THE HENSEL LEMMA IN ALGEBRAIC MANIPULATION

David Y. Y. Yun

Massachusetts Institute of Technology

Prepared for:

Office of Naval Research
Advanced Research Projects Agency

November 1974

BIBLIOGRAPHIC DATA SHEET	1. Report No. MAC TR- 138	2.	3. Recipient's Accession No. AD/A-002737
4. Title and Subtitle The Hensel Lemma in Algebraic Manipulation			5. Report Date: Issued November 1974 6.
7. Author(s) David Y. Y. Yun			8. Performing Organization Rept. No. MAC TR- 138
9. Performing Organization Name and Address PROJECT MAC; MASSACHUSETTS INSTITUTE OF TECHNOLOGY: 545 Technology Square, Cambridge, Massachusetts 02139			10. Project/Task/Work Unit No. 11. Contract/Grant No. N00014-70-A-0362-0006
12. Sponsoring Organization Name and Address Office of Naval Research Department of the Navy Information Systems Program Arlington, Va 22217			13. Type of Report & Period Covered: Interim Scientific Report 14.

15. Supplementary Notes

Ph.D. Thesis, MIT Department of Mathematics, November 1973

16. Abstracts: New and improved algorithms for computation in several fundamental polynomial operations are presented. The common basis for these algorithms are generalizations of the p-adic technique used in the constructive proof of the Hensel Lemma. Multivariate polynomial operations are stressed due to the special importance of the multivariate Hensel-type construction in replacing the modular evaluation-and-interpolation technique under certain conditions. Due to the availability of numerous methods for the computation of polynomial greatest common divisors (GCD), the EZGCD Algorithm is given special emphasis. Both theoretically and by actual computing data, this new algorithm demonstrates promising efficiencies by taking advantage of the sparseness of multivariate polynomials. An intuitive and more "engineering" approach to computing time analysis also appears to give quite accurate predictions of actual run times for many practical problems. Other applications of the Hensel-type constructions resulting in improved algorithms for computing polynomial factorizations, contents and primitive parts, and square-free decompositions are also described.

17. Key Words and Document Analysis. 17a. Descriptors

Algebraic manipulation
Algorithms
Greatest common divisor
P-adic construction
Polynomial operations
Modular arithmetic
Multivariate polynomials

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report)	21. No. of Pages
Approved for Public Release; Distribution Unlimited	UNCLASSIFIED	260
	20. Security Class (This Page) UNCLASSIFIED	22. Price

/

THE HENSEL LEMMA
IN ALGEBRAIC MANIPULATION

by

David Y. Y. Yun

November, 1973

PROJECT MAC

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Cambridge, Massachusetts 02139

THE HENSEL LEMMA IN ALGEBRAIC MANIPULATION

by

David Y. Y. Yun

Submitted to the Department of Mathematics on November 13, 1973, in partial fulfillment of the requirements for the Degree of Doctor of Philosophy.

ABSTRACT

New and improved algorithms for computation in several fundamental polynomial operations are presented. The common bases for these algorithms are generalizations of the p-adic technique used in the constructive proof of the Hensel Lemma. Multivariate polynomial operations are stressed due to the special importance of the multivariate Hensel-type construction in replacing the modular evaluation-and-interpolation technique under certain conditions. Due to the availability of numerous (not completely satisfactory) methods for the computation of polynomial greatest common divisors (GCD), the EZGCD Algorithm based on the Hensel construction is given special emphasis. An intuitive computing time analysis and many empirical experiments are made to compare the performance of this algorithm with two other major methods, especially the Modular GCD Algorithm. Both theoretically and by actual computing data, the new EZGCD Algorithm demonstrates promising efficiencies by taking advantage of the sparseness of multivariate polynomials. An intuitive and more "engineering" approach to computing time analysis also appears to give quite accurate predictions of actual run times for many practical problems. Other applications of the Hensel-type constructions resulting in improved algorithms for computing polynomial factorizations, contents and primitive parts, and square-free decompositions are also described.

THESIS SUPERVISOR: Joel Moses
TITLE: Associate Professor of Computer Science and Engineering

ACKNOWLEDGEMENT

Work reported herein was supported in part by Project MAC, an M.I.T. interdepartmental laboratory sponsored by the Advanced Research Projects Agency (ARPA), Department of the Defense, under Office of Naval Research Contract N00014-70-A-0362-0006. It was also supported in part by the Graduate Fellowship Program, Division of Graduate Education in Science, National Science Foundation.

The author wishes to express his gratitude to Professor Joel Moses for his continual innovations, guidance, encouragement, and criticism. He also wishes to thank Professor Seymour Papert for his constant advice and supervision in the Mathematics Department. The author wishes to thank Dr. Paul Wang for many helpful discussions and useful suggestions. Special thanks should also go to Roselyn, the author's wife, for her tireless efforts in helping to type and read the manuscript of this thesis as well as her providing constant comfort, understanding, and moral support through the author's graduate career.

TABLE OF CONTENTS

CHAPTER VIII. CONCLUSIONS AND

SUGGESTIONS FOR FUTURE RESEARCH

CHAPTER I. INTRODUCTION

I - 1 Introduction

With the development of modern computing machines, many numerical algorithms became powerful tools used by an increasing number of people. Technical terms such as polynomial approximation, evaluation and interpolation, and numerical quadratures became common words to almost anyone with a college mathematical background. A basic drawback in numerical methods lie in their painstaking concern for numerical errors, truncation and round-off, and the fact that in many cases, the resulting numbers do not provide enough information for interpreting physical phenomena or gaining significant insight. In such cases these methods have failed, since as R. W. Hamming said, "The purpose of computing is insight, not numbers".

Research in the field of symbolic and algebraic manipulation was initiated over a decade ago, based upon this spirit as well as the general perceived need. By dealing mainly with exact numbers, infinite precision integers and rational numbers, and algebraic expressions in terms of their symbolic representations, computers can now eliminate some of the sources of numerical error and aid scientists in performing many non-numeric computations. Many software systems devoted to various classes of symbolic

computations have been developed since 1960 or so. They have been used and tested in many different ways, but they do not have the wide acclaim that the numerical software routines have achieved. Various arguments can be made to explain this - some symbolic packages are highly specialized and some systems are still in embryonic development stages. There is, however, one common reason for the lack of widespread usage for many of these systems: the rapid exhaustion of the allotted data storage spaces for symbolic expressions due to the growth of expressions during computations. This expression growth problem can be divided into two categories. One is the blowup of the resulting expression of a computation. A good example for that is the computation of the determinant of a matrix. The determinant of an n by n numerical matrix is still a number, but a determinant of symbolic matrix may contain n factorial terms. The other category is the blowup of intermediate expressions when the final result of the computation is quite small and manageable. For this, the problem of computing the greatest common divisor (GCD) demonstrates this phenomenon [KNU69]. The GCD of two integers can be easily computed by the classical Euclidean algorithm, but the similar method for computing the GCD of two polynomials in just one variable over the integers can have enormous intermediate results even though the actual GCD could simply

be 1 [KNU69] [BRO71]. When polynomials in several variables are involved the blowup may be even worse. Actually, the distinction between the two categories is not as clear cut as the above examples have made it appear. When a computation involves many different steps, blowup of the first kind in any step can be considered in the second category, if only the final result of the entire computation is important. Thus any of these blowup problems will be referred to as the "expression growth" problem.

The solution to this common problem in symbolic manipulation seems to lie in careful analysis of existing computational algorithms and discoveries of more efficient new methods. (Although faster computers with more memory will also help, of course.) Much recent research in symbolic manipulation has been directed toward algorithm analysis. Perhaps the most important and useful collection of papers on algorithms (and also systems) in symbolic manipulation up to 1971 is the "Proceedings of the Second Symposium on Symbolic and Algebraic Manipulation" [PET71]. Many important surveys, tutorials, and current research papers in all areas of the field are presented. Since polynomial and rational function operations and simplification form the fundamental basis of any symbolic manipulation system, they represent a large portion of the papers on symbolic algorithms. Among these algorithms a noticeable common

theme is the use of modular arithmetic and interpolation techiques. At that time, tne common sentiment seemed to be a belief in the general applicability of the modular and evaluation mappings for many basic polynomial calculations. It is one of the major goals of this thesis to show how misleading this idea has been for several classes of problems, especially the computation of multivariate polynomial GCD's.

Another major algorithmic advance in this fieid has been in the problem of polynomial factorization. Berlekamp [BER67]. Zassenhaus [ZAS69], Musser [MUS71], Wang, and Rothschild [W&R73] have been instrumental in making this advance. The essential innovation in accomplishing the task of factoring polynomials is another classical idea of p-adic constructions in the Hensel's Lemma [VDW49]. It turns out that this method is not only applicable to factorization but also to many other polynomial operations, especially GCD's. We are especially interested in the computation of GCD's since there are many known methods of performing this task and none seems satisfactory for a wide spectrum of practical problems.

The use of Hensel-type constructions in polynomial factorization was crucial because it is very uneconomical to perform many factorizations using the Berlekamp algorithm. The Hensel-type construction takes advantage of the special

structure of a single modular factorization and extends this modular image to a full factorization. As some of these Hensel-type algorithms were being developed, dissatisfaction with the computing times of polynomial GCD's using known algorithms was mounting. The modular GCD algorithm, which also uses modular and evaluation mappings similar to the factorization algorithms, was performing poorly for large class of practical problems run on the symbolic manipulation system MACSYMA [MAC73]. It was upon Moses' suggestion that the similarities between the problems of factorization and GCD computation were studied and the Hensel-type constructions were tested for applications other than factorization. Not only were new algorithms discovered, in particular, major improvements of the Rothschild-Wang version of Hensel construction, but also methods of analyzing the algorithms were innovated. Previously, the analysis of many algorithms that has been geared excessively toward either idealized situations or worst-case situations. We will present a more intuitive approach of analyzing some of our main algorithms for cases which are too complex for step-by-step analysis. Besides, when we are dealing with multivariate polynomials, the basic operation for each step may be very different so as to make the detailed analysis misleading and polynomials are often sparse (most of the possible terms are zero) rather than dense so as to make

worst-case analysis inoperative. We will show that our intuitive "engineering" approach actually produces timing formulae which will predict the computing costs to within 15% in most cases.

With this brief description of the sequence of events leading to the current research, we will begin to present a more detailed overview of the thesis in the following section. An earlier version of some results in this thesis, in particular, those of Chapter III, appears in Moses and Yun [M&Y73].

I - 2 An Overview and Some Essential Concepts

Ever since the publication of Euclid's algorithm in Book 7 of his Elements (300 B. C.) for computing the positive greatest common divisor of two given positive integers, its various generalizations for computing polynomial GCD's have been useful computational tools for centuries (as many number theorists and algebraists can well substantiate). However it was the onset of research in symbolic and algebraic manipulation that touched off numerous significant results on the theoretical and mathematical basis, algorithmic descriptions, and computer implementations of these methods. In particular, Brown's [BRO71], Collins' [COL67], and Brown and Traub's [B&T71] works provide a complete background as well as excellent

technical material on this subject.

For the computation of univariate or multivariate polynomial GCD's over the domain of the integers or the rationals, the most popular computational methods are the Reduced or Subresultant PRS algorithms and the Modular algorithm. We will briefly discuss these methods in this section which will provide a basis for comparing analytical and computational results with those of the new Hensel-type GCD algorithms which we will present later.

In this section, we will give an overview of the material contained in this thesis. Our principal goal is to present several new methods of performing some essential polynomial operations based on the Hensel-type constructions. One major application of the Hensel construction is the polynomial GCD computation which we will describe in detail in Chapter III. Other applications will be presented in later chapters, but the analysis and empirical experiments will concentrate on the Hensel constructions and GCD computations. The concept of these polynomial operations can be easily extended to more general algebraic domains (as Musser did in [MUS71]). But, in keeping with the principles of symbolic manipulation where the stress is on exact arithmetic, we will be dealing mainly with polynomials over the integers, Z; the integers modulo q, Z/q,; or the rationals, Z/Z; in these domains many computational advan-

tages can be more easily demonstrated.

Since polynomials are our main concerns, we first need some basic algebraic concepts about polynomials. Only the most essential algebraic notions will be defined here, others will be referred to the Appendix or texts on modern algebra such as Herstein [HER64] or Birkoff and MacLane [B&M65].

The domain of integers will be denoted by Z; the set of equivalence classes in Z with respect to a congruence relation moodulo some integer q by (Z/q); and the field of rational numbers by (Z/Z). The ring of polynomials with indeterminants x,y,...,z over a coefficient domain J will be denoted by J[x,y,...,z]. Note that we can also write this polynomial ring as J'[x] where now J' denotes J[y,...,z]. This conceptually forces all polynomials to be expressed with x as the main variable with coefficients in J'. J[x,y,...,z] is an integral domain if J is [HER64, p. 123]. In most cases of the following chapters, we will consider univariate or multivariate polynomial domains I[x] or I[x,y,...,z] where I is often Z or Z/q and q is a prime power. Divisions of polynomials $F = f_m x^m + \ldots + f_1 x + f_0$ and $G = g_n x^n + \ldots + g_0$ over a field to obtain the quotient

polynomial $Q = q_{m-n} x^{m-n} + \ldots + q_0$ is done as follows:

For $k = m - n$ to $k = 0$, do $q_k = f_{n+k} \, g_n$ and

$f_j = f_j - q_k \, g_{j-k}$, $j = n+k-1, \ldots, k$. Let J be a unique factorization domain (u.f.d.), then J[x] is also a u.f.d. whose units are units of J. The process of polynomial division with remainder cannot always be made in J[x], since the coefficient domain is not necessarily a field where exact division can always be carried out. However, the process of pseudo-division can always be carried out [KNU69]. Given F, G in J[x] unique pseudo-quotient $Q = \text{pquo}(F, G)$ and pseudo-remainder $R = \text{prem}(F, G)$ can be found in J[x] such that $g^{d+1} F = Q G + R$ and $\deg(R) < \deg(G)$ where $g = \text{lc}(G)$, the leading coefficient of G, and $d = \deg(F) - \deg(G)$.

For non-zero F and G in J[x], F and G are said to be similar, denoted by $F \sim G$, if there exist f, g in J such that $f F = g G$. Let $F_1 = F$ and $F_2 = G$ and assume $\deg(F) > \deg(G)$, then a sequence of remainders $F_1, F_2, \ldots, F_k$ can be generated by pseudo-division such that

$F_i \sim \text{prem}(F_{i-2}, F_{i-1})$ for $i = 3, \ldots, k$, and

$\text{prem}(F_{k-1}, F_k) = 0$. This is called the polynomial remainder

sequence (PRS) [BRO71].

Let $G_1, G_2, \ldots, G_n$ be elements of an integral domain.

Then D is the greatest common divisor (GCD) of $G_1, \ldots, G_n$ iff

(1) D divides $G_1, G_2, \ldots$, and G_n.

(2) Every common divisor of $G_1, G_2, \ldots$, and G_n also divides D.

(3) D is unit normal. In a u.f.d., there always exists a unique GCD for a given set of elements. If GCD of two elements F, G in the u.f.d. equals 1, gcd(F,G) = 1, then we say that F and G are relatively prime. A polynomial P in J[x] is primitive if the GCD of all its non-zero coefficients is 1. Each polynomial P in J[x] has a unique representation of the form P = cont(P) pp(P), where cont(P), content of P, is the unit normal GCD of the non-zero coefficients of P and pp(P) = P/cont(P) is the remaining primitive part of P.

The process of computing the PRS for two polynomials F and G in J[x] is a generalization of the classical Euclid's algorithm which constructs a integer remainder sequence (IRS) [KNU69]. It is clear from the process of computing the PRS (or IRS) that

$$\gcd(F_1 = F, F_2 = G) \sim \gcd(F_2, F_3) \sim \ldots \sim \gcd(F_{k-1}, F_k) \sim F_k.$$

Thus the method of PRS yields the desired GCD up to simila-

rity. As pointed out by Brown [BR071], if F'_1 and F'_2 are non-zero polynomials in J[x] with $\deg(F'_1) \geq \deg(F'_2)$ and D' is their GCD, then the PRS algorithm can be used to determine this GCD D' uniquely as follows:

Let $c_1 = \mathrm{cont}(F'_1)$, $c_2 = \mathrm{cont}(F'_2)$, $c = \gcd(c_1, c_2)$, $F_1 = \mathrm{pp}(F'_1)$, $F_2 = \mathrm{pp}(F'_2)$, and $D = \gcd(F_1, F_2)$. If now $F_1, F_2, \ldots, F_k$ is a resulting PRS, then it is clear that $D = \mathrm{pp}(F_k)$ and $D' = c\,D$. This is the so called Euclidean PRS algorithm [COL67] as it is the obvious generalization of Euclid's algorithm to polynomials over a u.f.d. which is not necessarily a field.

Polynomial GCD computations basically rely on division or pseudo-division over a u.f.d. Thus the straight-forward generalization of the Euclid's algorithm for integers for computing the GCD of polynomials F and G over a field is simply:

$$F_1 = F,\ F_2 = G,\ F_i = -Q_{i-1}\,F_{i-1} + F_{i-2},\ \deg(F_i) < \deg(F_{i-1}),$$

$i=3,\ldots,k$. For polynomials over a u.f.d., then, it is necessary to use the PRS generated by pseudo-division. So we use the following rule instead:

$$\beta_i F_i = -Q_{i-1}\,F_{i-1} + \alpha_i F_{i-2},\ \deg(F_i) < \deg(F_{i-1}),\ i=3,\ldots,k.$$

The α_i and β_i are particular chosen elements of the coefficient u.f.d. and the epecification of them determines the particular PRS algorithm intended in the discussion. The α_i's are simply the necessary multiplier of pseudo-division. We briefly list several such PRS GCD algorithms according to the choices of β_i's. (For more detailed discussions of these algorithms, see [BRO71].)

(a) The Euclidian PRS Algorithm: $\beta_i = 1$, $i=3,\ldots,k$.

(b) The Primitive PRS Algorithm:

$\beta_i = \mathrm{cont}(\mathrm{prem}(F_{i-1},F_{i-2}))$, so that $F_i = \mathrm{pp}(\mathrm{prem}(F_{i-1},F_{i-2}))$, $i=3,\ldots,k$.

(c) Collins' Reduced PRS Algorithm:

$\beta_3 = 1$, $\beta_i = \alpha_{i-1}$, $i=4,\ldots,k$.

(d) The Subresultant PRS Algorithm:

$\beta_3 = (-1)^{\delta_1+1}$, $\beta_i = -f_{i-2}\psi_i^{\delta_{i-2}}$, $i=4,\ldots,k$ where

$d_i = \deg(F_i)$, $\delta_i = d_i - d_{i+1}$, $f_i = \mathrm{lc}(F_i)$, $i=1,\ldots,k$ and

$\psi_3 = -1$, $\psi_i = (-f_{i-2})^{\delta_{i-3}}\psi_{i-1}^{1-\delta_{i-3}}$, $i=4,\ldots,k$.

Among the above algorithms, the Euclidean PRS Algorithm suffers from coefficient growth due to the use of the pseudo-division multiplier. This expression explosion can sometimes, especially for multivariate polynomials where the multiplier is polynomial, be so big as to make the GCD computation prohibitively expensive. The Primitive PRS Algorithms requires a content computation at each step of the algorithm which may take a significant part of the total computing time, especially in multivariate cases again. Both the Reduced and the Subresultant PRS Algorithms are designed to avoid this content operation. As pointed out by Brown [BR071] even though there is a trade-off between the cost of computing the content at each step and the advantage of having possibly smaller coefficients to work with, in most cases the compensation is very little. The Subresultant PRS Algorithm is a further improvement of the Reduced PRS Algorithm in that it even avoids gross inefficiencies when the PRS is abnormal (when in a PRS $F_1, \ldots, F_k$, there is some $i > 1$, $\deg(F_i) - \deg(F_{i+1}) > 1$).

As an illustration of the severity of the growth of the coefficients in the PRS, let us consider univariate polynomials where N bounds the absolute magnitude of the coefficients in F_1 and F_2. Let $N^{**}a$ bound the coefficients

of F_j. Then, for the Euclidean PRS Algorithm with a normal PRS, we have $a_j = 2\,a_{j-1} + a_{j-2}$ as a recurrence relation for the sequence $\{a_j\}$. Thus $a_{j+1}/a_j \dashrightarrow 2.414$ as j becomes large. Therefore, if $\deg(F_1) = n + 1$ and $\deg(F_2) = n$, then the coefficient growth in the PRS is like N**(2.414**n) which is an exponential growth in the number of digits of the coefficients. For Collins' Reduced PRS GCD Algorithm with normal PRS, Knuth [KNU69] shows that the coefficient growth is proportional to N^{2n+1} which is only a linear growth in the number of digits. Nevertheless, the growth is still very large (see [KNU69] and [BRO71] for some examples), especially when the GCD itself is relatively small, say 1.

We now turn our attention to an algorithm which clearly avoids this basic difficulty - the Modular GCD Algorithm. The Modular GCD Algorithm, as well as many other modular algorithms, basically relies on two common forms of algebraic homomorphisms:

(1) The <u>modular</u> homomorphism, for integers modulo a prime p.

  (a) Computing a homomorphic image - getting the residue of an integer modulo p.

  (b) Inverting the mapping - applying the Chinese Re-

mainder Theorem (Garner's rule) to the residues w.r.t. (with respect to) different primes.

(2) The evaluation homomorphism, for polynomials in x modulo x-b.

(a) Computing a homomorphic image - evaluating the polynomial at a point b.

(b) Inverting the mapping - interpolating a polynomial from its values at different points.

Here, we will say that the prime, p, or the evaluation value, b, is lucky for a given polynomial if the homomorphic images of two distinct divisors of the given polynomial do not become identical in the image domain. (See also [BR071])

The modular GCD Algorithm basically applies the fundamental concepts of these two homomorphisms on the integral coefficients and recurses on each of the variables in the given polynomials. Without going into the details of the Modular Algorithm (for that see [BR071]), we list the steps of a general modular algorithm which is used for each variable and the integral coefficients:

(1) Estimate the number of homomorphic images required.

(2) Compute a new homomorphic image.

(3) Solve the problem in the image domain.

(4) Apply the inverse-mapping algorithm to include this new information.

(5) Test if sufficient number of image solutions have been

computed. If not go back to (2). If so, verify the current result and exit with it if correct.

Brown [BRO71] performed a "worst case" computing time analysis which indicated that the Modular GCD Algorithm would outperform the Subresultant Algorithm for sufficiently large problems. (Here, "large" often means dense or nearly dense polynomials of high degree in many variables.) Collins [COL71] presented a similar timing analysis and gave some empirical computing results which demonstrated the superiority of the Modular polynomial resultant algorithm to the Reduced PRS resultant algorithm.

Practical experiences with the Modular GCD and resultant algorithms have shown that in some cases these algorithms take a disastrously long time, often much longer than the comparable Reduced PRS algorithms. The reason for this is that the Modular algorithms are geared to the "worst" cases and are remarkably insensitive to the sparseness of the polynomial inputs and outputs. To illustrate this insensitivity, we give a simple case which should be familiar to anyone with some background in numerical analysis. Any interpolation technique used to compute the polynomial x**20 + 1 requires 21 values of this polynomial at 21 distinct points, even though there are only two terms in the polynomial. Thus a modular algorithm which inherently assumes denseness of the polynomials might be

inferior to a non-modular one, if the solution by the evaluation-interpolation process is in fact x**20 + 1. A completely dense polynomial in v variables of degree d in each has (d + 1)**v terms. This number is an exponential function of the number of variables.

If the GCD of two polynomials is a multivariate polynomial in v variables of degree d each, then the number of evaluations and interpolations performed by the Modular GCD Algorithm is at least (d + 1)**v, regardless of the number of terms in the original polynomials and in the GCD. Thus

if the GCD of two polynomials is $x_1^{10} + x_2^{10} + \ldots + x_{10}^{10}$

(e.g. P = this polynomial and Q = x P), the Modular Algorithm would take days or weeks to compute it on existing machines. On the other hand, the Reduced PRS GCD algorithm might take less than a second of computing time.

The Modular algorithm performs at its best when the original polynomials are univariate or when the resulting GCD is 1, since this cass involves few evaluations and univariate modular GCD calculations and no interpolation. The Reduced or Subresultant Algorithms are at their best when the input polynomials are very small (e.g. polynomials of degree 2 or less in each variable) since they require little overhead, or when the GCD is almost as large as the original polynomials (as is the case for P and Q defined

above). This is the case when the number of polynomial remaindering operations required to obtain the GCD is small, say 1. The Modular algorithm degrades as an exponential function of the number of variables and the logarithm of their degrees in the GCD. The Reduced or subresultant algorithms degrade quite rapidly, though the analysis here is not nearly as complete or elegant. The degradation is roughly a exponentially growing function of the length of the polynomial remainder sequence ([BRO71], [COL67]) required to obtain the GCD. For these two algorithms, small answers (e.g. 1) require the most work.

According to Brown's analysis, the computing time for the Reduced PRS GCD Algorithm is proportional to

$$L^2 (d+1)^{4v\,2^{2v}} 2^{v^3}$$

where L is the maximum integer length of the coefficients and d is the maximum degree in all v variables. The computing time for the Modular GCD Algorithm is proportional to $L^2 (d+1)^v + L (d+1)^{v+1}$. Comparing these computing times, we can see the important feature of the Modular algorithm. That is, as the number of terms in the two input polynomials grows toward the exponential number of terms in a dense polynomial, the Modular algorithm can obtain the answer much faster than the Reduced algorithm and even faster than a standard multiplication or division

of the two polynomials which would require on the order of $(d + 1)^{2v}$ steps. In practice, however, one is hardly likely to ever confront polynomials so large and dense that their multiplication will be slower than their GCD calculations by the Modular algorithm. To offset this potential advantage of the Modular algorithms is the fact, already noted, that they are insensitive to the possibility of sparseness in the original polynomials and the results. Polynomials in many variables of high degree are invariably sparse in practice, and in fact, exceedingly sparse. Furthermore, if one is going to deal with truly gigantic polynomials then there exist "fast" polynomial multiplication algorithms [BON73] which would lower the cost of polynomial multiplication (and division) to a level which is close to that of the Modular algorithm. This would improve the computation cost for any algorithm requiring multivariate multiplication and division in the dense cases.

Since not one of these algorithms performs well on the whole spectrum of input polynomials, there is much room for improvement. The EZGCD (Extended Zassenhaus GCD) Algorithm presented later appears to outperform the Modular GCD Algorithm in most practical situations. It will not outperform the Reduced or Subresultant Algorithms in those situations when these algorithms are at their best. It

does, however, give a much better accounting of itself in such cases than the Modular algorithm would.

As will be seen, situations which may present some difficulty to the EZGCD algorithm are those where the GCD is not relatively prime to either of the co-divisors (or cofactors, i.e. the quotients of original polynomials by the GCD) of the input polynomials. Then the EZGCD algorithm will resort to a variant of a square-free decomposition of the input polynomial. Additional difficulties arise when no variable has a constant leading coefficient in either of the input polynomials. And the most serious difficulty will be cases where no variable can have zero as an evaluation point.

The essential difference between the EZGCD algorithm and the Modular GCD algorithm is in the treatment of the multivariate case. When the Modular algorithm is at its best (univariate, small GCD, or dense inputs and outputs), the performance will usually be comparable.

As we mentioned before, the new methods for computing polynomial GCD and performing several other polynomial operations are based upon the Hensel-type constructions. It turns out that these Hensel-type methods are closely analogous to the evaluation-interpolation methods of numerical analysis and the above described modular methods. As numerical analysis draws from the mathematical field of

real and complex analysis for its computational methods and theoretical supports, it appears that symbolic manipulations new insights from modern algebra, in particular, the studies of geometric behavior of polynomials - algebraic geometry.

The classical Hensel's Lemma has its fundamental importance in the studies of p-adic analysis and valuation theory. The use of this constructive method in the problem of polynomial factorization was first suggested by H. Zassenhaus [ZAS69]. Musser [MUS71] presented this method and the Zassenhaus' quadratic extension technique as algorithms for extending factorizations of polynomials in general algebraic domains. Some computational inefficiencies result from considerations in such a general context especially for multivariate polynomials over the integers. A generalization of the Hensel algorithm based on the constructive proof of the Hensel's reducibility criterion [VDW49] was devised by Wang and Rothschild [W&R73] in the context of factoring multivariate polynomials. A (v+1)-variable polynomial can be considered as a generalized Taylor series, as we will see shortly. The generalized Hensel construction is devised to recover the codivisors of a given polynomial in the generalized Taylor series form with one main variable and a v-dimensional coefficient space. This is a distinct point of view from the original Wang and Rothschild version and our discussions of the

generalized Hensel construction will be presented from this new and different standpoint which will prove to be very beneficial.

The method essentially amounts to rediscovering the generalized Taylor series forms of the divisors term by term from their linearly independent "values" in the univariate polynomial space and the original polynomial. Such a process is quite a familiar one to many algebraic geometers as we discovered in the course of our work. The "evaluation" stage is known as the "specialization" process and the reconstruction method is often called the "deformation" process (one probable reason for calling it such is because the polynomial and its factors are now transformed to Taylor series forms corresponding to the specific chosen values, hence deformed to a space with a new origin). Interestingly the entire process of specialization and deformation is known as the "Newton's method". As we will see later, the parallelism between the evaluation-interpolation method and this specialization-deformation Newton's method is quite striking. However, we should note that the role of Hensel's Lemma in algebraic geometry was pointed out to us very late in our research.

Let us first get a good conceptual understanding of the generalized Taylor series representation of multivariate

polynomials. For a multivariate polynomial with integral coefficients, $P(x,y_1,\ldots,y_v)$ in $Z[x,y_1,\ldots,y_v]$, and for a given set of integers $b = (b_1,\ldots,b_v)$, we denote $P(x,y_1=b_1,\ldots,y_v=b_v)$ or $P(x,b_1,\ldots,b_v)$ by $P_b(x)$. We will also write P in a general Taylor series form:

$$P(x,y_1,\ldots,y_v) = P(x,b_1,\ldots,b_v)$$

$$+\sum_{i=1}^{v} p_i(x)\,(y_i - b_i) \qquad \text{(all terms of degree 1 in some } y)$$

$$+\sum_{i=1}^{v}\sum_{j=1}^{v} p_{i,j}(x)\,(y_i - b_i)\,(y_j - b_j) \qquad \text{(all terms of total degree 2 in the y's)}$$

$$+\sum_{i=1}^{v}\sum_{j=1}^{v}\sum_{k=1}^{v} p_{i,j,k}(x)\,(y_i - b_i)\,(y_j - b_j)\,(y_k - b_k)$$

$$+ \ldots$$

This series is finite and unique for a given set of b's, since P is a polynomial. We let

$P^{(0)}(x,y_1,\ldots,y_v) = P(x,b_1,\ldots,b_v) = P_b(x)$ and

$$P^{(m)}(x,y_1,\ldots,y_v) = \sum_{i_1,\ldots,i_m=1}^{v} p_{i_1,\ldots,i_m}(x) \prod_{j=1}^{m} (y_{i_j} - b_{i_j})$$

Then $P(x,y_1,\ldots,y_v) = \sum_{m=0}^{n} P^{(m)}(x,y_1,\ldots,y_v)$, for some

positive integer n. If we define "Im" to be a typical index

set $(i_1,\ldots,i_m)$ in $P^{(m)}$ where for $j = 1,2,\ldots,m$, i_j is in

{1,2,...,v} and "$Mm(y_1,\ldots,y_v)$" to be the <u>monomial</u> of the

$(y_{i_j} - b_{i_j})$'s indexed by Im, $Mm(y_1,\ldots,y_v) = \prod_{j=1}^{m} (y_{i_j} - b_{i_j})$,

then $P^{(m)} = \sum_{Im} p_{Im}(x)\, Mm(y_1,\ldots,y_v)$ and

$P(x,y_1,\ldots,y_v) = \sum_{m=0}^{n} \sum_{Im} p_{Im}(x)\, Mm(y_1,\ldots,y_v)$. The finite

degree bound n is determined by $n = \max td_i$ where td_i is the

<u>term degree</u> of the i<u>th</u> term in the complete expansion of P,

that is the sum of the degrees of the non-main variables

$y_1,\ldots,y_v$ in the i<u>th</u> term.

For $S = (y_1 - b_1, \ldots, y_v - b_v)$, we can extend the notion of modular congruence to multivariate polynomials with respect to a set of integers b. Thus $P \pmod S$ will simply be $P_b(x)$, i.e. dropping all terms in the Taylor series form of P which contain any element of S; $P \pmod{S^2} = P^{(0)} + P^{(1)}$; and, in general, $P \pmod{S^m} = \sum_{i=0}^{m-1} P^{(i)}(x, y_1, \ldots, y_v)$. Therefore, due to the finiteness of the series form of P, $P \pmod{S^{n+1}}$ is simply P written in the general Taylor series form with respect to b, so that it is equal to P independent of any chosen b. For $J = Z[y_1, \ldots, y_v]$ or $(Z/q)[y_1, \ldots, y_v]$, it is a direct extension of previous notation to denote the space of all polynomials modulo S^m by J/S^m. As an example of such a generalized Taylor series representation, we take a trivariate polynomial

$$P(x, y_1, y_2) = y_1^2 y_2^2 x + (-y_1^2 y_2 - y_1 y_2^2 + y_1 y_2 + y_1^2 + 2 y_1) x + y_1^3 y_2^3 - y_2^3 - y_1^2 y_2^2 - y_1 y_2 + 2 y_2, \text{ and}$$

$b = (b_1, b_2) = (1, 0)$. Then $P_b(x) = x^2 + 3x$,

and in the Taylor series form, which is typically obtained by a Hensel-type construction,

$$P(x,y_1,y_2) = P_b^{(1)}(x) + P^{(2)} + P^{(3)} + P^{(4)} \quad \text{where}$$

$$P^{(1)}(x,y_1,y_2) = (2x^2 - 3x)(y_1 - 1) + 0\,(y_2 - 0),$$

$$P^{(2)}(x,y_1,y_2) = (x^2 + 1)(y_1 - 1)^2 + (-x - 3)(y_1 - 1)\,y_2 + (-x)\,y_2^2,$$

$$P^{(3)}(x,y_1,y_2) = (-x - 1)(y_1 - 1)^2\,y_2 + (-x)(y_1 - 1)\,y_2^2,$$

$$P^{(4)}(x,y_1,y_2) = 1\,(y_1 - 1)^3\,y_2.$$

Clearly the term degree bound, n, is 4 for this P.

$S = (y_1 - 1,\ y_2)$ and, for instance,

$$P \pmod S = P^{(0)} + P^{(1)} = (x^2 + 3x) + (2x^2 - 3x)(y_1 - 1).$$

The above concepts and notations will be of crucial importance to our discussion of the various Hensel constructions in Chapter II. There, we will present, first, Hensel's Lemma itself, then the quadratic extension method due to Zassenhaus for the univariate polynomials. Section II-4 contains the detailed descriptions of the multivariate

Hensel construction - the Generalized Hensel Algorithm. Finally, in Section II-5, we discuss the uses of these Hensel constructions over the domain of integers which leads naturally to a discussion of the difficulty and solution of the non-trivial leading coefficient problem in this case. Some simple but comprehensive examples are carried out to help give a clearer understanding of these constructive methods which form the foundation of all the new algorithms presented later.

Chapter III contains the new algorithms for computing univariate or multivariate polynomial GCD's using the Hensel constructions. We will emphasize the multivariate algorithm, EZGCD, since it provides an efficient method for the large class of intermediate-sized problems. The general set-up of the new method of calculating GCD's and the fundamental theorems proving the validity of this method will be presented for the multivariate case only, since the univariate case obeys the same rules. In Section III-3, we present the UNIGCD Algorithm which uses the Zassenhaus' Quadratic Extension Algorithm to compute univariate GCD's. Then the main application of the Hensel construction in this thesis - EZGCD Algorithm, is finally discussed in full detail in Section III-4. The solution to the non-trivial leading coefficient problem for the computation of GCD's also appears in detail in this section.

The timing analysis of many algorithms in Chapters II and III are collected in Chapter IV. We analyze the computing costs of the Hensel-type algorithms, argue the case for our more intuitive timing formula for the multivariate Hensel construction. Then the computing cost of the EZGCD Algorithm is derived and it is shown that the dominating cost here is in the Hensel construction. These derived computing time formulas are verified by timing many cases of GCD computations. These empirical results strongly suggest the validity of the derived formulas since the actual computing times ordinarily correspond to the formula predictions to within 15% and frequently much closer.

The next three chapters contain three other useful applications of the Hensel constructions in algebraic computations. Again we concentrate on the multivariate cases , since the univariate cases are much less complex so that they can essentially be considered special cases of the multivariate methods.

Chapter V outlines the polynomial factorization algorithms using the Zassenhaus' Quadratic Extension algorithm in the univariate case and a new and more efficient multiple factor generalized Hensel construction for the multivariate case. Chapter VI shows how we can take advantage of the univariate images of the codivisors of one of several given polynomials and apply the Hensel

construction only once. Thus, polynomial contents and primitive parts or the GCD of more than two polynomials can be computed in a semi-parallel way. In Chapter VII, we point out some of the inefficiencies in the known methods of computing square-free decompositions of multivariate polynomials. Then we will generalize a method already introduced as the special case algorithm for EZGCD, and get still another application of the Hensel construction in a new square-free decomposition algorithm - EZSQFR.

Finally, in Chapter VIII we summarize this research, point out several more potential applications of the Hensel-type constructions, and indicate some open research problems of interest in this direction.

CHAPTER II. THE HENSEL ALGORITHMS

II - 1 Introduction

In this chapter, we will discuss the Hensel-type algorithms for polynomials over Z, or Z/q in particular those where computational advantages can be most easily demonstrated. Because of the importance of these Hensel constructions for the discussion of all the other material in the following chapters, some intuitive feeling as well as a conceptual grasp of these methods is imperative. To aid in achieving this understanding, we will intersperse numerous examples among the discussions. Also, for theoretical and pedagogical purposes, we compare the algorithms with a purely integral version. Analyses of these Hensel-type algorithms will be done in Chapter IV, where we will also present some empirical results to justify our arguments and the computing time formulas.

II - 2 Two Basic Supporting Algorithms

Lemma II-2.1: Let $F(x)$ and $G(x)$ be in $(Z/p)[x]$ where p is a prime in Z. If $\gcd(F,G) \equiv D(x) \pmod p$, then there exist unique (up to units) $A(x)$ and $B(x)$ in $(Z/p)[x]$ such that $A(x)\,F(x) + B(x)\,G(x) \equiv D(x) \pmod p$ where $\deg(A) < \deg(G) - \deg(D)$ and $\deg(B) < \deg(F) - \deg(D)$.

Proof: $(Z/p)[x]$ is an Euclidean ring, so that the

Extended Euclidean algorithm applies. A version of this algorithm for the domain of integers is presented by Knuth [KNU69] and can easily be generalized for $(Z/p)[x]$.

Assume there exist A' and B' in $(Z/p)[x]$ such that

$A'F + B'G \equiv 0 \pmod p$ and

$\deg(A') < \deg(G) - \deg(D)$, $\deg(B') < \deg(F) - \deg(D)$. Then $(A - A')(F/D) = (B' - B)(G/D)$. Since (F/D) and (G/D) are relatively prime, (F/D) divides $B - B'$. But $\deg(B - B') < \deg(F) - \deg(D)$, thus $B - B' \equiv 0$ or $B \equiv B'$. Similarly $A \equiv A'$. The A and B so found by the Extended Euclidean algorithm must then be unique in $(Z/p)[x]$. //

Example II-2.1a: As a simple example, let $F(x) = x$, $G(x) = x + 3$, and $p = 7$ then clearly $(-1)\,x + 1\,(x + 3) = 3$ or $2\,x + (-2)\,(x + 3) \equiv 1 \pmod 7$ so that $A = 2$, $B = -2$, and $D = 1$.

Corollary II-2.1: If p in the above lemma is not a prime, but all prime divisors of p are "lucky" (see [BRO71] or Section IV - 2), then the conclusions of the above lemma still hold.

Proof: Since p is lucky, the Extended Euclidean algorithm still applies (refer to Theorem 1 of [BRO71]). It is also clear that the proof for uniqueness of A and B holds. //

Since $(Z/p)[x]$ is a Euclidean ring, regular division

of polynomials with remainder can be carried out. Given polynomials $F_1 = F$ and $F_2 = G$ in $(Z/p)[x]$, the PRS, $F_1, F_2, \ldots, F_k$ can be found together with the quotients Q_i such that $F_{i-2} = Q_i F_{i-1} + F_i$, $i = 3,\ldots,k$. Then $D(x) = F_k(x)$ and $A(x)$, $B(x)$ are constructed from the $Q(x)$'s as specified in the Extended Euclidean algorithm. We present a tabulated version of this algorithm to make the algorithm easier to understand.

Algorithm II-2.1:

Input: F and G in $(Z/p)[x]$.

Output: A, B, and D in $(Z/p)[x]$ such that

$$A F + B G \equiv D \pmod p$$

		Q_3	Q_4	...	Q_i	...	Q_k
1	0	A_3	A_4	...	A_i	...	A_k
0	1	B_3	B_4	...	B_i	...	B_k
		F_3	F_4	...	F_i	...	F_k

$A_1 = 1$, $A_2 = 0$, $B_1 = 0$, $B_2 = 1$, and $A_i = A_{i-2} - Q_i A_{i-1}$, $B_i = B_{i-2} - Q_i B_{i-1}$, $i = 3,\ldots,k$. That is, as we compute Q_i and F_i by the division algorithm, we can fill in the values

of A_i and B_i by subtracting the product of the current quotient and the previous A or B from the one A or B before.

Finally $A = A_k$, $B = B_k$, and $D = F_k$ where

$A F + B G \equiv D \pmod p$.

Example II-2.1b: Let $F(x) = x^3 + 2$, $G(x) = x^2 + 2x - 2$, and $p = 5$. By division, we get

$F = Q_3 G + F_3 = (x - 2) G + (x - 2)$ and

$G = Q_4 F_3 + F_4 = (x - 1) F_3 + 1$. Thus the tabulated solutions are as follows:

		x-2	x-1
1	0	1	-x+1
0	1	-x+2	x^2+2x-2
		x-2	1

So $A = -x + 1$, $B = x^2 + 2x - 2$, and $D = 1$. It is easy to verify this result, and clearly $A F + B G \equiv D \pmod p$ with $\deg(A) = 1 < \deg(G) = 2$.

Remark: By the Corollary, p may be a composite integer and the algorithm will still be valid so long as lc(F) and lc(G) are units in (Z/p).

Lemma II-2.2: For non-zero F(x) and G(x) in (Z/p)[x], and any H(x) such that gcd(F,G) = D(x) which divides H(x)

(mod p), there exist unique A(x) and B(x) in (Z/p)[x] such that $A F + B G \equiv H \pmod p$ where $\deg(A) < \deg(G) - \deg(D)$.

Proof: By Lemma II-2.1, we can find A', B' in (Z/p)[x] such that $A' F + B' G \equiv D \pmod p$ where $\deg(A') < \deg(G) - \deg(D)$, $\deg(B') < \deg(F) - \deg(D)$. Let $C(x) = H(x)/D(x)$, then $(C A') F + (C B') G \equiv H \pmod p$ or $(C A') (F/D) + (C B') (G/D) \equiv C \pmod p$. Apply the division algorithm to C A' and G/D so that $C A' = Q (G/D) + R$ where $\deg(R) < \deg(G) - \deg(D)$. Let $A = R$ and $B = C B' + Q (F/D)$. Then $A (F/D) + B (G/D) \equiv C$ or $A F + B G \equiv H \pmod p$. Assume there also exist A"(x) and B"(x) in (Z/p)[x] satisfying $A" F + B" G \equiv H \pmod p$ and the degree constraints. Then

$(A - A") (F/D) \equiv (B" - B) (G/D) \pmod p$ .... (Eq. II-2.2).

F/D and G/D are relatively prime, so (G/D) must divide (A - A") (mod p). But $\deg(A - A") < \deg(G) - \deg(D)$, therefore, $A - A" \equiv 0 \pmod p$ or $A \equiv A" \pmod p$. Hence, from (Eq. II-2.2), $B \equiv B" \pmod p$, since G/D is non-zero. Therefore, the A and B so found must be unique in (Z/P)[x].

//

Example II-2.2a: Continuing from Example II-2.1a, assume $H(x) = x^2$, then following the notations in the proof of Lemma II-2.2 $C(x) = H/D = x^2$, since $D = 1$, $A' = 2$, and

$B' = -2$. $C\,A' = 2x^2 = Q\,G + R$ where $Q = 2x + 1$, $R = -3$.

Thus $A = -3$, $B = C\,B' + Q\,F = x$, and

$A\,F + B\,G = -3x + x(x + 3) = x^2$ where $\deg(A) < \deg(G)$.

It is quite obvious here that $A''\,F + B''\,G = H$ with $A'' = x$ and $B'' = 0$ also solves the equation. But then the condition $\deg(A'') < \deg(G)$ is not satisfied. In fact, $x = Q''\,G + R''$ with $Q'' = 1$ and $R'' = -3$. Using the same technique, we can get $R''\,F + (Q''\,F + B'')\,G = H$ where $Q''\,F + B'' = x$ and that is the same solution as the above A and B. This is a simple example showing that the equation $A\,F + B\,G = H$ has many solutions but division by G (or F) can be used to obtain a unique solution.

Corollary II-2.2: If H(x) in Lemma II-2.2 satisfies the degree constraint $\deg(H) < \deg(F) + \deg(G) - \deg(D)$, then B(x) satisfies a degree constraint similar to that for A(x), $\deg(B) < \deg(F) - \deg(D)$.

Proof: As in the proof of Lemma, $B = C\,B' + Q\,(F/D)$ satisfies $A\,F + B\,G \equiv H \pmod{p}$, thus $\deg(B) \leq \deg(H - A\,F) - \deg(G) < \deg(F) - \deg(D)$. //

Example II-2.2b: Continue from Example II-2.1b, we assume $H(x) = 2x^4 + x^3 + 2x^2 + x + 2$, then following the proof of Lemma II-2.2 and using the results of Example II-2.1b, $C(x) = H(x)$ since $D = 1$, $A' = -x + 1$, and

$B' = x^2 + 2x - 2$. $C\,A' = Q\,G + R$ where $Q = -2x^3 + 1$ and $R = 2x - 1$. So $A = R = 2x - 1$, $B = C\,B' + Q\,F = 2x - 2$, and $A\,F + B\,G \equiv H$ where $\deg(A) < \deg(G)$, and since $\deg(H) < \deg(F) + \deg(G)$, also $\deg(B) < \deg(H)$. This result will be used in Example II-3.1 in the next section.

Remark: In Lemma or Corollary II-2.2, if p is not a prime but $lc(G)$ is a unit in Z/p, then the conclusions still hold.

Example II-2.2c: In this example, p is not a prime, but computations similar to the above example can still be carried out. (The results of this example will later be used in Example II-3.2.) Assume we have

$A' = -x - 9$, $B' = x^2 - 3x + 3$, $F = x^2 + 10x - 8$,

$G = x^2 - 12x + 7$, and $p = 25$, then $A'\,F + B'\,G \equiv 1 \pmod{p}$. Clearly $lc(G) = 1$ is a unit in $(Z/25)$. Let

$H(x) = -x^3 - 11x^2 - 6x - 7$ in $(Z/25)[x]$ be given and we want to find $A(x)$, $B(x)$ in $(Z/25)[x]$ such that $A\,F + B\,G \equiv H \pmod{25}$. Since $C(x) = H/D = H$ and $lc(G) = 1$, the division algorithm can be carried out to compute $C\,A' = Q\,G + R$ where $Q = x^2 + 8x - 9$ and $R = 0$. Thus $A = 0$, $B = C\,B' + Q\,F = -x + 1$, and we get $A\,F + B\,G \equiv H$ with them. Therefore, when p is not a prime

but lc(G) is a unit in (Z/p), the process remaine the same so long as the division algorithm can be carried out.

The algorithm for achieving the goals set in Lemma II-2.2 is similar to Algorithm 2.7S (Solution of a polynomial equation) of Musser [MUS71], except ours can be used for general right-hand-sides other than 1.

Algorithm II-2.2:

Input: F, G, A', B', D, and H in (Z/p)[x] such that A' F + B' G = D and D divides H or C = H/D is in (Z/p)[x].

Output: Unique A and B in (Z/p)[x] such that A F + B G = H where deg(A) < deg(G) - deg(D).

(If deg(H) < deg(F) + deg(G) - deg(D), then also deg(B) < deg(F) - deg(D).)

(1) Set A <-- A' C. If D ≠ 1, set F <-- F/D, G <-- G/D.

(2) Apply mod p division algorithm to compute Q and R such that A = Q G + R, deg(R) < deg(G).

(3) Return A <-- R and B <-- C B' + Q F (mod p).

II - 3 Hensel's Lemma - The Univariate Case

We will now present the Hensel Lemma for extending a factorization modulo p to a factorization modulo p^k for any k > 1. The presentation is essentially based on that of Van der Waerden [VDW49]. This Lemma provides the foundation of most other algorithms we will discuss later. The

proof of the Lemma is constructive, but we will still formalize it into an algorithm. The reason for that is because it is very important to get a firm grasp of the underlying ideas of this Lemma. We will not use the construction of the proof of the Lemma for computations in the univariate case, rather we will use Zassenhaus' Quadratic Extension algorithm. However, the construction for extending multivariate factorizations is a direct generalization of this Lemma as we shall soon see.

Lemma II-3.1: (Hensel)

Let $F(x)$ be in $Z[x]$ and p be a prime in Z. Assume $F(x) \equiv G_1(x)\, H_1(x) \pmod p$, where $G_1(x)$ and $H_1(x)$ are relatively prime polynomials in $(Z/p)[x]$. Then, for any integer $k > 1$, there exist polynomials $G_k(x)$ and $H_k(x)$ in $(Z/q)[x]$ where $q = p^k$ such that $F(x) \equiv G_k(x)\, H_k(x) \pmod q$ and $G_k \equiv G_1 \pmod p$, $H_k \equiv H_1 \pmod p$.

Proof: Since G_1 and H_1 are relatively prime in $(Z/p)[x]$, we can find, by applying Algorithm II-2.1, $A(x)$, $B(x)$ in $(Z/p)[x]$ such that $A\,G_1 + B\,H_1 \equiv 1 \pmod p$ and $\deg(A) < \deg(H_1)$, $\deg(B) < \deg(G_1)$. From G_1 and H_1, we will

construct sequences of polynomials (G_j) and (H_j) by induction

such that $F \equiv G_j H_j \pmod{p^j}$ and $G_j \equiv G_1$, $H_j \equiv H_1 \pmod{p}$.

Assume for $j \geq 1$, we have $G_j(x)$, $H_j(x)$ in $(Z/p^j)[x]$ such that

$$F \equiv G_j H_j \pmod{p^j} \quad \ldots\ldots\ldots\ (*).$$

Let $C_j(x)$ in $(Z/p)[x]$ be such that

$$p^j C_j(x) \equiv F(x) - G_j(x) H_j(x) \pmod{p^{j+1}}$$

where all arithmetic operations are performed in (Z/p^{j+1})

with G_j and H_j considered to be polynomials in $(Z/p^{j+1})[x]$.

Note that, by (*), p^j divides into $F(x) - G_j(x) H_j(x)$

exactly. Now apply Algorithm II-2.2 on G_1, H_1, A, B and C_j

in $(Z/p)[x]$ and obtain $A_j(x)$, $B_j(x)$ in $(Z/p)[x]$ such that

$A_j(x) G_1(x) + B_j(x) H_1(x) \equiv C_j(x) \pmod{p}$ and

$\deg(A_j) < \deg(H_1)$. Now let

$G_{j+1} = G_j + p^j B_j$ and $H_{j+1} = H_j + p^j A_j$, then

$$G_{j+1}\,H_{j+1} \equiv G_j H_j + p^j (A_j G_1 + B_j H_1) \pmod{p^{j+1}}$$

$$\equiv G_j H_j + p^j C_j \equiv F(x) \pmod{p^{j+1}}$$

Because of this construction,

$$G_{j+1} = G_1 + p B_1 + \ldots + p^j B_j \text{ and}$$

$$H_{j+1} = H_1 + p A_1 + \ldots + p^j A_j, \text{ for all } j \geq 1$$

So clearly $G_j \equiv G_1$, $H_j \equiv H_1 \pmod{p}$ for all $j \geq 1$. //

Remark: If F(x) is a monic polynomial in Z[x] then $\deg(F - G_1 H_1) < \deg(G_1) + \deg(H_1)$. So C_2 and, in turn, C_j for all $j > 1$ satisfy the condition of Corollary II-2.2. Therefore, we have $\deg(B_j) < \deg(G_j)$ for all $j \geq 1$. This will prove to be a very important observation later, especially for solving the leading coefficient problem of the EZGCD algorithm. It should also be noted that the Hensel construction is a method for expressing the factors in a p-adic representation or a type of power series form. This is clearly seen from the last expressions for G_{j+1} and H_{j+1} in the above proof. An example here will make these points clearer, and also demonstrate the constructive nature of the

above proof.

Example II-3.1: Let $F(x)$ in $Z[x]$ be

$x^5 + 12x^4 - 22x^3 - 163x^2 + 309x - 119$ and $p = 5$. Then

in $(Z/p)[x]$, $F(x) \equiv G_1(x) H_1(x)$ where $G_1(x) = x^3 + 2$ and

$H_1(x) = x^2 + 2x - 2$. Let us carry out two iterations of

the Hensel construction, i.e. assume $k = 3$ and find $G_3(x)$,

$H_3(x)$ such that $F(x) = G_3(x) H_3(x) \pmod q$ where $q = p^3$. We

will cheat a little by revealing the answers,

$G_3(x) = x^3 - 15x + 17$ and $H_3(x) = x^2 + 12x - 7$. The

reason for doing this is to be able to see exactly how

these coefficients are discovered as the algorithm pro-

ceeds. The first thing to do is to compute $A(x)$ and $B(x)$ in

$(Z/p)[x]$ such that $A G_1 + B H_1 \equiv 1 \pmod p$. But this com-

putation was already done in Example II-2.1b and we have

$A(x) = -x + 1$ and $B(x) = x^2 + 2x - 2$. Next we have

$F - G_1 H_1 = 10x^4 - 20x^3 - 165x^2 + 305x - 115$ over Z with

G_1 and H_1 considered polynomials in $Z[x]$. Thus

$C_1(x) = (F - G_1 H_1)/p \pmod p = 2x^4 + x^3 + 2x^2 + x + 2$ and

$p\,C_1(x)$ is the "residue" of F and $G_1 H_1$ in the next bigger polynomial space $(Z/p^2)[x]$. The goal now is to correct or add to G_1 and H_1 what is missing so as to make this residue disappear and to have G_2 and H_2 such that

$F \equiv G_2 H_2 \pmod{p^2}$. To do so we need to find $A_1(x)$ and $B_1(x)$ in $(Z/p)[x]$ such that $A_1 G_1 + B_1 H_1 \equiv C_1 \pmod p$.

Conveniently, this computation has also been done in Example II-2.2b, with $A_1(x) = 2x - 1$ and $B_1(x) = 2x - 2$. Thus

$G_2 = G_1 + p B_1 = x^3 + 10x - 8$ and

$H_2 = H_1 + p A_1 = x^2 + 12x - 7$. Thus, as is easily verified,

$F \equiv G_2 H_2 \pmod{p^2}$. At this point, note that

G_2 is simply $G_2 \pmod{p^2}$ and H_2 is $H_3 \pmod{p^2}$. In fact,

$H_2 = H_3$ since $p^2 = 25$ is already bigger than twice the

coefficients of H_3. What this iteration has accomplished is then obvious. For the second iteration,

$$C_2(x) = (F - G_2 H_2)/p^2 \pmod p = -(x^3 + x^2 + x + 2).$$

Applying Algorithm II-2.2 on F, G, A, B, and D = 1, we get $A_2(x) = 0$ and $B_2(x) = -x + 1$ in $(Z/p)[x]$ such that

$A_2 F + B_2 G_2 \equiv C_2 \pmod p$. Then,

$G_3 = G_2 + p^2 B_2 = x^3 - 15x + 17$. As we already pointed out, $H_3 = H_2$ and surely $A_2 = 0$ so that H_2 need not be updated.

Therefore, we have constructed G_3 and H_3 such that

$F \equiv G_3 H_3$ in $(Z/q)[x]$, as we set out to do.

As we can see from the construction,

$G_3 = G_1 + p B_1 + p^2 B_2$ and $H_3 = H_1 + p A_1 + p^2 A_2$ which shows clearly $G_3 \equiv G_1$ and $H_3 \equiv H_1 \pmod p$. Actually,

$$G_3 = x^3 - 15x + 17 = (1 + 0\,5 + 0\,5^2)x^3 + (0 + 2\,5 + (-1)\,5^2)x + (2 + (-2)\,5 + 1\,5^2)$$

which is putting the coefficients of G_3 into p-adic representation with

$p = 5$. Then, the correspondence of this representation and the above formula for G_3 as constructed from G_1, B_1, and B_2 becomes clearer.

Algorithm II-3.1: (Hensel)

Input: $F(x)$ in $Z[x]$, p a prime in Z which does not divide $lc(F)$, $k > 0$ in Z, $G_1(x)$ and $H_1(x)$ in $(Z/p)[x]$ which are relatively prime, and $F \equiv G_1 H_1 \pmod p$.

Output: $G_k(x)$ and $H_k(x)$ in $(Z/q_k)[x]$ where $q_k = p^k$ such that $F \equiv G_k H_k \pmod q$ and $G_k \equiv G_1$, $H_k \equiv H_1 \pmod p$.

(1) Apply Algorithm II-2.1 to $G = G_1$ and $H = H_1$ and obtain A, B in $(Z/p)[x]$ such that $A G + B H \equiv 1 \pmod p$.

Set $j \leftarrow 1$, $q \leftarrow p$, and $q' \leftarrow q\,p$.

(2) If $j = k$ then output G and H.

Otherwise, set $C \leftarrow (F - G H)/q$ and $C \leftarrow C \pmod p$.

(3) Apply Algorithm II-2.2 on G, H, A, B, and C in $(Z/p)[x]$ to get A' and B' in $(Z/p)[x]$ such that $A' G + B' H \equiv C \pmod p$.

(4) Set $q \leftarrow q'$, $q' \leftarrow q\,p$, $j \leftarrow j + 1$, and go to (2).

We next discuss the Zassenhaus' quadratic extension algorithm [ZAS69] [MUS71] [W&R73] which is an improved version of the Hensel's method described above. This algorithm,

because of its added efficiency for small primes p or large required modulus q, is actually used in univariate cases of many later algorithms for extending factors from Z/p to Z/q, where $q = p^{2^k}$ for some $k > 0$, and consequently to Z (as we will see in Section II-5).

Lemma II-3.2: (Zassenhaus) Let $F(x)$ be in $Z[x]$ and p be a prime in Z which does not divide the leading coefficient of $F(x)$. Assume $F(x) \equiv G_0(x)\, H_0(x) \pmod p$ where G_0 and H_0 are relatively prime polynomials in $(Z/p)[x]$. Then, for any integer $k > 0$, there exist polynomials $G_k(x)$ and $H_k(x)$ in $(Z/q_k)[x]$, where $q_k = p^{2^k}$, such that

$$F(x) \equiv G_k(x)\, H_k(x) \pmod{q_k} \text{ and } G_k \equiv G_0,\ H_k \equiv H_0 \pmod p.$$

Proof: $lc(G_0)$ and $lc(H_0)$ must be units in (Z/p), since $lc(F)$ is one. By Lemma and Corollary II-2.1 and using Algorithm II-2.1, we can find A_0, B_0 in $(Z/p)[x]$ such that

$$A_0 G_0 + B_0 H_0 \equiv 1 \pmod p \text{ and}$$

$$\deg(A_0) < \deg(H_0),\ \deg(B_0) < \deg(G_0).$$

From G_0, H_0, A_0, and B_0, we will construct sequences of

polynomials (G_j), (H_j), (A_j), and (B_j) by induction such that

$F \equiv G_j H_j \pmod{q_j}$, $G_j \equiv G_0 \pmod{p}$, $H_j \equiv H_0 \pmod{p}$, and

$A_j G_j + B_j H_j \equiv 1 \pmod{q_j}$, $lc(H_j)$ is a unit

in (Z/q_j), and $\deg(A_j) < \deg(H_j)$, $\deg(B_j) < \deg(G_j)$. Assume

for $j \geq 0$ we have G_j, H_j, A_j, and B_j in $(Z/p^j)[x]$ such that

$F \equiv G_j H_j \pmod{q_j}$ and $A_j G_j + B_j H_j \equiv 1 \pmod{q_j}$.

Also assume $lc(H_j)$ is a unit in (Z/q_j) and

$\deg(A_j) < \deg(H_j)$, $\deg(B_j) < \deg(G_j)$.

Let $C_j(x)$ in $(Z/q_j)[x]$ be such that

$q_j C_j(x) = F(x) - G_j(x) H_j(x) \pmod{q_{j+1}}$ where all arithmetic operations are performed in $(Z/q_{j+1})[x]$. Now we apply

Algorithm II-2.2 on G_j, H_j, A_j, B_j, and C_j in $(Z/q_j)[x]$ and

obtain H'_j, G'_j in $(Z/q_j)[x]$ such that

$H'_j G_j + G'_j H_j \equiv C_j \pmod{q_j}$. Let $G_{j+1} = G_j + q_j G'_j$ and

$H_{j+1} = H_j + q_j H'_j$, then

$G_{j+1} H_{j+1} \equiv G_j H_j + q_j (H'_j G_j + G'_j H_j) \pmod{q_{j+1}}$

$$\equiv G_j H_j + q_j C_j \equiv F(x) \pmod{q_{j+1}}$$

Since $lc(H_j)$ is a unit in Z/q_j and $deg(H') < deg(H_j)$,

$lc(H_{j+1}) = lc(H_j)$ is also a unit in Z/q_{j+1} and

$deg(H_{j+1}) = deg(H_j) = deg(H_0)$. Now let $R_j(x)$ in $(Z/q_j)[x]$

be such that $A_j G_{j+1} + B_j H_{j+1} \equiv 1 + q_j R_j \pmod{q_{j+1}}$.

Apply Algorithm II-2.2 on G_j, H_j, A_j, B_j and R_j in $(Z/q_j)[x]$

and get A'_j, B'_j such that $A'_j G_j + B'_j H_j \equiv R_j$ in $(Z/q_j)[x]$

and $deg(A'_j) < deg(H_j)$. Let $A_{j+1} = A_j - q_j A'_j$ and

$B_{j+1} = B_j - q_j B'_j$, then

$$A_{j+1} G_{j+1} + B_{j+1} H_{j+1}$$

$$\equiv A_j G_{j+1} + B_j H_{j+1} - q_j (A'_j G_j + B'_j H_j).$$

$$\equiv 1 + q_j R_j - q_j (R_j) \equiv 1 \pmod{q_{j+1}}$$

and $deg(A_{j+1}) = deg(A_j) < deg(H_j) = deg(H_{j+1})$, hence,

$deg(B_{j+1}) < deg(G_{j+1})$, because $lc(H_{j+1})$ is a unit. By this

construction, $G_{j+1} = G_0 + q_0 G'_0 + \ldots + q_j G'_j$ and

$H_{j+1} = H_0 + q_0 H'_0 + \ldots + q_j H'_j$, so that clearly $G_j \equiv G_0$

and $H_j \equiv H_0 \pmod{p}$ for all $j \geq 0$. //

Remark: If $F(x)$ is a monic polynomial in $Z[x]$, then $\deg(F - G_0 H_0) < \deg(G_0) + \deg(H_0)$, so that C_1, and in turn, C_j for all $j > 1$ satisfy the condition of Corollary II-2.2.

Thus we must have $\deg(B') < \deg(G_j)$ in addition to $\deg(A') < \deg(H_j)$.

Example II-3.2: As an example for this quadratic construction we take the same F and p from Example II-3.1, so that we have $G_0 = x^3 + 2$ and $H_0 = x^2 + 2x - 2$ such that $F \equiv G_0 H_0 \pmod{q_0 = p = 5}$ as before. We want to carry the construction to $k = 2$ or $q_2 = 625$ which is already greater than 125 bounding twice (since we use positive and negative numbers centered at 0 as residue class representatives in any Z/q) the magnitude of coefficients in G_3 and H_3 of Example II-3.1. Thus our goal is still to construct G_3 and H_3 but using the quadratic construction with $A_0 = -x + 1$ and $B_0 = x^2 + 2x - 2$ such that $A_0 G_0 + B_0 H_0 \equiv 1 \pmod{q_0}$.

From these data we can compute G_1 and H_1 in $(Z/q_1)[x]$ as

we did in Example II-3.1, $G_1 = x^3 + 10x - 8$ and

$H_1 = x^2 + 12x - 7$. Next we find $R_0(x)$ in $(Z/q_0)[x]$ such

that $A_0 G_1 + B_0 H_1 \equiv 1 + q_0 R_0 \pmod{q_1 = 25}$,

$R_0 = ((15x^3 + 5x^2 - 20x + 6) - 1)/q_0 \pmod{q_0}$

$= -2x^3 + x^2 + x + 1$. Applying Algorithm II-2.2 on G_0,

H_0, A_0, B_0, and R_0 in $(Z/q_0)[x]$, we find $A' = 2$ and

$B' = x - 1$ such that $A' G_0 + B' H_0 \equiv R_0 \pmod{q_0}$. Thus

$A_1 = A_0 - q_0 A' = -x - 9$, $B_1 = B_0 - q_0 B' = x^2 - 3x + 3$,

and $A_1 G_1 + B_1 H_1 \equiv 1 \pmod{q_1 = p^2}$. Now we begin the next

iteration by computing $C_1(x)$ in $(Z/q_1)[x]$ such that

$q_1 C_1 = (F - G_1 H_1) \pmod{q_2 = 625}$.

$C_1 = (-25x^3 - 275x^2 + 475x - 175)/25 \pmod{25}$

$= -x^3 - 11x^2 - 6x - 7$. We need to apply Algorithm

II-2.2 on G_1, H_1, A_1, B_1, and C_1 in $(Z/q_1)[x]$ to obtain H'_1, G'_1 such that $H'_1 G_1 + G'_1 H_1 \equiv C_1 \pmod{q_1}$. Conveniently, Example II-2.2c has done just that with $H'_1 = 0$ and $G'_1 = -x + 1$. So $G_2 = G_1 + q_1 G'_1 = x^3 - 15x + 17$ and $H_2 = H_1 + q_1 H'_1 = H_1 = x^2 + 12x - 7$.

We have now constructed the desired polynomials using the quadratic method. The power of this method over the regular Hensel construction was not clearly demonstrated, but we can easily show it by changing the problem slightly. If we let $F = G H = (x^3 - 265x + 142)(x^2 + 187x - 107)$, then for the regular Hensel construction we see that the modulus would have to be at least 625 (since it is the least power of the modulus 5 which exceeds $2*265=530$) before we can construct from G_1 and H_1 any polynomial with coefficients of magnitude up to 265. That means we would carry out at least three steps of the Hensel construction, from modulo p to p^2 to p^3 to $p^4 = 625$. But for the quadratic method, only two iterations would be sufficient, since the modulus would already be 625. Actually, we intentionally chose G and H

so that $G \equiv G_1 = x^2 + 10x - 8$ and

$H \equiv H_1 = x^2 + 12x - 7$ in $(Z/25)[x]$. The only difference in the computation for this case is in the second iteration, where we need to compute a new C_1 and then, from G_1, H_1, A_1, B_1, and C_1, obtain $H' = 7x - 4$ and $G' = -9x + 6$ in $(Z/25)[x]$ such that $H' G_1 + G' H_1 \equiv C_1 \pmod{25}$. Thus we get finally $G_2 = G_1 + 25 G' = x^3 - 265x + 142$ and

$H_2 = H_1 + 25 H' = x^2 + 187x - 107$.

Of course, this example only shows a saving of one iteration for using the quadratic method instead of the regular Hensel construction. As the coefficients get large enough, the difference in the number of iterations will be bigger. For k iterations of the Hensel construction, the quadratic method needs only to go through (1 + the greatest integer of $\log_2(k)$) steps. However, let us point out an important fact - the quadratic growth in the modulus is obtained at the expense of having to solve

$A_j G_j + B_j H_j \equiv R_j \pmod{q_j}$ at each step, a cost which

is not negligible. Complicated cost trade-off is involved here. A more detailed study of this trade-off is currently underway. Preliminary results for the univariate cases show that the cost of solving these equations above is sufficiently dominant so as to make the quadratic method less efficient than the regular Hensel construction (with some modifications). There are also reasons to strongly suggest that the multivariate constructions will turn out to give the same indications. But that will again be left to a later study.

Algorithm II-3.2:

(Zassenhaus' Quadratic Extension Algorithm)

Input: $F(x)$ in $Z[x]$, p a prime in Z which does not divide $lc(F)$, $k > 0$ in Z, $G_0(x)$ and $H_0(x)$ in $(Z/p)[x]$ which are relatively prime, and $F \equiv G_0 H_0 \pmod{p}$.

Output: $G_k(x)$ and $H_k(x)$ in $(Z/q_k)[x]$ where $q_k = p^{2^k}$ such that $F \equiv G_k H_k \pmod{q_k}$ and $G_k \equiv G_0, H_k \equiv H_0 \pmod{p}$.

(1) Apply Algorithm II-2.1 to $G = G_0$ and $H = H_0$ and obtain A, B in $(Z/p)[x]$ such that $A G + B H \equiv 1 \pmod{p}$.

Set $j \leftarrow 0$, $q \leftarrow p$, and $q' \leftarrow q^2$.

(2) If $j = k$ then output G and H.

Otherwise, set $C \leftarrow (F - G H)/q$ and $C \leftarrow C \pmod q$.

(3) Apply Algorithm II-2.2 on G, H, A, B, and C in $(Z/q)[x]$ to get A' and B' in $(Z/q)[x]$ such that

$A' G + B' H \equiv C \pmod q$.

Set $G' \leftarrow G + q B'$ and $H' \leftarrow H + q A'$.

(4) Set $R \leftarrow (A' G' + B' H' - 1)/q \pmod{q'}$.

(5) Apply Algorithm II-2.2 on G, H, A, B, and R in $(Z/q)[x]$ to get A' and B' such that $A' G + B' H \equiv R$ in $(Z/q)[x]$.

Set $A \leftarrow A - q A'$, $B \leftarrow B - q B'$, $G \leftarrow G'$, and $H \leftarrow H'$.

(6) Set $q \leftarrow q'$, $q' \leftarrow q^2$, $j \leftarrow j + 1$, and go to (2).

II - 4 Generalized Hensel Algorithm

We now present the Generalized Hensel Algorithm (GHA) for extending univariate factorizations modulo q to multivariate factorizations expressed in the general Taylor series form modulo q where q is the r<u>th</u> power of a prime p. Musser [MUS71] presented a version of an "abstract" algorithm which is also intended to extend factorizations of multivariate polynomials. However, his basic underlying idea of recursively applying the Zassenhaus Quadratic Extension Algorithm has the drawback of requiring a general algorithm for solving multivariate polynomial equations in the form of $A F + B G = H$ with F, G and H given. We will

see later that there is a Hensel-type algorithm which will compute A and B. However, repeated use of a multivariate polynomial division with remainders algorithm in Musser's Algorithm Q keeps his version of the "Generalized Hensel Algorithm" from being computationally efficient. Therefore, the version of the Generalized Hensel Algorithm discussed here is based rather on a presentation by Wang and Rothschild [W&R73] in the context of factorization even though the presentation and the algorithm itself may appear to have been modified to a point of non-recognition.

Theorem II-4.1: (Generalized Hensel)

Let $F(x,y_1,y_2,\ldots,y_v)$ be a multivariate polynomial in $(Z[y_1,\ldots,y_v])[x]$. Let $b = \{b_1,\ldots,b_v\}$ be a given set of integral values and $S = \{y_1-b_1,\ldots,y_v-b_v\}$ such that the leading coefficient of F evaluated at $y_1 = b_1,\ldots,y_v = b_v$ satisfies $(lc(F))_b \not\equiv 0 \pmod q$ where q is a given rth power of a lucky prime p for F_b. Assume there exist relatively prime $G_1(x)$ and $H_1(x)$ in $(Z/q)[x] = (J/S)[x]$, where $J = (Z/q)[y_1,\ldots,y_v]$, such that $F \equiv G_1 H_1 \pmod q$. Then for any $k > 1$, there exist G_k and H_k in $(J/S^k)[x]$ such that

$F \equiv G_k H_k \pmod{q, S^k}$ and $G_k \equiv G_1, H_k \equiv H_1 \pmod{q, S}$.

Proof: Since $(lc(F)) \not\equiv 0 \pmod{q}$, it must be a unit in Z/q, hence $lc(G_1)$ and $lc(H_1)$ must be units also. By Lemma and Corollary II-2.1, we can use Algorithm II-2.1 to find $A_1(x)$ and $B_1(x)$ in $(J/S)[x]$ such that $A_1 G_1 + B_1 H_1 \equiv 1$ and $\deg(A_1) < \deg(H_1)$, $\deg(B_1) < \deg(G_1)$. From G_1, H_1, A_1, and B_1 in $(J/S)[x]$ we will construct sequences of polynomials $\{G_m\}$ and $\{H_m\}$ by induction such that $F \equiv G_m H_m$ in $(J/S^m)[x]$ and $G_m \equiv G_1, H_m \equiv H_1 \pmod{q, S}$. Assume for $m \geq 1$, we have $G_m(x, y_1, \ldots, y_v)$ and $H_m(x, y_1, \ldots, y_v)$ in $(J/S^m)[x]$ such that $F \equiv G_m H_m$. Let

$$R_m(x, y_1, \ldots, y_v) = F - G_m H_m \pmod{q, S^{m+1}}$$

$$= -\sum_{i_1=1}^{v} \cdots \sum_{i_m=1}^{v} C_{i_1, i_2, \ldots, i_m} (x) \prod_{j=1}^{m} (y_{i_j} - b_{i_j}).$$

$$= \sum_{Im} C_{Im}(x)\, M_m(y_1,\ldots,y_v)$$

For each C_{Im} in $(Z/q)[x]$ a typical coefficient polynomial of R_m, we can apply Algorithm II-2.2 on G_1, H_1, A_1, B_1, and C_{Im} in $(Z/q)[x]$ and obtain A_{Im}, B_{Im} such that

$A_{Im} G_1 + B_{Im} H_1 \equiv C_{Im}$ and $\deg(A_{Im}) < \deg(H_1)$ in $(Z/q)[x]$.

Now let $G^{(m)} = \sum_{Im} B_{Im}(x)\, M_m(y_1,\ldots,y_v)$

$$H^{(m)} = \sum_{Im} A_{Im}(x)\, M_m(y_1,\ldots,y_v)$$

$G_{m+1} = G_m + G^{(m)}$, and $H_{m+1} = H_m + H^{(m)}$.

Then $G_{m+1} H_{m+1} \equiv G_m H_m + H^{(m)} G_1 + G^{(m)} H_1$

$\equiv G_m H_m + R_m \equiv F$ in J/S^{m+1}. By this construction,

$G_{m+1} = G_1 + G^{(1)} + \ldots + G^{(m)}$ and

$H_{m+1} = H_1 + H^{(1)} + \ldots + H^{(m)}$, so that $G_m \equiv G_1$ and

$H_m \equiv H_1$ in (J/S) for all $m \geq 1$. //

Remark: It is important to note that, as in the univariate Hensel construction, if $F(x,y_1,\ldots,y_v)$ is a monic polynomial in $Z[x,y_1,\ldots,y_v]$, then $\deg(F - G_1 H_1) < \deg(G_1) + \deg(H_1)$. Thus each $C_{Im}(x)$ of R_j satisfies $\deg(C_{Im}) < \deg(G_1) + \deg(H_1)$, for all $m \geq 1$. Therefore, in addition to $\deg(H_1^{(m)}) < \deg(H_1)$ because of $\deg(A_{Im}) < \deg(H_1)$, we also have $\deg(G_1^{(m)}) < \deg(G_1)$ for all $m \geq 1$ because of $\deg(B_{Im}) < \deg(G_1)$.

Example II-4.1a: Let us first do a simple, monic example using the generalized Hensel construction for a multivariate polynomial. Let

$$F = x^2 + (-z^2 - y\,z + y + z + 2)\,x + y\,z^3 - z^3 + y^2\,z - y\,z + 2\,z.$$

Since F is monic, there is no problem in finding a valid evaluation, so let $b = (b_1, b_2) = (0, 0)$ and $S = (y-0, z-0) = (y, z)$. This will make the problem of picking out coefficients of monomials very simple. In fact, this device of using zeros as evaluation points will be used whenever possible. Assume we somehow determined a choice of

$q = p = 7$, then we have $G_1(x) = x$ and $H_1(x) = x + 2$ in $(Z/q)[x]$ such that $F \equiv G_1 H_1 \pmod q$. In this case we will let $k = 3$ and carry out two iterations of the construction. Again, we reveal in advance that F is actually the product of G and H with $G = x - y z + z$ and $H = x + y - z^2 + 2$; and we will see how these polynomials get recovered from G_1 and H_1 as we proceed. First, we need to find $A_1(x)$ and $B_1(x)$ in $(J/S)[x] = (Z/q)[x]$ such that $A_1 G_1 + B_1 H_1 \equiv 1 \pmod q$. By applying Algorithm II-2.1, we get $A_1(x) = 3$ and $B_1(x) = -3$. So we compute the current "residue" $R_1 = F - G_1 H_1 \pmod{q, S^2}$. We find that

$$F - G_1 H_1 = (-z^2 - y z + y + z) x + y z^3 - z^3 - y^2 z - y z + 2 z.$$

Modulo S^2 means dropping all terms with term degree in the non-main variables y and z higher than 1. Thus $R_1 = (y + z) x + 2 z$ in $(J/S^2)[x]$. Representing this in terms of monomials of the non-main variables, we get $R_1 = (x) y + (x + 2) z$, so we have $C_1(x) = x$ and

$C_2(x) = x + 2$ such that $R_1 = C_1(x)\, y + C_2(x)\, z$. By applying Algorithm II-2.2 (or it can be trivially shown), $A_1 = 1$, $B_1 = 0$ such that $A_1 G_1 + B_1 H_1 = C_1$ and $A_2 = 0$, $B_2 = 1$ such that $A_2 G_1 + B_2 H_1 = C_2$. Thus,

$G^{(1)} = B_1(x)\, y + B_2(x)\, z = z$ and

$H^{(1)} = A_1(x)\, y + A_2(x)\, z = y$, so that $G_2 = G_1 + G^{(1)} = x + z$

and $H_2 = H_1 + H^{(1)} = x + 2 + y$. Comparing this with G and H, we note that G_2 and H_2 already have more correct terms than G_1 and H_1. Although G_2 and H_2 still differ from G and H by some terms of higher degree, they now contain all terms of G and H which are linear in the non-main variable, i.e.

G_2 and H_2 approximates G and H correctly modulo S^2.

Continuing now to the second iteration, we find

$$R_2 = F - G_2 H_2 \pmod{q, S^3}$$

$$= (-y z - z^2)\, x + y z^2 - y^2 z - 2 y z^2 - z^3 \pmod{S^3}$$

$= (-y\,z - z^2)\,x - 2\,y\,z = (-x - 2)\,y\,z + (-x)\,z^2$. So, we have $C_{12}(x) = -(x + 2)$ and $C_{22}(x) = -x$ such that

$R_2 = C_{12}(x)\,y\,z + C_{22}(x)\,z^2$. Again it is trivial, even without using Algorithm II-2.2, to see that $A_{12} = 0$, $B_{12} = -1$ such that $A_{12}\,G_1 + B_{12}\,H_1 = C_{12}$ and $A_{22} = -1$, $B_{22} = 0$ such that $A_{22}\,G_1 + B_{22}\,H_1 = C_{22}$. Thus

$G_2^{(2)} = B_{12}\,y\,z + B_{22}\,z^2 = -y\,z$ and

$H_2^{(2)} = A_{12}\,y\,z + A_{22}\,z^2 = -z^2$ so that

$G_3 = G_2 + G_2^{(2)} = x + z - y\,z$ and

$H_3 = H_2 + H_2^{(2)} = x + y + 2 - z^2$. Note that already $G_3 = G$ and $H_3 = H$, since all terms quadratic in the non-main variables have been included, and G, H contain only up to quadratic terms any way so that equality modulo S^3 is the same as being equal over Z for all terms. Indeed, we find $F - G_3\,H_3 = 0$ over the integers.

Before we specify the details of the Generalized Hensel algorithm, we will discuss several computational improvements which will be important in the actual computational processes later. The first such improvement is due to a suggestion by Moses.

Lemma II-4.2: Let $G(x)$ and $H(x)$ be relatively prime polynomials in $(Z/q)[x]$ and let $A_i(x)$ and $B_i(x)$ be found such that $A_i G + B_i H \equiv x^i$ in $(Z/\ )[x]$ for all i where $0 \leq i \leq \deg(G) + \deg(H)$. Then for any $C(x) = \sum_{i=0}^{k} c_i x^i$ such that $\deg(C) = k \leq \deg(G) + \deg(H)$, polynomials $A(x)$ and $B(x)$ in $(Z/q)[x]$ can be constructed from the A_i and B_i's such that $A G + B H \equiv C \pmod q$.

Proof: Let $A = \sum_{i=0}^{k} c_i A_i$ and $B = \sum_{i=0}^{k} c_i B_i$ ... (*),

then $A G + B H = \sum_{i=0}^{k} c_i (A_i G + B_i H) \equiv C \pmod q$. //

Remark: If $A_i(x)$ and $B_i(x)$ are found using Algorithm II-2.2, then by Corollary II-2.2 $\deg(A_i) <$ deg(H) and

$\deg(B_i) < \deg(G)$ for all $0 \le i < \deg(G) + \deg(H)$, and for $i = \deg(G) + \deg(H)$, $\deg(A_1) < \deg(H)$, and $\deg(B_1) \le \deg(G)$.

Thus, since c_i's are independent of x, the computation of A and B involves simply the additions and multiplications in (*) and the degree constraints for A and B are automatically satisfied: $\deg(A) < \deg(H)$, $\deg(B) < \deg(G)$ for all C such that $\deg(C) < \deg(H) + \deg(G)$ and otherwise $\deg(B) \le \deg(G)$.

Also, the c_i's, instead of being elements of (Z/q), can be polynomials which are independent of x (or c_i in $(Z/q)[y_1,\ldots,y_v]$), and the computation of A and B is still simply a "scalar linear combination" of A_i's and B_i's.

Coming back to the generalized Hensel's construction, we can write any $R_m(x,y_1,\ldots,y_v)$ as polynomials in $(J/S_{m+1})[x]$, i.e. in the form of $\sum_{i=0}^{k} C_i(y_1,\ldots,y_v)\, x^i$ where C_i's are in $(Z/q)[y_1,\ldots,y_v]$ and k is the maximum degree of x in R_m (clearly $k \le \deg(F) = \deg(G_1) + \deg(H_1)$). Thus

$$G^{(m)} = \sum_{i=0}^{k} C_i(y_1,\ldots,y_v)\, B_i(x) \text{ and}$$

$$H^{(m)} = \sum_{i=0}^{k} C_i(y_1,\ldots,y_v)\, A_i(x) \text{ so that}$$

$$H^{(m)} G_1^{(m)} + G^{(m)} H_1^{(m)} \equiv \sum_{i=0}^{k} C_i(y_1,\ldots,y_v)\,(A_i G_1 + B_i H_1)$$

$$\equiv R_m \text{ in } (J/S^{m+1})[x].$$

So we see that, in order to compute $G^{(m)}$ and $H^{(m)}$ it is no longer necessary to solve all the polynomial equations $A_{Im} G_i + B_{Im} H_1 \equiv C_{Im}$, but simply use the solutions of $A_i G_1 + B_i H_1 \equiv x^i$ to form scalar linear combinations as needed.

Next, we observe that it is not necessary to compute R_{m+1} by the formula $F - G_{m+1} H_{m+1} \pmod{q, S^{m+2}}$, but we can get R_{m+1} by the following scheme using quantities already computed previously:

As in the proof of the Theorem (II-4.1), we write

$$G_{m+1} = G_m + G_m^{(m)} = G_1 + G_1^{(1)} + \ldots + G_m^{(m)}$$

$$H_{m+1} = H_m + H_m^{(m)} = H_1 + H_1^{(1)} + \ldots + H_m^{(m)}$$

Define $D_m = F - G_m H_m$ where the arithmetical operations are

done over Z with G_m H_m considered as polynomials in

$Z[x, y_1, \ldots, y_v]$. Then R_m is simply $D_m \pmod{q, S^{m+1}}$ and

$D_m \equiv 0 \pmod{S^m}$, i.e. D_m contains non-zero terms only in

$(J/S^k)[x]$ for $k > m$. Thus $D_1 = F - G_1 H_1$ and

$R_1 = D_1 \pmod{q, S^2}$. In general,

$$D_{m+1} = F - G_{m+1} H_{m+1} = D_m - G_m H_m^{(m)} - G_m^{(m)} H_m - G_m^{(m)} H_m^{(m)}$$

and $R_{m+1} = D_{m+1} \pmod{q, S^{m+2}}$ for all $m \geq 1$. (Since

$D_{m+1} \equiv 0 \pmod{S^{m+1}}$, R_{m+1} is obtained from D_{m+1} by simply

setting all $Mk(y_1, \ldots, y_v)$ to 0 in the generalized Taylor

series form of D_{m+1}, for all $k > m+1$.)

Finally, we note that the generalized Taylor series form is not always a very easy representation to obtain. To rewrite a multivariate polynomial $P(x,y_1,\ldots,y_v)$ in that form it will be necessary to divide P succesively by each and every $(y_i - b_i)$ until all combinations of n such $(y_i - b_i)$'s are used where n is the maximum term degree of the non-main variables. This is a combinatorially complex process. It is, therefore better to use the following more efficient technique: (However, it is important to note the this is not necessarily the optimal technique, as will be seen later when we discuss the so called "Non-zero Substitution" problem of this method.)

Let $y'_i = y_i - b_i$ for $i = 1,\ldots,v$. Then $y_i = y'_i + b_i$ so that if we completely expand $P(x,y'_1+b_1,\ldots,y'_v+b_v)$ and express that in the Taylor series form with b'_1 (the integral evaluation values for y'_i) $= 0$ for all i, we get

$$P(x,y_1,\ldots,y_v) = \sum_{m=0}^{n} C_{1m}(x)\, Mm(y'_1,\ldots,y'_v) \text{ where}$$

$$Mm(y'_1,\ldots,y'_v) = \prod_{j=1}^{m} y'_{ij} = \prod_{j=1}^{m} (y_{ij} - b_{ij}) = Mm(y_1,\ldots,y_v).$$

(Hence the name "monomial" used for Mm in Sec. I - 2.)

Note that with this representation of P, $S = (y'_1,\ldots,y'_v)$ and

$P \pmod{S^{m+1}}$ is a simple operation of dropping (or setting to zero) all monomials whose term degree exceed m with no re-representation or divisions by $(y_1 - b_i)$ necessary. Also, it is an equally simple matter to transform $P(x,y'_1,\ldots,y'_v)$ back to $P(x,y_1,\ldots,y_v)$ -- by substituting $y_i - b_i$ for y'_i.

Example II-4.1b: With all the computational mechanisms described above, we can now do a more complex example before we give the detailed specifications of the Generalized Hensel Algorithm. This example is the same as Example II-4.1a except the leading terms now have non-trivial coefficients. Let $G = y\,x + z - y\,z$ and $H = y\,x + y - z^2 + 2$, so that the given polynomial

$$F = G\,H = y^2\,x^2$$
$$+ (-y\,z^2 - y^2\,z + y\,z^2 + y^2 + 2\,y)\,x + y\,z^3 - z^3 - y^2\,z^2 - y\,z + 2\,z.$$

In order for the evaluation to be valid for this polynomial, we can only use non-zero values for y since $lc(F) = y^2$. Let $b = (b_1, b_2) = (1, 0)$ and $S = (y-1, z)$. Let $q = p = 7$, then

we have $G_1(x) = x$ and $H_1(x) = x + 3$ in $(Z/q)[x]$ such that

$F = G_1 H_1 \pmod q$. We let $k = 4$ and carry out three itera-

tions of the construction to compute G_3 and H_3 such that

$F = G_3 H_3 \pmod{q, S^3}$. First, we apply Algorithm II-2.1 on

G_1, H_1 in $(Z/q)[x]$ and find $A_{00}(x)$ and $B_{00}(x)$ such that

$A_{00} G_1 + B_{00} H_1 \equiv 1 \pmod q$. But Example II-2.1a has

already done so with $A_{10} = 2$ and $B_{10} = -2$. Then, according

to the first computational suggestion, we compute $A_{01}(x)$,

$B_{01}(x)$ such that $A_{01} G_1 + B_{01} H_1 \equiv x \pmod q$ and $A_{02}(x)$,

$B_{02}(x)$ such that $A_{02} G_1 + B_{02} H_1 \equiv X^2 \pmod q$. Since

$G_1 = x$, it is clear that $A_{01} = 1$ and $B_{01} = 0$, and conve-

niently Example II-2.2a readily gives us $A_{02} = -3$ and

$B_{02} = x$. Next, we perform the suggested substitution in

order to facilitate the necessary operations of putting

polynomials modulo S^m later. Let $y' = y - 1$ so that

$y = y' + 1$. Substituting this into F, we get

$$F' = (y'^2 + 2y' + 1)x^2$$

$$+ (-y'^2 z^2 - z^2 - y'^2 z - y' z + y'^2 - 3y' + 3)x$$

$$+ y' z^3 - y'^2 z - 3y' z.$$

With this substitution, $S = (y-1, z)$ becomes $S' = (y', z)$ which is a convenient form for picking out coefficients of monomials or performing modulo S'^m operations.

Having carried out these preliminary computations, we can proceed to go through the steps of the first iteration of the Generalized Hensel construction. According to another computational suggestion above, we compute $D_1 = F' - G_1 H_1$ over Z and get

$$D_1 = (y'^2 + 2y')x^2$$

$$+ (-y'^2 z^2 - z^2 - y'^2 z - y' z + y'^2 - 3y')x + y' z^3$$

$$- y'^2 z - 3y' z.$$

$$R_1 = D_1 \pmod{q, S'^2} = 2y' x^2 - 3y' x = (2x^2 - 3x)y'$$

$= C_1(x)\, y'$. From the preparations already done, we have $A_1(x) = 2A_{02}(x) + (-3)A_{01}(x) = -2$ and

$B_1(x) = 2B_{02}(x) + (-3)B_{01}(x) = 2x$ such that

$A_1 H_1 + B_1 G_1 \equiv C_1 \pmod q$. So $G_1^{(1)}$ is simply

$B_1 y' = 2 x y'$ and $H_1^{(1)} = A_1 y' = -2 y'$. Then

$G_2 = G_1 + G_1^{(1)} = x + 2 x y'$ and $H_2 = H_1 + H_1^{(1)} = x + 3 - 2 y'$

such that $G_2 H_2 \equiv F' \pmod{q, S'^2}$.

Next, we compute

$$D_2 = D_1 - G_1 H_1^{(1)} - G_1^{(1)} H_1 - G_1^{(1)} H_1^{(1)}$$

$$= y'^2 x^2 + (-y'^2 z - 2 y'^2 - y' z^2 - y' z - z^2) x$$

$$- y'^2 z + y' z^3 - 3 y' z. \text{ Then}$$

$$R_2 = D_2 \pmod{q, S'^3}$$

$$= y'^2 X^2 + (-2 y'^2 - y' z - z^2) x - 3 y' z$$

$$= (x^2 - 2x) y'^2 + (-x - 3) y' z + (-x) z^2$$

$$= C_{11}(x) y'^2 + C_{12}(x) y' z + C_{22}(x) z^2. \text{ By taking}$$

scalar linear combinations according to the coefficient polynomials C_{11}, C_{12}, and C_{13}, we easily get

$A_{11}(x) = 1\,A_{02} - 2\,A_{01} = 2$, $B_{11}(x) = 1\,B_{02} - 2\,B_{01} = x$, such that $A_{11} G_1 + B_{11} H_1 \equiv C_{12}$; $A_{12}(x) = 0$, $B_{12}(x) = -1$, such that $A_{12} G_1 + B_{12} H_1 \equiv C_{22}$; and $A_{22}(x) = -1$, $B_{22}(x) = 0$ such that $A_{22} G_1 + B_{22} H_1 \equiv C_{22} \pmod q$. Then

$$G_2^{(2)} = B_{11} y'^2 + B_{12} y' z + B_{22} z^2 = x y'^2 - y' z \text{ and}$$

$$H_2^{(2)} = A_{11} y'^2 + A_{12} y' z + A_{22} z^2 = 2 y'^2 - z^2 \text{, so that}$$

$$G_3 = G_2 + G_2^{(2)} = x + 2 x y' + x y'^2 - y' z,$$

$$H_3 = H_2 + H_2^{(2)} = x + 3 - 2 y' + 2 y'^2 - z^2 \text{, and}$$

$$G_3 H_3 \equiv F' \pmod{q, S'^3}.$$

Finally, we compute

$$D_3 = D_2 - G_2 H_2^{(2)} - G_2^{(2)} H_2 - G_2^{(2)} H_2^{(2)}$$

$$= (-2 y'^4 - 2 y'^3 + y'^2 z^2 - y'^2 z + y' z^2)\, x + 2 y'^3 z - 3 y'^2 z.$$

Then $R_3 = D_3 \pmod{q, S'^4} = (-2 y'^4 - y'^3 z + y'^2 z^2)\, x - 3 y'^3 z + 2 y'^2 z^2$

$$= (-2 x)\, y'^3 + (-x - 3)\, y'^2 z + (x)\, y' z^2$$

$$= C_{111}(x)\,y'^{3} + C_{112}(x)\,y'^{2}z + C_{122}(x)\,y'z^{2}.$$

Again, we can compute, by scalar linear combinations,

$A_{111} = -2$, $B_{111} = 0$, $A_{112} = 0$, $B_{112} = -1$, $A_{122} = 1$, and

$B_{122} = 0$. Thus

$$G^{(3)} = B_{111}y'^{3} + B_{112}y'^{2}z + B_{122}y'z^{2} = -y'^{2}z,$$

$$H^{(3)} = A_{111}y'^{3} + A_{112}y'^{2}z + A_{122}y'^{2}Z = -2y'^{3} + y'^{2}z,$$

$$\text{so that } G_4 = G_3 + G^{(3)} = x + 2xz' + xy'^{2} - y'^{2}z - y'^{2}z$$

$$= (y'^{2} + 2y' + 1)x - (y' + 1)y'z,$$

$$H_4 = H_3 + H^{(3)} = x + 3 - 2y' + 2y'^{2} - z - 2y'^{3} + y'^{2}z,$$

and $G_4 H_4 \equiv F' \pmod{q, S'^{4}}$.

At this point, one might object to the fact that neither G_4 nor H_4 has any resemblance to G' or H' (G or H with $y' + 1$ substituted for y) at all. This is a complication caused by the non-trivial leading coefficient of the given polynomial. All we can point out here is that if G_4 were transformed back into a polynomial in y by substituting

$y - 1$ for y' (we get $y^2 x - (y - 1) z$), then it becomes a simple multiple (by y) of G. We will give a more complete discussion of this in the next section, where this example will be continued and G, H will actually be recovered. Here, it is only important to observe that we can actually find the sequences of polynomials $\{G_2, G_3, G_4\}$ and $\{H_2, H_3, H_4\}$ such that $G_m H_m \equiv F' \pmod{q, S'^m}$, $m = 2, 3$, and 4.

Algorithm II-4.1 (Generalized Hensel)

Input: $F(x, y_1, \ldots, y_v)$ in $(Z[y_1, \ldots, y_v])[x]$; $b = (b_1, \ldots, b_v)$ a given set of integer values such that $(\mathrm{lc}(F))_b \not\equiv 0 \pmod{q}$ where q is a given lucky prime power for F_b; $S = (y_1 - b_1, \ldots, y_v - b_v)$; $G_1(x)$ and $H_1(x)$ are relatively prime polynomials in $(J/S)[x]$ where $J = (Z/q)[y_1, \ldots, y_v]$ such that $F \equiv G_1 H_1 \pmod{q, S}$; and n = the degree bound for F in the non-main variable.

Output: G_{n+1} and H_{n+1} in $(J/S^{n+1})[x]$ such that

$$F \equiv G_{n+1} H_{n+1} \pmod{q, S^{n+1}} \text{ and } G_{n+1} \equiv G_1,$$

$H_{n+1} \equiv H_1 \pmod{q, S}$.

(1) Set $F \leftarrow F(x, y_1 + b_1, \ldots, y_v + b_v)$, $S \leftarrow (y_1, \ldots, y_v)$, and

$m \leftarrow 1$.

(2) Apply Algorithm II-2.1 on $G = G_1$ and $H = H_1$ in $(Z/q)[x]$

to get A_0, B_0 such that $A_0 G + B_0 H \equiv 1$ in $(Z/q)[x]$ and

$\deg(A_0) < \deg(H)$, $\deg(B_0) < \deg(G)$.

(3) Apply Algorithm II-2.2 on G, H, A_0, B_0, and x^i for

$0 < i \leq \deg(F) = \deg(G) + \deg(H)$ and get A_i, B_i in $(Z/q)[x]$

such that $A_i G + B_i H \equiv x^i \pmod{q}$, $\deg(A_i) < \deg(H)$ for all

$i \leq \deg(F)$, $\deg(B_i) < \deg(G)$ for all $i < \deg(F)$, and

$\deg(B_i) \leq \deg(G)$ if $i = \deg(F)$.

(4) Set $D \leftarrow F - G H$, $G' \leftarrow 0$, $H' \leftarrow 0$.

(5) If $m = n + 1$, set $G \leftarrow G''$, $H \leftarrow H''$ and go to (7),

otherwise, set $m \leftarrow m + 1$. Set

$D \leftarrow D - G H' - G' H - G' H'$, $G \leftarrow G''$, and $H \leftarrow H''$, where

all arithmetic operations are carried out over Z/q and F,

G, H considered in $(Z/q)[x, y_1, \ldots, y_v]$. If $D = 0$, then go to

(7), otherwise, set R <-- D (mod S^m), i.e. drop all monomials in D with term degree $\geq$ m and express R as

$$\sum_{i=0}^{k} C_i(y_1,\ldots,y_v)\, x^i, \quad k \leq \deg(F)$$

by collecting all coefficients of like powers of x.

(6) Set $G' \leftarrow\!- \sum_{i=0}^{k} C_i B_i$, $H' \leftarrow\!- \sum_{i=0}^{k} C_i A_i$,

G" <-- G + G', H" <-- H + H', and go to (5).

(7) Output $G(x, y_1 - b_1, \ldots, y_v - b_v)$ and $H(x, y_1 - b_1, \ldots, y_v - b_v)$.

II - 5 Hensel Construction over the Integers.

Due to its theoretical importance and for pedagogical interests, we now present a theorem which contains all the underlying ideas of the generalized Hensel's construction for extending factors over the integers. The theorem shows that this method of construction can be carried out directly in Z. The Generalized Hensel Algorithm is simply a special case of this theorem where, for computational efficiencies, the underlying numerical coefficient space will be Z/q such that q is a prime power, p**r, greater than any of the integer coefficients in the original polynomial, its factors, and their evaluations at the given v integral

points.

Lemma II-5.1: Let J[x] denote a general polynomial domain and S denote a general modulus. Assume G H $\equiv$ G' H' in $(J/S^k)[x]$ where G $\equiv$ G' $\equiv$ G", H $\equiv$ H' $\equiv$ H" for some relatively prime polynomials G"(x), H"(x) in (J/S)[x], deg(G) = deg(G') = deg(G"), deg(H) = deg(H') = deg(H"). Then G and G', H and H' are associates in $(J/S^k)[x]$ (i.e. G $\equiv$ u G' and H $\equiv$ v H' where u and v are units and u v $\equiv$ 1 in $(J/S^k)[x]$ or G/lc(G) and H/lc(H), are identically equal to G'/lc(G') and H'/lc(H') respectively where divisions are carried out in $(J/S^k)[x]$ and the leading coefficients are inherently assumed to be units in (J/S^k)).

Proof: Let D = gcd(G,G') so that G = D C, G' = D C', and gcd(C,C') = 1. Thus, D C H = G H $\equiv$ G' H' = D C' H' ==> C divides H' ==> C (mod S) divides H' (mod S) = H". Clearly also C (mod S) divides G (mod S) = G". C (mod S), thus, divides gcd(G",H") = 1 ==> deg(C (mod S)) = 0. But deg(C) = deg(C (mod S)) = 0 in $(J/S^k)[x]$ and C must be a unit in $(J/S^k)[x]$, since C (mod S) divides 1. Similarly C' is also a unit. Therefore, G and G' are associates and similarly with H and H'. Since lc(G) and lc(G') are units,

we can compute the monic polynomials G/lc(G) and G'/lc(G')

in $(J/S^k)[x]$. These polynomials when considered in J[x]

must be equal. Similarly H/lc(H) = H'/lc(H'). //

Theorem II-5.2: Let $F(x,y_1,y_2,\ldots,y_v)$ be a monic

multivariate polynomial in J[X], $J = Z[y_1,\ldots,y_v]$,

$b = (b_1,\ldots,b_v)$ be a given set of integral values, and

$S = (y_1-b_1,\ldots,y_v-b_v)$ such that the leading coefficient of

F satisfies $(lc(F))_b \not\equiv 0$ or $lc(F) \not\equiv 0 \pmod S$. Assume F

has a factorization in J[x] such that F = G H where $G_b = G_1$,

$H_b = H_1$ and $\gcd(G_1, H_1) = 1$, then there exists a positive n

sufficiently large (in fact, n = degree bound of F or $\max_i t_i^{d_i}$

always suffices) such that $G = G_n$, $H = H_n$, $F = G_n H_n$ over Z

$G_n \equiv G_1$ and $H_n \equiv H_1 \pmod S$.

Proof: If we consider G_1 and H_1 to be polynomials in

(Z/Z)[x] which is an Euclidean domain, then we can apply

Algorithm II-2.1 and get $A_1(x)$, $B_1(x)$ in (Z/Z)[x] such that

$A_1 G_1 + B_1 H_1 \equiv 1$ and $\deg(A_1) < \deg(H_1)$, $\deg(B_1) < \deg(G_1)$.

Similar to the proof of Theorem II-2.5, we will construct sequences of polynomials $\{G_m\}$ and $\{H_m\}$ IN $J[x]$ by induction such that $F \equiv G_m H_m$ in $(J/S^m)[x]$ and $G_m \equiv G_1$, $H_m \equiv H_1 \pmod S$. For the inductive step, we let

$$R_m(x, y_1, \ldots, y_v) = F - G_m H_m \pmod{S^{m+1}}$$

$$= \sum_{Im} C_{Im}(x)\, Mm(y_1, \ldots, y_v) \text{ where each } C_{Im}(x) \text{ is in } Z[x].$$

For each $C_{Im}(x)$ we apply Algorithm II-2.2 on G_1, H_1, A_1, B_1, and C_{Im} as if we are in $(Z/Z)[x]$ and obtain A_{Im}, B_{Im} in $(Z/Z)[x]$ such that $A_{Im} G_1 + B_{Im} H_1 = C_{Im}$ and $\deg(A_{Im}) < \deg(H_1)$. Assume G and H are put in the generalized Taylor series form where $G = G_1 + G^{(1)} + \ldots + G^{(n)}$ and $H = H_1 + H^{(1)} + \ldots + H^{(n)}$. But then

$$G^{(m)} = \sum_{Im} B'_{Im}(x)\, Mm(y_1, \ldots, y_v) \text{ and}$$

$H^{(m)} = \sum_{Im} A'_{Im}(x)\, Mm(y_1,\ldots,y_v)$ where A'_{Im} and B'_{Im} are in

$Z[x]$. Furthermore, we must have $A'_{Im} G_1 + B'_{Im} H_1 = C_{Im}$ in

$Z[x]$, since otherwise G H when multiplied in their series

form can not equal to F in the integers. Also since F is

monic, $\deg(C_{Im}) \leq \deg(R_m) < \deg(F) = \deg(G_1) + \deg(H_1)$ for

all $m \geq 1$, hence $\deg(A'_{Im}) < \deg(H_1)$ and $\deg(B'_{Im}) < \deg(G_1)$.

Now, consider A'_{Im} and B'_{Im} to be polynomials in $(Z/Z)[x]$. We

then have $(A_{Im} - A'_{Im}) G_1 = (B'_{Im} - B_{Im}) H_1$ in $(Z/Z)[x]$. Since

G_1 and H_1 are relatively prime, H_1 must divide $A_{Im} - A'_{Im}$, yet

$\deg(A_{Im} - A'_{Im}) < \deg(H_1)$. Thus, $A_{Im} = A'_{Im}$ in $(Z/Z)[x]$ or,

in fact, in $Z[x]$. With this relation and the fact that H_1

is monic, we must also have $B_{Im} = B'_{Im}$ in $Z[x]$. Therefore,

the formula for $G^{(m)}$ and $H^{(m)}$ is uniquely determined by this

construction and it is routine to verify

$G_{m+1} H_{m+1} = (G_m + G^{(m)})(H_m + H^{(m)}) \equiv F \pmod{S^{m+1}}$. Also

it is clear that $G_m \equiv G_1$, $H_m \equiv H_1 \pmod{S}$ for all $m \geq 1$.

When $m = n$, the term degree bound of F, any divisor of F must be uniquely represented, or identically equal to an element in $(J/S^{n+1})[x]$. Consider, then, $G_n H_n = F \equiv G_1 H_1 \pmod{S^{n+1}}$, Lemma II-5.1 implies G_n and G_1, H_n and H_1 are associates in $(J/S^{n+1})[x]$. Since F is monic, G_n must also be monic. But G_n is, by construction, monic since G_1 is monic and $\deg(G_n^{(m)}) < \deg(G_1)$. Therefore $G_n = G_1$ over Z. Similarly $H_n = H_1$, and $F = G_n H_n$ over Z. //

In addition to demonstrating the feasibility of making multivariate Hensel construction directly over the integers, Theorem II-5.2 also shows that under suitable constraints factors or divisors of polynomials over Z can be reconstructed from their evaluations at well chosen points. We will now discuss extensions of the Zassenhaus' Algorithm and the Generalized Hensel Algorithm so that divisors of polynomials over Z can be reconstructed using these computationally more efficient "modular" algorithms.

The main additional operation necessary for achieving this is called the Restore Leading Coefficient (RLC) operation (the reason for this name will be clear later).

Theorem II-5.3 (EZ - Extended Zassenhaus)

Let $F(x)$ be a primitive polynomial in $Z[x]$ and p be a prime in Z which does not divide the leading coefficient of $F(x)$. Assume $G(x)$ in $Z[x]$ is a divisor of F, $F = G\,H$, such that $G_0 = G \pmod p$ and $H_0 = H \pmod p$ are relatively prime in $(Z/p)[x]$. Then for a sufficiently large integer $k > 0$, the $G_k(x)$ and $H_k(x)$ in $(Z/q_k)[x]$ of Lemma II-3.2, where $q_k = p^{2^k}$, found by applying Algorithm II-3.2 on F, G_0, and H_0 can be used to reconstruct G and H directly.

Proof: By Lemma II-3.2 we can find, for any $k > 0$, $G_k(x)$ and $H_k(x)$ in $(Z/q_k)[x]$ such that $F \equiv G_k H_k \pmod{q_k}$ and $G_k \equiv G_0$, $H_k \equiv H_0 \pmod p$. Let k be chosen such that $q_k > 2\,B\,lc(F)$ where B in Z bounds the magnitude of coefficients of F and any of its divisors with the particular degrees of G_0 and H_0. (One such absolute but large bound B can be computed from F by the formula $(2r)^d/(3d/2)^{1/2}$ where $d = \max(\deg(G), \deg(H))$ and r = maximum magnitude of all complex roots of F - for its computation, see [KNU69]. This formula is derived by applying Sterling's approximation to

the maximum coefficient of $(x + r)^d$.) For this k, any divisor D of F can at most differ from the canonical representative of its equivalence class in $(Z/q_k)[x]$ by a unit in (Z/q_k) i.e. $D/lc(D) \pmod{q_k}$ is unit normal or is the unique canonical class representative.

By Lemma II-5.1, G and G_k, H and H_k are associates in $(Z/q_k)[x]$. Since lc(F) is a unit in (Z/p) hence also in (Z/q_k), so lc(G) and lc(H) are also units in (Z/q_k). Since $lc(F) = lc(G_k)\, lc(H_k)$, $lc(G_k)$ and $lc(H_k)$ must also be units in (Z/q_k). Then, since q_k is sufficiently large,

$$H/lc(H) \pmod{q_k} = H_k/lc(H_k) \pmod{q_k} \Rightarrow$$

$$H' = lc(F)\, H/lc(H) \pmod{q_k} = H'' = lc(F)\, H_k/lc(H_k) \pmod{q_k}$$

where the equality now holds over Z. But $lc(F)/lc(H) = lc(G)$ in Z[x], hence lc(G) also divides into H". In fact, since F is primitive, so must H be. Hence, we must have

$lc(G) = cont(H') = cont(H")$ and

$$H = pp(H') = pp(H") = pp(lc(F)\, (H_k/lc(H_k)) \pmod{q_k}) \text{ over } Z.$$

Similarly $G = pp(lc(F)\, (G_k/lc(G_k)) \pmod{q_k})$ over Z.

(Actually, once H has been computed, lc(G) = lc(F)/lc(H) and

$$G = lc(G)\, G_k / lc(G_k) \pmod{q_k}$$

$$= cont(H')\, G_k / lc(G_k) \pmod{q_k}.) \qquad //$$

Example II-5.3: Let us now continue from Example II-3.2, where given $F = x^5 + 12x^4 - 22x^3 - 193x^2 + 309x - 119$ and $p = 5$ we used the quadratic construction to compute $G_2 = x^3 - 15x - 17$ and $H_2 = x^2 + 12x - 7$ from the modulo p cofactors of F, $G_0 = x^3 + 2$ and $H_0 = x^2 + 2x - 2$. Note that Example II-3.2 gives us $G_2 H_2 \equiv F \pmod{q_2 = 625}$ and the modulus 625 clearly exceeds the bound of the numerical coefficients in any involved polynomial. Also, since F, G_2, H_2 are monic, there is nothing to be done for the restore-leading-coefficient operation. Thus we get directly from the results of Example II-3.2, G_2 and H_2 such that $F = G_2 H_2$ over the integers. It should be realized that had F been nonmonic, G_2 and H_2 obtained from the mod p factorization of F (some G_0 and H_0) would not be factors of F over Z, so

that they must have the Restore Leading Coefficient operations performed on them. Since this fact will be demonstrated clearly in the next example involving multivariate polynomials, we will not dwell on it here but simply point out that so long as the modulus is sufficiently large, the modular factors of F can actually be used to recover the correct factors of F over Z regardless of whether F is monic or not.

Algorithm II-5.3: (Univariate EZ Algorithm)

Input: F(x) primitive in Z[x], p prime in Z which does not divide lc(F), and $G_0(x)$, $H_0(x)$ relatively prime in (Z/p)[x] such that $F \equiv G_0 H_0 \pmod p$.

output: If F has divisors G and H such that $F = G H$ in Z and $G \equiv G_0$, $H \equiv H_0 \pmod p$, then G and H will be returned. Otherwise the outputs will be G_k, H_k, and $q_k = p^{2^k}$ such that $F \equiv G_k H_k \pmod{q_k}$ and q_k bounds the coefficients of F and any of its divisors with degrees less than $\max(\deg(G_0), \deg(H_0))$.

(1) Set d <-- $\max(\deg(G_0), \deg(H_0))$,

r <-- maximum magnitude of all roots of F,

$$B \leftarrow (2r)^{d}/(3d/2)^{1/2}, \text{ and}$$

$k \leftarrow$ minimum integer such that $q_k > 2\,B\,lc(F)$.

(2) Apply Zassenhaus' Quadratic Extension Algorithm (II-3.2) on F, p, k, G_0, and H_0 to obtain $G_k(x)$ and $H_k(x)$ in $(Z/q_k)[x]$ such that $F \equiv G_k H_k \pmod{q_k}$ and $G_k \equiv G_0$, $H_k \equiv H_0 \pmod{p}$.

(3) Restore Leading Coefficient:

Set $H \leftarrow lc(F)\,(H_k/lc(H_k)) \pmod{q_k}$.

If H divides $(lc(F)\,F)$ over Z, then set $H \leftarrow pp(H)$ over Z, $G \leftarrow cont(H)\,(G_k/lc(G_k)) \pmod{q_k}$, and return G and H.

(4) Otherwise return G_k, H_k, and q_k.

In the multivariate case, we can similarly extend the Generalized Hensel Algorithm.

Theorem II-5.4: (Multivariate EZ)

Let $F(x,y_1,\ldots,y_v)$ be a multivariate polynomial in $(Z[y_1,\ldots,y_v])[x]$ primitive w.r.t. (with respect to) x. Let p be a prime in Z and $b = \{b_1,\ldots,b_v\}$ be a set of integral values such that $(lc(F))_b \not\equiv 0 \pmod{p}$. Assume there exist G and H in $(Z[y_1,\ldots,y_v])[x]$ such that $F = G\,H$ and

$G_1 = G \pmod{q_k, S}$, $H_1 = H \pmod{q_k, S}$ are relatively prime in $(Z/q_k)[x]$ or $(J/S)[x]$ where $J = (Z/q_k)[y_1,\ldots,y_v]$. For sufficiently large integers $k > 0$ and $n > 0$, the G_n and H_n in $(J/S^n)[x]$ found by applying the Generalized Hensel Algorithm (II-4.1) on F, G_1, and H_1 can be used to reconstruct G and H directly.

Proof: Let B in Z be larger than the magnitude of any numerical coefficients of F and any of its divisors with the particular degrees of G_1 and H_1. (One such coefficient bound can be computed using a method discussed in [W&R73]. In most cases, it is sufficient to let B be the maximum of the coefficient magnitudes of F and the bound computed from F_b using the method discussed in the proof of Theorem II-5.3.) Let $k > 0$ be such that

$$q_k > 2B^2 \geq 2B \text{ (coefficient bound of } lc(F)).$$

Let n' be the term degree bound for F and n" be the maximum term degree of lc(F). Now let $n = n' + n'' + 1$ which bounds all the term degrees of lc(F) F. Then for such k and n, any divisor D of F can at most differ from the canonical representative of its equivalence class in $(J/S^n)[x]$ by a unit

n n
in $(J/S^{\ })$ i.e. $D/lc(D) \pmod{q_k, S^{\ }}$ is unit normal or is

actually the canonical class representative.

By Theorem II-4.1, we can find G_n and H_n in $(J/S^{n})[x]$

from F, G_1, and H_1 such that $F \doteq G_n H_n$ in $(J/S^{n})[x]$ and

$G_n \doteq G_1$, $H_n \doteq H_1$ in $(J/S)[x]$. By Lemma II-5.1, G_1 and G_n,

H_1 and H_n are associates in $(J/S^{n})[x]$. Since $lc(F)$ is a

unit in J/S, hence also in J/S^{n}, so must $lc(G)$ and $lc(H)$ be

units in J/S^{n}. Since $lc(F) \doteq lc(G_n)\, lc(H_n)$, $lc(G_n)$ and

$lc(H_n)$ must also be units in (J/S^{n}). Then, because q_k and

S^{n} is sufficiently large, we have $H' = lc(F)\, H/lc(H)$ in

$(J/S^{n})[x]$ and $H'' = lc(F)\, H_n/lc(H_n)$ in $(J/S^{n})[x]$ identically

equal or equal in $(Z[y_1,\ldots,y_v])[x]$. But $lc(F)/lc(H) = lc(G)$

in $(Z[y_1,\ldots,y_v])[x]$ so that the expression H'' considered as

a polynomial in $(Z[y_1,\ldots,y_v])[x]$ must also be divisible by

lc(G). In fact, since F is primitive, so must H be, and lc(G) = cont(H') = cont(H"), H = pp(H') = pp(H"). Similarly, $G_n = pp(lc(F)\ (G_n/lc(G_n))\ (\mathrm{mod}\ q, S_k^n))$. (In fact, with H_n known now, $lc(G) = lc(F)/lc(H) = cont(H")$ and $G_n = cont(H")\ (G_n/lc(G_n))\ (\mathrm{mod}\ q, S_k^n)$ as it is considered as a polynomial in $(Z[y_1,\ldots,y_v])[x]$.) //

Example II-5.4a: This one will be the continuation from Example II-4.1a where F was given to be

$$x^2 + (-z^2 - y\,z + y + z + 2)\,x + y\,z^3 - z^3 + y\,z^2 - y\,z + 2\,z,$$

b = (0, 0), q = p = 7 and the univariate codivisors of F, $G_1 = x$ and $H_1 = x + 2$. Since q = 7, n = 3 happened to be large enough, and F was given monic, there is really nothing much to continue. The results of applying the Generalized Hensel Algorithm to F, G_1, and H_1 are $G_3 = x + z - y\,z$ and $H_3 = x + y - z^2 + 2$ such that $F \equiv G_3\,H_3 \ (\mathrm{mod}\ q, S_3)$. But there is no need for performing the Restore Leading Coefficient operations on G_3 and H_3 because they are monic. Thus, in fact $F = G_3\,H_3 = G\,H$ over the integers with $G = G_3$ and

$H = H_3$.

Example II-5.4b: Continuing from Example II-4.1b, we have the given non-monic multivariate polynomial

$$F = y^2 x^2 + (-y^2 z^2 - y^2 z + y z^2 + y^2 + 2y) x + y^3 z^3 - z^2 - y^2 z - y z + 2 z$$

with $b = (1, 0)$, $p = 7$, $G_1 = x$, and $H_1 = x + 3$. The goal here is to find the factors over the integers, G and H, corresponding to G_1 and H_1 in $(Z/p)[x]$. Since we know that $k = 1$ and $n = 4$ is sufficiently large for the general modulus, the results from Example II-4.1b, by applying the Generalized Hensel construction on F, G_1, and H_1, are exactly what we need here. We had

$$G_4 = (y'^2 + 2y' + 1) x - (y' + 1) y' z,$$

$$H_4 = x + 3 - 2y' + 2y'^2 - z^2 - 2y'^3 + y'^2 z, \text{ and}$$

$F' = G_4 H_4 \pmod{7, S'^4}$. We will now apply the Restore Leading Coefficient operations to G_4 and H_4, working with F' in the more convenient modular space S'^4 (rather than F in

S_4). For simplicity and by symmetry, we first compute

$G'' = lc(F') \; G_4 / lc(G_4)$ in $(J/S'^4)[x]$. Since

$lc(G_4) = lc(F') = (y'^2 + 2y' + 1)$, we have

$$G'' = G_4 \pmod{7, S'^4} = G_4 = (y'^2 + 2y' + 1)x - (y' + 1)y'z.$$

Considering G'' as polynomials over Z (since 7 is a sufficiently large modulus) and computing the primitive part of it, we get $(y' + 1)x - y'z$. But then G can be recovered from this by the substitution of $y - 1$ for y' in this expression giving $yx - yz + z = G$. For recovering H, we take the corresponding content of G'', $y' + 1$, and get

$$cont(G'') \; H_4 / lc(H_4) \pmod{7, S'^4}$$

$$= (y' + 1)x + 3(y' + 1) - 2y'(y' + 1)$$

$$+ 2y'^2(y' + 1) - z^2(y' + 1) - 2y'^3 + y'^2 z$$

$$= (y' + 1)x + 3 + y' - z^2 = (y' + 1)x + (y' + 1) + 2 - z^2$$

$$= yx + y - z^2 + 2 = H.$$ Then we have shown how the divisors of F over Z can be recovered from their univariate images, G_1 and H_1, via the application of the Generalized Hensel construction, to a sufficiently large modular space

(in this case S_4), and after Restore Leading Coefficient operations if necessary. The lack of resemblance between G_4 and G, H_4 and H mentioned in Example II-4.1b, turned out not to be a serious hindrance to the recovery of G and H. But the added complications and computations due to non-trivial leading coefficients have also been shown to be potentially costly. If we compare the computations involved in Examples II-5.4a and II-5.4b as well as in Examples II-4.1a and II-4.1b, the amount of extra work (for example the substitution transformations, the one additional iteration necessary for the Generalized Hensel construction, and the Restore Leading Coefficient operation) due to the simple but non-trivial leading coefficient y'^2 is quite substantial. This problem will be the topic of discussion in the next section (II-5(a)).

Algorithm II-5.4: (Multivariate EZ Algorithm)

Input: $F(x,y_1,\ldots,y_v)$ primitive in $(Z[y_1,\ldots,y_v])[x]$, $b = (b_1,\ldots,b_v)$ a set of integers in Z such that $(lc(F))_b \not\equiv 0 \pmod p$, p a lucky prime for F_b in Z, and $G_1(x)$, $H_1(x)$ relatively prime such that $F_b = G_1 H_1$ in

(J/S)[x].

Output: If F has divisors G and H such that F = G H in $Z[x,y_1,\ldots,y_v]$ and $G \equiv G_1$, $H \equiv H_1$ in (J/S)[x], then G and H will be returned. Otherwise, the outputs will be G_n, H_n, q_k, and n such that $F \equiv G_n H_n \pmod{q_k, S^n}$.

(1) Compute B as suggested in the above Proof.

Set L <-- maximum coefficient magnitude of lc(F).

k <-- minimum integer such that $q_k > 2BL$.

n' <-- term degree bound of $F = \max_i td_i$.

n" <-- maximum term degree of lc(F).

n <-- n' + n" + 1.

(2) Apply Generalized Hensel Algorithm II-4.1 on F, b, q_k, G_1, and H_1 to obtain $G_n(x,y_1,\ldots,y_v)$ and $H_n(x,y_1,\ldots,y_v)$ such that $F \equiv G_n H_n$ in $(J/S^n)[x]$ and $G_n \equiv G_1$, $H_n \equiv H_1$ in (J/S)[x].

(3) Restore Leading Coefficient:

Set $H \leftarrow lc(F)\,(H_n/lc(H_n)) \pmod{q_k, S^n}$.

If H divides (lc(F) F) over Z, then set

$H \leftarrow\!- pp(H)$ over $(Z[y_1,\ldots,y_v])[x]$,

$G \leftarrow\!- cont(H)\ (G_n / lc(G_n))\ (\mathrm{mod}\ q_k, S^n)$,

and return G and H.

(4) Otherwise return G_n, H_n, q_k, and n.

II - 5 (a) The Leading Coefficient Problem

We now observe that the basic underlying step of all of the Hensel-type algorithms is solving the Diophantine Univariate Polynomial Equations (DUPE) of the form $A G + B H = C$ where G, H, and C are given polynomials in $(Z/q)[x]$, $\gcd(G, H) = 1$ and q is some power of a prime p. As in the case of integral Diophantine equations, solutions to DUPE are not unique. However, if we impose the condition that lc(G) be a unit in Z/p and $\deg(A) < \deg(H)$ then the solutions A and B are unique up to units in Z/p. Unfortunately, this artificially imposed condition influences the Hensel-type of construction process so that the leading coefficient of one of the factors, say H, is never updated (see Example II-4.1b). For instance, in the Generalized Hensel Algorithm, we have

$H_{k+1} = H_1 + H^{(1)} + \ldots + H^{(k)}$ where each $H^{(m)}$ is a linear

combination of the A's which are computed by solving corres-

ponding DUPE's. Since deg(A_{Im}) < deg(H_1) for all Im, the leading coefficient of H_{k+1} is always the same as the leading coefficient of H_1 which is only congruent to the leading coefficient of the true factor H modulo S but usually not identically equal to it over the integers. Thus, as we have seen in the univariate and the multivariate EZ Theorem, for $F = G\,H$, the Hensel constructions result in G_n and H_n for sufficiently large n such that $F \equiv G_n H_n$ and $G \equiv G_n$, $H \equiv H_n$, but usually $G \neq G_n$ and $H \neq H_n$ over the integers. The reason for such a situation is that essentially all of the leading coefficient of F has been artificially forced onto the leading coefficient of G_n. In fact if the leading coefficient of H_1 were divided out of H_1, i.e. $H_1 = H_1/lc(H_1)$, $G_1 = lc(H_1)\,G_1$, and still $F \equiv G_1 H_1 \pmod S$, then the Hensel construction using these G_1 and H_1 will automatically force $lc(G_n) = lc(F)$ and $lc(H_n) = 1$. In this case, the Restore Leading Coefficient operation becomes simply $G' = pp(G_n)$ and $H' = cont(G_n)\,H_n \pmod{S_n}$. Thus, in general, we have

$G = G_n/u$ where $u = lc(H)$ which is a unit in J/S^n, and

$F = G\,H = ((1/u)\,G_n)\,(u\,H_n)$ where multiplications by units

are done in J/S^n. Many such phenomena (in their simplified versions) can be seen from Example II-5.4b and its preceeding related examples. We now point out that if

$u = 1 + y_1 + y_2 + \ldots + y_v \pmod{y_1^d, y_2^d, \ldots, y_v^d}$ then $1/u$ is a

dense polynomial in the y's of degree less than d in each,

which is a polynomial of d^v terms. The relationships of G, H, G_n, and H_n in J/S^n are $G_n = u\,G$ and $H_n = (1/u)\,H$. This means that the Hensel method is reconstructing the factors G_n and H_n where each coefficient of G is multiplied by u and each coefficient of H is multiplied by (1/u). This is an expression blowup due to non-uniqueness of solutions to DUPEs relating essentially to the leading coefficient of F. We will address this problem as the Leading Coefficient Blowup (LCB) problem.

Referring back to Theorem II-5.2 where we assumed F to be monic, we noted in the proof of that theorem that $\deg(C_{Im}) \leq \deg(R_m) < \deg(F) = \deg(G_1) + \deg(H_1)$. With this

condition holding, Corollary II-2.2 implies that the corresponding A_{lm} and B_{lm} satisfy $\deg(A_{lm}) < \deg(H_1)$ and $\deg(B_{lm}) < \deg(G_1)$. Thus, $\deg(G^{(m)}) < \deg(G_1)$ and $\deg(H^{(m)}) < \deg(H_1)$ so that $lc(G_n) = lc(G_1)$, $lc(H_n) = lc(H_1)$, and, in fact, $lc(G) = lc(G_n) = lc(H) = lc(H_n) = lc(F) = 1$.

The correct leading coefficients for G_n and H_n made G_n and H_n the unique canonical representative of the equivalence classes for G and H in $(J/S^n)[x]$ so that $G = G_n$ and $H = H_n$ over the integers when n is sufficiently large to bound the term degrees in F, G, and H. We now observe that if F is monic, then the same arguments hold even if the integral arithmetic was carried out in Z/q^k for k sufficiently large to bound all the numerical coefficients of F, G, and H (as in Theorem II-5.3 and II-5.4). Examples II-4.1a and II-5.4a clearly demonstrate these points. We further note that the important condition is not that F is monic but rather that $lc(G) = lc(G_n)$ and $lc(H) = lc(H_n)$. (See Remark following Theorem II-4.1.) This leads us to the next lemma which, though only constitutes an artificial solution to the LCB

problem, will later prove to avoid the LCB problem at a small cost for some applications of the Hensel construction. However, we will emphasize that this method of treating the leading coefficients is indeed "sparseness-preserving".

For univariate polynomials a similar version of the following lemma applies. Since we have pointed out the effects when F is monic in the remark following Lemma II-3.2, we will now concentrate only on the multivariate case. In fact, the Restore Leading Coefficient operation for univariate polynomials essentially only involves the additional step of computing integral contents and primitive parts that it is a much less expensive operation than in the multivariate case.

Lemma II-5.5: Let $F(x,y_1,\ldots,y_v)$ be in $(Z[y_1,\ldots,y_v])[x]$ and primitive w.r.t. x. Let $b = (b_1,\ldots,b_v)$ be a set of integral values and p be a lucky prime for F in Z such that $(lc(F))_b \not\equiv 0 \pmod p$. Assume there exist G and H in $(Z[y_1,\ldots,y_v])[x]$ such that $F = G\,H$ and $G_1 = G \pmod{q_k, S}$, $H_1 = H \pmod{q_k, S}$ are relatively prime in $(Z/q_k)[x]$ or $(J/S)[x]$ where $J = (Z/q_k)[y_1,\ldots,y_v]$. Then, with F, G_1, H_1, lc(G), and lc(H), we can apply a modi-

fied version of the Generalized Hensel Algorithm so that the outputs of it are G and H.

Proof: Before we proceed with the proof, we define a replacement function, the use of which was first suggested by Moses, replaceIc(P, L) = P with its leading coefficient replaced by L. As in proof of Theorem II-4.1, we have $F \equiv G_1 H_1 \pmod{q, S_k}$ so that A_1 and B_1 can be found in (J/S)[x] such that $A_1 G_1 + B_1 H_1 \equiv 1$ and $\deg(A_1) < \deg(H_1)$, $\deg(B_1) < \deg(G_1)$. Now let $G'_1 = \text{replaceIc}(G_1, \text{lc}(G))$ and $H'_1 = \text{replaceIc}(H_1, \text{lc}(H))$. For the mth step of the inductive process, we use $G'_m = G'_1 + G_1^{(1)} + \ldots + G_1^{(m-1)}$ and $H'_m = H'_1 + H_1^{(1)} + \ldots + H_1^{(m-1)}$ to compute $R_m = F - G'_m H'_m \pmod{q, S_k^{m+1}}$. Thus we have $\deg(C_{Im}) \leq \deg(R_m) < \deg(F) = \deg(G_1) + \deg(H_1)$ since now $\text{lc}(G'_m)\,\text{lc}(H'_m) = \text{lc}(F)$ over the integers. However, since $G'_1 \equiv G_1$ and $H'_1 \equiv H_1$ in J/S, we can still apply Algorithm II-2.2 on G_1, H_1, A_1, B_1, and C_{Im} in J/S to compute A_{Im} and

B_{Im} such that $A_{Im}G_1 + B_{Im}H_1 = C_{Im}$. But now Corollary 2.2 implies $\deg(A_{Im}) < \deg(H_1)$ and $\deg(B_{Im}) < \deg(G_1)$. Hence

$\deg(G_1^{(m)}) < \deg(G_1)$, $\deg(H_1^{(m)}) < \deg(H_1)$ and

$lc(G'_{m+1}) = lc(G'_m) = lc(G'_1) = lc(G)$, $lc(H'_{m+1}) = lc(H)$. Therefore, for sufficiently large k and n (see proof of Theorem II-5.4), we have $F \equiv G'_n H'_n \pmod{q^n, S_k}$. But because of the identities in the leading coefficients, we must have $G = G'_n$ and $H = H'_n$ over the integers. //

Example II-5.5: To demonstrate the feasibility and the necessary additional operations of this modified Hensel construction, we work out a problem which is closely related to that of Example II-5.4b. Let F be the product of y^2 and the given polynomial of Example II-5.4b, so

$$F = y^4x^2 + (-y^3z^2 - y^4z + y^3z + y^4 + 2y^3)x + y^3z^3 - y^2z^3 - y^4z - y^3z + 2y^2z.$$

Then, over the integers, F has codivisors G and H where

$$G = y(yx + z - yz) = y^2x + yz - y^2z \text{ and}$$

$$H = y(yx + y - z^2 + 2) = y^2x + y^2 - yz^2 + 2y.$$

With b = (1, 0) and p = 7, G_1 and H_1 are the same as those in Example II-5.4b, where $G_1 = x$ and $H_1 = x + 3$. Here, we assume it is known that $lc(G) = y^2$ and $lc(H) = y^2$ are the correct multivariate leading coefficients of G_1 and H_1. So we apply the replacement function to get

$G'_1 = replacelc(G_1, lc(G)) = y^2 x$ and

$H'_1 = replacelc(H_1, lc(H)) = y^2 x + 3$. As in Example II-5.4b, we get the same A_{00}, B_{00}, A_{01}, B_{01}, A_{02}, and B_{02} such that $A_{0i} G_1 + B_{0i} H_1 \equiv x^i \pmod 7$ for i=0, 1, 2.

Another preparatory step for the Hensel construction is, as before, the substitution transformation, y' + 1 for y. In this case, we apply this transformation to G'_1, H'_1, and F to get $G^*_1 = G'_1 (y=y'+1, z) = (y'^2 + 2y' + 1) x$,

$H'_1 = H'_1 (y=y'+1, z) = (y'^2 + 2y' + 1) x + 3$, and

F' = F(y=y'+1, z). Then the Hensel construction iterations are the same as before.

First we compute $D_1 = F' - G' H'$ and

$R_1 = D_1 \pmod{7, S'^2}$. We get $R_1 = -3 y' x = C_1(x) y'$.

Thus, simply $A_1(x) = -3$, $B_1(x) = 0$ such that

$A_1 G_1 + B_1 H_1 \equiv C_1 \pmod 7$ so that $G^{(1)} = B_1 y' = 0$,

$H^{(1)} = A_1 y' = -3 y'$,

$G'_2 = G'_1 + G^{(1)} = G'_1 = (y'^2 + 2y' + 1) x$, and

$H'_2 = H'_1 + H^{(1)} = (y'^2 + 2y' + 1) x + 3 - 3y'$ such that

$G'_2 H'_2 \equiv F' \pmod{7, S'^2}$. Next we compute

$D_2 = D_1 - G'_1 H^{(1)} - G^{(1)} H'_1 - G^{(1)} H^{(1)}$ and

$R_2 = D_2 \pmod{7, S'^3} = (y'^2 - y' z - z^2) x - 3 y' z$

$= (x) y'^2 + (-x - 3) y' z + (-x) z^2$

$= C_{11}(x) y'^2 + C_{12}(x) y' z + C_{22}(x) z^2$. By scalar linear

combinations using A_{0i}, B_{0i} $i = 0, 1, 2$, we get $A_{11} = 1$,

$B_{11} = 0$, $A_{12} = 0$, $B_{12} = -1$, $A_{22} = -1$, and $B_{22} = 0$. Thus

$G^{(2)} = B_{11} y'^2 + B_{12} y' z + B_{22} z^2 = -y' z$ and

$H^{(2)} = A_{11} y'^2 + A_{22} z^2 = y'^2 - z^2$, so that

$$G'_3 = G'_2 + G^{(2)} = (y'^2 + 2y' + 1)x - y' z,$$

$$H'_3 = H'_2 + H^{(2)} = (y'^2 + 2y' + 1)x + 3 - 3y' + y'^2 - z^2,$$

and $G'_3 H'_3 \equiv F' \pmod{7, S'^3}$. Finally, we compute

$$D_3 = D_2 - G'_2 H^{(2)} - G^{(2)} H'_2 - G^{(2)} H^{(2)} \text{ and}$$

$$R_3 = D_3 \pmod{7, S'^4} = (-y'^2 z - y' z^2)x - 3y'^2 z$$

$$= (-x - 3)y'^2 z + (-x)y' z^2$$

$$= C_{112}(x) y'^2 z + C_{122}(x) y' z^2.$$ it is easily computed

that $A_{112} = 0$, $B_{112} = -1$, $A_{122} = -1$, and $B_{122} = 0$. Thus

$G^{(3)} = B_{112} y'^2 z = -y'^2 z$ and $H^{(3)} = A_{122} y' z = -y' z^2$,

so that $G'_4 = G'_3 + G^{(3)} = (y'^2 + 2y' + 1)x - y' z - y'^2 z,$

$$H'_4 = H'_3 + H^{(3)} = (y'^2 + 2y' + 1)x + 3 - 3y' + y'^2 z^2 - y' z^2,$$

such that $G'_4 H'_4 \equiv F \pmod{7, S'^4}$. In fact,

$D_4 = F - G'_4 H'_4 = 0$, i.e. $F' = G'_4 H'_4$ over Z, so the Hensel construction terminates here and gives

$G = G'_4 (y'=y-1, z) = y^2 x + y z - y^2 z$ and

$H = H'_4 (y'=y-1, z) = y^2 x + y^2 - y z^2 + 2y$ as results,

which is indeed the correct codivisors of F over Z. Note that only three iterations of the Hensel construction were needed in this case, where the term degree bound for F is actually 6, so that even though the problem is a larger one than Example II-5.4b, the number of iterations remain the same. Therefore, we have shown with this example, that knowing the correct multivariate leading coefficients avoids the LCB problem (Compare the number of terms in H'_4 with that of H_4 in Example II-5.4b.) at the cost of working with a larger given polynomial but without having to go through any more iterations of the Hensel construction than before. This is definitely an improvement over the regular Hensel construction so long as we can be given the knowledge of the

correct way of splitting the leading coefficient of the given polynomial into the leading coefficients of its univariate codivisors. In Chapter IV, we will see how this knowledge can actually be obtained at a small cost thereby avoiding large cost due to the LCB problem and preserve the sparseness of the polynomials during the Hensel construction.

Algorithm II-5.5:

Input: Same as those for Multivariate EZ Algorithm II-5.4 except, in addition, g and h in $Z[y_1,\ldots,y_v]$ are given such that $g_b = lc(G_1)$ and $h_b = lc(H_1)$.

Output: If F has divisors G and H such that $F = G\,H$ and $lc(G) = g$, $lc(H) = h$, then G and H will be returned. Otherwise the outputs will be G_n, H_n, q_k and n such that

$$F \equiv G_n\,H_n \pmod{q_k, S^n}.$$

(1) Compute bounds k and n as in Step (1) of Algorithm II-5.4.

(2) Return the result of applying GHA on F, $G' = \text{replacelc}(G_1, g)$, and $H' = \text{replace}(H_1, h)$.

CHAPTER III POLYNOMIAL GREATEST COMMON DIVISORS

III - 1 Introduction

As already noted, polynomial GCD computations basically rely on division, especially for the PRS GCD algorithms. But the process of finding the GCD of two polynomials, F and G, also has many features in common with polynomial factorization. Clearly, if we have the complete factorizations of F and G, then their GCD can simply be recognized by taking the common factors raised to the respective minimum powers. Even without the complete factorizations, though, the GCD, D, of F and G still breaks up F and G into incomplete factorizations, F = D (F/D) and G = D (G/D). It is these factorizations that we usually seek in GCD computations, where we often require not only the GCD, but also the cofactors, F/D and G/D (for example, in reducing F/G to lowest terms). The Hensel-type GCD algorithm will always compute the cofactors as a byproduct of the built-in trial division process, just as Brown's Modular GCD Algorithm will obtain them as a byproduct of the process of verifying the GCD.

The Hensel-type GCD algorithm uses the modular and evaluation homomorphisms, just as the Modular GCD Algorithm does. The main distinction is in the process of inverting the mappings. Here, we use the Hensel-type constructions

which make good use of one well chosen homomorphic image instead of several images required by the Chinese Remainder Algorithm and the interpolation process. This is actually the only distinction in the univariate case. For multivariate GCD computations, however, the Generalized Hensel Algorithm enables the simultaneous constructions (or inversions) in several variables instead of recursively inverting one variable at a time. This is why the multivariate EZGCD Algorithm often compares much more favorably to other GCD algorithms than the univariate UNIGCD Algorithm would. For this reason (and because the computation of univariate GCDs are not so costly, anyway) the univariate case will be given less emphasis in the following discussion.

III - 2 Basic Concepts and an Overview of EZGCD Algorithm

We will now present the four basic steps of the GCD algorithm by Hensel-type constructions. This overview will show the similarities of this method with the Modular GCD Algorithm and with Wang and Rothschild's polynomial factorization algorithm. Then, we will also prove a fundamental theorem which will show why this new method for GCD computation is possible. For both of these tasks, we will mainly discuss the more important multivariate case and only point out that the univariate case follows similarly

and more simply so that it need not be repeated here.

First, we prove a result that is essential for the understanding of GCD computations via homomorphic images in more structured subdomains. This result is implicitly stated in Brown's [BRD71] discussions on "unlucky" primes and b-values, and its proof can be implicitly derived from his Theorem 1 and 4 if one understands the theory of polynomial PRS's and subresultants. But its importance in our ensuing discussions warrants its individual treatment here.

Lemma III-2.1: Let F and G be in J[x] where J is any u.f.d. Let h be a homomorphism from J to a subdomain of J whose kernel is some ideal I of J, i.e. h: J --> J' = J/I. Assume h(lc(F)) $\neq$ 0 and h(lc(G)) $\neq$ 0. If D = gcd(F, G) and D' = gcd(h(F), h(G)) where h(F) means applying h on all coefficients of F, then deg(D(x)) $\leq$ deg(D'(x)).

Proof: Since h is a homomorphism, D divides F and G so that there exists F' and G' in J such that F = D F' and G = D G' ==> h(F) = h(D) h(F') and h(G) = h(D) h(G') or h(D) divides h(F) and h(G) ==> h(D) divides D', by definition of GCD ==> deg(h(D)) $\leq$ deg(D'(x)). But h(lc(D)) also $\neq$ 0, for if h(lc(D)) = 0 then since lc(D) divides lc(F) and lc(G) so lc(F) = lc(G) = 0, contradicting assumptions. Therefore deg(D(x)) = deg(h(D)) $\leq$ deg(D'(x)). //

In our applications, we have the modular homomorphisms determined by a prime p and the evaluation homomorphisms determined by a chosen set of integers $b = (b_1,\ldots,b_v)$. According to Brown, the particular chosen p or b is lucky if the homomorphism determined by it satisfies $\deg(h(D)) = \deg(D' = \gcd(h(F), h(G)))$, and unlucky otherwise. Brown's Theorem 1 and 4 in [BR071] give another version of the definition of the "luckiness" in terms of subresultants of two polynomials. In general, we will call a particular homomorphism lucky for a single polynomial P if the homomorphic images of any two distinct factors of P are not the same in the image domain.

The EZGCD Algorithm will compute the GCD, $D'(x,y_1,\ldots,y_v)$, of two multivariate polynomials $F'(x,y_1,\ldots,y_v)$ and $G'(x,y_1,\ldots,y_v)$ in $Z[x,y_1,\ldots,y_v]$ with the cofactors F'/D' and G'/D' as byproducts of the GCD verification process.

Step I: (Contents and Primitive Parts)

Consider F' and G' as polynomials in $(Z[y_1,\ldots,y_v])[x]$

and set f' <-- cont(F'), F <-- pp(F'), g' <-- cont(G'), G <-- pp(G'), and d' <-- gcd(f', g').

The computation of content and primitive part is a recursive process, each step involving calculations of GCD's of

polynomials with one less variable. As we will see later, these computations can be accomplished using a variant of this algorithm. The remaining task for this algorithm is to compute $D = \gcd(F, G)$, where F and G are now primitive, and the corresponding cofactors F/D and G/D. Then D' is simply d' D and the corresponding cofactors are (f'/d') (F/D) and (g'/d') (G/D).

Step II: (Evaluation and Univariate GCD)

Choose a set of v integers $b = \{b_1, \ldots, b_v\}$ such that the degrees in x of F and G evaluated at $y_i = b_i$, $i=1,\ldots,v$ are not decreased. [Such an evaluation is called a valid evaluation.]

Compute $F_b(x)$, $G_b(x)$, and $D_\emptyset(x) = \gcd(F_b, G_b)$.

This univariate GCD computation can be done by using, say, the Modular GCD Algorithm which is very efficient in this case. However, it is equally efficient to use the UNIGCD Algorithm with Zassenhaus' quadratic extension algorithm in most cases, as we will see in the next section.

Step III: (Preparation for Hensel Construction)

If $\deg(D_\emptyset(x)) = 0$, then $D_\emptyset(x) = 1$ and $D = 1$.

If $\deg(D_\emptyset) = \deg(F_b)$ or $\deg(G_b)$, then F divides G, or G divides F, or a new valid evaluation should be made.

Clearly if $\deg(D_0) = 0$, then $D_0 = \text{constant}$. Since b is a valid evaluation for F and G, it must also be valid for D and so $D_0 = \text{constant}$ implies D contains no x. But F and G are primitive implies D is also primitive. Hence D must be 1. Because $\deg(D_0) \geq \deg(D)$ (see Lemma III-2.1), if $\deg(D_0) = \deg(F) \leq \deg(G)$ and F does not divide G, we know $\deg(D) < \deg(D_0)$ so that this evaluation must be unlucky so as to create an univariate GCD which is not the image of the actual GCD. Similarly, $\deg(D_0) = \deg(G) \leq \deg(F)$.

Otherwise determine if the following condition holds:

Condition III-A: $\gcd(D_0, F/D_0) = 1$ or

$$\gcd(D_0, G/D_0) = 1.$$

This is a necessary condition for the application of the Generalized Hensel Algorithm, as we have seen. This condition will not hold if $\gcd(D, F/D) \neq 1$ and $\gcd(D, G/D) \neq 1$. (The simplest class of problems for which this is true is the case where $F = U V^2$ and $G = U^2 V$, U and V are any non-trivial polynomials depentent on the main variable of the problem in question). If this is the case, the algorithm

will be forced to apply a special case method which amounts to computing a square-free decomposition of D. However, this special method also uses the Hensel-type constructions and, in many cases, proves to be comparable in cost to, if not more efficient than, the other GCD algorithms. This special method will be discussed and especially clearified when we present the Hensel-type square free decomposition algorithm in Chapter VII.

Assuming Condition III-A holds, we continue under a further assumption,

Condition III-B: $\deg(D_0) = \deg(D)$.

Since D is supposedly unknown, this assumption cannot be tested at this point. However, because it is an essential condition for the algorithm to work (see the following theorem), at various places in the algorithm we endeavor to increase the probability of having it hold. We also use a "safety-valve" test near the end of the algorithm, to ensure that any unlucky evaluation, which causes Condition III-B to be false, will be detected.

Step IV: (Application of the Hensel Construction)

Suppose F satisfies condition III-A.

Apply the Multivariate EZ Algorithm (II-5.4) on F, $D_0(x)$, and $F_b(x)/D_0(x)$ to get either

(a) multivariate cofactors, D" and H", of F such that

F = D" H" over Z, or

(b) some q_k, n, and D_n, H_n such that

$$F \equiv D_n H_n \pmod{q_k, S_n^n}.$$

We note here that only F (or G) is used in the application of the Hensel-type construction. This will be the fundamental observation for the new and, in most cases, improved Hensel-type polynomial content and primitive part algorithm which we will present later.

Step V: (GCD Verification)

For case (a), test whether D" also divide G. If so D = D" is the GCD of F and G we seek, so that D' = d' D and the cofactors can be computed by multiplying quantities already computed. Otherwise, or for case (b), go back to Step II for a new evaluation and a resulting univariate GCD having a smaller degree than this $D_\emptyset$.

From the properties of the Multivariate EZ Algorithm (Theorem II-5.4), $D"_b(x) = D_\emptyset(x)$ and D" is a proper primitive divisor of F. Thus, since $\deg(D") = \deg(D_\emptyset) \geq \deg(D)$, D" must be the GCD, D, of F and G if it also divides G.

This will be made more explicit in Theorem III-2.3.

For case (b), since q_k and n are chosen to be surely larger than the coefficient bound and the term degree bound of F respectively, b must be an unlucky evaluation so we look for another. In case it was decided to use a heuristic coefficient bound for the computation of q_k in the multivariate EZ algorithm, then case (b) does not necessarily mean that the chosen evaluation was unlucky. It may be worthwhile to apply the Hensel construction again with a larger integer modulus than q_k (perhaps, an actual coefficient bound). However, for practical multivariate problems, it is rarely the case that any reasonable heuristic coefficient bound such as the one suggested in the EZ algorithm will not suffice.

Example III-2.1: As an example of multivariate polynomial GCD computation using the above outlined EZGCD Algorithm, we assume

$$F' = x^2 + (-z^2 - y z + y + z + 2)\, x + y z^3 - z^3 + y^2 z - y z + 2 z \text{ and}$$

$$G' = x^3 + (-z^2 - y z + z)\, x^2 + (y z^3 - z^3 - y + 2)\, x + y^2 z - 3 y z + 2 z$$

are given and we want to find

$D' = \gcd(F', G')$. Since F and G are monic, they must be primitive, so that Step I can be passed through with $d' = \gcd(f', g') = \gcd(1, 1) = 1$, $F = F'$ and $G = G'$. We choose $b = (0, 0)$ which is clearly a valid evaluation for monic polynomials F and G. Then $F_b(x) = x^2 + 2x$, $G_b(x) = x^3 + 2x$, and $D_0 = \gcd(F_b, G_b) = x$. Since $\deg(D_0(x)) = 1 \neq \deg(F)$ or $\deg(G)$, we test Condition III-A and find $\gcd(D_0, F_b/D_0) = \gcd(x, x+2) = 1$. So we continue to Step III assuming Condition III-B, $\deg(D_0) = 1 = \deg(D)$ where $D = \gcd(F, G)$ is to be found. We now apply the Multivariate EZ Algorithm (II-5.4) on F, $D_0(x) = x$, and $F_b(x)/D_0(x) = x + 2$. But this problem was already done in Example II-5.4a and the results are $D'' = x + z - yz$ and $H'' = x + y - z^2 + 2$ such that $F = D''H''$ over Z (case (a) of Step IV). Since D" also divides G ($G = D''(x^2 - z^2 x - y + 2)$ over Z), we conclude that $D' = D = D'' = \gcd(F, G) = \gcd(F', G')$ and indeed it is.

We will now prove the fundamental theorems for the validity of the application of the Hensel-type construction

to polynomial GCD computations. Again, in order to avoid redundancy, we concentrate on the multivariate case.

Let $F(x,y_1,\ldots,y_v)$ and $G(x,y_1,\ldots,y_v)$ be two multivariate polynomials over the integers with GCD $D(x,y_1,\ldots,y_v)$.

For a set of integral values $b = (b_1,\ldots,b_v)$ such that $\deg(F(x)) = \deg(F_b(x))$ and $\deg(G(x)) = \deg(G_b(x))$ (i.e. $(lc(F))_b \neq 0$ and $(lc(G))_b \neq 0$.), let

$$D_0(x) = \gcd(F_b(x), G_b(x)).$$

Theorem III-2.2:

If $D_0(x)$ is assumed to have the following properties:

(i) $\deg(D_0(x)) = \deg(D(x))$,

(ii) $\gcd(D_0(x), F_b(x)/D_0(x)) = 1$,

then for any primitive divisor $D'(x,y_1,\ldots,y_v)$ of F such that $D'_b(x) = D_0(x)$, we must have

$D(x,y_1,\ldots,y_v) = D'(x,y_1,\ldots,y_v)\,Q(y_1,\ldots,y_v)$ where Q is a polynomial independent of x.

Proof: Refering to Lemma III-2.1 and its proof, we have

$D_b(x)$ dividing $D_0(x) = D'_b(x)$, $\deg(D) = \deg(D_b(x))$, and,

similarly, $\deg(D') = \deg(D'_b(x))$. By (i), we have

$$\deg(D_b(x)) = \deg(D) = \deg(D_0(x)) = \deg(D'_b(x)) = \deg(D').$$

Let $F = D F'$ and $G = D G'$, then

$D_0(x) = D_b(x) \gcd(F'_b(x), G'_b(x))$, and so $\gcd(F', G') = c$, c a constant. That is $D_0(x) = D'_b(x) = c\, D_b(x)$. Now let

$P(x, y_1, \ldots, y_v) = \gcd(D, D')$, then $D = P Q$ and $D' = P Q'$ for some Q and Q' in $Z[x, y_1, \ldots, y_v]$ such that $\gcd(Q, Q') = 1$.

Since D' is primitive, Q' must also be primitive so that either $Q' = 1$ or $\deg(Q') > 0$. Assuming the latter, we have P Q dividing P Q' (F/D') or Q dividing Q'(F/D'), since D divides F. But $\gcd(Q, Q') = 1$ so that Q divides F/D', hence Q_b divides $F_b/D'_b = F_b/D_0$. On the other hand, $P_b Q'_b = D'_b = D_0 = c\, D_b = c\, D_b Q_b$ or Q_b divides D_0. Thus Q_b divides $\gcd(D_0, F_b/D_0) = 1$, by (ii), or $Q_b = 1$ which means $Q'_b = c$ contradicting the fact that $\deg(Q'_b(x)) = \deg(Q') > 0$ by assumption. Therefore we must have $Q' = 1$ or $D = D' Q$.

But $\deg(D') = \deg(D)$ so that $\deg(Q) = 0$ and we have

$D(x, y_1, \ldots, y_v) = D'(x, y_1, \ldots, y_v)\, Q(y_1, \ldots, y_v)$ where Q is independent of x. //

Theorem III - 2.3: (Validity of the EZGCD Algorithm)

Let $F'(x,y_1,\ldots,y_v)$ and $G'(x,y_1,\ldots,y_v)$ be multivariate polynomials in $Z[x,y_1,\ldots,y_v]$ with GCD $D'(x,y_1,\ldots,y_v)$. Let $F = pp(F')$, $f' = cont(F')$, $G = pp(G')$, $g' = cont(G')$, and $D(x,y_1,\ldots,y_v) = D'/gcd(f', g')$. For any chosen set of integral values $b = (b_1,\ldots,b_v)$ satisfying the conditions $deg(F_b(x)) = deg(F)$ and $deg(G_b(x)) = deg(G)$ and resulting in a $D_\emptyset(x) = gcd(F_b(x), G_b(x))$ satisfying (i) and (ii) of Theorem III-2.2, the $D''(x,y_1,\ldots,y_v)$, which is obtained from $D_\emptyset(x)$ via the application of the multivariate EZ Algorithm (III-5.4) on F, $D_\emptyset$, and $F_b/D_\emptyset$, must satisfy $D = D''$ or

$$D'(x,y_1,\ldots,y_v) = D''(x,y_1,\ldots,y_v)\,gcd(f', g').$$

Proof: Apply the Multivariate EZ Algorithm on F, $D_\emptyset$, and $F_b/D_\emptyset$ to get D''. Since the evaluation mapping determined by b is a homomorphism, $D_b(x)$ divides $D_\emptyset(x)$ or $D_\emptyset = D_b\,C$. But (i) implies $deg(D_\emptyset) = deg(D_b)$ so that C must be a constant. But C divides $D_\emptyset$, in fact, C divides $lc(D_\emptyset)$, so C

must be a unit modulo the particular prime p of the EZ Algorithm. Thus we have $D \equiv D_0 \pmod{q_k, S}$ and so $F/D \equiv F_b/D_0 \pmod{q_k, S}$. Therefore, by Theorem II-5.4 the EZ Algorithm (II-5.4) yields D" which is primitive and divides F. By the result of Theorem III-2.2, D = D" Q. But F and G being primitive implies D must also be primitive, so that Q, being independent of x, must be 1. That is D = D" and D' = D" gcd(f', g'). //

III - 3 Univariate GCD Algorithm with Zassenhaus' Quadratic Constructions

Computing the GCD of univariate polynomials using the Hensel-type construction does not demonstrate the full power of the Hensel method. However, it does serve the purposes of showing that (1) in many cases it is not necessary to compute more than one homomorphic image of the GCD in order to "invert" the mapping and get the actual answer, (2) this method also avoids the coefficient growth problem of the PRS algorithms and at the same time retains the unique advantage of the Modular Algorithms for speedy detection when the GCD is actually 1, and (3) it provides a good conceptual basis for understanding the multivariate EZGCD Algorithm and its related problems, since the two cases are basically similar in many aspects.

Now that we have established some theoretical foundations for the utilization of the Hensel-type construction in GCD computations in the last section, we will directly present the univariate algorithm, UNIGCD, relying on the theoretical similarities of the multivariate and the univariate cases.

Algorithm III - 3.1: (Univariate GCD - UNIGCD)

Input: Univariate polynomials $F'(x)$ and $G'(x)$ in $Z[x]$ where $\deg(F')$ is assumed to be $\geq \deg(G')$. (Otherwise simply switch them and later also switch the order of the output cofactors.)

Output: $D'(x) = \gcd(F', G')$ in $Z[x]$, F'/D', and G'/D'.

(A1) Set $f' \leftarrow \text{cont}(F')$, $F \leftarrow \text{pp}(F')$, $g' \leftarrow \text{cont}(G')$, $G \leftarrow \text{pp}(G')$, and $d' \leftarrow \text{igcd}(f', g')$ where the "i" stands for integer operations.

(A2) Set $f_1 \leftarrow \text{lc}(F)$, $g_1 \leftarrow \text{lc}(G)$, and $d_1 = \text{igcd}(f_1, g_1)$.

Set $d_F \leftarrow \deg(F)$ and $d_G \leftarrow \deg(G)$.

Set p to a new prime such that p does not divide $f_1 g_1$.

Set $F_p \leftarrow F \pmod p$, $G_p \leftarrow G \pmod p$,

$D_p \leftarrow \text{mgcd}(F_p, G_p)$ where the "m" stands for modular operations, and $d \leftarrow \deg(D_p)$.

(A3) If $d = 0$ then $D_p = 1$ and return d', $(f'/d')\,F$, and

(g'/d') G. Otherwise, if (A4) has been passed through already, then skip to (A5).

(A4) Choose a new prime p' such that p' does not divide $f_1 g_1$ and compute $D_{p'} = \mathrm{mgcd}(F_{p'}, G_{p'})_{p'}$.

If $\deg(D_{p'}) < d$, then set $F_p \leftarrow F_{p'}$, $G_p \leftarrow G_{p'}$, $D_p \leftarrow D_{p'}$, $p \leftarrow p'$, $d \leftarrow \deg(D_{p'})$, and repeat (A4).

If $\deg(D_{p'}) = d$, then go to (A3).

If $\deg(D_{p'}) > d$, then repeat (A4).

(A5) If $d = d_G$, then test whether G divides F, if so return d' G, (f'/d') (F/G), and g'/d'.

If test-division fails then go to (A6).

If $d \neq d_G$, then continue to (A7).

(A6) Choose new primes p" until a new $D_{p"} = \mathrm{mgcd}(F_{p"}, G_{p"})_{p"}$ is found whose degree is less than d. Then set $F_p \leftarrow F_{p"}$, $G_p \leftarrow G_{p"}$, $D_p \leftarrow D_{p"}$, $p \leftarrow p"$, $d \leftarrow \deg(D_{p"})$, and go back to (A3).

(A7) If $\mathrm{mgcd}(D_p, G_p/D_p)_p = 1$, then set $U \leftarrow G$, $D_0 \leftarrow D_p$, $H_0 \leftarrow G_p/D_p$, and go to (A8). If $\mathrm{mgcd}(D_{p'}, F_{p'}/D_{p'})_{p'} = 1$,

then set $U = F$, $D_0 = D_p$, $H_0 = F_p/D_p$, and go to (A8).

Otherwise, go to the special case algorithm (S1).

(A8) Apply the Univariate EZ Algorithm (II-5.3) to $U(x)$ in $Z[x]$, and $D_0(x)$, $H_0(x)$ in $Z_p[x]$.

If the outputs are D and H such that $U = DH$ over Z then continue to (A9). Otherwise, the outputs are D_k, H_k, and q_k such that $U \equiv D_k H_k \pmod{q_k}$, then p must be unlucky so that a new prime p'' must be found, go to (A6).

(A9) If $U = G$, test whether D divides F, if so return $d'D$, $(f'/d')(F/D)$, and $(g'/d')H$. If $U = F$, test whether D divides G, if so return $d'D$, $(f'/d')H$, and $(g'/d')(G/D)$. Otherwise p must be unlucky, so go to (A6).

(S1) Set $G'' \leftarrow$ UNIGCD$(G, dG/dx)$ and $L \leftarrow G/G''$ (via Steps (A2) - (A9)).

(S2) Set $L \leftarrow$ UNIGCD(L, F) and $F'' \leftarrow F/L$ (via Steps (A2) - (A9)). Set $D'' \leftarrow L$ and $D \leftarrow L$.

(S3) Set $D_p \leftarrow$ mgcd(L_p, F_p'', G_p'') where p is the valid, lucky prime already used in UNIGCD Steps (A2) - (A9) of (S2).

(S4) If $\deg(D_p) = 0$, then $D_p = 1$, so return $d'D''$,

(f'/d') (F/D"), and (g'/d') (G/D").

(S5) If $L_p / D_p = 1$, then skip (S6) and go to (S7).

(S6) Apply Univariate EZ Algorithm to L in Z[x] and $D_0 = D_p$,

$H_0 = L_p / D_p$ in $Z_p[x]$ to get D_p and H_p in Z[x] such that

$L = D_p H_p$ over Z. Set L <-- D_p.

(S7) Set D" <-- D" D_p, F_p" <-- F"/D_p, G_p" <-- G"/D_p,

and go to (S3).

Now, we make some brief remarks and explanations on the individual steps of the algorithm especially those relevant to univariate case, leaving detailed discussions of points common to both cases to the multivariate algorithm later.

Remark (A4): This is the step where we attempt to establish Condition III-B, $\deg(D_0) = \deg(D)$. Since $D_0 = \gcd(F, G)$ is the unknown polynomial to be determined, we have no way of verifying that d = deg(D). In fact, by Lemma III-2.1, d could be > deg(D). However, the probability of the chosen prime p being unlucky is bounded by 1/p (see Theorem 3 of [BR071]). If we use fairly large primes and attempt as in (A4) to repeat d with another prime p' then we can greatly reduce the probability for p to be unlucky or greatly inhance the probability of having Condition III-B be true. Thus, in Step (A4) we try with another prime p'. If

p was unlucky so that d > deg(D), the probability of p' being unlucky and resulting in a $D_{p'}$ with degree equal to d is at least bounded by 1/(p p'). That means that, this way, it is very unlikely for us to use a unlucky prime p for our Hensel construction later.

Remark (A7): Here, we determine which, if one, of the given polynomials satisfy Condition III-A. Since it is a good strategy to apply Hensel constructions on smaller polynomials, we test the condition on G first, relying on the heuristic basis that polynomials of smaller degrees contain fewer terms. If Condition III-A fails on both polynomials, then we cannot directly apply the Hensel construction to this problem. However, our special case algorithm, which applies the Hensel construction also, will always apply for any two polynomials as we will soon see. Note that it is possible for Condition III-A to fail even though either gcd(D, F/D) = 1 or gcd(D, G/D) = 1, if the chosen prime is unlucky. Because of the fact that we have decreased the probability for p to be unlucky and that the special case algorithm works in general, we, therefore, have a complete GCD algorithm which uses the Hensel construction for all cases.

Remark (A8): By Theorem II-5.3 and when q_k is suffi-

ciently large to bound all possible integral coefficients, the Univariate EZ Algorithm returning only D_k and H_k such that $U \equiv D_k H_k \pmod{q_k}$ could only mean the lack of any divisor of U over the integers corresponding to this $D_0(x)$ or having the same degree as $D_0(x)$. Thus Lemma III-2.1 would imply that $\deg(D_0) > \deg(D)$ or p is unlucky.

Remark (S1): The basic underlying principle of the special case method is that if one of the two original polynomials is square-free, then it is always possible to find a prime such that Condition III-A holds. Viewing this another way, it is always possible, when given two polynomials, to compute the largest square-free part of their GCD first and then the largest square-free part of the GCD of the remaining cofactors successively. Thus we will always be able to compute essentially the square-free decomposition of the GCD of any two polynomials, hence the GCD itself.

The special case algorithm first computes the largest square-free part of G, then uses that with F to get the largest square-free part of the GCD of F and G. We leave many detailed discussions on the individual steps of this special case algorithm until later when we present the multivariate case and the Hensel-type square-free

decomposition algorithm (Chapter VI).

Remark (S2): The reason for only needing Steps (A2) - (A9) for UNIGCD(G, dG/dx) and for UNIGCD(L, F) should be quite clear. However, proofs for them will be presented in Chapter VII.

As compared to the modular GCD algorithm for univariate polynomials, the essential difference for the UNIGCD algorithm is the step of inverting the modular homomorphism. Except for the higher overhead of the Zassenhaus' Quadratic Extension Algorithm, i.e. the cost of applying the Extended Euclidean Algorithm (II-2.1) on the two modular codivisors, the remaining operations are mainly integer arithmetic only. The Chinese Remainder Algorithm also only involves arithmetic operations on each pair of corresponding coefficients, except each of these require one application of the integral extended Euclidean algorithm. In general, if p is an average-sized prime that we use in either algorithm (we assume that the primes used all have approximately the same integer length), and B is the bound on all integral coefficients for a given problem, then the major computational gain of using the Zassenhaus' algorithm is due to the fact that it requires only k steps where $k = \log_2(\log_p(B))$, whereas the modular algorithm will need to apply the Chinese Remainder Algorithm on corresponding

coefficients for k' steps where k' = log (B). Thus the
p

method using the Zassenhaus' algorithm can show some computational efficiencies when B is quite large or p is small so that k' = log (B), which is the number of steps the Modular
p

GCD Algorithm has to go through, is large compared to k (= log (k')). However, it is very costly for the
2

Zassenhaus' Quadratic Extension Algorithm to have to maintain relative primeness of the two codivisors being constructed (i.e. getting A_j and B_j such that

$A_j G_j + B_j H_j \equiv 1 \pmod{q_j}$; see Lemma II-3.2). In this respect, it seems that the ordinary Hensel method (Algorithm II-3.1) will prove to be more efficient when the number of steps, k', for this algorithm is relatively small (e.g. k' < 16 or k < 4). Also the computations of successive residual terms in both algorithms (the C_j's), require cross multiplications of the current factors which can be very costly, especially when the coefficients are large. These complex trade-offs in computational efficiencies of the three modular methods for computing univariate GCD's are currently under active study and further results will be reported in a future paper by Yun and Miola.

III - 4 The Multivariate GCD Algorithm - EZGCD

With Section III-2 as theoretical basis and the last section as brief introduction, we now present the detailed EZGCD Algorithm with remarks on some of the steps to follow afterwards.

Algorithm III-4.1: (Multivariate GCD - EZGCD)

Input: Multivariate polynomials $F'(x,y_1,\ldots,y_v)$ and $G'(x,y_1,\ldots,y_v)$ in $Z[x,y_1,\ldots,y_v]$ where $\deg(F')$ in x is assumed to be $\geq \deg(G')$ (Otherwise, simply switch the input polynomials and later also switch the order of the output cofactors.)

Output: $D'(x,y_1,\ldots,y_v) = \gcd(F', G')$ in $Z[x,y_1,\ldots,y_v]$, F'/D', and G'/D'.

(A1) Set f' <-- cont(F'), F <-- pp(F'), g' <-- cont(G'), G <-- pp(G'), and d' <-- gcd(f', g').

(A2) Set d_F <-- deg(F(x)) and d_G <-- deg(G(x)).

If $b = [b_1=0, b_2=0, \ldots, b_v=0]$ is a valid evaluation (i.e. $(lc(F))_b$ and $(lc(G))_b \neq 0$), then compute $F_b(x)$, $G_b(x)$, and $D_b(x) = \gcd(F_b, G_b)$. Otherwise, find a new valid evaluation b which contains as many zeros as

possible and compute $D_b(x) = \gcd(F_b(x), G_b(x))$. Set

$d \leftarrow \deg(D_b(x))$.

(A3) If $d = 0$, then return d', $(f'/d')\,F$, and $(g'/d')\,G$.

Otherwise, if (A4) has been passed through already,

then skip to (A5).

(A4) Choose a new valid evaluation b' and compute

$D_{b'}(x) = \gcd(F_{b'}(x), G_{b'}(x))$.

If $\deg(D_{b'}) < d$, then set $F_b \leftarrow F_{b'}$, $G_b \leftarrow G_{b'}$,

$D_b \leftarrow D_{b'}$, $P \leftarrow P'$, $d \leftarrow \deg(D_{b'})$, and repeat (A4).

If $\deg(D_{p'}) = d$, then go to (A3).

If $\deg(D_{p'}) > d$, then repeat (A4).

(A5) If $d = d_G$, then test whether G divides F, if so return

$d'\,G$, $(f'/d')\,(F/G)$, and g'/d'.

If test-division fails, then go to (A6).

If $d \neq d_G$, then continue to (A7).

(A6) Choose a new valid evaluation b'' until the correspond-

ing $D_{b''}(x) = \gcd(F_{b''}(x), G_{b''}(x))$ has degree less than

d. Then set $F_b \leftarrow F_{b''}$, $G_b \leftarrow G_{b''}$, $D_b \leftarrow D_{b''}$,

$b \leftarrow b''$, $d \leftarrow \deg(D_{b''})$, and go to (A3).

(A7) If $\gcd(D_b, G_b/D_b) = 1$, then set $U \leftarrow G$, $D_1 \leftarrow D_b$,

$H_1 \leftarrow G_b/D_b$, and go to (A8).

If $\gcd(D_b, F_b/D_b) = 1$, then set $U \leftarrow F$, $D_1 \leftarrow D_b$,

$H_1 \leftarrow F_b/D_b$, and go to (A8).

Otherwise, go to the special case algorithm (S1).

(A8) Apply the Multivariate EZ Algorithm II-5.4 to U in

$(Z[y_1,\ldots,y_v])[x]$ and $D_1(x)$, $H_1(x)$ in $(Z/q_k)[x]$ where

$q_k = p^{**}(2^{**}k)$ for some lucky prime p for U_b.

If the outputs are D and H such that $U = D\,H$ over Z,

then continue to (A9). Otherwise, the outputs are D_n,

H_n, q_k, and n such that $U \equiv D_n H_n \pmod{q_k, S_n}$, then

b must be an unlucky evaluation so that a new evaluation must be found, go to (A6).

(A9) If $U = G$, test whether D divides F, if so return d' D, $(f'/d')(F/D)$, and (g'/d') H.

If $U = F$, test whether D divides G, if so return d' D, (f'/d') H, and $(g'/d')(G/D)$.

Otherwise, b must be an unlucky evaluation, so go to (A6).

(S1) Set $G'' \leftarrow EZGCD(G, dG/dx)$ and $L \leftarrow G/G''$ (via Steps

(A2) - (A9) above).

(S2) Set L <-- EZGCD(L, F) and F" <-- F/L (via Steps (A2) - (A9) above). Set D" <-- L and D <-- L.

(S3) Set $D_b(x)$ <-- gcd($L_b(x)$, $F"_b(x)$, $G"_b(x)$) where b is the valid, lucky evaluation already used in EZGCD Steps (A2) - (A9) of (S2).

(S4) If deg($D_b(x)$) = 0, then $D_b(x) = 1$, so return d' D", (f'/d') (F/D"), and (g'/d') (G/D").

(S5) If $L_b / D_b = 1$, then skip (S6) and go to (S7).

(S6) Apply Multivariate EZ Algorithm to L in $Z[x,y_1,\ldots,y_v]$, and $D_1 = D_b$, $H_1 = L_b / D_b$ in $(Z/q_k)[x]$, where $q_k = p^{**}(2^{**}k)$ for some lucky prime p for L_b, to get D and H in $Z[x,y_1,\ldots,y_v]$ such that L = D H over Z. Set L <-- D.

(S7) Set D" <-- D" D, $F"_p$ <-- $F"_p/D_p$, $G"_p$ <-- $G"_p/D_p$, and go to (S3).

Remark (A1): Contents and primitive parts of any multivariate polynomials are computed recursively on the variables. Here, the contents are taken w.r.t. the main variable x and are polynomials in $Z[y_1,\ldots,y_v]$. That is we consider

the given polynomial to be in $(Z[y_1,\ldots,y_v])[x]$. The computation involves GCD's of all coefficients with respect to a particular main variable (here x). This is accomplished by using a variant of this algorithm which computes the GCD of all coefficients in a semiparallel fashion which will be discussed in more detail in Chapter VI.

Remark (A2): In Section II-4 and II-6 we have briefly mentioned the potential seriousness of the "non-zero substitution" problem. Section II-6 points out that the technique we use for putting polynomials into the generalized Taylor series forms may cause blowups in the number of terms resulting in computational inefficiencies for certain bad cases. We see from the way this transformation is carried out (Sec. II-4), it is desirable to have as many zeros in the set, b, of the integral values for evaluation as possible so that the number of terms in the expanded form of the series is not increased unnecessarily. Due to the requirement for valid and lucky evaluations, it is not always possible to choose zeros for evaluation. But more zeros in the set of values used for the Generalized Hensel construction certainly improves its efficiency because of the dependency of the computing cost for the GHA on the number of terms in the Taylor series form of the given polynomial as we shall see in the next chapter. Since it is not possi-

ble to check whether an evaluation is lucky or not at this point, we only make sure the set of values for evaluation contain as many zeros as possible and still remain valid.

Remark (A3): Since $\deg(\gcd(F, G)) \leq \deg(D_b(x))$, $d = 0$ implies $\gcd(F, G)$ is independent of x. But F and G are primitive so that $\gcd(F, G)$ must simply be 1. Thus, $\gcd(F', G')$ is just d' and the cofactors are $F'/d' = (f'/d')\,F$ and $G'/d' = (g'/d')\,G$ respectively.

Remark (A4): In this step, we attempt to establish Condition III-B, $\deg(D_0) = \deg(D)$. Since $D = \gcd(F, G)$ is the unknown polynomial to be determined, we have no way of verifying that $d = \deg(D)$ at this point. In fact, Lemma III-2.1 indicates that d could be $> \deg(D)$. However, the total number of possible unlucky integral values for evaluation is bounded by $(v\ m\ \deg(F))$ where

$m = \max_{i=1}^{v}(\deg_{y_i}(F) + \deg_{y_i}(G))$ (see Theorem 5 of [BRO71]). So even if we limit our possible choice of integral values to only those in Z/p for some prime p, the probability for any evaluation $b = (b_1, \ldots, b_v)$ to be unlucky is bounded by

$v^2\ m \deg(F)/p$. If the given GCD problem is of reasonable size so that $v^2\ m \deg(F)$ is not too large and the prime is

fairly large, then the task of (A4), which tries to repeat the same degree d with another evaluation b', will greatly reduce the probability for b to be unlucky or inhence the probability of having Condition II-B be true. Because of the fact that $\deg(D) \leq \deg(D_b = \gcd(F_b, G_b))$ for any given valid evaluation b, note that if we make several evaluations of F and G at different sets of values, then the one resulting in the minimal degree for D_b should be used. In fact, if the number of trial evaluations are larger than $v^2 m \deg(F)$ then we are sure to find one evaluation among them which has degree equal to deg(D). In Step (A4) we try to find a valid evaluation b which yields a $D_b(x)$ having the degree that is repeated by some other valid evaluation. Thus if the first evaluation is unlucky and results in a high degree, the chances of repeating this wrong degree are much smaller. This step virtually insures the success of getting a lucky evaluation. Together with Step (A2), we can be quite certain that the evaluation b we use in this algorithm, hence for the Hensel construction, is valid, lucky, and contains many zeros.

Remark (A5): Since $\deg(G(x)) \leq \deg(F(x))$ and F, G are primitive, when $d = d_G$ after Step (A4) it is very likely

that G divides F. If this test-division fails we still get the valuable information that d should be smaller than it is now. Therefore, this is a strategic, time saving step in any case.

Remark (A6): This step is used at several other places of the algorithm, whenever we can get the information about d being too large or the corresponding evaluation b is unlucky. In these cases we know there is an evaluation, say b", which can result in an univariate GCD of lower degree than d. So we keep on choosing new valid evaluations until such a b" is found.

Remark (A7): Here we determine which, if one of the given polynomials satisfy Condition III-A so that the Hensel construction can be applied. Since it is a good strategy to apply Hensel constructions on polynomials with fewer terms, we test the condition on G first, relying on the heuristic basis that polynomials of smaller degrees contain fewer terms. If Condition III-A fails on both polynomials, then we cannot directly apply the Hensel construction to this problem. However, our special case algorithm, which also applies the Hensel construction will always apply for any two polynomials. Note that if the chosen evaluation is unlucky it is possible for Condition III-A to fail even though either gcd(D, F/D) = 1 or gcd(D, G/D) = 1. But Step

(A4) has reduced the probability for using an unlucky evaluation and the special case algorithm can handle any polynomial inputs, so we have a complete GCD algorithm which applies the Hensel construction for all cases.

Remark (A8): Note that the application of the Multivariate EZ Algorithm (II-5.4) requires a lucky prime p for U_b - because it is necessary to compute A(x) and B(x) such that $A\,D_1 + B\,H_1 \equiv 1$ in $(Z/q_k)[x]$ by using the Extended Euclidean Algorithm (II-2.1). This lucky prime can be found in several ways. One such method is simply using the lucky prime from the GCD computation in Step (A7), assuming of course the univariate GCD computations are performed by either the Modular GCD Algorithm or the UNIGCD Algorithm which requires a lucky prime for its computations. In fact, such a lucky prime can also be provided by any other step of this algorithm (e.g. (A2)) where univariate GCD's are computed, so long as the univariate GCD algorithm used is the modular kind. By Theorem II-5.4 and since p is lucky, q_k and n are sufficiently large, the case where the Multivariate EZ Algorithm returns only D_n and H_n such that $U \equiv D_n H_n \pmod{q_k, S^n}$ can only mean the non-existence of

any divisor of U corresponding to this $D_1(x)$. Lemma III-2.1 would then imply that $\deg(D_1(x)) > \deg(D)$ or that b is an unlucky evaluation.

Remark (A9): If U = G, then G = D H over Z. Since $\deg(\gcd(F, G)) \leq \deg(\gcd(F_b, G_b))$ and primitive polynomials over Z are unique up to signs, D must be gcd(F, G) if D also divides F. If D does not divide F, then D must contain some factor of G which is not a factor of F. That is the degree of D must be too big or this evaluation b is unlucky, so we must re-evaluate by going back to Step (A6). This is a "safety-valve" step which catches all hitherto undetected unlucky evaluations even though the probability of such has been greatly reduced by Step (A4).

Remark (S1): Similar to Remark (S1) of the UNIGCD Algorithm, we point out again that the basic underlying principle of the special case method (S1) - (S7) is that if one of the two original polynomials is square-free, then it is always possible to find an evaluation such that Condition III-A holds. That is, for two arbitrary multivariate polynomials it is always possible to compute (using the Hensel construction) the largest square-free part of their GCD first, and then the largest square-free part of the GCD of the remaining cofactors, successively. Thus, we will

always be able to compute essentially the square-free decomposition of the GCD of any two polynomials using Hensel construction, hence the GCD itself.

The special case algorithm first computes the largest square-free part of G, then uses that with F to get the largest square-free part of the GCD of F and G. This step computes the largest square-free part of G, by computing the cofactor of G with respect to gcd(G, dG/dx). This GCD can always be computed by going through Steps (A2) - (A9) only. The reason for this is made clear by the following lemma:

Lemma (S1): Let P be primitive in $(Z[y_1,\ldots,y_v])[x]$ and R = gcd(P, dP/dx), then gcd(R, (dP/dx)/R) = 1.

Note that this virtually insures that of Condition III-A holds, so that there will be no need for special case consideration for gcd(G, dG/dx).

Here we point out that this special case algorithm (S1) - (S7) with the above lemma and some other conditions constitutes the essential ideas for an algorithm of finding the square-free decomposition of a multivariate polynomial using the Hensel construction. The clarification of this statement together with the proof of the above lemma will be deferred to Chapter VI on square-free decompositions of polynomials.

Remark (S2): Since L is square-free. GCD of L and any

other polynomial can be computed by Step (A2) - (A9) of this EZGCD Algorithm. That is, when L is square-free, it is always possible to find a valid lucky evaluation for L such that L evaluated is still square-free (see Chapter VI or [W&R73] for a proof). D" will be used to denote the accumulative GCD up to this point, D the current largest square-free part GCD, and F", G" the cofactors of D" in F, G respectively.

Remark (S3): If b is a valid, lucky evaluation for L and F in (S2) then it must be lucky for both F and G or for any square-free part of their GCD. Therefore, from this point on, we will always use this valid, lucky evaluation for our succeeding computations. What we intend to compute now is gcd(L, F", G"). But since b is a lucky and valid evaluation for this problem, all that needs to be computed at this point is the univariate GCD of the three polynomials evaluated. We can then use this univariate GCD for the Hensel construction with L to get the actual mutivariate gcd(L, F", G"). Here the GCD is a univariate GCD taken over the integers. GCD of several polynomials can be taken recursively by successive pairs. However, as we shall see in the next chapter, it is also possible to semi-parallelly compute the GCD of several polynomials and their corresponding cofactors together for either the multivariate or the univariate cases using Hensel constructions.

Remark (S5): If $L_b / D_b = 1$, then we know that the largest square-free part of the remaining cofactors of D" in F and G (i.e. F" and G") is the same as the current largest square-free part, L, or D = L. It is then not necessary to apply the Multivariate EZ Algorithm and simply use the current largest square-free part of the GCD, D, to update the accumulative GCD, D", in (S7). Thus whenever $L_b / D_b = 1$ we can save one application of the EZ Algorithm. For example, if $F = U^3 V^2$ and $G = U^2 V^3$, then $L = D = U^2 V^2$ in (S2), $F" = U^2 V$ and $G" = U V^2$, so that gcd(L, F", G") = U V again and in (S3) $D_b = U_b V_b = L_b$ resulting in $L_b / D_b = 1$.

Remark (S6): Here we apply the Multivariate EZ Algorithm on L, D_1, and H_1 to get D = gcd(L, F", G"), as we set out to do in Step (S3). The lucky prime p for L_b can be the one used in (S3) for the computation of univariate GCD, D_b. With these guaranteed lucky prime p, a valid lucky evaluation b, and a square-free L, all conditions of Theorem II-5.4 (Multivariate EZ) are satisfied so that the EZ Algorithm produces the correct multivariate polynomials D and H such that L = D H over Z. This D must be the GCD of L, F",

and G", or the largest square-free GCD of F" and G". This largest square-free GCD of F" and G", D, must be smaller than the previous one, L, and we can take advantage of it by using this D (setting L <-- D) for computation of later largest square-free GCD's for the new cofactors F" and G" corresponding to F" and G" of the next step.
b b

Remark (S7): Here we update the accumulative GCD to include the D just found. We then also update the cofactors to correspond to this D", in order to begin the computation of the next largest square-free GCD. But we have already observed that in (S3) we only need the univariate (or evaluated) cofactors, F" and G", so we only need to update
b b

these univariate polynomials instead of the corresponding multivariate polynomials. Therefore, we see that after Steps (S1) and (S2), the only multivariate polynomials used in the remaining computations are the largest square-free GCD's, L, in the application of the EZ algorithm. The final answer gcd(F, G) is simply D" which is the product of the results of the EZ Algorithm in (S6). This fact will account for the comparable efficiency of this special case algorithm with the regular algorithm ((A1) - (A9)) and the Modular GCD Algorithm.

III - 4(a) Solution to the Leading Coefficient Problem

Because of the non-uniqueness of polynomials in image domains, all algorithms based on modular techniques suffer by having to perform some extra process in order to determine a unique answer from its class of associates. This is often accomplished by recursively requiring a unique leading coefficient. Since the GCD algorithm was one of the earliest considered from this point of view, the leading coefficient problem naturally came up. In his now classic paper on the Modular GCD Algorithm [BRO71], Brown clearly recognized this problem. His implicitly proposed solution (only given in the algorithmic specifications for Algorithm M and P) uses the GCD of the leading coefficients of the two given polynomials. This polynomial is clearly divisible by the leading coefficient of the true GCD of the two given polynomials. The Modular GCD Algorithm then artificially imposes this onto a monicized GCD so as to have a unique polynomial which is a simple multiple of the actual GCD. In fact, the GCD is the primitive part of this polynomial.

This approach still results in the blowup problem, which we discussed in Section II-5(a). Although our solution to this leading coefficient blowup (LCB) problem is in essence similar to Brown's approach, some important differences as well as the significance of this blowup problems alone make this separate treatment worthwhile. Our

approach achieves the same goal in the end and also uses the GCD of the leading coefficients as does the Modular Algorithm, but the means for avoiding the blowup are different in many ways. Most improtantly, our method preserved the sparseness of the problem during computation at a small cost (of computing the GCD of the leading coefficients and multiplying it to the polynomials).

In Section II-5(a) we discussed the causes and the potential seriousness of the LCB problem. We also presented a method (Theorem and Algorithm II-5.5) of avoiding this problem and an example demonstrating its feasibility. But it is not always possible to know in advance the leading coefficients of the co-divisors that the Hensel construction is attempting to find. So the method proposed by Theorem II-5.5 must have appeared extremely artificial and impractical. In the case of the EZGCD Algorithm, however, there is actually much to be gained.

Let F and G be the primitive parts of the input polynomials for the EZGCD Algorithm and D be gcd(F, G) to be computed. Consider F, G, D to be in $(Z[y_1,\ldots,y_v])[x]$ and let $f_1 = lc(F)$, $g_1 = lc(G)$ and $d_1 = lc(D)$. These are polynomials in $Z[y_1,\ldots,y_v]$ independent of x. We can compute $d_1'' = gcd(f_1, g_1)$ for later use. Note that this is a GCD

computation of one less variable, which can hopefully be done more easily. In fact, this computation is, in most cases, less costly than the calculation of contents and primitive parts of the input polynomials F' and G' to the EZGCD Algorithm. Before Step IV of the EZGCD outline of Section III-2 (or before the application of the Multivariate EZ Algorithm in Step (A8) of the EZGCD Algorithm, assuming F satisfies the test in (A7)), we have F, $D_0(x)$, $H_0(x) = F(x)/D_0(x)$ such that $F \equiv D_0 H_0 \pmod{q, S}$ where $D_0(x) = \gcd(F(x), G(x))$. At this point we let $D_1(x) = dl''(D_0/lc(D_0))$ and $H_1(x) = lc(D_0) H_0 \pmod{q}$, then the following are all true:

(i) $lc(D_1(x)) = dl''$ and dl'' is a known polynomial.

(ii) $lc(H_1(x)) = lc(D_0)\, lc(H_0) = lc(F_b)\, lc(G_b) = fl$ and fl is a given, hence known polynomial.

(iii) $dl'' F \equiv dl'' F_b = D_1 H_1 \pmod{q, S}$.

Thus by theorem II-5.5, Algorithm II-5.5 can be used instead of the Multivariate EZ Algorithm in the EZGCD Algorithm in order to alleviate the LCB problem. If the prime and the evaluation values used are lucky, then $D_0(x) = D(x, b_1, \ldots, b_v)$ and the outputs obtained will be

D' = dl"(D/dl) and H' = dl H such that dl" F = D' H' over Z. Since dl divides dl", cont(D') = dl"/dl or dl = dl"/cont(D'), so that D = D'/cont(D') = pp(D') and H = H' cont(D')/dl". In this case, there is no blowup problem due to multiplication by some unit u in S_n or 1/u as pointed out in Section II-5(a). Therefore, the leading coefficient problem for the Hensel construction in EZGCD Algorithm is avoided at the relatively small cost of computing dl" = gcd(fl, gl) and using dl" F for the EZ Algorithm instead of F. We modify the EZGCD Algorithm by changing Step (A8) as follows:

(A8') Compute dl" = gcd(lc(F), lc(G)).

Set U" <-- dl" U, D_1 <-- dl" (D_1 /lc(D_1)), and

H_1 <-- lc(D_1) H_1.

If U = F, then apply Algorithm II-5.5 on U", D_1, H_1, dl, and lc(F), otherwise U = G, then apply Algorithm II-5.5 on U", D_1, H_1, dl, and lc(G), to replace the use of the Multivariate EZ Algorithm in (A8). If the results are D" and H" such that U" = D" H" over Z, then clearly U = pp(U") = pp(D") pp(H") = D H with D = pp(D") and H = pp(H"), then continue to A(9). Otherwise, as in (A8), b must be an unlucky evalua-

tion, so go to (A6).

Example III-4.1: In the rest of this section we will compute the multivariate GCD of two given non-monic polynomials using the EZGCD Algorithm. Hopefully this example will demonstrate many essential features of this algorithm, especially the way the leading coefficients are handled (as in (A8')). To avoid being too lengthy, we will go through the steps of the algorithm outline in Section III-2 instead of following the detailed steps of the EZGCD Algorithm. However, the modified version of the multivariate EZ Algorithm (II-5.5, as in (A8')) rather than the regular version (Algorithm II-5.4, as in (A8)) will be used. Also, the special case method is not needed for this example.

Assume the problem is to compute the GCD, D', of two given multivariate polynomials F' and G' where

$$F' = y^2 x^2 + (-y^2 z - y^2 z + y z + y^2 + 2y) x + y z^3 - z^3 - y^2 z - y z + 2 z \text{ and}$$

$$G' = y^3 x^3 + (-y^2 z - y^3 z + y^2 z) x + (y^3 z - z^3 - y^2 + 2 y) x + y^2 z - 3 y z + 2 z.$$

It is easily realized that F' and G' are primitive, so that d' = f' = g' = 1, F = F', G = G', and D' = D = gcd(F, G). By choosing b = (1, 0) which is clearly a valid evaluation

for F and G, we have $F_b(x) = x^2 + 3x$, $G_b(x) = x^3 + x$, and

$D_0(x) = \gcd(F_b, G_b) = x$. Since $\deg(D_0(x)) = 1 \neq \deg(F)$ or

deg(G), we test Condition III-A and find

$\gcd(D_0, F_b/D_0) = \gcd(x, x+3) = 1$. So we continue to Step IV

assuming Condition III-B, $\deg(D_0) = 1 = \deg(D)$. Now the

Hensel construction needs to be applied on F, D_0, and F_b/D_0,

but the multivariate leading coefficients of D_0 and F_b/D_0

are not known. So according to (A8'), we compute

$dl'' = \gcd(lc(F), lc(G)) = y^2$. Then we set

$U \leftarrow dl'' F$, $D_1 \leftarrow dl'' (D_B/lc(D_0)) = x$ and

$H_1 \leftarrow lc(D_0) F_B/D_0 = x + 2$, but now we know that the mul-

tivariate leading coefficients of D_1 and H_1 are both equal

to y^2. So we can apply the modified multivariate EZ Algo-

rithm (II-5.5) to U, D_1, H_1, with y^2 as leading coeffi-

cients. This problem is exactly the same as Example II-5.5

and the results of that are $D''(=G') = y^2 x + y^2 z - y^4 z$ and

$H''(=H') = y\ x + y\ - y\ z\ + 2\ y$ such that $U = D''\ H''$ over Z. Thus we avoided the LCB problem as pointed out by Example II-5.5 at the cost of computing $dl'' = gcd(y^2, y^3) = y^2$. Now we note that $F = pp(U) = pp(D'')\ pp(H'') = D\ H$ over Z, so that $D = y\ x + z - y\ z$ and $H = y\ x + y - z^2 + 2$. Since D also divides G ($G = D\ (y^2\ x^2\ - z^2\ x - y + 2)$ over Z), we conclude that $D' = D = gcd(F, G) = gcd(F', G') = y\ x + z - y\ z$.

III - 5 Conclusions

This chapter has demonstrated the applicability of the Hensel construction to the computation of polynomial greatest common divisors, especially in the multivariate case. A complete GCD algorithm was presented which uses the Hensel construction for all situations. UNIGCD Algorithm showed the feasibility of using Zassenhaus' Quadratic Extension Algorithm for calculating univariate GCD's, hence avoiding the computation of many modular images of a GCD over the integers. By taking advantage of the sparseness usually exhibited by multivariate polynomials, the EZGCD Algorithm achieves greater computational efficiency at the very small cost of having the given polynomials satisfy a quite minor condition (Condition III-A). Careful and detailed design and implementation of this algorithm

revealed the possibility of using the Hensel construction for other fundamental polynomial operations. These include the semi-parallel computation of contents and primitive parts, the square-free decompositions of polynomials, and a different view of the pseudo-division of polynomials via the solution of polynomial equations.

CHAPTER IV COMPUTING TIME ANALYSIS AND EMPIRICAL RESULTS

IV - 1 Introduction

The practical value of an algorithm is measured by its "efficiency" and the most natural quantitative measure of efficiency is the computing time of the algorithm for a particular problem. But computing times do not always reveal the whole truth because of the many varying factors that have an influence on their values, such as the computer hardware, the supporting software, and the algorithm implementation and implementer. Thus, in order to evaluate an algorithm, it is important to perform some theoretical analysis of the algorithm independent of the implementation. In analyzing an algorithm one should strive to correlate predicted performance with the actual computing time, so that one can attempt to verify the validity of the analysis and the correctness of the implementation simultaneously.

Methods of analysis which attempt to achieve these goals have been developed and popularized in the recent years by Knuth [KNU69] and (particularly for algebraic algorithms) by Collins [COL69], [COL71], and Brown [BRO71]. We will briefly discuss some of these methods and point out some of their weaknesses. Then a somewhat different and more intuitive new method (developed with some essential innovations of Moses) will be proposed for analyzing the

just described Hensel construction and the EZGCD Algorithm. Finally, in this chapter, we will show how well the analysis is substantiated by some empirical test data. A similar philosophy regarding computing time analysis was independently arrived at by Gentleman [GEN73]. Fateman [FAT73] also presents a similar point of view and criticizes the worst-case analysis of algorithms for asymptotically large problems as essentially misleading. He analyzes several algorithms for making one particular polynomial computation by comparing actual timing and counts of basic operations (additions, multiplications, etc.) to the total run-times of the implementations for these algorithms.

Although both time and space requirements of an algorithm are important, it is often much easier to concentrate on one aspect rather than the "two dimensional" problem. We shall concentrate mainly on the analysis of the computing time of algorithms, and simply neglect space requirements (or assume that the available storage space is infinite).

One notation used in the analysis of algorithms is the "big-oh" notation [KNU69], which has the following meaning: the number $x(n)$ represented by $O(f(n))$ (big-oh of some function of n), where $f(n)$ is a function of the positive integer n, satisfies the condition $|x(n)| \leq M\,|f(n)|$ for

some positive constant M. Unfortunately this O-notation is not very suitable for the analysis of algebraic algorithms. So Collins and Brown adopted the more general notion of dominance. Let f and g be real-valued functions defined on some set S. Then f is said to be dominated by g or g bounds f, if there is a positive real number c such that $f(x) \leq c\, g(x)$ for all x in S. If f is also a bound for g (f bounds g), then f and g are said to be codominant. Note the similarity to the O-notation but the generality of S, or x in S, makes it possible for f and g to be functions of several variables. It is clear that codominance is an equivalence relation. Musser [MUS71, Theorem 0] lists some properties of the dominance relation which are straightforward consequences of the definitions. This notion of a bound is useful and convenient for many problems but it also has some drawbacks. One defect is that the real constant c (which is ordinarily unknown) is sometimes necessarily so large that the bound is good only for a small portion of the x's in S and is unrealistic for all other elements of S. Also, since the definition for dominance encompasses all elements x of S, a bound for a given function often turns out to be an asymptotic or "worst case" bound and provides little information about the actual behavior of the function for the "average" cases. These phenomena have been mentioned already in Section I-2 where we quoted Brown's

computing time formulas for the Reduced PRS and the Modular GCD Algorithms. Nevertheless, for many simpler algorithms, this method of analysis still provides valuable intuition and we will utilize it whenever it is appropriate for our computing time analyses.

Before we proceed, we will now formalize the notions of dense and sparse polynomials. A v-variable polynomial having maximum degree d_i in the ith variable will be considered dense if it contains all the possible monomial terms (products of powers of variables) with ith degree $\leq d_i$. Thus, a dense univariate polynomial in x of degree d will have all powers of x -- $1, x, \ldots, x^d$, so that it has d+1 terms. A dense v variable polynomial with maximum degree d_i, then, has $\prod_{i=1}^{v} (d_i + 1)$ terms. Note that if $d_i = 4$ for all i and v = 6, the polynomial will have over 15,000 terms. A present-day symbolic computing facility can hardly perform any non-trivial algebraic operations on such a polynomial. This is an important objection to asymptotic analyses of some algebraic algorithms and the reason for the more intuitive analysis for practically sized problems.

Any polynomial which has a significantly smaller number of terms than that of the dense polynomial having the same degree in each of the variables is considered sparse. According to this description of sparseness, the dividing line between the classes of sparse and dense polynomials is certainly not distinct. It is often left to personal judgment to determine what is a reasonable percentage of non-zero terms for which a polynomial will be considered sparse.

These are what we prefer to call the "autonomous" definitions of denseness and sparseness in that they can be determined by knowing the polynomial itself. Another "relative" definition, originally used in relation to polynomial multiplications, is more preferable in analyzing in the Hensel-type algorithms. The two different points of view are actually very closely related as we will soon illustrate.

Two polynomials are considered completely sparse if, when we form their product, there is no combination of terms by addition at all. Thus if one polynomial has m terms and the other n terms, then their product will have mn terms. Two polynomials of m and n terms respectively will be considered dense if their product has m+n-1 terms. As pointed out by Johnson and Gentleman [J&G73], this can only be attained when the polynomials are univariate with no non-

zero coefficients. For autonomously dense multivariate polynomials of degree d in each of the v variables, the product has (2 d + 1)**v terms but the computation of the product by this usual algorithm involves (d + 1)**(2 v) term-by-term multiplications. Thus the ratio of non-cancelled terms to the possible terms is approximately (2/(d + 1))**v. So we extend the definition of relative denseness to equal-degreed (having the same maximum degree d in each variable) multivariate polynomials of m and n terms respectively which yield a product of k terms such that $k/(m\ n) \leq (2/(d + 1))^{**}v$.

Among the basic arithmetic operations for polynomials, multiplication turns out to be the most important and relevant, since addition is much less costly for our polynomial representation. The classical method for performing the multiplication of two polynomials P and Q to get their product R (= P Q) involves term-by-term multiplications. So, if T(P) denotes the number of terms in a polynomial P, then the cost of the polynomial multiplication, R <-- P Q, is at least T(P)T(Q). In fact, this is the often accepted computing cost for multiplication (e.g. by Brown [BR071], Johnson, and Gentleman [J&G73]), especially for the extreme cases where the polynomials involved are completely sparse or dense. For the completely sparse case, T(P) T(Q) = T(R) which is the actual computing time often

observed for sparse polynomials. Once these terms are generated, it is necessary to sort them into some particular canonically simplified representations. Thus, including the sorting, polynomial multiplication is bounded by T(P)T(Q)log(T(P)T(Q)). But that assumes no ordering on the terms in P or Q. When the ordering of, say, P is taken into account, the bound for multiplication of sparse polynomials can be reduced to T(P)T(Q)log(T(P)) (as ALTRAN has been able to accomplish). It, then, seems reasonable to expect the computing time of a multiplication algorithm to be proportional to T(P)T(Q) by taking advantages of the orderings from both input polynomials (see [FR000] for relevant discussions). In fact, such an improvement can be accomplished for the dense case. If the product is dense the classical method is not the best way for this computation. The reason is that terms which combine do not have to be computed separately. The technique of using the fast finite Fourier transform to do multiplication turns out to achieve the (presently) minimal computing time bound of T(R)log(T(R)) [BON73]. We see that when the polynomials involved are all dense, it is possible to use the above lower bound which in general is already bounded by T(P)T(Q) (since (2d+1)**v*log(2d+1)**v) is in general less than (d+1)**(2 v) for v > 1). An even stronger argument can be made for this when the algorithm being analyzed performs

polynomial multiplications only as a part of all the necessary polynomial operations. We agree completely with those who argue that the cost of multiplication is at least T(P)T(Q) when an algorithm is used only for multiplication or is dominated by some multiplication process. Thus it seems reasonable to use T(P)T(Q) as the estimate for the cost of multiplication for all cases, except possibly when the product is dense where we can use T(R)log(T(R)) as the bound. As already noted in Chapter I, multivariate polynomials of high degree are inevitably sparse in practical situations, in fact, exceedingly sparse, so that the T(P)T(Q) computing cost will be a fair measure for a wide range of practical situations. The correctness of using this formula for multiplication, however, will be confirmed by many empirical computing results in Section IV-3.

IV - 2 Analysis of Computing Costs

In this section, we will attempt to give some reasonable estimates of computing costs for the algorithms discussed in the previous two chapters. Many of the asymptotic computing times for common arithmetic operations, especially for dense univariate polynomials, will be based on Brown's analysis [BR071]. We will often assume univariate polynomials to be dense and use the asymptotic

computing times for many operations on them. But we will also discuss computing times for average sized, non-dense multivariate polynomials since dense polynomials with many variables of high degree seldom occur in practical computations.

The computing time for the Extended Euclidean Algorithm (Algorithm II-2.1) in (Z/q)[x] when q is bounded by some fixed length integer is of the same order as the computing time for the Euclidean algorithm, so that it is bounded by deg(G) (deg(F) - deg(D)). The reason for this is clear from the fact that the A and B produced by the algorithm are obtained by multiplications where the final products, A and B, satisfy the conditions deg(A) < deg(G), deg(B) < deg(F).

For Algorithm II-2.2, there are essentially three polynomial multiplications (A' C), (B' C), (Q F) and a division with remainder A' C = Q G + R. Their costs are bounded by deg(G) (deg(H) - deg(D) + 1), deg(F) (deg(H) - deg(D) + 1), (deg(H) - deg(D)) (deg(F) + 1), and (deg(H) - deg(D)) (deg(G) + 1), respectively. The approximate total cost is, then

2 (deg(F) + deg(G) + 1) (deg(H) - deg(D)) + deg(F) + deg(G).

For the univariate Zassenhaus' Quadratic Extension Algorithm II-3.2: Step (1) costs deg(G) deg(H); Step (2) and Step (4) are bounded by T(G) T(H); in Steps (3) and (5), since

deg(C) and deg(R) $\leq$ deg(G) + deg(H) and deg(D) = 0, the costs are bounded by

$2\,(\deg(G) + \deg(H))\,(\deg(G) + \deg(H)) + \deg^2(H)$ or

$(\deg(G) + \deg(H))^2$ or simply $\deg^2(F)$. Since Steps (2)-(5) are carried out k times and assume $T(G) = T(G_0)$, $T(H) = T(H_0)$ for each step, then the total cost is approximately $k\,(T(G_0)\,T(H_0) + 2\deg^2(F))$. For dense univariate polynomials $T(F) = \deg(F) + 1$, hence the total cost is essentially dominated by $k \deg^2(F)$ or $k\,d^2$ if deg(F) = d. Note that here we have not considered the integer length of the coefficients in F. This detail was omitted on the assumption that the coefficients occuring in F for practical cases can all be expressed by, say, at most four 36 bit binary computer words. In fact, if we have infinite precision integral coefficients the cost for this algorithm is still just a maximum integer length multiple of the above (for instance, see [MUS71]).

The Hensel's Algorithm (II-3.1) is just a simpler and shorter version of Algorithm II-3.2. Except the number of steps is k' where $k = \log(k')$ or $k' = 2^k$. Thus the cost of

the Hensel's Algorithm is $k'\,(T(G_1)\,T(H_1) + 2\deg^2(F))$.

For the analysis of the Generalized Hensel Algorithm II-4.1, it is necessary to first consider the computing cost of the substitution

$P'(y_1,\ldots,y_v) \leftarrow P(y_1+b_1,\ldots,y_v+b_v)$. Let d_i, $i = 1,\ldots,v$ be the maximum degrees of variables y_i in $P(y_1,\ldots,y_v)$.

For each i, the computation of $(y_i + b_i)^j$ for $j = 1,\ldots,d_i$ requires $2(2 + 3 + \ldots + d_i) = (d_i - 1)(d_i + 2)$ term by term multiplications. Thus $\sum_{i=1}^{v} (d_i - 1)(d_i + 2)$ term operations are needed for computing all the powers for the substitutions of the variables. Assume now the polynomial P is put in fully expanded form in $J[y_1,\ldots,y_v]$. Then a typical monomial is $y_1^{k_1} y_2^{k_2} \ldots y_v^{k_v}$ and the cost for multiplying the k_i degree dense polynomials of $(y_i + b_i)^{k_i}$ al-

ready prepared is $\prod_{i=1}^{v} (k_i + 1)$. If the fully expanded polynomial P contains T(P) terms or monomials, then the total cost of substitution to get P' is (*) .....

$$\sum_{j=1}^{T(P)} \prod_{i=1}^{v} (k_{ij} + 1) + \sum_{i=1}^{v} (d_i - 1)(d_i + 2).$$ (Note that a brief analysis of substitution of this type with Horner's rule not only shows the same order but also reveals its inefficiencies in redundant operations for these multiple substitutions. In a private communication with E. Horowitz, the above statement was also verified by his computing time formulas also [HOR73].)

If n is the maximum term degree of P in the variables $y_1, \ldots, y_v$, then n bounds all $k_{ij} + \ldots + k_{vj}$, so that the total cost of substitution is bounded by

$$T(P)\ (n/v)^v + d_1^2 + d_2^2 + \ldots + d_v^2.$$

If P is a dense polynomial of degree d in each variable, then the cost bound becomes

$$\sum_{k_1=0}^{d_1} \cdots \sum_{k_v=0}^{d_v} \prod_{i=1}^{v} (k_i + 1) + \prod_{i=1}^{v} d_i^2,$$

$$\prod_{i=1}^{v} (d_i + 1)(d_1 + 2)/2 + \prod_{i=1}^{v} d_i^2, \text{ or approximately}$$

$d^{2v}/2 + v\,d^{2}$ which is roughly the cost of multiplying P by itself.

It is important to point out that the number of terms in P' could become as large as a dense polynomial of degree $d_1, \ldots, d_v$ which could be much larger than T(P). Hence, not only may the computing cost of substitution be expensive, but also this type of blowup in the number of terms may cause computational inefficiencies in some unfortunate cases, as we will see later in this chapter. Therefore, it is important to choose as many zeros in the set of substitution values $b = \{b_1, \ldots, b_v\}$ whenever possible. In the expression (*), if any $b_{i'} = 0$ then we can accordingly set $k_{i'j}$ to 0 and $d_{i'}$ to 1 to indicate that essentially no work is involved for this variable. The resulting savings in computing cost for the Generalized Hensel Algorithm is obviously great

for each additional zero b_i. This blowup problem due to non-zero substitutions will be called the non-zero-substitution problem and will be discussed later for some individual algorithms.

Let the input polynomial F of the Generalized Hensel Algorithm have d_i, $i = 1,\ldots,v$, as the maximum degree in the non-main variable y_i. Then Step (1) of the GHA is the substitution operation whose cost is bounded by

$T(F)\ (n/v)^v + d_1^2 + \ldots + d_v^2$, if we assume essentially no additional cost for multiplying a coefficient polynomial in x, C(x), with a monomial in the other variables as compared with multiplying a constant and monomial. The cost of Step (2) is $\deg(G_1)\deg(H_1)$. For Step (3), we observe that the multiplication of an univariate polynomial in x by x^l involves essentially a shift operation or additions of the degrees by a constant, i. Thus, in the deg(F) applications of Algorithm II-2.2, only the division steps and one multiplication need to be counted. The cost of these operations is

$$\sum_{i=1}^{\deg(F)} [(\deg(H_1) + i)\deg(H_1) + i\deg(G_1)] =$$

$\deg(F)\ \deg^2(H_1) + \deg(F)\ (\deg(F)(\deg(F)+1)/2)$ or approximately bounded by $\deg^3(F)/2$. Because of the iterative update of D, carrying out Step (5) up to n times constitutes a multiplication of G_n and H_n, hence the total cost is $T(G_n)\ T(H_n)$. Step (4) has negligible cost and Step (6) only involves "scalar" multiplications. Step (7) is again a substitution step for polynomials smaller in size than F in Step (1), hence it is bounded by $T(G_n)\ (n/v)^v + T(H_n)\ (n/v)^v$. In fact the total cost of the Generalized Hensel Algorithm is bounded by

$$(T(F) + T(G_n) + T(H_n))\ (n/v)^v + d_1^2 + \ldots + d_v^2 + \deg^3(F)/2 +$$

$T(G_n)\ T(H_n)$ or simply by

$$T(G_n)T(H_n) + (T(F) + T(G_n) + T(H_n))(n/v)^v \ldots\ldots (E1).$$

For the Univariate EZ Algorithm II-5.3, there is one additional cost to the cost of applying Zassenhaus' Quadratic Extension Algorithm. That is the computing cost of cont(H) and pp(H) which essentially involves at most deg(H) integral GCD computations with each bounded by the maximum

integral length squared. For fixed length integers this is a cost depending essentially only on deg(H). However, in the case of Multivariate EZ Algorithm, the situation is somewhat different. Not only do cont(H) and pp(H) constitute additional costs to that of GHA, but also the cost of multivariate division and multiplication for computing

H and G mod S^n needs to be counted. However, Lemma II - 5.5 shows that it is possible to avoid this expense at the cost of predetermining leading coefficients of the divisors. So we will delay analysis of this problem until later when we discuss practical applications of the principles of Lemma II - 5.5. Furthermore, the following observations will decrease the severity of this problem.

The above analysis for the GHA is based essentially on the sum of the dominating costs for each individual step. From the expression (E1) we find that the costs for substitutions and multiplications seem to be the most significant. But the difficulty in using this expression for an estimate of computing cost lies in the fact that the basic operations of substitution and multiplication are different, so that their basic unit costs are also different. In the manipulation of multivariate polynomials, there can be many different basic operations that a complete timing analysis should include. But these operations often

have varying cost units, unlike many algorithmic analyees in complexity theory where the most costly basic operations are often either a comparison of two records or a multiplication of two fixed point integers. It is then clearly advantageoue to find some particular parameters, for an algorithm involving many different basic operations, on which essentially all relevant computing coets depend somewhat explicitly. For this basic reason as well as the very sound argument that the following variation of cost analysis can be more easily substantiated by actual run times, we propcse a new formula for the total proportional cost of the GHA. The main parameter for the computing coet formula, which we will attempt to establish by the following arguments and actual computing timee, is the number of terms for the polynomials.

Let F' denote $F(x,y_1+b_1,\ldots,y_v+b_v)$. Since the most important operations for the GHA is to correct for terms in $R_m = F' - G_m H_m$ at the mth iteration, after n iteratione the algorithm would have corrected for all the croee terms of $G_n H_n$ with $G_n = G_n^{(1)} + G_n^{(n-1)} + \ldots + G_n^{(1)}$ and $H_n = H_n^{(1)} + H_n^{(n-1)} + \ldots + H_n^{(1)}$. For each dietinct term in

this product, some work is involved. That work includes the multiplication that produced the term; isolating this term by finding the correct integral coefficient, the power of x, and the monomial in the other variables; solving the corresponding DUPE for the term; and finally assembling the solutions of the DUPE with the monomial and adding the results to the two current codivisors. Quite clearly, all this work for one term dominates in cost over any other basic operations such as sorting, comparing, multiplying or dividing by integers, etc. In fact, even the content and primitive part computations in the restore-leading-coefficient operations correspond roughly to these individual terms. Thus it is quite reasonable to expect that the total computing cost for GHA is proportional to the number of distinct terms that the algorithm has to deal with, and that is approximately (or at least bounded by) $T(G_n)\,T(H_n)$, except when F' is dense. In such a case F' would contain all possible terms that the product of G_n and H_n could generate. In this case, T(F') is the number of terms that GHA has to deal with. Therefore, the total cost of GHA should be approximately proportional to $(T(G_n)\,T(H_n)$ or T(F')) + cost of substitution. Actually the substitution cost can generally be included in the first part of the

above expression since it is $(T(F) + T(G_n) + T(H_n))\ (n/v)^{**}v$.

Clearly, if F' is dense, then $T(F) \leq T(F')$, $T(G_n) \leq T(F')$,

and $T(H_n) \leq T(F')$; whereas if

$\max(T(G_n)\ T(H_n), T(F')) = T(G_n)\ T(H_n)$ this $T(G_n)\ T(H_n)$ still

dominates $(T(F) + T(G_n) + T(H_n))$ in general. Therefore, we

can even go further and simply claim that the cost of GHA is

proportional to $T(G_n)\ T(H_n) \oplus T(F')$ .....(E2).

The "$\oplus$" is used here in the special sense that the total

cost is proportional to $T(G_n)\ T(H_n)$ in general except

when F' is dense, then it is mainly proportional to T(F').

There is one qualification necessary for the dense case.

Even though T(F') is the main parameter which many basic

operation costs in GHA are proportional to, the classical

term-by-term multiplication cost which is on the order of

$T(G_n)\ T(H_n)$ and is non-linear with respect to T(F') will

begin to take a significant part of the total GHA computing

cost if G_n and H_n are also quite dense and have a large

number of terms. This means that as the problem size in-

creases in the dense case, the multiplication cost can begin

to dominate the costs of other operations. For example, in

Case 2 of the empirical test data of the next section we explicitly recorded the time for the Hensel constructions on linearly dense quartic polynomials F and the related multiplication time (in seconds):

v = 1		2	3	4	5
Multp.	0.10	0.27	0.59	1.15	2.10
Hensel	0.46	0.76	1.35	2.28	3.80
%	22%	34%	43%	50%	55%

We note that the percentage of computing time for multiplication in the GHA in this problem is increasing as the number of variables increases. But as we will see from the "ratio tests" in the next section the linear behavior w.r.t. T(F') still dominates over the quadratic contributions from the multiplication of G_n and H_n. Similar recordings of multiplication times and total GHA time were also made for other cases of empirical test data in the next section. For the non-dense cases, the percentage of multiplication time has never exceeded 20% of the total GHA time; whereas the dense cases show a upper bound of 60%. At the point of 60%, however, the test problems involve polynomials of more than 1875 terms which are quite large as far as the polynomials that can reasonably be handled by a present-day symbolic system are concerned. Thus, for the class of problems where the polynomial sizes are not extremely large so as to prohibit computation in present-day symbolic systems,

formula (E2) gives a good estimate of the work involved to a reliable accuracy even for the dense case). Furthermore, we repeat, large multivariate polynomials are inevitably sparse, so that our computing cost estimate for GHA by (E2) will hold for quite a wide range of problems.

We now make estimates of the cost of computing GCD's of multivariate polynomials using the EZGCD Algorithm under the following assumptions:

(ASM1) The integer coefficients of the input polynomials and the resulting polynomials after solution with the chosen b-values are small (w.r.t. the maximum single precision integers in most present-day computers) so that the total computing time is not significantly dependent on the coefficient sizes or the time for arithmetical operations therof. (See (A3) and (A4) of [BRO71]).

(ASM2) Unlucky primes and evaluation values are never used in the Hensel constructions (i.e. Step (A4) of the EZGCD Algorithm ensures Condition III-B or a lucky evaluation for later steps). (See (A2) of [BRO71]).

(ASM3) The cost of content computations is negligible compared to the actual cost of computing the GCD of the primitive parts of input polynomials.

(ASM4) There is no need to use the special case algorithm (S1) - (S7) (i.e. Condition III-A holds).

Since Steps (A1) - (A9) are simply more detailed statements of Steps I - V of the outline of the EZGCD Algorithm, in Section III-2, we will examine these five steps. Assumption (ASM3) basically states that the cost of Step I is negligible. In fact, it assumes that the cost of GCD computations with one less variable is negligible. That includes computing d' = gcd(f', g') and dl" = gcd(lc(F), lc(G)) as well as the restore-leading-coefficient operations. This is in general not correct. Practical experiences tell us that the totality of all these GCD's of one less variable could in certain cases amount to as much as one-third the total cost. But we make this assumption on the grounds that the problem of computing the GCD of primitive polynomials merits special attention so that f', g', and d' can all be assumed to be 1, that lc(F) and lc(G) are in most cases much smaller than F and G, and that for GCD problems the modified Step (A8') of Section III-5(a) helps to reduce the cost of the restore-leading-coefficient operation to the very minimal. Also the computation of GCD's of several polynomials together will be discussed further in Chapter VI.

Steps II and III involves only evaluations and univariate GCD's which usually cost much less than the other steps. (ASM1) in effect states that computing univariate GCD's is cheap. Step V consists of only one test-division.

Although the cost of dividing two polynomials by the classical method is asymptotically bigger than the cost of computing GCD's for polynomials of the same sizes, in most practical cases, divisions are much cheaper than GCD calculations. Alternatively, we can perform two parallel Hensel constructions on both F and G so that the total cost will simply be multiplied by a constant, 2, and the test-division can be eliminated by simply comparing the resulting potential GCD's from the two Hensel constructions. Actually (ASM2) assumes away the need for performing the test-division.

Therefore, based on these assumptions the most significant amount of time of the entire algorithm is spent in Step (A8) or (A8') - the Hensel construction. According to the analysis already done above and assuming we use the alternative Step (A8'), the cost of that is (according to (E1))

T(sub((dl"/dl) D) T(sub(dl H)) +

$$[T(dl'' U) + T(sub((dl''/dl)D) + T(sub(dl H))](n/v)^{v} \quad \ldots.(E3)$$

where D = gcd(F, G), dl = lc(D), dl" = gcd(lc(F), lc(G)), U = either F or G, "sub" denotes the substitution transformation discussed in II-4, T denotes the number of terms in fully expanded form, and n is maximum term degree of dl" U.

This formula seems very complicated so that we may want to make some further simplifying assumptions. First we assume

(ASM5) Sufficiently many zeros can be used in the valid lucky evaluation for Hensel construction so as to keep the number of terms in the substitution transformed polynomials approximately the same as that of the untransformed polynomials.

This assumption has the wide ranging implication that makes the substitution cost negligible compared with the other costs, in addition to making T(sub(P)) = T(P). Thus formula (E3) becomes simply

T((dl"/d') D) T(dl H) .....(E4)

If in addition we assume:

(ASM6) There is no appreciable increase in cost of the EZ Algorithm due to non-constant leading coefficients, so that Step (A8) can be used instead of (A8') and still avoid the leading coefficient blowup problem.

Then formula (E4) becomes T(D) T(H) .....(E5)

Note that this assumption is stated more strongly than necessary for formula (E5) to hold. Actually, so long as the LCB problem is not very severe (i.e. the unwanted unit multipliers u and 1/u, as discussed in Section II-5(a) and here equal to 1/dl and dl respectively are small mod S , and only cause minor increases in the number of terms), we can

still use (A8) instead of (A8') of the EZGCD Algorithm. Anyway, (ASM3) assumes that the cost for restoring leading coefficients is small, especially when the content or the multiplier is small.

If assumption (ASM6) is made separately from (ASM5), then the cost for substitution transformation still remains and formula (E3) becomes

T(sub(D)) T(sub(H))

$$+ [T(U) + T(\text{sub}(D)) + T(\text{sub}(H))] (n/v)^{v} \quad \ldots\ldots\ (E6)$$

where n is the maximum term degree of U.

On the other hand, if we make the same intuitive arguments as we did for the GHA above, and reason that in formula (E3) substitution cost can be proportionally included or simply use the formula (E2) instead of (E1) for the proportional cost of GHA, then we have a proportional cost for EZGCD Algorithm:

T(sub((dl"/dl) D) T(sub(dl H)) $\oplus$ T(sub(dl" U)) ..... (E7)

under only assumptions (ASM1) - (ASM4). That is the cost of EZGCD Algorithm is proportional to

T(sub((dl"/dl) D) T(sub(dl H)), except when dl" U is dense, then it is proportional to T(sub(dl" U)) until the problems become so large that multiplication costs begin to be equally significant to the total costs, then (E7) is taken as an actual sum so that both terms contribute to the total

cost.

The above formula reveals a computing cost of polynomial GCD's which is essentially proportional to the number of terms of the polynomials. Specifically, the cost is proportional to the number of terms of the polynomial used for the Hensel construction in case it is dense, otherwise, proportional to the product of the number of terms in the codivisors obtained by the Hensel construction.

It is important to point out that the cost estimation formula involves both the given polynomials of the GCD problem and the answers (the GCD and cofactors). Sometimes, such an expression can have only questionable value. However, we first emphasize that what our formula reveals is the clear fact that polynomial GCD computation (and probably many other polynomial computations) is an input and output dependent process. Knowing only the given inputs simply will not allow a good estimate of the computing costs, except possibly in certain special cases. In our situation, the nearly completely sparse polynomial is such a special case where, because $T(P\ Q) = T(P)\ T(Q)$ in sparse cases, the input polynomials alone can provide sufficient information to estimate the computing cost. In fact, the computing cost is linearly proportional to the number of terms in the polynomial used for the Hensel construction. This constitutes a great advantage over the Modular GCD Algorithm

when the given polynomials are sparse (and the transformations necessary for the Hensel construction do not blow up the number of terms severly). For the other extreme - the dense case, the situation is more complicated. So long as the multiplication cost does not dominate, input polynomials still provide enough information to estimate the actual computing cost, as data of cases 2 and 5 will show in the next section. In Case 5, the biggest dense quadratic polynomials (in 4 variables only), for which we dared to attempt a GCD computation, has 1875 terms. But this fact makes clear why we can not and should not consider operations on large dense multivariate polynomials (yet) and why asymptotic analysis of algorithms may be misleading for realistic problems which are within the grasp of present-day computation.

Next we emphasize a previously neglected point in computing time analysis. One of the most important purpose of performing computing time analysis for an algorithm is to account for the major effort put into the computation, and one hopes, predict the cost for similar computations in the future. This goal, if achieved, is significant in many obvious ways. Not only can one reveal the asymptotic behavior of the algorithm and the correctness of the program implementation, but one can also estimate future computational efforts so that no wasteful or impossible

computations are attempted and all limitations of the algorithm can be realized. Of course, comparisons of different or similar algorithms then become more meaningful also. What we will show with test cases in the next section is that, for a particular class of problems whose general structures are known, having computed the GCD and cofactors of a simpler problem, our cost estimating formula enables us to predict the cost or the computing time of a more complex problem in the same class to an accuracy that is surprisingly good especially as the problems get large. That is exactly the point and the importance of what we call the "ratio test".

IV - 3 Empirical Computing Results

With the analytical formulas for costs of computing GCD's by the EZGCD Algorithm based on the analysis of the Generalized Hensel Algorithm, we now present several tables of empirically observed computing times (in seconds) for the Reduced, Modular, and the EZGCD Algorithms. With these actual run times, we hope to show not only the performances of these algorithms for various classes of problems but also substantially validate our more intuitively derived computing cost formulas for the wide range of practically sized problems. We will find that the actual computing times can usually be predicted to within 15% accuracies in

most empirical test cases when the intuitive cost estimation formula (E7) is used. Such accuracies are quite good considering the wide variety of basic operations underlying such an algorithm for the computation of polynomial GCD's.

The three algorithms tested are implemented (hopefully) without any bias because of their independent designs and codings in the symbolic manipulation system MACSYMA [MAC73] and run on a PDP-10 computer with a 2 microsecond memory. The particular examples are chosen to demonstrate specific features. For the most part they are chosen for simplicity and clarity, especially w.r.t. the integral coefficiente, and are otherwise (hopefully) quite random. Note that timings for the Reduced GCD Algorithm are included here mainly for the purpose of demonstrating the competitiveness of the PRS type of algorithms in small and certain sparse cases. The general blowup behavior of the PRS GCD algorithms, even the most efficient subresultant PRS GCD algorithm, for large and dense problems is quite well known and expected. Such a phenomenon will also be revealed by the data below. Our main purpose for this empirical data, however, is to compare the performances of the Modular and EZGCD Algorithms and to support the analytical computing time predictions with some actual figures. Numbers in parentheses indicate the number of "garbage collections", which cost aproximately 3 seconds each, made during the GCD

computation. This number gives a fairly good indication of the storage space consumed by intermediate expressions.

Case 1: gcd(F, G) = 1;

$$F = (x + \sum_{i=1}^{v} y_i + 1)\ (x + \sum_{i=1}^{v} y_i + 2);$$

$$G = (x^2 + \sum_{i=1}^{v} y_i^2 + 1)\ (-3\, y_1\, x^2 + y_1^2 - 1).$$

v = 1	2	3	4	5
EZ. 0.307	0.457	0.637	0.855	1.092
REO. 0.568	2.544	10.55	44.02(2)	147.5(10)
MOD. 0.213	0.390	0.601	0.849	1.144

This GCD = 1 case is supposedly best suited for any modular algorithm. Since the Modular and the EZGCD Algorithms are based on modular arithmetic, they both perform well as expected.

Case 2: Linearly dense quartics with quadratic GCD's.

$$D = (x + \sum_{i=1}^{v} y_i + 1)^2;$$

$$F = D\,(x - \sum_{i=1}^{v} y_i - 2)^2;$$

$$G = D\,(x + \sum_{i=1}^{v} y_i + 2)^2.$$

	v = 1	2	3	4	5
EZ.	1.320	2.520	5.124	9.475	19.75(1)
RED.	0.897	3.3[illegible]7	10.85	34.34(1)	---
MOD.	1.194	6.071	33.98(1)	180.4(7)	---

This is a dense monic case where both F and G have degree 4 in x, the main variable, and their GCD, D, is a quadratic polynomial. For the Reduced GCD Algorithm, the PRS sequences are normal and involves only two pseudo-division steps. For the Modular GCD Algorithm, we have degree d = 4 in each variable, so that according to the formula of computing time $(d + 1)^{v+1}$, the ratio of computing times for our polynomials above should be d + 1 = 5 and we see that the ratios between successive pairs of the actual computing times (less approximately three seconds for each garbage collection) approximate the prediction remarkably well. For the EZGCD Algorithm we will also perform this "ratio test". The ratios between succesive pairs of actual computing times are roughly RA(2, 1) = 1.9, RA(3, 2) = 2.0, RA(4, 3) = 1.8,

and RA(5, 4) = 1.7, rounding to two significant digits. Since the given polynomials are monic and dense while the number of terms are still reasonably small, formula (E7) would apply in this case as simply T(U). Here, F of the two original polynomials F and G, since their degrees are the same in x, is used for the Hensel construction. The number of terms are T(U) = T(F) = 13, 29, 57, 102, 170. Thus ratios of theoretically predicted costs are ratios of successive T(U) which are RT(2, 1) = 2.2, RT(3, 2) = 2.0, RT(4, 3) = 1.8, and RT(5, 4) = 1.7, also rounding to two digits. As we can see, this also agrees remarkably well with the ratio of actual computing times. Except possibly the first ratio, which is due to the high overhead (total preparation costs, including computing contents and substitutions) which is unusually high for the first small example. This phenomenon will also occur for later cases. But, in each case, as the problem size increases the ratios tend to agree better and better. This is in fact the reason for making these ratio tests - as the problem sizes increase in our "controlled experimental" environments the relatively small overhead costs become dominated and the total cost of the algorithm corresponds more and more to the number of terms of the polynomials involved . This gives strong support to our heuristically derived estimate of computing costs for the EZGCD Algorithm as well as Generallzed Hensel

Algorithm and other related Hensel constructions.

Case 3: Sparse polynomials where degrees and the number of variables increase with v.

$$D = x^{v+1} + \sum_{i=1}^{v} y_i^{v+1} + 1;$$

$$F = D\,(x^{v+1} + \sum_{i=1}^{v} y_i^{v+1} - 2);$$

$$G = D\,(x^{v+1} + \sum_{i=1}^{v} y_i^{v+1} + 2).$$

v =	1	2	3	4	5	6
EZ.	0.544	0.755	1.127	1.601	2.183	2.831
RED.	0.187	0.615	1.141	1.938	3.051	4.538
MOD.	0.631	6.148	124.3(8)	---	---	---

This is again a monic case, but now the polynomials F and G are quite sparse. Thus formula (E7) applies as T(D) T(H). T(D) = T(H) = 3, 4, 5, 6, 7, 8. Thus the ratios of theoretically predicted costs are approximately RT(2, 1) = 16/9 = 1.8, RT(3, 2) = 25/16 = 1.56, RT(4, 3) = 36/25 = 1.44, RT(5, 4) = 49/36 = 1.36, and RT(6, 5) = 64/49 = 1.30. Compared with ratios of actual computing times RA(2, 1) =

1.4, RA(3, 2) = 1.50, RA(4, 3) = 1.42, RA(5, 4) = 1.36, and RA(6, 5) = 1.30, we again have very good agreements. Also in this case the Reduced GCD Algorithm performs quite well, because of the sparseness of the problems and the symmetric regularity of the given polynomials. But the next table shows that slight variations of the input polynomials could cause the timing of the Reduced Algorithm to change drastically where as the EZGCD Algorithm remains quite stable.

Case 3': D and F as above;

$$G = D\,(x^{v} + \sum_{i=1}^{v} y_i^{v} + 2),$$

v = 1		2	3	4	5	6
EZ.	0.620	1.112	1.757	2.574	3.565	4.666
RED.	0.436	6.141	134.9(5)	---	---	---
MOD.	0.721	7.886	158.9(9)	---	---	---

Since G is of lower degree in x than F, it is the one that is used for Hensel construction according to the heuristics set in EZGCD Algorithm. Here G is nearly completely sparse so that the formula T(D) T(H) still applies and the RT ratios are the same as above. Compared with ratios of actual computing times for the data of this table

RA(2, 1) = 1.8, RA(3, 2) = 1.58, RA(4, 3) = 1.46, RA(5, 4) = 1.38, and RA(6, 5) = 1.31, we still have fairly good agreements.

Case 4: Quadratic non-monic GCD with quadratic cofactors.

$$D = y_1^2 x^2 + \sum_{i=2}^{v} y_i^2 + 1;$$

$$F = D\,(x^2 - y_1^2 + \sum_{i=2}^{v} y_i^2 - 1);$$

$$G = D\,(y_1 x + \sum_{i=2}^{v} y_i + 2)^2.$$

	v = 1	2	3	4	5
EZ.	1.184	1.612	2.506	3.822	5.549
RED.	0.697	6.480	55.58(1)	407.6(20)	---
MOD.	1.487	6.754	39.15(2)	190.9(10)	---

All three previous cases have been with monic polynomials which not only avoid the leading coefficient problem but also the non-zero substitution problem for the Hensel construction in the EZGCD Algorithm. So we now test a more complex situation for the EZGCD Algorithm. Here we must use

(E7) as T(sub((dI"/dI) D) T(sub(dI H)) with U = F, dI" = y_1^2,

and dI = y_1^2. Since Ic(F) = y_1^2, the evaluation value for y_1

can not be 0, so that there will be some blowup in the

number of terms when we substitute $y_1 + b_1$ for y_1. In this

case T(sub((dI"/dI) D) = 4, 5, 6, 7, 8, since dI"/dI = 1 and

each $y_1^2 \times$ in D becomes three terms after substitution.

T(sub(dI H)) = 8, 11, 14, 17, 20. Note that, in counting

the terms above, we have implicitly assumed that all other

evaluation values except b_1 for y_1 are zeros. Thus the

corresponding ratios of theoretically predicted costs are

RT(2, 1) = 55/32 = 1.73, RT(3, 2) = 84/55 = 1.53,

RT(4, 3) = 119/84 = 1.42, and RT(5, 4) = 160/119 = 1.35.

Compared with the ratios of actual computing times

RA(2,1) = 1.36, RA(3, 2) = 1.56, RA(4, 3) = 1.50, and

RA(5, 4) = 1.45, the agreements are still reasonably good

considering the many uncertainties in this case. Two spe-

cifically important uncertainties are due to the fact that

the total number of terms involved is still quite small so

the overhead costs are not completely dominated and the

fact that this is a case intermediate between dense and

sparse so formula (E7) is a simplifying estimation of the proportionalities rather than a more exact count. The Modular GCD Algorithm, however, behaves quite predictably.

According to the computing time formula $(d + 1)^{v+1}$ where $d = 4$ in this case, the actual computing time increases by a factor of approximately 5 as v increases from 1 to 4.

Case 5: Completely dense non-monic quadratic polynomials with dense non-monic linear GCD's.

$$D = (x + 1) \prod_{i=1}^{v} (y_i + 1) - 3;$$

$$F = D\,((x - 2) \prod_{i=1}^{v} (y_i - 2) + 3);$$

$$G = D\,((x + 2) \prod_{i=1}^{v} (y_i + 2) - 3).$$

v = 1		2	3	4
EZ.	1.032	3.712	16.77	100.5(5)
RED.	0.845	13.47	525.6(45)	---
MOD.	0.814	4.258	22.85(1)	102.1(5)

This is a completely dense case for which the Modular GCD

Algorithm is at its best. Since dl" = gcd(lc(F), lc(G))

$$= lc(D) = dl = \prod_{i=1}^{v} (y_i + 1)$$ and (dl" F), (dl" G) are used

for the Modular Algorithm, the computing time formula for this case should be $(d + 1)(2d + 1)^v$ with d = 2. Thus the predicted ratio of computing times for Modular GCD Algorithm is 5. We see that this ratio is approximately observed in the actual computing times of the Modular Algorithm. Next, we note that even though d = 2 for all variables in F and G, and D is linear and dense so that the PRS sequences consist of results for only one pseudo-division, the Reduced GCD Algorithm still loses badly to the other two algorithms. As for the EZGCD Algorithm, this case represents a difficult extreme where there are not only leading coefficient blowup problems but also dense leading coefficients in dense polynomials. The actual computing times, however, turned out to be good compared with the Modular times. The ratios of these times are RA(2, 1) = 3.6, RA(3, 2) = 4.5, RA(4, 3) = 5.0. On the other hand, since the polynomials are dense, substitutions do not change the number of terms, so that we can use formula (E7) for the cost of the EZGCD Algorithm with cost = T(dl" U) where U = F because deg(F) = deg(G). Thus, similar to the Modular Algorithm, we

have also T(dl" U) = 3*5**v. According to this, the ratio of theoretically predicted costs is 5 for all pairs of successive problems. Indeed, we see the actual ratios approach this number 5 as v gets large, or as the problem becomes sufficiently large so that the overhead costs are dominated.

Case 5': Sparse non-monic quadratic polynomials with linear GCD's

$$D = x \prod_{i=1}^{v} y_i - 1;$$

$$F = D\,(x \prod_{i=1}^{v} y_i + 3);$$

$$G = D\,(x \prod_{i=1}^{v} y_i - 3).$$

v = 1		2	3	4
EZ.	0.644	1.436	4.605	26.69(1)
RED.	0.164	0.245	0.337	0.431
MOD.	0.710	3.771	22.41(1)	113.0(5)

This case is only slightly different from Case 5 as we can easily see, except that the polynomials are very sparse here. In fact, each of F and G has only three terms. For

such sparse polynomials with low degrees (2) the number of PRS elements is very small for the Reduced GCD Algorithm. Actually, for each problem, the PRS is normal and requires only one pseudo-division, so that there can not be much growth at all. That accounts for the relatively fast computing times for the Reduced algorithm in the above table. For the Modular Algorithm, the timings are surprisingly similar to the dense problems of Case 5. The common ratio for predicted times and for the actual computing times remains 5, which clearly demonstrates the insensitivity of the modular algorithm to the sparseness of the polynomials and the computing time formula derived for this algorithm indicates this "worst-case" behavior. It shows that the number of variables in the problem and their maximum degrees are the only parameters affecting the computing times for the Modular GCD Algorithm. These are not yet the main points to be made with the empirical data of this case. The major observation to be made here is that this is a bad case for the EZGCD Algorithm also. Since the polynomials in x have products of powers of all other variables as leading coefficients, the validity requirement for the evaluation values forces them to be non-zero. Thus, after the substitution transformations ($y_i' + b_i$ for y_i) are performed on the polynomial used for Hensel construction

(in this case, it does not matter if it is F or G), the new polynomial that the Multivariate EZ Algorithm has to work with is just as dense as the problems of Case 5. In fact, the number of terms will be exactly the same with only different numerical coefficients (due to differences between various b 's of Case 5' and 2 of Case 5). So, T(sub(dl" J)) i

for this case is the same as T(dl" U) of Case 5 which is 3*5**v. Accordingly the ratio of predicted costs for pairs of successive problems is also 5. Indeed, we see that RA(2, 1) = 2.2, RA(3, 2) = 3.2, and RA(4, 3) = 5.1 which is approaching 5 as the problem get large. Note that the actual computing times of this case are proportionally less than those of the dense problems and those of the Modular Algorithm, because of the fact that the blowup due to non-zero substitutions only occurs in the step of EZGCD Algorithm where the Hensel construction is applied and not in other steps. Therefore, we emphasize the point that when the validity requirement of the evaluation values force them to be non-zero and cause the non-zero substitution problem, the computing time unfortunately goes up close to that of more dense problems. The phenomenon is quite similar to that of the Modular GCD Algorithm, but these bad cases for the EZGCD Algorithm occur much less frequently (since they depend on the number of non-zero values for valid and lucky

evaluations) than the bad cases for Modular Algorithm which depends solely on the number of variables and their degrees in the given polynomials.

Case 6: Trivariate Polynomials with increasing degrees

$$D = x^{j} y (z - 1);$$

$$F = D (x^{j} + y^{j+1} z^{j} + 1);$$

$$G = D (x^{j+1} + y^{j} z^{j+1} - 7).$$

j = 1		2	3	4	5
EZ.	1.192	1.523	1.934	2.449	3.067
RED.	0.896	1.750	2.697	3.884	5.321
MOD.	5.203	15.24(1)	31.42(3)	51.88(5)	82.85(9)

For all previous cases, we have used the number of variables, v, as the major parameter for experimentations. We observed that in those cases, the Modular as well as the Reduced GCD Algorithms are extremely sensitive to the increase of v. The EZGCD Algorithm depends more on the number of terms it has to process, rather than on v alone, so it is more efficient in the sparse polynomial situations. We also saw the predictabilities of the actual computing times with some of our formulas, especially for the Modular and the EZGCD Algorithms.

In this case, we concentrate on testing another

important parameter - the degree of the polynomials. Our purpose is also to see the performance of the EZGCD Algorithm for sparse polynomials but having a leading coefficient blowup problem that worsens with respect to the degree. First of all, the numbers in the above table reveal the fact that the EZGCD Algorithm still performs well even under these adverse conditions. Looking at the polynomial inputs in more detail, we have dl" = dl = y z. This forces the evaluations to be non-zero. In addition, the cofactors contain high powers of y and z, so that the blowup due to non-zero substitutions in Hensel constructions are quite severe. In fact, we get dense polynomials in y and z. Thus we use formula (E7) with proportional cost = T(sub(dl" U)) corresponding to dense polynomials. U = F, since deg(F) < deg(G). If we count carefully, T(sub(dl" U)) = 41, 59, 81, 107, and 137. From those, we get RT(2, 1) = 59/41 = 1.44, RT(3, 2) = 81/59 = 1.37, RT(4, 3) = 107/81 = 1.32, and RT(5, 4) = 137/107 = 1.28. Compared with ratios of actual computing times RA(2, 1) = 1.28, RA(3, 2) = 1.27, RA(4, 3) = 1.26, and RA(5, 4) = 1.25, the agreements are quite adequate. For the Modular GCD Algorithm, we should use dl" F and dl" G. The degrees of dl" F in x, y, and z respectively are 2j, j + 3, and j + 2. Using these degrees in the standard formula for the Modular algorithm, we find that the computing time should be growing somewhat as

(2 j + 1) (j + 4) (j + 3). Thus

RT(2, 1) = (5 6 5)/(3 5 4) = 2.50,

RT(3,2) = (7 7 6)/(5 6 5) = 1.96,

RT(4, 3) = (9 8 7)/(7 7 6) = 1.71, and

RT(5, 4) = (11 9 8)/(9 8 7) = 1.57. Compared with ratios of actual computing times RA(2, 1) = 2.35, RA(3, 2) = 1.83, RA(4, 3) = 1.65, and RA(5, 4) = 1.51, the agreements are quite good considering the fact that polynomials are actually sparse rather than dense as the formula expresses. Finally we also point out that the PRS sequences for the Reduced GCD Algorithm are normal in this case. That partially explains the good performance of the Reduced Algorithm.

Case 7: GCD's of trivariate polynomials requiring special case method of EZGCD Algorithm

$P = x - y z + 1$; $Q = x - y + 3 z$;

$F = P^{j} Q^{k}$; and $G = P^{k} Q^{j}$.

j, k = 1,2	1,3	1,4	2,4
EZ.	2.568	3.914	5.778	10.80
RED.	1.354	28.03(1)	729.7(77)	73.70(4)
MOD.	4.263	8.492	18.39(1)	33.58(2)

This is a very simple trivariate case where the special case method of the EZGCD Algorithm has to be used. We employ

this case to show that even though the special case method is much more inefficient than the regular algorithm ((A1) - (A9)), it is in fact not more inefficient than the other GCD algorithms in many cases similar to the one tested here.

CHAPTER V. Polynomial Factorization

V - 1 Introduction

In this chapter, we will discuss methods for factoring polynomials of one and several variables over the integers. The first computationally feasible method for the factorization of univariate polynomials over Z was due to Kronecker. As presented by Van der Waerden [VDW49, Section 25], the method is essentially as follows:

For a given polynomial P(x) of degree n, let m be the greatest integer $\leq n/2$. For $m + 1$ integral values $a_0, a_1, \ldots, a_m$; compute $P(a_0), \ldots, P(a_m)$ and let

S_i = the set of all distinct integral factors of $P(a_i)$.

For $k = 2, \ldots, m+1$ do the following. Choose set of k elements b_i, one from each S_i, and use interpolation to find a polynomial Q(x) of degree k-1 such that $Q(a_i) = b_i$ for all $0 \leq i \leq k$. If Q(x) divides P(x) then we have found a factor of P and we can recursively apply this method on P(x)/Q(x). Otherwise choose another set of k b_i's from the S_i's different from all previously chosen sets, interpolate, and test-divide again. When all possible combinations of

$m + 1$ or fewer integral values from the S_i's have been exhausted (the step for $k = m + 1$ is done), we conclude that P is irreducible.

The reason for the lack of success with the Kronecker's method even with today's high speed computers is the trial-and-error nature and the exponential number (w.r.t. n) of possible factors needed to be tried. It is also a nontrivial task to factor integers; in fact, its cost is an exponential function of the length of the given integer. These exponential growths dependent upon the degree and the integral length of the coefficients of the given polynomial prevent this method from being a practical computational algorithm even after several improvements were suggested [VDW49] [JOH66].

Berlekamp's factorization algorithms for polynomials over finite fields [BER67] [BER70] proved to be fundamental for developing efficient algorithms for factoring polynomials over the integers. Based on Berlekamp's algorithm Knuth [KNU69, Section 4.6.2] suggested a method of reconstructing the factors over Z using the Chinese Remainder Algorithm. However, the interpolative nature of the Chinese Remainder Algorithm (Garner's Rule) retains the exponential growth with respect to the degree of the given polynomial. Although the exponential growth due to

factoring integers is completely avoided in this method, the number of tentative factors and their combinatorial possibilities is still potentially large.

It was Zassenhaus' suggestion of using Hensel's p-adic construction (Lemma II-3.1) and his quadratic improvement thereof (Lemma II-3.2) that initiated new research for more efficient algorithms in polynomial factorization based on mod p factorizations. In his thesis, Musser [MUS71] presents abstract algorithms for factoring univariate and multivariate polynomials which theoreticaly establish the feasibility of using Hensel-type constructions for factorization in general algebraic spaces. He also presents detailed specifications, careful discussions, asymptotic computing time bounds, and some empirial timings for the algorithm for factoring univariate polynomials over the integers based mainly on Berlekamp's and Zassenhaus' algorithms.

For multivariate polynomial factorization, Musser's abstract algorithm has definite computational drawbacks, as pointed out previously. So our presentation of factorization algorithm in this case will essentially be based on that discussed by Wang and Rothschild [W&R73] which uses the computationally efficient Generalized Hensel Algorithm of Section II-4. In addition, we attempt to analyze the computing cost of the multivariate factorization

algorithm which has hitherto been lacking in the literature.

V - 2 Factorization of Univariate Polynomials

The algorithm for factoring an univariate polynomial F(x) over Z can be divided into six basic steps as follows:

(1) Set c <-- cont(F) and F <-- pp(F).

(2) Find the square-free decomposition of F, $\prod_{i=1}^{k} F_i^i$.

Each F_i will now be square-free and primitive.

For such a polynomial G we factor by

(3) Find a prime p in Z such that g(x) <-- G(x) (mod p) has the same degree as G and is square-free in (Z/p)[x]. (The existence of such a prime can be shown quite easily. For reference, see [W&R73].)

(4) Factor g(x) into $g_1(x)\, g_2(x) \ldots g_r(x)$ in (Z/p)[x] by applying the Berlekamp's algorithm over finite fields. (Both Musser [MUS71] and Wang and Rothschild [W&R73] have detailed discussions and algorithmic implementations of this algorithm for "small" prime fields.) If r = 1, G is irreducible over Z.

(5) For $r \neq 1$, apply Zassenhaus' algorithm (Algorithm II-3.2) successively on complementary products of these

g_i's to obtain $G_1(x), G_2(x), \ldots, G_r(x)$ such that

$G = G_1 G_2 \ldots G_r \pmod q$ where $q = p^{2^m}$ which bounds any integral coefficient of any factor of G. (See discussion in proof of Theorem II-5.3.)

Also $G_i = g_1 \pmod p$ for all $1 \leq l \leq r$.

(6) Combine extraneous factors and restore leading coefficient for these G_i's and get the true factors of G over Z.

Both Algorithm 2.6P of [MUS71] and Algorithm TRUFACTORS of [W&R73] discuss this step in detail.

Musser gives a very complete analysis of asymptotic computing time bounds for this univariate case in his thesis [MUS71]. We refer the reader to Musser's thesis for details. However, at this point we will call the reader's attention to the problem of combinatorially collecting extraneous factors into true factors. For an irreducible polynomial with k extraneous factors, as many as 2^k combinations of factors and divisions may be required before the irreducibility of the polynomial can be discovered.

V - 3 Multiple Factor Generalized Hensel Construction

Because of the fact that a polynomial may have many factors, we will present a new version of the Generalized Hensel Algorithm of Section II-4 which will construct more

than two multivariate factors from their evaluated univariate representations. This algorithm will be equivalent to GHA when the number of factors is two. Also, as we will see later, the computing cost of this algorithm will essentially be $1/(r-1)$ times less than that of applying GHA $r-1$ times on successive groupings of the r factors as the way this is done in [W&R73] and [MUS71]. Since we believe that the savings in cost for the univariate case will not be as significant as the multivariate case, we will only discuss this semi-parallel multiple factor construction for the multivariate case. However, we make the claim that a similar method applies to the multiple factor univariate constructions, and we challenge the reader to work that out in detail.

Theorem V-3.1: (Multiple Factor Generalized Hensel) Let $F(x,y_1,\ldots,y_v)$ be a multivariate polynomial in $(Z[y_1,\ldots,y_v])[x]$. Let $b = (b_1,\ldots,b_v)$ be a given set of integral values and $S = (y_1-b_1,\ldots,y_v-b_v)$ such that $lc(F) \not\equiv 0 \pmod{q, S}$ where q is a given rth power of a lucky prime p for F_b. Assume there exist pairwise relatively prime univariate polynomials $F1_1(x), F2_1(x), \ldots, Fr_1(x)$ in $(Z/q)[x] = (J/S)[x]$, where $J = (Z/q)[y_1,\ldots,y_v]$, such that

$F \equiv F1_1\ F2_1\ \ldots\ Fr_1 \pmod{q, S}$. Then, for any $k > 1$, we can step by step construct $F1_k, \ldots, Fr_k$ in $(J/S^k)[x]$ simultaneously such that $F \equiv F1_k\ F2_k\ \ldots\ Fr_k \pmod{q, S^k}$ and $Fi_k \equiv Fi_1 \pmod{q, S}$ for all $1 \leq i \leq r$.

Proof: Let $F[j|r]_1$ denote $Fj_1\ F[j+1]_1\ \ldots\ Fr_1$ for $j = 2,\ldots,r$. Since $lc(F) \not\equiv 0 \pmod{q, S}$, it must be a unit in Z/q, hence so must $lc(Fi)$ be, for $i=1,\ldots,r$. By Lemma and Corollary II-2.1, we can use Algorithm II-2.1 to find $Ai_1(x)$, $Bi_1(x)$ in $(J/S)[x]$ such that $Ai_1\ Fi_1 + Bi_1\ F[i|r]_1 \equiv 1$ and $\deg(Ai_1) < \deg F[i|r]_1$, $\deg(Bi_1) < \deg(Fi_1)$, for all $i=1,\ldots,r-1$. From these polynomials we will construct, by induction, sequences of polynomials $\{F1_m\}, \{F2_m\}, \ldots, \{Fr_m\}$ such that $F \equiv Fi_m\ \ldots\ Fr_m$ in $(J/S^m)[x]$ and $Fi_m \equiv Fi_1 \pmod{q, S}$ for all $i=1,\ldots,r$.

(Again let $F[j|r]_m$ denote $Fj_m\ F[j+1]_m\ \ldots\ Fr_m$ for $j=2,\ldots,r$.)

For inductive hypothesis, assume for $m \geq 1$ we have

$F_i(x, y_m, \ldots, y_1), i=1,\ldots,r$ in $(J/S_m^{m})[x]$ such that

$F = F_1 F_2 \ldots F_r$. Let

$$R_1(x, y_m, \ldots, y_1) = F - F_1(F_2 \ldots F_r) \pmod{q, S_m^{m+1}}$$

$$= \sum_{I_m} C_{1,I_m}(x)\, M_{1,m}(y_1, \ldots, y_v).$$

For each typical coefficient polynomial, $C_{1,I_m}(x)$, of $R_{1,m}$, we apply Algorithm II-2.2 on $F_{1,1}$, $F[2|r]_1$, $A_{1,1}$, $B_{1,1}$, and C_{1,I_m} in $(Z/q)[x]$ and obtain A_{1,I_m}, B_{1,I_m} such that

$A_{1,I_m} F_{1,1} + B_{1,I_m} F[2|r]_1 = C_{1,I_m}$ and

$\deg(A_{1,I_m}) < \deg(F[2|r]_1)$ in $(Z/q)[x]$. Now let

$$F_1^{(m)} = \sum_{I_m} B_{1,I_m}(x)\, M_{1,m}(y_1, \ldots, y_v),$$

$$F[2|r]^{(m)} = \sum_{I_m} A_{1,I_m}(x)\, M_{1,m}(y_1, \ldots, y_v),\ F_{1,m+1} = F_{1,m} + F_1^{(m)},$$

and $F[2|r]_{m+1} = F[2|r]_m + F[2|r]^{(m)}$. Then

$$F_{1,m+1} F[2|r]_{m+1} \equiv F_{1,m} F[2|r]_m + F_1^{(m)} F[2|r]_1 + F[2|r]^{(m)} F_{1,1}$$

$\equiv F1_m\ F2_m \ldots Fr_m + R1_m \equiv F$ in J/S^{m+1}. We observe that

in J/S^{m+1} $R2_m = F/F1_{m+1} - F2_m \ldots Fr_m = F[2|r]_{m+1} - F[2|r]_m$

$= F[2|r]^{(m)}$. In general (or inductively), assume we have

$F \equiv F1_{m+1}\ F2_{m+1} \ldots F[j-1]_{m+1}\ F[j|r]_{m+1}$ in J/S^{m+1}, then

$Rj_m = F/F1_{m+1}/\ldots/F[j-1]_{m+1} - F[j|r]_m = F[j|r]_{m+1} - F[j|r]_m$

$= F[j|r]^{(m)} = \sum_{Im} Cj_{Im}(x)\, Mj_m(y_1,\ldots,y_v)$. For each $Cj_{Im}(x)$

we can apply Algorithm II-2.2 on Fj_1, $F[(j+1)|r]_1$, Aj_1, Bj_1,

and Cj_{Im} to get Aj_{Im} and Bj_{Im} in $(Z/q)[x]$ such that

$Aj_{Im}\ Fj_1 + Bj_{Im}\ F[(j+1)|r]_1 \equiv Cj_{Im}$. Then let

$$Fj^{(m)} = \sum_{Im} Bj_{Im}(x)\, Mj_m(y_1,\ldots,y_v),$$

$$F[(j+1)|r]^{(m)} = \sum_{Im} Aj_{Im}\, Mj_m(y_1,\ldots,y_v),$$

$Fj_{m+1} = Fj_m + Fj^{(m)}$, and

$F[(j+1)|r]_{m+1} = F[(j+1)|r]_m + F[(j+1)|r]^{(m)}$, we have

$F \equiv F1 \ \ldots \ Fj \ F[(j+1)|r] \ \text{in } J/S^{m+1}$ and

$R[j+1] = F[(j+1)|r]^{(m)}$. Thus we can get $F \equiv F1 \ \ldots \ Fr \ \text{in } J/S^{m+1}$ by making $r - 1$ such iterations where for all $m \geq 1$ we have $Fj = Fj + Fj^{(1)} + \ldots + Fj^{(m)}$ for $j = 1,\ldots,r$ which exactly corresponds to the two-factor case of GHA. This completes the induction and the proof. //

We will not specify the algorithm for this multiple factor generalized Hensel construction in detail since the proof of the theorem is constructive. We will, however, point out that all of the special preparatory computational considerations discussed in Section II-4 still can be applied here. Thus this algorithm is a direct extension of GHA (Algorithm II-4.1). Referring to the analysis of the computing cost of GHA in Section IV-2, we see that because of the simultaneous construction of all factors which results in savings of the multiplication costs (corresponding to Step (5) of GHA), the corresponding total computing cost for this algorithm is approximately

$$\prod_{j=1}^{r} T(Fj)_n + (n/v)^v \left(T(F) + \sum_{j=1}^{r} T(Fj)_n\right).$$

We also point out that the total space requirement for storing preparatory data and intermediate results of this simultaneous construction is essentially the same as that of applying GHA successively r - 1 times. But, if C represents the cost of applying GHA once, then this simultaneous construction effectively improves the cost from (r - 1) C to C which can be most significant if the number of factors of F, r, is large.

V - 4 Factorization of Multivariate Polynomials

The only known implementation of the Hensel type multivariate factorization algorithm is by Wang and Rothschild on the symbolic manipulation system MACSYMA. An outline of such a factorization algorithm is as follows where the main difference here is in Step (4) where we apply the Multiple Factor Generalized Hensel Algorithm instead of using the ordinary GHA r - 1 times. Given a multivariate polynomial

$F(x_1, \ldots, x_v)$ in $Z[x_1, \ldots, x_v]$

(1) By taking content and primitive part of F, we can write $F = \text{cont}(F)\ \text{pp}(F)$ and factor each part separately. By computing the square-free decomposition of F, we can write

$F = F_1 F_2^2 \ldots F_k^k$ and factor each F_i separately then simply

attach the multiplicities of the factors. Thus we will assume that the F to be factored is primitive and square-free.

(2) Choose a set of integers $b = \{b_1,\ldots,b_v\}$ such that $f(x) = F_b(x) = F(x,b_1,\ldots,b_v)$ is also square-free and $\deg(f) = \deg(F)$ in x.

(3) Apply the univariate factoring algorithm described in Section V-2 to find a factorization of f such that $f(x) = f_1(x)\, f_2(x) \ldots f_r(x)$ over Z. In the process of this computation we can also determine a prime p and an integer m such that q = p**(2**m) bounds twice the coefficients of any factor of F and its evaluation at b so that f_i over Z is identical to $f_i \pmod q$ and the f_i's remain pairwise relatively prime modulo q.

(4) Let $S = \{y_1 - b_1,\ldots,y_v - b_v\}$, then we have $F(x,y_1,\ldots,y_v) \equiv f_1(x) \ldots f_r(x) \pmod S$. Apply the Multiple Factor Generalized Hensel Algorithm discussed in the last section on these univariate factors to compute pairwise relatively prime multivariate polynomials $F_i(x,y_1,\ldots,y_v)$, $i=1,\ldots,r$, corresponding to $f_i(x)$ such that

$F = F_1 F_2 \ldots F_r \pmod{S^n, q}$ where n = 1 + the degree bound for F in its non-main variables $y_1, \ldots, y_v$.

(5) From these F_i's, all true irreducible factors of F over Z can be computed by taking combinations, trial divisors, and the restore-leading-coefficient operations.

The cost for factoring a primitive, square-free multivariate polynomial F, then, mainly consists of the costs from Steps (3), (4), and (5). The cost of Step (3) is simply for factoring the univariate polynomial $F_b(x)$ into $f_1(x), f_2(x), \ldots, f_r(x)$ which was discussed in Section V-2. Step (4) involves the application of the Multiple Factor Generalization Hensel Algorithm in the last section. According to the brief analysis of this algorithm, the total cost of this step is proportional to

$$\prod_{i=1}^{r} T(F_i) + (n/v)^v \left(T(F) + \sum_{i=1}^{r} T(F_i)\right).$$

Step (5) is very much dependent on the choices of evaluation points and the prime modulus. The worst case is when F is irreducible but $\pmod{q, S^n}$; it has r extraneous factors. In this case, it will take as many as 2^{r-1} multiplications

of the extraneous factors and trial divisions to confirm the irreducibility of F. In practice, not very many trial operations are required and the true factors of F are quickly found. For each true factor of F, then, it may be necessary to restore the leading coefficient, and that involves a content and primitive part computation on some combination of the potential factors.

Finally, we observe that because of the high cost involved in processing an extraneous factor, especially in Steps (4) and (5), it seems wise and worthwhile to repeat Steps (2) and (3) several times in order to find an evaluation and a prime such that the number of potential factors is minimal.

CHAPTER VI. POLYNOMIAL CONTENTS AND PRIMITIVE PARTS

VI - 1 Introduction

In the previous chapters, we have seen the important parts played by the operation of computing contents and primitive parts of polynomials. Most notably, the first step of both the factorization and the GCD algorithms is mainly performing this operation. To refresh our memories, any polynomial P in J[x], with J being any u.f.d., can be rewritten in a unique representation P = cont(P) pp(P) where cont(P) is the unit normal GCD of all the non-zero coefficients of P and pp(P) = P/cont(P) is the remaining primitive part of P. Specifically, a multivariate polynomial P in $Z[x,y_1,\ldots,y_v]$ will actually be considered and represented in $(Z[y_1,\ldots,y_v])[x]$. That is P is considered to be a polynomial in x with coefficients as polynomials in the other variables. The importance of the content and primitive part operations comes exactly from this requirement of choosing a main variable and a unique representation of polynomials with respect to the main variable. Among the basic arithmetic operations on polynomials, division essentially makes the determination of a main variable mandatory. Factors and GCD's of polynomials are in fact divisors, hence the particular importance of the computation of contents and

primitive parts for these algorithms.

In general, contents and primitive parts are relative operations, i.e. they are operations with respect to a particular variable, cont(P) = cont(P, x) for some variable x. However, there are at least two other kinds of useful notions of contents and primitive parts which are essentially absolute instead of relative to any particular variable. One such is often called the term content, tcont. For an arbitrary polynomial $P(x_1, x_2, \ldots, x_v)$, $\text{tcont}(P) = x_1^{k1} x_2^{k2} \ldots x_v^{kv}$, i.e. the monomial which is the product of the variables each raised to the power which is the minimum of all corresponding variable powers in all the terms of the completely expanded P. In other words, according to Brown in a private communication pp(P) is now normal or it is not divisible by any monomial except 1 or -1. This is quite an easy quantity to obtain from a polynomial P for many different ways of representing a polynomial. Clearly, if a polynomial is represented in a system as a collection of fully expanded terms, then getting its term content simply involves taking minimums of small integers. Another such concept of content is called the super content, scont. This is computed by regular contents recursively on each variable in the polynomial. In other words,

$spp(P(x_1,\ldots,x_v)) = pp(\ldots(pp(pp(P, x_1), x_2),\ldots x_v)$ and

$scont(P) = P/spp(P)$. Super content of a polynomial P contains all factors of P each of which is independent of anyone of the variables in P. Thus spp(P) can only contain factors which have all the variables, or not divisible by any multivariate polynomial which depends on fewer than all v variables in P.

In the following sections of this chapter, we will present an algorithm for computing multivariate polynomial contents and primitive parts using the Hensel construction. We will compute term contents of the coefficients as a very useful preliminary step of computing regular contents with respect to a particular variable. We will only mention here that the computation of super contents, then, simply involves a recursive applications of the algorithm for computing regular contents on each variable.

VI - 2 The EZCONTENT Algorithm

The important observation to be made is that the Hensel construction for the EZGCD Algorithm Step IV, Section III-2 is applied using only one of the two given polynomials. That being the case, there is no reason why the EZGCD Algorithm cannot be generalized to compute the GCD of several polynomials at once. Because the entire preparation

for Hensel construction in EZGCD Algorithm consists of evaluations, univariate GCD computation, and some verification of the conditions, the only difference in handling several polynomials at once is in successively calculating the GCD of several univariate polynomials. Thus we have achieved some parallelism for dealing with several multivariate polynomials, hence the so called semi-parallel process. Since the content of a polynomial w.r.t. a particular variable is simply the GCD of all of the coefficients, we have a straight forward generalization of the EZGCD Algorithm to an algorithm for computing contents. Furthermore, since the GCD verification step of the EZGCD Algorithm (Step V, Sect. III-2) computes the cofactors as a byproduct, we can easily have the corresponding primitive parts as a byproduct of the content computation.

We now give an outline and overview of this generalized EZGCD Algorithm for computing contents and primitive parts, the EZCONTENT Algorithm, similar to those given in Section III-2. The detailed algorithm will clearly be quite similar to the EZGCD Algorithm and will be omitted.

The EZCONTENT Algorithm will compute the content $C'(z,x,y_1,\ldots,y_v)$ and the primitive part $P'(z,x,y_1,\ldots,y_v)$ with respect to the variable x of a given multivariate polynomial $F'(z,x,y_1,\ldots,y_v)$ in $Z[z,x,y_1,\ldots,y_v]$.

Step I: (Term Contents of Coefficients)

Express F' as $F0'(x,y_1,\ldots,y_v) + F1'(x,y_1,\ldots,y_v)\,z + \ldots + Fk'(x,y_1,\ldots,y_v)\,z^k$. For i = 1,...,k, set

Ti <-- tcont(Fi') and $Fi(x,y_1,\ldots,y_v)$ <-- tpp(Fi').

Note that some of the Fi' may be zero, but we assume at least two of the coefficients are non-zero. In the case where only one coefficient is non-zero, the problem is of course trivial, that coefficient is the content and the corresponding power of z is the primitive part.

Compute and set TC <-- gcd(F0,F1,...,Fk). Here, since all the polynomials are actually monomials in $x,y_1,\ldots,y_v$, their gcd is simply the product of variables raised to the minimum of corresponding powers in each monomial. The cofactors corresponding to TC are also easy to compute - by substraction of powers. Thus the remaining task for this algorithm is to compute $C(x,y_1,\ldots,y_v)$ = gcd(F0,F1,...,Fk) and the respective cofactors F0/C, F1/C, ... , Fk/C. Then, the content C' is simply TC * C and the primitive part $P'(z,x,y_1,\ldots,y_v)$ is

$$(T0/TC)\,(F0/C) + (T1/TC)\,(F1/C)\,z + \ldots + (Tk/TC)\,(Fk/C)\,z^k .$$

The reason for computing all the term contents of the coefficients is not only because they are convenient to calculate but also because we can thus eliminate some unnecessary blowup problem due to non-zero evaluation values later. We have noted before that non-zero evaluations cause blowups in the number of terms, hence in the computing cost of the GCD. If we can reduce the degrees of the variables in each coefficient polynomial by this term content operation, then we can also hope to lessen the severity of the blowups when we compute the GCD of the term primitive parts of the coefficient which are normal.

Step II: (Evaluation and Univariate GCD)

Choose a set of v integers $b = (b_1,\ldots,b_v)$ such that the degrees in x of F0, F1, ... , Fk evaluated at $y_i = b_i$ $i = 1,\ldots,v$ are not decreased (i.e. b is valid for F0, F1, ... , Fk).

Compute $F0_b, F1_b, \ldots, Fk_b$ and $C_0 = \gcd(F0_b, F1_b, \ldots, Fk_b)$.

This univariate GCD can be computed using UNIGCD or Modular GCD Algorithms serially or successively on pairs of polynomials, e.g. $\gcd(\ldots(\gcd(F0_b, F1_b),\ldots),Fk_b)$. If UNIGCD is used, then a similar generalization of that algorithm will enable it to handle several polynomials at once. However,

even in that case the gcd of several modular images would have to be computed serially, so that our parallel computation will eventually end in a serial calculation in a subdomain. That is why we only call this a semiparallel process.

Step III: (Preparation for Hensel Construction)

If $\deg(C_0(x)) = 0$, then $C_0(x) = 1$ and $C = 1$. If F_j is the polynomial of minimum degree in x among $F_0, \ldots, F_k$ and $\deg(C_0) = \deg(F_j)$, then either F_j divides all other polynomials or a new valid evaluation should be made. Otherwise, determine if the following condition holds:

Condition III - A: There exists a j in $\{0,1,\ldots,k\}$ such that $\gcd(C_0, F_j/C_0) = 1$.

If the condition fails to hold on any of the polynomials, then the special case method which is also similar to that of the EZGCD Algorithm will have to be used. The difference between handling two or several polynomials for the special case is again so small that we will omit presenting the method here.

Assuming Condition III-A holds for some j, we continue under the further assumption,

Condition III - B: $\deg(C_0(x)) = \deg(C)$.

Again similar to the EZGCD Algorithm, this C is supposedly unknown so that this condition cannot be tested at this point. However we will endeavor to increase the probability of having it hold, at various places of the algorithm. In addition, Step V will provide a safety-valve test to ensure that any unlucky evaluation, which causes Condition III-B to be false, will be detected.

Step IV: (Application of the Hensel Construction)

Suppose Fj satisfies condition III-A. Apply the Multivariate EZ Algorithm (II-5.4) on Fj, $C_b(x)$, and $F_j(x)/C_b(x)$ to get either

(a) multivariate cofactors, C" and H", of Fj such that

$F_j = C'' H''$ over Z, or

(b) some q_k, n, C_n, and H_n such that

$$F_j \equiv C_n H_n \pmod{q_k, S_n}.$$

Step V: (Verification of Results)

For case (a), test whether C" divides the other polynomials Fi not equal to Fj. If so, $C = C''$ is the GCD we seek, so that $C' = TC\ C$ is the content we seek and the primitive part can be computed by multiplying quantities already computed. Otherwise, or for case(b), go back to Step II for a new evaluation and a resulting

univariate GCD having a smaller degree than this C_0.

The theoretical justifications for this algorithm has esentially already been provided by Theorems III-2.2 and III-2.3. Therefore, we only refer to them keeping in mind the generalization to considering several given polynomials at once.

VI - 3 Computing Cost Estimation and Conclusions

Because of the similarities of the EZCONTENT and the EZGCD Algorithm, the cost of computing the content and primitive part of a v + 2 variable polynomial $P(z,x,y_1,\ldots,y_v)$ is about the same as computing the GCD of some v + 1 variable polynomials. For this algorithm, assumptions (ASM1) - (ASM4) will also be made, except that (ASM3) should be changed to

(ASM3') The cost of computing GCD's for polynomials of v variables or less is negligible compared to the cost of computing the GCD and cofactors of term primitive parts of the v + 1 variable coefficient polynomials.

Under these assumptions, we get a computing cost estimate for this algorithm parallel to (E3) of the EZGCD Algorithm:

T(sub((cl"/cl) C) T(sub(cl H)) +

$[T(cl'' Fj) + T(sub((cl''/cl) C) + T(sub(cl H))] (n/v)^{v}$

where C = gcd(F0, F1, ... , Fk), cl = lc(C), cl'' is the GCD of the leading coefficients of F0, F1, ... , Fk, H is the co-divisor of C in Fj which is one of the Fi's satisfying Condition III-A and used for the Hensel construction, and n is the maximum term degree for cl'' Fj.

If the additional assumptions (ASM5) and/or (ASM6) are made, we also get formulas similar to (E4) - (E6) for the EZGCD Algorithm. But, more importantly, we also get an expression similar to the well substantiated formula (E7) as the proportional computing cost for the EZCONTENT Algorithm:

T(sub((cl''/cl) C) T(sub(cl H)) $\oplus$ T(sub(cl'' Fj))

With this formula, we can conclude that the cost for computing content and primitive part of a polynomial is roughly proportional to the number of terms in the smallest coefficient polynomial. This interpretation, in essence, gives validity to assumption (ASM3) for the EZGCD Algorithm.

After the many cases of empirical examples shown in Section IV-3, one can readily observe that the computing time for contents and primitive parts using the EZCONTENT Algorithm can be much faster than the more usual ways of serially applying other GCD algorithms. Thus we will not attempt to create separate examples to re-emphasize this

point by exhibiting additional experimental data.

The definition for GCD of several polynomials (Section I-2) makes no stipulation that this GCD should be computed successive'y and pairwise. In fact, according to the definition it is more natural the other way, because the GCD is simply the greatest common divisor of all the polynomials taken simultaneously rather than taken in a specific order. We hope, our presentation of the EZCONTENT Algorithm further demonstrated and emphasized this view point. The polynomial content operation is an important example of the need for parallel GCD computations and the importance of the primitive parts shows the usefulness of getting the cofactors of a GCD as byproducts.

CHAPTER VII SQUARE-FREE DECOMPOSITIONS

VII - 1 Introduction and Overview of EZSQFR

Square-free decompositions of polynomials have many uses. Among them, the most important ones are in polynomial factorization and partial fraction decomposition of rational functions (refer to Appendix for definitions). These are, in turn, indispensable tools of rational function integrations [MOS67], [MOS71], [HOR71], [RIS69]. Horowitz [HOR71] made a quite complete survey and analysis on computing square-free decompositions, partial fraction decompositions and rational function integrations for the univariate case. Many existing symbolic manipulation systems such as MACSYMA [MAC73] and SAC-1 [HOR69] also contain algorithms for computing square-free decompositions of multivariate polynomials. Musser [MUS71] presented a slightly improved version of the above mentioned more classical algorithm. We will now present brief outlines of these square-free decomposition algorithms for multivariate polynomials and then, in parallel, give a rough overview of the new EZSQFR Algorithm which uses the Hensel construction.

Given a multivariate polynomial $P(x,y_1,\ldots,y_v)$ in $Z[x,y_1,\ldots,y_v]$, which will be assumed primitive w.r.t. x,

the following square-free decomposition algorithms will compute square-free polynomials $P_i(x,y_1,\ldots,y_v)$, $i=1,\ldots,k$, (some of them equal 1) such that $P = P_1 P_2^2 \ldots P_k^k$, for some $k \geq 1$ and $P_k \neq 1$.

First we give a simple and direct proof of a fundamental fact:

Theorem VII - 1.1: If $P = P_1 P_2^2 \ldots P_k^k$ is primitive in x then $\gcd(P, dP/dx) = P_2 P_3^2 \ldots P_k^{k-1}$.

Proof: Let $D = \gcd(P, dP/dx)$. For an arbitrary P_i write $P = P_i^i Q$, then

$dP/dx = i P_i^{i-1} (dP_i/dx) Q + P_i^i Q (dQ/dx)$, i.e. P_i^{i-1} divides dP/dx. Thus $\prod_{i=2}^{k} P_i^{i-1}$ divides D. On the other hand,

$dP/dx = \prod_{i=2}^{k} P_i^{i-1} \left(\sum_{i=1}^{k} i (dP_i/dx) \prod_{j \neq i} P_j \right)$, P_i^i does not divide dP/dx, since P_i divides every term in the summation

except one, i (dP /dx), because P is square-free and
i i

primitive w.r.t. x. Thus $P_2 P_3^2 \cdots P_k^{k-1}$ is actually the

greatest common divisor.

(i) Classical Method (Horowitz and Tobey):

Set $D \leftarrow \gcd(P, dP/dx)$, $P_0 \leftarrow D$, $i \leftarrow 1$.

Loop: Set $P \leftarrow D$, $D \leftarrow \gcd(P, dP/dx)$, $P_i = P_{i-1}/P/D$,

$i \leftarrow i + 1$, go Loop unless $D = 1$.

(ii) Improved Algorithm (Musser):

Set $C_1 = \gcd(P, dP/dx)$, $D_1 \leftarrow P/C_1$, $i \leftarrow 1$.

Loop: Set $D_{i+1} \leftarrow \gcd(C_i, D_i)$, $C_{i+1} \leftarrow C_i/D_{i+1}$,

$P_i \leftarrow D_i/D_{i+1}$, $i \leftarrow i+1$, go Loop unless $D_i = 1$.

(iii) EZSQFR (for primitive polynomials):

Set $D \leftarrow$ EZGCD(P, dP/dx) and $L \leftarrow P/D$.

Let $b = (b_1, \ldots, b_v)$ be the lucky evaluation for P used

in the above GCD computation. Set $i \leftarrow 1$.

Loop: Set $H_0 \leftarrow \text{igcd}(L_b(x), D_b(x))$, $G_0 \leftarrow L_b/H_0$,

$D_b \leftarrow D_b/H_0$, and $L_b \leftarrow H_0$. If $G_0 = 1$, then

set $P_i \leftarrow 1$; otherwise $G_0 \neq 1$, then apply the

Multivariate EZ Algorithm on L, G_0, and H_0 to

get multivariate G and H such that L = G H, and

set P_i <-- G and L <-- H. Set i <-- i + 1. If

$D_b(x) \neq 1$, go Loop, else set P_i <-- L and return

$P_1, P_2, \ldots, P_k$ where k = i.

Upon more careful examination of these three algorithms, it is quite clear that the difference between algorithm (i) and (ii) is rather small. Some unnecessary differentiations are done in (i), and the computations of GCD's in (ii) in general involves smaller polynomials than in (i). The GCD computations in the Loop of (i) and (ii) all involve multivariate polynomials. They can be very time consuming. If polynomial GCD Algorithms, such as the Modular and EZGCD Algorithms which compute the cofactors as byproducts, are used, then both (ii) and (iii) involves no polynomial divisions at all where as (i) has to do one division for each P_i. In addition, the most important advantage that EZSQFR Algorithm has over the other two methods is that, except the first GCD computation, all other GCD's are univariate operations. In fact, within the Loop of EZSQFR Algorithm, the only multivariate operations are done on L in the application of the Multivariate EZ Algo-

rithm for constructing G and H. Therefore we will now discuss this EZSQFR Algorithm for computing multivariate polynomial square-free decompositions in more detail.

VII - 2 The EZSQFR Algorithm

From the outline of the last section, it is clear that the EZSQFR Algorithm is also based on the evaluation homomorphism and the Hensel construction. As we have seen for the EZGCD Algorithm in Chapter III, it is quite obvious that the modular homomorphism and the Zassenhaus' Quadratic Extension Algorithm can be used for computing square-free decompositions of univariate polynomials, similar to the multivariate situations. Again, we will not discuss the univariate case in any detail, due to the ample similarities between the two cases.

For the multivariate EZSQFR Algorithm we will first discuss a useful, timesaving device for the computation of square-free decompositions, due to a suggestion by Wang. A given multivariate polynomial could already be square-free with respect to a particular variable. If this fact can be detected, at a relatively small cost, before the entire machinery of the square-free algorithm begins to work, there can be big savings in computing time. There is such a time saving test which we shall call <u>fail-safe square-free test</u> (f.s.s.f.). This test consists of (a) evaluating the

polynomial at random by chosen valid points. (b) computing the GCD of the resulting univariate polynomial and its derivative w.r.t. the main variable. (c) checking to see if the GCD is a integer or not, if so then the original polynomial is square-free. The validity of this test is shown by the following lemma:

Lemma VII - 2.1: Let $b = \{b_1,\ldots,b_v\}$ be an arbitrary valid evaluation for $P(x,y_1,\ldots,y_v)$ in $Z[x,y_1,\ldots,y_v]$. If $P_b(x) = P(x,b_1,\ldots,b_v)$ is square-free in $Z[x]$ then P is itself square-free.

Proof: If P is not square-free, $P = Q R^2$, then via the evaluation homomorphism $P_b = Q_b R_b^2$ which cannot be square-free. //

The usefulness of this test comes from the next lemma, which is also stated and proved by Wang and Rothschild [W&R73]:

Lemma VII - 2.2: If P is a square-free multivariate polynomial in $Z[x,y_1,\ldots,y_v]$, then a set of integers $b = \{b_1,\ldots,b_v\}$ can be chosen so that b is a valid evaluation for P (i.e. $\deg(P) = \deg(P_b(x))$ and $P_b(x)$ is also square-free.

Proof: Let $P = P_1 P_2 \ldots P_k$ be the factorization of P into irreducible factors where $\gcd(P_i, P_j) = 1$ for all i and j such that $i \neq j$. $P_b(x)$ is square-free if and only if $P_{ib}(x)$ is square-free for all i and $\gcd(P_{ib}(x), P_{jb}(x)) = \text{constant}$, for all i, j, $i \neq j$, which is equivalent to $\text{reslt}(P_{ib}(x), P_{jb}(x)) \neq 0$, for all i, j, $i \neq j$, where "reslt" denotes the resultant w.r.t. x [VDW49].

Let $R(x, y_1, \ldots, y_v) = \prod_i \text{reslt}(P_i, dP_i/dx) \prod_{i<j} \text{reslt}(P_i, P_j)$.

Then $R \neq 0$ since P is square-free and $P_b(x)$ is square-free if and only if $R_b(x) \neq 0$. Now it suffices to note that there are only finitely many integral values for $b_1, \ldots, b_v$ such that $R_b(x) = 0$ or $(\text{lc}(P))_b = 0$. //

The finiteness of integral roots for $R(x, y_1, \ldots, y_v)$ has the further implication that out of the infinite possibilities of integral values for each b_i, the probability for choosing a valid but unlucky set b, such that $P_b(x)$ becomes not square-free when P is, cannot be too large. We will apply this test at the beginning of the EZSQFR Algorithm with at most two valid evaluations. If the test fails, we

simply assume no information was gained and continue with the rest of the EZSQFR Algorithm. However, we point out that not all the work done for the test was wasted in this case, since the computations done for the f.s.s.f. test is useful for the EZGCD Algorithm when it is used to compute the square-free part of the given polynomial (the D of (iii) in the last section).

Next, we will prove Lemma (S1) already stated in Section III-4 which will eliminate the need for using the special case method of the EZGCD Algorithm when it is invoked to compute D = gcd(P, dP/dx).

Lemma VII - 2.3: (same as III-(S1))

Let P be primitive in $(Z[y_1,\ldots,y_v])[x]$ and

R = gcd(P, dP/dx), then gcd(R, (dP/dx)/R) = 1.

Proof: Let $P = P_1 P_2^2 \ldots P_k^k$ and Q = dP/dx. By Theorem VII-1.1, $R = P_1^1 P_2^2 \ldots P_k^{k-1}$. Assume D = gcd(R, Q/R) $\neq$ 1, then Q = R D Q" for some Q". Since D divides R, there must be an irreducible factor C $\neq$ 1 of some P_i, $i \geq 2$. But this C clearly also divides P/R so that P = R C P" for some P". Thus, R C divides both P and Q, contradicting to R being the GCD. Therefore, gcd(R, Q/R) must be 1. //

This lemma and the finiteness of the number of unlucky evaluations imply that Condition III-A of the EZGCD Algorithm can always be satisfied for the polynomial dP/dx. Hence, only Steps (A1) - (A9) of the EZGCD Algorithm will be used for computing gcd(P, dP/dx) and the special case method Steps (S1) - (S7) can be completely avoided for this square-free decomposition algorithm.

We are ready now to describe the EZSQFR Algorithm in detail. However we will not be so careful as to indicate, for example, how the computations for f.s.s.f. test is used in the EZGCD Algorithm or what is altered in the EZGCD Algorithm when it is known that Condition III-A will hold for one input polynomial eventually. These changes are actually very simple if we carefully look at the EZGCD Algorithm again.

If the given polynomial is not primitive, then clearly we can work on its content and primitive part separately. Because of the fact that square-free decompositions are main variable dependent and that for some uses it is not necessary to square-free decompose the content, we will assume primitive inputs to the EZSQFR Algorithm.

Algorithm VII - 2.4: (EZSQFR)

Input: A primitive multivariate polynomial $P(x,y_1,\ldots,y_v)$ in $Z[x,y_1,\ldots,y_v]$ and the main variables, x.

Output: A list of polynomials in $Z[x,y_1,\ldots,y_v]$,

$P_1, P_2, \ldots, P_k$, such that $P = P_1 P_2^2 \ldots P_k^k$ for some $k \geq 1$

and $P_k \neq 1$.

(1) Invoke the f.s.s.f. test twice with random valid evaluations. If the test is successful, then return P, otherwise continue.

(2) Apply the EZGCD Algorithm on P and dP/dx to obtain D = gcd(P, dP/dx) and L = P/D. Let b be the valid lucky evaluation for P and P be the lucky prime for $P_b(x)$ used in the EZGCD Algorithm above.

(3) Set $Lb \leftarrow L(x,b_1,\ldots,b_v)$, $Db \leftarrow D(x,b_1,\ldots,b_v)$, and $i \leftarrow 1$.

(4) Apply UNIGCD Algorithm (or some other univariate GCD Algorithm which also computes cofactors) on Lb and Db to obtain $H0 \leftarrow gcd(Lb, Db)$, $G0 \leftarrow Lb/H0$, and $Db \leftarrow Db/H0$. Set $Lb \leftarrow H0$.

(5) If G0 = 1, then set $P_i \leftarrow 1$. Otherwise, apply the Multivariate EZ Algorithm on L, G0, and H0 to get multivariate polynomials G and H such that L = G H over Z. Set $P_i \leftarrow G$, $L \leftarrow H$, and $i \leftarrow i + 1$.

(6) If $Db \neq 1$, then go to (4). Otherwise, set k <-- i, P_k <-- L, and return $P_1, P_2, \ldots, P_k$.

Remark: Note that even the evaluation of multivariate polynomial L and D in Step (3) need only be done once. The later values for Lb and Db are simply updated with known quantities. Step (4) involves only one univariate GCD computation. For the application of the Multivariate EZ Algorithm in Step (5), the required lucky evaluation b and prime p is again provided by the EZGCD Algorithm in Step (2). Thus the results of that computation are certain to be correct over Z. Also, it is possible to avoid the leading coefficient blowup problem by applying, instead of the Multivariate EZ Algorithm, Algorithm II-5.5 in a way similar to that of the EZGCD Algorithm in Section III-4(a). Here we can use the successively decreasing leading coefficient of L as the leading coefficient to be forced onto G0. That is instead of using L, G0, and H0, we use (lc(L), L), $(lc(L))_b$ (G0/lc(G0)), and lc(G0) H0 together with lc(L) as the leading coefficient replacing leading coefficients of both univariate polynomials.

VII - 3 Conclusions

It is quite obvious that the EZSQFR takes great advan-

tages of computations already performed to achieve added efficiencies. By performing Hensel constructions instead of full-fledged multivariate GCD computations, the gains in efficiency is clear. Even the preparations for these applications of the Multivariate EZ Algorithm are very simple. They involve essentially only one univariate GCD computation as preparation for each Hensel construction, hence for each P_i. That is much less costly than what EZGCD Algorithm goes through for preparing Hensel construction. All the choosing of lucky evaluations and primes, testing for various conditions et..., are eliminated because the evaluation values and prime are known to be lucky after the first call to EZGCD.

Whenever $P_i = 1$ for some i (for example, $P = U V^3$, then $P_1 = U$, $P_2 = 1$, $P_3 = V$), the computational process is even simpler - only one univariate GCD computation is necessary and no multivariate operation is required at all. Compared to method (ii), we would find, in this case, $D_{i+1} = \gcd(C_i, D_i) = D_i$, so that $P_i = D_i/D_{i+1} = 1$, i.e. D_i divides C_i. Even so, there is still at least one multivariate division involved, which is more time consuming than a univariate GCD computation in most cases. In conclusion, the EZSQFR Algorithm contains all the time saving devices in

the other two methods and several more in addition. Therefore, the decrease in computing costs should be rather obvious. We will not illustrate this with empirical tests but only re-emphasize the following facts: (1) Hensel constructions have been shown to be an efficient computational technique whenever applicable for practical multivariate polynomials, (2) the EZSQFR Algorithm needs only one call to EZGCD, no multivariate divisions at all, and several Hensel constructions replacing complete multivariate GCD computations, and (3) empirical data in Section IV-3 have shown clearly how the Hensel approach to GCD computations can be advantageous over other methods.

CHAPTER VIII. CONCLUSIONS AND SUGGESTIONS FOR FUTURE RESEARCH

VIII - 1 Summary and Conclusions

At this point, we hope that the usefulness of the Hensel-type constructions in symbolic and algebraic manipulation has been made clear. We have shown how far a relatively simple idea in p-adic analysis can go toward improving the performance of algebraic algorithms. A great deal of effort has been given, in Chapter II, to clarify this fundamental concept and its many generalizations. The basic idea in these Hensel constructions is to recover the co-divisors of a given polynomial in a higher domain from their algebraically independent images in some more structured subdomain. Many interspersed examples have been given to help in understanding these Hensel-type algorithms. The Generalized Hensel Algorithm for multivariate constructions brought back a seldom used but very appropriate way of viewing a multivariate polynomial - as a generalized Taylor series. Actually, that is also a natural generalization of the p-adic representation for integers. The only difference is that, instead of using a integral prime as the basis for representation, we use the degree-one irreducible polynomials of the form (y - b), with some chosen integral evaluation point b for the variable y. Although we have noted

the difficulties related to transforming the polynomials from one representation to the other, we also have realized the gains in efficiencies for many algebraic algorithms when the Taylor series forms are used. The major emphasis to be made here is that only one set of evaluation values are necessary for these suitable applications of the GHA when these integral values are well chosen, and that is exactly the reason for the efficiencies of the algorithms described in this thesis.

The main application of the Hensel construction is in the problem of computing polynomial greatest common divisors. This is a very well-studied topic ever since the time of Euclid (300 B.C.). In Chapter I, we pointed out the major drawbacks of known methods for computing GCD's. We noted that the method based upon the recently quite popular modular technique - the Modular GCD Algorithm, actually only gave us a "worst case" algorithm. That is, it essentially considers all polynomials to have the biggest size possible - dense up to the same maximum degree in each variable. Computational experience tell us that this is a very unrealistic and impractical assumption for GCD calculations.

Rational function manipulation and simplification, which is the basis many powerful recent symbolic manipulation systems, in most cases depends on the computation of polynomial GCD's. The storage capacity and speed limita-

tions of present-day computers impose bounds on the sizes of expressions for which any non-trivial computations can be done. As far as GCD computations are concerned, we saw in Section IV-3 that the largest dense polynomials, for which their GCD's can be computed within reasonable time periods, have on the order of 2000 terms and those are only dense quadratic polynomials in four variables. Thus, if every GCD computation assumes dense polynomials, it would be very difficult to do any non-trivial calculations with multivariate polynomials of high degrees with a symbolic computing system. Fortunately, most multivariate polynomials of any high degree are sparse. The EZGCD Algorithm based on the Hensel construction takes advantage of the sparseness of polynomials whenever possible and achieves remarkable efficiencies for many classes of problems. Some of the empirical test cases of Section IV-3 showed this efficiency very well. We should also point out that the MACSYMA symbolic manipulation system has been able to perform many previously nearly impossible (in time and space) computations with the newly implemented EZGCD Algorithm.

With these computational experiences and the test case data listed in Section IV-3, we were able to verify the usefulness and correctness of the EZGCD Algorithm with analytically derived computing cost estimation formulas. Chapter IV shows the accuracy of the predictions that we were able

to make with the intuitively derived expressions for computing costs. Because of these accurate predictions of actual run times, our confidence in this more intuitive (more "engineering") approach to computing time analysis has been strengthened. As already noted, Gentleman [GEN73] has independently arrived at a similar conclusion that an engineering approach to computing time analysis is desirable for complex algorithms. We can, therefore, conclude that in the absence of unlucky evaluation values and non-trivial leading coefficients causing severe blowup problems due to non-zero substitutions, and when special case considerations are not necessary, the EZGCD Algorithm generally surpasses the Modular GCD Algorithm in performance. Even in those cases where the Modular Algorithm is supposed to be at its best, such as when the GCD is 1 or for moderately dense situations, the Hensel-type algorithm can perform equally well. The only cases where the Modular GCD algorithm should theoretically win, are the extremely large and dense cases where the multiplication cost becomes a significant part of or even dominates the cost of Hensel construction. But, as the data in Section IV-3 showed clearly, such problems soon become prohibitively large so as to be impractical for computation by any method. So, we should concentrate mainly on problems of a more practical size. In this size range, it is possible for the PRS GCD algorithms to be more effi-

cient than the EZGCD Algorithm in small problems where the number of elements in the PRS sequence or the number of pseudo-divisions is very small (e.g. 1 or 2 divisions) so that it is impossible for large blowup to occur due to large pseudo-division multipliers. Since a large number of GCD problems deal with small polynomials (because the computation of GCD's and contents is essentially a recursive process on the variables), we believe that a combination of the Reduced (or Subresultant) PRS GCD Algorithm and the EZGCD Algorithm will be very advantageous. For example, we could test if the minimum of degrees of the two input polynomials is less than or equal to 2, recursively, each time the main variable changes. If so, the PRS GCD Algorithm can be used since at most two pseudo-divisions need to be performed in this case. Otherwise, we would apply the EZGCD Algorithm. In this manner, we can take advantage of the best aspects of both algorithms and achieve added efficiency.

The drawbacks of the EZGCD Algorithm should also be clearly recognized. First, there is the problem of unlucky primes and evaluations. But this problem is usually less severe for the EZGCD Algorithm than for the Modular GCD Algorithm because the EZGCD Algorithm only needs one lucky prime and one lucky evaluation value for each non-main variable. With careful screening of the chosen integral

values and several intermediate tests for the luckiness, as we have designed into the algorithm, this problem can be largely avoided at a very small cost. Next, there is the problem of the desired GCD being not relatively prime to either of its codivisors in the two given polynomials (i.e. Condition III-A of Section III-2 can not be satisfied). For that we have designed the special case method which still uses the Hensel construction but in a more indirect way. Although this method is slower by being round-about, it turns out that it still performs quite competitively by taking advantage of the special structures of the given polynomials, as Case 7 of Section IV-3 bears out. In such a case, the EZGCD Algorithm has the blowup problem due to non-trivial leading coefficient when applying the Hensel construction. As discussed in Sections II-5(a) and III-4(a), this problem is caused by the non-uniqueness of the solutions to Diophantine Univariate Polynomial Equations (DUPE) which results in arbitrary multiplications of the codivisors by units in the modular domain. This phenomenon is actually common to most modular algorithms and our solution for this problem for the GCD computation is somewhat similar to the way of getting around this problem used by the Modular GCD Algorithm. At the cost of computing the GCD of leading coefficients of the given polynomials and using the product of this GCD and one original polynomial (this is what the

Modular GCD Algorithm also uses) for the GHA, we can avoid this blowup problem by forcing the results from the Hensel construction to be uniquely determined. Finally, we have the yet unresolved blowup problem due to the combination of the requirement for valid and lucky evaluational values and the need for making a substitution transformation on the given polynomial for the Hensel construction. When the evaluation values are forced by the validity and luckiness requirements to be non-zero, the substitution transformation can increase the number of terms in the polynomial for GHA drastically (up to that of the dense polynomial with the same maximum degree in each variable, of course). This is the so called the non-zero substitution problem for which a generally applicable solution is still lacking. We point out that there are various ways for avoiding such blowup problems in some special situations. But, because of their lack of general applicability, we will not discuss them in this thesis. We will emphasize that even assuming the worst such blowup (to dense polynomials), the computation times for practical sized problems are still competitive with those of the Modular GCD Algorithm, as we can see from Case 5' in Section IV-3 and by verifying with the corresponding computing time formulas for both algorithms.

The last three chapters dealt with three other applications of the Hensel constructions. The application of

Hensel construction to the problem of factoring polynomials was actually the initial work in this direction. However the multiple factor generalized Hensel construction in Section V-3 can provide a noticeable improvement to the multivariate polynomial factorization algorithm when the number of potential factors is bigger than two. The next two applications of Hensel construction are actually out growths from the EZGCD Algorithm based essentially on two key observations while designing and implementing the GCD algorithm. Realizing that the computation of contents is just the same as computing the GCD of several polynomials together and that the application of the Hensel construction needs only one of the given polynomials, we can immediately extend the EZGCD Algorithm to compute polynomial contents and primitive parts. The square-free decomposition algorithm - EZSQFR is a direct extension of the special case method of the EZGCD Algorithm for handling non-square-free polynomials. The key observation in this case is Lemma VII-2.3 which ensures the applicability of the Hensel construction to the problem of computing square-free decompositions for all polynomials. A careful study of the known square-free decomposition algorithms revealed several inefficiencies of these methods which the EZSQFR Algorithm can avoid. These computational improvements make the EZSQFR Algorithm unquestionably more efficient than the other known methods.

VIII - 2 Prospects and Suggestions

The vast possibilities of applying Hensel-type methods are certainly not exhausted by this thesis. The univariate GCD algorithm discussed in Section III-3 has already provoked active study and analysis of other major GCD algorithms for univariate polynomials. Many interesting results have been discovered by comparing GCD algorithms using the ordinary Hensel's construction (Algorithm II-3.1) and Zassenhaus' Quadratic Extension Algorithm (II-3.2) with other known methods. A paper by Yun and Miola reporting these activities is under preparation.

A linear version of the Hensel construction is manifested in the problem of solving the polynomial equation $A F + B G = H$ with F, G, and H being the given polynomials and A, B the unknowns. The simplest such problem is described in Lemma II-2.2 and solvable by Algorithm II-2.2. Under suitable conditions, such equations can even be solved for multivariate polynomials over the integers. Work in this aspect of utilizing Hensel-type methods has already been initiated and will be further pursued by Yun in the near future. Two immediate major applications of this work lie in the problems of polynomial division with remainders (or pseudo-division) and in partial fraction decomposition of rational functions. We will only point out here that,

the main variable is quite arbitrary. As we can clearly realize in GCD computations, however, the choice of a main variable can greatly affect the computing time. With respect to the EZGCD Algorithm, for instance, different main variables result in different leading coefficients hence in different lucky evaluation values and more or fewer zeros for the substitution transformation. Up to now, the choice of main variable seems to be more of an art than a science. Thus, it will be a very worthwhile undertaking to see if a more methodical way of choosing a main variable can be found for the EZGCD Algorithm, the other GCD algorithms, and other polynomial operations.

(3) The multiplication of multivariate polynomials seems to be far from being well understood. For dense polynomials, the method of fast finite Fourier transform achieves the currently known minimal theoretical computing time - T(P*Q)log(T(P*Q)) [BON73]. For the completely sparse case, the best known method has a computing time proportional to T(P*Q)log(T(P)) (ALTRAN [BRO72]) when sorting of terms in the product into a canonical representation is also considered. Furthermore, we believe that a even better method (cost proportional to T(P) T(Q)) can be found if the ordering of terms in both input polynomials are taken into consideration. Then, what about intermediately sized polynomial multiplications? Are these known methods actually

optimal? Certainly, an understanding of polynomial multiplication is extremely important for the algorithmic studies of more complex algebraic operations.

(4) Polynomial division is equally as important as multiplication. As mentioned above, a variation of the Hensel construction already provides a new way of performing a division with remainder. A byproduct of that method is a fail-safe way of doing test-division which is an important verification step of the EZGCD and the Modular GCD Algorithms.

(5) Polynomial divisions and pseudo-divisions lead naturally to PRS sequences for computing GCD's and resultants. There are still many unanswered questions regarding PRS algorithms. For instance, is the Subresultant PRS GCD Algorithm the best one of its kind? Section 3.7 of [BR071] answers this negatively, but some theoretical and algorithmic details have yet to be worked out. A non-trivial improvement to the Subresultant PRS GCD Algorithm was discovered and presented by Hearn in 1972 [HEA72]. It has been shown by the EZGCD Algorithm that the Hensel-type construction works quite well in GCD computations. Can a similar Hensel-related method be used for the computation of polynomial resultants?

(6) As a matter of habit from algorithmic analysis in complexity theory, we have been taking the cost of polyno-

mial (versus integer, in complexity theory) multiplication and division as the basic operations to be accounted for in algebraic algorithmic analysis. Our more intuitive method of analysis has hopefully shown that complex polynomial (especially multivariate) operations often involve many other equally important basic operations which also contribute significantly to the total computing cost. In the case of the Hensel construction (GHA) and the EZGCD Algorithm, we were able to pin point one particular parameter, the number of terms, as the major variable on which the total computing cost depends. Can one find such a parameter for other polynomial algorithms? The work that remains to be done in this respect is to establish some order in this complex environment by finding the relative cost differences among several important basic operations. For example, the cost of forming the product of two polynomial terms in relation to comparing two terms and determining their relative positions in some canonical ordering of polynomial terms. Once such a ranking of cost units for major relevant basic operations is obtained, it will be much easier to perform accurate computing time analyses and to pin point important cost-affecting parameters. The ultimate goal for timing analysis should be predictive accuracy for computational algorithms with realistic problems.

(7) A very simple but generally useful substitution

operation is necessary for the Hensel construction. We found out that the more straight forward method of substituting term by term is actually faster than that by Horner's rule, contrary to the general impression due to the efficiency of the Horner's rule in polynomial evaluations [HOR73]. Thus, further understanding of and a better method for substitution of polynomials into polynomials still seems to be in demand. Related to the substitution problem and, of course, to the Hensel construction as well as the EZGCD Algorithm, is the open problem of blowup due to non-zero substitutions mentioned again in the last section. If this problem can be successfully avoided, then the class of problems where the EZGCD performs well can be greatly enlarged.

(8) The use of the Hensel construction in polynomial factorization and GCD computations theoretically requires a bound on the integral coefficients of the factors or divisors of the given polynomials. Although the computational algorithms easily and successfully get around this need by calculating only a heuristic bound supported by some final tests for the sufficiency of this bound, it is still highly desirable to be able to compute a good true bound at the start. Knuth [KNU69] gives several good bounds for the factors of univariate polynomials. But for multivariate polynomials, his methods are either not generalizable or too

time consuming. Other quickly computed absolute bounds for multivariate polynomial factors, such as the one given by Musser [MUS73], are unrealistically large. So we desire a quickly computable tight bound for the integral coefficients of factors of arbitrarily given multivariate polynomials.

(9) Now that the Hensel construction makes factorizations for polynomials a computationally feasible task, it appears reasonable to expect to encounter more and more factored polynomials in symbolic computations in the future. Factored polynomials are in general smaller, more comprehensible, and give more insight. However, we have been so accustomed to expanded representations of polynomials that little is known about how to perform such a basic operation as division efficiently when either or both polynomials are in factored forms. Work on this aspect will definitely be awaited with great anticipation.

APPENDIX: BASIC ALGEBRAIC CONCEPTS AND ESSENTIAL NOTATIONS

In this section, we will review some basic concepts in modern algebra [HER64] [B&M65], which will be used freely throughout this thesis. We also establish some notations which will be essential to facilitate our expositions.

In an algebraic ring, units are simply divisors of unity or elements having inverses in the domain, zero-divisors are divisors of zero. A commutative ring is an integral domain if it has no zero-divisors. In a field, all non-zero elements are units; in the domain of integers, denoted by Z, the only units are 1 and -1. Elements of an integral domain that divide each other are called associates.

A binary relation is called an equivalence relation if it satisfies reflexive, symmetric, and transitive properties. Equivalence relations divide a domain into equivalence classes and it is often convenient to choose one simple element out of each equivalence class as a canonical representative and define it to be unit normal. The relation of associativity is an equivalence relation and decomposes the integral domain into associate classes.

For a given element q in an integral domain I, the relation "$\equiv$", congruence modulo q, defined by a,b in I is $a \equiv b \pmod{q}$ iff (if and only if) q divides a-b, is an

equivalence relation. We will let (Z/q) denote the set of equivalence classes with respect to the congruence relation modulo some positive integer q. Z/Z will denote the field of rational numbers.

An element of an integral domain is said to be irreducible if its only divisors are units and associates. An integral domain in which every element can be represented uniquely (up to associativity) as a product of irreducibles is called a unique factorization domain (u.f.d.).

Let J be a unique factorization domain, then J[x] is also a u.f.d. [HER64] whose units are units of J. In J[x], we assume that the terms of a polynomial P are arranged in the order of decreasing exponents,

$$P = c_n x^n + c_{n-1} x^{n-1} + \ldots + c_1 x + c_0 .$$

The coefficient of the highest non-zero power of x is called the leading coefficient, lc(P), and the coefficient of the lowest non-negative power of x is called the trailing coefficient, tc(P). A polynomial P is called monic if lc(P) = 1. A polynomial is considered positive if its leading integral coefficient (leading coefficient of each variable taken recursively) is positive.

Finally we define square-free decomposition of a polynomial in J[x] and partial fraction decomposition for a

rational function where the numerator and the denominator are both polynomials in J[x]. A polynomial in J[x] is called square-free if all of its non-trivial factors (i.e. factors of positive degree in x) occur not more than once. The square-free decomposition of a polynomial Q(x) in J[x] is a product of the form $C \prod_{i=1}^{k} Q_i^i$ where C is in J, each Q_i is primitive and positive in J[x], and the Q_i's are pairwise relatively prime. A rational function over J[x] is a numerator-denominator pair of polynomials in J[x], P(x)/Q(x). Each such rational function has a canonically reduced form where P and Q are relatively prime and Q is positive. Two rational functions are said to be canonically equal if their canonically reduced forms are identically equal. A rational function P(x)/Q(x) is called proper if deg(P) < deg(Q) and improper otherwise. For a canonically reduced proper rational function P(x)/Q(x), a square-free partial fraction (s.f.p.f.) decomposition of P/Q is $\sum_{i=1}^{k} P_i(x)/(C_i Q_i^i(x))$ where Q has a square free decomposition

$$C \prod_{i=1}^{k} Q_i^i, \quad P_i \text{ is in } J[x], \quad C_i \text{ is in } J, \text{ and } \deg(P_i) < \deg(Q_i^i)$$

for all $1 \leq i \leq k$. By successive application of pseudo-division in $J[x]$, each $P_i/(C_i Q_i^i)$ can be further decomposed in the form $\sum_{j=1}^{i} P_{i,j}(x)/(C_{i,j} Q_i^j(x))$ where $\deg(P_{i,j}) < \deg(Q_i)$ for all $1 \leq j \leq i$. If each term of s.f.p.f. decomposition of P/Q is so expressed such that

$$P/Q = \sum_{i=1}^{k} \sum_{j=1}^{i} P_{i,j}(x)/(C_{i,j} Q_i^j(x))$$

then we have the complete square-free partial fraction (c.s.f.p.f.) decomposition of P/Q.

BIBLIOGRAPHY

[BER67] Berlekamp, E. R., "Factoring Polynomials over Finite Fields", Bell System Technical Journal, Vol. 46, 1967, MR 2314, pp. 1853-1859.

[BER70] Berlekamp, E. R., "Factoring Polynomials over Large Finite Fields", Mathematics of Computation, November, 1970.

[BON73] Bonneau, R., "The Polynomial Operations Using the Fast Fourier Transform", Department of Mathematics, M.I.T. (in preparation).

[BRO71] Brown, W. S., "On Euclid's Algorithm and the Computation of Polynomial Greatest Common Divisors", JACM, Vol. 18, No. 4, October, 1971, pp. 478-504.

[BRO72] Brown W. S., ALTRAN User's Manual, Second Edition, Bell Laboratories, 1972.

[B&M65] Birkoff, G., and MacLane, S., A Survey of Modern Algebra, Third Edition, The MacMillan Company, 1965.

[B&T71] Brown, W. S., and Traub, J. F., "On Euclid's Algorithm and the Theory of Subresultants", JACM, Vol. 18, No. 4, October, 1971, pp. 504-518.

[COL67] Collins, G. E., "Subresultants and Reduced Polynomial Remainder Sequences", JACM, Vol. 14, January, 1967, pp. 128-142.

[COL69] Collins, G. E., "Computing Time Analyses for Some Arithmetic and Algebraic Algorithms", Proceedings of the 1968 Summer Institute on Symbolic Mathematical Computation (Robert G. Tobey, ed.), IBM Federal Systems Center, June, 1968, pp. 195-232.

[COL71] Collins, G. E., "The Calculation of Multivariate Polynomial Resultants", JACM, Vol. 18, No. 4, October, 1971, pp. 515-532.

[FAT73] Fateman, R. J., "Polynomial Multiplication, Powers, and Asymptotic Analysis: Some Comments", to appear in SIAM Journal on Computing.

[FRO00] Frobenius, G., "Uber die Characteren der Symmetrischen Gruppe", Sitz. Pruss. Acad. Berlin, 1900, P. 516.

[GEN73] Gentleman, W. M., "On the Relevance of Various Cost Models of Complexity", Complexity of Sequential and Parallel Numerical Algorithms, Traub, J. F., ed., Academic Press, N.Y., 1973, pp. 103-109.

[HEA72] Hearn, A. C., "An Improved Non-Modular Polynomial GCD Algorithm", ACM SIGSAM Bulletin, No. 23, July 1972, pp. 10-15.

[HER64] Herstein, I. N., "Topics in Algebra, Blaisdell Publishing Company, 1964.

[HOR69] Horowitz, E., Algorithms for Symbolic Integration of Rational Functions", Ph. D. Thesis, Computer Science Dept., University of Wisconsin, 1969.

[HOR71] Horowitz, E., "Algorithms for Partial Fraction Decomposition and Rational Function Integration", [PET71], March, 1971, pp. 441-457.

[HOR73] Horowitz, E., "On the Substitution of Polynomial Forms", Presented at SIAM National Conference, June, 1973.

[JOH66] Johnson, S. C., "Tricks for Improving Kronecker's Method", Bell Laboratories Report, 1966.

[J&G73] Johnson, S. C., and Gentleman, W. M., "Analysis of Algorithms, A Case Study: Determinants of Polynomials", Proc. 5th Annual ACM Symposium on the Theory of Computing, Austin, Texas, May, 1973; also submitted to Journ. ACM.

[KNU69] Knuth, D. E., The Art of Computer Programming, Vol. 2: Seminumerical Algorithms, Addison Wesley, Reading, Massachusetts 1969.

[MAC73] MACSYMA User's Manual, by Bogen, R., et Al., Project MAC, M.I.T., Cambridge, Mass., 1973.

[MOS67] Moses, J., "Symbolic Integration", MAC-TR-47, Project MAC, M.I.T, Dec., 1967.

[MOS71] Moses, J., "Symbolic Integration: The Stormy Decade", [PET71], March, 1971, pp. 427-440.

[MUS71] Musser, D. R., "Algorithms for Polynomial Factorization", Ph. D. Thesis, Computer Science Department, University of Wisconsin, August, 1971.

[MUS73] Musser, D. R., "Multivariate Polynomial Factorization", TR-11, the University of Texas at Austin, January, 1973.

[M&Y73] Moses, J., & Yun, D. Y. Y., "The EZ GCD Algorithm", Proceedings of ACM Annual Conference, August 1973, Atlanta, pp. 159-166.

[PET71] Proc. of the Second Symposium on Symbolic and Algebraic Manipulation, Petrick, S. R., ed., ACM, March, 1971.

[RIS69] Risch, R., "The Problem of Integration in Finite Terms", Trans. AMS, Vol. 139, May, 1969, pp. 167-189.

[W&R73] Wang, P. S., and Rothschild, L. P., "Factoring Multivariate Polynomials over the Integers", ACM SIGSAM Bulletin, No. 28, Dec. 1973, pp. 21-29.

[ZAS69] Zassenhaus, H., "On Hensel Factorization I", Journal of Number Theory, Vol. 1, 1969, pp. 291-311.