ALGORITHMS FOR POLYNOMIALS OVER A REAL ALGEBRAIC NUMBER FIELD

by

Cyrenus M. Rubald

Computer Sciences Technical Report #206

January 1974

WIS-CS-206-74
COMPUTER SCIENCES DEPARTMENT
The University of Wisconsin
1210 West Dayton Street
Madison, Wisconsin 53706

Received January 1974

ALGORITHMS FOR POLYNOMIALS
OVER A REAL ALGEBRAIC NUMBER FIELD

by

Cyrenus M. Rubald

Computer Sciences Technical Report #206

January 1974

ACKNOWLEDGEMENT

I would like to express my thanks to Professor George E. Collins for the large amount of time and energy he has expended on my behalf in all aspects of my graduate career, and for his enthusiasm and guidance in the preparation of this thesis.

I would also like to acknowledge the National Science Foundation which has provided financial support for me. I want to thank those with whom I was associated at the Stanford University Artificial Intelligence Project for their assistance and the use of their facilities.

ALGORITHMS FOR POLYNOMIALS OVER A REAL ALGEBRAIC NUMBER FIELD

BY

CYRENUS MATTHEW RUBALD

ABSTRACT

This thesis defines, implements, and rigorously analyzes the computing times of a set of algorithms for performing operations in $Q(\alpha)$ and $Q(\alpha)[x]$, where Q is the field of rational numbers and α is any real algebraic number. α must be specified by means of a primitive, square-free, integral polynomial, Ψ, which has α as a root and no real rational roots, and an interval, Ω, containing α but no other root of Ψ. Since Ψ need not be irreducible, we do not, in general, enjoy the advantages of a canonical form; however, the problem of polynomial factorization is avoided. An algorithm is provided to determine a suitable Ψ from any integral polynomial P such that $P(\alpha) = 0$. Making use of an evaluation homomorphism, ψ_α, elements of $Q(\alpha)$ and $Q(\alpha)[x]$ are represented by elements of the residue class ring $Q[y]/(\Psi(y))$ and the polynomial ring $Q[y,x]/(\Psi(y))$ respectively. The operations performed in $Q(\alpha)$ and $Q(\alpha)[x]$ include addition, multiplication, and division. In $Q(\alpha)$, the order relation is algorithmically realized and an algorithm for computing reciprocals is presented. In $Q(\alpha)[x]$,

two algorithms for greatest common divisor calculation are presented. For each operation an algorithm with a polynomial computing time bound is obtained. Because of the unusual data representation employed and the utilization of such techniques as modular algorithms and the theory of subresultants, the procedures presented for these operations are not, in general, the classical algorithms. Empirical results from certain examples are presented.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

Section 1.1. Motivation

Let F, K be fields such that F is a subfield of K. An element α belonging to K is said to be algebraic over F if there exists a polynomial, A, of positive degree belonging to $F[x]$ such that $A(\alpha) = 0$. If A is monic and of degree less than or equal to the degree of any polynomial of $F[x]$ of which α is a root, then A is called a minimal polynomial for α over F. One can show (see [HER64], for example) that such an A is unique and irreducible over F. It also follows that A divides any other element of $F[x]$ of which α is a root.

Following standard conventions, let $F(\alpha)$ denote the field formed by the intersection of all fields which contain F and α. $F(\alpha)$ is called the field obtained by adjoining α to F, and is said to be a simple algebraic extension of F. It can be shown that $F(\alpha)$ is isomorphic to the residue class ring $F[x]/(A(x))$ if A is the minimal polynomial for α over F. If $\alpha_1,\ldots,\alpha_n$ are algebraic over F, then the field formed by the intersection of all fields which contain F and $\alpha_1,\ldots,\alpha_n$ is called a multiple algebraic extension of F, if $n > 1$, and is denoted by $F(\alpha_1,\ldots,\alpha_n)$.

Our interest lies in the case where F is Q, the field of rational numbers, K is the field of algebraic numbers, and α is real and irrational. We will present algorithms for performing addition, multiplication, and division in $Q(\alpha)$ and $Q(\alpha)[x]$. In $Q(\alpha)$ an algorithm is given for deciding the order relation between any given elements, and an algorithm for the computation of recip-

rocals is presented. In $Q(\alpha)[x]$, two algorithms for greatest common divisor calculation are specified. All of this is performed without recourse to the minimal polynomial of α. Furthermore, for each operation in $Q(\alpha)$ or $Q(\alpha)[x]$, at least one algorithm with a polynomial computing time bound is presented (see Section 1.3).

A motivation for devising these algorithms with the characteristics specified above, stems from the quantifier elimination method and decision procedure for the elementary theory of real closed fields devised by G. E. Collins (see [COL73c]). We will discuss this method very briefly and show the application of our work to the algorithm Collins has devised.

A quantifier elimination algorithm was originally developed by A. Tarski (see [TAR51]), and other algorithms for this purpose were presented in [SEI54] and [COH69]. All three procedures share the common fault of having an exponential maximum computing time for formulas in any fixed number, r, of variables, at least for $r \geq 2$, and are too impractical to be implemented on a computer and utilized. The algorithm of G. E. Collins, on the other hand, using the algebraic number system presented here, has only a polynomial computing time bound and is readily implementable on a digital computer. In fact, there is some indication that the latter algorithm may be nearly best in a certain sense.

The elementary theory of real closed fields is a first order theory where the variables range over some real closed field (see section 11.5 of [VDW70] for information about real closed fields).

But by the Completeness Theorem of elementary real closed fields, any property expressible in the elementary theory of real closed fields that is true in one real closed field is true in all real closed fields. Hence, it suffices to consider that the variables range over the field of real numbers. The non-logical constants consist of the operation symbols: $\cdot$, $+$; the relation symbol: $>$; and the individual constant: -1. The usual logical symbols, $\sim$, $\vee$, $\wedge$ are included, as well as the quantifiers $\forall x$ - the universal quantifier - and $\exists x$ - the existential quantifier. Algebraic terms are made up from variables, $\cdot$, $+$, and -1.

Collins defines <u>standard atomic formulas</u>, which are of the form $A > 0$, $A = 0$, $A < 0$, $A \neq 0$, $A \geq 0$, or $A \leq 0$, where A is an algebraic term. A <u>standard formula</u> is a formula built up from propositions of standard atomic formulas. The algorithm accepts as input a <u>standard prenex formula</u>, that is, one of the form $(Q_k x_k)\ldots(Q_r x_r)\phi(x_1,\ldots,x_r)$ where each Q_i is "$\forall$" or "$\exists$" for $k \leq i \leq r$; ϕ is a quantifier-free standard formula; and $1 \leq k \leq r$. The output is $\psi(x_1,\ldots,x_{k-1})$, where ψ is a quantifier-free standard formula which is equivalent to $(Q_k x_k)\ldots(Q_r x_r)\phi(x_1,\ldots,x_r)$. If $k = 1$, ψ is the truth-value of $(Q_1 x_1)\ldots(Q_r x_r)\phi(x_1,\ldots,x_r)$.

During the course of the computation of ψ, a certain set of univariate polynomials over the integers, $\{A_1,\ldots,A_m\}$, is determined. If $\alpha_1,\ldots,\alpha_h$ are the distinct real, irrational roots of the A_i, then certain operations, including g.c.d. calculations, are performed on univariate polynomials over $Q(\alpha_j)$, $1 \leq j \leq h$. At another point

in the algorithm, another set of univariate polynomials over the integers, $\{B_1,\ldots,B_n\}$, is determined. If $\beta_1,\ldots,\beta_\ell$ are the distinct real, irrational roots of the B_i, then Sturm's Theorem (see [HEI70]) is applied to isolating the roots of certain univariate polynomials in $Q(\beta_j)[x]$, $1 \leq j \leq \ell$. While not all of the algorithms needed for the latter application appear in this thesis, they follow readily from the material which does appear here.

Note that if one were to factor the A_i, B_j over Q, one could take advantage of the well known isomorphism mentioned above in constructing algorithms for operations in $Q(\alpha)$ and $Q(\alpha)[x]$. However, although there exist powerful algorithms for performing this factorization (see [MUS71]), all known methods have exponential maximum computing time functions. Consequently, one of the important aspects of this thesis is that the minimal polynomial of α need not be specified for the procedures in $Q(\alpha)$ and $Q(\alpha)[x]$ to be effective. All that is required is a polynomial of which α is a root, Ψ, with certain characteristics to be described later, and an interval, Ω, containing α and no other root of Ψ. Ψ can easily be computed from any polynomial having α as a root using an algorithm supplied in this thesis, while procedures for computing Ω appear in [HEI70]. Since the algorithms for determining Ψ and Ω have polynomial computing time functions, this property is preserved for the quantifier elimination method and decision procedure.

A secondary motivation for this thesis lies in the research of G. E. Collins and Rüdiger Loos associated with the field of all

algebraic numbers (see [COLO73] and [LOO73]). They have devised an algorithm which reduces a multiple extension field to an equivalent simple extention field. This algorithm assumes the availability of a subalgorithm for g.c.d. calculation in $Q(\alpha)[x]$. The Collins quantifier elimination algorithm uses this reduction algorithm.

Section 1.2. The SAC-1 System

SAC-1 (Symbolic and Algebraic Calculations - version 1) is a FORTRAN based, computer-independent system for performing a variety of tasks on many different algebraic structures. With the exception of a few assembly language programs called primitives, SAC-1 consists of a large number of FORTRAN subroutines, each performing some list processing or algebraic operation. Because these subroutines are written strictly in ASA FORTRAN and the number of primitives is small, the system is highly portable, as attested to by the large number of different computers on which it is implemented.

SAC-1 is actually a hierarchy of subsystems or modules, each consisting of a number of subroutines which perform related tasks. Each module depends directly on one or more (and perhaps indirectly, on many) other modules of the system, except for the List Processing System which stands alone.

To use SAC-1, one writes a FORTRAN program which makes the usual FORTRAN subroutine and function calls to the desired SAC-1 routines. Hence, no new syntax must be learned if that of FORTRAN is familiar to the user. In his program, the user must initialize certain variables, since he is afforded a large amount of freedom in determining run time parameters such as the amount of core to be utilized for list processing. The set of variables which must be initialized varies depending on what subroutines will be used; however, one of the most elementary is β (represented by BETA in a labeled common block) which is the radix to be used by the Integer

Arithmetic System. The interested reader is referred to the references given below for information on other of these parameters.

This large, powerful system cannot be adequately described here, but we will sketch each of the existing subsystems.

The List Processing System (see [COL71c]) utilizes the reference count method of list processing developed by G. E. Collins (see [KNU68] for a discussion of list processing techniques). This method allows for the overlapping of lists, which can mean a considerable saving in storage. Additionally, the reference count method returns cells to available space in time proportional to the number of cells returned, while other methods require time proportional to the number of cells being used. The price of this added efficiency is that the user must keep track of his own lists.

The Integer Arithmetic System (see [COL73d]) presents a recently revised set of subroutines for performing all of the commonly needed arithmetic operations on infinite-precision integers. The best known algorithms for performing these operations are incorporated into this system. An integer is represented as a first order list where the i^{th} element of the list is the i^{th} β-digit of the number.

The Polynomial System (see [COL71a]) provides subroutines for arithmetic, substitution, and differentiation of multivariate polynomials over the integers. Polynomials are represented in recursive canonical form; that is, a polynomial in v variables is considered to be a polynomial in one variable with coefficients which are polynomials in $v-1$ variables.

The Rational Function System (see [COL71d]) performs arithmetic operations and substitution on multivariate rational functions. Such a function is represented by a pair of relatively prime integral polynomials, with the leading coefficient of the denominator positive.

The Modular Arithmetic System (see [COAL69]) presents algorithms for performing a wide variety of tasks in $GF(p)[x_1,\ldots,x_v]$, $v \geq 0$, where p is a single precision prime.

The Partial Fraction Decomposition and Rational Function Integration System (see [COHO70]), the Polynomial Factorization System (see [COMU72]), and the Polynomial Linear Algebra System (see [COMC72]), perform operations which are obvious from their titles, the first two for univariate input only.

The Polynomial GCD and Resultant System (see [COL72]) implements the modular algorithms of G. E. Collins for computing g.c.d.'s and resultants of multivariate integral polynomials.

The Polynomial Real Zero System (see [COHE70]) isolates the real roots of a univariate integral polynomial into disjoint intervals, and refines the intervals to an arbitrary precision.

The Gaussian Polynomial Complex Zeros System (see [COPI73]) isolates the complex roots of a Gaussian polynomial into disjoint squares, and refines the squares to an arbitrary precision.

The Gaussian Integer and Polynomial System (see [CACO73]) extends the Integer Arithmetic and Polynomial Systems to Gaussian Integers and Polynomials.

The algorithms in this thesis are presented in a style consistent with that of most of the other publications utilizing SAC-1 which have been referenced here. To those unfamiliar with any such publications, the form should still be easily understood since the algorithm descriptions readily translate into high level computer languages, and this is reflected in the syntax. Also, for non-trivial algorithms, a discussion of the algorithm preceeds it, and each step has a short explanatory note accompanying it.

Countless references to algorithms and computing time bounds for algorithms appearing in [COL71c], [COL73d], [COL71a], [COAL69], [COHE70], [COL72], and [COPI73] appear throughout this thesis. We explain the relevant properties of the more complex algorithms referenced from these sources, and the mnemonic conventions of SAC-1 (for example PSUM sums two polynomials and IPROD forms the product of two integers) usually readily identify the more simple operations performed. Consequently, if one is willing to accept the computing time bounds and certain data structures from the above references on faith, an informal understanding of this thesis may be realized without recourse to the above documents. For a thorough understanding, however, using the above documents is encouraged.

For a good overview of SAC-1 and many of its conventions, capabilities, and algorithms, see [COL71b] and [COL73a].

Section 1.3. Computing Time Analysis

Each algorithm we will present contains a description which consists of a finite number of numbered steps. Each step consists of a finite sequence of executable statements, each of which is performed a finite number of times. Each such statement defines, explicitly or implicitly, a finite sequence of basic operations which will be performed, should the algorithm be implemented and executed with valid input.

Basic operations consist of such things as single precision addition and replacement, unconditional transfers, and subroutine calls. We could define specifically what we mean by basic operations, but such a definition would be dependent on the particular computer on which the algorithm is implemented. Instead, we note two important characteristics of these operations which we will use to divorce ourselves from any particular machine. First, the time to perform a basic operation on any fixed computer has an upper and lower bound. Second, if B_1, B_2 are two sets of basic operations, any operation from B_1 can be performed as a finite number of operations from B_2. If any computer should ever exist for which either of these conditions fails, it would have to be excluded from the set over which we are generalizing.

Given an algorithm description and a valid input (which may be an n-tuple) to the algorithm, the execution of the algorithm with the given input consists of the execution of a finite number of basic operations. One can determine exactly how many such operations

will be performed. This number is called the computing time of the algorithm for the input, since by using the two characteristics of basic operations mentioned above, one can determine upper and lower bounds on the execution time of the algorithm on any given machine. If A is the name of the algorithm and I is the input, the computing time is abbreviated $t_A(I)$.

In taking this approach we can meaningfully discuss the computing time of an algorithm independent of any implementation. Even though we lose the ability to predict exactly the computing time of A with input I should we implement A on some particular machine, we do have bounds for A with input I on any machine in general.

Freed of the restriction of determining computer-dependent constants by our computer-independent approach, we see that the theoretical computing time we have described above is a function strictly of the inputs to the algorithm. This ability to ignore machine-oriented aspects of a procedure, coupled with a desire to compare or incorporate the computing times of various algorithms, leads us naturally to the following definition.

Definition 1.3.1. (G. E. Collins) Let f, g be real-valued functions on an arbitrary common set S. f, g may be multivariate in which case the elements of S are n-tuples. If there exists a positive real constant c such that $f(x) \leq cg(x)$ for all x belonging to S, then we say f is dominated by g (g dominates f), and write $f \prec g$ ($g \succ f$). If $f \prec g$ and $g \prec f$ then we say that

f and g are <u>codominant</u>, and write $f \sim g$. If $f \preceq g$ but it is not the case that $g \preceq f$, then we say that f is <u>strictly dominated by</u> g, and write $f \prec g$.

Codominance is an equivalence relation on the set of all real-valued functions with domain S. Also, the dominance relation is a partial ordering of the set of codominance equivalence classes of these functions.

The following theorem lists some important properties of dominance and codominance which follow easily from the definition. We will make frequent use of these properties in later chapters without referencing this theorem.

<u>Theorem 1.3.1</u>. (G. E. Collins) Let f, f_1, f_2, g, g_1, g_2 be non-negative, real valued functions on S. Then

(1) c a positive real number implies that $cf \sim f$,

(2) $f_1 \preceq g_1$, $f_2 \preceq g_2$ implies that $f_1 + f_2 \preceq g_1 + g_2$ and $f_1 \cdot f_2 \preceq g_1 g_2$,

(3) $f_1 \prec g$, $f_2 \prec g$ implies that $f_1 + f_2 \prec g$,

(4) $\max(f,g) \sim f + g$

(5) $1 \preceq f$, $1 \preceq g$ implies that $f + g \preceq f \cdot g$,

(6) $1 \preceq f$, c a positive real number implies that $f \sim f + c$,

(7) $S = S_1 \times \cdots \times S_n$, $f \preceq g$ implies that $f(a_1,x_2,\ldots,x_n) \preceq g(a_1,x_2,\ldots,x_n)$ for any a_1 in S_1,

(8) $f \preceq g$ on S_1, $f \preceq g$ on S_2 implies that $f \preceq g$ on $S = S_1 \cup S_2$.

For each algorithm A appearing in this thesis we will determine a function f such that for valid input I, $t_A(I) \preceq f(I) = g(c_1(I),\ldots,c_n(I))$, where g is a real-valued function and $c_i(I)$ is some characteristic of I, $1 \leq i \leq n$. For example, suppose A is IPROD, an algorithm for computing the product of two infinite-precision integers, I is the two-tuple of integers (a,b). Then $n = 2$, $c_1(I)$ is the length of a, $c_2(I)$ is the length of b, $g(x,y) = xy$ is one possible determination of f. As stated in Section 1.1, one important aspect of this thesis is that at least one algorithm for each operation has a bound f, which is a polynomial, not an exponential function.

We will now define some related concepts we will find useful. Suppose the set S of valid inputs to A can be expressed as a denumerable union of disjoint finite sets: $S = \bigcup_{i=1}^{\infty} S_i$. We define the <u>maximum computing time function</u> of A relative to the decomposition of S to be $t_A^+(S_i) = \max\{t_A(I) : I \in S_i\}$. If the decomposition of S is clearly defined from context, then we also write $t_A^+(i)$ for $t_A^+(S_i)$. Note also that $t_A(I) \leq t^+(S_i)$ if I is an element of S_i. In analyzing A, we let N_i stand for the number of executions of step (i), t_i the maximum computing time for a single execution of step (i), T_i the total computing time of step (i). For other related concepts, see [COL73b].

Computing time functions for algorithms from other modules of SAC-1 will be referenced a countless number of times in determining bounds for the computing times of algorithms in this thesis. Each

such function appears in one of the following: [COAL69], [COL72], [COHE70], or [COPI73].

Section 1.4. Fundamental Concepts

Throughout the course of this thesis we shall employ various terms and relationships which have come to be accepted as standard, as well as others which are more recently defined but very useful. Since not all readers of this paper may be acquainted with these terms and relationships, this section is devoted to presenting definitions and theorems on varying subjects, as well as giving some standard notation.

We first note that by an integral domain we mean a commutative ring with identity which has no zero divisors.

Definition 1.4.1. The sign of an element, a, of an ordered integral domain, sign(a), is defined to be +1, 0, or -1 depending on whether a is positive, zero, or negative, respectively.

If A is any non-zero polynomial in $r \geq 1$ variables, $x_1,\ldots,x_r$, we denote the degree of A in x_i by "$\partial_i(A)$"; however we frequently use "deg(A)" in place of "$\partial_r(A)$", the degree of A in its main variable. If A = 0, then $\partial_i(A) = 0$ for $1 \leq i \leq r$.

Definition 1.4.2. If A is any polynomial in $r \geq 1$ variables, $x_1,\ldots,x_r$, then the degree vector of A, $\partial(A)$, is defined to be the r-tuple $(\partial_1(A),\partial_2(A),\ldots,\partial_r(A))$.

If A(x) is an element of R[x] where R is some ring, then we abbreviate the leading coefficient of A by, "ldcf(A)" (If A = 0 then ldcf(A) = 0).

If I is an ordered integral domain then the ordering of I can be extended to $I[x]$. If $A(x) \in I[x]$ then define $A(x)$ to be positive if and only if ldcf(A) is a positive element of I. Hence we give the following definition.

Definition 1.4.3. If R is an ordered integral domain and $A(x)$ is an element of $R[x]$, then the sign of $A(x)$, sign(A), is defined to be the sign (ldcf(A)).

Definition 1.4.4. If R is any commutative ring with identity and a is an element of R, then a is called a unit in case a divides the identity of R. Two elements a, b, of R are said to be associates if $a = ub$ for some unit, u, in R. In this case we write $a \sim b$. It is easy to see that $\sim$ is an equivalence relation.

Definition 1.4.5. Suppose I is an integral domain and that $A(x)$, $B(x)$ are elements of $I[x]$. If there exist non-zero elements a, b in I such that $aA(x) = bB(x)$ then $A(x)$ and $B(x)$ are said to be similar, and a,b are called coefficients of similarity. We write $A \approx B$.

If a,b are elements of some ring, R, and a divides b in R, then we abbreviate this $a|b$; if a does not divide b in R, then we abbreviate this $a \nmid b$.

Definition 1.4.6. Let I be an integral domain and let a,b be elements of I. A common divisor of a,b is any element, d, of I such that $d|a$, $d|b$. A greatest common divisor, g.c.d., of a,b is a common divisor, c, of a,b such that if d is any common divisor of a,b, then $d|c$. G. E. Collins gives us the definition of a g.c.d. domain, which is an integral domain in which any two elements have a g.c.d.

Note that if I is a g.c.d. domain and A is an ample set for I, that is, a set which contains exactly one element in each equivalence class of associates, we may speak about the unique g.c.d. of two elements in A. If a,b are elements of I and c is this unique g.c.d., we write $c = \gcd(a,b)$. For $n \geq 2$, we define $\gcd(a_1,\ldots,a_{n+1})$ by induction on n as $\gcd(\gcd(a_1,\ldots,a_n),a_{n+1})$. We define $\gcd(a)$ to be the associate of a in A. Hence we may talk about $\gcd(a_1,\ldots,a_n)$ for $n \geq 1$. Also, see [COL73a], Section 4.

Definition 1.4.7. Let I be a g.c.d. domain and let $A(x)$ be an element of $I[x]$ such that $A(x) = \sum_{i=0}^{n} a_i x^i$, and $A(x) \neq 0$. The content of A, cont(A), is $\gcd(a_n,\ldots,a_0)$. If cont(A) = 1 then A is called primitive. If $c = \text{cont}(A)$ then the primitive part of A, pp(A), is $A(x)/c$.

Definition 1.4.8. Let I be an integral domain and let a,b be elements of I. a is square-free in case there is no element, c, of I such that c is a non-unit and $c^2|a$. a is a square-free divisor of b if a is square-free and $a|b$. a is a greatest square-free divisor of b if a is a square-free divisor of b and whenever c is any square-free divisor of b, $c|a$.

If A is an ample set for I, we may speak about the unique greatest square-free divisor of a, gsfd(a), in A.

Let A,B be elements of $R[x]$, where R is an integral domain. Assume B is non-zero. Define $m = \deg(A)$, $n = \deg(B)$, $b = \mathrm{ldcf}(B)$, $k = \max(m-n+1,0)$. It can be shown that there exist unique polynomials Q_1, C_1 belonging to $R[x]$ such that

$$b^k A(x) = B(x)Q_1(x) + C_1(x),$$

where $C_1 = 0$ or $\deg(C_1) < n$.

If b is a unit in R and we let $Q_2 = b^{-k}Q_1$, $C_2 = b^{-k}C_1$, then

$$A(x) = B(x)Q_2(x) + C_2(x),$$

where $C_2 = 0$ or $\deg(C_2) < n$.

Note that if $m < n$ then $k = 0$, $Q_1 = Q_2 = 0$, and $C_1 = C_2 = A$.

Definition 1.4.9. Q_1, C_1, as defined above, are called the pseudo-quotient and pseudo-remainder, respectively, of A,B. This is abbreviated $Q_1 = \mathrm{pquo}(A,B)$ and $C_1 = \mathrm{prem}(A,B)$. Q_2, C_2, as defined above, are called the quotient and remainder, respectively, of A,B. This is abbreviated $Q_2 = \mathrm{quo}(A,B)$ and $C_2 = \mathrm{rem}(A,B)$. We also denote $\mathrm{rem}(A,B)$ by A modulo B.

We now present some definitions and an important theorem, given without proof, of results involving subresultants and polynomial remainder sequences. For a proof of the theorem and complete discussions of these concepts and how they relate, see [COL67], [COL71e], [COL73a], and [BRO71], [BRTR71]. Also [VDW70] contains useful information about resultants.

Definition 1.4.10. Let $A(x)$, $B(x)$ be non-zero elements of $I[x]$, where I is an integral domain. Let $A_1,A_2,\ldots,A_{k+2}$ be a sequence of polynomials such that $A_1 \approx A$, $A_2 \approx B$, and $A_{i+2} \approx \mathrm{prem}(A_i,A_{i+1})$ for $1 \le i \le k$, and $A_{k+2} = 0$. $A_1,A_2,\ldots,A_{k+2}$ is said to be a polynomial remainder sequence, p.r.s., for A and B. Let $n_i = \deg(A_i)$ for $1 \le i \le k+1$. $n_1,\ldots,n_{k+1}$ is called the degree sequence of A, B.

If $A_1,\ldots,A_{k+2}$ is any p.r.s. for A, B, then there exist non-zero $e_i \in I$, $Q_i \approx \mathrm{pquo}(A_i,A_{i+1})$, for $1 \le i \le k$, and non-zero $f_i \in I$ for $1 \le i \le k-1$, such that

$$e_i A_i = Q_i A_{i+1} + f_i A_{i+2},$$

for $1 \le i \le k$. Also, A_1,A_2,e_i,Q_i for $1 \le i \le k$, f_i for $1 \le i \le k-1$, are sufficient to uniquely determine the p.r.s. Depending on the choice of the e_i's and f_i's, the p.r.s. resulting may be over the fraction field of I.

Polynomial Remainder Sequences derive their importance by virtue of the fact that $\gcd(A_1,A_2) \approx \gcd(A_2,A_3) \approx \cdots \approx \gcd(A_{k+1},A_{k+2}) \approx A_{k+1}$. This can be seen by examining the definition of a p.r.s., and noting the analogy with the Euclidean algorithm for computing g.c.d.'s of integers. Note that if the p.r.s. is generated over a field, then A_{k+1} itself is a g.c.d. of A_1,A_2.

There are several different kinds of p.r.s.'s. The ones we will be most interested in are the natural p.r.s., where $e_i = f_i = 1$ for $1 \leq i \leq k$ and the fraction field of I is needed, as well as the subresultant p.r.s., where the e_i, f_i are chosen so that the A_i are certain subresultants. We now proceed to define resultants and subresultants.

Definition 1.4.11. Let $A(x)$, $B(x)$ be elements of $R[x]$, where R is a commutative ring with identity. Let $m = \deg(A)$, $n = \deg(B)$, and assume m,n are non-zero. We define the Sylvester matrix of A and B, $M(A,B)$, by

$$\begin{bmatrix} a_m & a_{m-1} & \cdots & & a_0 & & \\ & a_m & a_{m-1} & \cdots & & a_0 & \\ & & & & & \vdots & \\ & & a_m & a_{m-1} & \cdots & & a_0 \\ b_n & b_{n-1} & \cdots & b_0 & & & \\ & b_n & b_{n-1} & \cdots & b_0 & & \\ & & & & \vdots & & \\ & & & b_n & b_{n-1} & \cdots & b_0 \end{bmatrix}$$

where $A(x) = \sum_{i=0}^{m} a_i x^i$ and $B(x) = \sum_{j=0}^{n} b_j x^j$, $M(A,B)$ has n rows of A coefficients, m rows of B coefficients and hence is $m+n$ by $m+n$, and the elements not shown are zero. The resultant of A and B, $\mathrm{res}(A,B)$, is the determinant of $M(A,B)$, $\det(M(A,B))$, an

element of R.

Note that if one interchanges the rows of $M(A,B)$ so that the coefficients of $B(x)$ appear in the first m rows, we have that $\mathrm{res}(A,B) = (-1)^{mn}\,\mathrm{res}(B,A)$.

The first theorem we present is a well known result we will use frequently.

Theorem 1.4.1. Let A,B be polynomials over an integral domain such that $\deg(A)$, $\deg(B) > 0$. Then $\mathrm{res}(A,B) = 0$ if and only if A and B have a common factor of positive degree.

Definition 1.4.12. Define $A,B,M(A,B)$ as above. Let $m = \deg(A)$, $n = \deg(B)$. Then for $0 \leq j \leq \min(m,n)$ define the matrix $M_j(A,B)$ to be the submatrix of the Sylvester matrix obtained by deleting the last j rows of A coefficients, the last j rows of B coefficients, and the last j columns. Given j, define $M_{j,i}$ to be the matrix obtained from M_j by deleting all of the last $j+1$ columns except column $m+n-j-i$, for $0 \leq i \leq j$. The j^{th} subresultant of A and B, $S_j(A,B)$, is $\sum_{i=0}^{j} \det(M_{j,i})x^i$. Note that $S_0(A,B) = \mathrm{res}(A,B)$.

The following was independently discovered by Brown and Collins, and appears in [BRTR71].

Theorem 1.4.2. (The Fundamental Theorem of Subresultants and P.R.S.'s.) Let $A(x)$, $B(x)$ be non-zero elements of $I[x]$, where I is an integral domain. Define $m = \deg(A)$, $n = \deg(B)$ and assume $m \geq n$. Let $A_1 = A$, $A_2 = B, \ldots, A_{k+2} = 0$ be a p.r.s. for A and B defined by

$$e_i A_i = Q_i A_{i+1} + f_i A_{i+2} ,$$

where e_i, f_i are non-zero elements of I for $1 \leq i \leq k$. Furthermore, define $n_i = \deg(A_i)$, $a_i = \mathrm{ldcf}(A_i)$, $\delta_i = n_i - n_{i+1}$, for $1 \leq i \leq k$. Then the following relations hold.

(1) $$\{\prod_{i=1}^{j-2} e_i^{n_{i+1}-n_j}\} S_{n_j}(A,B) = \{\prod_{i=1}^{j-2} (-1)^{(n_i-n_j)(n_{i+1}-n_j)} f_i^{n_{i+1}-n_j} a_{i+1}^{\delta_i+\delta_{i+1}}\} a_j^{\delta_{j-1}-1} A_j \quad \text{for } 3 \leq j \leq k+1,$$

(2) $$\{\prod_{i=1}^{j-2} e_i^{n_{i+1}-n_{j-1}+1}\} a_{j-1}^{\delta_{j-1}-1} S_{n_{j-1}-1}(A,B) = \{\prod_{i=1}^{j-2} (-1)^{(n_i-n_{j-1}+1)(n_{i+1}-n_{j-1}+1)} f_i^{n_{i+1}-n_{j-1}+1} a_{i+1}^{\delta_i+\delta_{i+1}}\} A_j \quad \text{for } 3 \leq j \leq k+1,$$

(3) $$S_\ell(A,B) = 0 \quad \text{for } n_j < \ell < n_{j-1} - 1 \text{ and } 3 \leq j \leq k+1,$$

and for $\ell < n_{k+1}$.

Note that $A_1, A_2, S_{n_2-1}, S_{n_3-1}, \ldots, S_{n_{k+1}-1} = 0$, and $A_1, A_2, S_{n_3}, S_{n_4}, \ldots, S_{n_{k+1}}, S_{n_{k+2}} = 0$ form p.r.s.'s, the former usually being called the subresultant p.r.s. . Since we will employ both sequences in this thesis we make the following definition to distinguish them.

Definition 1.4.13. If $A(x)$, $B(x)$ are non-zero elements of $I[x]$ where I is an integral domain, and if $\deg(A) \geq \deg(B) > 0$, then we make the following definitions. $A, B, S_{n_2-1}, S_{n_3-1}, \ldots, S_{n_{k+1}-1}$ is called the subresultant p.r.s. of (A,B) of the first kind. $A, B, S_{n_3}, S_{n_4}, \ldots, S_{n_{k+2}}$ is called the subresultant p.r.s. of (A,B) of the second kind. S_i, n_i are as defined in Theorem 1.4.2.

If we speak ambiguously of the "subresultant p.r.s. of (A,B)", we shall mean either of the p.r.s.'s. In Theorem 1.4.2, note that if $\delta_{j-1} = 1$, then $n_j = n_{j-1} - 1$, and (1), (2) are identical.

Definition 1.4.14. A Gaussian integer is a number of the form $a + bi$ where a and b are rational integers. A Gaussian polynomial is a polynomial with Gaussian integer coefficients; these include the integral polynomials which are polynomials with rational integer coefficients.

Definition 1.4.15. Let A be a Gaussian polynomial in r variables, $x_1, \ldots, x_r$, $r \geq 0$. We define inductively the max norm of A, $|A|_\infty$, and the sum norm of A, $|A|_1$. If $r = 0$ (A is a Gaussian integer) then

$$|A|_\infty = \max(|a|, |b|),$$

and

$$|A|_1 = |a| + |b|,$$

where $A = a + bi$ and $|\ |$ is the usual absolute value of a rational integer. If $r > 0$ then $A(x_1, \ldots, x_r) = \sum_{i=0}^{n_r} A_i(x_1, \ldots, x_{r-1}) x_r^i$, where $n_r = \partial_r(A)$. In this case

$$|A|_\infty = \max_{0 \leq i \leq n_r} (|A_i|_\infty),$$

and

$$|A|_1 = \sum_{i=0}^{n_r} |A_i|_1 .$$

For integral polynomials, the sum norm and its properties were introduced by G. E. Collins in [COL69] and the max norm was suggested by W. S. Brown. Both were generalized to Gaussian polynomials in [COL73b]. A list of their properties appears in the following theorem.

Theorem 1.4.3. If A,B are Gaussian polynomials then

(a) $|A+B|_\infty \leq |A|_\infty + |B|_\infty$,

(b) $|A+B|_1 \leq |A|_1 + |B|_1$,

(c) $|AB|_\infty \leq |A|_\infty |B|_1$,

(d) $|AB|_1 \leq |A|_1 |B|_1$,

(e) $|A|_\infty \leq |A|_1$.

Definition 1.4.16. For a rational integer a, we define the length of a, $L(a)$, to be

$$L(a) = \lceil \log_\beta(|a|+1) \rceil ,$$

if $a \neq 0$, and

$$L(a) = 1$$

if $a = 0$, by convention. $\lceil \ \rceil$ is the ceiling function defined in [KNU68].

$L(a)$ is just the number of digits in the β-radix representation of a. Note that if b is a rational integer greater than zero and if $n = \log_\beta b$, $m = \log_\gamma b$ for β,γ rational integers greater than one, then $n = (\log_\beta \gamma)m$. Therefore $m \sim n$ and the definition of $L(a)$ is really independent of the radix being used to represent a. Hence when used in the computing time analysis of an algorithm, the length function is independent of any computer implementation of the algorithm.

It is easy to see that if a, b, are non-zero integers then

$$L(ab) \sim L(a) + L(b),$$

$$L(a^n) \sim nL(a),$$

if $n > 0$ and $a \geq 2$, and

$$L([a/b]) \sim L(a) - L(b) + 1,$$

if $|a| \geq |b| > 0$, and $[a/b]$ is the integral part of a/b.

It can also be shown that

(1) $$L(\prod_{i=1}^{n} a_i) \leq \sum_{i=1}^{n} L(a_i),$$

for $a_1,\ldots,a_n$ integers; and

(2) $$L(\prod_{i=1}^{n} a_i) \sim \sum_{i=1}^{n} L(a_i),$$

for $a_1,\ldots,a_n$ integers, n a variable, and $|a_i| \geq 2$ for $1 \leq i \leq n$.

Definition 1.4.16 and the material which follows it are from [COL73b], Section 3. The discussion there is more detailed and includes proofs for (1) and (2).

Definition 1.4.17. For real numbers a,b such that $a \leq b$, define $(a,b]$ to be the set $\{x: a < x \leq b\}$. Also define the interval $I = (a,b]$ to be the set $(a,b] \cup \{b\}$. If $J = (c,d]$ is also an interval, then we say $I < J$ if $b < c$ and $I \leq J$ if $b < c$ or $b = c$. Note that $(a,a] = \{\}$ but that $[a,a] = \{a\}$.

Definition 1.4.18. Let $I = (a,b]$ be an interval such that a,b are rational numbers, $a = a_1/a_2$, $b = b_1/b_2$, and $\gcd(a_1,a_2) = \gcd(b_1,b_2) = 1$. We define the maximum magnitude of an end-point of I, $\mathrm{mep}(I)$, to be $\max(|a_1|,|a_2|,|b_1|,|b_2|)$.

In SAC-1 the interval I is represented by the list (a,b). See also Section 2.1 of [HEI70].

The next theorem we present is very straightforward and is used frequently in the chapters which follow. Apparently it has never appeared in the literature, so we will give it now.

Theorem 1.4.4. If a,b are positive real functions and n is a fixed non-negative integer, then $(a+b)^n \sim a^n + b^n$.

Proof. The statement obviously holds for $n = 0$. Suppose it is true for n, and let $c = \max(a,b)$. Then $(a+b)^{n+1} = (a+b)^n(a+b) \sim (a^n+b^n)(a+b) \sim c^n c = c^{n+1} = \max(a^{n+1},b^{n+1}) \sim a^{n+1} + b^{n+1}$. ∎

The definition and theorem which follow appear in [COHO73]. The definition has been modified slightly and includes the case of a polynomial with one root, but the theorem still holds with this additional case.

Definition 1.4.19. Let $A(x)$ be an element of $C[x]$, where C is the field of complex numbers. Assume the degree of $A(x)$ is $n \geq 1$, and that $A(x)$ has $k \leq n$ distinct roots, $\alpha_1,\ldots,\alpha_k$. If $k \geq 2$, we define the minimum root separation of A, $\mathrm{sep}(A)$, by

$$\mathrm{sep}(A) = \min_{1 \leq i < j \leq k} |\alpha_i - \alpha_j|,$$

and

$$\mathrm{sep}(A) = \infty,$$

if $k = 1$.

Theorem 1.4.5. Let A be a Gaussian polynomial of degree $n \geq 1$ with only simple roots, and let $d = |A|_1$. Then

$$\mathrm{sep}(A) > \tfrac{1}{2}(d\, e^{1/2}\, n^{3/2})^{-n},$$

where e is the base of the natural logarithm.

Theorem 1.4.6. Let A be a non-zero polynomial over I of degree $n \geq 0$, B be a factor of A over I of degree k, and let $m = \lceil (n+k)/2 \rceil$. Then

$$|B|_1 < (m+1)^{n+k+1}\, |A|_1 .$$

Corollary 1.4.1. For any factor B of A,

$$|B|_1 < (n+1)^{2n}\, |A|_1 .$$

The above theorem and corollary are from [MUS71] and are due to Collins. See also [KNU69].

Theorem 1.4.7. (Chinese Remainder Theorem) Let $m_1, m_2, \ldots, m_r$ be positive integers which are pair-wise relatively prime:

$$\gcd(m_i, m_j) = 1 \quad \text{if} \quad i \neq j .$$

Let $m = \prod_{i=1}^{r} m_i$, and let $a, u_1, u_2, \ldots, u_r$ be integers. There exists a unique integer u such that

$$a \leq u < a + m,$$

and

$$u \equiv u_j \pmod{m_j} \quad \text{for} \quad 1 \leq j \leq r.$$

This definition appears on pages 249-250 of [KNU69]; see also [BRO71] and [COL71e], [COL73a]. We shall apply this theorem where the m_i are distinct odd prime numbers, p_i, $1 \leq i \leq r$, which are just slightly less than the radix, β. The availability of these primes is assumed by the algorithms to be on a list called PRIME. In the SAC-1 implementation of this system, PRIME appears as the first element in common block TR4.

Let $m = \prod_{i=1}^{r} p_i$. If $a = -(m-1)/2$ and u is any integer such that $|u| < m/2$, then knowing u_j (which is u modulo p_j) for $1 \leq j \leq r$, one can compute u. The SAC-1 system uses the Chinese Remainder Theorem as implemented with Garner's scheme; see [COAL69] and [COL72], as well as the above references.

Definition 1.4.20. Let I be an integral domain and let a,b be elements of I. A common multiple of a,b is any element, e, of I such that $a|e$, $b|e$. A least common multiple, l.c.m., of a,b is a common multiple, d, of a,b such that $d|e$, where e is any common multiple of a,b. Note that 0 is the unique l.c.m. of a,b if $a = 0$ or $b = 0$.

Theorem 1.4.8. (G. E. Collins) If I is a g.c.d. domain and a,b are non-zero elements of I, then $ab/\gcd(a,b)$ is a l.c.m. of a,b.

Proof. Let $c = \gcd(a,b)$, $\bar{a} = a/c$, $\bar{b} = b/c$, $d = ab/c = a\bar{b} = \bar{a}b$. d is a common multiple of a and b. Let e be any common multiple of a and b. $a|e$ so $ab|be$, and $b|e$ so $ab|ae$. Hence ab is a common divisor of ae and be. By Lemma 3, page 119 of [JAC51], $\gcd(ae,be) \sim e \gcd(a,b) = ec$. Hence $ab|ce$ from which it follows that $d|e$. Therefore d is a least common multiple. ∎

It follows from this theorem that if I is a g.c.d. domain and A is an ample set for I, we may speak about the unique l.c.m. of a,b, abbreviated $\mathrm{lcm}(a,b)$, in A. If $a \neq 0$ and $b \neq 0$ then $\mathrm{lcm}(a,b)$ is the associate of $ab/\gcd(a,b)$ in A, and $\mathrm{lcm}(a,b) = 0$ if $a = 0$ or $b = 0$. For $n \geq 2$, we define $\mathrm{lcm}(a_1, \ldots, a_{n+1})$ by induction on n as $\mathrm{lcm}(\mathrm{lcm}(a_1, \ldots, a_n), a_{n+1})$. We define $\mathrm{lcm}(a)$ to be the associate of a in A. Hence we may talk about $\mathrm{lcm}(a_1, \ldots, a_n)$ for $n \geq 1$.

CHAPTER 2: AUXILLIARY ALGORITHMS

Section 2.1. Introduction

As previously stated, the goal of this thesis is to construct and analyze a group of efficient algorithms for operations in $Q(\alpha)$ and $Q(\alpha)[x]$, where α is a real algebraic number. To this end, several algorithms have been devised which have proven useful in the quest of this goal, but which stand apart from the main algorithms and may be useful in themselves. This chapter deals with these procedures. They are grouped according to the type of operation they perform: list processing, integer arithmetic, rational number arithmetic or polynomial manipulation.

Section 2.2. List Processing Algorithms

The algorithms of this chapter are obvious extensions of algorithms from the SAC-1 List Processing System.

Algorithm 2.2.1. ADV2(a,b,L)

(Advance 2 Elements)

$L = (\ell_1,\ell_2,\ldots,\ell_n)$ where $n \geq 2$ is the input. After execution of ADV2, $a = \ell_1$, $b = \ell_2$ and $L = (\ell_3,\ldots,\ell_n)$.

Description

(1) ADV(a,L); ADV(b,L); return.

Theorem 2.2.1. Let L be a list. $t_{ADV2}(L) \sim 1$.

Proof. Obvious. ∎

Algorithm 2.2.2. DECAP2(a,b,L)

(Decapitate 2 Elements)

$L = (\ell_1,\ell_2,\ldots,\ell_n)$ where $n \geq 2$ is the input. After execution of ADV2, $a = \ell_1$, $b = \ell_2$ and $L = (\ell_3,\ldots,\ell_n)$. The first two cells of the inputlist are returned to the available space list without checking their reference counts.

Description

(1) DECAP(a,L); DECAP(b,L); return.

Theorem 2.2.2. Let L be a list. $t_{DECAP2}(L) \sim 1$.

Proof. Obvious. ∎

Algorithm 2.2.3. FIRST2(a,b,L)

(First 2 Elements)

L = $(\ell_1,\ell_2,\ldots,\ell_n)$ where $n \geq 2$ is the input. After execution of FIRST2, a = ℓ_1 and b = ℓ_2. L is unchanged.

Description

(1) T ← L; ADV(a,T); b ← FIRST(T); return.

Theorem 2.2.3. Let L be a list. $t_{FIRST2}(L) \sim 1$.

Proof. Obvious. ∎

The definition which follows is used in the next algorithm.

Definition 2.2.1. Let A = $(a_1,\ldots,a_m)$ and B = $(b_1,\ldots,b_n)$ be lists such that a_i, b_j are integers for $1 \leq i \leq m$ and $1 \leq j \leq n$. We define A < B if

(a) there exists i such that $1 \leq i \leq m,n$, $a_i < b_i$, and $a_j = b_j$ for $1 \leq j < i$, or

(b) $a_j = b_j$ for $1 \leq j \leq m$ and $m < n$. "<" is called the lexicographical ordering of all finite sequences of integers.

Algorithm 2.2.4. S = LEXORD(A,B)

(Lexicographical Ordering)

A and B are lists of FORTRAN integers. If "<" is the lexicographical ordering of all finite sequences of integers, then S is +1, 0 or -1 according as A > B, A = B or A < B.

Description

(1) A_1 ← A; B_1 ← B.

(2) If A_1 = 0, (if B_1 = 0, (S ← 0; return); S ← -1; return); if B_1 = 0, (S ← 1; return); ADV(a,A_1); ADV(b,B_1); if a = b, go to (2).

(3) If a > b, (S ← 1; return); S ← -1; return.

Theorem 2.2.4. Let A and B be lists of FORTRAN integers. Let m = length(A) and n = length(B). If k = min(m,n) then $t_{LEXORD}(A,B) \leq k + 1$.

Proof. $t_1,t_2,t_3 \sim 1$, $N_1,N_3 \sim 1$, and $N_2 \leq k + 1$. ∎

Section 2.3. Integer Arithmetic Algorithms

Definition 2.3.1. Let a,b be members of U, a unique factorization domain. Assume at least one of them is non-zero and let $c = \gcd(a,b)$. $\bar{a} = a/c$ and $\bar{b} = b/c$ are called the cofactors of a and b.

The concept of cofactors is often useful, in fact the motivation for performing a g.c.d. calculation is frequently to form the cofactors. An example of this is the determination of a canonical form for elements of the quotient field of U. In the algorithm which follows, the g.c.d. and cofactors of a and b are formed where U is the domain of integers.

Algorithm 2.3.1. IGCDCF(a,b,c,$\bar{a}$,$\bar{b}$)

(Integer G.C.D. and Cofactors)

The inputs are a and b, infinite-precision integers which are not both zero. The outputs are $c = \gcd(a,b)$ and the cofactors $\bar{a} = a/c$ and $\bar{b} = b/c$.

Description

(1) c ← IGCD(a,b); if FIRST(c) = 1 and TAIL(c) = 0, ($\bar{a}$ ← BORROW(a); $\bar{b}$ ← BORROW(b); return); $\bar{a}$ ← IQ(a,c); $\bar{b}$ ← IQ(b,c); return.

Theorem 2.3.1. Let a,b be integers at least one of which is non-zero. Let $m = \max(L(a),L(b))$, $n = \min(L(a),L(b))$ and $k = L(\gcd(a,b))$. $t_{IGCDCF}(a,b) \leq n(m-k+1)$.

Proof. $t_{IGCD}(a,b) \leq n(m-k+1)$, $t_{IQ}(a,c) \leq k(m-k+1) \leq n(m-k+1)$ and $t_{IQ}(b,c) \leq k(m-k+1) \leq n(m-k+1)$. ∎

A later section of this chapter will deal with rational number arithmetic. However, for the following algorithm it is necessary to know that if R is a non-zero rational number then R is represented by (r_1,r_2) where r_1, r_2 are integers, $\gcd(r_1,r_2) = 1$, $r_2 > 0$ and $R = r_1/r_2$.

Algorithm 2.3.2. a = IRNSPR(b,R)

(Integer-Rational Number, Special Product)

b is an infinite-precision integer and $R = (r_1,r_2)$ is a non-zero rational number such that $r_2 | b$. a is the infinite-precision integer $b \cdot R$.

Description

(1) a ← 0; if b = 0 return; FIRST2(r_1,r_2,R); t ← IQ(b,r_2); a ← IPROD(t,r_1); erase t; return.

Theorem 2.3.2. Let $R = (r_1,r_2)$ be a non-zero rational number, b an integer such that $r_2 | b$. If b = 0 then $t_{IRNSPR}(b,R) \sim 1$; if $b \neq 0$ then $t_{IRNSPR}(b,R) \leq n(m-L(r_2)+1)$, where $m = L(b)$ and $n = L(r_1) + L(r_2)$.

Proof. If b = 0 the proof is obvious. Suppose $b \neq 0$. $t_{IRNSPR}(b,R) \sim t_{IQ}(b,r_2) + t_{IPROD}(t,r_1) \leq L(r_2)\{m-L(r_2)+1\} + L(r_1)\{m-L(r_2)+1\} = n\{m-L(r_2)+1\}$. ∎

Section 2.4. Rational Number Algorithms

Let R be a rational number. If $R = 0$ the canonical form for R is the null list. If $R \neq 0$ let r_1, r_2 be integers such that $r_2 > 0$, $\gcd(r_1,r_2) = 1$ and $R = r_1/r_2$. The canonical form of R is the list (r_1,r_2) and we say $R = (r_1,r_2)$.

Definition 2.4.1. Let R be a rational number. We define the length of R, abbreviated $L(R)$, in the following manner. $L(R) = 1$ if $R = 0$, and $L(R) = L(r_1) + L(r_2)$ if $R \neq 0$ and $R = (r_1,r_2)$.

Although 0 is represented by the null list, it is frequently convenient when analyzing algorithms to treat 0 as if it were represented by the list (0,1). Thus if we speak about some general rational number $R = (r_1,r_2)$, if $R = 0$, r_1, r_2 are not undefined, but rather $r_1 = 0$ and $r_2 = 1$.

Algorithm 2.4.1. S = RNABS(R)

(Rational Number, Absolute Value)

R is a rational number, $S = |R|$.

Description

(1) If R = 0, (S ← 0; return); FIRST2(r_1,r_2,R); S ← PFL(IABSL(r_1), PFL(BORROW(r_2),0)); return.

Theorem 2.4.1. Let $R = (r_1,r_2)$ be a rational number. $t_{RNABS}(R) \preceq L(r_1)$.

Proof. If $R = 0$ then $t_{RNABS}(R) \sim 1 \sim L(r_1)$. For $R \neq 0$ we have $t_{RNABS}(R) \sim t_{IABSL}(r_1) \preceq L(r_1)$. ∎

Although ERASE, the general SAC-1 list erasure routine, will work when applied to a rational number, a more efficient algorithm for performing this task follows. RNERAS gains in efficiency by taking advantage of a knowledge of the structure of its input.

Algorithm 2.4.2. RNERAS(R)

(Rational Number Erasure)

R is a rational number. R is erased and set to zero.

Description

(1) If R = 0, return; n ← COUNT(R)-1; if n > 0, (SCOUNT(n,R) R ← 0; return); DECAP(r,R); ERLA(r); go to (1).

Theorem 2.4.2. Let R be a rational number, n the number of cells returned to the available space list by RNERAS(R). $t_{RNERAS}(R) \sim n+1$.

Proof. This follows directly from the computing time of ERLA. ∎

If a is a real number, then the floor function of a, $\lfloor a \rfloor$, is the greatest integer less than or equal to a (see [KNU68]).

Algorithm 2.4.3. n = RNFLOR(R)

(Rational Number Floor Function)

R is a rational number. $n = \lfloor R \rfloor$, an infinite-precision integer.

Description

(1) n ← 0; if R = 0, return; FIRST2(r_1,r_2,R); L ← IQR(r_1,r_2); DECAP2(n,t,L); if ISIGNL(t) < 0, n ← IMADD(n,1,-1); erase t; return.

Theorem 2.4.3. Let $R = (r_1,r_2)$ be a rational number. $t_{RNFLOR}(R) \preceq L(r_2)L(\lfloor R \rfloor)$.

Proof. If $R = 0$, $t_{RNFLOR}(R) \sim 1 = L(r_2)L(\lfloor R \rfloor)$. For $R \neq 0$, $t_{IQR}(r_1,r_2) \preceq L(r_2)L(\lfloor R \rfloor)$, $t_{ISIGNL}(t) \preceq L(r_2)$ and $t_{IMADD}(n,1,-1) \sim L(n) = L(\lfloor R \rfloor)$. ∎

Algorithm 2.4.4. R = RNINT1(a)

(Rational Number from Integer, 1 Argument)

a is an infinite-precision integer. R is a rational number in canonical form equivalent to a.

Description

(1) If a = 0, (R ← 0; return); R ← PFL(BORROW(a), PFL(PFA(1,0),0)); return.

Theorem 2.4.4. If a is an L-integer then $t_{RNINT1}(a) \sim 1$.

Proof. Obvious. ∎

Algorithm 2.4.5. R = RNINT2(a,b)

a and b are L-integers such that b is non-zero. R is the rational number in canonical form which is equivalent to a/b.

Description

(1) If a = 0, (R ← 0; return); IGCDCF(a,b,g,$\bar{a}$,$\bar{b}$); erase g.

(2) s ← ISIGNL($\bar{b}$); if s = 1, (R ← PFL($\bar{a}$,PFL($\bar{b}$,0)); return); R ← PFL(INEG($\bar{a}$), PFL(INEG($\bar{b}$),0)); erase $\bar{a}$, $\bar{b}$; return.

Theorem 2.4.5. Let a,b be L-integers, m = max(L(a),L(b)), n = min(L(a),L(b)), and k = L(gcd(a,b)). $t_{RNINT2}(a,b) \preceq n(m-k+1)$.

Proof. $t_{IGCDCF}(a,b) \preceq n(m-k+1)$ and $t_{ISIGNL}(\bar{b})$, $t_{INEG}(\bar{a})$, $t_{INEG}(\bar{b}) \preceq m-k+1$. Therefore $t_{RNINT2}(a,b) \preceq n(m-k+1)$. ∎

Algorithm 2.4.6. S = RNISPR(a,R)

(Rational Number-Integer, Special Product)

a is an infinite-precision integer and R is a rational number. S is the rational number a·R.

Description

(1) S ← 0; if a = 0 or R = 0, return; FIRST2(r_1,r_2,R).

(2) IGCDCF(a,r_2,g,$\bar{a}$,$\bar{r}_2$); s_1 ← IPROD($\bar{a}$,r_1); s_2 ← $\bar{r}_2$; erase g, $\bar{a}$; S ← PFL(s_1,PFL(s_2,0)); return.

Theorem 2.4.6. Let a be an infinite-precision integer and let $R = (r_1,r_2)$ be a rational number. $t_{RNISPR}(a,R) \preceq L(a)L(R)$.

Proof. Assume $a \neq 0$ and $R = (r_1,r_2) \neq 0$; $t_{IGCDCF}(a,r_2) \preceq L(a)L(r_2) \preceq L(a)L(R)$ and $t_{IPROD}(\bar{a},r_1) \preceq L(\bar{a})L(r_1) \preceq L(a)L(R)$. ∎

Algorithm 2.4.7. S = RNNEG(R)

(Rational Number Negation)

R is a rational number, S = -R.

Description

(1) S ← 0; if R = 0, return; FIRST2(r_1,r_2,R); S ← PFL(INEG(r_1), PFL(BORROW(r_2),0)); return.

Theorem 2.4.7. Let R = (r_1,r_2) be a rational number. $t_{RNNEG}(R) \preceq L(r_1)$.

Proof. Suppose $R \neq 0$. $t_{RNNEG}(R) \sim t_{INEG}(r_1) \preceq L(r_1)$. ∎

The Henrici-Brown multiplication algorithm is used to form the product of two rational numbers (see [COL71d]). This algorithm uses the fact that gcd(ac,bd) = gcd(a,d)gcd(b,c) when gcd(a,b) = gcd(c,d) = 1.

Algorithm 2.4.8. T = RNPROD(R,S)

(Rational Number Product)

R and S are rational numbers, T = R·S.

Description

(1) [Check for zero inputs.] If R = 0 or S = 0, (T ← 0; return); FIRST2(r_1,r_2,R); FIRST2(s_1,s_2,S).

(2) [Obtain the cofactors of r_1, s_2 and r_2, s_1.] IGCDCF(r_1,s_2, c,$\bar{r}_1$,$\bar{s}_2$); erase c; IGCDCF(r_2,s_1,c,$\bar{r}_2$,$\bar{s}_1$); erase c.

(3) [Form the numerator and denominator of T.] t_1 ← IPROD($\bar{r}_1$,$\bar{s}_1$); t_2 ← IPROD($\bar{r}_2$,$\bar{s}_2$); T ← PFL(t_1,PFL(t_2,0)); erase $\bar{r}_1$,$\bar{r}_2$,$\bar{s}_1$,$\bar{s}_2$; return.

Theorem 2.4.8. Let R and S be rational numbers. If R or S is zero then $t_{RNPROD}(R,S) \sim 1$. If R = (r_1,r_2) and S = (s_1,s_2) are non-zero define d = L(gcd(r_1,s_2)), e = L(gcd(r_2,s_1)), g = L(R), h = L(S) and k = min(d,e). $t_{RNPROD}(R,S) \preceq n(m-k+1)$, where n = min(g,h) and m = max(g,h).

Proof. If $R \neq 0$ and $S \neq 0$, then $t_{IGCDCF}(r_1,s_2) \preceq \min(L(r_1), L(s_2))\{\max(L(r_1),L(s_2))-d+1\} \leq \min(g,h)\{\max(g,n)-d+1\} \leq n(m-k+1)$, and similarly $t_{IGCDCF}(r_2,s_1) \preceq n(m-k+1)$. Also, $t_{IPROD}(\bar{r}_1,\bar{s}_1) \preceq (g-d+1)(h-e+1) \leq (g-k+1)(h-k+1) = (n-k+1)(m-k+1) \leq n(m-k+1)$, and similarly $t_{IPROD}(\bar{r}_2,\bar{s}_2) \preceq n(m-k+1)$. ∎

Algorithm 2.4.9. S = RNRECP(R)

(Rational Number Reciprocal)

R is a non-zero rational number. $S = R^{-1}$.

Description

(1) FIRST2(r_1,r_2,R); if ISIGNL(r_1) = 1, (S ← PFL(BORROW(r_2), PFL(BORROW(r_1),0)); return); S ← PFL(INEG(r_2), PFL(INEG(r_1),0)); return.

Theorem 2.4.9. Let R be a rational number. $t_{RNRECP}(R) \preceq L(R)$.

Proof. Obvious. ∎

Algorithm 2.4.10. T = RNQ(R,S)

(Rational Number Quotient)

R and S are rational numbers such that $S \neq 0$. $T = R \cdot S^{-1}$.

Description

(1) If R = 0, (T ← 0; return); S' ← RNRECP(S); T ← RNPROD(R,S'); erase S'; return.

Theorem 2.4.10. Let R and $S \neq 0$ be rational numbers. If R = 0, $t_{RNQ}(R,S) \sim 1$. If $R = (r_1,r_2) \neq 0$ and $S = (s_1,s_2)$ define $d = L(\gcd(r_1,s_1))$, $e = L(\gcd(r_2,s_2))$, $g = L(R)$, $h = L(S)$ and $k = \min(d,e)$. $t_{RNQ}(R,S) \preceq n(m-k+1)$, where $n = \min(g,h)$ and $m = \max(g,h)$.

Proof. If R = 0 the proof is obvious, so assume $R \neq 0$. $t_{RNRECP}(S) \preceq m \leq k + (m-k+1) \preceq k(m-k+1) \leq n(m-k+1)$ and $t_{RNPROD}(R,S') \preceq n(m-k+1)$ by Theorem 2.4.8. ∎

Algorithm 2.4.11. s = RNSIGN(R)

(Rational Number, Sign)

R is a rational number, s is the sign of R.

Description

(1) If R = 0, (s ← 0; return); s ← ISIGNL(FIRST(R)); return.

Theorem 2.4.11. Let $R = (r_1,r_2)$ be a rational number. $t_{RNSIGN}(R) \preceq L(r_1)$.

Proof. Obvious. ∎

Next we wish to construct an algorithm for comparing two rational numbers, $R = (r_1,r_2)$ and $S = (s_1,s_2)$. Let $r = \text{sign}(R)$, $s = \text{sign}(S)$. If $r = 0$ then $\text{sign}(R-S) = -s$, while $r \neq 0$ and $r \neq s$ implies that $\text{sign}(R-S) = r$. The only remaining case is $r = s \neq 0$, where we make use of $\text{sign}(R-S) = \text{sign}(r_1/r_2 - s_1/s_2) = \text{sign}(r_1s_2 - r_2s_1)$.

Algorithm 2.4.12. c = RNCOMP(R,S)

(Rational Number Compare)

R and S are rational numbers, c = sign(R-S).

Description

(1) r ← RNSIGN(R); s ← RNSIGN(S); if r = 0, (c ← -s; return); if $r \neq s$, (c ← r; return).

(2) FIRST2(r_1,r_2,R); FIRST2(s_1,s_2,S); r ← IPROD(r_1,s_2); s ← IPROD(r_2,s_1); c ← ICOMP(r,s); erase r,s; return.

Theorem 2.4.12. Let $R = (r_1,r_2)$, $S = (s_1,s_2)$. Then $t_{RNCOMP}(R,S) \preceq L(r_1) + L(s_1)$ if $RS \leq 0$ and $t_{RNCOMP}(R,S) \preceq L(r_1)L(s_2) + L(r_2)L(s_1)$ if $RS > 0$.

Proof. $t_1 \sim t_{RNSIGN}(R) + t_{RNSIGN}(S) \preceq L(r_1) + L(s_1)$. Thus if $R \cdot S \leq 0$, $t_{RNCOMP}(R,S) \preceq L(r_1) + L(s_1)$. Suppose $R \cdot S > 0$. $t_{IPROD}(r_1,s_2) \preceq L(r_1)L(s_2)$, $t_{IPROD}(r_2,s_1) \preceq L(r_2)L(s_1)$ and $t_{ICOMP}(r,s) \preceq L(r) + L(s) \preceq L(r_1) + L(s_2) + L(r_2) + L(s_1) \preceq L(r_1)L(s_2) + L(r_2)L(s_1)$. ∎

The Henrici-Brown addition algorithm is used to form the sum of two rational numbers (see [COL71d]). Let a, b, c and d be integers such that gcd(a,b) = gcd(c,d) = 1. Let $e = \gcd(b,d)$, $\bar{b} = b/e$ and $\bar{d} = d/e$. This algorithm uses the fact that $\gcd(ad+bc,bd) = e \gcd(a\bar{d}+\bar{b}c,e)$.

Algorithm 2.4.13. T = RNSUM(R,S)

(Rational Number Sum)

R and S are rational numbers. T = R + S.

Description

(1) [Check for zero inputs.] If R = 0, (T ← BORROW(S); return); If S = 0, (T ← BORROW(R); return); FIRST2(r_1,r_2,R); FIRST2(s_1,s_2,S).

(2) [Determine $\bar{r}_2$, $\bar{s}_2$ and form $r_1\bar{s}_2 + \bar{r}_2 s_1$, $r_2\bar{s}_2$.] IGCDCF(r_2,s_2,a,$\bar{r}_2$,$\bar{s}_2$); u_1 ← IPROD(r_1,$\bar{s}_2$); u_2 ← IPROD($\bar{r}_2$,s_1); t_1 ← ISUM(u_1,u_2); erase u_1, u_2, $\bar{r}_2$; if $t_1 = 0$, (erase a, $\bar{s}_2$; T ← 0; return); t_2 ← IPROD(r_2,$\bar{s}_2$); erase $\bar{s}_2$; if FIRST(a) = 1 and TAIL(a) = 0, (erase a; T ← PFL(t_1,PFL(t_2,0)); return).

(3) [Compute gcd(t_1,a) and finish up.] b ← IGCD(t_1,a); erase a; if FIRST(b) = 1 and TAIL(b) = 0, (erase b; T ← PFL(t_1,PFL(t_2,0)); return); $\bar{t}_1$ ← IQ(t_1,b); $\bar{t}_2$ ← IQ(t_2,b); erase t_1, t_2, b; T ← PFL($\bar{t}_1$,PFL($\bar{t}_2$,0)); return.

Theorem 2.4.13. Let $R = (r_1,r_2)$ and $S = (s_1,s_2)$ be rational numbers. If R = 0 or S = 0, $t_{RNSUM}(R,S) \sim 1$. Let $R \neq 0$, $S \neq 0$, and define $k = L(\gcd(r_2,s_2))$, $g = L(R)$, $h = L(S)$, $m = \max(g,h)$, and $n = \min(g,h)$. $t_{RNSUM}(R,S) \preceq n(m-k+1)$.

Proof. $t_{IGCDCF}(r_2,s_2) \preceq \min(L(r_2), L(s_2))\{\max(L(r_2), L(s_2)) - k+1\} \leq \min(g,h)\{\max(g,h)-k+1\} = n(m-k+1)$. $t_{IPROD}(r_1,\bar{s}_2) + t_{IPROD}(\bar{r}_2,s_1) \preceq L(r_1)\{L(s_2)-k+1\} + L(s_1)\{L(r_2)-k+1\} \leq g(h-k+1) + h(g-k+1) = n(m-k+1) + m(n-k+1) = n(m-k+1) + mn + m(-k+1) \leq n(m-k+1) + mn + n(-k+1) = n(m-k+1) + n(m-k+1) \sim n(m-k+1)$. $t_{ISUM}(u_1,u_2) \preceq L(u_1) + L(u_2) \preceq (m+n-k+1) + (m+n-k+1) \sim (m-k+1) \preceq n(m-k+1)$. $t_{IPROD}(r_2,\bar{s}_2) \preceq L(r_2)(L(s_2)-k+1) \preceq g(h-k+1) \preceq n(m-k+1)$.

In step (3), note that $L(t_i) \leq m-k+1$ for i = 1,2 and $L(b) \leq k$. $t_{IGCD}(t_1,a) \preceq L(t_1)L(a) \preceq k(m-k+1) \leq n(m-k+1)$, $t_{IQ}(t_i,b) \preceq L(t_i)L(b) \leq k(m-k+1) \leq n(m-k+1)$, for i = 1,2. ∎

Algorithm 2.4.14. T = RNDIF(R,S)

(Rational Number Difference)

R and S are rational numbers, T = R - s.

Description

(1) S' ← RNNEG(S); T ← RNSUM(R,S'); erase S'; return.

Theorem 2.4.14. Let $R = (r_1,r_2)$, $S = (s_1,s_2)$ be rational numbers. R = 0 or R ≠ 0 and S = 0 implies that $t_{RNDIF}(R,S) \preceq L(S)$. If R ≠ 0 and S ≠ 0 then define $k = L(\gcd(r_2,s_2))$, $g = L(R)$, $h = L(S)$, $m = \max(g,h)$, and $n = \min(g,h)$. $t_{RNDIF}(R,S) \preceq n(m-k+1)$.

Proof. Assume R ≠ 0 and S ≠ 0. $T_{RNNEG}(S) \preceq m \leq k + (m-k+1) \preceq$

$k(m-k+1) \leq n(m-k+1)$ and $t_{RNSUM}(R,S') \leq n(m-k+1)$ by Theorem 2.4.13 since $L(S') = L(S)$. ∎

Section 2.5. Polynomial Algorithms

Definition 2.5.1. Let $A \in I[x]$ such that A is positive, primitive, square-free, and $\deg(A) > 0$. Define B to be the unique polynomial which is the product of all the positive non-linear irreducible factors of A. $B|A$, B has no real rational roots, and if $C|A$ and C has no real rational roots then $C|B$; hence we call B the greatest rational root-free divisor of A. This is abbreviated $B = \text{grrfd}(A)$.

We shall now discuss an algorithm which finds the grrfd(A), a list of the real rational roots of A, and a list of intervals containing the real irrational roots of A. A is assumed to be an integral, univariate, positive, primitive, square-free polynomial of positive degree. The algorithm makes use of two basic facts. First, if b/c is a rational number then it is a root of A if and only if $(cx-b)|A(x)$. Second, if α is a root of A and $a = \text{ldcf}(A)$ then α is rational if and only if $a\alpha$ is an integer.

The input to the subroutine is a polynomial A with the properties described above. The outputs are $B = \text{grrfd}(A)$; $R = (r_1,\ldots,r_k)$, a list of all real rational roots of A with $r_1 < r_2 < \cdots < r_k$; and $I = (I_1,I_2,\ldots,I_\ell)$, a list of isolating intervals of the real irrational roots of A with $I_1 \leq I_2 \leq \cdots \leq I_\ell$.

The algorithm constructs these outputs by first determining $n = \deg(A)$. If $n = 1$ then $A(x) = cx-b$ so $B = 1$, $I = ()$, $R = (b/c)$ and the algorithm terminates.

If $n > 1$ ANALR (see [HEI70]) is applied to obtain $J = (J_1,\ldots,J_{k+\ell})$, a list of isolating intervals of the real roots of A where

each interval is of length at most $1/a$ and $a = \text{ldcf}(A)$. If $J = ()$ then $B = A$, $R = I = ()$ and the algorithm terminates. Otherwise, for $1 \leq i \leq k + \ell$ the algorithm does the following.

Let α_i be the root of A in J_i. If $J_i = \{s\}$, that is $\alpha_i = s$, a rational number, then s is treated as b/c, which is discussed below. If $J_i = (s,t]$ then $a\alpha_i \in aJ_i$, an interval of length at most one which therefore contains at most one integer. $d = \lfloor as \rfloor$ and $e = \lfloor at \rfloor$ are computed. If $d = e$ there is no integer in aJ_i and so J_i is adjoined to I. If $d \neq e$ then the integer $e \in aJ_i$ so $\text{sign}(A(e/a))$ is determined. If this sign is nonzero the root in J_i is irrational so J_i is adjoined to I. Otherwise let b/c be e/a in canonical form. At this point B is A divided by the already determined factors, so B is replaced by $B(x)/(cx-b)$ and b/c is adjoined to R. If $i < k + \ell$ the loop is repeated for $i+1$. After J is exhausted the algorithm terminates.

Algorithm 2.5.1. PGRRFD(A,B,R,I)

(Polynomial Greatest Rational Root-free Divisor)

The input is A, a positive, primitive, square-free, integral polynomial of positive degree. The outputs are: B, the positive greatest rational root-free divisor of A; $R = (r_1,\ldots,r_k)$, where $r_1,\ldots,r_k$ are the real rational roots of A with $r_1 < r_2 \ldots < r_k$; and $I = (I_1,\ldots,I_\ell)$, where $I_1,\ldots,I_\ell$ are isolating intervals of the real irrational roots of A with $I_1 \leq I_2 \leq \ldots \leq I_\ell$ and the length of I_j is not greater than $1/\text{ldcf}(A)$ for $1 \leq j \leq \ell$.

Description

(1) [Initialize.] $I \leftarrow 0$; $a \leftarrow \text{PLDCF}(A)$; $\text{ONE} \leftarrow \text{PFA}(1,0)$; if $\text{PDEG}(A) > 1$, go to (3).

(2) [Complete computations for the case $\deg(A) = 1$.] $T \leftarrow \text{TAIL}(\text{TAIL}(\text{TAIL}(A)))$; if $T = 0$, (erase a; $r_1 \leftarrow 0$); if $T \neq 0$, ($r_1 \leftarrow \text{PFL}(\text{INEG}(\text{FIRST}(T)), \text{PFL}(a,0))$); $R \leftarrow \text{PFL}(r_1,0)$; $B \leftarrow \text{PFL}(\text{PVBL}(A), \text{PFL}(\text{ONE},\text{PFA}(0,0)))$; return.

(3) [Compute the isolating intervals.] $\ell \leftarrow \text{PFL}(\text{ONE},\text{PFL}(\text{BORROW}(a), 0))$; $J \leftarrow \text{ANALR}(A,\ell)$; erase ℓ.

(4) [Initialize for rational root finding loop.] $R \leftarrow 0$; $B \leftarrow \text{BORROW}(A)$.

(5) [Check for completion and obtain the next interval.] If $J = 0$, (erase a; $R \leftarrow \text{INV}(R)$; $I \leftarrow \text{INV}(I)$; return); $\text{DECAP}(J_1,J)$; $\text{FIRST2}(s,t,J_1)$; if $\text{RNCOMP}(s,t) \neq 0$, go to (6); $\text{DECAP2}(r_1,x,J_1)$; erase x; go to (8).

(6) [Compute $d = \lfloor as \rfloor$ and $e = \lfloor at \rfloor$.] $s' \leftarrow \text{RNISPR}(a,s)$; $d \leftarrow \text{RNFLOR}(s')$; $t' \leftarrow \text{RNISPR}(a,t)$; $e \leftarrow \text{RNFLOR}(t')$; erase s', t'.

(7) [Determine if $A(e/a) = 0$.] If $\text{ICOMP}(d,e) = 0$, (erase d, e; $I \leftarrow \text{PFL}(J_1,I)$; go to (5)); $r_1 \leftarrow \text{RNINT2}(e,a)$; erase d, e; if $\text{REVAL}(A,r_1) \neq 0$, (erase r_1; $I \leftarrow \text{PFL}(J_1,I)$; go to (5)); erase J_1.

(8) [Prefix r_1 to R and divide B by $(cx-b)$.] $R \leftarrow \text{PFL}(r_1,R)$; if $r_1 = 0$, ($\text{ONE} \leftarrow \text{PFA}(1,0)$; $f \leftarrow \text{BORROW}(\text{ONE})$; $C \leftarrow \text{ONE}$); if $r_1 \neq 0$, ($\text{FIRST2}(b,c,r_1)$; $C \leftarrow \text{PFA}(1,\text{PFL}(\text{INEG}(b), \text{PFA}(0,0)))$; $f \leftarrow \text{BORROW}(c)$); $C \leftarrow \text{PFL}(\text{PVBL}(A), \text{PFL}(f,C))$; $T \leftarrow \text{PQ}(B,C)$; erase B,C; $B \leftarrow T$; go to (5).

Theorem 2.5.1. Let A be a positive, primitive, square-free polynomial of positive degree, $m = \deg(A)$ and $g = |A|_1$. If $m = 1$, $t_{PGRRFD}(A) \preceq L(g)$; if $m > 1$, $t_{PGRRFD}(A) \preceq m^{10}L(g)^3$.

Proof. If $m = 1$, $t_{PGRRFD}(A) \sim T_1 + T_2 \preceq L(g)$.

If $m > 1$, $T_1 \sim 1$, step (2) is not executed, $T_4 \sim 1$, and it follows from a result in [HEI70] that $T_3 \preceq m^{10}L(g)^3$.

The number of executions of steps (5) through (8) is $\preceq m$; therefore if we show the time for one execution of each step is $\preceq m^9L(g)^3$ we are done.

To analyze the steps which follow, we need a bound on the size of the endpoints of the isolating intervals of the roots of A. If $s = s_1/s_2$ is such a point, it follows from results in [HEI70] that $|s_1| \leq 32\, g^{m^2+2} \sim g^{m^2+2}$ and $s_2 \leq 8g^{m^2+1} \sim g^{m^2+1}$. Therefore $1 \leq L(s) = L(s_1) + L(s_2) \preceq m^2L(g)$.

We have $t_5 \prec t_{INV}(R) + t_{INV}(I) + t_{RNCOMP}(s,t) \preceq m + L(s)L(t) \preceq m + m^4L(g)^2 \prec m^9L(g)^3$.

In step (6), $t_{RNISPR}(a,s) \preceq L(a)L(s) \preceq L(g)\, m^2L(g) = m^2L(g)^2$, and similarly $t_{RNISPR}(a,t) \preceq m^2L(g)^2$. $t_{RNFLOR}(s') \preceq L(s')L(\lfloor s' \rfloor) \preceq m^2L(g)\, m^2L(g) = m^4L(g)^2$, and similarly $t_{RNFLOR}(t') \preceq m^4L(g)^2$. Thus $t_6 \prec m^9L(g)^3$.

In step (7), if α is a zero of A, then $|\alpha| \leq g-1$. Since $A(\alpha) = 0$ and $|\alpha - e/a| \leq 1$, we have $|e/a| \leq g$ and $|e| \leq ag \leq g^2+1$. Also, $d = e$ or $d = e-1$, so $|d| \leq g^2$. Hence $L(d), L(e) \preceq L(g)$ and $L(r_1) \preceq L(g)$. So $t_{ICOMP}(d,e) \preceq L(g)$, $t_{RNINT2}(e,a) \preceq L(g)^2$ and $t_{REVAL}(A,r_1) \preceq m^2L(r_1)^2 + m^2L(g)L(r_1) \preceq m^2L(g)^2$.

In each execution of step (8), both B and $T = PQ(B,C)$ are divisors of A. It follows from Corollary 1.4.1 that, $|B|_1$, $|T|_1 \leq (m+1)^{2m}g$. Also, C is a linear divisor of A, so $|C|_1 \leq g$. Hence $t_{PQ}(B,C) \preceq (\deg(C)+1)(\deg(T)+1)L(|C|_\infty)L(|T|_1) \preceq mL(g)\{mL(m) + L(g)\} \sim m^2L(m)L(g) + mL(g)^2 \prec m^9L(g)^3$. $t_{INEG}(b) \preceq L(g) \prec m^9L(g)^3$. ∎

The next algorithm computes the integer-content, b, and integer-primitive part, C, of A, a non-zero, integral polynomial in v variables, $v \geq 1$. That is, b is the greatest common divisor of the integer coefficients of A, and $C = A/b$. If $b = 1$ then we say that A is integer-primitive.

Algorithm 2.5.2. PICPP(A,b,C)

(Polynomial Integer-Content and Integer-Primitive Part)

The input is A, a non-zero, integral, multivariate polynomial. The outputs are b, the integer-content of A, and C, the integer-primitive part of A.

Description

(1) $b \leftarrow$ PICONT(A); if PONE(b) = 1, (C ← BORROW(A); return); C ← PID(A,b); return.

Theorem 2.5.2. Let A be an element of $I[x_1,\ldots,x_v]$, $v \geq 1$. Let $|A|_\infty = d$, $\deg_i(A) = n_i$ for $1 \leq i \leq v$. Let ℓ be the number of non-zero coefficients of A considered as a polynomial in x_v. Then $t_{PICPP}(A) \preceq L(d)^2\, \ell \prod_{i=1}^{v-1}(n_i+1)$.

Proof. It is clear that $t_{PICONT}(A) \leq L(d)^2 \ell \prod_{i=1}^{v-1} (n_i+1)$, and that $t_{PID}(A,b) \leq L(d)L(b) \ell \prod_{i=1}^{v-1} (n_i+1) \leq L(d)^2 \ell \prod_{i=1}^{v-1} (n_i+1)$. $t_{PONE}(b) \leq 1$. ∎

Suppose $b, a_1, \ldots, a_k$ are non-zero integers such that $|b| \geq 2$, $k \geq 1$, and one wants to compute $a_1 b^t, \ldots, a_k b^t$, where t is an integer such that $t \geq 2$. One method is to form the sequence $a_j b, a_j b^2, \ldots, a_j b^t$ for each j. The time to form $a_j b^i$, $i \geq 1$, is $\sim L(a_j b^{i-1}) L(b) \sim L(b)\{L(a_j) + (i-1)L(b)\}$. Hence the time to compute $a_j b^t$ is $\sim \sum_{i=1}^{t} L(b)\{L(a_j)+(i-1)L(b)\} \sim tL(b)L(a_j) + t^2 L(b)^2$. The time to compute $a_1 b^t, \ldots, a_k b^t$, therefore, is $\sim kt^2 L(b)^2 + tL(b) \sum_{j=1}^{k} L(a_j)$.

On the other hand, one might compute b^t first, then multiply $a_1, \ldots, a_k$ by b^t. It is clear that $t_{PPOWER}(b,t) \sim t^2 L(b)^2$, while the time to multiply a_j by b^t is $\sim tL(b)L(a_j)$. Hence the total time is $\sim t^2 L(b)^2 + tL(b) \sum_{j=1}^{k} L(a_j)$. This bound is superior for $k > 1$ and the same for $k = 1$. One can also show that both techniques have total time which is $\sim L(b) \sum_{j=1}^{k} L(a_j)$, if $t = 1$, $k \geq 1$, $b \neq 0$.

Let A, B be multivariate integral polynomials such that $n = \deg(A) \geq k = \deg(B) > 0$. Let $\ell = n - k + 1$, and let m be an integer such that $m \geq \ell$. The algorithm which follows, PSREMG, computes $b^{m-\ell}\, \mathrm{prem}(A,B)$, where $b = \mathrm{ldcf}(B)$. This is accomplished by first forming the sequence $C_0 = A, C_1, \ldots, C_{m_1}$, where

$$C_{i+1}(x) = bC_i(x) - c_i B(x) x^{n_i - k}, \tag{1}$$

$n_i = \deg(C_i)$, $c_i = \mathrm{ldcf}(C_i)$, and m_1 is the first integer such that $\deg(C_{m_1}) < k$. If $t = m - m_1 > 0$, $C_{m_1}(x) \neq 0$, then the next and final computation is to form $b^t C_{m_1}$; this is just a matter of determining $b^t a_{m_1}, \ldots, b^t a_0$, where $C_{m_1}(x) = \sum_{i=0}^{m_1} a_i x^i$. Assume A,B are univariate, the case with which we are primarily concerned. If $|b| \geq 2$ and $t \geq 2$, or $t = 1$, then we have shown that the second method described above is always as good as or superior to the first method for this computation. This second method is the one which is implemented in PSREMG.

If $m = \ell$, then the output of PSREMG is just prem(A,B), a polynomial which is also computed by the algorithm PSREM (see [(COL71a)]). This latter algorithm computes $C_0, \ldots, C_{m_1}$ just as does PSREMG, but employs the first of the methods described above to compute $b^t C_{m_1}$. Consequently, it follows from the above discussion that unless $t \geq 2$ and $b = \pm 1$, PSREMG(A,B,ℓ) is always as good as or superior to PSREM(A,B).

Algorithm 2.5.3. C = PSREMG(A,B,m)

(Polynomial Pseudo-remainder, Generalized)

A, B are non-zero elements of $I[x_1, \ldots, x_v]$, $v \geq 1$, such that $\deg(A) \geq \deg(B)$. Let $\ell = \deg(A) - \deg(B) + 1$, $b = \mathrm{ldcf}(B)$. m is a FORTRAN integer such that $m \geq \ell$. C is the unique polynomial such that $C = b^{m-\ell}\, \mathrm{prem}(A,B)$.

Description

(1) [Initialize.] n ← PDEG(A); k ← PDEG(B); m_1 ← 0; b ← PLDCF(B);

B' ← PRED(B); C ← BORROW(A).

(2) [Perform subtraction loop.] If C = 0, (erase b, B'; return); t ← PDEG(C) − k; if t < 0, go to (3); $m_1 \leftarrow m_1+1$; c ← PLDCF(C); C' ← PRED(C); U ← PSPROD(C',b,0); V ← PSPROD(B',c,t); erase C, C', c; C ← PDIF(U,V); erase U, V; go to (2).

(3) [Multiply C by b^{m-m_1}.] $t \leftarrow m - m_1$; if t = 0, (erase b, B'; return); d ← PPOWER(b,t); U ← PSPROD(C,d,0); erase b, B', d, C; C ← U; return.

In our applications we will be using PSREMG with univariate inputs only. Therefore we have limited our computing time analysis to this case.

Theorem 2.5.3. Let A, B be non-zero elements of I[x] such that $n = \deg(A) \geq k = \deg(B) > 0$. Let m be an integer such that $m \geq n - k + 1$, $f = |A|_\infty$, $g = |B|_\infty$. Then $t_{PSREMG}(A,B,m) \preceq nmL(g)\{mL(g)+L(f)\}$.

Proof. In step (2) of PSREMG, the sequence of polynomials $C_0 = A, \ldots, C_{m_1}$ defined by (1) is computed.

Let $f_i = |C_i|_\infty$. Observe that $f_i \leq 2g\, f_{i-1}$, and hence $f_i \leq 2^i g^i f$. This implies that $L(f_i) \leq L(2^i g^i f) \sim i\{L(2)+L(g)\} + L(f) \sim iL(g) + L(f)$, for $0 \leq i \leq m_1$.

We now observe that for the i^{th} execution of step (2), $t_{PSPROD}(C',B,0) \preceq (\deg(C')+1)L(|C_{i-1}|_\infty)L(b) \preceq (n-i+1)L(f_{i-1})L(b) \preceq (n-i+1)L(g)L(f_{i-1})$, and similarly $t_{PSPROD}(B',c,t) \preceq kL(g)L(f_{i-1})$. $t_{PDIF}(U,V) \preceq \{\max(\deg(U), \deg(V))+1\}\{L(|U|_\infty)+L(|V|_\infty)\} \preceq (n-i+1)\{L(f_{i-1}b)+L(f_{i-1}g)\} \preceq (n-i+1)\{L(f_{i-1})+L(g)\}$. Since $k \leq n - i + 1$, the time to compute C_i is $\preceq (n-i+1)L(g)L(f_{i-1})$. This implies that $T_2 \preceq \sum_{i=1}^{m_1} (n-i+1)L(g)L(f_{i-1}) \preceq nL(g) \sum_{i=1}^{m_1} L(f_{i-1}) \preceq nL(g) \sum_{i=1}^{m_1} \{(i-1)L(g)+L(f)\} \preceq nL(g)\{m_1^2L(g)+m_1L(f)\} \preceq nm\, L(g)\{mL(g)+L(f)\}$.

From [COHE70] we have that $t_{PPOWER}(b,t) \preceq t^2L(b)^2 \preceq (m-m_1)^2L(g)^2 \preceq m^2L(g)^2$. $t_{PSPROD}(C,d,0) \preceq \{\deg(C)+1\}L(|C|_\infty)L(d) \preceq kL(f_{m_1})(m-m_1)L(b) \preceq k\{m_1L(g)+L(f)\}(m-m_1)L(g) \preceq kmL(g)\{mL(g)+L(f)\} \preceq nm\, L(g)\{mL(g)+L(f)\}$. ∎

Lemma 2.5.1. Let A(x), B(x), C(x) be as defined in Algorithm 2.5.3. Then

(2) $$|C|_\infty \leq 2^m |B|_\infty^m |A|_\infty, \quad \text{and}$$

(3) $$|C|_\infty \leq |B|_1^m |A|_1 .$$

Proof. (2) is immediate from the proof of Theorem 2.5.3.

In the notation of Definition 1.4.12, $\text{prem}(A,B) = S_{k-1}(A,B) = \sum_{i=0}^{k-1} \det(M_{k-1,i})\, x^i$. $M_{k-1,i}$ is an $n - k + 2$ square matrix consisting of 1 row of A coefficients and $n - k + 1$ rows of B coefficients. Then by Theorem 1 of [COHO73], $|\text{prem}(A,B)|_\infty \leq |A|_1 |B|_1^{n-k+1}$. Hence $|C|_\infty \leq b^{m-(n-k+1)} |\text{prem}(A,B)|_\infty \leq |B|_1^m |A|_1$. ∎

CHAPTER 3: OPERATIONS IN $Q[x_1,\ldots,x_v]/(\Psi(x_1))$

Section 3.1. Introduction

Let $\Psi(x_1)$ be an integral polynomial of positive degree. Assume that $\mathrm{ldcf}(\Psi) > 0$ and that Ψ is primitive. Consider the residue class ring $Q[x_1]/(\Psi(x_1))$ and the polynomial ring $\{Q[x_1]/(\Psi(x_1))\}[x_2,\ldots,x_v]$, $v \geq 2$, where Q is the field of rational numbers. Making use of an obvious isomorphism, these rings may also be represented by the more common notation $Q[x_1,\ldots,x_v]/(\Psi(x_1))$, $v \geq 1$. The motivation for looking at these rings is their use as a basis for performing operations in $Q(\alpha)$ and $Q(\alpha)[x]$, where $\Psi(\alpha) = 0$.

Actually, we will only use the ring $Q[x_1]/(\Psi(x_1))$ - for performing operations in $Q(\alpha)$ - and the ring $Q[x_1,x_2]/(\Psi(x_1))$ - for performing operations in $Q(\alpha)[x_2]$. However, by approaching the problem in terms of v variables, we both simplify the presentation of the algorithms and generalize their applications.

In fact, with only minor modifications to algorithms in succeeding chapters, one would have a system, based on the algorithms in this chapter, for performing operations on multivariate polynomials. However, the multivariate case is not germane to the primary motivation of this thesis, namely the quantifier elimination algorithm of G. E. Collins, while such a generalization would make the computing time analysis of certain algorithms in succeeding chapters extremely complex; consequently, we have limited ourselves to $Q(\alpha)$ and $Q(\alpha)[x]$. We will then, need only analyze the time of the algorithms in the present chapter for the case $v = 2$. The time for the case $v = 1$ can then be obtained by substituting zero for the degree in x_2 of the relevant arguments.

It is well known that elements of $Q[x_1]/(\Psi(x_1))$ may be considered as univariate polynomials over Q of degree less than $\deg(\Psi)$. Similarly, elements of $Q[x_1,\ldots,x_v]/(\Psi(x_1))$ may be considered as multivariate polynomials over Q of degree in x_1 less than $\deg(\Psi)$. Suppose A is such a polynomial. If A is the zero polynomial, it is represented by the null list. If A is non-zero it is uniquely represented by the list $(r,\bar{A})$, where r is a positive rational number in canonical form and $\bar{A}$ is an integral, integer-primitive polynomial such that $A(x_1,\ldots,x_v) = r\bar{A}(x_1,\ldots,x_v)$. We write $A = (r,\bar{A})$ and refer to r as the rational part of A, $\mathrm{rp}(A)$, and $\bar{A}$ as the polynomial part of A, $\mathrm{polp}(A)$. The usual SAC-1 representation for polynomials is assumed for $\bar{A}$.

The algorithms given assume the availability of Ψ, and thus it need not be passed as a parameter. In the SAC-1 implementation of this system, this is accomplished by putting Ψ in the labeled common block, TR7, as the variable PSI.

To aid in analyzing the computing time of algorithms dealing with elements of $Q[x_1,\ldots,x_v]/(\Psi(x_1))$ we define the function ν below. ν is neither a norm nor a semi-norm as defined in [COHO73], but it is useful in determining a bound on the coefficients of elements of $Q[x_1,\ldots,x_v]/(\Psi(x_1))$. In this definition, and in some of the material in the following chapters, we tacitly assume that the zero polynomial is represented by the list (1,0) in order to simplify the discussion.

Definition 3.1.1. Let $A = (r,\bar{A})$ be an element of $Q[x_1,\ldots,x_v]/(\Psi(x_1))$, where $r = (r_1,r_2)$. Define ν_1, ν_2, ν to be functions from $Q[x_1,\ldots,x_v]/(\Psi(x_1))$ to the non-negative integers such that $\nu_1(A) = |r_1\bar{A}|_1$, $\nu_2(A) = r_2$, and $\nu(A) = \nu_1(A) + \nu_2(A)$.

Section 3.2. Additive Operations

The first algorithm we will discuss in this section will be one for the addition of two elements of $Q[x_1,\ldots,x_v]/(\Psi(x_1))$. The algorithm is based on the Henrici-Brown algorithm for the addition of rational functions (see [COL71d] or [KNU69]). Suppose $A = (r,\bar{A})$, $B = (s,\bar{B})$ are elements of $Q[x_1,\ldots,x_v]/(\Psi(x_1))$ and $r = (r_1,r_2)$, $s = (s_1,s_2)$. The algorithm makes use of the fact that if $g = \gcd(r_2,s_2)$, $\bar{r}_2 = r_2/g$ and $\bar{s}_2 = s_2/g$, then $\gcd(s_2r_1\bar{A} + r_2s_1\bar{B}, r_2s_2) = g\,\gcd(\bar{s}_2r_1\bar{A} + \bar{r}_2s_1\bar{B}, g)$.

Algorithm 3.2.1. C = PMSUM(A,B)

(Polynomial Modulus, Sum)

A, B are elements of $Q[x_1,\ldots,x_v]/(\Psi(x_1))$, $v \geq 1$. $C = A + B$.

Description

(1) [Initialize.] If $A = 0$, (C ← BORROW(B); return); if $B = 0$, (C ← BORROW(A); return); FIRST2$(r,\bar{A},A)$; FIRST2$(s,\bar{B},B)$; FIRST2(r_1,r_2,r); FIRST2(s_1,s_2,s).

(2) [Compute $C_1 = \bar{s}_2r_1\bar{A} + \bar{r}_2s_1\bar{B}$.] IGCDCF$(r_2,s_2,g,\bar{r}_2,\bar{s}_2)$; u ← IPROD$(\bar{s}_2,r_1)$; v ← IPROD$(\bar{r}_2,s_1)$; U ← PIP$(\bar{A},u)$; V ← PIP$(\bar{B},v)$; C_1 ← PSUM(U,V); erase $\bar{r}_2$, u, v, U, V.

(3) [Compute t_1, t_2, $\bar{C}$.] If $C_1 = 0$, (erase g, $\bar{s}_2$; C ← 0; return); t_2 ← IPROD$(r_2,\bar{s}_2)$; PICPP$(C_1,t_1,\bar{C})$; erase C_1, $\bar{s}_2$; if PONE(g) = 0, go to (4); t ← PFL$(t_1$,PFL$(t_2,0))$; C ← PFL$(t$,PFL$(\bar{C},0))$; erase g; return.

(4) [Compute $h = \gcd(t_1,t_2) = \gcd(t_1,g)$, $t_1/h, t_2/h$.] IGCDCF(t_1,g, h,$\bar{t}_1$,$\bar{g}$); erase t_1, g, $\bar{g}$; if PONE(h) = 1, ($\bar{t}_2 \leftarrow t_2$; go to (5)); $\bar{t}_2 \leftarrow$ IQ(t_2,h); erase t_2.

(5) [Finish-up.] Erase h; t $\leftarrow$ PFL($\bar{t}_1$,PFL($\bar{t}_2$,0)); C $\leftarrow$ PFL(t,PFL($\bar{C}$,0)); return.

Theorem 3.2.1. Let A, B be elements of $Q[x_1,x_2]/(\Psi(x_1))$. If A = 0 or B = 0 then $t_{PMSUM}(A,B) \sim 1$. Suppose that $A = (r,\bar{A})$, $B = (s,\bar{B})$ are non-zero. Let $\partial(A) = (m_1,m_2)$, $\partial(B) = (n_1,n_2)$, d = $\nu(A)$, e = $\nu(B)$. Let h be the number of non-zero coefficients of A considered as a polynomial in x_2, and define k similarly for B; let $\ell = \max(h,k)$. Then $t_{PMSUM}(A,B) \preceq \ell(m_1+n_1+1)\{L(d)^2+L(e)^2\}$.

Proof. Suppose $A \neq 0$, $B \neq 0$. In step (2), $t_{IGCDCF}(r_2,s_2) \preceq L(r_2)L(s_2) \preceq L(d)L(e)$. $t_{IPROD}(\bar{s}_2,r_1)$, $t_{IPROD}(\bar{r}_2,s_1) \preceq L(d)L(e)$. Also, $t_{PIP}(\bar{A},u) \preceq h(m_1+1)L(|\bar{A}|_1)L(u) \preceq \ell(m_1+1)L(d)L(de) \preceq \ell(m_1+1)\{L(d)^2+L(e)^2\}$. Similarly $t_{PIP}(\bar{B},v) \preceq \ell(n_1+1)\{L(d)^2+L(e)^2\}$. $U = \bar{s}_2 r_1 \bar{A}$, so $|U|_1 = |\bar{s}_2|\,|r_1\bar{A}|_1 \leq ed$, and similarly $|V|_1 \leq de$; hence $t_{PSUM}(U,V) \preceq \{h(m_1+1) + k(n_1+1)\}\{L(d)+L(e)\} \preceq \ell(m_1+n_1+1)\{L(d)+L(e)\}$.

In step (3), $t_{IPROD}(r_2,\bar{s}_2) \preceq L(d)L(e)$. $t_{PICPP}(C_1) \preceq \{h(m_1+1) + k(n_1+1)\}L(|C|_\infty)^2 \preceq (h+k)(m_1+n_1+1)L(|U|_\infty+|V|_\infty)^2 \preceq \ell(m_1+n_1+1)\{L(d)^2+L(e)^2\}$.

In step (4), since $|t_1| \leq |C_1|_1 \leq |U|_1 + |V|_1 \leq 2\,de$ and $g \leq r_2 \leq d \leq de$, $t_{IGCDCF}(t_1,g) \preceq L(t_1)L(g) \preceq L(2de)L(de) \sim L(d)^2 + L(e)^2$. Also, $t_2 = r_2\bar{s}_2 \leq de$ and $h \leq g \leq r_2 \leq de$, so $t_{IQ}(t_2,h) \preceq L(t_2)L(h) \preceq L(de)L(de) \preceq L(d)^2 + L(e)^2$. ∎

Lemma 3.2.1. Let A, B be elements of $Q[x_1,\ldots,x_v]/(\Psi(x_1))$, $v \geq 1$. Then $\nu(A\pm B) \leq \nu(A)\nu(B)$.

Proof. Let $A = (r,\bar{A})$, $B = (s,\bar{B})$, $r = (r_1,r_2)$, and $s = (s_1,s_2)$. Then $\nu(A\pm B) \leq |s_2 r_1\bar{A} \pm r_2 s_1\bar{B}|_1 + r_2 s_2 \leq s_2|r_1\bar{A}|_1 + r_2|s_1\bar{B}|_1 + r_2 s_2 \leq \nu(A)\nu(B)$. ∎

Algorithm 3.2.2. B = PMNEG(A)

(Polynomial Modulus, Negation)

A is an element of $Q[x_1,\ldots,x_v]/(\Psi(x_1))$, $v \geq 1$. B = -A.

Description

(1) If A = 0, (B $\leftarrow$ 0; return); FIRST2(r,$\bar{A}$,A); $\bar{B} \leftarrow$ PNEG($\bar{A}$); B $\leftarrow$ PFL(BORROW(r), PFL($\bar{B}$,0)); return.

Theorem 3.2.2. Let A be an element of $Q[x_1,x_2]/(\Psi(x_1))$. If A = 0 then $t_{PMNEG}(A) \sim 1$. Suppose $A \neq 0$, $\bar{A}$ = polp(A), $\partial_1(\bar{A}) = n_1$, $d = |\bar{A}|_\infty$. Let h be the number of non-zero coefficients of A considered as a polynomial in x_2. Then $t_{PMNEG}(A) \preceq h(n_1+1)L(d)$.

Proof. Assume $A \neq 0$. Then $t_{PMNEG}(A) \sim t_{PNEG}(\bar{A}) \preceq h(n_1+1)L(d)$. ∎

Algorithm 3.2.3. C = PMDIF(A,B)

(Polynomial Modulus, Difference)

A, B are elements of $Q[x_1,\ldots,x_v]/(\Psi(x_1))$, $v \geq 1$. C = A - B.

Description

(1) B' $\leftarrow$ PMNEG(B); C $\leftarrow$ PMSUM(A,B'); erase B'; return.

Theorem 3.2.3. Let A, B be elements of $Q[x_1,x_2]/(\Psi(x_1))$. If $B = 0$ then $t_{PMDIF}(A,B) \sim 1$. Suppose $B \neq 0$, $e = \nu(B)$, $\partial(B) = (n_1,n_2)$. Let k be the number of non-zero coefficients of B considered as a polynomial in x_2. If $A = 0$ then $t_{PMDIF}(A,B) \preceq k(n_1+1)L(e)$. Suppose $A \neq 0$, $d = \nu(A)$, $\partial(A) = (m_1,m_2)$. Let h be the number of non-zero coefficients of A considered as a polynomial in x_2. Let $\ell = \max(h,k)$. Then $t_{PMDIF}(A,B) \preceq \ell(m_1+n_1+1)\{L(d)^2 + L(e)^2\}$.

Proof. Assume $B \neq 0$. Then the result follows from Theorem 3.2.1 and Theorem 3.2.2, since $\nu(B) = \nu(B')$, $\partial(B) = \partial(B')$, and B' has the same number of non-zero coefficients as does B. ∎

Section 3.3. Multiplicative Operations

Let μ be the homomorphism from $Q[x_1]$ onto $Q[x_1]/(\Psi(x_1))$ defined by $\mu(A) = A$ modulo Ψ. For $v \geq 1$, let $A(x_1,\ldots,x_v) = \sum_{i=0}^{n} A_i(x_1,\ldots,x_{v-1})x_v^i$ be an element of $Q[x_1,\ldots,x_v]$. μ induces a homomorphism, μ^*, from $Q[x_1,\ldots,x_v]$ onto $Q[x_1,\ldots,x_v]/(\Psi(x_1))$ defined recursively as follows: $\mu^*(A) = \mu(A)$ if $v = 1$, and $\mu^*(A) = \sum_{i=0}^{n} \mu^*(A(x_1,\ldots,x_{v-1}))x_v^i$ if $v > 1$. We denote both μ^* and μ by μ. To determine a multiplication algorithm we first need an algorithm which is an implementation of μ. To this end we state the theorem which follows.

Theorem 3.3.1. Let A be a non-zero element of $I[x_1,\ldots,x_v]$, $v \geq 1$, and let r be a positive rational number. Assume $n = \partial_1(A) \geq k = \deg(\Psi)$, $g = \mathrm{ldcf}(\Psi)$, $m = n - k + 1$, $d = g^m$. If A is $\sum_{i=1}^{\ell} A_i(x_1)\, x_2^{e_{2,i}} \cdots x_v^{e_{v,i}}$, then let $m_i = \max(\deg(A_i) - k + 1, 0)$, and let $B(x_1,\ldots,x_v) = \sum_{i=1}^{\ell} g^{m-m_i}\, \mathrm{prem}(A_i(x_1),\Psi(x_1))x_2^{e_{2,i}} \cdots x_v^{e_{v,i}}$. Then $m \geq m_i$ for $1 \leq i \leq \ell$, and B is an integral polynomial. Also, if $B = 0$ then $\mu(rA) = 0$, while if $B \neq 0$ then $\mu(rA) = (s,\bar{B})$, where c is the integer-content of B, $\bar{B}$ is the integer-primitive part of B, and $s = rc/d$.

Proof. $\mu(A(x_1,\ldots,x_v)) = \sum_{i=1}^{\ell} \mu(A_i(x_1))x_2^{e_{2,i}} \cdots x_v^{e_{v,i}} = \sum_{i=1}^{\ell} (1/g^{m_i})\, \mathrm{prem}(A_i(x_1),\Psi(x_1))x_2^{e_{2,i}} \cdots x_v^{e_{v,i}}$. Now $n \geq \deg(A_i)$, so $m = n - k + 1 \geq \max(\deg(A_i) - k + 1,0) = m_i$. Hence g^{m-m_i} is integral. Since $\mathrm{prem}(A_i(x_1),\Psi(x_1))$ is an integral polynomial, this implies that B is an integral polynomial. $\mu(rA(x_1,\ldots,x_v)) =$

$r(1/g^m) \sum_{i=1}^{\ell} g^{m-m_i} \mathrm{prem}(A_i(x_1),\Psi(x_1))x_2^{e_{2,i}} \cdots x_v^{e_{v,i}} = (r/d)B(x_1, \ldots,x_v)$. The theorem now follows from the canonical form for $Q[x_1, \ldots,x_v]/(\Psi(x_1))$. ∎

The following algorithm will be used to compute the polynomial B of Theorem 3.3.1.

Algorithm 3.3.1. B = PMIPOL(A,m,d)

(Polynomial Modulus, Integral Polynomial Result)

A is a non-zero element of $I[x_1,\ldots,x_v]$, $v \geq 1$. m is a FORTRAN integer such that $m \geq \max(\partial_1(A) - \deg(\Psi) + 1,0)$. $d = g^m$ where $g = \mathrm{ldcf}(\Psi)$. If $A(x_1,\ldots,x_v) = \sum_{i=1}^{\ell} A_i(x_1)x_2^{e_{2,i}} \cdots x_v^{e_{v,i}}$, $m_i = \max(\deg(A_i) - \deg(\Psi) + 1,0)$, then B is the polynomial $\sum_{i=1}^{\ell} g^{m-m_i} \mathrm{prem}(A_i(x_1),\Psi(x_1))x_2^{e_{2,i}} \cdots x_v^{e_{v,i}}$.

Description

(1) [Determine if A is univariate.] If m = 0, (B ← BORROW(A); return); if TYPE (SECOND(A)) = 0, go to (2); T ← TAIL(A); B ← 0; go to (3).

(2) [Perform computation for univariate case.] If PDEG(A) < PDEG(Ψ), (B ← PIP(A,d); return); B ← PSREMG(A,Ψ,m); return.

(3) [Make recursive call with the coefficients of A.] If T = 0, go to (4); ADV2(a,n,T); b ← PMIPOL(a,m,d); if b ≠ 0, B ← PFA (n,PFL(b,B)); go to (3).

(4) [Finish-up.] B ← INV(B); if B ≠ 0, B ← PFL(PVBL(A),B); return.

Theorem 3.3.2. Let A be a non-zero element of $I[x_1,x_2]$, m be an integer such that $m \geq \max(\partial_1(A) - \deg(\Psi) + 1,0)$, $d = (\mathrm{ldcf}(\Psi))^m$. If m = 0 then $t_{\mathrm{PMIPOL}}(A,m,d) \sim 1$. Assume m > 0 and let $\partial_1(A) = n_1$, $f = |A|_\infty$, $g = |\Psi|_\infty$. If ℓ is the number of non-zero coefficients of A considered as a polynomial in x_2, then $t_{\mathrm{PMIPOL}}(A,m,d) \preceq \ell(n_1+1)mL(g)\{mL(g)+L(f)\}$.

Proof. Assume m > 0. Let $A(x_1,x_2) = \sum_{i=1}^{\ell} A_i(x_1)x_2^{e_i}$. Then $t_{\mathrm{PMIPOL}}(A,m,d) \preceq \sum_{i=1}^{\ell} t_{\mathrm{PMIPOL}}(A_i,m,d)$. Hence, by Theorem 2.5.3, $t_{\mathrm{PMIPOL}}(A,m,d) \preceq \sum_{i=1}^{\ell}[(\deg(A_i)+1)L(|A_i|_\infty)L(d) + \deg(A_i)mL(g)\{mL(g) + L(|A_i|_\infty)\}\} \preceq \ell\{(n_1+1)L(f)L(d) + n_1mL(g)\{mL(g) + L(f)\}\} \preceq \ell\{(n_1+1)L(f)mL(g) + (n_1+1)mL(g)\{mL(g) + L(f)\}\} \preceq \ell(n_1+1)mL(g)\{mL(g) + L(f)\}$. ∎

PMRPOL is our desired implementation of μ.

Algorithm 3.3.2. B = PMRPOL(r,A)

(Polynomial Modulus, Rational Polynomial Result)

r is a non-negative rational number and A is an integer-primitive element of $I[x_1,\ldots,x_v]$, $v \geq 1$. $B = \mu(rA)$.

Description

(1) [Initialize and compare degrees.] B ← 0; if r = 0 or A = 0, return; L ← INV(PDEGS(A)); DECAP(n,L); erase L; k ← PDEG(Ψ); if n ≥ k, go to (2); B ← PFL(BORROW(r), PFL(BORROW(A),0)); return.

(2) [Compute m and d.] m ← n − k + 1; e ← PLDCF(Ψ); d ← IPOWER(e,m); erase e.

(3) [Compute $\bar{B} = polp(\mu(rA))$.] $C \leftarrow PMIPOL(A,m,d)$; If $C = 0$, (erase d; return); $PICPP(C,c,\bar{B})$; erase C.

(4) [Compute $s = rp(\mu(rA))$.] $t \leftarrow RNINT2(c,d)$; erase c,d; $s \leftarrow RNPROD(t,r)$; erase t; $B \leftarrow PFL(s,PFL(\bar{B},0))$; return.

Theorem 3.3.3. Let r be a non-negative rational number and let A be an integer-primitive element of $I[x_1,x_2]$. If $r = 0$ or $A = 0$ then $t_{PMRPOL}(r,A) \sim 1$. Suppose $r = (r_1,r_2) \neq 0$, $A \neq 0$, and define $f = |r_1 A|_1 + r_2$, $g = |\Psi|_\infty$, $k = \deg(\Psi)$, $\partial(A) = (n_1,n_2)$. If $n_1 < k$ then $t_{PMRPOL}(r,A) \sim 1$. Let ℓ be the number of non-zero coefficients of A considered as a polynomial in x_2. If $n_1 \geq k$ then $t_{PMRPOL}(r,A) \preceq \ell n_1(n_1-k+1)\{(n_1-k+1)L(g)^2 + L(f)^2\}$.

Proof. Suppose $r \neq 0$, $A \neq 0$, $n_1 \geq k$. In step (2), from [COL73d], we have that $t_{IPOWER}(e,m) \sim m^2 L(e)^2 \preceq (n_1-k+1)^2 L(g)^2$. In step (3), by Theorem 3.3.2, $t_{PMIPOL}(A,m,d) \preceq \ell(n_1+1)\, mL(g)\{mL(g) + L(f)\} \sim \ell n_1(n_1-k+1)L(g)\{(n_1-k+1)L(g) + L(f)\} \preceq \ell n_1(n_1-k+1)\{(n_1-k+1)L(g)^2 + L(f)^2\}$. By Theorem 2.5.2, $t_{PICPP}(C) \preceq \ell(\partial_1(C)+1)L(|C|_\infty)^2 \preceq \ell k\, L(|C|_\infty)^2$. Hence, by Lemma 2.5.1, $t_{PICPP}(C) \preceq \ell k\, L(2^{n_1-k+1}|\Psi|_\infty^{n_1-k+1}|A|_\infty)^2 \preceq \ell k\{(n_1-k+1)^2 L(g)^2 + L(f)^2\} \leq \ell n_1(n_1-k+1)\{(n_1-k+1)L(g)^2 + L(f)^2\}$.

In step (4), $t_{RNINT2}(c,d) \preceq L(c)L(d)$ by Theorem 2.4.5, so $t_{RNINT2}(c,d) \preceq L(|C|_\infty)L(g^{n_1-k+1}) \preceq (n_1-k+1)L(g)\{(n_1-k+1)L(g) + L(f)\} \preceq (n_1-k+1)\{(n_1-k+1)L(g)^2 + L(f)^2\}$. $t_{RNPROD}(t,r) \preceq L(t)L(r) \preceq \{(n_1-k+1)L(g) + L(f)\}L(f) \preceq (n_1-k+1)\{L(g) + L(f)\}L(f) \preceq (n_1-k+1)\{L(g)^2 + L(f)^2\}$ by Theorem 2.4.8. ∎

We are now ready to present the multiplication algorithm PMPROD. The inputs are A, B and the output is $C = A \cdot B$. If $A = 0$ or $B = 0$, C is set to zero and the algorithm terminates. If $A = (r,\bar{A})$ and $B = (s,\bar{B})$ are non-zero, then $D = \bar{A}\bar{B}$ and $u = rs$ are computed. Since $\bar{A}$, $\bar{B}$ are integer-primitive, D is also. Hence, the algorithm then computes $C = \mu(uD) = PMRPOL(u,D)$.

Algorithm 3.3.3. $C = PMPROD(A,B)$

(Polynomial Modulus, Product)

A, B are elements of $Q[x_1,\ldots,x_v]/(\Psi(x_1))$, $v \geq 1$. $C = A \cdot B$.

Description

(1) [Determine if there is a zero input.] If $A = 0$ or $B = 0$, ($C \leftarrow 0$; return); $FIRST2(r,\bar{A},A)$; $FIRST2(s,\bar{B},B)$.

(2) [Form u, D, and $C = \mu(uD)$.] $u \leftarrow RNPROD(r,s)$; $D \leftarrow PPROD(\bar{A},\bar{B})$; $C \leftarrow PMRPOL(u,D)$; erase u, D; return.

Theorem 3.3.4. Let A, B be elements of $Q[x_1,x_2]/(\Psi(x_1))$. $A = 0$ or $B = 0$ implies that $t_{PMPROD}(A,B) \sim 1$. Assume $A \neq 0$, $B \neq 0$, and define $k = \deg(\Psi)$, $\partial_1(A) = m_1$, $\partial_1(B) = n_1$, $d = v(A)$, $f = v(B)$, $g = |\Psi|_\infty$. Let ℓ be the number of non-zero coefficients of A considered as a polynomial in x_2, define h similarly for B. If $m_1 + n_1 < k$ then $t_{PMPROD}(A,B) \preceq \ell h(m_1+1)(n_1+1)L(d)L(f)$. If $m_1 + n_1 \geq k$ then $t_{PMPROD}(A,B) \preceq \ell h(m_1+n_1)^2\{(m_1+n_1-k+1)L(g)^2 + L(d)^2 + L(f)^2\}$.

Proof. Assume $A \neq 0$, $B \neq 0$. By Theorem 2.4.8, $t_{RNPROD}(r,s) \preceq L(r)L(s) \preceq L(d)L(f)$. From [COL69], $t_{PPROD}(A,B) \preceq \ell(m_1+1)h(n_1+1)L(d)L(f)$. If $m_1,n_1 > 0$, then $t_{PPROD}(A,B) \preceq \ell h\, m_1n_1\{L(d)^2+L(f)^2\} \preceq \ell h(m_1+n_1)^2\{L(d)^2+L(f)^2\}$. If $m_1 + n_1 < k$ then $t_{PMRPOL}(u,D) \sim 1$. Assume $m_1 + n_1 \geq k$, and observe that if $u = (u_1,u_2)$, $r = (r_1,r_2)$, $s = (s_1,s_2)$, then $|u_1D|_1 + u_2 \leq |r_1s_1\bar{A}\bar{B}|_1 + r_2s_2 \leq |r_1\bar{A}|_1\,|s_1\bar{B}|_1 + r_2s_2 \leq df$. Also, $\partial_1(D) = m_1+n_1$, and the number of non-zero coefficients of D is $\leq \ell h$. Therefore, by Theorem 3.3.3, $t_{PMRPOL}(u,D) \preceq \ell h(m_1+n_1)(m_1+n_1-k+1)\{(m_1+n_1-k+1)L(g)^2 + L(df)^2\} \preceq \ell h(m_1+n_1)^2\{(m_1+n_1-k+1)L(g)^2 + L(d)^2 + L(f)^2\}$. ∎

Algorithm 3.3.4. C = PMSPRD(A,b,n)

(Polynomial Modulus, Special Product)

A is an element of $Q[x_1,\ldots,x_v]/(\Psi(x_1))$, $v \geq 2$, b is an element of $Q[x_1,\ldots,x_{v-1}]/(\Psi(x_1))$ compatible with the coefficients of A, and n is a non-negative FORTRAN integer. C is the polynomial Abx_v^n.

Description

(1) [Initialize.] If A = 0 or b = 0, (C ← 0; return); $\bar{A}$ ← SECOND(A); FIRST2(s,$\bar{b}$,b).

(2) [Compute $\bar{b}(x_1,\ldots,x_v) = b(x_1,\ldots,x_{v-1})x_v^n$.] $\bar{B}$ ← PFL(PVBL($\bar{A}$), PFL(BORROW($\bar{b}$), PFA(n,0))); B ← PFL(BORROW(s), PFL($\bar{B}$,0)).

(3) [Compute Abx_v^n.] C ← PMPROD(A,B); erase B; return.

Theorem 3.3.5. Let A be an element of $Q[x_1,x_2]/(\Psi(x_1))$, b an element of $Q[x_1]/(\Psi(x_1))$, n an integer such that $n \geq 0$. A = 0 or b = 0 implies that $t_{PMSPRD}(A,b,n) \sim 1$. Assume $A \neq 0$, $b \neq 0$, and define $k = \deg(\Psi)$, $\partial_1(A) = m_1$, $\deg(b) = n_1$, $d = \nu(A)$, $f = \nu(b)$, $g = |\Psi|_\infty$. Let ℓ be the number of non-zero coefficients of A considered as a polynomial in x_2. If $m_1 + n_1 < k$ then $t_{PMSPRD}(A,b,n) \preceq \ell(m_1+1)(n_1+1)L(d)L(f)$. If $m_1 + n_1 \geq k$ then $t_{PMSPRD}(A,b,n) \preceq \ell(m_1+n_1)^2\{(m_1+n_1-k+1)L(g)^2 + L(d)^2 + L(f)^2\}$.

Proof. This follows directly from Theorem 3.3.4. ∎

Lemma 3.3.1. Let A, B be elements of $Q[y,x]/(\Psi(y))$ such that A, B are non-zero. Define $g = |\Psi|_1$, $k = \deg(\Psi)$. Then $\nu(AB) \leq kg^{k-1}\nu(A)\nu(B)$.

Proof. Let $A = ((r_1,r_2),\bar{A})$, $B = ((s_1,s_2),\bar{B})$, $\partial(A) = (m_1,m_2)$, $\partial(B) = (n_1,n_2)$. Let $\bar{D}(y,x) = \bar{A}(y,x)\bar{B}(y,x) = \sum_{i=0}^{m_2+n_2} \bar{D}_i(y)x^i$. Assume first that $m_1 + n_1 \geq k$, and let $E(y,x) = \sum_{i=0}^{m_2+n_2} \mathrm{PSREMG}(\bar{D}_i(y), \Psi(y), n_1+m_1-k+1)x^i = \sum_{i=0}^{m_2+n_2} E_i(y)x^i$. By Lemma 2.5.1, $|E_i(y)|_\infty \leq g^{n_1+m_1-k+1}|\bar{D}_i(y)|_1$. Hence $|E_i(y)|_1 \leq k\,g^{n_1+m_1-k+1}|\bar{D}_i(y)|_1 \leq kg^{k-1}|\bar{D}_i(y)|_1$. From this it follows that $|E(y,x)|_1 = \sum_{i=0}^{m_2+n_2}|E_i(y)|_1 \leq kg^{k-1}\sum_{i=0}^{m_2+n_2}|\bar{D}_i(y)|_1 = kg^{k-1}\,|\bar{D}(y,x)|_1 \leq kg^{k-1}\,|\bar{A}(y,x)|_1\,|\bar{B}(y,x)|_1$.

Now if $AB = C = ((t_1,t_2),\bar{C})$, then $\nu(C) = |t_1\bar{C}|_1 + t_2 \leq |r_1s_1E|_1 + g^{m_1+n_1-k+1}r_2s_2 \leq r_1s_1\,kg^{k-1}|\bar{A}|_1\,|\bar{B}|_1 + g^{k-1}r_2s_2 \leq kg^{k-1}(|r_1\bar{A}|_1\,|s_1\bar{B}|_1 + r_2s_2) \leq kg^{k-1}\nu(A)\nu(B)$.

If $m_1 + n_1 < k$, then $\nu(C) \leq |r_1s_1\bar{A}\bar{B}|_1 + r_2s_2 \leq |r_1\bar{A}|_1\,|s_1\bar{B}|_1 + r_2s_2 \leq \nu(A)\nu(B)$. ∎

Section 3.4. Miscellaneous Algorithms.

Algorithm 3.4.1. n = PMDEG(A)

(Polynomial Modulus, Degree)

A is an element of $Q[x_1,\ldots,x_v]/(\Psi(x_1))$, $v \geq 1$. n is the degree of A.

Description

(1) n ← 0; if A = 0, return; n ← PDEG(SECOND(A)); return.

Theorem 3.4.1. $t_{PMDEG}(A) \sim 1$.

Proof. Obvious. ∎

Algorithm 3.4.2. a = PMLDCF(A)

(Polynomial Modulus, Leading Coefficient)

A is an element of $Q[x_1,\ldots,x_v]/(\Psi(x_1))$, $v \geq 2$. a is the leading coefficient of A.

Description

(1) If A = 0, (a ← 0; return); FIRST2(r,$\bar{A}$,A); b ← PLDCF($\bar{A}$); PICPP(b,c,$\bar{a}$); t ← PONE(c); if t = 1, s ← BORROW(r); if t = 0, s ← RNISPR(c,r); erase b,c; a ← PFL(s,PFL($\bar{a}$,0)); return.

Theorem 3.4.2. Let A be an element of $Q[x_1,x_2]/(\Psi(x_1))$. If A = 0 then $t_{PMLDCF}(A) \sim 1$. If $A \neq 0$, define $\partial_1(A) = n_1$, $v(A) = d$. $t_{PMLDCF}(A) \preceq (n_1+1)L(d)^2$.

Proof. Assume $A \neq 0$. By Theorem 2.5.2, $t_{PICPP}(b) \preceq (n_1+1)L(|b|_\infty)^2 \preceq (n_1+1)L(d)^2$. By Theorem 2.4.6, $t_{RNISPR}(c,r) \preceq L(c)L(r) \preceq L(d)^2$. ∎

If $A = \sum_{i=0}^{n} a_i x^i$ is a non-zero polynomial of degree n, then the reductum of A, red(A), is the polynomial $\sum_{i=0}^{n-1} a_i x^i$. If A = 0 then red(A) = 0.

Algorithm 3.4.3. B = PMRED(A)

(Polynomial Modulus, Reductum)

A is an element of $Q[x_1,\ldots,x_v]/(\Psi(x_1))$, $v \geq 1$. B is the reductum of A.

Description

(1) B ← 0; if A = 0, return; FIRST2(r,$\bar{A}$,A); C ← PRED($\bar{A}$); if C = 0, return; PICPP(C,c,$\bar{B}$); t ← PONE(c); if t = 1, s ← BORROW(r); if t = 0, s ← RNISPR(c,r); erase C,c; B ← PFL(s,PFL($\bar{B}$,0)); return.

Theorem 3.4.3. Let A be an element of $Q[x_1,x_2]/(\Psi(x_1))$. If A = 0 then $t_{PMRED}(A) \sim 1$. Assume $A \neq 0$ and define $\partial(A) = (n_1,n_2)$, $v(A) = d$. If red(A)=0 then $t_{PMRED}(A) \sim 1$, and if $red(A)\neq 0$ then $t_{PMRED}(A) \preceq n_2(n_1+1)L(d)^2$.

Proof. Assume $A \neq 0$ and $red(A)\neq 0$. By Theorem 2.5.2 $t_{PICPP}(C) \preceq (\partial_2(C)+1)(\partial_1(C)+1)L(|C|_\omega)^2 \preceq n_2(n_1+1)L(d)^2$. By Theorem 2.4.6, $t_{RNISPR}(c,r) \preceq L(c)L(r) \preceq L(d)^2$. ∎

Algorithm 3.4.4. V = PMVBL(A)

(Polynomial Modulus, Variable)

A is an element of $Q[x_1,\ldots,x_v]/(\Psi(x_1))$, $v \geq 1$. If $A = 0$ then $V = 0$. If $A \neq 0$ then $V = x_v$, the main variable of A.

Description

(1) V ← 0; if A = 0, return; V ← PVBL(SECOND(A)); return.

Theorem 3.4.4. $t_{PMVBL}(A) \sim 1$.

Proof. Obvious. ∎

Algorithm 3.4.5. A = PMPCL(L,W)

(Polynomial Modulus, Polynomial from Coefficient List)

L is a list $(e_1,A_1,\ldots,e_n,A_n)$ where $A_1,\ldots,A_n$ are non-zero elements of $Q[x_1,\ldots,x_v]/(\Psi(x_1))$, $v \geq 1$, and $e_1,\ldots,e_n$ are FORTRAN integers such that $0 \leq e_1 < \cdots < e_n$. W is a variable. $A = \sum_{i=1}^{n} A_i W^{e_i}$, an element of $Q[x_1,\ldots,x_v,W]/(\Psi(x_1))$.

Description

(1) [Initialize.] A ← 0; if L = 0, return; T ← U ← L; ℓ ← PFA(1,0); B ← 0.

(2) [Compute lcm of denominators of coefficients.] If T = 0, go to (3); ADV2(e,A_1,T); d ← SECOND(FIRST(A_1)); v ← ILCM(d,ℓ); erase ℓ; ℓ ← v; go to (2).

(3) [Multiply lcm times coefficients.] If U = 0, go to (4); ADV2(e,A_1,U); ADV2(r,$\bar{A}_1$,A_1); c ← IRNSPR(ℓ,r); C ← PIP($\bar{A}_1$,c); erase c; B ← PFL(C,PFA(e,B)); go to (3).

(4) [Form rp(A), polp(A) from B and ℓ.] B ← PFL(BORROW(W),B); PICPP(B,c,$\bar{A}$); r ← RNINT2(c,ℓ); erase c,ℓ,B; A ← PFL(r,PFL($\bar{A}$,0)); return.

Theorem 3.4.5. If $L = 0$ then $t_{PMPCL}(L,W) \sim 1$. Let $L = (e_1, A_1,\ldots,e_n,A_n)$ where $n \geq 1$ and $A_1,\ldots,A_n$ are non-zero elements of $Q[x_1]/(\Psi(x_1))$. If $h = \max(v(A_1),\ldots,v(A_n))$ and $k = \deg(\Psi)$, then $t_{PMPCL}(L,W) \preceq k\, n^3 L(h)^2$.

Proof. Assume $L \neq 0$. $\mathrm{lcm}(a,b) = ab/\gcd(a,b) \leq ab$, so prior to the i^{th} execution of step (2), $L(\ell) \leq iL(h)$. Then from page 30 of [COL73d], $t_{ILCM}(d,\ell) \preceq L(d)L(\ell) \leq L(h)iL(h) = iL(h)^2$. Therefore $T_2 \preceq \sum_{i=1}^{n} iL(h)^2 \sim n^2L(h)^2$.

By Theorem 2.3.2, the time for the i^{th} execution of IRNSPR(ℓ,r) is $\preceq L(\ell)L(r) \preceq nL(h)^2$. The time for the i^{th} execution of PIP($\bar{A}_1$,c) is $\preceq \{\deg(\bar{A}_1)+1\}L(c)L(|\bar{A}_1|_\infty) \preceq k(n+1)L(h)L(h) \sim k\,nL(h)^2$. Hence $T_3 \preceq k\,n^2L(h)^2$.

In step (4), $t_{PICPP}(B) \preceq n\,kL(|B|_\infty)^2 \preceq n\,kn^2L(h)^2 = k\,n^3L(h)^2$ by Theorem 2.5.2. By Theorem 2.4.5, $t_{RNINT2}(c,\ell) \preceq L(c)L(\ell) \preceq n^2L(h)^2$. ∎

Since the elements of $Q[x_1,\ldots,x_v]/(\Psi(x_1))$, $v \geq 1$, are represented as lists, they may be erased by application of the general SAC-1 erasure program, ERASE. However, since the elements have a very special structure, it is more efficient to construct an algorithm which takes advantage of a knowledge of this structure. PMERAS does this, primarily by application of the specialized erasure routines RNERAS to rp(A) and

PERASE to polp(A), where A is the element being erased.

Algorithm 3.4.6. PMERAS(A)

(Polynomial Modulus, Erase)

A is an element of $Q[x_1,\ldots,x_v]/(\Psi(x_1))$, $v \geq 1$. A is erased and set to the null list.

Description

(1) $c \leftarrow$ COUNT(A) - 1; if $c > 0$, (SCOUNT(c,A); A $\leftarrow$ 0; return); DECAP(t,A); if A $\neq$ 0, (RNERAS(t); go to (1)); PERASE(t); return.

Theorem 3.4.6. Let A be an element of $Q[x_1,\ldots,x_v]/(\Psi(x_1))$, $v \geq 1$, and let n be the number of cells returned to the available space list by PMERAS(A). $t_{PMERAS}(A) \sim n + 1$.

Proof. This follows directly from the computing times of RNERAS and PERASE. ∎

Section 3.5. Division Operations

It follows from the discussion preceeding Definition 1.4.9 that one can form the unique quotient and remainder of two elements, A, B, of $Q[x_1,\ldots,x_v]/(\Psi(x_1))$ if ldcf(B) is a unit. In particular, if B is monic, the quotient and remainder can be constructed. The algorithm which follows performs this task for input B having this property.

Suppose A, B are non-zero elements of $Q[x_1,\ldots,x_v]/(\Psi(x_1))$ such that $m = \deg(A) \geq n = \deg(B)$, and B is monic. PMQRMM computes the sequence of polynomials $C_0 = A, C_1, \ldots, C_e = \text{rem}(A,B)$, where

$$C_i = C_{i-1} - c_{i-1} B x_v^{m_{i-1}-n} \quad (1)$$

$\deg(C_i) = m_i$, $\text{ldcf}(C_i) = c_i$, and $e \leq m - n + 1$. Then we also have that the quotient of A, B is $\sum_{i=0}^{e-1} c_i x_v^{m_i-n}$.

Algorithm 3.5.1. L = PMQRMM(A,n,B_1)

(Polynomial Modulus, Quotient and Remainder, Monic)

A, B_1 are elements of $Q[x_1,\ldots,x_v]/(\Psi(x_1))$, $v \geq 1$. n is a non-negative FORTRAN integer; $n > \deg(B_1)$ or $B_1 = 0$. If $B = x_v^n + B_1$, then L is the list (Q,R), where Q = quo(A,B) and R = rem(A,B).

Description

(1) [Initialize.] C $\leftarrow$ BORROW(A); if A = 0 or PMDEG(A) < n, (L $\leftarrow$ PFL(0,PFL(C,0)); return); T $\leftarrow$ 0.

(2) [Construct C_{i+1} from C_i and B_1.] If $C = 0$, go to (3); $\ell \leftarrow$ PMDEG(C) - n; if $\ell < 0$, go to (3); $c \leftarrow$ PMLDCF(C); $C_1 \leftarrow$ PMRED(C); erase C; $D \leftarrow$ PMSPRD(B_1,c,ℓ); $C \leftarrow$ PMDIF(C_1,D); erase C_1,D; $T \leftarrow$ PFA(ℓ,PFL(c,T)); go to (2).

(3) [Finish up.] $R \leftarrow C$; $V \leftarrow$ PMVBL(A); $Q \leftarrow$ PMPCL(T,V); erase T, V; $L \leftarrow$ PFL(Q,PFL(R,0)); return.

Before proceeding to analyze the computing time of PMQRMM we must first prove a theorem concerning the sequence $C_0,\ldots,C_e$.

Lemma 3.5.1. Let A,B be elements of $Q[x_1,x_2]/(\Psi(x_1))$ such that B is monic. Let $k = \deg(\Psi)$, $g = |\Psi|_1$, $\ell = \mathrm{ldcf}(\Psi)$, $m_2 = \partial_2(A) \geq n_2 = \partial_2(B)$, $A_{m_2} = \mathrm{ldcf}(A)$ and $B_1 = \mathrm{red}(B)$. If $C(x_1,x_2) = A(x_1,x_2) - A_{m_2}(x_1)B(x_1,x_2)x_2^{m_2-n_2}$, then $\nu_1(C) \leq kg^{k-1}\nu_1(A)\nu(B_1)$, $\nu_2(C)|\ell^{k-1}\nu_2(A)\nu_2(B_1)$, and $\nu(C) \leq kg^{k-1}\nu(A)\nu(B_1)$.

Proof. Let $A = ((r_1,r_2),\bar{A})$, $B_1 = ((s_1,s_2),\bar{B}_1)$, $\bar{A}_{m_2} = \mathrm{ldcf}(\bar{A})$, and $\bar{A}_1 = \mathrm{red}(\bar{A})$. Then $A(x_1,x_2) - A_{m_2}(x_1)B(x_1,x_2)x_2^{m_2-n_2} = \frac{r_1\bar{A}_1(x_1,x_2)}{r_2} - \frac{r_1s_1\bar{D}(x_1,x_2)}{\ell^m r_2 s_2}$, where $m = \max(\deg(\bar{A}_{m_2}) + \partial_1(B)-k+1,0)$, $\bar{D}(x_1,x_2) = \mathrm{PMIPOL}(\bar{A}_{m_2}(x_1)\bar{B}_1(x_1,x_2)x_2^{m_2-n_2}, m, \ell^m)$ if $m > 0$, and $\bar{D}(x_1,x_2) = \bar{A}_{m_2}(x_1)\bar{B}_1(x_1,x_2)x_2^{m_2-n_2}$ if $m = 0$. This implies that $\nu_2(C)|\ell^m\nu_2(A)\nu_2(B_1)|\ell^{k-1}\nu_2(A)\nu_2(B_1)$. Also, $\nu_1(C) \leq |\ell^m s_2 r_1\bar{A}_1 - r_1s_1\bar{D}|_1 \leq g^{k-1}s_2|r_1\bar{A}_1|_1 + r_1s_1|\bar{D}|_1$. By Lemma 2.5.1, $\nu_1(C) \leq g^{k-1}s_2|r_1\bar{A}_1|_1 + r_1s_1kg^{k-1}|\bar{A}_{m_2}|_1\,|\bar{B}_1|_1 \leq kg^{k-1}|r_1\bar{A}|_1\{s_2 + |s_1\bar{B}_1|_1\} = kg^{k-1}\nu_1(A)\nu(B_1)$.

$\nu(C) = \nu_1(C) + \nu_2(C) \leq kg^{k-1}\nu_1(A)\nu(B_1) + g^{k-1}\nu_2(A)\nu_2(B_1) \leq kg^{k-1}\{\nu_1(A)\nu(B_1) + \nu_2(A)\nu_2(B_1)\} \leq kg^{k-1}\nu(A)\nu(B_1)$. ∎

Theorem 3.5.1. $\nu(C_i) \leq \{kg^{k-1}\nu(B_1)\}^i\nu(A)$ for $0 \leq i \leq e$.

Proof. This follows by induction from (1) and Lemma 3.5.1. ∎

Theorem 3.5.2. Let A, B_1 be elements of $Q[x_1,x_2](\Psi(x_1))$. Let n be a non-negative integer such that $n > \deg(B_1)$. Let $m_2 = \partial_2(A)$. If $A = 0$ or $m_2 < n$ then $t_{\mathrm{PMQRMM}}(A,n,B_1) \sim 1$. Assume $A \neq 0$ and $m_2 \geq n$. If $d = \nu(A)$, $f = \nu(B_1)$, $g = |\Psi|_1$, $k = \deg(\Psi)$, then $t_{\mathrm{PMQRMM}}(A,n,B_1) \preceq k^2(m_2-n+1)^2(m_2+1)\{(m_2-n+1)^2(k^2L(g)^2+L(f)^2)+L(d)^2\}$.

Proof. Assume $A \neq 0$, $m_2 \geq n$. Observe that by Theorem 3.5.1, $L(\nu(C_i)) \preceq i\{kL(g) + L(f)\} + L(d)$ and hence $L(\nu(C_i))^2 \preceq i^2\{k^2L(g)^2+L(f)^2\} + L(d)^2$. Also, $1 \leq e \leq m_2 - n + 1$, so $L(\nu(C_i))^2 \leq (m_2-n+1)^2\{k^2L(g)^2 + L(f)^2\} + L(d)^2$.

We now examine the computing time for the i^{th} execution of step (2). By Theorem 3.4.2, $t_{\mathrm{PMLDCF}}(C) \preceq kL(\nu(C_i))^2 \preceq k\{(m_2-n+1)^2(k^2L(g)^2 + L(f)^2) + L(d)^2\}$. By Theorem 3.4.3, $t_{\mathrm{PMRED}}(C) \preceq \{\deg(C_1)+1\}kL(\nu(C_i))^2 \preceq (m_2+1)k\{(m_2-n+1)^2(k^2L(g)^2+L(f)^2)+L(d)^2\}$. By Theorem 3.3.5, $t_{\mathrm{PMSPRD}}(B_1,c,\ell) \preceq (n+1)k^2\{kL(g)^2+L(f)^2+L(\nu(c))^2\} \preceq (m_2+1)k^2\{(m_2-n+1)^2(k^2L(g)^2 + L(f)^2)+L(d)^2\}$. By Theorem 3.2.3, $t_{\mathrm{PMDIF}}(C_1,D) \preceq \{(m_2+1)+(n+1)\}k\{L(\nu(C_1))^2+L(\nu(D))^2\}$. Hence, by Lemma 3.3.1, $t_{\mathrm{PMDIF}}(C_1,D) \preceq k(m_2+1)\{(m_2-n+1)^2(k^2L(g)^2+L(f)^2) + L(d)^2\}$.

Since $N_2 \le m_2 - n + 1$, $T_2 \le (m_2-n+1)k^2(m_2+1)\{(m_2-n+1)^2(k^2L(g)^2 + L(f)^2) + L(d)^2\}$.

By Theorem 3.4.5, $t_{PMPCL}(T,V) \le k(m_2-n+1)^3\{(m_2-n+1)^2(k^2L(g)^2 + L(f)^2) + L(d)^2\} \le k(m_2-n+1)^2(m_2+1)\{(m_2-n+1)^2(k^2L(g)^2 + L(f)^2) + L(d)^2\}$. ∎

Theorem 3.5.3. Let $(Q,R) = PMQRMM(A,n,B_1)$. Let $B(x_1,x_2) = x_2^n + B_1(x_1,x_2)$. If m,n,k,g are as defined above, then $\nu(Q) \le k(m-n+1)\{g^{k-1}\nu(B)\}^{m-n}\nu(A)$ and $\nu(R) \le \{kg^{k-1}\nu(B_1)\}^{m-n+1}\nu(A)$.

Proof. By Theorem 3.5.1, $\nu(C_i) \le \{kg^{k-1}\nu(B_1)\}^i\nu(A)$. Hence $\nu(R) = \nu(C_e) \le \{kg^{k-1}\nu(B_1)\}^{m-n+1}\nu(A)$.

Let M be the $m-n+2$ by $m+1$ matrix whose rows are the coefficients of $A(x_2)$, $B(x_2)x_2^{m-n},\ldots,B(x_2)x_2$ and $B(x_2)$. Define M_j to be the square submatrix of M consisting of the first $m-n+1-j$ rows and columns of M. Then

$$Q(x_2) = \sum_{j=0}^{m-n} (-1)^{m-n-j} q_j x_2^j ,$$

where $q_j = rem(det(M_j),\Psi)$. Let $\hat{A} = r_1\bar{A}$, $\hat{B} = s_1\bar{B}$, where $A = ((r_1,r_2),\bar{A})$, $B = ((s_1,s_2),\bar{B})$.

Let $\hat{M}$ be the matrix whose rows are the coefficients of $\hat{A}(x_2)$, $\hat{B}(x_2)x_2^{m-n},\ldots,\hat{B}(x_2)x_2$ and $\hat{B}(x_2)$. Let $\hat{M}_j$ be the square submatrix of $\hat{M}$ consisting of the first $m-n+1-j$ rows and columns of $\hat{M}$. Then, since $r_2\bar{A} = \hat{A}$ and $s_2\bar{B} = \hat{B}$,

$$det(\hat{M}_j) = r_2 s_2^{m-n-j} det(M_j),$$

and

$$q_j = r_2^{-1}s_2^{-(m-n-j)} rem(det(\hat{M}_j),\Psi).$$

If $n_j = deg(det(\hat{M}_j))$, $\ell = ldcf(\Psi)$ and $m_j = max(n_j-k+1,0)$, then

$$q_j = r_2^{-1}s_2^{-(m-n-j)}\ell^{-m_j} prem(det(\hat{M}_j),\Psi). \tag{2}$$

By Theorem 1 of [COHO73], $|det(\hat{M}_j)|_1 \le |\hat{A}|_1\,|\hat{B}|_1^{m-n-j}$. Therefore,

$$|prem(det(\hat{M}_j),\Psi)|_\infty \le g^{m_j}|\hat{A}|_1\,|\hat{B}|_1^{m-n-j} \tag{3}$$

by Lemma 2.5.1.

Now observe that since $n_j \le (m-n+1-j)(k-1)$, $(m-n-j)(k-1) - m_j \ge 0$. Therefore

$$q_j^* = \ell^{(m-n-j)(k-1)-m_j} prem(det(\hat{M}_j),\Psi) \tag{4}$$

is integral. And it follows from (3) that

$$|q_j^*|_\infty \le \{g^{k-1}|\hat{B}|_1\}^{m-n-j}|\hat{A}|_1 . \tag{5}$$

It follows from (2) and (4) that

$$q_j = q_j^* / r_2(\ell^{k-1}s_2)^{m-n-j} .$$

Hence $Q(x_2) = \sum_{j=0}^{m-n}(-1)^{m-n-j}\{q_j^*/r_2(\ell^{k-1}s_2)^{m-n-j}\}x_2^j = \{1/r_2(\ell^{k-1}s_2)^{m-n}\}\sum_{j=0}^{m-n}(-1)^{m-n-j}(\ell^{k-1}s_2)^j q_j^* x_2^j$. Therefore $\nu_2(Q) \le r_2(\ell^{k-1}s_2)^{m-n} = \nu_2(A)\{\ell^{k-1}\nu_2(B)\}^{m-n}$, and by (5), $\nu_1(Q) \le \sum_{j=0}^{m-n}(\ell^{k-1}s_2)^j\times\{g^{k-1}|\hat{B}|_1\}^{m-n-j}$

$|\hat{A}|_1 \leq k|\hat{A}|_1 \sum_{j=0}^{m-n} \{g^{k-1}(s_2 + |\hat{B}|_1)\}^{m-n} = k(m-n+1)\nu_1(A)\{g^{k-1}\nu(B)\}^{m-n}$.

$\nu(Q) = \nu_1(Q) + \nu_2(Q) \leq k(m-n+1)\{g^{k-1}\nu(B)\}^{m-n}\nu(A)$. ∎

CHAPTER 4: OPERATIONS IN Q(α) AND Q(α)[x]

Section 4.1. Introduction

In the previous chapter we discussed the residue class ring $Q[x_1,\ldots,x_v]/(\Psi(x_1))$ where Ψ is any primitive, integral polynomial of positive degree with $\mathrm{ldcf}(\Psi) > 0$. Suppose now that Ψ is also square-free and has no linear factors, that $v = 1$ or $v = 2$, and that α is a real algebraic number such that $\Psi(\alpha) = 0$ (note that if P is any non-zero integral polynomial such that $P(\alpha) = 0$, then a suitable Ψ can be determined by use of Algorithm 2.5.1). Suppose we further hypothesize the availability of an interval, $\Omega = (r,s)$ where r,s are rational numbers, which contains exactly one root of $\Psi(x_1)$, namely α. We then have sufficient materials, based on the algorithms in the previous chapter, for performing the arithmetic operations in both $Q(\alpha)$ and $Q(\alpha)[x]$, and for deciding whether any given element of $Q(\alpha)$ is zero, positive, or negative.

Any irrational real algebraic number can be chosen to be α, the choice being left to the user. The choices of Ψ and Ω to determine α are arbitrary, in so far as they satisfy the criteria stated above. Moreover, it should be noted that Ψ need not be irreducible for the operations to be effective. Ψ will frequently be referred to as the "polynomial modulus", or simply the "modulus", while Ω may be referred to as the "isolating interval", or the "interval". The algorithms assume the availability of Ψ and Ω, just as Ψ was assumed in the previous chapter, so they need not be passed as parameters. In the implementation, Ω appears in the same common block as does Ψ, namely TR7, as the variable OMEGA.

We make use of the evaluation homomorphism ψ_α for performing operations in $Q(\alpha)$. ψ_α is a mapping from $Q[x_1]/(\Psi(x_1))$ onto $Q(\alpha)$ defined by $\psi_\alpha(A(x_1)) = A(\alpha)$. ψ_α can be extended to polynomials over $Q[x_1]/(\Psi(x_1))$ in the usual manner; that is, $\psi_\alpha(A(x_1,x_2)) = \sum_{i=0}^{n} \psi_\alpha(A_i(x_1))x_2^i$ where $A(x_1,x_2) = \sum_{i=0}^{n} A_i(x_1)x_2^i$. ψ_α is an isomorphism if and only if Ψ is irreducible.

If a is an element of $Q(\alpha)$, then we represent a by an element $A(y)$, belonging to $Q[y]/(\Psi(y))$ such that $\psi_\alpha(A) = a$; furthermore, we make the restriction that if $a = 0$ then A is the zero polynomial. Similarly, if $a(x)$ is an element of $Q(\alpha)[x]$, then we represent $a(x)$ by an element, $A(y,x)$, belonging to $Q[y,x]/(\Psi(y))$ such that $\psi_\alpha(A(y,x)) = A(\alpha,x) = a(x)$; we also make the restriction that if $\hat{A}(y)$ is any coefficient of $A(y,x)$, then $\hat{A}(y)$ is the zero polynomial if $\hat{A}(\alpha) = 0$. We do not have a canonical form for $Q(\alpha)$, nor for $Q(\alpha)[x]$, if Ψ is not irreducible. As noted above, ψ_α is not an isomorphism in this instance since $\psi_\alpha(A) = 0$ if A is any multiple of the minimal polynomial of α.

However, since we have an ordering algorithm for $Q(\alpha)$, and hence for $Q(\alpha)[x]$, we can determine whether any element of $Q[y]/(\Psi(y))$ belongs to the kernel of ψ_α, and therefore we have effective procedures for all operations in $Q(\alpha)$ and $Q(\alpha)[x]$. This chapter, then, is concerned with developing algorithms for performing elementary operations in this field and polynomial ring. These include the usual arithmetic operations, determining the order relation, and other related procedures.

In this and the succeeding chapter, we will frequently refer to, say, a, as an element of $Q(\alpha)$ or $Q(\alpha)[x]$ when we really mean the element of $Q[y]/(\Psi(y))$ or $Q[y,x]/(\Psi(y))$ which represents a. This should cause no confusion as long as we remember that, for example, when we say that "a is an element of $Q(\alpha)$", we mean, "a is an element of $Q[y]/(\Psi(y))$ such that if $\psi_\alpha(a) = 0$ then a is identically zero".

Section 4.2. The Order Relation in $Q(\alpha)$

First we will show how to determine if $A(\alpha) = 0$, where A is a univariate, integral polynomial. This algorithm will be used by AREP, which determines the representation of an algebraic number from an element of $Q[y]/(\Psi(y))$, or a polynomial from an element of $Q[y,x]/(\Psi(y))$. Finally, we shall exhibit an effective procedure which given A, an element of $Q(\alpha)$ or $Q(\alpha)[x]$, determines sign(A).

Theorem 4.2.1. Let A be a non-zero element of $I[x]$ and let $\Omega = (r,t)$. Define $B = \gcd(A,\Psi)$, $s_1 = \text{sign}(B(r))$, and $s_2 = \text{sign}(B(t))$. $A(\alpha) = 0$ if and only if $s_1 \cdot s_2 = -1$.

Proof. Suppose $A(\alpha) = 0$. This implies $B(\alpha) = 0$ since α is a common root of A and Ψ. Ψ has no rational roots so neither does B since $B|\Psi$, hence $s_1 \neq 0$ and $s_2 \neq 0$. Furthermore, α is the unique root of B in Ω since $B|\Psi$, which also implies that B is square-free and therefore $s_1 \cdot s_2 = -1$.

Conversely, if $s_1 \cdot s_2 = -1$, then B has a root, γ, in Ω. But γ must be a root of Ψ so $\gamma = \alpha$. $B|A$ so $A(\alpha) = 0$. ∎

Algorithm 4.2.1. s = AZTEST(A)

(Algebraic Zero Test)

A is an element of $I[x]$. $s = 0$ if $A(\alpha) = 0$, and $s = 1$ otherwise.

Description

(1) [Compute gcd(A,Ψ).] $s \leftarrow 0$; if $A = 0$, return; $B \leftarrow \text{PGCD}(A,\Psi)$.

(2) [Determine s_1, s_2.] FIRST2(r,t,Ω); $s_1 \leftarrow \text{REVAL}(B,r)$; $s_2 \leftarrow \text{REVAL}(B,t)$; erase B; if $s_1 \cdot s_2 = 1$, $s \leftarrow 1$; return.

Theorem 4.2.2. Let A be an element of $I[x]$. If $A = 0$ then $t_{\text{AZTEST}}(A) \sim 1$. If $A \neq 0$ and $\Omega = (r,t]$, define $k = \deg(\Psi)$, $n = \deg(A)$, $d = |A|_1$, $g = |\Psi|_1$, $\ell = \max(k,n)$, $e = \text{mep}(\Omega)$. Then $t_{\text{AZTEST}}(A) \preceq \ell^3 L(dg)^2 + k^3 L(e) + k^2 L(e)^2$.

Proof. Suppose $A \neq 0$. $t_{\text{PGCD}}(A,\Psi) \preceq \max(n,k)^3 L(dg)^2 = \ell^3 L(dg)^2$. Note that $L(r), L(t) \preceq L(e)$. If $f = |B|_1$ then by Corollary 1.4.1 $f \leq (k+1)^{2k} g$, and if $m = \deg(B)$ then $m \leq k$. Therefore, from a result in [HEI70], $t_{\text{REVAL}}(B,r) \preceq m^2 L(r)^2 + m^2 L(f) L(r) \preceq k^2 L(e)^2 + k^2 L((k+1)^{2k} g)$ $L(e) \sim k^2 L(e)^2 + k^2 \{kL(k) + L(g)\} L(e) \preceq k^2 L(e)^2 + k^3 L(e) + k^2 L(g) L(e)$, assuming that $L(k) \sim 1$. $t_{\text{REVAL}}(B,t)$ has the same bound. Then $t_{\text{AZTEST}}(A) \sim t_{\text{PGCD}}(A,\Psi) = t_{\text{REVAL}}(B,r) + t_{\text{REVAL}}(B,t) \preceq \ell^3 L(dg)^2 + k^2 L(e)^2 + k^3 L(e) + k^2 L(g) L(e) \preceq \ell^3 L(dg)^2 + k^3 L(e) + k^2 L(e)^2$, since $k^2 L(g) L(e) \preceq k^2 L(dg)^2 + k^2 L(e)^2$. ∎

Algorithm 4.2.2. B = AREP(A)

(Algebraic Representation)

A is an element of $Q[y]/(\Psi(y))$ or $Q[y,x]/(\Psi(y))$. If A is an element of $Q[y]/(\Psi(y))$ then $B = 0$ if $\psi_\alpha(A) = 0$, and $B = A$ otherwise. If A is an element of $Q[y,x]/(\Psi(y))$ then B is the polynomial formed from A by deleting the coefficients of A which

belong to the kernel of ψ_α.

Description

(1) [Determine the ring to which A belongs.] B ← 0; if A = 0, return; FIRST2(r,$\bar{A}$,A); if TYPE(SECOND($\bar{A}$)) = 0, go to (2); C ← 0; T ← TAIL($\bar{A}$); go to (3).

(2) [Compute B for the univariate case.] s ← AZTEST($\bar{A}$); if s = 1, B ← BORROW(A); return.

(3) [Test the coefficients of A.] If T = 0, go to (4); ADV2(a,n,T); if AZTEST(a) = 1, C ← PFA(n,PFL(BORROW(a),C)); go to (3).

(4) [Compute rp(B), polp(B).] If C = 0, return; C ← INV(C); C ← PFL(PVBL($\bar{A}$),C); PICPP(C,c,$\bar{B}$); t ← PONE(c); if t = 1, s ← BORROW(r); if t = 0, s ← RNISPR(c,r); erase c, C; B ← PFL(s,PFL($\bar{B}$,0)); return.

Theorem 4.2.3. If A = 0 then $t_{AREP}(A) \sim 1$. If $A \neq 0$, define $e = mep(\Omega)$, $k = deg(\Psi)$, $d = \nu(A)$, and $g = |\Psi|_1$. If A is an element of $Q[y]/(\Psi(y))$ then $t_{AREP}(A) \leq k^2\{kL(dg)^2+kL(e)+L(e)^2\}$. If A is an element of $Q[y,x]/(\Psi(y))$ and ℓ is the number of non-zero coefficients of A considered as a polynomial in x, then $t_{AREP}(A) \leq \ell k^2\{kL(dg)^2+kL(e)+L(e)^2\}$.

Proof. Assume $A \neq 0$. If A is an element of $Q[y]/(\Psi(y))$ then the result follows directly from Theorem 4.2.2. Assume A is an element of $Q[y,x]/(\Psi(y))$. Then by Theorem 4.2.2, $T_3 \leq \ell k^2\{kL(dg)^2+kL(e)+L(e)^2\}$. By Theorem 2.5.2, $t_{PICPP}(C) \leq \ell(\partial_y(C)+1)L(|C|_\infty)^2 \leq \ell kL(d)^2$. By Theorem 2.4.6, $t_{RNISPR}(c,r) \leq L(c)L(r) \leq L(d)^2$. ∎

Suppose $\bar{A}$ is a univariate, integral polynomial. If $\bar{A}(\alpha) \neq 0$, then an interval $I \subseteq \Omega$ can be found such that $\bar{A}$ has no roots in I but $\alpha \in I$. I can be found by using interval bisection and the algorithm RRTS (from [HEI70]), which employs Sturm's Theorem to isolate the roots of a polynomial. Since $\bar{A}$ has no roots in I, sign($\bar{A}(\alpha)$) = sign($\bar{A}(r)$) for any rational number $r \in I$.

If its input, a, is non-zero, the algorithm ASIGN first determines whether a is an algebraic number or polynomial. In the former case, let $\bar{A}$ = polp(a), and let $\bar{A}$ = polp(ldcf(a)) otherwise. Let $\bar{A}'$ = gsfd($\bar{A}$). RRTS is used to determine n, the number of roots $\bar{A}'$ has in $\Omega = (r,t]$. If n = 0, then sign($\bar{A}(\alpha)$) = sign($\bar{A}(t)$). If $n \neq 0$ the interval is successively bisected, and RRTS is applied to $\bar{A}'$ and the subinterval containing α to determine if $\bar{A}'$ has a root in the interval. This process continues until an interval (r',t'] containing α and at most one root of $\bar{A}'$ is found.

If the number of roots in (r',t'] is zero then sign($\bar{A}(\alpha)$) = sign($\bar{A}(t')$). If the number of roots is one, the interval is successively bisected, and Ψ and $\bar{A}'$ are evaluated at the endpoints until an open interval, say I, containing no roots of $\bar{A}'$ but containing α is found. Since I is a subinterval of an interval containing exactly one root of $\bar{A}'$, at most one endpoint of I is a root of $\bar{A}$. Therefore the sign of $\bar{A}(\alpha)$ is the sign of $\bar{A}$ evaluated at an endpoint of I which is not a root of $\bar{A}$. Such an endpoint is determined and the evaluation performed; the algorithm then terminates.

Algorithm 4.2.3. s = ASIGN(a)

(Algebraic, Sign)

a is an element of $Q(\alpha)$ or $Q(\alpha)[x]$, s = sign(a).

Description

(1) [Determine whether a belongs to $Q(\alpha)$ or $Q(\alpha)[x]$.] s ← 0; if a = 0, return; $\bar{A}$ ← BORROW(SECOND(a)); if TYPE(SECOND($\bar{A}$)) = 1, ($\bar{A}$ ← PLDCF($\bar{A}$); erase $\bar{A}$; PICPP(A,c,$\bar{A}$); erase A,c); if PDEG($\bar{A}$) = 0, (s ← PSIGN($\bar{A}$); erase $\bar{A}$; return).

(2) [Initialize for RRTS and interval bisection.] FIRST2(r,t,Ω); r ← BORROW(r); t ← BORROW(t); $\bar{A}'$ ← PGSFD($\bar{A}$); S ← 0; s_1 ← REVAL(Ψ,r); s_2 ← $-s_1$; t_1 ← t_2 ← 2.

(3) [Determine the number of roots $\bar{A}$ has in I.] I ← PFL(r,PFL(t,0)); RRTS($\bar{A}'$,I,0,s,n,X); DECAP2(r,t,I); if n = 0, go to (7).

(4) [Bisect I and determine which half contains α.] v ← RNAVER(r,t); $\bar{s}$ ← REVAL(Ψ,v); if $s_1 \bar{s} < 0$, (erase t; t ← v); if $\bar{s}\, s_2 < 0$, (erase r; r ← v); if n ≠ 1, go to (3).

(5) [Determine if $\bar{A}$ has any roots in I.] If $t_1 = 2$, (t_1 ← REVAL($\bar{A}'$,r); t_2 ← REVAL($\bar{A}'$,t); go to (6)); if $\bar{s} = s_1$, (t_1 ← REVAL($\bar{A}'$,r); go to (6)); t_2 ← REVAL($\bar{A}'$,t).

(6) [Loop if a root of $\bar{A}$ remains in I.] If $t_1 t_2 < 0$, go to (4).

(7) [Obtain sign($\bar{A}$(α)).] If $t_2 \neq 0$, (s ← REVAL($\bar{A}$,t); go to (8)); s ← REVAL($\bar{A}$,r).

(8) [Finish up.] Erase S, r, t, $\bar{A}$, $\bar{A}'$; return.

Theorem 4.2.4. Let $a \in Q(\alpha)$. If a = 0 then $t_{ASIGN}(a) \sim 1$. If $a \neq 0$, define $k = \deg(\Psi)$, $g = |\Psi|_1$, $\bar{A} = \text{polp}(a)$, $h = |\bar{A}|_1$, $f = |\text{gsfd}(\bar{A})|_1$. Let c be the least positive integer such that $c^{-1} \leq \text{sep}(\Psi\bar{A})$. Then $t_{ASIGN}(a) \preceq k^3\{L(cg)^3 + L(h)^2\} + k^4 L(cg)^2 L(f) + k^5 L(f)^2$ if Ω is in the list L = ANALR(Ψ,0).

Proof. Assume $a \neq 0$ and let $\Omega = (r,t)$, $r = (r_1,r_2)$, $t = (t_1,t_2)$. Since Ω is in the list L = ANALR(Ψ,0), it follows from the proof of Theorem 2.6.8 of [HEI70] that $|r_1|, |t_1| \leq 8B^2c$ and $r_2, t_2 \leq 4Bc$, where B is a bound on the magnitude of the roots of Ψ. From Corollary 2.6.1 of the same paper, $B \leq 2g$ and hence $L(r), L(t) \preceq L(32g^2c) \sim L(cg)$.

If N_i is the number of times step(i) is executed, then $N_1 = N_2 = N_7 = N_8 = 1$. There are two loops in the algorithm, the first consisting of steps (3)-(4) and the second of steps (4)-(6). This implies that $N_3, N_5, N_6 \preceq N_4$. N_4 is just the number of times it is necessary to bisect Ω until α is isolated from any root of $\bar{A}$; hence $N_4 \leq m$, where m is the least positive integer such that $(t-r)/2^m < c^{-1}$, or equivalently, $(t-r)/2^{m-1} \geq c^{-1}$. However, $t - r \leq 4g$ so that $4cg \geq 2^{m-1}$, and $2 + \log_2 cg \geq m - 1$. But $\log_2 cg \sim L(cg)$ so that $N_4 \leq m \preceq L(cg)$.

Now let $n = \deg(\bar{A})$ and assume $L(k) = 1$. In step (2), it follows from [PIN73] that $t_{PGSFD}(\bar{A}) \preceq n^3 L(nh)^2 \preceq k^3 L(h)^2$, while $t_{REVAL}(\Psi,r) \preceq k^2 L(r)^2 + k^2 L(g) L(r) \preceq k^2 L(cg)^2 + k^2 L(g) L(cg) \sim k^2 L(cg)^2$

By Theorem 2.6.6 of [HEI70], for the first execution of RRTS in

step (3) we have $t_{RRTS}(\bar{A}',I,0,s) \preceq n^3L(cg)^2 + n^4L(cg)L(f) + n^5L(f)^2 \preceq k^3L(cg)^2 + k^4L(cg)L(f) + k^5L(f)^2$. To determine the time for subsequent executions, we must have a bound on the endpoints of I. Let $e = \max(r_2,t_2)$, r_1', t_1' be the integers such that $r_1'/e = r$, $t_1'/e = t$, $d = \max(r_1',t_1',e)$, and let I_i be the interval I prior to the $i+1^{st}$ bisection of I in step (4). 2^id is a bound on the magnitude of the numerators and denominators of I_i, $i \geq 0$. This can be shown inductively by observing that the denominators are positive powers of 2. Thus if a_i is a numerator or denominator from I_i, then $L(a_i) \preceq iL(2) + L(d) \preceq i + L(d) \preceq L(cg) + L(d) \preceq L(cg)$. Therefore, after the first execution of step (3), it follows from Theorem 2.6.5 of [HEI70] that $t_{RRTS}(\bar{A}',I,0,s) \preceq k^3L(cg)^2 + k^4L(cg)L(f)$. This implies that $T_3 \preceq k^3L(cg)^3 + k^4L(cg)^2L(f) + k^5L(f)^2$.

In step (4), we have from [PIN73] that $t_{RNAVER}(r,t) \preceq L(r)L(t) \preceq L(cg)^2$, while $t_{REVAL}(\Psi,v) \preceq k^2L(v)^2 + k^2L(g)L(v) \preceq k^2L(cg)^2$, and therefore $T_4 \preceq k^2L(cg)^3$. $t_{REVAL}(\bar{A}',r)$, $t_{REVAL}(\bar{A}',t) \preceq n^2L(cg)^2 + n^2L(cg)L(f) \preceq k^2L(cg)^2 + k^2L(cg)L(f)$, so that $T_5 \preceq k^2L(cg)^3 + k^2L(cg)^2L(f)$. $T_7 \preceq t_{REVAL}(\bar{A},t) + t_{REVAL}(\bar{A},r) \preceq k^2L(cg)^2 + k^2L(cg)L(h)$.

Thus $t_{ASIGN}(a) \preceq k^3L(h)^2 + k^3L(cg)^3 + k^4L(cg)^2L(f) + k^5L(f)^2 + k^2L(cg)L(h) \preceq k^3\{L(cg)^3+L(h)^2\} + k^4L(cg)^2L(f) + k^5L(f)^2$, since $k^3L(h)^2 + k^3L(cg)^3 \geq k^3\{L(h)^2+L(cg)^2\} \sim k^3\{L(h)+L(cg)\}^2 \geq k^2L(cg)L(h)$. ∎

Theorem 4.2.5. Let a be a non-zero element of $Q(\alpha)[x]$. Let $k = \deg(\Psi)$, $g = |\Psi|_1$, $A = \text{lcf}(\text{polp}(a))$, $h = |A|_1$, $\bar{A} = \text{pp}(A)$, and $f = |\text{gsfd}(\bar{A})|_1$. Let c be the least positive integer such that $c^{-1} \leq \text{sep}(\Psi\bar{A})$. Then $t_{ASIGN}(a) \preceq k^3\{L(cg)^3 + L(h)^2\} + k^4L(cg)^2L(f) + k^5L(f)^2$ if Ω is in the list L = ANALR(Ψ,0).

Proof. By Theorem 2.5.2, $t_{PICPP}(A) \preceq kL(h)^2$. The result now follows from Theorem 4.2.4. ∎

Lemma 4.2.1. Let A be a univariate integral polynomial of degree $n > 0$. Let c be the least positive integer such that $c^{-1} \leq \text{sep}(A)$. Then $L(c) \preceq h^2 + nL(h)$, where $h = |A|_1$.

Proof. Let $B = \text{gsfd}(A)$, $m = \deg(B)$, and $d = |B|_1$. If $c = 1$ then the theorem is clearly true. Hence assume $c > 1$. Since $\text{sep}(A) = \text{sep}(B)$, if $m = 1$ then $c = 1$. Therefore we have $n \geq m > 1$. By Theorem 1.4.5, $\text{sep}(B) > \frac{1}{2}(de^{1/2}\, m^{3/2})^{-m}$, where e is the base of the natural logorithm. Hence $(c-1)^{-1} > \frac{1}{2}(de^{1/2}\, m^{3/2})^{-m}$ and $c - 1 < 2(de^{1/2}\, m^{3/2})^m$. By Corollary 1.4.1, $d \leq (n+1)^{2n}h$, therefore $c - 1 < 2((n+1)^{2n}\, he^{1/2}\, n^{3/2})^n$. This implies that $L(c) \preceq n^2 + nL(h)$, assuming $L(n)$, $L(e) \sim 1$. ∎

Corollary 4.2.1. Let a be an element of $Q(\alpha)$. Let $k = \deg(\Psi)$, $g = |\Psi|_1$, $\bar{A} = \text{polp}(a)$, $h = |\bar{A}|_1$. If $a \neq 0$ and Ω is in the list L = ANALR(Ψ,0), then $t_{ASIGN}(a) \preceq k^9 + k^6L(gh)^3$.

Proof. Let c,f be as defined in Theorem 4.2.4. Let $d = |\Psi\bar{A}|_1$. Then $L(c) \preceq (2k-1)^2 + (2k-1)L(d)$ by Lemma 4.2.1. $d \leq gh$ so $L(c) \preceq k^2 + kL(gh)$. Also, by Corollary 1.4.1, $L(f) \preceq L(k^{2k-2}n) \preceq k + L(h) \preceq k + L(gh)$. $L(cg) \sim L(c) + L(g) \preceq k^2 + kL(gh) + L(g) \sim k^2 + kL(gh)$. Therefore, $k^3\{L(cg)^3 + L(h)^2\} \preceq k^3\{(k^2+kL(gh))^3 + L(h)^2\} \sim k^3\{k^2+kL(gh)\}^3 \sim k^9 + k^6L(gh)^3$ by Theorem 1.4.4.

The second term of the bound for $t_{ASIGN}(a)$ given by Theorem

4.2.4 is $k^4L(cg)^2L(f) \preceq k^4\{k^2+kL(gh)\}^2\{k+L(gh)\} \sim k^6\{k+L(gh)\}^3 \sim k^9 + k^6L(gh)^3$ by Theorem 1.4.4.

Finally, $k^5L(f)^2 \preceq k^5\{k+L(gh)\}^2 \sim k^7 + k^5L(gh)^2$. ∎

Corollary 4.2.2. Let a be a non-zero element of $Q(\alpha)[x]$. Let $k = \deg(\Psi)$, $g = |\Psi|_1$, $\bar{A} = \text{polp}(a)$, $h = \bar{A}_1$. If Ω is in the list $L = \text{ANALR}(\Psi,0)$, then $t_{\text{ASIGN}}(a) \preceq k^9 + k^6L(gh)^3$.

Proof. Let $A = \text{ldcf}(\text{polp}(a))$, $\hat{h} = |A|_1$ and $\bar{h} = |\text{pp}(A)|_1$. By Theorem 2.5.2, $t_{\text{PICPP}}(A) \preceq kL(\hat{h})^2$. Hence, by Corollary 4.2.1, $t_{\text{ASIGN}}(a) \preceq kL(\hat{h})^2 + k^9 + k^6L(g\bar{h})^3$. The theorem now follows from $\bar{h} \leq \hat{h} \leq h$. ∎

Section 4.3. Additive Operations

The first algorithm of this section, ASUM, computes the sum of two elements, a, b, of $Q(\alpha)$ or $Q(\alpha)[x]$. First, PMSUM is used to compute a + b, then AREP is used to determine the representation of the sum considered as an algebraic number or polynomial.

Algorithm 4.3.1. c = ASUM(a,b)

(Algebraic Sum)

a,b are elements of $Q(\alpha)$ or elements of $Q(\alpha)[x]$. c = a + b.

Description

(1) D ← PMSUM(a,b); c ← D; if $a \neq 0$ and $b \neq 0$, (c ← AREP(D); erase D); return.

Theorem 4.3.1. If a = 0 or b = 0 then $t_{\text{ASUM}}(a,b) \sim 1$. Assume $a \neq 0$, $b \neq 0$, and define $k = \deg(\Psi)$, $g = |\Psi|_1$, $d = v(a)$, $f = v(b)$ and $e = \text{mep}(\Omega)$. If a,b are elements of $Q(\alpha)$, then $t_{\text{ASUM}}(a,b) \preceq k^2\{kL(dfg)^2 + kL(e) + L(e)^2\}$. If a,b are elements of $Q(\alpha)[x]$, define ℓ_1 to be the number of non-zero terms of a, ℓ_2 the number of non-zero terms of b, and let $\ell = \max(\ell_1,\ell_2)$. Then $t_{\text{ASUM}}(a,b) \preceq \ell k^2\{kL(dfg)^2 + kL(e) + L(e)^2\}$.

Proof. Assume $a \neq 0$, $b \neq 0$. Let a,b be elements of $Q(\alpha)$. By Theorem 3.2.1, $t_{\text{PMSUM}}(a,b) \preceq k\{L(d)^2+L(f)^2\} \sim kL(df)^2$. By Theorem 4.2.3, $t_{\text{AREP}}(D) \preceq k^2\{kL(gv(D))^2 + kL(e) + L(e)^2\}$. Then by Lemma 3.2.1, $t_{\text{AREP}}(D) \preceq k^2\{kL(dfg)^2 + kL(e) + L(e)^2\}$.

Let a,b be elements of $Q(\alpha)[x]$. By Theorem 3.2.1, $t_{PMSUM}(a,b) \preceq \ell k\{L(d)^2+L(f)^2\} \preceq \ell kL(df)^2$. By Theorem 4.2.3, $t_{AREP}(D) \preceq \ell k^2\{kL(g\nu(D))^2 + kL(e) + L(e)^2\}$. Then by Lemma 3.2.1, $t_{AREP}(D) \preceq \ell k^2\{kL(dfg)^2 + kL(e) + L(e)^2\}$. ∎

Algorithm 4.3.2. b = ANEG(a)

(Algebraic Negation)

a is an element of $Q(\alpha)$ or $Q(\alpha)[x]$. b = -a.

Description

(1) b ← PMNEG(a); return.

Theorem 4.3.2. If a = 0, then $t_{ANEG}(a) \sim 1$. Assume $a \neq 0$. Define $k = deg(\Psi)$, $d = \nu(a)$. If a is an element of $Q(\alpha)$ then $t_{ANEG}(a) \preceq kL(d)$. If a is an element of $Q(\alpha)[x]$ and ℓ is the number of terms of a, then $t_{ANEG}(a) \preceq \ell kL(d)$.

Proof. This follows directly from Theorem 3.2.2. ∎

Algorithm 4.3.3. c = ADIF(a,b)

(Algebraic Difference)

a,b are elements of $Q(\alpha)$ or elements of $Q(\alpha)[x]$. c = a - b.

Description

(1) D ← PMDIF(a,b); c ← D; if $a \neq 0$ and $b \neq 0$, (c ← AREP(D); erase D); return.

Theorem 4.3.3. Let a,b be elements of $Q(\alpha)$. If b = 0 then $t_{ADIF}(a,b) \sim 1$. Assume $b \neq 0$ and define $k = deg(\Psi)$, $f = \nu(b)$. If a = 0 then $t_{ADIF}(a,b) \preceq kL(f)$. Suppose $a \neq 0$ and define $g = |\Psi|_1$, $d = \nu(a)$ and $e = mep(\Omega)$. $t_{ADIF}(a,b) \preceq k^2\{kL(dfg)^2 + kL(e) + L(e)^2\}$.

Proof. Assume $b \neq 0$. If a = 0 then the result follows directly from Theorem 3.2.3. Assume $a \neq 0$. By Theorem 3.2.3, $t_{PMDIF}(a,b) \preceq k\{L(d)^2 + L(f)^2\} \preceq kL(df)^2$. By applying Theorem 4.2.3 and then Lemma 3.2.1, we have that $t_{AREP}(D) \preceq k^2\{kL(g\nu(D))^2 + kL(e) + L(e)^2\} \preceq k^2\{kL(dfg)^2 + kL(e) + L(e)^2\}$. ∎

Theorem 4.3.4. Let a,b be elements of $Q(\alpha)[x]$. If b = 0 then $t_{ADIF}(a,b) \sim 1$. Assume $b \neq 0$ and define $k = deg(\Psi)$, $f = \nu(b)$. Let ℓ_2 be the number of non-zero terms of b. If a = 0 then $t_{ADIF}(a,b) \preceq \ell_2 kL(f)$. Suppose $a \neq 0$ and define $g = |\Psi|_1$, $d = \nu(a)$ and $e = mep(\Omega)$. Let ℓ_1 be the number of non-zero terms of a, and let $\ell = \max(\ell_1,\ell_2)$. $t_{ADIF}(a,b) \preceq \ell k^2\{kL(dfg)^2 + kL(e) + L(e)^2\}$.

Proof. Assume $b \neq 0$. If a = 0 then the result follows directly from Theorem 3.2.3. Assume $a \neq 0$. By Theorem 3.2.3, $t_{PMDIF}(a,b) \preceq \ell k\{L(d)^2 + L(f)^2\} \preceq \ell kL(df)^2$. By applying Theorem 4.2.3 and then Lemma 3.2.1, we have that $t_{AREP}(D) \preceq \ell k^2\{kL(g\nu(D))^2 + kL(e) + L(e)^2\} \preceq \ell k^2\{kL(dfg)^2 + kL(e) + L(e)^2\}$. ∎

Algorithm 4.3.4. b = AABS(a)

(Algebraic Absolute Value)

a is an element of $Q(\alpha)$ or $Q(\alpha)[x]$. $b = |a|$.

Description

(1) s ← ASIGN(a); if s ≥ 0, (b ← BORROW(a); return); b ← PMNEG(a); return.

Theorem 4.3.5. If $a = 0$ then $t_{AABS}(a) \sim 1$. Assume $a \neq 0$ and define $k = \deg(\Psi)$, $g = |\Psi|_1$, $d = \nu(a)$. If a is an element of $Q(\alpha)$ then $t_{AABS}(a) \preceq k^9 + k^6L(gd)^3$. If a is an element of $Q(\alpha)[x]$ and ℓ is the number of terms of a, then $t_{AABS}(a) \preceq k^9 + k^6L(gd)^3 + \ell kL(d)$.

Proof. Assume $a \neq 0$. By Corollary 4.2.1 and Corollary 4.2.2, $t_{ASIGN}(a) \preceq k^9 + k^6L(gd)^3$. By Theorem 3.2.2, $t_{ANEG}(a) \preceq kL(d)$ if a is an element of $Q(\alpha)$, and $t_{ANEG}(a) \preceq \ell kL(d)$ if a is an element of $Q(\alpha)[x]$. ∎

Section 4.4. Multiplicative Operations

Suppose a,b are elements of $Q(\alpha)$, that is, elements of $Q[y]/(\Psi(y))$ representing elements of $Q(\alpha)$. If $\psi_\alpha(a) \neq 0$ and $\psi_\alpha(b) \neq 0$ then $\psi_\alpha(ab) = \psi_\alpha(a)\,\psi_\alpha(b) \neq 0$. Now suppose a,b are elements of $Q(\alpha)[x]$, that is, certain elements of $Q[y,x]/(\Psi(y))$. Let $c(x) = a(x)b(x) = \sum_{i=0}^{n} c_i(y)x^i$. Clearly, for some i such that $c_i(y) \neq 0$, it may be the case that $\psi_\alpha(c_i(y)) = c_i(\alpha) = 0$ if Ψ is not irreducible. Consequently, in the algorithm APROD which follows, AREP must be utilized if the inputs belong to $Q(\alpha)[x]$.

Algorithm 4.4.1. c = APROD(a,b)

(Algebraic Product)

a,b are elements of $Q(\alpha)$ or elements of $Q(\alpha)[x]$. $c = a \cdot b$.

Description

(1) C ← PMPROD(a,b); c ← C; if C ≠ 0, if TYPE(SECOND(SECOND(C))) = 1, (c ← AREP(C); erase C); return.

Theorem 4.4.1. If $a = 0$ or $b = 0$ then $t_{APROD}(a,b) \sim 1$. Assume $a \neq 0$, $b \neq 0$ and define $k = \deg(\Psi)$, $m_1 = \partial_1(a)$, $n_1 = \partial_1(b)$, $d = \nu(a)$, $f = \nu(b)$. Suppose a,b are elements of $Q(\alpha)$. If $m_1 + n_1 < k$ then $t_{APROD}(a,b) \preceq (m_1+1)(n_1+1)L(f)L(t)$. If $m_1 + n_1 \geq k$ and $g = |\Psi|_1$, then $t_{APROD}(a,b) \preceq k^2\{(m_1+n_1-k+1)L(g)^2+L(df)^2\}$. Suppose a,b are elements of $Q(\alpha)[x]$. Let ℓ be the number of non-zero terms of a and define h similarly for b. If $e = \mathrm{mep}(\Omega)$ then $t_{APROD}(a,b) \preceq \ell hk^2\{k^3L(g)^2+kL(df)^2+L(e)^2\}$.

Proof. Suppose a,b are non-zero. If a,b are elements of $Q(\alpha)$ then the theorem follows directly from Theorem 3.3.4. Assume a,b are elements of $Q(\alpha)[x]$. By Theorem 3.3.4, $t_{PMPROD}(a,b) \preceq \ell h(m_1+n_1)^2\{(m_1+n_1-k+1)L(g)^2+L(d)^2+L(f)^2\} \preceq \ell hk^2\{kL(g)^2+L(df)^2\}$. Next, observe that by Lemma 3.3.1, $L(g\nu(C)) \leq L(kg^k df) \sim kL(g)+L(df)$. Note that we assume $L(k) \sim 1$. Using this result and Theorem 4.2.3 we obtain $t_{AREP}(C) \preceq \ell hk^2\{kL(g\nu(C))^2+kL(e)+L(e)^2\} \preceq \ell hk^2\{k\{kL(g)+L(df)\}^2 + kL(e)+L(e)^2\} \preceq \ell hk^2\{k^3L(g)^2+kL(df)^2+L(e)^2\}$. ∎

Algorithm 4.4.2. C = ASPROD(A,b,n)

(Algebraic Special Product)

A is an element of $Q(\alpha)[x]$, b is an element of $Q(\alpha)$, and n is a non-negative FORTRAN integer. $C = A \cdot bx^n$.

Description

(1) C ← PMSPROD(A,b,n); return.

Theorem 4.4.2. Let A be an element of $Q(\alpha)[x]$, b be an element of $Q(\alpha)$, and n be a non-negative integer. A = 0 or b = 0 implies that $t_{ASPROD}(A,b,n) \sim 1$. Assume $A \neq 0$, $b \neq 0$, and define $k = \deg(\Psi)$, $\partial_1(A) = m_1$, $\deg(b) = n_1$, $d = \nu(A)$, $f = \nu(b)$, $g = |\Psi|_\infty$. Let ℓ be the number of non-zero terms of A. If $m_1 + n_1 < k$ then $t_{ASPROD}(A,b,n) \preceq \ell(m_1+1)(n_1+1)L(d)L(f)$. If $m_1 + n_1 \geq k$ then $t_{ASPROD}(A,b,n) \preceq \ell k^2\{(m_1+n_1-k+1)L(g)^2+L(df)^2\}$.

Proof. This follows directly from Theorem 3.3.5. ∎

Section 4.5. Division Operations

First we discuss the computation of a^{-1}, where a is a non-zero element of $Q(\alpha)$.

Theorem 4.5.1. Let a be a non-zero element of $Q(\alpha)$ and define $\bar{A} = \text{polp}(a)$, $B = \gcd(\bar{A},\Psi)$, $\bar{\Psi} = \Psi/B$. Then $\bar{\Psi}$ is square-free and primitive, $\bar{\Psi}(\alpha) = 0$, and $\gcd(\bar{\Psi},\bar{A}) = 1$.

Proof. $\bar{\Psi} | \Psi$, so $\bar{\Psi}$ is square-free and primitive because Ψ is square-free and primitive. $B|\bar{A}$ and $\bar{A}(\alpha) \neq 0$, hence $B(\alpha) \neq 0$. Since $\Psi = B\bar{\Psi}$, this implies that $\bar{\Psi}(\alpha) = 0$. Suppose D is an integral polynomial such that $D|\bar{A}$ and $D|\bar{\Psi}$. Since $\bar{\Psi}|\Psi$, this implies that $D|\gcd(\bar{A},\Psi) = B$. Then $D^2|\Psi$ since $\Psi = B\bar{\Psi}$. But Ψ is square-free implies that $\deg(D) = 0$. Ψ is primitive implies that $D = \pm 1$. ∎

Now, since $\gcd(\bar{\Psi},\bar{A}) = 1$, if one applies the extended Euclidean algorithm to $\bar{\Psi}$ and $\bar{A}$, one obtains S and T, polynomials over the rationals such that $\bar{\Psi}S + \bar{A}T = 1$. S and T have the properties that $\deg(S) < \deg(\bar{A}) < \deg(\Psi)$ and $\deg(T) < \deg(\bar{\Psi}) \leq \deg(\Psi)$. Since $\bar{\Psi}(\alpha) = 0$, we have that $\bar{A}(\alpha)T(\alpha) = 1$, $A(\alpha)(1/r)T(\alpha) = 1$, and $(1/r)T(\alpha) = A(\alpha)^{-1} = a^{-1}$, if a is represented by $A = (r,\bar{A})$, an element of $Q[y]/(\Psi(y))$.

Instead of proceeding to compute T(y) in a direct manner, it is advantageous in terms of computing efficiency to take a more circuitous route and to instead compute $c = \text{res}(\bar{\Psi},\bar{A})$ and the polynomial cT. Then we use the fact that $\bar{\Psi}(cS) + \bar{A}(cT) = c$, where cS and

cT are integral polynomials. While the former method is conceptually much simpler, the latter approach affords us the opportunity to apply modular algorithms and the theory of polynomial remainder sequences. For explanations of these terms and a list of references, see Section 1.4. Note also that this approach is somewhat similar to that taken in [HEI70] for the computation of Sturm sequences.

Basically, our method consists of the following. For certain primes p, we construct a p.r.s. and a cosequence over $GF(p)$. We apply Theorem 1.4.2 to obtain a resultant. Also, we show that these resultants and copolynomials obtained from the cosequences are the modular images of c and cT. We then apply Theorem 1.4.7 to compute c and cT. The next several pages are devoted to a rigorous development and analysis of our approach, culminating with the algorithm ANRECP.

Definition 4.5.1. Let A_1, A_2 be non-zero polynomials over an integral domain such that $\deg(A_1)$, $\deg(A_2) > 0$. Let $A_1, A_2, \ldots, A_{k+2} = 0$ be the polynomial remainder sequence defined by $e_i A_i = Q_i A_{i+1} + f_i A_{i+2}$ for $1 \le i \le k$. Let S be the sequence $S_1, S_2, \ldots, S_{k+2}$ defined by $S_1 = 1$, $S_2 = 0$, $f_i S_{i+2} = e_i S_i - Q_i S_{i+1}$ for $1 \le i \le k$; let T be the sequence $T_1, T_2, \ldots, T_{k+2}$ defined by $T_1 = 0$, $T_2 = 1$, $f_i T_{i+2} = e_i T_i - Q_i T_{i+1}$ for $1 \le i \le k$. We call S and T the first and second cosequences of $A_1, A_2, \ldots, A_{k+2}$, respectively.

Theorem 4.5.2. Let A_i, S_i, T_i, for $1 \le i \le k+2$ be as defined in Definition 4.5.1. Then $A_1 S_i + A_2 T_i = A_i$ for $1 \le i \le k+2$.

Proof. The proof is by induction on i. The proof is obvious for $i = 1,2$. Suppose it is true for all $j < i$ where $i > 2$. By the definitions of S_i and T_i, $f_{i-2}\{A_1 S_i + A_2 T_i\} = A_1\{e_{i-2} S_{i-2} - Q_{i-2} S_{i-1}\} + A_2\{e_{i-2} T_{i-2} - Q_{i-2} T_{i-1}\} = e_{i-2}\{A_1 S_{i-2} + A_2 T_{i-2}\} - Q_{i-2}\{A_1 S_{i-1} + A_2 T_{i-1}\}$. Then by the induction hypothesis, $f_{i-2}\{A_1 S_i + A_2 T_i\} = e_{i-2} A_{i-2} - Q_{i-2} A_{i-1}$. From the definition of A_i, we have $f_{i-2}\{A_1 S_i + A_2 T_i\} = f_{i-2} A_i$. ∎

Definition 4.5.2. Let A_1, A_2 be non-zero polynomials over an integral domain such that $\deg(A_1)$, $\deg(A_2) > 0$. Let $A_1, A_2, \ldots, A_{k+2} = 0$ be the subresultant p.r.s. of (A_1, A_2) of the second kind. Let $S = (S_1, S_2, \ldots, S_{k+2})$ and $T = (T_1, T_2, \ldots, T_{k+2})$ be the first and second cosequences of $A_1, A_2, \ldots, A_{k+2}$. We call S_{k+1} and T_{k+1} the first and second subresultant copolynomials of (A_1, A_2) of the second kind, respectively.

Corollary 4.5.1. Let A_1, A_2 be polynomials over an integral domain such that $\deg(A_1)$, $\deg(A_2) > 0$, $\mathrm{res}(A_1, A_2) \neq 0$. Let $A_1, A_2, \ldots, A_{k+2} = 0$ be the subresultant p.r.s. of (A_1, A_2) of the second kind, and let S and T be the first and second subresultant copolynomials of (A_1, A_2) of the second kind. Then $A_1 S + A_2 T = \mathrm{res}(A_1, A_2)$.

Proof. By Theorem 1.4.1, $\mathrm{res}(A_1, A_2) \neq 0$ implies that $\deg(\gcd(A_1, A_2)) = 0$, and hence $\deg(A_{k+1}) = 0$. Therefore $\mathrm{res}(A_1, A_2) = S_0(A_1, A_2) = A_{k+1} = A_1 S_{k+1} + A_2 T_{k+1}$, by Theorem 4.5.2. By definition, $S_{k+1} = S$,

$T_{k+1} = T$. ∎

Theorem 4.5.3. Let A_i, S_i, T_i be as defined in Definition 4.5.1, $n_i = \deg(A_i)$ for $1 \le i \le k+2$. If $n_1 \ge n_2$, then $\deg(S_i) = n_2 - n_{i-1}$ and $\deg(T_i) = n_1 - n_{i-1}$, for $i \ge 3$. If $n_1 < n_2$, then $\deg(S_i) = n_2 - n_{i-1}$ and $\deg(T_i) = n_1 - n_{i-1}$ for $i \ge 4$.

Proof. Assume $n_1 \ge n_2$. The proof is by induction on i. $f_1S_3 = e_1$, so $\deg(S_3) = n_2 - n_2 = 0$. $f_1T_3 = -Q_1$, which implies that $\deg(T_3) = n_1 - n_2$. $f_2S_4 = -Q_2S_3$, so $\deg(S_4) = n_2 - n_3$. $A_1S_4 + A_2T_4 = A_4$. But $\deg(A_4) = n_4 < n_1 + n_2 - n_3 = \deg(A_1S_4)$, so $\deg(A_2T_4) = n_1 + n_2 - n_3$ which implies that $\deg(T_4) = n_1 - n_3$.

Suppose the theorem is true for $i-2$, $i-1$. $f_{i-2}S_i = e_{i-2}S_{i-2} - Q_{i-2}S_{i-1}$. $\deg(e_{i-2}S_{i-2}) = n_2 - n_{i-3}$, $\deg(Q_{i-2}S_{i-1}) = n_2 - n_{i-1}$ by assumption. $n_2 - n_{i-1} > n_2 - n_{i-3}$, so $\deg(f_{i-2}S_i) = \deg(S_i) = n_2 - n_{i-1}$. Since $A_1S_i + A_2T_i = A_i$ and $n_i = \deg(A_i) < n_1 + n_2 - n_{i-1} = \deg(A_1S_i)$, we have that $\deg(A_2T_i) = n_1 + n_2 - n_{i-1}$. This implies that $\deg(T_i) = n_1 - n_{i-1}$, completing the induction.

Now assume $n_1 < n_2$. Then $Q_1 = 0$, $A_3 = e_1A_1/f_1$, $S_3 = e_1/f_1$ and $T_3 = 0$. It follows that $(T_2,T_3,\ldots,T_{k+2})$ and $(f_1S_2/e_1, f_1S_3/e_1,\ldots,f_1S_{k+2}/e_1)$ are the first and second cosequences of $(A_2,A_3,\ldots,A_{k+2})$. The theorem for this case now follows from the previous case. ∎

The next theorem shows how to apply Corollary 4.5.1 to determine a solution to our problem.

Theorem 4.5.4. Let $a \in Q(\alpha)$ such that $a \ne 0$, $r = rp(a)$, $\bar{A} = polp(a)$, $\deg(\bar{A}) > 0$, $\bar{\Psi} = \Psi/gcd(\bar{A},\Psi)$. Define S and T to be the first and second subresultant copolynomials of $(\bar{\Psi},\bar{A})$ of the second kind and define $c = res(\bar{\Psi},\bar{A})$. Then $a^{-1} = (rc)^{-1}\,T(\alpha)$.

Proof. By Theorem 4.5.1, $gcd(\bar{A},\bar{\Psi}) = 1$, so by Theorem 1.4.1 $c \ne 0$. Therefore, by Corollary 4.5.1, $\bar{\Psi}S + \bar{A}T = c$. By Theorem 4.5.1, $\bar{\Psi}(\alpha) = 0$ and hence $\bar{A}(\alpha)T(\alpha) = c$. Since $A(\alpha) = r\bar{A}(\alpha)$ we obtain $A(\alpha)\,T(\alpha) = rc$ and $a^{-1} = A(\alpha)^{-1} = (rc)^{-1}\,T(\alpha)$. ∎

Observe that $\deg(T) = n_1 - n_k = \deg(\bar{\Psi}) - n_k < \deg(\bar{\Psi}) \le \deg(\Psi)$ by Theorem 4.5.3. Let $d = cont(T)$, $\bar{B} = sign(c)T(y)/d$, $s = \{d/r|c|\}$. If $b = a^{-1}$, then the representation of b is the list $(s,\bar{B})$.

In light of this result, the problem of determining a^{-1} becomes one of constructing c and T. To this end we define the mapping $\phi_p: I \to GF(p)$, where p is some prime, I is the ring of integers, and $\phi_p(a) = a$ modulo p, $a \in I$. This mapping is a homomorphism and induces a homomorphism, also called ϕ_p, on the ring $I[x]$; namely, if $A(x) = \sum_{i=0}^{n} a_i x^i$ then $\phi_p(A(x)) = \sum_{i=0}^{n} \phi_p(a_i)x^i$. We abbreviate $\phi_p(a)$ as a^* and $\phi_p(A)$ as A^*.

The homomorphisms ϕ_p, for any odd prime p, are used in the modular algorithm we employ to obtain c and T. This algorithm, ANRECP, determines the number of homomorphic images of c and T it must compute, computes these images, and applies the Chinese Remainder Theorem, Theorem 1.4.7, to compute the elements from their images. (For discussions of the Chinese Remainder Theorem and

modular algorithms, see [BRO71], [COL71e], and [KNU69].)

Our first task, therefore, is to generate, for an arbitrary prime p, $\phi_p(c)$ and $\phi_p(T)$. The next theorem solves part of the task, showing how to obtain $\phi_p(c)$. It is stated without proof and is a special case of Theorem 4 of [COL71e].

Theorem 4.5.5. Let A and B be polynomials of positive degree belonging to $I[x]$, let p be a prime number. If $\deg(A) = \deg(\phi_p(A))$ and $\deg(B) = \deg(\phi_p(B))$ then $\phi_p(\mathrm{res}(A,B)) = \mathrm{res}(\phi_p(A), \phi_p(B))$.

To obtain an expression for $\phi_p(T)$ we must first obtain some intermediate results.

Lemma 4.5.1. Let A and B be polynomials of positive degree over an integral domain I such that $c = \mathrm{res}(A,B) \neq 0$. Let S and T be the first and second subresultant copolynomials of (A,B) of the second kind. Let U and V be any polynomials over I such that $\deg(U) < \deg(B)$, $\deg(V) < \deg(A)$, and $AU + BV = c$. Then $S = U$ and $T = V$.

Proof. $c \neq 0$, so by Corollary 4.5.1, $AS + BT = AU + BV$, so $A(S-U) = B(V-T)$. By assumption $c \neq 0$, so by Theorem 1.4.1 A and B do not have a common factor of positive degree. Thus if $\bar{A} = pp(A)$, then $\bar{A} \mid (V-T)$. But $\deg(V) < \deg(\bar{A})$ by assumption, and $\deg(T) < \deg(\bar{A})$ by Theorem 4.5.3, so $\deg(V-T) < \deg(\bar{A})$. This implies that $V = T$. $AS + BT = AU + BT$, so $S = U$. ∎

Lemma 4.5.2. Let A and B be integral polynomials of positive degree such that $\mathrm{res}(A,B) \neq 0$. Let p be a prime number such that $p \nmid \mathrm{res}(A,B)$, $p \nmid \mathrm{ldcf}(A)$, and $p \nmid \mathrm{ldcf}(B)$. Define S and T to be the first and second subresultant copolynomials of (A,B) of the second kind, and define U and V similarly for A^* and B^*. Then $\phi_p(S) = U$ and $\phi_p(T) = V$.

Proof. Let $c = \mathrm{res}(A,B)$. Since $c \neq 0$, $AS + BT = c$ by Corollary 4.5.1. $\phi_p(AS + BT) = A^*S^* + B^*T^* = c^*$. $p \nmid \mathrm{ldcf}(A)$ and $p \nmid \mathrm{ldcf}(B)$ so $\mathrm{res}(A^*,B^*) = \phi_p(\mathrm{res}(A,B)) = c^*$ by Theorem 4.5.5. $p \nmid c$ so $c^* \neq 0$ and therefore by Corollary 4.5.1 $A^*U + B^*V = c^*$. $\deg(S^*) < \deg(B^*)$, $\deg(T^*) < \deg(A^*)$, so $U = S^*$, $V = T^*$ by Lemma 4.5.1. ∎

Definition 4.5.3. Let A_1, A_2 be non-zero polynomials over an integral domain such that $\deg(A_1)$, $\deg(A_2) > 0$. Let $A_1, A_2, \ldots, A_{k+2} = 0$ be the natural p.r.s. of (A_1, A_2). Let $S = (S_1, \ldots, S_{k+2})$ and $T = (T_1, \ldots, T_{k+2})$ be the first and second cosequences of $A_1, A_2, \ldots, A_{k+2}$. We call S_{k+1} and T_{k+1} the first and second natural copolynomials of (A_1, A_2), respectively.

Lemma 4.5.3. Let A and B be polynomials over an integral domain I such that $\deg(A)$, $\deg(B) > 0$, and $\mathrm{res}(A,B) \neq 0$. Let $A_1 = A$, $A_2 = B, \ldots, A_{k+2} = 0$ be the natural polynomial remainder sequence of A and B over the fraction field of I. Let U and V be the first and second subresultant copolynomials of (A,B) of the second kind. Let W and X be the first and second natural copolynomials of A and B. Define $n_i = \deg(A_i)$, $a_i = \mathrm{ldcf}(A_i)$

for $1 \le i \le k+1$, and define

$$a = (-1)^{\sum_{i=1}^{k-1} n_i n_{i+1}} \{\prod_{i=1}^{k-1} a_{i+1}^{n_i - n_{i+2}}\} a_{k+1}^{n_k - 1} .$$

Then $\mathrm{res}(A,B) = aa_{k+1}$, $U = aW$, and $V = aX$.

Proof. By Theorem 1.4.1, $\mathrm{res}(A,B) \neq 0$ implies that $\deg(\gcd(A,B)) = 0$. Since $A_{k+1} \approx \gcd(A,B)$ it follows that $\deg(A_{k+1}) = n_{k+1} = 0$. Then by substituting $k+1$ for j in (1) of Theorem 1.4.2, and noting that $n_{k+1} = 0$, $\delta_k = n_k$, and $e_i = f_i = 1$, $\delta_i + \delta_{i+1} = n_i - n_{i+2}$ for $1 \le i \le k-1$, we have that $S_0(A,B) = aA_{k+1} = aa_{k+1}$. (Although Theorem 1.4.2 is stated only for $\deg(A) \ge \deg(B)$, one can readily verify that its truth for this case implies its truth also for $\deg(A) < \deg(B)$ when j is at least 4, as it is in this application.) $S_0(A,B) = \mathrm{res}(A,B)$. Also, by Theorem 4.5.2 $A_{k+1} = a_{k+1} = AW + BX$, so $\mathrm{res}(A,B) = aa_{k+1} = A(aW) + B(aX)$. By Theorem 4.5.3, $\deg(aW) < \deg(B)$, $\deg(aX) < \deg(A)$. Hence $U = aW$ and $V = aX$ by Lemma 4.5.1. ∎

Theorem 4.5.6. Let A, B be integral polynomials such that $\deg(A)$, $\deg(B) > 0$ and $\mathrm{res}(A,B) \neq 0$. Let p be a prime number such that $p \nmid \mathrm{res}(A,B)$, $p \nmid \mathrm{ldcf}(A)$, and $p \nmid \mathrm{ldcf}(B)$. Define a as in Lemma 4.5.3 for the natural p.r.s. of A^* and B^*. Let S and T be the first and second subresultant copolynomials of (A,B) of the second kind, and let W and X be the first and second natural copolynomials of A^* and B^*. Then $S^* = aW$ and $T^* = aX$.

Proof. Let U and V be the first and second subresultant copolynomials of (A^*,B^*) of the second kind. By Lemma 4.5.2, $S^* = U$ and $T^* = V$. By Lemma 4.5.3, $U = aW$ and $V = aX$. ∎

Theorems 4.5.5 and 4.5.6 give us the tools necessary to devise an efficient algorithm for computing $\phi_p(c)$ and $\phi_p(T)$, where c is the desired resultant, and T the second subresultant copolynomial of $(\bar{\Psi},\bar{A})$ of the second kind. Algorithm CRESCP has as inputs a prime p, and univariate polynomials of positive degree over GF(p), A and B. Let $A_1 = A$, $A_2 = B$. The algorithm computes the natural p.r.s. of (A_1,A_2) using the recursion formula $A_i = Q_i A_{i+1} - A_{i+2}$; computes the second natural cosequence of (A_1,A_2) using the recursion formula $X_{i+2} = X_i - Q_i X_{i+1}$, with initial values $X_1 = 0$, $X_2 = 1$; and computes a where a is as described in Lemma 4.5.3. If it happens that A_{k+1}, and hence $\gcd(A_1,A_2)$, has positive degree then the outputs, c and T, are given in value -1 and the algorithm terminates. On the other hand, if $\deg(A_{k+1}) = 0$, then $c = aa_{k+1}$, $T = aX_{k+1}$. By Theorem 1.4.1 $\mathrm{res}(A_1,A_2) \neq 0$, so Lemma 4.5.3 applies and we see that $c = \mathrm{res}(A_1,A_2)$ and T is the second subresultant copolynomial of (A,B) of the second kind.

Suppose $A = \phi_p(\bar{\Psi})$ and $B = \phi_p(\bar{A})$, where p is a prime number such that $p \nmid \mathrm{ldcf}(\bar{\Psi})$ and $p \nmid \mathrm{ldcf}(\bar{A})$. If it happens that $\gcd(A,B)$ has positive degree, then $\mathrm{res}(A,B) = 0$ by Theorem 1.4.1. Then by Theorem 4.5.5 $p \mid \mathrm{res}(\bar{\Psi},\bar{A})$, so Theorem 4.5.6 does not apply. In this case the outputs of CRESCP are -1 and we know p is a prime to be rejected. Otherwise the outputs are the homomorphic images we desire,

since Theorem 4.5.6 does apply.

Algorithm 4.5.1. CRESCP(p,A,B,c,T)

(Congruence Algorithm, Resultant and Copolynomial)

The inputs are p, a prime number, and A, B, univariate polynomials over GF(p) such that deg(A), deg(B) > 0. The outputs c and T are both -1 in case res(A,B) = 0. If res(A,B) ≠ 0 then T is the second subresultant copolynomial of (A,B) of the second kind and c = res(A,B).

Description

(1) [Initialize.] $A_1 \leftarrow$ BORROW(A); $A_2 \leftarrow$ BORROW(B); $\nu \leftarrow 0$; $a \leftarrow 1$; $X_1 \leftarrow 0$; $X_2 \leftarrow$ PFA (0,PFA(1,0)); $n_1 \leftarrow$ CPDEG(A_1); $n_2 \leftarrow$ CPDEG(A_2).

(2) [Apply the recursion formula $A_i = Q_i A_{i+1} + A_{i+2}$ and determine if res(A,B) = 0.] If $n_1 < n_2$, ($Q \leftarrow 0$; $A_3 \leftarrow$ BORROW(A_1)); If $n_1 \geq n_2$, (L ← CPQREM(p,A_1,A_2); DECAP2(Q,A_3,L)); if $A_3 = 0$, (erase A_1,A_2,Q,X_1,X_2; c ← T ← -1; return); $n_3 \leftarrow$ CPDEG(A_3); $a_2 \leftarrow$ CPLNCF(A_2).

(3) [Apply the recursion formula $X_{i-2} = X_i - Q_i X_{i+1}$.] U ← CPPROD (p,Q,$X_2$); $X_3 \leftarrow$ CPDIF(p,X_1,U); erase U, X_1, Q; $X_1 \leftarrow X_2$; $X_2 \leftarrow X_3$.

(4) [Compute $\nu = \sum_{i=1}^{k-1} n_i n_{i+1}$.] $\nu \leftarrow \nu + n_1 n_2$.

(5) [Compute $\prod_{i=1}^{k-1} a_{i+1}^{n_i - n_{i+2}}$.] $t \leftarrow$ CPOWER(p, a_2, $n_1 - n_3$); $a \leftarrow$ CPROD (p,a,t).

(6) [Determine if A_3 is res(A,B).] Erase A_1; if $n_3 = 0$, go to (7); $A_1 \leftarrow A_2$; $A_2 \leftarrow A_3$; $n_1 \leftarrow n_2$; $n_2 \leftarrow n_3$; go to (2).

(7) [Finish computing a.] $a_3 \leftarrow$ CPLNCF(A_3); erase A_2, A_3, X_1; t ← CPOWER(p,a_3,n_2-1); a ← CPROD(p,a,t); s ← ν mod 2; if s = 1, a ← p - a.

(8) [Compute c and T from a, a_{k+1},X_{k+1}] T ← CSPROD(p,X_2,a,0); erase X_2; c ← CPROD(p,a,a_3); return.

Theorem 4.5.7. Let p be a prime number and A, B be polynomials of positive degree over GF(p). If m = deg(A), n = deg(B) then $t^+_{CRESCP}(p,m,n) \sim mn$ where t^+_{CRESCP} is the maximum computing time of CRESCP.

Proof. Without loss of generality, assume $m \geq n$. Let A_i, X_i, Q_i, n_i, k be as defined above. We have that $\deg(Q_i) = n_i - n_{i+1}$ for $1 \leq i \leq k$, and by Theorem 4.5.3, $\deg(X_i) = n_1 - n_{i-1}$ for $3 \leq i \leq k+2$. We use these facts to compute the following table, using the bounds given in [COAL69].

step	# of executions		time for i^{th} execution
(1)	1	~	1
(2)	k or k-1	~	$(n_{i+1}+1)(n_i-n_{i+1}+1)$
(3)	k-1	~	$(n_i-n_{i+1}+1)(n_1-n_i+1)+(n_1-n_{i+1}+1)$
(4)	k-1	~	1
(5)	k-1	≤	$L(n_i-n_{i+2})$
(6)	k-1	~	1
(7)	at most 1	≤	$L(n_k)$
(8)	at most 1	~	n_1-n_k

If $1 \le i \le k$, then $n_1 - n_{i+1} + 1 \ge n_1 - n_i + 1$ since $n_i \ge n_{i+1}$; therefore the time for the i^{th} execution of step (3) is $\preceq (n_i - n_{i+1} + 1)(n_1 - n_{i+1} + 1)$

Therefore $T_2 + T_3 \preceq \sum_{i=1}^{k} \{(n_{i+1}+1)(n_i-n_{i+1}+1) + (n_i-n_{i+1}+1)(n_1-n_i+1) + (n_1-n_{i+1}+1)\} \preceq (n_2+1)(n_1-n_2+1) + (n_1-n_2+1) + \sum_{i=2}^{k} \{(n_{i+1}+1)(n_i-n_{i+1}+1) + (n_i-n_{i+1}+1)(n_1-n_{i+1}+1)\} \preceq n_1n_2 + \sum_{i=2}^{k} (n_i-n_{i+1}+1)(n_1+1) \sim n_1n_2 + n_1 \sum_{i=2}^{k} (n_i-n_{i+1}) = n_1n_2 + n_1(n_2-n_{k+1}) \sim n_1n_2$. $T_5 \preceq \sum_{i=1}^{k-1} L(n_i-n_{i+2}) \preceq \sum_{i=1}^{k} (n_i-n_{i+2}) \le n_1 + n_2$. Then by observing that $k \le n_2 + 1$ we see that $t^{+}_{CRESCP}(p,m,n) \preceq mn$.

Let A, B be such that their degree sequence is normal and they are relatively prime. Then $k = n_2 + 1$, $n_{i+1} = n_i - 1$ for $2 \le i \le n_2 + 1$. This implies that $n_i = n_2 + 2 - i$ for $2 \le i \le n_2 + 2$. It can be shown that for given $n_1 \ge n_2 > 0$ such a pair can always be constructed; for example, if $A_{n_2+2} = 1$, $A_{n_2+1} = x$, $Q_i = x$ for $2 \le i \le n_2$, $Q_1 = x^{n_1-n_2}$, then one can construct A_1, A_2 using $A_i = A_{i+1}Q_i + A_{i+2}$, $1 \le i \le n_2$.

Then we have $t^{+}_{CRESCP}(p,m,n) \succeq T_2 \succeq \sum_{i=1}^{n_2} (n_{i+1}+1)(n_i-n_{i+1}+1) = (n_2+1)(n_1-n_2+1) + \sum_{i=2}^{n_2} (n_{i+1}+1)(n_i-n_{i+1}+1) = (n_2+1)(n_1-n_2+1) + 2\sum_{i=2}^{n_2} (n_2+2-i) = (n_2+1)(n_1-n_2+1) + 2\sum_{i=2}^{n_2} i = (n_2+1)(n_1-n_2+1) + n_2(n_2+1) - 2 = (n_2+1)(n_1+1) - 2 > n_1n_2 = mn$, since $n_1 + n_2 + 1 \ge 2$. ∎

Having determined how to obtain c^* and T^* for any prime p, the question of how many primes are needed by the modular algorithm remains. This number is determined by an upper bound on the absolute value of c and the absolute value of the coefficients of T. These upper bounds are provided by the following theorem.

Theorem 4.5.8. Let $A = \sum_{i=0}^{m} a_i x^i$ and $B = \sum_{j=0}^{n} b_j x^j$ be integral polynomials such that $\deg(A) = m > 0$, $\deg(B) = n > 0$, $c = \text{res}(A,B) \neq 0$. Let T be the second subresultant copolynomial of (A,B) of the second kind. Then

$$|T|_\infty \le |A|_1^n \, |B|_1^{m-1}$$

$$|c| \le |A|_1^n \, |B|_1^m .$$

Proof. Consider the $m+n$ by $m+n$ matrix M defined by

$$\begin{bmatrix}
a_m & a_{m-1} & \cdots & & a_{-n+2} & x^{n-1}A(x) \\
0 & a_m & a_{m-1} & \cdots & a_{-n+3} & x^{n-2}A(x) \\
\vdots & & & & & \\
0 & \cdots & 0 & a_m \cdots & a_1 & A(x) \\
b_n & b_{n-1} & \cdots & & b_{-m+2} & x^{m-1}B(x) \\
0 & b_n & b_{n-1} & \cdots & b_{-m+3} & x^{m-2}B(x) \\
\vdots & & & & \vdots & \\
0 & & 0 & b_n & b_1 & B(x)
\end{bmatrix}$$

where $a_i = 0$ if $i < 0$, and $b_j = 0$ if $j < 0$. If N is the Sylvester matrix of A, B, then M is the matrix obtained from N by multiplying column i by x^{m+n-i}, and adding the result to column $m+n$, for $1 \le i < m+n$. Expanding $\det(M)$ by column $m+n$ we obtain $A(x)S(x) + B(x)U(x) = \det(M) = \det(N) = c$, where $\deg(S) < n$, $\det(U) < m$. $c \neq 0$, so by Lemma 4.5.1, $U(x) = T(x)$, and $S(x)$ is the first subresultant copolynomial of (A,B) of the second kind. Therefore we see that the cofactors of the last m

elements in the expansion of the determinant by the last column are the coefficients of T.

Let D be the matrix corresponding to any such cofactor. D is an $n + m - 1$ by $n + m - 1$ matrix with n rows of A coefficients and $m - 1$ rows of B coefficients. It follows by Theorem 1 of [COHO73] that

$$|\det(D)|_1 \leq \Pi_{i=1}^{m-n+1} |D_i|_1 ,$$

where D_i is the i^{th} row of D. Hence $|\det(D)|_\infty \leq |\det(D)|_1 \leq |A|_1^n |B|_1^{m-1}$, and therefore $|T|_\infty \leq |A|_1^n |B|_1^{m-1}$.

$|c| \leq |A|_1^n |B|_1^n$ follows directly from Theorem 2 of [COHO73]. ∎

Suppose a is the input to the following algorithm, ANRECP, and $rp(a) = r$, $polp(a) = \bar{A}$, and $n = deg(\bar{A})$. If $n = 0$ then $\bar{A} = \pm 1$ so RNRECP is used to compute $a^{-1} = r^{-1}\bar{A}$ and the algorithm terminates. Otherwise $\bar{\Psi}$ is computed, and if $m = deg(\bar{\Psi})$ then $d = |\bar{\Psi}|_1^n |\bar{A}|_1^m$ is determined. Next, for a number of primes determined by d, $\bar{\Psi}^*$ and $\bar{A}^*$ are computed, CRESCP constructs c^* and T^*, and partial solutions of c and T are formed by application of the Chinese Remainder Theorem, Theorem 1.4.7. This theorem, as applied here, assures us that when the product of the primes used is at least as great as $2d$ that the partial solutions constructed are indeed c and T. At this point the canonical form of $b = a^{-1}$ is constructed and the algorithm terminates.

Algorithm 4.5.2. b = ANRECP(a)

(Algebraic Number, Reciprocal)

a is a non-zero element of $Q(\alpha)$, $b = a^{-1}$.

Description

(1) [Initialize.] FIRST2(r,$\bar{A}$,a); n ← PDEG($\bar{A}$); if n = 0, (s ← RNRECP(r); b ← PFL(s,PFL(BORROW($\bar{A}$),0)); return); PGCDCF(Ψ,$\bar{A}$, B,$\bar{\Psi}$,C); erase B, C; m ← PDEG($\bar{\Psi}$).

(2) [Compute $D = 2|\bar{\Psi}|_1^n |\bar{A}|_1^m$, the bound for c and T.] d_1 ← PNORMF($\bar{\Psi}$); d_2 ← PNORMF($\bar{A}$); e_1 ← IPOWER(d_1,n); e_2 ← IPOWER(d_2,m); erase d_1, d_2; D ← IPROD(e_1,e_2); erase e_1, e_2 IMFI(D,2).

(3) [Initialize for Chinese Remainder Theorem.] L ← PRIME; Q ← PFA(1,0); T ← 0; c ← 0; V ← PVLIST($\bar{\Psi}$).

(4) [Compute $\phi_p(\bar{\Psi})$ and $\phi_p(\bar{A})$.] If L = 0, stop; ADV(p,L); $\bar{\Psi}^*$ ← CPMOD(p,$\bar{\Psi}$); if CPDEG($\bar{\Psi}^*$) < m, (erase $\bar{\Psi}^*$; go to (4)); $\bar{A}^*$ ← CPMOD(p,$\bar{A}$); if CPDEG($\bar{A}^*$) < n, (erase $\bar{\Psi}^*$, $\bar{A}^*$; go to (4)); Q^* ← CMOD(p,Q); q ← CRECIP(p,Q^*).

(5) [Compute $\phi_p(c)$ and $\phi_p(T)$.] CRESCP(p,$\bar{\Psi}^*$,$\bar{A}^*$,c^*,T^*); erase $\bar{\Psi}^*$, $\bar{A}^*$; if $c^* = -1$, go to (4).

(6) [Apply the Chinese Remainder Theorem to compute T.] U ← CPCRA(Q,T,p,T^*,q,V); erase T, T^*; T ← U.

(7) [Apply the Chinese Remainder Theorem to compute c.] w ← CCRA(Q,c,p,c^*,q); erase c; c ← w.

(8) [Determine if the bound on $|c|$ and T has been reached.] IMFI(Q,p); if ICOMP(Q,D) = -1, go to (4).

(9) [Construct the canonical form and finish up.] Erase $\bar{\Psi}$, Q, D, V; d ← PCONT(T); if ISIGNL(c) = -1, IMFI(d,-1); $\bar{B}$ ← PSQ(T,d); erase T; t ← RNINT2(d,c); s ← RNQ(t,r); erase t, d, c; b ← PFL(s,PFL($\bar{B}$,0)); return.

Theorem 4.5.9. Let $a \in Q(\alpha)$ such that $a \neq 0$ and define $n = \deg(a)$, $k = \deg(\Psi)$, $f = \nu(a)$, $g = |\Psi|_1$, $\bar{g} = |\bar{\Psi}|_1$, and $h = \max(f,g,\bar{g})$. If $n = 0$ then $t_{ANRECP}(a) \preceq L(f)$. If $n > 0$ then $t_{ANRECP}(a) \preceq k^3 L(h)^2$.

Proof. If $n = 0$ then $t_{ANRECP}(a) \preceq t_{RNRECP}(r) \preceq L(r) \preceq L(f)$.

If $n > 0$ then $t_{PGCDCF}(\Psi,\bar{A}) \preceq k^3 L(h)^2$. In step (2) $t_{PNORMF}(\bar{\Psi}) \preceq kL(\bar{g}) \preceq kL(h)$ and similarly $t_{PNORMF}(\bar{A}) \preceq kL(h)$, while $t_{IPOWER}(d_1,n) \preceq n^2 L(d_1)^2 \preceq k^2 L(\bar{g})^2 \preceq k^2 L(h)^2$ and similarly $t_{IPOWER}(d_2,m) \preceq k^2 L(h)^2$. $t_{IPROD}(e_1,e_2) \preceq L(e_1)L(e_2) = L(d_1^n)L(d_2^m) \preceq L(\bar{g}^k)L(f^k) \preceq L(h^k)L(h^k) \sim k^2 L(h)^2$.

Let us say that a prime p is unlucky if $p \mid \mathrm{ldcf}(\bar{\Psi})$, $p \mid \mathrm{ldcf}(\bar{A})$, or $p \mid \mathrm{res}(\bar{\Psi},\bar{A})$, and lucky otherwise. Let n_i be the number of times step (i) is executed with a lucky prime and m_j be the number of times step (j) is executed for an unlucky prime for $i = 4,\ldots,8$ and $j = 4,5$. For $i = 4,\ldots,8$ we have that $n_i \preceq L(D) \preceq L(2\bar{g}^n f^m) \sim nL(\bar{g}) + mL(f) \preceq kL(h)$. Also, $m_4 \preceq L(\mathrm{ldcf}(\bar{\Psi})) + L(\mathrm{ldcf}(\bar{A})) \preceq L(\bar{g}) + L(f) \preceq L(h)$, while $m_5 \preceq L(c) \preceq L(\bar{g}^n f^m) \preceq kL(h)$. Therefore if N_i is the total number of executions of step (i), $N_i \preceq kL(h)$ for $i = 4,\ldots,8$.

Next we show that the time for one execution of step (i) is $\preceq k^2 L(h)$ for $i = 4,\ldots,8$. In step (4), $t_{CPMOD}(p,\bar{\Psi}) \preceq (\deg(\bar{\Psi}))\, L(|\bar{\Psi}|_\infty) \leq mL(\bar{g}) \leq kL(h)$, and similarly $t_{CPMOD}(p,\bar{A}) \preceq kL(h)$. In step (5), $t_{CRESCP}(p,\bar{\Psi}^*,\bar{A}^*) \preceq mn \preceq k^2$. In step (6), $t_{CPCRA}(Q,T,p,T^*,q) \preceq \deg(U)L(|U|_\infty) \preceq kL(\bar{g}^n f^m) \preceq k^2 L(h)$. $t_{CCRA}(Q,c,p,c^*,q) \preceq L(w) \preceq L(\bar{g}^n f^m) \preceq kL(h)$ in step (7). And finally, in step (8) $t_{IMFI}(Q,p)$, $t_{ICOMP}(Q,D) \preceq L(Q) \preceq L(D) \preceq kL(h)$.

To complete the proof we analyze step (9). $t_{PCONT}(T) \preceq (\deg(T)+1)L(|T|_\infty)^2 \preceq k^3 L(h)^2$, $t_{ISIGNL}(c) \preceq L(c) \preceq kL(h)$, and $t_{IMFI}(d,-1) \preceq L(d) \preceq kL(h)$. $t_{PSQ}(T,d) \preceq (\deg(T)+1)L(|T|_\infty)L(d) \preceq k^3 L(h)^2$, while $t_{RNINT2}(d,c) \preceq L(d)L(c) \preceq k^2 L(h)^2$, and $t_{RNQ}(t,r) \preceq L(t)L(r) \preceq kL(h)\, L(f) \preceq kL(h)^2$ by Theorems 2.4.5 and 2.4.10 respectively. ∎

Corollary 4.5.2. Let a, n, k, f, g, $\bar{g}$, h be as defined in Theorem 4.5.9. Let $\ell = \max(f,g)$ and assume $n > 0$. Then $t_{ANRECP}(a) \preceq k^5 + k^3 L(\ell)^2$.

Proof. By Corollary 1.4.1, $\bar{g} \leq (k+1)^{2k} g$, so $h \leq (k+1)^{2k} \ell$ and $L(h) \preceq k + L(\ell)$, assuming $L(k) = 1$. The result then follows from Theorem 4.5.9. ∎

Lemma 4.5.4. Let $a = ((r_1,r_2),\bar{A})$ be a non-zero element of $Q(\alpha)$. Let $k = \deg(\Psi)$, $g = |\Psi|_1$. If $b = \mathrm{ANRECP}(a)$, then $\nu(b) \leq (k+1)^{2k^2} g^{k-1} |\bar{A}|_1^{k-1} (r_2 k + r_1 |\bar{A}|_1)$ and $L(\nu(b)) \preceq k(L(\nu(a))+L(g)+k)$.

Proof. Let $\bar{g} = |\bar{\Psi}|_1$ and observe that $b(y) = \dfrac{r_2 T(y)}{r_1 |c|}$, where

$\bar{\Psi}$, T, c are as described above. By Theorem 4.5.8, $|T|_\infty \le \bar{g}^{k-1}|\bar{A}|_1^{k-1}$ and $|c| \le \bar{g}^{k-1}|\bar{A}|_1^{k}$. Hence $\nu(b) \le |r_2 T|_1 + r_1|c| \le r_2 k \bar{g}^{k-1}|\bar{A}|_1^{k-1} + r_1\bar{g}^{k-1}|\bar{A}|_1^{k}$. $\bar{g} \le (k+1)^{2k} g$ by Corollary 1.4.1, so $\nu(b) \le (k+1)^{2k^2} g^{k-1}|\bar{A}|_1^{k-1}(r_2 k + r_1|\bar{A}|_1)$.

Now observe that $L(\nu(a)) = L(|r_1\bar{A}|_1 + r_2) \sim \max(L(|r_1\bar{A}|_1), L(r_2)) \sim L(|r_1\bar{A}|_1) + L(r_2) \sim L(r_1) + L(r_2) + L(|\bar{A}|_1)$. Assuming $L(k) \sim 1$, $L(\nu(b)) \le k^2 + kL(g) + kL(|\bar{A}|_1) + L(r_2) + L(r_1) + L(|\bar{A}|_1) \le k\{k+L(g) + L(r_1)+L(r_2)+L(|\bar{A}|_1)\} \sim k\{L(\nu(a))+L(g)+k\}$. ∎

Let A and B be elements of $Q(\alpha)[x]$ such that B is non-zero and $\deg(A) \ge \deg(B)$. Before discussing an algorithm for computing the quotient and remainder of A and B, let us recall the representation we employ and consider the result we want to obtain. A and B are elements of $Q[y,x]/(\Psi(y))$ representing $A^* = \psi_\alpha(A)$ and $B^* = \psi_\alpha(B)$, such that no non-zero coefficient of A or B has α as a root. We seek to obtain elements Q, R of $Q[y,x]/(\Psi(y))$, whose non-zero coefficients do not have α as a root, such that $Q^* = \psi_\alpha(Q)$, $R^* = \psi_\alpha(R)$, where $A^* = Q^*B^* + R^*$ with $R^* = 0$ or $\deg(R^*) < \deg(B^*)$.

Since every element of $Q(\alpha)$ is a unit, Q and R are always defined. Therefore we could construct a division algorithm similar to PMQRMM, but for polynomials over $Q(\alpha)$. Such an algorithm would generate a sequence of polynomials $C_0 = A, C_1, \ldots, C_e = R$ over $Q(\alpha)$. The sequence is defined by

$$C_{i+1} = bC_i - c_i B\, x^{n_i - n},$$

where $b = \mathrm{ldcf}(B)$, $c_i = \mathrm{ldcf}(C_i)$, $n_i = \deg(C_i)$ and $n = \deg(B)$. However, while it has been shown by G. E. Collins that such an algorithm has a polynomial bounded computing time, the computing time analysis of it is by no means obvious. The problem arises in determining a bound on $\nu(C_i)$. We have side-stepped this more complex analysis by developing instead, an algorithm which computes a sequence corresponding to $C_0 = A, C_1, \ldots, C_e = R$ in $Q[y,x]/(\Psi(y))$, by application of PMQRMM.

Let $n = \deg(B)$, $b = \mathrm{ldcf}(B)$, $c = \mathrm{ANRECP}(b)$ and $\hat{B} = x^n + c\,\mathrm{red}(B)$, where the product is defined in $Q[y,x]/(\Psi(y))$. Then A and $\hat{B}$, considered only as elements of $Q[y,x]/(\Psi(y))$, have a unique quotient, $\bar{Q}$, and remainder, $\hat{R}$, so that $A = \bar{Q}\hat{B} + \hat{R}$. Let $\hat{Q} = c\bar{Q}$. Since $\psi_\alpha(b\hat{B}) = B^*$ and $\psi_\alpha(b)\,\psi_\alpha(c) = 1$, we have that $A^* = \psi_\alpha(\bar{Q})\,\psi_\alpha(\hat{B}) + \psi_\alpha(\hat{R}) = \psi_\alpha(c\bar{Q})\psi_\alpha(b\hat{B}) + \psi_\alpha(\hat{R}) = \psi_\alpha(\hat{Q})B^* + \psi_\alpha(\hat{R})$. Now $\deg(\psi_\alpha(\hat{R})) \le \deg(\hat{R}) < \deg(\hat{B}) = \deg(B)$, and Q^*, R^* are unique, so $R^* = \psi_\alpha(\hat{R})$, $Q^* = \psi_\alpha(\hat{Q})$.

Algorithm APQREM first determines whether $A = 0$ or $\deg(A) < \deg(B)$. If further computation is required, then c and $\hat{B}_1 = c\,\mathrm{red}(B)$ are constructed. Next PMQRMM is used to determine $\bar{Q}$, $\hat{R}$. $\hat{Q} = c\bar{Q}$ is computed, and finally $Q = \mathrm{AREP}(\hat{Q})$, $R = \mathrm{AREP}(\hat{R})$.

Algorithm 4.5.3. L = APQREM(A,B)

(Algebraic Polynomial, Quotient and Remainder)

A, B are elements of $Q(\alpha)[x]$ such that B is non-zero. L is the list (Q,R), where $Q = \mathrm{quo}(A,B)$ and $R = \mathrm{rem}(A,B)$.

then L is called a standard modular representation of C.

Definition 5.2.5. Let A and B be elements of $I[y,x]$ of positive degree and $S(A,B) = (S_1,S_2,\ldots,S_{h+1})$. If $S(A,B) = (S_1,S_2)$, let $L = ()$. Otherwise, let L_i be a modular representation of S_i for $3 \le i \le h+1$ and $L = (L_3,L_4,\ldots,L_{h+1})$. Then L is called a modular representation of $S(A,B)$. If L_i is standard, $3 \le i \le h+1$, then L is called a standard modular representation of $S(A,B)$.

If A^* and B^* are inputs to our algorithm and they are of positive degree, then the first thing the algorithm will do is compute L, a standard modular representation of $S(A,B)$ where $A = \text{polp}(A^*)$ and $B = \text{polp}(B^*)$. Before we can present an algorithm for computing L, however, we must first discuss a topic which we have so far ignored. That is, what happens when the degree sequence of $\phi_p(A)$ and $\phi_p(B)$ differs from that of A and B, and similarly for ψ_a. The next four theorems show that the number of such homomorphisms is finite and that we can detect their occurrence.

If ϕ is such a homomorphism then it must be rejected. Let $d = (n_1,n_2,\ldots,n_{h+1})$ be the degree sequence of A and B and $\hat{d} = (\hat{n}_1,\hat{n}_2,\ldots,\hat{n}_{\hat{h}+1})$ the degree sequence of $\phi(A)$ and $\phi(B)$. Suppose $n_i \notin \hat{d}$. If $n_i \ne \hat{n}_j - 1$ for some j then by (3) of Theorem 1.4.2, $S_{n_i}(\phi(A),\phi(B)) = \phi(S_{n_i}(A,B)) = 0$. But if $n_i = \hat{n}_j - 1$ for some j, then by (2) of Theorem 1.4.2 $\phi(S_{n_i}(A,B)) = S_{n_j-1}(\phi(A),\phi(B)) \ne 0$. In general, $S_{\hat{n}_j-1}(\phi(A),\phi(B))$ is not computed when we compute $S(\phi(A),\phi(B))$.

Theorem 5.2.6. For a prime number p, let A and B be elements of $GF(p)[y,x]$ such that $\partial(A) = (k_1,n_1)$, $\partial(B) = (k_2,n_2)$, and $n_1 \ge n_2 > 0$. Let $n_1,n_2,\ldots,n_{h+1}$ be the degree sequence of A and B, j an integer such that $3 \le j \le h+1$, and $f'_j = n_1k_2 + n_2k_1 - (n_{j-1}-1)(k_1+k_2) + 1$. Define $S_p = \{a: a \in GF(p) \ \& \ \deg(\psi_a(A)) = n_1 \ \& \ \deg(\psi_a(B)) = n_2\}$. Then if $a_1,a_2,\ldots,a_m$ are distinct elements of S_p such that n_j is not in the degree sequence of $\psi_{a_i}(A)$ and $\psi_{a_i}(B)$, then $m < f'_j$.

Proof. $\deg(S_{n_{j-1}-1}(A,B)) = n_j$ by (1) of Theorem 1.4.2. By Theorem 5.2.1, $\psi_{a_i}(S_{n_{j-1}-1}(A,B)) = S_{n_{j-1}-1}(\psi_{a_i}(A),\psi_{a_i}(B))$. Hence, if $\deg(\psi_{a_i}(S_{n_{j-1}-1}(A,B))) = n_j$ then n_j is an element of the degree sequence of $\psi_{a_i}(A)$ and $\psi_{a_i}(B)$ by Theorem 1.4.2. This implies that $\deg(\psi_{a_i}(S_{n_{j-1}-1}(A,B))) < n_j$ and therefore that a_i is a root of $\text{ldcf}(S_{n_{j-1}-1}(A,B))$. By Theorem 5.2.5, $\partial_y(S_{n_{j-1}-1}(A,B)) < f'_j$. ∎

It is easy to see that Theorem 5.2.6 implies that the total number of interpolation points which fail to yield the correct degree sequence is $\le n_1 + n_2 + \sum_{j=3}^{h+1}\{n_1k_2 + n_2k_1 - (n_{j-1}-1)(k_1+k_2)\} = n_1 + n_2 + (h-1)\{n_1k_2 + n_2k_1\} - (k_1+k_2)\sum_{j=3}^{h+1}(n_{j-1}-1)$.

Definition 5.2.6. Let $L = (a_1,\ldots,a_n)$ be a list. We define the j^{th} initial segment of L to be the list $(a_1,\ldots,a_j)$ if $j \le n$ and the list $(a_1,\ldots,a_n)$ if $j > n$.

We will be comparing lexicographically initial segments of degree sequences in the following material. Recall that this term is defined in Definition 2.2.1.

Theorem 5.2.7. Let p, A, B, S_p, j and f_j' be as defined above. Let d be the j^{th} initial segment of the degree sequence of A and B. For a an element of S_p, define $\hat{d}_a$ to be the j^{th} initial segment of the degree sequence of $\psi_a(A)$ and $\psi_a(B)$. Suppose $a_1, a_2, \ldots, a_m$ are distinct elements of S_p, $\hat{d}_i$ is the j^{th} initial segment of the degree sequence of $\psi_{a_i}(A)$, $\psi_{a_i}(B)$, and $\hat{d}_1 = \hat{d}_2 = \ldots = \hat{d}_m \neq d$. Then $\hat{d}_a \leq d$ and $m < f_j'$.

Proof. $\hat{d}_a \leq d$ follows immediately from Theorem 5.2.2. By Theorem 5.2.2, $\hat{d}_i < d$ implies that n_ℓ is not an element of $\hat{d}_i$ for some ℓ, $3 \leq \ell \leq j$. Since $\hat{d}_1 = \ldots = \hat{d}_m$, n_ℓ is not an element of $\hat{d}_i$, $1 \leq i \leq m$. Hence, by Theorem 5.2.6, $m < f_\ell' \leq f_j'$. ∎

Suppose we know n_{j-1}, where $3 \leq j \leq h+1$. Then we can compute $f_j' = n_1k_2 + n_2k_1 - (n_{j-1}-1)(k_1+k_2) + 1$. Next we compute $S(\psi_a(A),\psi_a(B))$ for f_j' elements, a, with a common degree sequence, say d. Then Theorem 5.2.7 implies that the j^{th} element of d must be n_j. Then knowing n_j we can compute $f_j = f_j' + (n_{j-1}-n_j-1)(k_1+k_2)$ and hence $S(\psi_a(A),\psi_a(B))$ for f_j elements, a, with a common degree sequence. At this point we know that the lists (14) for $\hat{S}_3^{(p)}$, $\hat{S}_4^{(p)}$, ..., $\hat{S}_j^{(p)}$ contain correct data. We can discover the unlucky a's since we always apply more homomorphisms than there

are unlucky a's which have a common degree sequence, and we only save $S(\psi_a(A),\psi_a(B))$ if its degree sequence is at least as great as any that was previously computed. We throw away any old ones when we find out that they were not maximal. If the length of d is j then we have computed a standard value representation of $S(A,B)$. Otherwise, since we know n_j, we can compute f_{j+1}' and repeat the above process. Initially we know n_2 so this process always works.

Theorem 5.2.8. Let A and B be elements of $I[y,x]$ such that $\deg(A) = n_1 \geq \deg(B) = n_2 > 0$. Let $n_1, n_2, \ldots, n_{h+1}$ be the degree sequence of A and B, j an integer such that $3 \leq j \leq h+1$, and E_j' an integer such that $E_j' > |A|_1^{n_2-n_{j-1}+1} |B|_1^{n_1-n_{j-1}+1}$. Define $S = \{p: p \text{ is a prime number } \& \deg(\phi_p(A)) = n_1 \ \& \ \deg(\phi_p(B)) = n_2\}$. Then if $p_1, \ldots, p_m$ are distinct elements of S such that n_j is not in the degree sequence of $\phi_{p_i}(A), \phi_{p_i}(B)$, then $\prod_{i=1}^m p_i < E_j'$.

Proof. $\deg(S_{n_{j-1}-1}(A,B)) = n_j$ by (2) of Theorem 1.4.2. By Theorem 5.2.1, $\phi_{p_i}(S_{n_{j-1}-1}(A,B)) = S_{n_{j-1}-1}(\phi_{p_i}(A),\phi_{p_i}(B))$. Hence, if $\deg(\phi_{p_i}(S_{n_{j-1}-1}(A,B))) = n_j$ then n_j is an element of the degree sequence of $\phi_{p_i}(A)$ and $\phi_{p_i}(B)$ by Theorem 1.4.2. This implies that $\deg(\phi_{p_i}(S_{n_{j-1}-1}(A,B))) < n_j$ and therefore that $p_i \mid \text{cont}(\text{ldcf}(S_{n_{j-1}-1}(A,B)))$. Suppose $M = \prod_{i=1}^m p_i \geq E_j'$. By Theorem 5.2.4, this implies that $M > |S_{n_{j-1}-1}(A,B)|_\omega$, which is a contradiction since the p_i are distinct. ∎

Theorem 5.2.8 implies that if $p_1', p_2', \ldots, p_m'$ are the primes which fail to yield the proper degree sequence then $\prod_{i=1}^{m} p_i' \leq |A|_1 + |B|_1 + \prod_{j=3}^{h+1} |A|_1^{n_2-n_{j-1}+1} |B|_1^{n_1-n_{j-1}+1} = |A|_1^{1+\sum_{j=3}^{h+1}(n_2-n_{j-1}+1)} |B|_1^{1+\sum_{j=3}^{h+1}(n_1-n_{j-1}+1)}$. From this relation we can determine a bound on m.

Theorem 5.2.9. Let A, B, S, j and E_j' be as defined above. Let d be the j^{th} initial segment of the degree sequence of A and B. For p an element of S, define $\hat{d}_p$ to be the j^{th} initial segment of the degree sequence of $\phi_p(A)$ and $\phi_p(B)$. Suppose $p_1, \ldots, p_m$ are distinct elements of S, $\hat{d}_i$ is the j^{th} initial segment of the degree sequence of $\phi_{p_i}(A)$ and $\phi_{p_i}(B)$, and $\hat{d}_1 = \hat{d}_2 = \ldots = \hat{d}_m \neq d$. Then $\hat{d}_p \leq d$ and $\prod_{i=1}^{m} p_i < E_j'$.

Proof. $\hat{d}_p \leq d$ follows immediately from Theorem 5.2.2. By Theorem 5.2.2, $\hat{d}_i < d$ implies that n_ℓ is not an element of $\hat{d}_i$ for some ℓ, $3 \leq \ell \leq j$. Since $\hat{d}_1 = \hat{d}_2 = \ldots = \hat{d}_m$, n_ℓ is not an element of $\hat{d}_i$, $1 \leq i \leq m$. Hence, by Theorem 5.2.8, $\prod_{i=1}^{m} p_i < E_\ell' \leq E_j'$. ∎

Suppose we know n_{j-1}, where $3 \leq j \leq h+1$. Then we can compute a standard value representation of $S(\phi_{p_i}(A), \phi_{p_i}(B))$ for e_j' primes, p_i, with a common degree sequence, say d. e_j' is the least integer such that $\prod_{i=1}^{e_j'} p_i > E_j' = 2|A|_1^{n_2-n_{j-1}+1} |B|_1^{n_1-n_{j-1}+1}$. Then Theorem 5.2.9 implies that the j^{th} element of d must be n_j. Then knowing n_j we can compute $E_j = 2|A|_1^{n_2-n_j} |B|_1^{n_1-n_j} = E_j'(|A|_1 |B|_1)^{n_{j-1}-n_j-1}$ and hence a standard value representation of $S(\phi_{p_i}(A), \phi_{p_i}(B))$ for e_j primes, p_i, with a common degree sequence. e_j is the least

integer such that $\prod_{i=1}^{e_j} p > E_j$. At this point we know that the lists (15) for $S_3, S_4, \ldots, S_j$ contain correct data. We can discover the unlucky p's since we always apply more homomorphisms than there are unlucky p's which have a common degree sequence, and we only save the value representation of $S(\phi_p(A), \phi_p(B))$ if its degree sequence is at least as great as any that was previously computed. We throw away any old ones when we find out that they were not maximal. If the length of d is j then we have computed a standard modular representation of $S(A,B)$. Otherwise, since we know n_j, we can compute E_{j+1}' and repeat the above process. Initially we know n_2 so this process always works.

Up to this point we have done the following. We sketched a procedure for computing g.c.d.'s in $Q(\alpha)[x]$. We showed two ways of improving it, and in the process developed the structure, in (15), the procedure would use. We formalized our results with a series of definitions. Then we showed with the last set of theorems that our procedure could be algorithmically realized. Our next step, then, is to present the algorithms for doing this.

To construct a standard modular representation of $S(A,B)$, we first need an algorithm for computing a standard value representation of $S(\phi_p(A), \phi_p(B))$. To construct this we need an algorithm for computing $S(\psi_a(\phi_p(A)), \psi_a(\phi_p(B)))$. In other words, before we can compute any other structure we need an algorithm for computing the subresultant p.r.s. of A and B of the second kind where A and B are elements of $GF(p)[x]$. The next theorem gives us a recursion

relation to do this; we will implement it in the algorithm which follows.

Theorem 5.2.10. Let A and B be non-zero elements of $GF(p)[x]$ such that $\deg(A) \geq \deg(B)$. Let $A_1, A_2, \ldots, A_{h+2} = 0$ be the natural p.r.s. of A and B, $n_i = \deg(A_i)$, $\delta_i = n_i - n_{i+1}$, and $a_i = \mathrm{ldcf}(A_i)$. Define $c_2 = a_2^{\delta_1 - 1}$ and $c_{j+1} = (-1)^{\delta_j(n_1 - n_j + j - 1)} a_j^{\delta_j + 1} a_{j+1}^{\delta_j - 1} c_j$ for $2 \leq j \leq h$. Then $S_{n_j}(A,B) = c_j A_j$ for $3 \leq j \leq h+1$.

Proof. By (1) of Theorem 1.4.2,

$$(16) \qquad S_{n_j}(A,B) = (-1)^{\sum_{i=1}^{j-2}(n_i - n_j)(n_{i+1} - n_j)} a_j^{\delta_{j-1}-1} \left\{\prod_{i=1}^{j-2} a_{i+1}^{\delta_i + \delta_{i+1}}\right\} A_j, \quad \text{for } 3 \leq j \leq h+1.$$

Now $\sum_{i=1}^{j-2}(n_i - n_j)(n_{i+1} - n_j) = \sum_{i=1}^{j-2}(\delta_i + n_{i+1} - n_j)(n_{i+1} - n_j) \equiv \sum_{i=1}^{j-2}(\delta_i + 1)(n_{i+1} - n_j)$ modulo 2, since $a^2 \equiv a$ modulo 2. So,

$$(17) \qquad \sum_{i=1}^{j-2}(n_i - n_j)(n_{i+1} - n_j) \equiv \sum_{i=1}^{j-2}(\delta_i + 1)(n_{i+1} - n_j) \text{ modulo } 2.$$

We now prove the theorem by induction on j. If $j = 3$, then by (16) and (17), $c_3 A_3 = (-1)^{\delta_2(n_1 - n_2 + 1)} a_2^{\delta_2 + 1} a_3^{\delta_2 - 1} c_2 A_3 = (-1)^{(\delta_1 + 1)(n_2 - n_3)} a_2^{\delta_2 + 1} a_3^{\delta_2 - 1} a_2^{\delta_1 - 1} A_3 = (-1)^{(\delta_1 + 1)(n_2 - n_3)} a_2^{\delta_1 + \delta_2} a_3^{\delta_2 - 1} A_3 = S_{n_3}(A,B)$.

Suppose it holds for j. Then by (16) and (17)

$$c_j = (-1)^{\sum_{i=1}^{j-2}(\delta_i + 1)(n_{i+1} - n_j)} a_j^{\delta_{j-1}} \left\{\prod_{i=1}^{j-2} a_{i+1}^{\delta_i + \delta_{i+1}}\right\}.$$

Hence

$$c_{j+1} = (-1)^{\{\sum_{i=1}^{j-2}(\delta_i + 1)(n_{i+1} - n_j)\} + \delta_j(n_1 - n_j + j - 1)} a_j^{\delta_j + 1} a_{j+1}^{\delta_j - 1} a_j^{\delta_{j-1} - 1} \left\{\prod_{i=1}^{j-2} a_{i+1}^{\delta_i + \delta_{i+1}}\right\}$$

$$= (-1)^{\{\sum_{i=1}^{j-2}(\delta_i + 1)(n_{i+1} - n_j)\} + \delta_j \sum_{i=1}^{j-1}(\delta_i + 1)} a_{j+1}^{\delta_j - 1} \left\{\prod_{i=1}^{j-1} a_{i+1}^{\delta_i + \delta_{i+1}}\right\}.$$

The result now follows by observing that $\{\sum_{i=1}^{j-2}(\delta_i + 1)(n_{i+1} - n_j)\} + \sum_{i=1}^{j-1}\delta_j(\delta_i + 1) = \sum_{i=1}^{j-2}(\delta_i + 1)(n_{i+1} - n_j) + \sum_{i=1}^{j-2}\delta_j(\delta_i + 1) + \delta_j(\delta_{j-1} + 1) = \sum_{i=1}^{j-2}(\delta_i + 1)(n_{i+1} - n_{j+1}) + \delta_j(\delta_{j-1} + 1) = \sum_{i=1}^{j-1}(\delta_i + 1)(n_{i+1} - n_{j+1})$. Therefore, by (16) and (17) the theorem is true for $j+1$ and by induction is true in general. ∎

Algorithm 5.2.1. CSUBS2(p,A,B,S,d)

(Congruence, Subresultant Sequence of the 2nd Kind)

The inputs are p, a prime number, and A and B, elements of $GF(p)[x]$ such that $\deg(A) \geq \deg(B) > 0$. The outputs are S, the subresultant p.r.s. of A and B of the second kind, and d, the degree sequence of A and B.

Description

(1) [Initialize.] $A_1 \leftarrow$ BORROW(A); $A_2 \leftarrow$ BORROW(B); $a_1 \leftarrow$ CPLNCF(A_1); $n_1 \leftarrow$ CPDEG(A_1); $n_2 \leftarrow$ CPDEG(A_2); $\bar{n}_1 \leftarrow n_1$; $j \leftarrow 2$; d $\leftarrow$ PFA(n_2,PFA(n_1,0)); S $\leftarrow$ PFL(BORROW(A_2),PFL(BORROW(A_1),0)).

(2) [Compute $c_2 = a_2^{\delta_1 - 1}$.] $\delta_1 \leftarrow n_1 - n_2$; $a_2 \leftarrow$ CPLNCF(A_2); $t \leftarrow \delta_1 - 1$; if $t < 0$; $c_2 \leftarrow$ CRECIP(p,a_2); if $t \geq 0$, $c_2 \leftarrow$ CPOWER(p,a_2,t).

(3) [Compute $A_{j+1} = \text{rem}(A_{j-1}, A_j)$.] $A_3 \leftarrow$ CPREM(p,A_1,A_2); erase A_1; if $A_3 = 0$, go to (6); $n_3 \leftarrow$ CPDEG(A_3); $a_3 \leftarrow$ CPLNCF(A_3).

(4) [Compute $c_{j+1} = (-1)^{\delta_j (n_1 - n_j + j - 1)} a_j^{\delta_j + 1} a_{j+1}^{\delta_j - 1} c_j$ and $S_{j+1} = c_{j+1} A_{j+1}$.] $\delta_2 \leftarrow n_2 - n_3$; $t \leftarrow$ CPOWER(p,a_2,δ_2+1); $u \leftarrow$ CPOWER(p,a_3,δ_2-1); $v \leftarrow$ CPROD(p,t,u); $c_3 \leftarrow$ CPROD(p,c_2,v); if $\delta_2 (n_1 - n_2 + j - 1) \equiv 1$ modulo 2, $c_3 \leftarrow p - c_3$; $S_3 \leftarrow$ CSPROD(p,A_3,c_3,0); $j \leftarrow j + 1$.

(5) [Determine if the loop is completed.] $S \leftarrow$ PFL(S_3,S); $d \leftarrow$ PFA(n_3,d); if $n_3 = 0$, go to (6); $A_1 \leftarrow A_2$; $A_2 \leftarrow A_3$; $n_1 \leftarrow n_2$; $n_2 \leftarrow n_3$; $a_1 \leftarrow a_2$; $a_2 \leftarrow a_3$; $c_2 \leftarrow c_3$; go to (3).

(6) [Finish-up.] $d \leftarrow$ INV(d); $S \leftarrow$ INV(S); erase A_2, A_3; return.

Theorem 5.2.11. Let p be a prime number and A and B non-zero elements of GF(p)[x] such that $m = \deg(A)$, $n = \deg(B)$ and $m \geq n > 0$. Then $t^+_{\text{CSUBS2}}(p,A,B) \sim mn$.

Proof. We use some results from [COAL69] to aid us in determining the following table of computing time bounds. Also, note that $N_3 =$ h or h - 1 and N_4, $N_5 = h - 1$.

step	# of executions	time for i^{th} execution
(1)	1	$\sim$ 1
(2)	1	$\preceq L(n_1 - n_2 + 1)$
(3)	h or h-1	$\preceq (n_{i+1}+1)(n_i - n_{i+1}+1)$
(4)	h-1	$\sim L(n_{i+1} - n_{i+2}+1) + n_{i+2} + 1$
(5)	h-1	$\sim$ 1
(6)	1	$\sim$ h

We assume that $L(n_i) \sim 1$ and note that $h \leq n_2 + 1$. Hence T_1, T_2, $T_6 \preceq h \preceq n_2 = n$. Also, $T_3 + T_4 + T_5 \preceq \sum_{i=1}^{h} \{(n_{i+1}+1)(n_i - n_{i+1}+1) + n_{i+2}+1\} \sim \sum_{i=1}^{h} (n_{i+1}+1)(n_i - n_{i+1}+1) \preceq (n_2+1) \sum_{i=1}^{h} (n_i - n_{i+1}+1) \preceq n_2 \{h + \sum_{i=1}^{h} (n_i - n_{i+1})\} \preceq n_2 \{n_2 + n_1\} \sim n_1 n_2 = mn$.

Suppose A and B are relatively prime and their degree sequence is normal, that is, $n_{i+1} = n_i - 1$ for $2 \leq i \leq h$. Then $h = n_2 + 1$ and $n_i = n_2 + 2 - i$ for $2 \leq i \leq h + 1 = n_2 + 2$.

Then we have $t^+_{\text{CSUBS2}}(p,m,n) \geq T_3 \geq \sum_{i=1}^{n_2} (n_{i+1}+1)(n_i - n_{i+1}+1) = (n_2+1)(n_1 - n_2 + 1) + \sum_{i=2}^{n_2} (n_{i+1}+1)(n_i - n_{i+1}+1) = (n_2+1)(n_1 - n_2 + 1) + 2 \sum_{i=2}^{n_2} (n_2 + 2 - i) = (n_2+1)(n_1 - n_2 + 1) + 2 \sum_{i=2}^{n_2} i = (n_2+1)(n_1 - n_2 + 1) + n_2(n_2+1) - 2 = (n_2+1)(n_1+1) - 2 > n_1 n_2 = mn$, since $n_1 + n_2 + 1 > 2$. ∎

The next algorithm, CSVRS2, computes the standard value representation of $S(A,B)$, where A and B are elements of GF(p)[y,x] such that $\partial(A) = (k_1, n_1)$, $\partial(B) = (k_2, n_2)$ and $n_1 \geq n_2 > 0$. Let $n_1, n_2, \ldots, n_{h+1}$ be the degree sequence of A and B; $S(A,B) = (S_1, S_2, \ldots, S_{h+1})$; $f'_j = (n_2 - n_{j-1} + 1)k_1 + (n_1 - n_{j-1} + 1)k_2 + 1$, and $f_j = (n_2 - n_j)k_1 + (n_1 - n_j)k_2 + 1$ for $3 \leq j \leq h + 1$. Note that $f_j = f'_j + (n_{j-1} - n_j - 1)(k_1 + k_2)$ and $f'_{j+1} = f_j + (k_1 + k_2)$. The algorithm computes $\psi_a(A) = \hat{A}$, $\psi_a(B) = \hat{B}$ and $S(\hat{A},\hat{B})$ for f_{h+1} elements, a, of GF(p) which have the correct degree sequence. It does this by computing f'_3 sequences. By Theorem 5.2.7 this determines n_3 and thus f_3. f_3 images of $S(A,B)$ with maximal degree sequences are computed. Then f'_4 and f'_4 degree sequences are computed. This determines n_4 and hence f_4, so f_4 more images are computed, and so on. It

saves exactly f_i images of S_i and throws the rest away.

$S(A,B)$ is constructed in the following manner. In step (2), $\hat{A}$ and $\hat{B}$ are computed.

In step (3), CSUBS2 computes $\hat{S} = S(\hat{A},\hat{B})$ and $\hat{d}$, the degree sequence of $\hat{A}$ and $\hat{B}$.

In step (4), $\hat{d}$ is compared with d. d is the common degree sequence of all the previously applied homomorphisms, or else $d = 0$ if a is the first one. If $d > \hat{d}$ then a is unlucky so we return to step (2) for the next homomorphism. If $d < \hat{d}$ then the old homomorphisms are all unlucky, or else $d = 0$, so the variables are initialized to their starting values: $d = \hat{d}$, $L = 0$, $f = f'_3$, $m = 0$, $j = 2$ and d' points to n_2 in d. f takes on the values f'_{j+1} and f_{j+1}, the bound on the number of mappings needed to compute S_{j+1}; j indicates how many of the S_i have enough images; m counts the number of homomorphisms which have been applied; and d' points to n_j.

Suppose d is not the degree sequence of $S(A,B)$, that the first m homomorphisms are unlucky. If $d_1,\ldots,d_m$ are the degree sequences generated by use of these homomorphisms, then the $j+1^{st}$ initial segments of the d_i all agree. Then by Theorem 5.2.7, $m < f'_{j+1}$. But since $f \leq f'_{j+1}$, CSVRS2 will detect this and act accordingly.

In step (5), $\hat{A}$ and $\hat{B}$ are decapped from $\hat{S}$ and erased, and $i = 2$ and L' are initialized for the loop consisting of step (6).

In step (6), for $i = 3,4,\ldots,h+1$, $\hat{S}_i$ is removed from $\hat{S}$. If $j = 2$ then $L = 0$. If $L \neq 0$ then L is the list $(L_3,L_4,\ldots,L_{h+1})$ where L_i is a partially computed standard value representation of S_i. The variable $L_3 = 0$ if $L = 0$, otherwise L_3 is given the value L_i by decapping L. If $i \leq j$ then enough images of S_i have been saved so $\hat{S}_i$ is erased. If $i > j$ then $\hat{S}_i$ and a are prefixed to $L_3 = L_i$.

In step (7), L is set to the inverse of L', m is incremented by one, and if $m < f = f'_{j+1}$ then control returns to step (2) for another iteration. If TAIL(d') = 0 then $d = (n_1,n_2)$, so we return. If $m = f = f'_{j+1}$ then f is set to f_{j+1} and control returns to step (2). If $m = f = f_{j+1}$ then enough images for S_{j+1} have been computed, so step (8) is executed next.

In step (8), d' is set to tail (d'). By Theorem 5.2.7, d' now points to n_{j+1}. If tail(d') = 0 then $j+1=h+1$ and L is complete, so the algorithm terminates. Otherwise f is replaced by $f_{j+1} + (k_1+k_2) = f'_{j+2}$ and j is incremented. Then control is returned to step (2) for $f'_{j+2} - f_{j+1}$ more executions of steps (2)-(7) before step (8) again checks for a terminal condition.

Algorithm 5.2.2. CSVRS2(p,A,B,L,d)

(Congruence, Standard Value Representation of Subresultant PRS of the 2nd Kind)

The inputs are p, a prime number, and A and B, elements of $GF(p)[y,x]$ such that $\partial_x(A) \geq \partial_x(B) > 0$. The outputs are L, a standard value representation of the subresultant p.r.s. of A and B of the second kind, and d, the degree sequence of A and B.

Description

(1) [Initialize.] $T \leftarrow$ CPDEGS(A); DECAP2(n_1,k_1,T); $T \leftarrow$ CPDEGS(B); DECAP2(n_2,k_2,T); $d \leftarrow 0$; $a \leftarrow -1$; $L \leftarrow 0$; $\delta_1 \leftarrow n_1 - n_2$; $\bar{n}_2 \leftarrow n_2$; $t \leftarrow 0$.

(2) [Compute $\hat{A}$ and $\hat{B}$.] $a \leftarrow a + 1$; if $a = p$, stop; $\hat{A} \leftarrow$ CPUEV(p,A,a); if CPDEG($\hat{A}$) $< n_1$, (erase $\hat{A}$; go to (2)); $\hat{B} \leftarrow$ CPUEV(p,B,a); if CPDEG($\hat{B}$) $< \bar{n}_2$, (erase $\hat{A}$, $\hat{B}$; go to (2)).

(3) [Compute $S(\psi_a(A),\psi_a(B))$.] CSUBS2(p,$\hat{A}$,$\hat{B}$,$\hat{S}$,$\hat{d}$); erase $\hat{A}$, $\hat{B}$.

(4) [Compare degree sequences.] $s \leftarrow$ LEXORD(d,$\hat{d}$); if $s = 0$, erase $\hat{d}$; if $s = +1$, (erase $\hat{S}$, $\hat{d}$; go to (2)); if $s = -1$, (erase d; $d \leftarrow \hat{d}$; $d' \leftarrow$ TAIL(d); $n_2 \leftarrow \bar{n}_2$; erase L; $L \leftarrow 0$; $f \leftarrow k_1 + (\delta_1+1)k_2 + 1$; $m \leftarrow 0$; $j \leftarrow 2$).

(5) [Remove $\hat{A}$ and $\hat{B}$ from $\hat{S}$.] DECAP2($\hat{A}$,$\hat{B}$,$\hat{S}$); erase $\hat{A}$, $\hat{B}$; $i \leftarrow 2$; $L' \leftarrow 0$.

(6) [Form L'.] If $\hat{S} = 0$, go to (7); $i \leftarrow i + 1$; DECAP($\hat{S}_3$,$\hat{S}$); $L_3 \leftarrow 0$; if $L \neq 0$, DECAP(L_3,L); if $i \leq j$, erase $\hat{S}_3$; if $i > j$, $L_3 \leftarrow$ PFA(a,PFL($\hat{S}_3$,L_3)); $L' \leftarrow$ PFL(L_3,L'); go to (6).

(7) [Is the $j-1^{st}$ list of L complete?] $L \leftarrow$ INV(L'); $m \leftarrow m + 1$; if $m < f$, go to (2); if TAIL(d') = 0, return; if $t = 0$, ($t \leftarrow 1$; $n_3 \leftarrow$ SECOND(d'); $f \leftarrow f + (n_2-n_3-1)(k_1+k_2)$; if $m < f$, go to (2)).

(8) [Compute bound for S_{j+2} and loop.] $d' \leftarrow$ TAIL(d'); if TAIL(d') = 0, return; $j \leftarrow j + 1$; $f \leftarrow f + (k_1+k_2)$; $t \leftarrow 0$; $n_2 \leftarrow n_3$; go to (2).

Theorem 5.2.12. Let p be a prime number and A and B elements of GF(p)[y,x] such that $\partial(A) = (k_1,n_1)$, $\partial(B) = (k_2,n_2)$ and $n_1 \geq n_2 > 0$. If $k = \max(k_1,k_2) + 1$ then $t_{CSVRS2}(p,A,B) \leq k^2n_1^2n_2 + kn_1^2n_2^2$.

Proof. Let $n_1,n_2,\ldots,n_{h+1}$ be the degree sequence of A and B. Clearly $T_1 \sim 1$. N_2, N_3, N_4, N_5 and N_7 are $\leq$ the total number of homomorphisms used. Hence by definition of a standard value representation and Theorem 5.2.6, for $i = 2,3,4,5$, and 7, $N_i \leq n_1 + n_2 + (h-1)(n_1k_2 + n_2k_1) - (k_1 + k_2)\sum_{j=3}^{h+1}(n_{j-1}-1) + (n_1k_2 + n_2k_1) - n_{h+1}(k_1 + k_2) + 1 = h(n_1k_2 + n_2k_1) - (k_1 + k_2)(n_{h+1} + \sum_{j=3}^{h+1}(n_{j-1}-1)) + 1 \leq n_2(n_1k_2 + n_2k_1) \leq kn_1n_2$. Now step (6) is executed h-1 times for every execution of step (5), hence $N_6 \leq kn_1n_2(h-1) \leq kn_1n_2^2$. $N_8 \leq h - 1 \leq n_2$. From page 31 of [COL72], $t_2 \leq kn_1 + kn_2 \leq kn_1$; hence $T_2 \leq k^2n_1^2n_2$. By Theorem 5.2.11, $t_{CSUBS2}(p,\hat{A},\hat{B}) \leq n_1n_2$ and therefore $T_3 \leq kn_1^2n_2^2$. By Theorem 2.2.4, $t_{LEXORD}(d,\hat{d}) \leq h \leq n_2$ so $T_4 \leq kn_1n_2^2$. $T_5 \leq kn_1n_2$ and $T_6 \leq kn_1n_2^2$. $t_7 \leq h - 1 \leq n_2$ which implies that $T_7 \leq kn_1n_2^2$. $T_8 \leq n_2$. ∎

Next comes PSMRS2, an algorithm which computes the standard modular representation of $S(A,B)$, where A and B are elements of $I[y,x]$ such that $\deg(A) > \deg(B) > 0$. PSMRS2 is the same algorithm as CSVRS2, except as applied to modular instead of evaluation homomorphisms. Hence we limit our discussion to a comparison of the differences between the two algorithms.

PSMRS2 computes a list whose elements are value representations, whereas CSVRS2 needs homomorphic images; hence in step (3) CSVRS2 calls CSUBS2 while PSMRS2 calls CSVRS2. Theorem 5.2.9 implies that the value representations computed have the proper degree sequence: this can be shown using an argument similar to the one used above with Theorem 5.2.7.

Let $E'_j = 2|A|_1^{n_2-n_{j-1}+1} |B|_1^{n_1-n_{j-1}+1}$ and $E_j = 2|A|_1^{n_2-n_j} |B|_1^{n_1-n_j}$, where $n_1,n_2,\ldots,n_{h+1}$ is the degree sequence of A and B. Instead of counting the number of homomorphisms applied, PSMRS2 forms the product of the primes used and compares this with E'_j or E_j. Let $\bar{E}_1$ and $\bar{E}_2$ be successive values of E. Suppose M is this product of primes and let $M > \bar{E}_1$. If $M > \bar{E}_1$, it may be that $M > \bar{E}_2$. Therefore, steps (8) and (9) are executed repeatedly until a j is determined such that $M \leq E'_{j+1}$ or $n_{j+1} = n_{h+1}$. Step (8) of CSVRS2 has become step (9) of PSMRS2.

We also note that $E'_3 = 2|A|_1 |B|_1^{n_1-n_2+1}$, $E_j = E'_j\{|A|_1 |B|_1\}^{n_{j-1}-n_j-1}$ and $E'_{j+1} = E_j\{|A|_1 |B|_1\}$.

Algorithm 5.2.3. PSMRS2(A,B,L,d)

(Polynomial, Standard Modular Representation of Subresultant PRS of the 2nd Kind)

The inputs are A and B, elements of I[y,x] such that deg(A) ≥ deg(B) > 0. The outputs are L, a standard modular representation of the subresultant p.r.s. of A and B of the second kind, and d, the degree sequence of A and B.

Description

(1) [Initialize.] $n_1 \leftarrow$ PDEG(A); $n_2 \leftarrow$ PDEG(B); f ← PNORMF(A); h ← PNORMF(B); c ← IPROD(f,h); t ← IPOWER(h,n_1-n_2+1); $E'_3 \leftarrow$ IPROD(f,t); IMFI(E'_3,2); erase t, f, h; W ← PRIME; L ← 0; E ← 0; d ← 0; M ← 0; $\bar{n}_2 \leftarrow n_2$; t ← 0.

(2) [Compute $\hat{A}$ and $\hat{B}$.] If W = 0, stop; ADV(p,W); A ← CPMOD(p,A); if CPDEG($\hat{A}$) < n_1, (erase $\hat{A}$; go to (2)); $\hat{B} \leftarrow$ CPMOD(p,B); if CPDEG($\hat{B}$) < $\bar{n}_2$, (erase $\hat{A}$, $\hat{B}$; go to (2)).

(3) [Compute a standard value representation of $S(\hat{A},\hat{B})$.] CSVRS2(p,$\hat{A}$,$\hat{B}$,$\hat{L}$,$\hat{d}$); erase $\hat{A}$, $\hat{B}$.

(4) [Compare degree sequences.] s ← LEXORD(d,$\hat{d}$); if s = 0, erase $\hat{d}$; if s = +1, (erase $\hat{L}$, $\hat{d}$; go to (2)); if s = −1, (erase d; d ← $\hat{d}$; d' ← TAIL(d); $n_2 \leftarrow \bar{n}_2$; erase L; L ← 0; erase E; E ← BORROW(E'_3); erase M; M ← PFA(1,0); j ← 2).

(5) [Initialize for the next loop.] i ← 2; L' ← 0.

(6) [Form L'.] If $\hat{L}$ = 0, go to (7); i ← i + 1; DECAP($\hat{L}_3$,$\hat{L}$); L_3 ← 0; if L ≠ 0, DECAP(L_3,L); if i ≥ j, erase $\hat{L}_3$; if i > j, L_3 ← PFA(p,PFL($\hat{L}_3$,L_3)); L' ← PFL(L_3,L'); go to (6).

(7) [Update L and M.] L ← INV(L'); IMFI(M,p).

(8) [Check for completion of L_{j+1}.] If ICOMP(M,E) < 0, go to (2); if TAIL(d') = 0, (erase c, E'_3, E, M; return); if t = 0, (t ← 1; n_3 ← SECOND(d'); u ← IPOWER(c,n_2-n_3-1); v ← IPROD(E,u); erase E, u; E ← v; go to (8)).

(9) [Compute bound for L_{j+2} and loop.] d' ← TAIL(d'); if TAIL(d') = 0, (erase c, E'_3, E, M; return); j ← j + 1; t ← 0; u ← IPROD(E,c); erase E; E ← u; $n_2 \leftarrow n_3$; go to (8).

Theorem 5.2.13. Let A and B be elements of I[y,x] such that $\partial(A) = (k_1,n_1)$, $\partial(B) = (k_2,n_2)$ and $n_1 \geq n_2 > 0$. If $f = |A|_1$, $h = |B|_1$ and $k = \max(k_1,k_2) + 1$ then $t_{PSMRS2}(A,B) \leq kn_1^2n_2L(fh)\{L(fh) + kn_1n_2 + n_1n_2^2\}$.

Proof. Let $n_1, n_2, \ldots, n_{\ell+1}$ be the degree sequence of A and B. In step (1), it follows from a result in [HEI70] that $t_{PNORMF}(A) \preceq kn_1 L(f)$ and $t_{PNORMF}(B) \preceq kn_2 L(h)$. Also, $t_{IPROD}(f,h) \preceq L(f)L(h) \preceq L(fh)^2$. From a result in [COL73d], $t_{IPOWER}(h,n_1-n_2+1) \preceq (n_1-n_2+1)^2 L(h)^2 \preceq n_1^2 L(h)^2$. $t_{IPROD}(f,t) \preceq L(t)L(f) \preceq (n_1-n_2+1)L(h)L(f) \preceq n_1 L(hf)^2$. From a result in [COL73d], $t_{IMFI}(E_3',2) \preceq L(E_3') \preceq L(f) + (n_1-n_2+1)L(h) \preceq n_1 L(h) + L(f) \preceq n_1 L(fh)^2$.

N_2, N_3, N_4, N_5, N_7 are $\preceq$ the total number of homomorphisms applied. Hence by definition of a standard modular representation and Theorem 5.2.8, for i = 2,3,4,5 and 7, $N_i \preceq \{1 + n_2 - n_{h+1} + \sum_{j=3}^{\ell+1} (n_2 - n_{j-1} + 1)\} L(|A|_1) + \{1 + n_1 - n_{h+1} + \sum_{j=3}^{\ell+1} (n_1 - n_{j-1} + 1)\} L(|B|_1) \preceq \{1 + n_2 + \sum_{j=3}^{\ell+1} n_2\} L(f) + \{1 + n_1 + \sum_{j=3}^{\ell+1} n_1\} L(h) \preceq \ell n_2 L(f) + \ell n_1 L(h) \preceq n_2^2 L(f) + n_1 n_2 L(h) \preceq n_1 n_2 \{L(f) + L(h)\} \sim n_1 n_2 L(fh)$. Step (6) is executed ℓ-1 times for each execution of step (5), hence $N_6 \preceq (\ell-1) n_1 n_2 L(fh) \preceq n_1 n_2^2 L(fh)$. $N_9 \preceq \ell-1 \preceq n_2$, while $N_8 \preceq N_7 + 2N_9 \preceq n_1 n_2 L(fh)$.

In step (2), it follows from [COAL69] that $t_{CPMOD}(p,A) + t_{CPMOD}(p,B) \preceq n_1 k L(f) + n_2 k L(h) \preceq kn_1 L(fh)$. Hence $T_2 \preceq n_1 n_2 L(fh)\, kn_1 L(fh) = kn_1^2 n_2 L(fh)^2$. In step (3), it follows from Theorem 5.2.12 that $t_{CSVRS2}(p,\hat{A},\hat{B}) \preceq k^2 n_1^2 n_2 + kn_1^2 n_2^2$. This implies that $T_3 \preceq n_1 n_2 L(fh)(k^2 n_1^2 n_2 + kn_1^2 n_2^2) = kn_1^2 n_2 L(fh)(kn_1 n_2 + n_1 n_2^2)$. In step (4), by Theorem 2.2.4, $t_{LEXORD}(d,\hat{d}) \preceq \ell \preceq n_2$ and therefore $T_4 + T_5 \preceq n_1 n_2^2 L(fh) \preceq n_1^2 n_2 L(fh)$. Now $t_6 \sim 1$ so $T_6 \preceq N_6 \preceq n_1 n_2^2 L(fh) \preceq n_1^2 n_2 L(fh)$.

Note that $E \preceq E_{j+1}$ for some j, and hence $L(E) \preceq (n_2 - n_{j+1}) L(f) + (n_1 - n_{j+1}) L(h) \preceq n_1\{L(f) + L(h)\} \sim n_1 L(fh)$. Therefore, in

step (7), from [COL73d], $t_{IMFI}(M,p) \preceq L(M) \preceq L(E) \preceq n_1 L(fh)$. Hence $T_7 \preceq n_1 n_2 L(fh)\, n_1 L(fh) = n_1^2 n_2 L(fh)^2$. In step (8), $t_{ICOMP}(M,E) \preceq L(E) \preceq n_1 L(fh)$. From [COL73d], $t_{IPOWER}(c,n_2-n_3-1) \preceq (n_2-n_3)^2 L(c)^2 \preceq n_2^2 L(fh)^2$, and $t_{IPROD}(E,u) \preceq L(E)L(u) \preceq n_1 n_2 L(fh)^2$. IPOWER (c,$n_2$-$n_3$-1) and IPROD(E,u) are executed only when t = 0, hence $T_8 \preceq n_1 n_2 L(fh)\, n_1 L(fh) + n_2 n_1 n_2 L(fh)^2 \preceq n_1^2 n_2 L(fh)^2$. In step (9), $t_{IPROD}(E,c) \preceq L(E)L(c) \preceq n_1 L(fh)\, L(fh) = n_1 L(fh)^2$; thus $T_9 \preceq n_1 n_2 L(fh)^2$. ∎

Having constructed a standard modular representation, L = (L_3, L_4,...,L_{h+1}), for S(A,B) = (S_1,S_2,...,S_{h+1}), the next question we face is the computation of any coefficient of S_i from L_i. An algorithm to do this will need a sub-algorithm for computing any coefficient of $\varphi_p(S_i)$ from a standard value representation of $\varphi_p(S_i)$. We first present this sub-algorithm, CICVR, and then the main algorithm, PCRCMR. The former is a straight-forward application of interpolation using CPINTI from [COL72]. The latter algorithm is a straightforward application of the Chinese remainder algorithm using CPCRA, also from [COL72].

Algorithm 5.2.4. c = CICVR(p,L,n)

(Congruence, Interpolation of a Coefficient from a Value Representation)

p is a prime number, L is a value representation of some non-zero element, S, of GF(p)[y,x], and n is a FORTRAN integer such that $0 \le n \le \deg(S)$. c is the coefficient of the term of S with exponent n, an element of GF(p)[y].

Description

(1) [Initialize.] $T \leftarrow L$; $c \leftarrow 0$; $D \leftarrow PFA(0,PFA(1,0))$; $m \leftarrow FIRST(SECOND(L)) - n$.

(2) [Obtain the coefficient of $\psi_a(S)$.] If $T = 0$, go to (4); $ADV2(a,\hat{S},T)$; for $i = 1,\ldots,m$ do, $\hat{S} \leftarrow TAIL(S)$; $c \leftarrow SECOND(\hat{S})$.

(3) [Interpolate.] $CPINTI(p,c,a,\hat{c},D,1,w)$; go to (2).

(4) [Finish-up.] Erase D; return.

Theorem 5.2.14. Let p be a prime number, L the value representation of some non-zero element, S, of $GF(p)[y,x]$, and n an integer such that $0 \leq n \leq deg(S)$. If f is the number of images of S in L and $\ell = deg(S)$, then $t_{CICVR}(p,L,n) \preceq f^2 + f(\ell-n)$.

Proof. $N_2, N_3 \sim f$. $t_2 \preceq \ell - n + 1$, so $T_2 \preceq f(\ell-n+1) \preceq f^2 + f(\ell-n)$. It follows from page 32 of [COL72] that $t_{CPINTI}(p,c,a,\hat{c},D,1) \preceq [deg(c)+1] + [deg(D)+1]$. Hence for the j^{th} execution of step (3), $t_{CPINTI}(p,c,a,\hat{c},D,1) \preceq j$. This implies that $T_3 \preceq N_3^2 \sim f^2$. ∎

Algorithm 5.2.5. $c = PCRCMR(L,n,V)$

(Polynomial, Chinese Remainder Algorithm for a Coefficient from a Modular Representation)

L is a modular representation of some non-zero element, S, of $I[y,x]$, n is a FORTRAN integer such that $0 \leq n \leq deg(S)$, and V is the list (y). c is the coefficient of the term of S with exponent n, an element of $I[y]$.

Description

(1) [Initialize.] $T \leftarrow L$; $\bar{c} \leftarrow 0$; $Q \leftarrow PFA(1,0)$.

(2) [Obtain the coefficient of $\phi_p(S)$.] If $T = 0$, go to (4); $ADV2(p,\hat{L},T)$; $\hat{c} \leftarrow CICVR(p,\hat{L},n)$.

(3) [Apply Chinese remainder algorithm.] $q \leftarrow CRECIP(p,CMOD(p,Q))$; $c \leftarrow CPCRA(Q,\bar{c},p,\hat{c},q,V)$; erase $\bar{c}$, $\hat{c}$; $\bar{c} \leftarrow c$; $IMFI(Q,p)$; go to (2).

(4) [Finish-up.] Erase Q; return.

Theorem 5.2.15. Let $L = (p_1,L_1,p_2,L_2,\ldots,p_e,L_e)$ be a modular representation of some non-zero element, S, of $I[y,x]$, n an integer such that $0 \leq n \leq \ell = deg(S)$ and V the list y. If the number of images of $\phi_{p_i}(S)$ in L_i is $\leq f$ for all i, then $t_{PCRCMR}(L,n,V) \preceq ef^2 + ef(\ell-n) + e^2 f$.

Proof. $N_2, N_3 \leq e$. By Theorem 5.2.14, $t_{CICVR}(p,\hat{L},n) \preceq f^2 + f(\ell-n)$ so $T_2 \preceq ef^2 + ef(\ell-n)$. From page 30 of [COL72] we have that the time for the i^{th} execution of CPCRA is $\preceq [deg(c)+1]\, L(|c|_\infty) \preceq fi$. This implies that $T_3 \preceq fN_3^2 \preceq e^2 f$. ∎

Suppose A^* and B^* are elements of $Q(a)[x]$. In the notation of Theorem 5.2.3, having determined r, computed S_r and then $S_{n^*_{r+1}}(A^*,B^*) = \psi_a(S_r)$, we want to compute the monic associate of $S_{n^*_{r+1}}(A^*,B^*)$. Therefore we have the following.

Suppose A is an element of $Q(a)[x]$, that is, an element of $Q[y,x]/(\Psi(y))$ representing $A^* = \psi_a(A)$. The next algorithm computes B, the monic associate of A; that is, $\psi_a(B) = B^*$ is the monic

associate of A^*. Suppose $a = ldcf(A)$ and $b = ANRECP(a)$. Then we could compute B by computing $ASPROD(A,b,0) = bA$. But if Ψ is not irreducible, then we do not have a unique representation for one. Hence $B = bA$, as an element of $Q[y,x]/(\Psi(y))$, will not be monic if $ab \neq 1$, although $\psi_\alpha(ab) = 1$ and $\psi_\alpha(B) = B^*$ is monic. Instead of doing this, we have the algorithm APMONA which yields a monic element of $Q[y,x]/(\Psi(y))$.

Let $n = deg(A)$. APMONA first computes $A' = red(A)$ and b. Next it computes $bA' = B' = ((s_1,s_2),\bar{B}')$. Therefore $B = x^n + B' = x^n + \frac{s_1\bar{B}'}{s_2}$. Since $gcd(s_1,s_2) = 1$, $B = ((1,s_2),\bar{B})$ where $\bar{B} = s_2x^n + s_1\bar{B}'$.

Algorithm 5.2.6. $B = APMONA(A)$

(Algebraic Polynomial, Monic Associate)

A is an element of $Q(\alpha)[x]$. If $A = 0$ then $B = 0$. If $A \neq 0$ then B is the monic associate of A.

Description

(1) [Compute B'.] $B \leftarrow 0$; if $A = 0$, return; $n \leftarrow PMDEG(A)$; $A' \leftarrow PMRED(A)$; if $A' = 0$, ($t \leftarrow PFA(1,0)$; $s \leftarrow PFL(BORROW(t), PFL(BORROW(t),0))$; $V \leftarrow PVLIST(SECOND(A))$; $\bar{B} \leftarrow PORDER(t,V)$; erase t, V; $ALTER(n,TAIL(TAIL(\bar{B})))$; $B \leftarrow PFL(s,PFL(\bar{B},0))$; return); $a \leftarrow PMLDCF(A)$; $b \leftarrow ANRECP(a)$; $B' \leftarrow PMSPRD(A',b,0)$; erase a, A', b.

(2) [Compute B from $B' = ((s_1,s_2),\bar{B}')$.] $DECAP2(s,\bar{B}',B')$; $DECAP2(s_1,s_2,s)$; $c \leftarrow PFL(PVBL(\Psi), PFL(s_2,PFA(0,0)))$; $\bar{B} \leftarrow PIP(\bar{B}',s_1)$; erase s_1, $\bar{B}'$; $DECAP(X,\bar{B})$; $\bar{B} \leftarrow PFL(X,PFL(c,PFA(n,\bar{B})))$;

$t \leftarrow PFA(1,0)$; $s \leftarrow PFL(t,PFL(BORROW(s_2),0))$; $B \leftarrow PFL(s,PFL(\bar{B},0))$; return.

Theorem 5.2.16. If $A = 0$ then $t_{APMONA}(A) \sim 1$. Let A be a non-zero element of $Q(\alpha)[x]$. Let $n = deg(A)$, $d = \nu(A)$, $g = |\Psi|_1$ and $k = deg(\Psi)$. If $red(A) = 0$, then $t_{APMONA}(A) \sim 1$. If $red(A) \neq 0$ then $t_{APMONA}(A) \preceq nk^4\{L(dg)^2tk^2\}$.

Proof. Assume $A \neq 0$. If $red(A) = 0$ then $t_{APMONA}(A) \sim 1$ since $t_{PMRED}(A) \sim 1$ by Theorem 3.4.3. Assume $red(A) \neq 0$. In step (1), by Theorem 3.4.3, $t_{PMRED}(A) \preceq nk\,L(d)^2$; by Theorem 3.4.2, $t_{PMLDCF}(A) \preceq kL(d)^2$; and by Corollary 4.5.2, $t_{ANRECP}(a) \preceq k^3\{L(dg)^2+k^2\}$. By Theorem 3.3.5 and Lemma 4.5.4, $t_{PMSPRD}(A',b,0) \preceq nk^2\{kL(g)^2+L(d)^2+L(\nu(b))^2\} \preceq nk^2\{k^2(L(d)^2+L(g)^2+k^2)\} \preceq nk^4\{L(dg)^2+k^2\}$. In step (2), by Lemmas 3.3.1 and 4.5.4, $t_{PIP}(\bar{B}',s_1) \preceq nL(|\bar{B}'|_1)L(s_1) \leq nL(\nu(B'))L(s_1) \preceq n\{kL(g)+L(\nu(A'))+L(b)\}L(d) \preceq n\{kL(g)+L(d)+k(L(d)+L(g)+k)\}L(d) \preceq nk\{L(d)^2+L(g)L(d)+kL(d)\} \preceq nk\{L(d)^2+L(g)^2+k^2\} \sim nk\{L(dg)^2+k^2\}$. ∎

Lemma 5.2.1. Let A be a non-zero element of $Q(\alpha)[x]$ and $B = APMONA(A)$. Let $k = deg(\Psi)$ and $g = |\Psi|_1$. Then $\nu(B) \leq (k+1)^{2k^2}(kg^{k-1})^2\,\nu(A)^{k+1}$.

Proof. Let $a = ldcf(A)$ and $b = ANRECP(a)$. Then if $A = ((r_1,r_2),\bar{A})$, by Lemma 3.3.1 and Lemma 4.5.4, $\nu(B) \leq \nu(bA) \leq kg^{k-1}\,\nu(A)\nu(b) \leq kg^{k-1}\,\nu(A)(k+1)^{2k^2}\,g^{k-1}\,|\bar{A}|_1^{k-1}(r_2k + r_1\,|\bar{A}|_1) \leq (kg^{k-1})^2\,\nu(A)(k+1)^{2k^2}\,|\bar{A}|_1^{k-1}\,\nu(A) \leq (k+1)^{2k^2}(kg^{k-1})^2\,\nu(A)^{k+1}$. ∎

Description

(1) [Initialize.] n ← PMDEG(B); if A = 0 or PMDEG(A) < n, (L ← PFL(0,PFL(BORROW(A),0)); return); b ← PMLDCF(B); c ← ANRECP(b); erase b.

(2) [Compute $\hat{Q}$, $\hat{R}$.] B_1 ← PMRED(B); $\hat{B}_1$ ← PMSPRD(B_1,c,0); erase B_1; H ← PMQRMM(A,n,$\hat{B}_1$); erase $\hat{B}_1$; DECAP2($\bar{Q}$,$\hat{R}$,H); $\hat{Q}$ ← PMSPRD($\bar{Q}$,c,0); erase $\bar{Q}$, c.

(3) [Form Q, R.] Q ← AREP($\hat{Q}$); erase $\hat{Q}$; R ← AREP($\hat{R}$); erase $\hat{R}$; L ← PFL(Q,PFL(R,0)); return.

Theorem 4.5.10. Let A, B be elements of $Q(\alpha)[x]$ such that B is non-zero. Let m = deg(A), n = deg(B). If A = 0 or m < n then $t_{APQREM}(A,B) \sim 1$. Assume $A \neq 0$, $m \geq n$, and define $k = \deg(\Psi)$, $e = \mathrm{mep}(\Omega)$, $d = \nu(A)$, $f = \nu(B)$, $g = |\Psi|_1$. We have that $t_{APQREM}(A,B) \preceq k^5(m-n+1)^4(m+1)(k^2+L(g)^2+L(f)^2) + k^3(m-n+1)^2(m+1)L(d)^2 + k^2(m-n+1)L(e)^2$.

Proof. Assume $A \neq 0$, $m \geq n$. By Lemma 4.5.4,

(1) $L(\nu(c))^2 \preceq k^2\{k^2 + L(g)^2 + L(f)^2\}$.

By Lemma 3.3.1, $L(\nu(\hat{B}_1))^2 = L(\nu(\mathrm{cred}(B)))^2 \preceq k^2L(g)^2 + L(\nu(c))^2 + L(f)^2$. Hence

(2) $L(\nu(\hat{B}_1)) \preceq k^2\{k^2 + L(g)^2 + L(f)^2\}$.

By Theorem 3.5.3, $L(\nu(\bar{Q}))^2 \preceq (m-n+1)^2\{k^2L(g)^2 + L(\nu(\hat{B}_1))^2\} + L(d)^2$. Therefore

(3) $L(\nu(\bar{Q}))^2 \preceq (m-n+1)^2k^2\{k^2+L(g)^2+L(f)^2\} + L(d)^2$.

Similarly, by Theorem 3.5.3,

(4) $L(\nu(\hat{R}))^2 \preceq (m-n+1)^2k^2\{k^2+L(g)^2+L(f)^2\} + L(d)^2$.

By Lemma 3.3.1, $L(\nu(\hat{Q}))^2 = L(\nu(\bar{Q}c))^2 \preceq k^2L(g)^2 + L(\nu(\bar{Q}))^2 + L(\nu(c))^2$. Hence from (1) and (3) we have

(5) $L(\nu(\hat{Q}))^2 \preceq (m-n+1)^2k^2\{k^2+L(g)^2+L(f)^2\} + L(d)^2$.

In step (1), by Theorem 3.4.2, $t_{PMLDCF}(B) \preceq kL(f)^2$, while by Corollary 4.5.2, $t_{ANRECP}(b) \preceq k^5 + k^3L(f)^2 + k^3L(g)^2 = k^3(k^2+L(g)^2+L(f)^2)$.

In step (2), by Theorem 3.4.3, $t_{PMRED}(B) \preceq (n+1)kL(f)^2 \leq (m+1)kL(f)^2$. By Theorem 3.3.5 and (1), $t_{PMSPRD}(B_1,c,0) \preceq (n+1)k^2\{kL(g)^2+L(\nu(c))^2+L(f)^2\} \preceq (m+1)k^4\{k^2+L(g)^2+L(f)^2\}$. By Theorem 3.5.2 and (2), $t_{PMQRMM}(A,n,\hat{B}_1) \preceq k^2(m-n+1)^2(m+1)\{(m-n+1)^2(k^2L(g)^2+L(\nu(\hat{B}_1))^2)+L(d)^2\} \preceq k^2(m-n+1)^2(m+1)\{(m-n+1)^2k^2(k^2+L(g)^2+L(f)^2)+L(d)^2\} = k^4(m-n+1)^4(m+1)(k^2+L(g)^2+L(f)^2) + k^2(m-n+1)^2(m+1)L(d)^2$. By Theorem 3.3.5, (1), and (3), $t_{PMSPRD}(\bar{Q},c,0) \preceq (m-n+1)k^2\{kL(g)^2+L(\nu(\bar{Q}))^2+L(\nu(c))^2\} \preceq (m-n+1)k^2\{(m-n+1)^2k^2(k^2+L(g)^2+L(f)^2)+L(d)^2\} = (m-n+1)^3k^4(k^2+L(g)^2+L(f)^2) + (m-n+1)k^2L(d)^2$.

By Theorem 4.2.3 and (5), $t_{AREP}(\hat{Q}) \preceq (m-n+1)k^2\{kL(g)^2+kL(\nu(\hat{Q}))^2+kL(e)+L(e)^2\} \preceq (m-n+1)k^2\{(m-n+1)^2k^3(k^2+L(g)^2+L(f)^2)+kL(d)^2+kL(e)+L(e)^2 \preceq k^5(m-n+1)^3(k^2+L(g)^2+L(f)^2)+k^3(m-n+1)L(d)^2+k^2(m-n+1)L(e)^2$.

By Theorem 4.2.3 and (4), $t_{AREP}(\hat{R}) \preceq (m-n+1)k^2\{kL(g)^2+kL(\nu(\hat{R}))^2+kL(e)+L(e)^2\} \preceq k^5(m-n+1)^3(k^2+L(g)^2+L(f)^2) + k^3(m-n+1)L(d)^2 + k^2(m-n+1)L(e)^2$. ∎

Algorithm 4.5.4. R = APREM(A,B)

(Algebraic Polynomial, Remainder)

A, B are elements of $Q(\alpha)[x]$ such that B is non-zero. R = rem(A,B).

Description

(1) L ← APQREM(A,B); DECAP2(Q,R,L); erase Q; return.

Theorem 4.5.11. Let A, B be elements of $Q(\alpha)[x]$ such that B is non-zero. Let m = deg(A), n = deg(B). If A = 0 or m < n then $t_{APREM}(A,B) \sim 1$. Assume $A \neq 0$, $m \geq n$, and define k = deg(Ψ), e = mep(Ω), d = ν(A), f = ν(B), g = $|\Psi|_1$. We have that $t_{APREM}(A,B) \preceq k^5(m-n+1)^4(m+1)(k^2+L(g)^2+L(f)^2) + k^3(m-n+1)^2(m+1)L(d)^2 + k^2(m-n+1)L(e)^2$.

Proof. This follows from Theorem 4.5.10. ∎

Algorithm 4.5.5. q = AQ(a,b)

(Algebraic Quotient)

a,b are both elements of $Q(\alpha)$ or both elements of $Q(\alpha)[x]$, with $b \neq 0$ and b|a. q = a/b.

Description

(1) q ← 0; if a = 0, return; if TYPE(SECOND(SECOND(b))) = 1, go to (2); c ← ANRECP(b); q ← APROD(a,c); erase c; return.

(2) L ← APQREM(a,b); DECAP2(q,r,L); return.

Theorem 4.5.12. Let a,b be elements of $Q(\alpha)$ such that b is non-zero. If a = 0, then $t_{AQ}(a,b) \sim 1$. Assume $a \neq 0$. If k = deg(Ψ), d = ν(a), f = ν(b), g = $|\Psi|_1$ then $t_{AQ}(a,b) \preceq k^4(k^2+L(g)^2 + L(f)^2) + k^2L(d)^2$.

Proof. By Corollary 4.5.2, $t_{ANRECP}(b) \preceq k^3(k^2+L(g)^2+L(f)^2)$. By Lemma 4.5.4, $L(\nu(c)) \preceq k\{k+L(g)+L(f)\}$. Therefore, by Theorem 4.4.1, $t_{APROD}(a,c) \preceq k^2\{kL(g)^2+L(d)^2+L(\nu(c))^2\} \preceq k^2\{k^2(k^2+L(g)^2+L(f)^2) + L(d)^2\} = k^4(k^2+L(g)^2+L(f)^2) + k^2L(d)^2$. ∎

Theorem 4.5.13. Let a,b be elements of $Q(\alpha)[x]$ such that b is non-zero. Let m = deg(a), n = deg(b). If a = 0 or m < n then $t_{AQ}(a,b) \sim 1$. Assume $a \neq 0$, $m \geq n$, and define k = deg(Ψ), e = mep(Ω), d = ν(a), f = ν(b), g = $|\Psi|_1$. We have that $t_{AQ}(a,b) \preceq k^5(m-n+1)^4(m+1)(k^2+L(g)^2+L(f)^2) + k^3(m-n+1)^2(m+1)L(d)^2 + k^2(m-n+1)L(e)^2$.

Proof. This follows from Theorem 4.5.10. ∎

Algorithm 4.5.6. C = ASQ(A,b)

(Algebraic, Special Quotient)

A is an element of $Q(\alpha)[x]$, b is a non-zero element of $Q(\alpha)$. $C = Ab^{-1}$.

Description

(1) C ← 0; if A = 0, return; d ← ANRECP(b); C ← PMSPRD(A,d,0); erase d; return.

Theorem 4.5.14. Let A be an element of $Q(\alpha)[x]$, let b be a non-zero element of $Q(\alpha)$. If $A = 0$ then $t_{ASQ}(A,b) \sim 1$. Assume $A \neq 0$ and define $h = \nu(A)$, $f = \nu(b)$, $g = |\Psi|_1$, $k = \deg(\Psi)$, $n = \deg(A)$. We have that $t_{ASQ}(A,b) \preceq (n+1)k^2\{k^2(k^2+L(g)^2+L(f)^2)+L(h)^2\}$.

Proof. Assume $A \neq 0$. By Corollary 4.5.2, $t_{ANRECP}(b) \preceq k^3(k^2+L(g)^2 + L(f)^2)$. By Lemma 4.5.4, $L(\nu(d)) \preceq k(k+L(g)+L(f))$. Therefore, by Theorem 3.3.5, $t_{PMSPRD}(A,d,0) \preceq (n+1)k^2\{kL(g)^2+L(h)^2+L(\nu(d))^2\} \preceq (n+1)k^2\{k^2(k^2+L(g)^2+L(f)^2) + L(h)^2\}$. ∎

CHAPTER 5: GCD CALCULATIONS

Section 5.1. Introduction

In this chapter we will present two algorithms for computing the greatest common divisor of two elements of $Q(\alpha)[x]$. But first, in this section we will discuss briefly the application of the generalized Brown-Collins g.c.d. algorithm to elements of $Q(\alpha)[x]$. This algorithm can be used when it is known that Ψ is irreducible and $I[\theta]$, for a certain algebraic integer θ in $Q(\alpha)$, is a u.f.d. Let us now discuss the algorithm and show why these properties are necessary for our application. We begin by briefly describing how it is applied to polynomials over the rational integers.

W. S. Brown ([BRO71]) and G. E. Collins ([COL72]) independently developed a g.c.d. algorithm for multivariate polynomials over the rational integers which is superior to any other known algorithm. Suppose A and B, elements of $I[x_1,\ldots,x_v]$, are the inputs. The algorithm determines that a certain finite number of prime integers must be used. For each such prime p, A and B are mapped into $GF(p)[x_1,\ldots,x_v]$ by the modular homomorphism ϕ_p. Then $C^* = \gcd(\phi_p(A), \phi_p(B))$ is computed. For all but a finite number of primes which can be detected and rejected by the algorithm, $C^* = \gcd(A,B)$ modulo p. Then the Chinese remainder algorithm for integers is applied to these images to construct C.

We now discuss the computation of C^*. Let $A^* = \phi_p(A)$ and $B^* = \phi_p(B)$. Suppose A^* and B^* are elements of $GF(p)[x]$. Since $GF(p)$ is a field, C^* is constructed by computing the natural p.r.s. of A^* and B^*. Now suppose A^* and B^* are elements of

$GF(p)[x_1,x_2]$. If $n > \deg(C^*)$, $a_1,\ldots,a_n$ are distinct elements of $GF(p)$, and one knows $C^*(a_1,x_2),\ldots,C^*(a_n,x_2)$, then interpolation can be used to compute $C^*(x_1,x_2)$. The evaluation homomorphism ψ_{a_i}, defined by $\psi_{a_i}(D(x_1,x_2)) = D(a_i,x_2)$, is used to determine $C^*(a_i,x_2)$ in the following manner. The algorithm determines a suitable n. For each a_i, $\psi_{a_i}(A^*)$ and $\psi_{a_i}(B^*)$ are computed, and then $\gcd(\psi_{a_i}(A^*), \psi_{a_i}(B^*))$ using the natural p.r.s. For all but a finite number of a_i, $\gcd(\psi_{a_i}(A^*), \psi_{a_i}(B^*)) \simeq C^*(a_i,x_2)$. These unlucky a_i's can be detected and discarded by the algorithm. In case $v > 2$, evaluation homomorphisms and interpolation are applied recursively to compute C^*.

The Chinese remainder theorem for integers and the method of interpolation sketched above, are both just special cases of a generalized Chinese remainder theorem dealing with isomorphisms of quotient rings. G. E. Collins has shown that by making use of this generalized theorem, the Brown-Collins g.c.d. algorithm can be generalized to any polynomial ring $I[x]$, where I is a g.c.d. domain and the generalized Chinese remainder theorem can be applied. This means determining an infinity of quotient rings and homomorphisms mapping I onto these quotient rings.

Suppose A and B are inputs to such an algorithm for $I[x]$. The number of homomorphisms which must be applied is determined. If ϕ is one such homomorphism, then $\phi(A)$ and $\phi(B)$ are computed. Next $\gcd(\phi(A), \phi(B))$ is computed. Just as in the specific cases mentioned above, it may be that $\gcd(\phi(A), \phi(B)) \neq \phi(\gcd(A,B))$. However, it can be shown that the number of such homomorphisms is finite

and that they can always be detected. Using the lucky homomorphisms, the Chinese remainder algorithm for I is used to compute $\gcd(A,B)$ from the homomorphic images. (Of course, the computation of $\gcd(\phi(A), \phi(B))$ may consist of another application of the Brown-Collins algorithm.)

Now in our application, this means determining a g.c.d. domain G of which $Q(\alpha)$ is the quotient field, so that every element of $Q(\alpha)[x]$ is similar to an element of $G[x]$; also, we must determine the homomorphisms and quotient rings we will need.

Toward this end, let us consider, for α a real algebraic number, the integral domains I_α and $I[\alpha]$. I_α is the ring of algebraic integers contained in $Q(\alpha)$; for discussions of algebraic integers see [HAWR45] and [POL50]. $I[\alpha]$ is the ring formed by extending the rational integers by α. Clearly, $Q(\alpha)$ is the quotient field for both $I[\alpha]$ and I_α. We now present a definition, theorem and corollary which will show us when I_α is a g.c.d. domain.

Definition 5.1.1. Let I be an integral domain such that every non-zero element of I which is not a unit can be written as a finite product of irreducible elements of I. I is then said to be a factorization domain, abbreviated f.d. If I is an f.d. and each non-zero non-unit element of I has a decomposition which is unique up to the order and associates of the irreducible elements, then I is called a unique factorization domain, abbreviated u.f.d.

Theorem 5.1.1 follows immediately from Theorem 1 of Chapter 4, Section 3 of [JAC51].

Theorem 5.1.1. Let I be an integral domain. I is a u.f.d. if and only if I is a f.d. and a g.c.d. domain.

By Theorem 7.5 of [POL50], I_α is a f.d. Therefore Corollary 5.1.1 follows easily from Theorem 5.1.1.

Corollary 5.1.1. I_α is a u.f.d. if and only if I_α is a g.c.d. domain.

By this result it suffices for I_α at least, to consider the case where a u.f.d. exists. In general, not much is known about whether I_α or $I[\alpha]$ is a g.c.d. domain for arbitrary α. However, if α is quadratic then quite a great deal can be said. By Theorem 245 of Section 14.7 of [HAWR45], if $Q(\alpha)$ is Euclidean then I_α is a u.f.d. Furthermore, in the following section of the same reference it is shown that if m is a positive, square-free integer greater than one, then $Q(\sqrt{m})$ is Euclidean if and only if $m = 2,3,5,6,7,11,13,17,19,21,29,33,37,41,57$ or 73. Hence at least for these cases we know that I_α is a g.c.d. domain. Also, it follows from Theorem 6.6 of [POL50] that if $\alpha = \sqrt{m}$, then for $m = 2,3,6,7,11$ or 19, $I[\alpha] = I_\alpha$. Hence $I[\alpha]$ is a g.c.d. domain for these cases.

On the other hand, it follows from the example in Section 14.6 of [HAWR45] that if $\alpha = \sqrt{10}$ then I_α is not a u.f.d. and hence not a g.c.d. domain. By Theorem 6.6 of [POL50], $I[\sqrt{10}] = I_{\sqrt{10}}$ so $I[\sqrt{10}]$ is not a g.c.d. domain.

We have shown that for some real algebraic numbers α, I_α is a u.f.d. and for others it is not. Even if I_α is a g.c.d. domain, it is still not quite the structure we need for G. But in some cases

there is a subset of I_α, $I[\theta]$ where θ is a certain algebraic integer, from which quotient rings and homomorphisms can be determined; hence it satisfies our conditions if it is a g.c.d. domain.

Let Ψ be the primitive irreducible integral polynomial satisfied by α, $k = \deg(\Psi)$ and $\ell = \mathrm{ldcf}(\Psi)$. Now consider $\theta = \ell\alpha$. It is easy to see that θ is a root of the monic irreducible integral polynomial $\hat{\Psi}(x) = \ell^{k-1}\Psi(\frac{x}{\ell})$. Hence θ is an algebraic integer. It also follows readily that $Q(\theta) = Q(\alpha)$. This implies that $Q(\alpha)$ is the quotient field of $I[\theta]$. We will assume that $I[\theta]$ is a g.c.d. domain.

Note that $I[\theta] \subseteq I_\alpha$. Also, if α is an algebraic integer then $I[\theta]=I_\alpha$. Then it follows from above that if $\alpha = \sqrt{m}$ and $m = 2,3,6,7,11$ or 19 then $I[\theta] = I_\alpha$ and $I[\theta]$ is a g.c.d. domain.

We now need to make just one further assumption, this time about $\hat{\Psi}$. We assume that for an infinite number of primes p, $\phi_p(\hat{\Psi})$ is irreducible. It should be noted that there exist polynomials which are reducible modulo p for infinitely many primes, even though they are irreducible over the rational integers. In fact, in [KNU69] an irreducible polynomial is presented which is reducible modulo p for every prime p. We are now ready to present the desired quotient rings and homomorphisms.

Let $I[y]_{\hat{\Psi}}$ denote the integral domain consisting of all univariate integral polynomials of degree less than $k = \deg(\hat{\Psi})$, where addition and multiplication are done modulo $\hat{\Psi}$. Since $\hat{\Psi}$ is irreducible and monic, it is clear that $I[\theta]$ is isomorphic to $I[y]_{\hat{\Psi}}$. Also, let p be a prime such that $\hat{\Psi}^* = \phi_p(\hat{\Psi})$ is irreducible and let

I_p denote the integers modulo p. Let $I_p[y]_{\hat{\Psi}^*}$ denote the set of all univariate polynomials over I_p of degree less than k, where addition and multiplication are done modulo $\hat{\Psi}^*$. It is well known that $I_p[y]_{\hat{\Psi}^*}$ is a finite field of cardinality p^k, and we denote this field by $GF(p^k)$. Finally, we let $\hat{\phi}_p$ denote the obvious homomorphism from $I[y]_{\hat{\Psi}}$ onto $I_p[y]_{\hat{\Psi}^*}$.

Since there are infinitely many primes p such that $\phi_p(\hat{\Psi})$ is irreducible, we are assured of having an infinity of quotient rings and corresponding homomorphisms. In the remainder of this discussion we will assume that all the primes used enjoy this property.

With the quotient rings and homomorphisms we have established, an application of the generalized Brown-Collins g.c.d. algorithm is possible. We begin with A and B, elements of $Q(\alpha)[x]$. In light of the above discussion, it suffices to consider A and B as elements of $I[\theta][x]$. We then substitute the indeterminate y for occurrences of θ, and consider A and B as elements of $I[y]_{\hat{\Psi}}[x]$. The algorithm determines the number of homomorphisms which must be applied. If $\hat{\phi}_p$ is one such homomorphism, $\hat{\phi}_p(A)$ and $\hat{\phi}_p(B)$ are computed, and then $\gcd(\hat{\phi}_p(A), \hat{\phi}_p(B))$ is determined. Since $GF(p^k)$ is a finite field and every element is of bounded length, the natural p.r.s. is used to obtain the g.c.d. If $\hat{\phi}_p$ is unlucky, we discard the result, otherwise, $\gcd(\hat{\phi}_p(A), \hat{\phi}_p(B)) \approx \hat{\phi}_p(\gcd(A,B))$. In the latter case we apply the Chinese remainder algorithm for $I[y]_{\hat{\Psi}}$. When enough homomorphisms have been used, we have the polynomial $C(y,x) \approx \gcd(A(y,x), B(y,x))$, and hence $C(\theta,x) \approx \gcd(A(\theta,x), B(\theta,x))$.

We note that $\alpha = \sqrt{2}$ is a case for which all of the above hypotheses are satisfied. In this case $\theta = \alpha$ and $\Psi(y) = \hat{\Psi}(y) = y^2 - 2$. By Theorem 95 of Section 6.11 of [HAWR45], 2 is a quadratic nonresidue of all primes p of the form $p = 8n \pm 3$, of which their are an infinite number. Hence for such primes $\hat{\Psi}^*$ is irreducible.

Section 5.2. An Algorithm Using Subresultants

In this section we are going to apply the theory of subresultants to obtain a g.c.d. algorithm for elements of $Q(\alpha)[x]$. We have discussed subresultants before; in particular, some references, definitions and important results about them appear in Section 1.4 beginning with Definition 1.4.11. The reader might want to review some of this material before proceeding further.

Suppose A and B are elements of $Q(\alpha)[x]$. We are not going to compute the subresultants of A and B. Instead, if $\bar{A} = \text{polp}(A)$ and $\bar{B} = \text{polp}(B)$, we will construct a subresultant p.r.s. of $\bar{A}$ and $\bar{B}$ considered as elements of $I[y,x]$. We can then apply the homomorphism ψ_α, which maps $I[y,x]$ onto $Q(\alpha)[x]$, to obtain $\gcd(A,B)$.

The reason we take this approach follows from Section 5.1. We were unable to apply the generalized Brown-Collins g.c.d. algorithm in every algebraic extension field; in particular we could not find a way to apply it when $\alpha = \sqrt{10}$. Consequently, we cannot always use the efficient method of modular homomorphisms in $Q(\alpha)[x]$ which we described in that section. However, if we perform our calculations in $I[y,x]$ and then apply ψ_α, we can make use of modular homomorphisms. Before explaining how we accomplish this, we present two general theorems which we will use numerous times in this section.

Let R, R^* be integral domains and let ϕ be a homomorphism from R to R^*. Let ϕ also refer to the usual induced homomorphism of ϕ from $R[x]$ to $R^*[x]$. Let A, B be elements of $R[x]$ such that $m = \deg(A)$, $n = \deg(B) > 0$. Let $A^* = \phi(A)$, $B^* = \phi(B)$ and assume $m = \deg(A^*)$, $n = \deg(B^*)$.

Theorem 5.2.1. For j such that $0 \le j \le \min(m,n)$, $\phi(S_j(A,B)) = S_j(A^*,B^*)$, where S_j is as defined in Definition 1.4.12.

Proof. Define M, M_j, $M_{j,i}$, $0 \le i \le j$, for A,B as in Definition 1.4.12. Define M^*, M_j^*, $M_{j,i}^*$, $0 \le i \le j$ in a similar manner for A^*, B^*.

For any matrix $N = (n_{k,\ell})$, define $\phi(N) = (\phi(n_{k,\ell}))$. Then $\phi(M) = M^*$ since $\deg(A^*) = m$ and $\deg(B^*) = n$. By the definition of M_j, M_j^*, this implies that $\phi(M_j) = M_j^*$ and hence $\phi(M_{j,i}) = M_{j,i}^*$ for $0 \le i \le j$. Therefore, $\phi(S_j(A,B)) = \phi(\sum_{i=0}^{j} \det(M_{j,i})x^i) = \sum_{i=0}^{j} \phi(\det(M_{j,i}))x^i = \sum_{i=0}^{j} \det(\phi(M_{j,i}))x^i = \sum_{i=0}^{j} \det(M_{j,i}^*)x^i = S_j(A^*,B^*)$. ∎

Theorem 5.2.2. Assume $m \ge n$. Let $n_1,n_2,\ldots,n_{h+1}$ be the degree sequence of A,B. Let $n_1^*,n_2^*,\ldots,n_{\ell+1}^*$ be the degree sequence of A^*, B^*. Then each n_i^* is some n_j. Also, $\ell \le h$.

Proof. Let $A_1 = A$, $A_2 = B,\ldots,A_{h+1}$, $A_{h+2} = 0$ be a p.r.s. for A,B, and $A_1^* = A^*$, $A_2^* = B^*,\ldots,A_{\ell+1}^*$, $A_{\ell+2}^* = 0$ a p.r.s. for A^*, B^*. By hypothesis, $n_1^* = n_1$ and $n_2^* = n_2$. Let $3 \le i \le \ell + 1$. By (1) of Theorem 1.4.2,

$$S_{n_i^*}(A^*,B^*) \sim A_i^* . \tag{1}$$

By Theorem 5.2.1,

(2) $$\phi(S_{n_i^*}(A,B)) = S_{n_i^*}(A^*,B^*) .$$

By (1) and (2),

(3) $$\phi(S_{n_i^*}(A,B)) \approx A_i^* .$$

By (3), $\deg(S_{n_i^*}(A,B)) \geq \deg(\phi(S_{n_i^*}(A,B))) = \deg(A_i^*) = n_i^*$. But $\deg(S_{n_i^*}(A,B)) \leq n_i^*$ by definition, so

(4) $$\deg(S_{n_i^*}(A,B)) = n_i^* .$$

By (3), $\phi(S_{n_i^*}(A,B)) \neq 0$ and hence

(5) $$S_{n_i^*}(A,B) \neq 0 .$$

By (5) and part (3) of Theorem 1.4.2, there exists a j such that

(6) $$n_i^* = n_j \quad \text{or} \quad n_i^* = n_{j-1} - 1 .$$

By part (2) of Theorem 1.4.2,

(7) $$S_{n_{j-1}-1}(A,B) \approx A_j .$$

Assume $n_i^* = n_{j-1} - 1$. Then by (7),

(8) $$S_{n_i^*}(A,B) \approx A_j .$$

By (4) and (8),

(9) $$n_i^* = n_j .$$

Thus (9) holds under the hypothesis $n_i^* = n_{j-1} - 1$ and hence by (6), (9) holds in any case. This completes the proof of the first part of the theorem, and the second part now follows by observing that the ℓ-element set $\{n_2^*,\ldots,n_{\ell+1}^*\}$ is contained in the h-element set $\{n_2,\ldots,n_{h+1}\}$. ∎

Definition 5.2.1. Let R be an integral domain and A and B elements of $R[x]$ of positive degree. Then we define $\underline{S(A,B)}$ to be the subresultant p.r.s. of A and B of the second kind.

Let A^* and B^* be elements of $Q(\alpha)[x]$. Assume $\deg(A^*) \geq \deg(B^*) > 0$. Let A and B stand for $\mathrm{polp}(A^*)$ and $\mathrm{polp}(B^*)$. We wish to consider A and B as elements of $I[y,x]$. Let ψ_α be the evaluation homomorphism mapping $I[y,x]$ onto $Q(\alpha)[x]$. Since $\psi_\alpha(A) \simeq A^*$ and $\psi_\alpha(B) \simeq B^*$, $\gcd(A^*,B^*) = \gcd(\psi_\alpha(A), \psi_\alpha(B))$, and by our conventions about representations of elements of $Q(\alpha)[x]$, Theorem 5.2.1 and Theorem 5.2.2 obviously apply.

Without loss of generality let us assume that $\psi_\alpha(A) = A^*$ and $\psi_\alpha(B) = B^*$. Let $d = (n_1,n_2,\ldots,n_{h+1})$ be the degree sequence of A and B (remember that d is determined by a p.r.s. over the integers). Let $d^* = (n_1^*,n_2^*,\ldots,n_{\ell+1}^*)$ be the degree sequence of A^* and B^*.

By Theorem 5.2.2, let $n_{\ell+1}^* = n_r$. By Theorem 5.2.1, $\psi_\alpha(S_{n_r}(A,B)) = S_{n_r}(A^*,B^*) = S_{n_{\ell+1}^*}(A^*,B^*)$. Hence $\deg(\psi_\alpha(S_{n_r}(A,B))) = \deg(S_{n_{\ell+1}^*}(A^*,B^*)) = n_{\ell+1}^*$. By part (1) of Theorem 1.4.2, $S_{n_r}(A,B) \approx A_r$ and therefore $\deg(S_{n_r}(A,B)) = n_r = n_{\ell+1}^*$. This implies that

(10) $$\psi_\alpha(\mathrm{ldcf}(S_{n_r}(A,B))) \neq 0.$$

Now suppose $r < j \leq h+1$. We know that $\psi_\alpha(S_{n_j}(A,B)) = S_{n_j}(A^*,B^*)$. But since $n_j < n_r = n^*_{l+1}$, part (3) of Theorem 1.4.2 implies that

$$\psi_\alpha(S_{n_j}(A,B)) = 0, \quad r < j \leq h+1. \tag{11}$$

Hence,

$$\psi_\alpha(\mathrm{ldcf}(S_{n_j}(A,B))) = 0, \quad r < j \leq h+1. \tag{12}$$

We have proved the following theorem.

Theorem 5.2.3. Let $S(A,B) = (S_1,S_2,\ldots,S_{h+1})$ and $c_j = \mathrm{ldcf}(S_j)$, $1 \leq j \leq h+1$. Let r be the greatest integer such that $\psi_\alpha(c_r) \neq 0$. Then $n_r = n^*_{l+1}$ and $\psi_\alpha(S_r) \simeq \gcd(A^*,B^*)$.

We could apply Theorem 5.2.3 in a straight-forward manner. That is, compute $S_1,S_2,\ldots,S_{h+1}$ and then apply ψ_α to the leading coefficient of each S_i in order of least degree. After we have determined r in this way we map S_r into $Q(\alpha)[x]$ and compute its monic associate. However, if we consider how we will construct the S_i, we see that we can rather simply improve the efficiency of our algorithm.

The use of modular homomorphisms and the Chinese remainder algorithm for integers is the best known method for computing subresultants of multivariate integral polynomials. Recall that we used this technique in Section 4.5 for computing reciprocals in $Q(\alpha)$; we also discussed this method briefly in Section 5.1. Also, it is applied to g.c.d. calculations in [BRO71] and [COL72] and

resultant calculations in [COL71e]. Briefly, the method as we will use it consists of the following.

We determine a bound on $|S_i|_\infty$, $3 \leq i \leq h+1$. These bounds determine that a number of distinct primes, p, will be needed to compute $S(A,B)$. For each p we compute $\phi_p(A) = \hat{A}$ and $\phi_p(B) = \hat{B}$, elements of $GF(p)[y,x]$. Then we compute $S(\hat{A},\hat{B}) = (\hat{S}_1,\hat{S}_2,\ldots,\hat{S}_{k+1})$. Let $\hat{d}$ be the degree sequence of $\hat{A}$ and $\hat{B}$. For all but a finite number of primes which we can detect and discard, $\hat{d} = d$ and $\hat{S}_i = \phi_p(S_i)$. Thus the Chinese remainder algorithm for integers can be applied to the number of these images needed to compute S_i.

From the next theorem we obtain the bound we need for $|S_i|_\infty$.

Theorem 5.2.4. Let I be an integral domain and let A and B be elements of $I[x]$ such that $n_1 = \deg(A)$, $n_2 = \deg(B)$, and $n_1,n_2 > 0$. If η is a semi-norm for $I[x]$, where semi-norm is defined in section 2 of [COHO73], and if c is any coefficient of $S_j(A,B)$, $0 \leq j \leq \min(n_1,n_2)$, then $\eta(C) \leq \eta(A)^{n_2-j}\,\eta(B)^{n_1-j}$.

Proof. $S_j(A,B) = \sum_{i=0}^{j} \det(M_{j,i})x^i$. By Theorem 1 of [COHO73], $\eta(\det(M_{j,i})) \leq \prod_{k=1}^{n_1+n_2-2j} \eta(M_{j,i,k})$ where $M_{j,i,k}$ is the k^{th} row of $M_{j,i}$. $M_{j,i}$ consists of n_1-j rows of B coefficients and n_2-j rows of A coefficients, hence $\eta(\det(M_{j,i})) \leq \eta(A)^{n_2-j}\,\eta(B)^{n_1-j}$. ∎

Note that Theorem 5.2.4 implies that $|S_i|_\infty \leq |A|_1^{n_2-n_i}\,|B|_1^{n_1-n_i}$. Therefore, in light of the SAC-1 implementation of the Chinese re-

mainder algorithm for integers (see Theorem 1.4.7 and the following material), to compute S_i we will need e_i distinct primes $p_1,\ldots,p_{e_i}$, where $\prod_{k=1}^{e_i} p_k > 2|A|_1^{n_2-n_i}|B|_1^{n_1-n_i}$ and $\prod_{k=1}^{e_i-1} p_k \le 2|A|_1^{n_2-n_i}|B|_1^{n_1-n_i}$. Clearly, $e_3 \le e_4 \le \cdots \le e_{h+1}$.

Let us now attempt to improve our method. For the sake of simplicity let us assume for the time being that for any prime we use, $\hat{d} = d$. The first subresultant we want to compute is S_{h+1}, so we need $\phi_{p_k}(S_{h+1})$ for $1 \le k \le e_{h+1}$. We will later show that to compute $\phi_{p_k}(S_{h+1})$ it is necessary to compute the entire subresultant p.r.s. of $\phi_{p_k}(A)$ and $\phi_{p_k}(B)$ of the second kind. Therefore, since $e_i \le e_{h+1}$ for all i, we have $\phi_{p_k}(S_i)$ for $1 \le k \le e_i$ for all i. Hence, we could construct the entire sequence $S(A,B)$ from the same p.r.s.'s computed to determine S_{h+1}. However, given the necessary images, the application of Theorem 1.4.7 to the computation of S_i is independent of its application to S_j, where $j \ne i$. In other words, we need not apply Theorem 1.4.7 to those S_i which we do not need. Instead, we save all the images generated and apply the Chinese remainder theorem to S_{h+1}, then S_h, and so on until we find S_r. In fact, in light of Theorem 5.2.3 we need only construct $\mathrm{ldcf}(S_{h+1})$, $\mathrm{ldcf}(S_h),\ldots,\mathrm{ldcf}(S_{r+1})$ and S_r.

Let us summarize what we have determined to this point. For $1 \le k \le e_{h+1}$, let $S(\phi_{p_k}(A), \phi_{p_k}(B)) = (\hat{s}_1^{(p_k)}, \hat{s}_2^{(p_k)},\ldots,\hat{s}_{h+1}^{(p_k)})$. Then let

$$\hat{L}_j = (p_1, \hat{s}_j^{(p_1)}, p_2, \hat{s}_j^{(p_2)},\ldots,p_{e_j}, \hat{s}_j^{(p_{e_j})}), \quad \text{for } 3 \le j \le h+1. \tag{13}$$

$\hat{L}_j$ contains all the data necessary to compute S_j, and hence any coefficient of S_j, by using Theorem 1.4.7. Our procedure, then, consists of constructing $\hat{L}_3,\hat{L}_4,\ldots,\hat{L}_{h+1}$. Then, beginning with $j = h+1$, we compute $c_j = \mathrm{ldcf}(S_j)$ from $\hat{L}_j$. Next we determine whether $\psi_\alpha(c_j) = 0$. If $\psi_\alpha(c_j) = 0$ we decrement j and repeat. Otherwise we have determined r, so we compute S_r from $\hat{L}_r$. Then we compute $\psi_\alpha(S_r) = S_{n_{\ell+1}^*}(A^*,B^*) \simeq \gcd(A^*,B^*)$.

Up to this point we have not considered the computation of $S(\phi_{p_k}(A), \phi_{p_k}(B))$. If we do so, we will see that we can again improve the efficiency of our implementation of Theorem 5.2.3.

In general, the question we wish to address is the computation of $S(A,B)$, where A and B are elements of $GF(p)[y,x]$ of positive degree. Assume for simplicity that $\deg(A) \ge \deg(B)$. The most efficient method for the computation of subresultants in $GF(p)[y,x]$ lies in the application of evaluation homomorphisms and interpolation. Recall that we discussed this technique briefly in Section 5.1. Also, it is applied to g.c.d. calculations in [BRO71] and [COL72] and resultant calculations in [COL71e]. Section 4.6.4 of [KNU69] also mentions its application.

The method of computing polynomials by interpolation is well known. Let $C(y,x)$ be an element of $GF(p)[y,x]$; n an integer such that $n > \partial_y(C)$; $a_1,a_2,\ldots,a_n$ distinct elements of $GF(p)$; and suppose we know $C(a_1,x), C(a_2,x),\ldots,C(a_n,x)$. Then by interpolation we can compute the coefficients of C, univariate polynomials in y, and hence $C(y,x)$. The method as we apply it consists of the following.

Let d be the degree sequence of A and B, and $S(A,B) = (S_1,S_2,\ldots,S_{h+1})$. We determine a bound on $\partial_y(S_i)$, $3 \le i \le h+1$. These bounds determine the number of distinct elements, a, of $GF(p)$ which will be needed to compute S_i. For each a we use an evaluation homomorphism ψ_a to compute $\psi_a(A) = \hat{A}$ and $\psi_a(B) = \hat{B}$, elements of $GF(p)[x]$. Then we compute $S(\hat{A},\hat{B}) = (\hat{S}_1,\hat{S}_2,\ldots,\hat{S}_{h+1})$. We compute the natural p.r.s. of $\hat{A}$ and $\hat{B}$ and apply Theorem 1.4.2 to do this. Let $\hat{d}$ be the degree sequence of $\hat{A}$ and $\hat{B}$. For all but a finite number of elements of $GF(p)$ which we can detect and discard, $\hat{d} = d$ and $\hat{S}_i = \psi_a(S_i)$. Thus interpolation can be applied to the number of these images needed to compute S_i.

From the next theorem we obtain the bound we need for $\partial_y(S_i)$.

Theorem 5.2.5. Let I be an integral domain and let A and B be elements of $I[y,x]$ such that $\partial(A) = (k_1,n_1)$, $\partial(B) = (k_2,n_2)$, and $n_1, n_2 > 0$. Then $\partial_y(S_\ell(A,B)) \le (n_2-\ell)k_1+(n_1-\ell)k_2$ for $0 \le \ell \le \min(n_1,n_2)$.

Proof. Recall from Definition 1.4.12 that $S_\ell(A,B) = \sum_{i=0}^{\ell} \det(M_{\ell,i})x^i$. $M_{\ell,i}$ is a $n_1 + n_2 - 2\ell$ square matrix consisting of $n_2-\ell$ rows of A coefficients and $n_1-\ell$ rows of B coefficients. $\det(M_{\ell,i})$ is the sum of $(n_1 + n_2 - 2\ell)!$ polynomials in y, each of which is the product of one element from each row of $M_{\ell,i}$. Hence if t is any such polynomial, $\partial_y(t) \le (n_2-\ell)k_1 + (n_1-\ell)k_2$. ∎

Note that Theorem 5.2.5 implies that $\partial_y(S_i) \le (n_2-n_i)k_1 + (n_1-n_i)k_2$. Let $f_i = (n_2-n_i)k_1 + (n_1-n_i)k_2 + 1$. Clearly $f_3 \le f_4 \le \cdots \le f_{h+1}$. For any i, the f_i points $a_1,\ldots,a_{f_i}$ are sufficient to compute S_i.

For the sake of simplicity, let us assume for the time being that for any a, $\hat{d} = d$. The first subresultant we want to compute is S_{h+1}, and hence for each a we must compute the entire sequence $S(\psi_a(A), \psi_a(B))$. Therefore, since $f_i \le f_{h+1}$ for all i, we have $\psi_{a_j}(S_i)$ for $1 \le j \le f_i$ for all i. Hence we could construct the entire sequence $S_1, S_2,\ldots,S_{h+1}$ from the same p.r.s.'s computed to determine S_{h+1}.

We are sure that the reader has noticed the similarity between the development of the application of modular homomorphisms for elements of $I[y,x]$, and that of evaluation homomorphisms for elements of $GF(p)[y,x]$. As mentioned in Section 5.1, these are both just applications of the generalized Chinese remainder theorem. We now carry this analogy one step further by minimizing the number of applications of interpolation.

Suppose A and B are the elements of $I[y,x]$ we discussed earlier and that $p_1,p_2,\ldots,p_{e_{h+1}}$ is the corresponding list of primes. For some k, let $\hat{A} = \psi_{p_k}(A)$, $\hat{B} = \psi_{p_k}(B)$ and $S(\hat{A},\hat{B}) = (\hat{S}_1^{(p_k)},\hat{S}_2^{(p_k)},\ldots,\hat{S}_{h+1}^{(p_k)})$. Let $a_1,\ldots,a_{f_{h+1}}$ be the interpolating points discussed above, and for each i let $S(\psi_{a_i}(A), \psi_{a_i}(B)) = (\hat{S}_1^{(a_i)},\hat{S}_2^{(a_i)},\ldots,\hat{S}_{h+1}^{(a_i)})$. Then let

$$\hat{L}_{k,j} = (a_1, \hat{S}_j^{(a_1)}, a_2, \hat{S}_j^{(a_2)},\ldots,a_{f_j}, \hat{S}_j^{(a_{f_j})}) \qquad (14)$$

for $3 \le j \le h+1$.

$\hat{L}_{k,j}$ contains all the data necessary to construct $\hat{S}_j^{(p_k)}$, and hence any coefficient of $\hat{S}_j^{(p_k)}$, by interpolation. We now combine (13) and (14) to determine the structure we will actually use in our g.c.d. algorithm.

For A and B consider the list $\hat{L}_j$ described in (13). For each k, $1 \le k \le e_j$, consider $\phi_{p_k}(A)$, $\phi_{p_k}(B)$ and the list $\hat{L}_{k,j}$ for $\phi_{p_k}(A)$ and $\phi_{p_k}(B)$ described in (14). Then by substituting $\hat{L}_{k,j}$ for $\hat{S}_j^{(p_k)}$ in $\hat{L}_j$, we obtain

(15) $\quad L_j = (p_1, \hat{L}_{1,j}, p_2, \hat{L}_{2,j}, \ldots, p_{e_j}, \hat{L}_{e_j,j}), \quad 3 \le j \le h+1.$

Given the list L_j, one could interpolate over $\hat{L}_{1,j}, \hat{L}_{2,j}, \ldots, \hat{L}_{e_j,j}$ and determine the list $\hat{L}_j = (p_1, \hat{S}_1^{(p_1)}, \ldots, p_{e_j}, \hat{S}_j^{(p_{e_j})})$, and then apply Theorem 1.4.7 to $\hat{L}_j$ to compute S_j. Alternatively, the same technique could be used to compute any individual coefficient of S_i.

We now have determined the data structure we will actually use in our algorithm. To apply Theorem 5.2.3 we will first compute L_3, $L_4, \ldots, L_{h+1}$. Beginning with $j = h+1$ and L_j, we interpolate over $\hat{L}_{1,j}, \hat{L}_{2,j}, \ldots, \hat{L}_{e_j,j}$ to determine $\mathrm{ldcf}(\phi_{p_1}(S_j))$, $\mathrm{ldcf}(\phi_{p_2}(S_j)), \ldots, \mathrm{ldcf}(\phi_{p_{e_j}}(S_j))$. We then apply Theorem 1.4.7 to these images to compute $c_j = \mathrm{ldcf}(S_j)$. If $\psi_\alpha(c_j) = 0$ then $j > r$ so we decrement j and repeat this process. Otherwise $j = r$ and $n_j = n^*_{\ell+1}$, so we finish computing S_r from L_r. Then we compute $\psi_\alpha(S_r) = S_{n^*_{\ell}-1}(A^*, B^*)$, and finally $C^* = \gcd(A^*, B^*)$, the monic associate of $S_{n^*_{\ell+1}}(A^*, B^*)$.

We formalize the above discussion with the following definitions.

Definition 5.2.2. Let C be an element of GF(p)[y,x]. If $C = 0$ let $L = 0$. If $C \neq 0$ let f be an integer such that $f > \partial_y(C)$. Let $a_1, \ldots, a_f$ be distinct elements of GF(p) such that $\deg(C(y,x)) = \deg(C(a_i,x))$ for $1 \le i \le f$, and $L = (a_1, C(a_1,x), a_2, C(a_2,x), \ldots, a_f, C(a_f,x))$. Then L is called a value representation of C. Suppose A and B are elements of GF(p)[y,x] of positive degree such that $\partial(A) = (k_1, n_1)$, $\partial(B) = (k_2, n_2)$, and C is the i^{th} element of $S(A,B)$, $i \ge 3$. If $f = (n_2 - n_i)k_1 + (n_1 - n_i)k_2 + 1$ then L is called a standard value representation of C.

Definition 5.2.3. Let A and B be elements of GF(p)[y,x] of positive degree and $S(A,B) = (S_1, S_2, \ldots, S_{h+1})$. If $S(A,B) = (S_1, S_2)$, let $L = ()$. Otherwise, let L_i be a value representation of S_i for $3 \le i \le h+1$ and $L = (L_3, L_4, \ldots, L_{h+1})$. Then L is called a value representation of $S(A,B)$. If L_i is standard, $3 \le i \le h+1$, then L is called a standard value representation of $S(A,B)$.

Definition 5.2.4. Let C be an element of I[y,x]. If $C = 0$ let $L = 0$. If $C \neq 0$ suppose $p_1, \ldots, p_e$ are distinct primes such that $p_1 \nmid \mathrm{ldcf}(C)$ and $\prod_{i=1}^{e} p_i > 2|C|_\infty$. Let L_i be a value representation of $\phi_{p_1}(C)$. Let $L = (p_1, L_1, \ldots, p_e, L_e)$. L is called a modular representation of C. Let A and B be elements of I[y,x] of positive degree. Suppose C is the i^{th} element of $S(A,B)$, $i \ge 3$, $n_1 = \deg(A)$ and $n_2 = \deg(B)$. If $\prod_{i=1}^{e} p_i > 2|A|_1^{n_2 - n_i} |B|_1^{n_1 - n_i}$, $\prod_{i=2}^{e} p_i \le 2|A|_1^{n_2 - n_i} |B|_1^{n_1 - n_i}$ and L_i is standard for $1 \le i \le e$

This completes the presentation of the sub-algorithms needed. We now present the main algorithm, APGCD. The inputs are A^* and B^*, elements of $Q(\alpha)[x]$. If A^* and B^* are not both of positive degree this is detected in step (1) and the appropriate action taken. Otherwise $A = polp(A^*)$ and $B = polp(B^*)$ if $deg(A^*) \geq deg(B^*)$ while A and B are turned around if $deg(A^*) < deg(B^*)$. In step (2), PSMRS2 computes the degree sequence, $d = (n_1,n_2,\ldots,n_{h+1})$, and a standard modular representation, $L = (L_3,L_4,\ldots,L_{h+1})$, of $S(A,B) = (S_1,S_2,\ldots,S_{n+1})$. For decreasing values of j beginning with $j = h + 1$, $c = ldcf(S_j)$ is constructed by PCRCMR from L_j and n_j in step (3). Then in step (4), AZTEST is applied to c to determine whether $\psi_\alpha(c) = 0$. If $\psi_\alpha(c) = 0$ then control returns to step (3) for the next value of j. Otherwise $j = r$ so step (6) is executed. PCRCMR is called with inputs $(L_r,n_r-1),(L_r,n_r-2),\ldots,(L_r,0)$, and AZTEST is applied to each resulting polynomial. C is constructed, an element of $I[y,x]$ whose coefficients are those of S_r which do not have α as a root. Then $\hat{C}^* = S_{n^*_{\lambda+1}}(A^*,B^*) = \psi_\alpha(S_r) = \psi_\alpha(C) = PMRPOL(u,C)$, where u is the rational number one, and $C^* = APMONA(\hat{C}^*)$ are computed.

Algorithm 5.2.7. $C^* = APGCD(A^*,B^*)$

(Algebraic Polynomial, Greatest Common Divisor)

A^* and B^* are elements of $Q(\alpha)[x]$. $C^* = gcd(A^*,B^*)$. C^* is zero or a monic element of $Q(\alpha)[x]$.

Description

(1) [Initialize.] $D^* \leftarrow A^*$; $E^* \leftarrow B^*$; if $PMDEG(A^*) < PMDEG(B^*)$, $(D^* \leftarrow B^*$; $E^* \leftarrow A^*)$; if $E^* = 0$, $(C^* \leftarrow APMONA(D^*)$; return); if $PMDEG(E^*) = 0$, $(C^* \leftarrow APMONA(E^*)$; return); $A \leftarrow SECOND(D^*)$; $B \leftarrow SECOND(E^*)$

(2) [Compute a standard modular representation of $S(A,B)$.] PSMRS2 (A,B,L,d); $L \leftarrow INV(L)$; $d \leftarrow INV(d)$; $V \leftarrow PVLIST(Y)$.

(3) [Compute $c = ldcf(S_j)$ from L_j.] If $L = 0$, $(C^* \leftarrow APMONA(E^*)$; erase V, d; return); $DECAP(L_1,L)$; $DECAP(n,d)$; $c \leftarrow PCRCMR(L_1,n,V)$.

(4) [Test if $\psi_\alpha(c) = 0$.] $s \leftarrow AZTEST(c)$; if $s \neq 0$, go to (5); erase c, L_1; go to (3).

(5) [Initialize C.] $C \leftarrow PFL(PVBL(A),0)$; $C \leftarrow PFA(n,PFL(c,C))$; $n \leftarrow n - 1$.

(6) [Compute the remaining coefficients of C.] If $n = 0$, go to (7); $c \leftarrow PCRCMR(L_1,n,V)$; $s \leftarrow AZTEST(c)$; if $s \neq 0$, $C \leftarrow PFA(n,PFL(c,C))$; if $s = 0$, erase c; $n \leftarrow n - 1$; go to (6).

(7) [Obtain the proper representation.] Erase V, L, d, L_1; $C \leftarrow INV(C)$; $t \leftarrow PFA(1,0)$; $u \leftarrow PFL(t,PFL(BORROW(t),0))$; $C^* \leftarrow PMRPOL(u,C)$; erase u, C; $C^* \leftarrow APMONA(C^*)$; erase $(\hat{C}^*)$; return.

Theorem 5.2.17. Let A^* and B^* be elements of $Z(\alpha)[x]$. Let $D^* = A^*$ and $E^* = B^*$ if $deg(A^*) \geq deg(B^*)$, and $D^* = B^*$ and $E^* = A^*$ otherwise. Define $f = v(D^*)$, $g = v(E^*)$, $n_1 = deg(D^*)$, $n_2 = deg(E^*)$, $k = deg(F)$ and $e = mep(E)$. If $E^* = 0$ then $t_{APGCD}(A^*,E^*) \leq (n_1+1)k^4(L(dg)^2+e^2)$. If $E^* \neq 0$ and $n_2 = 0$,

then $t_{APGCD}(A^*,B^*) \sim 1$. If $n_2 > 0$ then $t_{APGCD}(A^*,B^*) \preceq \{k^3+n_1\}k^3n_1^2n_2L(g)^2 + \{k+n_1^3\}k^3n_2\{n_2^2L(f)^2 + n_1^2L(h)^2\} + k^2n_2L(e)^2$.

Proof. Assume $n_2 > 0$. In step (2), by Theorem 5.2.13, $t_{PSMRS2}(A,B) \preceq kn_1^2n_2L(fh)\{L(fh)+kn_1n_2 + n_1n_2^2\} \preceq k^2n_1^3n_2^3\{L(f)^2+L(h)^2\} \preceq k^2n_1^3n_2\{n_2^2L(f)^2 + n_1^2L(h)^2\}$. By Theorem 5.2.16, $t_{APMONA}(E^*) \preceq n_2k^4\{L(gh)^2+k^2\} \preceq k^6n_2 + k^4n_2L(g)^2 + k^4n_2L(h)^2$.

Let $S(A,B) = (S_1,S_2,\ldots,S_{t+1})$, $L = (L_3,L_4,\ldots,L_{t+1})$, $d = (n_1, n_2,\ldots,n_{t+1})$ and suppose $\deg(\gcd(A^*,B^*)) = m = n_r$. Also, let e_i be the number of primes in L_i and f_i the number of evaluation homomorphic images for each prime in L_i. Then by definition 5.2.2, $\partial_y(S_i) < f_i \le (n_2-n_i)(k-1)+(n_1-n_i)(k-1) + 1 < k(n_1+n_2-2n_i) + 1 \preceq k(n_1+n_2) \sim kn_1$. Hence

(18) $$\partial_y(S_i) < f_i \preceq kn_1 , \quad 3 \le i \le t+1.$$

By Definition 5.2.4, $L(|S_i|_\infty) \preceq e_i \preceq L(|A|_1^{n_2-n_i}\,|B|_1^{n_1-n_i}) \preceq (n_2-n_i)L(|A|_1) + (n_1-n_i)L(|B|_1) \preceq n_2L(f) + n_1L(h)$. Hence

(19) $$L(|S_i|_\infty) \preceq e_i \preceq n_2L(f) + n_1L(h), \quad 3 \le i \le t+1.$$

Now N_3, $N_4 = t+1-r+1 = t+2-r \le n_r+1 \le n_2$. In step (3), by Theorem 5.2.16, $t_{APMONA}(E^*) \preceq n_2k^4\{L(gh)^2+k^2\} \preceq k^6n_2 + k^4n_2L(g)^2 + k^4n_2L(h)^2$. Also, by Theorem 5.2.15 and (18), (19), $t_{PCRCMR}(L_1,n,V) \preceq e_{t+1}f_{t+1}^2 + e_{t+1}^2f_{t+1} \preceq \{n_2L(f)+n_1L(h)\}k^2n_1^2 + \{n_2^2L(f)^2+n_1^2L(h)^2\}kn_1 \preceq \{k^2n_1^2n_2+kn_1n_2^2\}L(f)^2 + \{k^2n_1^3+kn_1^3\}L(h)^2 \sim k^2n_1^2n_2L(f)^2 + k^2n_1^3L(h)^2 \preceq k^2n_1^2\{n_2L(f)^2+n_1L(h)^2\}$. Thus $T_3 \preceq k^6n_2^2 + k^4n_2^2L(g)^2 + k^4n_2^2L(h)^2 + n_1^2k^2n_2\{n_2L(f)^2+n_1L(h)^2\} \preceq k^6n_1^2L(g)^2 + k^4n_2n_1L(h)^2 + n_1^2k^2n_2\{n_2L(f)^2+n_1L(h)^2\}$.

In step (4), by Theorem 4.2.2 and (18), (19), $t_{AZTEST}(c) \preceq \max(k,\deg(c))^3L(|c|_1g)^2 + k^3L(e) + k^2L(e)^2 \preceq k^3n_1^3n_2^3\{n_2^2L(f)^2 + n_1^2L(h)^2 + L(g)^2\} + k^3L(e) + k^2L(e)^2$. Therefore, since $k^3L(e) \preceq k^4 + k^2L(e)^2$, $T_4 \preceq k^3n_1^3n_2L(g)^2 + n_1^3k^3n_2\{n_2^2L(f)^2 + n_1^2L(h)^2\} + k^4n_2 + k^2n_2L(e)^2$.

$N_6 \le m+1 = n_r+1 \le n_2$. By applying Theorem 5.2.15, Theorem 4.2.2 and (18), (19), we would obtain the same bound for step (6) as for $T_3 + T_4$. Hence $T_6 \preceq k^6n_1^2L(g)^2 + k^4n_2n_1L(h)^2 + k^3n_1^2n_2L(g)^2 + n_1^3k^3n_2\{n_2^2L(f)^2+n_1^2L(h)^2\} + k^2n_2L(e)^2$.

In step (7), by Theorem 3.3.3 and (18), (19), $t_{PMRPOL}(u,C) \preceq (n_r+1)\partial_y(C)^2\{\partial_y(C)L(g)^2 + L(|C|_1)^2\} \preceq n_2k^2n_1^2\{kn_1L(g)^2 + n_2^2L(f)^2 + n_1^2L(h)^2\} = k^3n_1^3n_2L(g)^2 + k^2n_1^2n_2\{n_2^2L(f)^2 + n_1^2L(h)^2\}$. Now by Lemma 2.5.1, $\nu_1(\hat{C}^*) \le \{\deg(C)+1\}\,g^{\partial_y(C)-k+1}\,|C|_1 \le \{\deg(C)+1\}^2\,g^{\partial_y(C)-k+1}\,|C|_\infty$. Also, $\nu_2(\hat{C}^*) \le g^{\partial_y(C)-k+1}$. Hence by (18) and (19), $L(\nu(C^*)) \preceq \{\partial_y(C)-k+1\}L(g) + L(|C|_\infty) \preceq kn_1L(g) + n_2L(f) + n_1L(h)$. This implies that by Theorem 5.2.16, $t_{APMONA}(\hat{C}^*) \preceq (m+1)k^4\{L(g\nu(C^*))^2 + k^2\} \preceq n_2k^4\{k^2n_1^2L(g)^2 + n_2^2L(f)^2 + n_1^2L(h)^2\} \preceq k^6n_1^2n_2L(g)^2 + k^4n_2\{n_2^2L(f)^2 + n_1^2L(h)^2\}$. ∎

Section 5.3. The Monic PRS Algorithm

We now present the monic p.r.s. algorithm for elements of $Q(\alpha)[x]$. It is the only algorithm in this thesis which is given without an accompanying theorem proving that it has a polynomial bounded computing time function. Indeed, the question of the existence of such a bound remains open. As an examination of the theorem which does accompany APMGCD will reveal, the problem arises in determining a bound on the length of $v(A_i)$, where A_i is an element of the p.r.s.

Now in case Ψ is the minimal polynomial of α, the evaluation homomorphism φ_α is an isomorphism. Then one can apply Theorem 1.4.2 and the theory of matrices to determine a bound on $L(v(A_i))$. This bound is a polynomial function of the degrees of A and B, the lengths of $v(A)$ and $v(B)$, $\deg(\Psi)$, and $|\Psi|_1$, where A and B are the inputs to the algorithm. But in the event Ψ is not irreducible, the representation of A_i is not unique and this approach fails. The bound determined by application of Theorem 3.5.3 and Lemma 5.2.1 will be shown to be inadequate in Lemma 5.3.1, at least in so far as this author has been able to determine.

In spite of this shortcoming, there are several very legitimate reasons for including this algorithm in this thesis. First, the question of its computing time is unsolved, and while the author himself intends to pursue it further, he presents it here in hopes of stimulating interest in the subject. Of course even if the computing time is exponential this does not preclude the possibility that the algorithm would be more efficient than the one presented in Section

5.2 for certain limited cases.

Also, the monic p.r.s. generated by this algorithm can readily be adopted to the computation of Sturm sequences, as is shown in [HEI70]. The computation of such a sequence can be used to find the roots of a polynomial over $Q(\alpha)$, an application of use to the Collins quantifier elimination algorithm.

Finally, the algorithm is noteworthy when viewed in an historical context. Prior to the improved algorithms resulting from the research of G. E. Collins and later of W. S. Brown, the monic p.r.s. algorithm was the most efficient algorithm for the computation of g.c.d.'s of polynomials over the rationals. This was shown by G. E. Collins. As such, it should be considered as a worthy bench mark with which to compare empirically other g.c.d. algorithms, where the coefficient domain is $Q(\alpha)$ instead of Q.

Algorithm 5.3.1. C = APMGCD(A,B)

(Algebraic Polynomial, Monic PRS GCD Algorithm)

A and B are elements of $Q(\alpha)[x]$. C = gcd(A,B). C is zero or a monic element of $Q(\alpha)[x]$.

Description

(1) [Initialize.] $n \leftarrow$ PMDEG(B); if A = 0 or B ≠ 0 and n = 0, (C ← APMONA(B), return); $A_1 \leftarrow$ APMONA(A); $m \leftarrow$ PMDEG(A); if B = 0 or m = 0, ($C \leftarrow A_1$; return); $A_2 \leftarrow$ APMONA(B); if m ≥ n, go to (2); $T \leftarrow A_1$; $A_1 \leftarrow A_2$; $A_2 \leftarrow T$.

(2) [Compute next element of the p.r.s.] $\hat{A}_3 \leftarrow \text{APREM}(A_1,A_2)$; $A_3 \leftarrow \text{APMONA}(\hat{A}_3)$; erase A_1, $\hat{A}_3$.

(3) [Check for completion.] If $A_3 = 0$, ($C \leftarrow A_2$; return); if $\text{PMDEG}(A_3) = 0$, (erase A_2; $C \leftarrow A_3$; return); $A_1 \leftarrow A_2$; $A_2 \leftarrow A_3$; go to (2).

In the lemma and theorem which follow, we use Theorem 3.5.3 to determine a bound on $\nu(R)$, where $R = \text{APREM}(A,B)$ and A, B are elements of $Q(\alpha)[x]$. Theorem 3.5.3 applies to the output of PMQRMM. But by looking at Algorithm 4.5.3, we see that if $(\bar{Q},\hat{R})$ is the output of PMQRMM, then $R = \text{AREP}(\hat{R})$ and thus $\nu(R) \le \nu(\hat{R})$.

Lemma 5.3.1. Let A and B be elements of $Q(\alpha)[x]$ such that $\deg(A) \ge \deg(B) > 0$; $A_1,A_2,\ldots,A_{t+2} = 0$ the monic p.r.s. of A and B; $n_i = \deg(A_i)$ and $d_i = \nu(A_i)$. Define $d = \nu(A)$, $f = \nu(B)$, $g = |\Psi|_1$ and $k = \deg(\Psi)$. Also, let $h = kg^{k-1}$, $\ell = (k+1)^{2k^2}$, $\delta_i = n_i - n_{i+1}$, $e_i = (k+1)(\delta_i+1)$ and $E_i = \prod_{j=1}^{i} e_j$. Then for $1 \le i \le t-1$, $d_{i+2} \le \{\ell^3 h^5 d^{k+1} f^{k+1}\}^{2^i E_i}$ and $L(d_{i+2}) \le \{2(k+1)n_2\}^{i+1} n_1\{k+L(dfg)\}$.

Proof. Let $\hat{A}_1 = A$, $\hat{A}_2 = B$ and $\hat{A}_{i+2} = \text{rem}(A_i,A_{i+1})$. $d_i = \nu(A_i)$, so by Lemma 5.2.1,

$$d_i \le \ell h^2 \nu(\hat{A}_i)^{k+1} . \tag{1}$$

Thus $d_1 \le \ell h^2 d^{k+1}$ and $d_2 \le \ell h^2 f^{k+1}$. Since $d_1 d_2 \le \ell^2 h^4 d^{k+1} f^{k+1}$, for the first part of the theorem it suffices to show that $d_{i+2} \le \{\ell h d_1 d_2\}^{2^i E_i}$. We do this by induction on i.

Suppose $i = 1$. By (1) and Theorem 3.5.3, $d_3 \le \ell h^2 \nu(\hat{A}_3)^{k+1} \le \ell h^2 (hd_2)^{(\delta_1+1)(k+1)} d_1^{k+1} \le \ell h^{2+e_1} d_2^{e_1} d_1^{k+1} \le (\ell h d_1 d_2)^{2e_1} = (\ell h d_1 d_2)^{2E_1}$.

Suppose $i > 1$. By (1) and Theorem 3.5.3, $d_{i+2} \le \ell h^2 \nu(\hat{A}_{i+2})^{k+1} \le \ell h^2 \{(hd_{i+1})^{\delta_i+1} d_i\}^{k+1}$. Therefore, by the induction assumption, $d_{i+2} \le \ell h^2 \{(hd_{i+1})^{\delta_i+1} d_i\}^{k+1} = \ell h^{2+e_i} d_{i+1}^{e_i} d_i^{k+1} \le \ell h^{2e_i} d_{i+1}^{e_i} d_i^{k+1} \le \ell h^{2e_i} \{\ell h d_1 d_2\}^{2^{i-1}E_{i-1}e_i + 2^{i-2}E_{i-2}(k+1)} \le \ell^{1+2^{i-1}E_i+2^{i-2}E_{i-1}} h^{2e_i+2^{i-1}E_i+2^{i-2}E_{i-1}} (d_1d_2)^{2^{i-1}E_i+2^{i-2}E_{i-1}} \le \ell^{2^{i-1}E_i+2^{i-1}E_{i-1}} h^{2^{i-1}E_i+2^{i-1}E_i} (d_1d_2)^{2^{i-1}E_i+2^{i-1}E_i} \le \{\ell h d_1 d_2\}^{2^i E_i}$. Hence by induction, $d_{i+2} \le \{\ell^3 h^5 d^{k+1} f^{k+1}\}^{2^i E_i}$.

$L(d_{i+2}) \le 2^i E_i \{L(\ell)+L(n)+kL(d)+kL(f)\} \le 2^i \prod_{j=1}^{i}(k+1)(\delta_j+1) \{k^2+kL(g)+kL(d)+kL(f)\} \le 2^i (k+1)^i \prod_{j=1}^{i}(\delta_j+1)k\{k+L(dfg)\} \le 2^i (k+1)^{i+1} (n_1-n_2+1)n_2^{i-1}\{k+L(dfg)\} \le \{2(k+1)n_2\}^{i+1} n_1\{k+L(dfg)\}$. ∎

Theorem 5.3.1. Let A and B be elements of $Q(\alpha)[x]$. Let $d = \nu(A)$, $f = \nu(B)$, $g = |\Psi|_1$, $m = \deg(A)$, $n = \deg(B)$ and $k = \deg(\Psi)$. If $A = 0$ or $B \ne 0$ and $n = 0$, then $t_{\text{APMGCD}}(A,B) \le (n+1)k^4\{L(fg)^2+k^2\}$. If $B = 0$ or $A \ne 0$ and $m = 0$ then $t_{\text{APMGCD}}(A,B) \le (m+1)k^4\{L(dg)^2+k^2\}$. Let $e = \text{mep}(\Omega)$. Assume $A \ne 0$, $B \ne 0$ and let $n_1 = \max(m,n)$ and $n_2 = \min(m,n)$. Then if $n_2 > 0$, $t_{\text{APMGCD}}(A,B) \le k^5 n_2 n_1^7 \{2(k+1)n_2\}^{2n_2} \{k^2+L(dfg)^2\} + k^2 n_1 L(e)^2$.

Proof. Assume m, n > 0. In step (1), by Theorem 5.2.16, $t_{\text{APMONA}}(A) + t_{\text{APMONA}}(B) \le m\,k^4\{L(dg)^2+k^2\} + n\,k^4\{L(fg)^2+k^2\} \le k^4 n_1 n_2\{k^2+L(dfg)^2\}$. In step (2), assume that $n_1,n_2,\ldots,n_{h+1}$ is the degree sequence of $A_1,A_2,\ldots$

$A_1, A_2, \ldots, A_{h+2} = 0$ is the monic p.r.s. of A_1 and A_2; and $d_i = v(A_i)$. Then by Theorem 4.5.11 and Lemma 5.3.1, the time for the j^{th} execution of APREM(A_1, A_2) is $\preceq k^5(n_j - n_{j+1}+1)^4\, n_j(k^2+L(gd_{j+1})^2) + k^3 (n_j-n_{j+1}+1)^2\, n_j L(d_j)^2 + k^2(n_j-n_{j+1}+1)L(e)^2 \preceq k^5(n_j-n_{j+1}+1)^4\, n_j(k^2+\{2(k+1)n_2\}^{2j}\, n_1^2\{k^2+L(dfg)^2\}) + k^2(n_j-n_{j+1}+1)L(e)^2 \preceq k^5 n_j^5\{2(k+1)n_2\}^{2j}\, n_1^2\{k^2+L(dfg)^2\} + k^2(n_j-n_{j+1}+1)L(e)^2$. By Theorem 5.2.19, Theorem 3.5.3 and Lemma 5.3.1, $t_{\text{APMONA}}(\hat{A}_3) \preceq n_{j+2}\, k^4\{L(gv(\hat{A}_3))^2+k^2\} \preceq n_{j+2}\, k^4 \{(n_j-n_{j+1}+1)^2\{k^2L(g)^2+L(d_{j+1})^2\}+L(d_j^2)\} \preceq n_{j+2} k^4(n_j-n_{j+1}+1)^2\{2(k+1)n_2\}^{2j}\, n_1^2\{k^2+L(dfg)^2\} \preceq k^4\, n_j^3\{2(k+1)n_2\}^{2j}\, n_1^2\{k^2+L(dfg)^2\}$. Hence the total time of step (2) is $\preceq \sum_{j=1}^{h}(k^5 n_j^5\{2(k+1)n_2\}^{2j}\, n_1^2\{k^2+L(dfg)^2\} + k^2(n_j-n_{j+1}+1)L(e)^2) \preceq \sum_{j=1}^{h}(k^5 n_1^7\{2(k+1)n_2\}^{2n_2}\{k^2+L(dfg)^2\} + k^2(n_j-n_{j+1}+1)L(e)^2) \preceq k^5 n_1^7 n_2\{2(k+1)n_2\}^{2n_2}\{k^2+L(dfg)^2\} + k^2 n_1 L(e)^2$. ∎

CHAPTER 6: EMPIRICAL OBSERVATIONS

Section 6.1. Introduction

In this chapter we will present a test program and some empirical results obtained by using the algorithms in Chapter 5. The program will serve as an illustration of how to use this system. The empirical results will give an indication of the relative efficiency of the algorithms and show some of their characteristics.

For certain algebraic numbers α some elements of $Q(\alpha)[x]$ are generated and their g.c.d.'s are computed using APGCD and APMGCD. The test program was run on a Univac 1108 at the University of Wisconsin, where a subroutine, TIME, is available which reads the computer's clock; this was used to time these computations. Tables of this data are presented. The units in these tables are seconds of c.p.u. time. In some cases we also give approximate values of $v(A)$ and $v(B)$, where A and B are the inputs to the programs.

Section 6.2. Examples and Empirical Results

The following are the subroutines and test program used.

The function subprogram NU computes $\nu(A)$ where A is an arbitrary element of $Q[x_1,\ldots,x_v]/(\Psi(x_1))$. If A is $((r_1,r_2),\bar{A})$ then NU computes $r_1|\bar{A}|_1 + r_2 = |r_1\bar{A}|_1 + r_2 = \nu(A)$.

```
      INTEGER FUNCTION NU(A)
      INTEGER A,ABAR,R,R1,R2,T,U
      INTEGER FIRST,PFA,PNORMF,SECOND
10    IF (A.NE.0) GO TO 11
      NU=PFA(1,0)
      RETURN
11    R=FIRST(A)
      ABAR=SECOND(A)
      CALL ADV2(R1,R2,R)
      T=PNORMF(ABAR)
      U=IPROD(R1,T)
      NU=ISUM(U,R2)
      CALL ERLA(U)
      CALL ERLA(T)
      RETURN
      END
```

The function subprogram DATA computes "random" elements of $Q(\alpha)[x]$. It does this by calling a function PRAND, which generates random integral polynomials. If K is a FORTRAN integer, V the list (y,x), and D the list (n_1,n_2) where n_i is a FORTRAN integer, then PRAND(K,V,D) generates A, an element of I[y,x] such that $|A|_\infty < 2^K$, $\partial_y(A) \leq n_1$ and $\partial_x(A) \leq n_2$. DATA assumes $n_1 < \deg(\Psi)$. Given A, it forms $T = \mu(A)$, an element of $I[y,x]/(\Psi(y))$. Finally, AREP is used to determine an element of $I[\alpha][x]$. Since APGCD only uses polp(A) and APMGCD immediately computes APMONA(A), the fact that

$\nu_2(A) = 1$ does not bias our results.

```
      INTEGER FUNCTION DATA(K,V,D)
      INTEGER ABAR,C,D,R,T,V
      INTEGER AREP,PFL,PRAND,RNINT1
1     DATA=0
      T=PRAND(K,V,D)
      IF (T.EQ.0) RETURN
      CALL PICPP(T,C,ABAR)
      CALL PERASE(T)
      R=RNINT1(C)
      CALL ERLA(C)
      T=PFL(R,PFL(ABAR,0))
      DATA=AREP(T)
      CALL PMERAS(T)
      RETURN
      END
```

In the main program which follows, the code in block 1 initializes the necessary variables from other SAC-1 systems it uses. Of course the values of these variables are relative to the particular implementation at the University of Wisconsin.

In block 2 certain variables used by this program are initialized. IN is the unit number of the card reader; OUT the unit number of the line printer; LAVAIL determines the length of the available space list; TEST, the counter of the number of test cases, is set to zero; V is initialized for DATA using a subroutine which reads in a list of variables; and N, the number of test cases to be run, is input.

In block 3, a polynomial P is input from which Ψ and Ω are computed using PGRRFD. In this case Ω is arbitrarily set to be the second element of I, but this may be changed from run to run. PSI and OMEGA are output.

In block 4, a DO loop is executed. Relatively prime polynomials A and B are generated by DATA of degree L, where L is the loop index. In this case L assumes the values 5, 7, 9. Actually, A and B need not be relatively prime, but none have ever been generated by the author in this manner which were not. μ(A) and μ(B) are computed and output as well as the degrees of A and B. gcd (A,B) is computed in two ways. Also, deg(gcd(A,B)) and the time for the two computations are output.

Block 5 is similar to block 4 except that non-relatively prime data are used. DATA is used to generate A, B and E, polynomials of degree L. In this case L assumes the values 2, 3, 4. Then AE and BE are computed by APROD and gcd(AE,BE) = E is computed in 2 ways and timed.

In block 6 we loop if TEST < N, and check for lost cells and terminate otherwise.

```
      INTEGER A,ARRAY,AVAIL,B,BETA,C,D,E,F,H,OMEGA,OUT
      INTEGER P,PEXP,PRIME,PSI,R,RECORD,SPACE,STAK
      INTEGER SYMLST,T,TEST,THETA,V
      INTEGER APGCD,APMGCD,APROD,BORROW,DATA,FIRST
      INTEGER GENPR,PDEG,PFA,PMDEG,PREAD,SECOND
      INTEGER TIME,VLIST
      COMMON /TR1/ AVAIL,STAK,RECORD(72)
      COMMON /TR2/ SYMLST
      COMMON /TR3/ BETA,THETA
      COMMON /TR4/ PRIME,PEXP
      COMMON /TR7/ PSI,OMEGA
      DIMENSION SPACE(32400),ARRAY(2000)
C
C
 1    CALL BEGIN(SPACE,32400)
      SYMLST=0
```



```
      BETA=2**33
      THETA=10*9
      PRIME=GENRP(ARRAY,2000,2**31+1)
      PEXP=31
C
C
 2    IN=5
      OUT=6
      LAVAIL=LENGTH(AVAIL)
      TEST=0
      V=VLIST(IN)
      READ(IN,1000) N
 1000 FORMAT(I2)
C
C
 3    TEST=TEST+1
      WRITE(OUT,1001) TEST
 1001 FORMAT(30H1                    TEST NUMBER , I2)
      P=PREAD(IN)
      CALL PGRRFD(P,PSI,R,I)
      CALL PERASE(P)
      OMEGA=BORROW(SECOND(I))
      CALL ERASE(I)
      CALL ERASE(R)
      WRITE(OUT,1002)
 1002 FORMAT(6HOPSI =)
      CALL PWRITE(OUT,PSI)
      WRITE(OUT,1003)
 1003 FORMAT(8HOOMEGA =)
      CALL RWRITE(OUT,FIRST(OMEGA))
      CALL RWRITE(OUT,SECOND(OMEGA))
C
C
 4    K=4
      WRITE(OUT,1004)
 1004 FORMAT(31HOTEST OF RELATIVELY PRIME INPUT)
      M=PDEG(PSI)-1
      DO 41 L=5,9,2
      D=PFA(M,PFA(L,0))
      A=DATA(K,V,D)
      B=DATA(K,V,D)
      F=NU(A)
      H=NU(B)
      LL=PMDEG(A)
      WRITE(OUT,1005) LL
 1005 FORMAT(14HODEGREE OF A =,I3,8H NU(A) =)
      CALL IWRITE(OUT,F)
      CALL ERLA(F)
      LL=PMDEG(B)
      WRITE(OUT,1006) LL
 1006 FORMAT(14HODEGREE OF B =,I3,8H NU(B) =)
```

```
      CALL IWRITE(OUT,H)
      CALL ERLA(H)
      T=TIME(0)
      C=APGCD(A,B)
      T=TIME(0)-T
      LL=PMDEG(C)
      WRITE(OUT,1010) LL
      CALL PMERAS(C)
      WRITE(OUT,1007) T
 1007 FORMAT(23HOTHE TIME FOR APGCD IS ,I6)
      T=TIME(0)
      C=APMGCD(A,B)
      T=TIME(0)-T
      CALL PMERAS(A)
      CALL PMERAS(B)
      CALL PMERAS(C)
      WRITE(OUT,1008) T
 1008 FORMAT(24HOTHE TIME FOR APMGCD IS ,I6)
 41   CALL ERLA(D)
C
C
 5    K=4
      WRITE(OUT,1009)
 1009 FORMAT(33HOTEST WITH GCD OF POSITIVE DEGREE)
      DO 51 L=2,4
      D=PPA(M,PPA(L,0))
      A=DATA(K,V,D)
      B=DATA(K,V,D)
      E=DATA(K,V,D)
      T=APROD(A,E)
      CALL PMERAS(A)
      A=T
      T=APROD(B,E)
      CALL PMERAS(B)
      B=T
      CALL PMERAS(E)
      F=NU(A)
      H=NU(B)
      LL=PMDEG(A)
      WRITE(OUT,1005) LL
      CALL IWRITE(OUT,F)
      CALL ERLA(F)
      LL=PMDEG(B)
      WRITE(OUT,1006) LL
      CALL IWRITE(OUT,H)
      CALL ERLA(H)
      T=TIME(0)
      C=APGCD(A,B)
      T=TIME(0)-T
```



```
      LL=PMDEG(C)
      WRITE(OUT,1010) LL
 1010 FORMAT(21HODEGREE OF GCD(A,B) =,I3)
      CALL PMERAS(C)
      WRITE(OUT,1011) T
 1011 FORMAT(23HOTHE TIME FOR APGCD IS ,I6)
      T=TIME(0)
      C=APMGCD(A,B)
      T=TIME(0)-T
      CALL PMERAS(A)
      CALL PMERAS(B)
      CALL PMERAS(C)
      WRITE(OUT,1012) T
 1012 FORMAT(24HOTHE TIME FOR APMGCD IS ,I6)
 51   CALL ERLA(D)
C
C
 6    CALL PERASE(PS1)
      CALL ERASE(OMEGA)
      IF (TEST.LT.N) GO TO 3
      CALL ERASE(SYMLST)
      CALL ERASE(V)
      LAVAIL=LAVAIL-LENGTH(AVAIL)
      WRITE(OUT,1013) LAVAIL
 1013 FORMAT(12HOCELLS LOST ,I3)
      END
```

For our first example we ran the test program for $\alpha = (1+\sqrt{5})/2$, the well known golden ratio (see [KNU68]). We input N=3 and the following polynomials which were used for Ψ:

$$\Psi_1(y) = y^2 - y - 1 ,$$

$$\Psi_2(y) = y^4 - y^3 - y - 1$$

and

$$\Psi_3(y) = 2y^4 - 2y^3 - y^2 - y - 1 .$$

Note that $\Psi_1(y)$ is irreducible, $\Psi_2(y) = \Psi_1(y)(y^2+1)$ and $\Psi_3(y) = \Psi_1(y)(2y^2+1)$. Also observe that $\deg(\Psi_2) = \deg(\Psi_3) = 2 \deg(\Psi_1)$,

$|\Psi_2|_1 = |\Psi_1|_1 + 1$ and $|\Psi_3|_1 = 2|\Psi_2|_1 - 1$. The values computed for Ω using PGRRFD were:

$$\Omega_1 = \{1,2\}$$

$$\Omega_2 = \{1,2\}$$

and

$$\Omega_3 = \{3/2,2\} \quad .$$

The results are summarized in the following four tables. In the subroutine DATA, K=4 for all of these cases.

Relatively Prime Cases

APMGCD:

degree	Ψ_1	Ψ_2	Ψ_3
5	2.441	8.600	10.104
7	5.057	23.448	27.976
9	10.118	52.316	64.559

Table 6.2.1

APGCD:

degree	Ψ_1	Ψ_2	Ψ_3
5	.871	3.147	3.328
7	2.132	7.617	7.695
9	4.001	14.839	15.272

Table 6.2.2

Cases Where deg(gcd(A,B))=deg(A)/2=deg(B)/2

APMGCD:

degree	Ψ_1	Ψ_2	Ψ_3
4	1.062	2.851	2.974
6	1.943	7.330	6.930
8	3.334	13.379	14.477

Table 6.2.3

APGCD:

degree	Ψ_1	Ψ_2	Ψ_3
4	1.184	3.827	4.885
6	3.514	15.218	15.436
8	8.220	38.966	37.940

Table 6.2.4

Comparing Table 6.2.1 with 6.2.2 and Table 6.2.3 with 6.2.4 we see that APGCD is superior to APMGCD for the relatively prime inputs tested, but APMGCD was superior when the inputs had positive degree. This will also be reflected in the tables which follow.

Suppose A and B are input to APGCD and n = min(deg(A), deg(B)). Almost certainly polp(A) and polp(B) are relatively prime integral polynomials, so $S(\text{polp}(A), \text{polp}(B))$ probably contains n+2 elements. Therefore n standard modular representations must be computed regardless of the degree of gcd(A,B). Beginning with the subresultant of least degree, the leading coefficient is constructed and evaluated at α, then the leading coefficient of the next subresultant is computed, and so on until a non-zero one is found. Then this entire subresultant is computed, mapped into $Q(\alpha)[x]$ and its monic associate computed. Now the smaller the degree of gcd(A,B), the fewer such operations must be performed. Hence the smaller the degree of gcd(A,B) the more efficient APGCD is.

On the other hand, APMGCD performs all of its computations in $Q(\alpha)[x]$, and the number of iterations it performs is inversely related to deg(gcd(A,B)). Hence the larger this degree is, the more efficient APMGCD is. Now deg(gcd(A,B)) = deg(A)/2 = deg(B)/2 in Tables 6.2.3 and 6.2.4, which is quite large relative to the degree of the input. For smaller degree its relative superiority to APGCD will diminish.

In Tables 6.2.2 and 6.2.4 note that the time for inputs of each degree is adversely affected going from Ψ_1 to Ψ_2, when the

degree of Ψ doubles, $|\Psi|_\infty$ stays the same and $|\Psi|_1$ increases by one. However, going from Ψ_2 to Ψ_3 the times do not change very much when the degree of Ψ stays the same and the norms increase. This stands to reason since increasing deg(Ψ) increases $\partial_y(A)$ and $\partial_y(B)$ in our test cases and therefore $|A|_1$ and $|B|_1$ also increase. Then this may increase the number of primes needed, and always increases the number of evaluation homomorphisms needed for each prime. But increasing $|\Psi|_1$ only affects the application of AZTEST, PMRPOL and APMONA, and these applications are relatively minimal.

Table 6.2.1 shows that the efficiency of APMGCD was adversely affected by going from Ψ_1 to Ψ_2 and from Ψ_2 to Ψ_3. Again, this is to be expected since both deg(Ψ) and the size of the coefficients of Ψ, especially ldcf(Ψ), enter into the computation of remainders in $Q(\alpha)[x]$. Table 6.2.3 does not reflect much change going from Ψ_2 to Ψ_3, but then only 2, 3, and 4 remainders were computed for these cases.

For the next input to the test program we chose N=1 and the polynomial $P(y) = 16y^5 - 20y^3 + 5y$, the fifth Chebyshev polynomial. We arbitrarily chose α to be the root $-\cos 7\pi/10 \simeq -.5878$, which is in the second interval output by PGRRFD. The result was that

$$\Psi(y) = 16y^4 - 20y^2 + 5$$

and

$$\Omega = \{-5/8, -9/16\} .$$

The results are shown in the following two tables. In the subroutine DATA, K=4 for these cases.

Relatively Prime Cases

$\nu(A),\nu(B)$, degree	APGCD	APMGCD
$1.6\cdot10^2, 1.8\cdot10^2$, 5	3.515	13.923
$2.4\cdot10^2, 2.5\cdot10^2$, 7	7.776	38.907
$2.9\cdot10^2, 2.9\cdot10^2$, 9	15.818	95.816

Table 6.2.5

Cases Where deg(gcd(A,B))=deg(A)/2=deg(B)/2

$\nu(A),\nu(B)$, degree	APGCD	APMGCD
$5\cdot10^4, 4\cdot10^4$, 4	6.120	3.502
$3\cdot10^5, 4\cdot10^5$, 6	21.037	8.901
$5\cdot10^5, 4\cdot10^5$, 8	51.057	19.859

Table 6.2.6

In Table 6.2.5 note that APGCD is about 4, 5 and 6 times as fast as APMGCD for inputs of degree 5, 7 and 9 respectively. In Tabel 6.2.6, APMGCD is about 1.7, 2.4 and 2.6 times as fast as APGCD for inputs of degree 4, 6 and 8 respectively.

For our final case we generated a random univariate polynomial of degree 3 with coefficients less than 2^{10} in magnitude. We determined that this polynomial was irreducible, and chose α to be the root in the first interval output by PGRRFD. This resulted in

$$\Psi(y) = 379y^3 + 433y^2 + 601y + 486$$

and

$$\Omega = (-237/256, -473/512).$$

The results are shown in the following two tables. In the subroutine DATA, K=26 for Table 6.2.7 and K=4 for Table 6.2.8.

Relatively Prime Cases

$\nu(A),\nu(B)$, degree	APGCD	APMGCD
$6\cdot10^8, 6\cdot10^8$, 5	6.398	56.506
$7\cdot10^8, 8\cdot10^8$, 6	10.678	109.883
$8\cdot10^6, 7\cdot10^8$, 7	16.022	191.698
10^9, 10^9, 9	32.412	*

*case not run

Table 6.2.7

Cases Where deg(gcd(A,B))=deg(A)/2=deg(B)/2

$\nu(A),\nu(B)$, degree	APGCD	APMGCD
$5\cdot10^8$, $3\cdot10^8$, 4	6.667	3.024
$7\cdot10^8$, $8\cdot10^8$, 6	19.839	6.692
10^9, $9\cdot10^8$, 8	52.965	17.249

Table 6.2.8

In Table 6.2.7 note that APGCD is about 9, 10 and 12 times as fast as APMGCD for inputs of degree 5, 6 and 7 respectively. For degree 9 it was felt that computing APMGCD(A,B) was too impractical, given the previous entries in the table. In Table 6.2.8, APMGCD is about 2.2, 3 and 3 times as fast as APGCD for inputs of degree 4, 6 and 8 respectively.

It is interesting to note that in the relatively prime cases, going from Table 6.2.5 to 6.2.7, the relative superiority of APGCD over APMGCD more than doubled, even though the degree of Ψ decreased. In the cases of g.c.d.'s of positive degree, going from Table 6.2.6 to 6.2.8, the relative advantage of APMGCD over APGCD increased only slightly.

Note also the entries for inputs of degree 6 in Table 6.2.7 and Table 6.2.8. For inputs of the same degree and coefficients of approximately the same size, APGCD was twice as slow when the inputs had a g.c.d. of degree 3 as when they were relatively prime; about 19.8 to 10.7 seconds. On the other hand, APMGCD was more than 16 times as slow when they were relatively prime as when they had a g.c.d. of degree 3; about 110 to 6.7 seconds.

In light of these observations, if these examples are representative of the inputs to be used, then APMGCD will probably be faster when the degree of the g.c.d. is large relative to the degree of the inputs, whereas if the degree of the g.c.d. is small then APMGCD will probably be much slower.

APPENDIX: FORTRAN PROGRAM LISTINGS

```
      INTEGER FUNCTION AABS(A)
      INTEGER A,S,ASIGN,BORROW,PMNEG
10    SEASIGN(A)
      IF (S.LT.0) GO TO 11
      AABS=BORROW(A)
      RETURN
11    AABS=PMNEG(A)
      RETURN
      END


      INTEGER FUNCTION ADIF(A,B)
      INTEGER A,B,D,AREP,PMDIF
1     DEFMDIF(A,B)
      ADIF=D
      IF (A.EQ.0.OR.D.EQ.0) RETURN
      ADIF=AREP(D)
      CALL PMERAS(D)
      RETURN
      END


      SUBROUTINE ADV2(A,B,L)
      INTEGER A,B
10    CALL ADV(A,L)
      CALL ADV(B,L)
      RETURN
      END


      INTEGER FUNCTION ANEG(A)
      INTEGER A,PMNEG
1     ANEG=PMNEG(A)
      RETURN
      END


      INTEGER FUNCTION ANREP2(A)
      INTEGER A,ABAR,ABARST,B,BBAR,C,CCAP,CSTAR,D,DCAP
      INTEGER D1,D2,E1,E2,OMEGA,P,PEXP,PRIME,PSI
      INTEGER PSIBAR,PSIBST,Q,QCAP,QCAPST
      INTEGER R,S,T,TSTAR,U,V,W
      INTEGER BORROW,CORA,CMOD,CPCRA,CPDEG,CPMOD
      INTEGER CPTCIP,PCONT,PDEG,PFA,PFL,PNORMF
      INTEGER PS2,PVLIST,RNINT2,RNO,RNREP
      COMMON /TR4/PRIME,PEXP
      COMMON /TR7/PSI,OMEGA
10    CALL FIRST2(P,ABAR,A)
      NEPDEG(ABAR)
```

```
      IF (N.NE.0) GO TO 11
      SETNRECP(R)
      ANRECP=PFL(S,PFL(BORROW(ABAR),C))
      RETURN
11    CALL PRODCF(PSI,ABAR,B,PSIBAR,CCAP)
      CALL ERASE(B)
      CALL ERASE(CCAP)
      M=PDEG(PSIBAR)
20    D1=PNORMF(PSIBAR)
      D2=PNORMF(ABAR)
      E1=IPOWER(D1,N)
      E2=IPOWER(D2,M)
      CALL ERLA(D1)
      CALL ERLA(D2)
      DCAP=IPROD(E1,E2)
      CALL ERLA(E1)
      CALL ERLA(E2)
      CALL IMFI(DCAP,2)
30    L=PRIM
      GCAP=PFA(1,C)
      T=0
      C=C
      V=PVLIST(PSIBAR)
40    IF (L.NE.C) GO TO 42
      PRINT 41
41    FORMAT(23H ALGORITHM ANRECP FAILS)
      STOP
42    CALL ADV(P,L)
      PSIBST=CPMOD(P,PSIBAR)
      IF (CPDEG(PSIBST).EQ.M) GO TO 43
      CALL CPERAS(PSIBST)
      GO TO 40
43    ABARST=CPMOD(P,ABAR)
      IF (CPDEG(ABARST).EQ.N) GO TO 44
      CALL CPERAS(PSIBST)
      CALL CPERAS(ABARST)
      GO TO 40
44    GCAPST=CMOD(P,GCAP)
      Q=CRECIP(P,GCAPST)
50    CALL CRESCP(P,PSIBST,ABARST,CSTAR,TSTAR)
      CALL CPERAS(PSIBST)
      CALL CPERAS(ABARST)
      IF (CSTAR.EQ.-1) GO TO 40
60    U=CPCRA(GCAP,T,P,TSTAR,Q,V)
      CALL ERASE(T)
      CALL CPERAS(TSTAR)
      T=U
70    W=CCRA(DCAP,C,P,CSTAR,D)
      CALL ERLA(C)
      C=W
80    CALL IMFI(GCAP,P)
      IF (ICOMP(DCAP,DCAP).EQ.-1) GO TO 40
```



```
90    CALL ERLA(QCAP)
      CALL ERLA(DCAP)
      CALL ERASE(V)
      CALL ERASE(PSIBAR)
      D=PCONT(T)
      IF (ISIONL(C).EQ.-1) CALL IMFI(D,-1)
      BBAR=PSQ(T,D)
      CALL ERASE(T)
      T=RNINTZ(D,C)
      C=PNQ(T,R)
      CALL RNERAS(T)
      CALL ERLA(D)
      CALL ERLA(C)
      ANRECP=PFL(S,PFL(BBAR,C))
      RETURN
      END


      INTEGER FUNCTION APSCD(ASTAR,BSTAR)
      INTEGER A,ASTAR,B,BSTAR,C,CCAP,CHATST,D,DSTAR
      INTEGER ESTAR,OMEGA,PSI,R,S,T,V
      INTEGER APMONA,AZTEST,BORROW,PCRCMB,PFA,PFL
      INTEGER PMDEG,PMRPOL,PVBL,PVLIST,SECOND
      COMMON /TR7/ PSI,OMEGA
10    DSTAR=ASTAR
      ESTAR=BSTAR
      IF (PMDEG(ASTAR).GE.PMDEG(BSTAR)) GO TO 11
      DSTAR=BSTAR
      ESTAR=ASTAR
11    IF (ESTAR.NE.0) GO TO 12
      APSCD=APMONA(DSTAR)
      RETURN
12    IF (PMDEG(ESTAR).NE.0) GO TO 13
      APSCD=APMONA(ESTAR)
      RETURN
13    A=SECOND(DSTAR)
      B=SECOND(ESTAR)
20    CALL PSMRS2(A,B,L,D)
      L=INV(L)
      D=INV(D)
      V=PVLIST(PSI)
30    IF (L.NE.0) GO TO 31
      APSCD=APMONA(ESTAR)
      CALL ERLA(D)
      CALL ERASE(V)
      RETURN
31    CALL DECAP(L,L)
      CALL DECAP(N,D)
      C=PCRCMB(L,N,V)
40    S=AZTEST(C)
      IF (S.NE.0) GO TO 50
      CALL PERASE(C)
```

```
      CALL ERASE(L1)
      GO TO 30
50    CCAP=PFL(PVBL(A),0)
      CCAP=PFA(N,PFL(C,CCAP))
      N=N-1
60    IF (N.LT.0) GO TO 70
      C=PORDER(L1,N,V)
      S=AZTEST(C)
      IF (S.NE.0) CCAP=PFA(N,PFL(C,CCAP))
      IF (S.EQ.0) CALL PERASE(C)
      N=N-1
      GO TO 60
70    CALL ERASE(V)
      CALL ERASE(L)
      CALL ERLA(0)
      CALL ERASE(L1)
      CCAP=INV(CCAP)
      T=PFA(1,0)
      R=PFL(T,PFL(BORROW(T),0))
      CHATST=PMRPOL(R,CCAP)
      CALL RMERAS(R)
      CALL PERASE(CCAP)
      APSCD=APMONA(CHATST)
      CALL PMERAS(CHATST)
      RETURN
      END


      INTEGER FUNCTION APMSCD(A,B)
      INTEGER A,A1,A2,A3,A3HAT,B
      INTEGER APMONA,APREM,PMDEG
10    IF (A.NE.0.AND.B.EQ.0.OR.PMDEG(B).NE.0) GO TO 11
      APMSCD=APMONA(B)
      RETURN
11    A1=APMONA(A)
      IF (B.NE.0.AND.PMDEG(A).NE.0) GO TO 12
      APMSCD=A1
      RETURN
12    A2=APMONA(B)
20    A3HAT=APREM(A1,A2)
      A3=APMONA(A3HAT)
      CALL PMERAS(A1)
      CALL PMERAS(A3HAT)
30    IF (A2.NE.0) GO TO 31
      APMSCD=A2
      RETURN
31    IF (PMDEG(A3).NE.0) GO TO 32
      CALL PMERAS(A2)
      APMSCD=A3
      RETURN
```



```
32    A1=A2
      A2=A3
      GO TO 20
      END


      INTEGER FUNCTION APMONA(ACAP)
      INTEGER A,ACAP,APRIME,B,BBAR,BBARPR,BPRIME
      INTEGER C,OMEGA,PSI,S,S1,S2,T,V,X
      INTEGER ANRECP,BORROW,PFA,PFL,PIP,PMDEG,PMLDCF
      INTEGER PMRED,PMSPRD,PORDER,PVBL,PVLIST
      INTEGER SECOND,TAIL
      COMMON /TR7/ PSI,OMEGA
10    APMONA=0
      IF (ACAP.EQ.0) RETURN
      N=PMDEG(ACAP)
      APRIME=PMRED(ACAP)
      IF (APRIME.NE.0) GO TO 11
      T=PFA(1,0)
      S=PFL(BORROW(T),PFL(BORROW(T),0))
      V=PVLIST(SECOND(ACAP))
      BBAR=PORDER(T,V)
      CALL ERLA(T)
      CALL ERASE(V)
      CALL ALTER(N,TAIL(TAIL(BBAR)))
      APMONA=PFL(S,PFL(BBAR,0))
      RETURN
11    A=PMLDCF(ACAP)
      B=ANRECP(A)
      BPRIME=PMSPRD(APRIME,B,0)
      CALL PMERAS(A)
      CALL PMERAS(APRIME)
      CALL PMERAS(B)
20    CALL DECAP2(S,BBARPR,BPRIME)
      CALL DECAP2(S1,S2,S)
      C=PFL(PVBL(PSI),PFL(S2,PFA(0,C)))
      BBAR=PIP(BBARPR,S1)
      CALL ERLA(S1)
      CALL PERASE(BBARPR)
      CALL DECAP(X,BBAR)
      BBAR=PFL(X,PFL(C,PFA(N,BBAR)))
      T=PFA(1,0)
      S=PFL(T,PFL(BORROW(S2),0))
      APMONA=PFL(S,PFL(BBAR,0))
      RETURN
      END


      INTEGER FUNCTION APREM(A,BCAP)
      INTEGER A,B,BCAP,B1,B1HAT,C
      INTEGER N,O,QBAR,QHAT,R,RHAT
      INTEGER ARSP,ANRECP,BORROW,PFL,PMDEG
      INTEGER PMLDCF,PMRMM,PMRED,PMSPRD
```

```
10    N=PMDEG(BCAP)
      IF (A.NE.0.AND.PMDEG(A).GE.N) GO TO 11
      APOREM=PFL(Q,PFL(BORROW(A),0))
      RETURN
11    B=PMLDCF(BCAP)
      C=ANRECP(B)
      CALL PMERAS(B)
20    B1=PMRED(BCAP)
      B1HAT=PMSPRD(B1,C,0)
      CALL PMERAS(B1)
      H=PMSRMN(A,N,B1HAT)
      CALL PMERAS(B1HAT)
      CALL DECAP2(QBAR,RHAT,H)
      QHAT=PMSPRD(QBAR,C,0)
      CALL PMERAS(QBAR)
      CALL PMERAS(C)
30    Q=AREP(QHAT)
      CALL PMERAS(QHAT)
      R=AREP(RHAT)
      CALL PMERAS(RHAT)
      APOREM=PFL(Q,PFL(R,0))
      RETURN
      END


      INTEGER FUNCTION APREM(A,B)
      INTEGER A,B,Q,R,APOREM
1     L=APOREM(A,B)
      CALL DECAP2(Q,R,L)
      CALL PMERAS(Q)
      APREM=R
      RETURN
      END


      INTEGER FUNCTION APROD(A,B)
      INTEGER A,B,CCAP
      INTEGER AREP,PMPROD,SECOND,TYPE
1     CCAP=PMPROD(A,B)
      APROD=CCAP
      IF (CCAP.EQ.0) RETURN
      IF (TYPE(SECOND(SECOND(CCAP))).EQ.0) RETURN
      APROD=AREP(CCAP)
      CALL PMERAS(CCAP)
      RETURN
      END


      INTEGER FUNCTION AQ(A,B)
      INTEGER A,B,C,Q,R
      INTEGER ANRECP,APOREM,APROD,SECOND,TYPE
1     AQ=0
```



```
      IF (A.EQ.0) RETURN
      IF (TYPE(SECOND(SECOND(B))).EQ.1) GO TO 2
      C=ANRECP(B)
      AQ=APROD(A,C)
      CALL PMERAS(C)
      RETURN
2     L=APOREM(A,B)
      CALL DECAP2(Q,R,L)
      AQ=Q
      RETURN
      END


      INTEGER FUNCTION AREP(ACAP)
      INTEGER A,ABAR,ACAP,BBAR,C,CCAP,R,S,T
      INTEGER AZTEST,BORROW,PFA,PFL,PONE
      INTEGER PVBL,RNISPR,SECOND,TAIL,TYPE
1     AREP=0
      IF (ACAP.EQ.0) RETURN
      CALL FIRST2(B,ABAR,ACAP)
      IF (TYPE(SECOND(ABAR)).EQ.0) GO TO 2
      CCAP=C
      T=TAIL(ABAR)
      GO TO 3
2     S=AZTEST(ABAR)
      IF (S.EQ.1) AREP=BORROW(ACAP)
      RETURN
3     IF (T.EQ.0) GO TO 4
      CALL ADV2(A,N,T)
      IF (AZTEST(A).EQ.1) CCAP=PFA(N,PFL(BORROW(A),CCAP))
      GO TO 3
4     IF (CCAP.EQ.0) RETURN
      CCAP=INV(CCAP)
      CCAP=PFL(PVBL(ABAR),CCAP)
      CALL PICPF(CCAP,C,BBAR)
      T=PONE(C)
      IF (T.EQ.1) C=BORROW(B)
      IF (T.EQ.0) C=RNISPR(C,B)
      CALL CR_AIC
      CALL PERASE(CCAP)
      AREP=PFL(S,PFL(BBAR,C))
      RETURN
      END


      INTEGER FUNCTION ACISN(A)
      COMMON /TP7/PSI,OMEGA
      INTEGER A,ABAR,ABARPM,ACAP,C,OMEGA,PSI,R,S
      INTEGER SBAR,S1,S2,T,T1,T2,V,X,Y,Z
      INTEGER BORROW,PDEG,PFL,PSSFD,PLOCF
      INTEGER PSION,REVAL,RNAVER,SECOND,TYPE
```

```
      ASIGN=0
      IF (A.EQ.0) RETURN
      ABAR=BORROW(SECOND(A))
      IF (TYPE(SECOND(ABAR)).NE.1) GO TO 11
      ACAP=PLOOF(ABAR)
      CALL PERASE(ABAR)
      CALL PICPP(ACAP,C,ABAR)
      CALL PERASE(ACAP)
      CALL ERLA(C)
11    IF (POEG(ABAR).GT.0) GO TO 20
      ASIGN=PSIGN(ABAR)
      CALL PERASE(ABAR)
      RETURN
20    CALL FIRSTZ(R,T,OMEGA)
      R=BORROW(R)
      T=BORROW(T)
      ABARPM=PGSPD(ABAR)
      S=0
      S1=REVAL(PSI,R)
      S2=-S1
      T1=2
      T2=2
30    I=PFL(R,PFL(T,0))
      CALL RRTS(ABARPM,I,C,S,N,X)
      CALL DECAPZ(Y,Z,I)
      IF (N.EQ.0) GO TO 70
40    V=PNAVER(R,T)
      SBAR=REVAL(PSI,V)
      IF (S1*SBAR.GE.0) GO TO 41
      CALL RNERAS(T)
      T=V
41    IF (SBAR*S2.GE.0) GO TO 42
      CALL RNERAS(R)
      R=V
42    IF (N.NE.1) GO TO 30
50    IF (T1.NE.2) GO TO 51
      T1=REVAL(ABARPM,R)
      T2=REVAL(ABARPM,T)
      GO TO 60
51    IF (SBAR.NE.S1) GO TO 52
      T1=REVAL(ABARPM,R)
      GO TO 60
52    T2=REVAL(ABARPM,T)
60    IF (T1*T2.LT.0) GO TO 40
70    IF (T2.EQ.0) GO TO 71
      ASIGN=REVAL(ABAR,T)
      GO TO 80
71    ASIGN=REVAL(ABAR,R)
80    CALL ERASE(S)
      CALL RNERAS(R)
      CALL RNERAS(T)
```



```
      CALL PERASE(ABAR)
      CALL PERASE(ABARPM)
      RETURN
      END


      INTEGER FUNCTION ASPROD(A,B,N)
      INTEGER A,B,PMSPRD
1     ASPROD=PMSPRD(A,B,N)
      RETURN
      END


      INTEGER FUNCTION ASQ(A,B)
      INTEGER A,B,C,ANRECP,PMSPRD
1     ASQ=0
      IF (A.EQ.0) RETURN
      C=ANRECP(B)
      ASQ=PMSPRD(A,C,0)
      CALL PMERAS(C)
      RETURN
      END


      INTEGER FUNCTION ASUM(A,B)
      INTEGER A,C,D,AREP,PMSUM
1     D=PMSUM(A,B)
      ASUM=0
      IF (A.EQ.0.OR.B.EQ.0) RETURN
      ASUM=AREP(D)
      CALL PMERAS(D)
      RETURN
      END


      INTEGER FUNCTION AZTEST(A)
      INTEGER A,B,OMEGA,PSI,R,S1,S2,T
      INTEGER PROD,REVAL
      COMMON /TR7/PSI,OMEGA
1     AZTEST=0
      IF (A.EQ.0) RETURN
      B=PROD(A,PSI)
2     CALL FIRSTZ(R,T,OMEGA)
      S1=REVAL(B,R)
      S2=REVAL(B,T)
      CALL PERASE(B)
      IF (S1*S2.GE.1) AZTEST=1
      RETURN
      END
```

```
      INTEGER FUNCTION CICVR(P,L,N)
      INTEGER A,C,CHAT,D,P,SHAT,T,W
      INTEGER FIRST,PFA,SECOND,TAIL
10    T=L
      C=0
      D=PFA(C,PFA(1,0))
      M=FIRST(SECOND(L))-N
20    IF (T.EQ.0) GO TO 40
      CALL ADV2(A,SHAT,T)
      IF (M.EQ.0) GO TO 22
      DO 21 I=1,M
21    SHAT=TAIL(SHAT)
22    CHAT=SECOND(SHAT)
30    CALL CPINTI(P,C,A,CHAT,D,1,W)
      GO TO 20
40    CALL ERLA(D)
      CICVR=C
      RETURN
      END


      SUBROUTINE CRESCP(P,ACAP,BCAP,C,TCAP)
      INTEGER A,ACAP,ACAP1,ACAP2,ACAP3,AC,A3,BCAP
      INTEGER C,P,Q,S,T,TCAP,U,V,X1,X2,X3
      INTEGER BORROW,CPDEG,CPDIF,CPLNCF,CPOWER,CPPROD
      INTEGER CPREM,CPROD,CSPROD,PFA
10    ACAP1=BORROW(ACAP)
      ACAP2=BORROW(BCAP)
      V=0
      A=1
      X1=0
      X2=PFA(C,PFA(1,C))
      N1=CPDEG(ACAP1)
      N2=CPDEG(ACAP2)
20    IF(N1.GE.N2) GO TO 21
      G=C
      ACAP3=BORROW(ACAP1)
      GO TO 22
21    L=CPREM(P,ACAP1,ACAP2)
      CALL DECAP2(Q,ACAP3,L)
22    IF (ACAP3.NE.0) GO TO 23
      CALL CPERAS(ACAP1)
      CALL CPERAS(ACAP2)
      CALL CPERAS(X1)
      CALL CPERAS(X2)
      CALL CPERAS(Q)
      C=-1
      TCAP=-1
      RETURN
23    N3=CPDEG(ACAP3)
      A2=CPLNCF(ACAP2)
30    U=CPPROD(P,Q,X2)
```



```
      X3=CPDIF(P,X1,U)
      CALL CPERAS(U)
      CALL CPERAS(X1)
      CALL CPERAS(Q)
      X1=X2
      X2=X3
40    V=V+N1-N2
50    T=CPOWER(P,A2,N1-N3)
      A=CPROD(P,A,T)
60    CALL CPERAS(ACAP1)
      IF (N3.EQ.0) GO TO 70
      ACAP1=ACAP2
      ACAP2=ACAP3
      N1=N2
      N2=N3
      GO TO 20
70    A3=CPLNCF(ACAP3)
      CALL CPERAS(ACAP2)
      CALL CPERAS(ACAP3)
      CALL CPERAS(X1)
      T=CPOWER(P,A3,N2-1)
      A=CPROD(P,A,T)
      S=V/2*2
      IF (S.EQ.1) A=P-A
80    TCAP=CSPROD(P,X2,A,C)
      CALL CPERAS(X2)
      C=CPROD(P,A,A3)
      RETURN
      END


      SUBROUTINE CSUBS2(P,A,B,S,C)
      INTEGER A,ACAP1,ACAP2,ACAP3,A1,A2,A3,B
      INTEGER C2,CC,D,DELTA1,DELTA2,P,S,S2,T,U,V,W
      INTEGER BORROW,CPDEG,CPLNCF,CPOWER
      INTEGER CPREM,CPROD,CRECIP,CSPROD,PFA,PFL
1     ACAP1=BORROW(A)
      ACAP2=BORROW(B)
      A1=CPLNCF(ACAP1)
      N1=CPDEG(ACAP1)
      N2=CPDEG(ACAP2)
      N1PAR=N1
      V=0
      DEFL(N2,PFA(N1,0))
      S=PFL(BORROW(ACAP2),PFL(BORROW(ACAP1),0))
2     DELTA1=N1-N2
      A2=CPLNCF(ACAP2)
      T=DELTA1-1
      IF (T.LT.0) C2=CRECIP(P,A2)
      IF (T.GE.0) C2=CPOWER(P,A2,T)
3     ACAP3=CPREM(P,ACAP1,ACAP2)
      CALL CPERAS(ACAP1)
```

```
      IF (ACAP3.EQ.0) GO TO 6
      N3=CPDEG(ACAP3)
      A3=CPLNCF(ACAP3)
4     DELTA2=N2-N3
      T=CPOWER(P,A2,DELTA2+1)
      U=CPOWER(P,A3,DELTA2-1)
      V=CPROD(P,T,U)
      C3=CPROD(P,C2,V)
      W=DELTA2*(N1BAR-N2+J-1)
      W=W-W/2*2
      IF (W.EQ.1) C3=P-C3
      S3=CSPROD(P,ACAP3,C3,D)
      J=J+1
5     S=PFL(S3,S)
      D=PFA(N3,D)
      IF (N3.EQ.0) GO TO 6
      ACAP1=ACAP2
      ACAP2=ACAP3
      N1=N2
      N2=N3
      A1=A2
      A2=A3
      C2=C3
      GO TO 3
6     D=INV(D)
      S=INV(S)
      CALL CPERAS(ACAP2)
      CALL CPERAS(ACAP3)
      RETURN
      END


      SUBROUTINE CSVRS2(P,ACAP,BCAP,L,D)
      INTEGER A,ACAP,AHAT,BCAP,BHAT,D,DELTA1,DHAT
      INTEGER DPRIME,F,P,S,SHAT,S3HAT,T
      INTEGER CPDEG,CPDEGS,CPUEV,PFA,PFL,SECOND,TAIL
10    T=CPDEGS(ACAP)
      CALL DECAP2(N1,K1,T)
      T=CPDEGS(BCAP)
      CALL DECAP2(N2,K2,T)
      D=0
      A=-1
      L=0
      DELTA1=N1-N2
      N2BAR=N2
      T=0
20    A=A+1
      IF (A.NE.F) GO TO 22
      PRINT 21
21    FORMAT(22H ALGORITHM CSVRS1 FAILS)
      STOP
22    AHAT=CPUEV(P,ACAP,A)
```



```
      IF (CPDEG(AHAT).EQ.N1) GO TO 23
      CALL CPERAS(AHAT)
      GO TO 20
23    BHAT=CPUEV(P,BCAP,A)
      IF (CPDEG(BHAT).EQ.N2BAR) GO TO 30
      CALL CPERAS(AHAT)
      CALL CPERAS(BHAT)
      GO TO 20
30    CALL CSUBS2(P,AHAT,BHAT,SHAT,DHAT)
      CALL CPERAS(AHAT)
      CALL CPERAS(BHAT)
40    S=LEXORD(D,DHAT)
      IF (S.EQ.0) CALL ERLA(DHAT)
      IF (S.NE.1) GO TO 41
      CALL ERASE(SHAT)
      CALL ERLA(DHAT)
      GO TO 20
41    IF (S.NE.-1) GO TO 50
      CALL ERLA(D)
      D=DHAT
      DPRIME=TAIL(D)
      N2=N2BAR
      CALL ERASE(L)
      L=0
      F=K1*(DELTA1+1)*K2+1
      M=0
      J=2
50    CALL DECAP2(AHAT,BHAT,SHAT)
      CALL CPERAS(AHAT)
      CALL CPERAS(BHAT)
      I=2
      LPRIME=0
60    IF (SHAT.EQ.0) GO TO 70
      I=I+1
      CALL DECAP(S3HAT,SHAT)
      L3=0
      IF (L.NE.0) CALL DECAP(L3,L)
      IF (I.LE.J) CALL CPERAS(S3HAT)
      IF (I.GT.J) L3=PFA(A,PFL(S3HAT,L3))
      LPRIME=PFL(L3,LPRIME)
      GO TO 60
70    L=INV(LPRIME)
      M=M+1
      IF (M.LT.F) GO TO 20
      IF (TAIL(DPRIME).EQ.0) RETURN
      IF (T.NE.0) GO TO 80
      T=1
      N2=SECOND(DPRIME)
      F=F+(N2-N3-1)*(K1+K2)
      IF (M.LT.F) GO TO 20
80    DPRIME=TAIL(DPRIME)
      IF (TAIL(DPRIME).EQ.0) RETURN
```

```
      J=J+1
      F=F+K1+K2
      T=0
      N2=N3
      GO TO 20
      END


      SUBROUTINE DECAP2(A,B,L)
      INTEGER A,B
10    CALL DECAP(A,L)
      CALL DECAP(B,L)
      RETURN
      END


      SUBROUTINE FIRST2(A,B,L)
      INTEGER A,B,T,FIRST
 10   T=L
      CALL ADV(A,T)
      B=FIRST(T)
      RETURN
      END


      SUBROUTINE IOCOCF(A,B,C,ABAR,BBAR)
      INTEGER A,ABAR,B,BBAR,C,BORROW,FIRST,TAIL
 10   C=IOCO(A,B)
      IF (FIRST(C).NE.1.OR.TAIL(C).NE.C) GO TO 11
      ABAR=BORROW(A)
      BBAR=BORROW(B)
      RETURN
 11   ABAR=I9(A,C)
      BBAR=I9(B,C)
      RETURN
      END


      INTEGER FUNCTION IRNSPR(B,R)
      INTEGER B,R1,R2,R,T
 1    IRNSPR=0
      IF (B.EQ.0) RETURN
      CALL FIRST2(R1,R2,R)
      T=I9(B,R2)
      IRNSPR=IPROD(T,R1)
      CALL ERLA(T)
      RETURN
      END
```



```
      INTEGER FUNCTION LEXORD(ACAP,BCAP)
      INTEGER A,ACAP,A1,B,BCAP,B1,S
10    A1=ACAP
      B1=BCAP
20    IF (A1 .NE. 0) GO TO 22
      IF (B1 .NE. 0) GO TO 21
      S=0
      GO TO 40
21    S=-1
      GO TO 40
22    IF (B1 .NE. 0) GO TO 23
      S=1
      GO TO 40
23    CALL ADV(A,A1)
      CALL ADV(B,B1)
      IF(A.EQ.B) GO TO 20
      IF (A.LT.B) GO TO 31
      S=1
      GO TO 40
31    S=-1
40    LEXORD=S
      RETURN
      END


      INTEGER FUNCTION PORCMR(L,N,V)
      INTEGER C,CBAR,CHAT,P,Q,QCAP,T,V
      INTEGER CICVR,CMOD,CPCRA,CRECIP,PFA
1     T=L
      CBAR=0
      QCAP=PFA(1,0)
2     IF (T.EQ.0) GO TO 4
      CALL ADV2(P,_HAT,T)
      CHAT=CICVR(P,LHAT,N)
3     Q=CRECIP(P,CMOD(P,QCAP))
      C=CPCRA(QCAP,CBAR,P,CHAT,Q,V)
      CALL PERASE(CBAR)
      CALL CPERAS(CHAT)
      CBAR=C
      CALL IMFI(QCAP,P)
      GO TO 2
4     CALL ERLA(QCAP)
      PCRCMR=C
      RETURN
      END


      SUBROUTINE PORRFD(ACAP,BCAP,RCAP,I)
      INTEGER A,ACAP,B,BCAP,C,CCAP,D,E,F,RCAP,ONE,R1
      INTEGER S,SPRIME,T,TCAP,TPPIMF,X
      INTEGER ANALR,BORROW,FIRST,PDEG,PFA
      INTEGER PFL,PLDCF,PG,PVBL,REVAL,RNCOMP
```

```
      INTEGER RNFLOR,RNINT2,RNISPR,TAIL
10    I=0
      A=PLDOF(ACAP)
      ONE=PFA(1,0)
      IF (PDEG(ACAP).GT.1) GO TO 30
20    TCAP=TAIL(TAIL(TAIL(ACAP)))
      IF (TCAP.NE.0) GO TO 21
      CALL ERLA(A)
      R1=0
      GO TO 22
21    R1=PFL(INEG(FIRST(TCAP)),PFL(A,C))
22    RCAP=PFL(R1,0)
      CCAP=PFL(PVBL(ACAP),PFL(ONE,PFA(0,0)))
      RETURN
30    L=PFL(ONE,PFL(BORROW(A),0))
      J=ANALP(ACAP,L)
      CALL RNERAS(L)
40    RCAP=C
      BCAP=BORROW(ACAP)
50    IF (J.NE.0) GO TO 51
      CALL ERLA(A)
      RCAP=INV(RCAP)
      I=INV(I)
      RETURN
51    CALL DECAP(J1,J)
      CALL FIRST2(S,T,J1)
      IF (RNCOMP(S,T).NE.0) GO TO 60
      CALL DECAP2(R1,X,J1)
      CALL RNERAS(X)
      GO TO 30
60    SPRIME=RNISPR(A,S)
      D=RNFLOR(SPRIME)
      TPRIME=RNISPR(A,T)
      E=RNFLOR(TPRIME)
      CALL RNERAS(SPRIME)
      CALL RNERAS(TPRIME)
70    IF (ICOMP(D,E).NE.0) GO TO 71
      CALL ERLA(D)
      CALL ERLA(E)
      I=PFL(J1,I)
      GO TO 50
71    R1=RNINT2(E,A)
      CALL ERLA(D)
      CALL ERLA(E)
      IF (RIVAL(ACAP,R1).EQ.0) GO TO 72
      CALL RNERAS(R1)
      I=PFL(J1,I)
      GO TO 50
72    CALL ERASE(J1)
80    RCAP=PFL(R1,RCAP)
      IF (R1.NE.0) GO TO 81
      ONE=PFA(1,0)
```



```
      ERLA(ONE)
      =ONE
      TO 82
81    CALL FIRST2(B,C,R1)
      CCAP=PFA(1,PFL(INEG(B),PFA(0,0)))
      E=BORROW(C)
82    CCAP=PFL(PVBL(ACAP),PFL(E,CCAP))
      TCAP=PQ(BCAP,CCAP)
      CALL ERASE(BCAP)
      CALL ERASE(CCAP)
      BCAP=TCAP
      GO TO 50
      END


      SUBROUTINE PICPP(A,B,C)
      INTEGER A,B,C,BORROW,PICONT,PID,PONE
10    B=PICONT(A)
      IF (PONE(B).NE.1) GO TO 11
      C=BORROW(A)
      RETURN
11    C=PID(A,B)
      RETURN
      END


      INTEGER FUNCTION PMDEG(A)
      INTEGER A,PDEG,SECOND
1     PMDEG=0
      IF (A.EQ.0) RETURN
      PMDEG=PDEG(SECOND(A))
      RETURN
      END


      INTEGER FUNCTION PMDIF(A,B)
      INTEGER A,B,BPRIME,PMNEG,PMSUM
1     BPRIME=PMNEG(B)
      PMDIF=PMSUM(A,BPRIME)
      CALL PMERAS(BPRIME)
      RETURN
      END


      SUBROUTINE PMERAS(A)
      INTEGER A,C,T,COUNT
10    C=COUNT(A)-1
      IF (C.EQ.0) GO TO 11
      CALL SCOUNT(C,A)
      A=C
      RETURN
11    CALL DECAP(T,A)
```

```
      IF (A.EQ.0) GO TO 12
      CALL RNERAS(T)
      GO TO 10
12    CALL PERASE(T)
      RETURN
      END


      INTEGER FUNCTION PMIPOL(X,Y,Z)
      INTEGER A,ACAP,B,BCAP,D,OMEGA,P,PSI,T,X,Y,Z
      INTEGER BORROW,PDEG,PFA,PFL,PIP,PSREMG
      INTEGER PVBL,SECOND,TAIL,TYPE
      COMMON /TR7/PSI,OMEGA
1     ACAP=X
      M=Y
      D=Z
      P=1
      GO TO 10
2     PMIPOL=BCAP
      RETURN
10    IF (M.NE.0) GO TO 11
      PMIPOL=BORROW(ACAP)
      RETURN
11    IF (TYPE(SECOND(ACAP)).EQ.0) GO TO 20
      T=TAIL(ACAP)
      BCAP=0
      GO TO 30
20    IF (PDEG(ACAP).GE.PDEG(PSI)) GO TO 21
      BCAP=PIP(ACAP,D)
      GO TO (2,31),P
21    BCAP=PSREMG(ACAP,PSI,M)
      GO TO (2,31),P
30    IF (T.EQ.0) GO TO 40
      CALL ADV2(A,N,T)
      CALL STACK3(A,ACAP,BCAP)
      CALL STACK3(N,P,T)
      ACAP=A
      P=2
      GO TO 10
31    B=BCAP
      CALL UNSTK3(N,P,T)
      CALL UNSTK3(A,ACAP,BCAP)
      IF (D.NE.0) BCAP=PFA(N,PFL(B,BCAP))
      GO TO 30
40    BCAP=INV(BCAP)
      IF (BCAP.NE.0) BCAP=PFL(PVBL(ACAP),BCAP)
      GO TO (2,31),P
      END
```



```
      INTEGER FUNCTION PMLDCF(ACAP)
      INTEGER ABAR,ACAP,ACAPBR,B,C,P,S
      INTEGER BORROW,PFL,PLDCF,PONE,RNISPR
1     PMLDCF=0
      IF (ACAP.EQ.0) RETURN
      CALL FIRST2(R,ACAPBR,ACAP)
      B=PLDCF(ACAPBR)
      CALL PICPP(B,C,ABAR)
      IF (PONE(C).EQ.1) S=BORROW(R)
      IF (PONE(C).EQ.0) S=RNISPR(C,R)
      CALL PERASE(B)
      CALL ERLA(C)
      PMLDCF=PFL(S,PFL(ABAR,0))
      RETURN
      END


      INTEGER FUNCTION PMNEG(A)
      INTEGER A,ABAR,BBAR,R,BORROW,PFL,PNEG
1     PMNEG=0
      IF (A.EQ.0) RETURN
      CALL FIRST2(R,ABAR,A)
      BBAR=PNEG(ABAR)
      PMNEG=PFL(BORROW(R),PFL(BBAR,0))
      RETURN
      END


      INTEGER FUNCTION PMPOL(LCAP,W)
      INTEGER ABAR,A1,A1BAR,B,C,CCAP,D,E,R,T,U,V,W,X
      INTEGER BORROW,FIRST,PFA,PFL,PIP,RNINT2,SECOND
1     PMPOL=0
      IF (LCAP.EQ.0) RETURN
      T=LCAP
      U=LCAP
      L=PFA(1,0)
      B=0
2     IF (T.EQ.0) GO TO 3
      CALL ADV2(X,A1,T)
      C=SECOND(FIRST(A1))
      V=PLCM(D,L)
      CALL ERLA(L)
      L=V
      GO TO 2
3     IF (U.EQ.0) GO TO 4
      CALL ADV2(E,A1,U)
      CALL ADV2(R,A1BAR,A1)
      C=RNSPR(L,R)
      CCAP=PIP(A1BAR,C)
      CALL ERLA(C)
      B=PFL(CCAP,PFA(E,B))
      GO TO 3
```

```
4     B=PFL(BORROW(W),B)
      CALL PICPP(B,C,ABAR)
      R=RNINT2(C,L)
      CALL ERLA(C)
      CALL ERLA(L)
      CALL PERASE(B)
      PMPCL=PFL(R,PFL(ABAR,C))
      RETURN
      END


      INTEGER FUNCTION PMPROD(A,B)
      INTEGER A,ABAR,B,BBAR,C,R,S,U
      INTEGER PMRPOL,PPROD,RNPROD
1     PMPROD=0
      IF (A.EQ.0.OR.B.EQ.0) RETURN
      CALL FIRST2(R,ABAR,A)
      CALL FIRST2(S,BBAR,B)
2     U=RNPROD(R,S)
      C=PPROD(ABAR,BBAR)
      PMPROD=PMRPOL(U,C)
      CALL RNERAS(U)
      CALL PERASE(C)
      RETURN
      END


      INTEGER FUNCTION PMORMM(A,N,B1)
      INTEGER A,B1,C,CCAP,C1,D,G,R,T,V
      INTEGER BORROW,PFA,PFL,PMDEG,PMDIF,PMLDCF
      INTEGER PMPOL,PMRED,PMSPRD,PVBL,SECOND
10    CCAP=BORROW(A)
      IF (A.NE.C.AND.PMDEG(A).GE.N) GO TO 11
      PMORMM=PFL(G,PFL(CCAP,C))
      RETURN
11    T=0
20    IF (CCAP.EQ.0) GO TO 30
      L=PMDEG(CCAP)-N
      IF (L.LT.0) GO TO 30
      C=PMLDCF(CCAP)
      C1=PMRED(CCAP)
      CALL PMERAS(CCAP)
      D=PMSPRD(B1,C,L)
      CCAP=PMDIF(C1,D)
      CALL PMERAS(C1)
      CALL PMERAS(D)
      T=PFA(L,PFL(C,T))
      GO TO 20
30    R=CCAP
      V=PVBL(SECOND(A))
      G=PMPOL(T,V)
```



```
      CALL ERASE(T)
      CALL ERASE(V)
      PMORMM=PFL(G,PFL(R,C))
      RETURN
      END


      INTEGER FUNCTION PMRED(A)
      INTEGER A,ABAR,BBAR,C,CCAP,R,S
      INTEGER BORROW,PFL,PONE,PRED,RNISPP
1     PMRED=0
      IF (A.EQ.0) RETURN
      CALL FIRST2(R,ABAR,A)
      CCAP=PRED(ABAR)
      IF (CCAP.EQ.0) RETURN
      CALL PICPP(CCAP,C,BBAR)
      IF (PONE(C).EQ.1) S=BORROW(R)
      IF (PONE(C).EQ.0) S=RNISPR(C,R)
      CALL PERASE(CCAP)
      CALL ERLA(C)
      PMRED=PFL(S,PFL(BBAR,C))
      RETURN
      END


      INTEGER FUNCTION PMRPOL(R,A)
      INTEGER A,BBAR,C,CCAP,D,E,OMEGA,PSI,R,S,T
      INTEGER BORROW,PDEC,PDEGS,PFL,PLDCF
      INTEGER PMPOL,RNINT2,RNPROD
      COMMON /TR7/ PSI,OMEGA
10    PMRPOL=0
      IF (R.EQ.0.OR.A.EQ.0) RETURN
      L=INV(PDEGS(A))
      CALL DECAP(N,L)
      CALL ERLA(L)
      K=PDEC(PSI)
      IF (N.GE.K) GO TO 20
      PMRPOL=PFL(BORROW(R),PFL(BORROW(A),0))
      RETURN
20    M=N-K+1
      E=PLDCF(PSI)
      D=IPOWER(E,M)
      CALL ERLA(E)
30    CCAP=PMPOL(A,M,D)
      IF (CCAP.NE.0) GO TO 31
      CALL ERLA(D)
      RETURN
31    CALL PICPP(CCAP,C,BBAR)
      CALL PERASE(CCAP)
40    T=RNINT2(C,D)
      CALL ERLA(C)
      CALL ERLA(D)
```

```
      S=RNPROD(T,R)
      CALL RNERAS(T)
      PMRPOL=PFL(S,PFL(3BAR,0))
      RETURN
      END


      INTEGER FUNCTION PMSPRD(A,B,N)
      INTEGER A,ABAR,B,3BAR,D,DBAR,S
      INTEGER BORROW,PFA,PFL,PMPROD,PVBL,SECOND
1     PMSPRD=0
      IF (A.EQ.0.OR.B.EQ.0) RETURN
      ABAR=SECOND(A)
      CALL FIRST2(S,BBAR,B)
2     DBAR=PFL(PVBL(ABAR),PFL(BORROW(3BAR),PFA(N,0)))
      D=PFL(BORROW(S),PFL(DBAR,D))
3     PMSPRD=PMPROD(A,D)
      CALL PMERAS(D)
      RETURN
      END


      INTEGER FUNCTION PMSUM(A,B)
      INTEGER A,ABAR,B,BBAR,CBAR,C1,G,GBAR,H,R,R1
      INTEGER R2,R2BAR,S,S1,S2,S2BAR,T,T1,T1BAR
      INTEGER T2,T2BAR,U,UCAP,V,VCAP
      INTEGER BORROW,PFL,PIP,PONE,PSUM
10    IF (A.NE.0) GO TO 11
      PMSUM=BORROW(A)
      RETURN
11    IF (B.NE.0) GO TO 12
      PMSUM=BORROW(A)
      RETURN
12    CALL FIRST2(R,ABAR,A)
      CALL FIRST2(S,BBAR,B)
      CALL FIRST2(R1,R2,R)
      CALL FIRST2(S1,S2,S)
20    CALL IGCDCF(R2,S2,G,R2BAR,S2BAR)
      U=IPROD(S2BAR,R1)
      V=IPROD(R2BAR,S1)
      UCAP=PIP(ABAR,U)
      VCAP=PIP(BBAR,V)
      C1=PSUM(UCAP,VCAP)
      CALL ERLA(R2BAR)
      CALL ERLA(U)
      CALL ERLA(V)
      CALL PERASE(UCAP)
      CALL PERASE(VCAP)
30    IF (C1.NE.0) GO TO 31
      CALL ERLA(G)
      CALL ERLA(S2BAR)
      PMSUM=0
```



```
      RETURN
31    T1=IPROD(R2,S2BAR)
      CALL PICPP(C1,T1,CBAR)
      CALL PERASE(C1)
      CALL ERLA(S2BAR)
      IF (PONE(G).EQ.0) GO TO 40
      T=PFL(T1,PFL(T2,0))
      PMSUM=PFL(T,PFL(CBAR,0))
      CALL ERLA(G)
      RETURN
40    CALL IGCDCF(T1,G,H,T1BAR,GBAR)
      CALL ERLA(T1)
      CALL ERLA(G)
      CALL ERLA(GBAR)
      IF (PONE(H).NE.1) GO TO 41
      T2BAR=T2
      GO TO 50
41    T2BAR=IQ(T2,H)
      CALL ERLA(T2)
50    CALL ERLA(H)
      T=PFL(T1BAR,PFL(T2BAR,0))
      PMSUM=PFL(T,PFL(CBAR,0))
      RETURN
      END


      INTEGER FUNCTION PMVBL(A)
      INTEGER A,PVBL,SECOND
1     PMVBL=0
      IF (A.EQ.0) RETURN
      PMVBL=PVBL(SECOND(A))
      RETURN
      END


      SUBROUTINE PSMPRS(A,D,L,O)
      INTEGER A,AHAT,B,BHAT,C,D,DHAT,DPRIME,E,EIPRIM
      INTEGER F,H,P,PEXP,PRIME,S,T,U,V,W
      INTEGER BORROW,CPDEG,CPMOD,PDEG,PFA
      INTEGER PFL,PNORM,SECOND,TAIL
      COMMON /TP4/ PRIME,PEXP
10    N1=PDEG(A)
      N2=PDEG(B)
      PEXP=NORM(A)
      NEXP=NORM(B)
      C=IPROD(F,H)
      T=IPOWER(H,N1-N2+1)
      EIPRIME=IPROD(E,T)
      CALL IMFI(EIPRIM,2)
      CALL ERLA(T)
      CALL ERLA(F)
      CALL ERLA(H)
```

```
      W=PRIME
      L=0
      E=0
      D=0
      M=0
      N2BAR=N2
      T=0
20    IF(W.NE.0) GO TO 22
      PRINT 21
21    FORMAT(23H ALGORITHM PSMRS1 FAILS)
      STOP
22    CALL ADV(P,W)
      AHAT=CPMOD(P,A)
      IF (CPDEG(AHAT).EQ.N1) GO TO 23
      CALL CPERAS(AHAT)
      GO TO 20
23    BHAT=CPMOD(P,B)
      IF (CPDEG(BHAT).EQ.N2BAR) GO TO 30
      CALL CPERAS(AHAT)
      CALL CPERAS(BHAT)
      GO TO 20
30    CALL CGVRS2(P,AHAT,BHAT,LHAT,DHAT)
      CALL CPERAS(AHAT)
      CALL CPERAS(BHAT)
40    S=LEXORD(D,DHAT)
      IF (S.EQ.0) CALL ERLA(DHAT)
      IF (S.NE.1) GO TO 41
      CALL ERASE(LHAT)
      CALL ERLA(DHAT)
      GO TO 20
41    IF (S.NE.-1) GO TO 50
      CALL ERLA(D)
      D=DHAT
      DPRIME=TAIL(D)
      N2=N2BAR
      CALL ERASE(L)
      L=0
      CALL ERLA(E)
      E=PORROW(E3PRIM)
      CALL ERLA(M)
      M=PFA(1,D)
      J=2
50    I=2
      LPRIME=0
60    IF (LHAT.EQ.0) GO TO 70
      I=I+1
      CALL DECAP(L3HAT,LHAT)
      L3=0
      IF (L.NE.0) CALL DECAP(L3,L)
      IF (I.LE.J) CALL ERASE(L3HAT)
      IF (I.GT.J) L3=PFA(P,PFL(L3HAT,L3))
      LPRIME=PFL(L3,LPRIME)
```



```
      GO TO 60
70    I=INV(LPRIME)
      CALL IMFI(M,P)
80    IF (ICOMP(M,E).LE.0) GO TO 20
      IF (TAIL(DPRIME).NE.0) GO TO 81
      CALL ERLA(C)
      CALL ERLA(E3PRIM)
      CALL ERLA(E)
      CALL ERLA(M)
      RETURN
81    IF (T.NE.0) GO TO 90
      T=1
      N3=SECOND(DPRIME)
      U=IPOWER(C,N2-N3-1)
      V=IPROD(E,U)
      CALL ERLA(E)
      CALL ERLA(U)
      E=V
      GO TO 80
90    DPRIME=TAIL(DPRIME)
      IF (TAIL(DPRIME).NE.0) GO TO 91
      CALL ERLA(C)
      CALL ERLA(E3PRIM)
      CALL ERLA(E)
      CALL ERLA(M)
      RETURN
91    J=J+1
      T=0
      U=IPROD(E,C)
      CALL ERLA(E)
      E=U
      N2=N3
      GO TO 80
      END


      INTEGER FUNCTION PSREMG(A,BCAP,M)
      INTEGER A,B,BCAP,BPRIME,C,CCAP,CPRIME,D,T,U,V
      INTEGER BORROW,PDEG,PDIF,PLDCF,PPOWER,PRED,PSPROD
10    N=PDEG(A)
      K=PDEG(BCAP)
      M1=0
      B=PLDCF(BCAP)
      BPRIME=PRED(BCAP)
      CCAP=BORROW(A)
20    IF (CCAP.NE.0) GO TO 21
      CALL PERASE(B)
      CALL PERASE(BPRIME)
      PSREMG=CCAP
      RETURN
21    T=PDEG(CCAP)-K
      IF (T.LT.0) GO TO 30
```

```
      M1=M1+1
      C=PLDCF(CCAP)
      CPRIME=PRED(CCAP)
      U=PSPROD(CPRIME,B,C)
      V=PSPROD(BPRIME,C,T)
      CALL PERASE(CCAP)
      CALL PERASE(CPRIME)
      CALL PERASE(C)
      CCAP=PDIF(U,V)
      CALL PERASE(U)
      CALL PERASE(V)
      GO TO 20
30    T=M-M1
      IF (T.NE.0) GO TO 31
      CALL PERASE(B)
      CALL PERASE(BPRIME)
      PSPROD=CCAP
      RETURN
31    C=PPOWER(B,T)
      U=PSPROD(CCAP,B,C)
      CALL PERASE(B)
      CALL PERASE(BPRIME)
      CALL PERASE(C)
      CALL PERASE(CCAP)
      PSPROD=U
      RETURN
      END


      INTEGER FUNCTION RNABS(R)
      INTEGER R,R1,R2,BORROW,PFL
10    IF (R.NE.0) GO TO 11
      RNABS=0
      RETURN
11    CALL FIRST2(R1,R2,R)
      RNABS=PFL(IABSL(R1),PFL(BORROW(R2),0))
      RETURN
      END


      INTEGER FUNCTION RNCOMP(RCAP,SCAP)
      INTEGER R,RCAP,R1,R2,S,SCAP,S1,S2,RNSIGN
10    R=RNSIGN(RCAP)
      S=RNSIGN(SCAP)
      IF (R.NE.S) GO TO 11
      RNCOMP=-S
      RETURN
11    IF (R.EQ.S) GO TO 20
      RNCOMP=R
      RETURN
20    CALL FIRST2(R1,R2,RCAP)
      CALL FIRST2(S1,S2,SCAP)
      R=IPROD(R1,S2)
      S=IPROD(R2,S1)
      RNCOMP=ICOMP(R,S)
      CALL ERLA(R)
      CALL ERLA(S)
      RETURN
      END


      INTEGER FUNCTION RNDIF(R,S)
      INTEGER R,S,SPRIME,RNNEG,RNSUM
1     SPRIME=RNNEG(S)
      RNDIF=RNSUM(R,SPRIME)
      CALL RNERAS(SPRIME)
      RETURN
      END


      SUBROUTINE RNERAS(RCAP)
      INTEGER R,RCAP,COUNT
10    IF (RCAP.EQ.0) RETURN
      N=COUNT(RCAP)-1
      IF (N.EQ.0) GO TO 11
      CALL SCOUNT(N,RCAP)
      RETURN
11    CALL DECAP(R,RCAP)
      CALL ERLA(R)
      GO TO 10
      END


      INTEGER FUNCTION RNFLOR(RCAP)
      INTEGER RCAP,R1,R2,T
1     RNFLOR=0
      IF (RCAP.EQ.0) RETURN
      CALL FIRST2(R1,R2,RCAP)
      L=IDR(R1,R2)
      CALL DECAP2(N,T,L)
      IF(ISIGNL(T).LT.0) N=IMADD(N,1,-1)
      CALL ERLA(T)
      RNFLOR=N
      RETURN
      END


      INTEGER FUNCTION RNINT1(A)
      INTEGER A,BORROW,PFA,PFL
1     RNINT1=0
      IF (A.EQ.0) RETURN
      RNINT1=PFL(BORROW(A),PFL(PFA(1,0),0))
      RETURN
      END
```

```
      INTEGER FUNCTION RNINT2(A,B)
      INTEGER A,ABAR,B,BBAR,G,S,PFL
10    IF (A.NE.0) GO TO 11
      RNINT2=0
      RETURN
11    CALL IGCDCF(A,B,G,ABAR,BBAR)
      CALL ERLA(G)
20    S=ISIGNL(BBAR)
      IF (S.NE.1) GO TO 21
      RNINT2=PFL(ABAR,PFL(BBAR,C))
      RETURN
21    RNINT2=PFL(INEG(ABAR),PFL(INEG(BBAR),C))
      CALL ERLA(ABAR)
      CALL ERLA(BBAR)
      RETURN
      END


      INTEGER FUNCTION RNISPR(A,RCAP)
      INTEGER A,ABAR,G,RCAP,R1,R2,R2BAR,S1,S2,PFL
1     RNISPR=0
      IF (A.EQ.0.OR.RCAP.EQ.0) RETURN
      CALL FIRST2(R1,R2,RCAP)
2     CALL IGCDCF(A,R2,G,ABAR,R2BAR)
      S1=IPROD(ABAR,R1)
      S2=R2BAR
      CALL ERLA(G)
      CALL ERLA(ABAR)
      RNISPR=PFL(S1,PFL(S2,0))
      RETURN
      END


      INTEGER FUNCTION RNNEG(R)
      INTEGER R,R1,R2,BORROW,PFL
1     RNNEG=0
      IF (R.EQ.0) RETURN
      CALL FIRST2(R1,R2,R)
      RNNEG=PFL(INEG(R1),PFL(BORROW(R2),0))
      RETURN
      END


      INTEGER FUNCTION RNPROD(R,S)
      INTEGER C,R,R1,R1BAR,R2,R2BAR,S
      INTEGER S1,S1BAR,S2,S2BAR,T1,T2,PFL
10    IF (R.NE.0.AND.S.NE.0) GO TO 11
      RNPROD=0
      RETURN
11    CALL FIRST2(R1,R2,R)
      CALL FIRST2(S1,S2,S)
20    CALL IGCDCF(R1,S2,C,R1BAR,S2BAR)
```



```
      CALL ERLA(C)
      CALL IGCDCF(R2,S1,C,R2BAR,S1BAR)
      CALL ERLA(C)
30    T1=IPROD(R1BAR,S1BAR)
      T2=IPROD(R2BAR,S2BAR)
      RNPROD=PFL(T1,PFL(T2,0))
      CALL ERLA(R1BAR)
      CALL ERLA(R2BAR)
      CALL ERLA(S1BAR)
      CALL ERLA(S2BAR)
      RETURN
      END


      INTEGER FUNCTION RNQ(R,S)
      INTEGER R,S,SPRIME,RNPROD,RNRECP
10    IF (R.NE.0) GO TO 11
      RNQ=0
      RETURN
11    SPRIME=RNRECP(S)
      RNQ=RNPROD(R,SPRIME)
      CALL RNERAS(SPRIME)
      RETURN
      END


      INTEGER FUNCTION RNRECP(R)
      INTEGER R,R1,R2,BORROW,PFL
10    CALL FIRST2(R1,R2,R)
      IF (ISIGNL(R1).NE.1) GO TO 11
      RNRECP=PFL(BORROW(R2),PFL(BORROW(R1),0))
      RETURN
11    RNRECP=PFL(INEG(R2),PFL(INEG(R1),0))
      RETURN
      END


      INTEGER FUNCTION RNSIGN(R)
      INTEGER R,FIRST
10    IF (R.NE.0) GO TO 11
      RNSIGN=0
      RETURN
11    RNSIGN=ISIGNL(FIRST(R))
      RETURN
      END


      INTEGER FUNCTION RNSUM(R,S)
      INTEGER A,B,R,R1,R2,R2BAR,S,S1,S2,S2BAR
      INTEGER T1,T1BAR,T2,T2BAR,U1,U2
      INTEGER BORROW,FIRST,PFL,TAIL
```

[COL71a] Collins, G. E., "The SAC-1 Polynomial System", University of Wisconsin Computing Center Technical Report No. 2 (66 pages). Madison, Wisconsin, March 1971.

[COL71b] Collins, G. E., "The SAC-1 System: An Introduction and Survey", Proceedings of the Second Symposium on Symbolic and Algebraic Manipulation, Los Angeles, March 1971, pp. 144-152.

[COL71c] Collins, G. E., "The SAC-1 List Processing System", University of Wisconsin Madison Academic Computing Center Technical Report No. 24 (34 pages), Madison, Wisconsin, July, 1971.

[COL71d] Collins, G. E., "The SAC-1 Rational Function System", University of Wisconsin Madison Academic Computing Center Technical Report No. 8 (31 pages), Madison, Wisconsin, September, 1971.

[COL71e] Collins, G. E., "The Calculation of Multivariate Polynomial Resultants", Journal of the Association for Computing Machinery, Vol. 18, No. 4 (October 1971), pp. 515-532.

[COL72] Collins, G. E., "The SAC-1 Polynomial GCD and Resultant System", University of Wisconsin Madison Academic Computing Center Technical Report No. 27 (94 pages), Madison, Wisconsin, February, 1972.

[COL73a] Collins, G. E., "Computer Algebra of Polynomials and Rational Functions", American Mathematical Monthly, Vol. 80, No. 7 (August-September, 1973), pp. 725-755.

[COL73b] Collins, G. E., "The Computing Time of the Euclidean Algorithm", Siam Journal on Computing, to appear.

[COL73c] Collins, G. E., "Quantifier Elimination for Real Closed Fields by Cylindrical Algebraic Decomposition", in preparation.

[COL73d] Collins, G. E., "The SAC-1 Integer Arithmetic System-Version III", University of Wisconsin Madison Academic Computing Center Technical Report No. 31 (63 pages), Madison, Wisconsin, March 1973.

[COMC72] Collins, G. E. and McClellan, M. T., "The SAC-1 Polynomial Linear Algebra System", University of Wisconsin Madison Academic Computing Center Technical Report No. 28 (107 pages), Madison, Wisconsin, April 1972.

[COMU72] Collins, G. E. and Musser, D. R., "The SAC-1 Polynomial Factorization System", University of Wisconsin Madison Academic Computing Center Technical Report No. 30 (65 pages), Madison, Wisconsin, March 1972.

[COPI73] Collins, G. E. and Pinkert, J. R., "The SAC-1 Gaussian Polynomial Complex Zero System", University of Wisconsin Madison Academic Computing Center Technical Report, in preparation.

[HAWR45] Hardy, G. H. and Wright, E. M., An Introduction to the Theory of Numbers, Clarendon Press, (Oxford: 1945).

[HEI70] Heindel, L. E., "Algorithms for Exact Polynomial Root Calculation", Ph.D. Thesis, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 1970.

[HER64] Herstein, I. N., Topics in Algebra, Blaisdell Publishing Company, (Waltham: 1964).

[JAC51] Jacobson, N., Lectures in Abstract Algebra, Vol. I - Basic Concepts, D. Van Nostrand Company, (New York: 1951).

[KNU68] Knuth, D. E., The Art of Computer Programming, Vol. I: Fundamental Algorithms, Addison-Wesley, (Reading: 1968).

[KNU69] Knuth, D. E., The Art of Computer Programming, Vol. II: Seminumerical Algorithms, Addison-Wesley, (Reading: 1969).

[LOO73] Loos, R., "A Constructive Approach to Algebraic Numbers", (April 1973).

[MUS71] Musser, D. R., "Algorithms for Polynomial Factorization", Ph.D. Thesis, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 1971.

[PIN73] Pinkert, J. R., "Algebraic Algorithms for Computing the Complex Zeros of Gaussian Polynomials", Ph.D. Thesis, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 1973.

[POL50] Pollard, H., The Theory of Algebraic Numbers, Mathematical Association of America, (1950).

[SEI54] Seidenberg, A., "A New Decision Method for Elementary Algebra", Annals of Mathematics, Vol. 60, No. 2 (September 1954), pp. 365-374.

[TAR51] Tarski, A., A Decision for Elementary Algebra and Geometry, University of California Press, (Berkeley: 1951).

[VDW70] van der Waerden, B. L., Algebra, Vol. 1, Frederick Ungar Publishing Company, (New York: 1970).

```
10    IF (R.NE.0) GO TO 11
      RNSUM=BORROW(C)
      RETURN
11    IF (S.NE.0) GO TO 12
      RNSUM=BORROW(R)
      RETURN
12    CALL FIRST2(R1,R2,R)
      CALL FIRST2(S1,S2,S)
20    CALL IGCDCF(R2,S2,A,R2BAR,S2BAR)
      U1=IPROD(R1,S2BAR)
      U2=IPROD(R2BAR,S1)
      T1=ISUM(U1,U2)
      CALL ERLA(U1)
      CALL ERLA(U2)
      CALL ERLA(R2BAR)
      IF (T1.NE.0) GO TO 21
      CALL ERLA(A)
      CALL ERLA(S2BAR)
      RNSUM=0
      RETURN
21    T2=IPROD(R2,S2BAR)
      CALL ERLA(S2BAR)
      IF (FIRST(A).NE.1.OR.TAIL(A).NE.0) GO TO 30
      CALL ERLA(A)
      RNSUM=PFL(T1,PFL(T2,0))
      RETURN
30    B=IGCD(T1,A)
      CALL ERLA(A)
      IF (FIRST(B).NE.1.OR.TAIL(B).NE.0) GO TO 31
      CALL ERLA(B)
      RNSUM=PFL(T1,PFL(T2,0))
      RETURN
31    T1BAR=IQ(T1,B)
      T2BAR=IQ(T2,B)
      CALL ERLA(T1)
      CALL ERLA(T2)
      CALL ERLA(B)
      RNSUM=PFL(T1BAR,PFL(T2BAR,0))
      RETURN
      END
```


REFERENCES

[BRO71] Brown, W. S., "On Euclid's Algorithm and the Computation of Polynomial Greatest Common Divisors", Journal of the Association for Computing Machinery, Vol. 18, No. 4 (October 1971), pp. 478-504.

[BRTR71] Brown, W. S. and Traub, J. F., "On Euclid's Algorithm and the Theory of Subresultants", Journal of the Association for Computing Machinery, Vol. 18, No. 4 (October 1971), pp. 505-514.

[CACO73] Caviness, B. F., Collins, G. E., et al, "The SAC-1 Gaussian Integer and Gaussian Polynomial System", University of Wisconsin, Computer Sciences Department Technical Report, in preparation.

[COAL69] Collins, G. E. et al, "The SAC-1 Modular Arithmetic System", University of Wisconsin Computing Center Technical Reference No. 10 (50 pages), Madison, Wisconsin, June 1969.

[COHE70] Collins, G. E. and Heindel, L. E., "The SAC-1 Polynomial Real Zero System", University of Wisconsin Computing Center Technical Report No. 18 (72 pages), Madison, Wisconsin, August 1970.

[COHO70] Collins, G. E. and Horowitz, E., "The SAC-1 Partial Fraction Decomposition and Rational Function Integration System," University of Wisconsin Computing Center Technical Report No. 12 (47 pages), Madison, Wisconsin, February, 1970.

[COHO73] Collins, G. E. and Horowitz, E., "On the Minimum Root Separation of a Polynomial", Mathematics of Computation, to appear.

[COH69] Cohen, P., "Decision Procedures for Real and p-adic Fields", Communications on Pure and Applied Mathematics, Vol. XXII, No. 2 (March 1969), pp. 131-151.

[COLO73] Collins, G. E. and Loos, R., "Resultant Algorithms for Exact Arithmetic on Algebraic Numbers", in preparation.

[COL67] Collins, G. E., "Subresultants and Reduced Polynomial Remainder Sequences", Journal of the Association for Computing Machinery, Vol. 14, No. 1 (January 1967), pp. 128-142.

[COL69] Collins, G. E., "Computing Time Analyses for Some Arithmetic and Algebraic Algorithms", Proceedings of the 1966 Summer Institute on Symbolic Mathematical Computation, R. G. Tobey, Editor, IBM Boston Programming Center, (June 1969), pp. 195-231.

INDEX OF ALGORITHMS