determine the Hamburger-Noether expansion of C. We start by setting $\overline{Z_{-1}} = \begin{bmatrix} x_2 \\ x_3 \end{bmatrix}$ and $z_0 = x_1$, and continue with dividing the entries of $\overline{Z_{-1}}$ by z_0, which gives $\frac{x_2}{x_1} = \frac{t^6+t^8}{t^4+t^6} = t^2$ and $\frac{x_3}{x_1} = \frac{t^9+t^{11}}{t^4+t^6} = t^5$. As we have at least one series with order less than the order of x_1, we continue with the parametrization $(t^2, t^4 + t^6, t^5)$, but before that, we have an expansion for $\overline{Z_{-1}}$ as:*

$$\overline{Z_{-1}} = z_0 Z_1$$

$\overline{Z_{-1}}$ and z_0 are as explained before and $Z_1 = \begin{bmatrix} t^2 \\ t^5 \end{bmatrix}$ here. Hence to continue with the parametrization system $(t^2, t^4 + t^6, t^5)$, we take $\overline{Z_0} = \begin{bmatrix} t^4 + t^6 \\ t^5 \end{bmatrix}$, $z_1 = t^2$ and start dividing the entries of $\overline{Z_0}$ with z_1 to obtain $\frac{t^4+t^6}{t^2} = t^2 + t^4$ and $\frac{t^5}{t^2} = t^3$. None of the series has order less than the order of $z_1 = t^2$, so we continue dividing the series with z_1. We obtain $\frac{t^2+t^4}{t^2} = 1 + t^2$ (we subtract the constant term 1 and get the series t^2) and $\frac{t^3}{t^2} = t$. We now have a series with order less than the order of $z_1 = t^2$, so we can write $\overline{Z_0}$ as,

$$\overline{Z_0} = A_{12} {z_1}^2 + Z_2 {z_1}^2$$

where $A_{12} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ and $Z_2 = \begin{bmatrix} t^2 \\ t \end{bmatrix}$. As we have reached an order one series, we have one more step and the algorithm stops. We continue with the parametrization system (t, t^2, t^2). We have $\begin{bmatrix} t^2 \\ t^2 \end{bmatrix}$ as $\overline{Z_1}$ and t as z_2. So, we can write

$$\overline{Z_1} = A_{22} {z_2}^2$$

where $A_{22} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$. Hence, the Hamburger-Noether expansion for C is:

$$\begin{aligned} \overline{Z_{-1}} &= z_0 Z_1 \\ \overline{Z_0} &= A_{12} {z_1}^2 + Z_2 {z_1}^2 \\ \overline{Z_1} &= A_{22} {z_2}^2 \end{aligned}$$

Remark 4.2.2 *Note that the Hamburger-Noether expansion for a curve branch is not unique. It depends on the smallest ordered element that we pick from the parametrization. But when we fix the smallest ordered elements that we choose in each step, we have a unique Hamburger-Noether expansion.*

Proposition 4.2.3 *[10, Remark 2.4.9] Let* $\overline{Z_{j-1}} = \sum_{i=1}^{h_j} A_{ji}{z_j}^i + {z_j}^{h_j} Z_{j+1}, \quad j = 0, ..., r$ *be a Hamburger-Noether expansion for the curve branch C.* n_j *being the order of the element* z_j*, the different multiplicities that we have in the blowing up process of C are the natural numbers* n_j*'s.*

Corollary 4.2.4 *[10, Remark 2.4.9] Let* $\overline{Z_{j-1}} = \sum_{i=1}^{h_j} A_{ji}{z_j}^i + {z_j}^{h_j} Z_{j+1}, \quad j = 0, ..., r$ *be a Hamburger-Noether expansion for the curve branch C.* n_j *being the order of the element* z_j*, the multiplicity sequence of C is:*

$$\underbrace{n_0, ..., n_0}_{h_0 \ times}, \underbrace{n_1, ..., n_1}_{h_1 \ times}, ..., \underbrace{n_s, ..., n_s}_{h_r \ times}$$

Example 4.2.5 *Let C be the curve branch given with parametrization* $x_1(t) = t^4 + t^6, x_2(t) = t^6 + t^8, x_3(t) = t^9 + t^{11}$ *in Example 4.2.1. Then the multiplicity sequence of C is: (4,2,2,1,1).*

Corollary 4.2.6 *[10, Remark 2.4.9] Let C and* C^* *be two curve branches with Hamburger-Noether expansions*

$$\begin{aligned} \overline{Z_{j-1}} &= \sum_{i=1}^{h_j} A_{ji}{z_j}^i + {z_j}^{h_j} Z_{j+1}, \quad j = 0, ..., r \\ \overline{Z^*_{j-1}} &= \sum_{i=1}^{h^*_j} A^*_{ji}{z^*_j}^i + {z^*_j}^{h^*_j} Z^*_{j+1}, \quad j = 0, ..., r^* \end{aligned}$$

respectively. Then, C and C^* *are equiresoluble if and only if* $h_j = h^*_j$*, and* $n_j = n^*_j$*.*

4.2.1 Hamburger-Noether Matrices

Let C be a curve branch given with parametrization $(x_1(t), ..., x_n(t))$ and with the Hamburger-Noether expansion

$$\overline{Z_{j-1}} = \sum_{i=1}^{h_j} A_{ji}{z_j}^i + {z_j}^{h_j} Z_{j+1}, \quad j = 0, ..., r$$

The data contained in the Hamburger-Noether expansion of C can be given with a matrix, which is called Hamburger-Noether Matrix. The matrix has n rows, $r+1$ boxes and the jth box represents the expansion $\overline{Z_{j-1}}$. Hence, the jth box has h_j columns. Each box has a marked row starting with a 1, exactly to mark the divider. Finally, the entries of the boxes represent the constants that we have been subtracted in the expansion. To understand the process better, we give an example.

Example 4.2.7 *For the curve branch C with parametrization $x_1(t) = t^4 + t^6, x_2(t) = t^6 + t^8, x_3(t) = t^9 + t^{11}$ in Example 4.2.1, we know that the Hamburger-Noether expansion is:*

$$\begin{aligned} \overline{Z_{-1}} &= z_0 Z_1 \\ \overline{Z_0} &= A_{12} {z_1}^2 + Z_2 {z_1}^2 \\ \overline{Z_1} &= A_{22} {z_2}^2 \end{aligned}$$

As we are dealing with a space curve, our Hamburger-Noether matrix will have 3 *rows and as the Hamburger-Noether expansion consists only of $\overline{Z_{-1}}$, $\overline{Z_0}$, $\overline{Z_1}$, Hamburger-Noether matrix has* 3 *boxes. It looks like roughly:*

$$\left[\begin{array}{c|c|c} & & \\ & & \\ & & \end{array}\right]$$

Before starting to fill out the boxes, recall that the parameterizations we have used throughout the process are:

$(t^4 + t^6, t^6 + t^8, t^9 + t^{11})$

$(t^4 + t^6, t^2, t^5)$

(t^2, t^2, t)

For the first box, we have $h_0 = 1$, so the box has only one column. We have used $t^4 + t^6$ as z_0, which is the <u>first</u> in parametrization, so the <u>first</u> row should be marked and 1*. All the constants are* 0*, so we have:*

$$\left[\begin{array}{c|c|c} \boldsymbol{1} & & \\ 0 & & \\ 0 & & \end{array}\right]$$

in the first box. Continuing with the second box, we know that $h_1 = 2$, hence the box has 2 *columns. The dividend t^2 is in the <u>second</u> row in the parametrization, so the <u>second</u> row of the second box should be marked and* $[\boldsymbol{1}\ \ \boldsymbol{0}]$*. And recall that we subtract* 1 *from the first series* *in the second division, so we put a* 1 *to the first row second column of the current box. Other entries should be* 0 *as the other constants are* 0 *for the second expansion.*

$$\left[\begin{array}{c|cc|c}
\boldsymbol{1} & 0 & 1 & \\
0 & \boldsymbol{1} & \boldsymbol{0} & \\
0 & 0 & 0 &
\end{array}\right]$$

For the last box, the third row should be marked, as the dividend t is the in the third row of the parametrization. And we have ones in the second columns of the first and second row, as we have subtracted 1*'s in the second division. Other entries should be* 0*. So the Hamburger-Noether matrix is:*

$$\left[\begin{array}{c|cc|cc}
\boldsymbol{1} & 0 & 1 & 0 & 1 \\
0 & \boldsymbol{1} & \boldsymbol{0} & 0 & 1 \\
0 & 0 & 0 & \boldsymbol{1} & \boldsymbol{0}
\end{array}\right]$$

CHAPTER 5

ARF THEORY

In this chapter, we give a review of Arf theory. As we have mentioned in Chapter 2, with this theory, Arf answers the important question asked by Du Val: ‘Given the parametrization of a space branch, how can we find its characters?’ While answering this question, he develops new algebraic concepts, which we present in this chapter.

For a branch given by the parametrization $\{x_1 = \varphi_1(t),\ x_2 = \varphi_2(t),\ ...,\ x_n = \varphi_n(t)\}$, the completion of the local ring of the curve branch is $k[[\varphi_1(t), ..., \varphi_n(t)]]$, Arf shows that the characters can be obtained by following these steps:

- Construct a ring with a method described later, called the canonical closure (Arf closure) from the ring $k[[\varphi_1(t), ..., \varphi_n(t)]]$.
- Form the semigroup of orders of the elements in the canonical closure of the ring $k[[\varphi_1(t), ..., \varphi_n(t)]]$.
- Pick some of the numbers from that semigroup with a method to be described later.

We obtain the multiplicity sequence of the curve branch by applying the Jacobian Algorithm to these characters. Before describing each step in detail, we give the necessary definitions and background.

5.1 Notation and Basic Definitions

In this section, we use the notation in [38]. Let R be a subring of the formal power series ring with one indeterminate $k[[t]]$. We define $W(R)$ as the set consisting of the orders of the elements of R:

$$\begin{aligned} W(R) &= \{ord(r) | r \in R\} \\ &= \{i_0 = 0 < i_1 < ... < i_r < ...\} \end{aligned}$$

Clearly, $W(R)$ is a semigroup of nonnegative integers.

Theorem 5.1.1 *[1, Theorem 1] Let v be the greatest common divisor of the elements of $W(R)$. Then for a sufficiently large natural number m, there exists a power series of order 1*

$$\tau = t(1 + \sum_{l=1}^{\infty} \beta_l t^l), \qquad \beta_l \in k$$

such that $i_{m+1} = i_m + v, \quad i_{m+2} = i_m + 2v, \quad ..., \quad i_{r+k} = i_r + kv, \quad ...$ *and every element of R is of the form $\sum_{j=0}^{\infty} \alpha_j \tau^{jv}$.*

As a consequence of this theorem, we can say that if $v = 1$, then for sufficiently large m, R contains the power series of all orders greater than or equal to m. We will generally be working with rings having $v = 1$.

Note that a ring R with $v = 1$ can be given as

$$R = k + kS_{i_1} + kS_{i_2} + ... + k[[t]]S_{i_r},$$

where S_{i_j} is an arbitrary element in R with order i_j, and $i_1 < i_2 < ... < i_r$ are elements in the semigroup $W(R)$, which contains every integer greater than or equal to i_r.

For an element n of $\mathbb{N}$, define I_n as

$$I_n = \{r \in R \mid ord(r) \geq n\},$$

which is the subset of R consisting of all elements of order greater than or equal to n.

Theorem 5.1.2 *[1, Theorem 2] Let r be an element of R of order* 0*. The multiplicative inverse of r is again an element of R, .*

Remark 5.1.3 *[1] For an order* 0 *element r of R, nth root of r is again an element of R,*

Remark 5.1.4 *I_n is an ideal of R.*

Consider the set

$$I_n/S_n \;=\; \{r.S_n^{-1} \mid ord(r) \geq n\}$$

where S_n is an element of R of order n. That is, I_n/S_n is the set consisting of the quotients of elements of I_n by S_n. Note that, I_n/S_n is not always a ring. Hence, we consider the smallest subring of R inside $k[[t]]$ containing I_n/S_n, and denote it by $[I_n/S_n]$.

Theorem 5.1.5 *[1, Theorem 3] The ring $[I_n/S_n]$ is independent of the chosen element S_n of order n .*

As a consequence, without loss of generalization, we can set $[I_n]$ instead of $[I_n/S_n]$. So, $[I_n]$ is the smallest subring of $k[[t]]$ containing I_n/S_n.

Example 5.1.6 *[1] Let R be the subring of the formal power series ring with one indeterminate generated by the elements $X = t^4$ and $Y = t^{10} + t^{15}$. Then*

$$W(H) = \{0, 4, 8, 10, 12, 14, 16, 18, 20, 22, 24, 25, 26, 28, 29, 30, 32, 33, 34, 35, 36, 37, 38, ...\}$$

Setting $S_4 = X$, the orders of the elements of I_4/S_4 are,

$$0, 4, 6, 8, 10, 12, 14, 16, 18, 20, 21, 22, 24, 25, 26, 28, 29, 30, 31, 32, 33, 34, ...$$

The semigroup generated by these elements are:

$$0, 4, 6, 8, 10, 12, 14, 16, 18, 20, 21, 22, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, ...$$

but the element $(Y/X)^2 - X^3 = 2t^{17} + t^{22}$ is also in the ring $[I_4]$, so $17 \in W([I_4])$.

This example of Cahit Arf shows that the semigroup $W([I_n])$ contains the integers $i_n - i_n = 0, \quad i_{n+1} - i_n, \quad i_{n+2} - i_n, ...$, which are the orders of the elements of I_{i_n}/S_{i_n}, but it may contain more than those. This is an example of a ring, for which $I_n/S_n \neq [I_n]$ for $n = 4$. Indeed, I_n/S_n is always a ring only for some special rings R.

5.2 Arf Rings, Arf Semigroups

Definition 5.2.1 *[1] A ring R is called an Arf Ring, if for all $n \in W(R)$, $[I_n]$ is equal to I_n/S_n.*

Example 5.2.2 *The ring $k[[t]]$ is a trivial example of an Arf ring.*

Cahit Arf calls the rings mentioned above as 'canonical rings', which are named by Lipman as Arf Rings in [33]. Similarly, Arf defines 'canonical semigroups', which are called later as Arf Semigroups.

Definition 5.2.3 *[1] A semigroup $G = \{i_0 = 0 < i_1 < i_2 < ... < i_n < ...\}$ is called an Arf Semigroup, if $\{i_n - i_n, i_{n+1} - i_n, i_{n+2} - i_n, ...\}$ is a semigroup for all $i_n \in G$.*

Example 5.2.4 *The semigroup $\mathbb{N}$ is a trivial example of an Arf Semigroup.*

Remark 5.2.5 *[1] If R is an Arf Ring, $W(R)$ is an Arf Semigroup, but the reverse is not true.*

Theorem 5.2.6 *[1] (i) Intersection of Arf rings is again an Arf ring.*

(ii) Intersection of Arf semigroups is again an Arf semigroup.

Theorem 5.2.7 *[1] If R is an Arf ring, then $[I_n]$ is also an Arf ring for any n.*

We can now understand the structure of Arf rings and Arf semigroups. Let R be an Arf ring. For a sufficiently large N, we have $(N-1)v,\ Nv, ... \in W(R)$ and $[I_{(N-1)v}] = k[[T]]$ where $T = \tau^v$ by Theorem 5.1.1, [1]. We choose an element T_{r-1} of order v_{r-1}in the Arf ring $[I_{(N-1)v}]$. Then, we have

$$[I_{i_r-1}] = k + T_{r-1}[I_{(N-1)v}]$$

where $i_r = (N-1)v$. The ring $[I_{i_r-1}]$ and the semigroup $W([I_{i_r-1}]) = \{0, v_{r-1} + v\mathbb{N}\}$ are Arf by Remark 5.2.5 and Theorem 5.2.7. Now, we choose an element $T_{r-2} \in [I_{i_r-1}]$ of order v_{r-2}. Then, we have

$$\begin{aligned} [I_{i_r-2}] &= k + T_{r-2}[I_{i_r-1}] \\ &= k + kT_{r-2} + kT_{r-2}T_{r-1}k[[T]] \end{aligned}$$

and $W([I_{i_r-2}]) = \{0, v_{r-2}, v_{r-2} + v_{r-1} + v\mathbb{N}\}$. Continuing in this manner, we obtain the Arf Ring

$$R = k + kT_1 + kT_1T_2 + ... + kT_1T_2...T_{r-2} + k[[T]]T_1T_2...T_{r-2}T_{r-1}$$

and the Arf Semigroup

$$W(H) = \{0, v_1, v_1 + v_2, ..., v_1 + v_2 + ... + v_{r-2}, v_1 + v_2 + ... + v_{r-1} + v\mathbb{N}\}$$

If $ord(T_h) = v_h$ and if $T_h = kT_{h+1} + kT_{h+1}T_{h+2} + ... + k[[T]]]T_1T_2...T_{r-1}$, then $v_h \in \{v_{h+1}, v_{h+1} + v_{h+2}, ..., v_{h+1} + v_{h+2} + ... + v_{r-1} + v\mathbb{N}\}$.

Noting that $k[[t]]$ is an Arf ring and $\mathbb{N}$ is an Arf semigroup, we can give the following definitions.

Definition 5.2.8 *For a subring R of formal power series ring, the smallest Arf ring containing R is called the Arf Closure of R. In other words, the Arf closure of a ring is the intersection of all Arf rings containing it. We denote the Arf closure of R by R^*.*

Definition 5.2.9 *For a subsemigroup G of nonnegative integers, the smallest Arf semigroup containing G or equivalently, intersection of all Arf Semigroups containing G is called the Arf Closure of G. We denote the Arf Closure of G by G^*.*

5.3 Characteristic Semigroup, Arf Characters

It is now time to define the characters of the branch, which are the integers giving the multipicity sequence after an application of the modified Jacobian algorithm.

Theorem 5.3.1 *[1] Let G be a subsemigroup of the non negative integers. The intersection of all semigroups g with $G^* = g^*$ is a semigroup g_χ and $g_\chi^* = G^*$.*

Definition 5.3.2 *[1] The semigroup $g_\chi = \bigcap_{g^*=G^*} g$ in Theorem 5.3.1 is called the characteristic subsemigroup of G.*

Observe that the characteristic semigroup is the smallest semigroup, whose Arf closure is equal to the Arf Closure of G. For a subsemigroup G of $\mathbb{N}$, the minimal generators $\chi_1, \chi_2, ..., \chi_h$ of G are defined in the following way:

(i) $G = \{\alpha_1\chi_1 + ... + \alpha_h\chi_h \mid \alpha_1, ..., \alpha_h \in \mathbb{N}^+\}$

(ii) χ_1 is the smallest nonzero integer in G.

(iii) χ_n is the smallest nonzero integer in $G - \langle\chi_1, ..., \chi_{n-1}\rangle$ for $n \geq 2$

Definition 5.3.3 *[1] The minimal generators $\chi_1, ..., \chi_h$ of the characteristic subsemigroup g_χ of G are said to be the characters of G.*

Theorem 5.3.4 *[1] Let G be a semigroup generated by $\{\chi_1, \chi_2, ..., \chi_l\}$. Then the set of characters of G is a subset of $\{\chi_1, \chi_2, ..., \chi_l\}$.*

5.4 Computation of the Arf Closure

We start with the computation of the Arf closure of a semigroup, which is much simpler.

5.4.1 Computation of the Arf Closure of a Semigroup

Theorem 5.4.1 *Let G be a semigroup with characters $\chi_1 < \chi_2 < ... < \chi_h$ and the integers $v_1, ..., v_{N-1}, v$ be obtained by applying the modified Jacobian algorithm to $\chi_1 < \chi_2 < ... < \chi_h$. Then $G^* = \{0, v_1, v_1 + v_2, ..., v_1 + v_2 + ... + v_{N-1} + \mathbb{N}v\}$ is the Arf closure of the semigroup G. The integers $v_1, ..., v_{N-1}, v$ are the divisors in the algorithm explained below and the quotients represent the number of times each divisor is repeated in the sequence $v_1, ..., v_{N-1}, v$.*

At this point, it is a good idea to give a quick summary of Du Val's [23] modified Jacobian algorithm.

5.4.2 The modified Jacobian Algorithm

To apply the modified Jacobian algorithm to the set $G = \{g_1, g_2, ..., g_n\} \subset \mathbb{N}$, start by forming $g_1, g_2, ..., g_n$ as the first row and choosing the smallest g_{i_0} among them as the divisor of the first row. Divide the smallest element g_{j_0} in $G - \{g_{i_0}\}$ with g_{i_0}. Let the quotient be m. Subtract mg_{i_0} from all elements in the first row except g_{i_0} and obtain the second row. Apply the same process to the second row and continue in the same manner. If 0 is obtained, omit that column. The algorithm stops, when a row with only one element is obtained.

Example 5.4.2 *We apply the modified Jacobian Algorithm to the set {100, 150, 275, 290,* *312*}.

100	150	275	290	312	1
	$\underline{100}$	$\underline{100}$	$\underline{100}$	$\underline{100}$	
100	**50**	175	190	212	2
$\underline{100}$		$\underline{100}$	$\underline{100}$	$\underline{100}$	
	50	75	90	112	1
		$\underline{50}$	$\underline{50}$	$\underline{50}$	
	50	**25**	40	62	1
	$\underline{25}$		$\underline{25}$	$\underline{25}$	1
	25	25	**15**	37	1
	$\underline{15}$	$\underline{15}$		$\underline{15}$	
	10	10	15	22	1
		$\underline{10}$	$\underline{10}$	$\underline{10}$	
	10		**5**	12	2
	$\underline{10}$			$\underline{10}$	
			5	**2**	2
			$\underline{4}$		
			1	2	2
				$\underline{2}$	
			1		

The divisors in the algorithm give us the different multiplicities of the points and the quotients corresponding to each divisor tells us how many times each multiplicity is repeated in the sequence. Hence, the multiplicity sequence corresponding to the characters in this example is (100, 50, 50, 50, 25, 15, 10, 5, 5, 2, 2, 1, 1).

5.4.3 Computation of the Arf Closure of a ring

Let R be a subring of $k[[t]]$. If the semigroup of orders of R is $W(R) = \{0 = i_0, i_1, ..., i_{h-1}, i_h + \mathbb{N}v\}$ with $i_0 < i_1 < ... < i_h$, we have seen that R can be presented as

$$R = k + kS_{i_1} + kS_{i_2} + ... + kS_{i_{h-1}} k[[T]]S_{i_h}$$

where S_{i_n} is an element of R of order i_n and $T = \tau^v$ is as explained in Theorem 5.1.1. Let R_1 be the ring

$$[I_{i_1}] = \sum k\left(\frac{S_{i_2}}{S_{i_1}}\right)^{\alpha_2}\left(\frac{S_{i_3}}{S_{i_1}}\right)^{\alpha_3}...\left(\frac{S_{i_{h-1}}}{S_{i_1}}\right)^{\alpha_{h-1}} + k[[T]]\frac{S_{i_h}}{S_{i_1}}$$

where the sum is over all $\alpha_2, \alpha_3, ..., \alpha_{h-1}$ satisfying

$$\alpha_2(i_2 - i_1) + \alpha_3(i_3 - i_1) + \cdots + \alpha_{h-1}(i_{h-1} - i_1) < (i_h - i_1)$$

Then, $R \subset k + R_1 S_{i_1} \subset R^*$. Taking the Arf closure of both sides, we obtain $R^* = k + R_1^* S_{i_1}$ and applying the same procedure to R_1 and continuing inductively, we have $R_N = k[[T]]$ for sufficiently large N. Hence,

$$R^* = k + kT_1 + kT_1T_2 + \cdots + kT_1T_2\cdots T_{N-1} + k[[T]]T_1T_2\cdots T_{N-1}T_N$$

where T_{i+1} is an element with the smallest order in R_i.

5.4.4 Examples

Example 5.4.3 *Let R be the ring generated by the elements $X = t^4$, $Y = t^6$ and $Z = t^{11}$, i.e. $R = k[[t^4, t^6, t^{11}]]$. Then $W(R) = \{0, 4, 6, 8, 10, 11, 12, 14 + \mathbb{N}\}$, and thus R can be written as*

$$R = k + kt^4 + kt^6 + kt^8 + kt^{10} + kt^{11} + kt^{12} + k[[t]]t^{14}$$

and

$$R_1 = [I_4] = \sum k\left(\frac{t^6}{t^4}\right)^{\alpha_2}\left(\frac{t^8}{t^4}\right)^{\alpha_3}\left(\frac{t^{10}}{t^4}\right)^{\alpha_4}\left(\frac{t^{11}}{t^4}\right)^{\alpha_5}\left(\frac{t^{12}}{t^4}\right)^{\alpha_6} + k[[t]]\left(\frac{t^{14}}{t^4}\right)$$

Thus, $R_1 = k + kt^2 + kt^4 + k[[t]]t^6$ with $W(R_1) = \{0, 2, 4, 6 + \mathbb{N}\}$ implying

$$R_2 = \sum k\left(\frac{t^4}{t^2}\right)^{\alpha_2} + k[[t]]\left(\frac{t^6}{t^2}\right) = k + kt^2 + k[[t]]t^4$$

Continuing the same way, we get $R_3 = k[[t]]t^2$ and finally we reach $R_4 = k[[t]]$. Hence,

$$R^* = k + kt^4 + kt^4t^2 + kt^4t^2t^2 + k[[t]]t^4t^2t^2t^2 = k + kt^4 + kt^6 + kt^8 + k[[t]]t^{10}$$

Note that, in this example, $W(R^*) = \{0, 4, 6, 8, 10 + \mathbb{N}\}$ and the Arf closure of the semigroup generated by the set $\{4, 6, 11\}$ is also $\{0, 4, 6, 8, 10 + \mathbb{N}\}$. In fact, this is not a coincidence:

Remark 5.4.4 *Let $a_1, a_2, ..., a_n$ be natural numbers. For the curve branch having parametrization*

$$x_1 = t^{a_1},\ x_2 = t^{a_2},\ ...,\ x_n = t^{a_n}$$

as in Example 5.4.3, $W(R^*)$ is equal the Arf closure of the semigroup generated by the natural numbers $\{a_1, a_2, ..., a_n\}$.

On the other hand, Example 5.4.3 should not give us the wrong impression that finding the Arf closure is quite easy. If the parametrization of the curve branch consists of polynomials in t, computation of $W(R^*)$ can involve lots of computations and may be very difficult. We should carefully consider the effects of higher degree terms. In the next chapter, we give an alternative method for constructing the Arf closure R^* of a ring R that does not need the computation of the semigroup $W(R^*)$.

Example 5.4.5 *Let us calculate the Arf closure of the ring R generated by the elements $X = t^4$, $Y = t^8 + t^9$ and $Z = t^{15}$. Namely, $R = k[[t^4, t^8 + t^9, t^{15}]]$ and as $Y - X^2 = t^9 \in R$, $W(R) = \{4, 8, 9, 12, 13, 15 + \mathbb{N}\}$.*

$$R = k + kt^4 + k(t^8 + t^9) + kt^9 + kt^{12} + kt^{13} + k[[t]]t^{15}$$

$$R_1 = \sum k\left(\frac{t^8 + t^9}{t^4}\right)^{\alpha_2}\left(\frac{t^9}{t^4}\right)^{\alpha_3}\left(\frac{t^{12}}{t^4}\right)^{\alpha_4}\left(\frac{t^{13}}{t^4}\right)^{\alpha_5} + k[[t]]\frac{t^{15}}{t^4}$$

We can write R_1 as,

$$R_1 = k + kt^4 + kt^5 + k[[t]]t^8$$

Applying the same process to R_1, we get $R_2 = k[[t]]$, so

$$R^* = k + kt^4 + k[[t]]t^8$$

In Example 5.4.5 we have seen that we can not always form $W(R)$ directly from the orders of the generating elements X, Y, Z, we need to consider all the elements and their orders in R, which is not very easy algorithmically. This difficulty can be better observed in the next example.

Example 5.4.6 *If $R = k[[t^4, t^6 + t^9, t^{14}]]$, $W(R)$ consists of the elements like 15 and 17 because $Y^2 - X^3 = 2t^{15} + t^{18}$ and $X^2Y - Z = t^{17}$ are the elements of the ring R. So, $W(R) = \{4, 6, 8, 10, 12, 14 + \mathbb{N}\}$ implying that*

$$R = k + kt^4 + k(t^6 + t^9) + kt^8 + kt^{10} + kt^{12} + k[[t]]t^{14}$$

and

$$R_1 = \sum\left(\frac{t^6 + t^9}{t^4}\right)^{\alpha_2}\left(\frac{t^8}{t^4}\right)^{\alpha_3}\left(\frac{t^{10}}{t^4}\right)^{\alpha_4}\left(\frac{t^{12}}{t^4}\right)^{\alpha_5} + k[[t]]\left(\frac{t^{14}}{t^4}\right)$$

and as $(t^2+t^5)^2 - t^4$ *is in* R_1, $W(R_1) = \{2, 4, 6+\mathbb{N}\}$. *So,* R_1 *can be written as*

$$R_1 = k + k(t^2+t^5) + kt^4 + k[[t]]t^6$$

Then,

$$R_2 = \sum \left(\frac{t^4}{t^2+t^5}\right)^{\alpha_2} + k[[t]]\left(\frac{t^6}{t^2+t^5}\right) = \sum \left(t^2 - t^5 + t^8 - ...\right)^{\alpha_2} + k[[t]]\left(t^4 - t^7 + t^{10} - ...\right)$$

Thus, $R_3 = k + k[[t]]t^2$ *and finally* $R_4 = k[[t]]$ *implying that*

$$\begin{aligned} R^* &= k + kt^4 + kt^4(t^2+t^5) + kt^4(t^2+t^5)(t^2-t^5+t^8-..) + k[[t]]t^4(t^2+t^5)(t^2-t^5+t^8-..)t^2 \\ &= k + kt^4 + kt^4(t^2+t^5) + kt^8 + k[[t]]t^{10} \end{aligned}$$

Example 5.4.6 shows that in the computation of the Arf closure, we need to determine the terms of a series obtained by division, so the question arises "Up to which degree should we consider?" This is a crucial question that will be discussed in detail in the next chapter.

Having understood the computation of the Arf closure, we can now give Cahit Arf's important theorem that makes it possible to compute the characters of a space branch.

Theorem 5.4.7 *[1] Let C be a branch given with the parametrization* $x_1 = \varphi_1(t)$, $x_2 = \varphi_2(t)$, ..., $x_n = \varphi_n(t)$ *and with the corresponding local ring* $R = k[[\varphi_1(t), \varphi_2(t), ..., \varphi_n(t)]]$. *If* $W(R^*) = \{v_1, v_1+v_2, ..., v_1+v_2+...v_{N-1}+\mathbb{N}\}$, *then the multiplicity sequence of the curve branch is*

$$v_1, v_2, ..., v_{N-1}, 1, 1, ...$$

Combining all the theory presented, Arf gives the following algorithm for finding the characters of a branch given with the parametrization $x_1 = \varphi_1(t)$, $x_2 = \varphi_2(t)$, ..., $x_n = \varphi_n(t)$:

1. Start with $R = k[[\varphi_1(t), \varphi_2(t), ..., \varphi_n(t)]]$.
2. Find the Arf closure R^*.
3. Find $W(R^*)$ from R^*.
4. Construct the characteristic sub-semigroup $g_\chi(W(R^*))$ of $W(R^*)$.
5. Find the minimal set of generators $\chi_1, \chi_2, ..., \chi_h$ of g_χ.

$\chi_1, \chi_2, ..., \chi_h$ are the Arf characters.

With this theory, Arf gives a complete answer to Du Val's question about the computation of the characters of a space branch, and generalizes the plane case. Hence, the following are equivalent:

i) Two branches have the same multiplicty sequence.

ii) Two branches have the same Arf characters.

We can now give some examples:

Example 5.4.8 *Consider the plane branch with the corresponding local ring* $R_1 = k[[t^4, t^9]]$. *Since* $W(R_1^*) = \{4, 8 + \mathbb{N}\}$, *the Arf characters are* $\{4, 9\}$. *Consider the space branch in Example 5.4.5 with the corresponding local ring* $R_2 = k[[t^4, t^8 + t^9, t^{15}]]$. *We have shown in Example 5.4.5 that* $W(R_2^*) = \{4, 8 + \mathbb{N}\}$, *so the Arf characters are* $\{4, 9\}$. *Hence, these two branches have the same multiplicity sequence.*

Example 5.4.9 *The curve branch with the corresponding local ring* $R = k[[t^4, t^6 + t^9, t^{14}]]$ *in Example 5.4.6 has* $W(R^*) = \{4, 6, 8, 10 + \mathbb{N}\}$, *and so* $\{4, 6, 11\}$ *are the Arf characters of the curve.*

5.5 Arf Closure and Hamburger-Noether matrices

It is also possible to construct the Arf closure of a ring by using the Hamburger-Noether matrices. Before explaining the process with the notation of [16], we give the definition of an Apery basis.

Definition 5.5.1 *Let* S *be a semigroup and* $n \in S$. *Consider the elements of* S *in the congruence classes of* $(\overline{0_S}, \overline{1_S}, ..., \overline{(n-1)_S})$ *in mod* n. *Among the elements of each congruence class, we choose the smallest element* s_i. *The set* $(S, n) = \{s_0, ..., s_{n-1}\}$ *is called the Apery Basis of* S *with respect to* n.

Example 5.5.2 *Let* $S = \{4, 6, 8 + \mathbb{N}\}$ *and* $n = 4$. *Then*
$\overline{0_S} = \{4, 8, 12, 16, ...\}$

$\overline{1_S} = \{9, 13, 17, ...\}$

$\overline{2_S} = \{6, 10, 14, 18, ...\}$

$\overline{3_S} = \{11, 15, 19, ...\}$

Hence, $s_0 = 4,\ s_1 = 9,\ s_2 = 6,\ s_3 = 11$ *and* $(S, 4) = \{4, 6, 9, 11\}$.

Let C be a branch given with parametrization $\varphi_1, ..., \varphi_n$, with Hamburger-Noether expansion

$$\overline{Z_{j-1}} = \sum_{i=1}^{h_j} A_{ji} {z_j}^i + {z_j}^{h_j} Z_{j+1}, \quad j = 0, ..., r$$

and with Hamburger-Noether matrix H. Recall that the multiplicity sequence of C is

$$\underbrace{n_0, ..., n_0}_{h_0\ times}, \underbrace{n_1, ..., n_1}_{h_1\ times}, ..., \underbrace{n_s, ..., n_s}_{h_r\ times}.$$

Let $R = k[[\varphi_1, ..., \varphi_n]]$ and R^* be the Arf closure of R. Construct another matrix H^* from H as follows:

- Let $i_1, ..., i_d$ be the different marked rows of H.
- Let $B_{j_1}, ..., B_{j_d}$ be the boxes of H, where the marked rows $i_1, ..., i_d$ appear for the first time respectively.
- Define $\alpha_{j_1}, \alpha_{j_2}, ..., \alpha_{j_d}$ such that $\alpha_k = h_0 n_0 + h_1 n_1 + ... + h_{k-1} n_{k-1} + n_k$, where $k = j_1, ..., j_d$.
- Construct the Apery basis $(W(R^*), n_0)$ of the semigroup $W(R^*)$ with respect to n_0 (Recall that the semigroup $W(R^*)$ is completely determined by the multiplicity sequence of C).
- For each element γ_i in $(W(R^*), n_0)$ that is not in the set $\{\alpha_{j_1}, ..., \alpha_{j_d}\}$, find the natural numbers q_j such that $\gamma_i = h_0 n_0 + h_1 n_1 + ... + h_{j-1} n_{j-1} + q_j n_j$.
- The new matrix H^* has n_0 rows, for which the first d rows are the same with the marked rows of H. For each element γ_i, we add a new row with a 1 in the column $(h_0 + h_1 + ... + h_{j-1} + q_j)$ and 0's for the rest.

Example 5.5.3 *Let R be the ring $k[[t^4, t^6, t^{13}]]$. It is not hard to show that the Hamburger-Noether matrix H of the branch given by the parametrization (t^4, t^6, t^{13}) is:*

$$H = \left[\begin{array}{c|cccc|cc} \boldsymbol{1} & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & \boldsymbol{1} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & \boldsymbol{1} & \boldsymbol{0} \end{array}\right]$$

and the multiplicity sequence is $(4, 2, 2, 2, 2, 1, 1)$*. We have three rows and three boxes in H. As all three rows have been marked and they all been marked just once, we use all three boxes. Namely, from the first box,* $\alpha_1 = 4$*,* $\alpha_2 = 4 + 2 = 6$ *from the second and* $\alpha_3 = 4 + 4.2 + 1 = 13$ *from the third box.* $W(R^*)$ *on the other hand, is equal to* $\{0, 4, 4+2, 4+2+2, 4+2+2+2, 4+2+2+2+2+\mathbb{N}\}$ *and the Apery basis of* $W(R^*)$ *with respect to* 4 *is* $(W(R^*), 4) = \{4, 6, 13, 15\}$*. Hence, the matrix* H^* *has rows as* $(W(R^*), 4)$ *has four elements. The first three rows of* H^* *are the same with the rows of H. We add one last row for* $15 \in (W(R^*), 4) - \{\alpha_1, \alpha_2, \alpha_3\}$*. As* $15 = 1.4 + 4.2 + 3.1$*, the last row has a* 1 *in the* $1 + 4 + 3 = 8$*th column and* 0*'s for the rest. That is,*

$$H^* = \left[\begin{array}{c|cccc|ccc} \boldsymbol{1} & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & \boldsymbol{1} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & \boldsymbol{1} & \boldsymbol{0} & \boldsymbol{0} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right]$$

Indeed, it can be shown that the Arf closure R^* *of* R *is* $k[[t^4, t^6, t^{13}, t^{15}]]$ *and* H^* *is the Hamburger-Noether matrix of* R^**.*

Example 5.5.4 *Let* $R = k[[t^6, t^9, t^{11}]]$*. It can be shown that the Hamburger-Noether matrix of R is*

$$H = \left[\begin{array}{c|c|c|cc} \boldsymbol{1} & 0 & 0 & 1 & 0 \\ 0 & \boldsymbol{1} & 0 & \boldsymbol{1} & \boldsymbol{0} \\ 0 & 0 & \boldsymbol{1} & 0 & 1 \end{array}\right]$$

The multiplicity sequence is $(6, 3, 2, 1, 1)$ *so that* $W(R^*) = \{0, 6, 9, 11 + \mathbb{N}\}$ *and* $(W(R^*), 6) = \{6, 9, 11, 13, 14, 16\}$*. There are three marked rows in H and they have been marked in box 1, box 2 and box 3 for the first time. Thus,* $\alpha_1 = 6$ *from the first box,* $\alpha_2 = 6 + 3 = 9$ *from the second box and* $\alpha_3 = 6 + 3 + 2 = 11$ *from the second box. Then,* $\{13, 14, 16\} \in (W(R^*), 6) - \{\alpha_1, \alpha_2, \alpha_3\}$*. The new matrix* H^* *has six rows as the cardinality of* $(W(R^*), 6)$ *is* 6*. The first three rows are the same with the rows of H as H has three marked rows. We add a row for* $13 = 1.6 + 1.3 + 1.2 + 2.1$ *with a* 1 *in the* $1 + 1 + 1 + 2 = 5$*th column, and zeros for the rest, another row for* $14 = 1.6 + 1.3 + 1.2 + 3.1$ *with a* 1 *in the* $1 + 1 + 1 + 3 = 6$*th column and zeros for the rest and one last row for* $16 = 1.6 + 1.3 + 1.2 + 5.1$ *with a* 1 *in the*

$1 + 1 + 1 + 5 = 8$*th column and zeros for the other entries. Thus,*

$$H^* = \left[\begin{array}{c|c|c|ccccc} \boldsymbol{1} & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & \boldsymbol{1} & 0 & \boldsymbol{1} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ 0 & 0 & \boldsymbol{1} & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right]$$

CHAPTER 6

A FAST ALGORITHM FOR COMPUTING THE ARF CLOSURE

In Chapter 5, we have seen the method given by Arf to compute the Arf closure of a given branch. It is not easy to convert this method to an algorithm, and implement this in a computer algebra program. In the literature, there is only one algorithm given by Arslan, implemented in Maple [2]. The method given by Arf starts with determining the semigroup of values and the conductor of the branch C. Arslan gives an algorithm for this for his implementation. In fact, this problem has been studied by many mathematicians, and different algorithms have been given [15], [28]. We should note that all of these algorithms are quite slow. This is not unexpected, since even the special case of this problem is the famous Fobenius problem or coin problem, still being studied by many mathematicians. The difficulty in determining the conductor lies in the need to check so many relations between the generators.

The second difficulty in computing the Arf closure is the problem we have observed Example 5.4.6. We have seen that we need to determine the terms of a series obtained by division up to some degree, so this degree must be chosen so that no information is lost. Arslan uses the conductor of the local ring coresponding to the branch in his algorithm [2], but this is a quite large bound, which makes the algorithm slower.

We should note that the computation of the Arf closure by using Hamburger-Noether matrices given at the end of Chapter 5 is also restricted with the same problems.

In this chapter, we give an algorithm, which does not involve determining the conductor, and computing the elements in the local ring at each step. Also, by giving a quite small bound for doing the divisions, we obtain an algorithm, which works very fast compared to other algorithms.

Let C be an algebroid curve branch with the local ring $\Theta = k[[\varphi_1(t), ..., \varphi_n(t)]]$ and the semigroup of values $W(\Theta) = \{i_0, i_1, ..., i_{h-1}, i_h + \mathbb{N}\}$ where $i_0 = 0$ and $i_1 = ord(\varphi_1(t))$. We have seen that the ring Θ can be presented as

$$\Theta = k + kS_{i_1} + kS_{i_2} + ... + kS_{i_{h-1}} k[[T]]S_{i_h}$$

where S_{i_j}'s are elements of Θ of order i_j chosen such that $\varphi_1(t), ..., \varphi_n(t)$ are among them. (For example, $S_{i_1} = \varphi_1(t)$). Recall also that

$$[I_{i_1}] = \sum k \left(\frac{S_{i_2}}{\varphi_1}\right)^{\alpha_2} \left(\frac{S_{i_3}}{\varphi_1}\right)^{\alpha_3} ... \left(\frac{S_{i_{h-1}}}{\varphi_1}\right)^{\alpha_{h-1}} + k[[T]]\frac{S_{i_h}}{\varphi_1}$$

where the sum is over all $\alpha_2, \alpha_3, ..., \alpha_{h-1}$ satisfying

$$\alpha_2(i_2 - i_1) + \alpha_3(i_3 - i_1) + ... + \alpha_{h-1}(i_{h-1} - i_1) < (i_h - i_1).$$

Using the given presentation of Θ and the definition of $[I_{i_1}]$, we give the following lemma.

Lemma 6.0.5 $[I_{i_1}] = k[[\varphi_1, \frac{\varphi_2}{\varphi_1}, ..., \frac{\varphi_n}{\varphi_1}]]$.

Proof. The inclusion $k[[\varphi_1, \frac{\varphi_2}{\varphi_1}, ..., \frac{\varphi_n}{\varphi_1}]] \subset [I_{i_1}]$ is clear. Hence, let us prove the other direction. For this, it is sufficient to show that $\frac{S_{i_j}}{\varphi_1}$ is in $k[[\varphi_1, \frac{\varphi_2}{\varphi_1}, ..., \frac{\varphi_n}{\varphi_1}]]$. S_{i_j} is an element of Θ of order i_j by definition. As it is an element of Θ, it can be written as $S_{i_j} = \sum a_i(\varphi_1^{\alpha_{i_1}} ... \varphi_n^{\alpha_{i_n}})$. Hence,

$$\frac{S_{i_j}}{\varphi_1} = \sum a_i (\varphi_1)^{\alpha_{i_1} + \alpha_{i_2} + ... + \alpha_{i_n} - 1} \left(\frac{\varphi_2}{\varphi_1}\right)^{\alpha_{i_2}} ... \left(\frac{\varphi_n}{\varphi_1}\right)^{\alpha_{i_n}}$$

Then each summand in the expansion of $\frac{S_{i_j}}{\varphi_1}$ is an element of the ring $k[[\varphi_1, \frac{\varphi_2}{\varphi_1}, ..., \frac{\varphi_n}{\varphi_1}]]$. As a consequence, $\frac{S_{i_j}}{\varphi_1}$ is an element of $k[[\varphi_1, \frac{\varphi_2}{\varphi_1}, ..., \frac{\varphi_n}{\varphi_1}]]$ for any i, j. Since each $\frac{S_{i_j}}{\varphi_1}$ is an element of $k[[\varphi_1, \frac{\varphi_2}{\varphi_1}, ..., \frac{\varphi_n}{\varphi_1}]]$, then $\sum k \left(\frac{S_{i_2}}{\varphi_1}\right)^{\alpha_2} \left(\frac{S_{i_3}}{\varphi_1}\right)^{\alpha_3} ... \left(\frac{S_{i_{h-1}}}{\varphi_1}\right)^{\alpha_{h-1}}$ is also an element of $k[[\varphi_1, \frac{\varphi_2}{\varphi_1}, ..., \frac{\varphi_n}{\varphi_1}]]$. Thus, $[I_{i_1}] \subset k[[\varphi_1, \frac{\varphi_2}{\varphi_1}, ..., \frac{\varphi_n}{\varphi_1}]]$. And this proves our claim. ■

By using Lemma 6.0.5, we propose the following algorithm to compute the Arf closure:

- We choose the smallest ordered element of Θ, which is φ_1 and assign it to F_0.
- We find the blow-up ring $\Theta^{(1)} = k[[\varphi_1^{(1)}, ..., \varphi_n^{(1)}]]$ of $\Theta = \Theta^{(0)}$ by dividing all the series by φ_1 and then subtracting the constant terms, if necessary.
- If the blow up is smooth, the algorithm stops.

- If not, we find the smallest ordered element of $\Theta^{(1)}$ and assign it to F_1.
- We find the blow-up ring $\Theta^{(2)} = k[[\varphi_1^{(2)}, ..., \varphi_n^{(2)}]]$ of $\Theta^{(1)}$. If the curve is smooth, the algorithm stops.
- If not, we continue till we resolve the singularity.

The Algorithm is as follows:

Algorithm 3 ARF CLOSURE

Input: $(\varphi_1(t), ..., \varphi_n(t))$ is a primitive parametrization with $ord(\varphi_1) \leq ord(\varphi_i(t))$

Output: $F_0, ..., F_s \in k[[t]]$ such that $k + kF_0 + kF_0F_1 + ... + kF_0...F_{s-1} + k[[t]]F_0...F_s$ is the Arf Closure of $k[[\varphi_1, ...\varphi_n]]$

```
1: $F_0 \leftarrow \varphi_1,\ \varphi_1^{(1)} \leftarrow \varphi_1,\ \varphi_i^{(1)} \leftarrow \frac{\varphi_i}{F_0}\quad for\ 2 \leq i \leq n$
2: for $2 \leq i \leq n$ do
3:     Change $\varphi_i^{(1)}$ with $\varphi_i^{(1)} - constant\ term\ of\ \varphi_i^{(1)}$.
4: end for
5: Find the smallest index $j$ with $ord(\varphi_j^{(1)})$ is minimum among the orders of $\varphi_1^{(1)}, ..., \varphi_n^{(1)}$
6: if $ord(\varphi_j^{(1)}) = 1$ then
7:     Stop the algorithm and return $F_0$
8: else
9:     $F_1 \leftarrow \varphi_j^{(1)}$ and go to step 1 by taking $(\varphi_1^{(1)}, ..., \varphi_n^{(1)})$ instead of $(\varphi_1, ..., \varphi_n)$, $\varphi_j^{(1)}$ instead
       of $\varphi_1$, $F_1$ instead of $F_0$.
10: end if
```

The algorithm terminates because after finitely many blowing ups, we obtain a smooth curve branch. Hence, we obtain be a power series of order 1.

One can immediately realize that the given form of the algorithm is still not implementable, since it does not a give a bound that determines up to which degree the divisions of the power series are done. We will also solve this problem, but we first give some examples to see how the algorithm works.

Example 6.0.6 *Let R be the ring* $k[[t^4, t^6, t^{11}]]$. *We know from Example 5.4.3 that*

$$R^* = k + kt^4 + kt^4t^2 + kt^4t^2t^2 + k[[t]]t^4t^2t^2t^2 = k + kt^4 + kt^6 + kt^8 + k[[t]]t^{10}.$$

With the new algorithm, $F_0 = t^4$, $\varphi_1^{(1)} = t^4$, $\varphi_2^{(1)} = \frac{t^6}{t^4} = t^2$ and $\varphi_3^{(1)} = \frac{t^{11}}{t^4} = t^7$. None of the $\varphi_i^{(1)}$'s have order 1*, so we take $F_1 = t^2$, as $\varphi_2^{(1)} = t^2$ has the minimum order among the $\varphi_i^{(1)}$'s. We continue the algorithm with the parametrization (t^4, t^2, t^7). Then, $\varphi_1^{(2)} = \frac{t^4}{t^2} = t^2$, $\varphi_2^{(2)} = t^2$ and $\varphi_3^{(2)} = \frac{t^7}{t^2} = t^5$. None of the $\varphi_i^{(2)}$'s has order one. $\varphi_1^{(2)} = t^2$ has the smallest order. So we take $F_2 = \varphi_1^{(2)} = t^2$ and continue dividing the series with F_2. Then $\varphi_1^{(3)} = t^2$ again and this time $\varphi_2^{(3)} = \frac{t^2}{t^2} - 1$, $\varphi_3^{(3)} = \frac{t^5}{t^2} = t^3$. As $\varphi_2 = 0$, we continue without it, that is with (t^2, t^3) . None of the series have order one, taking $F_3 = \varphi_1^{(2)} = t^2$, we continue the algorithm. $\varphi_1^{(4)} = t^2$, $\varphi_3^{(4)} = \frac{t^3}{t^2} = t$. Now we have a series of order one, so the algorithm stops. Hence, the Arf closure of R is*

$$R^* = k + kF_0 + kF_0F_1 + kF_0F_1F_2 + k[[t]]F_0F_1F_2F_3 = k + kt^4 + kt^6 + kt^8 + k[[t]]t^{10}$$

as expected.

Example 6.0.7 *Let R be the ring $k[[t^4, t^8 + t^9, t^{15}]]$ in Example 5.4.5. We have shown that its Arf closure is $R^* = k + kt^4 + k[[t]]t^8$. Now, let us show it with our new algorithm. $F_0 = t^4$, $\varphi_1^{(1)} = t^4$, $\varphi_2^{(1)} = \frac{t^8+t^9}{t^4} = t^4 + t^5$ and $\varphi_3^{(1)} = \frac{t^{15}}{t^4} = t^{11}$. None of the series has order one, and the smallest ordered series is t^4. Hence, we continue the algorithm by taking $F_1 = t^4$. $\varphi_1^{(2)} = t^4$, $\varphi_2^{(2)} = \frac{t^4+t^5}{t^4} - 1 = t$, $\varphi_3^{(2)} = t^7$. As $ord(\varphi_2^{(2)}) = 1$, the algorithm stops. So the Arf closure is:*

$$R^* = k + kF_0 + k[[t]]F_0F_1 = k + kt^4 + k[[t]]t^8$$

6.1 Challenges

Even though it seems that the algorithm works very well in Example 6.0.6 and Example 6.0.7, we must solve the problem of determining a suitable bound for the divisions, as the following example shows..

Example 6.1.1 *Let R be the ring $k[[t^4, t^6 + t^9, t^{14}]]$ in Example 5.4.6. $F_0 = t^4$, $\varphi_1^{(1)} = t^4$, $\varphi_2^{(1)} = \frac{t^6+t^9}{t^4} = t^2 + t^5$, $\varphi_3^{(1)} = \frac{t^{14}}{t^4} = t^{10}$ after the first division. The smallest ordered series is $t^2 + t^5$. Hence, $F_1 = t^2 + t^5$, $\varphi_1^{(2)} = \frac{t^4}{t^2+t^5}$, $\varphi_2^{(2)} = t^2 + t^5$ and $\varphi_3^{(2)} = \frac{t^{10}}{t^2+t^5}$. Observe that $\varphi_1^{(2)}$ and $\varphi_3^{(2)}$ are not polynomials, but series. So, we should ignore some terms in these series, but some terms play significant roles in the next divisions. Losing them may cause mistakes in finding the Arf closure. In this example, we do the divisions by ignoring the terms with* *powers greater than* 14*, and we will explain later, why we have made this choice. With our choice,* $\varphi_1^{(2)} = t^2 - t^5 + t^8 - t^{11} + t^{14}$*,* $\varphi_2^{(2)} = t^2 + t^5$*,* $\varphi_3^{(2)} = t^8 - t^{11} + t^{14}$*.* $t^2 + t^5$ *and* $t^2 - t^5 + t^8 - t^{11} + t^{14}$ *have both order* 2 *as the smallest. We can use any of them to divide the series. Hence, without loss of generality, we choose* $t^2 + t^5$*. Then we have* $F_2 = t^2 + t^5$*,* $\varphi_1^{(3)} = \frac{t^2-t^5+t^8-t^{11}+t^{14}}{t^2+t^5} - 1 = -2t^3 + 3t^6 - 4t^9 + 5t^{12} +$ *higher degree terms,* $\varphi_2^{(3)} = t^2 + t^5$*,* $\varphi_3^{(3)} = \frac{t^8-t^{11}+t^{14}}{t^2+t^5} = t^6 - 2t^9 + 3t^{12} +$ *higher degree terms. We ignore the higher degree terms. Again,* $t^2 + t^5$ *have the smallest order. Hence,* $F_3 = t^2 + t^5$*,* $\varphi_1^{(4)} = \frac{-2t^3+3t^6-4t^9+5t^{12}}{t^2+t^5} = -2t + 5t^4 - 9t^7 + 14t^{10} - 14t^{13} +$ *higher degree terms. The algorithm stops, as we have obtained an order one series. Then the Arf closure of R is*

$$R^* = kt^4 + k(t^6 + t^9) + k(t^8 + 2t^{11} + t^{14}) + k[[t]](t^{10} + 3t^{13}) = kt^4 + k(t^6 + t^9) + kt^8 + k[[t]]t^{10}$$

Note that, we have used 14 as a bound for dividing the series in Example 6.1.1. Recall that we have already shown in Example 5.4.6 that the conductor is 14. This is not a coincidence. The conductor of the local ring corresponding to the branch can be used without losing any data. This is a consequence of the following lemma and the fact that the conductor of the blow up ring is always smaller.

Theorem 6.1.2 *[15, Proposition 1.1] Let C be a curve branch with the parametrization* $(\varphi_1(t), \varphi_2(t), ..., \varphi_n(t))$*. Let also c be the conductor of the semigroup of values of C. Then any parametrization* $(\varphi_1'(t), \varphi_2'(t), ..., \varphi_n'(t))$ *with* $\varphi_i(t) \equiv \varphi'_i(t)$ *(mod* t^c*) for* $1 \leq i \leq n$ *gives the same curve with C.*

Proof. Let $\overline{\varphi_i(t)}$ be the series obtained by $\varphi_i(t)$ by truncation in mod(t^c) for $1 \leq i \leq n$. Consider the curve branch $\overline{C}$ given with the parametrization $(\overline{\varphi_1(t)}, ..., \overline{\varphi_n(t)})$. The key point here is observing that the conductor of the semigroup of $\overline{C}$ is also c.
We are going to show that $\overline{\varphi_i(t)} \in k[[\varphi_1, ..., \varphi_n]]$ for any $1 \leq i \leq n$. For this, consider the series $f_i^m \in k[[x_1, ..., x_n]]$ for any $m \geq c$ such that

$$\overline{\varphi_i(t)} = \varphi_i(t) - f_i^{(c)}(\varphi_1(t), ..., \varphi_n(t)) - f_i^{(c+1)}(\varphi_1(t), ..., \varphi_n(t)) - ....$$

Such series exist since the rings $k[[\varphi_1, ..., \varphi_n]]$ and $k[[\overline{\varphi_1}, ..., \overline{\varphi_n}]]$ are complete. Then we can say that $\overline{\varphi_i} \in k[[\varphi_1, ..., \varphi_n]]$ and thus $k[[\overline{\varphi_1}, ..., \overline{\varphi_n}]] \subset k[[\varphi_1, ..., \varphi_n]]$. The other direction can be shown similarly. ■

Hence, to find the Arf closure of the ring $R = k[[x_1(t), ..., x_n(t)]]$ with the semigroup of values $W(R)$ and the conductor c, we can use Algorithm 3 with the bound c, while dividing the series. Although this seems to solve our problem, it still requires to find the conductor of the semigroup of values $W(R)$. But, it is not known so far how to find the conductor, without knowing the semigroup, which is a difficult problem as we have mentioned above. So, our aim is finding another bound, which is easier to compute, that we can use while dividing the series.

We use the following notation, throughout the remaining part of this chapter. Let $\Theta = k[[\varphi_1^{(0)}(t), \varphi_2^{(0)}(t), ..., \varphi_s^{(0)}(t)]]$ be an algebraic curve branch, where $\varphi_i^{(0)}(t)$'s, $(1 \leq i \leq s)$ are polynomials in t with increasing order. Setting S to the semigroup of Θ and c to the conductor of S, Let Θ^* be the Arf closure of Θ, S^* semigroup of Θ^*, and c^* conductor of S^*. Assume that k times blowing-up Θ solves its singularity. Let Θ_0 be Θ, Θ_i be the ith blow-up of Θ for $i \leq k$, F_i be the smallest ordered element of Θ_i and let a_i be the order of F_i. Schematically;

$$\begin{array}{lllll}
\textit{Column 1} & \textit{Column 2} & & \textit{Column s} & \\
\varphi_1^{(0)}(t), & \varphi_2^{(0)}(t), & \ldots, & \varphi_s^{(0)}(t), & \longrightarrow \quad F_0 = t^{a_0} + \text{higher degree terms} \\
\varphi_1^{(1)}(t), & \varphi_2^{(1)}(t), & \ldots, & \varphi_s^{(1)}(t), & \longrightarrow \quad F_1 = t^{a_1} + \text{higher degree terms} \\
\vdots & \vdots & & \vdots & \qquad\quad \vdots \\
\varphi_1^{(k)}(t), & \varphi_2^{(k)}(t), & \ldots, & \varphi_s^{(k)}(t), & \longrightarrow \quad F_k = t + \text{higher degree terms}
\end{array} \tag{6.1}$$

Here,

$$\varphi_j^{(i)}(t) = \begin{cases} \varphi_j^{(i-1)}(t), & \text{if } F_{i-1} = \varphi_j^{(i-1)}(t) \\ \\ \frac{\varphi_j^{(i-1)}(t)}{F_{i-1}} - c_{ij}, & \text{if } F_{i-1} \neq \varphi_j^{(i-1)}(t) \end{cases} \tag{6.2}$$

where $c_{ij} \in k$ and $c_{ij} \neq 0$ if and only if $ord(\varphi_j^{(i-1)}(t)) = ord(F_{i-1})$. Note also that $ord(F_{k-1}) \geq 2$ with $a_0 \geq a_1 \geq ... \geq a_{k-1} \geq 2$. Then $\Theta_i = k[[\varphi_1^{(i)}(t), \varphi_2^{(i)}(t), ..., \varphi_s^{(i)}(t)]]$ and the Arf closure of Θ is:

$$\begin{aligned} \Theta^* &= k + kF_0 + kF_0F_1 + ... + kF_0F_1...F_{k-2} + t^{c^*}k[[t]] \\ &= k + kG_0 + kG_0\,G_1 + ... + kG_0\,G_1...G_{k-2} + t^{c^*}k[[t]] \end{aligned} \tag{6.3}$$

where $G_i \equiv F_i \pmod{t^{c^*}}$, $0 < i < k$. We need to determine G_i's to construct the Arf closure and we need to know the term that we should stop while doing the divisions $\frac{\varphi_j^{(i-1)}(t)}{F_{i-1}}$. We should choose such a power that we do not lose any information about the Arf closure in the subsequent steps. To explain what we mean exactly, consider following example:

Example 6.1.3 $\Theta_0 = k[[t^6 + t^{11} + t^{32}, t^8 + t^{13} + t^{34}]]$. *Let's construct the Arf closure of* Θ_0 .

$$\varphi_1^{(0)}(t) = t^6 + t^{11} + t^{32}, \quad \varphi_2^{(0)}(t) = t^8 + t^{13} + t^{34} \longrightarrow F_0 = \varphi_1^{(0)}(t) = t^6 + t^{11} + t^{32}$$

$$\varphi_1^{(1)}(t) = \varphi_1^{(0)}(t) = t^6 + t^{11} + t^{32}, \quad \varphi_2^{(1)}(t) = \frac{\varphi_2^{(0)}(t)}{F_0} = t^2 \longrightarrow F_1 = \varphi_2^{(1)}(t) = t^2$$

$$\varphi_1^{(2)}(t) = \frac{\varphi_1^{(1)}(t)}{F_1} = t^4 + t^9 + t^{30}, \quad \varphi_2^{(2)}(t) = \varphi_2^{(1)}(t) = t^2 \longrightarrow F_2 = \varphi_2^{(2)}(t) = t^2$$

$$\varphi_1^{(3)}(t) = \frac{\varphi_1^{(2)}(t)}{F_2} = t^2 + t^7 + t^{28}, \quad \varphi_2^{(3)}(t) = \varphi_2^{(2)}(t) = t^2 \longrightarrow F_3 = \varphi_2^{(3)}(t) = t^2$$

$$\varphi_1^{(4)}(t) = \frac{\varphi_1^{(3)}(t)}{F_3} - 1 = t^5 + t^{26}, \quad \varphi_2^{(4)}(t) = \varphi_2^{(3)}(t) = t^2 \longrightarrow F_4 = \varphi_2^{(4)}(t) = t^2$$

$$\varphi_1^{(5)}(t) = \frac{\varphi_1^{(4)}(t)}{F_4} = t^3 + t^{24}, \quad \varphi_2^{(5)}(t) = \varphi_2^{(4)}(t) = t^2 \longrightarrow F_5 = \varphi_2^{(5)}(t) = t^2$$

$$\varphi_1^{(6)}(t) = \frac{\varphi_1^{(5)}(t)}{F_5} = t + t^{22}, \quad \varphi_2^{(6)}(t) = \varphi_2^{(5)}(t) = t^2 \longrightarrow F_6 = \varphi_1^{(6)}(t) = t + t^{22}$$

Arf closure is,

$$\begin{aligned}
\Theta^* &= k + k(t^6 + t^{11} + t^{32}) + k(t^6 + t^{11} + t^{32})(t^2) + k(t^6 + t^{11} + t^{32})(t^2)(t^2) + \\
&\quad k(t^6 + t^{11} + t^{32})(t^2)(t^2)(t^2) + k(t^6 + t^{11} + t^{32})(t^2)(t^2)(t^2)(t^2) + (t^6 + t^{11} \\
&\quad + t^{32})(t^2)(t^2)(t^2)(t^2)(t^2)k[[t]] \\
&= k + k(t^6 + t^{11}) + k(t^8 + t^{13}) + k(t^{10} + t^{15}) + k(t^{12}) + k(t^{14}) + t^{16}k[[t]]
\end{aligned}$$

If we had done our divisions till the power 9,

$$\varphi_1^{(0)}(t) = t^6 + t^{11} + t^{32}, \quad \varphi_2^{(0)}(t) = t^8 + t^{13} + t^{34} \longrightarrow F_0 = \varphi_1^{(0)}(t) = t^6 + t^{11} + t^{32}$$

$$\varphi_1^{(1)}(t) = \varphi_1^{(0)}(t) = t^6 + t^{11} + t^{32}, \quad \varphi_2^{(1)}(t) = \frac{\varphi_2^{(0)}(t)}{F_0} = t^2 \longrightarrow F_1 = \varphi_2^{(1)}(t) = t^2$$

$$\varphi_1^{(2)}(t) = \frac{\varphi_1^{(1)}(t)}{F_1} (mod\, t^9) \equiv t^4, \quad \varphi_2^{(2)}(t) = \varphi_2^{(1)}(t) = t^2 \longrightarrow F_2 = \varphi_2^{(2)}(t) = t^2$$

$$\varphi_1^{(3)}(t) = \frac{\varphi_1^{(2)}(t)}{F_2} = t^2, \quad \varphi_2^{(3)}(t) = \varphi_2^{(2)}(t) = t^2 \longrightarrow F_3 = \varphi_2^{(3)}(t) = t^2$$

Hence, we wouldn't be able to construct the Arf Closure correctly. We would face with this problem, because we would lose 't^{11}', which takes part in the algorithm at step 4 as 't^5'. Observe that step 4 *is the step that the constant c_{14} is different than* 1*. We can also choose* 12 *instead of* 9 *as the bound that determines, where the divisions stop. Indeed,*

$$\begin{array}{lll}
\varphi_1^{(0)}(t) = t^6 + t^{11} + t^{32}, & \varphi_2^{(0)}(t) = t^8 + t^{13} + t^{34} & \longrightarrow\ F_0 = \varphi_1^{(0)}(t) = t^6 + t^{11} + t^{32} \\
\varphi_1^{(1)}(t) = \varphi_1^{(0)}(t) = t^6 + t^{11} + t^{32}, & \varphi_2^{(1)}(t) = \frac{\varphi_2^{(0)}(t)}{F_0} = t^2 & \longrightarrow\ F_1 = \varphi_2^{(1)}(t) = t^2 \\
\varphi_1^{(2)}(t) = \frac{\varphi_1^{(1)}(t)}{F_1}(mod\, t^{12}) \equiv t^4 + t^9, & \varphi_2^{(2)}(t) = \varphi_2^{(1)}(t) = t^2 & \longrightarrow\ F_2 = \varphi_2^{(2)}(t) = t^2 \\
\varphi_1^{(3)}(t) = \frac{\varphi_1^{(2)}(t)}{F_2} = t^2 + t^7, & \varphi_2^{(3)}(t) = \varphi_2^{(2)}(t) = t^2 & \longrightarrow\ F_3 = \varphi_2^{(3)}(t) = t^2 \\
\varphi_1^{(4)}(t) = \frac{\varphi_1^{(3)}(t)}{F_3} - 1 = t^5, & \varphi_2^{(4)}(t) = \varphi_2^{(3)}(t) = t^2 & \longrightarrow\ F_4 = \varphi_2^{(4)}(t) = t^2 \\
\varphi_1^{(5)}(t) = \frac{\varphi_1^{(4)}(t)}{F_4} = t^3, & \varphi_2^{(5)}(t) = \varphi_2^{(4)}(t) = t^2 & \longrightarrow\ F_5 = \varphi_2^{(5)}(t) = t^2 \\
\varphi_1^{(6)}(t) = \frac{\varphi_1^{(5)}(t)}{F_5} = t, & \varphi_2^{(6)}(t) = \varphi_2^{(5)}(t) = t^2 & \longrightarrow\ F_6 = \varphi_1^{(6)}(t) = t
\end{array}$$

In this case, the Arf Closure is;

$$k + k(t^6 + t^{11}) + k(t^8 + t^{13}) + k(t^{10} + t^{15}) + k(t^{12}) + k(t^{14}) + k[[t]](t^{16})$$

which is the same ring as Θ^*. In spite of the fact that the term 't^{30}' is lost, we still obtain the same result. So, we can say that 't^{11}' is playing an important role, while 't^{30}' is not.

As we have seen from the examples, we should choose such a bound that no important terms are lost, if the divisions are done by using this bound.

6.2 Solution

Let $\Theta = k[[\varphi_1^{(0)}(t), \varphi_2^{(0)}(t), ..., \varphi_s^{(0)}(t)]]$ be a branch, where $\varphi_i^{(0)}(t)$'s, $(1 \leq i \leq s)$ are polynomials in t with increasing order. Let S be the semigroup of Θ and c be the conductor of S. Let Θ^* be the Arf closure of Θ, S^* semigroup of Θ^*, and c^* conductor of S^*. And let $\overline{\varphi_i}$ be the series obtained by truncation of the series φ_i by mod(t^{c^*+1}). We can give our main theorem and its proof, which actually says that the bound '$c^* + 1$' works.

Theorem 6.2.1 *Let $\Theta = k[[\varphi_1^{(0)}(t), \varphi_2^{(0)}(t), ..., \varphi_s^{(0)}(t)]]$ be the local ring corresponding to a branch and c^* be the conductor of the semigroup of values of its Arf closure Θ^*. If we have $\varphi_i^{(0)}(t) \equiv \overline{\varphi}_i^{(0)}(t)$ (mod t^{c^*+1}) for $1 \le i \le s$, then the $k[[\varphi_1^{(0)}(t), \varphi_2^{(0)}(t), ..., \varphi_s^{(0)}(t)]]$ and $k[[\overline{\varphi}_1^{(0)}(t), \overline{\varphi}_2^{(0)}(t), ..., \overline{\varphi}_s^{(0)}(t)]]$ have the same Arf Closure.*

Proof. As $\Theta^* = k + kF_0 + kF_0F_1 + ... + kF_0F_1...F_{k-2} + t^{c^*}k[[t]]$, its semigroup of values is $W(\Theta^*) = \{0, a_0, a_0 + a_1, ..., a_0 + ... + a_{k-2}, c^* + \mathbb{N}\}$ and $c^* = a_0 + a_1 + ... + a_{k-1}$, where here $a_i = ord(F_i)$. We are going to show that, while finding the Arf Closure of $k[[\varphi_1^{(0)}(t), \varphi_2^{(0)}(t), ..., \varphi_s^{(0)}(t)]]$, by doing divisions in mod (t^{c^*+1}), we don't lose any important monomials and obtain the exact Arf Closure. We are going to prove our claim in two parts by considering the constants 'c_{ij}'s. We will use the blow up schema 6.1 and the notation there. Before starting to prove our claim, observe that, if a column in the blow up schema 6.1 does not enter the algorithm, it means that the algorithm is the same without that column. Therefore, losing monomials in that column has no effect on the computation of the Arf closure. Hence, without loss of generalization, we assume that all of the columns enter the algorithm at least once. Using the schema 6.1 again, this is equivalent to saying that for all i, $\varphi_i^{(j)} = F_j$ for at least one j.

We first consider the case with zero constants. In other words, recalling Equation 6.2, $c_{ij} = 0$ for all i, j, so that all the important monomials are the smallest ordered terms of $\varphi_i^{(0)}$'s.

In this situation, for all i, $ord(\varphi_i^{(0)}) \ge ord(\varphi_i^{(1)}) \ge ... \ge ord(\varphi_i^{(k)})$ and,

$$\varphi_i^{(j)}(t) = \varphi_i^{(k)}(t) \prod_{l \in I_{ij}} F_l \text{ where } I_{ij} = \left\{ l : j \le l < k \text{ and } \mathrm{F_l} \neq \varphi_{\mathrm{i}}^{(\mathrm{l})}(\mathrm{t}) \right\}$$

Then, $ord(\varphi_i^{(0)}) = \sum_{l \in I_{i0}} a_l + ord(\varphi_i^{(k)}(t))$. We have observed that $\varphi_i^{(j)}(t) = F_j$ for some j. Therefore, $\sum_{l \in I_{i0}} a_l + \underbrace{\underbrace{ord(F_j)}_{a_j}}_{ord(\varphi_i^{(j)})} \le c^*$. Also, since $ord(\varphi_i^{(k)}) \le ord(\varphi_i^{(j)})$ for all $j < k$, we have $ord(\varphi_i^{(0)}) = \sum_{l \in I_{i0}} a_l + ord(\varphi_i^{(k)}) \le c^*$ for all i. So, dividing the series till the $(c^* + 1)$th power is enough to construct the Arf closure correctly.

If $c_{ij} \neq 0$ for some i and j, we do an induction on the number of j's for which $c_{ij} \neq 0$ for some i.

- $n = 1$ ($c_{ij} \neq 0$ for only one i and for some j)

$$\begin{array}{llllll}
\varphi_1^{(0)}(t), & \dots & \varphi_i^{(0)}(t), & \dots & \varphi_s^{(0)}(t), & \longrightarrow \quad F_0 \\
\vdots & & \vdots & & \vdots & \qquad\;\; \vdots \\
\varphi_1^{(j)}(t), & \dots & \varphi_i^{(j)}(t) = \frac{\varphi_i^{(j-1)}(t)}{F_{j-1}} - c_{ij}, & \dots & \varphi_s^{(j)}(t), & \longrightarrow \quad F_j \\
\vdots & & \vdots & & \vdots & \qquad\;\; \vdots \\
\varphi_1^{(k)}(t), & \dots & \varphi_j^{(k)}(t), & \dots & \varphi_s^{(k)}(t), & \longrightarrow \quad F_k = t + ...
\end{array}$$

Let

$$\varphi_i^{(j)}(t) = b_1 t^{\alpha_1} + b_2 t^{\alpha_2} + \dots = \frac{\varphi_i^{(j-1)}(t)}{F_{j-1}} - c_{ij}, \ (\alpha_1 < \alpha_2). \tag{6.4}$$

It's enough to show that, we don't lose the term t^{α_1} in the steps $0, 1, ..., j-1$ by doing the divisions till the $(c^* + 1)$th power. The reason is that after the jth step, there are no nonzero c_{ij}'s and from the previous part, we know that $\alpha_1 < a_j + a_{j+1} + \dots + a_k < c^*$. (Note that, without loss of generalization, we can assume that the ith column enters the algorithm at least once after the jth step.)

Hence, by Equation (6.4),

$$\varphi_i^{(j-1)}(t) = (c_{ij} + b_1 t^{\alpha_1} + b_2 t^{\alpha_2} + \dots) F_{j-1}$$

and

$$\varphi_i^{(0)}(t) = (c_{ij} + b_1 t^{\alpha_1} + b_2 t^{\alpha_2} + \dots) \prod_{l \in \Lambda_{i,0}} F_l$$

where $\Lambda_{i,0} = \left\{ l : 0 \leq l \leq j-1 \text{ and } \mathrm{F_l} \neq \varphi_{\mathrm{i}}^{(\mathrm{l})}(\mathrm{t}) \right\}$. As $\alpha_1 \leq a_j + \dots + a_{k-1}$ and $c^* = a_0 + ... + a_{j-1} + a_j + ... + a_{k-1}$, we can say that $\varphi_i^{(0)}(t) \bmod t^{c^*+1}$ contains the term, which gives t^{α_1} in the j-th step. This shows that doing the divisions till the (c^*+1)th power in the steps $0, 1, ..., j-1$, guarantees that the term t^{α_1} is obtained in the j-th step.

- Assume the claim is true for a branch having $c_{ij} \neq 0$ for $n-1$ different j's. We take any branch having $c_{ij} \neq 0$ for n different j's.

Let the first constant appears at the i_0-th column, j_0-th step. Then,

$$\begin{array}{llllll}
\varphi_1^{(0)}(t), & \dots & \varphi_i^{(0)}(t), & \dots & \varphi_s^{(0)}(t), & \longrightarrow \quad F_0 \\
\vdots & & \vdots & & \vdots & \qquad\;\; \vdots \\
\varphi_1^{(j_0)}(t), & \dots & \varphi_{i_0}^{(j_0)}(t) = \frac{\varphi_{i_0}^{(j_0-1)}(t)}{F_{j_0-1}} - c_{i_0 j_0}, & \dots & \varphi_s^{(j_0)}(t), & \longrightarrow \quad F_{j_0} \\
\vdots & & \vdots & & \vdots & \qquad\;\; \vdots \\
\varphi_1^{(k)}(t), & \dots & \varphi_j^{(k)}(t), & \dots & \varphi_s^{(k)}(t), & \longrightarrow \quad F_k = t + ...
\end{array}$$

From the induction assumption, for the steps starting with j_0th one, it is sufficient to make the divisions till the power $a_{j_0} + a_{j_0+1} + ... + a_{k-1}$, since that is the conductor of the Arf closure of the ring $k[[\varphi_1^{(j_0)}, \varphi_2^{(j_0)}, ..., \varphi_s^{(j_0)}]]$. In other words, all the important monomials that determine the Arf closure have orders less than or equal to $a_{j_0} + ... + a_{k-1}$ at j_0th step. Then, as in the first part of the induction hypothesis, since

$$\varphi_{i_0}^{(j_0)}(t) = b_1 t^{\alpha_1} + b_2 t^{\alpha_2} + \ldots = \frac{\varphi_{i_0}^{(j_0-1)}(t)}{F_{j_0-1}} - c_{i_0 j_0},\ (\alpha_1 < \alpha_2)\,, \tag{6.5}$$

we can write $\varphi_{i_0}^{(0)}$ as:

$$\varphi_{i_0}^{(0)}(t) = (c_{ij} + b_1 t^{\alpha_1} + b_2 t^{\alpha_2} + \ldots) \prod_{l \in \Lambda_{i_0,0}} F_l,$$

where $\Lambda_{i_0,0} = \left\{ l : 0 \leq l \leq j_0 - 1 \text{ and } \mathrm{F_l} \neq \varphi_{\mathrm{i_0}}^{(1)}(\mathrm{t}) \right\}$. Then all the important monomials in the first steps have order less than or equal to $ord(\prod_{l \in \Lambda_{i_0,0}} F_l) = a_0 + a_1 + ... + a_{j_0-1}$ plus $a_{j_0} + a_{j_0+1} + ... + a_{k-1}$, which is equal to c^*. Hence, by doing our divisions in mod t^{c^*+1}, we keep the important terms to construct the Arf closure correctly. ■

Remark 6.2.2 *We should observe that, Theorem 6.2.1 does not say that the parametrizations* $(\varphi_1^{(0)}(t), \varphi_2^{(0)}(t), ..., \varphi_s^{(0)}(t))$ *and* $(\overline{\varphi}_1^{(0)}(t), \overline{\varphi}_2^{(0)}(t), ..., \overline{\varphi}_s^{(0)}(t))$ *provide the same curve. We do not guarantee getting the same blow up rings by doing the divisions in mod(*t^{c^*+1}*). We only say that, we obtain the same Arf closure and multiplicity sequence.*

Now, we have a bound for dividing the series and determining the Arf Closure. It is definitely much better to use $c^* + 1$ than to use c, since mostly $c^* + 1$ is much smaller than c. ($\Theta \subset \Theta^*$ and $W(\Theta) \subset W(\Theta^*)$. Thus, $c^* \leq c$.) But, it looks like as if we are in a vicious circle: We want to determine the Arf Closure and the multiplicity sequence, but we need the conductor of the Arf closure for this. Hence, we ask the following question: 'Is there a way to find the conductor of the Arf Closure or to give a bound for it without knowing the Arf Closure?' We first investigate the plane branches, but before that let us state the next obvious remark for all branches.

Remark 6.2.3 *Let C be a branch given with the parametrization $(\varphi_1, ..., \varphi_n)$ and with the multiplicity sequence $a_0, a_1, ..., a_{k-1}, 1, 1....$ Then the conductor c^* of the Arf Closure of the ring $\Theta = k[[\varphi_1, ..., \varphi_n]]$ is equal to the sum $a_0 + a_1 + ... + a_{k-1}$*

Proof. We know from Theorem 5.4.7 that $W(\Theta^*) = \{0, a_0, a_0 + a_1, ..., a_0 + ... + a_{k-1} + \mathbb{N}\}$. Then, clearly $c^* = a_0 + a_1 + ... + a_{k-1}$. ■

6.2.1 Conductor of the Arf Closure for the Plane Algebroid Branches

Let C be a plane branch with characteristic exponents β_0,...,β_q.

Theorem 6.2.4 *Let Θ be the local ring corresponding to C, Θ^* be Arf closure of Θ and c^* be conductor of $W(\Theta^*)$. Then, $c^* = \beta_0 + \beta_q - 1$.*

Proof. We know from Theorem 6.2.3 that c^* is the sum of the multiplicities in the multiplicity sequence of C^* but Corollary 3.1.18 says that the sum of the multiplicities in the multiplicity sequence is equal to $\beta_0 + \beta_q - 1$. Hence, $c^* = \beta_0 + \beta_q - 1$. ■

As a consequence, for the plane case, if we know the characteristic exponents of the branch, we know the conductor of the Arf Closure, so we can determine the Arf closure by using our algorithm. The next step is finding a bound for c^* directly from the parameterization.

Let C be a plane branch given in Puiseux form $x(t) = t^n$, $y(t) = \sum a_i t^i$ with $n \leq ord(y(t))$, and let $(\beta_0, ..., \beta_q)$ be the characteristic exponents of C. From the definition of the characteristic exponents, $\beta_0 = n$ and β_q is the degree of one of the terms appearing in $y(t)$. If we start with polynomials $x(t)$ and $y(t)$, β_q is less than or equal to the highest power appearing in $y(t)$. Then we can obviously state the next theorem:

Theorem 6.2.5 *Let C be a plane curve branch given with parametrization $x(t) = t^n$, $y(t) = a_1 t^{m_1} + a_2 t^{m_2} + ... + a_r t^{m_r}$. Then the conductor of the Arf closure of $k[[x(t), y(t)]]$ is less than or equal to $n + m_r - 1$*

As a consequence, for the branches given with the parametrization $x(t) = t^n$, $y(t) = a_1 t^{m_1} + a_2 t^{m_2} + ... + a_r t^{m_r}$, while constructing the Arf Closure, we can do our divisions till the power $n + m_r$.

Now, if we have a plane branch given in Puiseux form, we can obtain our bound by just considering the powers appearing in the series. For the parameterizations in standard form,

$$\begin{aligned} x(t) &= a_0 t^p + a_1 t^{p+1} + a_2 t^{p+2} + ... \\ y(t) &= b_0 t^q + b_1 t^{q+1} + b_2 t^{q+2} + ... \end{aligned}$$

we need a method for converting it to the Puiseux form:

$$\begin{aligned} x(t) &= t^p \\ y(t) &= c_q t^q + c_{q+1} t^{q+1} + ... \end{aligned}$$

In general, this process involves finding a formula for t in terms of x, but this requires a lot of computation. Hence, we present here an algorithm given by Borodzik in [6]. The Algorithm works for $k = \mathbb{C}$.

Theorem 6.2.6 *[6] Let C be a plane branch with parametrization*

$$\begin{aligned} x(t) &= a_0 t^p + a_1 t^{p+1} + a_2 t^{p+2} + ... \\ y(t) &= b_0 t^q + b_1 t^{q+1} + b_2 t^{q+2} + ... \end{aligned}$$

in $\mathbb{C}[[t]]$. Define

$$P_0(t) = y, \quad r_0 = ord(P_0(t)) = q$$

and $\dot{y}$ denoting $\frac{dy}{dt}$, $\dot{x}$ denoting $\frac{dx}{dt}$,

$$P_1(t) = \dot{y}x - \frac{q}{p}\dot{x}y, \quad r_1 = ord(P_1(t)) - (p-1)$$

and for $k \geq 1$,

$$P_{k+1}(t) = x\dot{x}\frac{d}{dt}P_k - \left(\frac{r_k}{p}\dot{x}^2 + (2k-1)\ddot{x}x\right)P_k, \quad r_{k+1} = ord(P_{k+1}) - (2k+1)(p-1).$$

The next algorithm gives the positive integers $(r_0, r_1, ..., r_n)$ such that

$$\begin{aligned} x(t) &= t^p \\ y(t) &= c_{r_0} t^{r_0} + c_{r_1} t^{r_1} + ... + c_{r_n} t^{r_n} \end{aligned}$$

is a Puiseux Expansion of C.

Algorithm 4 BORODZIK

Input: $(x(t), y(t)) \in \mathbb{C}[[t]]$ primitive parametrization for the branch C.

Output: The positive integers $(r_0, r_1, ..., r_n)$ such that

$$\begin{aligned} x(t) = &\quad t^p \\ y(t) = &\quad c_{r_0}t^{r_0} + c_{r_1}t^{r_1} + ... + c_{r_n}t^{r_n} \end{aligned}$$

is a Puiseux Expansion of C.

```
1:  $p_0 \leftarrow p$ and $r_0 \leftarrow q$. Let $p_1 = gcd(p_0, q_0)$
2:  **if** $p_1 = 0$ **then**
3:      we stop and end the algorithm
4:  **end if**
5:  Compute $P_1(t) = \dot{y}x - \frac{q}{p}\dot{x}y$ and $r_1 = ord(P_1(t)) - (p-1)$. Let $p_2 = gcd(p_1, r_1)$.
6:  **if** $p_2 = 1$ **then**
7:      we stop and end the algorithm
8:  **else**
9:      We put $k = 1$
10:     Compute $P_{k+1}(t) = x\dot{x}\frac{d}{dt}P_k - \left(\frac{r_k}{p}\dot{x}^2 + (2k-1)\ddot{x}x\right)P_k$ and $r_{k+1} = ord(P_{k+1}) - (2k+1)(p-1)$.
11:     Let $p_{k+1} = gcd(p_k, r_k)$
12:     **if** $p_{k+1} = 1$ **then**
13:         we stop and end the algorithm
14:     **else**
15:         we increase $k$ and go to step 10
16:     **end if**
17: **end if**
```

Hence, when we start with a plane branch in parametric form with coefficient field $\mathbb{C}$, we can determine the Puiseux expansion with the Algorithm 4. Then by taking the smallest power appearing in the series plus highest power appearing in the series as a bound for dividing the series, we can construct the Arf Closure of the branch C.

Note that, you can find the SINGULAR procedure, we have written to find the Puiseux expansion by using the Algorithm 4 in Appendix B.

6.2.2 A Bound for the Conductor of the Arf Closure for Space Branches

For the plane branches, we have determined a bound for c^*. This bound is equal to smallest characteristic exponent plus highest characteristic exponent minus one. But characteristic exponents are not defined for the space branches. Hence, this idea can not be used in higher dimensions. Instead, our idea is finding a bound for c^* by using our result for plane branches. Let C be a space branch, for which the corresponding local ring is Θ. If we can find a plane branch $\tilde{C}$ with $\tilde{\Theta}$ corresponding local ring such that $\tilde{\Theta} \subset \Theta$, then we will have

$$\begin{aligned}\tilde{\Theta} \subset \Theta \quad &\Rightarrow \quad \tilde{\Theta}^* \subset \Theta^* \\ &\Rightarrow \quad \tilde{S}^* \subset S^* \\ &\Rightarrow \quad \tilde{c}^* \geq c^*.\end{aligned}$$

As a consequence, while constructing the Arf Closure of Θ, instead of using the bound $c^* + 1$ to divide the series, we can use $\tilde{c}^* + 1$ which is greater than or equal to $c^* + 1$ and we don't lose any information. Therefore, the next thing to do is, finding a method to construct $\tilde{C}$ and $\tilde{\Theta}$.

Let $\Theta = k[[\varphi_1(t), \varphi_2(t), ..., \varphi_s(t)]]$ be a branch, where

$$\begin{aligned}&\varphi_1(t) = t^{m_{11}} \\ &\varphi_2(t) = a_{21}t^{m_{21}} + a_{22}t^{m_{22}} + ... + a_{2r_2}t^{m_{2r_2}} \\ &\vdots \\ &\varphi_s(t) = a_{s1}t^{m_{s1}} + a_{s2}t^{m_{s2}} + ... + a_{sr_s}t^{m_{sr_s}}\end{aligned}$$

$m_{11} \leq m_{21} \leq .... \leq m_{s1}$ and $gcd(m_{11}, m_{21}, m_{22}, ..., m_{2r_2}, ..., m_{s1}, m_{s2}, ..., m_{sr_s}) = 1$. The first and naive idea that comes to mind is to take $\tilde{\Theta} = k[[\varphi_1(t), \varphi_{i_0}(t)]]$, where $\varphi_{i_0}(t) \in \{\varphi_2(t), ..., \varphi_s(t)\}$ and m_{ij}'s are as above. Unfortunately, finding such $\varphi_{i_0}(t)$ is not always possible, because

$$gcd(m_{11}, m_{21}, ..., m_{2r_2}, ..., m_{s1}, ..., m_{sr_s}) = 1 \nRightarrow gcd(m_{11}, m_{i_01}, ..., m_{i_0r_{i_0}}) = 1.$$

That is, the parametrization obtained in this way may not be primitive. We should choose $\tilde{\Theta}$ in such a way that the powers of the terms in its parametrization must be relatively prime.

For $\varphi(t) = \varphi_2(t) + ... + \varphi_s(t)$, the second idea is to take $\tilde{\Theta} = k[[\varphi_1(t), \varphi(t)]]$, but still we can not guarantee that the greatest common divisor of m_{11} and the powers of the terms of $\varphi(t)$ is one, because some of the m_{ij}'s may vanish while summing $\varphi_2(t), ..., \varphi_s(t)$. But we can always determine constants $b_2, ..., b_s$ such that after the addition $\varphi(t) = b_2\varphi_2(t) + b_3\varphi_3(t) + ... + b_s\varphi_s(t)$, none of the m_{ij}'s vanish and the greatest common divisor of the powers of the terms of $\varphi(t)$ and $\varphi_s(t)$ is equal to $gcd(m_{11}, ..., m_{s1}, m_{s2}, ..., m_{sr_s})$ which is equal to 1. Then we can give our next main theorem.

Theorem 6.2.7 *Let* $\Theta = k[[\varphi_1(t), \varphi_2(t), ..., \varphi_s(t)]]$ *be a branch, where*

$$\begin{aligned}
\varphi_1(t) &= t^{m_{11}} \\
\varphi_2(t) &= a_{21}t^{m_{21}} + a_{22}t^{m_{22}} + ... + a_{2r_2}t^{m_{2r_2}} \\
&\vdots \\
\varphi_s(t) &= a_{s1}t^{m_{s1}} + a_{s2}t^{m_{s2}} + ... + a_{sr_s}t^{m_{sr_s}}
\end{aligned}$$

$m_{11} \leq m_{21} \leq .... \leq m_{s1}$ *and* $gcd(m_{11}, m_{21}, m_{22}, ..., m_{2r_2}, ..., m_{s1}, m_{s2}, ..., m_{sr_s}) = 1$. *Using the bound* $m_{11} + m_{sr_s}$ *in algorithm 3 while dividing the series, we constuct the Arf Closure correctly.*

Proof. We have seen that, we can use $\tilde{c}^*$ instead of c^*. But we already know, $\tilde{c}^* \leq m_{11}+m_{sr_s}-1$ from Theorem 6.2.5. Then from Theorem 6.2.1, we can use $m_{11} + m_{sr_s}$ while dividing the series. ■

If we want to work with a general space branch, given with the parametrization ($\varphi_1(t)$,$\varphi_2(t)$, ..., $\varphi_s(t)$) with the coefficient field $\mathbb{C}$, where

$$\begin{aligned}
\varphi_1(t) &= a_{11}t^{m_{11}} + a_{12}t^{m_{12}} + ... + a_{1r_1}t^{m_{1r_1}} \\
&\vdots \\
\varphi_s(t) &= a_{s1}t^{m_{s1}} + a_{s2}t^{m_{s2}} + ... + a_{sr_s}t^{m_{sr_s}}
\end{aligned}$$

with $m_{11} \leq m_{21} \leq .... \leq m_{s1}$ and $gcd(m_{11}, m_{12}, ..., m_{1r_1}, ..., m_{s1}, m_{s2}, ..., m_{sr_s}) = 1$, again, we form the ring $\tilde{\Theta} = \mathbb{C}[[\varphi_1(t), \varphi(t)]]$, where $\varphi(t) = b_2\varphi_2(t) + b_3\varphi_3(t) + ... + b_s\varphi_s(t)$ (b_i's have been chosen in such a way that none of the m_{ij}'s vanish). Then we convert its parametrization to the Puiseux Form by using Algorithm 4 and use the bound smallest power appearing in the converted series plus the highest power appearing in the converted series.

CHAPTER 7

HILBERT FUNCTIONS OF ARF RINGS

In this section, we present a conjecture of Arslan and Sertoz and give many examples supporting this conjecture obtained by using the algorithm given by us. First, we describe the Hilbert function of an Arf ring.

Let R be a local ring with the maximal ideal $\mathfrak{m}$. The associated graded ring of R is defined to be the ring

$$gr_{\mathfrak{m}}(R) = \bigoplus_{i=0}^{\infty} \mathfrak{m}^i/\mathfrak{m}^{i+1}.$$

Geometrically, if R is a local ring of a variety V, then the associated graded ring of R is the coordinate ring of the tangent cone of V.

Theorem 7.0.8 *[36] Let R be a local Cohen-Macaulay ring of dimension d with the multiplicity e. Let embdim(R) denote the embedding dimension of R (the number of generators in the minimal basis of the maximal ideal of R). If*

$$embdim(R) = e - d + 1,$$

then the associated graded ring of R is also Cohen-Macaulay.

Theorem 7.0.9 *[33] Let R be a local ring and R^* its Arf closure. Then the multiplicity of R^* is equal to the embedding dimension of R^**

Corollary 7.0.10 *The associated graded rings of Arf rings are Cohen Macaulay.*

Proof. This is a direct consequence of Theorem 7.0.8 and 7.0.9, as $d = 1$ for Arf rings. ∎

The Hilbert function $H_R(n)$ of the local ring R is defined to be the Hilbert function of the associated graded ring $gr_{\mathfrak{m}}(R)$.

$$H_R(n) = H_{gr_{\mathfrak{m}}(R)}(n) = dim_{R/\mathfrak{m}}(\mathfrak{m}^n/\mathfrak{m}^{n+1}) \;\; n \geq 0$$

The Hilbert series of R is defined to be

$$P_R(t) = \sum_{n\in\mathbb{N}} H_R(n)t^n.$$

It has been proved by Hilbert and Serre that, $P_R(t) = \frac{h(t)}{(1-t)^d}$, where $h(t)$ is a polynomial with the coefficients from $\mathbb{Z}$ and d is the Krull dimension of R. Hence, in our case, we have $P_R(t) = \frac{h(t)}{1-t}$. The multiplicity e is equal to $h(1)$ and also $h(t) = 1 + (e-1)t + a_2t + ... + a_st^s$.

Theorem 7.0.11 *[25, Theorem 1.5] If R is a graded Cohen-Macaulay ring, coefficients of $h(t)$ are positive.*

As a consequence, we can state the next theorem.

Theorem 7.0.12 *Let R be a local ring and R^* its Arf Closure. Then the Hilbert series of R^* is:*

$$P_{R^*}(t) = \frac{1 + (e-1)t}{1-t}$$

Proof. As the associated graded ring of R^* is Cohen-Macaulay from Corollary 7.0.10, the coefficients of $h(t)$ are positive from Theorem 7.0.11 and from the remarks preceding Theorem 7.0.11, we have $a_2 + ... + a_s = 0$. This is only possible if we have $a_2 = a_3 = ... = a_s = 0$. Thus, $h(t) = 1 + (e-1)t$ and

$$P_{R^*}(t) = \frac{1 + (e-1)t}{1-t}.$$

■

7.1 Conjecture

Knowing the basic definitions and theorems, now we can state the conjecture due to Arslan and Sertoz:

If R_1 and R_2 are two local rings having the same Arf closure with $R_1 \subset R_2$ and $P_{R_1}(t) = \frac{h_1(t)}{1-t}$, $P_{R_2}(t) = \frac{h_2(t)}{1-t}$, then we have

$$\text{degree}(h_1) \geq \text{degree}(h_2).$$

Note that, this is not true in general for the local rings containing each other.

Example 7.1.1 *Consider the rings $R_1 = k[[t^{10}, t^{15}, t^{17}, t^{18}]]$ and $R_2 = k[[t^{10}, t^{11}, t^{15}, t^{17}, t^{18}]]$. $R_1 \subset R_2$, but $P_{R_1}(t) = \frac{1+3t+4t^2+2t^3}{1-t}$ and $P_{R_2}(t) = \frac{1+4t+4t^2+t^4}{1-t}$.*

Example 7.1.2 *Consider the rings $R_3 = k[[t^9, t^{10}, t^{15}, t^{17}, t^{18}]]$ and $R_4 = k[[t^7, t^{10}, t^{15}, t^{17}, t^{18}]]$ with $P_{R_3} = \frac{1+3t+3t^2+t^3+t^5}{1-t}$ and $P_{R_4} = \frac{1+3t+2t^2+t^4}{1-t}$. Comparing with R_1 in the previous example, we see that $R_1 \subset R_3$ but deg$(h_1) <$ deg(h_3). In the same way, $R_1 \subset R_4$ but deg$(h_1) <$ deg(h_4).*

The next tables give examples of rings having the same Arf closure and their Hilbert series. The Arf closure computations are done via our library "ArfClosure.lib", which you can find in Appendix B.

Table 7.1: Rings having the Arf Closure $k[[t^4 + t^6, t^9, t^{10}, t^{11}]]$

Rings with Arf Closure $k[[t^4 + t^6, t^9, t^{10}, t^{11}]]$	**Hilbert Series**
$k[[t^4 + t^6 + t^{12}, t^9]]$	$1 + t + t^2 + t^3$
$k[[t^4 + t^6 + t^{12}, t^9, t^{10}]]$	$1 + 2t + t^2$
$k[[t^4 + t^6 + t^{12}, t^9, t^{10}, t^{11}]]$	$1 + 3t$

Table 7.2: Rings having the Arf Closure $k[[t^6, t^8, t^{10}, t^{11}, t^{13}, t^{15}]]$

Rings with Arf Closure $k[[t^6, t^8, t^{10}, t^{11}, t^{13}, t^{15}]]$	**Hilbert Series**
$k[[t^6 + t^8, t^{10} + t^{11}]]$	$1 + 2t + 3t^2$
$k[[t^6, t^8, t^{10}, t^{11}]]$	$1 + 3t + 2t^2$
$k[[t^6, t^8, t^{10}, t^{11}, t^{13}]]$	$1 + 4t + t^2$
$k[[t^6, t^8, t^{10}, t^{11}, t^{13}]]$	$1 + 5t$

Table 7.3: Rings having the Arf Closure $k[[t^7, t^{23}, t^{25}, t^{27}, t^{29}, t^{31}, t^{33}]]$

Rings with Arf Closure $k[[t^7, t^{23}, t^{25}, t^{27}, t^{29}, t^{31}, t^{33}]]$	**Hilbert Series**
$k[[t^7, t^{23}]]$	$1 + t + t^2 + t^3 + t^4 + t^5 + t^6$
$k[[t^7, t^{23}, t^{25}]]$	$1 + 2t + 2t^2 + 2t^3$
$k[[t^7, t^{23}, t^{25}, t^{27}]]$	$1 + 3t + 3t^2$
$k[[t^7, t^{23}, t^{25}, t^{27}, t^{29}]]$	$1 + 4t + 2t^2$
$k[[t^7, t^{23}, t^{25}, t^{27}, t^{29}, t^{31}]]$	$1 + 5t + t^2$
$k[[t^7, t^{23}, t^{25}, t^{27}, t^{29}, t^{31}, t^{33}]]$	$1 + 6t$

Table 7.4: Rings with Arf Closure $k[[t^{12}, t^{18}, t^{25}, t^{26}, t^{27}, t^{28}, t^{29}, t^{31}, t^{32}, t^{33}, t^{34}, t^{35}]]$

Rings with Arf Closure $k[[t^{12}, t^{18}, t^{25}, t^{26}, t^{27}, t^{28}, t^{29}, t^{31}, t^{32}, t^{33}, t^{34}, t^{35}]]$	**Hilbert Series**
$k[[t^{12}, t^{18}, t^{25}, t^{26}]]$	$1 + 3t + 4t^2 + 3t^3 + t^4$
$k[[t^{12}, t^{18}, t^{25}, t^{26}, t^{27}]]$	$1 + 4t + 5t^2 + 2t^3$
$k[[t^{12}, t^{18}, t^{25}, t^{26}, t^{27}, t^{28}]]$	$1 + 5t + 5t^2 + t^3$
$k[[t^{12}, t^{18}, t^{25}, t^{26}, t^{27}, t^{28}, t^{29}]]$	$1 + 6t + 5t^2$
$k[[t^{12}, t^{18}, t^{25}, t^{26}, t^{27}, t^{28}, t^{29}, t^{31}]]$	$1 + 7t + 4t^2$
$k[[t^{12}, t^{18}, t^{25}, t^{26}, t^{27}, t^{28}, t^{29}, t^{31}, t^{32}]]$	$1 + 8t + 3t^2$
$k[[t^{12}, t^{18}, t^{25}, t^{26}, t^{27}, t^{28}, t^{29}, t^{31}, t^{32}, t^{33}]]$	$1 + 9t + 2t^2$
$k[[t^{12}, t^{18}, t^{25}, t^{26}, t^{27}, t^{28}, t^{29}, t^{31}, t^{32}, t^{33}, t^{34}]]$	$1 + 10t + t^2$
$k[[t^{12}, t^{18}, t^{25}, t^{26}, t^{27}, t^{28}, t^{29}, t^{31}, t^{32}, t^{33}, t^{34}, t^{35}]]$	$1 + 11t$

Table 7.5: Rings having the Arf Closure $k[[t^6, t^{15}, t^{19}, t^{20}, t^{22}, t^{23}]]$

Rings with Arf Closure $k[[t^6, t^{15}, t^{19}, t^{20}, t^{22}, t^{23}]]$	**Hilbert Series**
$k[[t^6, t^{15}, t^{19}]]$	$1 + 2t + 2t^2 + t^3$
$k[[t^6, t^{15}, t^{19}, t^{20}]]$	$1 + 3t + 2t^2$
$k[[t^6, t^{15}, t^{19}, t^{20}, t^{22}]]$	$1 + 4t + t^2$
$k[[t^6, t^{15}, t^{19}, t^{20}, t^{22}, t^{23}]]$	$1 + 5t$

Table 7.6: Rings with Arf Cl. $k[[t^{14}, t^{21}, t^{30}, t^{32}, t^{34}, t^{36}, t^{37}, t^{38}, t^{39}, t^{40}, t^{41}, t^{43}, t^{45}, t^{47}]]$

Rings	**Hilbert Series**
$k[[t^{14}, t^{21}, t^{30}]]$	$1 + 2t + 2t^2 + 2t^3 + 2t^4 + 2t^5 + 2t^6 + t^7$
$k[[t^{14}, t^{21}, t^{30}, t^{32}]]$	$1 + 3t + 4t^2 + 4t^3 + 2t^4$
$k[[t^{14}, t^{21}, t^{30}, t^{32}, t^{34}]]$	$1 + 4t + 6t^2 + 3t^3$

Table 7.7: Rings with Arf Closure $k[[t^{12}, t^{18}, t^{21}, t^{25}, t^{26}, t^{27}, t^{28}, t^{29}, t^{31}, t^{32}, t^{34}, t^{35}]]$

Rings with Arf Closure $k[[t^{12}, t^{18}, t^{21}, t^{25}, t^{26}, t^{27}, t^{28}, t^{29}, t^{31}, t^{32}, t^{34}, t^{35}]]$	**Hilbert Series**
$k[[t^{12}, t^{18}, t^{21}, t^{25}]]$	$1 + 3t + 4t^2 + 3t^3 + t^4$
$k[[t^{12}, t^{18}, t^{21}, t^{25}, t^{26}]]$	$1 + 4t + 6t^2 + t^3$
$k[[t^{12}, t^{18}, t^{21}, t^{25}, t^{26}, t^{27}]]$	$1 + 5t + 6t^2$
$k[[t^{12}, t^{18}, t^{21}, t^{25}, t^{26}, t^{27}, t^{28}]]$	$1 + 6t + 5t^2$
$k[[t^{12}, t^{18}, t^{21}, t^{25}, t^{26}, t^{27}, t^{28}, t^{29}]]$	$1 + 7t + 4t^2$
$k[[t^{12}, t^{18}, t^{21}, t^{25}, t^{26}, t^{27}, t^{28}, t^{29}, t^{31}]]$	$1 + 8t + 3t^2$
$k[[t^{12}, t^{18}, t^{21}, t^{25}, t^{26}, t^{27}, t^{28}, t^{29}, t^{31}, t^{32}]]$	$1 + 9t + 2t^2$
$k[[t^{12}, t^{18}, t^{21}, t^{25}, t^{26}, t^{27}, t^{28}, t^{29}, t^{31}, t^{32}, t^{34}]]$	$1 + 10t + t^2$
$k[[t^{12}, t^{18}, t^{21}, t^{25}, t^{26}, t^{27}, t^{28}, t^{29}, t^{31}, t^{32}, t^{34}, t^{35}]]$	$1 + 11t$

Table 7.8: Rings having the Arf Closure $k[[t^{10}, t^{15}, t^{28}, t^{31}, t^{32}, t^{33}, t^{34}, t^{36}, t^{37}, t^{39}]]$

Rings with Arf Closure $k[[t^{10}, t^{15}, t^{28}, t^{31}, t^{32}, t^{33}, t^{34}, t^{36}, t^{37}, t^{39}]]$	**Hilbert Series**
$k[[t^{10}, t^{15}, t^{28}]]$	$1 + 2t + 2t^2 + 2t^3 + 2t^4 + t^5$
$k[[t^{10}, t^{15}, t^{28}, t^{31}]]$	$1 + 3t + 4t^2 + 2t^3$
$k[[t^{10}, t^{15}, t^{28}, t^{31}, t^{32}]]$	$1 + 4t + 5t^2$
$k[[t^{10}, t^{15}, t^{28}, t^{31}, t^{32}, t^{33}]]$	$1 + 5t + 4t^2$
$k[[t^{10}, t^{15}, t^{28}, t^{31}, t^{32}, t^{33}, t^{34}, t^{36}, t^{37}, t^{39}]]$	$1 + 9t$

Table 7.9: Rings with Arf Cl. $k[[t^{12}, t^{16} + t^{30}, t^{20}, t^{31}, t^{33}, t^{34}, t^{35}, t^{37}, t^{38}, t^{39}, t^{41}, t^{42}]]$

Rings	**Hilbert Series**
$k[[t^{12}, t^{16} + t^{30}, t^{31}]]$	$1 + 2t + 2t^2 + 2t^3 + 2t^4 + 2t^5 + t^6$
$k[[t^{12}, t^{16} + t^{30}, t^{20}, t^{31}]]$	$1 + 3t + 5t^2 + 3t^3$
$k[[t^{12}, t^{16} + t^{30}, t^{20}, t^{31}, t^{33}]]$	$1 + 4t + 7t^2$
$k[[t^{12}, t^{16} + t^{30}, t^{20}, t^{31}, t^{33}, t^{34}, t^{35}]]$	$1 + 6t + 5t^2$
$k[[t^{12}, t^{16} + t^{30}, t^{20}, t^{31}, t^{33}, t^{34}, t^{35}, t^{37}, t^{38}, t^{39}, t^{41}, t^{42}]]$	$1 + 11t$

Observe that in each table, a ring in an arbitrary row is contained in the rings appearing in the rows following that row. However, degrees of the corresponding Hilbert series is nondecreasing as the rings are getting closer to the Arf Closure. Hence, all these examples support the conjecture due to Arslan an Sertoz.

CHAPTER 8

RESOLUTION AND PARAMETRIZATION OF REDUCIBLE CURVES GIVEN IN CLOSED FORM

Most of the computer algebra systems are working with the reducible curves given in closed form, so does the computer algebra system SINGULAR [37]. The SINGULAR library hnoether.lib [34] computes the Hamburger Noether expansion of a reduced curve singularity according to the theory in [10] by Campillo, but starting from the defining polynomial $f \in k[x, y]$. The library also contains procedures for computing the invariants of the plane curve singularities from its Hamburger Noether Expansion. Unfortunately, there wasn't any released library or procedure that computes the invariants of a curve branch given in parametric form. The existing libraries work with only plane branches not with the space ones. Finding the semigroup of values of a branch and its conductor from the parametrization may also be useful for Arf Closure computations but the library hnoether.lib only works with plane algebroid curves. Moreover, its input is defining ideal, not the parametrization. We have been informed by Professor Gerhard Pfister that there has been another library called "space-curve.lib", which has been written by Maryna Viazovska with the guidance of Professor Gerhard Pfister as a master project [39] but has never been released as a SINGULAR library. The library was not totally completed and it contained the following procedures:

Blowingup(f,I,l);	Blowing up of V(I) at the point 0;
CurveRes(I);	Resolution of V(I);
Curveparam(I);	Parametrization of algebraic branches of $V(I)$;
WSemigroup(X,b);	Weirstrass semigroup of the curve

The procedures in the library work in all dimensions. Moreover, the procedure *curveparam(I)* gives a bound for the conductors of the branches using the multiplicity sequences.

Professor Pfister offered me to check the library and complete it by adding some new procedures that compute the invariants of the branches in the plane case. The first problem in the library that should be solved was that the existing procedure CurveRes(I) in library ”spacecurve.lib” starts with the defining ideal and it gives the sum of the multiplicities of the branches, not the multiplicities corresponding to each branch. This causes a big problem, because it is not possible to obtain the invariants of the algebroid curve and also the bound obtained for the conductor is very large, as it uses the sum of the multiplicities. That bound is not practical for dividing the series. What we need was to find the multiplicities of each branch to get a good bound for the conductor. We have decided to solve this problem by using Hamburger-Noether matrices. The library ”spacecurve.lib” has already contained the procedure $curveparam(I)$ that computes the parameterizations of the branches of the algebroid curve from the defining ideal. Our idea was to compute the parameterizations by using this procedure and then computing the Hamburger-Noether matrices by using those parameterizations applying the algorithm 1 in chapter 4. There was not a procedure doing this in the literature, so I have written a procedure doing all these. Once we obtain the Hamburger-Noether matrix all the invariants can be computed by using the library ”alexpoly.lib” in SINGULAR [30].

After writing well working procedures to compute the invariants, the next step was to make the library faster from the other libraries. Checking some examples, we have noticed that, both ”hnoether.lib” and ”spacecurve.lib” work faster with the irreducible curves and slower with the reducible ones. If we had a chance to factorize the defining polynomial of the reducible curve in the power series ring, the both libraries could work faster. Unfortunately, there is no way of doing this without expensive computations yet. Professor Pfister had the idea that if we could factorize the defining polynomial in polynomial ring, we would have a faster algorithm. Binyamin had given examples which show that factorizing makes the algorithm faster in the library ”hnoether.lib” [5]. In [5], Binyamin also gives an algorithm that computes the contact numbers from the multiplicity sequence and the intersection matrix. We also use this algorithm while computing contact numbers in our procedures.

We now give a brief summary of the theory behind the ”spacecurve.lib” and our contributions to this library. You can find the whole library and the procedures in Appendix A.

8.1 Algorithms for the library ”spacecurve.lib”

Before starting to explain the algorithms that are used for the procedures in ”spacecurve.lib”, let us give the notations first.

For any polynomial $f \in k[x_1, ..., x_n]$, $f^{(in)}$ denotes the 'initial polynomial' of f. That is, homogeneous part of f of smallest degree. $I^{(in)}$ denotes the homogeneous ideal generated by $f^{(in)}$'s for all $f \in I$.

The next Lemma constitutes the main idea of the algorithm ExceptHypersurface that computes the homogeneous ideal of the exceptional hypersurface.

Lemma 8.1.1 *[39, Lemma 2.2.2] The exceptional hypersurface of the blow-up of an affine variety $X \subset \mathbb{A}^n$ in the origin is $V_p(I(X)^{(in)})$.*

Let C be a curve branch, $\tilde{C}$ its blow-up and E exceptional divisor.

Theorem 8.1.2 *[39, Theorem 2.3.1] There is one to one correspondence between the points of exceptional divisor $\tilde{C} \cap E$ and the first rows of Hamburger-Noether expansion of C.*

Algorithm 5 ExceptHypersurface(I)

Input: Defining ideal I of the curve.

Output: Homogeneous ideal J of the exceptional hypersurface

```
1. Compute a standard basis $f_1, f_2,...,f_k$ for $I$.
2. Compute the ideal $J =< f_1^{(in)}, ..., f_k^{(in)} >$.
3. Return($J$)
```

To compute the equations of the strict transform in each chart, we set $\tilde{C}_i = \tilde{C} \cap U_i$. Let ψ be the ring homomorphism with $\psi(x_i) = x_i$ and $\psi(x_j) = x_i x_j$ for $j \neq i$. Then $V(I_i) = \tilde{C}_i \cup E_i$. Setting S to the set of all positive powers of x_i, we define the ideal $\tilde{I}_i = S^{-1}I_i \cup k[x_1, ..., x_n]$. Then it can be shown that $V(\tilde{I}_i) = \tilde{C}_i$ implying $I(\tilde{C}_i) = \tilde{I}_i$. You can check [39] for all of the proofs. The next algorithm computes the full vanishing ideal of the strict transform in the ith affine chart.

Algorithm 6 STRICTTransform(I,i)

Input: Defining ideal I of the curve

Output: Ideal $\tilde{I}_i$ of the strict transform in the i-th affine chart.

```
1. Compute a standard basis $f_1, f_2, ..., f_k$ of $I$.
2. Define a map $\psi$ which maps $x_i$ to $x_j$ and $x_j$ to $x_i x_j$ for all $j \neq i$.
3. for $s = 1$ to $k$, do
4. poly $g_s := \psi(f_s)/x_i^{n_s}$ where $n_s = ord_{x_i}(f_s)$.
5. $\tilde{I}_i :=< g_1, ..., g_k >$
6. return($\tilde{I}_i$)
```

In practice, it not possible to work with an algebraically closed field. Hence it is assumed that the field is a finite extension $k[a]$, where a is a root of an irreducible polynomial with coefficients from k. During the algorithm we may need to enlarge our field with a new parameter b with the new minimal polynomial g such that $k[a] \subset k[b]$.

Remark 8.1.3 *[39, Remark 3.1.1] A point on exceptional divisor may exist in more than one affine chart. To consider the points of $E \cap U_{i_s}$ that do not lie in $E \cap U_{i_1}$,...,$E \cap U_{i_{s-1}}$, we add equations $< x_{i_1}, ..., x_{i_{s-1}} >$ to the ideal $J_{i_s} = J + < x_{i_s} - 1 >$ of exceptional divisor in i_s-th chart. In this way, each point of E is counted once.*

Remark 8.1.4 *[39, Remark 3.1.2] Let P be a prime zero dimensional ideal. Then P is an intersection of maximal ideals $P = \bigcap m_i$. Then for each i, $m_i = I(a_i)$ where $a_i = (a_{i1}, ..., a_{in})$. There is a finite field extension L such that $k \subset L \subset \overline{k}$ which contains coordinates of all points a_i. If k is a perfect field, there is a primitive element $\alpha \in L$, such that $L = k(\alpha)$. As L contains all a_{ij}, $a_{ij} = f_{ij}(\alpha)$ for some $f_{ij} \in k[x]$. Thus, we can write P as $< x_1 - f_1(\alpha), ..., x_n - f_n(\alpha), f(\alpha) >$. It is possible to do this for a perfect field of any characteristic but the procedures in "spacecurve.lib" does this for the case of characteristic zero.*

Algorithm 7 BLOWINGUp(f, I, l)

Input: Irreducible polynomial f, defining ideal I of the curve, list l of previous charts.

Output: List BlowUp. Each element BlowUp[i] of this list contains the the following information: new irreducible poly f_i, ideal of a new curve $\tilde{I}_i$ and a map ϕ_i from the new curve to the old one.

Algorithm 7 BLOWINGUp(f, I, l) (cont.)

```
1. List BlowUp:=$\emptyset$
2. Compute $J$ :=EXCEPTHypersurface($I$)
3. Compute the list $i_1$,...,$i_n$ where $i_j$ is the index of the smallest ordered element when doing the blowing up.
4.for $s = 1$ to $n$ do
5. $J_{i_s} := J+ < x_{i_1}, ..., x_{i_{s-1}}, x_{i_s} - 1 >$
    if $dim(J_{i_s}) \neq 0$ then break.
6. ideal $\tilde{I}_{i_s}$=StrictTransform($I, i_s$)
7. $\tau$ :=identity map
8. Compute primary decomposition for $RADICAL(J_{i_s}) = \bigcap_{j=1}^{k} P_{sj}$
9. for $j = 1$ to $k$, do
10. Present $P_{sj}$ at the form $P_{sj} =< x_1 - g_1(b), ..., x_n - g_n(b), g(b) >$
11. Define map $\theta$ to be a shift $x_i \to x_i + g_i(b)$
12. Define map $\phi := \tau o \theta$ and ideal $\tilde{I} := \theta(\tilde{I}_{i_s})$
13. $\tau = \tau o \theta$
14. $BlowUp$:=ADD($BlowUp$,list($g, \tilde{I}, \phi$))
15. return($BlowUp$)
```

As BLOWINGUp(f,I,l) makes one blowing up, we need another algorithm to do the blowing up successively.

Algorithm 8 CURVERes(I)

Input: Defining ideal I of the curve C

Output: List of resolutions $Resolve$. Each element $Resolve[i]$ consists of irreducible poly f_i, ideal of a smooth curve I'_i, map π_i from the new curve to the old one.

```
1. if $C$ is smooth then
2. return($f, I, id$)
3. else
4. compute list $BlowUp$ =BLOWINGUp($f, I, l$)
5. For each element of $BlowUp$, compute CURVERes.
```

The next two algorithms are for finding the parametrization of the curve. The first one is for the smooth curves, and the second one is for singular curves.

Algorithm 9 SMOOTHParam(I, N)

```
Input: ideal $I$ of a smooth curve and an integer $N$
Output: parametrization $x_1(t),...x_n(t)$ till the order $N$
1. Compute $M$ =JACOB($I$) at point 0.
2. Find $n-1$ linearly independent rows and columns of the jacobian matrix $M$ and define a submatrix $A$.
3. define $X_s := t$ and $X_j := 0$ for $j \neq s$
4. Compute $A^{-1}$
5. for $p = 1$ to $N$ do
6. vector $\vec{c} := (\frac{\partial f_{i_1}}{\partial x_s}, ..., \frac{\partial f_{i_{n-1}}}{\partial x_s})^T$
7. vector $\vec{b} := -A^{-1}\vec{c}$
8. Define map $\tau$ which sends $x_s \to x_s$ and $x_j \to x_s(x_j + b_j)$ for $j \neq s$
9. ideal $I = \tau(I)/x_s$
10. $X_j := X_j + b_j.t^p$ for $j \neq s$
11.return($X_1(t), ..., X_n(t)$)
```

By using the algorithm 9, we define the next algorithm for the singular curves. The algorithm first desingularizes the curve by the algorithm 8, then uses the parametrization of the smooth curve we got from CURVERes.

Algorithm 10 CURVEParam(I, N)

```
Input: ideal $I$ of the curve and an integer $N$
Output: list of parameterizations till the order $N$ for all algebraic branches of the curve.
1. list $Resolve := CURVERes(I)$
2. for $i = 1$ to SIZE($Resolve$), do
3. $\mathcal{P}$ :=SMOOTHParam($I'_i, N$)
4. $\mathcal{P}'_i := \pi_i(\mathcal{P})$
5. return($\mathcal{P}'_1, ..., \mathcal{P}'_k$), where $k$ is a size of $Resolve$
```

The next and the last algorithm in [39] computes the Weierstrass semigroup of a curve.

Algorithm 11 WSemigroup(X,b)

Input: an integer b and polynomials $x_1, ..., x_n \in k[t]$ of degree less than or equal to b.

Output: Weierstrass semigroup of the curve C given by parametrization $x_1(t)$,...,$x_n(t)$

Denote $m_i = ord_t(x_i)$ the order of x_i with respect to a local ordering and assume that $m_1 \leq m_2 \leq ... \leq m_n$.

1. Compute the semigroup $\Gamma =< m_1, ..., m_n >$ till the bound b and the list L of length b, where $L[i]$ is a list of all nonnegative integer vectors $(\alpha_1, ..., \alpha_n)$ such that

$$\alpha_1 m_1 + ... + \alpha_n m_n = i$$

   Integer i is an element of S iff the list $L[i]$ is not empty. If semigroup S has a conductor $c \leq b$, then set $b := min(b, c + m_1)$

2. Compute a list N of length b, for which $N[i]$ is a list consisting of polynomials $x_1^{\alpha_1}...x_n^{\alpha_n}$ for all $(\alpha_1, ..., \alpha_n) \in L[i]$.

3. For each pair p_l, $p_k \in L[i]$, compute a polynomial $f := lc(p_l)p_k - lc(p_k)p_l$, where $lc(p)$ is a leading coefficient with respect to local ordering. Denote $m := ord_t(f)$, then $m > i$. Then m is either an element of S or not.

   - If $m \in \Gamma$ check if f is a linear combination of elements of $N[m]$. If it is not, add f to $N[i]$.
   - If $m \notin \Gamma$

   This process continues till all the pairs $p_k, p_l \in N[i]$ for all $i \leq b$ are considered.

To check the correctness of the algorithms, you can see [39].

8.2 The new library ”curvepar.lib”

We can now give the brief summary of the new library ”curvepar.lib” obtained by using the library ”spacecurve.lib” and by our procedures:
We start with the defining polynomial f of the curve ”C”.

1. We factorize f over $Q[x, y]$ for faster computations. Let f_1,...,f_k be the irreducible

factors of f in $Q[x, y]$.

2. For each f_i, compute $CURVEParam(< f_i >)$ which gives the parametrization for the branch corresponding to f_i.

3. Compute the Hamburger-Noether matrix of each branch by using the algorithm 1.

4. Use the procedures from "hnoether.lib" to compute the matrix of the intersection multiplicities of the branches.

5. Use the procedures from "hnoether.lib" to compute the characteristic exponents and the multiplicity sequences of the branches.

6. Compute the contact numbers using the multiplicity sequences and intersection multiplicities from Theorem 3.2.11 (The algorithm can be found in [5]).

7. Compute the matrix of the resolution graph of f using the contact numbers and characteristic exponents with the procedures in "alexpoly.lib".

We have written SINGULAR procedures doing all these. It is now in use under the name "curvepar.lib" in SINGULAR [18].

8.3 Timings

In this section, we give timing comparison of the already existing library "hnoether.lib" and our new library "curvepar.lib" in SINGULAR. The computations are done by using Singular 3-1-3. We have obtained timings for the ideals $< f_i >$, where

$$f_1 = ((x^7 - y^6)^4 - x^{20}y^{10} + x^{35})((x^7 - y^6)^3 - x^{24} + y^{25})((x^7 - y^5)^3 - x^{24} - y^{25})$$
$$f_2 = (x^{14} + x^5y^5 + y^{10} + y^{15})(x^{16} + x^{18} + y^{20} + y^{23})$$
$$f_3 = (x^{14} - y^{12} + x^{10}y^4 + y^{19})(x^{12} - y^{14} + x^{15} + y^{15})((x^6 - y^5)^4 - x^{26}y^{12} + x^{28} + y^{30})(x^{14} + y^{18} + y^{21})$$
$$f_4 = (x^8 + 2y^{14})(x^{10} + 5y^{10})(y^2 - x^3)(x^2 - y^3)$$
$$f_5 = (x^{18} + y^{24} + x^{29})(x^{14} + y^{18} + y^{21})(x^9 - x^3y^3 + y^{11})(x^{15} - y^{10} - y^{19})$$
$$f_6 = (x^{16} + 3y^{18})(x^5 + 7y^5)$$
$$f_7 = ((x^7 - y^6)^4 - x^{20}y^{10} + x^{35})((x^7 - y^6)^3 - x^{24} + y^{25})$$
$$f_8 = (x^{18} + y^{24} + x^{29})(x^{14} + y^{18} + y^{21})(x^9 - x^3y^3 + y^{11})$$

As we can observe from the next table, the computation of the invariants of these examples have shown that, the library ”curvepar.lib” is faster in half of the examples, while the library ”hnoether.lib” is faster in the other half.

Table 8.1: Time Comparison

poly	hnoether	curveparam
f_1	3 sec	> 1 hour
f_2	245 sec	124 sec
f_3	1 sec	> 1 hour
f_4	> 1 hour	11 sec
f_5	2788 sec	546 sec
f_6	> 1 hour	1 sec
f_7	0 secs	> 1 hour
f_8	0 secs	72 sec

Checking the examples for which the ”curvepar.lib” is slower, we have noticed that while the procedures ”CurveRes(I)” and ”CurveParam(I)” in ”curveparam.lib” work faster than the procedures ”develop(f)” and ”param(f)” in the library ”hnoether.lib”, we still get slower results. What makes ”curvepar.lib” slower in these four examples is that the successive division of the series. Although we have better timings in half of the examples when compared to ”hnoether.lib”, we should avoid successive divisions to find the multiplicity sequence and the contact numbers for our library to work faster in all of the computations. We still search for the ways of reading the multiplicity sequence of each branch and the contact numbers from the output of ”CurveRes(I)”.
For future research, our purpose is to find a way to read the contact numbers from the output of ”CurveRes(I)”. In this way, it will be possible to construct the resolution graph from the resolution process. Then finding the multiplicity sequence of each branch from the resolution graph without using successive division will be possible. This will save a lot of time in the computations and we will get a faster library.

CHAPTER 9

CONCLUSION

In this thesis, we have studied the space curve singularities. We have focused on Arf rings, which makes it possible generalize the concepts like the chracaters in the plane case to space case. Our main contribution has been to give a new and efficient algorithm to compute the Arf closure of a local ring. This depends on determining a bound for doing the successive division of series. We have shown that, to construct the Arf closure correctly, it is sufficient to consider the terms up to degree $c^* + 1$ while dividing the series. We have also obtained a bound for the conductor c^* of the semigroup of the Arf closure R^* of $R \subset k[[t]]$, without computing R^*. We have shown that it is possible to read this bound by just considering the degrees of the series appearing in the parametrization corresponding to R, when the parametrization is of Puiseux form, in other words. $R = k[[\varphi_1(t) = t^m, \varphi_2(t), ..., \varphi_n(t)]]$ for some $m \in \mathbb{N}$

Our algorithm works much faster than the algorithm given in [2], which was the only implemented algorithm for constructing the Arf closure in the literature. The speed of our algorithm is a a result of two facts: first, our algorithm avoids the computation of the semigroup of the given branch. Second, it uses a much smaller bound in doing the series divisions. Also, this bound, which gives the opportunity to determine the multiplicity sequence with a fast algorithm can be used for writing a fast algorithm to determine the numerical semigroup of a branch, which is a difficult problem.

With our new library, we have computed examples to check a conjecture given by Arslan and Sertoz. All of our examples supported the conjecture proposing that among the local rings having the same Arf closure, the one with the smallest embedding dimension has the largest regularity index. As a future goal, we want to attack this conjecture, which supports the idea that taking the Arf closure is in fact “taming the singularity” by filling some gaps, so between two rings having the same Arf closure, the one closer to the Arf closure can not have a “worse“ singularity.

In this thesis, we have also dealt the singularity theory of plane algebroid curves. We have written a SINGULAR library ”curvepar.lib” that computes the invariants of plane algebroid curves. Comparing to the library ”hnoether.lib”, which also computes the invariants of plane algebroid curves, we have observed that our library is faster in almost half of the examples, while the ”hnoether.lib” is faster in the other half. For algorithmic purposes, this is quite important, since both libraries can be combined to obtain a more efficient one by correctly deciding to use the fast one for a given plane algebroid curve.

Another goal for future research is to add new procedures that computes the multiplicity sequence of the branches of a reducible algebroid curve given with the defining ideal $I \subset k[[x_1, .., x_n]]$ where n is greater than 2 to our SINGULAR library ”curvepar.lib”.

To conclude, it wouldn’t be possible to obtain most of the results in this thesis without the use of computational methods. This thesis is an example showing that the geometric problems can be attacked by computational methods.

REFERENCES

[1] Cahit Arf. Une interprétation algébrique de la suite des ordres de multiplicité d'une branche algébrique. *Proc. London Math. Soc. (2)*, 50:256–287, 1948.

[2] Sefa Feza Arslan. On arf rings, 1994.

[3] V. Barucci, M. D'Anna, and R. Fröberg. ARF characters of an algebroid curve. *JP J. Algebra Number Theory Appl.*, 3(2):219–243, 2003.

[4] V. Barucci, M. D'Anna, and R. Fröberg. On plane algebroid curves. In *Commutative ring theory and applications (Fez, 2001)*, volume 231 of *Lecture Notes in Pure and Appl. Math.*, pages 37–50. Dekker, New York, 2003.

[5] M. A. Binyamin. Improving the computation of invariants of plane curve singularities. *Analele Stiintifice Ale Universitatii "Ovidius" Constanta*.

[6] Maciej Borodzik. An efficient method of finding a puiseux expansion of a parametric singularity.

[7] Maria Bras-Amorós. Improvements to evaluation codes and new characterizations of Arf semigroups. In *Applied algebra, algebraic algorithms and error-correcting codes (Toulouse, 2003)*, volume 2643 of *Lecture Notes in Comput. Sci.*, pages 204–215. Springer, Berlin, 2003.

[8] Egbert Brieskorn and Horst Knörrer. *Plane algebraic curves*. Birkhäuser Verlag, Basel, 1986. Translated from the German by John Stillwell.

[9] A. Campillo, F. Delgado, and C. A. Núñez. The arithmetic of Arf and saturated semigroups. Applications. *Rev. Real Acad. Cienc. Exact. Fís. Natur. Madrid*, 82(1):161–163, 1988.

[10] Antonio Campillo. *Algebroid curves in positive characteristic*, volume 813 of *Lecture Notes in Mathematics*. Springer, Berlin, 1980.

[11] Antonio Campillo and Julio Castellanos. Arf closure relative to a divisorial valuation and transversal curves. *Amer. J. Math.*, 116(2):377–395, 1994.

[12] Antonio Campillo and Julio Castellanos. Valuative Arf characteristic of singularities. *Michigan Math. J.*, 49(3):435–450, 2001.

[13] Antonio Campillo and Julio Castellanos. *Curve Singularities: An Algebraic and Geometric Approach*. Actualités Mathématiques. Hermann, 2005.

[14] Antonio Campillo, José Ignacio Farrán, and Carlos Munuera. On the parameters of algebraic-geometry codes related to Arf semigroups. *IEEE Trans. Inform. Theory*, 46(7):2634–2638, 2000.

[15] Angel Castellanos and Julio Castellanos. Algorithm for the semigroup of a space curve singularity. *Semigroup Forum*, 70(1):44–60, 2005.

[16] Julio Castellanos. A relation between the sequence of multiplicities and the semigroup of values of an algebroid curve. *J. Pure Appl. Algebra*, 43(2):119–127, 1986.

[17] I. S. Cohen. On the structure and ideal theory of complete local rings. *Trans. Amer. Math. Soc.*, 59:54–106, 1946.

[18] Gerhard Pfister; Nil Şahin; Maryna Viazonska. curvepar.lib SINGULAR 3-1-4 a library for computing the resolution of space curve singularities and the numerical invariants of plane curve singularities. 2012. http://www.singular.uni-kl.de.

[19] Steven Dale Cutkosky. *Resolution of singularities*, volume 63 of *Graduate Studies in Mathematics*. American Mathematical Society, Providence, RI, 2004.

[20] Marco D'Anna. Canonical module and one-dimensional analytically irreducible Arf domains. In *Commutative ring theory (Fès, 1995)*, volume 185 of *Lecture Notes in Pure and Appl. Math.*, pages 215–225. Dekker, New York, 1997.

[21] Theo de Jong and Gerhard Pfister. *Local analytic geometry*. Advanced Lectures in Mathematics. Friedr. Vieweg & Sohn, Braunschweig, 2000. Basic theory and applications.

[22] David E. Dobbs and Gretchen L. Matthews. On comparing two chains of numerical semigroups and detecting Arf semigroups. *Semigroup Forum*, 63(2):237–246, 2001.

[23] Patrick Du Val. The Jacobian algorithm and the multiplicity sequence of an algebraic branch. *Rev. Fac. Sci. Univ. Istanbul. Ser. A.*, 7:107–112, 1942.

[24] Patrick Du Val. Note on Cahit Arf's "Une interprétation algébrique de la suite des ordres de multiplicité d'une branche algébrique.". *Proc. London Math. Soc. (2)*, 50:288–294, 1948.

[25] J. Elias, J. M. Giral, R. M. Miró-Roig, and S. Zarzuela, editors. *Six lectures on commutative algebra*. Modern Birkhäuser Classics. Birkhäuser Verlag, Basel, 2010. Papers from the Summer School on Commutative Algebra held in Bellaterra, July 16–26, 1996, Reprint of the 1998 edition.

[26] Federigo Enriques and Oscar Chisini. *Lezioni sulla teoria geometrica delle equazioni e delle funzioni algebriche. 2. Vol. III, IV*, volume 5 of *Collana di Matematica [Mathematics Collection]*. Nicola Zanichelli Editore S.p.A., Bologna, 1985. Reprint of the 1924 and 1934 editions.

[27] Robin Hartshorne. *Algebraic geometry*. Springer-Verlag, New York, 1977. Graduate Texts in Mathematics, No. 52.

[28] Abramo Hefez and Marcelo Escudeiro Hernandes. *Computational methods in the local theory of curves*. Publicações Matemáticas do IMPA. [IMPA Mathematical Publications]. Instituto de Matemática Pura e Aplicada (IMPA), Rio de Janeiro, 2001. 23o Colóquio Brasileiro de Matemática. [23rd Brazilian Mathematics Colloquium].

[29] Heisuke Hironaka. Resolution of singularities of an algebraic variety over a field of characteristic zero. I, II. *Ann. of Math. (2) 79 (1964), 109–203; ibid. (2)*, 79:205–326, 1964.

[30] Fernando Hernando Carrillo; Thomas Keilen. alexpoly.lib SINGULAR 3-1-3 a library for computing the hamburger-noether expansion of a reduced plane curve singularity and the numerical invariants of plane curve singularities. 2011. http://www.singular.uni-kl.de.

[31] János Kollár. *Lectures on resolution of singularities*, volume 166 of *Annals of Mathematics Studies*. Princeton University Press, Princeton, NJ, 2007.

[32] Martin Lamm. Hamburger-noether-entwicklung von kurvensingularitaten, 1999.

[33] Joseph Lipman. Stable ideals and Arf rings. *Amer. J. Math.*, 93:649–685, 1971.

[34] Martin Lamm; Christoph Lossen. hnoether.lib SINGULAR 3-1-3 a library for computing the hamburger-noether expansion of a reduced plane curve singularity and the numerical invariants of plane curve singularities. 2011. http://www.singular.uni-kl.de.

[35] J. C. Rosales, P. A. García-Sánchez, J. I. García-García, and M. B. Branco. Arf numerical semigroups. *J. Algebra*, 276(1):3–12, 2004.

[36] Judith D. Sally. On the associated graded ring of a local Cohen-Macaulay ring. *J. Math. Kyoto Univ.*, 17(1):19–21, 1977.

[37] Wolfram Decker; Gert-Martin Greuel; Gerhard Pfister; Hans Schönemann. SINGULAR 3-1-3 — A computer algebra system for polynomial computations. 2011. http://www.singular.uni-kl.de.

[38] Sinan Sertöz. Arf rings and characters. *Note Mat.*, 14(2):251–261 (1997), 1994.

[39] Maryna Viazovska. Computation of weierstrass semigroup for space curve singularities. Master's thesis, University of Kaiserslautern, March 2007.

[40] C. T. C. Wall. *Singular points of plane curves*, volume 63 of *London Mathematical Society Student Texts*. Cambridge University Press, Cambridge, 2004.

APPENDIX A

The Singular Library ”curvepar.lib”

```
/////////////////////////////////////////////////////////////////////////////////
version="$Id: curvepar.lib 15103 2012-07-11 10:00:13Z motsak $";
category="Singularity Theory";
info="
LIBRARY:  curvepar.lib  Resolution of space curve singularities, semi-group

AUTHOR:     Gerhard Pfister    email: pfister@mathematik.uni-kl.de
            Nil Sahin          email: e150916@metu.edu.tr
            Maryna Viazovska   email: viazovsk@mathematik.uni-kl.de

SEE ALSO: spcurve_lib

PROCEDURES:
BlowingUp(f,I,l);             BlowingUp of V(I) at the point 0;
CurveRes(I);                  Resolution of V(I)
CurveParam(I);                Parametrization of algebraic branches of V(I)
WSemigroup(X,b);              Weierstrass semigroup of the curve
primparam(x,y,c);             HN matrix of parametrization(x(t),y(t))
MultiplicitySequence(I);      Multiplicity sequences of the branches of plane curve V(I)
CharacteristicExponents(I);   Characteristic exponents of the branches of plane curve V(I)
IntersectionMatrix(I);        Intersection Matrix of the branches of plane curve V(I)
ContactMatrix(I);             Contact Matrix of the branches of plane curve V(I)
plainInvariants(I);           Invariants of the branches of plane curve V(I)
";

LIB "sing.lib";
LIB "primdec.lib";
LIB "linalg.lib";
LIB "ring.lib";
LIB "alexpoly.lib";
LIB "matrix.lib";

//////////////////////////////////////////////////////////////
//----------Resolution of singular curve--------------------//
//////////////////////////////////////////////////////////////

proc BlowingUp(poly f,ideal I,list l,list #)
"USAGE:   BlowingUp(f,I,l);
          f=poly
          b=ideal
          l=list

ASSUME:   The basering is r=0,(x(1..n),a),dp
          f is an irrreducible polynomial in k[a],
          I is an ideal of a curve(if we consider a as a parameter)

COMPUTE:  Blowing-up of the curve at point 0.

RETURN:   list C of charts.
          Each chart C[i] is a list of size 5 (reps. 6 in case of plane curves)
          C[i][1] is an integer j. It shows, which standard chart do we consider.
          C[i][2] is an irreducible poly g in k[a]. It is a minimal polynomial
                 for the new parameter.
          C[i][3] is an ideal H in k[a].
                 c_i=F_i(a_new) for i=1..n,
                 a_old=H[n+1](a_new).
          C[i][4] is a map teta:k[x(1)..x(n),a]-->k[x(1)..x(n),a] from the new
                 curve to the old one.
                 x(1)-->x(j)*x(1)
                 . . .
                 x(j)-->x(j)
                 . . .
                 x(n)-->x(j)*(c_n+x(n))
          C[i][5] is an ideal J of a new curve. J=teta(I).
          C[i][6] is the list of exceptional divisors in the chart
```

```
EXAMPLE: example BlowingUp; shows an example"
{
  def r=basering;
  int n=nvars(r)-1;
  ring r1=(0,a),(x(1..n)),ds;
  number f=leadcoef(imap(r,f));
  minpoly=f;
  ideal I=imap(r,I);
  ideal locI=std(I);
  ideal J=tangentcone(I);
  setring r;
  ideal J=imap(r1,J);
  ideal locI=imap(r1,locI);
  int j;
  int i;
  list C,E;
  list C1;
  ideal B;
  poly g;
  ideal F;
  poly b,p;
  list Z;
  list Z1;
  ideal D;
  map teta;
  ideal D1;
  map teta1;
  int k,e;
  ideal I1;
  ideal I2;
  int ind;
  list w=mlist(1,n);
  for(j=1;j<=n;j++)
  {
    B=J;
    for(i=1;i<j;i++) {B=B+x(w[i]);}
    B=B+(x(w[j])-1);
    B=B+f;
    Z=Z1;
    if(dim(std(B))==0)
    {
      Z=ZeroIdeal(B);
      for(i=1;i<j;i++)
      {
        D[w[i]]=x(w[j])*x(w[i]);
      }
      D[w[j]]=x(w[j]);
      for(i=j+1;i<=n;i++)
      {
        D[w[i]]=x(w[j])*x(w[i]);
      }
      D[n+1]=a;
      teta=r,D;
      I1=teta(locI);
      I1=reduce(I1,std(f));
      ind=1;
      for(i=1;i<=size(I1);i++)
      {
        ind=1;
        while(ind==1)
        {
          if(gcd(x(w[j]),I1[i])==x(w[j])){I1[i]=I1[i]/x(w[j]);}
          else{ind=0;}
        }
      }
    }
    for(k=1;k<=size(Z);k++)
    {
      g=Z[k][1];
      for(i=1;i<=n;i++){F[i]=Z[k][2][i];}
      b=Z[k][3];
      C1[1]=w[j];
      C1[2]=g;
      C1[3]=F;
      for(i=1;i<j;i++)
      {D[w[i]]=x(w[j])*x(w[i]);}
      D[w[j]]=x(w[j]);
      for(i=j+1;i<=n;i++)
      {D[w[i]]=x(w[j])*(F[w[i]]+x(w[i]));}
      D[n+1]=Z[k][2][n+1];
      teta=r,D;
      C1[4]=D;
      for(i=1;i<=j;i++)
      {D1[w[i]]=x(w[i]);}
      for(i=j+1;i<=n;i++)
      {D1[w[i]]=F[w[i]]+x(w[i]);}
      D1[n+1]=a;
      teta1=r,D1;
      if(nvars(basering)==3)
      {
        I2=quickSubst(I1[1],teta1[1],teta1[2],std(g));
      }
      else
      {
        I2=teta1(I1);
        I2=reduce(I2,std(g));
      }
      C1[5]=I2;
      if(nvars(basering)==3)
      {
        if(size(#)>0)
        {
          E=#;
          E=teta(E);
          for(e=1;e<=size(E);e++)
          {
            p=E[e];
            while(subst(p,x(w[j]),0)==0)
            {
              p=p/x(w[j]);
            }
            if((deg(E[e])>0)&&(deg(p)==0))
            {
              E[e]=size(E);
            }
            else
            {
              E[e]=p;
            }
          }
          E[size(E)+1]=x(w[j]);
          C1[6]=E;
        }
        else
        {
          C1[6]=list(x(w[j]));
        }
      }
      C=insert(C,C1);
    }
  }
  return(C);
}
example
{
  "EXAMPLE:";echo = 2;
  ring r=0,(x(1..3),a),dp;
  poly f=a2+1;
  ideal i=x(1)^2+a*x(2)^3,x(3)^2-x(2);
  list l=1,3,2;
  list B=BlowingUp(f,i,l);
  B;
}
//============= ACHTUNG ZeroIdeal ueberarbeiten / minAssGTZ rein  ========================
/////////////////////////////////////////////////////////////////////////////////////////
static proc ZeroIdeal(ideal J)

"USAGE:   ZeroIdeal(J);
          J=ideal

ASSUME:   J is a zero-dimensional ideal in k[x(1),...,x(n)].

COMPUTE:  Primary decomposition of radical(J). Each prime ideal J[i] has the form:
          x(1)-f[1](b),...,x(n)-f[n](b),
          f(b)=0, f irreducible
          for some b=x(1)*a(1)+...+x(n)*a(n), a(i) in k.

RETURN:   list Z of lists.
          Each list Z[k] is a list of size 3
          Z[k][1] is a poly f(b)
          Z[k][2] is an ideal H, H[n]=f[n],
          Z[k][3] is a poly x(1)*a(1)+...+x(n)*a(n)

EXAMPLE:"
{
  intvec opt = option(get);
  def r=basering;
  int n=nvars(r);
  if(dim(std(J))!=0){return(0);}
  ring s=0,(x(1..n)),lp;
  ideal A; ideal S; int i; int j;
  for(i=1;i<=n;i++) {A[i]=x(i);}
  map phi=r,A;
  ideal J=phi(J);
  ideal I=radical(J);
  list D=zerodec(I);
  list Z; ideal H; intvec w; intvec v; int ind; ideal T; map tau; int q; list u;
  ideal Di; poly h;
  for(i=1;i<=size(D);i++)
  {
    option(redSB);
    ind=0;q=n;
    while(ind==0 and q>0)
    {
      for(j=1;j<=n;j++){T[j]=x(j);}
      T[q]=x(n);
      T[n]=x(q);
      tau=s,T;
      Di=D[i];
      S=std(tau(Di));
      ind=1;
      v=leadexp(S[1]);
      if(leadmonom(S[1])!=x(n)^v[n]){ind=0;}
      for(j=2;j<=n;j++)
      {
        if(leadmonom(S[j])!=x(n-j+1)){ind=0;}
        H[n-j+1]= -S[j]/leadcoef(S[j])+x(n-j+1);
        v=leadexp(H[n-j+1]);
        if(leadcoef(H[n-j+1])*leadmonom(H[n-j+1])!=leadcoef(H[n-j+1])*x(n)^v[n])
        {ind=0;}
      }
      if(ind==1)
      {
        u[1]=S[1];
        H[n]=x(n);
        H[n]=H[q];
        H[q]=x(n);
        u[2]=H;
        u[3]=x(q);
        Z[i]=u;
      }
      q--;
    }
    if(ind==0)
    {
      vector a;
      while(ind==0)
      {
        h=x(n);
        for(j=1;j<=n-1;j++){a=a+random(-10,10)*gen(j);h=h+a[j]*x(j);}
        T=subst(S,x(n),h);
        option(redSB);
        T=std(T);
        ind=1;
        w=leadexp(T[1]);
        if(leadmonom(T[1])!=x(n)^w[n]){ind=0;}
        for(j=2;j<=n;j++)
        {
          if(leadmonom(T[j])!=x(n-j+1)){ind=0;}
          H[n-j+1]= -T[j]/leadcoef(T[j])+x(n-j+1);
          w=leadexp(H[n-j+1]);
          if(leadmonom(H[n-j+1])*leadcoef(H[n-j+1])!=leadcoef(H[n-j+1])*x(n)^w[n])
          {ind=0;}
        }
        if(ind==1)
        {
          list l;
          l[1]=T[1];
          H[n]=x(n);h=x(n);
          for(j=1;j<=n-1;j++){H[n]=H[n]+a[j]*H[j];h=h-a[j]*x(j);}
          l[2]=H;
          l[3]=h;
          Z[i]=l;
        }
      }
    }
  }
  setring r;
  ideal A;
  list Z;
  for(i=1;i<=n;i++)
  {A[i]=var(i);}
  map psi=s,A;
  Z=psi(Z);
  option(set, opt);
  return(Z);
}
///////////////////////////////////////////////////////////////////////////////////////////////
//assume that the basering is k[x(1),...,x(n),a]

static proc main(ideal I,ideal Psi,poly f,list m,list l,list HN,intvec v,list HI,list #)
{
  def s=basering;
```

```
int i,z;
int j;
list C,E,resTree;
list C1;
list C2;
list C3;
list l1;
C2[8]=HI;
list m1;
list HN1;
ideal J;
map psi;
intvec w;
z=(SmoothTest(I,f)==1);
if((nvars(basering)==3)&&z&&(size(#)>0))
{
  z=transversalTest(I,f,#);
}
if(z)
{
  C2[1]=I;
  C2[2]=Psi;
  C2[3]=f;
  C2[4]=m;
  C2[5]=l;
  C2[6]=HN;
  if(nvars(basering)==3)
  {
    if(size(#)>0)
    {
      C2[9]=#;
    }
    C2[7]=v;
  }
  //C2[8][size(C2[8])+1]=list(C2[7],C2[9]);
  C[1]=C2;
}
if(!z)
{
  int mm=mmult(I,f);
  m1=insert(m,mm,size(m));
  if(nvars(basering)==3)
  {
    if(size(#)>0)
    {
      E=#;
      C1=BlowingUp(f,I,l,E);
    }
    else
    {
      C1=BlowingUp(f,I,l);
    }
  }
  else
  {
    C1=BlowingUp(f,I,l);
  }
  for(j=1;j<=size(C1);j++)
  {
    C2[1]=C1[j][5];
    J=C1[j][4];
    psi=s,J;
    C2[2]=psi(Psi);
    C2[3]=C1[j][2];
    C2[4]=m1;
    l1=insert(l,C1[j][1],size(l));
    C2[5]=l1;
    HN1=psi(HN);
    HN1=insert(HN1,C1[j][3],size(HN1)-1);
    C2[6]=HN1;
    if(deg(C2[3])>1)
    {
      w=v,-j;
    }
    else
    {
      w=v,j;
    }
    C2[7]=w;
    if(nvars(basering)==3)
    {
      C2[9]=C1[j][6];
      C2[8][size(C2[8])+1]=list(C2[7],C2[9]);
      C3=main(C2[1],C2[2],C2[3],C2[4],C2[5],C2[6],C2[7],C2[8],C2[9]);
      C=C+C3;
    }
    else
    {
        C3=main(C2[1],C2[2],C2[3],C2[4],C2[5],C2[6],C2[7],C2[8]);
        C=C+C3;
      }
    }
  }
  return(C);
}
///////////////////////////////////////////////////////////////////////////////////////////////

static proc transversalTest(ideal I,poly f,list L)
{
  def r=basering;
  int n=nvars(r)-1;
  int i;
  ring r1=(0,a),(x(1..n)),ds;
  number f=leadcoef(imap(r,f));
  minpoly=f;
  ideal I=imap(r,I);
  list L=imap(r,L);
  ideal K=jet(L[size(L)],deg(lead(L[size(L)])));
  ideal T=1;
  if(size(L)>1)
  {
    for(i=1;i<=size(L)-1;i++)
    {
      if(subst(L[i],x(1),0,x(2),0)==0) break;
    }
    if(i<=size(L)-1)
    {
      T=jet(L[i],deg(lead(L[i])));
    }
  }
  ideal J=jet(I[1],deg(lead(I[1])));
  setring r;
  ideal J=imap(r1,J);
  ideal K=imap(r1,K);
  ideal T=imap(r1,T);
  int m=size(reduce(J,std(K)))+size(reduce(K,std(J)));
  if(m)
  {
    m=size(reduce(J+K+T,std(ideal(x(1),x(2)))));
  }
  return(m);
}
///////////////////////////////////////////////////////////////////////////////////////////////
static proc SmoothTest(ideal I,poly f)
//Assume I is a radical ideal of dimension 1 in a ring k[x(1..n),a]
//Returns 1 if a curve V(I) is smooth at point 0 and returns 0 otherwise
{
  int ind;
  int l;
  def t=basering;
  int n=nvars(t)-1;
  ring r1=(0,a),(x(1..n)),dp;
  number f=leadcoef(imap(t,f));
  minpoly=f;
  ideal I=imap(t,I);
  matrix M=jacob(I);
  for(l=1;l<=n;l++){M=subst(M,x(l),0);}
  if(mat_rk(M)==(n-1)){ind=1;}
  return(ind);
}
///////////////////////////////////////////////////////////////////////////////////////////////
proc CurveRes(ideal I)
"USAGE:   CurveRes(I);
          I ideal
ASSUME:   The basering is r=0,(x(1..n))
          V(I) is a curve with a singular point 0.
COMPUTE:  Resolution of the curve V(I).
RETURN:   a ring R=basering+k[a]
          Ring R contains a list Resolve
          Resolve is a list of charts
          Each Resolve[i] is a list of size 6
          Resolve[i][1] is an ideal J of a new curve. J=teta(I).
          Resolve[i][2] ideal which represents the map
                        teta:k[x(1)..x(n),a]-->k[x(1)..x(n),a] from the
                        new curve to the old one.
          Resolve[i][3] is an irreducible poly g in k[a]. It is a minimal polynomial for the
                        new parameter a. deg(g) gives the number of branches in Resolve[i]
          Resolve[i][4] sequence of multiplicities (sum over all branches in Resolve as long as
                        they intersect each other !)
          Resolve[i][5] is a list of integers l. It shows, which standard charts we considered.
          Resolve[i][6] HN matrix
          Resolve[i][7] (only for plane curves) the development of exceptional divisors
                        the entries correspond to the i-th blowing up. The first entry is an
                        intvec. The first negative entry gives the splitting of the (over Q
                        irreducible) branches. The second entry is a list of the exceptional
                        divisors. If the entry is an integer i, it says that the divisor is not
```

```
                       visible in this chart after the i-th blowing up.

EXAMPLE: example CurveRes; shows an example"
{
  def r=basering;
  int n=nvars(r);
  ring s=0,(x(1..n),a),dp;
  ideal A;
  int i;
  int j;
  for(i=1;i<=n;i++){A[i]=x(i);}
  map phi=r,A;
  ideal I=phi(I);
  poly f=a;
  list l;
  list m;
  list HN=x(1);
  ideal psi;
  for(i=1;i<=n;i++){psi[i]=x(i);}
  psi[n+1]=a;
  intvec v;
  list L,Resolve;
  if(n==2)
  {
    ideal J=factorize(I[1],1);
    list resolve;
    for(int k=1;k<=size(J);k++)
    {
      I=J[k];
      resolve=main(I,psi,f,m,l,HN,v,L);
      for(i=1;i<=size(resolve);i++)
      {
        resolve[i][6]=delete(resolve[i][6],size(resolve[i][6]));
        if(size(resolve[i])>=9){resolve[i]=delete(resolve[i],9);}
        resolve[i]=delete(resolve[i],7);
      }
      if(k==1){Resolve=resolve;}
      else{Resolve=Resolve+resolve;}
    }
  }
  else
  {
    Resolve=main(I,psi,f,m,l,HN,v,L);
    for(i=1;i<=size(Resolve);i++)
    {
      Resolve[i][6]=delete(Resolve[i][6],size(Resolve[i][6]));
      Resolve[i]=delete(Resolve[i],8);
    }
  }
  export(Resolve);
  return(s);
}
example
{
  "EXAMPLE:"; echo=2;
  ring r=0,(x,y,z),dp;
  ideal i=x2-y3,z2-y5;
  def s=CurveRes(i);
  setring s;
  Resolve;
}
///////////////////////////////////////////////////////////////////
static proc mlist(list l,int n)
{
  list N;
  list M;
  int i;
  int j;
  for(i=1;i<=n;i++) {M[i]=i;}
  N=l+M;
  for(i=1;i<=size(N)-1;i++)
  {
    j=i+1;
    while(j<=size(N))
    {
      if(N[i]==N[j]){N=delete(N,j);}
      else
      {j++;}
    }
  }
  return(N);
}
//////////////////////////////////////////////////////////////////////
//Assume that the basering is k[x(1..n),a]

static proc mmult(ideal I,poly f)
{
  def r=basering;
  int n=nvars(r)-1;
  ring r1=(0,a),(x(1..n)),ds;
  number f=leadcoef(imap(r,f));
  minpoly=f;
  ideal I=imap(r,I);
  int m=mult(std(I));
  return(m);
}
///////////////////////////////////////////////////////////////
//--------Parametrization of smooth curve-------------------//
///////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////////////
//computes jacobian matrix, considering x(1..n) as variables and a(1..m) as parameters

static proc mjacob(ideal I)
{
  def r=basering;
  int n=nvars(r);
  int k=size(I);
  matrix M[k][n];
  int i;
  int j;
  int l;
  for(i=1;i<=k;i++)
  {
    for(j=1;j<=n;j++)
    {
      M[i,j]=diff(I[i],x(j));
      for(l=1;l<=n;l++){M[i,j]=subst(M[i,j],x(l),0);}
    }
  }
  return(M);
}
///////////////////////////////////////////////////////////
static proc mmi(matrix M,int n)
{
  ideal l;
  int k=nrows(M);
  int i;
  int j;
  for(i=1;i<=k;i++)
  {
    l[i]=0;
    for(j=1;j<=n;j++)
    {
      l[i]=l[i]+x(j)*M[i,j];
    }
  }
  l=std(l);
  int t=size(l);
  i=1;
  int mi=0;
  while( mi==0 and i<=n-1)
  {
    if(diff(l[i],x(n-i))!=0){mi=n-i+1;}
    else{i++;}
  }
  if(mi==0){mi=1;}
  matrix Mi[k][n-1];
  for(i=1;i<=k;i++)
  {
    for(j=1;j<=mi-1;j++)
    {
      Mi[i,j]=M[i,j];
    }
    for(j=mi;j<=n-1;j++)
    {
      Mi[i,j]=M[i,j+1];
    }
  }
  list lmi=mi,Mi;
  return(lmi);
}
///////////////////////////////////////////////////////////
static proc mC(matrix Mi,int n)
{
  int k=nrows(Mi);
  ideal c;
  int i,j;
  for(i=1;i<=n-1;i++)
  {
    c[i]=0;
    for(j=1;j<=k;j++)
    {
      c[i]=c[i]+y(j)*Mi[j,i];
    }
  }
  c=std(c);
  return(c);
}
//////////////////////////////////////////////////////////
static proc mmF(ideal C, matrix Mi,int n,int k)
{
  int s=size(C);
  intvec mf;
  int p=0;
  int t=0;
  int i;
  int j;
  int v=0;
  for(i=s;i>0;i--)
  {
    p=t;
    j=1;
    while(t==p and p+j<=k)
    {
      if(diff(C[i],y(p+j))==0){j++;}
      if(diff(C[i],y(p+j))!=0){t=p+j;v++;mf[v]=t;}
    }
  }
  matrix B[n-1][n-1];
  for(i=1;i<=n-1;i++)
  {
    for(j=1;j<=n-1;j++)
    {
      B[i,j]=Mi[ mf[i],j];
    }
  }
  list mmf=mf,B;
  return(mmf);
}
/////////////////////////////////////////////////////
static proc cparam(ideal I,poly f,int n,int m,int N)
{
  def r=basering;
  ring s=(0,a),(x(1..n)),lp;
  number f=leadcoef(imap(r,f));
  minpoly=f;
  ideal I=imap(r,I);
  matrix M=mjacob(I);
  list l0=mmi(M,n);
  int mi=l0[1];
  matrix Mi=l0[2];
  int k=nrows(Mi);
  ring q=(0,a),(y(1..k)),lp;
  number f=leadcoef(imap(r,f));
  minpoly=f;
  matrix Mi=imap(s,Mi);
  ideal D=mC(Mi,n);
  list l1=mmF(D,Mi,n,k);
  intvec mf=l1[1];
  matrix B=l1[2];
  setring s;
  matrix B=imap(q,B);
  matrix C=inverse(B);
  int i;
  int j;
  ideal P;
  for(i=1;i<mi;i++){P[i]=x(i);}
  P[mi]=x(n);
  for(i=1;i<=n-mi;i++){P[mi+i]=x(mi+i-1);}
  map phi=s,P;
  ideal I1=phi(I);
  if(nvars(basering)==2)
  {
    setring r;
    ideal I1=imap(s,I1);
    matrix C=imap(s,C);
    list X;
    matrix d[n-1][1];
    matrix b[n-1][1];
    ideal Q;
    map psi;
    int l;
    for(i=1;i<=N;i++)
    {
      for(j=1;j<=n-1;j++)
      {
        d[j,1]=diff(I1[mf[j]],x(n));
        for(l=1;l<=n;l++)
        {
          d[j,1]=subst(d[j,1],x(l),0);
        }
      }
      b=-C*d;
      I1=jet(I1,N-i+2);
      X[i]=b;
      for(j=1;j<=n-1;j++){Q[j]=x(n)*(b[j,1]+x(j));}
      Q[n]=x(n);
      I1[1]=quickSubst(I1[1],Q[1],Q[2],std(f));
      I1=I1/x(n);
    }
    list Y=X,mi;
    return(Y);
  }
  list X;
  matrix d[n-1][1];
  matrix b[n-1][1];
  ideal Q;
  map psi;
  int l;
  for(i=1;i<=N;i++)
  {
    for(j=1;j<=n-1;j++)
    {
      d[j,1]=diff(I1[mf[j]],x(n));
      for(l=1;l<=n;l++){d[j,1]=subst(d[j,1],x(l),0);}
    }
    b=-C*d;
    I1=jet(I1,N-i+2);
    X[i]=b;
    for(j=1;j<=n-1;j++){Q[j]=x(n)*(b[j,1]+x(j));}
    Q[n]=x(n);
    psi=s,Q;
    I1=psi(I1);
    I1=I1/x(n);
  }
  list Y=X,mi;
  setring r;
  list Y=imap(s,Y);
  return(Y);
}
///////////////////////////////////////////////////////////////
//--------Parametrization of singular curve-----------------//
///////////////////////////////////////////////////////////////
proc CurveParam (list #)
"USAGE:   CurveParam(I);
          I ideal
ASSUME:   I is an ideal of a curve C with a singular point 0.
COMPUTE:  Parametrization for algebraic branches of the curve C.
RETURN:   list L of size 1.
          L[1] is a ring ring rt=0,(t,a),ds;
          Ring R contains a list Param
          Param is a list of algebraic branches
          Each Param[i] is a list of size 3
          Param[i][1] is a list of polynomials
          Param[i][2] is an irredusible polynomial f\in k[a].It is a minimal polynomial for
                      the parameter a.
          Param[i][3] is an integer b--upper bound for the conductor of Weierstrass semigroup

EXAMPLE: example curveparam; shows an example"
{
  int i;
  int j;
  if(typeof(#[1])=="ideal")
  {
    int d=deg(#[1][1]);
    def s=CurveRes(#[1]);
  }
  else
  {
    def s=#[1];
  }
  setring s;
  int n=nvars(s)-1;
  list Param;
  list l;
  ideal D,P,Q,T;
  poly f;
  map tau;
  list Z;
  list Y;
  list X;
  int mi;
  int b;
  int k;
  int dd;
  for(j=1;j<=size(Resolve);j++)
  {
    b=0;
    for(k=1;k<=size(Resolve[j][4]);k++)
    {
      b=b+Resolve[j][4][k]*(Resolve[j][4][k]+1);
    }
    if((n==2)&&(size(Resolve[j][4])==0))
    {b=d;}
    Y=cparam(Resolve[j][1],Resolve[j][3],n,1,b);
    X=Y[1];
    mi=Y[2];
    f=Resolve[j][3];
    for(i=1;i<mi;i++)
    {
      P[i]=0;
      for(k=1;k<=b;k++){P[i]=P[i]+X[k][i,1]*(x(1)^k);}
    }
    P[mi]=x(1);
    for(i=mi+1;i<=n;i++)
    {
      P[i]=0;
      for(k=1;k<=b;k++){P[i]=P[i]+X[k][i-1,1]*(x(1)^k);}
    }
    P[n+1]=a;
    tau=s,P;
    T=Resolve[j][2];
//HERE A TEST FOR dd
    if(nvars(basering)==3)
    {
      dd=boundparam(P[2]);
      if(dd==1){dd=boundparam(P[1]);}
      P[1]=jet(P[1],dd);
      P[2]=jet(P[2],dd);
      Q[1]=quickSubst(T[1],P[1],P[2],std(f));
      Q[2]=quickSubst(T[2],P[1],P[2],std(f));
      Q[3]=a;
    }
    else
    {
      Q=tau(T);
    }
    for(i=1;i<=n;i++){Z[i]=jet(reduce(Q[i],std(f)),b+1);}
    l[1]=Z;
    l[2]=f;
    l[3]=b;
    Param[j]=l;
  }
  ring rt=0,(t,a),ds;
  ideal D;
  D[1]=t;
  D[n+1]=a;
  map delta=s,D;
  list Param=delta(Param);
  export(Param);
  return(rt);
}
example
{
  "EXAMPLE:";echo=2;
  ring r=0,(x,y,z),dp;
  ideal i=x2-y3,z2-y5;
  def s=CurveParam(i);
  setring s;
  Param;
}
////////////////////////////////////////////////////////////////////////////////////////////
//----------Computation of Weierstrass Semigroup from parametrization--------------------//
////////////////////////////////////////////////////////////////////////////////////////////
static proc Semi(intvec G,int b)
"USAGE:   Semi(G,b);
          G=intvec
          b=int
ASSUME:   G[1]<=G[2]<=...<=G[k],
COMPUTE:  elements of semigroup S generated by the enteries of G till the bound b.
          For each element i of S computes the list of integer vectors v of dimension
          k=size(G), such that g[1]*v[1]+g[2]*v[2]+...+g[k]*v[k]=i. If there exists
          conductor  of semigroup S  c<b-n, where n is minimal element of G, then
          computes also c+n.
RETURN:   list M of size 2.
          L=M[1] is a list  of size min(b,c+n).
          L[i] is a list of integer vectors.
          If i is not in a semigroup S than L[i] is empty.
          M[2] is an integer =min(b,c+n)
          M[3] minimal generators of S
EXAMPLE:"
{
  list L;
  list l;
  int i;
  for(i=1;i<=b;i++){L[i]=l;}
  int k=size(G);
  int n=G[1];
  int j;
  int t;
  int q;
  int c=0;
  intvec w;
  intvec v;
  for(i=1;i<=k;i++)
  {
    for(j=1;j<=k;j++)
    {
      if(j==i){w[j]=1;}
      else{w[j]=0;}
    }
    L[G[i]]=insert(L[G[i]],w);
  }
  list L1=L;
  int s=0;
  int s1=0;
  i=1;
  int p;
  while(i<=b and s<n)
  {
    for(j=1;j<=k;j++)
    {
      if(i-G[j]>0)
      {
        if(size(L[i-G[j]])>0)
        {
          for(t=1;t<=size(L[i-G[j]]);t++)
          {
            v=L[i-G[j]][t];
            p=1;
            for(q=1;q<j;q++)
            {
              if(v[q]>0){p=0;}
            }
            if(p==1){v[j]=v[j]+1;L[i]=insert(L[i],v);}
          }
        }
      }
    }
    if(size(L[i])>0){s1=1;}
    s=s1*(s+1);
    s1=0;
    i++;
  }
  intvec Gmin;
  int jmin=1;
  for(j=1;j<=k;j++)
  {
    if(size(L[G[j]])==size(L1[G[j]]) && G[j]<i)
    {
      Gmin[jmin]=G[j];
      L1[G[j]]=insert(L1[G[j]],0);
      jmin++;
    }
  }
  list M=L,i-1,Gmin;
  return(M);
}
///////////////////////////////////////////////////////////////////////////////////////////
static proc AddElem(list L,int b,int k,int g,int n)
"ASSUME:  L list of size b. L[i] list of integer vectors of dimension k.
          b=int
          k=int as above
          g=int new generator
          n=int. minimal generator
RETURN: list M
        M[1]=new L;
        M[2]=new b;"
{
  int i,j;
  intvec v;
  for(i=1;i<=b;i++)
  {
    if(size(L[i])>0)
    {
      for(j=1;j<=size(L[i]);j++)
      {
        v=L[i][j];
        v[k+1]=0;
        L[i][j]=v;
      }
    }
  }
  intvec w;
  w[k+1]=1;
  L[g]=insert(L[g],w);
  int s=0;
  int s1=0;
  i=1;
  while(i<=b and s<n)
  {
    if(i-g>0)
    {
      if(size(L[i-g])>0)
      {
        for(j=1;j<=size(L[i-g]);j++)
        {
          v=L[i-g][j];
          v[k+1]=v[k+1]+1;
          L[i]=insert(L[i],v);
        }
      }
    }
    if(size(L[i])>0){s1=1;}
    s=s1*(s+1);
    s1=0;
    i++;
  }
  int b1=i-1;
  list M=L,b1;
  return(M);
}
/////////////////////////////////////////////////////////////////////////////////////////////
proc WSemigroup(list X,int b0)
"USAGE:    WSemigroup(X,b0);
           X a list of polinomials in one vaiable, say t.
           b0 an integer
COMPUTE:   Weierstrass semigroup of space curve C,which is given by a parametrization
           X[1](t),...,X[k](t), till the bound b0.

ASSUME:    b0 is greater then conductor
RETURN:    list M of size 5.
           M[1]= list of integers, which are minimal generators set of the Weierstrass semigroup.
           M[2]=integer, conductor of the Weierstrass semigroup.
           M[3]=intvec, all elements of the Weierstrass semigroup till some bound b,
           which is greather than conductor.
WARNING:   works only over the ring with one variable with ordering ds
EXAMPLE: example WSemigroup; shows an example"

{
  int k=size(X);
  intvec G;
  int i,i2;
  poly t=var(1);
  poly h;
  int g;
  for(i=1;i<=k;i++)
  {G[i]=ord(X[i]);}
  for(i=1;i<k;i++)
  {
    for(i2=i;i2<=k;i2++)
    {
      if(G[i]>G[i2])
      {
        g=G[i];G[i]=G[i2];G[i2]=g;
        h=X[i];X[i]=X[i2];X[i2]=h;
      }
    }
  }
  list U=Semi(G,b0);
  list L=U[1];
  int b=U[2];
  G=U[3];
  int k1=size(G);
  list N;
  list l;
  for(i=1;i<=b;i++){N[i]=1;}
  int j;
  for(j=b0;j>b;j--){L=delete(L,j);}
  poly p;
  int s;
  int e;
  for(i=1;i<=b;i++)
  {
    for(j=1;j<=size(L[i]);j++)
    {
      p=1;
      for(s=1;s<=k;s++)
      {
        for(e=1;e<=L[i][j][s];e++)
        {
          p=p*X[s];
          p=jet(p,b);
        }
      }
      N[i]=insert(N[i],p);
    }
  }
  int j1;
  int j2;
  list M;
  poly c1;
  poly c2;
  poly f;
  int m;
  int b1;
  ideal I;
  matrix C;
  matrix C1;
  int q;
  int i1;
  i=1;
  while(i<=b)
  {
    for(j1=2;j1<=size(N[i]);j1++)
    {
      for(j2=1;j2<j1;j2++)
      {
        c1=coeffs(N[i][j1],t)[i+1,1];
        c2=coeffs(N[i][j2],t)[i+1,1];
        f=c2*N[i][j1]-c1*N[i][j2];
        m=ord(f);
        if(m>=0)
        {
          if(size(N[m])==0)
          {
            N[m]=insert(N[m],f);
            if(size(L[m])==0)
            {
              M=AddElem(L,b,k,m,G[1]);
              L=M[1];
              b1=M[2];
              G[k1+1]=m;
              X[k+1]=f;
              N[m]=insert(N[m],f);
              k=k+1;
              k1=k1+1;
              if(b1<b)
              {
                for(i1=1;i1<=b1;i1++)
                {
                  for(s=1;s<=size(N[i1]);s++){N[i1][s]=jet(N[i1][s],b1);}
                }
                for(s=size(N);s>b1;s--){N=delete(N,s);}
                for(s=size(L);s>b1;s--){L=delete(L,s);}
              }
              b=b1;
            }
          }
          else
          {
            for(q=1;q<=size(N[m]);q++){I[q]=N[m][q];}
            I[size(N[m])+1]=f;
            C=coeffs(I,t);
            C1=gauss_col(C);
            if(C1[size(N[m])+1]!=0){N[m]=insert(N[m],f);}
          }
        }
      }
    }
    i++;
  }
  intvec S;
  j=1;
  for(i=1;i<=b;i++)
  {
    if(size(L[i])>0){S[j]=i;j++;}
  }
  U=Semi(G,b);
  G=U[3];
  list Q=G,b-G[1]+1,S;
  return(Q);
}
example
{
  "EXAMPLE:";echo=2;
  ring r=0,(t),ds;
  list X=t4,t5+t11,t9+2*t7;
  list L=WSemigroup(X,30);
  L;
}
///////////////////////////////////////////////////////////////////////////////////////////////
```

```
static proc quickSubst(poly h, poly r, poly s,ideal I)
{
//=== computes h(r,s) mod I for h in Q[x(1),x(2),a]
  attrib(I,"isSB",1);
  if((r==x(1))&&(s==x(2))){return(reduce(h,I));}
  poly q1 = 1;
  poly q2 = 1;
  poly q3 = 1;
  int i,j,e1,e2,e3;
  list L,L1,L2,L3;
  if(r==x(1))
  {
    matrix M=coeffs(h,x(2));
    L[1]=1;
    for(i=2;i<=nrows(M);i++)
    {
      q2 = reduce(q2*s,I);
      L[i]=q2;
    }
    i=1;
    h=0;
    while(i <= nrows(M))
    {
      if(M[i,1]!=0)
      {
         h=h+M[i,1]*L[i];
      }
      i++;
    }
    h=reduce(h,I);
    return(h);
  }
  if(s==x(2))
  {
    matrix M=coeffs(h,x(1));
    L[1]=1;
    for(i=2;i<=nrows(M);i++)
    {
      q1 = reduce(q1*r,I);
      L[i]=q1;
    }
    i=1;
    h=0;
    while(i <= nrows(M))
    {
      if(M[i,1]!=0)
      {
        h=h+M[i,1]*L[i];
      }
      i++;
    }
    h=reduce(h,I);
    return(h);
  }
  for(i=1;i<=size(h);i++)
  {
    if(leadexp(h[i])[1]>e1){e1=leadexp(h[i])[1];}
    if(leadexp(h[i])[2]>e2){e2=leadexp(h[i])[2];}
    if(leadexp(h[i])[3]>e3){e3=leadexp(h[i])[3];}
  }
  for(i = 1; i <= size(h); i++)
  {
    L[i] = list(leadcoef(h[i]),leadexp(h[i]));
  }
  L1[1]=1;
  L2[1]=1;
  L3[1]=1;
  for(i=1;i<=e1;i++)
  {
    q1 = reduce(q1*r,I);
    L1[i+1]=q1;
  }
  for(i=1;i<=e2;i++)
  {
    q2 = reduce(q2*s,I);
    L2[i+1]=q2;
  }
  for(i=1;i<=e3;i++)
  {
    q3 = reduce(q3*var(3),I);
    L3[i+1]=q3;
  }
  int m=size(L);
  i = 1;
  h = 0;
  while(i <= m)
  {
    h=h+L[i][1]*L1[L[i][2][1]+1]*L2[L[i][2][2]+1]*L3[L[i][2][3]+1];
    i++;
  }
  h=reduce(h,I);
  return(h);
}

static proc semi2char(intvec v)
{
  intvec k=v[1..2];
  intvec w=v[1];
  int i,j,p,q;
  for(i=2;i<size(v);i++)
  {
    w[i]=gcd(w[i-1],v[i]);
  }
  for(i=3;i<=size(v);i++)
  {
    k[i]=v[i];
    for(j=2;j<i;j++)
    {
      k[i]=k[i]-(w[j-1] div w[j]-1)*v[j];
    }
  }
  return(k);
}
///////////////////////////////////////////////////////////////////////////////////////////////////
proc primparam(poly x,poly y,int c)
"USAGE:  MultiplicitySequence(x,y,c);  x poly, y poly, c integer
ASSUME:  x and y are polynomials in k(a)[t] such that (x,y) is a primitive parametrization of
         a plane curve branch and ord(x)<ord(y).
RETURN:  Hamburger-Noether Matrix of the curve branch given parametrically by (x,y).
EXAMPLE: example primparam;  shows an example
"
{
  int i,h;
  poly F,z;
  list L;
  while(ord(x)>1)
  {
    list v;
    while(ord(y)>=ord(x))
    {
      F=divide(y,x,c);
      if(ord(F)==0)
      {
        v=insert(v,subst(F,t,0),size(v));
        y=F-subst(F,t,0);
      }
      else
      {
        v=insert(v,0,size(v));
        y=F;
      }
    }
    v=insert(v,t,size(v));
    L=insert(L,transform(v),size(L));
    z=x;
    x=y;
    y=z;
    kill v;
  }
  if(ord(x)==1)
  {
    list v;
    while(i<c)
    {
      F=divide(y,x,c);
      if(ord(F)==0)
      {
        v=insert(v,subst(F,t,0),size(v));
        y=F-subst(F,t,0);
      }
      else
      {
        v=insert(v,0,size(v));
        y=F;
      }
      if(y==0)
      {
        v=insert(v,t,size(v));
        break;
      }
      i++;
    }
    L=insert(L,transform(v),size(L));
  }
  return(compose(L));
}
```

```
example
{
  "EXAMPLE:"; echo=2;
  ring r=(0,a),t,ds;
  poly x=t6;
  poly y=t8+t11;
  int c=15;
  primparam(x,y,c);
}

//////////////////////////////////////
//L is a list of polynomials
//////////////////////////////////////
static proc transform(list L)
{
  matrix m2;
  matrix m1=matrix(L[1]);
  for(int i=2;i<=size(L);i++)
  {
    m2=matrix(L[i]);
    m1=concat(m1,m2);
  }
  return(m1);
}
////////////////////////////////////////////////////////////////////
//L is a list of matrices
///////////////////////////////////////
static proc compose(list L)
{
  matrix M[ncols(L[1])][1]=transpose(L[1]);
  for(int i=2;i<=size(L);i++)
  {
    M=concat(M,transpose(L[i]));
  }
  return(transpose(M));
}
/////////////////////////////////////////////////////////////
static proc rduce(poly p)
{
  int n=ord(p);
  poly q=p/(t^n);
  return(q);
}
//////////////////////////////////////////////////////
static proc divide(poly p,poly q,int c)i
{
  int n=ord(p);
  int m=ord(q);
  poly p'=rduce(p);
  poly q'=rduce(q);
  poly r=t^(n-m)*p'*jet(1,q',c);
  return(jet(r,c));
}
///////////////////////////////////////////////////////
static proc contact(list L)
{
  def M=L[1];
  intvec v=L[2];
  int s,j,i;
  for(i=1;i<=size(v);i++)
  {
    if(v[i]<0){v[i]=-1-v[i];}
    for(j=1;j<=v[i];j++)
    {
      s=s+1;
      if(find(string(M[i,j]),"a")!=0){return(s);}
    }
  }
}
//////////////////////////////////////////////////////////////
static proc converter(list L)
{
def s=basering;
list D;
int i,c;
for(i=1;i<=size(L);i++)
   {D=insert(D,deg(L[i][2]),size(D));}
ring r=(0,a),(t),ds;
list L=imap(s,L);
poly x,y,z;
list A,B;
for(i=1;i<=size(L);i++)
   {A[5]=D[i];
   x=L[i][1][1];
   y=L[i][1][2];
   if(ord(x)<=ord(y)){A[3]=0;}
   else{A[3]=1;
       z=x;
       x=y;
       y=z;
       }
   c=bound(x,y);
   if(c==-1){ERROR("Bound is not enough");}
   A[1]=primparam(x,y,c);
   A[2]=lengths(A[1]);
   A[4]=0;
   B=insert(B,A,size(B));
   A=list();
   }
ring r1=(0,a),(x,y),ds;
list hne=fetch(r,B);
export(hne);
return(r1);
}
//////////////////////////////////////////////////////////
static proc intermat(list L)
{
    int s=size(L);
    intvec v=L[1][5];
    intvec w1;
    int i,j,d,b,l,k,c,o,p;
    for(i=2;i<=s;i++)
    {v=v,L[i][5];}
    intvec u=v[1];
    for(i=2;i<=s;i++)
    {
      l=u[size(u)]+v[i];
     u=u,l;
    }
    int m=u[size(u)];
    intmat M[m][m];
    for(i=1;i<=m;i++)
    {
      for(j=i+1;j<=m;j++)
      {
        d=sorting(u,i);
        b=sorting(u,j);
        if(d==b){k=contact(L[d]);
        w1=multsequence(L[d]);
        if(size(w1)<k){for(p=size(w1)+1;p<=k;p++)
        {w1=w1,1;} }
        for(o=1;o<=k;o++){c=c+w1[o]*w1[o];}
        M[i,j]=c;
        c=0;
      }
      else
      {M[i,j]=intersection(L[d],L[b]);}
    }
  }
  return(M);
}
/////////////////////////////////////////////////////////////////
static proc lengths(matrix M)
{
  intvec v;
  int s,i,j;
  for(i=1;i<=nrows(M);i++)
  {
    for(j=1;j<=ncols(M);j++)
    {
      if(M[i,j]==t)
      {
        v[i]=j-1;
        if(i==nrows(M)){s=1;}
        break;
      }
    }
  }
  if(s==0){v[nrows(M)]=-j;}
  return(v);
}
//////////////////////////////////////////////////////////////////////
static proc sorting(intvec u,int k)
{
  int s=size(u);
  int i;
  for(i=1;i<=s;i++)
  {
    if(u[i]>=k){break;}
  }
  return(i);
}
//////////////////////////////////////////////////////////////////////
proc MultiplicitySequence(ideal i)
"USAGE:  MultiplicitySequence(i); i ideal
ASSUME:  i is the defining ideal of a (reducible) plane curve singularity.
```

```
RETURN:  list X of charts. Each chart contains the multiplicity sequence of
         the corresponding branch.
EXAMPLE: example MultiplicitySequence;  shows an example
"
{
  def s=CurveParam(i);
  setring s;
  int j,k;
  def r1=converter(Param);
  setring r1;
  list Y=hne;
  list X;
  for(j=1;j<=size(Y);j++)
  {
    for(k=1;k<=Y[j][5];k++)
    {
      X=insert(X,multsequence(Y[j]),size(X));
    }
  }
  return(X);
}
example
{
  "EXAMPLE:"; echo = 2;
  ring r=0,(x,y),ds;
  ideal i=x14-x4y7-y11;
  MultiplicitySequence(i);
}
///////////////////////////////////////////////////////////////////////////
proc IntersectionMatrix(ideal i)
"USAGE:  IntersectionMatrix(i); i ideal
ASSUME:  i is the defining ideal of a (reducible) plane curve singularity.
RETURN:  intmat of the intersection multiplicities of the branches.
EXAMPLE: example IntersectionMatrix;  shows an example
"
{
  def s=CurveParam(i);
  setring s;
  int j,k;
  def r1=converter(Param);
  setring r1;
  list Y=hne;
  return(intermat(Y));
}
example
{
  "EXAMPLE:"; echo = 2;
  ring r=0,(x,y),ds;
  ideal i=x14-x4y7-y11;
  IntersectionMatrix(i);
}
/////////////////////////////////////////////////////////////////////////////
proc CharacteristicExponents(ideal i)
"USAGE:  CharacteristicExponents(i); i ideal
ASSUME:  i is the defining ideal of a (reducible) plane curve singularity.
RETURN:  list X of charts. Each chart contains the characteristic exponents
         of the corresponding branch.
EXAMPLE: example CharacteristicExponents;  shows an example
"
{
  def s=CurveParam(i);
  setring s;
  int j,k;
  def r1=converter(Param);
  setring r1;
  list X;
  list Y=hne;
  for(j=1;j<=size(Y);j++)
  {
    for(k=1;k<=Y[j][5];k++)
    {
      X=insert(X,invariants(Y[j])[1],size(X));
    }
  }
  return(X);
}
example
{
  "EXAMPLE:"; echo = 2;
  ring r=0,(x,y),ds;
  ideal i=x14-x4y7-y11;
  CharacteristicExponents(i);
}
///////////////////////////////////////////////////////////////////////////////
static proc contactNumber(int a,intvec v1,intvec v2)
{
//====  a is the intersection multiplicity of the branches
//====  v1,v2 are the multiplicity sequences
```

```
  int i,c,d;
  if(size(v1)>size(v2))
  {
    for(i=size(v2)+1;i<=size(v1);i++)
    {
      v2[i]=1;
    }
  }
  if(size(v1)<size(v2))
  {
    for(i=size(v1)+1;i<=size(v2);i++)
    {
      v1[i]=1;
    }
  }
  while((c<a)&&(d<size(v1)))
  {
    d++;
    c=c+v1[d]*v2[d];
  }
  if(c<a)
  {
    d=d+a-c;
  }
   return(d);
}
///////////////////////////////////////////////////////////////////////////////
proc ContactMatrix(ideal I)
"USAGE:  ContactMatrix(I); I ideal
ASSUME:  i is the defining ideal of a (reducible) plane curve singularity.
RETURN:  intmat N of the contact matrix of the braches of the curve.
EXAMPLE: example ContactMatrix;  shows an example
"
{
  def s=CurveParam(I);
  setring s;
  int j,k,i;
  def r1=converter(Param);
  setring r1;
  list Y=hne;
  list L;
  for(j=1;j<=size(Y);j++)
  {
    for(k=1;k<=Y[j][5];k++)
    {
      L=insert(L,multsequence(Y[j]),size(L));
     }
  }
  intmat M=intermat(Y);
  intmat N[nrows(M)][ncols(M)];
  for(i=1;i<=nrows(M);i++)
  {
  for(j=i+1;j<=ncols(M);j++)
  {
    N[i,j]=contactNumber(M[i,j],L[i],L[j]);
    N[j,i]=N[i,j];}
  }
  return(N);
}
example
{
  "EXAMPLE:"; echo = 2;
  ring r=0,(x,y),ds;
  ideal i=x14-x4y7-y11;
  ContactMatrix(i);
}
//////////////////////////////////////////////////////////////////////////////
proc plainInvariants(ideal i)
"USAGE:  plainInvariants(i); i ideal
ASSUME:  i is the defining ideal of a (reducible) plane curve singularity.
RETURN:  list L of charts. L[j] is the invariants of the jth branch and the last entry
         of L is a list containing the intersection matrix,contact matrix,resolution
         graph of the curve.
         L[j][1]:     intvec (characteristic exponents of the jth branch)
         L[j][2]:    intvec (generators of the semigroup of the jth branch)
         L[j][3]:    intvec (first components of the puiseux pairs of the jth branch)
         L[j][4]:    intvec (second components of the puiseux pairs of the jth branch)
         L[j][5]:    int    (degree of conductor of the jth branch)
         L[j][6]:    intvec (multiplicity sequence of the jth branch.
         L[last][1]: intmat (intersection matrix of the branches)
         L[last][2]: intmat (contact matrix of the branches)
         L[last][3]: intmat (resolution graph of the curve)
SEE ALSO: MultiplicitySequence, CharacteristicExponents, IntersectionMatrix,
          ContactMatrix
EXAMPLE: example plainInvariants;  shows an example
"
{
  def s=CurveParam(i);
  setring s;
  int j,k;
  def r1=converter(Param);
  setring r1;
  list Y=hne;
  list L,X,Q;
  for(j=1;j<=size(Y);j++)
  {
    for(k=1;k<=Y[j][5];k++)
    {
      L=insert(L,invariants(Y[j]),size(L));   //computes the same thing again
      X=insert(X,invariants(Y[j])[1],size(X));
    }
  }
  L=insert(L,list(),size(L));
  L[size(L)]=insert(L[size(L)],intermat(Y),size(L[size(L)]));
  intmat N[nrows(intermat(Y))][ncols(intermat(Y))];
  for(k=1;k<=nrows(intermat(Y));k++)
  {
    for(j=k+1;j<=ncols(intermat(Y));j++)
    {
      N[k,j]=contactNumber(intermat(Y)[k,j],L[k][6],L[j][6]);
      N[j,k]=N[k,j];
    }
  }
  L[size(L)]=insert(L[size(L)],N,size(L[size(L)]));
  Q=L[size(L)][2],X;
  L[size(L)]=insert(L[size(L)],resolutiongraph(Q),size(L[size(L)]));
  return(L);
}
example
{
  "EXAMPLE:"; echo = 2;
  ring r=0,(x,y),ds;
  ideal i=x14-x4y7-y11;
  plainInvariants(i);
}
///////////////////////////////////////////////////////////////////////////////////
static proc bound(poly x,poly y)
{
  poly z=x+y;
  int m=ord(z);
  int i;
  int c=-1;
  for(i=2;i<=size(z);i++)
  {
    if(gcd(m,leadexp(z[i])[1])==1)
    {
      c=2*leadexp(z[i])[1];
      break;
    }
    else
    {
      m=gcd(m,leadexp(z[i])[1]);
    }
  }
  return(c);
}
////////////////////////////////////////////////////////////////////////////////////
static proc boundparam(poly f)
{
  int i;
  int l=size(f);
  int m=leadexp(f[l])[1];
  for(i=l-1;i>=1;i--)
  {
    if(gcd(m,leadexp(f[i])[1])==1)
    {
      i=i-1;
      break;
    }
    else
    {
      m=gcd(m,leadexp(f[i])[1]);
    }
  }
  return(2*leadexp(f[i+1])[1]);
}
```

APPENDIX B

The Singular Library ”ArfClosure.lib”

```
///////////////////////////////////////////////////////////////////////////////////////////
version="$Id:   $";
category="Singularity Theory";
info="
LIBRARY:  space_curve.lib

AUTHOR:     Feza Arslan       email: feza.arslan@msgsu.edu.tr
            Nil Sahin         email: e150916@metu.edu.tr


PROCEDURES:
Blwup(L,c);                   BlowingUp of the ring generated by the elements of L
Arfjet(L,c);                  Arf Closure of the ring generated by the elements of L in
                              simplest form
genlist(L,c);                 Generators of the maximal ideal of the Arf Closure of the
                              ring generated by the elements of L
Puiseux(x,y);                 Puiseux exponents of the curve generated by (x(t),y(t))
PuiseuxToChar(L);             Converts puiseux exponents to characteristic exponents
";

LIB "sing.lib";
LIB "qhmoduli.lib";

/////////////////////////////////////////////////////////////////
//-------------Computing the Arf Closure--------------------//
/////////////////////////////////////////////////////////////////
proc Blwup(list L,intc)
"USAGE:   Blwup(L,c);
L=list
c=integer

ASSUME:   The basering is r=0,t,ds
L is a list of polynomials in r

COMPUTE:  First blowing up of the ring with for which has the maximal ideal generated
by the elements of L.

RETURN:   list of polynomials which generates the maximal ideal of the blowing up

EXAMPLE: example BlowingUp; shows an example"
{list F=list();
int s=size(L);
int i=1;
while(i<=s){int s_i=ord(L[i]);
def A=F;
def F=insert(A,s_i);
i++;}


list E=list();
int j=1;
int m=Min(F);
while(j<=s){if(ord(L[j])==m){int k=1;
while(k<=s){if(k!=j){def K=insert(E,divide(L[k],L[j],c));
def E=K;}
else{def K=insert(E,L[j]);
def E=K;}
k++;}
break;}
else{j++;}}
return(E);
}
example
{
"EXAMPLE:";
"ring r=0,t,ds;
list L=t^3,t^5
int c=10;
Blwup(L,c);";

ring r=0,t,ds;
list L=t^3,t^5
int c=10;
Blwup(L,c);
}
////////////////////////////////////////////////////////////////
static proc rduce(poly p){
int n=ord(p);
poly q=p/(t^n);
return(q);
}
//////////////////////////////////////////////////////////////
static proc invers(poly p,int k)
{
poly q=1/p[1];
poly re=q;
p=q*(p[1]-jet(p,k));
poly s=p;
while(p!=0)
{re=re+q*p;
p=jet(p*s,k);}
return(re);
}
//////////////////////////////////////////////////////////////
static proc divide(poly p,poly q,int c)
//Assume the order of p is greater than or equal to the order of q
//Divides p by q till the terms of power c.
{
int n=ord(p);
int m=ord(q);
poly p'=rduce(p);
poly q'=rduce(q);
poly r=t^(n-m)*p'*invers(q',c);
return(jet(r,c));
}
//////////////////////////////////////////////////////////////
static proc sub(list L)
//Assume L is a list of polynomials
//Subtracts the constant terms from the entries of L.
{
int s=size(L);
int i=1;
while(i<=s){
if(leadmonom(L[i])==1){L[i]=L[i]-lead(L[i]);}
i++;}
return(L);
}
//////////////////////////////////////////////////////////////
static proc mmbrshp(list L,poly f)
// returns 1, if f is an elemnt of L; returns 0 if not.
{
int s=size(L);
int i=1;
while(i<=s){if(L[i]==f){break;}
else{i++;}
}
if(i==s+1){return(0);}
else{return(1);}
}
//////////////////////////////////////////////////////////////
static proc Orders(list L)
//Assume L is a list of polynomials
//Returns a new list consisting of the orders of the elements of L(in reverse order)
{
list F=list();
int s=size(L);
int i=1;
while(i<=s){int s_i=ord(L[i]);
def A=F;
def F=insert(A,s_i);
i++;}
return(F);
}
//////////////////////////////////////////////////////////////
static proc MinOrdElt(list L)
//Assume L is a list of polynomials
//Returns minimum ordered element of L
{
int m=Min(Orders(L));
poly f=FindOr(L,m);
return(f);
}
```

```
///////////////////////////////////////////////////////////////
static proc irredundant(list L)
//Deletes the 0's from the list L
{
poly f=0;
while(mmbrshp(L,f)==1){
int j=Place(L);
def T=L;
def L=delete(T,j);
}
return(L);}
///////////////////////////////////////////////////////////////
static proc Place(list L)
//L is a list of polynomials
//Returns the index of the minimum ordered element of L
{
int j=1;
while(ord(L[j])!=Min(Orders(L))){j++;}
return(j);
}
///////////////////////////////////////////////////////////////
static proc Arfclosure(list L,int c)
//Assume L is a list of polynomials and r=0,t,ds is the base ring
{
poly g=1;
list X=list();
while(Min(Orders(irredundant(sub(L))))!=1){
def K=irredundant(sub(L));
def L=Blwup(K,c);
poly h=g;
poly g=h*MinOrdElt(K);
def Y=X;
def X=insert(Y,g);
}
return(X);
}
///////////////////////////////////////////////////////////////
proc Arfjet(list L,int c)
"USAGE:   Arfjet(L,c);
L list
c integer

ASSUME:   The basering is r=0,t,ds
L is a list of polynomials.

COMPUTE:  Arf Closure of the local ring whose maximal ideal is generated by the elements of L.

RETURN:   A list S of polynomials of degree less than or equal to c
If the output is S=S[1],S[2],...,S[m], the Arf Closure is k+k S[1]+...+k S[m]

EXAMPLE: example Arfjet; shows an example"
{
list S=Arfclosure(L,c);
int s=size(S);
poly f=S[1];
int k=ord(f);
int i=1;
while(i<=s){S[i]=jet(S[i],k);
i++;}
return(S);
}
example
{
"EXAMPLE:";

"ring r=0,t,ds;
list L=t5,t7+t9,t11;
int c=17;
Arfjet(L,c);
";

ring r=0,t,ds;
list L=t5,t7+t9,t11;
int c=17;
Arfjet(L,c);
}
///////////////////////////////////////////////////////////////
proc genlist(list L,int c)
"USAGE:   genlist(L,c);
L list
c integer

ASSUME:   The basering is r=0,t,ds
L is a list of polynomials.

COMPUTE:  Generetors of the Arf Closure of the local ring whose maximal ideal is generated
by the elements of L.
```

```
RETURN:   A list S of polynomials of degree less than or equal to c
If the output is K=K[1],K[2],...,K[m], the Arf Closure is k[[K[1],...,K[m]]].

EXAMPLE: example genlist; shows an example"
{
list C=list();
list A=Arfjet(L,c);
int s=size(A);
int i=1;
while(i<=s){list B=insert(C,ord(A[i]));
def C=B;
i++;}
def D=reorder(B);
int n=Min(D);
int h=D[s];
int j=1;
while(j<h){int k=D[1];
def E=insert(D,k+1);
def D=E;
j++;}
list F=generator(reorder(D));
int k=1;
list M=list();
while(k<=size(F)){int p=F[k];
if(mmbrshp(Orders(A),p)==0){list K=insert(M,t^p);
def M=K;}
else{poly q=FindOr(A,p);
list K=insert(M,q);
def M=K;}
k++;}
return(K);}
example
{
"EXAMPLE:";

"ring r=0,t,ds;
list L=t5,t7+t9,t11;
int c=17;
genlist(L,c);
";

ring r=0,t,ds;
list L=t5,t7+t9,t11;
int c=17;
genlist(L,c);
}
///////////////////////////////////////////////////////////////////////////
static proc FindOr(list L,int n)
//Gives the first element of L of order n
{
int s=size(L);
int i=1;
while(i<=s){if(ord(L[i])==n){break;}
else{i++;}
}
return(L[i]);
}
////////////////////////////////////////////////////////////////////////
static proc generator(list L)
//Assume L is a list of integers which is sorted in increasing order
//Finds an Apery set for L with respect to L[1]
{
int s=size(L);
int m_1=Min(L);
int i=2;
list M=0;
list G=m_1;
while(i<=s){int m_i=L[i] mod m_1;int t=mmbrshp(M,m_i);
if(t==1){i++;}
else{def T=insert(M,m_i);
def M=T;
def K=insert(G,L[i]);
def G=K;
i++;}}
return(G);
}
////////////////////////////////////////////////////////////////////////
static proc reorder(list L)
//aligns the elements of L in the revers order. That is, first element of
//L becomes the last and so on.
{
int s=size(L);
int i=2;
int k=L[1];
list E=k;
while(i<=s){def T=E;
def E=insert(T,L[i]);
i++;
```

```
}
return(E);
}
//////////////////////////////////////////////////////////////
//-------------Finding the Puiseux Expansion----------------//
//////////////////////////////////////////////////////////////
proc Puiseux(poly x,poly y)
"USAGE:   Puiseux(x,y);
x polynomial
y polynomial

ASSUME:   Base ring is r=0,t,ds.

COMPUTE:  Puiseux form of the parametrization (x(t),y(t)) by using the
Borodzik's Algorithm.

RETURN:   list L containing the powers of the terms having nonzero coefficients
in the Puiseux Expansion.

EXAMPLE: example Puiseux; shows an example"
{
poly P0=y;
number p=ord(x);
number q=ord(y);
list L=p,q;
number p1=gcd(p,q);
if(p1!=1)
{
poly x'=diff(x,t);
poly y'=diff(y,t);
poly x''=diff(x',t);
poly P1=y'*x-x'*y*(q/p);
number r1=ord(P1)-(p-1);
L[3]=r1;
int k=1;
number p2=gcd(p1,r1);
poly P2;
number r2;
while(p2!=1)
{
P2=diff(P1,t)*x*x'-x'^2*(r1/p)*P1-(2*k-1)*(x'')*x*(P1);
r2=ord(P2)-(2*k+1)*(p-1);
L[3+k]=r2;
p2=gcd(p2,r2);
P1=P2;
r1=r2;
k=k+1;
}
}
return(L);
}
example
{
"EXAMPLE:";
"ring r=0,t,ds;
poly x=t12+t13+(37/28)*t14;
poly y=t18+(3/2)*t19+(33/14)*t20+(13/14)*t21+(675/1568)*t22-(675/3136)*t23;
Puiseux(x,y);
";
ring r=0,t,ds;
poly x=t12+t13+(37/28)*t14;
poly y=t18+(3/2)*t19+(33/14)*t20+(13/14)*t21+(675/1568)*t22-(675/3136)*t23;
Puiseux(x,y);
}
/////////////////////////////////////////////////////
proc PuiseuxToChar(list L)
"USAGE:   PuiseuxToChar(L);
L list

ASSUME:   L is the list of the powers in a Puiseux expansion

COMPUTE:  Converts the Puiseux powers to the characteric exponents.

RETURN:   list L containing the characteristic exponents.

EXAMPLE: example PuiseuxToChar; shows an example"
{
list A;
A[1]=L[1];
int i=2;
int r=int(L[1]);
int p;
while(r!=1)
{
p=gcd(r,int(L[i]));
if(p<r)
{
A[size(A)+1]=L[i];
r=p;
}
i++;
}
return(A);
}
example
{
"EXAMPLE:";
"list L=6,12,18,21,25;
PuiseuxToChar(L);";
list L=6,12,18,21,25;
PuiseuxToChar(L);
}
///////////////////////////////////////////////////
```

VITA

PERSONAL INFORMATION

Name, Surname: Nil Şahin

Date and Place of Birth: 19 September 1983, Ankara

Nationality: Turkish (TC)

Phone: (+90) 507 5797263

E-mail : e150916@metu.edu.tr

EDUCATION

Degree	Institution	Year
Visiting Scholar	Technische Universitat Kaiserslautern, Department of Mathematics	2011
BS	Ankara University, Department of Mathematics	2001-2005
High School	Çankaya High School	1997-2001

FOREIGN LANGUAGES

English (Fluent), German (Beginner)

RESEARCH INTERESTS

Computational Algebraic Geometry and Commutative Algebra

- Arf Rings and their Hilbert Functions,
- Gluing of Monomial Curves and their Hilbert Functions,
- Invariants of Plane Algebraic Curves

SCHOLARSHIPS

- Ph. D. Scholarship Program, TUBITAK, 2005 - 2010
- Research Scholarship for Ph.D. students, TUBITAK, march 2011-december 2011

PUBLICATIONS, WRITTEN CODES

- Symmetric Numerical Semigroup Families Supporting Rossi's Conjecture (submitted)
- curvepar.lib , SINGULAR 3-1-4 Library for Curve Resolutions