ALGORITHMIC ASPECTS OF POLYNOMIAL RESIDUE CLASS RINGS

by

Stuart Carl Schaller

Computer Sciences Technical Report #370

October 1979

ALGORITHMIC ASPECTS OF POLYNOMIAL RESIDUE CLASS RINGS

by

STUART CARL SCHALLER

A thesis submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN - MADISON

1979

To Charmian and Edward

ACKNOWLEDGEMENTS

I would like to express my appreciation to the following people: to Prof. George E. Collins, my advisor, for his continual advice and guidance during the preparation of this thesis; to Prof. Bobby F. Caviness who originally introduced me to the problems discussed in this thesis; to Tsu-wu Chow, Dennis Arnon, and Markus Lauer for many enlightening discussions; and finally to Prof. John C. Thompson and Dr. Harold S. Butler for encouragement and moral support. I also owe an unpayable debt to my wife, Charmian, for her patience and support.

I would also like to acknowledge the financial support of the National Science Foundation and the University of Wisconsin.

ALGORITHMIC ASPECTS OF POLYNOMIAL RESIDUE CLASS RINGS

Stuart C. Schaller

ABSTRACT

This thesis investigates algorithmic aspects of polynomial residue class rings giving special attention to the problem of simplifying polynomial equations in the presence of algebraic relations among the variables. Algorithms are described for transforming polynomial ideal bases to allow the computation of canonical residue class representatives through the use of simplifying ample functions. The fundamental theory is given a new treatment in a general setting using the concept of simple constructibility. It is shown how resultant systems may be used during the basis transformation yielding, for certain polynomial rings, polynomially bounded computing times. For the case of polynomials over a field, detailed algorithm descriptions are presented. Empirical results are presented for an implementation based on the SAC-2 computer algebra system.

TABLE OF CONTENTS

CHAPTER ONE:

Introduction

1.1 The Problem of Simplification in Computer Algebra

In symbolic mathematical computations one often desires to simplify a mathematical expression by taking into account relations among the variables involved. These relations may be mathematical identities such as $\cos^2 x + \sin^2 x = 1$, or they may be identities expressing physical laws such as conservation of quantum numbers in high energy physics, equations of state in thermodynamics, or basic dynamical laws in celestial mechanics. In each instance the aim is to use the identities or laws to find a simplified form for the expression being considered. By "simplified form" one might mean that the expression is more readable, that it contains more physical meaning, that it is more tractable mathematically, or even that it just consists of fewer symbols. In some cases one may wish to realize many or all of these goals -- a wish that cannot often be fulfilled.

This problem -- the simplification of expressions in the presence of side relations -- is part of the larger problem of simplification in the field of computer algebra. The general problems of simplification have been discussed elsewhere, (see [CAV70], [FAT72], and [MOS71]). In [CAV70], Caviness formalized the notion of a canonical form for a class of expressions and showed, following Richardson [RCD68], that for a sufficiently rich class of expressions, the problem of finding a canonical form is recursively undecidable.

In this thesis we are interested in a portion of the simplification problem known to be decidable. Namely we are concerned with the problem of doing unique simplification in the ring of multivariate polynomials over some effectively given coefficient domain in the presence of algebraic relations among the variables.

Section 1.2 introduces the notions of canonical form and simplifying ample function in order to give a precise meaning to simplification when a set and an equivalence relation on that set are given. Section 1.3 specializes these ideas to the case of polynomials with an equivalence relation defined by congruence modulo an ideal. Sections 1.4 and 1.5 discuss past work in this area and possible applications, respectively. Section 1.6 deals with basic ideas concerning the presentation and analysis of algorithms in this thesis.

Chapter 2 shows how simplification in polynomial rings actually hinges on the computation of a special form of ideal basis (called "complete"). This chapter contains a new formulation of the basic theory using the concept of "simple constructibility." Chapter 3 describes an implementation of the ideal basis transformation algorithms in the case where the coefficient ring is a field. Chapter 4 discusses greatest common divisor and generalized resultant algorithms for several (i.e. more than two) polynomials, and the application of these algorithms to the problem of computing complete bases. Chapter 5 includes empirical investigations in the polynomial ring $GF(p)[x,y]$. Chapter 6 concludes with a short summary.

1.2 Canonical Forms and Simplifying Ample Functions

As a first step in refining our ideas about simplification in the presence of an equivalence relation we introduce Lauer's concept of a simplifying ample function, [LAU76a] and relate it to Caviness' definition of a canonical form.

Let us consider the ordered pair $(S,\equiv)$ where S is a non-empty set and $\equiv$ is an equivalence relation on S (i.e. $\equiv$ is reflexive, symmetric, and transitive). The relation $\equiv$ partitions S into equivalence classes. If a subset A of S contains exactly one element of each of these equivalence classes, we say that A is an ample set for the ordered pair $(S,\equiv)$, [MUS71]. A function σ mapping S into a subset A of S is called an ample function for $(S,\equiv)$ if A is an ample set for $(S,\equiv)$ and for all elements s in S

$$\sigma(s) \equiv s. \tag{1.2.1}$$

Furthermore, we can see that if σ is an ample function for $(S,\equiv)$ then for all s and t in S

$$\text{if } s \equiv t \text{ then } \sigma(s) = \sigma(t). \tag{1.2.2}$$

In fact, properties (1.2.1) and (1.2.2) taken together are sufficient to guarantee that $A = \text{range}(\sigma)$ is an ample set for $(S,\equiv)$. We shall often assume that an ample function, σ, is defined by these two properties without specific reference to the associated ample set. In addition, when no ambiguity is possible we also drop the explicit reference to the ordered pair $(S,\equiv)$.

For a moment we change our point of view and consider the set S* of equivalence classes of S generated by $\equiv$. An element s* of S* can be thought of as having a large number of equivalent representations. Hence, since σ satisfies (1.2.1) and (1.2.2), it is a canonical form for S in a slightly more general sense than [CAV70]. To put this in Caviness' terms S is a class of expressions given by a set of rules for determining the set of well-formed expressions in the class. S* is a set of functions determined by interpreting the expressions over some domain. The relation $\equiv$ is determined by functional equivalence. Since we have defined no algebraic structure on S, we omit the part of Caviness' definition which requires that $\sigma(0) = 0$. Lastly, we need to require that σ be computable, a specification we have omitted up to now.

Thus either ample functions or canonical forms can serve as a starting point in defining what we mean by the phrase "simplification of elements of S with respect to the equivalence relation $\equiv$." The ample function $\sigma(s)$ selects a unique representative of the equivalence class composed of all elements of S which are equivalent to s under $\equiv$.

In general, however, there are many possible ample sets and hence ample functions for a given pair $(S,\equiv)$. Any one of these ample functions provides an unambiguous description of the simplification of elements of S with respect to $\equiv$.

Many of the ample functions possible for the pair $(S,\equiv)$, however, do not reflect the usual notions of what is meant by

"simplification." If S has an associated partial order relation, $\leq$, (i.e. reflexive, anti-symmetric, and transitive) we can limit our attention in a natural way to ample functions which for all s in S satisfy

$$\sigma(s) \leq s. \tag{1.2.3}$$

A function which satisfies (1.2.1), (1.2.2), and (1.2.3) will be called a simplifying ample function for the triple $(S,\equiv,\leq)$, [LAU76a]. The partial order on S thus gives an analytic basis for our concept of simplification.

The justification for calling such a function a simplifying ample function is that the element it selects as a representative of a given residue class is minimal with respect to the partial order. This is expressed as follows:

Lemma 1.2.1 If σ is a simplifying ample function for $(S,\equiv,\leq)$ and if $s \equiv t$ then $\sigma(s) \leq t$.

Proof $\sigma(S) = \sigma(t) \leq t$. □

Since in general many partial orderings may be defined on S, there are many different ways of specifying, in this manner, what we mean by "simplification" of elements of S with respect to $\equiv$. However, as Lauer observes [LAU76a], once a partial order is specified, it uniquely determines the associated simplifying ample function.

Lemma 1.2.2 If σ_1 and σ_2 are both simplifying ample functions for $(S,\equiv,\leq)$, then for all s in S

$$\sigma_1(s) = \sigma_2(s).$$

Proof Since σ_1 and σ_2 are both ample functions we know that $\sigma_1(s) \equiv s \equiv \sigma_2(s)$ for all s in S. However, since σ_1 and σ_2 are simplifying ample functions Lemma 1.2.1 shows that $\sigma_1(s) \leq \sigma_2(s)$ and similarly $\sigma_2(s) \leq \sigma_1(s)$. Hence $\sigma_1(s) = \sigma_2(s)$ for all s in S.□

As an example let us take S = Z, the ring of integers, and consider the equivalence relation defined by congruence modulo m for some positive integer m. We will write this relation as $a \equiv b \bmod m$ if and only if $m|a-b$. Let rem(a,m) be the remainder of a divided by m, i.e. $a = q \cdot m + \text{rem}(a,m)$ with $0 \leq \text{rem}(a,m) < m$. For a fixed m we see that $\sigma_0(a) = \text{rem}(a,m)$ is an ample function for the pair $(Z, \equiv \bmod m)$. Indeed, for any integer n the function $\sigma_n(a) = \text{rem}(a,m) + n \cdot m$ is also an ample function for $(Z, \equiv \bmod m)$. Now let us define a partial ordering, $\leq^*$, on Z as follows.

$$a <^* b \text{ iff } \begin{cases} a = 0 \text{ and } b \neq 0 \\ a > 0 \text{ and } b < 0 \\ \text{sign}(a) = \text{sign}(b) \text{ and } |a| < |b| \end{cases}$$

where < is the normal ordering of the real line. We write $a \leq^* b$ if $a <^* b$ or a = b. One can easily verify that $\sigma_0(a)$ is a simplifying ample function for the triple $(Z, \equiv \bmod m, \leq^*)$. The ample set associated with $\sigma_0(a)$ is the set of integers $A_0 = \{0,1,2,\ldots,m-1\}$, in fact this set is often used to represent Z/(m), the residue class ring of the integers modulo m.

As a final note before turning to the specification of simplifying ample functions in rings of polynomials we point out that the specification of a partial order does not guarantee the existence of a simplifying ample function. In particular, it can be seen that the normal ordering of the real line does not allow a simplifying ample function for the integers modulo m. This is because in any given equivalence class there is no minimal element, a consequence of the fact that the integers are not well-ordered by this ordering. Note, however, that in general it is not necessary for S to be well-ordered by $\leq$ in order to have a simplifying ample function.

In summary then, simplifying ample functions give us a way to uniquely specify what we mean when we wish to simplify elements of a set with respect to an equivalence relation and a partial order.

1.3 Specialization to Polynomial Ideal Theory

We now cast our previous discussion in a more concrete form. Our problem is that of simplifying polynomial expressions in the presence of algebraic relations among the variables involved. In particular, we would like to be able to recognize when two expressions are equivalent as determined by the given relations. We formulate this problem in terms of finding a simplifying ample function for a polynomial ring with respect to an equivalence relation over the ring generated by the algebraic relations among the variables.

Let R be a ring and let $R[x_1,\ldots,x_r]$ be the ring of polynomials in the r indeterminates $x_1,\ldots,x_r$. We will often abbreviate $R[x_1,\ldots,x_r]$ by R[x]. We let R[x] be our set S in the definition of a simplifying ample function. Since we will need to recognize when two elements of R[x] are identically equal, we assume the existence of a canonical form for elements of R[x].

We assume further that the relations among the indeterminates $x_1,\ldots,x_r$ are given by a set of algebraic equations:

$$A_i(x_1,\ldots,x_r) = 0,\ 1 \leq i \leq n\ , \qquad (1.3.1)$$

where for $1 \leq i \leq n$, A_i is in R[x]. We may reasonably interpret the equivalence of two polynomials P and Q in R[x] to mean that their difference can be shown to be the sum of multiples of the polynomials $A_1,\ldots,A_n$. In other words P and Q are equivalent if their

difference

$$P - Q = \sum_{i=1}^{n} S_i A_i$$

where S_i is an element of $R[x]$ for $1 \leq i \leq n$. Since $A_i = 0$ for $1 \leq i \leq n$, this means that $P - Q = 0$.

If we let $\overline{A} = \{ \sum_{i=1}^{n} S_i A_i : S_i \in R[x] \} \subseteq R[x]$ then we see that P and Q are equivalent in this interpretation if and only if $P - Q$ is an element of $\overline{A}$. But $\overline{A}$ is just the ideal in $R[x]$ generated by the ideal basis $\{A_1,\ldots,A_n\}$. Our equivalence relation thus is equivalence modulo the ideal $\overline{A}$. Expressed formally we say for polynomials P and Q in $R[x]$,

$$P \equiv Q \bmod \overline{A} \text{ iff } P - Q \in \overline{A}. \qquad (1.3.2)$$

Thus we have our equivalence relation which we write $\equiv \bmod \overline{A}$.

Finally, we need to introduce a partial ordering, $\leq$, on the polynomial ring $R[x]$. This will be done in Chapter 2 by specifying a partial order on the monomials (or products of indeterminates) contained in $R[x]$ (see Definition 2.2.2).

With a set, an equivalence relation, and an ordering specified we can now restate the definition of a simplifying ample function for the triple $(R[x], \equiv \bmod \overline{A}, \leq)$.

Definition 1.3.1 The function $\sigma: R[x] \to R[x]$ is a simplifying ample function for $(R[x], \equiv \bmod \overline{A}, \leq)$ if for all P and Q in $R[x]$

(a) $\sigma(P) \equiv P \bmod \overline{A}$,

(b) if $P \equiv Q \bmod \overline{A}$ then $\sigma(P) = \sigma(Q)$, and

(c) $\sigma(P) \leq P$.

Our task in the remainder of this thesis is, first, to show under what circumstances it is possible to find a computable simplifying ample function for $(R[x], \equiv \bmod \overline{A}, \leq)$. In particular we would like to know what restrictions are required on the ring R and the ordering relation $\leq$. Our second task is to investigate the practicality of special cases of the resulting algorithms.

1.4 Past Work

The approach to algebraic simplification discussed in the previous section relies on finding constructive methods to answer questions in polynomial ideal theory. During the first third of this century many people investigated explicit constructions in ideal theory, but as Lazard [LAZ76] points out, the limits of practical computability were quickly reached. Characteristic of this work was Macaulay's The Algebraic Theory of Modular Systems [MAC16] which formed the basis of what later became elimination theory. In 1926 Hermann [HER26] gave explicit bounds on the number of steps required for several ideal theoretic constructions in rings composed of multivariate polynomials over a field. In particular, she gave a finite method for deciding when a given polynomial was contained in a given ideal. This work was later corrected and extended by Seidenberg (see below).

During the second third of this century constructive problems in ideal theory gave way to the more abstract methods of algebraic geometry. One exception was Szekeres [SZK52] who in 1952 presented a constructive method of finding canonical bases for ideals formed by univariate polynomials over a principal ideal domain. In 1970 Buchberger [BUC70] gave a generalization of work he did in 1965 [BUC65] which included an algorithm for computing a vector space basis for the residue class ring of an ideal in the ring of multivariate polynomials over a field. This algorithm also provided a procedure

for deciding when a polynomial is contained in a given ideal.

The rapid expansion in the application of digital computers to symbolic mathematics during the late 1960's revived interest in several neglected areas of constructive mathematics and, in particular, in polynomial ideal theory. In 1966 Kleiman [KLE66] proposed a canonical form for rational functions in several algebraically dependent variables over a field, and gave an algorithm to put a given polynomial in this form. The algebraic relations in this case must generate a prime ideal, however, making this approach less than satisfactory.

A technical report by Shtokhamer [SHT76] in 1976 generalized the work of Szekeres. Shtokhamer showed that if R is a Noetherian ring in which residue class representatives can be constructed, then a similar construction is possible in the ring of univariate polynomials over R.

The efforts of Buchberger and Shtokhamer were joined in a 1976 paper by Lauer [LAU76a] which, in addition, showed how the computation of residue class representatives allows the specification of simplifying ample functions. Shortly thereafter, Lauer [LAU76b] generalized Buchberger's work to allow the computation of residue class representatives for ideals in the ring of multivariate polynomials over a Euclidean domain.

In the same year Buchberger revised and extended his original presentation [BUC76a, BUC76b, BUC76c]. Recently, Kollreider and

Buchberger [KOB78] have given an improved version of the basic algorithm for the case of multivariate polynomials over a field.

The most general formulation to date is that of Trinks [TRI78]. He presented an algorithm for computing residue class representatives for ideals in the ring of multivariate polynomials where the coefficient ring satisfies Shtokhamer's criteria. Chapter 2 of this thesis is a reformulation of this general case.

The central algorithm in each of the papers since 1960 performs an ideal basis transformation. The termination proof in each case relies, implicitly or explicitly, on the ascending chain condition for Noetherian rings. Bounds on the length of ascending chains are not well understood even in simple cases. Some work on this has been done by Seidenberg [SEI56,SEI71], but the bounds obtained are not primitive recursive functions. Heuristic arguments for "efficient" algorithms have been given in [BUC70] and [KOB78]. Some computational studies have been reported by Buchberger [BUC65], Schrader [SCH76], and Trinks [TRI78].

Co-temporaneous with the revival of interest in constructive ideal theory by those interested in computer algebra has been the increase of attention to constructive aspects of algebra by pure mathematicians. As mentioned above, Seidenberg [SEI74] revised and extended Hermann's work, giving detailed attention to strictly constructive algorithms for operations on ideals in rings of multivariate polynomials over a field. In [RCM74] Richman solves similar

problems for polynomials over a wider class of coefficient rings; in particular, his work applies to multivariate polynomials over the integers. A summary of these results and comments on the optimization of the associated algorithms is given in Lazard [LAZ76].

Richman and Seidenberg do not consider the problem of computing representatives of ideal residue classes nor do they deal with the computation of canonical ideal bases. The work originating with Buchberger and Szekeres, however, deals directly with these questions, and, thus is immediately applicable to the problems of simplification.

As a final note, the problem of determining canonical forms for a wide variety of algebraic structures is discussed by Bergman [BER78]. He deals with the general case of associative algebras over commutative rings given a presentation of the algebra by a family of generators and a family of relations. He discusses the criteria which must be satisfied to uniquely "reduce" (or "simplify") elements of the algebra with respect to the relations. The general form of the algorithm of Buchberger appears as a special case of Bergman's work.

1.5 Applications

The original motivation for this study grew out of interest in extending the algebraic domains in which the zero recognition problem in algebraic simplification is decidable. This motivation is the reason for the introduction of the concept of a simplifying ample function in Sections 1.2 and 1.3. The simplifying ample function guarantees that we can use the information given by a set of algebraic relations to obtain a unique "simplified" form for a polynomial expression.

This area is not the only application of the concepts and algorithms discussed in this thesis. There are also applications to computations in residue class rings, to solving systems of polynomial equations, and to constructing the primary decomposition of ideals.

The equivalence relation $\equiv \bmod \overline{A}$ (using the notation of Section 1.3) partitions the polynomial ring R[x] into disjoint equivalence (or residue) classes. As is well known, [VDW70a], these classes form a ring called the residue class ring of R[x] modulo ideal $\overline{A}$, denoted by $R[x]/\overline{A}$. Addition and multiplication in $R[x]/\overline{A}$ are defined in terms of the addition and multiplication of elements of the residue classes. A computable ample function allows one to uniquely represent each residue class and thus to effectively perform computations in $R[x]/\overline{A}$. If we use a simplifying ample function, the residue class representatives will be minimal in the sense specified by the partial ordering on R[x].

Now assume that R is a field and consider the finite ideal basis for $\overline{A}$ as a system of polynomial equations for which we desire the solutions. In this case $R[x]/\overline{A}$ can be interpreted as a vector space over R and if $\overline{A}$ is a zero dimensional ideal, then $R[x]/\overline{A}$ is a finite dimensional vector space. In [BUC70] it is shown how the residue class representatives for a certain finite set of monomials in R[x] can be used to compute a series of polynomials $P_1, P_2, \ldots, P_r$ with $P_i \in R[x_1, \ldots, x_i]$ for $1 \leq i \leq r$, which can be successively solved. The set of solutions thus derived can be shown to contain the set of solutions for the system of polynomials generating $\overline{A}$. Buchberger also extends this technique to the case of systems of polynomials which generate ideals of dimension greater than zero.

If σ is an ample function for $(R[x], \equiv \bmod \overline{A})$, then it is possible to determine whether an arbitrary polynomial P in R[x] is contained in $\overline{A}$ merely by comparing $\sigma(P)$ and $\sigma(0)$. Furthermore, if we are given another ideal $\overline{A}'$ presented by a finite basis, then we can determine whether $\overline{A}' \subseteq \overline{A}$ by testing if each of the basis elements which generate $\overline{A}'$ is in $\overline{A}$. Schrader, [SCH76], uses this property of an ample function in an algorithm which computes the primary decomposition of an arbitrary ideal.

1.6 Algorithm Presentation and Analysis

The algorithms in Chapters 3 and 4 and in Appendix B of this thesis are presented using the Aldes language for Algorithm Description developed by Loos and Collins [LOS76]. The algorithms in Chapter 2 use an informal varient of this language.

The SAC-2 System for Symbolic and Algebraic Computation [COL78] has been written entirely in Aldes. Algorithms used but not described in this thesis may be found in either [LOS76] or [COL78].

The Aldes language processor and the SAC-2 system assume the existence of a run-time list processing system. In this system, atoms and lists are distinguished by the system parameter β, which is approximately a factor of four smaller than γ, the largest single precision integer available on a given computer. Integers with absolute value less than β are interpreted as atoms and are called β-digits. Integers greater than β are interpreted as list pointers. The empty list, denoted by (), is represented by β.

Each list cell is composed of two fields. These fields are referenced through the functions FIRST(L) and RED(L). If L is a non-empty list, then FIRST(L) returns the first element of list L and RED(L) returns the reductum of L, i.e., the list L with the first element removed. Lists are created using the composition function COMP(a,L) which, if L is a list, returns a new list whose first element is a and whose reductum is L. A summary of other list processing algorithms may be found in [LOS76].

Where possible in this thesis, we will analyze the maximum computing times of the algorithms presented. We express these computing time bounds in a computer-independent manner by using the concept of dominance. If f and g are two real valued functions defined on some set S, we say that f is dominated by g (or g is a bound for f) in case there exists a positive real number c such that $f(x) \leq c \cdot g(x)$ for all x in S. If f is dominated by g and g is dominated by f, we say that f and g are codominant. The properties of the dominance relation are discussed in detail in [MUS71] and [COL78].

CHAPTER TWO:

Computation of Simplifying Ample Functions in Polynomial Rings

2.1 Introduction

The constructions in this chapter will show how to find simplifying ample functions in very general polynomial rings. A ring R in which, roughly speaking, we can find simplifying ample functions and in which we can also solve any linear homogeneous equation in several unknowns will be called a simplification ring. We will prove that, if R is a simplification ring, then it is possible to construct simplifying ample functions in $R[x_1,\ldots,x_r]$. As a consequence we will be able to show that $R[x_1,\ldots,x_r]$ is itself a simplification ring.

In this section we present an informal discussion. An attempt to construct a simplifying ample function in $R[x_1,\ldots,x_r]$ begins by writing the algebraic relations among the indeterminates given in equation 1.3.1 in distributive canonical form using some given ordering of the power products of the indeterminates. For each polynomial A_i we then have a well defined leading term, $lt(A_i)$, and reductum, $rd(A_i) = A_i - lt(A_i)$. Since $A_i = 0$, we know that we can write $lt(A_i) = -rd(A_i)$. When the coefficient ring is a field we can make $lt(A_i)$ monic, and it would seem that substituting $-rd(A_i)$ for occurrences of $lt(A_i)$ in an arbitrary polynomial, P, would serve to "simplify" P. This process does result in a simpler form for P in terms of the ordering given for the power products of the indeter-

minates. It does not guarantee, however, that two polynomials which are equivalent with respect to the algebraic relations given in equation 1.3.1 will have the same "simplified" form.

We will see that if two equivalent polynomials have different simplified forms, it is because the ideal generated by the A_i contains elements (i.e. algebraic relations among the indeterminates consistent with the initial set) which are "irreducible" with respect to the initial set of relations. By combining and simplifying all possible subsets of relations, we eventually will reach a point where no new relations can be generated. The substitution operation described above, when used with respect to this new set of relations, can be shown to be a simplifying ample function.

In the development of this and the following chapters the repeated substitution process will be called "remaindering" and the combining of relations will be referred to as "consensus formation." The ideal basis resulting after all possible relations have been combined will be called a "complete" basis.

We conclude this informal introduction with a concrete example. Let Q be the field of rational numbers and consider the relations $A_1, A_2 \in Q[x,y]$ given by $A_1 = x^2y + 1 = 0$ and $A_2 = xy^2 + 1 = 0$, which generate the ideal $\overline{A}$ in $Q[x,y]$. Define an ordering, $\leq$, of the power products of x and y such that $o < 1 < x < y < x^2 < xy < y^2 < x^3$. Let $P = xyA_1 - x^2A_2 = xy - x^2$. Since both P and 0 are in $\overline{A}$, we would like their simplified forms to be the same. However P cannot be reduced

using either A_1 or A_2. If we let $A_3 = yA_1 - xA_2 = y - x$, we see that A_3 is an element of $\overline{A}$, and A_3 is "irreducible" with respect to A_1 and A_2. But since $P = x \cdot lt(A_3) - x^2$, we have that $P' = x(-rd(A_3)) - x^2 = x(x) - x^2 = 0$ is the "simplified" form of P with respect to the new set of relations $\{A_1, A_2, A_3\}$.

Continue by combining relations A_1 and A_3 to get $A_4 = A_1 - x^2A_3 = x^3 + 1$ which is also in $\overline{A}$, but is "irreducible" with respect to A_1, A_2, and A_3. All other combinations of the relations A_1, A_2, A_3, and A_4 turn out to be reducible to zero, so there are no more relations to be added. The theorems in the remainder of this chapter will show that the remainder operation with respect to the ideal basis $\langle A_1, A_2, A_3, A_4 \rangle$ is a simplifying ample function for $(Q[x,y], \equiv \bmod \overline{A}, \leq)$.

2.2 Simplification Rings and Monomial Orderings

Let R be a ring and let $R[x_1,\ldots,x_r]$ be the ring of polynomials in the r indeterminates $x_1,\ldots,x_r$. As in Chapter One we will abbreviate $R[x_1,\ldots,x_r]$ by R[x]. In what follows we will often use lower case Roman letters to represent elements of R, and similar upper case letters for elements of R[x]. Letters with an arrow above them represent finite length sequences of elements of either R or R[x] depending on whether the letter is lower or upper case. A letter (lower or upper case) with a bar above it represents the ideal generated by the elements of the sequence represented by the same letter with an arrow above it. In this situation the sequence represented by the letter surmounted by the arrow will be called an ideal basis. The word "basis" throughout the remainder of this thesis will mean ideal basis unless otherwise noted. A sequence of elements of some domain (and in particular an ideal basis) will be denoted by the given elements surrounded by angle brackets (<>).

For example, if $a_1,\ldots a_n \in R$, then we let $\vec{a} = \langle a_1,\ldots,a_n\rangle$ and $\bar{a} = \{\sum_{i=1}^{n} s_i a_i : s_i \in R,\ 1 \leq i \leq n\}$, i.e. $\bar{a}$ is the ideal generated by basis $\vec{a}$.

If $\vec{a} = \langle a_1,\ldots,a_n\rangle$ and $\vec{b} = \langle b_1,\ldots,b_n\rangle$ are sequences of the same length, then we write the vector inner product of $\vec{a}$ and $\vec{b}$ as $\vec{a}\cdot\vec{b} = \sum_{i=1}^{n} a_i b_i$. We can thus rewrite $\bar{a}$ in the previous paragraph as $\bar{a} = \{\vec{s}\cdot\vec{a} : \vec{s} \in R^n\}$.

Let R^* be the set of all finite non-null sequences of elements of R, i.e. $R^* = R \cup R^2 \cup R^3 \cup \ldots$. If $\vec{a}, \vec{b} \in R^*$ and we write $\vec{a}\cdot\vec{b}$, we are assuming that $\vec{a}$ and $\vec{b}$ are sequences of the same length.

The following definition gives the criteria which must be satisfied by the coefficient ring R.

Definition 2.2.1 Let R be a Noetherian ring with identity, and let $\leq$ be a partial order on R. Such a ring is called a simplification ring if the following additional conditions are satisfied.

(a) There exists a function $\theta : R \times R^* \to R^*$ such that if $\vec{a} \in R^*$ and $b \in R$ then the function $\rho(b,\vec{a}) = b - \vec{a}\cdot\theta(b,\vec{a})$ is a simplifying ample function for $(R, \equiv \bmod \bar{a}, \leq)$ where $\bar{a}$ is the ideal in R generated by $\vec{a}$. In addition we require $\theta(0,\vec{a}) = \langle 0,\ldots,0\rangle$.

(b) There exists a function $\beta : R^* \to (R^*)^*$ such that if $\vec{a} \in R^*$ then $\beta(\vec{a})$ is a finite sequence of generators for the R-module $V(\vec{a}) = \{\vec{s} \in R^* : \vec{s}\cdot\vec{a} = 0\}$.

Some remarks on this definition are in order.

A Noetherian ring R is a commutative ring which possesses the following three equivalent properties [VDW70b].

(a) Every ideal in R is generated by a finite basis.

(b) Every strictly ascending chain of ideals of R must contain only a finite number of terms. (A strictly ascending chain of ideals is a sequence of ideals $\bar{a}_i \subseteq R$ such that $\bar{a}_1 \subset \bar{a}_2 \subset \ldots$.

(c) Every non-empty set of ideals of R must contain a maximal element (i.e. an ideal which is contained in no other ideal of the set).

If R contains an identity element, then property (a) can be used to prove the Hilbert Basis Theorem.

Theorem 2.2 If R is a Noetherian ring with identity, then the polynomial ring $R[x_1,\ldots,x_r]$ is also Noetherian.

Proof See [VDW70b]. □

We will use the Hilbert Basis Theorem in conjunction with property (b) of a Noetherian ring to provide a termination proof for an algorithm which constructs complete ideal bases in R[x] (see Section 2.5).

The function $\theta(b,\vec{a})$ provides a representation of $b - \rho(b,\vec{a})$ in terms of the basis $\vec{a}$. Of more importance in the following developments is the behavior of $\rho(b,\vec{a})$.

As a simplifying ample function, $\rho(b,\vec{a})$ must satisfy the following conditions (given $b,c \in R$ and $\vec{a} \in R^*$):

(a) $\rho(b,\vec{a}) \equiv b \bmod \overline{a}$,

(b) if $b \equiv c \bmod \overline{a}$ then $\rho(b,\vec{a}) = \rho(c,\vec{a})$, and

(c) $\rho(b,\vec{a}) \leq b$.

In addition, since $\theta(0,\vec{a}) = \langle 0,\ldots,0\rangle$ we have

(d) $\rho(0,\vec{a}) = 0 - \vec{a}\cdot\theta(0,\vec{a}) = -\vec{a}\cdot\langle 0,\ldots,0\rangle = 0$.

Now let $\vec{a} = \langle 1\rangle$ so that $\overline{a} = R$. Then, by (d), (b), and (c) respectively, since $0 \equiv b \bmod \overline{a}$, $0 = \rho(0,\vec{a}) = \rho(b,\vec{a}) \leq b$ for all $b \in R$. This is an important restriction on the order $\leq$ in R.

By Lemma 1.2.2 we know that a simplifying ample function $\rho(b,\vec{a})$ is unique for the triple $(R, \equiv \bmod \overline{a}, \leq)$. Therefore $\rho(b,\vec{a})$

must be independent of the basis $\vec{a}$ used to represent the ideal $\bar{a}$. Hence it is possible to write $\rho(b,\bar{a})$ instead of $\rho(b,\vec{a})$. We will retain the explicit reference to $\vec{a}$ because $\theta(b,\vec{a})$ explicitly depends on the basis used to represent $\bar{a}$. Note that although we have called $\rho(b,\vec{a})$ a simplifying ample function, we really mean that for any fixed ideal $\bar{a}$, the function $\rho_{\bar{a}}$ defined by $\rho_{\bar{a}}(b) = \rho(b,\bar{a})$ is the actual simplifying ample function for $(R, \equiv \bmod \bar{a}, \leq)$.

The second part of Definition 2.2.1 simply states that if $\vec{a} = \langle a_1,\ldots,a_n\rangle$ then we can find a finite representation of all solutions $s_1,\ldots,s_n \in R$ of the equation $\sum_{i=1}^{n} s_i a_i = 0$. This is what was meant by our reference to solving linear homogeneous equations in Section 2.1. The function $\beta(\vec{a})$ will prove to be vital in constructing combinations of basis elements for ideals in R[x].

The algorithms presented in this chapter will be "abstract algorithms" in the sense specified by Musser [MUS71]. The domains of the inputs and outputs of these algorithms are abstract algebraic structures, and the validity of the algorithms depends on the existence of effective algorithms for computing ρ, θ, and β. That such algorithms actually exist is evident from the following examples.

If R = F, a field, then the only ideals are {0} and F. A valid partial ordering of F is given by $a \leq b$ iff $a = 0$ or $a = b$, for $a,b \in F$. Assume that an ideal $\bar{a}$ in F is given by basis $\vec{a} = \langle a_1,\ldots,a_n\rangle$. If $a_i = 0$ for $1 \leq i \leq n$, then $\bar{a} = \{0\}$ and if $b \in F$, then

$$\theta(b,\vec{a}) = \langle 0,\ldots,0\rangle,$$

and $$\beta(\vec{a}) = \{\langle 1,0,\ldots,0\rangle, \langle 0,1,0,\ldots,0\rangle, \ldots, \langle 0,\ldots,0,1\rangle\}$$

satisfy Definition 2.2.1 with $\rho(b,\{0\}) = b$. Otherwise we may assume that $a_1 \neq 0$. Then $\overline{a} = F$ and the definition is satisfied by

$$\theta(b,\vec{a}) = \langle \frac{b}{a_1}, 0,\ldots,0\rangle$$

and $$\beta(\vec{a}) = \{\langle -a_2,a_1,0,\ldots,0\rangle, \langle -a_3,0,a_1,0,\ldots,0\rangle, \ldots,$$

$$\langle -a_n,0,\ldots,0,a_1\rangle, \langle 0,\ldots,0\rangle\}$$

with $$\rho(b,F) = 0.$$

These forms for ρ, θ, and β will be used implicitly in Chapter Three when we treat the special case of computing residue class representatives in the ring of multivariate polynomials over a field.

If $R = Z$, the ring of integers, then all ideals in R are principal ideals. If $\vec{a} = \langle a_1,\ldots,a_n\rangle$ is a basis for ideal $\overline{a}$ in Z, and $\overline{a} \neq \{0\}$ then $\overline{a}$ is also generated by the basis $\langle d\rangle$ where $d = \gcd(a_1,\ldots,a_n)$. As in Section 1.2, let us define the function $\rho(b,\vec{a}) = r$ where $b = qd + r$ with $q,r \in Z$ and $0 \leq r < d$. Also let us define a partial order $\leq^*$ on Z such that

$$a <^* b \text{ iff } \begin{cases} a = 0 \text{ and } b \neq 0 \\ a > 0 \text{ and } b < 0 \\ \text{sign}(a) = \text{sign}(b) \text{ and } |a| < |b|, \end{cases}$$

with $a \leq^* b$ iff $a <^* b$ or $a = b$. In Section 1.2 we pointed out that

$\rho(b,\vec{a})$ is a simplifying ample function for $(Z_1 \equiv \bmod d, \leq^*)$, and hence for $(Z, \equiv \bmod \bar{a}, \leq^*)$.

If the extended Euclidean algorithm for n integers [BRD70] is applied to $\vec{a}$, the output will be $\vec{s} \in R^n$ such that $\vec{s}\cdot\vec{a} = d$. Then $\rho(b,\vec{a}) = r = b - qd = b - q\vec{s}\cdot\vec{a} = b - (q\vec{s})\cdot\vec{a}$. Hence if we set $\theta(b,\vec{a}) = q\vec{s}$ then $\rho(b,\vec{a}) = b - \theta(b,\vec{a})\cdot\vec{a}$, as required in Definition 2.2.1(a).

To determine $\beta(\vec{a})$ we must be able to find a general solution to the linear homogeneous Diophantine equation $\sum_{i=1}^{n} x_i a_i = 0$. Algorithms for solving such equations have been discussed by Blankinship [BLK66], Bradley [BRD71], Knuth [KNU69], and most recently by Kannan and Bachem [KBA78].

We now shift our attention to the polynomial ring $R[x] = R[x_1,\ldots,x_r]$. Our first task in making R[x] a simplification ring is to specify a partial ordering of polynomials in R[x]. A large class of partial orderings in R[x] which lend themselves to the determination of simplifying ample functions is defined by combining the partial order, $\leq$ in R with a total ordering of all monomials in R[x].

To be more explicit, by a <u>monomial</u> in R[x] we mean a power product $x_1^{e_1}\ldots x_r^{e_r}$ where $e_i \geq 0$, $1 \leq i \leq r$. Note that when regarded as a polynomial in R[x], a monomial in our usage is monic, i.e. its coefficient is the identity element of R. Let $M = \{x_1^{e_1}\ldots x_r^{e_r} : e_r \geq 0, 1 \leq i \leq r\}$ be the set all monomials in R[x]. By the <u>exponent vector</u> of a monomial we mean the sequence of its exponents, $\langle e_1,\ldots,e_r\rangle$ in the above case. Corresponding to

the set M of all monomials, we have the set of all exponent vectors, $E = \{\langle e_1,\ldots,e_r\rangle : e_i \geq 0,\ 1 \leq i \leq r\}$. If $I = \langle e_1,\ldots,e_r\rangle \in E$ then by x^I we mean the monomial $x_1^{e_1}\ldots x_r^{e_r}$. Exponent vectors will be represented by upper case Roman letters I, J, K, etc.

Both multiplicative operations on monomials and orderings of monomials can be represented by means of the associated exponent vectors. This will be done almost exclusively throughout this thesis. Let $I,J \in E$ with $I = \langle e_1,\ldots,e_r\rangle$ and $J = \langle f_1,\ldots,f_r\rangle$. If $x^K = x^I x^J$ then we write $K = I + J = \langle e_1 + f_1,\ldots,e_r + f_r\rangle$. If $e_i \leq f_i$ for $1 \leq i \leq r$ and $x^K = x^J/x^I$, then we write $K = J - I = \langle f_1 - e_1,\ldots,f_r - e_r\rangle$. Finally if $x^K = \gcd(x^I,x^J)$ and $x^{K'} = \operatorname{lcm}(x^I,x^J)$, the greatest common divisor and the least common multiple of x^I and x^J, respectively, then we write $K = \gcd(I,J) = \langle\min(e_1,f_1),\ldots,\min(e_r,f_r)\rangle$ and $K' = \operatorname{lcm}(I,J) = \langle\max(e_1,f_1),\ldots,\max(e_r,f_r)\rangle$. Note that these definitions extend readily to expressing the gcd and lcm of more than two monomials.

The relation of divisibility is a partial order of the set M, and hence induces a partial order in E. We use a vertical bar to denote this order and say that for $I,J \in E$ $(x^I,x^J \in M)$

$$I|J \text{ (or } x^I|x^J) \quad \text{iff} \quad e_i \leq f_i,\ 1 \leq i \leq r\ .$$

We write $I \nmid J$ when $I|J$ is false.

We now use this partial ordering to limit the possible total monomial orderings.

Definition 2.2.2 An acceptable monomial ordering in R[x] is a total (linear) ordering, $\leq$, of the set E such that for $I,J,K \in E$,

(a) if $I \mid J$ then $I \leq J$,

(b) if $I \leq J$ then $I+K \leq J+K$.

If $I \leq J$ and $I \neq J$ then we write $I < J$ or $J > I$.

The first property will be used in Lemma 2.2.1 to show that E (and hence M) is well-ordered by $\leq$. The second property of $\leq$ will be used in Lemma 2.2.2 to show that the exponent vector of the leading term of a polynomial product has a well defined upper bound relative to $\leq$.

Although we use $\leq$ to represent both the partial ordering in R and the total ordering in E, the context will always be sufficient to resolve which is intended. In particular, note that if E' is a finite subset of E, then by max E' and min E' we mean the maximum and minimum $I \in E'$, respectively, relative to the total ordering $\leq$ of E.

The specification of the monomial ordering is crucial, for on it hinges the meaning of simplification in R[x]. Given two different orderings, the algorithms to be presented will produce two different residue class representatives, i.e. there will be two different simplifying ample functions. In addition, even when the actual residue class representative is unimportant (for instance when using these algorithms to determine when a polynomial is in a given ideal), the choice of the ordering can have a significant effect on the associated computing times. This will be discussed further in Chapter 5.

The restriction given in part (a) of Definition 2.2.2 is essential because several proofs in the following sections are by transfinite induction and depend on the following lemma.

Lemma 2.2.1 The set E of all exponent vectors (and hence the set M of all monomials) is well-ordered by $\leq$.

Proof It is sufficient to show that there is no strictly decreasing infinite sequence of elements of E. Let $D = \langle I_1, I_2, \ldots \rangle$ be an infinite sequence of elements of E such that $I_j > I_k$ for $j < k$. Note that by Definition 2.2.2(a) this implies $I_j \dagger I_k$ for $j < k$. If $r = 1$, i.e. E contains exponent vectors for polynomials in one indeterminate, then this is clearly impossible.

Let $r \geq 2$ be the least number of indeterminates for which there is such an infinite sequence D. Let $I_j = \langle e_{j1}, \ldots, e_{jr} \rangle$. Suppose first that the set $\{e_{jr} : j \geq 1\}$ has no upper bound. Then we can select a sub-sequence of D, $D' = \langle I_1', I_2' \ldots \rangle$ with $I_j' = \langle e_{j1}', \ldots e_{jr}' \rangle$ such that $e_{jr}' < e_{kr}'$ for $j < k$. But then since $I_j' \dagger I_k'$ for $j < k$ the sequence $D'' = \langle I_1'', I_2'', \ldots \rangle$ where $I_j'' = \langle e_{j1}', \ldots, e_{j,r-1}' \rangle$ must be an infinite sequence with $I_j'' \dagger I_k''$ for $j < k$. But D" is a sequence of exponent vectors representing monomials in r-1 indeterminates. This violates our assumption that r is the least such integer.

Now suppose that the set $\{e_{jr} : j \geq 1\}$ is bounded. Then there must be some integer $\bar{e}$ such that $e_{jr} = \bar{e}$ for an infinite number of j's. Now let $D' = \langle I_1', I_2', \ldots \rangle$ be a sub-sequence of D such that $I_j' = \langle e_{j1}', \ldots, e_{j,r-1}', \bar{e} \rangle$. The sub-sequence D" derived from this

sequence as in the previous paragraph also violates the assumption that r is the minimum number of indeterminates for which there exists a strictly decreasing infinite sequence of exponent vectors.

These two contradictions imply that there are no strictly decreasing infinite sequences of elements of E, and hence that E is well-ordered.□

We now return to the problem of specifying a partial order in R[x]. First we note that any polynomial $A \in R[x]$ can be written as a sum of monomials, $A = \sum_{I \in E} a_I x^I$ where only a finite number of the $a_I \in R$ are non-zero. Given any acceptable monomial ordering, $\leq$, we can define several important components of A. If $A \neq 0$, the <u>leading exponent vector</u> (or <u>degree</u>) of A is $\partial A = \max_{\leq} \{I \in E : a_I \neq 0\}$. We then define the <u>leading coefficient</u> of A, $lc(A) = a_{\partial A}$, the <u>leading monomial</u> of A, $lm(A) = x^{\partial A}$, and the <u>leading term</u> of A, $lt(A) = a_{\partial A} x^{\partial A}$. Finally, we define the <u>reductum</u> of A, $rd(A) = A - lt(A)$. In case $A = 0$ we adopt the conventions that $lc(0) = lt(0) = rd(0) = 0$. We leave $lm(0)$ undefined, but we consider $\partial 0$ to be a special exponent vector with the properties $\partial 0 < I$, $\partial 0 \nmid I$ and $I \nmid \partial 0$ for $I \in E$. Exponent vector sums and differences involving $\partial 0$ are not defined, but if $I_1,\ldots,I_n \in E \cup \{\partial 0\}$ then $\gcd(I_1,\ldots,I_n)$ and $\operatorname{lcm}(I_1,\ldots,I_n)$ are the greatest lower bound and least upper bound respectively of $\{I_1,\ldots,I_n\}$ relative to the partial order $|$. We define $\gcd(\partial 0,\ldots,\partial 0) = \operatorname{lcm}(\partial 0,\ldots,\partial 0) = \partial 0$.

Note that when we refer to the degree of a polynomial, ∂A, we are referring to its leading exponent vector. This is distinct from

the degree vector of A, which is defined to be $\langle \partial_{x_1} A, \ldots, \partial_{x_r} A \rangle$ where $\partial_{x_i} A$ is the maximum degree of the indeterminate x_i in A, $1 \leq i \leq r$.

We now are in a position to use restriction (b) in Definition 2.2.2 to give a bound on the leading exponent vector of a product of two polynomials.

Lemma 2.2.2 Let $A,B \in R[x]$. Then $\partial(AB) \leq \partial A + \partial B$. If $lc(A)lc(B) \neq 0$ then $\partial(AB) = \partial A + \partial B$ and $lc(AB) = lc(A)lc(B)$.

Proof Let $A = \sum_{I \in E} a_I x^I$ and $B = \sum_{J \in E} b_J x^J$. Also let $\overline{I} = \{I \in E : a_I \neq 0\}$, $\overline{J} = \{J \in E : b_j \neq 0\}$, $I \in \overline{I}$ and $J \in \overline{J}$. Then $\partial A \geq I$ and $\partial B \geq J$ so, by Definition 2.2.2(b), $\partial A + \partial B \geq I + \partial B \geq I + J$.

Now $AB = \sum_{\substack{I \in \overline{I} \\ J \in \overline{J}}} a_I b_J x^I x^J = \sum_{\substack{I \in \overline{I} \\ J \in \overline{J}}} a_I b_J x^{I+J}$. Let

$\overline{K} = \{I + J : I \in \overline{I}, J \in \overline{J}\}$. Then $\partial(AB) \leq \max \overline{K} = \partial A + \partial B$, proving the first part of the lemma. To prove the second part, it is sufficient to show that if $I < \partial A$ and $J < \partial B$ then $I + J < \partial A + \partial B$. If $I < \partial A$ then, by Definition 2.2.2, $I + J \leq \partial A + J$. If equality holds, then we can subtract J from both sides giving $I = \partial A$ contrary to our assumption. Hence $I + J < \partial A + J$. Similarly, since $J < \partial B$, $\partial A + J < \partial A + \partial B$. Hence $I + J < \partial A + \partial B$.□

With these remarks out of the way, we can finally define a partial order in R[x].

Definition 2.2.3 If $\leq$ is a partial ordering in R and $\leq$ is also an acceptable monomial ordering, then we define an acceptable polynomial ordering, also denoted by $\leq$, as follows. If $A,B \in R[x]$ then

$$A < B \text{ iff } \begin{cases} \partial A < \partial B, \\ \partial A = \partial B \text{ and } lc(A) < lc(B), \text{ or} \\ \partial A = \partial B \text{ and } lc(A) = lc(B) \text{ and } rd(A) < rd(B). \end{cases}$$

We define $A \leq B$ by $A < B$ or $A = B$. Again, the context in which $\leq$ is used will serve to distinguish the ordering intended.

We end this section by defining several sets of polynomials and coefficients which will be of importance later.

Let $\vec{A} = \langle A_1, \ldots, A_n \rangle$ be a finite sequence of polynomials in $(R[x])^*$. As before, we let $\overline{A}$ denote the ideal in $R[x]$ generated by the polynomials in $\vec{A}$, i.e. $\overline{A} = \{\vec{S} \cdot \vec{A} : \vec{S} \in (R[x])^n\}$.

Definition 2.2.4 For $I \in E$ the degree I leading coefficient ideal of ideal $\overline{A}$ is defined to be $\bar{\bar{a}}_I = \bar{\bar{a}}_I(\overline{A}) = \{lc(A) : A \in \overline{A} \text{ and } \partial A \mid I\}$.

That $\bar{\bar{a}}_I$ is indeed an ideal in R can be justified as follows. If $b,c \in \bar{\bar{a}}_I$ and $b,c \neq 0$ then there exist polynomials $B,C \in \overline{A}$ such that $b = lc(B)$, $c = lc(C)$, $\partial B \mid I$ and $\partial C \mid I$. Note that $B' = x^{I-\partial B}B$ and $C' = x^{I-\partial C}C$ are such that $b = lc(B')$, $c = lc(C')$ and $\partial B' = \partial C' = I$. If $d = b + c = 0$ then $b + c \in \bar{\bar{a}}_I$. If $d \neq 0$ then $d = lc(B' + C')$ and since $\partial(B' + C') = I$ we have that $b + c \in \bar{\bar{a}}_I$. Similarly we may show that $b'b \in \bar{\bar{a}}_I$ for all $b' \in R$. Hence $\bar{\bar{a}}_I$ is actually an ideal of R.

Definition 2.2.5 For $I \in E$ we define the degree I sub-basis of basis $\vec{A} = \langle A_1, \ldots, A_n \rangle$ to be $\vec{A}_I = \langle A_1', \ldots, A_n' \rangle$ where for $1 \leq j \leq n$, $A_j' = A_j$ if $\partial A_j \mid I$ and $A_j' = 0$ otherwise. The degree I leading coefficient sub-basis of $\vec{A}$ is $\vec{a}_I = \vec{a}_I(\vec{A}) = \langle lc(A_1'), \ldots, lc(A_n') \rangle$.

The degree I leading coefficient ideal of basis $\vec{A}$ is just that ideal, $\bar{a}_I$, in R generated by the elements of $\vec{a}_I$.

Note carefully that $\bar{\bar{a}}_I(\bar{A})$ and $\bar{a}_I(\vec{A})$ denote two ideals in R. Clearly, $\bar{a}_I \subseteq \bar{\bar{a}}_I$. We will see shortly that $\bar{a}_I = \bar{\bar{a}}_I$, i.e. that $\vec{a}_I$ generates $\bar{\bar{a}}_I$, whenever $\vec{A}$ is a complete basis for $\bar{A}$.

In Section 2.1 we indicated that some polynomials in R[x] can be considered to be "irreducible" with respect to a given basis $\vec{A}$. This set of kernel polynomials defined here will prove to be those polynomials which cannot be further reduced by the remainder algorithm described in the next section.

Definition 2.2.6 Given a basis $\vec{A}$ as above with leading coefficient sub-bases $\vec{a}_I$, the set of kernel polynomials with respect to basis $\vec{A}$ is defined to be $\ker(\vec{A}) = \{B \in \bar{A} : B = 0 \text{ or } B = \sum_{I \in E} b_I x^I$ where $b_I = \rho(b_I, \vec{a}_I)$ for all $I \in E\}$.

The coefficient of x^I in a kernel polynomial is thus defined to be the representative of the residue class that it is contained in modulo the leading coefficient ideal $\bar{a}_I$.

2.3 Complete Bases and Simplifying Ample Functions

In this section we define a special basis (complete basis) which allows us to find a simplifying ample function for the ideal in R[x] generated by this basis. We assume $\vec{A} = \langle A_1,\ldots,A_n\rangle$ is a basis for ideal $\overline{A}$ in R[x]. We first need a way to describe a given element of $\overline{A}$ in terms of $\vec{A}$.

Definition 2.3.1 Let $A \in \overline{A}$ and assume there exists $\vec{S} = \langle S_1,\ldots,S_n\rangle \in (R[x])^n$ such that $A = \vec{S}\cdot\vec{A}$. $\vec{S}$ is said to be a representation of A in terms of $\vec{A}$. The degree of $\vec{S}$, $\partial\vec{S}$, is defined as follows. If for $1 \le i \le n$, either $A_i = 0$ or $S_i = 0$ then $\partial\vec{S} = \partial 0$. Otherwise $\partial\vec{S} = \max_{1\le i\le n} \{\partial S_i + \partial A_i : S_i \neq 0 \text{ and } A_i \neq 0\}$. We also define the multiplicity of representation $\vec{S}$, $\mu\vec{S} = n$ if $\partial\vec{S} = \partial 0$, and $\mu\vec{S}$ = cardinality of $\{1 \le i \le n : A_i \neq 0,\ S_i \neq 0 \text{ and } \partial\vec{S} = \partial S_i + \partial A_i\}$.

Note that $\partial\vec{S}$ is actually the maximum of the "formal degrees" of the products S_iA_i, $1 \le i \le n$, where by the formal degree of S_iA_i we mean $\partial 0$ if $A_i = 0$ or $S_i = 0$ and $\partial S_i + \partial A_i$ otherwise. If R is an integral domain, then the formal degree of S_iA_i is just $\partial(S_iA_i)$, and $\partial\vec{S} = \max_{1\le i\le n} \partial(S_iA_i)$. In this case $\mu\vec{S}$ = cardinality of $\{1 \le i \le n : \partial(S_iA_i) = \partial\vec{S}\}$.

We now define a special class of representations.

Definition 2.3.2 If $\partial\vec{S} = \partial A$ then $\vec{S}$ is said to be a simple representation of A with respect to the basis $\vec{A}$ (or that A is simply constructible from $\vec{A}$).

Note that $\partial\vec{S} \leq \partial A$ is an equivalent requirement for simple constructibility since in every case we must have $\partial\vec{S} \geq \partial A$.

Finally we describe a special class of bases.

Definition 2.3.3 If all polynomials $A \in \overline{A}$ are simply constructible from basis $\vec{A}$, then $\vec{A}$ is said to be a complete basis for ideal $\overline{A}$ (or that ideal $\overline{A}$ is simply constructible from basis $\vec{A}$).

It should be emphasized that the monomial ordering is intimately connected with the notion of a complete basis. The existence of a complete basis guarantees that every element of the ideal has a representation which involves no monomials of higher degree than its leading term.

We now present the description of a function which will become our simplifying ample function in R[x]. The basic idea is to reduce a given polynomial by subtracting multiples of various basis elements. Eventually a polynomial is found that is a kernel polynomial with respect to the given basis.

Recall that for $J \in E$, $\vec{a}_J$ is the degree J leading coefficient sub-basis of basis $\vec{A}$, and $\overline{a}_J$ is the corresponding ideal in R.

Algorithm 2.3

$$Q \leftarrow \text{rem}(P, \vec{A})$$

[Remainder of polynomial with respect to ideal basis.

Input: $P \in R[x]$; $\vec{A}=\langle A_1,\ldots,A_n\rangle \in (R[x])^n$, a basis for ideal $\overline{A}$.

Output: $Q \in R[x]$ such that $Q \equiv P \bmod \overline{A}$; $Q \in \ker(\vec{A})$; $Q \leq P$; and $P-Q$ is simply constructible from $\vec{A}$.]

(1) [Initialize.] $Q \leftarrow P$.

(2) [Is $Q=0$?] if $Q=0$ then return.

(3) [Assuming $Q=\sum_{I \in E} q_I x^I$, determine possible reductions.]
$D \leftarrow \{I \in E : q_I \neq \rho(q_I, \vec{a}_I)\}$; if D is empty then return.

(4) [Apply remainder choice function.] Choose $J \in D$.

(5) [Use $\theta(q_J, \vec{a}_J)$ to determine $\rho(q_J, \vec{a}_J)$ in terms of $\vec{a}_J$.]
$\vec{b}=\langle b_1,\ldots,b_n\rangle \leftarrow \theta(q_J, \vec{a}_J)$.

(6) [Make new degree J coefficient the residue class representative for the residue class of q_J mod $\bar{a}_J$, i.e. $q_J \leftarrow \rho(q_J, \vec{a}_J)=q_J-\vec{b}\cdot\vec{a}_J$.]
for $i=1,\ldots,n$ do if $\partial A_i | J$ then $B_i \leftarrow b_i x^{J-\partial A_i}$ else $B_i \leftarrow 0$;
$\vec{B} \leftarrow \langle B_1,\ldots,B_n\rangle$; $Q \leftarrow Q-\vec{B}\cdot\vec{A}$; go to step 2 □

Although we will eventually demonstrate that Algorithm 2.3 is indeed an algorithm and that at termination the output conditions are satisfied, let us assume these results for the moment. Note that Algorithm 2.3 is an abstract algorithm implementing a function $\mathrm{rem}(P,\vec{A})$. The abstractness appears in step 4, where we must select an element of D to determine which term of Q to reduce, and in step 5, where we must use $\theta(q_J, \vec{a}_J)$ to compute a representation $\vec{b}$ for $q_J - \rho(q_J, \vec{a}_J)$ in terms of $\vec{a}_J$. In both of these steps we are assuming that the abstract sub-algorithms are functions, i.e. that they are single valued. In particular, every time the choice function is given some subset $D \subseteq E$ as input, it always returns the same element $J \in D$ as output. In both the case of the choice function and the θ function we have a large family of possible functions to choose from.

Note that the actual function computed by Algorithm 2.3 depends on the specifications made for these abstract sub-algorithms. For example, let $\vec{A} = \langle A_1, A_2 \rangle$ where $A_1 = y^2x + 2$ and $A_2 = yx^2 + 1$ are polynomials in $Q[x,y]$. If $Q = y^2x^2 + y$ and $J = \langle 2,2 \rangle$ then $\vec{a}_J = \langle 1,1 \rangle$. Since $q_J = 1$, we can reduce q_J to zero in at least two ways. Let $\theta_1(q_J, \vec{a}_J)$ be such that $\theta_1(1,\langle 1,1 \rangle) = \langle 1,0 \rangle$, and let $\theta_2(q_J, \vec{a}_J)$ be such that $\theta_2(1,\langle 1,1 \rangle) = \langle 0,1 \rangle$. Both of these θ functions give $\rho(1,\langle 1,1 \rangle) = 0$. However, when used in Algorithm 2.3 we find

$Q_1 = Q - \langle xA_1, yA_2 \rangle \cdot \theta_1(1,\langle 1,1 \rangle) = Q - xA_1 = y - 2x$ while

$Q_2 = Q - \langle xA_1, yA_2 \rangle \cdot \theta_2(1,\langle 1,1 \rangle) = Q - yA_2 = 0$. Thus two different choices for θ yield two different remainder functions.

We discuss the resolution of these ambiguities in the definition of the remainder function in Chapter 3 under the heading of <u>remainder selection rules</u>.

We now turn to the task of establishing the validity and termination of Algorithm 2.3. We begin by giving two lemmas. The first deals with the results of a single reduction step in the remainder algorithm. The second deals with certain sequences of subsets of a well-ordered set.

<u>Lemma 2.3.1</u> Let non-zero polynomial $Q = \sum_{I \in E} q_I x^I \in R[x]$, and let $J \in E$ be such that $q_J \neq \rho(q_J, \vec{a}_J)$ where $\vec{a}_J$ is the degree J leading coefficient sub-basis of ideal basis $\vec{A} = \langle A_1, \ldots, A_n \rangle$. Let $\vec{b} = \langle b_1, \ldots, b_n \rangle = \theta(q_J, \vec{a}_J)$ and $\vec{B} = \langle B_1, \ldots, B_n \rangle$ where for $1 \leq i \leq n$ if $\partial A_i | J$ then $B_i = b_i x^{J - \partial A_i}$ otherwise $B_i = 0$. If $Q' = Q - \vec{B} \cdot \vec{A} = \sum_{I \in E} q_I' x^I$, then

(a) $\partial(\vec{B}\cdot\vec{A}) = J$,

(b) $q_J' = \rho(q_J, \vec{a}_J)$,

(c) $Q' < Q$, and

(d) $\vec{B}$ is a simple representation for $Q - Q'$, and in fact $\partial\vec{B} = J$.

Proof For $1 \leq i \leq n$ let $a_i = lc(A_i)$. If $\partial A_i | J$ then $b_i = lc(B_i)$ and $\partial(B_iA_i) = \partial(b_i x^{J-\partial A_i} A_i) \leq (J - \partial A_i) + \partial A_i = J$ by Lemma 2.2.2, while otherwise $B_iA_i = 0$. Hence $\partial(\vec{B}\cdot\vec{A}) \leq J$. Now since $q_J \neq \rho(q_J, \vec{a}_J)$, $0 \neq q_J - \rho(q_J, \vec{a}_J) = \vec{a}_J \cdot \theta(q_J, \vec{a}_J) = \vec{b}\cdot\vec{a}_J = \sum_{\partial A_i | J} b_i a_i = \sum_{\partial A_i | J} lc(B_i)lc(A_i)$ which is the coefficient of x^J in $\vec{B}\cdot\vec{A}$. Hence $\partial(\vec{B}\cdot\vec{A}) = J$, proving (a).

Now since $\partial(\vec{B}\cdot\vec{A}) = J$, $q_J' = q_J - \sum_{i=1}^{n} b_i a_i = q_J - (q_J - \rho(q_J, \vec{a}_J))$ by the previous paragraph. Hence $q_J' = \rho(q_J, \vec{a}_J)$, proving (b).

Again since $\partial(\vec{B}\cdot\vec{A}) = J$, we have that $q_I' = q_I$ for $I > J$. But $q_J' = \rho(q_J, \vec{a}_J) \neq q_J$. Hence, since ρ is a simplifying ample function, $q_J' < q_J$, where $<$ is the partial order in R. Thus by Definition 2.2.3, $Q' < Q$, establishing (c).

By the proof of (a), $\sum_{\partial A_i | J} lc(B_i)lc(A_i) \neq 0$. Hence for some i, $\partial A_i | J$, $B_i \neq 0$ and $A_i \neq 0$. But if $B_i \neq 0$ and $A_i \neq 0$ then $\partial A_i | J$ and $\partial B_i + \partial A_i = (J - \partial A_i) + \partial A_i = J$. Hence $\max_{1 \leq i \leq n} \{\partial B_i + \partial A_i : B_i \neq 0$ and $A_i \neq 0\} = J = \partial(\vec{B}\cdot\vec{A}) = \partial(Q - Q')$. So $\partial\vec{B} = J = \partial(Q - Q')$, i.e. $\vec{B}$ is a simple representation for $Q - Q'$.□

The following lemma will be used to prove that Algorithm 2.3 terminates. For the purpose of this lemma, the set E is any well-

ordered set. We do not use any special properties of exponent vectors in the proof.

Lemma 2.3.2 Let E be a set well-ordered by the relation $\leq$. Let $D_1, D_2, \ldots$ be a sequence of finite subsets of E such that for every $i \geq 1$ either D_i is empty or there exists $J_i \in D_i$ and finite sets $F_i, F_i' \subseteq E$ such that $I < J_i$ for all $I \in F_i \cup F_i'$ and $D_{i+1} = (D_i - \{J_i\} - F_i) \cup F_i'$. Then for some $j \geq 1$, D_j is empty.

Proof (The proof is by transfinite induction on max D_1.) Let $K = \max D_1$, and assume the theorem is true whenever $\max D_1 < K$. Assume that D_i is non-empty and that $J_i \neq K$ for all $i \geq 1$. Let $D_i' = D_i - \{K\}$ for all i. If D_1' is empty then $D_1 = \{K\}$ and $J_1 = K$, contrary to hypothesis. Therefore, D_1' is not empty and $\max D_1' < K$. By the induction hypothesis, there must exist a k such that D_k' is empty. But then $D_k = \{K\}$ and $J_k = K$, contradicting again our assumption that $J_i \neq K$ for all i. Hence, if D_i is non-empty for all i, then, for some k, $J_k = K$. But then $\max D_{K+1} < K$, and the induction hypothesis applied to $D_{k+1}, D_{k+2}, \ldots$ shows that for some $j \geq k+1$, D_j is empty. Hence the theorem is true for $\max D_1 = K$, completing the induction.□

Now we come to the termination proof for the algorithm presenting the remainder function.

Lemma 2.3.3 Algorithm 2.3 eventually terminates.

Proof Let Q_i represent the value of Q at the i-th execution of step 2. Let D_i, J_i, and $\vec{B}_i$ be the values of D, J, and $\vec{B}$ after the

i-th execution of steps 3, 4, and 6, respectively.

If $Q_i = 0$ then Algorithm 2.3 terminates during the i-th execution of step 2. So assume $Q_i = \sum_{I \in E} q_I x^I \neq 0$. If $D_i = \{I \in E : q_I \neq \rho(q_I, \vec{a}_I)\}$ is empty, then the algorithm terminates during the i-th execution of step 3. So assume D_i is non-empty.

Now $Q_{i+1} = Q_i - \vec{B}_i \cdot \vec{A}$. If $Q_{i+1} = 0$ then Algorithm 2.3 terminates during the $(i+1)$-st execution of step 2. Assume $Q_{i+1} = \sum_{I \in E} q_I' x^I \neq 0$. We now describe D_{i+1} in terms of D_i.

Let $D_i(> J_i) = \{I \in D_i : I > J_i\}$ and $D_i(< J_i) = \{I \in D_i : I < J_i\}$. Define $D_{i+1}(> J_i)$ and $D_{i+1}(< J_i)$ similarly. By Lemma 2.3.1(a), $\partial(\vec{B}_i \cdot \vec{A}) = J_i$; hence $D_{i+1}(> J_i) = D_i(> J_i)$. By Lemma 2.3.2(b), $q_J' = \rho(q_J, \vec{a}_J)$, and since residue class representatives are unique, $q_J' = \rho(q_J', \vec{a}_J)$. Hence $J_i \notin D_{i+1}$.

Now $D_i(< J_i)$ and $D_{i+1}(< J_i)$ are clearly finite, so $F_i = \{I \in D_i(< J_i) : I \notin D_{i+1}(< J_i)\}$ and $F_i' = \{I \in D_{i+1}(< J_i) : I \notin D_i(< J_i)\}$ are also finite sets.

Thus we can write $D_{i+1} = (D_i - \{J_i\} - F_i) \cup F_i'$ where for all $I \in F_i \cup F_i'$, $I < J_i$. Hence D_i, F_i, F_i', and J_i satisfy the hypotheses of Lemma 2.3.2. Since E is well-ordered by the monomial ordering $\leq$, there must be a $j \geq 1$ such that D_j is empty. Hence Algorithm 2.3 will terminate during the j-th execution of step 3.□

Before we prove the validity of the output assertions in Algorithm 2.3, we need one preliminary lemma.

Lemma 2.3.4 Let $\vec{A}$ be a basis for ideal $\overline{A}$ in R[x]. If $P \in \ker(\vec{A})$ then $\partial(P - \mathrm{rem}(P,\vec{A})) = \partial 0$, otherwise $\partial(P - \mathrm{rem}(P,\vec{A})) = \max\{I \in E : p_I \neq \rho(p_I,\vec{a}_I)\}$ where $P = \sum_{I\in E} p_I x^I$.

Proof We use the notation and results of the proof of Lemma 2.3.3. If $P \in \ker(\vec{A})$ then $P = \mathrm{rem}(P,\vec{A})$ and $\partial(P - \mathrm{rem}(P,\vec{A})) = \partial 0$. If $P \notin \ker(\vec{A})$ then clearly $P - \mathrm{rem}(P,\vec{A}) = \sum_{i=1}^{j} \vec{B}_i \cdot \vec{A}$ and $\partial\vec{B}_i = J_i \leq J$ for $1 \leq i \leq j$, where $J = \max\{I \in E : p_I \neq \rho(p_I,a_I)\}$. Since D_j is empty, there is a least k such that $J_k = J$. Then $J_i < J$ for $i \neq k$. Hence $\partial(P - \mathrm{rem}(P,\vec{A})) = \partial(\sum_{i=1}^{j} \vec{B}_i \cdot \vec{A}) = \max_{1\leq i\leq n} \partial(\vec{B}_i \cdot \vec{A}) = \max_{1\leq i\leq n} \partial\vec{B}_i = \max_{1\leq i\leq n} J_i = J$. □

We now restate and prove the properties given in the output specification of Algorithm 2.3.

Lemma 2.3.5 Let $P \in R[x]$ and let $\vec{A}$ be a basis for ideal $\overline{A} \subseteq R[x]$. Then

(a) $\mathrm{rem}(P,\vec{A}) \equiv P \bmod \overline{A}$;

(b) $\mathrm{rem}(P,\vec{A}) \in \ker(\vec{A})$;

(c) $\mathrm{rem}(P,\vec{A}) \leq P$, where $\leq$ refers to the ordering of polynomials;

(d) $P - \mathrm{rem}(P,\vec{A})$ is simply constructible from $\vec{A}$.

Proof Part (a) is immediate from (d).

(b) Let $Q = \mathrm{rem}(P,\vec{A})$. If $Q = 0$ then $Q \in \ker(\vec{A})$. Let $Q = \sum_{I\in E} q_I x^I \neq 0$. When Algorithm 2.3 terminates, D is empty. Hence $q_I = \rho(q_I,\vec{a}_I)$ for all $I \in E$. Thus $Q \in \ker(\vec{A})$.

(c) We proceed by induction on the number of times step 6 of Algorithm 2.3 is executed. If step 6 is never executed, then $\text{rem}(P,\vec{A}) = P$. Assume step 6 is executed at least once. After the first execution of step 6, $Q = P - \vec{B}\cdot\vec{A}$ where $\vec{B}$ is determined in steps 4, 5, and 6. By Lemma 2.3.1(c), we know $Q < P$. But $\text{rem}(P,\vec{A}) = \text{rem}(Q,\vec{A})$, and the computation of $\text{rem}(Q,\vec{A})$ requires one less execution of step 6 than does $\text{rem}(P,\vec{A})$. Hence, by our induction hypothesis, $\text{rem}(Q,\vec{A}) \leq Q$. Thus $\text{rem}(P,\vec{A}) = \text{rem}(Q,\vec{A}) \leq Q < P$. Therefore, $\text{rem}(P,\vec{A}) \leq P$, and we are done.

(d) Going back to the proof of Lemma 2.3.4, $P - \text{rem}(P,\vec{A}) = \sum_{i=1}^{j} \vec{B}_i\cdot\vec{A} = (\sum_{i=1}^{j} \vec{B}_i)\cdot\vec{A}$. But $\partial(\sum_{i=1}^{j} \vec{B}_i) \leq \max_{1\leq i\leq n} \partial\vec{B}_i = J$. Hence $\sum_{i=1}^{j} \vec{B}_i$ is a simple representation for $P - \text{rem}(P,\vec{A})$ with respect to $\vec{A}$.□

We now combine the remainder with a complete ideal basis to obtain our simplifying ample function. But first we present a lemma dealing with the leading coefficient sub-bases of a complete basis.

Lemma 2.3.6 If $\vec{A}$ is a complete basis for ideal $\overline{A}$, then for all $I \in E$, $\vec{a}_I$ is a basis for the leading coefficient ideal $\overline{\overline{a}}_I$ (i.e. $\overline{a}_I = \overline{\overline{a}}_I$).

Proof Let $p \in \overline{\overline{a}}_I$. If $p = 0$ then $p \in \overline{a}_I$. If $p \neq 0$ then there exists $P \in \overline{A}$ such that $\text{lc}(P) = p$ and $\partial P | I$. Since $\vec{A} = \langle A_1,\ldots,A_n\rangle$ is complete, P has a simple representation $\vec{S} = \langle S_1,\ldots,S_n\rangle \in (R[x])^n$ and $\partial\vec{S} = \partial P$. Permute $\vec{A}$ such that $S_i \neq 0$, $A_i \neq 0$, and $\partial S_i + \partial A_i = \partial\vec{S}$

for $1 \leq i \leq \mu\vec{S}$. Then

$$p = lc(P) = \sum_{i=1}^{\mu\vec{S}} lc(S_i)lc(A_i).$$

Now for $1 \leq i \leq \mu\vec{S}$, $\partial S_i + \partial A_i = \partial P$, hence $\partial A_i | I$, and therefore $lc(A_i) \in \vec{\bar{a}}_I$. Therefore $p \in \bar{\bar{a}}_I$, and $\vec{\bar{a}}_I \subseteq \bar{\bar{a}}_I$. We have already noted that $\bar{\bar{a}}_I \subseteq \vec{\bar{a}}_I$ so $\bar{\bar{a}}_I = \vec{\bar{a}}_I$.□

The next theorem gives the crucial properties of the remainder when used with a complete basis.

Theorem 2.3 Let $\vec{A}$ be a basis for ideal $\bar{A}$ in $R[x]$, and let $P,P' \in R[x]$. Then the following are equivalent.

(a) $\vec{A}$ is a complete basis for $\bar{A}$;

(b) If $P \equiv P' \bmod \bar{A}$ then $rem(P,\vec{A}) = rem(P',\vec{A})$;

(c) If $P \in \bar{A}$ then $rem(P,\vec{A}) = 0$.

Proof (b) =>(c). If $P \in \bar{A}$ then $P \equiv 0 \bmod \bar{A}$. Hence $rem(P,\vec{A}) = rem(0,\vec{A}) = 0$.

(c) => (a). By Lemma 2.3.5(d), $P - rem(P,\vec{A})$ is simply constructible from A. If $P \in \bar{A}$ then $P - rem(P,\vec{A}) = P - 0 = P$ is simply constructible from $\vec{A}$. Hence $\vec{A}$ is a complete basis for $\bar{A}$.

(a) => (b). Let $Q = rem(P,\vec{A}) = \sum_{I \in E} q_I x^I$, $Q' = rem(P',\vec{A}) = \sum_{I \in E} q_I' x^I$, $Q'' = Q - Q'$, $q'' = lc(Q'')$, $J = \partial Q''$, and assume $Q'' \neq 0$. Then $q'' \in \vec{\bar{a}}_J$. By Lemma 2.3.6, since $\vec{A}$ is complete, $\vec{\bar{a}}_J = \bar{a}_J$. Hence $q_J - q_J' = q'' \in \bar{a}_J$ and $q_J \equiv q_J' \bmod \bar{a}_J$. By Lemma 2.3.5(b), $Q,Q' \in \ker(\vec{A})$. Hence $q_J = \rho(q_J,\vec{a}_J)$ and $q_J' = \rho(q_J',\vec{a}_J)$. Since

$q_J \equiv q_J'$ mod $\overline{a}_J$ and ρ is an ample function, $\rho(q_J,\vec{a}_J) = \rho(q_J',\vec{a}_J)$. Hence $q_J = q_J'$, $q'' = 0$ and $Q'' = 0$.□

We now conclude this section with a corollary establishing our desired simplifying ample function (see Definition 1.3.1).

Corollary 2.3 If $\vec{A}$ is a complete basis for ideal $\overline{A}$ in R[x], then rem(P,$\vec{A}$) is a simplifying ample function for the triple (R[x], $\equiv$ mod $\overline{A}$, $\leq$).

Proof (a) rem(P,$\vec{A}$) $\equiv$ P mod $\vec{A}$ by Lemma 2.3.5(a). (b) if P,Q $\in$ R[x] and P $\equiv$ Q mod $\overline{A}$, then rem(P,$\vec{A}$) = rem(Q,$\vec{A}$) by Theorem 2.3(b). (c) rem(P,$\vec{A}$) $\leq$ P by Lemma 2.3.5(c) □

We note in passing that Lemma 1.2.2 guarantees that the simplifying ample function given by rem(P,$\vec{A}$) is independent of the complete basis $\vec{A}$. Indeed if $\vec{A} = \langle A_1,\ldots,A_n\rangle$ is a complete basis for $\overline{A}$ and $A \in \overline{A}$, then $\vec{A}' = \langle A_1,\ldots,A_n,A\rangle$ is also a complete basis for $\overline{A}$ since any polynomial having the simple representation $\langle S_1,\ldots,S_n\rangle$ in terms of $\vec{A}$ has the simple representation $\langle S_1,\ldots,S_n,0\rangle$ in terms of $\vec{A}'$.

2.4 Consensus Formation and Complete Bases

In the previous section we have shown that we have a simplifying ample function for $(R[x], \equiv \bmod \overline{A}, \leq)$ if we can find a complete basis for the ideal $\overline{A}$. Theorem 2.3 states that a basis is complete if and only if the remainder of all elements of $\overline{A}$ is zero with respect to this basis. This criterion is hardly suitable for deciding when a basis is complete -- let alone the problem of trying to find such a basis. In this section we characterize a complete basis in terms of a smaller set of polynomials. In the next section this characterization will provide an algorithm for computing a complete basis.

The informal remarks in Section 2.1 indicated that combinations of the original algebraic relations can be added to the initial set without changing the relations between the variables. We make this idea more precise by defining the consensus operation.

Let $\vec{B} = \langle B_1,\ldots,B_n\rangle \in (R[x])^n$ and let $\vec{b} = \langle b_1,\ldots,b_n\rangle$ where $b_i = lc(B_i)$, $1 \leq i \leq n$. Let $V(\vec{b})$ be the R-module $V(\vec{b}) = \{\vec{s} \in R^n : \vec{s}\cdot\vec{b} = 0\}$. Since R is a simplification ring, there is a function $\beta(\vec{b})$ which determines a finite sequence of generators for $V(\vec{b})$. Let $\beta(\vec{b}) = \langle\vec{c}_1,\ldots,\vec{c}_t\rangle$ where $\vec{c}_j = \langle c_{j1},\ldots,c_{jn}\rangle \in R^n$ for $1 \leq j \leq t = t(\vec{b})$, the number of vectors in the set of generators.

Furthermore let $J = lcm(\partial B_1,\ldots,\partial B_n)$, i.e. J is the exponent vector of the least common multiple of the leading monomials of the non-zero B_i's. If each $B_i = 0$ then we set $J = \partial 0$.

Definition 2.4.1 For $1 \leq j \leq t(\vec{b})$ let $\vec{C}_j = \langle C_{j1},\ldots,C_{jn}\rangle$ where if $B_i \neq 0$, then $C_{ji} = c_{ji}x^{J-\partial B_i}$ otherwise $C_{ji} = 0$. Then the j-th consensus of $\vec{B}$ is defined to be $\text{cons}_j(\vec{B}) = \vec{C}_j \cdot \vec{B}$.

Note $\vec{C}_j$ is a representation for $\text{cons}_j(\vec{b})$ with respect to $\vec{B}$, and $\partial\vec{C}_j = J$. If $\vec{C}_j \neq \langle 0,\ldots,0\rangle$, the terms of degree J of the components of the inner product $\vec{C}_j \cdot \vec{B}$ cancel, and it is clear that $\text{cons}_j(\vec{B})$ is a polynomial in R[x] such that $\partial(\text{cons}_j(\vec{B})) < J$.

We now relate the consensus to an ideal basis. As before, let $\vec{A} = \langle A_1,\ldots,A_n\rangle$ be a basis for ideal $\overline{A}$ in R[x].

Definition 2.4.2 Let $W(\vec{A}) = \{\langle B_1,\ldots,B_n\rangle \in (R[x])^n$: for $1 \leq i \leq n$ either $B_i = A_i$ or $B_i = 0\}$. For $\vec{B} \in W(\vec{A})$ we define $\vec{b}$, $V(\vec{b})$, and $t(\vec{b})$ as above. Then the set $\text{cons}(\vec{A}) = \{\text{cons}_j(\vec{B}) : \vec{B} \in W(\vec{A})$ and $1 \leq j \leq t(\vec{b})\}$ will be called the consensus set of basis A.

Note that $\text{cons}(\vec{A})$ is a finite subset of $\overline{A}$. Furthermore, any element $C \in \text{cons}(\vec{A})$ can be written as $C = \vec{C}\cdot\vec{A}$ and if $\vec{C} \neq \langle 0,\ldots,0\rangle$ then $\partial\vec{C} > \partial C$. The representation $\vec{C}$ can be found using Definition 2.4.1.

The word "consensus" has been suggested by the similarity between this operation and the consensus operation in the consensus method of finding prime implicants in Boolean algebra [MEN70]. In both cases the output expression is found by canceling, in some sense, certain elements of the input expressions.

We now give a technical lemma relating consensus formation to other combinations of polynomials. Essentially this lemma says that

if the elements of $\vec{B} \in W(\vec{A})$ are combined such that the leading terms of the components of the sum cancel, then the combination can be written in terms of the consensus polynomials of $\vec{B}$.

Lemma 2.4.1 Let $\vec{B} = \langle B_1,\ldots,B_n\rangle \in W(\vec{A})$ with $\vec{b}$, $V(\vec{b})$, and $t = t(\vec{b})$ defined as above and $\vec{B} \neq \langle 0,\ldots,0\rangle$. Let $K \in E$ be a multiple of $J = \text{lcm}(\partial B_1,\ldots,\partial B_n)$, and suppose there exists $\vec{d} = \langle d_1,\ldots,d_n\rangle \in V(\vec{B})$, i.e. $\vec{d}\cdot\vec{b} = 0$. Also let $\vec{D} = \langle D_1,\ldots,D_n\rangle$ where if $B_i \neq 0$ then $D_i = d_i x^{K-\partial B_i}$, otherwise $D_i = 0$, $1 \leq i \leq n$. Then there exist $g_j \in R$, $1 \leq j \leq t$, such that

$$\vec{D}\cdot\vec{B} = x^{K-J} \sum_{j=1}^{t} g_j \, \text{cons}_j(\vec{B}).$$

Proof Let us assume that R-module $V(\vec{b})$ has a set of generators $\beta(\vec{b}) = \langle \vec{c}_1,\ldots,\vec{c}_t\rangle$ where $\vec{c}_j = \langle c_{j1},\ldots,c_{jn}\rangle \in R^n$ for $1 \leq j \leq t$. Now $\vec{d} \in V(\vec{b})$, hence there exist $g_j \in R$, $1 \leq j \leq t$ such that $\vec{d} = \sum_{j=1}^{t} g_j \vec{c}_j$. Thus $d_i = \sum_{j=1}^{t} g_j c_{ji}$ for $1 \leq i \leq n$.

If $B_i \neq 0$ then $D_i = d_i x^{K-\partial B_i} = \sum_{j=1}^{t} g_j (c_{ji} x^{K-\partial B_i})$, for $1 \leq i \leq n$.

But then

$$\vec{D}\cdot\vec{B} = \sum_{i=1}^{n} D_i B_i = \sum_{\substack{i=1 \\ B_i \neq 0}}^{n} \sum_{j=1}^{t} g_j (c_{ji} x^{K-\partial B_i}) B_i$$

$$= \sum_{j=1}^{t} g_j \left(\sum_{\substack{i=1 \\ B_i \neq 0}}^{n} c_{ji} x^{K-\partial B_i} B_i \right)$$

$$= x^{K-J} \sum_{j=1}^{t} g_j \Big(\sum_{\substack{i=1 \\ B_i \neq 0}}^{n} c_{ji} x^{J-\partial B_i} B_i \Big)$$

$= x^{K-J} \sum_{j=1}^{t} g_j \, cons_j(\vec{B})$, because the zero elements of $\vec{B}$ do not contribute to $cons_j(\vec{B})$ □

We can now use the consensus operation to give another characterization of a complete basis. The statement of Theorem 2.4 is intentionally parallel to that of Theorem 2.3.

Theorem 2.4 Let $\vec{A}$ be a basis for ideal $\overline{A}$ in R[x]. The following statements are equivalent.

(a) $\vec{A}$ is a complete basis for $\overline{A}$;

(b) If $C \in cons(\vec{A})$ then C is simply constructible from $\vec{A}$;

(c) If $C \in cons(\vec{A})$ then $rem(C,\vec{A}) = 0$.

Proof (a) => (c). Clearly, $cons(\vec{A}) \subseteq \overline{A}$. Hence if $C \in cons(\vec{A})$ and $\vec{A}$ is a complete basis, then by Theorem 2.3(c), $rem(C,\vec{A}) = 0$.

(c) => (b). By Lemma 2.3.5(d), $C - rem(C,\vec{A})$ is simply constructible from $\vec{A}$ for any C and $\vec{A}$. If $C \in cons(\vec{A})$, then $rem(C,\vec{A}) = 0$. Hence $C - rem(C,\vec{A}) = C - 0 = C$ is simply constructible from $\vec{A}$.

(b) => (a). Let $P \in \overline{A}$. To show $\vec{A}$ is a complete basis we must prove that there exists a simple representation for P with respect to $\vec{A}$. If P = 0 then the representation $\vec{S} = \langle 0,\ldots,0\rangle$ is clearly a simple representation for P. Now suppose $P \neq 0$, and, using the well-ordering of E by $\leq$, let $\vec{S} = \langle S_1,\ldots,S_n\rangle \in (R[x])^n$ be a representation

of P with respect to $\vec{A}$ of least degree. If $\partial\vec{S} = \partial P$ then $\vec{S}$ is a simple representation for P, and we are done. Therefore, assume that $\partial\vec{S} > \partial P$, i.e. that P is not simply constructible from $\vec{A}$. We will construct from $\vec{S}$ a representation $\vec{U}$ for P such that $\partial\vec{U} < \partial\vec{S}$, obtaining a contradiction.

For convenience, we order the basis $\vec{A}$ such that $S_i \neq 0$, $A_i \neq 0$, and $\partial S_i + \partial A_i = \partial\vec{S}$ for $1 \leq i \leq \mu\vec{S}$, and either $S_i = 0$, $A_i = 0$, or $S_i \neq 0$, $A_i \neq 0$, and $\partial S_i + \partial A_i < \partial\vec{S}$ for $\mu\vec{S} < i \leq n$. As before $\mu\vec{S}$ is the multiplicity of representation $\vec{S}$. Let $K = \partial\vec{S}$ and $m = \mu\vec{S}$. Thus we can write

$$P = \vec{S}\cdot\vec{A} = \sum_{i=1}^{m} lt(S_i)A_i + \sum_{i=1}^{m} rd(S_i)A_i + \sum_{i=m+1}^{n} S_iA_i.$$

By definition of m, the only part of this expression which contributes terms of degree K to the representation $\vec{S}$ is the first summation. Since $K = \partial A_i + \partial S_i$ for $1 \leq i \leq m$ we can write

$$\sum_{i=1}^{m} lt(S_i)A_i = \sum_{i=1}^{m} lc(S_i)A_i x^{K-\partial A_i}.$$

Now since $K > \partial P$, we know that $\sum_{i=1}^{m} lc(S_i)lc(A_i) = 0$. Let $\vec{B} = \langle A_1,\ldots,A_m, 0,\ldots,0\rangle$ where we have appended n - m zeros. Then $\vec{B} \in W(\vec{A})$ with $\vec{b} = \langle lc(A_1),\ldots,lc(A_m), 0,\ldots,0\rangle$. As in Definition 2.4.1, let $t = t(\vec{b})$ be the number of vectors in the finite representation given by $\beta(\vec{b})$ for $V(\vec{b})$. Let $\vec{D} = \langle D_1,\ldots,D_n\rangle$ such that, for $1 \leq i \leq m$, $D_i = lc(S_i)x^{K-\partial A_i}$ and, for $m < i \leq n$, $D_i = 0$. Then

$\vec{d} = \langle lc(D_1),\ldots,lc(D_n)\rangle \in V(\vec{b})$. Therefore, when Lemma 2.4.1 is applied, we see that there exist $g_j \in R$, $1 \le j \le t$ such that

$$\vec{D}\cdot\vec{B} = \sum_{i=1}^{n} D_i B_i = \sum_{i=1}^{m} lc(S_i)A_i x^{K-\partial A_i} = x^{K-J} \sum_{j=1}^{t} g_j \, cons_j(\vec{B})$$

where $J = lcm(\partial A_1,\ldots,\partial A_m)$, and $\partial(cons_j(\vec{B})) < J$.

Hypothesis (b) implies that $cons_j(\vec{B})$ has a simple representation with respect to $\vec{A}$, i.e. that there exist $\vec{T}_j = \langle T_{j1},\ldots,T_{jn}\rangle \in R[x])^n$, $1 \le j \le t$ such that $cons_j(\vec{B}) = \vec{T}_j\cdot\vec{A}$ and $\partial\vec{T}_j = \partial(cons_j(\vec{B}))$.

Therefore,

$$\begin{aligned}\sum_{i=1}^{m} lt(S_i)A_i &= x^{K-J} \sum_{j=1}^{t} g_j \, cons_j(\vec{B}) \\ &= x^{K-J} \sum_{j=1}^{t} g_j \, \vec{T}_j\cdot\vec{A} \\ &= x^{K-J} \sum_{j=1}^{t} g_j \Big(\sum_{i=1}^{n} T_{ji}A_i\Big).\end{aligned}$$

Thus

$$P = x^{K-J} \sum_{i=1}^{n} \Big(\sum_{j=1}^{t} g_j T_{ji}\Big)A_i + \sum_{i=1}^{m} rd(S_i)A_i + \sum_{i=m+1}^{n} S_i A_i \ .$$

Now let representation $\vec{U} = \langle U_1,\ldots,U_n\rangle$ be defined by

$$U_i = x^{K-J} \sum_{j=1}^{t} g_i T_{ji} + rd(S_i) \ , \qquad 1 \le i \le m \ ,$$

$$U_i = x^{K-J} \sum_{j=1}^{t} g_j T_{ji} + S_i \ , \qquad m < i \leq n.$$

Hence $P = \vec{U} \cdot \vec{A}$, and we claim $\partial \vec{U} < K$.

It suffices to show that if $A_i \neq 0$ then $\partial U_i < K - \partial A_i$. Since $\vec{T}_j$ is a simple representation of $cons_j(\vec{B})$ and $\partial(cons_j(\vec{B})) < J$, $\partial(T_{ji}) < J - \partial A_i$. Hence $\partial(x^{K-J} \sum_{j=1}^{t} g_j T_{ji}) < (K-J) + (J - \partial A_i) = K - \partial A_i$. Now assume also that $S_i \neq 0$. Since $\vec{S}$ is a representation of P of degree K, $\partial(rd(S_i)) < \partial(S_i) \leq K - \partial A_i$, and if $i > m$ then $\partial(S_i) < K - \partial A_i$ by our assumptions about the multiplicity of $\vec{S}$, and the arrangement of the basis $\vec{A}$. Hence $\partial U_i < K - \partial A_i$ by definition of U_i.

But $\partial \vec{U} < K = \partial \vec{S}$ contradicts the assumption that $\partial \vec{S}$ is minimal, completing the proof that (a) => (b).□

2.5 Construction of Complete Bases

Theorem 2.4 provides a way to check whether a given ideal basis $\vec{A}$ is complete or not, i.e. $\vec{A}$ is complete iff all polynomials in cons($\vec{A}$) remainder to zero. This theorem does not, however, give a method for constructing a complete basis. If an element, say C, of cons($\vec{A}$) does not remainder to zero, then A = rem(C,$\vec{A}$) is, in a sense, "irreducible" with respect to $\vec{A}$. More precisely, we know that A is a kernel polynomial with respect to $\vec{A}$. If we add A to basis $\vec{A} = \langle A_1,\ldots,A_n\rangle$, then $\overline{A}$ does not change but A is now simply constructible from $\vec{A}' = \langle A_1,\ldots,A_n,A\rangle$. We will see in this section that, since lt(A) $\in$ ker ($\vec{A}$), repeating this process leads to the construction of an ascending chain of ideals generated by these leading terms. Thus the process of adding non-zero consensus remainders must terminate.

These ideas are made precise in the following algorithm and its validity proof.

Algorithm 2.5

$\vec{B} \leftarrow$ cbasis($\vec{A}$)

[Complete ideal basis formation.

Input: $\vec{A}=\langle A_1,\ldots,A_n\rangle \in (R[x])^n$, a basis for ideal $\overline{A}$ in R[x].

Output: $\vec{B}=\langle B_1,\ldots,B_s\rangle \in (R[x])^s$, a complete basis for $\overline{A}$.]

(1) [Initialize.] $\vec{B} \leftarrow \vec{A}$.

(2) [Find consensus polynomials with non-zero remainders.]
$\hat{C} \leftarrow \{C \in \text{cons}(\vec{B}) : \text{rem}(C,\vec{B}) \neq 0\}$; if $\hat{C}$ is empty then return.

(3) [Apply consensus choice function.] Choose $C \in \hat{C}$.

(4) [Compute remainder.] A←rem(C,$\vec{B}$).

(5) [Adjoin to basis.] Append A to $\vec{B}$; go to step 2 □

Before proving termination and correctness for Algorithm 2.5 we note that as with Algorithm 2.3 we have presented an abstract algorithm. We wish to emphasize that each of the abstract sub-algorithms implements a function, and hence cbasis, if it terminates, is a function mapping (R[x])* into itself. There are three areas which require further specification for a concrete realization of the algorithm. In step 2 the consensus operation assumes that the function β associated with the simplification ring R will be applied to determine a finite set of generators for various R-modules $V(\vec{b})$, where $\vec{b}$ is a vector of zeros and leading coefficients of elements of $\vec{B}$. We have imposed as yet no restrictions on the generating sequence produced by β. In step 3 we apply a choice function that specifies which of the (possibly many) non-zero consensus remainders will be added to $\vec{B}$. Finally, in step 4, as in step 2 the remainder function itself is viewed abstractly. The first two of these areas will be dealt with precisely under the heading of consensus selection rules in Chapter 3.

Before proving that Algorithm 2.5 terminates we need a lemma about ideals generated by bases composed of single-term polynomials.

Lemma 2.5.1 Let $\vec{L} = \langle L_1,\ldots,L_n\rangle$, where for $1 \leq i \leq n$ either $L_i = a_i x^{\partial L_i}$ or $L_i = 0$, generate the ideal $\bar{L}$ in R[x]. Then $\vec{L}$ is a complete basis.

Proof We claim that $\text{cons}(\vec{L}) = \{0\}$. To do this we construct an element of $\text{cons}(\vec{L})$ following Definitions 2.4.1 and 2.4.2. Let $\vec{B} \in W(\vec{L})$, i.e. $\vec{B} = \langle B_1,\ldots,B_n\rangle$, where $B_i = L_i$ or $B_i = 0$ for $1 \le i \le n$. Let $\vec{b} = \langle b_1,\ldots,b_n\rangle = \langle \text{lc}(B_1),\ldots,\text{lc}(B_n)\rangle$ and $\langle \vec{c}_1,\ldots,\vec{c}_t\rangle = \beta(\vec{b})$ where $\vec{c}_j = \langle c_{j1},\ldots,c_{jn}\rangle$ for $1 \le j \le t$. Let $J = \text{lcm}(\partial B_1,\ldots,\partial B_n)$, since we may assume $\vec{B} \neq \langle 0,\ldots,0\rangle$. Then

$$\text{cons}_j(B) = \sum_{\substack{i=1 \\ B_i \neq 0}}^{n} (c_{ji}x^{J-\partial B_i})B_i$$

$$= \sum_{\substack{i=1 \\ B_i \neq 0}}^{n} (c_{ji}x^{J-\partial B_i})(b_i x^{\partial B_i})$$

$$= x^J \sum_{i=1}^{n} c_{ji}b_i$$

$$= x^J (\vec{c}_j \cdot \vec{b})$$

$$= 0, \text{ since } \vec{c}_j \in V(\vec{b}).$$

Thus $\text{cons}(\vec{L}) = \{0\}$ and, by Theorem 2.4, $\vec{L}$ is complete.□

We now prove that the cbasis algorithm terminates.

Lemma 2.5.2 Algorithm 2.5 always terminates.

Proof Let $\vec{B}_i$ be the basis $\vec{B}$ during the i-th execution of step 3 of Algorithm 2.5. Then $\vec{B}_1 = \langle A_1,\ldots,A_n\rangle$ and $\vec{B}_{i+1} = \langle A_1,\ldots,A_n, A_{n+1},\ldots,A_{n+i}\rangle$, i.e. A_{n+i} is the non-zero consensus remainder appended to the basis $\vec{B}_i$ on the i-th execution of step 5.

Finally let $\vec{L}_i = \langle lt(A_1),\ldots,lt(A_{n+i-1})\rangle$ generate $\bar{L}_i$, the leading term ideal of $\vec{B}_i$.

Now for $i \geq 1$, A_{n+1} is the remainder of some polynomial with respect to $\vec{B}_i$. Hence by Lemma 2.3.5(b), $A_{n+i} \in \ker(\vec{B}_i)$. We claim that $lt(A_{n+i}) \notin \bar{L}_i$.

Let $J = \partial A_{n+i}$ and let $\bar{a}_J(\vec{B}_i)$, $\bar{a}_J(\vec{L}_i)$, and $\bar{\bar{a}}_J(\bar{L}_i)$ be the degree J leading coefficient ideals of basis $\vec{B}_i$, basis $\vec{L}_i$, and ideal $\bar{L}_i$, respectively. Since $\vec{B}_i$ and $\vec{L}_i$ have the same leading terms $\bar{a}_J(\vec{B}_i)$ and $\bar{a}_J(\vec{L}_i)$ have the same bases (i.e. $\vec{a}_J(\vec{B}_i) = \vec{a}_J(\vec{L}_i)$). Hence $\bar{a}_J(\vec{B}_i) = \bar{a}_J(\vec{L}_i)$. Lemma 2.5.1 indicates that $\vec{L}_i$ is a complete basis for $\bar{L}_i$. Thus by Lemma 2.3.6 $\bar{a}_J(\vec{L}_i) = \bar{\bar{a}}_J(\bar{L}_i)$. Therefore, $\bar{a}_J(\vec{B}_i) = \bar{\bar{a}}_J(\bar{L}_i)$.

Now assume $lt(A_{n+i}) \in \bar{L}_i$, and let $a = lc(A_{n+i})$. Then $a \in \bar{\bar{a}}_J(\bar{L}_i) = \bar{a}_J(\vec{B}_i)$, and hence $\rho(a,\vec{a}_J(\vec{B}_i)) = 0$ where $\vec{a}_J(\vec{B}_i)$ is the degree J leading coefficient sub-basis of basis $\vec{B}_i$. But $lt(A_{n+i}) \in \ker(\vec{B}_i)$, so $a = \rho(a,\vec{a}_J(\vec{B}_i))$. Since $a = lc(A_{n+i}) \neq 0$, we have a contradiction which invalidates our assumption that $lt(A_{n+i}) \in \bar{L}_i$. And hence, $\bar{L}_i \subset \bar{L}_{i+1}$.

Therefore, if Algorithm 2.4 does not terminate, then it generates an infinite strictly ascending chain of ideals in R[x], namely,

$$\bar{L}_1 \subset \bar{L}_2 \subset \bar{L}_3 \subset \ldots$$

But this contradicts the fact that R[x] is Noetherian, so Algorithm 2.4 must terminate.□

Corollary 2.5.1 When Algorithm 2.5 terminates, $\vec{B}$ is a complete basis for $\overline{A}$.

Proof We have already noted that $\vec{B}$ is a basis for $\overline{A}$ (adding ideal elements to $\vec{A}$ does not change the ideal generated).

When Algorithm 2.5 terminates, $\hat{C}$ is empty. Hence by Theorem 2.4, $\vec{B}$ is a complete basis.□

Notice that the termination proof for Algorithm 2.5 given in Lemma 2.5.2 only requires that $lt(A_{n+i}) \in \ker(B_i)$. If, in the remainder algorithm (Algorithm 2.3), we replace D in step 3 by $D = \{\partial Q\}$ if $lc(Q) \neq \rho(lc(Q), \vec{a}_{\partial Q})$, or otherwise D is empty, then the output condition $Q \in \ker(\vec{A})$ is changed to $lt(Q) \in \ker(\vec{A})$. The modified remainder algorithm thus obtained can then be used in the cbasis algorithm, with correctness and termination of the new cbasis algorithm still being guaranteed by Lemma 2.5.2.

The polynomials output by the original remainder algorithm are kernel polynomials with respect to basis $\vec{A}$. The outputs of the modified remainder algorithm will be called semi-kernel polynomials.

Definition 2.5.1 Given a basis $\vec{A}$ with leading coefficient subbases $\vec{a}_I$, the set of semi-kernel polynomials with respect to $\vec{A}$ is given by $sker(\vec{A}) = \{B \in \overline{A} : B = 0 \text{ or } lc(B) = \rho(lc(B), \vec{a}_{\partial B})\}$.

A remainder algorithm which generates semi-kernel polynomials is called a semi-remainder algorithm, and is denoted by $Q \leftarrow srem(P, \vec{A})$.

2.6 Computing Representations in R[x]

In the next two sections we establish the main result of this chapter, namely, if R is a simplification ring then so is R[x]. In Section 2.2 we found that if R is a simplification ring, then R[x] is a Noetherian ring with identity. We have also defined a partial ordering, $\leq$, on R[x]. It remains to be shown that we can find analogues in R[x] for the functions θ, ρ, and β associated with R. To be precise we restate Definition 2.2.1 for R[x].

Definition 2.6.1 Let R[x] be a Noetherian ring with identity and let $\leq$ be a partial order on R[x]. Such a ring is called a simplification ring if the following additional conditions are satisfied.

(a) There exists a function $\theta^* : R[x] \times (R[x])^* \to (R[x])^*$ such that if $\vec{A} \in (R[x])^*$ and $B \in R[x]$ then the function $\rho^*(B,\vec{A}) = B - \vec{A} \cdot \theta^*(B,\vec{A})$ is a simplifying ample function for $(R[x], \equiv \bmod \overline{A}, \leq)$ where $\overline{A}$ is the ideal in R[x] generated by $\vec{A}$. In addition, we require $\theta^*(0,\vec{A}) = \langle 0,\ldots,0\rangle$.

(b) There exists a function $\beta^* : (R[x])^* \to ((R[x])^*)^*$ such that if $\vec{A} \in (R[x])^*$ then $\beta^*(\vec{A})$ is a finite sequence of generators for the R[x]-module $V(\vec{A}) = \{\vec{S} \in (R[x])^* : \vec{S} \cdot \vec{A} = 0\}$.

Section 2.7 deals with the formulation of β^*. The current section will show how θ^*, and hence ρ^*, can be constructed from variants of the remainder and complete basis functions presented in Sections 2.3 and 2.5.

The determination of ρ^* is quite straightforward. Since the function cbasis $(\vec{A})$ yields a complete basis for $\overline{A}$, the ideal generated by $\vec{A}$, Corollary 2.3 tells us that $\rho^*(P,\vec{A}) = \text{rem}(P,\text{cbasis}(\vec{A}))$ is a simplifying ample function for $(R[x], \equiv \bmod \overline{A}, \leq)$ and by Lemma 1.2.2, ρ^* is independent of the complete basis generated by cbasis $(\vec{A})$.

The function θ^*, however, must satisfy the condition that $P - \rho^*(P,\vec{A}) = \vec{A} \cdot \theta^*(P,\vec{A})$, i.e. that $\theta^*(P,\vec{A})$ is a representation of $P - \rho^*(P,\vec{A}) = P - \text{rem}(P,\text{cbasis}(\vec{A}))$ with respect to $\vec{A}$. In Algorithm 2.6.1 we will see how a slight modification to the remainder algorithm allows the computation of a representation for $P - \text{rem}(P,\vec{B})$ with respect to $\vec{B}$, where $\vec{B} \in (R[x])^*$. In fact, the representation obtained will be a simple representation. A similar change in the complete basis algorithm allows us to find a representation of each element of basis $\vec{B}$ with respect to $\vec{A}$ where $\vec{B} = \text{cbasis}(\vec{A})$. This will be seen in Algorithm 2.6.2. Finally Algorithm 2.6.3 combines these two results to produce θ^*.

The functions defined in Algorithms 2.6.1 and 2.6.2 will be referred to as extended remainder and extended complete basis functions. They are extensions of the remainder and complete basis functions in the same sense that the extended Euclidean algorithm is an extension of the Euclidean algorithm.

Algorithm 2.6.1

$$\text{remx}(P,\vec{A};Q,\vec{S})$$

[Remainder of polynomial with respect to ideal basis, extended.

Input: $P \in R[x]$; $\vec{A}=\langle A_1,\ldots,A_n\rangle \in (R[x])^n$, a basis for ideal $\bar{A}$.

Output: $Q \in R[x]$ and $\vec{S}=\langle S_1,\ldots,S_n\rangle \subset (R[x])^n$ such that $Q=\mathrm{rem}(P,\vec{A})$ and $\vec{S}$ is a simple representation for $P-Q$ with respect to basis $\vec{A}$.]

(1) [Initialize.] $Q \leftarrow P$; for $i=1,\ldots,n$ do $S_i \leftarrow 0$; $\vec{S} \leftarrow \langle S_1,\ldots,S_n\rangle$.

(2) [Is $Q=0$?] if $Q = 0$ then return.

(3) [Assuming $Q=\sum_{I\in E} q_I x^I$, determine possible reductions.]
$D \leftarrow \{I \in E : q_I \neq \rho(q_I,\vec{a}_I)\}$; if D is empty then return.

(4) [Apply remainder choice function.] Choose $J \in D$.

(5) [Use $\theta(q_J,\vec{a}_J)$ to determine $\rho(q_J,\vec{a}_J)$ in terms of $\vec{a}_J$.]
$\vec{b}=\langle b_1,\ldots,b_n\rangle \leftarrow \theta(q_J,\vec{a}_J)$.

(6) [Make new degree J coefficient the residue class representative for the residue class of q_J mod $\bar{a}_J$, i.e. $q_J \leftarrow \rho(q_J,\vec{a}_J)=q_J-\vec{b}\cdot\vec{a}_J$.]
for $i=1,\ldots,n$ do if $\partial A_i \mid J$ then $B_i \leftarrow b_i x^{J-\partial A_i}$ else $B_i \leftarrow 0$;
$\vec{B} \leftarrow \langle B_1,\ldots,B_n\rangle$; $Q \leftarrow Q-\vec{B}\cdot\vec{A}$.

(7) [Update multipliers.] $\vec{S} \leftarrow \vec{S}+\vec{B}$; go to step 2□

Lemma 2.6.1 Algorithm 2.6.1 always terminates, and when it does, $Q = \mathrm{rem}(P,\vec{A})$ and $\vec{S}$ is a simple representation for $P - Q$ with respect to $\vec{A}$.

Proof Algorithm 2.6.1 differs from Algorithm 2.3 only in the added initialization of $\vec{S}$ in step 1 and the modification of $\vec{S}$ in step 7. The termination of Algorithm 2.6.1 does not depend on either of these steps. Hence the proof in Lemma 2.3.3 that Algorithm 2.3 terminates suffices to prove Algorithm 2.6 also terminates.

The changes mentioned above do not affect the computation of Q from P. Hence $Q = \mathrm{rem}(P,\vec{A})$.

Using the notation of the proof of Lemma 2.3.3 we showed, in the proof of Lemma 2.3.5(d), that $\sum_{i=1}^{j} \vec{B}_i$ is a simple representation for $P - \mathrm{rem}(P,\vec{A})$. When Algorithm 2.6.1 terminates, $\vec{S} = \sum_{i=1}^{j} \vec{B}_i$. Hence $\vec{S}$ is a simple representation for $P - \mathrm{rem}(P,\vec{A})$.□

We now give an extended version of the cbasis algorithm.

Algorithm 2.6.2

cbasisx($\vec{A};\vec{B},\vec{T}$)

[Complete ideal basis formation, extended.

Input: $\vec{A}=\langle A_1,\ldots,A_n\rangle \in (R[x])^n$, a basis for ideal $\bar{A}$ in R[x].

Output: $\vec{B}=\langle B_1,\ldots,B_t\rangle \in (R[x])^t$, a complete basis for $\bar{A}$, and $\vec{T}=\langle\vec{T}_1,\ldots,\vec{T}_t\rangle$ such that for $1\le j\le t$, $\vec{T}_j \in (R[x])^n$ and $B_j=\vec{T}_j\cdot\vec{A}$, i.e. $\vec{T}_j$ is a representation for B_j with respect to $\vec{A}$.]

(1) [Initialize.] $\vec{B}\leftarrow\vec{A}$; for j=1,...,n do{ for i=1,...,n do if i=j then $T_{ji}\leftarrow 1$ else $T_{ji}\leftarrow 0$; $\vec{T}_j\leftarrow\langle T_{j1},\ldots,T_{jn}\rangle$ }; $\vec{T}\leftarrow\langle\vec{T}_1,\ldots,\vec{T}_n\rangle$.

(2) [Find consensus polynomials with non-zero remainders.] $\hat{C}\leftarrow\{\ C \in \mathrm{cons}(\vec{B}):\mathrm{rem}(C,\vec{B})\neq 0\ \}$; if $\hat{C}$ is empty then return.

(3) [Apply consensus choice function.] Choose $C \in \hat{C}$; set $\vec{C}$ to the representation of C with respect to $\vec{B}$, given in Definition 2.4.1.

(4) [Compute A=rem(C,$\vec{B}$), and a representation $\vec{S}$ for C-A.] remx($C,\vec{B};A,\vec{S}$).

(5) [Make $\vec{U}$ a representation of A with respect to $\vec{A}$. Assume $\vec{B}=\langle B_1,\ldots,B_s\rangle$, $\vec{C}=\langle C_1,\ldots,C_s\rangle$, $\vec{S}=\langle S_1,\ldots,S_s\rangle$, and $\vec{T}=\langle\vec{T}_1,\ldots,\vec{T}_s\rangle$.] $\vec{U}\leftarrow\sum_{j=1}^{s}(C_j-S_j)\vec{T}_j$.

(6) [Adjoin basis element and multipliers.] Append A to $\vec{B}$; append $\vec{U}$ to $\vec{T}$; go to 2 □

Lemma 2.6.2 Algorithm 2.6.2 always terminates, and, when it does, $\vec{B}$ is a complete basis for $\overline{A}$, and for $1 \leq j \leq t$, $B_j = \vec{T}_j \cdot \vec{A}$.

Proof Algorithm 2.6.2 differs from Algorithm 2.5 in the added initialization of $\vec{T}$ in step 1 and the added computation of $\vec{C}$, $\vec{S}$, $\vec{U}$ and $\vec{T}$ in steps 3 through 6. The termination of Algorithm 2.6.2 is not affected by any of these changes. Hence Lemma 2.5.2 shows that Algorithm 2.6.2 terminates.

The computation of $\vec{B}$ is also not affected by these changes, hence Corollary 2.5.1 shows that $\vec{B}$ is a complete basis for $\overline{A}$.

The final statement in the lemma is proved by induction on the number of times step 2 is executed in Algorithm 2.6.2. Let $\vec{B}_0 = \langle B_1,\ldots,B_n\rangle$ and $\vec{T}^{(0)} = \langle \vec{T}_1,\ldots,\vec{T}_n\rangle$ be the values of $\vec{B}$ and $\vec{T}$ just before the first execution of step 2. Then for $1 \leq j \leq n$, $\vec{B}_j = \vec{T}_j \cdot \vec{A}$ since $B_j = A_j$ in step 1.

Let $\vec{B}_i = \langle B_1,\ldots,B_s\rangle$ and $\vec{T}^{(i)} = \langle \vec{T}_1,\ldots,\vec{T}_s\rangle$ be the values of $\vec{B}$ and $\vec{T}$ at the i-th execution of step 2. Also let $\vec{C} = \langle C_1,\ldots,C_s\rangle$, $\vec{S} = \langle S_1,\ldots,S_s\rangle$, and $\vec{U} = \langle U_1,\ldots,U_n\rangle$ be the values of these variables at the i-th execution of step 6.

Note $\vec{T}^{(i+1)} = \langle \vec{T}_1,\ldots,\vec{T}_s,\vec{U}\rangle$ and by the induction hypothesis $B_j = \vec{T}_j \cdot \vec{A}$ for $1 \leq j \leq s$. Now

$$\vec{U}\cdot\vec{A} = \left(\sum_{j=1}^{s}(C_j-S_j)\vec{T}_j\right)\cdot\vec{A} = \sum_{j=1}^{s}(C_j-S_j)\vec{T}_j\cdot\vec{A} = \sum_{j=1}^{s}(C_j-S_j)B_j$$

$$= (\vec{C}-\vec{S})\cdot\vec{B} = \vec{C}\cdot\vec{B} - \vec{S}\cdot\vec{B} = C - (C-A) = A.$$

Hence $\vec{U}$ is a representation of A with respect to $\vec{A}$. And thus $\vec{T}^{(i+1)}$ is a sequence of representations of the elements of $\vec{B}_i = \langle B_1,\ldots,B_s,A\rangle$ with respect to $\vec{A}$ □

The two previous algorithms can be combined to yield θ*.

Algorithm 2.6.3

$$\vec{S}\leftarrow\text{theta}(P,\vec{A})$$

[θ function for the ring R[x].

Input: $P \in R[x]$; $\vec{A}=\langle A_1,\ldots,A_n\rangle \in (R[x])^n$, a basis for ideal $\overline{A}$.

Output: $\vec{S}=\langle S_1,\ldots,S_n\rangle \in (R[x])^n$ such that $\rho^*(P,\vec{A}) = P-\vec{A}\cdot\vec{S}$ is a simplifying ample function for $(R[x], \equiv \bmod A, \leq)$ and if P=0 then $\vec{S}=\langle 0,\ldots,0\rangle$.]

(1) [Compute a complete basis $\vec{B}$ for $\overline{A}$ and a representation $\vec{T}$ for $\vec{B}$ in terms of $\vec{A}$.] cbasisx($\vec{A};\vec{B},\vec{T}$).

(2) [Compute the residue class representative Q for P, along with a representation $\vec{U}$ of P-Q with respect to $\vec{B}$.] remx($P,\vec{B};Q,\vec{U}$).

(3) [Compute a representation $\vec{S}$ of P-Q in terms of $\vec{A}$. Let $\vec{T}=\langle\vec{T}_1,\ldots,\vec{T}_s\rangle$ and $\vec{U}=\langle U_1,\ldots,U_s\rangle$.] $\vec{S} \leftarrow \sum_{j=1}^{s} U_j\vec{T}_j$; return □

Lemma 2.6.3 Algorithm 2.6.3 always terminates, and when it does $\rho^*(P,\vec{A}) = P-\vec{A}\cdot\vec{S}$ is a simplifying ample function for

$(R[x], \equiv \bmod \overline{A}, \leq)$. In addition, if $P = 0$ then $\vec{S} = \langle 0,\ldots,0\rangle$.

Proof The termination proof is trivial, since each of the sub-algorithms is known to terminate. At the end of step 3 let $\vec{A} = \langle A_1,\ldots,A_n\rangle$, $\vec{B} = \langle B_1,\ldots,B_s\rangle$, $\vec{T} = \langle \vec{T}_1,\ldots,\vec{T}_s\rangle$ where $\vec{T}_j = \langle T_{j1},\ldots,T_{jn}\rangle$ for $1 \leq j \leq s$; also let $\vec{U} = \langle U_1,\ldots,U_s\rangle$.

Then $\rho^*(P,\vec{A}) = P - \vec{S}\cdot\vec{A} = P - (\sum_{j=1}^{s} U_j\vec{T}_j)\cdot\vec{A} = P - \sum_{j=1}^{s} U_j(\vec{T}_j\cdot\vec{A}) =$ $P - \sum_{j=1}^{s} U_jB_j = P - (P - Q) = Q = \mathrm{rem}(P,\vec{B}) = \mathrm{rem}(P,\mathrm{cbasis}(\vec{A}))$. Hence by Corollary 2.3, $\rho^*(P,\vec{A})$ is a simplifying function for $(R[x], \equiv \bmod \overline{A}, \leq)$.

Furthermore, if $P = 0$ then remx$(P,\vec{B}; Q,\vec{U})$ returns $\vec{U} = \langle 0,\ldots,0\rangle$ (see Algorithm 2.6.1). Hence $\vec{S} = \sum_{j=1}^{s} U_j\vec{T}_j = \langle 0,\ldots,0\rangle$. Thus the lemma is proved.□

2.7 R[x] is a Simplification Ring

In this section we complete the proof that $R[x] = R[x_1,\ldots,x_r]$ is a simplification ring. To do this we must find an algorithm for the function β^* of Definition 2.6.1. If $\vec{A} \in (R[x])^n$ then $\beta^*(\vec{A}) = \vec{Y} = \langle \vec{Y}_1,\ldots,\vec{Y}_u \rangle$ must be a finite sequence of generators for the R[x]-module $V(\vec{A}) = \{\vec{S} \in (R[x])^n : \vec{S}\cdot\vec{A} = 0\}$. Note that any element $\vec{S} \in V(\vec{A})$ is actually a representation of zero with respect to $\vec{A}$. Hence, we are seeking a sequence of representations of zero, namely $\vec{Y}_k \in (R[x])^n$, $1 \leq k \leq u$, where if $\vec{S} \in V(\vec{A})$ then there exist polynomials $P_1,\ldots,P_u \in R[x]$ such that $\vec{S} = \sum_{j=1}^{u} P_j\vec{Y}_j$.

Consider the case where $\vec{A}$ is a complete basis. Let $C \in \text{cons}(\vec{A})$, and note that if $C \neq 0$ then we have two different representations for C with respect to $\vec{A}$. The first representation, $\vec{C}$, is given in the consensus definition (2.4.2), where it was also observed that, if $\vec{C} \neq \langle 0,\ldots,0 \rangle$, $\partial\vec{C} > \partial C$. Since $\vec{A}$ is a complete basis, and since $C \in \overline{A}$, the ideal generated by $\vec{A}$, we know that C has a simple representation, say $\vec{T}$, with respect to $\vec{A}$. Since $\vec{T}$ is a simple representation, $\partial\vec{T} = \partial C < \partial\vec{C}$. Let $\vec{T}$ be our second representation for C, and note that, since $\text{rem}(C,\vec{A}) = 0$, $\text{remx}(C,\vec{A};\ Q,\vec{T})$ determines such a simple representation for C. But then $\vec{C}\cdot\vec{A} = C = \vec{T}\cdot\vec{A}$, and hence $0 = \vec{C}\cdot\vec{A} - \vec{T}\cdot\vec{A} = (\vec{C} - \vec{T})\cdot\vec{A}$ so $\vec{C} - \vec{T} \in V(\vec{A})$. In Lemma 2.7.1 we show that $\langle \vec{C} - \vec{T} : C \in \text{cons}(\vec{A}) \rangle$ is a sequence of generators for $V(\vec{A})$ when $\vec{A}$ is a complete basis. When A is not complete, the situation is only a little more complicated as will be seen when β^* is realized in Algorithm 2.7.2.

The process of finding a sequence of generators for $V(\vec{A})$ when $\vec{A}$ is a complete basis is made precise in the following algorithm.

Algorithm 2.7.1

$$\vec{Y} \leftarrow \text{modgen}(\vec{A})$$

[Computation of sequence of module generators for the R[x]-module $V(\vec{A})$ when $\vec{A}$ is a complete basis.

Input: $\vec{A}=\langle A_1,\ldots,A_n\rangle \in (R[x])^n$, a complete basis for ideal $\overline{A}$.

Output: $\vec{Y}=\langle \vec{Y}_1,\ldots,\vec{Y}_u\rangle$ where $\vec{Y}_k \in (R[x])^n$ for $1 \leq k \leq u$, a finite sequence of generators for $V(\vec{A})= \{\vec{S} \in (R[x])^n : S \cdot A=0\}$.]

(1) [Initialize.] $\vec{Y} \leftarrow$ empty; for i=1,...,n if $A_i=0$ then { for j=1,...,n if i=j then $y_j \leftarrow 1$ else $y_j \leftarrow 0$; append $\langle y_1,\ldots,y_n\rangle$ to $\vec{Y}$ }; $\hat{C} \leftarrow \text{cons}(\vec{A})$.

(2) [Select consensus.] if $\hat{C}$ is empty then return; choose $C \in \hat{C}$; $\hat{C} \leftarrow \hat{C} - \{C\}$.

(3) [Non-simple representation for C.] Set $\vec{C}$ to the representation of C with respect to $\vec{A}$ given in Definition 2.4.1.

(4) [Simple representation for C.] remx$(C,\vec{A};Q,\vec{T})$.

(5) [Form new generator.] $\vec{Y}' \leftarrow \vec{C} - \vec{T}$; append $\vec{Y}'$ to $\vec{Y}$; go to 4 □

Lemma 2.7.1 Algorithm 2.7.1 always terminates, and when it does, $\vec{Y} = \langle \vec{Y}_1,\ldots,\vec{Y}_u\rangle$ is a finite sequence of generators for $V(\vec{A})$.

Proof Since each sub-algorithm always terminates, and since $\text{cons}(\vec{A})$ is finite, Algorithm 2.7.1 always terminates.

If $\vec{Y}_i \in \vec{Y}$, then either $\vec{Y}_i = \langle 0,\ldots,0,1,0,\ldots,0\rangle$ with a 1 in the i-th position and $A_i = 0$, or $\vec{Y}_i = \vec{C} - \vec{T}$ where $\vec{C}$ and $\vec{T}$ are two representations for some element $C \in \text{cons}(\vec{A})$. In either case $\vec{Y}_i \cdot \vec{A} = 0$

and hence $\vec{Y}_i \in V(\vec{A})$. We must now show $\vec{Y}$ generates all of $V(\vec{A})$.

Let $\vec{A} = \langle A_1,\ldots,A_n\rangle$ and let $\vec{S} = \langle S_1,\ldots,S_n\rangle \in V(\vec{A})$. We will prove that $\vec{S}$ is generated by $\vec{Y}$ by induction on $\partial\vec{S}$.

If $\partial\vec{S} = \partial 0$ then for $1 \leq i \leq n$, $S_i = 0$ or $A_i = 0$. Let us assume that $A_i = 0$ for $1 \leq i \leq m$ and $A_i \neq 0$ for $0 \leq m < i \leq n$. Then since $\partial\vec{S} = \partial 0$, $S_i = 0$ for $m < i \leq n$. So $\vec{S} = \langle S_1,\ldots,S_m,0,\ldots,0\rangle$. In step 1 of Algorithm 2.7.1 the zero representations $\vec{Y}_j = \langle y_{j1},\ldots,y_{jn}\rangle$ where $y_{ji} = 0$ if $i \neq j$, $y_{ji} = 1$ if $i = j$, for $1 \leq j \leq m$, $1 \leq i \leq n$, have been appended to $\vec{Y}$. But then $\vec{S} = \sum_{j=1}^{m} S_j\vec{Y}_j$ and $\vec{S}$ is generated by $\vec{Y}$.

Now assume $\partial S > \partial 0$. Let $a_i = lc(A_i)$ and $s_i = lc(S_i)$, $1 \leq i \leq n$, and let $K = \partial\vec{S}$ and $m = \mu\vec{S}$. Assume now without loss of generality that $\vec{A}$ is organized such that $S_i \neq 0$, $A_i \neq 0$ and $\partial S_i + \partial A_i = K$ for $1 \leq i \leq m$, and either $S_i = 0$, $A_i = 0$, or $S_i \neq 0$, $A_i \neq 0$ and $\partial S_i + \partial A_i < K$ for $m < i \leq n$.

Write $\vec{S} = \vec{S}' + \vec{S}''$ where $\vec{S}' = \langle s_1x^{K-\partial A_1},\ldots,s_mx^{K-\partial A_m},0,\ldots,0\rangle$ and $\vec{S}'' = \langle rd(S_1),\ldots,rd(S_m),S_{m+1},\ldots,S_n\rangle$. Since K is a multiple of $J = lcm(\partial A_1,\ldots,\partial A_m)$, we can write $\vec{S}' = x^{K-J}\langle s_1x^{J-\partial A_1},\ldots,s_mx^{J-\partial A_m},0,\ldots,0\rangle$. And since $\partial\vec{S} < K$ then $\vec{S}'\cdot\langle lt(A_1),\ldots,lt(A_m),0,\ldots,0\rangle = 0$ and $\sum_{i=1}^{m} s_ia_i = 0$.

Let $\vec{B} = \langle A_1,\ldots,A_m,0,\ldots,0\rangle \in (R[x])^n$, and $\vec{b} = \langle a_1,\ldots,a_m,0,\ldots,0\rangle \in R^n$. Then $\vec{s} = \langle s_1,\ldots,s_m,0,\ldots,0\rangle \in R^n$ and $\vec{s} \in V(\vec{b})$ since $\vec{s}\cdot\vec{b} = 0$. Let $\beta(\vec{b}) = \langle\vec{c}_1,\ldots,\vec{c}_t\rangle$ where the $\vec{c}_j = \langle c_{j1},\ldots,c_{jn}\rangle \in R^n$, $1 \leq j \leq t$, generate $V(\vec{b})$. Thus there exist $g_j \in R$, $1 \leq j \leq t$ such that

$$\vec{s} = \sum_{j=1}^{t} g_j \vec{c}_j .$$

Now, by Definition 2.4.1, $cons_j(B) = \vec{C}_j \cdot \vec{B}$ where $\vec{C}_j = \langle c_{j1}x^{J-\partial A_1},\ldots,c_{jm}x^{J-\partial A_m},0,\ldots,0\rangle \in (R[x])^n$. Hence

$$\begin{aligned}\vec{S}' &= x^{K-J}\langle s_1 x^{J-\partial A_1},\ldots,s_m x^{J-\partial A_m},0,\ldots,0\rangle \\ &= x^{K-J}\langle \sum_{j=1}^{t} g_j c_{j1} x^{J-\partial A_1},\ldots,\sum_{j=1}^{t} g_j c_{jm} x^{J-\partial A_m},0,\ldots,0\rangle \\ &= x^{K-J}\sum_{j=1}^{t} g_j \langle c_{j1}x^{J-\partial A_1},\ldots,c_{jm}x^{J-\partial A_m},0,\ldots,0\rangle \\ &= x^{K-J}\sum_{j=1}^{t} g_j \vec{C}_j .\end{aligned}$$

As pointed out in Definition 2.4.2, $\vec{C}_j$ is a representation for $cons_j(\vec{B})$ in terms of $\vec{A}$, so we write $cons_j(\vec{B}) = \vec{C}_j \cdot \vec{A}$. But since $\vec{A}$ is a complete basis $cons_j(\vec{B})$ also has a simple representation $\vec{T}_j$ given by $remx(cons_j(\vec{B}),\vec{A}; P,\vec{T}_j)$. Since $\partial(cons_j(\vec{B})) < J$, $\partial\vec{T}_j < J$.

Now for $1 \le j \le t$, let $\vec{Y}_j' = \vec{C}_j - \vec{T}_j$. Notice that since $cons_j(\vec{B}) \in cons(\vec{A})$, $\vec{Y}_j' \in \vec{Y}$ when Algorithm 2.7.1 terminates.

Now $\vec{Y}_j' \cdot \vec{A} = (\vec{C}_j - \vec{T}_j)\cdot\vec{A} = \vec{C}_j\cdot\vec{A} - \vec{T}_j\cdot\vec{A} = cons_j(\vec{B}) - cons_j(\vec{B}) = 0$. Therefore, if we set $\vec{U} = \vec{S} - x^{K-J}\sum_{j=1}^{t} g_j\vec{Y}_j'$, then

$\vec{U}\cdot\vec{A} = \vec{S}\cdot\vec{A} - x^{K-J}\sum_{j=1}^{t} g_j\vec{Y}_j'\cdot\vec{A} = 0$. Hence $\vec{U} \in V(\vec{A})$.

Let us assume for the moment that $\partial\vec{U} < K$. If this is true, then we can apply the induction hypothesis, namely that $\vec{Y}$ generates $\vec{U}$, and

write $\vec{U} = \sum_{k=1}^{u} P_k \vec{Y}_k$ where for $1 \leq k \leq u$, $P_k \in R[x]$. But then

$\vec{S} = \vec{U} + x^{K-J} \sum_{j=1}^{t} g_j \vec{Y}_j' = \sum_{k=1}^{u} P_k \vec{Y}_k + \sum_{j=1}^{t} g_j x^{K-J} \vec{Y}_j'$, and since

$\vec{Y}_j' \in \vec{Y}$ for $1 \leq j \leq t$, $\vec{S}$ is generated by $\vec{Y}$ and the lemma is proved.

We proceed to prove that $\partial\vec{U} < K$. For $1 \leq j \leq t$ let

$\vec{T}_j = \langle T_{j1}, \ldots, T_{jn} \rangle \in (R[x])^n$. Now

$$\vec{U} = \vec{S} - x^{K-J} \sum_{j=1}^{t} g_j \vec{Y}_j'$$

$$= \vec{S}' + \vec{S}'' - x^{K-J} \sum_{j=1}^{t} g_j (\vec{C}_j - \vec{T}_j)$$

$$= (\vec{S}' - x^{K-J} \sum_{j=1}^{t} g_j \vec{C}_j) + \vec{S}'' + x^{K-J} \sum_{j=1}^{t} g_j \vec{T}_j$$

$$= \vec{S}'' + \sum_{j=1}^{t} g_j x^{K-J} \vec{T}_j.$$

Thus for $1 \leq i \leq m$

$$U_i = rd(S_i) + \sum_{j=1}^{t} g_j x^{K-J} T_{ji}$$

and for $m < i \leq n$

$$U_i = S_i + \sum_{j=1}^{t} g_j x^{K-J} T_{ji} .$$

The conditions on J, K, S_i, g_j, and T_{ji} are the same as at the end of the proof of Theorem 2.4. Hence $\partial\vec{U} < K = \partial\vec{S}$ and our claim

is established.□

If $\vec{A}$ is any basis for ideal $\overline{A}$ and if $\vec{B}$ is any complete basis for $\overline{A}$, let $\vec{C}$ be the concatenation of bases $\vec{A}$ and $\vec{B}$. Algorithm 2.7.1 can then be applied to find a sequence of generators for $V(\vec{C})$. But $V(\vec{A})$ is isomorphic to a sub-module of $V(\vec{C})$, hence the generators of $V(\vec{C})$ must also generate this sub-module.

This concept is made precise in the following definition of β^*.

Algorithm 2.7.2

$$\vec{Z} \leftarrow \text{beta}(\vec{A})$$

[β function for the ring R[x].

Input: $\vec{A}=\langle A_1,\ldots,A_n\rangle \in (R[x])^n$, a basis for ideal $\overline{A}$.

Output: $\vec{Z}=\langle \vec{Z}_1,\ldots,\vec{Z}_t\rangle$ where for $1\leq j\leq t$ $\vec{Z}_j \in (R[x])^n$, and $\vec{Z}$ is a finite sequence of generators for the R[x]-module $V(\vec{A})=\{\vec{S} \in (R[x])^n : \vec{S}\cdot\vec{A}=0\}$.]

(1) [Construct a complete basis $\vec{B}$ for $\overline{A}$.] $\vec{B} \leftarrow \text{cbasis}(\vec{A})$.

(2) [Assuming $\vec{B}=\langle B_1,\ldots,B_m\rangle$, form a new complete basis $\vec{C}$.]
$\vec{C} \leftarrow \langle A_1,\ldots,A_n,B_1,\ldots,B_m\rangle$.

(3) [Determine a sequence $\vec{Y}$ of generators for $V(\vec{C})$.] $\vec{Y} \leftarrow \text{modgen}(\vec{C})$.

(4) [Assuming $\vec{Y}=\langle \vec{Y}_1,\ldots,\vec{Y}_t\rangle$ and $\vec{Y}_j=\langle Y_{j1},\ldots,Y_{jn},\ldots,Y_{j,n+m}\rangle$ get $\vec{Z}$.]
for j=1,...,t do $\vec{Z}_j \leftarrow \langle Y_{j1},\ldots,Y_{jn}\rangle$; return □

Lemma 2.7.2 Algorithm 2.7.2 always terminates and when it does, $\vec{Z}$ is a finite sequence of generators for the R[x]-module $V(\vec{A})$.

Proof Termination is obvious.

Let $\vec{A} = \langle A_1,\ldots,A_n\rangle$ and $\vec{B} = \langle B_1,\ldots,B_m\rangle$ so that $\vec{C} = \langle A_1,\ldots,A_n,B_1,\ldots,B_m\rangle$. Let $\vec{Y} = \langle Y_1,\ldots,Y_t\rangle$ where $\vec{Y}_j = \langle Y_{j1},\ldots,Y_{jn},\ldots,Y_{j,n+m}\rangle$ for $1 \leq j \leq t$ be a finite sequence of generators for $V(\vec{C})$. If $\vec{S} = \langle S_1,\ldots,S_n\rangle \in V(\vec{A})$ then $\vec{S}' = \langle S_1,\ldots,S_n,0,\ldots,0\rangle \in V(\vec{C})$, where we have appended m zeros to $\vec{S}$. Hence there exist $P_j \in R[x]$, $1 \leq j \leq t$, such that $\vec{S}' = \sum_{j=1}^{t} P_j\vec{Y}_j$. Thus $S_i = \sum_{j=1}^{t} P_jY_{ji}$ for $1 \leq i \leq n$, and $\vec{S} = \sum_{j=1}^{t} P_j\langle Y_{j1},\ldots,Y_{jn}\rangle$ $= \sum_{j=1}^{t} P_j\vec{Z}_j$. Hence $\vec{Z}$ generates $V(\vec{A})$. □

We summarize our main result in the following theorem.

Theorem 2.7 If R is a simplification ring, then $R[x] = R[x_1,\ldots,x_r]$ is also.

Proof Ring R is a Noetherian ring with identity. Hence, by the Hilbert Basis Theorem, R[x] is also. Let $\leq$ represent the partial order in R[x] given by Definition 2.2.3. It is a combination of the partial order in R and the total order of all monomials given by any acceptable monomial ordering.

Let $\vec{A}$ be a basis for ideal $\bar{A}$ in R[x]. Let θ^* and β^* be defined by Algorithms 2.6.3 and 2.7.2, respectively. By Lemma 2.6.3 we have that $\rho^*(P,\vec{A}) = P - \vec{A}\cdot\theta^*(P,\vec{A})$ is a simplifying ample function for $(R[x], \equiv \bmod \bar{A}, \leq)$ and, in addition, $\theta^*(0,\vec{A}) = \langle 0,\ldots,0\rangle$. By Lemma 2.7.2, $\beta^*(\vec{A})$ determines a finite sequence of generators for the

R[x]-module $V(\vec{A}) = \{\vec{S} \in (R[x])^* : \vec{S} \cdot \vec{A} = 0\}$. Hence R[x], $\leq$, θ^*, and β^* satisfy the requirements of Definition 2.6.1, and thus R[x] is a simplification ring.□

2.8 Special Case Improvements

In this section we present several lemmas which will be useful in implementing the algorithms described in Sections 2.3 and 2.5.

Our first result deals with the possibility of replacing a given basis element by its remainder (or semi-remainder, see the end of Section 2.5 and Section 3.4) with respect to the rest of the basis. The lemma also shows that, if the initial basis is complete, so is the resulting basis.

Lemma 2.8.1 Let $\vec{A} = \langle A_1,\ldots,A_n\rangle$, $n \geq 2$, be a (complete) basis for ideal $\overline{A}$ in $R[x]$. Let $\vec{B} = \langle A_2,\ldots,A_n\rangle$ and $A_1' = \mathrm{rem}(A_1,\vec{B})$. Then $\vec{A}' = \langle A_1',A_2,\ldots,A_n\rangle$ is a (complete) basis for $\overline{A}$.

Proof Let $P \in \overline{A}$ and let $\vec{S} = \langle S_1,\ldots,S_n\rangle$ be a representation for P with respect to $\vec{A}$. By Lemma 2.3.5(d) $A_1 - A_1'$ has a simple representation, say $\vec{T} = \langle T_2,\ldots,T_n\rangle$, in terms of $\vec{B}$.

Now $P = \sum_{i=1}^{n} S_iA_i = S_1A_1 + \sum_{i=2}^{n} S_iA_i = S_1(A_1' + \sum_{i=2}^{n} T_iA_i)$ $+ \sum_{i=2}^{n} S_iA_i = S_1A_1' + \sum_{i=2}^{n} (S_1T_i + S_i)A_i$. Let $\vec{U} = \langle U_1,\ldots,U_n\rangle$ where $U_1 = S_1$ and for $2 \leq i \leq n$ $U_i = S_1T_i + S_i$. Then $\vec{U}$ is a representation for P in terms of $\vec{A}'$, and since $A_1' \in \overline{A}$, $\vec{A}'$ is a basis for $\overline{A}$.

Now assume $\vec{A}$ is a complete basis for $\overline{A}$ and $\vec{S}$ is a simple representation for P with respect to $\vec{A}$. We wish to show that $\vec{U}$ is a simple representation for P with respect to $\vec{A}'$.

By Lemma 2.3.5, $\partial A_1' \leq \partial A_1$. If $A_1' \neq 0$ and $S_1 \neq 0$ then $\partial S_1 + \partial A_1' \leq \partial S_1 + \partial A_1 \leq \partial P$. For $2 \leq i \leq n$ it is sufficient to show that if

$A_i \neq 0$ then either $U_i = 0$ or $\partial U_i + \partial A_i \leq \partial P$. Assume $U_i \neq 0$. Then either $\partial U_i \leq \partial S_1 + \partial T_i$ or $\partial U_i \leq \partial S_i$. Since $\vec{T}$ is a simple representation for $A_1 - A_1'$ with respect to $\vec{B}$, in the first case $\partial T_i + \partial A_i \leq \partial(A_1 - A_1') \leq \partial A_1$. Thus $\partial U_i + \partial A_i \leq (\partial S_1 + \partial T_i) + \partial A_i \leq \partial S_1 + \partial A_1 \leq \partial P$ since $\vec{S}$ is a simple representation for P. In the second case $\partial U_i + \partial A_i \leq \partial S_i + \partial A_i \leq \partial P$ for the same reason.

Therefore $\partial \vec{U} \leq \partial P$, and hence $\partial \vec{U} = \partial P$. Thus $\vec{A}'$ is a complete basis for $\bar{A}$. □

Note that this lemma also holds if the hypothesis $A_1' = \text{rem}(A_1, \vec{B})$ is replaced by $A_1' = \text{srem}(A_1, \vec{B})$. That is, A_1 may be replaced by the semi-remainder of A_1 with respect to the rest of the basis without changing the ideal generated or the completeness of the basis.

Our next lemma deals with the case where R (and hence R[x]) is a g.c.d. domain. It shows that in constructing complete bases we need only consider bases whose elements have a greatest common divisor of one.

Lemma 2.8.2 Let $\vec{A} = \langle A_1, \ldots, A_n \rangle$, $n \geq 2$, be a basis for ideal $\bar{A}$ in R[x], a g.c.d. domain. Let $A = \gcd(A_1, \ldots, A_n)$ and, for $1 \leq i \leq n$, let $B_i = A_i/A$. Let $\vec{B} = \langle B_1, \ldots, B_n \rangle$ generate ideal $\bar{B}$ in R[x]. If $\vec{B}' = \langle B_1', \ldots, B_s' \rangle$ is a complete basis for $\bar{B}$ then $\vec{A}' = \langle AB_1', \ldots, AB_s' \rangle$ is a complete basis for $\bar{A}$.

Proof Clearly $\vec{A}'$ is a basis for $\bar{A}$. We must show that if $P \in \bar{A}$ then P has a simple representation with respect to $\vec{A}'$.

Since $P \in \overline{A}$, $P' = P/A \in \overline{B}$. Since $\vec{B}'$ is complete, let $\vec{S} = \langle S_1,\ldots,S_s\rangle$ be a simple representation of P' with respect to $\vec{B}'$. Thus $P/A = P' = \vec{S}\cdot\vec{B}'$, and hence $P = \vec{S}\cdot(A\vec{B}') = \vec{S}\cdot\vec{A}' = \sum_{i=1}^{s} S_i(AB_i')$. If $S_i = 0$ or $B_i' = 0$ then $S_i = 0$ or $AB_i' = 0$. If $S_i \neq 0$ and $AB_i' \neq 0$, then $\partial S_i + \partial(AB_i') \leq \partial S_i + \partial A + \partial B_i' \leq \partial P' + \partial A$ since $\vec{S}$ is a simple representation for P'. But since $P = AP'$, $\partial P = \partial A + \partial P'$, hence $\partial S_i + \partial(AB_i') \leq \partial P' + \partial A = \partial P$. Therefore, $\vec{S}$ is a simple representation for P with respect to $\vec{A}'$.□

Our final result gives a condition under which the consensus of two polynomials is simply constructible from the given pair of polynomials. During the execution of the cbasis algorithm we need not construct the consensus of such polynomials.

Lemma 2.8.3 Let $\vec{A} = \langle A_1,A_2\rangle$, where A_1 and A_2 are non-zero polynomials in $R[x]$. Let $a_1 = lc(A_1)$, $a_2 = lc(A_2)$ and $\beta(\langle a_1,a_2\rangle) = \langle\vec{c}_1,\ldots,\vec{c}_t\rangle$. If a_1 and a_2 are units in R, and if $lcm(\partial A_1,\partial A_2) = \partial A_1 + \partial A_2$, then, for $1 \leq j \leq t$, $cons_j(\vec{A})$ is simply constructible from $\vec{A}$.

Proof Let $\vec{c}_j = \langle c_1,c_2\rangle \in R^2$, and let $J = lcm(\partial A_1,\partial A_2)$. Then

$$cons_j(\vec{A}) = c_1x^{J-\partial A_1}A_1 + c_2x^{J-\partial A_2}A_2$$

$$= c_1x^{\partial A_2}A_1 + c_2x^{\partial A_1}A_2$$

Now $\partial(cons_j(\vec{A})) < J$, hence $c_1a_1 + c_2a_2 = 0$, or $c_1a_1 = -c_2a_2$. Since a_1 and a_2 are units in R, we can divide by them. Hence $c_1/a_2 = -c_2/a_1$.

Let $d = c_1/a_2$, then $c_1 = da_2$, and $c_2 = -da_1$. Hence $cons_j(\vec{A}) =$ $da_2 x^{\partial A_2} A_1 - da_1 x^{\partial A_1} A_2 = d\ lt(A_2)A_1 - d\ lt(A_1)A_2 =$ $d(A_2 - rd(A_2))A_1 - d(A_1 - rd(A_1))A_2 = -d\ rd(A_2)A_1 + d\ rd(A_1)A_2 = \vec{S}\cdot\vec{A}$ where $\vec{S} = \langle S_1, S_2\rangle$ and $S_1 = -d\ rd(A_2)$, $S_2 = d\ rd(A_1)$.

If $S_1 = 0$ then $\langle 0, S_2\rangle$ is a simple representation for $cons_j(\vec{A})$, and if $S_2 = 0$ then $\langle S_1, 0\rangle$ is a simple representation for $cons_j(\vec{A})$. Assume now $S_1 \neq 0$, $S_2 \neq 0$ and let $lt(S_1) = s_1 x^{\partial S_1}$, $lt(S_2) = s_2 x^{\partial S_2}$. If $\partial\vec{S} > \partial(cons_j(\vec{A}))$ then $s_1 x^{\partial S_1}\ lt(A_1) + s_2 x^{\partial S_2} lt(A_2) = 0$, hence $\partial S_1 + \partial A_1 = \partial S_2 + \partial A_2$. Thus $\partial A_2 | (\partial S_2 + \partial A_2) = \partial S_1 + \partial A_1$, and $lcm(\partial A_1, \partial A_2) \leq \partial S_1 + \partial A_1 = \partial(rd(A_2)) + \partial A_1 < \partial A_2 + \partial A_1$. But we assumed $lcm(\partial A_1, \partial A_2) = \partial A_1 + \partial A_2$. This contradiction implies $\partial\vec{S} \leq \partial(cons_j(\vec{A}))$ and hence $\vec{S}$ is a simple representation for $cons_j(\vec{A})$ with respect to $\vec{A}$. □

CHAPTER THREE:

Remainder and Complete Basis Algorithms in F[x]

3.1 Introduction

In this chapter we describe algorithms for computing simplifying ample functions in the polynomial ring $F[x_1,\ldots,x_r]$, where F is a field. We again abbreviate $F[x_1,\ldots,x_r]$ by F[x]. Throughout this chapter we assume F is an effectively given field, i.e. that algorithms exist for performing arithmetic operations on field elements. Since F may be any effectively given field, many of the algorithms presented in this chapter will still be abstract. These algorithms will be abstract, however, only in regard to arithmetic operations in F. We will specify all other parts of these algorithms precisely.

In Section 3.2 we discuss list representations for exponent vectors, polynomials, and ideal bases. We also give the specifications for the algorithms which perform simple operations on these structures. In Section 3.3 we investigate the special properties of F when it is regarded as a simplification ring. The remainder algorithm for F[x] is given in Section 3.4. Problems dealing with the uniqueness of complete bases in F[x] are handled in Section 3.5, culminating in Section 3.6 with an algorithm for computing what we will call a complete "kernel" basis in F[x].

The algorithms in this chapter will be presented using the Aldes language for algorithm description. Lower case algorithm names

are used for algorithms which refer to the abstract arithmetic properties of F. Upper case letters are used for fully specified algorithms given here, or for algorithms in the SAC-2 computer algebra system.

The algorithms discussed in this and the following chapters have been implemented in the SAC-2 system for the cases F = Q, the field of rational numbers, and for F = GF(p), a finite field with p elements, p a prime β-integer. The table given in Appendix A relates the algorithm names used in this thesis to the actual algorithm names in the SAC-2 system. The algorithms themselves are given in Appendix B.

3.2 Exponent Vectors, Polynomials, and Ideal Bases

In this section we describe the basic algorithms and representations needed to support the implementation of the remainder and complete basis algorithms in the following sections.

We first define a representation for exponent vectors in terms of SAC-2 system list structures. The dense exponent vector canonical form, I^*, of exponent vector $I = \langle e_1,\ldots,e_r\rangle \in E$ is defined as follows. If $e_i = 0$ for $1 \leq i \leq r$ then $I^* = ()$, the empty list. If at least one e_i is non-zero, then $I^* = (e_r,\ldots,e_1)$. Note that when the notation $F[x_1,\ldots,x_r]$ is used, x_r is often interpreted as the main variable. This representation for exponent vectors has the advantage that the exponent of the main variable of a monomial is immediately available. We assume that the elements of an exponent vector are β-integers (see Section 1.6). This allows the individual exponents to be represented as atoms.

Two exponent vectors are compatible if they represent monomials in the same number of variables. The exponent vector represented by the empty list is defined to be compatible with any other exponent vector. Any exponent vector is compatible with $\partial 0$, but notice that $\partial 0$ does not have an explicit representation.

We assume the existence of the following algorithms for operations on exponent vectors. In the following I, J, and K are compatible exponent vectors.

Algorithm	Output
K←EVDIF(I,J)	if J|I then K=I-J else K=-1
K←EVSUM(I,J)	K=I+J
K←EVGCD(I,J)	K=gcd(I,J)
K←EVLCM(I,J)	K=lcm(I,J)

For the studies in this thesis we have selected two of the many possible acceptable monomial orderings. Let $I = \langle e_1,\ldots,e_r\rangle$ and $J = \langle f_1,\ldots,f_r\rangle$ be two compatible exponent vectors. We say that $I < J$ with respect to lexicographic ordering if there exists a j, $1 \leq j \leq r$, such that $e_i = f_i$ for $j < i$ and $e_j < f_j$.

By the total degree of an exponent vector we mean the sum of the individual elements. Let $e = \sum_{i=1}^{r} e_i$ and $f = \sum_{i=1}^{r} f_i$ be the total degrees of I and J respectively. Then $I < J$ with respect to total degree ordering if $e < f$ or $e = f$ and $I < J$ with respect to lexicographic ordering.

In both cases, $I = J$ iff $e_i = f_i$, $1 \leq i \leq r$ and we write $I \leq J$ iff $I < J$ or $I = J$.

We assume the existence of the following algorithm which uses the global variable EVORD to select the ordering desired. EVORD = 1 gives lexicographic ordering and EVORD = 2 gives total degree ordering.

Algorithm	Output
b←EVCOMP(I,J)	$b = \begin{cases} +1 & \text{if } I>J \\ 0 & \text{if } I=J \\ -1 & \text{if } I<J \end{cases}$

The computing times of EVDIF, EVSUM, EVGCD, EVLCM, and EVCOMP are dominated by r.

We assume that elements of the field F are represented in some canonical form. The following abstract algorithms perform arithmetic in F (a,b, and c are elements of F).

Algorithm	Output
c←fdif(a,b)	c=a-b
b←fneg(a)	b=-a
c←fprod(a,b)	c=ab
c←fquo(a,b)	c=a/b , b≠0
c←fsum(a,b)	c=a+b

We can now describe a representation for a polynomial $A = \sum_{I \in E} a_I x^I \in F[x]$. The distributive, dense exponent vector, canonical form A* of A is defined as follows. If A = 0 then A* = 0. If $A \neq 0$ then let $E' = \{I \in E: a_I \neq 0\}$ and let $E' = \{I_1,\ldots,I_m\}$ where $I_1 > I_2 > \ldots > I_m$ and > is some acceptable monomial ordering. Let I_j^* be the dense exponent vector canonical form for I_j, $1 \leq j \leq m$, and let a_j^* be a canonical representation for the element $a_{I_j} \in F$. Then $A^* = (I_1^*,a_1^*,\ldots,I_m^*,a_m^*)$.

In the remainder of this thesis the phrase "distributive polynomial" will refer to a polynomial represented in the distributive, dense exponent vector, canonical form. (In the SAC-2 system

the word "polynomial" is reserved for polynomials represented in recursive canonical form, see Section 4.2).

We assume the existence of the following algorithms which extract various parts of a non-zero distributive polynomial, A.

Algorithm	Output
a←DPLC(A)	a=lc(A)
DPLEC(A;I,a)	I=∂A, a=lc(A)
DPLECR(A;I,a,B)	I=∂A, a=lc(A), B=rd(A)
I←DPLEV(A)	I=∂A
B←DPRD(A)	B=rd(A)

Since we are assuming the use of distributive canonical form for A, the computing times of DPLC, DPLEC, DPLECR, DPLEV, and DPRD are all codominant with one.

We also assume the existence of the following algorithms to perform arithmetic on distributive polynomials in R[x] where R is an as yet unspecified coefficient domain. Later in this chapter we will use these abstract algorithms with the assumption that R is a field F.

We say that two polynomials are compatible if they have the same number of variables. A non-zero constant polynomial, i.e. a polynomial whose leading exponent vector is represented by the empty list, is compatible with any polynomial. The zero polynomial is also compatible with any polynomial.

In these algorithms we assume A, B, and C are compatible distributive polynomials. We also assume that J is an exponent vector compatible with ∂A, and that b is an element of the coefficient ring.

Algorithm	Output
C←dpdif(A,B)	C=A-B
B←dpneg(A)	B=-A
C←dpprod(A,B)	C=AB
C←dpsum(A,B)	C=A+B
B←dptermpr(J,b,A)	$B=bx^JA$

A non-zero $A \in R[x]$ is said to be monic if lc(A) is the identity in R. If R is a field then the monic associate B of a non-zero polynomial A is given by B = A/lc(A). We let 0 be the monic associate of 0. In the case R is a field we assume the existence of the following abstract algorithm (A is a distributive polynomial).

Algorithm	Output
B←dpmon(A)	if A=0 then B=0 else B=A/lc(A)

Since we have not specified the coefficient domain, it is not possible to give bounds on the computing times for these algorithms. However, it is possible to bound the number of exponent vector operations required. Let $A = \sum_{I \in E} a_I x^I$ and $B = \sum_{I \in E} b_I x^I$ and let m_A = cardinality of $\{I \in E: a_I \neq 0\}$ and m_B = cardinality of $\{I \in E: b_I \neq 0\}$. Then the number of exponent vector operations

required by dpneg, dptermpr, and dpmon is just m_A: The maximum number of exponent vector operations required by dpsum and dpdif is dominated by $m_A + m_B$, which is codominant with $\max(m_A, m_B)$. The number of exponent vector operations required by dpprod is dominated by $m_A^2 m_B$.

Let $\vec{A} = \langle A_1, \ldots, A_n \rangle$ be a basis for ideal $\overline{A}$ in $F[x]$, and let A_i^* be the distributive, dense exponent vector, canonical form for $A_i \in F[x]$, $1 \leq i \leq n$. Then the representation $\vec{A}^*$ of the basis $\vec{A}$ is just the list $\vec{A}^* = (A_1^*, \ldots, A_n^*)$. We assume that a basis is non-empty.

In this section we have made a distinction between an algebraic object and the list structure representing that object. For the remainder of this thesis (except where otherwise noted) we will ignore this distinction and use the same notation for both concepts.

3.3 F as a Simplification Ring

To make the field F into a simplification ring it is necessary to specify a simplifying ample function ρ and the functions θ and β such that the criteria described in Definition 2.2.1 are satisfied. A brief discussion of this problem was given in Section 2.2. A more extensive discussion is given here.

Recall that since F is a field, the only possible ideals are $\{0\}$ and F. Let us consider the residue classes defined by these ideals. If $\overline{a}$ is an ideal in F and $\overline{a} = \{0\}$, then, for $a,b \in F$, $a \equiv b \bmod \overline{a}$ iff $a-b \in \overline{a}$, i.e. iff $a = b$. Thus every element in F is in a residue class by itself, and ρ, considered as an ample function, must be such that $\rho(a,\{0\}) = a$ for all $a \in F$. If $\overline{a} = F$ then $a \equiv b \bmod \overline{a}$ iff $a-b \in F$, i.e. there is only one residue class, namely F. Since $\rho(0,\overline{a}) = 0$ for any ideal $\overline{a}$, we must have $\rho(a,F) = 0$ for any $a \in F$.

Now ρ must be a simplifying ample function for the triple $(F, \equiv \bmod \overline{a}, \leq)$ in the two cases $\overline{a} = \{0\}$ and $\overline{a} = F$. This implies that any partial order defined on F must be consistent with the partial order $\leq$ given by $a \leq b$ iff $a = 0$ or $a = b$. We will assume that our partial order on F is determined by just this relation.

The simplifying ample function, ρ, is uniquely determined by the ideal of F under consideration. Much more freedom is allowed in the specification of the functions θ and β.

Let $\vec{a} = \langle a_1,\ldots,a_n\rangle \in F^n$ be a basis for ideal $\overline{a}$ in F. If $\overline{a} = \{0\}$ then $\vec{a} = \langle 0,\ldots,0\rangle$. Since $a = \rho(a,\vec{a}) = a-\vec{a}\cdot\theta(a,\vec{a})$, we have

$\vec{a}\cdot\theta(a,\vec{a}) = 0$. We straightforwardly set $\theta(a,\vec{a}) = \langle 0,\ldots,0\rangle$ in this case. If $\bar{a} = F$ then $0 = \rho(a,\vec{a}) = a - \vec{a}\cdot\theta(a,\vec{a})$. Hence $\theta(a,\vec{a})$ $= \langle s_1,\ldots,s_n\rangle \in F^n$ such that $a = \sum_{i=1}^{n} a_i s_i$. Of the many ways to solve this equation over a field, the simplest is to find any integer j such that $a_j \neq 0$, set $s_j = a/a_j$ and set $s_i = 0$ for $i \neq j$.

Let $\vec{b} = \langle b_1,\ldots,b_n\rangle \in F^n$. Recall that $\beta(\vec{b})$ is defined to be a function whose value is a sequence of generators for the R-module $V(\vec{b}) = \{\vec{s} \in F^n: \vec{s}\cdot\vec{b} = 0\}$, and note that, since F is a field, $V(\vec{b})$ is actually a vector space. In Section 2.2 we gave an example of such a $\beta(\vec{b})$ under the assumption $\vec{b} \neq \langle 0,\ldots,0\rangle$. It is our task here to remove this assumption from our specification of $\beta(\vec{b})$.

In the following definition we assume, without loss of generality, that $b_i \neq 0$ for $1 \leq i \leq m$ and $b_i = 0$ for $m < i \leq n$. If $\vec{b} = \langle 0,\ldots,0\rangle$ then $m = 0$. For $1 \leq j \leq n$ let $\vec{u}_j$ be the j-th unit vector of length n, i.e. $\vec{u}_j = \langle 0,\ldots,1,\ldots,0\rangle$ where the j-th element is 1.

Definition 3.3.1 Let $\vec{b} = \langle b_1,\ldots,b_n\rangle \in F^n$. Then $\beta(\vec{b}) = \langle \vec{c}_1,\ldots,\vec{c}_t\rangle$ is defined as follows.

(a) If $\vec{b} = \langle 0,\ldots,0\rangle$, i.e. $m = 0$, then $t = n$ and $\vec{c}_j = \vec{u}_j$, $1 \leq j \leq n$;

(b) If $m = 1$ and $n = 1$ then $t = 1$ and $\vec{c}_1 = \langle 0\rangle$; if $m = 1$ and $n > 1$ then $t = n-1$ and $\vec{c}_j = \vec{u}_{j+1}$ for $1 \leq j < n$;

(c) If $m > 1$ then $t = n-1$, $\vec{c}_j = b_{j+1}\vec{u}_1 - b_1\vec{u}_{j+1}$ for $1 \leq j < m$, and $\vec{c}_j = \vec{u}_{j+1}$ for $m \leq j < n$.

Notice that if $\vec{b} = \langle 0,\ldots,0\rangle$ then $V(\vec{b}) = F^n$ which is generated by $\langle \vec{u}_1,\ldots,\vec{u}_n\rangle$. If $\vec{b} \neq \langle 0,\ldots,0\rangle$ then the vector space dimension of $V(\vec{b})$ is n-1, and $\beta(\vec{b})$ contains exactly n-1 linearly independent elements of $V(\vec{b})$. Hence in both cases $\beta(\vec{b})$ generates $V(\vec{b})$.

We note in passing that we can regard the basis $\langle 1\rangle$ as a canonical basis for ideal $\overline{a} = F$, and the basis $\langle 0\rangle$ as a canonical basis for ideal $\overline{a} = \{0\}$.

Note also that the remark following Definition 2.3.1 applies to representations in F[x] since a field is also an integral domain. That is, if $\vec{A} = \langle A_1,\ldots,A_n\rangle \in (F[x])^n$ and $\vec{S} = \langle S_1,\ldots,S_n\rangle$ is a representation for $A \in F[x]$ in terms of $\vec{A}$, then $\partial\vec{S} = \max_{1\leq i\leq n} \partial(S_iA_i)$.

3.4 Remainder Algorithm for F[x]: Remainder Selection Rules

Let $\overline{A}$ be the ideal generated by the basis $\vec{A} = \langle A_1,\ldots,A_n\rangle \in (F[x])^n$ where F is an effectively given field. In Algorithm 2.3 we defined the remainder, $\mathrm{rem}(P,\vec{A})$, in R[x] where R is a simplification ring. We present in this section a version of Algorithm 2.3 for the polynomial ring F[x].

In this section we assume $\overline{A} \neq \{0\}$ and that every element of basis $\vec{A} \in (F[x])^n$ is non-zero.

Let $Q_i = \sum_{I\in E} q_I x^I \in F[x]$ be the value of the variable Q during the i-th execution of step 3 of Algorithm 2.3, and let $D_i = D(Q_i)$ $= \{I \in E: q_I \neq \rho(q_I,\vec{a}_I)\}$ be the value of D during the i-th execution of step 4. Recall that $\vec{a}_I$ is the degree I leading coefficient sub-basis of $\vec{A}$. The simplifying ample function ρ is the one discussed in the previous section. If $\overline{a}_I = \{0\}$ then for all $q \in F$, $\rho(q,\vec{a}_I) = q$. Hence $J \in D_i$ iff $q_J \neq 0$ and there exists a j, $1 \leq j \leq n$, such that $\partial A_j | J$.

To specify the remainder choice function in step 4, we need to give criteria for selecting $J \in D$. Since Q_i is represented as a list in descending exponent vector order, it is reasonable to select the largest possible exponent vector from D_i. Note also that since F is a field, $\rho(q_I,\vec{a}_I) = 0$ if $\overline{a}_I \neq \{0\}$. Hence, by Lemma 2.3.1, $\max D_i > \max D_{i+1}$, i.e. a given term of Q will only be reduced once. We make this version of the remainder choice function precise in the following selection rule.

Remainder Selection Rule 1 (RSR1) Given Q_i and D_i as above, at the i-th execution of step 4 of Algorithm 2.3, $J = \max D_i$.

Note that this is the same as requiring $J = \max \{I \in E: q_I \neq 0$ and $\exists\, j,\ 1 \leq j \leq n,\ \partial A_j | I\}$. This is the form in which RSR1 will be used in the remainder algorithm in this section.

Once we have selected $J \in D_i$, it remains to determine $\theta(q_J, \vec{a}_J)$ for use in steps 5 and 6 of Algorithm 2.3. Recall that $\vec{A}_J = \langle A_1', \ldots, A_n' \rangle$ is the degree J sub-basis of ideal basis $\vec{A}$, where $A_j' = A_j$ if $\partial A_j | J$ and $A_j' = 0$ otherwise, $1 \leq j \leq n$. Let k be such that $A_k' \neq 0$ and let $a_k = lc(A_k') = lc(A_k)$. In the previous section we have seen that $\theta(q_J, \vec{a}_J) = \langle 0, \ldots, q_J/a_k, \ldots, 0 \rangle$ is a straightforward realization of the θ function where the non-zero entry, q_J/a_k, occurs in the k-th position.

The problem now becomes one of deciding which of the possibly many non-zero elements of $\vec{A}_J$ to choose. Two immediate possibilities present themselves. We could choose k such that $A_k' \neq 0$ and $\partial A_k'$ is a maximum or $\partial A_k'$ is a minimum. Experimental comparisons of these two results will be made in Chapter 5. Here we argue that, if we select k such that $\partial A_k'$ is a minimum, then A_k' will most likely have fewer non-zero terms than if $\partial A_k'$ is a maximum. This would have the effect of reducing the number of exponent vector comparisons and field operations required in the sum computed in step 6 of Algorithm 2.3.

We make this discussion precise in the following selection rule.

Remainder Selection Rule 2 (RSR2) Given $J \in D_i$, let $I = \min\{\partial A_j: 1 \leq j \leq n \text{ and } \partial A_j | J\}$, and let $k = \min\{j: \partial A_j = I\}$. Then $\theta(q_J, \vec{a}_J) = \langle b_1, \ldots, b_n \rangle$ where $b_k = q_J/a_k$ and $b_j = 0$ for $j \neq k$.

Note that $\theta(q_J, \vec{a}_J)$ as presented here actually depends on $\vec{A}_J$ as well as $\vec{a}_J$ so it is not strictly a function of q_J and $\vec{a}_J$ by themselves. We have here actually two remainder selection rules in one. The first specifies an ordering of $\vec{A}_J$ and hence of $\vec{a}_J$ and the second actually determines $\theta(q_J, \vec{a}_J)$. Since these two sub-rules are so closely bound in the current situation, we have found it convenient to treat them as one.

We can summarize RSR1 and RSR2 by saying that we will always reduce the largest unreduced term of Q with respect to the element of basis $\vec{A}$ with the smallest suitable leading exponent vector.

Before giving the remainder algorithm for F[x], we give an implementation of RSR2 in the following algorithm. For efficiency, we assume the elements of the basis $\vec{A}$ are organized such that their leading exponent vectors are in non-decreasing order.

Algorithm 3.4.1

BESEL($J, \vec{A}; K, a, B$)

[Basis element selection.

Input: $J \in E$, $\vec{A} = \langle A_1, \ldots, A_n \rangle \in (F[x])^n$, $A_j \neq 0$, $1 \leq j \leq n$, and $\partial A_1 \leq \partial A_2 \leq \ldots \leq \partial A_n$.

Output: If, for $1 \leq j \leq n$, $\partial A_j \uparrow J$ then K=-1 and a and B are undefined. Otherwise let k be the minimum integer such that $\partial A_k | J$. Then $K=J-\partial A_k$, $a=lc(A_k)$, and $B=rd(A_k)$. This implements remainder selection rule 2.]

(1) [Initialize.] $\vec{A}' \leftarrow \vec{A}$.

(2) [Check each basis element.] repeat { ADV($\vec{A}'$;A,$\vec{A}'$); DPLECR(A;I,a,B); K←EVDIF(J,I); if K≠-1 then return; if EVCOMP(J,I)=-1 then return } until $\vec{A}'$=(); return □

We now present the remainder algorithm for F[x]. In the following P,Q, and the elements of $\vec{A}$ are assumed to be compatible distributive polynomials. These algorithms are still abstract because the field F has not been specified.

Algorithm 3.4.2

Q←dprem(P,$\vec{A}$)

[Distributive polynomial over a field, remainder.
Input: $P \in F[x]$, $\vec{A}=\langle A_1,\ldots,A_n\rangle \in (F[x])^n$, $A_i \neq 0$, $1 \leq i \leq n$, and $\partial A_1 \leq \partial A_2 \leq \ldots \leq \partial A_n$.
Output: $Q \in F[x]$, Q=rem(P,$\vec{A}$) using remainder selection rules 1 and 2.]

(1) [P=0.] if P=0 then { Q←P; return }.

(2) [Initialize.] Q'←P; Q←().

(3) [Attempt to reduce lt(Q').] repeat { DPLECR(Q';J,q,Q'); BESEL(J,$\vec{A}$;K,a,B); if K=-1 then Q←COMP2(q,J,Q) else { b←fneg(fquo(q,a)); Q'←dpsum(Q',dptermpr(K,b,B)) } } until Q'=0.

(4) [Finish.] if Q=() then Q←0 else Q←INV(Q); return □

We also present an algorithm to compute semi-remainders in F[x]. In computing a semi-remainder, we terminate as soon as $lt(Q) \in ker(\vec{A})$. Hence remainder selection rule 1 does not apply.

Algorithm 3.4.3

$$Q \leftarrow dpsrem(P,\vec{A})$$

[Distributive polynomial over a field, semi-remainder.
Input: $P \in F[x]$, $\vec{A}=<A_1,\ldots,A_n> \in (F[x])^n$, $A_i \neq 0$, $1 \leq i \leq n$, and $\partial A_1 \leq \partial A_2 \leq \ldots \leq \partial A_n$.
Output: $Q \in F[x]$, $Q=srem(P,\vec{A})$ using remainder selection rule 2.]

(1) [P=0.] Q←P; if P=0 then return.

(2) [Attempt to reduce lt(Q).] repeat { DPLECR(Q;J,q,Q');
BESEL(J,$\vec{A}$;K,a,B); if K=-1 then return; b←fneg(fquo(q,a));
Q←dpsum(Q',dptermpr(K,b,B)) } until Q=0; return □

The termination and correctness proofs of Algorithm 3.4.1 are obvious. The termination proofs for Algorithms 3.4.2 and 3.4.3 are related to the termination proof for Algorithm 2.3 given in Lemma 2.3.3. Let $Q_i' = \sum_{I \in E} q_I x^I$ be the value of variable Q' in Algorithm 3.4.2 at the beginning of the i-th execution of the repeat loop in step 3. Let $J = \partial Q_i'$ and $a_j = lc(A_j)$, $1 \leq j \leq n$. If there exists a j such that $\partial A_j | J$ then $Q_{i+1}' = Q_i' - (q_J/a_j)x^{J-\partial A_j} A_j$ and hence $\partial Q_{i+1}' < \partial Q_i'$. If such a j does not exist then $Q_{i+1}' = rd(Q_i')$ and

hence $\partial Q_{i+1}' < \partial Q_i'$. Since the degree of Q_i' is being reduced by each iteration of the repeat loop, the algorithm must eventually terminate. A similar, and even simpler, argument can be made for Algorithm 3.4.3 with Q taking the place of Q'.

The detailed discussion at the beginning of this section shows how Algorithm 3.4.2 is a specialization of Algorithm 2.3. Hence, when Algorithm 3.4.2 terminates, $Q = \text{rem}(P,\vec{A})$. To prove Algorithm 3.4.3 is correct, we need to show that at termination $Q \equiv P \bmod \overline{A}$ and $\text{lt}(Q) \in \ker(\vec{A})$. The first of these properties is obvious. When this algorithm terminates, either Q = 0 or there is no j such that $\partial A_j | \partial Q$. The latter case is equivalent to saying that $\overline{a}_{\partial Q} = \{0\}$ and hence $\text{lc}(Q) = \rho(\text{lc}(Q), \vec{a}_{\partial Q})$. Therefore, $\text{lt}(Q) \in \ker(\vec{A})$ and we are done.

Without specifying the coefficient domain, it is not possible to completely analyze the computing times of Algorithms 3.4.2 and 3.4.3. We can, however, derive a function which dominates the number of exponent vector operations required by Algorithm 3.4.2 (and hence also by Algorithm 3.4.3) when total degree ordering is being used.

Consider now exponent vectors representing monomials in r variables. The number of exponent vectors with total degree m is just the number, N(r,m), of distinct ways of choosing non-negative integers $e_1,\ldots,e_r$ (not necessarily all distinct) such that $\sum_{i=1}^{r} e_i = m$. By induction we can show that $N(r,m) = \binom{m+r-1}{r-1}$. If we let $\overline{N}(r,m)$ be the number of exponent vectors with total degree less than or equal to m,

then $\overline{N}(r,m) = \sum_{k=0}^{m} N(r,m) = \sum_{k=0}^{m} \binom{k+r-1}{r-1} = \binom{m+r}{r} = N(r+1,m)$ (see [KNU68], Chapter 1).

Let m_i be the total degree of exponent vector ∂A_i for $1 \leq i \leq n$ and let m' be the total degree of ∂P. If $m = \max\{m', m_1, \ldots, m_n\}$, then $\overline{N}(r,m)$ is a bound on both the number of terms in P and the number of terms in any basis element. During the i-th execution of the repeat loop in Algorithm 3.4.2, $\partial Q_i' \leq \partial P$. Thus the total degree of Q_i' is less than or equal to m, hence the number of terms in Q_i' is less than or equal to $\overline{N}(r,m)$.

Since $\partial Q_{i+1}' < \partial Q_i'$ the repeat loop can be repeated a maximum of $\overline{N}(r,m)$ times. During the i-th execution, the number of exponent vector operations required by dpsum and dptermpr is dominated also by $\overline{N}(r,m)$.

The number of exponent vector operations required by Algorithm 3.4.1 is clearly dominated by n. Combining these results we see that the number of exponent vector operations required by Algorithm 3.4.2 is dominated by $\overline{N}(r,m)(\overline{N}(r,m)+n)$.

The case of lexicographic ordering cannot be analyzed in the same way because the number of terms in a polynomial cannot be bounded by a function of its leading exponent vector. In Chapter 5 we present some computational experiments using both total degree and lexicographic ordering.

3.5 Canonical Complete Bases in F[x]

Let R be a simplification ring, let $\overline{A} \neq \{0\}$ be an ideal in R[x], and assume we have fixed a monomial ordering, $\leq$. At the end of Section 2.3 we remarked that there are many possible complete bases for $\overline{A}$. The complete basis computed by Algorithm 2.5 depends on the initial basis and on the remainder and consensus selection rules chosen.

In this section we restrict our attention again to the case where R is a field, F. We describe a special basis called a "monic complete kernel basis" which can be shown to be uniquely determined once an ideal in F[x] and the monomial ordering are chosen. Section 3.6 describes algorithms for computing such a basis. The importance of a complete kernel basis is that it is, in a sense, a minimal complete basis. (In [TRI78] a similar type of basis is described in case R is a ring very much like a simplification ring and it is possible to compute canonical bases for ideals in R.)

Let $\vec{A} = \langle A_1, \ldots, A_n \rangle$ be a basis for ideal $\overline{A}$ in F[x]. If either n=1 and $A_1 = 0$ or, for $1 \leq i \leq n$, $A_i \neq 0$ and A_i is a monic polynomial, then we say that $\vec{A}$ is a monic basis. Note that since F is a field, it is always possible to compute a monic basis for $\overline{A}$ from any other basis for $\overline{A}$.

We now describe two important kinds of bases. We give algorithms for computing these bases in Section 3.6. If $n \geq 2$ then, for

$1 \leq i \leq n$, let $\vec{A}_i = \langle A_1,\ldots,A_{i-1}, A_{i+1},\ldots,A_n\rangle$, i.e. $\vec{A}_i$ is just basis $\vec{A}$ with A_i removed.

Definition 3.5.1 $\vec{A} = \langle A_1,\ldots,A_n\rangle$ is said to be a semi-kernel basis for $\overline{A}$ if either $n = 1$, or $n \geq 2$ and, for $1 \leq i \leq n$, $A_i \neq 0$ and $A_i \in \text{sker}(\vec{A}_i)$.

An important property of semi-kernel bases is given in the following lemma.

Lemma 3.5.1 If $\vec{A} = \langle A_1,\ldots,A_n\rangle$, $n \geq 2$, is a semi-kernel basis then for $1 \leq i, j \leq n$, $i \neq j$, $\partial A_i \dagger \partial A_j$.

Proof Since $n \geq 2$, $A_i \neq 0$ for $1 \leq i \leq n$. Let $a = \text{lc}(A_i)$, $I = \partial A_i$, and let $\vec{a}_I$ be the leading coefficient sub-basis of $\vec{A}_i$. Recall $A_i \in \text{sker}(\vec{A}_i)$ just when $\text{lt}(A_i) \in \text{ker}(\vec{A}_i)$, i.e. $a = \rho(a,\vec{a}_I)$. In Section 3.3 we noted that $\rho(a,\vec{a}_I) \neq 0$ iff $\overline{a}_I = \{0\}$. But, going back to Definition 2.2.5, we see this implies that, for $1 \leq j \leq n$ and $j \neq i$, $\partial A_j \dagger I$, i.e. that $\partial A_j \dagger \partial A_i$.□

Hence, in a semi-kernel basis, the leading term of each basis element is irreducible with respect to the other basis elements, i.e. $\text{srem}(A_i,\vec{A}_i) = A_i$.

An even more restrictive class of bases is given next.

Definition 3.5.2 $\vec{A} = \langle A_1,\ldots,A_n\rangle$ is said to be a kernel basis for $\overline{A}$ if either $n = 1$ or $n \geq 2$ and, for $1 \leq i \leq n$, $A_i \neq 0$ and $A_i \in \text{ker}(\vec{A}_i)$.

Note that this means that $\text{rem}(A_i, \vec{A}_i) = A_i$, i.e. A_i cannot be reduced with respect to the rest of the basis.

We now describe the special properties of a complete semi-kernel basis. By the leading exponent vector set, $L(\vec{A})$, of basis $\vec{A} = \langle A_1, \ldots, A_n \rangle$ we mean the set $\{\partial A_1, \ldots, \partial A_n\}$.

Lemma 3.5.2 Let $\vec{A} = \langle A_1, \ldots, A_n \rangle$ be a complete semi-kernel basis for ideal $\overline{A}$ in $F[x]$, and let $\vec{B} = \langle B_1, \ldots, B_s \rangle$ be any other complete basis for $\overline{A}$. Then $L(\vec{A}) \subseteq L(\vec{B})$ and $n \leq s$.

Proof If $n = 1$ and $A_1 = 0$ then $\overline{A} = \{0\}$ and $\vec{B}$ must be of the form $\langle 0, \ldots, 0 \rangle$ so the lemma is true. If $n = 1$ and $A_1 \neq 0$ then clearly $n \leq s$. Since $\overline{A}$ is a principal ideal generated by A_1, $\partial A_1 | \partial B_i$ for $1 \leq i \leq s$. But since $\vec{B}$ is complete there must exist a j, $1 \leq j \leq s$ such that $\partial B_j | \partial A_1$. Thus for this j, $\partial B_j = \partial A_1$ and hence $L(\vec{A}) \subseteq L(\vec{B})$.

Now assume $n \geq 2$. For $1 \leq i \leq n$, $A_i \in \overline{A}$, and since $\vec{B}$ is a complete basis there exists a j, $1 \leq j \leq s$ such that $\partial B_j | \partial A_i$. But $B_j \in \overline{A}$ hence, since $\vec{A}$ is a complete basis, there exists a k, $1 \leq k \leq n$ such that $\partial A_k | \partial B_j$, and hence $\partial A_k | \partial A_i$. Therefore, since $\vec{A}$ is a semi-kernel basis, by Lemma 3.5.1, $\partial A_k | \partial A_i$ iff $i = k$. Hence $\partial B_j = \partial A_i$ and $L(\vec{A}) \subseteq L(\vec{B})$. □

We have an immediate corollary.

Corollary 3.5.1 If $\vec{A}$ and $\vec{B}$ are both complete semi-kernel bases in Lemma 3.5.2 then $L(\vec{A}) = L(\vec{B})$ and $n = s$.

Thus we see that the number of elements in a complete semi-kernel basis is determined only by the ideal and the monomial ordering.

A complete semi-kernel basis is minimal in the sense that the number of elements in it is less than or equal to the number of elements in any other complete basis for the same ideal.

Note that the number of elements in a complete semi-kernel basis depends on the monomial ordering because completeness is defined in terms of simple constructibility (Definition 2.3.2) which depends directly on the monomial ordering. As an example, let $\vec{A} = \langle y+x^2, x^3 \rangle$ and $\vec{B} = \langle y^2, xy, x^2+y \rangle$. One can verify that $\vec{A}$ and $\vec{B}$ generate the same ideal in $Q[x,y]$. In fact, $\vec{A}$ is a complete semi-kernel basis with respect to lexicographic ordering and $\vec{B}$ is a complete semi-kernel basis with respect to the total degree ordering. ($\vec{A}$ and $\vec{B}$ are actually monic complete kernel bases.)

With the monomial ordering fixed, we can now prove the following.

Theorem 3.5 If $\vec{A}$ and $\vec{B}$ are monic complete kernel bases for ideal $\overline{A}$ in $F[x]$, then $\vec{A}$ and $\vec{B}$ differ only in the order of their elements.

Proof Since a kernel basis is also a semi-kernel basis, then, by Corollary 3.5.1, we can assume $\vec{A} = \langle A_1, \ldots, A_n \rangle$ and $\vec{B} = \langle B_1, \ldots, B_n \rangle$ where $\partial A_i = \partial B_i$ for $1 \leq i \leq n$. If $n = 1$ then A_1 is a multiple of B_1, and B_1 is a multiple of A_1. Since both A_1 and B_1 are monic, we have $A_1 = B_1$ and we are done.

Now assume $n \geq 2$, and consider $C_i = A_i - B_i$ for $1 \leq i \leq n$. We wish to show $C_i \in \ker(\vec{A})$. First note that since $\vec{A}$ and $\vec{B}$ are monic

bases, $lt(A_i) = lt(B_i)$, hence $C_i = rd(A_i) - rd(B_i)$. Furthermore, with $\vec{A}_i$ given in Definition 3.5.1, $A_i \in \ker(\vec{A}_i)$, and since $\partial A_i > \partial(rd(A_i))$, $rd(A_i) \in \ker(\vec{A})$. Similarly, $rd(B_i) \in \ker(\vec{B})$. Since $\vec{A}$ and $\vec{B}$ are complete, we know $\ker(\vec{A}) = \ker(\vec{B})$ by Lemma 2.3.6 and Definition 2.2.6. Hence $rd(B_i) \in \ker(\vec{A})$.

Let $rd(A_i) = \sum_{I \in E} a_I x^I$ and $rd(B_i) = \sum_{I \in E} b_I x^I$. Thus $a_I = \rho(a_I, \vec{a}_I)$ and $b_I = \rho(b_I, \vec{a}_I)$. If $a_I \neq 0$ or $b_I \neq 0$ then, since F is a field, Section 3.3 shows $\overline{a}_I = \{0\}$. But then $a_I - b_I = \rho(a_I - b_I, \vec{a}_I)$, hence $C_i \in \ker(\vec{A})$.

But $C_i \in \overline{A}$, and $\vec{A}$ is complete so $rem(C_i, \vec{A}) = 0$. But if $C_i \in \ker(\vec{A})$ then $C_i = rem(C_i, \vec{A})$. Hence $C_i = 0$ and we have $rd(A_i) - rd(B_i) = 0$ or $A_i = B_i$.□

We conclude this section with a lemma which provides a foundation for an algorithm to compute a complete kernel basis from a complete semi-kernel basis.

Lemma 3.5.3 Let $\vec{A} = \langle A_1, \ldots, A_n \rangle$, $n \geq 2$, be a complete semi-kernel basis for ideal $\overline{A} \neq \{0\}$, arranged such that $\partial A_1 < \partial A_2 < \ldots < \partial A_n$. Let $A_1' = A_1$ and for $1 \leq i < n$ let $A_{i+1}' = dprem(A_{i+1}, \vec{A}_i)$ where $\vec{A}_i = \langle A_1', \ldots, A_i' \rangle$. Then $\vec{A}_n$ is a complete kernel basis for $\overline{A}$.

Proof Let $\vec{A}_0' = \vec{A}$ and let $\vec{A}_i' = \langle A_1', \ldots, A_i', A_{i+1}, \ldots, A_n \rangle$ for $1 \leq i \leq n$. Now $\vec{A}_0'$ is a complete basis for $\overline{A}$, so let us assume $\vec{A}_i'$ is a complete basis for $\overline{A}$. Since $\partial A_j > \partial A_{i+1}$ for $i + 1 < j \leq n$, Definition 2.2.2(a) implies that A_j cannot be used to remainder A_{i+1}.

Hence $A_{i+1}' = \text{dprem}(A_{i+1}, \vec{A}_i) = \text{dprem}(A_{i+1}, \langle A_1', \ldots, A_i', A_{i+2}, \ldots, A_n \rangle)$. Therefore, by Lemma 2.8.1, $\vec{A}_{i+1}'$ is also a complete basis for $\bar{A}$.

Furthermore, since $\partial A_i' = \partial A_i \neq \partial 0$, $A_i' = \text{dprem}(A_i', \langle A_1', \ldots, A_{i-1}', A_{i+1}', \ldots, A_n' \rangle)$ hence $\vec{A}_n$ is a kernel basis.□

We now turn to the problem of computing complete semi-kernel and complete kernel bases.

3.6 Complete Basis Algorithm for F[x]: Consensus Selection Rules

In this section we describe an algorithm for computing monic complete kernel bases in F[x]. Let $\vec{A}$ be an arbitrary basis for ideal $\overline{A}$ in F[x]. Algorithm 2.5 provides a method for computing a complete basis for $\overline{A}$ once the function β and the consensus choice function have been specified. Note that Algorithm 2.5 requires every element of $cons(\vec{A})$ to be remaindered. By combining results of Sections 2.8 and 3.3 we will show that it is possible to decide if $\vec{A}$ is a complete basis by using a rather small subset of $cons(\vec{A})$.

As in Section 3.4 we assume that $\overline{A} \neq \{0\}$ and that every element of $\vec{A}$ is non-zero. We wish to point out that Algorithm 2.5 only forms a loose framework for the algorithm to be presented in this section. In particular, we will never form the set $\hat{C} = \{C \in cons(\vec{A}): rem(C,\vec{A}) \neq 0\}$. Rather, we will discuss strategies for choosing subsets of elements of $\vec{A}$ whose consensus polynomials are possible candidates for inclusion in $\hat{C}$. The consensus choice function mentioned in Section 2.5 thus becomes a function which selects a subset of $\vec{A}$ whose consensus in to be evaluated.

Our first step in specifying a complete basis algorithm is to specify the remainder algorithm to be used. The remarks following the termination proof of Algorithm 2.5 show that, in the computation of a complete basis, it is sufficient to form the semi-remainder of a given consensus polynomial instead of the full remainder. Since

the semi-remainder algorithm produces a semi-kernel polynomial, it, in general, has a smaller computing time than the remainder algorithm. We will use the semi-remainder algorithm, dpsrem(P,$\vec{A}$), presented in Section 3.4.

To continue the specification of the complete basis algorithm we discuss consensus selection rules.

Consensus Selection Rule 1 (CSR1) If $\vec{b} \in F^n$ then a sequence of generators for $V(\vec{b})$ will be computed using $\beta(\vec{b})$ in Definition 3.3.1.

We develop the implications of this selection rule in the following definitions and lemmas.

Definition 3.6.1 Let A and B be two non-zero polynomials in F[x] with a = lc(A), b = lc(B) and J = lcm(∂A,∂B). The consensus of the pair (A,B) is defined to be the polynomial $P = bx^{J-\partial A}A - ax^{J-\partial B}B$. We denote this by cons(A,B).

Definition 3.6.2 Let $\vec{A} = \langle A_1,\ldots,A_n\rangle$ be a basis for ideal $\overline{A}$ in F[x]. Then the set $\{cons(A_i,A_j): 1 \leq i < j \leq n, A_i \neq 0, A_j \neq 0\}$ is called the modified consensus set of $\vec{A}$, and is denoted mcons($\vec{A}$).

Note that mcons($\vec{A}$) is clearly a subset of cons($\vec{A}$) determined by the β given in CSR1.

Lemma 3.6.1 Let $\vec{A}$ be a basis for ideal $\overline{A}$ in F[x]. Then every element of mcons($\vec{A}$) is simply constructible from $\vec{A}$ iff every element of cons($\vec{A}$) is also.

Proof Since mcons($\vec{A}$) $\subseteq$ cons($\vec{A}$), the only if part is obvious.

Let $C \in \text{cons}(\vec{A})$. If $C = 0$ then C is simply constructible from $\vec{A}$, so assume $C \neq 0$. According to Definition 2.4.1 there exists a $\vec{B} \in W(\vec{A})$ and a $\vec{C} \in (F[x])^n$ such that $C = \vec{C} \cdot \vec{B}$. For this $\vec{B}$ let us rearrange the basis $\vec{A}$ such that if $\vec{B} = \langle B_1, \ldots, B_n \rangle$ then $B_i = A_i \neq 0$ for $1 \leq i \leq m$ and $B_i = 0$ for $m < i \leq n$. Since $C \neq 0$ we know $m \geq 2$. Also let $\vec{b} = \langle b_1, \ldots, b_n \rangle$ where $b_i = \text{lc}(B_i)$, $1 \leq i \leq n$.

By Definition 2.4.1 $\vec{C} = \langle c_1 x^{K-\partial A_1}, \ldots, c_m x^{K-\partial A_m}, 0, \ldots, 0 \rangle$ where $K = \text{lcm}(\partial A_1, \ldots, \partial A_m)$ and $\vec{c} = \langle c_1, \ldots, c_m, 0, \ldots, 0 \rangle$ is an element of $\beta(\vec{b})$. Now $C \neq 0$ so, by Definition 3.3.1 and CSR1, $\vec{c} = b_j \vec{u}_1 - b_i \vec{u}_j$ for some j, $2 \leq j \leq m$, and $C = b_j x^{K-\partial A_1} A_1 - b_1 x^{K-\partial A_j} A_j$.

Let $J = \text{lcm}(\partial A_1, \partial A_j)$. Now $J | K$ and for $1 \leq j \leq m$, $b_j = \text{lc}(B_j)$ $= \text{lc}(A_j) = a_j$. Thus we can write $C = x^{K-J}(a_j x^{J-\partial A_1} A_1 - a_1 x^{J-\partial A_j} A_j)$ $= x^{K-J} \text{cons}(A_1, A_j)$.

But $\text{cons}(A_1, A_j) \in \text{mcons}(\vec{A})$ and since $\text{cons}(A_1, A_j)$ is simply constructible from $\vec{A}$, then so is $C = x^{K-J} \text{cons}(A_1, A_j)$. □

Theorem 2.3 tells us that $\vec{A}$ is complete iff all elements of $\text{cons}(\vec{A})$ are simply constructible from $\vec{A}$, or, what is the same thing, iff all elements of $\text{cons}(\vec{A})$ remainder to zero. Thus to check if $\vec{A}$ is a complete basis using Algorithm 2.5 requires 2^n remainder operations where n is the number of basis elements in $\vec{A}$. As a consequence of Theorem 2.3 and Lemma 3.6.1 it is easy to see that in $F[x]$, $\vec{A}$ is a complete basis if every element of $\text{mcons}(\vec{A})$ remainders to zero. The complete basis algorithm to be presented relies on this and hence

requires at most $n(n-1)/2$ remainder operations to determine whether $\vec{A}$ is a complete basis.

We can further reduce the number of consensus polynomials which need to be remaindered by applying Lemma 2.8.3. Let $\vec{A} = \langle A_1,\ldots,A_n\rangle$ with $a_i = lc(A_i)$, $1 \leq i \leq n$. Assume $A_i \neq 0$ and $A_j \neq 0$. Since F is a field, a_i and a_j are units; if $lcm(\partial A_i,\partial A_j) = \partial A_i + \partial A_j$, then $cons(A_i,A_j)$ is simply constructible from the basis $\langle A_i,A_j\rangle$ and, therefore, also from the basis $\vec{A}$.

Every time a non-zero consensus remainder is found in step 5 of Algorithm 2.5, the remainder polynomial is appended to the current basis. No provision is made for the possible deletion of basis elements. When the coefficient ring is a field, however, deletions are sometimes possible, as the next lemma shows.

Lemma 3.6.2 Let $\vec{A} = \langle A_1,\ldots,A_n\rangle$ generate ideal $\overline{A}$ in F[x]. Assume $A_1 \neq 0$, $A_2 \neq 0$, and $\partial A_2 | \partial A_1$. Let $A_1' = dpsrem(cons(A_1,A_2),\vec{A})$. Then $\overline{A}$ is also generated by $\vec{A}' = \langle A_1',A_2,\ldots,A_n\rangle$. If $\vec{A}$ is complete, so is $\vec{A}'$.

Proof Let $A_1'' = cons(A_1,A_2)$. Then $A_1'' = a_2A_1 - a_1x^{\partial A_1-\partial A_2}A_2$ since $lcm(\partial A_1,\partial A_2) = \partial A_1$. Hence $A_1 = (A_1'' + a_1x^{\partial A_1-\partial A_2}A_2)/a_2$. Hence $A'' = \langle A_1'',A_2,\ldots,A_n\rangle$ generates $\overline{A}$, and it is easy to see that if $P \in \overline{A}$ is simply constructible from $\vec{A}$ then it is also simply constructible from $\vec{A}''$.

Now $A_1' = dpsrem(A_1'',\vec{A})$. But $\partial A_1'' < \partial A_1$, hence $dpsrem(A_1'',\vec{A}) = dpsrem(A_1'', \langle A_2,\ldots,A_n\rangle)$. Therefore, by Lemma 2.8.1, $\vec{A}'$ generates $\overline{A}$, and if $\vec{A}''$ is complete, then so is $\vec{A}'$.□

We take Lemma 3.6.2 to define our second selection rule.

Consensus Selection Rule 2 (CSR2) Let $\vec{A} = \langle A_1,\ldots,A_n\rangle$, and let (A_i,A_j) be the next consensus pair selected, $i \neq j$. If $\partial A_i \mid \partial A_j$ then delete A_j after computing $\text{dpsrem}(\text{cons}(A_j,A_i),\vec{A})$, otherwise, if $\partial A_j \mid \partial A_i$ then delete A_i after computing $\text{dpsrem}(\text{cons}(A_i,A_j),\vec{A})$.

Note that the consistent application of this rule will produce a complete basis that is also a semi-kernel basis.

By applying Lemmas 3.6.1 and 2.8.3 along with consensus selection rule 1, we have managed to considerably restrict the size of the consensus set generated in Algorithm 2.5. It remains now to specify a consensus choice function which will define the order in which the remaining consensus operations will be evaluated.

Let $\vec{A} = \langle A_1,\ldots,A_n\rangle$ and let $L = \{(A_i,A_j): 1 \leq i < j \leq n$ such that $A_i \neq 0$, $A_j \neq 0$, and $\text{lcm}(\partial A_i,\partial A_j) \neq \partial A_i + \partial A_j\}$. L is called the consensus list of basis $\vec{A}$. The previous results in this section show that $\vec{A}$ is simply constructible iff for all $(A_i,A_j) \in L$, $\text{cons}(A_i,A_j)$ remainders to zero with respect to $\vec{A}$.

Now if $(A_i,A_j) \in L$, let $J = \text{lcm}(\partial A_i,\partial A_j)$ and $C = \text{cons}(A_i,A_j)$. By Definition 3.6.1, $\partial C < J$, and hence the remainder of C with respect to $\vec{A}$ will have degree strictly less than J. If this remainder is non-zero, it will be added to $\vec{A}$. One can argue that, by choosing the pair (A_i,A_j) with minimal J, the smallest possible polynomials will be added to $\vec{A}$.

Notice also that L can be partitioned into two disjoint subsets as follows: $L_1 = \{(A_i,A_j) \in L: \partial A_i | \partial A_j \text{ or } \partial A_j | \partial A_i\}$ and $L_2 = L-L_1$ where the minus sign indicates set theoretic difference. L_1 contains the consensus pairs which will result in the deletion of an element from basis $\vec{A}$, and is called the deletion consensus list of $\vec{A}$. If we form the consensus of pairs from this list whenever possible, then we will limit, to some extent, the growth of the number of elements in $\vec{A}$. This should usually have the effect of speeding the remainder algorithm and reducing the number of consensus operations required.

We combine these two strategies in our third consensus selection rule. This completes the specification of the consensus choice function.

Consensus Selection Rule 3 (CSR3) Let L_1 and L_2 be defined as above. If L_1 is non-empty then let $L' = L_1$ else let $L' = L_2$. Let (A,B) be the first element of L' such that $J = lcm(\partial A,\partial B) = \min\{lcm(\partial A',\partial B') : (A',B') \in L'\}$. The next consensus to be formed will be cons(A,B).

Before we can give the actual complete basis algorithm, we need to give the specifications for an algorithm for inserting lists containing exponent vectors in other lists. We assume the availability of the following algorithm.

L'←EVLIN(L,M,k)

[Exponent vector, list insertion. L and M are lists. If L=() then L'=(M). Otherwise $L=(M_1,\ldots,M_s)$ where M_i is a list such that

K_i=FIRST(M_i) is an exponent vector compatible with the exponent vector K=FIRST(M). The elements of L are arranged such that the exponent vectors K_i are in non-decreasing (k=1) or non-increasing (k=-1) order. M is inserted in L such that the ordering is preserved. L' is the new list; L is modified.]

Assuming fixed length exponent vectors, if s = LENGTH(L) then the computing time of EVLIN is dominated by s+1. EVLIN will be used to perform insertions in lists representing ideal bases and in lists representing sublists of consensus lists.

We represent the consensus list of basis $\vec{A}$ by a two element list L = (L_1,L_2) where L_1 is the deletion consensus list and L_2 consists of all other consensus pairs. If $\vec{A} = \langle A_1,\ldots,A_n\rangle$, $A_i \neq 0$, $A_j \neq 0$ and $lcm(\partial A_i,\partial A_j) \neq \partial A_i + \partial A_j$, then the consensus pair (A_i,A_j) is represented in L_1 or L_2 by the triple (K,A,B) where $K = lcm(A_i,A_j)$. If $\partial A_i \mid \partial A_j$ then $A = A_j$, $B = A_i$, otherwise $A = A_i$, $B = A_j$. If $L_1 = ()$ and $L_2 = ()$ then L = ().

The consensus list for basis $\vec{A}$ is maintained by algorithms CLGEN, CLSEL, and CLDEL. CLGEN maintains the consensus lists L_1 and L_2 such that the least common multiples of the consensus pairs are in non-decreasing order in each list. Thus CLSEL can implement CSR3 by merely selecting the first consensus pair in L_1 or L_2. CLDEL deletes unnecessary consensus pairs.

Algorithm 3.6.1

$L' \leftarrow CLGEN(L,A,\vec{B})$

[Consensus list generate new elements.

Input: L, a consensus list, $A \in F[x]$ and $\vec{B}=<B_1,\ldots,B_n> \in (F[x])^n$, $A\neq 0$, $B_i\neq 0$, $1\leq i\leq n$.

Output: L' a consensus list composed of the elements of L and, in addition, the consensus pairs (A,B_i), $1\leq i\leq n$, such that $lcm(\partial A,\partial B_i)\neq \partial A+\partial B_i$. The sublists of L and L' are arranged such that the least common multiples of the leading exponent vectors of the consensus pairs are in non-decreasing order. This partially implements consensus selection rules 2 and 3. The list representing L is modified.]

(1) [Initialize.] if $L\neq()$ then $FIRST2(L;L_1,L_2)$ else $\{ L_1\leftarrow(); L_2\leftarrow() \}$; $I\leftarrow DPLEV(A)$; $\vec{B}'\leftarrow\vec{B}$.

(2) [Pair A with elements of $\vec{B}$.] repeat { $ADV(\vec{B}';B,\vec{B}')$; $J\leftarrow DPLEV(B)$; $K'\leftarrow EVGCD(I,J)$; if $K'\neq()$ then { $K\leftarrow EVLCM(I,J)$; $I'\leftarrow EVDIF(K,I)$; $J'\leftarrow EVDIF(K,J)$; if $J'=()$ then $M\leftarrow LIST3(K,B,A)$ else $M\leftarrow LIST3(K,A,B)$; if $I'=() \vee J'=()$ then $L_1\leftarrow EVLIN(L_1,M,1)$ else $L_2\leftarrow EVLIN(L_2,M,1)$ } } until $\vec{B}'=()$.

(3) [Return L'.] if $L_1=()$ & $L_2=()$ then $L'\leftarrow()$ else $L'\leftarrow LIST2(L_1,L_2)$; return □

Note that in step 2 we avoid adding the pair (A,B_i) to the consensus list if $\gcd(\partial A,\partial B_i) = () = \partial(1)$. This is equivalent to checking whether $\text{lcm}(\partial A,\partial B_i) = \partial A + \partial B_i$. Let $s = \max(\text{LENGTH}(L_1), \text{LENGTH}(L_2))$. Then, for a fixed number of variables in the exponent vectors, the computing time of CLGEN is dominated by $n(n+s)$.

The algorithm to select the next consensus pair is very simple.

Algorithm 3.6.2

CLSEL(L;L',K,A,B)

[Consensus list, selection.
Input: $L=(L_1,L_2)$ a consensus list, where either $L_1\neq()$ or $L_2\neq()$.
Output: $A,B \in F[x]$, the next consensus pair, and $K=\text{lcm}(\partial A,\partial B)$. L' is a consensus list equivalent to L with the pair (A,B) removed. This partially implements consensus selection rule 3.]

(1) [Initialize.] FIRST2(L;L_1,L_2).

(2) [Choose pair.] if $L_1\neq()$ then ADV(L_1;M,L_1) else ADV(L_2;M,L_2); FIRST3(M;K,A,B).

(3) [Finish.] if $L_1=()$ and $L_2()$ then L←() else L←LIST2(L_1,L_2); return □

The computing time for CLSEL is codominant with 1.

Algorithm 3.6.3

M←CLDEL(L,C)

[Consensus list, deletion.
Input: L, a consensus list, and C, a distributive polynomial.

Output: M, the consensus list gotten from L by deleting all consensus pairs containing C. L is modified.]

(1) [L=().] if L=() then { M←(); return }.

(2) [Scan L_1 then L_2.] FIRST2(L;L_1,L_2); M'←L_1; i←1.

(3) [M'=().] if M'=() then go to 6.

(4) [C in FIRST(M').] FIRST3(FIRST(M');K,A,B); if C=A ∨ C=B then { M'←RED(M'); go to 3 }.

(5) [Scan M'.] L"←M' L'←RED(L"); while L'≠() do { FIRST3(FIRST(L');K,A,B); if C=A ∨ C=B then SRED(L";RED(L')) else L"←L'; L'←RED(L") }.

(6) [Next list.] if i=1 then { L_1←M'; M'←L_2; i←2; go to 3 }; L_2←M'.

(7) [Return M.] if L_1=() & L_2=() then M←() else M←LIST2(L_1,L_2); return □

If L = (L_1,L_2), the computing time of CLDEL is dominated by max(LENGTH(L_1),LENGTH(L_2)) + 1.

Finally we assume the existence of an algorithm to delete γ-integers (see Section 1.6) from a list of γ-integers. This algorithm will be used to delete elements of a polynomial ideal basis. The computing time of LDEL is dominated by the number of elements in the list L.

M←LDEL(L,A)

[List deletion. L=(L_1,...,L_n), n≥1, is a list of γ-integers. A is

a γ-integer. If i is the least integer such that $L_i=A$, then $M=(L_1,\ldots,L_{i-1},L_{i+1},\ldots,L_n)$. Otherwise M=L. L is modified.]

We can now present an algorithm for computing monic complete kernel bases in F[x]. This algorithm is abstract only in that it depends on arithmetic in the coefficient field F.

Algorithm 3.6.4

$\vec{C}\leftarrow$mckbasis($\vec{A}$)

[Polynomial over a field ideal basis, monic complete kernel basis. Input: $\vec{A}=\langle A_1,\ldots,A_n\rangle \in (F[x])^n$, a basis for ideal $\overline{A}$ in F[x], with $A_i\neq 0$, $1\leq i\leq n$, and $\partial A_1\leq\partial A_2\leq\ldots\leq\partial A_n$. Output: $\vec{C}=\langle C_1,\ldots,C_s\rangle \in (F[x])^s$, a monic complete kernel basis for $\overline{A}$. $\partial C_1<\partial C_2<\ldots<\partial C_s$. This algorithm uses consensus selection rules 1, 2, and 3, and remainder selection rules 1 and 2. The list representing $\vec{A}$ is modified.]

(1) [Initialize consensus list.] $\vec{C}\leftarrow\vec{A}$; L$\leftarrow$(); ADV($\vec{A}$;A,$\vec{A}'$); while $\vec{A}'\neq()$ do { L$\leftarrow$CLGEN(L,A,$\vec{A}'$); ADV($\vec{A}'$;A,$\vec{A}'$) }.

(2) [Select consensus pair.] if L=() then go to 8; CLSEL(L;L,K,A,B).

(3) [Form consensus and semi-remainder.]
DPLECR(A;I,a,A'); I'$\leftarrow$EVDIF(K,I); a$\leftarrow$fneg(a);
DPLECR(B;J,b,B'); J'$\leftarrow$EVDIF(K,J);
C$\leftarrow$dpsum(dptermpr(I',b,A'),dptermpr(J',a,B'));
C$\leftarrow$dpsrem(C,$\vec{C}$).

(4) [Delete if possible.] if I'=() then { $\vec{C}$←LDEL($\vec{C}$,A); L←CLDEL(L,A) }.

(5) [C is constant?] if C=0 then go to 2; if DPLEV(C)=() then go to 7.

(6) [Modify consensus list and basis.] L←CLGEN(L,C,$\vec{C}$); $\vec{C}$←EVLIN($\vec{C}$,C,1); go to 2.

(7) [Identity element is in ideal.] $\vec{C}$←LIST1(dpmon(C)); return.

(8) [Make $\vec{C}$ into monic kernel basis.] ADV($\vec{C}$;C,$\vec{B}$); $\vec{C}$←LIST1(dpmon(C)); while $\vec{B}\neq()$ do { ADV($\vec{B}$;C,$\vec{B}$); C←dprem(C,$\vec{C}$); C←dpmon(C); $\vec{C}$←EVLIN($\vec{C}$,C,1) }; return □

The termination and correctness of Algorithm 3.6.4 are directly related to the termination and correctness of Algorithm 2.5. In step 6 of Algorithm 3.6.4 the polynomial C added to basis $\vec{C}$ is a non-zero semi-kernel polynomial with respect to $\vec{C}$. Lemma 2.5.2 shows that we can add only a finite number of such polynomials and hence Algorithm 3.6.4 terminates. At termination the consensus list L is empty indicating that all elements in mcons($\vec{C}$), and hence all elements of cons($\vec{C}$), are simply constructible from $\vec{C}$.

Since we are deleting elements whenever possible in step 4, the basis $\vec{C}$ at the beginning of step 8 will be a semi-kernel basis. Step 8 of Algorithm 3.6.4 is a straightforward implementation of the algorithm embedded in Lemma 3.5.3 with the added provision that the basis elements are made monic. Therefore, at the end of step 8, $\vec{C}$ is a monic complete kernel basis for $\overline{A}$. Note that in this step we

make explicit use of the order of the basis elements.

To analyze the computing time of Algorithm 3.6.4 it is necessary to specify the field F being used. But, without specifying F, we can discuss the possibility of bounding the number of exponent vector operations required. To find such a bound we need to be able to find an upper limit on the number of basis elements in any of the intermediate bases and on the maximum degree of any polynomial contained in these bases.

In the termination proof for Algorithm 2.5 we showed that the leading term ideals generated by the intermediate bases formed a strictly ascending chain of ideals in R[x]. The same thing holds true in F[x] during the execution of Algorithm 3.6.4. In [SEI56] and [SEI71], Seidenberg discusses bounds for the maximum length of any strictly ascending chain of ideals in F[x] in case a bound exists for the maximum degree in any variable of the basis elements of any ideal in the ascending chain. In [SEI56] a bound is given for the case where each ideal in the chain is generated by monomials--just the situation we have in Algorithms 2.5 and 3.6.4. Unfortunately, the chain length bound Seidenberg derives is not even a primitive recursive function of the degree bounds.

In the next chapter we give a modification of Algorithm 3.6.4 which controls the degree growth and the number of elements in the intermediate bases. This provides us with an algorithm whose computing time is polynomially bounded for the polynomial ring GF(p)[x,y].

CHAPTER FOUR:

Resultant Systems and Complete Bases

4.1 Introduction

In the previous chapter, we indicated that the analysis of the complete basis algorithm of Section 3.6 is very difficult. The present chapter introduces the notion of a resultant system of a set of polynomials, and shows how this system can be used to yield a polynomially bounded complete basis algorithm for certain polynomial rings

The motivation for this work comes from the following observation. Suppose ideal $\overline{A}$ in $F[x,y]$, F a field, is generated by the polynomials $A_1(x,y)$ and $A_2(x,y)$. We wish to find a complete basis for $\overline{A}$. By Lemma 2.8.2, we can assume, without loss of generality, that $gcd(A_1,A_2)=1$. Let us assume A_1 and A_2 are of positive degree in both x and y. If R_x and R_y are the resultants of A_1 and A_2 with respect to x and y, respectively, then $R_x \neq 0$ and $R_y \neq 0$ (see [VdW70a]). We also know that $R_x \varepsilon F[y]$ and $R_y \varepsilon F[x]$ and, further, that there exist S_x, S_y, T_x, and $T_y \varepsilon F[x,y]$ such that $S_x A_1 + T_x A_2 = R_x$ and $S_y A_1 + T_y A_2 = R_y$. Hence, $R_x, R_y \varepsilon \overline{A}$, and since they are univariate, we can use them to limit the number of intermediate basis elements and their degrees during the computation of a complete basis.

Sections 4.2 and 4.3 discuss the problems of computing multipolynomial gcd's and resultant systems thus generalizing the resultant to the case of more than two polynomials. Section 4.4 describes a version of the complete basis algorithm which can be applied when the ideal contains an appropriate set of univariate polynomials. Section 4.5 combines the results of Sections 4.2, 4.3, and 4.4 in a complete

basis algorithm which, for bivariate polynomials over GF(p), has a polynomially bounded computing time.

4.2 Multipolynomial Greatest Common Divisors

To make full use of Lemma 2.8.2 in computing complete bases, we need to be able to find the greatest common divisor of several (i.e., more than two) polynomials. In this section, we present an informal discussion of this problem along with information concerning the SAC-2 system to aid in understanding the detailed algorithms presented in Appendix B.

Let A be a polynomial in $R[x]=R[x_1,\ldots,x_r]$. In this section, we will use the notation $\partial_i A$ to represent the degree of A in variable x_i. Also, if R=Z, we use $|A|_\infty$ to denote the max norm of A, i.e., the maximum absolute value of the integer coefficients of A, and we use $|A|_1$ to denote the sum norm of A, i.e., the sum of the absolute values of the integer coefficients of A. If d is a non-zero integer, then by the length of d, L(d), we mean the number of digits required to represent d in some fixed number base.

If A_1 and A_2 are polynomials in R[x] where R=Z, the ring of integers, or R=GF(p), the finite field with p elements, p a prime, then the problem of computing $G=\gcd(A_1,A_2)$ is relatively well understood. Let us assume that $\partial_i A_1, \partial_i A_2 < m$ for $1 \le i \le r$ and if R=Z, then we assume $|A_1|_1, |A_2|_1 \le d$. Under the assumption that all primes are single precision, that a negligible number of "unlucky" primes are processed, and that $G, A_1/G$, and A_2/G have sum norms of d or less, Brown [BRN71] has shown that modular algorithms lead to the following results. If R=GF(p) then the time to compute G is dominated by m^{r+1}. If R=Z, then the time to compute G is dominated by $m^{r+1}L(d) + m^r L(d)^2$. A prime is said to be

"unlucky" if the gcd of the homomorphic images of A_1 and A_2 has degree in some variable different from the degree of the homomorphic image of G (see [BRN71]).

The restrictions mentioned in the previous paragraph imply that the computing times given are not true bounds on the maximum computing time for finding G. We now give a derivation of maximum computing times for gcd algorithms similar to Brown's.

A modular algorithm to compute $G=\gcd(A_1,A_2)$ in $R[x_1,\ldots,X_r]$ can be described informally as follows:

(1) If A_1 and A_2 are in the coefficient ring, then we assume we have an algorithm to compute G; if not, let G=0.

(2) Choose a prime element, p, of the coefficient ring and set $A_1'=A_1 \bmod p$, $A_2'=A_2 \bmod p$.

(3) Compute $G'=\gcd(A_1',A_2')$ in $R[x_1,\ldots,x_r]/(p)$.

(4) If G=0, then set $G=G'$; otherwise combine G and G' using the Chinese Remainder Algorithm.

(5) If enough lucky primes have been processed, then $G=\gcd(A_1,A_2)$; otherwise, repeat steps 2 through 5 with a new prime.

If R=GF(p), p a single precision prime integer, then the coefficient ring for the gcd algorithm described is $GF(p)[x_r]$. A prime element of this ring is a polynomial x_r-b where $b\in GF(p)$. The number of primes required by the algorithm depends on the degree of G. Hence, p must be fairly large for the algorithm to succeed.

Let T(r,m) be the maximum computing time required to find G when R=GF(p) and $\partial_i A_1, \partial_i A_2 < m, 1 \le i \le r$. If r=1, then it is possible to compute

G using the Euclidean algorithm in $GF(p)[x_1]$ in time dominated by m^2. Hence, $T(1,m) \leq fm^2$ where f is a constant.

Assume $r \geq 2$. Reduction modulo $x_r - b$ is actually evaluation at $x_r = b$. Hence, one execution of step 2 takes time dominated by m^r, the number of terms in A_1 or A_2. The same bound holds for a single execution of the Chinese Remainder Algorithm in step 4, which in this case is actually polynomial interpolation. The remaining step, step 3, requires the amount of time given by $T(r-1,m)$.

To determine the number of times the loop from step 2 to 5 is executed, we proceed as follows. Since $G|A_1$ and $G|A_2$, $\partial_r G < m$. Hence, at most m lucky primes must be processed before $G = \gcd(A_1, A_2)$. However, Brown [BRN71, p495] shows that at most $2rm^2$ unlucky primes will be encountered. Hence, we can express $T(r,m)$ as follows:

$$T(1,m) \leq fm^2,$$

$$T(r,m) \leq frm^{r+2} + 2rm^2 T(r-1,m).$$

By induction on r, we can show that $T(r,m) \leq (2^r - 1)fr!m^{2r}$ which is dominated by $(2rm^2)^r$. This, then, is our maximum computing time bound for the case $R = GF(p)$.

If $R = Z$, then the coefficient ring is just the integers and the primes used are just the integral primes. If p is a single precision prime integer, then the gcd computation in step 3 takes place in the ring $GF(p)[x_1, \ldots, x_r]$.

Let $T'(r,m)$ be the maximum computing time required to find $G = \gcd(A_1, A_2)$ when $R = Z$, $\partial_i A_1, \partial_i A_2 < m$ and $|A_1|_1, |A_2|_1 < d$. Let us first consider the numbers of lucky primes required. Let $g = |G|_1$. Since

$G|A_1$, a result of Gelfond [GEL60, p135], shows that $g \leq e^{mr}|A_1|_1 \leq e^{mr}d$. Hence, L(g) is dominated by mr+L(d), and the number of lucky primes is also dominated by mr+L(d). The number of unlucky primes is shown by Brown [BRN71, p492] to be dominated by mrL(d).

The computing time for the modular reduction in step 2 for one single precision prime is dominated by $m^r L(d)$, the number of terms in A_1 or A_2 times the time to reduce one term. The time to compute the gcd over GF(p) for a single prime is given by T(r,m) which we showed to be dominated by $(2rm^2)^r$. Finally, the time to apply the Chinese Remainder Algorithm for a single prime is dominated by $m^r L(g)$ which is dominated by $m^r(mr+L(d))$.

Combining these results we find that $T'(r,m)$ is dominated by $mrL(d)(m^r L(d) + (2rm^2)^r + m^r(mr+L(d)))$ which, in turn, is dominated by $2^r r^{r+1} m^{2r+1} L(d) + rm^{r+1} L(d)^2$.

We now turn to the problem of computing the greatest common divisor of more than two polynomials. If $A_1,\ldots A_n \in R[x]$, $m \geq 2$, then the most straightforward way to determine $G=\gcd(A_1,\ldots,A_n)$ is to compute $G=\gcd(\gcd(\ldots(\gcd(A_1,A_2),A_3)\ldots,A_{n-1}),A_n)$. The cofactors of $A_1,\ldots,A_n$ with respect to G are just the polynomials $A_1/G,\ldots,A_n/G$. These can be computed by division. Let $\partial_i A_j < m$, $1 \leq i \leq r$, $1 \leq j \leq n$. If R=GF(p), then the time to compute G is dominated by $n(2rm^2)^r$ since if $B|A$ then $\partial_i B \leq \partial_i A$. The time to compute A_i/G is codominant with the time required to multiply A_i/G by G which is dominated by m^{2r}. Hence, the time to find the cofactors is also dominated by $n(2rm^2)^r$.

In the case R=Z, the coefficients of the intermediate gcd's can grow in size. However, Gelfond's result shows that if $B|A$, then $|B|_1 \leq e^{rm}|A|_1$. If $G=\gcd(A_1,\ldots,A_n)$, then $g=|G|_1 \leq e^{mr}|A_i| \leq e^{mr}d$, and hence, $L(g)$ is dominated by $mr+L(d)$. Note, also, that the sum norm of the gcd of any subset of $A_1,\ldots,A_n$ will also be dominated by $e^{mr}d$. Hence, the time to compute G will be dominated by $n(2^r r^{r+1} m^{2r+1} L(g) + rm^{r+1} L(g)^2)$. Substituting $mr+L(d)$ for $L(g)$, we find that the time to compute $\gcd(A_1,\ldots,A_n)$ in $Z[x_1,\ldots,x_r]$ is dominated by $n2^r r^{r+2} m^{2r+2} + n2^r r^{r+1} m^{2r+1} L(d) + nrm^{r+1} L(d)^2$. The time to compute the cofactors is dominated by $nL(g)L(d)m^{2r}$ which is dominated by $nrm^{2r+1} L(d) + nm^{2r} L(d)^2$.

The complete basis algorithms described in this thesis require all polynomials to be in distributive canonical form. Unfortunately, the SAC-2 gcd algorithms assume that their inputs and outputs are in <u>recursive canonical form</u>. Recursive canonical form results from considering a polynomial $A \in R[x]$ as a polynomial in a single variable x_r, $A=\sum_{i=0}^{m} A_i x_r^i$ where $A_m \neq 0$ and $A_i \in R[x_1,\ldots,x_{r-1}]$, $0 \leq i \leq m$. If $r=1$, let A_i^* be a canonical form for the element $A_i \in R$, otherwise let A_i^* be the recursive canonical form for the polynomial $A_i \in R[x_1,\ldots,x_{r-1}]$, $0 \leq i \leq m$. Finally, let $(i_1,\ldots,i_n)$ be the sub-sequence of $(0,\ldots,m)$ such that $A_{i_j} \neq 0$. The recursive canonical form A^* of A is defined as follows. If $A=0$, then $A^*=0$, otherwise $A^*=(i_n, A_{i_n}^*, \ldots, i_1, A_{i_1})$.

The algorithms to compute the gcd's (and also resultants) of polynomials in distributive form must convert back and forth between distri-

butive and recursive form. The time for conversion in either direction is dominated by the time it takes to sort the terms of the distributive canonical form polynomial in case it is not in lexicographic order. A merge sort can be performed in time dominated by tlog(t) where t is the number of items to be sorted. If $A \in R[x]$ and $\partial_i A < m$, then A contains at most m^r terms. Since the sort must compare exponent vectors, each comparison will have a computing time dominated by r. Therefore, the computing time for sorting a distributive polynomial, and hence the computing time for conversion between recursive and distributive canonical form, is dominated by $rm^r \log(m^r) = r^2 m^r \log(m)$.

Combining the preceding results, we assume the existence of the following algorithm.

ldpgcdc(L;G,L′)

[List of distributive polynomials, greatest common divisor and cofactors. $L=(A_1,\ldots,A_n)$, $n \geq 2$, is a list of polynomials in distributive canonical form. $G=\gcd(A_1,\ldots,A_n)$ and $L'=(A_1',\ldots,A_n')$ where $A_i'=A_i/G$, $1 \leq i \leq n$.]

If $\partial_i A_j < m$ for $1 \leq i \leq r$ and $1 \leq j \leq n$, then the computing time of ldpgcdc is dominated by $n \cdot (2rm^2)^r$ for the case $GF(p)[x_1,\ldots,x_r]$. In the case $Z[x_1,\ldots,x_r]$, the time is dominated by $n2^r r^{r+2} m^{2r+2} + n2^r r^{r+1} m^{2r+1} L(d) + (nrm^{r+1} + nm^{2r}) L(d)^2$.

4.3 Multipolynomial Resultant Systems

Let $A_1,\ldots,A_n \in R[x]$, $n\geq 2$, where x is a single indeterminate and R is a gcd domain, and $\partial_x A_i > 0$ for $1\leq i\leq n$. Let $y_2,\ldots,y_n$ be distinct indeterminates different from x, and consider the polynomials $U=A_1$ and $V=\sum_{i=2}^{n} y_i A_i$ contained in $R[y_2,\ldots,y_n,x]$. Let $R_x=res_x(U,V)$ be the resultant of U and V with respect to x. Since $R_x \in R[y_2,\ldots,y_n]$, we can write $R_x=\sum_{I\in E} r_I y^I$ where E is the set of $(n-1)$-variate exponent vectors, $y^I=y_2^{e_2}\ldots y_n^{e_n}$ for some $\langle e_2,\ldots,e_n\rangle \in E$, and $r_I \in R$. Modifying Van der Waerden's usage slightly [VDW50], we call $RS_x(A_1,\ldots,A_n)=\{ r_I : r_I \neq 0 \}$ the resultant system of $A_1,\ldots,A_n$ with respect to x. Note that if $n=2$, then $R_x=y_2^{m_1} res_x(A_1,A_2)$ where $m_1=\partial_x A_1$.

The importance of the resultant system for our purposes is given in the following lemmas.

Lemma 4.3.1 Let $A_1,\ldots,A_n$ be polynomials in $R[x]$, with x a single indeterminate. Suppose that for $1\leq i\leq n$, $\partial_x A_i > 0$ and $\partial_x(gcd(A_1,\ldots,A_n))=0$. Then $RS_x(A_1,\ldots,A_n)$ is non-empty.

Proof Let U, V, and R_x be defined as above. By the well known theorem on resultants, $\partial_x(gcd(U,V))=0$ if and only if $R_x \neq 0$. Now let $G=gcd(U,V)$ and note that since U is independent of y_i, $\partial_{y_i} G=0$, $2\leq i\leq n$. Thus, if G is a common divisor of U and V, then G must be a common divisor of $A_1,\ldots,A_n$, and, in particular, G must divide $gcd(A_1,\ldots,A_n)$. Hence, $\partial_x G=0$. But then R_x cannot be identically zero, so $RS_x(A_1,\ldots,A_n)$ is non-empty □

The next lemma relates resultant systems to polynomial ideals.

Lemma 4.3.2 Let $\vec{A}=\langle A_1,\ldots,A_n\rangle$ be a basis for ideal $\bar{A}$ in $R[x]$ and suppose the elements of $\vec{A}$ satisfy the hypotheses of Lemma 4.3.1. If $r\varepsilon RS_x(A_1,\ldots,A_n)$, then $r\varepsilon\bar{A}$.

Proof Let U, V, and R_x again be defined as above. By another well known theorem on resultants, there exist polynomials S and T in $R[y_2,\ldots,y_n,x]$ such that $SU+TV=R_x$. This can also be written $SA_1+\sum_{i=2}^{n} Ty_iA_i=R_x$. Since the A_i are independent of the y_j, there exist polynomials S_{Ii} in $R[x]$ such that $R_x=\sum_{i=1}^{n}(\sum_{I\varepsilon E} S_{Ii}y^I)A_i = \sum_{I\varepsilon E}(\sum_{i=1}^{n} S_{Ii}A_i)y^I$. Comparing this with the expansion for R_x given above, we see that $r_I=\sum_{I\varepsilon E} S_{Ii}A_i$. Hence, if $r_I\varepsilon RS_x(A_1,\ldots,A_n)$, then $r_I\varepsilon\bar{A}$. □

The following corollary shows how it is possible to find univariate ideal elements in certain cases.

Corollary 4.3.1 Let $\langle A_1,\ldots,A_n\rangle$ generate ideal $\bar{A}$ in $R[w,x]$ where w and x are single indeterminates. Let $G=\gcd(A_1,\ldots,A_n)$. If $\partial_x A_i>0$, $1\le i\le n$ and $\partial_x G=0$, then $RS_x(A_1,\ldots,A_n)$ is non-empty, and, for $r\varepsilon RS_x(A_1,\ldots,A_n)$, $r\varepsilon\bar{A}$ and $r\varepsilon R[w]$.

Proof Apply Lemmas 4.3.1 and 4.3.2 with R replaced by R[w]. □

Let us now consider the problem of computing $RS_x(A_1,\ldots,A_n)$ when we replace x by x_r and R by $R[x_1,\ldots,x_{r-1}]$ with R=Z or R=GF(p). Let $\partial_{x_i} A_j<m$ for $1\le i\le r$ and $1\le j\le n$. In case R=Z, let $|A_j|_1\le d$ for $1\le j\le n$.

To compute U and V, we need to introduce the variables $y_2,\ldots,y_n$ into the A_i, $1\leq i\leq n$, and then to sum y_2A_2 through y_nA_n. The time for this process depends on the number of terms in the A_i. In case R=GF(p), the time for this is dominated by n^2m^r. If R=Z, the time is also dominated by n^2m^r because no coefficient operations are necessary. When this step is completed, we will have that $\partial_{x_i} U, \partial_{x_i} V<m$ and $\partial_{y_i} U=0$ and $\partial_{y_i} V=1$. Furthermore, if R=Z, then $|U|_1\leq d$ and $|V|_1\leq nd$.

In [COL71, pp 526-527], Collins has derived computing time bounds for resultant algorithms employing modular techniques, assuming the input polynomials are in recursive canonical form, and that they contain a fixed number of variables. If A and B are polynomials in r variables where r is fixed, and if the degrees and A and B in any variable are bounded by m, then for R=GF(p) Collins shows that the time to compute res(A,B) is dominated by m^{2r}. If R=Z and $|A|_1,|B|_1\leq d$, then the time to compute res(A,B) is dominated by $m^{2r+1}L(d) + m^{2r}L(d)^2$. If we now consider r to be variable, we can modify Collins' analysis and show that these bounds become $r2^rm^{2r}$ and $r2^rm^{2r+1}L(d) + 2^rm^{2r}L(d)^2$ for R=GF(p) and R=Z, respectively.

The polynomials U and V contain r+n-1 variables. Hence the time to compute $R_r=res_{x_r}(U,V)$ is dominated by $(r+n-1)2^{(r+n-1)}m^{2(r+n-1)}$ for

the case R=GF(p). If R=Z the bound becomes $(r+n-1)2^{(r+n-1)}m^{2(r+n-1)+1}$ $L(nd) + 2^{(r+n-1)}m^{2(r+n-1)}L(nd)^2$. Note that for a fixed number of polynomials and variables, these bounds are polynomial functions of m and L(d).

After R_r has been determined, we can compute the resultant system by finding the coefficients of R_r when we regard it as a polynomial in the variables $y_2,\ldots,y_n$ over the coefficient ring $R[x_1,\ldots,x_{r-1}]$. Now R_r is the determinant of a matrix composed of at most 2(m-1) rows and columns whose elements have degree less than m in the x_i and at most degree one in the y_j. Hence, $\partial_{x_i} R_r<2m^2$ and $\partial_{y_j} R_r<m$. Thus, we have fewer than m^{n-1} elements in the resultant system and each element has degree in x_i less than $2m^2$. These elements can be isolated in time dominated by m^{n-1}.

Our final subject in this section concerns the time required for conversions between distributive and recursive canonical form. Using the results at the end of section 4.2, we see that the time to convert all of the A_i to recursive canonical form is dominated by $n\cdot r^2m^r\cdot\log(m)$. Similarly, the time required to convert the resultant system to distributive canonical form is dominated by $m^{n-1}(r-1)^2(2m^2)^{(r-1)}\log(2m^2)$.

From these observations, we can bound the time required to compute the resultant system of $A_1,\ldots,A_n$ and perform the canonical form

conversions. Let $s = r+n-1$. In the case $R=GF(p)$ the time required is dominated by $n^2m^r + nr^2m^r\log(m) + s2^sm^{2s} + m^{(n-1)} + (r-1)^2(2m^2)^{(r-1)}\log(2m^2)$ which is dominated by $ns^22^sm^{2s}\log(m)$. If $R=Z$ then the time is dominated by $n^2m^r + nr^2m^r\log(m) + s2^sm^{2s+1}L(nd) + 2^sm^{2s}L(nd)^2 + m^{(n-1)} + (r-1)^2(2m^2)^{(r-1)}\log(2m^2)$. This bound is in turn dominated by $ns^22^sm^{2s+1}\log(m)L(nd)^2$. Again we note that for a fixed number of polynomials and a fixed number of variables, these bounds are polynomial functions of m and L(d).

We assume the existence of an algorithm with the preceding computing times and the following specification.

$L' \leftarrow$ ldpres(L)

[List of distributive polynomials, resultant system. $L=(A_1,\ldots,A_n)$, $n\geq 2$, is a list of non-zero polynomials in distributive canonical form in r variables with main variable x_r. For $1\leq i\leq n$ $\partial_{x_r} A_i>0$. $\partial_{x_r}\gcd(A_1,\ldots,A_n)=0$. L' is the resultant system of $A_1,\ldots,A_n$ with respect to x_r.]

4.4 Univariate Polynomials in Complete Basis Computations

Let $\vec{A}=<A_1,\ldots,A_n>$ be a basis for ideal $\bar{A}$ in $F[x]=F[x_1,\ldots,x_r]$ where F is a field. Suppose we also have a sequence of non-zero polynomials $\vec{R}=<R_1,\ldots,R_r>$ where, for $1\leq i\leq r$, $R_i\in\bar{A}$ and $R_i\in F[x_i]$, i.e., R_i is a non-zero univariate polynomial in the indeterminate x_i. In this section, we show how the complete kernel basis algorithm of Section 3.6 can be modified to use the polynomials in $\vec{R}$. The computing time of the modified algorithm, for certain coefficient fields, can be bounded by a polynomial function of the degrees of the A_i's and R_i's.

The key to the modified algorithm is the observation that if polynomial A is in a basis $\vec{A}$ for $\bar{A}$ and if $R_i\in\vec{R}$ is an element of $\bar{A}$ distinct from A, then A can be replaced by $\text{rem}(A,<R_i>)$. This operation has no effect on the degree of A in x_j, $j\neq i$, but it can lower the degree of A in x_i. If this operation is consistently applied during the course of a complete basis algorithm, we can limit the degrees of the intermediate basis elements and also the number of such elements.

We embody this observation in the special versions of the remainder and semi-remainder algorithms presented next.

Algorithm 4.4.1

$$Q\leftarrow\text{dpremu}(P,\vec{A},\vec{R})$$

[Distributive polynomial over a field, remainder using univariate ideal elements.

Input: $P\in F[x]=F[x_1,\ldots,x_r]$, $\vec{A}=<A_1,\ldots,A_n>\in(F[x])^n$, a basis for ideal $\bar{A}$ in $F[x]$, with $A_i\neq 0$, $1\leq i\leq n$, $\partial A_1\leq\ldots\leq\partial A_n$; $R=<R_1,\ldots,R_r>$, $R_k\neq 0$, $R_k\in\bar{A}$, $1\leq k\leq r$, and $\partial R_1\leq\ldots\leq\partial R_r$. For each j, $1\leq j\leq r$, there is exactly one R in $\vec{R}$ such that $R\in F[x_j]$.

Output: $Q=\mathrm{rem}(P,<A_1,\ldots,A_n,R_1,\ldots,R_r>).$]

(1) [$P=0.$] if $P=0$ then { $Q\leftarrow P$; return }.

(2) [Initialize.] $Q\leftarrow \mathrm{dprem}(P,\vec{R})$; if $Q=0$ then return; $Q'\leftarrow Q$; $Q\leftarrow()$.

(3) [Attempt to reduce $\mathrm{lt}(Q')$.] repeat { $\mathrm{DPLECR}(Q';J,q,Q')$;
$\mathrm{BESEL}(J,\vec{A};K,a,B)$; if $K=-1$ then $Q\leftarrow \mathrm{COMP2}(q,J,Q)$ else
{ $b\leftarrow \mathrm{fneg}(\mathrm{fquo}(q,a))$; $Q'\leftarrow \mathrm{dpsum}(Q',\mathrm{dptermpr}(K,b,B))$;
$Q'\leftarrow \mathrm{dprem}(Q',\vec{R})$ } } until $Q'=0$.

(4) [Finish.] if $Q=()$ then $Q\leftarrow 0$ else $Q\leftarrow \mathrm{INV}(Q)$; return □

Algorithm 4.4.2

$$Q\leftarrow \mathrm{dpsremu}(P,\vec{A},\vec{R})$$

[Distributive polynomial over a field semi-remainder using univariate ideal elements.

Input: Same as Algorithm 4.4.1.

Output: $Q=\mathrm{srem}(P,<A_1,\ldots,A_n,R_1,\ldots,R_r>).$]

(1) [Initialize.] $Q\leftarrow \mathrm{dprem}(P,\vec{R})$; if $Q=0$ then return.

(2) [Attempt to reduce $\mathrm{lt}(Q)$.] repeat { $\mathrm{DPLECR}(Q;J,q,Q')$;
$\mathrm{BESEL}(J,\vec{A};K,a,B)$; if $K=-1$ then return; $b\leftarrow \mathrm{fneg}(\mathrm{fquo}(q,a))$;
$Q\leftarrow \mathrm{dpsum}(Q',\mathrm{dptermpr}(K,b,B))$; $Q\leftarrow \mathrm{dprem}(Q,\vec{R})$ } until $Q=0$; return □

One will note that these algorithms differ from algorithms 3.4.2 and 3.4.3 (dprem and dpsrem) only in the remainder operations with respect to $\vec{R}$ performed in the initialization and following each reduction of a term with respect to an element of $\vec{A}$. Note also that when we are looking for a possible reduction with respect to $\vec{A}$, we are using remainder selection rules 1 and 2.

We now discuss the computing times of Algorithms 4.4.1 and 4.4.2. Since we have not specified a coefficient field, it is not possible to completely analyze the computing times of these algorithms. However, we can bound the number of exponent vector operations and the number of field operations required. Note that the number of field operations is dominated by the number of exponent vector operations. In cases where the field operations have a computing time codominant with one, such as F=GF(p) with p a single precision prime, this exponent vector bound also gives a bound on the computing times.

Let s be an integer such that for $1 \leq i \leq r$ and $1 \leq j \leq n$, $\partial_{x_i} A_j < s$ and, for R in $\vec{R}$, $\partial_{x_i} R < s$. Further, let t be an integer such that $\partial_{x_i} P < t$. Note that any element of $\vec{A}$ can have at most s^r terms. P and, hence, the initial Q have at most t^r terms. Since the elements of R are univariate, each one can have at most s terms. We concentrate our attention on the remainder operation, dpremu, since the time to compute a semi-remainder is clearly dominated by the time to compute a remainder.

Algorithm 4.4.1 uses remainder selection rule 1 which has the effect of processing the highest ranking terms of Q first. Therefore, once a term of Q is reduced, it cannot be reduced again. In addition, note that any partial reduction of Q leaves it with degree still less than t in all variables.

In the worst case each of the at most t^r terms of P gets reduced by some element of $\vec{A}$. Each of these reductions requires:

(a) Exponent division operations using possibly all elements of $\vec{A}$: at most n exponent operations.

(b) A reduction requiring the subtraction of a multiple of an element of $\vec{A}$. This requires at most $\max(s,t)^r$ exponent vector operations. Since $\max(s,t)$ is codominant with $s+t$, and since $(s+t)^r$ is codominant with s^r+t^r, $\max(s,t)^r$ is also codominant with s^r+t^r.

(c) The remaindering of at most t^r terms of the reduced Q with respect to $\vec{R}$. For each of these terms, we need at most r exponent vector division operations and $\max(s,t^r)$ exponent vector operations for the reduction. Note that $\max(s,t^r)$ is codominant with $s+t^r$.

Combining these results, we find that the number of exponent vector operations required to execute dpremu is dominated by $t^r(n+s^r+t^r+t^r(r+s+t^r))$.

In the portion of algorithm dpremu which reduces intermediate results with respect to $\vec{R}$, we simply employed dprem and ignored the special univariate structure of the elements of $\vec{R}$. If we replace dprem with a special algorithm which makes use of this information, it is possible to obtain a better bound than that found in part c of the analysis above. The central idea of the special algorithm is to consider the polynomial to be remaindered as a polynomial with coefficients in $F[x_i]$ and to only remainder these coefficients with respect to the polynomial $R \in \vec{R}$ which is an element of $F[x_i]$.

We give an informal sketch of such an algorithm below. Here P and Q are defined as in Algorithm 4.4.1, but we assume $\vec{R}=<R_1,\ldots,R_r>$

where $R_i \varepsilon F[x_i]$ and, in addition, R_i is represented as a distributive polynomial in x_i only.

$$Q \leftarrow \text{dpremul}(P, \vec{R})$$

[Distributive polynomial over a field remainder with respect to univariate polynomials.]

(1) $Q \leftarrow P$; if $Q=0$ then return.

(2) For $i=1,\ldots,r$ repeat steps 3 through 9.

(3) Permute the components of the exponent vectors of Q such that x_i is the least major variable.

(4) Sort Q into lexicographical order.

(5) Represent Q as a distributive polynomial with coefficients in $F[x_i]$.

(6) Q is now a list $(I_k,q_k,\ldots,I_0,q_0)$ where q_j is an element of $F[x_i]$ and I_j is an r-1 variable exponent vector. We now reduce each q_j with respect to R_i.] $Q' \leftarrow Q$; $Q \leftarrow ()$; repeat { DPLECR(Q';I,q,Q'); $q \leftarrow \text{dprem}(q,<R_i>)$; if $q \neq 0$ then $Q \leftarrow \text{COMP}(q,I,Q)$ } until $Q'=0$.

(7) if $Q=()$ then { $Q \leftarrow 0$; return }; $Q \leftarrow \text{INV}(Q)$.

(8) Represent Q as a distributive polynomial with coefficients in F.

(9) Permute the components of the exponent vectors of Q such that x_i is again the i-th variable.

(10) When Q has been reduced with respect to each of the elements of $\vec{R}$, sort the terms of Q into the original exponent vector order and return □

This algorithm can be analyzed as follows. As in the analysis of dpremu, we assume s and t are integers such that $\partial_{x_i} R_i < s$ and $\partial_{x_i} P < t$ for $1 \leq i \leq r$.

The major contribution to the computing time of this algorithm comes from the loop from step 3 through 9 which is repeated at most r times. The only effect a given iteration of this loop can have on Q is possibly to reduce the degree of Q in one or more variables. Thus during any iteration Q can have at most t^r terms.

In steps 3, 5, 7, 8, and 9, we must look at each term of Q, hence each of these steps requires at most t^r exponent vector operations. As discussed in Section 4.2 the terms of Q can be sorted in step 4 (and step 10) in at most $rt^r \log(t)$ exponent vector operations using a merge sort.

In step 6 Q is represented as a polynomial in r-1 variables and hence has at most t^{r-1} terms. The coefficients of these terms are polynomials in x_i and hence have at most t terms each. These coefficients all must be remaindered with respect to R_i, a polynomial with at most s terms. The reduction of a given coefficient term requires one exponent vector division and at most max(s,t) exponent vector addition operations. Hence step 6 requires at most $t^{r-1}t(1+\max(s,t))$ exponent vector operations, which is codominant with $t^r(s+t)$.

Combining the results for steps 4 and 6, we see that the number of exponent vector operations required for the loop from step 3 through step 9 is dominated by $r(rt^r \log(t)+t^r(s+t))$ which is

$r^2t^r\log(t)+rst^r+rt^{r+1}$. This compares with the bound of $t^r(r+s+t^r)$ found above for a straightforward application of dprem instead of dpremul.

Applying this result to the bound for dpremu, we find that the number of exponent vector operations required by dpremu is dominated by $t^r(n+s^r+t^r+r^2t^r\log(t)+rst^r+rt^{r+1})$ which is dominated by $t^r(n+s^r+r^2t^r\log(t)+rst^r+rt^{r+1})$.

We can now give the modified complete basis algorithm.

Algorithm 4.4.3

$\vec{C} \leftarrow$ mckbasisu($\vec{A}$,$\vec{R}$)

[Polynomial over a field ideal basis, monic complete kernel basis using univariate ideal elements.

Input: $\vec{A}$ and $\vec{R}$ are as in Algorithm 4.4.1.

Output: $\vec{C}=<C_1,\ldots,C_s>\varepsilon(F[x])^s$, a monic complete kernel basis for $\overline{A}$. $\partial C_1<\ldots<\partial C_s$.]

(1) [Include elements of $\vec{R}$ in $\vec{A}$ and initialize consensus list.]
$\vec{C}\leftarrow\vec{A}$; $\vec{R}'\leftarrow\vec{R}$; repeat { ADV($\vec{R}'$;B,$\vec{R}'$); $\vec{C}\leftarrow$EVLIN($\vec{C}$,B,1) } until $\vec{R}'=()$;
$L\leftarrow()$; ADV($\vec{C}$;$\vec{A}$,$\vec{A}'$); while $\vec{A}'\neq()$ do { $L\leftarrow$CLGEN(L,$\vec{A}$,$\vec{A}'$); ADV($\vec{A}'$;$\vec{A}$,$\vec{A}'$) }

(2) [Select consensus pair.] if L=() then go to 8; CLSEL(L;L,K,A,B).

(3) [Form consensus and semi-remainder.] DPLECR(A;I,a,A′);
I′$\leftarrow$EVDIF(K,I); a$\leftarrow$fneg(a); DPLECR(B;J,b,B′); J′$\leftarrow$EVDIF(K,J);
$\vec{C}\leftarrow$dpsum(dptermpr(I′,b,A′),dptermpr(J′,a,B′));
$\vec{C}\leftarrow$dpsremu($\vec{C}$,$\vec{C}$,$\vec{R}$).

(4) [Delete if possible.] if I′=() then { $\vec{C}\leftarrow$LDEL($\vec{C}$,$\vec{A}$); L$\leftarrow$CLDEL(L,A) }.

(5) [C is constant?] if C=0 then go to 2; if DPLEV(C)=() then to to 7.

(6) [Modify consensus list and basis.]
L$\leftarrow$CLGEN(L,$\vec{C}$,$\vec{C}$); $\vec{C}\leftarrow$EVLIN($\vec{C}$,$\vec{C}$,1); go to 2.

(7) [Identity element is in ideal.] $\vec{C}\leftarrow$LIST1(dpmon($\vec{C}$)); return.

(8) [Make $\vec{C}$ into monic kernel basis.] ADV($\vec{C}$;$\vec{C}$,$\vec{B}$); $\vec{C}\leftarrow$LIST1(dpmon($\vec{C}$));
while $\vec{B}\neq()$ do { ADV($\vec{B}$;$\vec{C}$,$\vec{B}$); C′$\leftarrow$dpremu(DPRD($\vec{C}$),$\vec{C}$,$\vec{R}$);
C$\leftarrow$DPLT(C); if C′≠0 then SRED(RED(C),C′); C$\leftarrow$dpmon(C);
$\vec{C}\leftarrow$EVLIN($\vec{C}$,$\vec{C}$,1) }; return □

The only differences between this algorithm and Algorithm 3.6.4 (mckbasis) occur in steps 1, 3, and 8. In step 1, we include the elements of $\vec{R}$ in $\vec{A}$. This is necessary because we are using $\vec{R}$ in the in the remainder operation. In steps 3 and 8, we replace the use of dpsrem and dprem by dpsremu and dpremu, respectively, thus limiting the degree growth of the remainder polynomials. Consensus selection rules 1, 2, and 3 are applied as they are in Algorithm 3.6.4. Since $\vec{C}$ is a semi-kernel basis at the beginning of step 8, we need only remainder the reductum of each basis element with respect to the rest of the basis. We treat this situation explicitly in Algorithm 4.4.3 because it is possible for a polynomial, say C, to be both an element of $\vec{R}$ and $\vec{C}$. In this case, C would remainder to zero with respect to $\vec{R}$. However, since rd (C) cannot be reduced with respect to C, we can form the remainder of C with respect to the rest of the basis by computing lt(C)+dpremu(rd(C),$\vec{C}$,$\vec{R}$).

The termination of Algorithm 4.4.3 depends, as does Algorithm 3.6.4, on the ascending chain condition for Noetherian rings. The polynomial C added to the basis in step 6 is a semi-remainder with respect to $\vec{C}$, but also a remainder with respect to $\vec{R}$. However, when the algorithm terminates, the consensus of all pairs of elements of $\vec{C}$ remainder to zero with respect to $\vec{C}$, and hence $\vec{C}$ is complete.

We now discuss the computing time of Algorithm 4.4.3. Since we have not specified a coefficient field, it is not possible to analyze

the computing time for this algorithm completely. However, as discussed for dpremu, we can bound the number of exponent vector operations required.

Let m be an integer such that for $1 \leq i \leq r$ and $1 \leq j \leq n$, $\partial_{x_i} A_j < m$ and $\partial_{x_i} R_i < m$. Note that any element of $\vec{A}$ or $\vec{R}$ has at most m^r terms.

Let C be an element of the semi-kernel basis computed in Algorithm 4.4.3. Since $\partial R_i \not| \partial C$ we have at most m^r choices for ∂C. However, if C' is also an element of the same basis, then $\partial C' \not| \partial C$. Hence, we have at most m^{r-1} choices for the leading exponent vectors in such a basis. Therefore, the final semi-kernel (and hence kernel) bases must have at most m^{r-1} elements.

Now, if polynomial C is added to the basis in step 6, and if C' is added later then $\partial C' \neq \partial C$. Hence, since C is a kernel polynomial with respect to $\vec{R}$, at most m^r elements can be added to the basis. Thus, the total number of elements in $\vec{C}$ during the course of the algorithm is bounded by $n+r+m^r$. Hence, the number of consensus operations performed is bounded by $\binom{m+r+m^r}{2}$ which is codominant with $(n+r+m^r)^2$. If we assume $m \geq 2$, then r is dominated by m^r, and the number of consensus operations is dominated by n^2+m^{2r}. Note that this formula dominates the number of times the loop from step 2 to step 6 is executed.

Since we are employing consensus selection rule 2 in Algorithm 4.4.3, the consensus pair selected in step 2 will result in the deletion of an element from $\vec{C}$ if that is at all possible. If the number of

elements in an intermediate basis is greater than m^{r-1}, then a deletion will always be possible. Hence, the number of elements in any intermediate basis may not exceed $\max(n+r,m^{r-1})$ which is codominant with $n+r+m^{r-1}$. If we assume $m \geq 2$, then r is dominated by m^{r-1}, and the number of elements in any intermediate basis is dominated by $n+m^{r-1}$. This, in turn, implies that the length of the consensus list at any point is dominated by $\binom{n+m^{r-1}}{2}$ which is codominant with $n^2+m^{2(r-1)}$.

Let us now consider the computing time of each of the steps of Algorithm 4.4.3. We assume classical algorithms for polynomial arithmetic and linear list representations for list insertion and deletion.

Step 1. We must first insert r items in a list of length at most n+r. To form the consensus list, we must insert $\binom{n+r}{2}$ items in a list of length $\binom{n+r}{2}$. Hence, step 1 requires time dominated by $r(n+r)+\binom{n+r}{2}^2$ which is dominated by $(n+r)^4$.

Step 2. Assuming the consensus list is maintained in the proper order, the time to select the next consensus pair is codominant with one.

Step 3. The degrees of A and B in any variable are less than m, hence the degree of the consensus, C, in any variable is less than 2m. The number of exponent vector operations required to compute C is dominated by m^r.

To bound the number of exponent vector operations required to execute dpsremu, we apply the results obtained above for dpremu with s=m, t=2m, where n is now $n+m^{r-1}$, the maximum number of elements in $\vec{C}$.

We assume that the version of dpremu under consideration uses the special univariate remainder algorithm dpremul. The number of exponent vector operations required is then dominated by $(2m)^r(n+m^{r-1}+m^r+r^2(2m)^r\log(m)+rm(2m)^r+r(2m)^{r+1})$. This, in turn, is dominated by $n(2m)^r+r^2(2m)^{2r}\log(m)+r(2m)^{2r+1}$. This quantity clearly dominates the total number of operations needed for step 3.

Step 4. If a deletion is possible, we must traverse at most the current basis and the current consensus list. The time required for this is dominated by $n^2+m^{2(r-1)}$.

Step 5 requires time codominant with one.

Step 6. Insertion in the basis requires at most $n+m^{r-1}$ exponent vector operations. To update the consensus list, we must pair C with each member of the basis and insert each pair on the consensus list. This pairing and insertion requires at most $(n+m^{r-1})\cdot(n^2+m^{2(r-1)})$ exponent vector operations. This bound is codominant with $n^3+m^{3(r-1)}$.

Step 7 requires time codominant with one.

Step 8. At the beginning of step 8, C is a semi-kernel complete basis for $\overline{A}$, and hence contains at most m^{r-1} elements of degree less than m in all variables. To bound the number of exponent vector operations required for each application of dpremu, we apply the results obtained above for dpremu with s=m, t=m, and $n=m^{r-1}$. We again assume the version of dpremu under consideration uses the special univariate remainder algorithm dpremul. The number of exponent vector operations

is then dominated by $m^r(m^{r-1}+m^r+r^2m^r\log(m)+rm^{r+1}+rm^{r+1})$. This, in turn, is dominated by $r^2m^{2r}\log(m)+rm^{2r+1}$. There will be fewer than m^{r-1} basis elements, so the total number of exponent vector operations required by step 8 is dominated by $r^2m^{3r-1}\log(m)+rm^{3r}$.

The number of exponent vector operations required for a single execution of the loop between steps 2 and 6 is dominated by the number of exponent vector operations required for steps 3 and 6. Using the bound on the number of loop executions derived above, we see that the number of exponent vector operations required for all executions of the loop is dominated by $(n^2+m^{2r})(n(2m)^r+r^2(2m)^{2r}\log(m)+r(2m)^{2r+1}+n^3+m^{3(r-1)})$. This quantity clearly dominates the number of exponent vector operations required by steps 1, 2, and 8. Hence, it provides a bound on the number of exponent vector operations required by Algorithm 4.4.3.

For the bivariate case to be discussed in the next section r=2. In this case the bound on Algorithm 4.4.3 becomes $(n^2+m^4)(nm^2+m^4\log(m)+m^5+n^3+m^3)$. This bound is dominated by $(n^2+m^4)(n^3+m^5)$.

4.5 Resultant Systems and Complete Bases

We are now in a position to combine the results of the previous three sections. In this section, we give an algorithm to compute a complete basis using the resultant systems of the cofactors of the initial basis for the case of bivariate polynomials over a field. The number of exponent vector operations required by this algorithm is dominated by a polynomial function of the degrees of the polynomials in the initial basis whenever the number of basis elements is fixed.

We begin by giving an informal description of the algorithm.

Algorithm 4.5

$$C \leftarrow \text{mckbasisr}(\vec{A})$$

[Monic complete kernel basis using resultant systems. $\vec{A}=\langle A_1,\ldots,A_n\rangle$ is a basis for ideal $\overline{A}$ in $F[x]=F[x_1,x_2]$, F a field. $\vec{C}$ is the unique monic complete kernel basis for the ideal generated by $\vec{A}$.]

(1) [Find gcd and cofactors.] Compute $G=\gcd(A_1,\ldots,A_n)$ and $\vec{B}=\langle B_1,\ldots,B_n\rangle$ where $B_i=A_i/G$, $1\le i\le n$; remove zero elements from $\vec{B}$; if for some i, $B_i \epsilon F-\{0\}$, then $\vec{C}\leftarrow \text{dpmon}(B_i)$ and return.

(2) [Find univariate elements in ideal generated by $\vec{B}$.] For j=1,2 do { $\vec{R}_j=\{B_i \epsilon \vec{B}: \partial_{x_j} B_i=0\}$; if $\vec{R}_j$ is empty, then set $\vec{R}_j$ to the resultant system of $B_1,\ldots,B_n$ with respect to x_j; let R_j be the element of $\vec{R}_j$ with minimum degree in x_j }; $\vec{R}\leftarrow\langle R_1\rangle$; $\vec{R}\leftarrow \text{EVLIN}(\vec{R},R_2,1)$.

(3) [Compute monic complete kernel basis for the ideal generated by $\vec{B}$.] $\vec{D} \leftarrow$ mckbasisu($\vec{B}$,$\vec{R}$).

(4) [Use gcd to find $\vec{C}$. Assume $\vec{D}=<D_1,\ldots,D_s>$.] Set $\vec{C}=<C_1,\ldots,C_s>$ where $C_i=G\cdot D_i$, $1\leq i\leq s$; return □

In this algorithm, we have used Lemma 2.8.2 which allows us to remove the gcd of the elements of $\vec{A}$ before computing the complete basis. Note that for the results of the previous section to apply, $\vec{R}$ in step 3 must be composed of univariate polynomials. By Corollary 4.3.1, R_1 is a non-zero polynomial in x_2 alone and R_2 is univariate in x_1. Hence, in this case, the input conditions for Algorithm 4.4 are satisfied and we have a polynomial bound on the number of exponent vector operations required in step 3 of Algorithm 4.5.

For the rest of this section, let us assume $F[x]=GF(p)[x_1,x_2]$, where again GF(p) represents the finite field with p elements, p a fixed prime. Arithmetic operations in this field have a time codominant with one. Given this situation, we can proceed to analyze the computing time of Algorithm 4.5.

Let k be an integer such that $\partial_j A_i<k$ for $1\leq i\leq n$ and $j=1,2$. Then, according to Section 4.2, the computing time for step 1 is dominated by nk^4. Furthermore, the degrees of the B_i's in x_1 and x_2 are less than k.

In order to compute the two resultant systems in step 2, we need to arrange for first x_1 and then x_2 to be the main variables in the B_i's. These tasks can be easily accomplished in time dominated by nk^2.

The time to compute each resultant system, according to Section 4.3, is dominated by $n^3 2^{n+1} k^{2(n+1)} \log(k)$ which dominates the time required to select the element of minimal degree from the resultant system. Further, note that $\text{res}_{x_1}(U,V)$ in Section 4.3 is the determinant of a matrix with fewer than 2k rows and 2k columns, each of whose entries contains polynomials in x_2 whose degrees are less than k. Hence, the degrees of the elements of any resultant system are bounded by $2k^2$.

This bound on the degrees of the elements of $\vec{R}$ makes it possible to apply the results of Section 4.4 to the computing time of step 3 of Algorithm 4.5. We thus have $m=2k^2$ to be inserted in the time bound $(n^2+m^4)(n^3+m^5)$. This gives a bound of $(n^2+k^8)(n^3+k^{10})$. Since the coefficient field is finite, this bound on exponent vector operations required by step 3 is also a bound on the computing time required for step 3.

The computing time for step 4 of Algorithm 4.5 is determined by the time required to multiply s polynomials of degree less than $2k^2$ by a polynomial of degree less than k, where $s<(2k^2)^2$. The time to perform this operation is dominated by k^{10} which is less than the time required for step 3.

Finally, we have that the computing time for Algorithm 4.5 when $F[x]=GF(p)[x_1,x_2]$ is dominated by $n^3 2^{n+1} k^{2(n+1)} \log(k)+(n^2+k^8)(n^3+k^{10})$. For any fixed number of initial polynomials, we have a complete basis algorithm whose computing time is a polynomial function of the degrees of the initial polynomials.

CHAPTER FIVE:

Empirical Results

5.1 Introduction

In conjunction with the theoretical analyses described in the previous chapters, several empirical studies were conducted to compare various implementations of the remainder and complete basis algorithms. In particular, we were interested in investigating how changes in the remainder and consensus selection rules affect the computing times of the associated algorithms. The data presented here do not represent comprehensive comparisons of these alternatives, but they do indicate some of the characteristics of each.

Section 5.2 describes timing results for the remainder algorithm using two possibilities for remainder selection rule 2. Section 5.3 deals with three possible modifications of the consensus selection rules; it also compares the computing times for finding a complete basis using lexicographic order and using total degree order for the exponent vector ordering. Section 5.4 concludes with a presentation of timing results for the complete basis algorithm described in Section 4.5 which makes use of resultant systems.

The algorithms in this thesis were implemented as part of the SAC-2 system for Symbolic and Algebraic Computation (see Appendices A and B). The measurements reported in this chapter were made on a Digital Equipment Corporation VAX-11/780 computer running under the VAX/VMS operating system. The VAX-11 has a maximum memory access time of 1.8 μs, but an average memory access time of .3 μs.

The timing runs were made in a multiprogramming environment. Hence, in an attempt to eliminate timing effects due to the load of the computer, the results presented here are the average of several runs. It was observed that the timings were reproducible to within 5 to 10 percent.

The "random" polynomials used as input for the computations in this chapter were generated as follows. Let $F[x_1,\ldots,x_r]$ be the polynomial ring in which we wish to generate a random polynomial, A, such that $\partial A=I$, for some specified $I\varepsilon E=\{<e_1,\ldots,e_r>: e_i\geq o, 1\leq i\leq r\}$. Let $E_T(I)=\{J\varepsilon E: J<I\}$ where $<$ represents total degree ordering. Let $E_L(I)=\{J\varepsilon E: J<I \text{ and } J|I\}$ where now $<$ stands for lexicographic ordering. Note that both of these sets of exponent vectors are finite. Let $P_T(I)=\{a_I x^I+\sum_{J\varepsilon E_T(I)} a_J x^J: a_I\neq o, a_I, a_J\varepsilon F\}$, and let $P_L(I)$ be defined similarly with $E_L(I)$ substituted for $E_T(I)$. A randomly selected element $A\varepsilon P_T(I)$ will be called a random polynomial of degree I with respect to total degree ordering. A randomly selected element of $P_L(I)$ will be called a random polynomial of degree I with respect to lexicographic ordering. For the computations reported here, we set r=2 and F=GF(p) where $p\approx 2^{28}$. This choice of p implies that the random polynomials generated will be dense in the associated orderings, i.e., that nearly all exponent vectors in $E_T(I)$ or $E_L(I)$ will be present in a generated polynomial whose leading exponent vector is I.

5.2 Remainder Algorithm Comparisons

In forming the remainder of a polynomial P with respect to basis $\vec{A}$, we must first choose a term of P to reduce and then we must choose an element of $\vec{A}$ with which to reduce it. Remainder selection rule 1 of Section 3.4 specifies that we always choose to remainder the term of P which is the largest possible with respect to the exponent vector ordering. This rule has the advantage that once a term is reduced, it can never be reduced again.

In Section 3.4, we discussed two possibilities for selecting the element of $\vec{A}$ to be used at a given step of the remainder algorithm. We finally formulated remainder selection rule 2 which specified that we choose the element of $\vec{A}$ whose leading exponent vector is the minimum possible. The other obvious alternative is to select the element of $\vec{A}$ whose leading exponent vector is maximum. Using the notation of Section 3.4, we formalize this selection rule.

Remainder Selection Rule 2´(RSR2´) Given $J \varepsilon D_i$, let $I=\max\{\partial A_j : 1 \le j \le n \text{ and } \partial A_j | J\}$, and let $k=\min\{j : \partial A_j = I\}$. Then $\Theta(q_J, \vec{a}_J)=\langle b_1,\ldots,b_n\rangle$ where $b_k=q_J/a_k$ and $b_j=0$ for $j \neq k$.

This selection rule can be implemented as follows. In algorithms BESEL, dprem, and dpsrem of Section 3.4 (Algorithms 3.4.1, 3.4.2, and 3.4.3) change the input specifications to read $\partial A_1 \ge \partial A_2 \ge \ldots \ge \partial A_n$. In algorithm BESEL, step number 2, remove the statement "if EVCOMP(I,J)=-1 then return". These changes cause the basis $\vec{A}$ to be searched from the largest element to the smallest.

The remainder algorithms based on RSR2 and RSR2′ were tested as follows. Since the coefficient field was GF(p), we used algorithm DDMPRM of Appendix B to realize abstract algorithm dprem (Algorithm 3.4.2). The remainder selection rules were realized in two versions of DDBESL which realize the abstract algorithm BESEL (Algorithm 3.4.1).

For a fixed n, we generated a basis $\vec{A}=<A_1,\ldots,A_n>$ and a polynomial P in GF(p)[x,y]. For the case of lexicographic exponent vector ordering, ∂A_i for $1\leq i\leq n$ was selected randomly from $E_L((4,4))-E_L((1,1))$; A_i was then selected from $P_L(\partial A_i)$. For each basis, a polynomial P was selected from $P_L((10,10))$. For the case of total degree exponent vector ordering, ∂A_i was chosen from $E_T((3,3))-E_T((1,1))$, and A_i was selected from $P_T(\partial A_i)$; P was selected from $P_T((7,7))$. Each polynomial/basis pair was input to DDMPRM based on RSR2 and then input to a version of DDMPRM based on RSR2′.

The results of these tests are presented in Tables 5.2.1 and 5.2.2. Each entry is the average of ten separate polynomial/basis pairs. The times are in seconds.

Table 5.2.1 Remainder Computing Times (DDMPRM, Lexicographic Order)

n	RSR2	RSR2′
6	2.54	6.18
20	2.24	7.29

Table 5.2.2 Remainder Computing Times (DDMPRM, Total Degree Order)

n	PSR2	RSR2´
6	3.61	6.59
20	3.48	6.73

From these tables we observe the following. Remainder selection rule 2 consistently performed significantly better than rule 2´, thus reinforcing our decision to use RSR2 in the algorithms of Chapter 3. Also, note that as the number, n, of polynomials in the basis increased, the time required by RSR2 decreased while that for RSR2´ increased. This may be directly related to the importance of reducing a polynomial with respect to the smallest basis elements first. If there are small elements in the basis, then RSR2´ requires us to test all the larger elements of the basis before the smaller elements can be used.

5.3 Complete Basis Algorithm Comparisons

In Section 3.6, three consensus selection rules were specified during the formulation of the complete basis algorithm (Algorithm 3.6.4). The first two deal with the specification of the modified consensus operation and with the possibility of deleting basis elements during the course of the algorithm. Consensus selection rule 3, however, specified the order in which consensus pairs are to be chosen. Let L_1 be the deletion consensus list of a basis (see Section 3.6), and let L_2 be the list containing all other consensus pairs for the same basis. Consensus selection rule 3 requires us to form the consensus of the consensus pair (A,B) in L_1 such that lcm ($\partial A, \partial B$) is minimum. If L_1 is empty, then we look for such a pair on L_2.

Two alternatives to this rule are possible. First, we could choose the pair (A,B) such that lcm ($\partial A, \partial B$) is a maximum, and second, we could use the deletion consensus list last instead of first. Using the notation of Section 3.6, we formalize these selection rules as follows.

Consensus Selection Rule 3´(CSR3´) If L_1 is non-empty, then let $L' = L_1$ else let $L' = L_2$. Let (A,B) be the first element of L´ such that $J = \text{lcm}(\partial A, \partial B) = \max \{ \text{lcm}(\partial A', \partial B') : (A', B') \in L' \}$. The next consensus to be formed will be cons (A,B).

Consensus Selection Rule 3´´(CSR3´´) If L_2 is non-empty, then let $L' = L_2$ else let $L' = L_1$. Let (A,B) be the first element of L´ such that $J = \text{lcm}(\partial A, \partial B) = \min \{ \text{lcm}(\partial A', \partial B') : (A', B') \in L' \}$. The next consensus to be formed will be cons(A,B).

Thus, CSR3′ chooses the maximum lcm(∂A,∂B) first in L_1 then in L_2. CSR3′′ chooses the minimum lcm(∂A,∂B) in L_2 first and then in L_1, effectively postponing deletions.

Consensus selection rule 3′ may be implemented by a change to algorithm CLGEN (Algorithm 3.6.1) which generates the consensus list. In step 2, it is only necessary to change the third parameter in each of the two calls to EVLIN from 1 to -1. This ensures that the sublists of the consensus lists L and L′ are arranged such that the least common multiples of the leading exponent vectors of the consensus pairs are in non-increasing order. Thus, algorithm CLSEL will select the consensus pair (A,B) such that lcm(∂A,∂B) is a maximum.

Consensus selection rule 3′′ can be implemented by making sure that algorithm CLSEL (Algorithm 3.6.2) always selects consensus pairs from the non-deletion consensus list first. This is accomplished by interchanging the references to L_1 and L_2 in step 2 of Algorithm 3.6.2.

The complete basis algorithms based on CSR3, CSR3′, and CSR3′′ were tested as follows. Since the coefficient field was GF(p), we used algorithm DDMBCK of Appendix B to realize abstract algorithm mckbasis (Algorithm 3.6.4). The consensus selection rules were realized in versions of algorithm DDBCLG (for abstract algorithm CLGEN) and algorithm DDBCLS (for CLSEL). Both of these algorithms are given in Appendix B.

We restricted our input to the complete basis algorithm to bases in GF(p)[x,y] which contained two basis elements with the same leading exponent vector. A bivariate exponent vector, I, was specified, and the basis elements A_1 and A_2 were selected at random from $P_L(I)$ for

tests conducted using lexicographic exponent vector ordering, and from $P_T(I)$ for tests using total degree ordering. Each of these bases was then input to the three versions of DDMBCK based on CSR3, CSR3′, and CSRC′′.

The results of these tests are presented in Tables 5.3.1 and 5.3.2. Each entry is the average of five different initial bases. The times are in seconds. Dashes indicate times in excess of 200 seconds.

Table 5.3.1 Complete Basis Computing Times (DDMBCK, Lexicographic Order)

I	CSR3	CSR3′	CSR3′′
(2,2)	1.11	1.10	102.8
(3,3)	9.71	10.18	-
(4,4)	70.6	70.1	-

Table 5.3.2 Complete Basis Computing Times (DDMBCK, Total Degree Order)

I	CSR3	CSR3′	CSR3′′
(2,2)	2.97	7.13	2.95
(3,3)	20.2	85.0	20.7
(4,4)	86.8	-	86.7

From these tables, we note that the time required to compute a complete basis with any of these algorithms rises rapidly as the degrees of the initial basis elements increase. But note, however, that there are definite distinctions among these three versions. If complete bases are being computed with exponent vectors in lexicographic order, then CSR3 and CSR3′ yield virtually identical computing times. This indicates that in this case it makes little difference in

what order the consensus pairs are evaluated, as long as consensus pairs which lead to deletions are processed first. Note that if deletions are processed last (CSR3''), then the time to compute a complete basis with lexicographic order becomes much longer.

Nearly the opposite behavior is seen if we are interested in computing complete bases with total degree exponent vector ordering. In this case, it seems to make little difference whether deletions are processed first or last (CSR3 and CSR3''), but it seems important to first process consensus pairs whose leading exponent vector least common multiple is small. Although these tests are not exhaustive, they do support the use of consensus selection rule 3 as a rule which handles both exponent vector orderings in a satisfactory manner.

In Section 2.2, we mentioned that two different exponent vector (or monomial) orderings will produce two distinct sets of residue class representatives when the remainder algorithm is utilized with a complete basis to give a simplifying ample function. However, we also indicated that even when the actual residue class representatives are unimportant, the choice of the ordering used during the complete basis computation can have a pronounced effect on the computing time.

This phenomenon is illustrated in Table 5.3.3. The times in this table were obtained by generating initial bases containing two random polynomials in GF(p)[x,y] and using mckbasis with CRS3, as realized by algorithm DDMBCK, to compute a complete basis. The source of the two polynomials is given under the column headed "Initial." The times (in seconds) required to compute complete bases using lexicographic and

total degree exponent vector orderings are given in the final two columns. As before, each of these entries is the average of five complete basis computations.

Table 5.3.3 Complete Basis Computing Times (DDMBCK, Selection Rule CSR3)

I	Initial	Lexicographic	Total Degree
(2,2)	$P_L(I)$	1.11	1.55
(3,3)	$P_L(I)$	9.71	16.5
(4,4)	$P_L(I)$	70.6	89.9
(2,2)	$P_T(I)$	3.48	2.97
(3,3)	$P_T(I)$	53.12	20.2
(4,4)	$P_T(I)$	822.5	86.8

5.4 Complete Basis Computations Using Resultant Systems

In this section, we present some empirical results regarding the computing time of the complete basis algorithm (mckbasisr, Algorithm 4.5), which makes use of multipolynomial resultant systems. For the coefficient field GF(p) this abstract algorithm is realized by algorithm DDMBCR of Appendix B.

Two comments are in order. DDMBCR uses a bubble sort, not a merge sort, when converting between distributive and recursive canonical form. This only affects the time required to remove the ideal basis gcd and compute the associated resultant systems. As will be seen below, the time required for this phase of the process is only a small part of the time required to find a complete basis. In addition, the univariate remainder algorithm used by DDMBCR is just the remainder algorithm presented in Chapter 3, and not the special univariate remainder algorithm, dpremul, described in Section 4.4. It is not known whether the use of dpremul would have a significant effect on the empirical computing times for DDMBCR.

The times listed in Tables 5.4.1 and 5.4.2 were obtained in the same way as the times in Tables 5.3.1 and 5.3.2. We list the times for mckbasis using CSR3 (as realized by DDMBCK) for convenience.

Table 5.4.1 Complete Basis with Resultant System Computing Times (Lexicographic Order)

I	DDMBCK(CSR3)	DDMBCR
(2,2)	1.11	6.49
(3,3)	9.71	67.2
(4,4)	70.6	472.0

Table 5.4.2 Complete Basis with Resultant System Computing Times (Total Degree Order)

I	DDMBCK(CSR3)	DDMBCR
(2,2)	2.97	35.2
(3,3)	20.2	472.0
(4,4)	86.8	-

The computing times required by DDMBCR are much larger than those required by DDMBCK. More information can be gained from Tables 5.4.3 and 5.4.4 which break down the times (in seconds) required by DDMBCR into three categories. The column labeled "Prep" gives the time necessary to "prepare" the initial basis, i.e., to compute the gcd and the resultant system of the cofactors. The column marked "Comp" gives the time actually required to compute a complete basis for the ideal generated by the cofactors of the initial basis elements. This portion of the algorithm makes use of the univariate ideal elements computed from the resultant systems. The column marked "Post" gives the time necessary to multiply the elements of the complete basis by the gcd of the initial basis elements. The total times do not correspond exactly to the sum of the partial times because of rounding.

Table 5.4.3 Complete Basis with Resultant System Component Computing Times (Lexicographic Order)

I	Prep	Comp	Post	Total
(2,2)	1.41	5.06	.02	6.49
(3,3)	4.23	62.88	.30	67.15
(4,4)	10.31	461.41	.55	471.77

Table 5.4.4 Complete Basis with Resultant System Component Computing Times (Total Degree Order)

I	Prep	Comp	Post	Total
(2,2)	4.88	30.3	.04	35.23
(3,3)	22.17	450.04	.11	472.31

The pairs of random polynomials used as inputs had a gcd of one. Hence, most of the preparation time was spent in computing resultant systems. The post time was small in every case because it was only necessary to multiply each of the final basis elements by one. The majority of the computing time was spent finding a complete basis using the univariate elements computed from the associated resultant systems. The frequent attempts to reduce polynomials with respect to the univariate ideal elements probably contributes a great deal to the magnitude of this figure.

CHAPTER SIX:

Conclusions

The work reported in this thesis is related most closely to that of Buchberger, Shtokhamer, Lauer, and Trinks. In [BUC65] and [BUC70], Buchberger first detailed an algorithm to compute what in this thesis we have been calling a complete basis. His work dealt with multivariate polynomials over a field, and only considered the total degree exponent vector ordering. A short while later, Shtokhamer [SHT76] formulated an algorithm for computing complete bases in rings of univariate polynomials over coefficient rings which are very much like the simplification rings defined in Chapter 2. In [LAU76a] and [LAU76b], Lauer recognized that both of these approaches could be used to specify simplifying ample functions in the associated rings. He also generalized Buchberger's algorithm to coefficient rings that are Euclidean domains. Trinks [TRI78] synthesized this previous work in an algorithm for computing complete bases for ideals composed of multivariate polynomials whose coefficients were elements of a ring that was similar to a simplification ring.

Both Buchberger and Trinks call complete bases Gröbner bases. Lauer refers to them as Buchberger bases, and Shtokhamer used the phrase Szekeres basis. The remainder operation described in this thesis is similar to the M-reduction used by Buchberger, Lauer, and Trinks. The polynomials formed by the consensus operation have been variously described as S polynomials [BUC70], S and T polynomials [LAU76b], and S_j^{τ} polynomials [TRI78].

In the work of Buchberger, Lauer, and Trinks, a basis is defined to be complete if the M-reductions of all elements of the associated ideal are zero. This relates the completeness of the basis directly to the specification of an algorithm. In 1976, however, G. E. Collins suggested the concept of simple constructibility for multivariate polynomials over a field. In Section 2.3 of this thesis, we have generalized this concept to the case of multivariate polynomials over a simplification ring. The concept of simple constructibility thus abstracts that property of a basis which makes it a complete basis. The formulation of the remainder and complete basis algorithms in terms of simple constructibility occupies most of Chapters 2 and 3.

Perhaps one of the main contributions of this thesis is the application of greatest common divisor and resultant system calculations to the problem of computing complete bases. In Section 2.8, we show that the gcd of the basis elements may be removed before computing a complete basis. In Chapter 4, we show how the computation of multipolynomial resultant systems can result in univariate polynomials which can then be used to limit degree growth during the computation of complete bases. This technique resulted in a polynomial time bound on an algorithm for computing complete bases in GF(p)[x,y].

APPENDIX A:

Algorithm Index

A.1 Introduction

In this appendix we present an index of the algorithms discussed in the body of this thesis. In addition we show the correspondence between the algorithms in the thesis and the algorithms as they have actually been realized in the SAC-2 computer algebra system. Aldes descriptions of the SAC-2 versions may be found in Appendix B. Appendix B also contains Aldes descriptions of several algorithms not mentioned in the thesis. These algorithms support the thesis algorithms by providing capabilities for polynomial output, sorting, and canonical form conversion, among others. A list of the names of these algorithms is to be found in Section A.3 of this appendix.

In the algorithms presented in Appendix B we have added two special symbols, "$\vec{\ }$" and "∂", to the publication Aldes character set given in [LOS76]. We transliterate the ornament "$\vec{\ }$" as "A" for arrow; "∂" is transliterated as "DG". Thus $\vec{B}$ becomes BA, and ∂A becomes DG(A).

A.2 Index of Thesis Algorithms

Algorithm	Section	SAC-2 Version
BESEL	3.4.1	DDBESL
beta	2.7.2	
cbasis	2.5	
cbasisx	2.6.2	
CLDEL	3.6.3	DDBCLD
CLGEN	3.6.1	DDBCLG
CLSEL	3.6.2	DDBCLS
COMP	1.6	COMP
dpdif	3.2	DDMPDF
DPLC	3.2	DPLC
DPLEC	3.2	DPLEC
DPLECR	3.2	DPLECR
DPLEV	3.2	DPLEV
dpmon	3.2	DDMPMN
dpneg	3.2	DDMPNG
dpprod	3.2	DDMPPR
DPRD	3.2	DPRD
dprem	3.4.2	DDMPRM
dpremu	4.4.1	DDMPRU
dpremul	4.4	
dpsrem	3.4.3	DDMPSR
dpsremu	4.4.2	DDMPSU
dpsum	3.2	DDMPSM
dptermpr	3.2	DDMPTP
EVCOMP	3.2	EVDCMP

Algorithm	Section	SAC-2 Version
EVDIF	3.2	EVDDIF
EVGCD	3.2	EVDGCD
EVLCM	3.2	EVDLCM
EVLIN	3.6	EVDLIN
EVSUM	3.2	EVDSUM
fdif	3.2	MDDIF
FIRST	1.6	FIRST
fneg	3.2	MDNEG
fprod	3.2	MDPROD
fquo	3.2	MDQ
fsum	3.2	MDSUM
LDEL	3.6	LDEL
ldpgcdc	4.2	LDDMPG
ldpres	4.3	see DDMBFV
mckbasis	3.6.4	DDMBCK
mckbasisr	4.5	DDMBCR
mckbasisu	4.4.3	DDMBCU
modgen	2.7.1	
RED	1.6	RED
rem	2.3	
remx	2.6.1	
srem	2.5	
theta	2.6.3	

A.3 Support Algorithms

Algorithm	Description
DDBSRT	Distributive, dense exponent vector, polynomial ideal basis sort.
DDIBWR	Distributive, dense exponent vector, integral polynomial ideal basis write.
DDIPWR	Distributive, dense exponent vector, integral polynomial write.
DDLFPL	Distributive, dense exponent vector, polynomial list from polynomial list.
DDLMVS	Distributive, dense exponent vector, polynomial list, main variable substitution.
DDMBFV	Distributive, dense exponent vector, modular polynomial ideal basis, ideal elements free of main variable.
DDMBRA	Distributive, dense exponent vector, modular polynomial ideal basis, random.
DDMPRA	Distributive, dense exponent vector, modular polynomial, random.
DDPFP	Distributive, dense exponent vector, polynomial from polynomial.
DDPFPL	Distributive, dense exponent vector, polynomial from polynomial, lexicographic order.
DDPMVS	Distributive, dense exponent vector, polynomial, main variable substitution.
DDPSRT	Distributive, dense exponent vector, polynomial sort.
EVDMVS	Exponent vector, dense representation, main variable substitution.

Algorithm	Description
LMPGDC	List of modular polynomials, greatest common divisor and cofactors.
LMPRES	List of modular polynomials, resultant system.
PBCFL	Polynomial base coefficient list.
PFDDP	Polynomial from distributive, dense exponent vector, polynomial.
PLFDDL	Polynomial list from distributive, dense exponent vector, polynomial list.

APPENDIX B:

SAC-2 Algorithms

M←DDBCLD(L,C)

[Distributive, dense exponent vector, polynomial ideal basis consensus list deletion. L is a consensus list, C is a distributive polynomial. M is the consensus list gotten from L by deleting all consensus pairs containing C. L is modified.]

safe DDBCLD.

(1) [L=().] if L=(), then { M←(); return }.

(2) [Scan L_1, then L_2.] FIRST2(L;L_1,L_2); M'←L_1; i←1.

(3) [M'=().] if M'=(), then go to 6.

(4) [C in FIRST(M').] FIRST3(FIRST(M'); K,A,B); if C=A V C=B, then { M'←RED(M'); go to 3 }.

(5) [Scan M'.] L"←M'; L'←RED(L"); while L'≠()
do { FIRST3(FIRST(L');K,A,B); if C=A V C=B, then
SRED(L",RED(L')) else L"←L'; L'←RED(L") }.

(6) [Next List.] if i=1, then { L_1←M'; M'←L_2; i←2; go to 3 };
L_2←M'.

(7) [Return M.] if L_1=()& L_2=() then M←() else M←LIST2(L_1,L_2);
return □

L'←DDBCLG(L,A,$\vec{B}$)

[Distributive, dense exponent vector, polynomial ideal basis consensus list, generate elements. L is a consensus list. A is a non-zero polynomial in distributive, dense exponent vector, canonical form; $\vec{B}=<B_1,\ldots,B_n>$ is an ideal basis composed of non-zero distributive polynomials compatible with A. L' is a new consensus list composed of the elements of L, and, in addition, the consensus pairs (A,B_i), $1\leq i\leq n$, such that $lcm(\partial A,\partial B_i)\neq \partial A+\partial B_i$. The sublists of L and L' are arranged such that the least common multiples of the leading exponent vectors of the consensus pairs are in non-decreasing order. This partially implements consensus selection rules 2 and 3. The list L is modified.]

safe A,B,$\vec{B}'$,I,J.

(1) [Initialize.] if L≠() then FIRST2(L;L_1,L_2) else { L_1←(); L_2←() }; I←DPLEV(A); $\vec{B}'$←$\vec{B}$.

(2) [Pair A with elements of $\vec{B}$.] repeat { ADV($\vec{B}'$;B,$\vec{B}'$); J←DPLEV(B); K'←EVDGCD(I,J); if K'≠() then { K←EVDLCM(I,J); I'←EVDDIF(K,I); J'←EVDDIF(K,J); if J'=() then M←LIST3(K,B,A) else M←LIST3(K,A,B); if I'=() V J'=() then L_1←EVDLIN(L_1,M,1) else L_2←EVDLIN(L_2,M,1) } } until $\vec{B}'$=().

(3) [Return L'.] if L_1=() & L_2=() then L'←() else L'←LIST2(L_1,L_2); return □

DDBCLS(L;L',K,A,B)

[Distributive, dense exponent vector, polynomial ideal basis consensus list, selection. $L=(L_1,L_2)$ is a consensus list, where either L_1 or L_2 is non-empty. A and B, the next consensus pair, are compatible non-zero polynomials in distributive, dense exponent vector, canonical form. $K=lcm(\partial A,\partial B)$. L' is the consensus list equivalent to L with the pair (A,B) removed. This partially implements consensus selection rule 3. The list L is modified.]

safe DDBCLS.

(1) [Initialize.] FIRST2(L;L_1,L_2).

(2) [Choose pair.] if $L_1\neq()$ then ADV(L_1;M,L_1) else ADV(L_2;M,L_2); FIRST3(M;K,A,B).

(3) [Finish.] if $L_1=()$ & $L_2=()$ then $L'\leftarrow()$ else $L'\leftarrow$LIST2(L_1,L_2); return □

DDBESL(J,$\vec{A}$;K,a,B)

[Distributive, dense exponent vector, polynomial ideal basis element selection. J is an exponent vector and $\vec{A}=<A_1,\ldots,A_n>$ is an ideal basis composed of polynomials in distributive, dense exponent vector, canonical form whose leading exponent vectors are compatible with J. For $1\leq j\leq n$, $A_j\neq 0$, and $\partial A_1\leq\partial A_2\leq\ldots\leq\partial A_n$. If $\partial A_j\nmid J$ for $1\leq j\leq n$, then K=-1, and a and B are undefined. Otherwise, let k be the least integer such that $\partial A_k\mid J$. Then $K=J-\partial A_k$, $a=lc(A_k)$, and $B=rd(A_k)$. This implements remainder selection rule 2.]

safe DDBESL.

(1) [Initialize.] $\vec{A}'\leftarrow\vec{A}$; K$\leftarrow$ -1.

(2) [Check each basis element.] repeat { ADV($\vec{A}'$;A,$\vec{A}'$); DPLECR(A;I,a,B); K$\leftarrow$EVDDIF(J,I); if K$\neq$-1 then return; if EVDCMP(J,I)=-1 then return } until $\vec{A}'$=(). return □

$\vec{B} \leftarrow$DDBSRT($\vec{A}$,k)

[Distributive, dense exponent vector, polynomial ideal basis sort. $\vec{A}$ is an ideal basis composed of compatible non-zero polynomials in distributive, dense exponent vector, canonical form. The basis elements are sorted such that their exponent vectors are in non-decreasing (k=0) or non-increasing (k≠0) order. $\vec{B}$ is the sorted basis. The list representing $\vec{A}$ is modified.]

safe DDBSRT.

(1) [Initialize.] $\vec{B} \leftarrow \vec{A}$; $\vec{B}_0 \leftarrow ()$.

(2) [Bubble sort non-decreasing.] repeat { $\vec{B}' \leftarrow \vec{B}$; $\vec{B}'' \leftarrow$RED($\vec{B}'$); A'←FIRST($\vec{B}'$); I'←DPLEV(A'); $\vec{B}_0' \leftarrow ()$; while $\vec{B}'' \neq \vec{B}_0$ do { A"←FIRST($\vec{B}''$); I"←DPLEV(A"); if EVDCMP(I',I")=1 then { SFIRST($\vec{B}'$,A"); SFIRST($\vec{B}''$,A'); $\vec{B}_0' \leftarrow \vec{B}''$ } else { A'←A"; I'←I" }; $\vec{B}' \leftarrow \vec{B}''$; $\vec{B}'' \leftarrow$RED($\vec{B}'$) }; $\vec{B}_0 \leftarrow \vec{B}_0'$ } until $\vec{B}_0 = ()$.

(3) [Finish.] if k≠0 then $\vec{B} \leftarrow$INV($\vec{B}$); return □

DDIBWR($\vec{A}$,V)

[Distributive, dense exponent vector, integral polynomial ideal basis write. $\vec{A}$ is an ideal basis composed of compatible integral polynomials in distributive, dense exponent vector, canonical form. V is a variable list for the polynomials in $\vec{A}$. The polynomials in $\vec{A}$ are written in the output stream. The output buffer is emptied after each polynomial is output.]

safe DDIBWR.

(1) $\vec{A}'\leftarrow\vec{A}$; repeat { ADV($\vec{A}'$;A,$\vec{A}'$); TAB(5); DDIPWR(A,V); EMPTOB }
until $\vec{A}'$=(); return □

DDIPWR(A,V)

[Distributive, dense exponent vector, integral polynomial write. A is an integral polynomial in distributive, dense exponent vector, canonical form. V is a variable list for A. A is written in the output stream.]

```
    safe e,i,s,v,A',I,S,V'.

(1) [A=0.] if A=0 then { CWRITE('0'); return }.
(2) [Output each term.] A'←A; i←0; repeat {   ADV2(A';I,a,A');
    s←ISIGNF(a); if s=-1 then { a←INEG(a); S←'-' } else S←'+';
    if i=1 then CWRIT2(' ',S) else if s=-1 then CWRITE(S); i←1;
    if I=() V a≠1 then { CWRITE(' '); IWRITE(a) };
    V'←V; while I≠() do {  ADV(I;e,I); ADV(V';v,V'); if e>0 then
    { CWRITE(' '); CLOUT(v) }; if e>1 then
    { CWRIT2('*','*'); IWRITE(e) }  }   } until A'=(); return □
```

M←DDLFPL(r,L)

[Distributive, dense exponent vector, polynomial list from polynomial list. L is a list of polynomials in r variables, r≥0, represented in recursive canonical form. M is a list of the polynomials in L in distributive, dense exponent vector, canonical form.]

safe A,L'.

(1) L'←L; M←(); repeat { ADV(L';A,L'); B←DDPFP(r,A); M←COMP(B,M) }
until L'=(); M←INV(M); return □

M←DDLMVS(L,k)

[Distributive, dense exponent vector, polynomial list, main variable substitution. $L=(A_1,\ldots,A_n)$ is a list of compatible polynomials in variables $x_1,\ldots,x_r$ in distributive, dense exponent vector canonical form. $M=(B_1,\ldots,B_n)$ is a list of distributive, dense exponent vector, polynomials. For $1\leq i\leq n$, B_i is gotten from A_i interchanging the main variable x_r with the variable x_k, $1\leq k\leq r$.]

safe A,L'.

(1) L'←L; M←(); repeat { ADV(L';A,L'); M←COMP(DDPMVS(A,k),M) }
until L'=(); M←INV(M); return □

$\vec{C}$←DDMBCK(p,$\vec{A}$)

[Distributive, dense exponent vector, modular polynomial ideal basis, monic complete kernel basis. $A=<A_1,\ldots,A_n>$ is an ideal basis composed of non-zero compatible polynomials over Z_p, p a prime β-integer, in distributive, dense exponent vector, canonical form. $\vec{A}$ is arranged such that $\partial A_1 \leq \partial A_2 \leq \ldots \leq \partial A_n$. $\vec{C}=<C_1,\ldots,C_s>$ is a monic complete kernel basis for the ideal generated by $\vec{A}$ such that $\partial C_1 < \partial C_2 < \ldots < \partial C_s$. This algorithm uses consensus selection rules 1, 2, and 3, and remainder selection rules 1 and 2. The list representing A is modified.]

safe a,b,A,A',A',B,B',I,J.

(1) [Initialize consensus list.] $\vec{C}$←$\vec{A}$; L←(); ADV($\vec{A}$;A,$\vec{A}'$); while $\vec{A}'\neq()$ do { L←DDBCLG(L,A,$\vec{A}'$); ADV($\vec{A}'$;A,$\vec{A}'$) }.

(2) [Select consensus pair.] if L=() then go to 8; DDBCLS(L;L,K,A,B).

(3) [Form consensus and semi-remainder.] DPLECR(A;I,a,A'); I'←EVDDIF(K,I); a←MDNEG(p,a); DPLECR(B;J,b,B'); J'←EVDDIF(K,J); C←DDMPSM(p,DDMPTP(p,I',b,A'), DDMPTP(p,J',a,B')); C←DDMPSR(p,C,$\vec{C}$).

(4) [Delete if possible.] if I'=() then { $\vec{C}$←LDEL($\vec{C}$,A); L←DDBCLD(L,A) }.

(5) [C is constant.] if C=0 then go to 2; if DPLEV(C)=() then go to 7.

(6) [Modify consensus list and basis.] L←DDBCLG(L,C,$\vec{C}$);
$\vec{C}$←EVDLIN($\vec{C}$,C,1); go to 2.

(7) [Identity is in ideal.] $\vec{C}$←LIST1(LIST2(1),1)); return.

(8) [Make $\vec{C}$ into monic kernel basis.] ADV($\vec{C}$;C,$\vec{B}$);
$\vec{C}$←LIST1(DDMPMN(p,C)); while $\vec{B}\neq()$ do
{ ADV($\vec{B}$;C,$\vec{B}$); C←DDMPRM(p,C,$\vec{C}$); C←DDMPMN(p,C);
$\vec{C}$←EVDLIN($\vec{C}$,C,1) ; return □

$\vec{C}$←DDMBCR(p,$\vec{A}$)

[Distributive, dense exponent vector, modular polynomial ideal basis, monic complete kernel basis using resultant systems. $\vec{A}=<A_1,\ldots,A_n>$ is an ideal basis composed of compatible polynomials over Z_p, p a prime β-integer, in distributive, dense exponent vector, canonical form. $\vec{C}=<C_1,\ldots,C_s>$ is a monic complete kernel basis for the ideal generated by $\vec{A}$ such that $\partial C_1<\ldots<\partial C_s$. This algorithm uses consensus selection rules 1, 2, and 3. The list representing $\vec{A}$ is modified.]

safe r.

(1) [LENGTH($\vec{A}$)=1.] if RED($\vec{A}$)=() then { C←FIRST($\vec{A}$); go to 9 }.

(2) [Compute gcd and cofactors.] LDDMPG(p,$\vec{A}$;G,$\vec{B}$).

(3) [Check for zeros and constants.] $\vec{B}'$←$\vec{B}$; $\vec{B}$←(); repeat { ADV($\vec{B}'$;A,$\vec{B}'$); if A≠0 then { if DPLEV(A)=() then { C←G; go to 9 } else $\vec{B}$←EVDLIN($\vec{B}$,A,1) } } until $\vec{B}'$=(); if $\vec{B}$=() then { C←0; go to 9 }; if RED($\vec{B}$)=() then { C←DDMPPR(p,FIRST($\vec{B}$),G); go to 9 }.

(4) [Determine number of variables.] r←LENGTH(DPLEV(FIRST($\vec{B}$))); if r=1 then { C←G; go to 9 }.

(5) [Find r-1 variate ideal element systems.] $\vec{R} \leftarrow ()$; for i=1,...,r do { $\vec{B}' \leftarrow$DDLMVS($\vec{B}$,i); $\vec{R}' \leftarrow$DDMBFV(p,$\vec{B}'$); $\vec{R}' \leftarrow$DDLMVS($\vec{R}'$,i); $\vec{R} \leftarrow$COMP($\vec{R}'$,$\vec{R}$) }.

(6) [Select one element from each system.] $\vec{S} \leftarrow \vec{R}$; $\vec{R} \leftarrow ()$; repeat { ADV($\vec{S}$;$\vec{R}'$,$\vec{S}$); ADV($\vec{R}'$;A,$\vec{R}'$); I$\leftarrow$DPLEV(A); while $\vec{R}' \neq ()$ do { ADV($\vec{R}'$;B,$\vec{R}'$); J$\leftarrow$DPLEV(B); if EVDCMP(I,J)=1 then { A$\leftarrow$B; I$\leftarrow$J } }; $\vec{R} \leftarrow$EVDLIN($\vec{R}$,A,1) } until $\vec{S}=()$.

(7) [Compute monic complete kernel basis for $\vec{B}$.] $\vec{B} \leftarrow$DDMBCU(p,$\vec{B}$,$\vec{R}$).

(8) [Construct basis for ideal generated by $\vec{A}$.] $\vec{C} \leftarrow ()$; repeat { ADV($\vec{B}$;A,$\vec{B}$); C$\leftarrow$DDMPPR(p,G,A); $\vec{C} \leftarrow$COMP(C,$\vec{C}$) } until $\vec{B}=()$; $\vec{C} \leftarrow$INV($\vec{C}$); return.

(9) [Ideal is principal.] C$\leftarrow$DDMPMN(p,C); $\vec{C} \leftarrow$LIST1(C); return □

$\vec{C}$←DDMBCU(p,$\vec{A}$,$\vec{R}$)

[Distributive, dense exponent vector, modular polynomial ideal basis, monic complete kernel basis using univariate ideal elements. $\vec{A}=<A_1,\ldots,A_n>$ is an ideal basis composed of non-zero compatible polynomials over Z_p, p a prime β-integer, in distributive, dense exponent vector, canonical form. $\partial A_1 \leq \ldots \leq \partial A_n$. $\vec{R}=<R_1,\ldots,R_r>$ is a sequence of univariate elements of the ideal generated by $\vec{A}$ with $\partial R_1<\ldots<\partial R_r$. $\vec{C}=<C_1,\ldots,C_s>$ is a monic complete kernel basis for the ideal generated by $\vec{A}$ such that $\partial C_1<\ldots<\partial C_s$. This algorithm uses consensus selection rules 1, 2, and 3 and remainder selection rule 3. The list representing $\vec{A}$ is modified.]

safe a,b,A,A',$\vec{A}'$,B,B',I,J.

(1) [Include elements of $\vec{R}$ in $\vec{A}$ and initialize consensus list.] $\vec{C}$←$\vec{A}$; $\vec{R}'$←$\vec{R}$; repeat { ADV($\vec{R}'$;B,$\vec{R}'$); $\vec{C}$←EVDLIN($\vec{C}$,B,1) } until $\vec{R}'$=(); L←(); ADV($\vec{C}$;A,$\vec{A}'$); while $\vec{A}'\neq()$ do { L←DDBCLG(L,A,$\vec{A}'$); ADV($\vec{A}'$;A,$\vec{A}'$) }.

(2) [Select consensus pair.] if L=() then go to 8; DDBCLS(L;L,K,A,B).

(3) [Form consensus and semi-remainder.] DPLECR(A;I,a,A'); I'←EVDDIF(K,I); a←MDNEG(p,a); DPLECR(B;J,b,B'); J'←EVDDIF(K,J); C←DDMPSM(p,DDMPTP(p,I',b,A'), DDMPTP(p,J',a,B')); C←DDMPSU(p,C,$\vec{C}$,$\vec{R}$).

(4) [Delete if possible.] if I'=() then { $\vec{C}$←LDEL($\vec{C}$,A); L←DDBCLD(L,A) }.

(5) [C is constant.] if C=0 then go to 2; if DPLEV(C)=() then go to 7.

(6) [Modify consensus list and basis.] L←DDBCLG(L,C,$\vec{C}$); $\vec{C}$←EVDLIN($\vec{C}$,C,1); go to 2.

(7) [Identity is in ideal.] $\vec{C}$←LIST1(LIST2((),1)); return.

(8) [Make $\vec{C}$ into monic kernel basis.] ADV($\vec{C}$;C,$\vec{B}$); $\vec{C}$←LIST1(DDMPMN(p,C)); while $\vec{B}$≠() do { ADV($\vec{B}$;C,$\vec{B}$); C'←DDMPRU(p,DPRD(C),$\vec{C}$,$\vec{R}$); C←DPLT(C); if C'≠0 then SRED(RED(C),C'); C←DDMPMN(p,C); $\vec{C}$←EVDLIN($\vec{C}$,C,1) }; return □

$\vec{B}$←DDMBFV(p,$\vec{A}$)

[Distributive, dense exponent vector, modular polynomial ideal basis, ideal elements free of main variable. $\vec{A}=<A_1,\ldots,A_n>$ is a basis of non-zero compatible polynomials over Z_p, p a prime β-integer. $\vec{B}=<B_1,\ldots,B_s>$ is a sequence of elements of the ideal generated by $\vec{A}$ such that the degree of B_i in the main variable is zero, $1\leq i\leq s$. If there exist A_i such that the degree of A_i in the main variable is zero, then $\vec{B}$ is the sequence of such A_i's. Otherwise, $\vec{B}$ is the resultant system of $\vec{A}$.]

safe r.

(1) [Check elements of $\vec{A}$.] PLFDDL($\vec{A}$;r,$\vec{A}$"); $\vec{A}'$←$\vec{A}$"; $\vec{B}$←();
repeat { ADV($\vec{A}'$;A,$\vec{A}'$); if PDEG(A)=0 then $\vec{B}$←COMP(A,$\vec{B}$) }
until $\vec{A}'$=(); if $\vec{B}\neq$() then go to 3.

(2) [Compute resultant system.] $\vec{A}'$←LMPRES(r,p,$\vec{A}$"); $\vec{B}$←();
repeat { ADV($\vec{A}'$;A,$\vec{A}'$); A←LIST2(0,A); $\vec{B}$←COMP(A,$\vec{B}$) }
until $\vec{A}'$=(); $\vec{B}$←INV($\vec{B}$).

(3) [Convert to distributive form and return.]
$\vec{B}$←DDLFPL(r,$\vec{B}$); return □

$\vec{A} \leftarrow$ DDMBRA(m,n,I_1,I_2)

[Distributive, dense exponent vector, modular polynomial ideal basis, random. Inputs m and n are β-integers, I_1 and I_2 are compatible exponent vectors in the dense representation such that $I_1 \leq I_2$. $\vec{A}=\langle A_1,\ldots,A_n\rangle$ is an ideal basis composed of n compatible non-zero random polynomials over Z_m represented in distributive, dense exponent vector, canonical form. For $1 \leq i \leq n$, ∂A_i is compatible with I_2, and ∂A_i is selected at random from the set of all exponent vectors J such that $I_1 \leq J \leq I_2$. In the case of lexicographic order, we also require $J | I_2$.]

safe EVORD. global EVORD.

SAFE d,e,f,i,j,r,z,I'.

(1) [I_2=().] $\vec{A} \leftarrow$(); if I_2=() v EVDCMP(I_1,I_2)=0 then { for i=1,...,n do $\vec{A} \leftarrow$COMP(DDMPRA(m,I_2),$\vec{A}$); return }.

(2) [Decide order.] if EVORD=2 then go to 4.

(3) [Lexicographic order.] for i=1,...,n do { repeat { J←(); I'←I_2; z←0; repeat { ADV(I';e,I'); f←MDRAN(e+1); if f>0 then z←1; J←COMP(f,J) } until I'=(); if z=0 then J←() else J←INV(J) } until EVDCMP(I_1,J)≤0 & EVDCMP(J,I_2)≤0; A←DDMPRA(m,J); $\vec{A} \leftarrow$COMP(A,$\vec{A}$) }; return.

(4) [Total degree order.] r←LENGTH(I_2); d←1; I'←I_2;
for j=1,...,r do { ADV(I';e,I'); d←d+e }; for i=1,...,n
do { repeat { J←(); z←0; for j=1,...,r do
{ f←MDRAN(d); if f>0 then z←1; J←COMP(f,J) };
if z=0 then J←() else J←INV(J) } until EVDCMP(I_1,J)≤0
& EVDCMP(J,I_2)≤0; A←DDMPRA(m,J); $\vec{A}$←COMP(A,$\vec{A}$) }; return □

C←DDMPDF(m,A,B)

[Distributive, dense exponent vector, modular polynomial difference. A and B are compatible polynomials over Z_m, m a β-integer, represented in distributive, dense exponent vector, canonical form. C=A-B.]

(1) C←DDMPSM(m,A,DDMPNG(m,B)); return □

B←DDMPHM(m,A)

[Distributive, dense exponent vector, modular polynomial homomorphism. A is an integral polynomial represented in distributive, dense exponent vector, canonical form. m is a positive β-integer. $B=H_m(A)$.]

safe a,b,A',I.

(1) [A=0.] if A=0 then { B←0; return }.

(2) [Apply homomorphism to each term of A.] A'←A; B←();
repeat { ADV2(A';I,a,A'); b←MDHOM(m,a); if b≠0 then
B←COMP2(b,I,B) } until A'=(); if B=() then B←0 else
B←INV(B); return □

A'←DDMPMN(p,A)

[Distributive, dense exponent vector, modular polynomial monic. A is a polynomial over Z_p, p a prime β-integer, in distributive, dense exponent vector, canonical form. If A≠0 then A' is the polynomial similar to A with lc(A')=1; if A=0 then A'=0.]

safe a,a'.

(1) [A=0.] if A=0 then { A'←0; return }.

(2) [A≠0.] a←DPLC(A); a'←MDINV(p,a); A'←DDMPTP(p,(),a',A); return □

B←DDMPNG(m,A)

[Distributive, dense exponent vector, modular polynomial negative. A is a polynomial over Z_m, m a β-integer, represented in distributive, dense exponent vector, canonical form. B=-A.]

safe a,b,A',I.

(1) [A=0.] if A=0 then { B←0; return }.

(2) [A≠0.] A'←A; B←(); repeat { ADV2(A';I,a,A'); b←MDNEG(m,a); B←COMP2(b,I,B) } until A'=(); B←INV(B); return □

C←DDMPPR(m,A,B)

[Distributive, dense exponent vector, modular polynomial product. A and B are compatible polynomials over A_m, m a β-integer, in distributive, dense exponent vector, canonical form. C=AB.]

safe A',I.

(1) [A=0 V B=0.] C←0; if A=0 V B=0 then return.

(2) [A≠0 & B≠0.] A'←A; repeat { DPLECR(A';I,a,A');
C←DDMPSM(m,C,DDMPTP(m,I,a,B)) } until A'=0; return □

A←DDMPRA(m,I)

[Distributive, dense exponent vector, modular polynomial, random. The input m is a β-integer, I is an exponent vector in the dense representation. A is a random polynomial over Z_m represented in distributive, dense exponent vector, canonical form, such that ∂A=I and A is compatible with I. The ordering of the monomials of A is determined by the global variable EVORD. EVORD=1 gives lexicographic ordering, EVORD=2 gives total degree ordering. If J is the exponent vector of any non-zero term of A in the lexicographic case, then J|I.]

safe EVORD. global EVORD.

safe a,e,f,i,m',n,r,L,L'.

(1) [I=().] m'←m-1; if I=() then { a←MDRAN(m')+1;
A←LIST2((),a); return }.

(2) [I≠().] r←LENGTH(I); if EVORD=2 then go to 4.

(3) [Lexicographic ordering.] I'←CINV(I); L←LIST1(());
repeat { M←(); ADV(I';e,I'); for f=0,...,e do { L'←L;
repeat { ADV(L';J,L'); M←COMP(COMP(f,J),M) } until
L'=() }; L←INV(M) } until I'=(); go to 7.

(4) [Total degree ordering.] n←0; I'←I; repeat { ADV(I';e,I');
n←n+e } until I'=(); M←(); for e=0,...,n do
M←COMP(LIST1(LIST1(e)),M); for i=2,...,r do { L←M; M←();
for e=n,n-1,...,0 do { $\overline{L}$←L; M'←(); for f=0,...,e do
{ ADV($\overline{L}$;L',$\overline{L}$); while L'≠() do { ADV(L';J,L');
M'←COMP(COMP(f,J),M') } }; M←COMP(INV(M'),M); L←RED(L) };
M←INV(M) }.

(5) [Delete excess exponent vectors.] L←FIRST(M); repeat
{ L'←L; ADV(L;J,L) } until EVDCMP(I,J)=0; SRED(L',()).

(6) [Connect sublists of M.] L←(); while M≠() do
{ ADV(M;M',M); L←CONC(M',L) }.

(7) [Insert coefficients.] SFIRST(L,()); A←();
repeat { ADV(L;J,L); a←MDRAN(m); if a≠0 then
A←COMP2(J,a,A) } until L=().

(8) [Make I the leading exponent vector.] If A=() V
EVDCMP(I,FIRST(A))≠0 then { a←MDRAN(m')+1; A←COMP2(I,a,A) };
return □

Q←DDMPRM(p,P,$\vec{A}$)

[Distributive, dense exponent vector, modular polynomial remainder. P is a polynomial in distributive, dense exponent vector, canonical form over Z_p, p a prime β-integer. $\vec{A}=<A_1,\ldots,A_n>$ is a basis of polynomials over Z_p compatible with P. For $1\leq i\leq n$, $A_i\neq 0$ and $\partial A_1\leq\partial A_2\leq\ldots\leq\partial A_n$. Q=rem(P,$\vec{A}$) using remainder selection rules 1 and 2.]

safe a,b,q,B,J.

(1) [P=0.] if P=0 then { Q←P; return }.

(2) [Initialize.] Q'←P; Q←().

(3) [Attempt to reduce lt(Q').] repeat { DPLECR(Q';J,q,Q'); DDBESL(J,$\vec{A}$;K,a,B); if K=-1 then Q←COMP2(q,J,Q) else { b←MDNEG(p,MDQ(p,q,a)); Q'←DDMPSM(p,Q',DDMPTP(p,K,b,B)) } } until Q'=0.

(4) [Finish.] if Q=() then Q←0 else Q←INV(Q); return □

Q←DDMPRU(p,P,$\vec{A}$,$\vec{R}$)

[Distributive, dense exponent vector, modular polynomial remainder using univariate ideal elements. P is a polynomial in distributive, dense exponent vector, canonical form over Z_p, p a prime β-integer. $\vec{A}=<A_1,\ldots,A_n>$ and $\vec{R}=<R_1,\ldots,R_r>$ are sequences of non-zero polynomials over Z_p compatible with P. The elements of $\vec{R}$ are univariate elements of the ideal generated by $\vec{A}$. $\partial A_1 \leq \ldots \leq \partial A_n$ and $\partial R_1 < \ldots < \partial R_r$. Q=rem(P,$<A_1,\ldots,A_n,R_1,\ldots,R_r>$).]

safe a,b,q,B,J.

(1) [P=0.] if P=0 then { Q←P; return }.

(2) [Initialize.] Q←DDMPRM(p,P,$\vec{R}$); if Q=0 then return; Q'←Q; Q←().

(3) [Attempt to reduce lt(Q').] repeat { DPLECR(Q';J,q,Q'); DDBESL(J,$\vec{A}$;K,a,B); if K=-1 then Q←COMP2(q,J,Q) else { b←MDNEG(p,MDQ(p,q,a)); Q'←DDMPSM(p,Q',DDMPTP(p,K,b,B)); Q'←DDMPRM(p,Q',$\vec{R}$) } } until Q'=0.

(4) [Finish.] if Q=() then Q←0 else Q←INV(Q); return □

C←DDMPSM(m,A,B)

[Distributive, dense exponent vector, modular polynomial sum. A and B are compatible polynomials over Z_m, m a β-integer, represented in distributive, dense exponent vector, canonical form. C=A+B.]

safe a,b,c,k,A',B',I,J.

(1) [A or B zero.] if A=0 then { C←B; return };
if B=0 then { C←A; return }.

(2) [Compare exponent vectors.] A'←A; B'←B; C'←();
repeat { I←DPLEV(A'); J←DPLEV(B'); k←EVDCMP(I,J); case k
of { -1 do { ADV2(B';J,b,B'); C'←COMP2(b,J,C') };
0 do { ADV2(A';I,a,A'); ADV2(B';J,b,B'); c←MDSUM(m,a,b);
if c≠0 then C'←COMP2(c,I,C') }; 1 do { ADV2(A';I,a,A');
C'←COMP2(a,I,C') } } } until A'=() V B'=().

(3) [Finish.] if A'=() then A'←B'; if C'=() then C←A' else
{ C←INV(C'); SRED(C',A') }; if C=() then C←0; return □

Q←DDMPSR(p,P,$\vec{A}$)

[Distributive, dense exponent vector, modular polynomial semi-remainder. P is a polynomial in distributive, dense exponent vector, canonical form over Z_p, p a prime β-integer. $\vec{A}=<A_1,\ldots,A_n>$ is an ideal basis composed of polynomials over Z_p compatible with P. For $1\leq i\leq n$, $A_i\neq 0$, and $\partial A_1\leq\partial A_2\leq\ldots\leq\partial A_n$. Q=srem(P,$\vec{A}$) using remainder selection rule 2.]

safe a,b,q,B,J,Q'.

(1) [P=0.] Q←P; if P=0 then return.

(2) [Attempt to reduce lt(Q).] repeat { DPLECR(Q;J,q,Q');
DDBESL(J,$\vec{A}$;K,a,B); if K=-1 then return;
b←MDNEG(p,MDQ(p,q,a)); Q←DDMPSM(p,Q',DDMPTP(p,K,b,B)) }
until Q=0; return □

Q←DDMPSU(p,P,$\vec{A}$,$\vec{R}$)

[Distributive, dense exponent vector, modular polynomial semi-remainder using univariate ideal elements. P is a polynomial in distributive, dense exponent vector, canonical from over Z_p, p a prime β-integer. $\vec{A}=<A_1,\ldots,A_n>$ and $\vec{R}=<R_1,\ldots,R_r>$ are sequences of non-zero polynomials over Z_p compatible with P. The elements of $\vec{R}$ are univariate elements of the ideal generated by $\vec{A}$. $\partial A_1 \leq \ldots \leq \partial A_n$ and $\partial R_1 < \ldots < \partial R_r$. Q=srem(P,$<A_1,\ldots,A_n,R_1,\ldots,R_r>$).]

safe a,b,q,B,J,Q'.

(1) [Initialize.] Q←DDMPRM(p,P,$\vec{R}$); if Q=0 then return.

(2) [Attempt to reduce lt(Q).] repeat { DPLECR(Q;J,q,Q'); DDBESL(J,$\vec{A}$;K,a,B); if K=-1 then return; b←MDNEG(p,MDQ(p,q,a)); Q←DDMPSM(p,Q',DDMPTP(p,K,b,B)); Q←DDMPRM(p,Q,$\vec{R}$) } until Q=0; return □

B←DDMPTP(m,J,b,A)

[Distributive, dense exponent vector, modular polynomial term product. A is a polynomial over Z_m, m a β-integer, represented in distributive, dense exponent vector, canonical form; J is an exponent vector in the dense exponent vector representation compatible with ∂A; b is an element of Z_m. $B=bx^JA$.]

safe a,c,A',I.

(1) [A or b is zero.] if A=0 V b=0 then { B←0; return }.

(2) [Multiply each term of A.] A'←A; B←(); repeat
{ ADV2(A';I,a,A'); c←MDPROD(m,a,b); K←EVDSUM(I,J);
if c≠0 then B←COMP2(c,K,B) } until A'=(); if B=() then
B←0 else B←INV(B); return □

B←DDPFP(r,A)

[Distributive, dense exponent vector, polynomial from polynomial. A is a polynomial in r variables, $r \geq 1$. B is a polynomial in distributive, dense exponent vector, canonical form such that B=A. The order of the exponent vectors in B is determined by the global variable EVORD.]

safe EVORD. global EVORD.

(1) B←DDPFPL(r,A); if EVORD≠1 then B←DDPSRT(B); return □

B←DDPFPL(r,A)

[Distributed, dense exponent vector, polynomial from polynomial, lexicographic order. A is a polynomial in r variables, r≥1. B is a polynomial in distributive, dense exponent vector, canonical form such that B=A. The exponent vectors in B are in lexicographic order.]

safe a,i.

(1) [A=0 or r=0.] if A=0 then { B←0; return }; if r=0 then { B←LIST2((),A); return }.

(2) [A≠0, r≥1.] B←(); r'←r-1; A'←A; repeat { ADV2(A';e,a,A'); if r'=0 then { if e=0 then I←() else I←LIST1(e); B←COMP2(a,I,B) } else { b←DDPFPL(r',a); while b≠() do { ADV2(b;J,c,b); if e≠0 & J=() then for i=1,...,r' do J←COMP(0,J); if J≠() then J←COMP(e,J); B←COMP2(c,J,B) } } } until A'=(); B←INV(B); return □

B←DDPMVS(A,k)

[Distributive, dense exponent vector, polynomial main variable substitution. A is a polynomial in distributive, dense exponent vector, canonical form in variables $x_1,\ldots,x_r$. B is gotten from A by interchanging the main variable x_r with the variable x_k, $1 \leq k \leq r$.]

safe a,A',I.

(1) [A=0.] if A=0 then { B←A; return }.

(2) [Interchange in each term.] A'←A; B←(); repeat { DPLECR(A';I,a,A'); J←EVDMVS(I,k); B←COMP2(J,a,B) } until A'=0; B←DDPSRT(B); return □

B←DDPSRT(A)

[Distributive, dense exponent vector, polynomial sort. A is a polynomial in distributive, dense exponent vector, form. The terms of A are in arbitrary order. The terms are sorted in descending order. The order used is determined by the setting of the global variable EVORD. The polynomial A is modified.]

safe DDPSRT.

(1) [A=0.] B←A; if A=0 then return.

(2) [Bubble sort A.] B_0←(); repeat { B'←B; B"←RED2(B'); DPLEC(B';I',a'); B_0'←(); while B"≠B_0 do { DPLEC(B";I",a"); if EVDCMP(I',I")=-1 then { SFIRST(B',I"); SFIRST(RED(B'),a"); SFIRST(B",I'); SFIRST(RED(B"),a'); B_0'←B"} else { I'←I"; a'←a" }; B'←B"; B"←RED2(B') }; B_0←B_0' } until B_0=(); return □

a←DPLC(A)

[Distributive polynomial, leading coefficient. A is a non-zero polynomial in distributive canonical form. a=lc(A).]

safe DPLC.

(1) a←SECOND(A); return □

DPLEC(A;I,a)

[Distributive polynomial, leading exponent vector and leading coefficient. A is a non-zero polynomial in distributive canonical form. I=∂A and a=lc(A).]

safe DPLEC.

(1) FIRST2(A;I,a); return □

DPLECR(A;I,a,B)

[Distributive polynomial, leading exponent vector, leading coefficient, and reductum. A is a non-zero polynomial in distributive canonical form. I=∂A, a=lc(A), and B=rd(A).]

safe DPLECR.

(1) ADV2(A;I,a,B); if B=() then B=0; return □

I←DPLEV(A)

[Distributive polynomial, leading exponent vector. A is a non-zero polynomial in distributive canonical form. I=∂A.]

safe DPLEV.

(1) I←FIRST(A); return □

B←DPLT(A)

[Distributive polynomial, leading term. A is a non-zero polynomial in distributive canonical form. B=lt(A).]

safe DPLT.

(1) FIRST2(A;I,a); B←LIST2(I,a); return □

B←DPRD(A)

[Distributive polynomial, reductum. A is a non-zero polynomial in distributive canonical form. B=rd(A).]

safe DPRD.

(1) B←RED2(A); if B=() then B←0; return □

b←EVDCMP(I,J)

[Exponent vector, dense representation, comparison. I and J are compatible exponent vectors in the dense exponent vector representation. The ordering used is determined by the global variable EVORD. EVORD=1 gives lexicographic ordering; EVORD=2 gives total degree ordering. If I<J then b=-1. If I=J then b=0. If I>J then b=1.]

safe EVDCMP.

safe EVORD. global EVORD.

(1) [Initialize.] b←0; if I=J then return; if I=() then { b← -1; return }; if J=() then { b←1; return }.

(2) [Decide ordering.] if EVORD=2 then go to 4.

(3) [Lexicographic ordering.] I'←I; J'←J;
repeat { ADV(I';e,I'); ADV(J';f,J'); if e<f then
{ b←-1; return }; if e>f then { b←1; return } } until
I'=(); return.

(4) [Total degree ordering.] I'←I; J'←J; s←0; t←o; repeat
{ ADV(I';e,I'); ADV(J';f,J'); s←s+e; t←t+f; if b=0 then
b←SIGN(e-f) } until I'=(); if s=t then return;
b←SIGN(s-t); return □

K←EVDDIF(I,J)

[Exponent vector, dense representation, difference. I and J are compatible exponent vectors in the dense exponent vector representation. If J divides I then K=I-J. Otherwise K= -1.]

safe e,f,g,z,I',J'.

(1) [Initialize.] K←(); if I=J then return; if J=() then { K←I; return }; if I=() then { K← -1; return }; I'←I; J'←J; z←0.

(2) [Subtract vector components.] repeat { ADV(I';e,I'); ADV(J';f,J'); g←e-f; if g<0 then { K← -1; return }; if g≠0 then z←1; K←COMP(g,K) } until I'=().

(3) [Are all elements of K zero?] if z=0 then K←() else K←INV(K); return □

K←EVDGCD(I,J)

[Exponent vector, dense representation, greatest common divisor. I and J are compatible exponent vectors in the dense exponent vector representation. K=gcd(I,J).]

safe e,f,g,z,I',J'.

(1) [I=() or J=().] K←(); if I=() V J=() then return.

(2) [Compute gcd.] I'←I; J'←J; z←0; repeat { ADV(I';e,I'); ADV(J';f,J'); g←MIN(e,f); K←COMP(g,K); if g>0 then z←1 } until I'=().

(3) [Check for zero vector.] if z=0 then K←() else K←INV(K); return □

K←EVDLCM(I,J)

[Exponent vector, dense representation, least common multiple. I and J are compatible exponent vectors in the dense exponent vector representation. K=lcm(I,J).]

safe e,f,g,I',J'.

(1) [I=() or J=().] if I=() then { K←J; return };
if J=() then { K←I; return }.

(2) [Compute lcm.] I'←I; J'←J; K←();
repeat { ADV(I';e,I'); ADV(J';f,J'); g←MAX(e,f);
K←COMP(g,K) } until I'=(). K←INV(K); return □

L'←EVDLIN(L,M,k)

[Exponent vector, dense representation, list insertion. L and M are lists. If L=() then L'=(M). Otherwise L=(M_1,...,M_s) where M_i is a list such that K_i=FIRST(M_i) is an exponent vector, in the dense exponent vector representation, compatible with the exponent vector K=FIRST(M). The elements of L are arranged such that the exponent vectors K_i are in non-decreasing (k=1) or non-increasing (k=-1) order. M is inserted in L such that the ordering is preserved. L' is the new list; L is modified.]

safe b,K,K',M',N,N',N".

(1) [L=().] L"←LIST1(M); if L=() then { L'←L"; return }.

(2) [Find insertion point.] K←FIRST(M); L'←L; N"←(); N'←L; repeat { ADV(N';M',N); K'←FIRST(M'); b←EVDCMP(K,K'); if k=1 & b≠1 V k=-1 & b≠-1 then go to 4; N"←N'; N'←N } until N=().

(3) [Insert M at end.] SRED(N",L"); return.

(4) [Insert M between N" and N'.] SRED(L",N'); if N"=() then L'←L" else SRED(N",L"); return □

J←EVDMVS(I,k)

[Exponent vector, dense representation, main variable substitution. I is an exponent vector in the dense representation. If I=() then J=(). If $I=(e_r,\ldots,e_1)$ then $J=(e_k,e_{r-1},\ldots,e_{k+1},e_r,e_{k-1},\ldots,e_1)$, $1\leq k\leq r$.]

safe e,f,r,I',J'.

(1) [I=().] J←(); if I=() then return.

(2) [k=r.] r←LENGTH(I); if k=r then { J←I; return }.

(3) [Interchange.] I'←I; f←FIRST(I); for i=1,...,r-k
do { ADV(I';e,I'); J←COMP(e,J) }. ADV(I';e,I');
J←COMP(f,J); J'←J; J←INV(J); SFIRST(J,e); SRED(J',I');
return □

K←EVDSUM(I,J)

[Exponent vector, dense representation, sum. I and J are compatible exponent vectors in the dense exponent vector representation. K=I+J.]

safe e,f,I',J'.

(1) [Initialize.] if I=() then { K←J; return }; if J=() then { K←I; return }; I'←I; J'←J; K←().

(2) [Sum vector components.] repeat { ADV(I';e,I'); ADV(J';f,J'); K←COMP(e+f,K) } until I'=().

(3) [Invert K and return.] K←INV(K); return □

LDDMPG(p,L;G,M)

[List of distributive, dense exponent vector, modular polynomials, greatest common divisor and cofactors. $L=(A_1,\ldots,A_n)$, $n\geq 2$, is a list of compatible polynomials over Z_p, p a prime β-integer, in distributive, dense exponent vector, canonical form. $G=\gcd(A_1,\ldots,A_n)$ and $M=(A_1/G,\ldots,A_n/G)$. G and the elements of M are in distributive, dense exponent vector, canonical form.]

safe r.

(1) PLFDDL(L;r,L'); LMPGDC(r,p,L';G,M); G←DDPFP(r,G); M←DDLFPL(r,M); return □

M←LDEL(L,A)

[List deletion. $L=(L_1,\ldots,L_n)$, $n\geq 1$, is a list of γ-integers. A is a γ-integer. If i is the least integer such that $L_i=A$, then $M=(L_1,\ldots,L_{i-1},L_{i+1},\ldots,L_n)$. If A does not occur in L, M=L. L is modified.]

safe LDEL.

(1) [$A=L_1$.] if A=FIRST(L) then { M←RED(L); return }.

(2) [Scan L.] M←L; L"←L; L'←RED(L"); while L'≠()
{ if A=FIRST(L') then { SRED(L",RED(L')); return };
L"←L'; L'←RED(L") }; return □

LMPGDC(r,p,L;G,M)

[List of modular polynomials, greatest common divisor and cofactors. $L=(A_1,\ldots,A_n)$, $n\geq 2$, is a list of polynomials in r variables, $r\geq 1$, over Z_p, p a prime β-integer. $G=\gcd(A_1,\ldots,A_n)$ and $M=(A_1/G,\ldots,A_n/G)$, a list of cofactors.]

safe A,L',V,W.

(1) [Find G.] ADV(L;G,L'); repeat { ADV(L';A,L'); MPGCDC(r,p,G,A;G,V,W) } until L'=().

(2) [Compute cofactors.] M←(); L'←L; repeat { ADV(L';A,L'); M←COMP(MPQ(r,p,A,G),M) } until L'=(); M←INV(M); return □

M←LMPRES(r,p,L)

[List of modular polynomials, resultant system. $L=(A_1,\ldots,A_n)$, $n\geq 2$, is a list of polynomials in r variables over Z_p, p a prime β-integer. For $1\leq i\leq n$ the degree of A_i in the main variable is positive. The resultant system of L is computed with respect to the main variable. If the resultant system is empty, then M=(). Otherwise, $M=(B_1,\ldots,B_s)$ is the resultant system of L. The elements of M are polynomials in r-1 variables.]

safe b,e,n,n',r',A,L'.

(1) [LENGTH(L)=2.] n←LENGTH(L); if n=2 then { FIRST2(L;A,B); C←MPRES(r,p,A,B); if C=0 then M←() else M←LIST1(C); return }.

(2) [Introduce indeterminates.] n'←n-1; L'←L; N←();
for i=1,...,n do { ADV(L';A,L'); B←PINV(1,A,n'); B'←B;
repeat { ADV2(B';e,b,B'); for j=3,...,i do b←SECOND(b);
if i>1 then SFIRST(b,1) } until B'=(); N←COMP(B,N) };
N←INV(N).

(3) [Sum last n' elements of N.] r'←r+n'; ADV(N;A,N'); B←0;
for i=1,...,n' do { ADV(N';B',N'); B←MPSUM(r',p,B',B) }.

(4) [Compute resultant.] C←MPRES(r',p,A,B).

(5) [Get list of (r-1)-variate coefficients of C.]
M←PBCFL(n',C); return □

L←PBCFL(r,A)

[Polynomial base coefficient list. A is a polynomial in r variables. If A=0 then L=(). If A≠0 then L is the list of base coefficients of A. L is in descending lexicographic order with respect to the monomials of A.]

safe a,e.

(1) [A=0 or r=0.] L←(); if A=0 then return; if r=0 then { L←LIST1(A); return }.

(2) [A≠0 and r≥1.] r'←r-1; A'←A; repeat { ADV2(A';e,a,A'); if r'=0 then L←COMP(a,L) else { L'←PBCFL(r',a); L"←INV(L'); SRED(L',L); L←L" } } until A'=(); L←INV(L); return □

PFDDP(A;r,B)

[Polynomial from distributive, dense exponent vector, polynomial. A is a polynomial in r variables, $r \geq 0$. A is represented in distributive, dense exponent vector, canonical form. B is a polynomial in recursive canonical form such that B=A. If A is a constant then B=lc(A) and r=0.]

safe a,e,i,r,A'.

(1) [A is constant.] if A=0 then { B←0; r←0; return }; I←DPLEV(A); if I=() then { B←DPLC(A); r←0; return }.

(2) [Convert each term of A and sum.] B←0; A'←A; r←LENGTH(I); repeat { ADV2(A';I,a,A'); T←a; if I=() then for i=1,...,r do I←COMP(0,I) else I←CINV(I); repeat { ADV(I;e,I); T←LIST2(e,T) } until I=(); B←MPSUM(r,B,T) } until A'=(); return □

PLFDDL(L;r,M)

[Polynomial list from distributive, dense exponent vector, polynomial list. L is a list of compatible polynomials in r variables, r≥0, represented in distributive, dense exponent vector, canonical form. M is a list of the polynomials in L in recursive canonical form.]

safe r,r',A,L'.

(1) [Convert polynomials in L.] r←0; L'←L; M'←();
repeat { ADV(L';A,L'); PFDDP(A;r',B); M'←COMP2(r',B,M');
if r'>r then r←r' } until L'=().

(2) [Make constants compatible with r variables.]
M←(); repeat { ADV2(M';r',B,M'); if B≠0 & r'<r then
B←PINV(0,B,r); M←COMP(B,M) } until M'=(); return □

APPENDIX C:

Definition/Symbol Index

Definition	Section
remainder selection rule 2'	5.2
representation	2.3
resultant system	4.3
semi-kernel basis	3.5
semi-kernel polynomial	2.5
semi-remainder	2.5
simple constructibility	2.3
simple representation	2.3
simplification ring	2.2
simplifying ample function	1.2
sum norm	4.2
total degree ordering	3.2

Symbol	Section	Symbol	Section
$\bar{a}$	2.2	$I \mid J$	2.2
$\vec{a}$	2.2	$I \not\mid J$	2.2
$\bar{a}_I$	2.2	$I+J$	2.2
$\vec{a}_I$	2.2	$J-I$	2.2
$\bar{\bar{a}}_I$	2.2	$\ker(\vec{A})$	2.2
$\bar{A}$	2.2	$L(d)$	4.2
$\vec{A}$	2.2	$lc(A)$	2.2
$\vec{A}_I$	2.2	$lcm(I,J)$	2.2
$\|A\|_1$	4.2	$lm(A)$	2.2
$\|A\|_\infty$	4.2	$lt(A)$	2.2
$\vec{a}\cdot\vec{b}$	2.2	$mcons(\vec{A})$	3.6
$\beta(a)$	2.2	$\mu\vec{S}$	2.3
$\beta^*(A)$	2.6	$N(r,m)$	3.4
$cons(A,B)$	3.6	$\bar{N}(r,m)$	3.4
$cons(\vec{A})$	2.4	$P_L(I)$	5.1
$cons_j(\vec{B})$	2.4	$P_T(I)$	5.1
CSR1	3.6	$rd(A)$	2.2
CSR2	3.6	$R[x]$	1.3
CSR3	3.6	R^*	2.2
CSR3'	5.3	RSR1	3.4
CSR3''	5.3	RSR2	3.4
∂A	2.2	RSR2'	5.2
$\partial_i A$	4.2	$RS_x(A_1,\ldots,A_n)$	4.3
$\partial_{x_i} A$	2.2	$\rho(b,\vec{a})$	2.2
		$\rho^*(B,\vec{A})$	2.6
$\partial\vec{S}$	2.3	$sker(\vec{A})$	2.5
E	2.2	$\theta(b,\vec{a})$	2.2
$E_L(I)$	5.1	$\theta^*(B,A)$	2.6
$E_T(I)$	5.1	$V(\vec{a})$	2.2
$gcd(I,J)$	2.2	$V(\vec{A})$	2.6

REFERENCES:

[BER78] G. M. Bergman, "The Diamond Lemma for Ring Theory," Advances in Mathematics (to appear).

[BRD70] G. H. Bradley, "Algorithms and Bounds for the Greatest Common Divisor of n Integers," Numerical Mathematics, 13, pp 433-436 (1970).

[BRD71] G. H. Bradley, "Algorithms for Hermite and Smith Normal Matrices and Linear Diophantine Equations," Mathematics of Computation, 25, pp 897-907 (1971).

[BLK66] W. A. Blankinship, "Algorithm 288, Solution of Linear Diophantine Equations [F4]," Communications of the ACM, 9, p 514 (1966).

[BRN71] W. S. Brown, "On Euclid's Algorithm and the Computation of Polynomial Greatest Common Divisors," Journal of the ACM, 18, No. 4, pp 478-504 (Oct. 1971).

[BUC65] B. Buchberger, "Ein Algorithmus zum Auffinden der Basiselemente des Restklassenrings nach einem null-dimensionalen Polynomialideal," Dissertation, Universität Innsbruck (1965).

[BUC70] B. Buchberger, "Ein Algorithmisches Kriterium für die Lösbarkeit eines Algebraischen Gleichungssystems, "Aequationes Mathematicae, 4, pp 374-383 (1970).

[BUC76a] B. Buchberger, "Some Properties of Gröbner-Bases for Polynomial Ideals," SIGSAM Bulletin, 10, No. 4, pp 19-24 (Nov. 1976).

[BUC76b] B. Buchberger, "A Theoretical Basis for the Reduction of Polynomials to Canonical Forms," SIGSAM Bulletin, 10, No. 3, pp 19-29 (Aug. 1976).

[BUC76c] B. Buchberger, "On Certain Bases of Polynomial Ideals," Bericht 53, Institut für Mathematik, Johannes Kepler Universität Linz (1976).

[CAV68] B. F. Caviness, "On Canonical Forms and Simplification," PhD Thesis, Carnegie-Mellon University (1968).

[CAV70] B. F. Caviness, "On Canonical Forms and Simplification," Journal of the ACM, 17, No. 2, pp 385-396 (Apr. 1970).

[COL71] G. E. Collins, "The Calculation of Multivariate Polynomial Resultants," Journal of the ACM, 18, No. 4, pp 515-532 (Oct. 1971).

[COL78] G. E. Collins, Algebraic Algorithms, Prentice-Hall, (to appear).

[FAT72] R. J. Fateman, "Essays in Algebraic Simplification," PhD Thesis, Massachusetts Institute of Technology (1972).

[GEL60] A. O. Gelfond, Transcendental and Algebraic Numbers, Dover, New York (1960).

[HAL68] M. Hall, Jr., Combinatorial Theory, Blaisdell/Ginn. New York (1968).

[HER26] G. Hermann, "Die Frage der endlich vielen Schritte in der Theorie der Polynomideale," Math. Ann., 95, pp 736-788 (1926).

[KBA78] R. Kannan and A. Bachem, "Polynomial Algorithms for Computing the Smith and Hermite Normal Forms of an Integer Matrix," SIAM Journal on Computing (to appear).

[KLE66] S. L. Kleiman, "Computing with Rational Expressions in Several Algebraically Dependent Variables," Bell Telephone Laboratories Computer Science Technical Report No. 42 (1966).

[KOB78] C. Kollreider and B. Buchberger, "An Improved Algorithmic Construction of Gröbner-Bases for Polynomial Ideals, "SIGSAM Bulletin, 12, No. 2 pp 27-36 (May 1978).

[KNU68] D. E. Knuth, The Art of Computer Programming, Vol. 1, Addison Wesley, Reading, MA (1968).

[KNU69] D. E. Knuth, The Art of Computer Programming, Vol. 2, Addison Wesley, Reading, MA (1969).

[LAU76a] M. Lauer, "Canonical Representatives for Residue Classes of a Polynomial Ideal," Proceedings of the 1976 ACM Symposium on Symbolic and Algebraic Computation, pp 339-345 (Aug 1976).

[LAU76b] M. Lauer, "Kanonische Repräsentanten für die Restklassen nach einem Polynomideal," Diplomarbeit, Universität Kaiserslautern (1976).

[LAZ76] D. Lazard, "Algorithmes Fondamentaux en Algèbre Commutative," Société Mathématique de France Astérisque, 38-39, pp 131-138, (1976).

[LOS74] R. Loos, "Towards a Formal Implementation of Computer Algebra," SIGSAM Bulletin, 8, No. 3, pp-9-16 (Aug 1974).

[LOS76] R. Loos, "The Algorithm Description Language ALDES (report)," SIGSAM Bulletin, 10, No. 1, pp 15-39 (Feb 1976).

[MAC16] F. S. Macaulay, The Algebraic Theory of Modular Systems, Cambridge Univ. Press, London (1916).

[MEN70] E. Mendelson, Boolean Algebra and Switching Circuits, McGraw-Hill, New York (1970).

[MOS71] J. Moses, "Algebraic Simplification, A Guide for the Perplexed," Communications of the ACM, 14, No. 8, pp 527-538 (Aug 1971).

[MUS71] D. R. Musser, "Algorithms for Polynomial Factorization," PhD Thesis, University of Wisconsin (1971).

[MUS75] D. R. Musser, "Multivariate Polynomial Factorization," Journal of the ACM, 22, No. 2, pp 291-308 (Apr 1975).

[RCD68] D. Richardson, "Some Unsolvable Problems Involving Elementary Functions of a Real Variable," Journal of Symbolic Logic, 33, pp 514-520 (1968).

[RCM74] F. Richman, "Constructive Aspects of Noetherian Rings," Proceedings of the AMS, 44, No. 2, pp 436-441 (June 1974).

[SCH76] R. Schrader, "Zur Konstruktiven Idealtheorie," Diplomarbeit, Universität Karlsruhe (1976).

[SEI56] A. Seidenberg, "An Elimination Theory for Differential Algebra," Univ. Calif. Publ. Math., 3, pp 31-65 (1956).

[SEI71] A. Seidenberg, "On the Length of a Hilbert Ascending Chain," Proceedings of the AMS, 29, No. 3 pp 443-450 (Aug 1971).

[SEI72] A. Seidenberg, "Constructive Proof of Hilbert's Theorem on Ascending Chains," Transactions of the AMS, 174, pp 305-312, (1972).

[SEI74] A. Seidenberg, "Constructions in Algebra," Transactions of the AMS, 197, pp 273-313 (1974).

[SHT76] R. Shtokhamer, "A Canonical Form of Polynomials in the Presence of Side Relations," Technion-PH-76-25, Physics Department, Technion, Haifa, Israel (1976).

[SHT78] R. Shtokhamer, "The Use of 'Let' Statements in Producing Short Comprehensible Outputs," SIGSAM Bulletin, 11/12, pp 20-22 (Nov 1977/Feb 1978).

[SPE77] D. A. Spear, "A Constructive Approach to Commutative Ring Theory," Proceedings of the 1977 MACSYMA Users' Conference, pp 369-376 (July 1977).

[SZK52] G. Szekeres, "A Canonical Basis for the Ideals of a Polynomial Domain," Am. Math. Monthly, 59, No. 6, pp 379-386 (June 1952).

[TRI78] W. Trinks, "Uber B. Buchbergers Verfahren, Systeme algebraischer Gleichungen zu lösen," Manuscript, Fachbereich Informatik I, Universität Karlsruhe (1978).

[VDW50] B. L. van der Waerden, Modern Algebra, Vol 2, Ungar, New York (1950).

[VDW70a] B. L. van der Waerden, Algebra, Vol. 1, Ungar, New York (1970).

[VDW70b] B. L. van der Waerden, Algebra, Vol. 2, Ungar, New York (1970).