ON EFFICIENT COMPUTATION OF GROBNER BASES

Justin M. Gash

Submitted to the faculty of the University Graduate School
in partial fulfillment of the requirements
for the degree
Doctor of Philosophy
in the Department of Mathematics,
Indiana University
July 2008

Accepted by the Graduate Faculty, Indiana University, in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Doctoral Committee

Jee Koh, Ph.D.

Darrell Haile, Ph.D.

Jiri Dadok, Ph.D.

Daniel Leivant, Ph.D.

June 27, 2008

To all those who believed in me,

and all those who didn't.

Acknowledgements

I should start by saying that there is an awful lot of people I would like to thank, so this acknowledgements section might be unusually long. But I believe that the successes in my life are in large part due to the love, support and encouragement of people around me, and I would be remiss if I didn't take the time at the outset of this article to pay tribute to their efforts.

I would like to start by thanking my thesis advisor, Professor Jee Koh, for his guidance and support. I would also like to thank the other thesis defense committee members: Professors Darrell Haile, Jiri Dadok and Daniel Leivant. I truly appreciate the time investment on my behalf (a time investment that is not limited to time spent on my committee).

I also need to think Till Stegers for his input on Grobner basis algorithms and for the use of his code. Thanks to Gwenole Ars for his input as well. I would also like to think Christopher Wolf for his encouragement – I truly appreciate it.

Besides my committee, there are other professors here at Indiana University that have had a large impact on my graduate career. Many of them spent extra time with me in office hours and outside of class. I would like to thank Professors Valery Lunts, Vinay Deodhar, Victor Goodman, Chuck Livingston, Grahame Bennett, Dirk van Gucht (Computer Science), Randall Bramley (Computer Science), Minaxi Gupta (Computer Science), Paul Purdom (Computer Science), Markus Jakobsson (Informatics), Steven Myers (Informatics) and Peter Sternberg for their counsel and support. Whether it was preparing for exams, additional help in class, advice in my career, a good word on my behalf or simply good conversation, they were there. I would like to extend a special thanks to Dean Kirstine Lindemann for her efforts on my behalf and her great work with students. I would also like to thank Professor Emeritus Billy Rhoades and his lovely wife Mary Lou for their friendship throughout my time here at Indiana University – I will never forget your kindness.

I have also been blessed with terrific models for being a mathematics teacher, and I would like to thank those faculty members who have help to mold me into a better educator. I will begin with the two coordinators with whom I have spent the most time – Linda McKinley and Greg Kattner. Thank you both for your tremendous effort (both in working with me and for your students), for your friendship and for listening when I needed advice or someone with whom to talk. I am forever indebted to you. I would also like to thank Professor William Wheeler for his work on WeBWorK homework assignments, Andrew Dabrowski and John Steele for their good company and Steve McKinley for his guidance and friendship. I would also like to extend a special thanks to Professor Frank Lester (Education) for the time spent with him in his office discussing mathematics education – your input and perspective have been an invaluable addition to my mathematics teaching career.

And I would be foolish to forget the many hard-working (and under-appreciated on occasion, if I may say) staff members during my time at IU. They do a great job, and I believe that my success here is in no small part due to their support and friendship. I would like to specifically thank Misty Cummings and Kate Bowman for answering my constant nagging questions (many of which they had answered before but have to answer again because I forgot what they told me). Thanks to Jim "Father" Opiat for his help in cataloging my student evaluations, general assistance, and being an overall good guy. I would like to thank Clay Collier for his friendship and for his support of the National League Central (even if it is the Reds). A special thanks goes out to Jeff Taylor and Zeke Henline – my research went much smoother thanks to their extra efforts with IT. I want to thank Connie Wright for being a great mailroom manager; she is without question the best Connie Wright ever. Thanks to thank Mary Jane Wilcox for her support, friendship, good company, excellent taste in local cuisine and her foolish (yet charming) belief that IU will win the NCAA men's basketball tournament every Spring. I would also like to thank Ginny Jones, Cheryl Miller, Mandie Frye, Teresa Bunge, Bob Noel and Jaime Chapman for there assistance and camaraderie. In a different vein, I would

like to thank Katie Kearns and Carol Subino from instructional support services for their outstanding advice in preparing my teaching portfolio and job applications. Finally, I would like to thank Sherry Kay and Lucy Battersby in the computer science department for their logistical help in earning my M.A.

The faculty and staff at IU have been instrumental in my academic, professional and personal success thus far; but, without question, the greatest source of support in the IU family has been from my fellow graduate students. There have been many to touch my life during my seven years here at IU, but I am going to try to list all of those who have made a permanent impression on me. (Please accept my heartfelt apologies if I leave you out.) I would like to thank, in no particular order, Jiun-Chau "J.C." Wang, Cornelia van Cott, Micah (and Kirk) Knobel, Jennifer "Smiley" Betcher, Prudence Heck, Brian Milleville, Lori Ramsay, Eric (and Lisa) Oglesbee, Kevin Foster, Adam (and Julia) Weyhaupt, Soyeon Lee, Bo-hae Im, Du "Wizard" Pham, Chelsea Smock, Noah Salvaterra, Shantia Yarahmadian, Serdar Altok, Saleh Aliyari, Tob Hagge, Michelle Hackman, Jiho (and Maryanne) Kim, Jung-Miao "Maxi" Kuo, Bongsuk Kwon, Sam Vaughn, Josh Sack, Matt Mauntel, Peter Rankenburg, Dan Smith and Cristina Tone, Kelly Steinmetz, Jonathan (and Kat) Yazinski, Saied Yasamin, Masoud Yari, Lisa O'Rourke, Jonathan Boggess, Brennan White, Eric Wilson, Onur Yavuz, Lester Mercado, Eric Wilson and Meghan Kiernan.

I would like to give special thanks to the members of my office – Jesse "Crarford" Crawford, Shawn "Alspizzle" Alspaugh, Justin Mazur, David Meyer and Rob O'Connell. Thank you for your friendship, support, candor, sense of humor and spirit – I have been truly blessed to know you. In particular, I would like to thank David Meyer for his exhaustive help during the Spring of 2005 during preparations for my oral exams.

I would also like to give my eternal thanks to the following special friends – my experience at IU would not have been successful without them. I would like to thank

Josh Riddell and Steve Maguire for their sense of humor and unyielding support; Rahmet Savas for her inner strength and wonderful spirit; Kati Leed for her passion, support, advice and a patient ear; Jennifer Franko for her extra efforts to keep me up to speed during my illness of 2002, friendship and counsel; Jason "Outlaw" Shaw for his friendship, fun-loving attitude and for being really awesome; Rob Eleuteri for his guidance in improving my health, his faith and his positive attitude; Chris and Donna Wilson for their friendship and widespread support – since my first week of graduate classes. I am greatly indebted to all of you, and I will never forget it. Thank you so very much.

To finish up with the people from my time in Bloomington, I would like to thank Reverend David Bremer and the congregation at the United Presbyterian Church. I have grown in Christ through my time here in Bloomington thanks in large part to their spirit of love and kindness. I also need to thank my neighbors who have been a boon to me during my time here. They are good friends and were especially helpful during my illness of 2002. I would like to thank Mike and Joyce Danielson, Dick and Ethel Satter, Joe and Bev Beckes and Sara Scott.

Outside of IU I have been blessed with many other good, long-time friends. Though I may not be in constant contact with all of them anymore, I need to thank the people who have been there for me for a long time, giving me a consistent source of support, love and advice. (Again, please accept my apologies if I leave you out.) I would like to thank, in no particular order, Mark and Leslie Boeckel, David and Tracy Townsend, J.K. and Christina Wall, David Clucas, Sarah Bell, Todd Foose, Kendall Noyes, Heather Landry and my brothers in the Beta-Beta Chapter of Sigma Nu fraternity. I would also like to thank the Seib family (Frank, Nancy, Ben, Sara and the late Jim and June Pyle), the Haymaker family (John, Susan and Sarah), Joe Fredrich, the Scott family (Jackimaye, Dale, Alexander, Sara and Uncle Nick), the Maurer family (Frances, Robert and Tim), the Herrin family (Dan, Patricia, Ryan Pelky and Marissa Mitchell) and the Hruska family (Dave, Robin, Joel and

Rebecca) for their love and support of me and my family. I also need to give special thanks to Dr. Ahmet Percinel of Tri-State Orthopaedics and Mark Grundman of Grundman's Shoes; without your expert craft and vision, I would not be the man that I am today – my eternal gratitude is extended to both of you.

I would like to especially thank Mike and Marissa Mitchell and Brian and Katie Dixon for their close friendship to me and my wife – I am truly blessed to have you in my life, and your long support has helped me through quite a lot. And I would like to extend thanks to Alexander and Jana Scott, Tim and Dawn Maurer and Joel Hruska for almost two decades of blessing, good times and friendship. You are all so close to me that my thanks are implied, but I include them here nonetheless.

I consider myself so blessed to have a loving family. I would like to thank my father, Vernon, and my mother, Janice, for 29 years of undying love and counsel. Your guidance has helped mold me into the mathematician I am today, and any success I have is due in large part to you. I would like to thank my brother, Ryan, for his kinship, good heart and charming spirit. You are an inspiration to me, and I hope to make you proud. I also want to thank my grandmother Viola "Nonnie" Fansler and my late grandfather Harmon "Papaw" Fansler for their guidance and shining example. Thanks to Teresa "Druid-mother" Taylor, my loving aunt, for her love and support. I need also thank my late great aunts Jane "Aunts" Arthur and Helen Elder for their nurturing love during my childhood. Finally, I would like to thank some of the newest members of my family: Bob and Pat Chapman, Mark and Judy Chapman and Keith Chapman for their love and support.

But the person to whom I owe the most gratitude is my beautiful, loving wife Andrea. Thank you for your undying love, friendship and loyalty. You are God's most powerful blessing in my life, and this thesis is a reflection of your strong and steadying influence. I pray that this work – this beginning of my career as a mathematician – is the start of a new and exciting chapter in our lives. I am yours, with love, always.

May God bless each and every person as He has me.

Justin M. Gash

ON EFFICIENT COMPUTATION OF GROBNER BASES

On October 2, 2000 the National Institute of Standards and Technology chose to adopt a new cryptographic algorithm as the standard for the United States government. This new system was called Rijndael, named after its creators Vincent Rijmen and Joan Daemon. Though it remains unbroken, cryptanalysts are using so-called Grobner-basis attacks in an attempt to break it. Thus, the onus is now on finding Grobner bases quickly.

In 2002 J.C. Faugere published an algorithm called F5 that found Grobner bases dramatically quicker in most cases; however, in some cases, his program failed to terminate. The problem posed is two-fold:

1. Can F5 be improved so that it terminates?
2. Can the hypotheses for termination be tightened?

My research has produced a modified F5 algorithm (called F5t) that guarantees termination in all cases. In addition I demonstrate a current accepted major theorem in Grobner basis theory is false. I replace the erroneous theorem by a new theorem, proving that a slightly modified F5 can be made to terminate (correctly) over finite fields.

_____

_____

_____

_____

Contents

List of Tables

1. Introduction

On October 2, 2000 the National Institute of Standards and Technology chose to adopt a new cryptographic algorithm as the standard for the United States government. This new system was called Rijndael, named after its creators Vincent Rijmen and Joan Daemon. It remains unbroken, and has since become a favored method of encryption the world around; it is now referred to as the Advanced Encryption Standard (AES) [4].

Among the most interesting properties of Rijndael is it's method of encryption - a basic algebraic algorithm that is completely public! Moreover, every step in the algorithm is completely invertible. This means that:

(1) Everyone knows the steps the Rijndael algorithm uses to encrypt a message.

(2) All the steps Rijndael uses are completely invertible.

Thus it would seem that Rijndael would provide no security at all. At first glance it seems that one would be able to simply take a ciphertext – the encoded message – and invert it back to the plaintext – the original message – from which it came. And that technique does work, but it is computationally infeasible. Let us consider the following overview of Rijndael.

Rijndael's security comes from a secret key. Different key sizes can be used, but we will view Rijndael's algorithm with a 128-bit key. Rijndael is a symmetric key cryptosystem, meaning the two people communicating are assumed to already have shared the key. During this first portion of our overview, the key is extended to create ten children keys (for a total of 11 keys).

To start, the 128-bit key is split into 16 bytes, and the bytes are placed in a 4×4 matrix K. These bytes can actually be viewed as polynomials in F_{256}, the field of 256 elements. This ensures that matrix operations can be done and the elements remain in F_{256}. Call the columns of this matrix $W(0)$, $W(1)$, $W(2)$ and $W(3)$.

Then the matrix is extended into 40 more columns of bytes using very basic bitwise operations; this extension is based upon the elements in K. (For brevity and simplicity, we will omit the details of the extension. Full details can be found in [16].) Thus a 4×44 matrix has been constructed, which can be viewed as a set of eleven 4×4 matrices, $\{M_0, \ldots, M_{10}\}$, each with columns $W(4k)$, $W(4k+1)$, $W(4k+2)$ and $W(4k+3)$. Now a block of data - call it m for message - is also stored into a 4×4 matrix of bytes. The following operations are applied to the matrix m over ten different rounds:

(1) Round Key Addition: XOR m with M_k, where k is the round number.

(2) Shift Row: Permute the elements of m in a public way.

(3) Mix Column: Multiply m by a public 4×4 invertible matrix of bytes.

(4) Byte Sub: Map each byte from m to another byte dictated by a one-to-one public mapping.

This yields the output of the algorithm (see [16] for full details).

We can see everything is known except the key K and every operation is public and one-to-one, thus invertible. And all these invertible, public operations are done within the framework of matrices (with entries in a field). Thus solving for the original message m is equivalent to solving a system of equations. With knowledge of the special key (and thus K), the sender and receiver can easily solve this system. However, without knowledge of K solving this system requires solving 8000 quadratic equations with 1600 binary unknowns [4]. This is very difficult computationally.

As shown in [5], Grobner bases of polynomial ideals offer a quick solution to solving a system of equations. Grobner bases of polynomial ideals are special and powerful generating sets for those ideals. We will discuss, at length, some of the properties of Grobner bases in this paper; but it is easy to observe how the Grobner basis of

an ideal assists in the solving of a system of equations. The Grobner basis of an ideal triangulates the indeterminates so that a naive substitution method will solve for each indeterminate. Using an example from [5], we see that the Grobner basis of the ideal $I = \langle x^2 + y + z - 1, x + y^2 + z - 1, x + y + z^2 - 1 \rangle$ is:

$$g_1 : x + y + z^2 - 1$$
$$g_2 : y^2 - y - z^2 + z$$
$$g_3 : 2yz^2 + z^4 - z^2$$
$$g_4 : z^6 - 4z^4 + 4z^3 - z^2$$

It is clear how one could naively solve this system of equations. So the new idea for breaking AES is to take the equations resulting from the AES algorithm, look at the Grobner basis of the ideal generated by those equations, and solve. Since the solution of the system is trivial, this method of solving a system of equations reduces to efficiently finding the Grobner basis of an ideal. In short, we trade the difficulties of directly solving an extremely large and complicated system of equations for the difficulty of finding a Grobner basis for a set of polynomials.

In 2002, J.C. Faugere published an algorithm called F5 in [7]. This algorithm has been shown in empirical tests to be the fastest Grobner-basis-generating algorithm devised. (It is worth noting that the complexity class of the Grobner basis generating problem is still approximately double exponential in the number of indeterminates; this was recently discussed in [2], an example was produced in [11] and the topic was originally discussed in [14].) Though his algorithm works well, it doesn't always terminate.

This thesis spends a great deal of time in further development of the mathematics supporting the F5 algorithm. In addition, this thesis focuses on improving the F5 algorithm so that it terminates in all cases, and without a significant sacrifice to the speed of the F5 algorithm in many cases. The pitfall of the F5 algorithm is that it can continue indefinitely for some inputs. By using carefully placed and

sparingly used high-priced computations, a stopping condition for the algorithm can be achieved.

For those who are already familiar with Grobner-basis-algorithm theory, we offer the following outline of this thesis. In chapter 2, entitled Background Information, we will introduce the background necessary for our discussions within the paper. This preliminary discussion can be supplemented by material in [3, 5, 6].

In chapter 3, entitled Faugere's F5, we discuss the algorithm introduced in [7] in 2002. This chapter contains the following clarifications of previous publication items:

(1) In subsection 3.3, we clear up some notational and minor content errors in previous proofs of the F5 criterion.

(2) In sub-subsection 3.4.2, we reproduce the example used by Faugere in [7]. The exposition here is lengthier and a few minor computational errors have been corrected.

Chapter 3 also contains the following new material:

(1) In subsection 3.1, Theorem 3.1.1 proves Proposition 1 introduced in [7].

(2) In subsection 3.2, Lemma 3.2.1 proves that normalizing a non-normalized polynomial can yield a different signed representation; this lemma plays a very important role in the thesis. In addition, Theorem 3.2.1 proves that, under certain specialized conditions, signed (and admissible) polynomials have minimal signature.

(3) In sub-subsection 3.4.1, we produce the first proof (of this author's knowledge) of the validity of the Rewritten subroutine in F5.

(4) In subsection 3.5, we introduce an error in the proof of F5's termination from [7]. We also introduce a method to fix the proof via a minor change in the F5 algorithm.

In chapter 4, entitled Improving F5 and Introducing F5t, we discuss various improvements to F5. In section 4.1 we prove a minor improvement to the F5 algorithm first introduced in [15]. In sections 4.2 and 4.3 we verify applications of Buchberger's first and second criteria within F5. Buchberger's second criterion is slightly modified to accommodate the use of signatures within F5. In addition, with respect to Buchberger's first criterion, we have Corollary 4.2.2 that speaks to the usefulness of applying the criterion within F5. In section 4.4 we introduce a new F5-Buchberger algorithm hybrid called F5t. This new algorithm is guaranteed to terminate for any input. Included within the section is the proof of a new criterion, pseudocode for F5t, proof of correctness, proof of termination and timing data comparing F5 and F5t.

In chapter 5 – the conclusion section – we sum-up the contents of the paper and will give some final commentary.

There are also two appendices. The first appendix, Appendix A, is an exposition on the error in the proof of F5's termination made in [7]. Though much of the content of the error is explored in subsection 3.5, Appendix A discusses precisely where the error in logic occurs. Appendix A uses the example from sub-subsection 3.4.2 for illustration. The second appendix, Appendix B, contains the code used to implement the F5t algorithm. The code is written in the MAGMA computer algebra language (see [13]).

2. Background Information

In this chapter we provide some background for the subsequent chapters. This chapter is by no means self-sufficient (see [3, 5, 6] for additional background). For readers who are familiar with these topics, this chapter should provide the necessary background to comfortably continue through the remainder of the paper.

2.1. **Term Orders.**

Definition 2.1.1. *Let M be a set. Then a binary relation on the set $M \times M$ is a subset $R \subset M \times M$. A relation is a total order on M if R has the following properties:*

(1) $\forall x \in M$, $(x, x) \in R$ *(reflexivity)*

(2) $\forall x, y \in M$, *either* (x, y) *or* $(y, x) \in R$ *(connexicity)*

(3) *if* $x, y, z \in M$ *such that* $(x, y) \in R$ *and* $(y, z) \in R$, *then* $(x, z) \in R$ *(transitivity)*

(4) *if* $x, y \in M$ *and* $(x, y), (y, x) \in R$, *then* $x = y$ *(antisymmetry)*

Definition 2.1.2. *Let $M_1 \subset M$ and let $<$ be a relation on M. We say M_1 has an $<-$minimal element m if $m < m'$ $\forall m' \in M_1$, $m' \neq m$. (A similar definition defines the notion of an $<-$maximal element.) If an $<-$minimal element exists for any subset $M_1 \subset M$, then we say $<$ is a well-ordering on M.*

A similar but more complete set of definitions for binary relations and various order types can be found in [15].

Let K be a field and let $\mathcal{P} = K[x_1, x_2, \ldots, x_n]$. From now on, for ease of reading, we will often denote $K[x_1, x_2, \ldots, x_n]$ as $K[\overline{x}]$. We will let T denote the set of terms in $K[\overline{x}]$, $T = \{x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_n^{\alpha_n} | \alpha_1, \alpha_2, \ldots, \alpha_n \in \mathbb{Z}_{\geq 0}\}$

Definition 2.1.3. *An order $\leq$ on T is called a term order if $\leq$ is a total well-ordering on T such that $1 \leq t$ $\forall t \in T$ and for $t_1, t_2 \in T$ such that $t_1 \leq t_2$, $t_1 t \leq t_2 t$ $\forall t \in T$.*

This above definition is paraphrased from [15].

Term orders are generally viewed as binary relations on $\mathbb{Z}^n_{\geq 0}$ where a term $x^\alpha = x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_n^{\alpha_n}$ is mapped to $(\alpha_1, \alpha_2, \ldots, \alpha_n) \in \mathbb{Z}^n_{\geq 0}$. Note that we use the shorthand notation of x^α for $x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_n^{\alpha_n}$. The above map is clearly an isomorphism, so any order properties of the term order $\leq$ in T are maintained by the corresponding total order in $\mathbb{Z}^n_{\leq 0}$. Thus term orders are often defined in a way similar to that in [5]:

Definition 2.1.4. *A term ordering on $K[x_1, x_2, \ldots, x_n]$ is a relation $\leq$ on $\mathbb{Z}^n_{\geq 0}$ such that:*

(1) *$\leq$ is a total order on $\mathbb{Z}^n_{\leq 0}$*

(2) *$\leq$ is a well-ordering on $\mathbb{Z}^n_{\leq 0}$*

(3) *if $\alpha \leq \beta$ and $\gamma \in \mathbb{Z}^n_{\leq 0}$, then $\alpha + \gamma \leq \beta + \gamma$*

We also define the total degree of a term:

Definition 2.1.5. *The total degree of a term $x^\alpha = x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_n^{\alpha_n}$ is $\sum_{i=1}^n \alpha_i$. We will denote the total order of x^α as $|\alpha|$.*

Remark 2.1.1. Here we give a few examples of term orders.

(1) *Lexicographical Order.* Lexicographical order, or lex order for short, is similar to the dictionary's method for ordering words (hence the name). We begin with an order on the indeterminates, say $x_1 > x_2 > \cdots > x_n$. Then, for $\alpha, \beta \in \mathbb{Z}^n_{\leq 0}$, we say $\alpha >_{lex} \beta$ if and only if $\exists j$, $1 \leq j \leq n$, such that $\alpha_j > \beta_j$ and $\alpha_i = \beta_i$ $\forall i$, $1 \leq i \leq j-1$.

(2) *Graded Lexicographical Order.* This often shorthanded as grlex, and simply adds one layer to lex ordering. We begin with an order on the indeterminates, say $x_1 > x_2 > \cdots > x_n$. Then, for $\alpha, \beta \in \mathbb{Z}^n_{\leq 0}$, we say $\alpha >_{grlex} \beta$ if and only if either $|\alpha| > |\beta|$ or $|\alpha| = |\beta|$ but $\exists j$, $1 \leq j \leq n$, such that $\alpha_j > \beta_j$ and $\alpha_i = \beta_i \ \forall i$, $1 \leq i \leq j-1$.

(3) *Graded Reverse Lexicographical Order.* This is a widely used ordering, and is shorthanded as grevlex. We begin with an order on the indeterminates, say $x_1 > x_2 > \cdots > x_n$. Then, for $\alpha, \beta \in \mathbb{Z}^n_{\leq 0}$, we say $\alpha >_{grevlex} \beta$ if and only if either $|\alpha| > |\beta|$ or $|\alpha| = |\beta|$ but $\exists j$, $1 \leq j \leq n$, such that $\alpha_j < \beta_j$ and $\alpha_i = \beta_i \ \forall i$, $j+1 \leq i \leq n$.

It is important to note that grevlex is **not** simply a reverse of grlex. Let's consider a brief example, with $n = 3$. Let $\alpha = (3, 4, 5)$ and $\beta = (2, 4, 6)$. Then $\alpha >_{grlex} \beta$ because $|\alpha| = |\beta| = 12$ but $3 > 2$. It is also true, however, that $\alpha >_{grevlex} \beta$ because $|\alpha| = |\beta| = 12$ but $5 < 6$. Notice that the tie-breaking occurs at different spots in the two orderings; grlex starts at the left-most entry and grevlex starts at the right-most entry.

For more examples of term orders, refer to [5].

2.2. **Grobner Bases.** With the idea of a term order firmly in hand, we can make the following important consequential definition:

Definition 2.2.1. *Let polynomial $p \in K[\overline{x}]$, T be the set of terms and $\leq$ be a term order on T. Then the head term of p, denoted $HT(p)$, is the term t of p such that, for all other terms t' of p, $t' \leq t$. We will define the head coefficient of p, denoted $HC(p)$, is the coefficient (from K) of $HT(p)$. We will say that the head monomial of p, denoted $HM(p)$, is $HM(p) = HC(p)HT(p)$. Finally, for a set I of multiple polynomials, we define the notation $HT(I) = \{HT(p) | p \in I\}$.*

Remark 2.2.1. Some sources on term orders will interchange our definition of a head term and a head monomial. This definition given here is consistent with [3, 15]. This interchange of definitions causes no ill (other than confusion) in Grobner basis theory.

Now that we have defined the notions of a head term, head coefficient and head monomial, we introduce an important analogue of single-variable polynomial division: namely, reduction one of polynomial by another. These definitions are taken from [3].

Definition 2.2.2. *Let $f, g, p \in K[\overline{x}]$ with $f, p \neq 0$; let $P \subset K[\overline{x}]$. We say that f reduces to g modulo p by eliminating t, denoted by $f \underset{p}{\rightarrow} g[t]$, if t is a term of f, $\exists s \in T$ with $sHT(p) = t$ and $g = f - \frac{a}{HC(p)} sp$, where a is the coefficient of t in f. We say f reduces to g modulo p, denoted $f \underset{p}{\rightarrow} g$, if $f \underset{p}{\rightarrow} g[t]$ for some t, a term of f. We say f reduces to g modulo P, denoted $f \underset{P}{\longrightarrow} g$, if $f \underset{p}{\rightarrow} g$ for some $p \in P$. We say f is reducible modulo p if $\exists g \in K[\overline{x}]$ such that $f \underset{p}{\rightarrow} g$; similarly, we say f is reducible modulo P if $\exists g \in K[\overline{x}]$ such that $f \underset{P}{\longrightarrow} g$. We will say f is top-reducible by p to g if $f \underset{p}{\rightarrow} g[t]$ and $t = HT(f)$. If f is not reducible modulo p (P), then we will say that f is in normal form with respect to p (P); thus a normal form of f modulo P is a polynomial g that is in normal form modulo P and satisfies the condition that $f \underset{P}{\overset{*}{\longrightarrow}} g$, where $\overset{*}{\rightarrow}$ is the reflexive-transitive closure of $\underset{P}{\longrightarrow}$.*

Now we can give a definition of a Grobner basis. The following definition is given in [15], but is based upon the fuller definition given in [3]. For our purposes the shorter definition will do. First we give a theorem.

Theorem 2.2.1. *Let G be a finite subset $K[\overline{x}]\backslash\{0\}$. The following conditions are all equivalent.*

(1) *Every $f \in \langle G \rangle$ has a unique normal form with respect to $\underset{G}{\longrightarrow}$.*

(2) *For every $f \in \langle G \rangle$, there is a reduction so that $f \underset{G}{\overset{*}{\longrightarrow}} 0$.*

(3) *Every non-zero* $f \in \langle G \rangle$ *is reducible by* G.

(4) *Every non-zero* $f \in \langle G \rangle$ *is top-reducible by* G.

(5) *For every* $s \in HT(\langle G \rangle)$ *there is a* $t \in HT(G)$ *such that* $t|s$.

(6) $HT(\langle G \rangle) \subset \langle HT(G) \rangle$.

(7) *The polynomials* $h \in K[\overline{x}]$ *that are in normal form with respect to* G *form a system of representatives for the partition* $\{f + \langle G \rangle | f \in K[\overline{x}]\}$ *of* $K[\overline{x}]$.

Proof. See [3]. □

Definition 2.2.3. *Let* G *be a finite subset* $K[\overline{x}] \backslash \{0\}$. *Then* G *is a Grobner basis if it satisfies the equivalent conditions of Theorem 2.2.1.*

It is important to note that Grobner bases are not unique. In particular, if G were a Grobner basis for some ideal $I \in K[\overline{x}]$, then one could simply multiply every element of G by a non-zero element of K to achieve another Grobner basis. It would be nice if we could generate some semblance of uniqueness in the world of Grobner bases. As described in [10], this can be done.

Definition 2.2.4. *A Grobner basis* $G = \{g_1, g_2, \ldots, g_{\mathcal{L}}\}$ *of* $I \in K[\overline{x}]$ *is said to be minimal if* g_i *is monic* $\forall i$, $1 \leq i \leq \mathcal{L}$, *and if* $HT(g_i)$ *does not divide* $HT(g_j)$ *for* $i \neq j$, $1 \leq i, j \leq \mathcal{L}$. *A Grobner basis* $G = \{g_1, g_2, \ldots, g_{\mathcal{L}}\}$ *of* $I \in K[\overline{x}]$ *is said to be reduced if* g_i *is monic* $\forall i$, $1 \leq i \leq \mathcal{L}$, *and if none of the terms of* g_i *is divisible by* $HT(g_j)$ *for* $i \neq j$.

Theorem 2.2.2. *Let* I *be an ideal,* $I \subset K[\overline{x}]$. *Then* $\exists G \subset K[\overline{x}]$ *such that* G *is the unique reduced Grobner basis of* I. *[Note that this theorem not only states that Grobner bases exist for any polynomial ideal* I *in* $K[\overline{x}]$, *but it says that the reduced Grobner basis for* I *is unique.]*

Proof. For existence of a Grobner basis, see [3, 5]. For the uniqueness of the reduced Grobner basis, see [10]; this source will also show how to create a reduced Grobner basis from a non-reduced one. □

2.3. **Buchberger's Algorithm.** In section 2.2 we defined the notion of a Grobner basis and gave citations that verify that Grobner bases exist for any polynomial ideal $I \subset K[\overline{x}]$. Moreover we gave a citation that shows that the reduced Grobner basis is unique and can be formed from a non-reduced Grobner basis. But what we have not described is how to create a Grobner basis algorithmically.

The first Grobner basis algorithm was constructed by Buchberger in the 1960's; thus it bears his name to this day – Buchberger's algorithm. We will spend this subsection of this chapter outlining Buchberger's algorithm. This is of critical importance to our examination of efficient Grobner basis algorithms that are in use today because they rely on the same foundations as Buchberger's algorithm. We will begin our examination with a definition:

Definition 2.3.1. *Let $f, g \in K[\overline{x}]\backslash\{0\}$. The S-polynomial of f and g, denoted by $S(f,g)$, is*

$$S(f,g) = \frac{lcm(HM(f), HM(g))}{HT(f)} f - \frac{lcm(HM(f), HM(g))}{HT(g)} g$$

Why do we care about S-polynomials? Let us consider a simple example to illustrate why S-polynomials are of primary concern. (This example can also be found in [5].) Consider the polynomial ideal $I = \langle f_1, f_2 \rangle$ where $f_1 = x^3 - 2xy$ and $f_2 = x^2y - 2y^2 + x$. We will assume we are working in the polynomial ring $K[x,y]$ and that we are using grlex order on the terms.

Recall that in Theorem 2.2.1 and in Definition 2.2.3 we stated that a Grobner basis G for I will have the property that $HT(I) \subset \langle HT(G) \rangle$. The immediate question to ask is: Is (f_1, f_2) and Grobner basis for I? Unfortunately, the answer is no. Why not? Consider the following calculation:

$$xf_2 - yf_1 = x(x^2y - 2y^2 + x) - y(x^3 - 2xy) = x^3y - 2xy^2 + x^2 - x^3y + 2xy^2 = x^2$$

This calculation shows that $x^2 \in I$ (and, thus, $x^2 \in HT(I)$), yet (clearly) $x^2 \notin \langle x^3, x^2y \rangle$. This occurs specifically because we multiplied f_1 and f_2 by the factors needed to cause cancelations in the head terms. So even though f_1 and f_2 have head terms x^3 and x^2y respectively, it is possible to make a linear combination of f_1 and f_2 that has head term x^2. In fact, this linear combination above is precisely $S(f_2, f_1)$.

In short, if we want to assure that $HT(I) \subset \langle HT(G) \rangle$, we need to make sure all such occurrences of the above – that is, a linear combination of basis elements resulting in a cancelation of head terms and a new element of $HT(I)$ – are captured within G. That is the primary thrust of Buchberger's algorithm – an exhaustive amalgamation of all such occurrences (see Theorem 2.5.2 and [5] for more details). And it's effectiveness is proven in the following theorem.

Theorem 2.3.1. *Let $I \subset K[\overline{x}]$ be a non-zero ideal and let $G = \{g_1, g_2, \ldots, g_{\mathcal{L}}\}$ be a basis for I (i.e., $I = \langle G \rangle$). Then G is a Grobner basis of I if and only if $S(g_i, g_j) \xrightarrow[G]{*} 0$ for every $g_i, g_j \in G$.*

Proof. This proof can be found in [3, 5, 10]. □

With Theorem 2.3.1 in hand, we introduce the pseudocode for Buchberger's algorithm. (This pseudocode is taken from [3].)

Buchberger's Algorithm

Input: F a finite subset of $K[\overline{x}]$.

Output: G a Grobner basis of F.

$G := F$;

$B := \{(g_1, g_2) | g_1, g_2 \in G, g_1 \neq g_2\}$;

while $B \neq \emptyset$ do

  select (g_1, g_2) from B;

  $B := B \backslash (g_1, g_2)$;

$h := S(g_1, g_2)$;

$h_0 :=$ normal form of h modulo G;

if $h_0 \neq 0$ then

$B := B \cup \{(g, h_0) | g \in G\}$;

$G := G \cup \{h_0\}$;

Theorem 2.3.2. *Given appropriate input F, as indicated above, Buchberger's algorithm will always terminate and output G will always be a Grobner basis of F.*

Proof. This proof can be found in [3]. □

Definition 2.3.2. *The ordered pairs of elements that get placed in B in the pseudocode above are referred to as critical pairs.*

Though Buchberger's algorithm looks relatively simple, it can take a very large amount of time. The step that creates h_0 via a normal form calculation is computationally very difficult. This is particularly frustrating (and wasteful) if the normal form calculation results in $h_0 = 0$ because all that computation ends up adding nothing to the Grobner basis (h_0 is only added if it is non-zero). It would be nice if there were some way to "see ahead" and eliminate critical pairs whose S-polynomials top-reduce to 0 rather than actually computing the normal form of those S-polynomials and getting 0.

Buchberger helped to ameliorate the situation by creating two criteria by which critical pairs can be eliminated. These criteria allow us to look at a proposed critical pair and skip the normal form calculation altogether if the critical pair meets the proper hypotheses. We will now introduce the first of the criteria (taken from [5]).

Theorem 2.3.3. ***Buchberger's First Criterion.*** *Let $G \subset K[\overline{x}]$ be finite. Suppose that $f, g \in G$ such that $lcm(HT(f), HT(g)) = HT(f)HT(g)$ (i.e., the head terms of the two polynomials are disjoint). Then $S(f,g) \xrightarrow[G]{*} 0$.*

Proof. (This proof is of specific importance later in this thesis. Thus I include the entire proof here. One can also find it in [5].)

Without loss of generality, assume $HC(f) = HC(g) = 1$ (since K is a field, this is no problem). We denote $f = HT(f) + p$ and $g = HT(g) + q$. Since we know $lcm(HT(f), HT(g)) = HT(f)HT(g)$, we have

$$\begin{aligned} S(f,g) &= HT(g)f - HT(f)g \\ &= (g-q)f - (f-p)g \\ &= gf - qf - fg + pg \\ &= pg - qf \end{aligned} \tag{2.1}$$

We will show that equation 2.1 implies $S(f,g)$ is top-reducible by either f or g.

Assume, for contradiction, that $HT(p)HT(g) = HT(q)HT(f)$. Since we assumed $HT(f)$ and $HT(g)$ were disjoint, $HT(f)|HT(p)$. But this is ridiculous because, by construction, $HT(f) > HT(p)$ in the term order. Thus we conclude our assumption that $HT(p)HT(g) = HT(q)HT(f)$ is false. So, without loss of generality, we assume $HT(S(f,g)) = HT(p)HT(g)$.

Then $S(f,g)$ is top-reducible by g via p; if this reduction takes place, we are left with qf, which trivially top-reduces by f via q to 0. Thus $S(f,g) \xrightarrow[G]{*} 0$.

This completes the proof. □

This leads immediately to the following:

Corollary 2.3.1. *If (f, g) is a critical pair in Buchberger's algorithm satisfying the hypotheses of Buchberger's First Criterion, (f, g) can simply be stricken from B and no normal form calculation is required.*

Proof. By Theorem 2.3.3, $S(f, g) \xrightarrow[G]{*} 0$. This means that if we did perform the normal form calculation, $h_0 = 0$; thus, nothing would be added to G in Buchberger's algorithm. This is the same result as if we had simply stricken (f, g) from B and moved immediately to the next critical pair. □

Before introducing Buchberger's other criterion for eliminating critical pairs, we need a new definition.

Definition 2.3.3. *Let $f \in K[\overline{x}], f \neq 0$. In addition, let $p \subset K[\overline{x}]$. Then we say that*

$$f = \sum_{p \in P} q_p p$$

with $q_p \in K[\overline{x}], q_p \neq 0$ and $HT(q_p p) \leq HT(f)\ \forall p \in P$ is a standard representation of f. If we replace $HT(f)$ in the previous definition with some term t (i.e., $HT(q_p p) \leq t\ \forall p \in P$), then we have what is known as a t-representation of f.

The definition provided here is a slight variant of the one provided in [3], but the reader can disregard the discrepancy.

This yields another categorization of Grobner bases, as presented in [3]. We include it here because it is an important lens from which to view the properties of a Grobner basis; we will look at similar behavior later in section 3.

Theorem 2.3.4. *Let G be a finite subset of $K[\overline{x}]$ with $0 \notin G$, and assume that $\forall g_1, g_2 \in G$, $S(g_1, g_2)$ equals 0 or has a t-representation with respect to G, $t < lcm(HT(g_1), HT(g_2))$. Then G is a Grobner basis.*

Proof. See [3]. □

Now that we are armed with the notion of the representation of a polynomial, we can introduce the second of Buchberger's criteria (taken from [3]).

Theorem 2.3.5. ***Buchberger's Second Criterion.*** *Let $g_1, g_2, p \in G \subset K[\overline{x}]$ and assume the following additional properties hold:*

(1) $HT(p) | lcm(HT(g_1), HT(g_2))$

(2) *for $i = 1, 2$, $S(g_i, p)$ has a t_i-representation with respect to G,*
$t_i < lcm(HT(g_i), HT(p))$

Then $S(g_1, g_2)$ need not be added to G in order for G to become a Grobner basis.

Rather than prove this version of Buchberger's Second Criterion, I will state and prove a slightly different (but equivalent in content) version.

Theorem 2.3.6. *Let $g_1, g_2, p \in G \subset K[\overline{x}]$ and assume the following additional properties hold:*

(1) $HT(p) | lcm(HT(g_1), HT(g_2))$

(2) *$S(g_1, p)$ and $S(g_2, p)$ have been added to G*

Then $S(g_1, g_2)$ need not be added to G in order for G to become a Grobner basis.

Clearly, if $S(g_1, p)$ and $S(g_2, p)$ have been added to the G already, then we have the condition that, for $i = 1, 2$, $S(g_i, p)$ has a t_i-representation with respect to G such that $t_i < lcm(HT(g_i), HT(p))$.

Once again, since Buchberger's Second Criterion (revised) will appear later in the thesis, we provide a full proof of the criterion here.

Proof. We begin by noting the following representation of $S(g_1, g_2)$:

$$S(g_1, g_2) = \frac{lcm(HM(g_1), HM(g_2))}{lcm(HM(g_1), HM(p))} S(g_1, p) - \frac{lcm(HM(g_1), HM(g_2))}{lcm(HM(g_2), HM(p))} S(g_2, p)$$

(The above argument is a simple algebraic trace; if one wants verification of the statement, simply look at the beginning of the proof of Theorem 4.3.1.)

The important thing to note about this representation of $S(g_1, g_2)$ is that it is a t-representation of $S(g_1, g_2)$ with respect to G where $t < lcm(HT(g_1), HT(g_2))$. Thus, by Theorem 2.3.4, $S(g_1, g_2)$ need not be added to G for G to become a Grobner basis.

□

Thus we now have valid criteria that we can use to eliminate critical pairs from consideration before normal form calculations are done. This can help to save great amounts of time.

However, when implementing the code to check for Buchberger's Second Criterion, one needs to be careful not to fall into the trap of the so-called two-out-of-three problem – consider the following definition.

Definition 2.3.4. *If $g_1, g_2, p \in G \subset K[\overline{x}]$ such that:*

(1) $HT(p) | lcm(HT(g_1), HT(g_2))$

(2) $lcm(HT(g_1), HT(p)) | lcm(HT(g_1), HT(g_2))$

(3) $lcm(HT(g_2), HT(p)) | lcm(HT(g_1), HT(g_2))$

then we refer to (g_1, p, g_2) as a Buchberger triple. (Note that the conditions listed above are all equivalent; we list them as separate items for clarity.)

We would say that, based upon Theorem 2.3.6, if (g_1, p, g_2) is a Buchberger triple and $S(g_1, p)$ and $S(g_2, p)$ have been or will be "dealt with," then the critical pair (g_1, g_2) could be dismissed.

The two-out-of-three problem arises when you have, in addition to the conditions given in Definition 2.3.4, two of the three lcm's of head terms equal. Let's say that $lcm(HT(g_2), HT(p)) = lcm(HT(g_1), HT(g_2))$. Then we would have both

(g_1, p, g_2) and (g_1, g_2, p) as Buchberger triples. Then we might encounter the Buchberger triple (g_1, p, g_2) first, promise to deal with the critical pairs (g_1, p) and (g_2, p) later in the algorithm run, and dismiss (g_1, g_2) from consideration. That much is perfectly fine. However we cannot subsequently encounter the Buchberger triple (g_1, g_2, p) and promise to deal with the critical pairs (g_1, g_2) and (g_2, p) in order to dismiss (g_1, p); we have ALREADY dismissed (g_1, g_2) from being considered when we encountered the Buchberger triple (g_1, p, g_2) (not to mention that our dismissal of (g_1, g_2) was contingent on our dealing with (g_1, p)). This is the two-out-of-three problem. This means that any implementation of Buchberger's second criterion must take care to ensure that critical pairs that must be dealt with via one Buchberger triple are not dismissed by another.

A full exposition on the two-out-of-three problem can be found in [3]. This exposition also includes the pseudocode for a subalgorithm UPDATE. UPDATE is an implementation of Buchberger's two criteria, taking into account the two-out-of-three problem. Readers may give this pseudocode a perusal – it is still heavily referenced today.

2.4. **Applications of Grobner Bases.** Before moving into the main section of this paper, we will spend some time describing some applications of Grobner bases. We will illustrate two such applications here. First we will describe the Ideal Membership problem. Second we will spend some time on using Grobner bases to solve systems of polynomial equations; it is this application that is of interest in cryptography.

2.4.1. *Ideal membership.* The Ideal Membership problem is a very natural problem for the commutative algebraist. We let $I = \langle f_1, f_2, \ldots, f_m \rangle \subset K[\overline{x}]$ and we have a polynomial $f \in K[\overline{x}]$. We ask the question: is $f \in I$?

As illustrated earlier by the example in section 2.3, this question is not always easily answered. In that example the polynomial x^2 was in fact a member of $I = \langle x^3 - 2xy, x^2y - 2y^2 + x \rangle$; if we hadn't looked at the S-polynomial of these

two generating polynomials, we would not have been able to see that $x^2 \in I$. The generating set really didn't offer enough information on face for us to answer the question; rather, an investigation (however brief) had to be conducted. (This investigation can become orders of magnitude more exasperating when the ideals and polynomials in question become more complex.)

But a Grobner basis offers us more information than a typical basis. Consider the following theorem based on Definitions 2.2.1 and 2.2.3.

Theorem 2.4.1. *Let G be a Grobner basis for an ideal $I \subset K[\overline{x}]$. Let $f \in K[\overline{x}]$. Then $f \in I$ if and only if $f \xrightarrow[G]{*} 0$.*

Proof. This proof can be found in [5]. □

This suggests the following algorithm, presented in pseudocode, for solving the Ideal Membership problem; this algorithm requires that the basis G for the ideal I is a Grobner basis.

Ideal Membership Algorithm

Input: Grobner basis G for polynomial ideal $I \subset K[\overline{x}]$; polynomial $f \in K[\overline{x}]$.

Output: True if $f \in I$ and false otherwise.

```
if f = 0 then
  return true;
f_0 := normal form of f with respect to G;
if f_0 = 0 then
  return true;
else
  return false;
```

Now there is no guess work involved. The uniqueness of the normal form of polynomials with respect to the Grobner basis (see Definition 2.2.1) has yielded an algorithm to solve Ideal Membership that is straightforward and deterministic.

2.4.2. *Solving Systems of Polynomial Equations.* This application is of primary interest to cryptographers and code breakers. Many cryptographic primitives, specifically AES, are based upon solving a system of equations with coefficients in a field (see [4]). Before discussing this application, we need a little more background. We begin with a definition taken from [5].

Definition 2.4.1. *Let K be field and $(f_1, f_2, \ldots, f_m)$ be a sequence of polynomials in $K[x_1, x_2, \ldots, x_n]$. Then we define the variety of $(f_1, f_2, \ldots, f_m)$, denoted $V(f_1, f_2, \ldots, f_m)$, to be $\{(a_1, a_2, \ldots, a_n) \in K^n | f_i(a_1, a_2, \ldots, a_n) = 0 \ \forall i, 1 \leq i \leq m\}$. Similarly, if an ideal $I \subset K[x_1, x_2, \ldots, x_n]$, $I = \langle f_1, f_2, \ldots, f_m \rangle$, then we define the variety of the ideal I, denoted $V(I)$, to be $\{(a_1, a_2, \ldots, a_n) \in K^n | p(a_1, a_2, \ldots, a_n) = 0, p \in I\}$. An ideal is called zero-dimensional if $V(I)$ is finite.*

This leads us to an important theory of algebraic geometry and Grobner bases:

Theorem 2.4.2. *Let $I \subset K[x_1, x_2, \ldots, x_n]$ such that $I = \langle f_1, f_2, \ldots, f_m \rangle$. Then $V(I) = V(f_1, f_2, \ldots, f_m)$.*

Proof. This proof can be found in [5]. □

Theorem 2.4.2 is important because it tells us that we can choose whatever basis for ideal I we would like when trying to find the variety of I; in particular, we could use the Grobner basis for I.

On this note, let's look at two examples. Both are taken from [5]. We will consider the equations:

$$x^2 + y^2 + z^2 = 1$$
$$x^2 + z^2 = y$$
$$x = z$$

These equations determine the polynomial ideal $I = \langle x^2+y^2+z^2-1, x^2+z^2-y, x-z\rangle$. Given this basis for I, we generate a Grobner basis under lex order, $x > y > z$ (we know we can do this because we have Buchberger's algorithm). The Grobner basis turns out to be the following:

$$g_1 = x - z$$
$$g_2 = -y + 2z^2$$
$$g_3 = z^4 + \tfrac{1}{2}z^2 - \tfrac{1}{4}$$

This system of equations is relatively easy to solve because the indeterminates have been diagonalized. That is to say, g_3 can be solved in terms of it's only variable z; these roots can be placed into g_2 to solve for the only remaining variable y; these ordered pairs of roots for g_3 and g_2 can be placed into g_1 to solve for the only remaining variable x.

We consider a second example where $I = \langle x^2+y+z-1, x+y^2+z-1, x+y+z^2-1\rangle$. Once again we generate a Grobner basis for I under lex order, $x > y > z$. The Grobner basis turns out to be:

$$g_1 = x + y + z^2 - 1$$
$$g_2 = y^2 - y - z^2 + z$$
$$g_3 = 2yz^2 + z^4 - z^2$$
$$g_4 = z^6 - 4z^2 + 4z^3 - z^2$$

Once again we see the same phenomenon – diagonalization of the variables in the Grobner basis.

It turns out that this is a property of Grobner bases. Thus Grobner bases algorithms can be used to find the variety of a sequence of multi-variate polynomials. (Note that the field K must be a finite field or a field extension of $\mathbb{Q}$ [15]; this additional constraint is toothless for cryptographic applications.)

It is worth noting, as an addendum to this subsection, that solving a system of multivariate quadratic equations is known to be NP-complete. (For a discussion

of NP-completeness, see [9].) This is known as the MQ-problem. Given our above examples, this might seem hard to believe. It appears as though Grobner bases make the process of finding the variety for a sequence of polynomials rather trivial. However, one must remember that Buchberger's algorithm (and for that matter all subsequently introduced Grobner-basis-producing algorithms) still must make normal form computations during its run; this step can be quite time-consuming and complex, and it is this step that bears witness to the NP behavior of the MQ-problem.

2.5. **Additional Definitions.** There are few straggling definitions and concepts that are useful for discussing Grobner bases algorithms. In this section we include those definitions.

We begin by including the following definition:

Definition 2.5.1. *Let $f \in K[\overline{x}]$. We define the total degree of f to be the greatest total degree of all of f's terms. We say f is a homogeneous polynomial (of degree d) if all the terms in f are of the same degree (and that degree is d). (Thus if f is homogeneous of degree d, f also has total degree d.) The total degree of a homogeneous polynomial f is denoted $deg(f)$.*

Oftentimes we are given non-homogeneous polynomials when we want them to be homogeneous. There is a method of transforming a non-homogeneous polynomial into a homogeneous one.

Definition 2.5.2. *Let $f \in K[\overline{x}]$ be non-homogeneous with total degree d. Then we can do the following steps:*

(1) *Create a new indeterminate not in $K[\overline{x}]$. For our purposes here, we will call that indeterminate z. Thus we will consider ourselves working in $K[\overline{x}, z]$.*

(2) *Multiply each term in f by the appropriate power of z such that each term now has total degree d.*

(3) *Call this resulting polynomial* f^h.

This procedure is called homogenization. In this particular case, we homogenized f *(hence the name of the result* f^h*). The added indeterminate* z *is called the homogenizing variable. One can homogenize a sequence of polynomials by performing the above procedure on all polynomials in the sequence.*

Although f^h is a different object (polynomial) than f, it is clear that f is easily retrievable from f^h. All that one must do is set the homogenizing variable equal to 1. This is a natural projection map from the homogenized polynomial back to the original.

We also have a special case of Grobner bases to discuss. Consider the following:

Theorem 2.5.1. *Let* G *be a finite, homogeneous subset* $K[\overline{x}]\backslash\{0\}$. *Let* $I = \langle G\rangle$ *and let* I_d *be the restriction of* I *to degree* d*; that is,* $I_d = \{p \in I | deg(p) < d\}$. *Let* $\xrightarrow[G]{*d}$ *be the restriction of* $\xrightarrow[G]{*}$ *to* $K[\overline{x}]_d = \{p \in K[\overline{x}] | deg(HT(p)) < d\}$. *The following statements are equivalent:*

(1) *Every* $f \in I_d$ *has a unique normal form with respect to* $\xrightarrow[G]{*d}$.

(2) *For every* $f \in I_d$*, there is a reduction so that* $f \xrightarrow[G]{*d} 0$.

(3) *Every non-zero* $f \in I_d$ *is reducible by* G.

(4) *Every non-zero* $f \in I_d$ *is top-reducible by* G.

(5) *For every* $s \in HT(I_d)$ *there is a* $t \in HT(G)$ *such that* $t|s$.

(6) $HT(I_d) \subset \langle HT(G)\rangle$.

(7) *The polynomials* $h \in K[\overline{x}]_d$ *that are in normal form with respect to* G *form a system of representatives for the partition* $\{(f+I) \cap K[\overline{x}]_d | f \in K[\overline{x}]_d\}$ *of* $K[\overline{x}]_d$.

Proof. See [3]. □

Definition 2.5.3. *If G is a finite, homogeneous subset $K[\overline{x}]\backslash\{0\}$ that satisfies one of the equivalent conditions of Theorem 2.5.1 for some degree d, then G is a degree-d Grobner basis for $\langle G\rangle$.*

Finally we need to define a syzygy. This term comes from the Greek word meaning "yoke" [5]. We have already seen syzygies before in this paper, we just didn't use that terminology; in the term "S-polynomial", the "S" is short for syzygy.

Definition 2.5.4. *Let $G = (f_1, f_2, \ldots, f_m) \in K[\overline{x}]^m\backslash\{0\}$. A syzygy on the leading terms of the f_i's is an m-tuple $(p_1, p_2, \ldots, p_m) \in K[\overline{x}]^m$ such that*

$$\sum_{j=1}^{m} p_j HT(f_j) = 0$$

A syzygy on G is an m-tuple $(p_1, p_2, \ldots, p_m) \in K[\overline{x}]^m$ such that

$$\sum_{j=1}^{m} p_j f_j = 0$$

The symbol Syz represents the module of syzygies (where G is understood). We will say a principal syzygy is of the form $f_i f_j - f_j f_i$; we will denote the module generated by the principal syzygies $PSyz$ (where G is understood). Finally, G is called a regular sequence if $Syz = PSyz$ or, equivalently (and perhaps more usefully), if

$$gf_i \in \langle f_{i+1}, f_{i+2}, \ldots, f_m\rangle \Leftrightarrow g \in \langle f_{i+1}, f_{i+2}, \ldots, f_m\rangle$$

$\forall i$.

The notion of a syzygy plays a big part in our upcoming discussion of Faugere's F5 algorithm from [7]. Moreover, it is the relationship between syzygies and S-polynomials that lies at the heart of the proof of correctness of Buchberger's algorithm. We end with the following theorem.

Theorem 2.5.2. *Given $G = (f_1, f_2, \ldots, f_m) \in K[\overline{x}]^m\backslash\{0\}$, every syzygy on leading terms of G can be written as*

$$\sum_{i<j} u_{ij} S(f_i, f_j)$$

where $u_{ij} \in K[\overline{x}]$ $\forall i, j$.

In short, the S-polynomials on G actually do form a basis for all the cancelation of terms possible amongst the elements in $\langle G \rangle$.

Proof. See [5]. □

3. Faugere's F5

While Buchberger's algorithm is the foundation for Grobner-basis-algorithm theory, it is by no means the fastest method for finding Grobner bases for ideals. In fact it is terribly slow in general. Buchberger's algorithm's dependence on normal form computations is primarily responsible for the slow run times. Thus people have been looking for ways to remove as many normal form computations as possible.

This leads us to the F5 algorithm, first published in 2002 in [7]. This algorithm has become the standard for Grobner basis computation, as it runs orders of magnitude faster than other algorithms for many input sequences of polynomials. In this section we will introduce, develop and repair various aspects of the F5 algorithm.

3.1. **Signatures and Signed Polynomials.** Before we introduce the F5 algorithm and delve into a full examination of the criteria used within it, we must introduce some terminology that will be used extensively in the algorithm itself. The terminology was introduced (in its current form) in [7] by Faugere; it was later explored with more exposition by Stegers in [15]. Throughout this thesis, unless otherwise noted, we will strive to use the original notation(s) used by Faugere in [7] while maintaining the expository clarity of Stegers. This is the case here.

We begin by introducing some standard notation that will be used throughout this paper. Let K be a field and let $\mathcal{P} = K[x_1, x_2, \ldots, x_n]$. From now on, for ease of reading, we will denote $K[x_1, x_2, \ldots, x_n]$ as $K[\overline{x}]$. We will let T denote the set of terms in $K[\overline{x}]$, and we will say $<$ is some admissible term ordering on T. (Though not required, the reader should probably view $<$ as the degree-reverse-lexicographical ordering on T.)

Consider $(f_1, \ldots, f_m) \in \mathcal{P}^m$, where $\mathcal{P}^m$ is viewed as a $K[\overline{x}]$-module, and let the ideal generated the polynomials f_1 through f_m be $\langle f_1, \ldots, f_m \rangle = I$. We also assume that $(f_1, \ldots, f_m)$ are homogeneous (all the f_i's are homogeneous, not necessarily all of the same degree). We will let F_i represent the i^{th} canonical unit vector in

$\mathcal{P}^m$; in other words, $F_i = (0, 0, \ldots, 0, 1, 0, \ldots, 0)$, where the 1 is in the i^{th} position in the vector. (As a side note to the reader, the definitions of m and n here in this paper are standard within current Grobner-basis literature.) We will also let $T(I)$ be the set of head terms in the ideal I. Now we consider the map

$$v : \mathcal{P}^m \longrightarrow \mathcal{P}$$

by

$$g = (g_1, \ldots, g_m) \mapsto \sum_{i=1}^{m} g_i f_i$$

So, for example, $v(F_i) = v((0, 0, \ldots, 0, 1, 0, \ldots, 0)) = f_i$. We may also view an element $g \in \mathcal{P}^m$ as $\sum_{i=1}^{m} g_i F_i$. This introduces another way to discuss syzygies (see section 2.5). We can observe that $g \in \mathcal{P}^m$ is a syzygy if $v(g) = 0$.

Now we extend $\leq$, which is strictly a term ordering, into an admissible ordering $\prec$ on $\mathcal{P}^m$.

Definition 3.1.1. *Let $g, h \in \mathcal{P}^m$, with*

$$g = \sum_{k=i}^{m} g_k F_k, h = \sum_{k=j}^{m} h_k F_k$$

Then we will say $g \prec h$ if:

(1) *$i > j$ or*

(2) *$i = j$ and $HT(g_i) < HT(g_j)$*

(Note: Under this ordering, $g = h$ if $i = j$ and $HT(g_i) = HT(g_j)$.)

Then under this new ordering we have $F_1 \succ F_2 \succ \cdots \succ F_m$. Now we will define the notions of an index and a head term for an element $g \in \mathcal{P}^m$.

Definition 3.1.2. *Let $g \in \mathcal{P}^m$. Then $\exists$ i such that*

$$g = \sum_{k=i}^{m} g_k F_k$$

and $g_i \neq 0$. This natural number i will be defined as the index of (this representation of) g, and will be denoted $index(g)$. (Note that if g is represented differently as a sum of $F - k$'s, this index might change.)

Definition 3.1.3. *If $g \in \mathcal{P}^m$ with $index(g) = i$, then $HT(g) = HT(g_i)$. Moreover, the degree of g, denoted $deg(g)$, will be $max\{deg(g_i) + deg(f_i) | i \in \{1, 2, \ldots, m\}\}$. This definition equates, except in cases of term cancelation, $deg(g)$ with $deg(v(g))$.*

Now we need to introduce two more pieces of notation. Let $T_i = \{tF_i | t \in T\}$ and let $\mathbf{T} = \cup_{i=1}^{m} T_i$. Now we are ready.

Definition 3.1.4. *Let $t \in T(I)$ and consider the following map W:*

$$W : T(I) \longrightarrow A \subset \mathcal{P}^m$$

by

$$t \mapsto \{g \in \mathcal{P}^m | HT(v(g)) = t\}$$

In other words, W is the map that takes an element t of the monomial ideal of head terms of I and maps it to a subset of $\mathcal{P}^m$ that contains all elements whose image under v has head term t. This set, of course, may have cardinality greater than 1; thus we define a variant of W, w, where:

$$w : T(I) \longrightarrow \mathcal{P}^m$$

by

$$t \mapsto min_{\prec} W(t)$$

Finally we will define the map v_1:

$$v_1 : I \longrightarrow \boldsymbol{T}$$

by

$$p \mapsto HT(w(HT(p)))$$

Now that we have all this additional machinery with which to work, we will model the polynomials in F5 in such a way to make use of (i.e., keep record of) the additional data. It is this additional data that will permit the algorithm to ignore full normal-form reductions of polynomials, thus avoiding the most costly subroutines of Buchberger's algorithm (and other Grobner-basis-producing algorithms like F4 – see [8]). In this final definition from this section, we formalize this new polynomial construct.

Definition 3.1.5. *We will represent polynomials as elements in $R = \boldsymbol{T} \times \mathcal{P}$, and we call the elements of R signed polynomials. If $r = (tF_i, f) \in R$, we define:*

$$poly(r) = f$$
$$\mathcal{S}(r) = tF_i \in \boldsymbol{T}$$
$$index(r) = i$$

We will say that $\mathcal{S}(r)$ is the signature of r. (If it is understood, for some $r' \in R$, that $\mathcal{S}(r') = t'F_j$ and $poly(r') = f'$, then we will also say that the signature of f' is $t'F_j$.)

We say $r \in R$ is admissible if $\exists\ g \in v^{-1}(poly(r))$ such that $HT(g) = \mathcal{S}(r)$; it is admissibility that connects the signature of an element of R with that element's polynomial.

For nonzero $\lambda \in K$, $v \in T$, $t = wF_k \in \boldsymbol{T}$, $q \in K[\overline{x}]$ and $r = (uF_i, p) \in R$ we define the following arithmetic rules:

(1) $\lambda r = (uF_i, \lambda p)$

(2) $vt = vwF_k$

(3) $vr = (uvF_i, vp)$

(4) $qr = (HT(q)uF_i, qp)$

We also will define the following natural extensions (perhaps a more appropriate word would be "abuses") of notation on elements of R:

$$\text{for } r \in R,\ HT(r) = HT(poly(r))$$
$$\text{for } r \in R,\ HC(r) = HC(poly(r))$$
$$\text{for } r \in R \text{ and } G \subset \mathcal{P},\ NF(r, G) = (\mathcal{S}(r), NF(poly(r), G))$$

There are several remarks that should be made in order to maintain clarity and to link the above definitions (especially Definition 3.1.4 and Definition 3.1.5) together. Though these remarks are not required for correctness, they will motivate the definitions presented above.

For the following remarks, we will assume that $m = 2$, $n = 2$, $K = \mathbb{Q}$ and $I = \langle x^2y + y^2, 2xy - z\rangle$ with degree-reverse-lexicographical order on the terms ($x > y > z$), $f_1 = x^2y + y^2$ and $f_2 = 2xy - z$.

Remark 3.1.1. F5 will only use signed polynomials that are admissible because the signature is only useful if the signed polynomial is admissible. Using the above example, the signed polynomial $(yF_1, x^2y^2 + y^3 + 2xyz^4 - z^5)$ is admissible because $x^2y^2+y^3+2xyz^4-z^5 = yf_1+z^4f_2$. (Note that the head term of $x^2y^2+y^3+2xyz^4-z^5$ is $2xyz^4$, not x^2y^2.)

The signature of a signed polynomial need not have anything meaningful to say. For example, $r = (y^{3000}z^2F_2, x^2y^2 + y^3 + 2xyz^4 - z^5)$ is still an element of R; but it is clear that the $\mathcal{S}(r)$ has nothing to do with $poly(r)$.

Remark 3.1.2. Once we decide to focus on the cases where the signature of a signed polynomial is admissible (i.e., useful), the arithmetic rules introduced in

Definition 3.1.5 are easy to understand. We will go through each rule one-by-one, illustrating with our example.

(1) $\lambda r = (uF_i, \lambda p)$: We will use $(yF_1, x^2y^2 + y^3 + 2xyz^4 - z^5)$, the admissible signed polynomial from remark 3.1.1, and let $\lambda = 3$. Then we can see that $3(x^2y^2 + y^3 + 2xyz^4 - z^5) = 3yf_1 + 3z^4f_2$; from here it is clear that $(yF_1, 3(x^2y^2 + y^3 + 2xyz^4 - z^5))$ is also an admissible pair thus giving credence to this arithmetic rule.

(2) $vt = vwF_k$: This is an arithmetic fact that is motivated by the final fact. See below.

(3) $vr = (uvF_i, vp)$: Once again we will look at $(yF_1, x^2y^2+y^3+2xyz^4-z^5)$. Let $v = y$. Then it's easy to see that $y(x^2y^2 + y^3 + 2xyz^4 - z^5) = y^2f_1 + yz^4f_2$, which would be represented as a signed polynomial by $(y^2F_1, y(x^2y^2 + y^3 + 2xyz^4 - z^5))$. This is exactly what the arithmetic rule prescribes.

Thus these arithmetic rules truly do emulate the relationship between $\mathcal{S}(r)$ and $poly(r)$ for an admissible element $r \in R$.

Remark 3.1.3. It may be unclear what the derived map v_1 has to do with any of this – the astute reader will notice that we never used v_1 in the definition of anything subsequent to its inception. But we note here that $(v_1(p)F_{index(w(HT(p)))}, p)$ is an admissible element of R. Moreover, if the sequence of polynomials $(f_1, \ldots, f_m)$ is regular, then F5 will **only** use this signature for every polynomial appearing in the output Grobner basis! (We will prove this in Corollary 3.2.1).

Before moving on to the next section, there is one more piece of business to be addressed. In [7], Faugere states the following proposition:

Proposition 3.1.1. *If* $(t_1, t_2) \in T(I)^2$ *and* $t_1 \neq t_2$, *then* $HT(w(t_1)) \neq HT(w(t_2))$.

Neither [7] nor [15] proves Proposition 3.1.1. Given the relationship between the map w and the signature of an admissible polynomial, especially if the input sequence $(f_1, \ldots, f_m)$ is regular, it seems prudent to verify this proposition. A proof is given now:

Proof. Without loss of generality, let $t_1 > t_2$. Assume, for contradiction, that $HT(w(t_1)) = HT(w(t_2))$. Say that $HT(w(t_1)) = eF_i$.

Then $\exists\ g^1 \in \mathcal{P}^m$ such that $g^1_j = 0\ \forall\ j < i$, $HT(g^1_i) = e$ and $v(g^1) = p^1$ where $HT(p^1) = t_1$.

Similarly, $\exists\ g^2 \in \mathcal{P}^m$ such that $g^2_j = 0\ \forall\ j < i$, $HT(g^2_i) = e$ and $v(g^2) = p^2$ where $HT(p^2) = t_2$.

Now consider $g^1 - g^2 \in \mathcal{P}^m$. Note that $HT(g^1 - g^2) \prec eF_i$ (or, if not, since K is a field, we could adjust $HC(p^1)$ and $HC(p^2)$ so that this is true). But

$$\begin{aligned} v(g^1 - g^2) &= \sum_{k=1}^{m} (g^1_k - g^2_k) f_k \\ &= \sum_{k=i}^{m} (g^1_k - g^2_k) f_k \\ &= \sum_{k=i}^{m} g^1_k f_k - \sum_{k=i}^{m} g^2_k f_k \\ &= p^1 - p^2 \end{aligned}$$

and $HT(p^1 - p^2) = t_1$.

Thus $\exists\ g \in \mathcal{P}^m$ (namely $(g^1 - g^2)$) such that $HT(v(g)) = t_1$ and $g \prec w(t_1)$. This contradicts the minimality of the map w under $\prec$.

Thus we are forced to conclude that our assumption that $HT(w(t_1)) = HT(w(t_2))$ is incorrect.

This completes the proof.

□

Faugere also introduced this corollary to Proposition 3.1.1, whose proof is immediate by definition of v_1.

Corollary 3.1.1. *Let $p_1, p_2 \in I$ with $HT(p_1) \neq HT(p_2)$. Then $v_1(p_1) \neq v_1(p_2)$.*

Proof. Immediate.

□

3.2. **Signed Representations and Normalization.** In this section we pay special attention to definitions and properties that will play an especially important role in our application and understanding of the F5 algorithm. We will begin by extending the definition of a t-representation of a polynomial f introduced in Definition 2.3.3. Since we developed the notion of a signature in section 3.1, we would like to attach this new idea to representations of polynomials as well. To this end, and once again borrowing some notation from [7], consider the following definition:

Definition 3.2.1. *Let $P \subset R$, $r \in R$ and $t \in T$. Also assume that we have the following standard representation of $poly(r)$:*

$$poly(r) = \sum_{p \in P} q_p poly(p)$$

where $q_p \in K[\overline{x}]$ $\forall p \in P$.

Then the signature of the representation is defined to be

$$max_{poly(p) \in P}\{\mathcal{S}(q_p p)\}$$

In addition we define the above standard representation as a signed representation of r if the representation is a $HT(r)$-representation and $\mathcal{S}(r) \succeq \mathcal{S}(q_p p)$ $\forall p \in P$. This naturally yields itself to our definition of a signed t-representation (where $HT(r)$ in the above definition is replaced by a term t). If such a signed t-representation exists, we will say $poly(r) = O_P(t)$.

Finally, we say $poly(r) = o_P(t)$ *for some* $t \in T$ *if there is* $poly(r) = O_P(t')$ *and* $t' < t$ *in the term order.*

For the rest of the paper, all of our representations for various $r \in R$ will be signed representations. We will make some effort to specifically note that both requirements for signed representations are met, but we ask the reader to be generous in his/her interpretations.

To be clear with the definitions presented in Definition 3.2.1, we include the following remark. For the following remark, just as in the remarks 3.1.1, 3.1.2 and 3.1.3, assume that $m = 2$, $n = 2$, $K = \mathbb{Q}$ and $I = \langle x^2y + y^2, 2xy - z\rangle$ with degree-reverse-lexicographical order on the terms $(x > y > z)$, $f_1 = x^2y + y^2$ and $f_2 = 2xy - z$.

Remark 3.2.1. Consider the admissible $r \in R$, $r = (yF_1, x^2y^2 + y^3 + 2xyz^4 - z^5)$. Then a standard representation for $poly(r)$ would clearly be

$$poly(r) = yf_1 + z^4 f_2$$

This representation would have a signature of yF_1 because

$$max\{\mathcal{S}(yf_1), \mathcal{S}(z^4 f_2)\} = max\{yF_1, z^4F_2\} = yF_1$$

Since $\mathcal{S}(r) = yF_1$ as well and $HT(yf_1), HT(z^4f_2) < HT(r)$, this makes our representation a signed representation (or a signed xyz^4-representation); so we say $poly(r) = O_{\{f_1,f_2\}}(xyz^4)$.

However, our above representation does not witness that $poly(r) = o_{\{f_1,f_2\}}(xyz^4)$. (In theory, this property could be illustrated by another representation. In this case it is true that $poly(r) \neq o_{f_1,f_2}(xyz^4)$. But it should be pointed out that just because a specific representation for $poly(r)$ for some $r \in R$ does not illustrate $poly(r) = o_P(t)$ for some collection of polynomials P and some term t does not

mean $poly(r) \neq o_P(t)$; $poly(r) = o_P(t)$ is dependent upon r, P and t, not a specific representation of $poly(r)$.)

So what would be an example that illustrates this final definition from Definition 3.2.1? Instead, let us look at $S(f_1, f_2) = 2f_1 - xf_2 = 2x^2 - xz$. This means that an admissible signature for $S(f_1, f_2)$ is F_1. Let's define a new collection of polynomials, call it P', where $P' = \{f_1, f_2, 2x^2 - xz\}$. Then, under this new P', we would have $S(f_1, f_2) = o'_P(x^2y)$. In particular the representation that is witness to this is the most trivial, namely

$$S(f_1, f_2) = 1(2x^2 - xz)$$

Note that, because of the cancelation of head terms created by the S-polynomial, $HT(2x^2 - xz) < x^2y$.

This illustration is an accurate description of the strategy that F5 will use. In order to gain what will come to be known as the "little-o" condition (see the end of section 3.3), F5 will simply add S-polynomials to an ever-increasing collection of polynomials.

There is only one more critical definition to introduce before turning our attention to F5. Just like Definition 3.2.1, we are using our new machinery – the signature of a signed polynomial – to create new mathematical constructs. Unlike every other definition we have had, however, the next definition (a normalized polynomial) has no analog in previous Grobner basis algorithms. As we will see, the concept of a normalized polynomial is a direct result of a polynomial's signature and cannot be emulated without the signature itself. Thus, not surprisingly, it is an idea that plays a primary role in the functioning of the F5 algorithm.

After defining a normalized polynomial and several notational devices associated with the definition, we will also prove some useful facts associated with normalization. Once again, we base our definition upon the notation introduced by Faugere.

Definition 3.2.2. *Let $r \in R$. We will say that r is normalized if $\mathcal{S}(r) = eF_k$ and e is not top-reducible by $\langle f_{k+1}, \ldots, f_m \rangle$. We say that $(u, r) \in T \times R$ is normalized if ur (defined by Definition 3.1.5) is normalized. We will also say that $(r_1, r_2) \in R^2$ is normalized if $\mathcal{S}(r_1) \succ \mathcal{S}(r_2)$ and $(u_1, r_1), (u_2, r_2) \in T \times R$ are normalized where*

$$u_i = \frac{lcm(HT(r_1, r_2))}{HT(r_i)}$$

for $i = 1, 2$.

We will also define, as another convenient abuse of notation, that $\mathcal{S}(r)$ is normalized if r is normalized (when $\mathcal{S}(r)$ is understood).

We now present Lemma 3.2.1, the **Normalization Lemma**. This is one of the most important lemmas in this thesis. At the same time, it also highlights a specific property of non-normalized, admissible signed polynomials; namely that if the admissible $r \in R$ is non-normalized, there is a representation for $poly(r)$ that has a smaller signature.

Lemma 3.2.1. ***Normalization Lemma.*** *Consider an admissible $r \in R$, $r = (uF_i, p)$ where $i = index(w(HT(p)))$. (In other words, there does not exist an admissible $(u'F_j, p)$ such that $j > i$.) Also assume that uF_i is not normal. Then there exists (a canonical) $(u'F_i, p)$ that is normalized.*

Proof. Since $r = (uF_i, p)$ is admissible,

$$p = \sum_{k=i}^{m} g_k f_k$$

where $HT(g_i) = u$ and $g_k \in K[\overline{x}], i \leq k \leq m$.

Let $(b_1, \ldots, b_s) \in R^s$ such that $b_{\mathcal{L}}$ is admissible $\forall \mathcal{L}$ and $(b_1, \ldots, b_s)$ forms the Grobner basis for $\langle f_{i+1}, \ldots, f_m \rangle$. Then we can say

$$p = g_i f_i + \sum_{k=1}^{s} g'_k b_k$$

where $g'_k \in K[\overline{x}], 1 \leq k \leq s$.

We assumed that uF_i is not normalized, so u is top-reducible by $\langle f_{i+1}, \ldots, f_m \rangle$; thus, since $(b_1, \ldots, b_s)$ forms the Grobner basis for $\langle f_{i+1}, \ldots, f_m \rangle$, we know $\exists b_j$ such that $HT(b_j)|u$, say $u = HT(b_j)t$ for some term t.

Now we have:

$$p = (HT(b_j)t + q(\overline{x}))f_i + \sum_{k=1}^{s} g'_k b_k \tag{3.1}$$

$$= (q(\overline{x}))f_i + (tf_i)b_j - (t(b_j - HT(b_j)))f_i + \sum_{k=1}^{s} g'_k b_k$$

$$= (t(b_j - HT(b_j)) + q(\overline{x}))f_i + (tf_i)b_j + \sum_{k=1}^{s} g'_k b_k \tag{3.2}$$

where $q(\overline{x}) \in K[\overline{x}]$ and $HT(q(\overline{x})) < u$.

Note that this representation 3.2 of p has a smaller signature (either $HT(t(b_j - HT(b_j)))F_i$ or $HT(q(\overline{x}))F_i$) than the original representation 3.1 of p.

If $\mathcal{S}(3.2)$ is normalized, then we are done; otherwise, simply repeat the above process again. Since it is clear that this process of rewriting the representation of p leads to a decrease in signature at each step, we know that this process must eventually stop. We call the signature of this last representation of p $u'F_i$.

This completes the proof.

□

So what has been demonstrated is that if an admissible $r \in R$, $r = (uF_i, p)$, is not normalized then there is another admissible $r' \in R$, $r' = (u'F_i, p)$, such that $u'F_i \prec uF_i$. So in some sense being non-normalized implies that there must be a "better" representation. It is this interpretation of Lemma 3.2.1 that will be of use to us.

Then one might also ask: If $r, r' \in R$ are admissible, $r = (uF_i, p)$ and $r' = (u'F_i, p)$, with $u'F_i \prec uF_i$, does this mean that r is non-normalized. Unfortunately, this is

not always true. In particular, if the sequence of polynomials $(f_1, \ldots, f_m)$ is non-regular, it could very well be that both r and r' are normalized. However, if it is the case that $(f_1, \ldots, f_m)$ is a regular sequence, we would be able to conclude that r is non-normalized. We prove this now.

Theorem 3.2.1. *Let $r, r' \in R$ be admissible, $r = (uF_i, p)$ and $r' = (u'F_i, p)$, with $u'F_i \prec uF_i$. In addition, assume that $(f_1, \ldots, f_m)$ is a regular sequence. Then r is not normalized.*

Proof. Since r and r' are admissible, we know $\exists g, g' \in \mathcal{P}^m$ such that:

(1) $v(g) = v(g') = p$

(2) $HT(g) = u$

(3) $HT(g') = u'$

(4) $index(g) = index(g') = i$

This implies that $v(g - g') = 0$, meaning that $(g - g') \in Syz$. Since $(f_1, \ldots, f_m)$ is a regular sequence, $PSyz = Syz$ (making $(g - g') \in PSyz$).

Now we make the following observation. (This portion of the proof is taken from [7].) Let $s \in PSyz$ with $index(s) = i$. Then

$$\begin{aligned} s &= \sum_{k=i}^{m} \sum_{j=k+1}^{m} \lambda_{k,j} s_{k,j} \\ &= \sum_{k=i}^{m} \sum_{j=k+1}^{m} \lambda_{k,j} f_j F_k - \sum_{k=i}^{m} \sum_{j=k+1}^{m} \lambda_{k,j} f_k F_j \\ &= \sum_{j=i+1}^{m} \lambda_{i,j} f_j F_i + \sum_{k=i+1}^{m} q_k F_k \end{aligned}$$

where $\lambda_{k,j}, q_k \in K[\overline{x}] \forall k, j$.

This allows us to conclude that if $s \in PSyz$, and $HT(s) = u''F_i$, then $HT(f_{j'})|u''$ for some $j' > i$. We know that $HT(g-g') = uF_i$; since $(g-g') \in PSyz$, we conclude that $\exists j' > i$ such that $HT(j')|u$. This is the definition of r being non-normalized.

This completes the proof.

□

This leads immediately into the following corollary:

Corollary 3.2.1. *Assume $(f_1, \ldots, f_m)$ is a regular sequence. Let $r = (uF_i, p)$ be admissible. Then if r is normalized, $uF_i = v_1(p)$.*

Proof. Assume, for contradiction, that $uF_i \neq v_1(p)$. Then $\exists g' \in \mathcal{P}^m$ such that $v(g') = p$ and $HT(g') = v_1(p)$; this yields an admissible $r' \in R, r' = (v_1(p), p)$. By Definition 3.1.4, $uF_i \succ v_1(p)$. By Theorem 3.2.1, r is not normalized. This is a contradiction to the hypothesis that r is normalized.

Thus we are forced to conclude that our assumption that $uF_i \neq v_1(p)$ was incorrect.

This completes the proof.

□

This presents itself as an interesting chain of facts. This means that if we have a regular sequence of polynomials $(f_1, \ldots, f_m)$ and we wish to talk about signed polynomials that are:

(1) admissible (i.e., useful) and

(2) normalized (i.e., coming from the minimal representation under $\prec$)

both of which would seem to be the "preferred" (or perhaps even "canonical" is a fair word to use) properties to be using, then v_1 is the map that produces a unique signature.

At this point in our discussion the reader should take caution. While the signature seems to have quite a few nice properties so far, we must recall that these properties

have largely been generated by the assumption that our input sequence $(f_1, \ldots, f_m)$ is regular. That is a *very strong* hypothesis. So from a mathematics point of view, our work so far has a serious limitation. Counteracting this, we will see that from an applications standpoint, there is still quite a lot of value in the notion of a signature and a signed polynomial, especially when normalization is required.

3.3. **F5 Criterion.** In this section we re-introduce the F5 criterion for computing Grobner bases. This criterion was introduced by Faugere in [7] and discussed again in detail by Stegers in [15]. In [15], Stegers does a good job reproducing the arguments from Faugere, paying special attention to the various orderings Faugere uses in his presentation; however, Stegers uses different notation than Faugere. We will revert back to the notation(s) used in [7], while trying to maintain the expository clarity of [15].

First we need a definition.

Definition 3.3.1. *Let $F = (f_1, \ldots, f_m)$ be a list of monic polynomials in $K[\overline{x}]$ and let $G = (r_1, \ldots, r_{|G|}) \in R^{|G|}$ such that:*

(1) *$F \subset poly(G)$, $g_i = poly(r_i) \forall i$ and $G_1 = (g_1, \ldots, g_{|G|})$*

(2) *all the r_i are admissible and monic*

We define the new order $\leq_1$. For $f \in \langle G_1 \rangle$, let

$$\mathcal{V} = \{(s, \sigma) \in \mathcal{P}^{|G|} \times S_{|G|} | \sum_{i=1}^{|G|} s_i g_{\sigma(i)} = f, \mathcal{S}(s_1 r_{\sigma(1)}) \succ \cdots \succ \mathcal{S}(s_{|G|} r_{\sigma(|G|)})\}$$

Our new ordering will define $(s, \sigma) \leq_1 (s', \sigma')$. Say

$$v = (\mathcal{S}(s_1, r_{\sigma(1)}), \cdots, \mathcal{S}(s_{|G|} r_{\sigma(|G|)}))$$

and

$$v' = (\mathcal{S}(s'_1, r_{\sigma'(1)}), \cdots, \mathcal{S}(s'_{|G|} r_{\sigma'(|G|)}))$$

Then $(s, \sigma) \leq_1 (s', \sigma')$ *if:*

(1) $v <_{lex} v'$

(2) $v =_{lex} v'$ *and* $max_i HT(s_i g_{\sigma(i)}) < max_i HT(s'_i g_{\sigma'(i)})$

(3) $v =_{lex} v'$ *and* $t = max_i HT(s_i g_{\sigma(i)}) < max_i HT(s'_i g_{\sigma'(i)})$ *and* $|\{i | HT(s_i g_{\sigma(i)}) = t\}| < |\{i | HT(s'_i g_{\sigma'(i)}) = t\}|$

The ordering $\leq_1$ on the representations of f is a key idea throughout this paper, specifically in the proof of the F5 and F5t criterions (F5t is introduced in section 4.4).

Note that it is possible for two elements of $\mathcal{V}$ that are not the same to be equal under $\leq_1$; this will not be a problem in any of the applications of $\leq_1$, and the reader need not worry about such occurrences.

We are now prepared to prove Faugere's F5 criterion. We will begin with two lemmas.

Lemma 3.3.1. *Let* $F = (f_L, \ldots, f_m)$ *be a list of polynomials in* $K[\overline{x}]$, K *a field. Let* $G = (r_1, \ldots, r_{|G|}) \in R^{|G|}$ *such that:* $G_L = poly(G)$; $F \subset poly(G)$; *and all the* r_i *are admissible. Let* $f \in \langle G_L \rangle$,

$$f = \sum_{g_i \in G_L} s_i g_i$$

$s_i \in K[\overline{x}]$, *with the following conditions on the representation of* f*:*

(1) $HT(f) = t$

(2) $t' = max_{g_i \in G_L} HT(s_i g_i)$

(3) $t' > t$ *(in the term order)*

(4) *The above representation of* f *is minimal (under* $\leq_1$*) among all representations of* f *satisfying conditions (*1*) through (*3*).*

Then (s_i, r_i) *is normalized* $\forall r_i \in G$.

Proof. Assume $\exists \mathcal{L}$ such that $(s_{\mathcal{L}}, r_{\mathcal{L}})$ is not normalized. In other words, $\mathcal{S}(r_{\mathcal{L}}) = uF_k$ and $HT(s_{\mathcal{L}})u \in \langle f_{k+1}, \ldots, f_m \rangle$. Since all the r_i are admissible,

$$g_{\mathcal{L}} = \sum_{j=k}^{m} w_j f_j$$

where $HT(w_k) = u$. Then we can say the following about f:

$$\begin{aligned} f &= \sum_{j \neq k} s_j g_j + s_{\mathcal{L}} g_{\mathcal{L}} \\ &= \sum_{j \neq k} s_j g_j + s_{\mathcal{L}} \sum_{j=k}^{m} w_j f_j \\ &= \sum_{j \neq k} s_j g_j + s_{\mathcal{L}} w_k f_k + s_{\mathcal{L}} \sum_{j=k+1}^{m} w_j f_j \end{aligned}$$

By Lemma 3.2.1, we know that:

$$s_{\mathcal{L}} w_k f_k = \left(a + \sum_{\substack{r \in G \\ \mathcal{S}(r) \prec F_L}} \lambda_r poly(r) \right) f_k$$

where a is a polynomial in $K[\overline{x}]$, $HT(a) < HT(s_{\mathcal{L}} u)$ and $\lambda_r \in K[\overline{x}]$ $\forall r \in G$.

Note that conditions (1) through (3) still hold. Yet we created a new representation of f that is smaller under $\leq_1$. This contradicts condition (4) above.

Thus we must conclude that (s_i, r_i) is normalized $\forall r_i \in G$.

□

Lemma 3.3.2. *Adopt all notation and assumptions from Lemma 3.3.1. Let* $I = \{i | HT(s_i g_i) = t'\}$, $w = max\{\mathcal{S}(s_i r_i) | i \in I\}$ *and let* $J = \{i \in I | \mathcal{S}(s_i r_i) = w\}$. *Then* $|J| = 1$.

Proof. Assume for contradiction that $|J| \geq 2$. Let $k_1 = min\{i | i \in J\}$ and $k_2 = min\{i | i \in J \backslash \{k_1\}\}$. Let $w = uF_L$. So,

$$f = \sum_{i \neq k_1, k_2} s_i g_i + s_{k_1} g_{k_1} + s_{k_2} g_{k_2}$$

We will denote

$$g_{k_1} = \sum_{j=L}^{m} h_j^1 f_j, g_{k_2} = \sum_{j=L}^{m} h_j^2 f_j$$

This leaves us with the following chain of equalities, for some $c, d \in K$:

$$\begin{aligned}
f &= \sum_{i \neq k_1, k_2} s_i g_i + s_{k_1}(\sum_{j=L}^{m} h_j^1 f_j) + s_{k_2}(\sum_{j=L}^{m} h_j^2 f_j) \\
&= \sum_{i \neq k_1, k_2} s_i g_i + sk_1 h_L^1 f_L + s_{k_2} h_L^2 f_L + s_{k_1}(\sum_{j=L+1}^{m} h_j^1 f_j) + s_{k_2}(\sum_{j=L+1}^{m} h_j^2 f_j) \\
&= \sum_{i \neq k_1, k_2} s_i g_i + (cu + (s_{k_1} - HT(s_{k_1}))h_L^1 + HT(s_{k_1})(h_l^1 - HT(h_L^1)))f_L \\
&+ (du + (s_{k_2} - HT(s_{k_2}))h_L^2 + HT(s_{k_2})(h_l^2 - HT(h_L^2)))f_L \\
&+ s_{k_1}(\sum_{j=L+1}^{m} h_j^1 f_j) + s_{k_2}(\sum_{j=L+1}^{m} h_j^2 f_j) \\
&= \sum_{i \neq k_1, k_2} s_i g_i + ((c+d)u + (s_{k_1} - HT(s_{k_1}))h_L^1 + HT(s_{k_1})(h_l^1 - HT(h_L^1)))f_L \\
&+ ((s_{k_2} - HT(s_{k_2}))h_L^2 + HT(s_{k_2})(h_l^2 - HT(h_L^2)))f_L \\
&+ s_{k_1}(\sum_{j=L+1}^{m} h_j^1 f_j) + s_{k_2}(\sum_{j=L+1}^{m} h_j^2 f_j) \\
&= \sum_{i \neq k_1, k_2} s_i g_i + ((c+d)u + (s_{k_1} - HT(s_{k_1}))h_L^1 + HT(s_{k_1})(h_l^1 - HT(h_L^1)))f_L \\
&+ (s_{k_2} - HT(s_{k_2}))(\sum_{j=L}^{m} h_j^2 f_j) + (HT(s_{k_2}))(h_L^2 - HT(h_L^2))f_L \\
&+ s_{k_1}(\sum_{j=L+1}^{m} h_j^1 f_j) + (HT(s_{k_2}))(\sum_{j=L+1}^{m} h_j^2 f_j) \\
&= \sum_{i \neq k_1, k_2} s_i g_i + s'_{k_1} g_{k_1} + (s_{k_2} - HT(s_{k_2}))g_{k_2} - d(HT(s_{k_1}))(\sum_{j=L+1}^{m} h_j^1 f_j) \\
&+ (HT(s_{k_2}))(h_L^2 - HT(h_L^2))f_L
\end{aligned}$$

where $s'_{k_1} = s_{k_1} + dHT(s_{k_1})$.

This representation of f is smaller (under $\leq_1$) than $\sum_{i \neq k_1,k_2} s_i g_i + s_{k_1} g_{k_1} + s_{k_2} g_{k_2}$. This fact can be divined by looking at four separate cases:

(1) $f_L = g_x$ for some $x \neq k_1, k_2$ where $\mathcal{S}(s_x r_x) \succ \mathcal{S}(s_{k_1} r_{k_1})$. In this case, $(HT(s_{k_2}))(h_L^2 - HT(h_L^2))f_L$ can be "absorbed" into $\sum_{i \neq k_1,k_2} s_i g_i$ by altering s_x to $s_x + (HT(s_{k_2}))(h_L^2 - HT(h_L^2))$. This change to s_x does not change $\mathcal{S}(s_x r_x)$. Of the remaining summands in equation 4.4, $\mathcal{S}(s'_{k_1} g_{k_1}) = \mathcal{S}(s_{k_1} g_{k_1})$, $\mathcal{S}((s_{k_2} - HT(s_{k_2}))g_{k_2}) \prec \mathcal{S}(s_{k_2} g_{k_2})$ and $dHT(s_{k_1})(\sum_{j=L+1}^{m} h_j^1 f_j)$ has the smallest signature of the three. Thus this new representation of f is smaller (under $\leq_1$) than $\sum_{i \neq k_1,k_2} s_i g_i + s_{k_1} g_{k_1} + s_{k_2} g_{k_2}$.

(2) $f_L = g_{k_1}$. In this case the exact same argument as in (1) holds, except "absorption" would occur into $s'_{k_1} g_{k_1}$.

(3) $f_L = g_{k_2}$. In this case, f could have been written without $s_{k_1} g_{k_1}$ at all. This would have immediately led to a contradiction as in case (1).

(4) $f_L = g_x$ for some $x \neq k_1, k_2$ where $\mathcal{S}(s_x r_x) \prec \mathcal{S}(s_{k_1} r_{k_2})$. Then the exact same argument as in case (1) holds, except tracking the resulting signature of $s_x r_x$ after "absorption" is irrelevant (because $\mathcal{S}(s_x n_x) \prec \mathcal{S}(the new representation of f, under \leq_1)$).

This list covers all possible cases for f_L. Thus we are forced to conclude that our assumption that $|J| \geq 2$ is false.

Thus, $|J| = 1$.

□

Theorem 3.3.1. *Let $F = (f_1, \ldots, f_m)$ be a list of monic polynomials in $K[\overline{x}]$ and let $G = (r_1, \ldots, r_{|G|}) \in R^{|G|}$ such that:*

(1) *$F \subset poly(G)$, $g_i = poly(r_i) \forall i$ and $G_1 = (g_1, \ldots, g_{|G|})$*

(2) *all the r_i are admissible and monic*

(3) $\forall (i,j) \in \{1,2,\ldots,|G|\}^2$ *such that* (r_i, r_j) *is normalized,* $S(g_i, g_j) = o_G(u_i r_i)$ *(or* 0*), where* $u_i = \frac{lcm(HT(g_i), HT(g_j))}{HT(g_i)}$

Then G_1 *is a Grobner basis of* $\langle f_1, \ldots, f_m \rangle$.

Proof. Let $f \in \langle G_1 \rangle$. We will show that f is top-reducible by G_1, thus proving that G_1 is a Grobner basis of $\langle f_1, \ldots, f_m \rangle$.

Let $(s, \sigma) \in \mathcal{P}^{|G|} \times S_{|G|}$ be a minimal representation of f under $\leq_1$, as defined in Definition 3.3.1. Without loss of generality, assume $\sigma = id$ by renumbering G. Let $t = max_i HT(s_i g_i)$, $I = \{i | HT(s_i g_i) = t\}$ and $r = |I|$.

Assume, for contradiction, that $t > HT(f)$ (under the term order). Note that if $t = HT(f)$, then f is top-reducible by G_1 and we are done immediately. Then we know that $r \geq 2$ because there must be at least one syzygy of head terms in the representation of f.

By Lemma 3.3.1 we know that (s_i, r_i) is normalized for all $i \in G$.

Let $w = max\{\mathcal{S}(s_i r_i) | i \in I\}$ and $J = \{i \in I | \mathcal{S}(s_i r_i) = w\}$. By Lemma 3.3.2, we know that $|J| = 1$.

Let $k \in J$ and $\mathcal{L} \in I \backslash \{k\}$; recall that $r \geq 2$, so such an $\mathcal{L}$ must exist. By construction we have $\mathcal{S}(s_k r_k) \succ \mathcal{S}(s_\mathcal{L} r_\mathcal{L})$. We rewrite f in the following representation derived from (s, σ):

$$f = s_k g_k - \frac{HC(s_k)}{HC(s_\mathcal{L})} s_\mathcal{L} g_\mathcal{L} + \left[1 + \frac{HC(s_k)}{HC(s_\mathcal{L})}\right] s_\mathcal{L} g_\mathcal{L} + \sum_{i \neq k, \mathcal{L}} s_i g_i \tag{3.3}$$

Let

$$m_k = HM(s_k)$$

and

$$m_{\mathcal{L}} = \frac{HC(s_k)}{HC(s_{\mathcal{L}})} HM(s_{\mathcal{L}})$$

and, for all i,

$$s_i' = s_i - HM(s_i)$$

Thus $t = HT(m_k g_k) = HT(m_{\mathcal{L}} g_{\mathcal{L}})$, so $lcm(HT(g_k), HT(g_{\mathcal{L}}))$ divides t.

Now, for notational ease, define $\tau_{k,\mathcal{L}} = lcm(HT(g_k), HT(g_{\mathcal{L}}))$. Then we have the following series of equalities:

$$S(g_k, g_{\mathcal{L}}) = \frac{\tau_{k,\mathcal{L}}}{HM(g_k)} g_k - \frac{\tau_{k,\mathcal{L}}}{HM(g_{\mathcal{L}})} g_{\mathcal{L}}$$

$$\Leftrightarrow \frac{tHC(s_k)}{\tau_{k,\mathcal{L}}} S(g_k, g_{\mathcal{L}}) = tHC(s_k) \left(\frac{g_k}{HM(g_k)} - \frac{g_{\mathcal{L}}}{HM(g_{\mathcal{L}})} \right)$$

$$\Leftrightarrow \frac{tHC(s_k)}{\tau_{k,\mathcal{L}}} S(g_k, g_{\mathcal{L}}) = m_k g_k - m_{\mathcal{L}} g_{\mathcal{L}}$$

Since (s_k, g_k) and $(s_{\mathcal{L}}, g_{\mathcal{L}})$ are normalized, so is $(g_k, g_{\mathcal{L}})$. Thus by hypothesis (3), we have:

$$m_k g_k - m_{\mathcal{L}} g_{\mathcal{L}} = \frac{t}{\tau_{k,\mathcal{L}}} o_G(u_k r_k)$$

where $u_k = \frac{\tau_{k,\mathcal{L}}}{HT(r_k)}$. This implies $m_k g_k - m_{\mathcal{L}} g_{\mathcal{L}} = o_G(s_k r_k)$.

Returning to our rewrite of f in equation 3.3, we can now say:

$$\begin{aligned} f &= HM(s_k) g_k + s_k' g_k - \frac{HC(s_k)}{HC(s_{\mathcal{L}})} HM(s_{\mathcal{L}} g_{\mathcal{L}}) - \frac{HC(s_k)}{HC(s_{\mathcal{L}})} s_{\mathcal{L}}' g_{\mathcal{L}} \\ &\quad + \left(1 + \frac{HC(s_k)}{HC(s_{\mathcal{L}})}\right) s_{\mathcal{L}} g_{\mathcal{L}} + \sum_{i \neq \mathcal{L}, k} s_i g_i \\ &= o_G(s_k r_k) + s_k' g_k - \frac{HC(s_k)}{HC(s_{\mathcal{L}})} s_{\mathcal{L}} g_{\mathcal{L}} + \left(1 + \frac{HC(s_k)}{HC(s_{\mathcal{L}})}\right) s_{\mathcal{L}} g_{\mathcal{L}} + \sum_{i \neq \mathcal{L}, k} s_i g_i \end{aligned}$$

This is a representation of f that is derived from an element $(s', \sigma') \in \mathcal{P}^{|G|} \times S_{|G|}$ less than (s, σ) under $\leq_1$. This is a contradiction.

Thus we are forced to conclude that our assumption that $t > HT(f)$ (under the term order) was incorrect. This means that $t = HT(f)$, which in turn means that f is top-reducible by G_1.

Since f was arbitrary, we conclude G_1 is a Grobner basis for $\langle f_1, \ldots, f_m \rangle$.

□

We will occasionally refer to condition (3) of this theorem as the "little-o" condition.

Note that, unlike previous criteria for generating Grobner bases, Faugere's F5 criterion does not require a mandatory normal form reduction of all prospective polynomials to be added to the basis. Rather the signature of the polynomials is used as a guide for keeping, removing and reducing critical pairs and polynomials.

3.4. **The F5 Algorithm.** At this point we are ready to look at the algorithm for F5. We will begin by giving the pseudocode for F5. Then we will give an example runthrough for a basic regular input sequence. (Note to the reader: the example we give in 3.4.2 will be of great importance later.) The final three pieces of this subsection (3.4.3, 3.4.4 and 3.4.5) are all meant to be looked at after examining the pseudocode.

At the risk of being long-winded, and since this document has more space allotted than Faugere's original had, we will be fairly detailed in the pseudocode. In particular, we will pay special attention to how lists and lists of lists are organized. We will begin our discussion with a final definition of an F5-specific object.

Definition 3.4.1. *In the language of the pseudocode of F5, the critical pair of* $(r_1, r_2) \in R^2$ *is a* $5-$*tuple* $(t, u_1, r_1, u_2, r_2) \in T^2 \times R \times T \times R$ *such that* $t = u_1 HT(r_1) = u_2 HT(r_2) = lcm(HT(r_1), HT(r_2))$ *and* $\mathcal{S}(u_1 r_1) \prec \mathcal{S}(u_2 r_2)$. *We will define the degree of a critical pair* (t, u_1, r_1, u_2, r_2) *to be* $deg(t)$.

3.4.1. *Pseudocode for F5.* Here we will use (with minor corrections and notational changes) the pseudocode introduced by Faugere in [7]. Some changes were suggested by Stegers in [15].

The outside shell of the F5 algorithm is controlled by the routine *Incremental F5.*

Incremental_F5

Input: A sequence $F = (f_1, \ldots, f_m)$ *of homogeneous polynomials under an admissible term ordering.*

Output: A set of signed polynomials $poly(G_1)$ *where* $poly(G_1)$ *is a Grobner basis for* F*.*

$N := m$;

Reset_simplification_rules(m);

$r_m := (F_m, f_m)$;

$G_m := [r_m]$;

for i: $(m-1) \cdots 1$ do

  G_i:=Algorithm_F5(i, f_i, G_{i+1});

return($poly(G_1)$);

(Please note that N is a global variable, and can be accessed/altered by any subroutine.)

Let's first turn our attention to *Algorithm_F5.*

Algorithm_F5

Input: i *an integer;* f_i *a polynomial;* G_{i+1} *a collection of signed polynomials such that* $poly(G_{i+1})$ *is a Grobner basis for* $\langle f_{i+1}, \ldots, f_m \rangle$*.*

Output: A set of signed polynomials G_i *where* $Poly(G_i)$ *is a Grobner basis for* $\langle f_i, \ldots, f_m \rangle$*.*

$r_i := (F_i, f_i)$;

$\phi_{i+1} := NF(\cdot, poly(G_{i+1}))$;

$G_i := G_{i+1} \cup \{r_i\}$;

$P :=$Deg_Sort([CritPair$(r_i, r, i, \phi_{i+1}, \phi_{index(r)+1})|r \in G_{i+1}]$);

while $P \neq \emptyset$ do

  $d := deg(P[1])$;

  $P_d := \{p \in P|deg(p) = d\}$;

  $P := P \backslash P_d$;

  $F :=$SPol(P_d);

  $R_d :=$Reduction(F, G_i, i, ϕ_{i+1});

  for $r \in R_d$ do

    $P := P \cup \{$CritPair$(r, p, i, \phi_{i+1}, \phi_{index(p)+1}|p \in G_i\}$;

    $G_i := G_i \cup \{r\}$;

  Deg_Sort(P);

return G_i;

This is the portion of the algorithm that is called $(m-1)$ times and at the end of each call a new Grobner basis is produced. After the first call to this algorithm, the Grobner basis for $\langle f_m, f_{m-1}\rangle$ is returned. In general, after the k^{th} call to *Algorithm_F5*, the Grobner basis for $\langle f_m, \ldots, f_{m-k}\rangle$. This is why F5 is called an iterative algorithm.

The process used by *Algorithm_F5* is similar to many Grobner basis algorithms: it moves degree-by-degree (see section on *Deg_Sort*); it generates a new set of S-polynomials F to consider (see sections on *CritPair* and *SPol*); it reduces this new set F of S-polynomials by G_i and ϕ_{i+1} (see section on *Reduction*).

Just for clarity, even though it is intuitively obvious what *Deg_Sort* does, we include it's description below.

Deg_Sort

Input: A list of critical pairs of the form (t, u_1, r_1, u_2, r_2).

Output: A list of critical pairs of the form (t, u_1, r_1, u_2, r_2) *given in ascending order with respect to the degree of the critical pairs. Thus the first element in the list has minimum degree.*

Now we present the subroutine *CritPair*.

CritPair

Input: k *an integer;* $r_1, r_2 \in R^2$;$\phi_{index(r_1)+1}, \phi_{index(r_2)+1}$ *normal form mappings.*

Output: A critical pair (t, u_1, r_1, u_2, r_2) *or* $\emptyset$.

$t := lcm(HT(r_1), HT(r_2))$;

$u_1 := \frac{t}{HT(r_1)}$;

$u_2 := \frac{t}{HT(r_2)}$;

if $\mathcal{S}(u_1 r_1) \prec \mathcal{S}(u_2 r_2)$ then

  $tempterm := u_1$;

  $u_1 := u_2$;

  $u_2 := tempterm$;

  $tempr := r_1$;

  $r_1 := r_2$;

  $r_2 := tempr$;

if $(index(r_1) > k)$ then

  return $\emptyset$;

if $\phi_{index(r_1)+1}(u_1 HT(r_1)) \neq u_1 HT(r_1)$ then

  return $\emptyset$;

if $\phi_{index(r_2)+1}(u_2 HT(r_2)) \neq u_2 HT(r_2)$ then

  return $\emptyset$;

return (t, u_1, r_1, u_2, r_2);

It is the subroutine *CritPair* that is responsible for imposing the F5 Criterion from Theorem 3.3.1. Note that in condition (3) of Theorem 3.3.1, it is required that all pairs (r_i, r_j) be normalized. The reader will recall from Definition 3.2.2 that a pair is normalized if:

(1) $\mathcal{S}(r_1) = e_1 F_{k_1}$ is not top-reducible by $\langle f_{k_1+1}, \ldots, f_m \rangle$

(2) $\mathcal{S}(r_2) = e_2 F_{k_2}$ is not top-reducible by $\langle f_{k_2+1}, \ldots, f_m \rangle$

(3) $\mathcal{S}(u_1 r_1) \succ \mathcal{S}(u_2 r_2)$

If these three conditions are not met in *CritPair* (note that the third condition will always be met because *CirtPair* forces it to be met), the nominated critical pair is dropped and $\emptyset$ is returned. In addition, there is one more line in *CritPair* that could cause the nominated pair to be dropped:

if $(index(r_1) > k)$ then
  return $\emptyset$;

This is for a very simple reason: if $(index(r_1) > k)$ then $S(r_1, r_2) \in \langle f_{k+1}, \ldots, f_m \rangle$ and the critical pair created by (r_1, r_2) is thus not needed. In [15] this line was removed from the pseudocode. That removal is fine, but was not proven; we will prove it is okay to remove this line of code in subsection 4.1.

Once we have collected the nominated critical pairs that pass the F5 criterion test of *CritPair*, we send them to *SPol*.

SPol

Input: A list $[p_1, p_2, \ldots, p_h]$ *of critical pairs of the form* (t, u_1, r_1, u_2, r_2).
Output: A list F *of new, signed polynomials that is sorted in ascending order by* $\mathcal{S}$.

for $\mathcal{L} = 1$ to h do

  $p_{\mathcal{L}} := (t_{\mathcal{L}}, u_{\mathcal{L}}, r_{i_{\mathcal{L}}}, v_{\mathcal{L}}, r_{j_{\mathcal{L}}})$;

$F := \emptyset$;

for $\mathcal{L} = 1$ to h do

  if (not Rewritten?$(u_{\mathcal{L}}, r_{i_{\mathcal{L}}})$ and not Rewritten?$(v_{\mathcal{L}}, r_{j_{\mathcal{L}}})$) then

    $N := N + 1$;

    $r_N := (u_{\mathcal{L}}\mathcal{S}(r_{i_{\mathcal{L}}}), u_{\mathcal{L}}poly(r_{i_{\mathcal{L}}}) - v_{\mathcal{L}}poly(r_{j_{\mathcal{L}}}))$;

    Add_Rule(r_N);

    $F := F \cup \{r_N\}$;

$F :=$Sig_Sort(F);

return F;

Though at first glance this subroutine may look complicated, *SPol* essentially does one thing: form the new S-polynomials output from *CritPair* as admissible signed polynomials. We note that, because *CritPair* ensured that $\mathcal{S}(u_1 r_1) \succ \mathcal{S}(u_2 r_2)$, we know that the signature of all new polynomials will always be of the form $u_{\mathcal{L}}\mathcal{S}(r_{i_{\mathcal{L}}})$ in *SPol*.

At this point the number of subroutines called upon that have not been introduced is beginning to mount. We have now seen *Reset_simplification_rules*, *Rewritten?* and *Add_Rule* in the pseudocode. The reader is encouraged to make note of these but to ignore them for the time being. These three subroutines are all related, and will be dealt with in due course.

Just as with *Deg_Sort*, we give the description of *Sig_Sort*.

Sig_Sort

Input: A list F of admissible R-elements.

Output: A list F of admissible R-elements given in ascending order with regards to signature ($\prec$).

Now we introduce F5's variant on the well-known, time-intensive portion of the Grobner-basis-producing process. In F5 the subroutine is called *Reduction*. *Reduction* has several sub-subroutines that it uses. We will present all of these at once, and then give a full discussion afterwards.

Reduction

Input: $ToDo$ and G_i finite lists of signed polynomials; k an integer;ϕ_{i+1} a normal form mapping.

Output: $Done$ a finite list of reduced signed polynomials.

$Done := \emptyset$;

while $ToDo \neq \emptyset$ do

  h :=Sig_Sort($ToDo$)[1];

  $ToDo := ToDo \backslash \{h\}$;

  $(h_1, ToDo_1)$:=TopReduction($\phi_{i+1}(h), G_i \cup Done, k, \phi_{i+1}$);

  $Done := Done \cup h_1$;

  $ToDo := ToDo \cup ToDo_1$;

return $Done$;

TopReduction

Input: r_{k_0} a signed polynomial; G_i a finite list of signed polynomials; k an integer;ϕ_{i+1} a normal form mapping.

Output: An ordered pair, each either being a list of signed polynomials.

if $poly(r_{k_0}) = 0$ then

  print("Warning! Input sequence is not regular!");

  return $(\emptyset, \emptyset)$;

r' :=IsReducible?$(r_{k_0}, G_i, k, \phi_{i+1})$;

if $r' = \emptyset$ then

  $r_{k_0} := (\frac{1}{HC(r_{k_0})}) r_{k_0}$;

return $(r_{k_0}, \emptyset)$;

else

$r_{k_1} := r'$;

$u = \frac{HT(r_(k_0))}{HT(r_{k_1})}$;

if $(u\mathcal{S}(r_{k_1}) \prec \mathcal{S}(r_{k_0}))$ then

$poly(r_{k_0}) = poly(r_{k_0}) - u\ poly(r_{k_1})$;

return $(\emptyset, r_{k_0})$;

else

$N := N + 1$;

$r_N := (u\mathcal{S}(r_{k_1}), u\ poly(r_{k_1}) - poly(r_{k_0}))$;

Add_Rule(r_N);

return $(\emptyset, \{r_{k_0}, r_N\})$;

IsReducible?

Input: r_{k_0} a signed polynomial; $G_i = [r_1, \ldots, r_{|G_i|}]$ a finite list of signed polynomials; k an integer;ϕ_{i+1} a normal form mapping.

Output: Either a signed polynomial or the $\emptyset$.

for $j = 1$ to $|G_i|$ do

$t_j F_{k_j} := \mathcal{S}(r_j)$;

for $j = 1$ to $|G_i|$ do

if $((u := \frac{HT(r_{k_0})}{HT(r_{k_j})} \in T)$ and $(\phi_{i+1}(ut_j) = ut_j)$ and

(not Rewritten?(u, r_{k_j})) and $(ut_j F_{k_j} \neq \mathcal{S}(r_{k_0})))$ then

return r_{k_j};

return $\emptyset$;

Let's begin our discussion by focusing our attention to the outer layer of the reduction subroutine(s): *Reduction*. The signed polynomial with smallest signature, denoted h, is grabbed and removed from the $ToDo$ list of polynomial to be reduced.

It's normal form, with respect to the previous Grobner basis, and other information is sent to the sub-subroutine *TopReduction*. If *TopReduction* determines that the signed polynomial can be reduced, then nothing will be added to *Done* and the reduced (still signed) version of h will be placed back into $ToDo$. If no top reduction is possible, h is made monic by K-multiplication and the resulting signed polynomial is placed in *Done*.

This description of *Reduction* seems very similar to other reduction routines from other algorithms. The difference lies in the phrase, "If *TopReduction* determines that the signed polynomial can be reduced ..." So we must examine how *TopReduction* makes this decision.

We will go through *TopReduction* step-by-step. If the signed polynomial being examined has polynomial part 0, then there is no data left in that particular signed polynomial – an empty ordered pair is returned. Otherwise *TopReduction* calls upon another sub-subroutine *IsReducible?*. Essentially, if *IsReducible?* comes back negative, the current signed polynomial is made monic and returned to *Reduction* to be placed in *Done*. If a top-reduction is deemed possible, then there are two possible cases: either the reduction will increase the signature of polynomial or it won't. In the latter case, the signature of r_{k_0} is maintained, the polynomial portion is top-reduced and the signed polynomial is returned to *Reduction* to be added back into $ToDo$; this case corresponds to top-reduction in previous algorithms. In the former case, however, the signature will change. This is marked by adding a new polynomial r_N (our notation here describes N after N was incremented) with appropriate signature based upon the reductor, not $\mathcal{S}(r_{k_0})$. A new rule is added (as I mentioned previously, this will be explained later) and then **both** r_{k_0} and r_N are sent back to *Reduction* to be added back into $ToDo$. This is done because r_N has a different signature that r_{k_0} and r_{k_0} might still be reducible by another signed polynomial.

All this discussion still hasn't gotten us a good answer to our question: "What are the criteria for F5 top-reducing a polynomial?" We just analyzed *TopReduction* looking for an answer, but *TopReduction* pawned it off on the third and final sub-subroutine: *IsReducible?*. Now we look at *IsReducible?* in detail.

For a previously added signed polynomial in G_i to become a reductor of r_{k_0}, it must meet four requirements:

(1) $u := \frac{HT(r_{k_0})}{HT(r_{k_j})} \in T$

(2) $\phi_{i+1}(ut_j) = ut_j$

(3) not Rewritten?(u, r_{k_j})

(4) $ut_j F_{k_j} \neq \mathcal{S}(r_{k_0})$

We will go through each requirement one-by-one.

Requirement (1) is simply the normal top-reduction requirement. The only thing of note here is that, in testing for the top-reducibility, u is assigned a particular value to be used in subsequent tests.

Requirement (2) is making sure that the signature of the reductor is normalized. Recall that we only want signatures of our polynomials to be normalized – we are discarding non-normalized S-polynomials. If we ignored this condition and our reductor wound up having larger signature than $\mathcal{S}(r_{k_0})$, then *TopReduction* would create a new signed polynomial with our reductor's non-normalized signature. (We might add that, if the reductor had **smaller** signature than $\mathcal{S}(r_{k_0})$, it would be fine to reduce by it; however, F5 doesn't miss anything by forgoing this opportunity because, by Lemma 3.2.1 (The Normalization Lemma), there will be another normalized reductor with the same head term and smaller signature.)

Requirement (3) will be discussed when we discuss *Rewritten?*. That discussion is approaching rapidly.

Requirement (4) is a check that makes sure we don't reduce by something that has the same signature as r_{k_0}. Recall that we want all signed polynomials used during the run of F5 to be admissible. If we reduced by a polynomial that has the same signature, we would be left with a new polynomial for which we would have no idea what the signature is. The act of reduction would have certainly lowered the signature, thus causing admissibility to be lost. (We will comment on this requirement later in subsection 3.5. With a little care, we can loosen this requirement.)

This leaves us with these final subroutines to discuss: *Reset_simplification_rules*, *Add_Rule*, *Rewritten* and *Rewritten?*. All of these ideas are related to one subroutine – *Rewritten*.

Rewritten gives us information to be used as an additional criterion for eliminating critical pairs. Proof of this fact is given in section 3.4.3. In short, we could remove all discussion of rules and *Rewritten* and F5 would work fine. (But it would work much more slowly.) So we will treat these final four subroutines as a separate module that works in conjunction with F5, but is not an official part of the F5 criteria per se.

We begin by making the following definition:

Definition 3.4.2. *A rule in F5 is an element $(t, k) \in T \times \mathbb{N}$, where $\exists$ a signed polynomial r_k that F5 has added to the Grobner basis (or has reduced to 0) such that $\mathcal{S}(r_k) = tF_j$ for some natural number j.*

During each run of F5, the algorithm keeps a global list of lists called *Rule*. There is one sublist for each index running from 1 to m. Now that we have established that F5 is doing this, we can introduce the following:

Reset_simplification_rules

Input: m, the number of polynomials in the input sequence

Output: A list Rule of m empty sublists.

for $i = 1$ to m do
$Rule[i] := []$;

It is not an accident that there are exactly m sublists. Every time a new signed polynomial of index j is added, a new rule is added to $Rule[j]$ – the j^th sublist in *Rule*. As we look back at the pseudocode we have seen, we will note that every time a new signed polynomial is added to G_i, we call *Add_Rule*.

Add_Rule

Input: A signed polynomial $r_k = (tF_i, p)$.
Output: Rule has been updated in sublist $Rule[i]$.

$Rule[i] := concat(Rule[i], [(t, k)])$;

It is important to note that this concatenation is done *on the end* of the sublist $Rule[i]$, not at the front. This will play a very important role now that we have gotten to *Rewritten*. A proof of *Rewritten* and it's nature (i.e., why it works) is discussed at some length in section 3.4.3. Here we will just introduce the pseudocode for the remaining subroutines.

Rewritten

Input: A term u; a signed polynomial $r_k = (tF_i, p)$.
Output: An ordered pair from $T \times R$.

for $j = 1$ to $|Rule[i]|$ do
  $(t_j, k_j) := Rule[i][j]$;
for $j = |Rule[i]|$ to 1 do
  if $t_j | ut$ then

return $(\frac{ut}{t_j}, r_{k_j})$;

return (u, r_k);

Rewritten?

Input: A term u; a signed polynomial $r_k = (tF_i, p)$.

Output: True or False.

$(v, r_{\mathcal{L}})$:=Rewritten(u, r_k);

if $(\mathcal{L} \neq k)$ then

  return true;

else

  return false;

3.4.2. *Example Run of F5.* In this subsection we will trace an example run of F5 with a small $(m = 3, n = 4)$ input sequence. This is the same example given in [7] but we will trace it in more detail, fixing a few typographical errors along the way. This example is also of particular importance because it will be used as a reference later in the paper.

We will begin with the following input:

$$f_3 = x^2y - z^2t$$
$$f_2 = xz^2 - y^2t$$
$$f_1 = yz^3 - x^2t^2$$

while using degree-reverse-lexicographical ordering on the terms, $x > y > z > t$ (in this framework, it would be fair to say that t appears to have the role of a homogenizing variable). We now begin the trace.

$N := 3$

$Rule := [[], [], []]$

$r_3 := (F_3, f_3)$

$G_3 := [r_3]$

Now we will begin the first iteration of the loop from *Incremental_F5*. Note that we already have the Grobner basis of $\langle f_3 \rangle$ (trivially) stored in G_3.

Algorithm_F5$(2, f_2, G_3)$

$r_2 := (F_2, f_2)$

$\phi_3 := NF(\cdot, G_3)$

$G_2 := [r_3, r_2]$

CritPair for r_2 and G_3 generates the following pair:

$p_1 := (x^2yz^2, xy, r_2, z^2, r_3)$

Now we have $d := 5$, $P_5 := [p_1]$ and $P := \emptyset$

SPol(P_5)

$p_1 \to r_4 := (xyF_2, -xy^3t + z^4t)$

Add a rule so that $Rule := [[], [(xy, 4)], []]$

Reduction on this new set:

There are no top-reducers eligible for r_4; just make it monic.

$r_4 := (xyF_2, xy^3t - z^4t)$

$R_5 := [r_4]$

CritPair for R_5 and G_2 generates the following pairs:

$p_2 := (x^2y^3t, x, r_4, y^2t, r_3)$, $p_3 := (xy^3z^2t, z^2, r_4, y^3t, r_2)$

Note that p_2 is eliminated because $\phi_3(x\mathcal{S}(r_4)) \neq x\mathcal{S}(r_4)$.

$G_2 := [r_3, r_2, r_4]$

Now we have $d := 7$, $P_7 := [p_3]$ and $P := \emptyset$

SPol(P_7)

$p_3 \to r_5 := (xyz^2F_2, -z^6t + y^5t^2)$

Add a rule so that $Rule := [[], [(xy, 4), (xyz^2, 5)], []]$

Reduction on this new set:

There are no top-reducers eligible for r_5; just make it monic.

$r_5 := (xyz^2F_2, z^6t - y^5t^2)$

$R_7 := [r_5]$

CritPair for R_7 and G_2 generates the following pairs:

$p_4 := (x^2yz^6t, x^2y, r_5, z^6t, r_3)$, $p_5 := (xz^6t, x, r_5, z^4t, r_2)$,

$p_6 := (xy^3z^6t, xy^3, r_5, z^6, r_4)$

Note that p_4 is eliminated because $\phi_3(x^2y\mathcal{S}(r_5)) \neq x^2y\mathcal{S}(r_5)$.

Note that p_5 is eliminated because $\phi_3(x\mathcal{S}(r_5)) \neq x\mathcal{S}(r_5)$.

Note that p_6 is eliminated because $\phi_3(xy^3\mathcal{S}(r_5)) \neq xy^3\mathcal{S}(r_5)$.

$G_2 := [r_3, r_2, r_4, r_5]$

return G_2 ($P = \emptyset$, so we are done)

At this stage we have the Grobner basis G_2 for $\langle f_2, f_3\rangle$. We have one more iteration of *Algorithm_F5* left – index 1.

Algorithm_F5$(1, f_1, G_2)$

$r_1 := (F_1, f_1)$

$\phi_2 := NF(\cdot, G_2)$

$G_1 := [r_3, r_2, r_4, r_5, r_1]$

CritPair for r_1 and G_2 generates the following pairs:

$p_7 := (x^2yz^3, x^2, r_1, z^3, r_3)$, $p_8 := (xyz^3, x, r_1, yz, r_2)$,

$p_9 := (xy^3z^3t, xy^2t, r_1, z^3, r_4)$, $p_{10} := (yz^6t, z^3t, r_1, y, r_5)$

Now we have $d := 5$, $P_5 := [p_8]$ and $P := [p_7, p_9, p_10]$

SPol(P_5)

$p_8 \to r_6 := (xF_1, y^3zt - x^3t^2)$

Add a rule so that $Rule := [[(x, 6)], [(xy, 4), (xyz^2, 5)], []]$

Reduction on this new set:

There are no top-reducers eligible for r_6; it's already monic.

$R_5 := [r_6]$

CritPair for R_5 and G_1 generates the following pairs:

$p_{11} := (x^2y^3zt, x^2, r_6, y^2zt, r_3)$, $p_{12} := (xy^3z^2t, xz, r_6, y^3t, r_2)$,

$p_{13} := (xy^3zt, x, r_6, z, r_4)$, $p_{14} := (y^3z^6t, z^5, r_6, y^3, r_5)$,

$p_{15} := (y^3z^3t, z^2, r_6, y^2t, r_1)$

Note that p_{14} is eliminated because $\phi_2(z^5\mathcal{S}(r_6)) \neq z^5\mathcal{S}(r_6)$

Note that p_{15} is eliminated because $\phi_2(z^2\mathcal{S}(r_6)) \neq z^2\mathcal{S}(r_6)$

$G_1 := [r_3, r_2, r_4, r_5, r_1, r_6]$

Now we have $d := 6$, $P_6 := [p_7, p_{13}]$ and $P := [p_9, p_{10}, p_{11}, p_{12}]$

SPol(P_6)

p_7 is Rewritten on xr_1 by rule $(x, 6)$

$p_{13} \to r_7 := (x^2F_1, z^5t - x^4t^2)$

Add a rule so that $Rule := [[(x, 6), (x^2, 7)], [(xy, 4), (xyz^2, 5)], []]$

Reduction on this new set:

There are no top-reducers eligible for r_7; it's already monic.

$R_6 := [r_7]$

CritPair for R_6 and G_1 generates the following pairs:

$p_{16} := (x^2yz^5t, x^2y, r_7, z^5t, r_3)$, $p_{17} := (xz^5t, x, r_7, z^3t, r_2)$

$p_{18} := (xy^3z^5t, xy^3, r_7, z^5, r_4)$, $p_{19} := (z^6t, z, r_7, 1, r_5)$

$p_{20} := (yz^5t, y, r_7, z^2t, r_1)$, $p_{21} := (y^3z^5t, y^3, r_7, z^4, r_6)$

Note that p_{16} is eliminated because $\phi_2(x^2y\mathcal{S}(r_7)) \neq x^2y\mathcal{S}(r_7)$

Note that p_{18} is eliminated because $\phi_2(xy^3\mathcal{S}(r_7)) \neq xy^3\mathcal{S}(r_7)$

Note that p_{20} is eliminated because $\phi_2(y\mathcal{S}(r_7)) \neq y\mathcal{S}(r_7)$

Note that p_{21} is eliminated because $\phi_2(y^3\mathcal{S}(r_7)) \neq y^3\mathcal{S}(r_7)$

$G_1 := [r_3, r_2, r_4, r_5, r_1, r_6, r_7]$

Now we have $d := 7$, $P_7 := [p_{11}, p_{12}, p_{17}, p_{19}]$ and $P := [p_9, p_{10}]$

SPol(P_7)

p_{11} is Rewritten on x^2r_6 by rule $(x^2, 7)$

p_{12} is Rewritten on xzr_6 by rule $(x^2, 7)$

$p_{17} \to r_8 := (x^3F_1, -x^5t^2 + y^2z^3t^2)$

Add a rule so that $Rule := [[(x, 6), (x^2, 7), (x^3, 8)], [(xy, 4), (xyz^2, 5)], []]$

$p_{19} \to r_9 := (x^2zF_1, y^5t^2 - x^4zt^2)$

Add a rule so that $Rule := [[(x,6),(x^2,7),(x^3,8),(x^2z,9)],$

$[(xy,4),(xyz^2,5)],[]]$

Reduction on this new set:

We reorder the new set by signature; thus we start with r_9.

There are no top-reducers eligible for r_9; it's already monic.

There are no top-reducers eligible for r_8; just make it monic.

$r_8 := (x^3F_1, x^5t^2 - y^2z^3t^2)$

$R_7 := [r_9, r_8]$

CritPair for R_7 and G_1 generates the following pairs:

Starting with r_9:

$p_{22} := (x^2y^5t^2, x^2, r_9, y^4t^2, r_3)$, $p_{23} := (xy^5z^2t^2, xz^2, r_9, y^5t^2, r_2)$

$p_{24} := (xy^5t^2, x, r_9, y^2t, r_4)$, $p_{25} := (y^5z^6t^2, z^6, r_9, y^5t, r_5)$

$p_{26} := (y^5z^3t^2, z^3, r_9, y^4t^2, r_1)$, $p_{27} := (y^5zt^2, z, r_9, y^2t, r_6)$

$p_{28} := (y^5z^5t^2, z^5, r_9, y^5t, r_7)$

Note that p_{23} is eliminated because $\phi_2(xz^2\mathcal{S}(r_9)) \neq xz^2\mathcal{S}(r_9)$

Note that p_{25} is eliminated because $\phi_2(z^6\mathcal{S}(r_9)) \neq z^6\mathcal{S}(r_9)$

Note that p_{26} is eliminated because $\phi_2(z^3\mathcal{S}(r_9)) \neq z^3\mathcal{S}(r_9)$

Note that p_{27} is eliminated because $\phi_2(z\mathcal{S}(r_9)) \neq z\mathcal{S}(r_9)$

Note that p_{28} is eliminated because $\phi_2(z^5\mathcal{S}(r_9)) \neq z^5\mathcal{S}(r_9)$

$G_1 := [r_3, r_2, r_4, r_5, r_1, r_6, r_7, r_9]$

Now for r_8:

$p_{29} := (x^5yt^2, y, r_8, x^3t^2, r_3)$, $p_{30} := (x^5z^2t^2, z^2, r_8, x^4t^2, r_2)$

$p_{31} := (x^5y^3t^2, y^3, r_8, x^4t, r_4)$, $p_{32} := (x^5z^6t, z^6, r_8, x^5, r_5)$

$p_{33} := (yz^5t^2, y, r_8, z^2t, r_1)$, $p_{34} := (x^5y^3zt^2, y^3z, r_8, x^5t, r_6)$

$p_{35} := (x^5z^5t^2, z^5, r_8, x^5t, r_7)$, $p_{36} := (x^5y^5t^2, y^5, r_8, x^5, r_9)$

Note that p_{29} is eliminated because $\phi_2(y\mathcal{S}(r_8)) \neq y\mathcal{S}(r_8)$

Note that p_{30} is eliminated because $\phi_2(z^2\mathcal{S}(r_8)) \neq z^2\mathcal{S}(r_8)$

Note that p_{31} is eliminated because $\phi_2(y^3\mathcal{S}(r_8)) \neq y^3\mathcal{S}(r_8)$

Note that p_{32} is eliminated because $\phi_2(z^6\mathcal{S}(r_8)) \neq z^6\mathcal{S}(r_8)$

Note that p_{33} is eliminated because $\phi_2(y\mathcal{S}(r_8)) \neq y\mathcal{S}(r_8)$

Note that p_{34} is eliminated because $\phi_2(y^3z\mathcal{S}(r_8)) \neq y^3z\mathcal{S}(r_8)$

Note that p_{35} is eliminated because $\phi_2(z^5\mathcal{S}(r_8)) \neq z^5\mathcal{S}(r_8)$

Note that p_{36} is eliminated because $\phi_2(y^5\mathcal{S}(r_8)) \neq y^5\mathcal{S}(r_8)$

$G_1 := [r_3, r_2, r_4, r_5, r_1, r_6, r_7, r_9, r_8]$

Now we have $d := 8$, $P_8 := [p_9, p_{10}, p_{24}]$ and $P := [p_{22}]$

SPol(P_8)

p_9 is Rewritten on xy^2tr_1 by rule $(x, 6)$

$p_{10} \to r_{10} := (z^3tF_1, y^6t^2 - x^2z^3t^3)$

Add a rule so that $Rule := [[(x, 6), (x^2, 7), (x^3, 8), (x^2z, 9), (z^3t, 10)],$

$[(xy, 4), (xyz^2, 5)], []]$

$p_{24} \to r_{11} := (x^3zF_1, -x^5zt^2 + y^2z^4t^2)$

Add a rule so that $Rule := [[(x, 6), (x^2, 7), (x^3, 8), (x^2z, 9), (z^3t, 10),$

$(x^3z, 11)], [(xy, 4), (xyz^2, 5)], []]$

Reduction on this new set:

$r_{10} := (z^3tF_1, \phi_2(poly(r_{10})) = y^6t^2 - xy^2zt^4)$

From here there are no top-reducers eligible for r_{10}; it's already monic.

There are no top-reducers eligible for r_{11} (r_8 is disallowed by *IsReducible?*).

Just make it monic. $r_{11} := (x^3zF_1, x^5zt^2 - y^2z^4t^2)$

$R_8 := [r_{10}, r_{11}]$

CritPair for R_8 and G_1 generates the following pairs:

Starting with r_{10}:

$p_{37} := (x^2y^6t^2, x^2, r_{10}, y^5t^2, r_3)$, $p_{38} := (xy^6z^2t^2, xz^2, r_{10}, y^6t^2, r_2)$

$p_{39} := (xy^6t^2, x, r_{10}, y^3t, r_4)$,$p_{40} := (y^6z^6t^2, z^6, r_{10}, y^6t, r_5)$

$p_{41} := (y^6z^3t^2, z^3, r_{10}, y^5t^2, r_1)$, $p_{42} := (y^6zt^2, y^3t, r_6, z, r_{10})$

$p_{43} := (y^6z^5t^2, y^6t, r_7, z^5, r_{10})$,$p_{44} := (y^6t^2, y, r_9, 1, r_{10})$

$p_{45} := (x^5y^6t^2, y^6, r_8, x^5, r_{10})$

Note that p_{37} is eliminated because $\phi_2(x^2\mathcal{S}(r_{10})) \neq x^2\mathcal{S}(r_{10})$

Note that p_{38} is eliminated because $\phi_2(xz^2\mathcal{S}(r_{10})) \neq xz^2\mathcal{S}(r_{10})$

Note that p_{39} is eliminated because $\phi_2(x\mathcal{S}(r_{10})) \neq x\mathcal{S}(r_{10})$

Note that p_{40} is eliminated because $\phi_2(z^6\mathcal{S}(r_{10})) \neq z^6\mathcal{S}(r_{10})$

Note that p_{41} is eliminated because $\phi_2(z^3\mathcal{S}(r_{10})) \neq z^3\mathcal{S}(r_{10})$

Note that p_{42} is eliminated because $\phi_2(y^3t\mathcal{S}(r_6)) \neq y^3t\mathcal{S}(r_6)$

Note that p_{43} is eliminated because $\phi_2(z^5\mathcal{S}(r_7)) \neq z^5\mathcal{S}(r_7)$

Note that p_{44} is eliminated because $\phi_2(y\mathcal{S}(r_9)) \neq y\mathcal{S}(r_9)$

Note that p_{45} is eliminated because $\phi_2(y^6\mathcal{S}(r_8)) \neq y^6\mathcal{S}(r_8)$

$G_1 := [r_3, r_2, r_4, r_5, r_1, r_6, r_7, r_9, r_8, r_{10}]$

Now for r_{11}:

$p_{46} := (x^5yzt^2, y^2, r_{11}, x^3zt^2, r_3)$, $p_{47} := (x^5z^2t^2, z, r_{11}, x^4t^2, r_2)$

$p_{48} := (x^5y^3zt^2, y^3, r_{11}, x^4zt, r_4)$, $p_{49} := (x^5z^6t^2, z^5, r_{11}, x^5t, r_5)$

$p_{50} := (x^5yz^3t^2, yz^2, r_{11}, x^5t^2, r_1)$, $p_{51} := (x^5y^3zt^2, y^3, r_{11}, x^5t, r_6)$

$p_{52} := (x^5z^5t, x^5, r_7, z^4, r_{11})$, $p_{53} := (x^5y^5zt^2, y^5, r_{11}, x^5z, r_9)$

$p_{54} := (x^5zt^2, z, r_8, 1, r_{11})$, $p_{55} := (x^5y^6zt^2, y^6, r_{11}, x^5z, r_{10})$

Note that p_{46} is eliminated because $\phi_2(y^2\mathcal{S}(r_{11})) \neq x^2\mathcal{S}(r_{10})$

Note that p_{47} is eliminated because $\phi_2(z\mathcal{S}(r_{11})) \neq x^2\mathcal{S}(r_{10})$

Note that p_{48} is eliminated because $\phi_2(y^3\mathcal{S}(r_{11})) \neq x^2\mathcal{S}(r_{10})$

Note that p_{49} is eliminated because $\phi_2(z^5\mathcal{S}(r_{11})) \neq x^2\mathcal{S}(r_{10})$

Note that p_{50} is eliminated because $\phi_2(yz^2\mathcal{S}(r_{11})) \neq x^2\mathcal{S}(r_{10})$

Note that p_{51} is eliminated because $\phi_2(y^3\mathcal{S}(r_{11})) \neq x^2\mathcal{S}(r_{10})$

Note that p_{52} is eliminated because $\phi_2(z^4\mathcal{S}(r_{11})) \neq x^2\mathcal{S}(r_{10})$

Note that p_{53} is eliminated because $\phi_2(y^5\mathcal{S}(r_{11})) \neq x^2\mathcal{S}(r_{10})$

Note that p_{55} is eliminated because $\phi_2(y^6\mathcal{S}(r_{11})) \neq x^2\mathcal{S}(r_{10})$

$G_1 := [r_3, r_2, r_4, r_5, r_1, r_6, r_7, r_9, r_8, r_{10}, r_{11}]$

Now we have $d := 8$, $P_8 := [p_{54}]$ and $P := [p_{22}]$

SPol(P_8)

p_{54} is Rewritten on zr_8 by rule $(x^3z, 11)$

Now we have $d := 9$, $P_9 := [p_{22}]$ and $P := \emptyset$

SPol(P_9)

p_{22} is Rewritten on $x^2 r_9$ by rule $(x^3 z, 11)$

return G_1 ($P = \emptyset$, so we are done)

It is worth noting that our trace has yielded a significant difference from Faugere's trace in [7]. The signed polynomial r_{11} is not added to the Grobner basis in his trace. This is because the F5 algorithm does not force an order, within the degrees, of *Deg_Sort*. In Faugere's trace he decides to switch the order of the critical pairs in P_7 so that *SPol* adds r_9 (and, hence, the rule $(x^2 z, 9)$) before r_8 (and its rule $(x^3, 8)$). (Obviously, switching the order would change the numbering of these two signed polynomials. To keep things from being confusing, we will keep r_8 and r_9 as given in the trace.) There is nothing overtly incorrect in doing this, as there is no standard required method for ordering other than to order by degree.

A standardized ordering system was used in our trace – namely, all lists were read in ascending order of subscripts. This standardization is also correct. However, under this uniform standard, the algorithm fails to notice that r_{11} is not needed because the rules $(x^2 z, 9)$ and $(x^3, 8)$ fall in the "wrong" order.

The reader should be assured of two facts, both of which are critically important to this paper:

(1) Both methods are correct.

(2) Both methods can be used as a counterexample to a currently (as of Spring 2008) accepted Grobner-basis-algorithm theorem.

3.4.3. *Rewritten.* Though the procedure *Rewritten* was introduced in its present form in 2002 in [7], there is no proof for its validity therein. Moreover, Stegers laments his difficulty in proving *Rewritten's* validity in [15]. In this section we prove that *Rewritten* is indeed a valid procedure for eliminating critical pairs in F5. Moreover, the proof itself serves as another example of the overall theme of F5: rewrite current critical pairs in terms of ones for which are already accounted.

There is another point of importance in the following proof of *Rewritten*. Though it is reasonable to assume that *Rewritten* is a valid procedure (after all, it has been used since 2002 as part of the F5 algorithm and its variants), it was previously unclear if special hypotheses were required to make *Rewritten* legitimate criterion for eliminating critical pairs. It was unclear, for example, whether *Rewritten* was only valid if the input sequence was regular – a very strong requirement. On face this makes *Rewritten* a suspect of interest when inquiring into F5's lack of termination for some non-regular inputs (a suspicion that is put to rest here).

It is for these reasons that this author highlights this proof as one of the key contributions offered by this thesis.

Theorem 3.4.1. *Assume that F5 has computed the Grobner basis for* $\langle f_{i+1}, \ldots, f_m \rangle$ *and is computing the Grobner basis for* $\langle f_i, \ldots, f_m \rangle$*. Let* (t, u_1, r_1, u_2, r_2) *be an output of CritPair that activates Rewritten in SPol. Then* $S(r_1, r_2)$ *need not be added to the Grobner basis of* $\langle f_i, \ldots, f_m \rangle$*.*

Proof. There are 3 distinct cases.

(1) *Rewritten* is activated by only (u_1, r_1). Then there exists a rule $(\mathcal{L}_1, k_1)$ such that $\mathcal{L}_1 | (u_1 \mathcal{S}(r_1))$. Let $c = \frac{(u_1 \mathcal{S}(r_1))}{\mathcal{L}_1}$. Let $\mathcal{S}(r_1) = dF_i$. Then we have the following:

$$\begin{aligned} S(r_1, r_2) &= u_1 r_1 - u_2 r_2 \\ &= u_1 \sum_{j=i}^{m} q_j f_j - u_2 r_2 \\ &= u_1((d + \cdots) f_i + \sum_{j=i+1}^{m} q_j f_j) - u_2 r_2 \\ &= u_1 d f_i + u_1((\cdots) f_i + \sum_{j=i+1}^{m} q_j f_j) - u_2 r_2 \\ &= c r_{k_1} + \sum_{j=i}^{m} q'_j f_j - u_2 r_2 \end{aligned} \tag{3.4}$$

where $q_j, q'_j \in K[\overline{x}] \forall j$ and $HT(q'_i) F_i \prec \mathcal{S}(c r_{k_1})$.

Note that this new representation 3.4 of $S(r_1, r_2)$ has the same signature as the original, namely $\mathcal{S}(u_1 r_1)$. Using Lemma 3.2.1, we can assume without loss of generality that all the components of 3.4 are normalized. We will also assume, without loss of generality, that no $q'_j f_j$ can be rewritten as per *Rewritten*. (If it could simply rewrite as we rewrote $u_1 r_1$ above.) We will use this new representation of $S(r_1, r_2)$ to show that $S(r_1, r_2) = o_G(u_1 r_1)$, where G is the set of signed polynomials from the previous iteration's Grobner basis (index $i + 1$) and other signed polynomials that will be added to the Grobner basis of the input sequence during iteration i. Thus it will immediately follow, as per Theorem 3.3.1, that $S(r_1, r_2)$ is not needed.

There must be a syzygy of head terms on the right-hand-side of equation 3.4 because $HT(S(r_1, r_2)) \neq HT(u_2 r_2)$. We will show $S(r_1, r_2) = o_G(u_1 r_1)$ by induction on the number of syzygies of pairs (exactly two, no more) of head terms on the right-hand-side; in other words, in the spirit of Theorem 2.5.2, we are going to induct on the number of S-polynomial rewrites that are required on the right-hand-side of equation 3.4 in order that no addend on the right-hand-side have a head term greater than the head term of $S(r_1, r_2)$.

Base Case(the number of syzygies of head terms is exactly one): $u_2 r_2$ has to be involved, so there are two cases:

(a) The syzygy of head terms is between r_{k_1} and r_2.

(b) The syzygy of head terms is between f_j and r_2 for some j.

In the first case, we note the following facts regarding $S(r_{k_1}, r_2)$: $S(r_{k_1}, r_2)$ is normalized; $S(r_{k_1}, r_2)$ cannot trigger *IsRewritable?* (if r_{k_1} triggered it, (u_1, r_1) would have activated *IsRewritable?* differently); r_2 cannot activate *IsRewritable?* because that was the assumption of case 1. So either $S(r_{k_1}, r_2)$ has already been added to G, will be added to G later in the

iteration (during this degree) or reduced to 0 in *Reduction*; in either case, $S(r_{k_1}, r_2) = o_G(u_1 r_1)$, thus implying $S(r_1, r_2) = o_G(u_1 r_1)$.

(The assumption that $u_2 r_2$ is not rewritable is technical in nature and is meant to assure the reader that all the conditions set forth in the pseudocode are met. In truth, if some new polynomial were to come along, later in the same degree – call it d, that would rewrite $u_2 r_2$, we could simply make a new representation of $S(r1, r2)$ using that new polynomial and push through the same argument as above. Since no degree d has infinitely many polynomials added, this process of rewriting is not infinite (and will stop). And since this rewriting doesn't affect signatures at all, our dependence on the "little-o" condition is unaffected as well.)

(To simplify notation, we simply assume $u_2 r_2$ has already been represented in such a way that rewrites are impossible. If the reader finds this unacceptable, case 3 below demonstrates a more robust case of rewriting.)

The second case uses the exact same arguments as the first to conclude $S(r_1, r_2) = o_G(u_1 r_1)$.

This finishes the base case.

Assume $S(r_1, r_2) = o_G(u_1 r_1)$ if α syzygies of head terms are needed to remove all syzygies of head terms from the right-hand-side.
(The induction hypothesis.)

Prove if $\alpha + 1$ syzygies of head terms are needed. So we are looking at the following (derived from equation 3.4):

$$\sum_{\beta \in A} \lambda_\beta S(x_\beta, y_\beta) - u_2 r_2$$

where the first α syzygies of head terms have produced an index set A of S-polynomials (and/or f_j's), all of which have been added (or will be

added) to the Grobner basis. If $HT(u_2 r_2)$ would have been a part of the first α syzygies of head terms, then the head term of the right-hand-side would already be less than $HT(u_1 r_1)$ (i.e., $HT(u_2 r_2)$) and we would have $S(r_1, r_2) = o_G(u_1 r_1)$ immediately. Thus we assume that, after the first α syzygies of head terms, $u_2 r_2$ is still intact in the representation. Since, by assumption, there is only one syzygy of head terms remaining on the right-hand-side, it must be between $u_2 r_2$ and $\lambda_{\beta'} S(x_{\beta'}, y_{\beta'})$ for some $\beta' \in A$.

Since we assumed without loss of generality in equation 3.4 that all components of the representation were normalized and would not trigger *Is-Rewritable?*, we can assume $S(S(x_{\beta'}, y_{\beta'}), u_2 r_2)$ is normalized and is not *Rewritten*. Thus $S(S(x_{\beta'}, y_{\beta'}), u_2 r_2)$ has already been added to G, will be added to G later in the iteration (during this degree) or reduced to 0 in *Reduction*; in either case, $S(S(x_{\beta'}, y_{\beta'}), u_2 r_2) = o_G(u_1 r_1)$, thus implying $S(r_1, r_2) = o_G(u_1 r_1)$.

This completes the induction proof.

This completes case 1.

(2) *Rewritten* is activated by only (u_2, r_2). This case mirrors case 1.

(3) *Rewritten* is activated by both (u_1, r_1) and (u_2, r_2). Define $\mathcal{L}$, k_1, c and d as in case 1. In addition there exists a rule $(\mathcal{L}_2, k_2)$ such that $\mathcal{L}_2 | (u_2 \mathcal{S}(r_2))$. Let $a = \frac{(u_2 \mathcal{S}(r_2))}{\mathcal{L}_2}$ and $\mathcal{S}(r_2) = bF_i$. Then in a similar construction as equation 3.4 introduced in case 1,

$$S(r_1, r_2) = cr_{k_1} + \sum_{j=i}^{m} q'_j f_j - ar_{k_2} + \sum_{j=i}^{m} q''_j f_j$$

where

$$cr_{k_1} + \sum_{j=i}^{m} q'_j f_j$$

is a new representation of $u_1 r_1$ and

$$ar_{k_2} + \sum_{j=i}^{m} q_j'' f_j$$

is a new representation of $u_2 r_2$, with $q_j', q_j'' \in K[\overline{x}] \forall j$ and $HT(q_i')F_i \prec \mathcal{S}(cr_{k_1})$. From here the case mirrors 1.

Thus in all three cases $S(r_1, r_2) = o_G(u_1 r_1)$, implying by Theorem 3.3.1 $S(r_1, r_2)$ is not needed.

□

Now we have shown the validity of *Rewritten* as a criterion for removing critical pairs in F5. Again we emphasize that no assumption of regularity in the input sequences was required. Thus, when used in regards to the "little-o" condition of Theorem 3.3.1 for regulating a Grobner basis, *Rewritten* is quite robust – it is a criterion that works harmoniously and in tandem with the "little-o" condition. We will utilize this property later when we introduce a new algorithm for computing Grobner bases in section 4.

3.4.4. *Proof of F5's Correctness.* In this section we prove that F5 is correct (if it terminates). Our primary bases for this proof will be Theorems 3.3.1 and 3.4.1. The reader is encouraged to note that no assumption of regularity is required for correctness; this is an important comment because, as mentioned in section 3.2, the hypothesis of regularity is mathematically quite strong and we would like to distance ourselves from it as much as possible.

Theorem 3.4.2. *If F5 terminates on input $(f_1, \ldots, f_m)$, the set poly(G) is a Grobner basis for $\langle f_1, \ldots, f_m \rangle$.*

Proof. We will prove the correctness of F5 using the same language and notation from Theorem 3.3.1.

We begin by noting that $F \subset poly(G)$ because (F_i, f_i) is added to the output at the very beginning of the i^{th} iteration of *Algorithm_F5*.

It is also clear that all the $r_i \in G$ are monic. Before any signed polynomial is added to the output during an iteration of *Algorithm_F5*, it is made monic by the line:

$$r_{k_0} := (\frac{1}{HC(r_{k_0})})r_{k_0};$$

in *TopReduction*.

It requires a little more analysis to prove the claim that all the $r_i \in G$ are admissible. We will do so here; a reader may also find the proof in [7, 15].

We will prove admissibility by induction on the index i.

Base Case ($i = m$). This is clear because the only polynomial of index m that appears in G is (F_m, f_m), which is trivially admissible.

Assume all polynomials appearing in G of index k or greater are admissible.

Prove that all polynomials appearing in G of index $k-1$ are admissible. If $k = 1$, there are no polynomials of this index or lower; thus we are trivially done. So, without loss of generality, we assume that $k - 1 \geq 1$.

To prove this we will need another proof by induction (so the proof of Theorem 3.4.2 is a proof by nested induction). We will induct on the signatures of the polynomials of index $k-1$ (i.e., we will induct on $\prec$).

> Base Case (signature of the polynomial is F_{k-1}). As mentioned prior, the signed polynomial (F_{k-1}, f_{k-1}) is clearly admissible.
>
> Assume all $r_p \in G$ of index $k-1$, $\mathcal{S}(r_p) \preceq tF_{k-1}$, are admissible.
>
> Prove all $r_p \in G$ of index $k-1$, $\mathcal{S}(r_p) \succ tF_{k-1}$, are admissible. The operation to create new signed polynomials for possible addition to G is done in *SPol* in the line:
>
> $$r_N := (u_{\mathcal{L}}\mathcal{S}(r_{i_{\mathcal{L}}}), u_{\mathcal{L}}poly(r_{i_{\mathcal{L}}}) - v_{\mathcal{L}}poly(r_{j_{\mathcal{L}}}));$$

where

$$u_{\mathcal{L}}\mathcal{S}(r_{i_{\mathcal{L}}}) \succ v_{\mathcal{L}}\mathcal{S}(r_{j_{\mathcal{L}}})$$

If $r_{i_{\mathcal{L}}}$ and $r_{j_{\mathcal{L}}}$ are of index k or greater, then they are admissible by the outer induction hypothesis. If $r_{i_{\mathcal{L}}}$ and $r_{j_{\mathcal{L}}}$ are of index $k-1$, but have signatures $\preceq tF_{k-1}$, they are admissible by the inner induction hypothesis. If they are of index $k-1$, but have signatures $\succ tF_{k-1}$, then we recall that $r_{i_{\mathcal{L}}}$ was generated from an S-polynomial of polynomials of strictly smaller signature (along with possible reduction in *Reduction*). So

$$r_{i_{\mathcal{L}}} = u_1 r_{i_1} - u_2 r_{i_2} - \sum_{r_p \in G} u_{r_p} r_p$$

where $u_1 r_{i_1} - u_2 r_{i_2}$ is representative of the S-polynomial calculation from *SPol* and $\sum_{r_j \in G} u_{r_j} r_j$ represents the reduction of $r_{i_{\mathcal{L}}}$ in *Reduction*. If any of the elements of R in the above representation of $r_{i_{\mathcal{L}}}$ still fall outside of the range of the two induction hypotheses, then this process can be repeated; since the signatures of the polynomials r_j are strictly decreasing, we will eventually find ourselves with a representation of $r_{i_{\mathcal{L}}}$ for which the induction hypotheses apply. A similar argument will hold for $r_{j_{\mathcal{L}}}$. So, in short, without loss of generality, $r_{i_{\mathcal{L}}}$ and $r_{j_{\mathcal{L}}}$ are admissible.

It follows immediately that r_N is also admissible, since it's signature is inherited from $r_{i_{\mathcal{L}}}$. Since r_N was arbitrary, this concludes the inner induction.

This concludes the outer induction, and the proof of admissibility in F5.

The only remaining item to prove is the third condition of Theorem 3.3.1. The procedure *CritPair* ensures that only normalized critical pairs are being considered in F5. Once a critical pair is being considered, there are only a few things that can happen to it:

(1) The critical pair is eliminated by *Rewritten?* in *SPol*. By Theorem 3.4.1, this meets the little-o condition of Theorem 3.3.1.

(2) The polynomial is reduced to 0 in *Reduction*. By definition, this meets the little-o condition of Theorem 3.3.1.

(3) The polynomial is reduced (not to 0) and then added. This is fine too. Note that, once a signed polynomial makes it into *Reduction*, it is guaranteed that a reduced version **of the same signature** will be added to the Grobner basis. (It is also possible the reduction will yield the addition of a signed polynomial with larger signature to the Grobner basis, but that is irrelevant to meeting the little-o condition for the original signed polynomial.) The addition of this signed polynomial immediately meets the little-o condition of Theorem 3.3.1.

This list covers all possible cases; thus, we can conclude the third condition of Theorem 3.3.1 is satisfied. Thus all hypotheses of Theorem 3.3.1 have been met.

This proves the correctness of the F5 algorithm and completes our proof.

□

3.4.5. *A Regular Input Sequence Yields No Reductions to 0.* The real benefit of using F5 is an increase in speed (measured empirically, on average) over previously used Grobner-basis-producing algorithms. As mentioned in [7], it is a full order of magnitude faster than the F4 algorithm (see [8]) and is more than two orders of magnitude faster than other algorithms. The natural question to ask would be: where does F5 makes such positive gains in speed.

The answer is that F5 uses the signature of the polynomials to avoid some full normal-form computations – the computations primarily responsible for giving Grobner-basis-producing algorithms their double-exponential time complexity on the number of variables in the polynomials ring (see [2]). The number of full reductions to 0 – the criterion for not adding a polynomial to the Grobner basis in the

Buchberger algorithm – varies depending on the properties of the input sequence. Once again we will see that assuming regularity of the input sequence yields (not surprisingly) a very strong result.

Theorem 3.4.3. *If the input sequence $(f_1, \ldots, f_m)$ is regular, there are no reductions to 0 in Reduction in the F5 algorithm.*

Proof. Let $r \in R$, $r = (uF_i, p)$ be a signed polynomial that is a candidate for admission into G in F5 (i.e., r has made it into *Reduction*). Then we know that r is admissible, and we know that r is normalized.

Assume, for contradiction, that r is reduced to 0 in F5. Then we have the following witness of this reduction:

$$p - \Big(\sum_{g_i \in poly(G)} q_i g_i\Big) = 0 \tag{3.5}$$

where $q_i \in K[\overline{x}] \forall i$.

By property (b) of *IsReducible*, all the $q_i g_i$ are normalized. By property (d) of *IsReducible*, $\mathcal{S}(3.5)$ is either $\mathcal{S}(\sum_{g_i \in poly(G)} q_i g_i)$ or uF_i. Without loss of generality, let's say that $\mathcal{S}(3.5) = \mathcal{S}(\sum_{g_i \in poly(G)} q_i g_i)$.

Then $\exists g, g' \in \mathcal{P}^m$ such that the following hold:

(1) $v(g) = v(g') = p = \sum_{g_i \in poly(G)} q_i g_i$

(2) $HT(g) = u$

(3) $HT(g') = \mathcal{S}(\sum_{g_i \in poly(G)} q_i g_i)$

(4) $index(g) = index(g') = i$

This implies that $(g' - g) \in Syz$. Since $(f_1, \ldots, f_m)$ is regular, $(g' - g) \in PSyz$. By Theorem 3.2.1, we know $\mathcal{S}(\sum_{g_i \in poly(G)} q_i g_i)$ is not normalized. This is a contradiction to the fact that, by property (b) of *IsReducible*, all the $q_i g_i$ are normalized.

Thus our assumption that r is reduced to 0 in F5 was incorrect.

Since r was arbitrary in an arbitrary index i, we know that there are no reductions to 0 in F5 if the input sequence is regular.

This completes the proof.

□

So if we assume the input sequence is regular, there will be no reductions to 0 in F5. Perhaps not surprisingly, if we start with an input sequence that is non-regular, F5 may have some reductions to 0. (Any such reduction to 0 is output with a message in *TopReduction*.) Such examples are plentiful.

3.5. **(Error in) Proof of Termination in F5.** In this subsection we will discuss an error in reasoning that currently exists in the proof of termination of the F5 algorithm. This proof was originally presented in [7] and was labeled as a conjecture in [15]. Understanding the cases when F5 terminates is of key importance. In section 4.4, we will introduce a new hybridization of F5 and Buchberger's algorithm. This algorithm will always terminate, but it will come at great cost to the timing of F5 under certain input. Any attempt to minimize the time F5 takes should begin with a strong understanding of the situations in which it terminates.

To that end, we begin our discussion with the following proposition from [7]:

Proposition 3.5.1. *For all degree d, the result of Reduction will be denoted as R_d. Suppose that there are no reductions to* 0 *during the run of F5. Then, at index i, $\langle HT(G_i) \rangle = \langle HT(G_i \cup R_d) \rangle$.*

This proposition, if true, would lead immediately (via Theorem 3.4.3) to the following corollary:

Corollary 3.5.1. *If an input sequence to F5 is regular, F5 terminates for that sequence.*

The problem is that Proposition 3.5.1 is plainly false. To witness this fact, we can look at the example run of F5 from sub-subsection 3.4.2. Note that this example has no reductions to 0 within it, yet R_8 fails to meet the conclusion of Proposition 3.5.1. Thus we have a counterexample to the proposition. For more exposition on this situation, see Appendix A.

This causes an immediate dilemma because it destroys Corollary 3.5.1. This corollary is the standard by which F5 is considered to terminate. Now we must strive to find a replacement standard. We will attempt to do this in this section.

We will now change course. Consider this new proposition that is introduced by [15] (during the conjecture of termination) and strongly implied by [7] (during the discussion of Proposition 3.5.1).

Conjecture 3.5.1. *Let F5 run on input $(f_1, f_2, \ldots, f_m)$ and let $r_j = (u_j F_i, p_j)$ be a signed polynomial that has been added to G_i during index i's iteration. Then there will not be a signed polynomial $r_k = (u_k F_i, p_k)$, $r_k \neq r_j$ added to G_i where $u_k = uu_j$ and $HT(p_k) = uHT(p_j)$.*

We will deal with the validity of Conjecture 3.5.1 at a later time. No proof will be offered now. For the moment, let's accept it as true.

Then we have the following theorem that will shed some light on the terminating conditions of F5.

Consequence 3.5.1. *Assume that K is a finite field with $|K| = \rho = p^k$ for some prime p. Let $(f_1, f_2, \ldots, f_m)$ be a proposed input sequence into F5, $f_i \in K[x_1, x_2, \ldots, x_n]$ $\forall i$. Assume, instead, that the sequence*

$$(x_1^\rho - x_1 t^{\rho-1}, x_2^\rho - x_2 t^{\rho-1}, \ldots, x_n^\rho - x_n t^{\rho-1}, f_m, f_{m-1}, \ldots, f_1)$$

is input into F5, ***from left to right in order****, where t is a homogenizing variable. Then F5 will terminate under this input.*

Proof. Assume, for contradiction, that F5 does not terminate under this input at index i. Also assume, without loss of generality, that $i = 1$. There exists a degree d such that the degree-d Grobner basis (GB_d) for $\langle x_1^\rho - x_1 t^{\rho-1}, x_2^\rho - x_2 t^{\rho-1}, \ldots, x_n^\rho - x_n t^{\rho-1}, f_m, f_{m-1}, \ldots, f_1\rangle$ is in fact a Grobner basis.

Since we assumed that F5 does not terminate under this input, there exists $q \in GB_d$ and an infinite sequence of terms $\{u_k\}_{k=1}^\infty$ and an infinite sequence of polynomials $\{p_k\}_{k=0}^\infty$ such that $q = p_0$ and $HT(p_j) = u_j HT(p_{j-1})$, with p_k being added to the Grobner basis for all k.

It must be the case that $\nexists(N, \alpha)$ pair such that

$$x_\alpha^\rho | \prod_{\ell=1}^{N} u_\ell$$

In this case, p_N would not have been added to the Grobner basis because it would be top-reducible by $x_\alpha^\rho - x_\alpha t^{\rho-1}$. So there exists and N such that, for all $n' > N$, $u_{n'}$ is only divisible by term t (and not $x_1, x_2, \ldots, x_n$).

Now we must consider the following fact ($\forall k$):

$$p_k = h_{1,k}f_1 + h_{2,k}f_2 + \cdots + h_{m,k}f_m + h_{m+1,k}(x_1^\rho - x_1 t^{\rho-1}) + \cdots + h_{m+n,k}(x_n^\rho - x_n t^{\rho-1})$$

where $h_{\ell,k} \in K[\overline{x}] \forall \ell$ and $\mathcal{S}(p_k) = HT(h_{1,k})F_{m+n}$. Without loss of generality, we will consider $h_{1,k}$ to be a totally normalized polynomial; that is to say that all terms of $h_{1,k}$ are normalized. By Lemma 3.2.1, this assumption does not change the head term of p_k. Moreover, since we assumed p_k was added to the Grobner basis, we can assume that normalization does not change the signature of p_k.

We will call a polynomial "docked in term y" if each term from the polynomial is re-written without y. For example, the polynomial $3x_1^2x_2 + 4x_1x_2^2 \in \mathbf{F_5}[x_1, x_2]$ would be docked in x_2 into $3x_1^2 + 4x_1$.

If we consider all possible $h_{1,k}$'s (i.e., ranging over k) docked in t, since K is a finite field and we assumed total normalization, there are only finitely many distinct $h_{1,k}$'s docked in t. This means we can find two polynomials p_β and p_γ, $\beta, \gamma > N$, $\beta > \gamma$, such that $h_{1,\beta}$ docked in t equals $h_{1,\gamma}$ docked in t. Since $\beta, \gamma > N$ and $\beta > \gamma$,

$$HT(p_\beta) = t^\Delta HT(p_\gamma)$$

for some exponent Δ. Moreover, $h_{1,\beta} = t^\Delta h_{1,\gamma}$. This means that we have two polynomials p_β and p_γ such that

$$HT(p_\beta) = t^\Delta HT(p_\gamma)$$

and

$$\mathcal{S}(p_\beta) = t^\Delta \mathcal{S}(p_\gamma)$$

By Conjecture 3.5.1, this is impossible.

Thus our assumption that F5 doesn't terminate at index i must be incorrect. Therefore F5 does terminate at index i. Since i was arbitrary, this means F5 terminates.

This completes the proof.

□

We now make a few remarks regarding Consequence 3.5.1.

Remark 3.5.1. If Conjecture 3.5.1 is in fact true, we have a new standard for F5 terminating. Perhaps even more impressively, this standard does not appeal to a hypothesis of regularity. The removal of such a hypothesis is very attractive mathematically.

Remark 3.5.2. However, this author has been unable to prove (or disprove) Conjecture 3.5.1. Folklore certainly claims it's validity. But without proof, we cannot show that Consequence 3.5.1 is valid.

Consider the problem we witnessed during our example from section 3.4.2. Recall that we had two polynomials r_8 and r_9 whose orders were switched in Faugere's example from [7]. In our example here, we admitted r_{11} into the Grobner basis. In the example from [7], r_{11} is Rewritten by r_8 and not added to the Grobner basis. The point here is that just because something **can** be rewritten doesn't mean that it will **trigger** the subroutine *Rewritten*. The author has run into a similar problem in attempting to prove Conjecture 3.5.1.

Remark 3.5.3. The added condition that the polynomials $x_1^\rho - x_1 t^{\rho-1}, x_2^\rho - x_2 t^{\rho-1}, \ldots, x_n^\rho - x_n t^{\rho-1}$ is a extremely weak hypothesis, and should not be considered a hindrance. These are the so-called field polynomials (derived from the field equations of K by letting $t = 1$). If, as instructed in the statement of Consequence 3.5.1, F5 is run on these polynomials first, the first n iterations of *Algorithm_F5* will pass extremely quickly; the only polynomial added to the Grobner basis at each of these first n iterations will be the corresponding field polynomial. So there is no significant time added to the run of F5.

It is also not unfair to suspect that K is a finite field for many applications. As mentioned in the Introduction, the system of equations produced by AES is over a finite field.

Remark 3.5.4. While Consequence 3.5.1 does offer proof of termination (assuming the folklore concerning Conjecture 3.5.1 is true), it does not offer a tight bound on when this termination must occur. The degree at which the contradiction occurs may be well above the degree needed for a Grobner basis.

In an attempt to make Consequence 3.5.1 provably correct, we will make a slight modification to the F5 algorithm. In the subroutine *IsReducible?* we will remove the fourth if-condition:

$$ut_j F_{k_j} \neq \mathcal{S}(r_{k_0})$$

This means that we will allow reductors that have the same signature as the signed polynomial they are reducing. If we were to make this change, it would clearly act as Conjecture 3.5.1 does in the proof of Consequence 3.5.1; p_γ would have been further reduced by p_β. Thus we do not need Conjecture 3.5.1 anymore.

But we must also make sure that making this change would not produce an error in the F5 algorithm. Recall that this if-condition exists because, if we reduce a signed polynomial using a reductor with the same signature, we will lose track of the resulting signed polynomial's signature. The following theorem speaks to this issue.

Theorem 3.5.1. *The fourth if-condition of IsReducible?:*

$$ut_j F_{k_j} \neq \mathcal{S}(r_{k_0})$$

(hereafter referred to as "the condition to be removed") can be removed without hurting the correctness of F5.

Proof. By Theorem 3.4.4, we know that F5 is a correct algorithm. This correctness is established by meeting the hypotheses of Theorem 3.3. In this proof we will show that if the condition to be removed is removed, the hypotheses of Theorem 3.3 are still met, and thus our modified F5 is still a valid algorithm.

Let $r = (eF_i, p)$ be a signed polynomial that is to be reduced (if possible) by *TopReduction*. Assume that $r' = (e'F_i, p')$ has already been added to the Grobner basis and is such that $HT(p) = yHT(p')$ and $\mathcal{S}(yr') = \mathcal{S}(r)$ for some $y \in T$.

(Assume, without loss of generality, that this candidate reductor is not Rewritten; if it is, simply replace with it's rewrite.)

If the condition to be removed has been removed, and no other reductor is found prior to r' being checked, *IsReducible* will return r' as a legal reductor of r. Then $\mathcal{S}(r - yr') = sF_j \prec eF_i$ for some term $s < e$ and for some integer $j \geq i$.

Thus we have

$$r - yr' = \sum_{k=j}^{m} h_k f_k$$

for $h_k \in K[\overline{x}] \forall k$. Adding yr' to both sides, we get

$$r = \sum_{k=j}^{m} h_k f_k + yr'$$

Note that $\mathcal{S}(righthandside) = \mathcal{S}(lefthandside)$.

In addition, we make the following observation regarding $\sum_{k=j}^{m} h_k f_k$. Without loss of generality, by Lemma 3.2.1, assume that $\sum_{k=j}^{m} h_k f_k$ is normalized. It must be the case, given that *TopReduction* is working with polynomials of signature greater than that of $\sum_{k=j}^{m} h_k f_k$, that

$$\sum_{k=j}^{m} h_k f_k \xrightarrow[G_{i+1} \cup R]{*} 0$$

without an increase in signature during the reduction. (Otherwise there would be a normalized S-polynomial of signature less than eF_i that was left out of F5's computation, which is clearly absurd.)

Thus we are left to conclude

$$r \xrightarrow[G_{i+1} \cup R]{*} 0$$

without an increase in signature. This meets the requirement of the "little-o" condition of Theorem 3.3. (The head term condition is trivially met because all

polynomials in *Reduction* have head terms smaller than the head terms canceled in the S-polynomial creation.)

This completes the proof.

□

In short, the fact that we have lost the signature is irrelevant because, no matter what sF_j is, the resulting polynomial reduces to 0. Thus we have the following corollary:

Corollary 3.5.2. *Assume the hypotheses of Consequence 3.5.1 with the added hypothesis that the if-condition*

$$ut_j F_{k_j} \neq \mathcal{S}(r_{k_0})$$

has been removed from IsReducible?. Then this altered F5 will terminate on this input.

Proof. Immediate. □

4. Improving F5 and Introducing F5t

In this section of the paper, we focus our attention on improving (both theoretically and in implementation) the F5 algorithm. We will begin with a small clean-up of the pseudocode in *CritPair*. Then we will strive to implement Buchberger's first and second criteria within F5. Finally we will introduce a new variant of F5 called F5t – this version of F5 is guaranteed to terminate regardless of input but still heavily relies upon F5-criteria for accepting/rejecting critical pairs. Thus F5t saves many of the speed improvements of F5.

4.1. **Removal of Check of Index in CritPair.**

In this subsection we will show that the line

$$\text{if } (index(r_1) > k) \text{ then}$$
$$\text{return } \emptyset;$$

from *CritPair* can be eliminated from the pseudocode of F5. This fact is not new – it is made in [15]. However it is not proven in [15] and in the reprint of [7], the change was not made. Thus we aim to provide proof of this deletion's correctness here.

Proposition 4.1.1. *The line*

$$\textit{if } (index(r_1) > k) \textit{ then}$$
$$\textit{return } \emptyset;$$

(henceforth referred to as "the line to be removed") can be removed.

Proof. We will prove the proposition by induction on the index i.

Base Case ($i = m$). The only polynomial added with this index is the signed polynomial (F_m, f_m). This polynomial isn't even checked by the line to be removed because it occurs outside the for-loop in *Incremental_F5*.

Assume the line to be removed can be removed for all polynomials having index greater than or equal to k.

Prove for index $k-1$. First we note that all signatures of signed polynomials formed during the $k-1^{st}$ iteration of *Algorithm_F5* inherit the index from one of the parents. Polynomial creation only happens in *SPol* in the line:

$$r_N := (u_{\mathcal{L}}\mathcal{S}(r_{i_{\mathcal{L}}}), u_{\mathcal{L}}poly(r_{i_{\mathcal{L}}}) - v_{\mathcal{L}}poly(r_{j_{\mathcal{L}}}));$$

and in *TopReduction* in the line:

$$r_N := (u\mathcal{S}(r_{k_1}), upoly(r_{k_1}) - poly(r_{k_0}));$$

Thus it is impossible for a signed polynomial during the $k-1^{st}$ iteration of *Algorithm_F5* to have an index less than $k-1$ because, if this were possible, an initial signed polynomial of index less than $k-1$ would have to be added. This occurs for the first time in subsequent iterations of *Algorithm_F5*.

Moreover, we note that the line:

$$(h_1, ToDo_1) := TopReduction(\phi_{i+1}(h), G_i \cup Done, k, \phi_{i+1});$$

has a normal form operation imbedded within it. This means that if any signed polynomial during the $k-1^{st}$ iteration of *Algorithm_F5* could be written with larger index, then it would be eliminated by the normal form operation (i.e., the normal form operation would reduce it to 0 and *TopReduction* would subsequently drop it).

(In the above paragraph, we paint with a broad brush for ease. But one can be finer with the argument. In particular, since our signatures, regardless of minimality, are inherited from the parents and all S-polynomials considered during the $k-1^{st}$ iteration are children of (F_{k-1}, f_{k-1}), there won't be any signed polynomials admitted of larger index.)

In short, we can conclude that all signed polynomials that will be added to the Grobner basis during the $k-1^{st}$ iteration have index $k-1$. This is immediately implies that the line to be removed is not needed and can, in fact, be removed.

This completes the proof.

□

In some sense the key result of the above proposition is that, within each iteration of *Algorithm_F5*, there will only be one index admitted – the index of that iteration. Of course it is possible, in cases where the input sequence is non-regular, that a signed polynomial could have a signature of the wrong index; but, as mentioned in the above proof, this case is absorbed by the normal form operation taken during *Reduction*.

4.2. **Applying Buchberger's First Criterion in F5.**

Theorem 4.2.1. *Let (xF_i, f) and (yF_j, g) be signed polynomials that have been added to the Grobner basis G during a run of F5. Furthermore, assume that $HT(f)$ and $HT(g)$ are disjoint. Then either $\mathcal{S}(S(f,g))$ is not normalized or $S(f,g) = o_G(HT(g)f)$.*

Proof. As per common notation, let $f = HT(f) + p$ and $g = HT(g) + q$. Then $S(f,g) = HT(g)f - HT(f)g$. If $i > j$, $\mathcal{S}(S(f,g)) = HT(g)xF_i$; since $g \in GB\langle f_j, \ldots, f_m\rangle$, $\mathcal{S}(S(f,g))$ is not normalized. The case of $i < j$ is similar.

Without loss of generality, assume $i = j$. Also, without loss of generality, assume $\mathcal{S}(S(f,g)) = HT(g)xF_i$. We remind ourselves that:

$$\begin{aligned} S(f,g) &= (g-q)f - (f-p)g \\ &= gf - qf - fg + pg \\ &= pg - qf \end{aligned} \tag{4.1}$$

Since $HT(f)$ and $HT(g)$ are disjoint, $HT(pg) \neq HT(qf)$. Note that both $\mathcal{S}(pg)$ and $\mathcal{S}(qf) \prec \mathcal{S}(S(f,g)) = HT(g)xF_i$. Since $f, g \in G$, $S(f,g) = o_G(HT(g)f)$.

□

An immediate corollary to this would be:

Corollary 4.2.1. *Buchberger's First Criterion is a valid criterion for eliminating critical pairs in F5.*

Proof. Immediate.

□

Thus we have shown that Buchberger's first criterion is quite compatible with the F5 criterion. This is not only of note because it gives us a new method for eliminating critical pairs during a run of F5, but it forms another concrete link between F5 and previous Grobner-basis-producing algorithms.

Of course, the next question to ask is: Does using Buchberger's first criterion actually save time in a run of F5? That is to say, we know it is fair to apply Buchberger's first criterion for eliminating pairs; do we know that it is removing pairs that F5 wouldn't have removed before Reduction? The following corollary to Theorem 4.2.1 explores this question.

Corollary 4.2.2. *Let $(f_1, \ldots, f_m)$ be a regular sequence of polynomials. Assume this sequence is input for F5. Then no critical pairs of the form (t, u_1, r_1, u_2, r_2), with $HT(r_1)$ and $HT(r_2)$ disjoint, will be considered by SPol.*

Proof. Assume, for contradiction, such a pair exists – call it (t, u_1, r_1, u_2, r_2), as in the corollary's hypothesis. Then $\mathcal{S}(S(r_1, r_2)) = \mathcal{S}(u_1 r_1)$ and $u_1 r_1$ is normalized. We know from equation 4.1 that $S(r_1, r_2) = pr_2 - qr_1$, where r_1 and r_2 are taking the place of f and g respectively. Note that this representation of $S(r_1, r_2)$ has smaller signature than $\mathcal{S}(u_1 r_1)$.

Thus we have two elements $g_1, g_2 \in \mathcal{P}^m$, g_1 with signature $\mathcal{S}(u_1 r_1)$ and g_2 with signature less than $\mathcal{S}(u_1 r_1)$ such that $v(g_1) = v(g_2) = S(r_1, r_2)$, where v is the map

introduced in section 3.1. Then $(g_1 - g_2) \in Syz$. Since we assumed the input is regular, $Syz = PSyz$; this directly implies by Corollary 3.2.1 that $\mathcal{S}(g_1 - g_2) = \mathcal{S}(u_1 r_1)$ is not normalized. This contradicts our opening assumption: that such a critical pair (t, u_1, r_1, u_2, r_2) exists.

Thus, *SPol* considers no such critical pair (i.e., *CritPair* eliminates it because it is not normalized).

□

So in the case that the input sequence to F5 is regular, testing Buchberger's first criterion fails to eliminate any critical pair that F5 wouldn't eliminate anyway. (Also, by looking at the details of the proof of Corollary 4.2.2, we can see that implementing Buchberger's first criterion would have negligible effect on the timings of regular input sequences as well.) However, it is worth noting that we have not concluded Buchberger's first criterion wouldn't eliminate new (i.e., different) critical pairs if the input sequence were not regular. Thus it seems reasonable to consider the application of Buchberger's first criterion as an improvement to F5; it may help with non-regular input sequences, and it doesn't hurt if the input sequence is regular.

4.3. **Applying a Modified Buchberger's Second Criterion in F5.** This discussion of Buchberger's first criterion leads naturally into a discussion of trying to implement Buchberger's second criterion in F5. Unfortunately, Buchberger's second criterion cannot be implemented in it's classic form in F5. The reason is simple (and I use the notation introduced in section 2.3): in the classic list of hypothesis in Buchberger's second criterion, there is no constraint placed on the signature of p. In short, if p's signature is too large, the "little-o" condition will fail to materialize as easily as it did in the proof of Theorem 4.2.1.

However, we will introduce a modification to Buchberger's second criterion by taking care to look at the signature of r_k. Consider the following:

Theorem 4.3.1. ***Modified Buchberger's Second Criterion.*** *Assume, during a run of F5, that r_i, r_j and r_k have been added to the Grobner basis. Assume that $HT(r_k)|lcm(HT(r_i), HT(r_j))$ and let $t = \frac{lcm(HM(r_i),HM(r_j))}{HM(r_k)}$. Further assume that $S(r_i, r_j) = u_i r_i - u_j r_j$ and $\mathcal{S}(S(r_i, r_j)) = \mathcal{S}(u_i r_i)$. If $\mathcal{S}(tr_k) \prec \mathcal{S}(u_j r_j)$, then $S(r_i, r_j)$ need not be added.*

Proof. We will use some notation from [5]. Define $x^{\gamma_{ij}} = lcm(HM(r_i), HM(r_j))$ and similarly define $x^{\gamma_{ik}}$ and $x^{\gamma_{jk}}$. Define $S(r_i, r_k) = u_i^k r_i - u_k^i r_k$ and $S(r_j, r_k) = u_j^k r_j - u_k^j r_k$.

Now let's look at $S(r_i, r_j)$:

$$\begin{aligned} S(r_i, r_j) &= u_i r_i - u_j r_j \\ &= \frac{x^{\gamma_{ij}}}{HM(r_i)} r_i - \frac{x^{\gamma_{ij}}}{HM(r_j)} r_j \\ &= \frac{x^{\gamma_{ij}}}{x^{\gamma_{ik}}} u_i^k r_i - \frac{x^{\gamma_{ij}}}{x^{\gamma_{jk}}} u_j^k r_j \\ &= \frac{x^{\gamma_{ij}}}{x^{\gamma_{ik}}} (u_i^k r_i - u_k^i r_k) - \frac{x^{\gamma_{ij}}}{x^{\gamma_{jk}}} (u_j^k r_j - u_k^j r_k) \\ &= \frac{x^{\gamma_{ij}}}{x^{\gamma_{ik}}} S(r_i, r_k) - \frac{x^{\gamma_{ij}}}{x^{\gamma_{jk}}} S(r_j, r_k) \end{aligned} \tag{4.2}$$

Note that

$$\mathcal{S}(righthandside) = \mathcal{S}\left(\frac{x^{\gamma_{ij}}}{x^{\gamma_{ik}}}(u_i^k r_i)\right) = \mathcal{S}(u_i r_i)$$

because

$$\frac{x^{\gamma_{ij}}}{x^{\gamma_{ik}}}(u_i^k r_i) = \frac{x^{\gamma_{ij}}}{x^{\gamma_{jk}}}(u_k^j r_k) = t r_k$$

So if

$$\frac{x^{\gamma_{ij}}}{x^{\gamma_{ik}}} S(r_i, r_k) = o_G(u_i r_i)$$

and

$$\frac{x^{\gamma_{ij}}}{x^{\gamma_{jk}}} S(r_j, r_k) = o_G(u_i r_i)$$

then $S(r_i, r_j) = o_G(u_i r_i)$, which implies it is not needed (by the F5 criterion). But since

$$\frac{x^{\gamma_{ij}}}{x^{\gamma_{ik}}}(u_i^k r_i) = u_i r_i$$

and

$$u_j^k r_j = u_j r_j$$

this problem reduces to whether $S(r_i, r_k) = o_G(u_i^k r_i)$ and $S(r_j, r_k) = o_G(u_j^k r_j)$.

Let's focus our attention on $S(r_i, r_k) = u_i^k r_i - u_k^i r_k$. We know that $u_i^k r_i$ is normalized; if it wasn't, since $x^{\gamma_{ik}} | x^{\gamma_{ij}}$, $u_i r_i$ wouldn't be. So this leaves us with two distinct cases: $u_k^i r_k$ is normalized and $u_k^i r_k$ is not.

Case 1:$u_k^i r_k$ is normalized. In this case, there are a variety of things that can happen to $S(r_i, r_k)$:

(1) It is eliminated by Rewritten. In this case, by Theorem 3.4.1, $S(r_i, r_k) = o_G(u_i^k r_i)$. (See the proof.)

(2) It is eliminated by an implementation of Buchberger's first criterion (if applicable). In this case, by Theorem 4.2.1, $S(r_i, r_k) = o_G(u_i^k r_i)$.

(3) It is reduced to 0 in Reduction. Then, by definition, $S(r_i, r_k) = o_G(u_i^k r_i)$.

(4) It is added to the Grobner basis G. Then, by construction of G, $S(r_i, r_k) = o_G(u_i^k r_i)$.

(5) It is eliminated by Buchberger's second criterion. We will deal with this case later in the proof.

Case 2:$u_k^i r_k$ is not normalized. Then, as per Lemma 3.2.1,

$$u_k^i r_k = \lambda_\beta S_\beta + \sum_{l=1}^{m} p_{\mathcal{L}} f_{\mathcal{L}}$$

where $p_{\mathcal{L}} \in K[\overline{x}] \forall \mathcal{L}$, $HT(u_k^i r_k) = HT(\lambda_\beta S_\beta)$, $\mathcal{S}(\lambda_\beta S_\beta) \prec \mathcal{S}(u_k^i r_k)$,

$$\mathcal{S}\left(\sum_{l=1}^{m} p_{\mathcal{L}} f_{\mathcal{L}}\right) \prec \mathcal{S}(\lambda_\beta S_\beta)$$

and

$$HT\left(\sum_{l=1}^{m} p_{\mathcal{L}} f_{\mathcal{L}}\right) < HT(\lambda_\beta S_\beta)$$

Thus we could rewrite $S(r_i, r_j) as$:

$$S(r_i, r_j) = \frac{x^{\gamma_{ij}}}{x^{\gamma_{ik}}} S(r_i, S_\beta) - \frac{x^{\gamma_{ij}}}{x^{\gamma_{jk}}} S(r_j, r_k) - \frac{x^{\gamma_{ij}}}{x^{\gamma_{ik}}} \sum_{l=1}^{m} p_{\mathcal{L}} f_{\mathcal{L}}$$

Without loss of generality, we assume $\lambda_\beta S_\beta$ is normalized and is not *Rewritten*. So we are now back in case 1, but with $S(r_i, S_\beta)$ instead of $S(r_i, r_k)$. Now, without loss of generality, we can ignore case 2 and simply focus back on case 1.

That leaves us ready to deal with Case 1, option 5 (that is, $S(r_i, r_k)$ is eliminated by Buchberger's second criterion). In this case, just like the original $S(r_i, r_j)$ was written in terms of $S(r_i, r_k)$ and $S(r_j, r_k)$ in equation 4.2, $S(r_i, r_k)$ is written in terms of $S(r_i, r_{k_1})$ and $S(r_k, r_{k_1})$ for some $r_{k_1} \in G$. We will call such a rewrite a BSC step (BSC for Buchberger's second criterion). Since the polynomial used to rewrite must involve a decrease in signature – this was the special condition that embodied our modification to the original Buchberger's second criterion (i.e., $\mathcal{S}(tr_k) \prec \mathcal{S}(u_i r_i)$ and $\mathcal{S}(u_j r_j)$) – there will only be finitely many of these BSC steps.

We claim that $S(r_i, r_k) = o_G(u_i^k r_i)$ is true regardless of the number of BSC steps taken. We will prove this claim by induction on the number of BSC steps.

Base Case (the number of BSC is 1): This is the situation discussed under Case 1 above, but with the added condition that option 5 is not possible. Then, trivially, $S(r_i, r_k) = o_G(u_i^k r_i)$.

Assume true for the number of BSC $\leq n$, for some n. (The induction hypothesis.)

Prove for the number of BSC steps $= n + 1$. Let $S(r_{\alpha_1}, r_{\alpha_2})$ be a child of $S(r_i, r_k)$ in the BSC stepping process after n BSC steps. Assume that $S(r_{\alpha_1}, r_{\alpha_2})$ is eliminated from consideration in F5 due to Buchberger's second criterion; then $\exists r_{\alpha_3} \in G$ such that

$$S(r_{\alpha_1}, r_{\alpha_2}) = x^{\gamma} S(r_{\alpha_1}, r_{\alpha_3}) + x^{\gamma'} S(r_{\alpha_2}, r_{\alpha_3})$$

Without loss of generality, based on the arguments under Case 2, assume both $S(r_{\alpha_1}, r_{\alpha_3})$ and $S(r_{\alpha_2}, r_{\alpha_3})$ are not dismissed from F5 due to lack of normalization. Then both are under the options of Case 1, with the exception of option 5 since we are assuming that the number of BSC steps is $n + 1$. Thus, both $S(r_{\alpha_1}, r_{\alpha_3})$ and $S(r_{\alpha_2}, r_{\alpha_3})$ meet the little-o condition. It follows then, by the arguments given earlier in this proof, that $S(r_{\alpha_1}, r_{\alpha_2})$ meets the little-o condition. Since $S(r_{\alpha_1}, r_{\alpha_2})$ was arbitrary, we activate the induction hypothesis and we have $S(r_i, r_k) = o_G(u_i^k r_i)$.

This completes the proof of the claim.

So we can conclude that $S(r_i, r_k) = o_G(u_i^k r_i)$. By a similar argument, we can also conclude $S(r_j, r_k) = o_G(u_j^k r_j)$.

Thus we conclude that $S(r_i, r_j) = o_G(u_i r_i)$, and this completes the proof.

□

Now we have shown that both Buchberger's first and second criteria can be applied within the body of F5. Note that this proof does not demonstrate that applying Buchberger's second criterion would necessarily we useful; that is, we have not shown that applying Buchberger's second criterion would eliminate critical pairs that F5 would otherwise keep.

It is worth noting that Gwenole Ars applies Buchberger's second criterion in [1], but with several distinct differences between our implementation here. First, his version of F5 uses Buchberger's second criterion to determine a bound for the degree of the Grobner basis under some specialized assumptions (see [1, 2]). Second, he does

not use this application of Buchberger's second criterion to eliminate critical pairs (except those eliminated by degree under his specialized conditions).

One other point to note is that, under the current hypotheses of this modified second criterion, there is no two-out-of-three problem. This is because of the strict condition on r_k that $\mathcal{S}(tr_k) \prec \mathcal{S}(u_j r_j)$. An astute observer will note that this modified theorem could have it's hypotheses loosened a bit to:

Theorem 4.3.2. ***Modified Buchberger's Second Criterion, version 2.*** *Assume, during a run of F5, that r_i, r_j and r_k have been added to the Grobner basis. Assume that $HT(r_k)|lcm(HT(r_i), HT(r_j))$ and let $t = \frac{lcm(HT(r_i),HT(r_j))}{HT(r_k)}$. Further assume that $S(r_i, r_j) = u_i r_i - u_j r_j$ and $\mathcal{S}(S(r_i, r_j)) = \mathcal{S}(u_i r_i)$. If $\mathcal{S}(tr_k) \prec \mathcal{S}(u_i r_i)$, then $S(r_i, r_j)$ need not be added.*

This would still work because, borrowing our notation from the proof of Theorem 4.3.1,

$$\mathcal{S}(righthandside) = \mathcal{S}\left(\frac{x^{\gamma_{ij}}}{x^{\gamma_{ik}}}(u_i^k r_i)\right) = \mathcal{S}(u_i r_i)$$

However, if one were to choose this second version for application, one would have to also implement code to avoid the two-out-of-three problem.

4.4. **New Hybrid Algorithm – F5t.** Now we introduce a new hybrid of F5 and Buchberger's original algorithm. We call this hybrid F5t – the "t" standing for terminating. This new version of F5 will always terminate (**provably**), regardless of the input sequence.

It is worth noting that, on it's own, it is not difficult to create a hybrid of F5 and another Grobner-basis-generating algorithm that is guaranteed to terminate. From a strictly mathematical point of view, one could claim to have a hybrid of F5 and Buchberger's algorithm that was essentially almost entirely dominated by using Buchberger's original methods and criterions (like checking the normal forms of new polynomials relative to the polynomials already added to the Grobner basis).

The problem with such a trivial hybrid is obvious – the timings would be slower than F5 timings. The reason to use F5 in the first place is to achieve better timings (measured by doing empirical tests).

F5t however is not a trivial hybrid of algorithms, as will be demonstrated in this subsection. Rather it uses F5's criterion for including/excluding critical pairs extensively, using normal form computations only when required to achieve termination. Thus F5t has timings that conserve much of the gain made by F5.

The author programmed F5t in MAGMA (see [13]), using Stegers' program in [15] as a starting point. An array of empirical experiments were conducted, comparing Stegers' implementation of F5 and F5t.

In section 4.4.1, the F5t criterion – the analog to the F5 criterion – is proven.

4.4.1. *The F5t Criterion.* We will need a few tools in order to construct our new criterion for the hybrid algorithm. First we must recall the definition of $\leq_1$ from section 3.3. Then, just as in section 3.3, we will need two lemmas. These lemmas are very similar in both hypotheses and proofs to Lemmas 3.3.1 and 3.3.2 respectively. The reader will notice that the hypotheses to Lemmas 4.4.1 and 4.4.2 are more complicated than their respective F5 analogs. The proofs of Lemmas 3.3.2 and 4.4.2 are extremely similar. The reader should focus more on the change in the hypotheses than the proofs; it is the new hypotheses that hold the new ideas that will be used to form F5t.

Lemma 4.4.1. *Let $F = (f_L, \ldots, f_m)$ be a list of polynomials in $K[\overline{x}]$, K a field. Let $G = (r_1, \ldots, r_{|G|}) \in R^{|G|}$ such that: $G_L = Poly(G)$; $G = N \cup D \cup B$ with N, D and B pairwise disjoint; $f_L \subset Poly(N)$; $f_{L+1}, \ldots, f_m \subset (B)$; B is a Grobner Basis for $\langle f_{L+1}, \ldots, f_m \rangle$; all the r_i are admissible; and $\mathcal{S}(r_j) \succ F_L \forall r_j \in D \cup N$. Let $f \in \langle G_L \rangle$,*

$$f = \sum_{d_i \in D} t_i d_i + \sum_{n_i \in N} s_i n_i + \sum_{b_i \in B} z_i b_i$$

$s_i, t_i, z_i \in K[\overline{x}]$, with the following conditions on the representation of f:

(1) $HT(f) = t$

(2) $t' = max(max_{d_i \in D} HT(t_i d_i), max_{n_i \in N} HT(s_i n_i), max_{b_i \in B} HT(z_i b_i))$
$= max_{n_i \in N}(HT(s_i' n_i))$

(3) $t' > t$ *(in the term order)*

(4) *The above representation of f is minimal (under $\leq_1$) among all representations of f satisfying conditions (1) through (3).*

Then (s_i, r_i) is normalized $\forall r_i \in N$.

Proof. Assume $\exists \mathcal{L}$ such that $(s_{\mathcal{L}}, r_{\mathcal{L}})$ is not normalized, $r_{\mathcal{L}} \in N$. In other words, $\mathcal{S}(r_{\mathcal{L}}) = uF_L$ and $HT(s_{\mathcal{L}})u \in \langle f_{L+1}, \ldots, f_m \rangle$. Since all the r_i are admissible,

$$n_{\mathcal{L}} = \sum_{j=L}^{m} w_j f_j$$

where $HT(w_L) = u$. Then we can say the following about f:

$$\begin{aligned} f &= \sum_{d_i \in D} t_i d_i + \sum_{\substack{n_i \in N \\ i \neq \mathcal{L}}} s_i n_i + \sum_{b_i \in B} z_i b_i + s_{\mathcal{L}} n_{\mathcal{L}} \\ &= \sum_{d_i \in D} t_i d_i + \sum_{\substack{n_i \in N \\ i \neq \mathcal{L}}} s_i n_i + \sum_{b_i \in B} z_i b_i + s_{\mathcal{L}} \sum_{j=L}^{m} w_j f_j \qquad (4.3) \\ &= \sum_{d_i \in D} t_i d_i + \sum_{\substack{n_i \in N \\ i \neq \mathcal{L}}} s_i n_i + \sum_{b_i \in B} z_i b_i + s_{\mathcal{L}} w_L f_L + s_{\mathcal{L}} \sum_{j=L+1}^{m} w_j f_j \end{aligned}$$

By Lemma 3.2.1, we know that

$$s_{\mathcal{L}} w_L f_L = (a + \sum_{\substack{r \in G \\ \mathcal{S} \prec F_L}} \lambda_r poly(r)) f_L$$

where a is a polynomial in $K[\overline{x}]$, $HT(a) < HT(s_{\mathcal{L}} u)$ and $\lambda_r \in K[\overline{x}] \forall r \in G$.

Since $\mathcal{S}(r_j) \succ F_L \forall j \in D$, this rewrite doesn't use polynomials from D. Thus conditions (1) through (3) still hold. Yet we created a new representation of f that is smaller under $\leq_1$. This contradicts condition (4) above.

Thus we must conclude that (s_i, r_i) is normalized $\forall r_i \in N$. □

Lemma 4.4.2. *Adopt all notation and assumptions from Lemma 4.4.1. Let* $I = \{n_j \in N | HT(s_j n_j) = t'\}$, $w = max\{\mathcal{S}(s_i n_i) | i \in I\}$ *and let* $J = \{i \in I | \mathcal{S}(s_i n_i) = w\}$. *Then* $|J| = 1$.

Proof. Assume for contradiction that $|J| \geq 2$. Let $k_1 = min\{i | i \in J\}$ and $k_2 = min\{i | i \in J \backslash k_1\}$. Let $w = uF_L$. So,

$$f = \sum_{d_i \in D} t_i d_i + \sum_{\substack{n_i \in N \\ n_i \neq k_1, k_2}} s_i n_i + \sum_{b_i \in B} z_i b_i + s_{k_1} n_{k_1} + s_{k_2} n_{k_2}$$

For ease of reading throughout this proof, denote

$$\Phi = \sum_{d_i \in D} t_i d_i + \sum_{\substack{n_i \in N \\ n_i \neq k_1, k_2}} s_i n_i + \sum_{b_i \in B} z_i b_i$$

Thus we are now working with the following representation of f:

$$f = \Phi + s_{k_1} n_{k_1} + s_{k_2} n_{k_2}$$

We will denote

$$n_{k_1} = \sum_{j=L}^{m} h_j^1 f_j, n_{k_2} = \sum_{j=L}^{m} h_j^2 f_j$$

This leaves us with the following chain of equalities, for some $c, d \in K$:

$$
\begin{aligned}
f &= \Phi + s_{k_1}(\sum_{j=L}^{m} h_j^1 f_j) + s_{k_2}(\sum_{j=L}^{m} h_j^2 f_j) \\
&= \Phi + sk_1 h_L^1 f_L + s_{k_2} h_L^2 f_L + s_{k_1}(\sum_{j=L+1}^{m} h_j^1 f_j) + s_{k_2}(\sum_{j=L+1}^{m} h_j^2 f_j) \\
&= \Phi + (cu + (s_{k_1} - HT(s_{k_1}))h_L^1 + HT(s_{k_1})(h_l^1 - HT(h_L^1)))f_L \\
&+ (du + (s_{k_2} - HT(s_{k_2}))h_L^2 + HT(s_{k_2})(h_l^2 - HT(h_L^2)))f_L \\
&+ s_{k_1}(\sum_{j=L+1}^{m} h_j^1 f_j) + s_{k_2}(\sum_{j=L+1}^{m} h_j^2 f_j) \\
&= \Phi + ((c+d)u + (s_{k_1} - HT(s_{k_1}))h_L^1 + HT(s_{k_1})(h_l^1 - HT(h_L^1)))f_L \\
&+ ((s_{k_2} - HT(s_{k_2}))h_L^2 + HT(s_{k_2})(h_l^2 - HT(h_L^2)))f_L \\
&+ s_{k_1}(\sum_{j=L+1}^{m} h_j^1 f_j) + s_{k_2}(\sum_{j=L+1}^{m} h_j^2 f_j) \\
&= \Phi + ((c+d)u + (s_{k_1} - HT(s_{k_1}))h_L^1 + HT(s_{k_1})(h_l^1 - HT(h_L^1)))f_L \\
&+ (s_{k_2} - HT(s_{k_2}))(\sum_{j=L}^{m} h_j^2 f_j) + (HT(s_{k_2}))(h_L^2 - HT(h_L^2))f_L \\
&+ s_{k_1}(\sum_{j=L+1}^{m} h_j^1 f_j) + (HT(s_{k_2}))(\sum_{j=L+1}^{m} h_j^2 f_j) \\
&= \Phi + s'_{k_1} n_{k_1} + (s_{k_2} - HT(s_{k_2}))n_{k_2} - d(HT(s_{k_1}))(\sum_{j=L+1}^{m} h_j^1 f_j) \\
&+ (HT(s_{k_2}))(h_L^2 - HT(h_L^2))f_L
\end{aligned}
$$

where $s'_{k_1} = s_{k_1} + dHT(s_{k_1})$.

This leaves us with the following representation of f:

$$f = \sum_{d_i \in D} t_i d_i + \sum_{\substack{n_i \in N \\ n_i \neq k_1, k_2}} s_i n_i + \sum_{b_i \in B} z_i b_i$$

$$+ s'_{k_1} n_{k_1} + (s_{k_2} - HT(s_{k_2}))n_{k_2} \tag{4.4}$$

$$- d(HT(s_{k_1}))(\sum_{j=L+1}^{m} h_j^1 f_j) + (HT(s_{k_2}))(h_L^2 - HT(h_L^2))f_L$$

This representation of f is smaller (under $\leq_1$) than $\sum_{d_i \in D} t_i d_i + \sum_{\substack{n_i \in N \\ n_i \neq k_1, k_2}} s_i n_i + \sum_{b_i \in B} z_i b_i$. This fact can be divined by looking at four separate cases:

(1) $f_L = n_x$ for some $x \neq k_1, k_2$ where $\mathcal{S}(s_x n_x) \succ \mathcal{S}(s_{k_1} n_{k_1})$. In this case, $(HT(s_{k_2}))(h_L^2 - HT(h_L^2))f_L$ can be "absorbed" into $\sum_{\substack{n_i \in N \\ n_i \neq k_1, k_2}} s_i n_i$ by altering s_x to $s_x + (HT(s_{k_2}))(h_L^2 - HT(h_L^2))$. This change to s_x does not change $\mathcal{S}(s_x r_x)$. Of the remaining summands in equation 4.4, $\mathcal{S}(s'_{k_1} n_{k_1}) = \mathcal{S}(s_{k_1} n_{k_1})$, $\mathcal{S}((s_{k_2} - HT(s_{k_2}))n_{k_2}) \prec \mathcal{S}(s_{k_2} n_{k_2})$ and $dHT(s_{k_1})(\sum_{j=L+1}^{m} h_j^1 f_j)$ has the smallest signature of the three. Thus this new representation of f is smaller (under $\leq_1$) than $\sum_{d_i \in D} t_i d_i + \sum_{\substack{n_i \in N \\ n_i \neq k_1, k_2}} s_i n_i + \sum_{b_i \in B} z_i b_i$.

(2) $f_L = n_{k_1}$. In this case the exact same argument as in (1) holds, except "absorption" would occur into $s'_{k_1} n_{k_1}$.

(3) $f_L = n_{k_2}$. In this case, f could have been written without $s'_{k_1} n_{k_1}$ at all. This would have immediately led to a contradiction as in case (1).

(4) $f_L = n_x$ for some $x \neq k_1, k_2$ where $\mathcal{S}(s_x n_x) \prec \mathcal{S}(s_{k_1} n_{k_2})$. Then the exact same argument as in case (1) holds, except tracking the resulting signature of $s_x r_x$ after "absorption" is irrelevant (because $\mathcal{S}(s_x n_x) \prec \mathcal{S}(the new representation of f, under \leq_1)$).

This list covers all possible cases for f_L. Thus we are forced to conclude that our assumption that $|J| \geq 2$ is false.

Thus, $|J| = 1$. □

Now we can prove the F5t criterion. Instead of simply looking at a set G of signed polynomials, we split G into three disjoint subsets B, N and D.

Theorem 4.4.1. *Let $F = (f_L, \ldots, f_m)$ be a list of polynomials in $K[\overline{x}]$, K a field. Let $G = \{r_1, \ldots, r_{|G|}\}$, $r_i \in R$ $\forall i$, such that:*

(1) *$G = N \cup D \cup B$; N, D and B pairwise disjoint; Poly$(N) = (n_1, \ldots, n_{|N|})$; Poly$(D) = (d_1, \ldots, d_{|D|})$; Poly$(B) = (b_1, \ldots, b_{|B|})$; $f_L \subset$ Poly(N); $f_{L+1}, \ldots, f_m \subset$ Poly(B); $G_L =$Poly(G)*

(2) *all the r_i are admissible*

(3) *$\forall r_i, r_j$ such that $r_i, r_j \in N$ (or $r_i \in N$ and $r_j \in B$), if (r_i, r_j) is normalized then $S(n_i, n_j) = o_{G_L}(u_i HT(n_i))$ (or 0) where $u_i = \frac{lcm(HT(n_i), HT(n_j))}{HT(n_i)}$*

(4) *$\forall r_i \in D, r_j \in G, S(r_i, r_j) \xrightarrow[G_L]{*} 0$*

(5) *$\forall r_i \in D, \mathcal{S}(r_i) \succ F_L$*

(6) *B is a Grobner Basis for $\langle f_{L+1}, \ldots, f_m \rangle$*

Then G_L is a Grobner Basis for $\langle f_L, \ldots, f_m \rangle$.

Proof. Let $t \in HT(\langle F \rangle)$. We will show that t is top-reducible via G_L, thus making G_L a Grobner Basis of $\langle F \rangle$.

Let $f \in \langle F \rangle$,

$$f = \sum_{d_i \in D} t_i d_i + \sum_{n_i \in N} s_i n_i + \sum_{b_i \in B} z_i b_i$$

such that:

(1) $HT(f) = t$

(2) $t' = max(max_{d_i \in D} HT(t_i d_i), max_{n_i \in N} HT(s_i n_i), max_{b_i \in B} HT(z_i b_i))$ and no other polynomial in $\langle F \rangle$ would have a smaller t' (under the term order) while still holding condition (1).

Note that this is possible because of assumption 1 of the theorem and the fact that $f \in \langle F \rangle$.

Assume, for contradiction, that $t' > t$. Then there is a syzygy on the leading terms of the $s_i n_i$'s, $t_i d_i$'s and $z_i b_i$'s. By Theorem 2.5.2, the module of syzygies on the leading terms of these polynomials is generated by all the S-polynomials of these polynomials. Thus, by assumptions 4 and 6 of this theorem, f can be reduced (not necessarily top-reduced) to a new f',

$$f' = \sum_{d_i \in D} t'_i d_i + \sum_{n_i \in N} s'_i n_i + \sum_{b_i \in B} z'_i b_i$$

such that

(3) $t'' = max(max_{d_i \in D} HT(t'_i d_i), max_{n_i \in N} HT(s'_i n_i), max_{b_i \in B} HT(z'_i b_i))$
$= max_{i \in N}(HT(s'_i n_i))$

(4) Among all representations of f' that satisfy condition (3), the chosen representation is minimal under $\leq_1$ (the order introduced in Definition 3.3.1).

Note that $f = f'$; f' is only a different representation of f. If $t'' < t'$, then we have contradicted the minimality of t' under constraint (2). Thus we know $t'' = t'$.

Then by Lemma 4.4.1, (s'_i, r_i) is normalized $\forall r_i \in N$. Let $I = \{j \in N | HT(s'_j n_j) = t'\}$, $w = max\{\mathcal{S}(s'_i n_i) | i \in I\}$ and $J = \{i \in I | \mathcal{S}(s'_i n_i) = w\}$. By Lemma 4.4.2, $|J| = 1$. Note that $|I| \geq 2$ because $t' > t$ (i.e., there must be a syzygy on the leading terms of the $s'_i n_i$'s). Let $k \in J$ and $\mathcal{L} \in I \backslash \{k\}$.

Then,

$$f' = \sum_{d_i \in D} t'_i d_i + \sum_{\substack{n_i \in N \\ i \neq k,l}} s'_i n_i + \sum_{b_i \in B} z'_i b_i + s'_k n_k - \frac{HC(s'_k)}{HC(s'_\mathcal{L})} s'_\mathcal{L} n_\mathcal{L} + \left[1 + \frac{HC(s'_k)}{HC(s'_\mathcal{L})}\right] s'_\mathcal{L} n_\mathcal{L}$$

For ease of notation in the upcoming argument, define $m'_k = HM(s'_k)$ and $m'_\mathcal{L} = \frac{HC(s'_k)}{HC(s'_\mathcal{L})} HM(s'_\mathcal{L})$. Then, by definition of I and J, $t' = HT(m'_k n_k) = HT(m'_\mathcal{L} n_\mathcal{L})$. Consequently, $lcm(HT(n_k), HT(n_\mathcal{L}))|t'$; in other words,

$$m'_k n_k - m'_\mathcal{L} n_\mathcal{L} = \frac{HC(s'_k) t'}{lcm(HT(n_k), HT(n_\mathcal{L}))} S(n_k, n_\mathcal{L})$$

Since (s'_k, n_k) and $(s'_{\mathcal{L}}, n_{\mathcal{L}})$ are normalized, $(r_k, r_{\mathcal{L}})$ is normalized. Thus, using assumption 3 of the theorem, in a manner similar to the argument in the proof of Theorem 3.3.1,

$$m'_k n_k - m'_{\mathcal{L}} n_{\mathcal{L}} = \frac{t'}{lcm(HT(n_k), HT(n_{\mathcal{L}}))} o_{G_L}(u_k, r_k) = o_{G_L}(s'_k r_k)$$

This means that there is a smaller representation of f', under $\leq_1$, such that condition (3) still holds (a contradiction to condition (4)) or a new representation of f' has been discovered such that t' under this new representation is smaller than the original t' (a contradiction to condition (2)).

We are forced to conclude that our original assumption that $t' > t$ is false. Thus $t' = t$, making t top-reducible via G_L.

Since t was arbitrary, G_L is a Grobner Basis of $\langle F \rangle$. □

4.4.2. *The F5t Algorithm.* Here we will outline the pseudocode for F5t. As mentioned previously, F5t relies heavily on the skeleton of F5 because it tries to utilize as much of the F5 criterion as possible. However, the reader should note that the following algorithm uses the F5t criterion, outlined in section 4.4.1 in Theorem 4.4.1.

We will begin by showing the modification to *Reduction*.

F5tReduction

Input: ToDo and G_i finite lists of signed polynomials; k an integer; ϕ_{i+1} a normal form mapping; a set of polynomials D; an integer bound M.
Output: Done a finite list of reduced signed polynomials; addtoD a boolean.

$Done := \emptyset$;
while $ToDo \neq \emptyset$ do
  h :=Sig_Sort($ToDo$)[1];
  $ToDo := ToDo \backslash \{h\}$;

$(h_1, ToDo_1)$:=TopReduction$(\phi_{i+1}(h), G_i \cup Done, k, \phi_{i+1})$;

$Done := Done \cup h_1$;

$ToDo := ToDo \cup ToDo_1$;

$temp := Done$;

$reducerset := Done \cup G_i \cup D$;

$addtoD := 0$;

if degree of polynomials in $Done < M$ and $Done \neq \emptyset$ then

$h := Done[1]$;

$Done := Done \backslash \{h\}$;

h :=NormalForm$(h, reducerset)$;

if $h \neq 0$ then

$Done := temp$;

break;

if $Done = \emptyset$ then

$addtoD := 1$;

$Done := temp$;

for $r \in Done$ do

RemoveRule(k, r);

return $Done$;

The subroutine *F5tReduction* regulates entry of signed polynomials into the set D, D being the set described in Theorem 4.4.1. Once the statements from F5's *Reduction* have been completed, *F5tReduction* carries on additional steps. These steps are meant to decide whether any new candidate polynomial is truly adding to the ideal of head terms being created. If it is the case that the degree is high enough and all candidate polynomials of that degree reduce to 0 (please note that this reduction operation is the normal form operation from Buchberger's algorithm – no additional conditions, like those in *IsReducible*, are used) then the boolean

addtoD is set to 1 (true); otherwise, *addtoD* remains 0 (false) and everything carries on as usual as per the original F5 algorithm.

F5tReduction carries with it two other items of note. First, the boolean *addtoD* acts exactly as is implied by it's name: if *addtoD* is set to 1, then all the signed polynomials of *Done* will be placed in the set D. Second, there is a new routine *RemoveRule*. This routine is the opposite of *AddRule*. Given an integer input k and a signed polynomial r, *RemoveRule* searches through the list of lists that make up the rules, finds sublist k (i.e., the sublist for index k), searches for a rule for polynomial r and removes it. This must be done for the following reason. As was described in section 3.4.1, *Rewritten* uses rules to eliminate signed polynomials and this elimination is valid through the "little-o" condition. The hypotheses of Theorem 4.4.1 do not imply any "little-o" condition existing relative to the polynomials in D. Thus the rules for polynomials in D cannot be used during the running of F5t and must be removed.

This spirit of rule management with respect to D can be seen again in *F5tSPol*. Note the slight change from *SPol*.

F5tSPol

Input: A list $[p_1, p_2, \ldots, p_h]$ *of critical pairs of the form* (t, u_1, r_1, u_2, r_2)*; a list hasbeeninD of signed polynomials that are in D (or have been during a previous iteration).*

Output: A list F of new, signed polynomials that is sorted in ascending order by $\mathcal{S}$*.*

for $\mathcal{L} = 1$ to h do

  $p_{\mathcal{L}} := (t_{\mathcal{L}}, u_{\mathcal{L}}, r_{i_{\mathcal{L}}}, v_{\mathcal{L}}, r_{j_{\mathcal{L}}})$;

$F := \emptyset$;

for $\mathcal{L} = 1$ to h do

if (not$((r_{i_{\mathcal{L}}} \in hasbeeninD)$ or $(r_{j_{\mathcal{L}}} \in hasbeeninD))$) then

    if (not Rewritten?$(u_{\mathcal{L}}, r_{i_{\mathcal{L}}})$ and not Rewritten?$(v_{\mathcal{L}}, r_{j_{\mathcal{L}}})$) then

$N := N + 1$;

$r_N := (u_{\mathcal{L}}\mathcal{S}(r_{i_{\mathcal{L}}}), u_{\mathcal{L}}poly(r_{i_{\mathcal{L}}}) - v_{\mathcal{L}}poly(r_{j_{\mathcal{L}}}))$;

Add_Rule(r_N);

$F := F \cup \{r_N\}$;

F :=Sig_Sort(F);

return F;

In *F5tSPol* we outlaw using current or past elements of D for Rewritten. Again this is because the element of D are not subject to the "little-o" condition like other signed polynomials of the Grobner basis.

Another pseudocode augmentation needed to convert F5 to F5t is a change in *Algorithm_F5*.

Algorithm_F5t

Input: i an integer; f_i a polynomial; G_{i+1} a collection of signed polynomials such that $poly(G_{i+1})$ is a Grobner basis for $\langle f_{i+1}, \ldots, f_m \rangle$.

Output: A set of signed polynomials G_i where G_i is a Grobner basis for $\langle f_i, \ldots, f_m \rangle$.

$r_i := (F_i, f_i)$;

$\phi_{i+1} := NF(\cdot, poly(G_{i+1}))$;

$G_i := G_{i+1} \cup \{r_i\}$;

P :=Deg_Sort([CritPair($r_i, r, i, \phi_{i+1}, \phi_{index(r)+1}$)$|r \in G_{i+1}$]);

$D := \emptyset$;

$addedtoD := \emptyset$;

$hasbeeninD := \emptyset$;

$holder := \emptyset$;

while ($P \neq \emptyset$ or $holder \neq \emptyset$) do

  $d := deg(P[1])$;

  $P_d := \{p \in P | deg(p) = d\}$;

$P := P \backslash P_d$;

$F :=$SPol$(P_d, hasbeeninD)$;

$< R_d, addtoD >:=$Reduction(F, G_i, i, ϕ_{i+1});

if $addtoD = 0$ then

  for $r \in R_d$ do

    create all critical pairs between r and D and store in $holder$;

  Deg_Sort($holder$);

  for $r \in R_d$ do

    $P := P \cup \{\text{CritPair}(r, p, i, \phi_{i+1}, \phi_{index(p)+1} | p \in G_i\}$;

    $G_i := G_i \cup \{r\}$;

  Deg_Sort(P);

  $reducer := R_d \cup D \cup G_i$;

  while ($holder \neq \emptyset$) and ($deg(holder[1]) = d$) do

    $r := holder[1]$;

    $holder := holder \backslash \{r\}$;

    if r doesn't reduce to 0 via $reducer$ then

      $D := D \cup \{r\}$;

      $hasbeeninD := hasbeeninD \cup \{r\}$;

      $reducer := reducer \cup \{r\}$;

      create all critical pairs between $\{r\}$ and $D \cup R_d \cup G_i$ and store in $holder$;

      Deg_Sort($holder$);

else

  $reducer := R_d \cup D \cup G_i$;

  create all critical pairs between $S(r, h)$ and $D \cup R_d \cup G_i$ and store in $holder$;

  Deg_Sort($holder$);

  $D := D \cup R_d$;

  $hasbeeninD := hasbeeninD \cup R_d$;

  while ($holder \neq \emptyset$) and ($deg(holder[1]) = d$) do

    $r := holder[1]$;

$holder := holder \backslash \{r\}$;

if r doesn't reduce to 0 via $reducer$ then

$D := D \cup \{r\}$;

$hasbeeninD := hasbeeninD \cup \{r\}$;

$reducer := reducer \cup \{r\}$;

create all critical pairs between $\{r\}$ and $D \cup R_d \cup G_i$ and store in $holder$;

Deg_Sort($holder$);

return $G_i \cup D$;

To describe this section of pseudocode, I will refer to the notation used in Theorem 4.4.1.

In the above section of code B is the Grobner basis from the previous iteration. The set N is composed of all polynomials added to G_i in the line

$$G_i := G_i \cup \{r\};$$

The bottom portion of code ensures that all S-polynomials borne from the signed polynomials in D (and children of such S-polynomials) reduce to 0 via G_i. Thus this section of code is enforcing hypothesis (4) from Theorem 4.4.1. Every time an S-polynomial does not reduce to 0, it is manually added to D (and thus, eventually, G_i) and the process begins again. This process is regulated by a special list $holder$. As long as $holder$ has a critical pair of the current degree to be considered, the bottom portion of this pseudocode continues to run.

One aspect of the pseudocode that is important to note is that most of the additional code is meant to regulate additional normal form computations. It was these computations that cause Grobner basis algorithms to have enormous time complexity. Thus we have chosen, at least in the case of *F5tReduction* to be very conservative. If even one of a batch of candidate polynomials in R_d truly contributes a new head term to the monomial ideal of head terms, then the entire

batch is accepted as per F5. This standard keeps D at a minimum. This is of great importance since every signed polynomial in D is responsible for a host of normal form calculations. Keeping D of minimal size will greatly help in reducing the time taken by F5t.

One other important item to point out is that these normal form operations are required to maintain signature. That is to say, just as in F5, we want all signed polynomials admitted into the Grobner basis to be admissible. It is quite possible that some elements of D are not normalized; after all, we are not placing extra conditions on the reductors in the bottom part of *Algorithm_F5t.* That is just fine, but the polynomials in D must still be admissible.

This completes the pseudocode needed to create F5t.

4.4.3. *Proof of F5t's Correctness.* The first order of business is to show that F5t is correct. We will prove this in a manner similar to our proof of Theorem 3.4.2. But instead of using Theorem 3.3.1, we will use Theorem 4.4.1 and check off each hypothesis one-by-one.

Theorem 4.4.2. *If $(f_L, \ldots, f_m)$ is input into F5t, the set poly(G) is a Grobner basis for $\langle f_L, \ldots, f_m \rangle$.*

Proof. We will prove the correctness of F5t using the same notation and language as Theorem 4.4.1. We will prove correctness by induction on the index i.

Base Case $(i = m)$.: This is trivially true since the only polynomial of index m admitted is (F_m, f_m), which is clearly a Grobner basis of $\langle f_m \rangle$.

Assume F5t correct for indices up to $L + 1$.

Prove F5t correct for index L. Condition (1) of Theorem 4.4.1 in essentially all notation. We can see that (F_L, f_L) is manually added to G_L during the L^{th} iteration of *Algorithm_F5t.* It is also clear that N, B and D are kept disjoint (where B is

the output of *Algorithm_F5t* after index $L+1$) during the running of F5t. Thus we have achieved hypothesis (1) of Theorem 4.4.1.

We have already seen in the proof of Theorem 3.4.2 that all polynomials added to G_L during F5's final iteration of *Algorithm_F5* are admissible. In the closing comments of section 4.4.2 we assert that the additional code augmenting F5 into F5t maintains admissibility in a similar fashion. Thus we have achieved hypothesis (2) of Theorem 4.4.1. The same set of arguments also implies hypothesis (5) of Theorem 4.4.1.

By induction hypothesis, B is a Grobner basis of $\langle f_{L+1}, \ldots, f_m \rangle$. Thus we have achieved hypothesis (6) of Theorem 4.4.1.

The tail-end of the pseudocode of *Algorithm_F5* ensures, by exhaustion, that hypothesis (4) of Theorem 4.4.1 is achieved.

This only leaves hypothesis (3) of Theorem 4.4.1. This is clearly witnessed by theorem 3.4.2 as this hypothesis has nothing to do with the set D; as long as a polynomial is not in D, the "little-o" condition is satisfied.

Thus F5t is correct. This completes the proof.

□

We are now ready to deal with the issue of F5t's termination.

4.4.4. *Proof of F5t's Termination.* Unlike the F5 algorithm, F5t is always guaranteed to terminate. We will prove this fact now.

Theorem 4.4.3. *For an input* $(f_1, \ldots, f_m)$*, F5t terminates.*

Proof. To prove this theorem we could prove that there is termination at every index. We will prove this by induction on the index i.

Base Case $(i = m)$.: This is trivially true since the only polynomial of index m admitted is (F_m, f_m).

Assume F5t terminates for indices up to $L+1$.

Prove F5t terminates for index L. Let d be the smallest degree such that $D \cup G_L$ form a Grobner basis of $\langle f_L, \ldots, f_m \rangle$ and $d \geq M$ (if this is impossible then F5t must have terminated prior to degree d, in which case we are done). In other words, the degree-d Grobner basis for $\langle f_L, \ldots, f_m \rangle$ is actually a Grobner basis for $\langle f_L, \ldots, f_m \rangle$. By Theorem 2.2.2, we know that such a degree d exists; by Theorem 4.4.3, we know that F5t will achieve the Grobner basis for $\langle f_L, \ldots, f_m \rangle$ at degree d.

Then we can turn our attention to the subroutine *F5tReduction*. In the pseudocode (in index L), we note that $reducerset$ is comprised of G_L and D. Thus, once we are at (or beyond) degree d, the reducerset is a Grobner basis for $\langle f_L, \ldots, f_m \rangle$. Thus, from degree d on, $reducerset$ will reduce all polynomials to 0. This means that all subsequent calls to *F5tReduction* will result in $addtoD$ being set to 1.

This will cause the tail-end of *Algorithm_F5t* to never add any signed polynomials to P (P is only used if $addtoD = 0$). This means at some degree $d' \geq d$, $P = \emptyset$.

Once $P = \emptyset$, the only carrier of critical pairs left is $holder$. But, just as in *F5tReduction*, the $reducer$ in *Algorithm_F5t* is comprised of G_L and D; this means that the $reducer$ in *Algorithm_F5t* is also a Grobner basis for $\langle f_L, \ldots, f_m \rangle$. Thus, at degree d' (starting at degree d), all S-polynomials borne from elements in D will reduce to 0 via $reducer$. This means nothing new is added to $holder$. This means at some degree $d'' \geq d' \geq d$, $P = holder = \emptyset$.

This is the condition that terminates the outer while-loop in *Algorithm_F5t*. Thus we have proven termination at index L.

This completes the induction: *Algorithm_F5t* terminates at every index i.

This completes the proof.

□

Note that this proof of termination relies heavily on Theorem 4.4.3 – our theorem of F5t's correctness. This relationship is similar to the relationship between termination and correctness in Buchberger's algorithm. The appearance of the relationship in our proof should come as no surprise. This additional code at the bottom of *Algorithm_F5t* is essentially a hybridization Buchberger's algorithm with F5. In short, we are using Buchberger's exhaustive approach to patch the holes in F5.

4.4.5. *Timing Data for F5 and F5t.* In this section we compare the timings between F5 and F5t. The table below yields a collection of data. The systems of polynomials used for the empirical test were found in an appendix of [15].

The code for F5 was taken from [15] and the code for F5t was created by the author; both programs were written in MAGMA ([13]). The times given are in seconds. They were compiled on an Athlon-chip home computer.

The variable M was set to twice the Macaulay bound for the input sequence (see [2]).

The author was only able to afford the student version of MAGMA, which allows a maximum of 100MB of memory per program run. In some cases the input to F5 and F5t required more memory than that, in which case the program was forced to prematurely terminate. These cases are denoted with an "ML" (memory limit). In the single case where F5 did not terminate, an "x" is shown.

The confirmation column describes the author's attempts to verify externally that the output of F5t was in fact a valid Grobner basis for the input sequence. The test for verifying that the output is, in fact, a Grobner basis is extremely costly. When possible, confirmation was made (c); there were also times when an output was impossible (due to time constraints) to verify (n); finally, there were cases where no output was obtained to be tested (x).

TABLE 1. Timing Data for F5 and F5t

TIMING DATA FOR F5 AND F5t				
Polynomial System	F5 Timing	Confirmation	F5t Timing	Comment
Augot	ML	x	ML	
BenchmarkD1	5396.125	n	5345.000	
Bronstein86	0.000	c	0.000	
Buchberger87	0.000	c	0.000	
Butcher	ML	x	ML	
Caprasse	0.297	c	0.313	
Chemkin	175.765	n	XL	174.672
Cyclic5	0.063	c	0.094	
Cyclic6	2.984	c	3.016	
Cyclic7	336.360	n	336.922	
Cyclic8	ML	x	ML	
Czapor86	0.047	c	0.000	
Ducos1	ML	x	ML	
Eco6	0.125	c	0.141	
f633	0.156	c	0.172	
f744	958.282	n	877.968	
f855	ML	x	ML	
Fabrice24	128.781	n	XL	128.969
Faugere'sF5Paper(MMT)	0.016	c	0.000	
Filter9	ML	x	ML	
Gerdt93	0.031	c	0.032	
Gonnet83	ML	x	ML	
Hairer1	0.000	c	0.016	
Hairer2	ML	x	ML	
ISSACChallenge	0.063	c	0.015	
Katsura10	ML	x	ML	
Katsura3	0.016	c	0.016	
Katsura5	0.109	c	0.110	
Katsura6	0.343	c	0.297	
Katsura7	4.968	n	4.969	
Katsura8	81.297	n	81.750	
Katsura9	ML	x	ML	
KernE150	ML	n	ML	
Lichtblau	3.860	c	3.843	
Liu	0.015	c	0.015	
Neff89	0.000	c	0.000	
Non-terminatingExample	x	c	0.843	3.840
Noon3	0.109	c	0.031	
Noon4	0.106	c	0.031	
Pavelle	0.000	c	0.015	
Schrans-Troost	ML	n	ML	
SegersHFE	1.594	c	1.656	
Sym1-411	0.000	c	0.000	
Sym3-3	0.016	c	0.015	
Sym3-4	0.016	c	0.015	
Trinks	0.032	c	0.015	
Trivial1	0.000	c	0.000	
Trivial2	0.000	c	0.000	
Uteshev-Bikker	0.000	c	0.015	
Vermeer	1.703	c	1.657	
Wang89	0.000	c	0.000	
Weispfenning94	0.188	c	0.281	

There were two cases where F5t took an extremely long time and the program was forcefully terminated. Those cases are marked with an "XL" (extremely long). These cases were re-tested with M set to three times the Macaulay bound. The times of those runs are given in the comments column.

Finally, the Non-terminating example was run on a coded version of Buchberger's algorithm (in MAGMA). The runtime for this implementation is given in the comments column.

Remark 4.4.1. Just as Stegers does in [15], we make no claim that the code for F5 or F5t is most efficient. In fact, it is quite likely that a matrix-oriented implementation would work more quickly in both cases ([7, 15]). However the data does show that, in most cases, both F5 and F5t are the same order of magnitude for the standard bound M.

Remark 4.4.2. Anytime an element is added to D, the Buchberger portion of the algorithm starts doing it's exhaustive tests. In some cases this results in a very huge delay in receiving the output. Specifically we can see that Chemkin and Fabrice24 took so much additional time that they became impossible (on a home computer with limited resources) for which to receive timings; on the other hand, the Non-terminating example that doesn't even terminate in F5 terminates very quickly in F5t. This set of extreme cases highlights the fragility of the set D. Though D allows us to guarantee termination it can also dramatically increase the time unnecessarily.

The Chemkin and Fabrice24 systems were run again with M set to three times the Macaulay bound. In these cases, the program terminated before F5t even began testing for polynomial entries into D. It seems that a conservative approach to testing for D is appropriate. One seems to be better off setting a relatively high value for M in the hopes that F5t (and, thus, F5) will terminate on its own before reaching that degree.

Remark 4.4.3. We should also take note that Buchberger's algorithm takes about four times as long computing the Non-terminating example for F5. This illustrates that F5t is using it's F5-component to achieve a speed advantage over Buchberger's algorithm. The advantage is by about a factor of 4. Thus it seems the hybrid program is demonstrating a level of success – it is providing provable termination (which F5 cannot provide in all cases) while still moving faster than Buchberger.

5. Conclusion

In this final section we will attempt to take stock in what we have accomplished in this thesis.

On a practical note we have implemented a new hybrid Grobner-basis-producing algorithm called F5t (t for terminating) that provably terminates for all input sequences. F5t is a hybrid of the classic Buchberger's algorithm and F5 found in [7]. We prove that this hybrid is correct using a new Grobner basis criterion introduced in sub-subsection 4.4.1; we also prove its termination for all input sequences. This guaranteed termination comes at the cost of increased run times for some input sequences, as illustrated in sub-subsection 4.4.5. We conclude that extra effort should be made to make D minimal in size.

Much has been gained in terms of mathematical theory. In subsection 3.5, we highlight an error in a previous (and often cited – see [1, 2, 15]) proof of F5's termination for regular input sequences; in this same section, we offer a minor alteration to the code of F5 that should alleviate the problem under certain weak additional constraints to the input sequences over finite fields. (In Appendix A we give this topic a more thorough discussion.) In sub-subsection 3.4.1, we prove the validity of the subroutine *Rewritten* in the language and criteria of F5 for the first time (to our knowledge). In subsection 4.1 we verify a minor improvement to F5 first introduced in [15]. In subsections 4.2 and 4.3 we apply Buchberger's first and (modified) second criteria to F5; these applications come with some corollary results. In subsections 3.1 and 3.2 we prove Proposition 1 from [7] (which, to our knowledge, had not been proven in print before) and we prove two new important ideas: the normalization lemma (Lemma 3.2.1) and that normalized polynomials derived from regular sequences have minimal signature (Theorem 3.2.1).

There is much room for future work, and we hope that this thesis serves a springboard for additional strides in Grobner basis theory. There is specific room for study in minimizing the set D. Finding a bound for the variable M given certain

properties of the input (like regularity, for example) would be a great start. In addition, there may be ways to better integrate D with the rewriting rules so that, if D does become non-empty, more polynomials can be removed without reduction. It would also seem prudent to use a better hybridization that the one in this paper (now that a hybridization has been established); replacing the Buchberger's-algorithm portion with an F4 [8] portion might save a significant amount of run time. In a different vein of study, it is still worthwhile to determine precisely when F5 does terminate.

References

[1] Ars, Gwenole. *Applications des Bases de Grobner a la Cryptographie.* Diss. U. de Rennes, 2005

[2] Bardet, Magali. *On the Complexity of a Grobner Basis Algorithm.* Ed. Bruno Salvy. In (Ed. F. Chyzak) INRIA 2005: 85-92. 20 November 2006. <http://algo.inria.fr/seminars/summary/Bardet2002.pdf>.

[3] Becker, Thomas, Volker Weispfenning (in cooperation with Heinz Kredel) *Grobner Bases: A Commutative Approach to Commutative Algebra.* New York: Springer-Verlag, 1993.

[4] Courtois, Nicolas T. *Is AES Secure?* 20 November 2006 <http://www.cryptosystem.net/aes/>.

[5] Cox, David, John Little, and Daniel O'Shea. *Ideals, Varieties, and Algorithms: An Introduction to Computational and Commutative Algebra.* 2nd ed. New York: Springer, 1998.

[6] Dummit, D.S. and Foote, R.M. *Abstract Algebra*, Upper Saddle River: Prentice-Hall, 1999

[7] Faugere, Jean-Charles. *A new efficient algorithm for computing Grobner bases without reduction to zero (F5).* In ISSAC. Villeneuve d' Ascq, France. July 2002.

[8] Faugere, Jean-Charles. *A new efficient algorithm for computing Grobner bases.* Journal of Pure and Applied Algebra 139 (1999): 61-88.

[9] Garey, Michael R. and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* New York: W.H. Freeman and Company, 1990.

[10] Koblitz, Neal. *Algebraic Aspects of Cryptography.* Berlin: Springer, 1999.

[11] Koh, Jee. *Ideals Generated by Quadrics Exhibiting Double Exponential Degrees.* Journal of Algebra 200 (1998): 225-245.

[12] Kurt, Yesem. *New Key Exchange Primitives.* Diss. Indiana U, 2005.

[13] Wieb Bosma, John Cannon, and Catherine Playoust. The Magma algebra system. I. The user language. J. Symbolic Comput., 24(3-4):235-265, 1997.

[14] Mayr, E. and A. Meyer. *The complexity of the word problems for commutative semigroups and polynomials ideals.* Adv. Math 46 (1982): 305-329.

[15] Stegers, Till. *Faugere's F5 Algorithm Revisited.* Diss. TU Darmstadt, 2005.

[16] Trappe, Wade and Lawrence C. Washington. *Introduction to Cryptography with Coding Theory.* Upper Saddle River: Prentice Hall, 2002.

Appendices

Appendix A: Exposition on Error in Original Proof of F5 Termination

In this section we will spend some time looking at the errors in previous arguments of F5's termination for regular input sequences. It is of great importance to do this; since the inception of the argument by Faugere in [7], it has been cited in numerous places, including [2]. In [15], Stegers acknowledges difficulty in proving F5's termination for regular input sequences. Thus he leaves the claim as a conjecture.

For expository clarity, we will begin this section by paraphrasing (almost quoting) the statement and partial proof of Conjecture 3.31 from pages 38 and 39 of [15]. Though the same ideas can be found in [7], [15] gives a lengthier exposition on the topic. Once we have a handle on the reasoning involved, we will be able to show a specific counterexample that challenges the reasoning of the proof of termination originally offered by Faugere in [7].

Conjecture. *Let $F = (f_1, f_2, \ldots, f_m)$ be a sequence of polynomials such that, if F is given as input to F5, there are no reductions to* 0 *in F5. Then on input F, F5 terminates.*

This conjecture, if true, has an immediate corollary. We know by Theorem 3.5 that there are no reductions to 0 during a run of F5 if the input sequence to F5 is regular. Thus, if the above conjecture is true, F5 would always terminate for regular input sequences.

We will begin our exposition with Stegers' attempt at proof.

Proof. **Partial Proof of Conjecture.** Let $\hat{R}$ be the result of a call of the function *Reduction* and $\hat{G}_i$ the intermediate Grobner basis prior to including the elements of $\hat{R}$. It suffices to show that if $R \neq \emptyset$, then $\langle HT(\hat{G}_i)\rangle \neq \langle HT(\hat{G}_i \cup \hat{R})\rangle$, for then, since $K[\overline{x}]$ is Noetherian, eventually $\hat{R}$ will be the empty set and remain empty.

Suppose $\hat{R}$ is nonempty. Without loss of generality we may furthermore assume that $\hat{R}$ is finite and $i = 1$. Choose $r_k \in \hat{R}$ such that $\mathcal{S}(r_k)$ is $\prec$-maximal. We show that r_k is not top-reducible by any other element of $\hat{G_1} \cup \hat{R}$ and thus in particular $\langle HT(\hat{G_i})\rangle \neq \langle HT(\hat{G_1} \cup \hat{R})\rangle$.

Suppose by way of contradiction that there exists an $r_\ell \in (\hat{G_1} \cup \hat{R})\backslash\{r_k\}$ such that $HT(r_\ell)|HT(r_k)$, say $uHT(r_\ell) = HT(r_k)$ with $u \in T$. Clearly $poly(r_\ell) \notin \langle \hat{G_2}\rangle$, otherwise r_k could have been reduced by G_2 during *Reduction*.

We distinguish four cases.

(1) ur_ℓ is normalized. Note that $u\mathcal{S}(r_\ell) \neq \mathcal{S}(r_k)$, otherwise r_k would have been discarded due to rewriting rules. [Author's note: This statement is given (without proof) in [15] and implied (again, without proof) by [7]. This author has been unable to verify the truth of the statement. However, it is not of major concern for us here. For more exposition on the topic, turn to section 3.5.]

   (a) $\mathcal{S}(r_k) \prec \mathcal{S}(ur_\ell)$. Note the critical pair $(HT(r_k), u, r_\ell, 1, r_k)$ was introduced into the list. Indeed, (u, r_ℓ) is normalized since $u\mathcal{S}(r_\ell)$ is not top-reducible by G_2; clearly, $(1, r_k)$ is normalized as well. Together with the assumption that $\mathcal{S}(r_k) \prec \mathcal{S}(ur_\ell)$, this implies that the critical pair is normalized, hence it was created by *CritPair*. The corresponding S-polynomial produced by the procedure *SPol* is $(u\mathcal{S}(r_\ell), ur_\ell - r_k)$. As the signature of a labeled polynomial is not changed during the reduction and by assumption no reduction to 0 occurs, $\hat{R}$ contains some polynomial with signature $\mathcal{S}(ur_\ell)$. But $\mathcal{S}(r_k) \prec \mathcal{S}(ur_\ell)$, contradicting the assumption that $\mathcal{S}(r_k)$ is $\prec$-maximal.

   (b) $u\mathcal{S}(r_\ell) \prec \mathcal{S}(r_k)$. Since $\mathcal{S}(r_k)$ is maximal, r_k was the last labeled polynomial to be included in $\hat{R}$. In particular, when *TopReduction* is passed

r_k, the labeled polynomial r_ℓ is already in the Grobner basis. But then $u\mathcal{S}(r_\ell) \prec \mathcal{S}(r_k)$ implies r_k can be reduced by r_ℓ, a contradiction.

(2) ur_ℓ is not normalized. Since r_ℓ is admissible, there exists $s' \in K[\overline{x}]^m$ such that $s'_1 \neq 0$, $v(s') = poly(r_\ell)$ and $HT(s'_1)F_1 = \mathcal{S}(r_\ell)$. Let

$$us'_1 = v' + \sum_{j=2}^{m} \lambda_j f_j$$

where $v', \lambda_2, \ldots, \lambda_m \in K[\overline{x}]$, v' not top-reducible by G_2. Since $uHT(s'_1)$ is reducible by G_2, either $v' = 0$ or $HT(v) < HT(us'_1)$. If $v' = 0$, the $upoly(r_\ell) \in \langle G_2 \rangle$, so $poly(r_k)$ would have been reduced by G_2. Therefore, $v' \neq 0$. We have

(A-1) $$upoly(r_\ell) = us'_1 f_1 + \sum_{j=2}^{m} us'_j f_j = v' f_1 + \sum_{j=2}^{m} (\lambda_j f_1 + us'_j) f_j$$

Now we define a set $\hat{T} = \{t \in T(v') | HT(tf_1) > HT(r_k)\}$. Since v' is in normal form with respect to $\langle f_2, f_3, \ldots, f_m \rangle$, the signed polynomial tr_1 is normalized for every $t \in \hat{T}$.

(a) $\hat{T} = \emptyset$. In this case, there is a representation

$$\sum_{j=2}^{m} (\lambda_j f_1 + us'_j) f_j = \sum_{g \in G_2} \mu_g g$$

where μ_g are polynomials such that $\mu_g = 0$ or $HT(\mu_g g) < HT(r_k)$. As the head term of $upoly(r_\ell)$ is $HT(r_k)$, it must occur on the right-hand-side of equation A-1 somewhere. We conclude that $HT(v'f_1) = HT(r_k)$, in which case either r_k would have been reduced by the normalized signed polynomial $HT(v')r_1$, or some $g \in G_2$ satisfies $HT(\mu_g g) = HT(r_k)$, in which case r_k would have been reduced by g.

(b) $\hat{T} \neq \emptyset$. [It is this case that makes the above proposition a conjecture.] For all $t \in \hat{T}$, the signed polynomial tr_1 is normalized. All these signed polynomials were included in some critical pair, and therefore there exists a signed polynomial $r_j \in G_1$ such that $\mathcal{S}(r_j) = HT(v')F_1$ and $HT(r_j) = HT(r_k)$. If so, then r_k would have been reduced by r_j, a contradiction.

Thus, by the reasoning given at the beginning of this proof, we can conclude that if there is no reduction to 0 within F5, F5 will terminate. This ends the partial proof. □

This proof seems to be well-constructed. There is an error, however. Before we delve into where the error lies, let's first observe that **there must be an error**. We can do this by producing a counterexample to the statement of the proposition.

Now we will revisit the example from subsection 3.4.2. In that example, we added polynomial $r_9 = (x^2zF_1, y^5t^2 - x^4zt^2)$ to the Grobner basis at degree 7. Then at degree 8 we added polynomial $r_{10} = (z^3tF_1, y^6t^2 - x^2z^3t^3)$. Notice that $HT(r_9)|HT(r_{10})$. We also consider polynomial $r_{11} = (x^3zF_1, x^5zt^2 - y^2z^4t^2)$; note that $r_8 = (x^3F_1, x^5t^2 - y^2z^3t^2)$, so that $HT(r_8)|HT(r_{11})$. In the language of the above partial proof, this means that in degree 8 we have $\hat{R} \neq \emptyset$ and yet $\langle HT(\hat{G_1})\rangle = \langle HT(\hat{G_1} \cup \hat{R})\rangle$. This is the exact opposite of what we proved! According to our proof, if there is no reduction to 0 (which there isn't during our example run) then every new subsequent nonempty output $\hat{R}$ of *Reduction* should add something new to the monomial ideal of head terms associated with the Grobner basis – we should get new elements of $\langle HT(\hat{G_1})\rangle$. This is, in fact, demonstrably not the case in our above example. Moreover, the two polynomials r_{10} and r_{11} reduce to 0 by the polynomials previously add to the Grobner basis!

(As an aside, we may recall that the example given in this paper deviates slightly from the example given in [7]. This deviation has no effect on the above argument – Faugere's version of the example in [7] also yields the exact same counterexample.

In fact since the example found here and the example found in [7] represent the only two legal orderings of the operations of F5 under the input from subsection 3.4.2, we can feel assured that we have an input sequence that acts as a counterexample regardless of the ordering of the polynomials.)

So we know that the above proposition is incorrect. The question is: where does the argument fail? In order to answer this, let's use the notation from the proof and look again at the example from subsection 3.4.2. We will let $r_k = r_{10}$, $r_\ell = r_9$ and $u = y$. Then we will definitely be in case 2 since $ur_\ell = yr_9$ is not normalized. Then we have $s' = (x^2z, -xyz^2 - y^3t, 0)$ (note that $v(s') = poly(r_\ell)$); thus, we have $s'_1 = x^2z$. Just as in the proof, we will put us'_1 in a normal form representation. Thus we will have $us'_1 = yx^2z = z^3t + zpoly(r_3)$. Thus we will have $v' = z^3t$ (note that $v' \neq 0$).

The only candidate for $\hat{T}$ is z^3t. Since $HT(z^3tr_1) = yz^6t > HT(r_{10})$, we see that indeed $\hat{T} = \{z^3t\}$. Thus we are in subcase b ($\hat{T} \neq \emptyset$). So according to our "proof", there exists a signed polynomial $r_j \in G_1$ such that $\mathcal{S}(r_j) = HT(v')F_1$ and $HT(r_j) = HT(r_k)$. Now we look through all of the polynomials that have been added to G_1 so far. There is only one candidate for r_j... r_{10} itself!

And now the problem with the partial proof is clear. There is nothing incorrect in the proof at all until the very last line of subcase $2b$ – the proposed r_j can be r_k itself. This case does not lead to contradiction because F5 would never try to reduce a candidate polynomial by itself during *Reduction*.

Thus we can conclude that the reasoning used in [7] is not quite correct.

It is important to note that we have not shown that the claim made by Faugere in [7] is incorrect; rather, we have explored why the proof given in [7] (and partial proof in [15]) is unsatisfactory. It is still plausible that F5 terminates for regular input sequences, and given that the result has not been challenged in approximately six years (at the time of this writing), the empirical acceptance certainly bodes well

for the claim. However, a different line of reasoning must be explored. (This author has been unable to show that F5 terminates for all regular input sequences.)

Appendix B: MAGMA Code for F5t

This is the code for F5 written in MAGMA [13]. The homepage for the MAGMA computer algebra system can be found at:

http://magma.maths.usyd.edu.au/magma/

The code presented below is an extension of the code written by Till Stegers in [15]. The entire code for the program is not given here; rather, only the significantly altered routines are given. The code presented here has been altered in its spacing (not in content) to accommodate print.

```
/*
Updated version written by: Justin M. Gash
(with permission from Till Stegers)
This version is called F5t
Created from Fall 2006 through Spring 2008 as part of the doctoral thesis:

"On Efficient Grobner Basis Computation"

at Indiana University Bloomington.

All of the following header commentary was written by Till Stegers and
still applies.
*/

/*
    For a procedure name, the identifier in brackets
    refers to the names J.-C. Faugere chose in his paper:

        "A new efficient algorithm for computing Gr"obner bases
        without reduction to zero ($F_5$)", ISSAC '02: Proceedings
        from the International Symposium on Symbolic and Algebraic
        Computation, ACM Press, 2002.
```

```
        Version 1.2 of the paper available at
        http://www-calfor.lip6.fr/~jcf/Papers/@papers/f5.pdf


During the algorithm, polynomials are augmented using so-called
signatures. These "labeled polynomials", or "rules", as Faug‘ere calls
them, are implemented as tuples in Magma as follows: a labeled
polynomial f is a tuple  < t, i, p >, where
    t is a term,
    i the index of the i-th canonical basisvector F_i of the module P^m,
    p the polynomal,
    such that the signature of the labeled polynomial is S(p) = t*e_i.

Most of the time, these labeled polynomials will be elements of a unique
global array r, and be referenced by their indices in r, or r-indices for
short.

Critical pairs are stored as tuples <l, u, j, v, k>, where
   l is the LCM of the critical pair
   u,v are terms, and
   j,k are (distinct) r-indices of labeled polynomials.

The variable R contains a sequence sequences of simplification rules,
similar to the array Rule in Faug‘ere’s paper. A simplification rule
in R[i] is a tupel <t,k>, where t is a term and k is an r-index of a
labeled polynomial r_k that has the signature t*e_i.

*/


import "critpairs.mag": NUMBER_OF_CRITERIA;
```

```
/*-----------------------------------------------+
|                                               |
|                                               |
|                  Main Algorithm               |
|                                               |
|                                               |
|                                               |
+-----------------------------------------------*/



/*************************************************************

    AlgorithmF5

    returns a (not necessarily reduced) Groebner basis
    for the ideal generated by {f_i} union G_iplus1.

    Eventually, G_iplus1 is set to the resulting basis.


    Input:

        m_i_fi = <m,i,f_i>

        m           total number of input polynomials

        i           # of current global iteration

        f_i         polynomial to be added in this
                    iteration
```

```
        G_iplus1    sequence of r-indices of the labeled
                    polynomials forming the (non-reduced)
                    Gr"obner basis of the ideal generated
                    by f_{i+1}, ..., f_m

        G_iplus1Reduced: the unique reduced Gr"obner basis
                    of the ideal  generated by
                    f_{i+1}, ..., f_m

        rules       sequence of simplificationr rules

        statsPerCall: receives statistics generated during
                    this call, conforms with stats.mag



************************************************************/
intrinsic AlgorithmF5(~m_i_fi::Tup,
    ~r::[],
    ~G_iplus1::[],
    ~G_iplus1Reduced::[],
    ~rules::[],
    ~statsPerCall::[])
{Compute a Groebner basis of the ideal <G_iplus1 cat [f_i]>.}

    m   := m_i_fi[1];
    i   := m_i_fi[2];
    f_i := m_i_fi[3];
    hasbeeninD := m_i_fi[4];
    Mbound:= m_i_fi[5];
```

```
pol_ring := Parent(f_i);
pair_univ := PairUniverse(pol_ring);
rule_univ := RuleUniverse(pol_ring);

DD := [];
addedtoDD := [];
holder :=[pair_univ|   ]; //will hold ordered S-polynomials

// Statistics per call of F5
// (data structure discussed in IncrementalF5)

// Statistics per degree
statsPerDeg := Stats_Init(m);

r[i]:= <1, i, f_i / LeadingCoefficient(f_i)>;
G_i := Append(G_iplus1,i);

Stats_PolCreate(~statsPerCall,[i],~r);

newPolIndices := [i];

// Given G_{i+1}, compute the unique Groebner basis of the
// ideal generated by G_{i+1}.
procedure ReduceIntermediateBasis(~reduced,~r,~newPols)

    // Inefficient variant as in the paper (no reduction):
    //reduced cat:= [pol_ring| r[l][3]: l in newPols];

    // Efficient variant (using Magma's Reduce()):
    reduced := ReduceGroebnerBasis(reduced cat [pol_ring| r[l][3]:
               l in newPols]);
```

```
        // Honest variant (using interpreted reduction code, not Magma's):
        //reduced := ReduceGroebnerNaive(reduced cat [pol_ring| r[l][3]:
                     l in newPols]);
    end procedure;


    // Collect head terms of G_{i+1}, ..., G_m
    // to speed up reduction (phi) and reducibility test (psi)
    heads := [ [car<Integers(),pol_ring,pol_ring>|
                <l,LM(G_iplus1Reduced[comp][l]),
                LC(G_iplus1Reduced[comp][l])>:
                l in [1..#G_iplus1Reduced[comp]]]: comp in [1..m]];

    // Hand-written top reduction of a
    // labeled polynomial x w.r.t. G_{i+1}
    function phi(x)
        if x[3] ne Parent(x[3])!0 then
            LMx := LM(x[3]);
            LCx := LC(x[3]);
        end if;

        while (x[3] ne 0) and exists(g){h: h in heads[i+1] |
                                        IsDivisibleBy(LMx,h[2])} do
            y := x[3];
            x[3] -:= LCx/g[3]*(LMx div g[2])*G_iplus1Reduced[i+1][g[1]];

            if x[3] ne Parent(x[3])!0 then
                LMx := LM(x[3]);
                LCx := LC(x[3]);
            end if;

            if GetAssertions() and not AssertReducedTo(y,x[3]) then
```

```
            print "x:",y;
            print "g:",g;
            print "x':",x[3];
            error  "not reduced! stopping.";
        end if;
    end while;

    return x;

    // very expensive test:
    assert x[3] eq NormalForm(x[3],G_iplus1Reduced[i+1]);
end function;



// For a labeled polynomial x = <t, k, p>, psi(x) is true iff the
// term t is top-reducible mod G_{k+1} and k<m, false otherwise
// (i.e. if k=m). In other words, psi(x) is true iff x is
// not normalized.
function psi(x)
    if x[2] eq m then
        return false;
    end if;

    return exists{h: h in heads[x[2]+1] | IsDivisibleBy(x[1],h[2])};
end function;

P := [ pair_univ | ];

for j in G_iplus1 do
    result := <-42>;
    CritPair(i,j,~r,i,~psi,~result);
```

```
        pair, is_pair,critUsed := Explode(result);

        assert TestCritPair(pair,is_pair,r,pol_ring);

        if is_pair then // check if the pair wasn't redundant
            Append(~P,pair);
            if GetAssertions() and not AssertPairNormalized
              (pair,r,m,pol_ring) then
                print "";
                print "critUsed:",critUsed;
                printf "Accepted non-normalized pair %o at spot 1\n", pair;
                PrintPairInfo(pair,~r,~G_iplus1Reduced,~heads,~psi);
                assert false;
           end if;
           Stats_PairCreate(~statsPerCall,pair);
        else
            if GetAssertions() and AssertPairNormalized
              (pair,r,m,pol_ring) then
                printf "Missed normalized pair %o at spot 1\n", pair;
                PrintPairInfo(pair,~r,~G_iplus1Reduced,~heads,~psi);
                assert false;
            end if;
            Stats_PairDiscard(~statsPerCall,critUsed);
        end if;
    end for;

    Sort(~P, ~HasLowerDegreeCritPair);

    while ((not IsEmpty(P)) or (not IsEmpty(holder))) do

        statsPerDeg := Stats_Init(m);
```

```
d := DegreeCritPair(P[1]); // smallest occurring degree

D := DegreeCritPair(P[#P]);
P_d := [ pair_univ | pair: pair in P | DegreeCritPair(pair) eq d ];

ChangeUniverse(~P_d,pair_univ);

printf "\n----- Selecting new P_d...\n";
printf "Size of G_%o so far: %o\n", i, #G_i;
printf "Size of G_%o:        %o\n", i+1, #G_iplus1;
printf "Size of G_%o reduced:%o\n", i+1, #G_iplus1Reduced[i+1];
printf "Number of remaining critical pairs: %o\n",#P;
printf "Minimal degree: %3o\n",d;
printf "Maximal degree: %3o\n",D;
printf "Selecting %o pair(s) of degree d=%o...\n\n",#P_d,d;

P := P[#P_d+1..#P];
assert #SequenceToSet(P) eq #P;

//printf "P after removing P_%o: %o\n", d, P;
//printf "P_%o: %o\n",d,P_d;


print "Computing S-polynomials...";

R_d := [];


spols_redPairs := [* *];

iterandhasbeeninD := <i, hasbeeninD>;
//a tuple for argument passing
```

```
Spols(~G_i, ~P_d, iterandhasbeeninD, ~r, ~rules,~spols_redPairs);

spols := spols_redPairs[1];
redundantPairs := spols_redPairs[2];



printf "Number of S-polynomials before reduction: %o\n",#spols;
Stats_PolCreate(~statsPerDeg,spols,~r);
assert #P_d ge #spols;

Stats_PolDiscard(~statsPerDeg,#P_d-#spols);
putinD := 0;
phi_psi_stats := <phi, psi,statsPerDeg, putinD, DD,Mbound>;
//a tuple for argument passing

//ReductionF5(~spols, G_i, i, ~r, ~phi_psi_stats, ~rules);
//SpecialGiplus1 := G_iplus1Reduced[i+1];
ReductionF5(~spols, newPolIndices, i, ~r, ~phi_psi_stats,~rules);
putinD := phi_psi_stats[4];
//determines whether output from ReductionF5 is put in DD or not

// Update statsPerDeg from the tuple
statsPerDeg := phi_psi_stats[3];

printf "Number of S-polynomials after reduction: %o\n\n", #spols;

print "#R_d:",#R_d;

if (putinD eq 1) then //spols should be put in DD

  //create Spols of spols and DD
```

```
reducer := G_iplus1 cat DD;
//this is the reducing set for the upcoming section of code

reducer cat:= newPolIndices;
reducer cat:= spols;
reducerset:=[k:k in reducer];
//indices of reducer set, used in upcoming section

for k in DD do
  for j in spols do
    STerm := LCMCritPair(r[k],r[j]);
    u_1 := pol_ring!STerm div pol_ring!LM(r[k][3]);
    u_2 := pol_ring!STerm div pol_ring!LM(r[j][3]);
    pair := <STerm, u_1, k, u_2, j>;
    Append(~holder, pair);
  end for;
end for;

//create Spols of spols and newPolIndices

for k in newPolIndices do
  for j in spols do
    STerm := LCMCritPair(r[k],r[j]);
    u_1 := pol_ring!STerm div pol_ring!LM(r[k][3]);
    u_2 := pol_ring!STerm div pol_ring!LM(r[j][3]);
    pair := <STerm, u_1, k, u_2, j>;
    Append(~holder, pair);
  end for;
end for;

//create Spols of spols and G_iplus1
```

```
for k in G_iplus1 do
  for j in spols do
    STerm := LCMCritPair(r[k],r[j]);
    u_1 := pol_ring!STerm div pol_ring!LM(r[k][3]);
    u_2 := pol_ring!STerm div pol_ring!LM(r[j][3]);
    pair := <STerm, u_1, k, u_2, j>;
    Append(~holder, pair);
  end for;
end for;



//sort holder by degree

Sort(~holder, ~HasLowerDegreeCritPair);
//sort S-polynomials by degree

//attach spols to appropriate lists

DD := DD cat spols; //place the output of Reduction in DD
hasbeeninD cat:= spols;
//place the output of Reduction in hasbeeninD

//reduction for those elements of holder of degree d
//if any polynomial is added to r, it's S-polynomials
//are also added to holder

holderempty:=IsEmpty(holder);
while ((not holderempty)and(DegreeCritPair(holder[1]) eq d)) do
    N := #r + 1;
    pair := holder[1];
    STerm, u_1, k, u_2, j := Explode(pair);
```

```
holder := Reverse(holder);
Prune(~holder);
holder := Reverse(holder);
if SignatureLess(<u_1*r[k][1],r[k][2]>,
  <u_2*r[j][1],r[j][2]>) lt 0 then
  r[N] := rule_univ!<u_2*r[j][1], r[j][2], u_1*r[k][3] -
          (LC(u_1*r[k][3])/ LC(u_2*r[j][3]))*u_2*r[j][3]>;
else
  r[N] := rule_univ!<u_1*r[k][1], r[k][2], u_1*r[k][3] -
          (LC(u_1*r[k][3])/ LC(u_2*r[j][3]))*u_2*r[j][3]>;
end if;
F5tSpecialReduction(~N, ~reducerset, ~r, ~phi_psi_stats);
if (r[N][3] ne 0) then
  DD cat:= [N];
  hasbeeninD cat:= [N];
  print "F5t has added an extra polynomial to D";
  reducerset cat:= [N];
  for k in DD do
    STerm := LCMCritPair(r[k],r[N]);
    u_1 := pol_ring!STerm div pol_ring!LM(r[k][3]);
    u_2 := pol_ring!STerm div pol_ring!LM(r[N][3]);
    pair := <STerm, u_1, k, u_2, N>;
    Append(~holder, pair);
  end for;
  for k in G_iplus1 do
    STerm := LCMCritPair(r[k],r[N]);
    u_1 := pol_ring!STerm div pol_ring!LM(r[k][3]);
    u_2 := pol_ring!STerm div pol_ring!LM(r[N][3]);
    pair := <STerm, u_1, k, u_2, N>;
    Append(~holder, pair);
  end for;
  for k in newPolIndices do
```

```
            STerm := LCMCritPair(r[k],r[N]);
            u_1 := pol_ring!STerm div pol_ring!LM(r[k][3]);
            u_2 := pol_ring!STerm div pol_ring!LM(r[N][3]);
            pair := <STerm, u_1, k, u_2, N>;
            Append(~holder, pair);
          end for;
          Sort(~holder, ~HasLowerDegreeCritPair);
          //sort S-polynomials by degree

        else
          print "Attempt to add an extra polynomial failed.
                 It reduced to 0.";
          Prune(~r);
        end if;
        holderempty:=IsEmpty(holder);
    end while;

  else //spols will be added to newPolIndices

    //create Spols of spols and DD

    for k in DD do
      for j in spols do
        STerm := LCMCritPair(r[k],r[j]);
        u_1 := pol_ring!STerm div pol_ring!LM(r[k][3]);
        u_2 := pol_ring!STerm div pol_ring!LM(r[j][3]);
        pair := <STerm, u_1, k, u_2, j>;
        Append(~holder, pair);
      end for;
    end for;

    //sort holder by degree
```

```
Sort(~holder, ~HasLowerDegreeCritPair);
//sort S-polynomials by degree

//normal F5 stuff

R_d := R_d cat spols;

printf "Adding critical pairs
        for reduced polynomials...\n",#R_d;

nNewPairs := 0;

for k in R_d do
    new_pairs := [pair_univ | ];

    for l in G_i do
        result := <-42>; //initialize variable receiving result
        CritPair(k, l, ~r, i, ~psi, ~result);
        pair, is_pair,critUsed := Explode(result);

        assert TestCritPair(pair,is_pair,r,pol_ring);

        if is_pair then
            Include(~new_pairs,pair_univ!pair);
            Stats_PairCreate(~statsPerDeg,pair);
            if GetAssertions() and not AssertPairNormalized
              (pair,r,m,pol_ring) then
                printf "Accepted non-normalized
                        pair %o\n", pair;
                PrintPairInfo(pair,~r,~G_iplus1Reduced,
                              ~heads,~psi);
```

```
                    assert false;
                end if;
            else
                Stats_PairDiscard(~statsPerDeg,critUsed);

                if GetAssertions() and AssertPairNormalized
                  (pair,r,m,pol_ring) then
                    printf "Missed normalized pair %o\n", pair;
                    PrintPairInfo(pair,~r,~G_iplus1Reduced,
                                  ~heads,~psi);
                    assert false;
                end if;
            end if;
        end for;

        nNewPairs +:= #new_pairs;
        P cat:= new_pairs;

        // Add rule k to intermediate basis:
        Append(~G_i, k);
        Append(~newPolIndices,k);

    end for;

    printf "Number of new critical pairs:      %6o\n", nNewPairs;

    reducer := G_iplus1 cat DD;
    //this is the reducing set for the upcoming section of code

    reducer cat:= newPolIndices;
    reducer cat:= spols;
    reducerset:=[k:k in reducer];
```

```
//indices of reducer set, used in upcoming section


holderempty:=IsEmpty(holder);
while ((not holderempty)and(DegreeCritPair(holder[1]) eq d)) do
    N := #r + 1;
    pair := holder[1];
    STerm, u_1, k, u_2, j := Explode(pair);
    holder := Reverse(holder);
    Prune(~holder);
    holder := Reverse(holder);
    if SignatureLess(<u_1*r[k][1],r[k][2]>,
      <u_2*r[j][1],r[j][2]>) lt 0 then
      r[N] := rule_univ!<u_2*r[j][1], r[j][2], u_1*r[k][3] -
              (LC(u_1*r[k][3])/ LC(u_2*r[j][3]))*u_2*r[j][3]>;
    else
      r[N] := rule_univ!<u_1*r[k][1], r[k][2], u_1*r[k][3] -
              (LC(u_1*r[k][3])/ LC(u_2*r[j][3]))*u_2*r[j][3]>;
    end if;
    F5tSpecialReduction(~N, ~reducerset, ~r, ~phi_psi_stats);
    if (r[N][3] ne 0) then
      DD cat:= [N];
      hasbeeninD cat:= [N];
      print "F5t has added an extra polynomial to D";
      reducerset cat:= [N];
      for k in DD do
        STerm := LCMCritPair(r[k],r[N]);
        u_1 := pol_ring!STerm div pol_ring!LM(r[k][3]);
        u_2 := pol_ring!STerm div pol_ring!LM(r[N][3]);
        pair := <STerm, u_1, k, u_2, N>;
        Append(~holder, pair);
      end for;
      for k in G_iplus1 do
```

```
            STerm := LCMCritPair(r[k],r[N]);
            u_1 := pol_ring!STerm div pol_ring!LM(r[k][3]);
            u_2 := pol_ring!STerm div pol_ring!LM(r[N][3]);
            pair := <STerm, u_1, k, u_2, N>;
            Append(~holder, pair);
          end for;
          for k in newPolIndices do
            STerm := LCMCritPair(r[k],r[N]);
            u_1 := pol_ring!STerm div pol_ring!LM(r[k][3]);
            u_2 := pol_ring!STerm div pol_ring!LM(r[N][3]);
            pair := <STerm, u_1, k, u_2, N>;
            Append(~holder, pair);
          end for;
          Sort(~holder, ~HasLowerDegreeCritPair);
          //sort S-polynomials by degree

        else
          print "Attempt to add an extra polynomial failed.
                 It reduced to 0.";
          Prune(~r);
        end if;
        holderempty:=IsEmpty(holder);
    end while;

end if;


//printf "#G_i after adding s-polynomials: %o\n",#G_i;
//printf "G_i after adding s-polynomials: %o\n",G_i;
//print "r after one d-loop:",r;
print "Sorting critical pairs...";
Sort(~P, ~HasLowerDegreeCritPair);
```

```
        //printf "Statistics for degree %o:\n",d;
        //Stats_Print(statsPerDeg);
        Stats_Update(~statsPerCall,statsPerDeg);

    end while;

    stufftoadd:= newPolIndices cat DD;

    G_iplus1 cat:= newPolIndices;
    G_iplus1 cat:= DD;

    assert #SequenceToSet(G_iplus1) eq #G_iplus1;

    // initalize for reduction
    G_iplus1Reduced[i] := G_iplus1Reduced[i+1];

    //Rereduce intermediate basis

    ReduceIntermediateBasis(~G_iplus1Reduced[i],~r,~stufftoadd);

    print "Size of reduced basis:", #G_iplus1Reduced[i];
    m_i_fi := <m, i, f_i, hasbeeninD>;
    //print #r;
    //printf "\n\n\nStatistics for stage %o:\n\n",i;
    //Stats_Print(statsPerCall);

end intrinsic;
```

```
/***********************************************************

    Prereduce [no equivalent in Faug'ere's text]

    Mutual reduction of the input polynomials before
    starting F5. Corresponds to Magma's option ReduceInitial
    for the Buchberger algorithm.

    Input:

        F           sequence of polynomials

    Returns (in ~F):

        F           the reduced polynomials

***********************************************************/
intrinsic Prereduce(~F::[])
{Reduce the input before starting F5. Corresponds to Magma's option
 ReduceInitial for the Buchberger algorithm.}
    // Faster in most examples (exceptions: Cyclic n, Segers HFE)
   F := Reduce(F);
end intrinsic;




/***********************************************************

    IncrementalF5 [IncrementalF5]
```

```
    Input:

        F           sequence of nonzero homogeneous
                    polynomials (if F is not regular,
                    termination is not guaranteed)

        ensure_gb   If true, IncrementalF5 will check
                    using Magma's IsGroebner() that after
                    each step i, G_i is a Groebner basis.
                    Slows down the algorithm considerably.

    Returns:

        Gred        the unique reduced Groebner basis
                    of the ideal generated by F, identical
                    to the result of the call
                        GroebnerBasis(F);
                    in Magma

        G1          the non-reduced Groebner basis of F
                    as computed by F5. Consists of labeled
                    polynomials.

        is_gb       (for testing purposes) true if G1 is
                    a Groebner basis, false otherwise

        stats       a statistics object (see stats.mag)
                    summarizing this run of IncrementalF5


**********************************************************/
intrinsic IncrementalF5(F,ensure_gb) -> [],[],BoolElt,[]
```

```
{Outer routine of F5.}

    P := Parent(F[1]);
    rule_univ := RuleUniverse(P);

    Prereduce(~F);

    m := #F;

    // Initialize statistics
    stats := Stats_Init(m);

    rules := ResetRules(m,P);

    // Define the Macaulay matrix
    r := [rule_univ| ];

    r[m] := < 1, m, F[m] / LeadingCoefficient(F[m])>;

    Mbound:=TotalDegree(LT(F[m]));
    printf "=================== Stage %o/%o ================== \n",m,m;
    printf "Adding polynomial r[1].\n\n";

    G_iplus1 := [ m ];

    G_iplus1Reduced := [ [P| ]: i in [1..m+1] ];
    G_iplus1Reduced[m] := [ F[m] ];

    Stats_PolCreate(~stats,[m],~r);

    hasbeeninD := [];
    // will hold a list of all polynomials that are in DD
```

```
for i := m-1 to 1 by -1 do

    statsPerCall := Stats_Init(m);
    printf "================= Stage %o/%o ================= \n",i,m;

    //print "before iteration, has is: ", hasbeeninD;
    Mbound:=Mbound + TotalDegree(LT(F[i])) - 1;

    m_i_fi := <m,i,F[i], hasbeeninD,Mbound>;

      AlgorithmF5(~m_i_fi,~r,~G_iplus1,~G_iplus1Reduced,
                  ~rules,~statsPerCall);
      hasbeeninD := m_i_fi[4];
      G_rules := [r[l]: l in Sort(G_iplus1)];

      print "";



      print "";
      if ensure_gb then
          is_gb := IsGroebner([r[l][3]: l in G_iplus1]);

          if is_gb then
              printf "G_%o *is* a Groebner basis\n",i;
              printf "G_%o generates <f_%o, ..., f_m>: %o\n",i,i,
              Ideal([r[l][3]: l in G_iplus1]) eq
                    Ideal([F[l]: l in [i..m]]);
              printf "Size of basis is %o.\n",#G_iplus1;
              printf "Size of reduced basis is %o.\n",
                      #G_iplus1Reduced[i];
          else
```

```
                printf "G_%o is *not* a Groebner basis.
                        Size is %o.\n",i,#G_iplus1;

                if ensure_gb then
                printf "Aborted after step i = %o
                        and returning G_i\n",i;
                return G_iplus1Reduced[i],G_rules,is_gb;
                end if;
            end if;
            print "";
        else
            is_gb := true;
        end if;
        //printf "G_%o = %o\n\n",i,G_iplus1;

        Stats_Update(~stats,statsPerCall);
    end for;


    print "Statistics for all stages:";
    Stats_Print(stats);
    return G_iplus1Reduced[1],G_rules,is_gb,stats;

end intrinsic;



/*************************************************************

    F5opt

    Wrapper for the F5 algorithm, e.g. for use in a
```

```
    comparison with other algorithms.


    Input:

        F           sequence of nonzero homogeneous
                    polynomials (if F is not regular,
                    termination is not guaranteed)

        ensure_gb   If true, IncrementalF5 will check
                    using Magma's IsGroebner() that after
                    each step i, G_i is a Groebner basis.
                    Slows down the algorithm considerably.

    Returns:

        Gred        the unique reduced Groebner basis
                    of the ideal generated by F, identical
                    to the result of the call
                        GroebnerBasis(F);
                    in Magma

        stats       a statistics object (see stats.mag)
                    summarizing this run of IncrementalF5


        name        a string identifying the algorithm,
                    currently simply "F5"

**********************************************************/
intrinsic F5opt(F::[],ensure_gb::BoolElt) -> [],[],MonStgElt
{Faugere's F5 algorithm}
```

```
    t:= Cputime();
    placeholder:= 0;
    // will be used to pause the output so user can see the running time

    error if IsEmpty(F), "F must not be empty!";

    P := Parent(F[1]);

    homogenized := false;
    isHomogeneous := IsHomogeneousSystem(F);

    // Make sure we didn't forget to homogenize
    if not isHomogeneous[1]  then
        read answer,"System is not homogeneous!
                     Homogenize first y/n? [y]";
        if answer ne "n" then
            homogenized := true;
            F,P := HomogenizeExample(F,P);
        end if;
    end if;

    Gpols,Grules,b,stats := IncrementalF5(F,ensure_gb);
    if isHomogeneous[1] then
        print "Input system was homogeneous.";
    else
        printf "Input system was *not* homogeneous,";

        if homogenized then
            print " it was homogenized first.";
        else
            print " but it was not homogenized.
```

```
                    You're lucky it terminated at all!";
        end if;
    end if;

    if ensure_gb then
        same_ideal := Ideal(F) eq Ideal(Gpols);
        print "Generates same ideal:", same_ideal;
    else
        print "Did not check if G_1 is actually a Groebner basis.";
    end if;

    print "The time taken is: ", Cputime(t);

    readi placeholder; //artificial pause to see running time output

    return Gpols,stats, "F5";

end intrinsic;



/***********************************************************

    F5


    Wrapper for the F5 algorithm, useful for calls from
    the Magma console.


    Input:
```

```
        F           sequence of nonzero homogeneous
                    polynomials (if F is not regular,
                    termination is not guaranteed)

    Returns:

        Gred        the unique reduced Groebner basis
                    of the ideal generated by F, identical
                    to the result of the call
                        GroebnerBasis(F);
                    in Magma


************************************************************/
intrinsic F5(F::[]) -> [],[],MonStgElt
    {Faugere's F5 algorithm, not checking that G_i is a Groebner
     basis after each step i.}

    return F5opt(F,false);
end intrinsic;




/************************************************************

    F5ensure


    Wrapper for the F5 algorithm, useful for calls from
    the Magma console when debugging the algorithm.
```

```
    Input:

        F           sequence of nonzero homogeneous
                    polynomials (if F is not regular,
                    termination is not guaranteed)

    Returns:

        Gred        the unique reduced Groebner basis
                    of the ideal generated by F, identical
                    to the result of the call
                        GroebnerBasis(F);
                    in Magma


**********************************************************/
intrinsic F5ensure(F::[]) -> [],[],MonStgElt
    {Faugere's F5 algorithm, checking that G_i is a Groebner
     basis after each step i.}

    return F5opt(F,true);
end intrinsic;

/**********************************************************

Buchberger's Algorithm

This is a homemade, totally unoptimized version
of Buchberger's algorithm.

Input:
```

```
F           sequence of nonzero homogeneous
                  polynomials (if F is not regular,
                  termination is not guaranteed)

      ensureGB    a boolean element; if true, the output
                  is externally verified

Returns:

[Nothing]

************************************************************/

intrinsic BuchAlg(F::[], ensureGB::BoolElt)
{Buchberger's algorithm.}

    t:= Cputime();
    placeholder:= 0; // artificial user input variable used to pause
                     // the program's output to see running time
    G:=[];

    error if IsEmpty(F), "F must not be empty!";

    pol_ring:=Parent(F[1]);
    Z:= Integers();
    pair_univ:=car<pol_ring,Z,Z>;
    CP:=[pair_univ| ];

    // make all input polynomials monic

    for i:=1 to #F do
```

```
  F[i]:=(1/LeadingCoefficient(F[i]))*F[i];
end for;

// build original set of critical pairs

G:=F;
for i:=1 to #G-1 do
  for j:=i+1 to #G do
    lcm:=LCM(LM(G[i]),LM(G[j]));
    Append(~CP,<lcm,i,j>);
  end for;
end for;

Sort(~CP, ~BuchHLDCP);

//meat of Buchberger's algorithm

while (not IsEmpty(CP)) do

  //get first critical pair

  pair:=CP[1];
  CP:= Reverse(CP);
  Prune(~CP);
  CP:=Reverse(CP);

  //build S-polynomial

  u1:= pol_ring!pair[1] div pol_ring!LM(G[pair[3]]);
  u2:= pol_ring!pair[1] div pol_ring!LM(G[pair[2]]);
  h:=u1*G[pair[2]]-u2*G[pair[3]];
```

```
    //get S-polynomial in normal form

    h:= NormalForm(h,G);

    // if it's not 0, create more critical pairs and
    // add the new polynomial to the Grobner basis

    if (h ne 0) then
      h:=(1/LeadingCoefficient(h))*h;
      for i:=1 to #G do
        lcm:=LCM(LM(G[i]),LM(h));
        Append(~CP,<lcm,i,#G+1>);
      end for;
      Append(~G,h);
    end if;

    Sort(~CP, ~BuchHLDCP);

  end while;

  // if the user wanted the output verified, notify the user

  if (ensureGB eq true) then
    GBtest:=IsGroebner(G);
    if GBtest then
      print "A confirmed Grobner basis was computed!";
    else
      print "The output is not a Grobner basis!";
    end if;
    same_ideal := Ideal(F) eq Ideal(G);
    if same_ideal then
      print "The output generates the same ideal as the input!
```

```
                Woohoo!";
      else
        print "The output generates a different ideal as the input!
                Boohoo...";
      end if;
    end if;

    print "The time taken is: ", Cputime(t);

    readi placeholder;

    print G;

end intrinsic;

/*-----------------------------------------------+
|                                                |
|                                                |
|          Reduction of Polynomial Sequences     |
|                                                |
|                                                |
|                                                |
+-----------------------------------------------*/



/************************************************

    FindReductor [IsReducible]

    Input:
```

```
        k0_k:    a tuple <k0,k> of integers (see below)

        k0:     r-index of labeled polynomial to reduce

        k:      # of global iteration

        G:      sequence of r-indices of polynomials w.r.t.
                which r[k0] is to be reduced

        r:      "global" list of polynomials

        phi_psi_stats
<phi, psi,statsPerDeg, putinD, DD,Mbound>

  phi:        function that, given a labeled
                    polynomial <t,j,p>, returns
                    <t,j,NormalForm(p,G_{k+1})>

        psi:        function that returns true iff a
                    labeled polynomial <t,j,p> is
                    top-reducible w.r.t. G_{j+1}

        stats       a statistics object (see stats.mag)
                    that might be updated

        putinD
a boolean that indicates where the output from Reduction
        should be added to DD

  DD
an index list of polynomials that have been added to DD
```

```
  Mbound
a multiple of the Macaulay bound used in Reduction

        rules:  "global" list of simplification rules

        result: variable to receive return value


    Returns (in ~result):

        A tupel < red, is_top_red >.

        is_top_red: false if the polynomial of r[k0] cannot
                be top-reduced by the polynomials specified
                by G (modulo optimizations)

        red:    a reductor of r[k0] if r[k0] can be
                top-reduced (modulo optimizations)


**************************************************/
intrinsic FindReductor(k0_k::Tup, ~G::[],  ~r::[], ~phi_psi_stats::Tup,
                       ~rules::[], ~result)
{Find a reductor for a given labeled polynomial.}

    k0 := k0_k[1];
    k := k0_k[2];

    psi := phi_psi_stats[2];

    pol_ring := Parent(r[k0][3]);
```

```
rule_univ := RuleUniverse(pol_ring);

lt_0 := pol_ring!LeadingMonomial(r[k0][3]);
for j in G do

    lt_i := pol_ring!LeadingMonomial(r[j][3]);
    k_j := r[j][2];

    if not IsDivisibleBy(lt_0,lt_i) then
        // discard reductor by criterion (a)
        continue;
    else
        u := lt_0 div lt_i;
        ut := u * r[j][1];

        if (ut eq r[k0][1]) and (r[j][2] eq r[k0][2]) then
            // discard reductor by criterion (d)
            continue;
        end if;

        canBeRewritten := false;

        IsRewritable(u,j,k,~r,~rules,~canBeRewritten);

        if canBeRewritten then
            // discard reductor by criterion (c)
            continue;

        elif psi(<ut,k_j,1>) then
            // discard reductor by criterion (b)
            continue;
```

```
            else
                // conditions (a) through (d) are satisfied
               result := < j,true >;
               return;
            end if;
        end if;
    end for;

    // no reductor found or all discarded
    result := < 0,false >;

end intrinsic;

/*************************************************

    TopReduction [TopReduction]


    Performs a top-reduction using a normalized
    reductor on a given labeled polynomial,
    if possible. If no normalized reductor can be
    found, the polynomial is divided by its leading
    coefficient. If the reductor has a larger signature
    than the reductee, the reductum is added as a new
    labeled polynomial, and the list of simpification
    rules is updated.

    Input:

        k0_k:       tuple <k0,k>, see below

        k0:         the index of a polynomial in r
```

```
                that is to be top-reduced

        k:      # of global iteration

        G:      list of r-indices of labeled polynomials
                w.r.t. which to reduce r[k0]

        r:      list of labeled polynomials,
                might be updated

        phi_psi_stats
<phi, psi,statsPerDeg, putinD, DD,Mbound>

  phi:      function that, given a labeled
                polynomial <t,j,p>, returns
                <t,j,NormalForm(p,G_{k+1})>

        psi:    function that returns true iff a
                labeled polynomial <t,j,p> is
                top-reducible w.r.t. G_{j+1}

        stats   a statistics object (see stats.mag)
                that might be updated

        putinD
a boolean that indicates where the output from Reduction
        should be added to DD

  DD
an index list of polynomials that have been added to DD

  Mbound
```

```
a multiple of the Macaulay bound used in Reduction

        R:          simplification rules, might be
                    updated

        result:     variable to receive return value



   Result (in ~result):

        result = [* no_red_to_zero, h, red_list *]

        no_red_to_zero: false iff the r[k0] was reduced
                    to zero

        red_list:   Sequence of r-indices of polynomials
                    to be reduced, possibly empty;
                    unspecified if no_red_to_zero is false


        h:          r-index of any rules for which no normalized
                    reductor could be found, and hence may be
                    declared done by Reduction.
                    If no_red_to_zero is false or red_list is
                    empty, h is unspecified.


**************************************************/
intrinsic TopReduction(k0_k::Tup, ~G::[], ~r::[], ~phi_psi_stats::Tup,
                       ~R::[], ~result)
{Reduces a polynomial w.r.t. a list of polynomials.
```

```
Adds new rules as it sees fit.}

    k0 := k0_k[1];
    k  := k0_k[2];

    error if k0 gt #r, "TopReduction: Rule index k0 is invalid!";

    pol_ring := Parent(r[k0][3]);

    old := r[k0];



    if r[k0,3] eq Zero(pol_ring) then
        printf "Warning, r[%4o] is 0! Input is not a regular sequence.
                Stage: %o\n",k0,k;

        Stats_RedToZero(~phi_psi_stats[3],k);

        result := [* false, 0,[ ] *];
        return;
    end if;



    // k1 corresponds to the index of r' in Faugere's paper
    FindReductor(<k0,k>, ~G, ~r, ~phi_psi_stats, ~R, ~result);
    k1, top_reducible := Explode(result);

    lc_0 := LeadingCoefficient(r[k0][3]);

    if (not top_reducible) then
```

```
    // lc_0 is nonzero (see if-clause above)
    normalized_rule := <r[k0][1], r[k0][2], r[k0][3] div lc_0>;
    r[k0] := normalized_rule;


    result := [*  true, k0, [ ] *];
    return;

else

    lt_0 := LeadingMonomial(r[k0][3]);
    lt_1 := LeadingMonomial(r[k1][3]);

    lc_1 := LeadingCoefficient(r[k1][3]);

    u := pol_ring!lt_0 div pol_ring!lt_1;

    new_sig := <u * r[k1][1], r[k1][2]>;

    if SignatureLess(new_sig, <r[k0][1],r[k0][2]>) lt 0 then

        r[k0][3] := r[k0][3] - lc_0 / lc_1 * u * r[k1][3];

        assert AssertReducedTo(old[3],r[k0][3]);

        result := [*  true, 0, [k0] *];
        return;
    else

        N := #r + 1;

        r[N]:=<new_sig[1],new_sig[2],u*r[k1][3]-lc_1/lc_0*r[k0][3]>;
```

```
            if r[N][3] eq 0 then //DEBUG
                //print "TopRed: Unexpected reduction to zero occurred!";
                //TODO what's so unexp. here?
                Stats_RedToZero(~phi_psi_stats[3],k);
                result := [* false, 0, [] *];
                return;
            end if;



            AddRule(~R,~r,N,pol_ring);
            Stats_PolCreate(~phi_psi_stats[3],[N],~r);

            assert AssertReducedTo(r[k0][3],r[N][3]);

            result := [* true, 0, [N,k0] *];
            // (Segers says k0 is superfluous, but if left out F5 loops)

            return;

        end if;
    end if;

end intrinsic;

/***************************************************

F5tSpecialFindReductor

A helper subroutine for F5tSpecialReduction, this subroutine
is similar to FindReductor above except that no criteria
are used other than head term divisibility.
```

```
Input:

k
an r-index representing a signed polynomial to (attempt to) reduce

reducerset
a list of r-indices representing the polynomial candidates for
        reduction of r[k]

r
a list of signed polynomials

phi_psi_stats
<phi, psi,statsPerDeg, putinD, DD,Mbound>

phi:        function that, given a labeled
                    polynomial <t,j,p>, returns
                    <t,j,NormalForm(p,G_{k+1})>

      psi:        function that returns true iff a
                    labeled polynomial <t,j,p> is
                    top-reducible w.r.t. G_{j+1}

      stats       a statistics object (see stats.mag)
                    that might be updated

      putinD
a boolean that indicates where the output from Reduction
        should be added to DD

DD
```

```
an index list of polynomials that have been added to DD

Mbound
a multiple of the Macaulay bound used in Reduction

result
a 2-tuple holding the result; the first element holds the r-index of a
        reductor, if found; the second element is a boolean, where true
        indicates a reductor has been found

Returns: [Nothing]

*************************************************/

intrinsic F5tSpecialFindReductor(~k::RngIntElt, ~reducerset::[], ~r::[],
                                 ~phi_psi_stats::Tup, ~result)
{Special subroutine used by F5tSpecialReduction.}

  pol_ring := Parent(r[k][3]);
  lt_k := pol_ring!LeadingMonomial(r[k][3]);

  for j in reducerset do
    lt_j := pol_ring!LeadingMonomial(r[j][3]);
    if IsDivisibleBy(lt_k, lt_j) then
      result := <j, true>;
      return;
    end if;
  end for;

  result := <0, false>;

end intrinsic;
```

```
/***************************************************

F5tSpecialReduction

This is a helper routine for the portion of the F5 code that
reduces signed polynomials in DD.  It is similar to TopReduction.

Input:

k
an r-index representing a signed polynomial to (attempt to) reduce

reducerset
a list of r-indices representing the polynomial candidates
        for reduction of r[k]

r
a list of signed polynomials

      phi_psi_stats
<phi, psi,statsPerDeg, putinD, DD,Mbound>

phi:        function that, given a labeled
                    polynomial <t,j,p>, returns
                    <t,j,NormalForm(p,G_{k+1})>

      psi:        function that returns true iff a
                    labeled polynomial <t,j,p> is
                    top-reducible w.r.t. G_{j+1}

      stats       a statistics object (see stats.mag)
```

```
                    that might be updated

      putinD
a boolean that indicates where the output from Reduction
        should be added to DD

DD
an index list of polynomials that have been added to DD

Mbound
a multiple of the Macaulay bound used in Reduction

Returns: [Nothing]

************************************************/
intrinsic F5tSpecialReduction(~k::RngIntElt, ~reducerset::[],
                              ~r::[], ~phi_psi_stats::Tup)
{Special subroutine used to reduce signed polynomials in DD.}

  pol_ring := Parent(r[k][3]);
  result := <-42, -42>;
  done := 0;
  if (r[k][3] eq 0) then
    done := 1;
  end if;

  while (done ne 1) do
    r[k] := phi_psi_stats[1](r[k]);

    if (r[k][3] eq 0) then
      done := 1;
    else
```

```
      F5tSpecialFindReductor(~k,~reducerset,~r,~phi_psi_stats,~result);
      j, tr := Explode(result);

      if (tr eq false) then
        lc_k := LeadingCoefficient(r[k][3]);
        r[k][3] := r[k][3] div lc_k;
        done := 1;
      else
        lt_k := pol_ring!LeadingMonomial(r[k][3]);
        lt_j := pol_ring!LeadingMonomial(r[j][3]);
        lc_k := LeadingCoefficient(r[k][3]);
        lc_j := LeadingCoefficient(r[j][3]);
        u := pol_ring!lt_k div pol_ring!lt_j;
        newsig := <u*r[j][1], r[j][2]>;
        if SignatureLess(newsig, <r[k][1], r[k][2]>) lt 0 then
          r[k][3] := r[k][3] - lc_k/lc_j * u * r[j][3];
        else
          r[k]:= <newsig[1],newsig[2],u*r[j][3]-lc_j/lc_k *r[k][3]>;
        end if;
      end if;
    end if;
    if (r[k][3] eq 0) then
      done := 1;
    end if;

  end while;

end intrinsic;

/************************************************

    Reduction [Reduction]
```

```
    Input:

        todo:       sequence of t-indices of representing
                    polynomials to reduce

        G:          SeqEnum of indices of r representing
                    polynomials w.r.t. which to reduce

        r           set of all labeled polynomials known
                    (will possibly be updated)

        k           # of global iteration

        phi_psi_stats
<phi, psi,statsPerDeg, putinD, DD,Mbound>

  phi:        function that, given a labeled
                    polynomial <t,j,p>, returns
                    <t,j,NormalForm(p,G_{k+1})>

        psi:        function that returns true iff a
                    labeled polynomial <t,j,p> is
                    top-reducible w.r.t. G_{j+1}

        stats       a statistics object (see stats.mag)
                    that might be updated

        putinD
a boolean that indicates where the output from Reduction
```

```
        should be added to DD

  DD
an index list of polynomials that have been added to DD

  Mbound
a multiple of the Macaulay bound used in Reduction


        rules:      simplification rules, might be
                    updated


   Returns (in ~todo):

            sequence of r-indices of labeled polynomials
            to be included in the Groebner basis


**************************************************/
intrinsic ReductionF5(~todo::[], G::[], k::RngIntElt, ~r::[],
                      ~phi_psi_stats::Tup, ~rules::[])
{Reduction step in F5.}

        done := [];
        pol_ring := Parent(r[k][3]);

        // list of reduced S-polynomials known to be in NF w.r.t.
        // G_i+1, currently
        inNFwrtG_iplus1 := [Integers()| ];
        nPolsCreated := 0;
```

```
while (not IsEmpty(todo)) do
        //print "#todo is:", #todo;
        //TODO could be made more efficient.
        //Isn't this redundant?
        IndexRemoveDoubles(~todo,~r,pol_ring,~todo);
        cmpFunc := -42; //initialize variable
        IndexSignatureLess(~r,~cmpFunc);
        Sort(~todo,cmpFunc);

        size := #todo;



        h := todo[1];
        // r-index of polynomial with minimal signature

        Remove(~todo,1);

        // Moved here from TopRed to be able to avoid
        // calling phi more often than necessary
        pos := 0;
        BinSearch(~inNFwrtG_iplus1,h,~pos);
        //TODO #inNFwrtG_iplus1 <= 1, so this is unnecessary?!!

        if pos lt 1 then
                // r[h] was not reduced by phi yet or has
                // changed since then
                r[h] := phi_psi_stats[1](r[h]);
            else
                nPolsCreated +:= 1;
        end if;

        //DELTA_Pearce:one could pass todo as well, claims Pearce
```

```
        GcatDone := done cat G;

        result := [];


        TopReduction(<h,k>, ~GcatDone, ~r,~phi_psi_stats,
                     ~rules,~result);
        is_reg_seq, h1, todo1 := Explode(result);

        if #todo1 gt 1 then
            // h was not reduced, so we don't have to call phi(h)
            // the next time h is treated
            IncludeSorted(~inNFwrtG_iplus1,h);
        else
            ExcludeSorted(~inNFwrtG_iplus1,h);
        end if;


        if is_reg_seq then
            if IsEmpty(todo1) then
            // Faug'ere doesn't need this 'if' as he just
            // returns the empty set for h

                Append(~done,h1);
            else
                todo := todo cat todo1;
            end if;
        end if;

        size := #todo;
end while;
```

```
// the additional portion of F5t

ender := 0; // controls when this portion of code can be exited
temp := done;
reducerset1 := [pol_ring|r[i][3]: i in G];
reducerset2 := [pol_ring|r[i][3]:i in phi_psi_stats[5]];
reducerset:=reducerset1 cat reducerset2;
if (IsEmpty(done)) then
  ender := 1;
end if;
if ((not IsEmpty(done)) and (2*phi_psi_stats[6] gt
    TotalDegree(LT(r[done[1]][3])))) then
  ender := 1;
end if;
if (ender eq 0) then
  print "Entering special F5t-phase of reduction.";
end if;

while(ender eq 0) do

  // grab the first output element of Reduction

  b:=done[#done];
  Prune(~done);
  blah := r[b];
  keepgoing := 1;
  //controls when the reduction of a polynomial can be terminated

  while (keepgoing eq 1) do
    print "Still working in F5t-phase of reduction.";
```

```
  checker := blah[3];
  blah[3]:= NormalForm(blah[3],reducerset);
  blah := phi_psi_stats[1](blah);
  if (checker eq blah[3]) then
    keepgoing := 0;
  end if;
  if (blah[3] eq 0) then
    print "F5t has reduced a polynomial to 0!";
    keepgoing := 0;
  end if;
end while;

// if the normal form of the chosen polynomial is not 0,
// then halt this portion of code

if (blah[3] ne 0) then
  ender := 1;
  done:= temp;
end if;

// if all the output from Reduction reduced to 0,
// halt this portion and send the batch to DD

if (IsEmpty(done)) then
  ender := 1;
  phi_psi_stats[4] := 1;
  done := temp;

  // remove the rules associated with these polynomials

  for p in done do
    RemoveRule(~rules, p, ~r);
```

```
            end for;
          end if;
        end while;

        todo := done;


end intrinsic;

/*-----------------------------------------------+
|                                                |
|                                                |
|            Computation of Critical Pairs       |
|                  and S-Polynomials             |
|                                                |
|                                                |
+-----------------------------------------------*/



/*********************************************************


    Criteria used to detect non-normalized critical pairs


*********************************************************/


    // Both components have index <= i
    CRITERION_ONE   := 1; //
```

```
  // Larger component is not normalized
  CRITERION_TWO   := 2;

  // Smaller component is not normalized
  CRITERION_THREE := 3;


  // Number of criteria above, used for stats
  NUMBER_OF_CRITERIA := 3;



/***********************************************************

  Rewrite [Rewritten]


  Simplifies a given a pair (u,r[k]) using a list of
  simplification rules.

  Input:

        u:          term
        k:          r-index
        iter:       # of iteration in F5
        r:          global list of labeled polynomials
        rules:      global list of simplification rules
        result:     tupel to hold the result

  Returns (in ~result):
    A tupel of the form <u',k'>, where u is a term
    and k' the highest r-index with r[k'][2]=iter and
```

```
    r[k'][1] a divisor of u*r[k']. In particular
    k=k' if the given pair could not be simplified.

**********************************************************/

intrinsic Rewrite(u::RngMPolElt, k::RngIntElt, iter::RngIntElt,
                  ~r::[], ~rules::[], ~result::Tup)
{Simplifies a given a pair (u,r) using the given list of rules.}

    assert LC(u) eq 1;

    t := LM(r[k][1]);
    i := r[k][2];

    assert LC(t) eq 1;

    for j in [1..#rules[i]] do

        t_j := rules[i,j][1];
        k_j := rules[i,j][2];
        assert LC(t_j) eq 1;

        // assert entry rules[i,j] is correct:
        assert LM(r[k_j][1]) eq t_j;
        assert r[k_j][2] eq i;



        if IsDivisibleBy(u*t,t_j) then
            result := < (u*t)div t_j, k_j >;
            return;
        end if;
    end for;
```

```
    result := < u, k >; // could not be simplified
end intrinsic;



/*********************************************************
  IsRewritable [Rewritten?]

  Checks if a term u can be rewritten using a given
  labeled polynomial r[k].

  Input:
        u:          term
        k:          r-index
        iter:       # of global iteration
        r:          global list of labeled polynomials
        rules:      global list of rules

  Returns (in ~result):
        True if the term u can be rewritten using the
        labeled polynomial r_k, false otherwise.

*********************************************************/
intrinsic IsRewritable(u::RngMPolElt,k::RngIntElt, iter::RngIntElt,
                       ~r::[],~rules::[],~result::BoolElt)
{Checks if a term u can be rewritten using
a given labeled polynomial r[k].}

    subresult := <-42>;
    Rewrite(u,k,iter,~r,~rules,~subresult);
    term, l := Explode(subresult);
```

```
    assert LC(term) eq 1;

    result := l ne k;
      //print "Rewritten by ",l;

end intrinsic;




/***********************************************************
    CritPair

    Given two r-indices i,j, check if one of the two
    critical pairs (r_i,r_j), (r_j,r_i) is normalized.
    If so, assemble and return it.


    Input:

        i:          r-index of a normalized labeled
                    polynomial
        j:          r-index of a normalized  labeled
                    polynomial
        r:          "global" array of labeled polynomials
        k:          current iteration in IncrementalF5
        psi:        function that returns true for a
                    labeled polynomial x iff x is
                    top-reducible mod G_{l+1}, where
                    l = x[2]

    Returns (in ~result):
```

```
        result      a list [* p, is_pair, crit *]

        is_pair:    is true if this critical pair is
                    normalized, false otherwise.

        p:          If is_pair is true, p is a critical
                    pair, i.e. a tuple <t,u,k,v,l>,
                    where t is the LCM of the HTs of
                    the polynomials indexed by k,l,
                    u is t div HT(r[k]), v is t div
                    HT(r[l]). {k,l} is {i,j} (not
                    necessarily [k,l]=[i,j], as the
                    components may be swapped (cf.
                    Faug'ere's  Def. 3). The pair is
                    guaranteed to be normalized.

        crit:       number of the criterion used to
                    identify the pair as redundant
                    (see constants above)

**********************************************************/
intrinsic CritPair(i::RngIntElt, j::RngIntElt, ~r::[],
                   iter::RngIntElt, ~psi::Program, ~result::Tup)
{Compute critical pair for two rules,  if necessary.}

    pol_ring := Parent(r[i][3]);
    rule_univ := RuleUniverse(pol_ring);
    pair_univ := PairUniverse(pol_ring);

    t := LCMCritPair(r[i],r[j]);
```

```
    u_1 := pol_ring!t div pol_ring!LM(r[i][3]);
    u_2 := pol_ring!t div pol_ring!LM(r[j][3]);


    if SignatureLess(<u_1*r[i][1],r[i][2]>,
                    <u_2*r[j][1],r[j][2]>) lt 0 then
        // swap rules
        tmp  := i;
        i := j;
        j := tmp;

        tmp := u_1;
        u_1 := u_2;
        u_2 := tmp;
    end if;

    t_1 := LM(r[i][1]);
    t_2 := LM(r[j][1]);


    k_1 := r[i][2];
    k_2 := r[j][2];


    function Criterion1()
        if k_1 gt iter then
            //TODO: can't happen!
            //printf "crit (1) ruled out pair %o.\n", [i,j];

            //if ((i eq 4) and (j eq 23)) then
            //  print "HA!1";
            //  readi debugger;
```

```
        //end if;
        return false;
    else
        return true;
    end if;
end function;



function Criterion2()
    if psi(<u_1*t_1,k_1,1>) then
        //printf "crit (2) ruled out pair %o\n", [i,j];

        //if ((i eq 4) and (j eq 23)) then
        //  print "HA!2";
        //  readi debugger;
        //end if;
        return false;
    else
        return true;
    end if;
end function;



function Criterion3()
    if psi(<u_2*t_2,k_2,1>) then  //DELTA_Faugere
        //printf "crit (3) ruled out pair %o\n", [i,j];

        //if ((i eq 4) and (j eq 23)) then
        //  print "HA!3";
        //  readi debugger;
        //end if;
        return false;
```

```
        else
            return true;
        end if;
    end function;


    pair := pair_univ!<t, u_1, i, u_2, j>;

    if not Criterion1() then
        result := < pair, false, CRITERION_ONE >;
        return;
    elif not Criterion2() then
        result := < pair, false, CRITERION_TWO >;
        return;
    elif not Criterion3() then
        result := < pair, false, CRITERION_THREE >;
        return;
    else
        //printf "New crit pair: <%o, %o, r_%o, %o, r_%o>\n"
                 ,t,u_1,i,u_2,j;

        result := < pair, true, 0 >;
        return;
    end if;

end intrinsic;



/*************************************************************

   Spols [Spol]
```

```
 Calculate S-polynomials for a sequence of
 critical pairs, eliminating redundant
 S-polynomials using simplification rules.

 Input:

      G_i:        current intermediate Groebner
                  basis of <f_i,...,f_m>

      critpairs:  sequence of critical pairs

      iterandhasbeeninD
<iter,hasbeeninD>

      iter:       # of current global iteration

hasbeeninD: list of r-indices that have been in DD

      r:          "global" list of labeled polynomials,
                  receives any S-polynomials created

      rules:      "global" list of simplification rules,
                  is updated if any S-polynomials are
                  created

      F_dropped:  tuple, receives the return value


 Returns (in ~F_dropped):

      F_dropped[1]: sequence of the r-indices of the
```

```
                    S-polynomials for the given critical
                    pairs except those that were detected
                    as unnecessary

        F_dropped[2]: sequence of those critical pairs
                    in critpairs for which the S-polynomial
                    was dropped, i.e., not inserted into r



************************************************************/
intrinsic Spols(~G_i::[],~critpairs::[], iterandhasbeeninD::Tup,
                ~r::[], ~rules::[], ~F_dropped)
{Calculate non-redundant S-polynomials for a sequence of critical pairs.}

    iter := iterandhasbeeninD[1];
    hasbeeninD := iterandhasbeeninD[2];
    cantest := true;
    // boolean that will indicate whether a particular critical pair
    // can be tested for Rewrite

    pol_ring := Parent(r[iter][3]);




    rule_univ := RuleUniverse(pol_ring);

    // # of S-polynomials that are 0
    nZero := 0;

    F := [ ];

    droppedL := [];
```

```
print "Spols: Total number of pairs to examine:", #critpairs;
for pair in critpairs do

    u := pair[2];
    v := pair[4];

    il := pair[3];
    jl := pair[5];

    assert il ne jl;

    // if one of the two r-indices of the S-polynomial have
    // been in DD, cannot Rewrite

    for index in hasbeeninD do
      if ((index eq il) or (index eq jl)) then
        cantest := false;
        break;
      end if;
    end for;

    // ** Determine if we need the S-polynomial **

    // Criterion 1: is none of the polys zero?
    needSpol := (r[il][3] ne 0) and (r[jl][3] ne 0);

    lc_il := LC(u*r[il][3]);
    lc_jl := LC(v*r[jl][3]);

    sp := u*r[il][3]-  lc_il / lc_jl * v*r[jl][3];
```

```
if sp eq 0 then
    // Skip zeroes
    //printf "\nS-Polynomial of pair %o is 0, skipping\n",pair;
    //printf "r_%o = %o\n", il, r[il];
    //printf "r_%o = %o\n", jl, r[jl];
    nZero +:= 1;
    continue;
end if;



// Criterion 2: Can the left summand be rewritten?
if ((needSpol) and (cantest)) then
    result := false;
    IsRewritable(u, il, iter, ~r, ~rules, ~result);
    needSpol := not result;
end if;
//end if;

// Criterion 3: Can the right summand be rewritten?
if ((needSpol) and (cantest)) then
    IsRewritable(v, jl, iter, ~r, ~rules, ~result);
    needSpol := not result;
end if;
//end if;




if needSpol then


    N := #r + 1; // "increment" N (it's actually not global)


    assert AssertReducedTo(u*r[il][3],sp);
```

```
            assert AssertReducedTo(v*r[jl][3],sp);

            r[N] := rule_univ!<u * r[il][1], r[il][2], sp>;

            AddRule(~rules,~r,N,pol_ring);

            Append(~F,N);

        else

            droppedL cat:= [pair];


        end if;
    end for;

    cmpFunc := -42; //initialize with dummy value
    IndexSignatureLess(~r,~cmpFunc);
    Sort(~F, cmpFunc);

    if GetAssertions() then
        degs := {TotalDegree(r[l][3]): l in F};
        error if #degs gt 1,
              "S-polynomials have different degrees!:",degs;
    end if;

    F_dropped := [* F, droppedL *];
end intrinsic;
```

Justin M. Gash–Curriculum Vitae

2500 Stonelake Circle
Bloomington, IN 47404

EDUCATION

Ph.D. in Mathematics, Indiana University - Bloomington, June 2008
Thesis Ttile: On Efficient Computation of Grobner Bases
Thesis Advisor: Professor Jee Koh

M.A. in Computer Science, Indiana University - Bloomington, May 2007

Bachelor of Arts, DePauw University, May 2001
Summa Cum Laude
Phi Beta Kappa
Majors: Mathematics, Computer Science
Minor: Economics

TEACHING EXPERIENCE

Indiana University (2001-present)

Associate Instructor: Responsible for lectures, writing quizzes, grading quizzes, assigning homework, grading homework, writing exams, grading exams, holding office hours, and giving review sessions for 14 different courses. Courses include:

Finite Mathematics (D116, D117, X018): These courses focus on basic set theory, permutations, combinations, probability, solving systems of equations, solving systems of inequalities, matrix methods and Markov chains.

D116:Fall 2007 (2sections), Spring 2006 (2 sections), Fall 2006 (2 sections), Spring 2004 (1 section)
D117: Fall 2004 (2 sections)
X018: Fall 2003, Fall 2002, Spring 2002 (1 section each)

Math for Elementary Education Majors (T101): This course focuses on methods of teaching arithmetic, graphical models for arithmetic, methods for presenting solutions, fractions, the place-value system and number bases, and an introduction to proof writing.

T101: Spring 2006, Fall 2005 (2 sections each)

College Algebra (M014, M025): These courses focus on basic topics like solving equations and inequalities, factoring, graphing lines, graphing polynomials, graph manipulations, logarithms, and matrix methods.

M025: Summer 2006 (1 section)
M014: Spring 2003 (1 section)

Basic Arithmetic (J010): This course is for first-generation college students and is taught during the summer preceding their first official semester. Topics include a review of basic arithmetic, basic geometry, fractions, and word problems.

J010: Summer 2007, Summer 2005, Summer 2004, Summer 2003, Summer 2002 (1 section each)

Recitations (Fall 2001): Responsible for administering quizzes, grading quizzes, proctoring exams, grading exams, and giving three recitations per week for three different calculus courses.

DePauw University (1999-2001)

Associate for Academic Resource Center: Responsible for working four hours per week and offering general mathematics help to students.

Review Sessions for Calculus: Responsible for running review sessions for the calculus section before exam. This was supervised by the mathematics department.

Private Tutoring (1998-present):Offered private tutoring lessons in mathematics. Topics include finite mathematics, algebra, calculus, trigonometry, pre-calculus and basic arithmetic.

RESEARCH INTERESTS

Cryptography: Specifically I enjoy trying to find cryptographic primitives and finding weaknesses in existing primitives.

Computational Algebra: This is the research area of my thesis. It involves looking at algorithms that implement operations on algebraic structures. I specifically like trying to improve algorithms and understand their computational complexity.

Computer Security and Phishing: This area involves understanding current network security standards and general Internet security and privacy measures. One particular area of interest to study ways in which phishing can be implemented (and thus, countermeasures can be found).

AWARDS

Rothrock Teaching Award for Associate Instructors, Indiana University - Bloomington, May 2003: In recognition in outstanding performance in teaching mathematics and in expectation of continued academic success.

PRESENTATIONS

An Introduction to Advanced Encryption Standard, March 23

2007,University of Indianapolis, INMAA Spring Meeting

Teaching Portfolio Share Fair, February 23 2007, Indiana University - Bloomington, Campus Instructional Consulting

Delayed Password Disclosure, April 4, 2006, Indiana University - Bloomington, I690 Informatics - Topics in Security

Oblivious Transfer and It's Applications, March 1 2006, Indiana University - Bloomington, Graduate Student Algebra Seminar

PAKE (Password Authenticated Key Exchange), February 26, 2006, Indiana University - Bloomington, I690 Informatics - Topics in Security

AES (Advanced Encryption Standard), November 4, 2005, Indiana University - Bloomington, Informatics Security Seminar

AES and Grobner Bases (part II): November 2, 2005, Indiana University - Bloomington, Graduate Student Algebra Seminar

AES and Grobner Bases (part I): October 26, 2005, Indiana University - Bloomington, Graduate Student Algebra Seminar

Creating Cryptosystem Primitives from Non-Abelian Groups (Part II), January 26, 2005, Indiana University - Bloomington, Graduate Student Algebra Seminar

Creating Cryptosystem Primitives from Non-Abelian Groups (Part I), January 19, 2005, Indiana University - Bloomington, Graduate Student Algebra Seminar

PRIMES is in P, May 5, 2004, Indiana University - Bloomington, Graduate Student Algebra Seminar

PROGRAMMING EXPERIENCE

C, C++, Matlab, Maple (v 9.0), Magma

SERVICE

Graduate Student Algebra Seminar: Organized seminars, scheduled rooms, advertised talks and personally gave talks for the graduate student algebra seminar during the 2005-2006 school year.

Free Tutoring: Tutored a computer science student in cryptography during the Fall of 2005.

Additional Help Sessions: In addition to an assignment to run one evening review session per week during the Spring of 2004, I held a second review session every week.

PROFESSIONAL ORGANIZATIONAL MEMBERSHIP

American Mathematical Society

Mathematics Association of America