## Bangor University

## DOCTOR OF PHILOSOPHY

## Noncommutative involutive bases

Evans, Gareth Alun

Award date:
2005

Awarding institution:
University of Wales, Bangor

Link to publication

## General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

# Noncommutative Involutive Bases 

Thesis submitted to the University of Wales in support of the application for the degree of Philosophiæ Doctor
by

Gareth Alun Evans

School of Informatics
The University of Wales, Bangor
September 2005

- PRIFYSGOL CYMRU •

BANGOR


## Summary

The theory of Gröbner Bases originated in the work of Buchberger [11] and is now considered to be one of the most important and useful areas of symbolic computation. A great deal of effort has been put into improving Buchberger's algorithm for computing a Gröbner Basis, and indeed in finding alternative methods of computing Gröbner Bases. Two of these methods include the Gröbner Walk method [1] and the computation of Involutive Bases [58].

By the mid 1980's, Buchberger's work had been generalised for noncommutative polynomial rings by Bergman [8] and Mora [45]. This thesis provides the corresponding generalisation for Involutive Bases and (to a lesser extent) the Gröbner Walk, with the main results being as follows.
(1) Algorithms for several new noncommutative involutive divisions are given, including strong; weak; global and local divisions.
(2) An algorithm for computing a noncommutative Involutive Basis is given. When used with one of the aforementioned involutive divisions, it is shown that this algorithm returns a noncommutative Gröbner Basis on termination.
(3) An algorithm for a noncommutative Gröbner Walk is given, in the case of conversion between two harmonious monomial orderings. It is shown that this algorithm generalises to give an algorithm for performing a noncommutative Involutive Walk, again in the case of conversion between two harmonious monomial orderings.
(4) Two new properties of commutative involutive divisions are introduced (stability and extendibility), respectively ensuring the termination of the Involutive Basis algorithm and the applicability (under certain conditions) of homogeneous methods of computing Involutive Bases.

Source code for an initial implementation of an algorithm to compute noncommutative Involutive Bases is provided in Appendix B. This source code, written using ANSI C and a series of libraries (AlgLib) provided by MSSRC [46], forms part of a larger collection of programs providing examples for the thesis, including implementations of the commutative and noncommutative Gröbner Basis algorithms [11, 45]; the commutative Involutive Basis algorithm for the Pommaret and Janet involutive divisions [58]; and the Knuth-Bendix critical pairs completion algorithm for monoid rewrite systems [39].

## Acknowledgements

Many people have inspired me to complete this thesis, and I would like to take this opportunity to thank some of them now.

I would like to start by thanking my family for their constant support, especially my parents who have encouraged me every step of the way. Mae fy nyled yn fawr iawn i chi.

I would like to thank Prof. Larry Lambe from MSSRC, whose software allowed me to test my theories in a way that would not have been possible elsewhere.

Thanks to all the Mathematics Staff and Students I have had the pleasure of working with over the past seven years. Particular thanks go to Dr. Bryn Davies, who encouraged me to think independently; to Dr. Jan Abas, who inspired me to reach goals I never thought I could reach; and to Prof. Ronnie Brown, who introduced me to Involutive Bases.

I would like to finish by thanking my Supervisor Dr. Chris Wensley. Our regular meetings kept the cogs in motion and his insightful comments enabled me to avoid wrong turnings and to get the little details right. Diolch yn fawr!

This work has been gratefully supported by the EPSRC and by the School of Informatics at the University of Wales, Bangor.
"No one has ever done anything like this."
"That's why it's going to work."
The Matrix [54]

## Contents

Summary ..... ii
Acknowledgements ..... iii
Introduction ..... 1
Background ..... 1
Structure and Principal Results ..... 5
1 Preliminaries ..... 9
1.1 Rings and Ideals ..... 9
1.1.1 Groups and Rings ..... 9
1.1.2 Polynomial Rings ..... 10
1.1.3 Ideals ..... 12
1.2 Monomial Orderings ..... 14
1.2.1 Commutative Monomial Orderings ..... 14
1.2.2 Noncommutative Monomial Orderings ..... 16
1.2.3 Polynomial Division ..... 17
2 Commutative Gröbner Bases ..... 22
2.1 S-polynomials ..... 22
2.2 Dickson's Lemma and Hilbert's Basis Theorem ..... 27
2.3 Buchberger's Algorithm ..... 30
2.4 Reduced Gröbner Bases ..... 32
2.5 Improvements to Buchberger's Algorithm ..... 34
2.5.1 Buchberger's Criteria ..... 35
2.5.2 Homogeneous Gröbner Bases ..... 39
2.5.3 Selection Strategies ..... 40
2.5.4 Basis Conversion Algorithms ..... 42
2.5.5 Optimal Variable Orderings ..... 43
2.5.6 Logged Gröbner Bases ..... 43
3 Noncommutative Gröbner Bases ..... 46
3.1 Overlaps ..... 47
3.2 Mora's Algorithm ..... 52
3.2.1 Termination ..... 53
3.3 Reduced Gröbner Bases ..... 54
3.4 Improvements to Mora's Algorithm ..... 56
3.4.1 Buchberger's Criteria ..... 56
3.4.2 Homogeneous Gröbner Bases ..... 59
3.4.3 Selection Strategies ..... 62
3.4.4 Logged Gröbner Bases ..... 63
3.5 A Worked Example ..... 63
3.5.1 Initialisation ..... 63
3.5.2 Calculating and Reducing S-polynomials ..... 64
3.5.3 Applying Buchberger's Second Criterion ..... 66
3.5.4 Reduction ..... 68
4 Commutative Involutive Bases ..... 70
4.1 Involutive Divisions ..... 71
4.1.1 Involutive Reduction ..... 73
4.1.2 Thomas, Pommaret and Janet divisions ..... 74
4.2 Prolongations and Autoreduction ..... 77
4.3 Continuity and Constructivity ..... 80
4.4 The Involutive Basis Algorithm ..... 86
4.5 Improvements to the Involutive Basis Algorithm ..... 96
4.5.1 Improved Algorithms ..... 96
4.5.2 Homogeneous Involutive Bases ..... 96
4.5.3 Logged Involutive Bases ..... 102
5 Noncommutative Involutive Bases ..... 104
5.1 Noncommutative Involutive Reduction ..... 105
5.2 Prolongations and Autoreduction ..... 110
5.3 The Noncommutative Involutive Basis Algorithm ..... 112
5.4 Continuity and Conclusivity ..... 112
5.4.1 Properties for Strong Involutive Divisions ..... 114
5.4.2 Properties for Weak Involutive Divisions ..... 117
5.5 Noncommutative Involutive Divisions ..... 119
5.5.1 Two Global Divisions ..... 119
5.5.2 An Overlap-Based Local Division ..... 124
5.5.3 A Strong Local Division ..... 135
5.5.4 Alternative Divisions ..... 142
5.6 Termination ..... 148
5.7 Examples ..... 149
5.7.1 A Worked Example ..... 149
5.7.2 Involutive Rewrite Systems ..... 154
5.7.3 Comparison of Divisions ..... 157
5.8 Improvements to the Noncommutative Involutive Basis Algorithm ..... 158
5.8.1 Efficient Reduction ..... 158
5.8.2 Improved Algorithms ..... 161
5.8.3 Logged Involutive Bases ..... 162
6 Gröbner Walks ..... 165
6.1 Commutative Walks ..... 166
6.1.1 Matrix Orderings ..... 166
6.1.2 The Commutative Gröbner Walk Algorithm ..... 167
6.1.3 A Worked Example ..... 169
6.1.4 The Commutative Involutive Walk Algorithm ..... 176
6.2 Noncommutative Walks ..... 176
6.2.1 Functional Decompositions ..... 177
6.2.2 The Noncommutative Gröbner Walk Algorithm for Harmonious Monomial Orderings ..... 179
6.2.3 A Worked Example ..... 182
6.2.4 The Noncommutative Involutive Walk Algorithm for Harmonious Monomial Orderings ..... 185
6.2.5 A Worked Example ..... 187
6.2.6 Noncommutative Walks Between Any Two Monomial Orderings? ..... 190
7 Conclusions ..... 191
7.1 Current State of Play ..... 191
7.2 Future Directions ..... 192
A Proof of Propositions 5.5.31 and 5.5.32 ..... 194
A. 1 Proposition 5.5.31 ..... 194
A. 2 Proposition 5.5.32 ..... 203
B Source Code ..... 212
B. 1 Methodology ..... 212
B.1.1 MSSRC ..... 213
B.1.2 AlgLib ..... 214
B. 2 Listings ..... 215
B.2.1 README ..... 216
B.2.2 arithmetic_functions.h ..... 219
B.2.3 arithmetic_functions.c ..... 220
B.2.4 file_functions.h ..... 226
B.2.5 file_functions.c ..... 227
B.2.6 fralg-functions.h ..... 238
B.2.7 fralg-functions.c ..... 240
B.2.8 list_functions.h ..... 268
B.2.9 list_functions.c ..... 271
B.2.10 ncinv_functions.h ..... 290
B.2.11 ncinv_functions.c ..... 291
B.2.12 involutive.c ..... 340
C Program Output ..... 356
C. 1 Sample Sessions ..... 356
C.1.1 Session 1: Locally Involutive Bases ..... 356
C.1.2 Session 2: Involutive Complete Rewrite Systems ..... 358
C.1.3 Session 3: Noncommutative Involutive Walks ..... 361
C.1.4 Session 4: Ideal Membership ..... 365
Bibliography ..... 368
Index ..... 374

## List of Algorithms

1 The Commutative Division Algorithm ..... 19
2 The Noncommutative Division Algorithm ..... 20
3 A Basic Commutative Gröbner Basis Algorithm ..... 31
4 The Commutative Unique Reduced Gröbner Basis Algorithm ..... 35
5 Mora's Noncommutative Gröbner Basis Algorithm ..... 52
6 The Noncommutative Unique Reduced Gröbner Basis Algorithm ..... 57
7 The Commutative Involutive Division Algorithm ..... 73
8 The Commutative Autoreduction Algorithm ..... 78
9 The Commutative Involutive Basis Algorithm ..... 87
10 The Noncommutative Involutive Division Algorithm ..... 108
11 The Noncommutative Autoreduction Algorithm ..... 110
12 The Noncommutative Involutive Basis Algorithm ..... 113
13 The Left Overlap Division $\mathcal{O}$ ..... 127
14 'DisjointCones' Function for Algorithm 15 ..... 136
15 The Strong Left Overlap Division $\mathcal{S}$ ..... 137
16 The Two-Sided Left Overlap Division $\mathcal{W}$ ..... 145
17 The Commutative Gröbner Walk Algorithm ..... 168
18 The Commutative Involutive Walk Algorithm ..... 175
19 The Noncommutative Gröbner Walk Algorithm for Harmonious Monomial Orderings ..... 180
20 The Noncommutative Involutive Walk Algorithm for Harmonious Mono- mial Orderings ..... 185

## Introduction

## Background

## Gröbner Bases

During the second half of the twentieth century, one of the most successful applications of symbolic computation was in the development and application of Gröbner Basis theory for finding special bases of ideals in commutative polynomials rings. Pioneered by Bruno Buchberger in 1965 [11], the theory allowed an answer to the question "What is the unique remainder when a polynomial is divided by a set of polynomials?". Buchberger's algorithm for computing a Gröbner Basis was improved and refined over several decades [ $1,10,21,29$ ], aided by the development of powerful symbolic computation systems over the same period. Today there is an implementation of Buchberger's algorithm in virtually all general purpose symbolic computation systems, including Maple [55] and Mathematica [57], and many more specialised systems.

## What is a Gröbner Basis?

Consider the problem of finding the remainder when a number is divided by a set of numbers. If the dividing set contains just one number, then the problem only has one solution. For example, " 5 " is the only possible answer to the question "What is $20 \div 4$ ?". If the dividing set contains more than one number however, there may be several solutions, as the division can potentially be performed in more than one way.

Example. Consider a tank containing 21L of water. Given two empty jugs, one with a capacity of 2 L and the other 5 L , is it possible to empty the tank using just the jugs, assuming only full jugs of water may be removed from the tank?


Trying to empty the tank using the 2 L jug only, we are able to remove $10 \times 2=20 \mathrm{~L}$ of water from the tank, and we are left with 1 L of water in the tank. Repeating with the 5 L jug, we are again left with 1 L of water in the tank. If we alternate between the jugs however (removing 2 L of water followed by 5 L followed by 2 L and so on), the tank this time does become empty, because $21=2+5+2+5+2+5$.

The observation that we are left with a different volume of water in the tank dependent upon how we try to empty it corresponds to the idea that the remainder obtained when dividing the number 21 by the numbers 2 and 5 is dependent upon how the division is performed.

This idea also applies when dividing polynomials by sets of polynomials - remainders here will also be dependent upon how the division is performed. However, if we divide a polynomial with respect to a set of polynomials that is a Gröbner Basis, then we will always obtain the same remainder no matter how the division is performed. This fact, along with the fact that any set of polynomials can be transformed into an equivalent set of polynomials that is a Gröbner Basis, provides the main ingredients of Gröbner Basis theory.

Remark. The 'Gröbner Basis' for our water tank example would be just a 1L jug, allowing us to empty any tank containing $n \mathrm{~L}$ of water (where $n \in \mathbb{N}$ ).

## Applications

There are numerous applications of Gröbner Bases in all branches of mathematics, computer science, physics and engineering [12]. Topics vary from geometric theorem proving to solving systems of polynomial equations, and from algebraic coding theory to the design of experiments in statistics.

Example. Let $F:=\left\{x+y+z=6, x^{2}+y^{2}+z^{2}=14, x^{3}+y^{3}+z^{3}=36\right\}$ be a set of polynomial equations. One way of solving this set for $x, y$ and $z$ is to compute a lexicographic Gröbner Basis for $F$. This yields the set $G:=\left\{x+y+z=6, y^{2}+y z+z^{2}-\right.$ $\left.6 y-6 z=-11, z^{3}-6 z^{2}+11 z=6\right\}$, the final member of which is a univariate polynomial in $z$, a polynomial we can solve to deduce that $z=1,2$ or 3 . Substituting back into the second member of $G$, when $z=1$, we obtain the polynomial $y^{2}-5 y+6=0$, which enables us to deduce that $y=2$ or 3 ; when $z=2$, we obtain the polynomial $y^{2}-4 y+3=0$, which enables us to deduce that $y=1$ or 3 ; and when $z=3$, we obtain the polynomial $y^{2}-3 y+2=0$, which enables us to deduce that $y=1$ or 2 . Further substitution into $x+y+z=6$ then enables us to deduce the value of $x$ in each of the above cases, enabling us to give the following table of solutions for $F$.

| x | 3 | 2 | 3 | 1 | 2 | 1 |
| :--- | :--- | :--- | :--- | :--- | :--- | :--- |
| y | 2 | 3 | 1 | 3 | 1 | 2 |
| z | 1 | 1 | 2 | 2 | 3 | 3 |

## Involutive Bases

As Gröbner Bases became popular, researchers noticed a connection between Buchberger's ideas and ideas originating from the Janet-Riquier theory of Partial Differential Equations developed in the early 20th century (see for example [44]). This link was completed for commutative polynomial rings by Zharkov and Blinkov in the early 1990's [58] when they gave an algorithm to compute an Involutive Basis that provides an alternative way of computing a Gröbner Basis. Early implementations of this algorithm (an elementary introduction to which can be found in [13]) compared favourably with the most advanced implementations of Buchberger's algorithm, with results in [25] showing the potential of the Involutive method in terms of efficiency.

## What is an Involutive Basis?

Given a Gröbner Basis $G$, we know that the remainder obtained from dividing a polynomial with respect to $G$ will always be the same no matter how the division is performed. With an Involutive Basis, the difference is that there is only one way for the division to be performed, so that unique remainders are also obtained uniquely.

This effect is achieved through assigning a set of multiplicative variables to each polynomial
in an Involutive Basis $H$, imposing a restriction on how polynomials may be divided by $H$ by only allowing any polynomial $h \in H$ to be multiplied by its corresponding multiplicative variables. Popular schemes of assigning multiplicative variables include those based on the work of Janet [35], Thomas [52] and Pommaret [47].

Example. Consider the Janet Involutive Basis $H:=\left\{x y-z, y z+2 x+z, 2 x^{2}+x z+\right.$ $\left.z^{2}, 2 x^{2} z+x z^{2}+z^{3}\right\}$ with multiplicative variables as shown in the table below.

| Polynomial | Janet Multiplicative Variables |
| :---: | :---: |
| $x y-z$ | $\{x, y\}$ |
| $y z+2 x+z$ | $\{x, y, z\}$ |
| $2 x^{2}+x z+z^{2}$ | $\{x\}$ |
| $2 x^{2} z+x z^{2}+z^{3}$ | $\{x, z\}$ |

To illustrate that any polynomial may only be involutively divisible by at most one member of any Involutive Basis, we include the following two diagrams, showing which monomials are involutively divisible by $H$, and which are divisible by the corresponding Gröbner Basis $G:=\left\{x y-z, y z+2 x+z, 2 x^{2}+x z+z^{2}\right\}$.



Note that the irreducible monomials of both bases all appear in the set $\left\{1, x, y^{i}, z^{i}, x z^{i}\right\}$, where $i \geqslant 1$; and that the cube, the 2 planes and the line shown in the right hand diagram do not overlap.

## Noncommutative Bases

There are certain types of noncommutative algebra to which methods for commutative Gröbner Bases may be applied. Typically, these are algebras with generators $\left\{x_{1}, \ldots, x_{n}\right\}$ for which products $x_{j} x_{i}$ with $j>i$ may be rewritten as ( $x_{i} x_{j}+$ other terms). For example, version 3-0-0 of Singular [31] (released in June 2005) allows the computation of Gröbner Bases for $G$-algebras.

To compute Gröbner Bases for ideals in free associative algebras however, one must turn to the theory of noncommutative Gröbner Bases. Based on the work of Bergman [8] and Mora [45], the theory answers the question "What is the remainder when a noncommutative polynomial is divided by a set of noncommutative polynomials?", and allows us to find Gröbner Bases for such algebras as path algebras [37].

The final piece of the jigsaw is to mirror the application of Zharkov and Blinkov's Involutive methods to the noncommutative case. This thesis provides the first extended attempt at accomplishing this task, improving the author's first basic algorithms for computing noncommutative Involutive Bases [20] and providing a full theoretical foundation for these algorithms.

## Structure and Principal Results

This thesis can be broadly divided into two parts: Chapters 1 through 4 survey the building blocks required for the theory of noncommutative Involutive Bases; the remainder of the thesis then describes this theory together with different ways of computing noncommutative Involutive Bases.

## Part 1

Chapter 1 contains accounts of some necessary preliminaries for our studies - a review of both commutative and noncommutative polynomial rings; ideals; monomial orderings; and polynomial division.

We survey the theory of commutative Gröbner Bases in Chapter 2, basing our account on many sources, but mainly on the books [7] and [22]. We present the theory from the viewpoint of S-polynomials (for example defining a Gröbner Basis in terms of S-
polynomials), mainly because Buchberger's algorithm for computing a Gröbner Basis deals predominantly with S-polynomials. Towards the end of the Chapter, we describe some of the theoretical improvements of Buchberger's algorithm, including the usage of selection strategies, optimal variable orderings and Logged Gröbner Bases.

The viewpoint of defining Gröbner Bases in terms of S-polynomials continues in Chapter 3, where we encounter the theory of noncommutative Gröbner Bases. We discover that the theory is quite similar to that found in the previous chapter, apart from the definition of an S-polynomial and the fact that not all input bases will have finite Gröbner Bases.

In Chapter 4, we acquaint ourselves with the theory of commutative Involutive Bases. This is based on the work of Zharkov and Blinkov [58]; Gerdt and Blinkov [25, 26]; Gerdt [23, 24]; Seiler [50, 51]; and Apel [2, 3], with the notation and conventions taken from a combination of these papers. For example, notation for involutive cones and multiplicative variables is taken from [25], and the definition of an involutive division and the algorithm for computing an Involutive Basis is taken from [50].

As for the content of Chapter 4, we introduce the Janet, Pommaret and Thomas divisions in Section 4.1; describe what is meant by a prolongation and autoreduction in Section 4.2; introduce the properties of continuity and constructivity in Section 4.3; give the Involutive Basis algorithm in Section 4.4; and describe some improvements to this algorithm in Section 4.5. In between all of this, we introduce two new properties of involutive divisions, stability and extendibility, that ensure (respectively) the termination of the Involutive Basis algorithm and the applicability (under certain conditions) of homogeneous methods of computing Involutive Bases.

## Part 2

The main results of the thesis are contained in Chapter 5, where we introduce the theory of noncommutative Involutive Bases. In Section 5.1, we define two methods of performing noncommutative involutive reduction, the first of which (using thin divisors) allows the mirroring of theory from Chapter 4, and the second of which (using thick divisors) allows efficient computation of involutive remainders. We also define what is meant by a noncommutative involutive division, and give an algorithm for performing noncommutative involutive reduction.

In Section 5.2, we generalise the notions of prolongation and autoreduction to the non-
commutative case, introducing two different types of prolongation (left and right) to reflect the fact that left and right multiplication are different operations in noncommutative polynomial rings. These notions are then utilised in the algorithm for computing a noncommutative Involutive Basis, which we present in Section 5.3.

In Section 5.4, we introduce two properties of noncommutative involutive divisions. Continuity helps ensure that any Locally Involutive Basis is an Involutive Basis; conclusivity ensures that for any given input basis, a finite Involutive Basis will exist if and only if a finite Gröbner Basis exists. A third property is also introduced for weak involutive divisions to ensure that any Locally Involutive Basis is a Gröbner Basis (Involutive Bases with respect to strong involutive divisions are automatically Gröbner Bases).

Section 5.5 provides several involutive divisions for use with the noncommutative Involutive Basis algorithm, including two global divisions and ten local divisions. The properties of these divisions are analysed, with full proofs given that certain divisions satisfy certain properties. We also show that some divisions are naturally suited for efficient involutive reduction, and speculate on the existence of further involutive divisions.

In Section 5.6, we briefly discuss the topic of the termination of the noncommutative Involutive Basis algorithm. In Section 5.7, we provide several examples showing how noncommutative Involutive Bases are computed, including examples demonstrating the computation of involutive complete rewrite systems for groups. Finally, in Section 5.8, we discuss improvements to the noncommutative Involutive Basis algorithm, including how to introduce efficient involutive reduction and Logged Involutive Bases.

Chapter 6 introduces and generalises the theory of the Gröbner Walk, where a Gröbner Basis with respect to one monomial ordering may be computed from a Gröbner Basis with respect to another monomial ordering. In Section 6.1, we summarise the theory of the commutative Gröbner Walk (based on the papers [1] and [18]), and we describe a generalisation of the theory to the Involutive case due to Golubitsky [30]. In Section 6.2, we then go on to partially generalise the theory to the noncommutative case, giving algorithms to perform both Gröbner and Involutive Walks between two harmonious monomial orderings.

After some concluding remarks in Chapter 7, we provide full proofs for two Propositions from Section 5.5 in Appendix A. Appendix B then provides ANSI C source code for an initial implementation of the noncommutative Involutive Basis algorithm, together with
a brief description of the AlgLib libraries used in conjunction with the code. Finally, in Appendix C, we provide sample sessions showing the program given in Appendix B in action.

## Chapter 1

## Preliminaries

In this chapter, we will set out some algebraic concepts that will be used extensively in the following chapters. In particular, we will introduce polynomial rings and ideals, the main objects of study in this thesis.

### 1.1 Rings and Ideals

### 1.1.1 Groups and Rings

Definition 1.1.1 A binary operation on a set $S$ is a function $*: S \times S \rightarrow S$ such that associated with each ordered pair $(a, b)$ of elements of $S$ is a uniquely defined element $(a * b) \in S$.

Definition 1.1.2 A group is a set $G$, with a binary operation $*$, such that the following conditions hold.
(a) $g_{1} * g_{2} \in G$ for all $g_{1}, g_{2} \in G$ (closure).
(b) $g_{1} *\left(g_{2} * g_{3}\right)=\left(g_{1} * g_{2}\right) * g_{3}$ for all $g_{1}, g_{2}, g_{3} \in G$ (associativity).
(c) There exists an element $e \in G$ such that for all $g \in G, e * g=g=g * e$ (identity).
(d) For each element $g \in G$, there exists an element $g^{-1} \in G$ such that $g^{-1} * g=e=g * g^{-1}$ (inverses).

Definition 1.1.3 A group $G$ is abelian if the binary operation of the group is commutative, that is $g_{1} * g_{2}=g_{2} * g_{1}$ for all $g_{1}, g_{2} \in G$. The operation in an abelian group is often written additively, as $g_{1}+g_{2}$, with the inverse of $g$ written $-g$.

Definition 1.1.4 A rng is a set $R$ with two binary operations + and $\times$, known as addition and multiplication, such that addition has an identity element 0 , called zero, and the following axioms hold.
(a) $R$ is an abelian group with respect to addition.
(b) $\left(r_{1} \times r_{2}\right) \times r_{3}=r_{1} \times\left(r_{2} \times r_{3}\right)$ for all $r_{1}, r_{2}, r_{3} \in R$ (multiplication is associative).
(c) $r_{1} \times\left(r_{2}+r_{3}\right)=r_{1} \times r_{2}+r_{1} \times r_{3}$ and $\left(r_{1}+r_{2}\right) \times r_{3}=r_{1} \times r_{3}+r_{2} \times r_{3}$ for all $r_{1}, r_{2}, r_{3} \in R$ (the distributive laws hold).

Definition 1.1.5 A rng $R$ is a ring if it contains a unique element 1 , called the unit element, such that $1 \neq 0$ and $1 \times r=r=r \times 1$ for all $r \in R$.

Definition 1.1.6 A ring $R$ is commutative if multiplication (as well as addition) is commutative, that is $r_{1} \times r_{2}=r_{2} \times r_{1}$ for all $r_{1}, r_{2} \in R$.

Definition 1.1.7 A ring $R$ is noncommutative if $r_{1} \times r_{2} \neq r_{2} \times r_{1}$ for some $r_{1}, r_{2} \in R$.
Definition 1.1.8 If $S$ is a subset of a ring $R$ that is itself a ring under the same binary operations of addition and multiplication, then $S$ is a subring of $R$.

Definition 1.1.9 A ring $R$ is a division ring if every nonzero element $r \in R$ has a multiplicative inverse $r^{-1}$. A field is a commutative division ring.

### 1.1.2 Polynomial Rings

## Commutative Polynomial Rings

A nontrivial polynomial $p$ in $n$ (commuting) variables $x_{1}, \ldots, x_{n}$ is usually written as a sum

$$
\begin{equation*}
p=\sum_{i=1}^{k} a_{i} x_{1}^{e_{1}^{1}} x_{2}^{e_{i}^{2}} \ldots x_{n}^{e_{n}^{n}}, \tag{1.1}
\end{equation*}
$$

where $k$ is a positive integer and each summand is a term made up of a nonzero coefficient $a_{i}$ from some ring $R$ and a monomial $x_{1}^{e_{i}^{1}} x_{2}^{e_{i}^{2}} \ldots x_{n}^{e_{i}^{n}}$ in which the exponents $e_{i}^{1}, \ldots, e_{i}^{n}$ are
nonnegative integers. It is clear that each monomial may be represented in terms of its exponents only, as a multidegree $e_{i}=\left(e_{i}^{1}, e_{i}^{2}, \ldots, e_{i}^{n}\right)$, so that a monomial may be written as a multiset $\mathbf{x}^{e_{i}}$ over the set $\left\{x_{1}, \ldots, x_{n}\right\}$. This leads to a more elegant representation of a nontrivial polynomial,

$$
\begin{equation*}
p=\sum_{\alpha \in \mathbb{N}^{n}} a_{\alpha} \mathrm{x}^{\alpha}, \tag{1.2}
\end{equation*}
$$

and we may think of such a polynomial as a function $f$ from the set of all multidegrees $\mathbb{N}^{n}$ to the ring $R$ with finite support (only a finite number of nonzero images).

Example 1.1.10 Let $p=4 x^{2} y+2 x+\frac{19}{80}$ be a polynomial in two variables $x$ and $y$ with coefficients in $\mathbb{Q}$. This polynomial can be represented by the function $f: \mathbb{N}^{2} \rightarrow \mathbb{Q}$ given by

$$
f(\alpha)= \begin{cases}4, & \alpha=(2,1) \\ 2, & \alpha=(1,0) \\ \frac{19}{80}, & \alpha=(0,0) \\ 0 & \text { otherwise }\end{cases}
$$

Remark 1.1.11 The zero polynomial $p=0$ is represented by the function $f(\alpha)=0_{R}$ for all possible $\alpha$. The constant polynomial $p=1$ is represented by the function $f(\alpha)=1_{R}$ for $\alpha=(0,0, \ldots, 0)$, and $f(\alpha)=0_{R}$ otherwise.

Remark 1.1.12 The product $m_{1} \times m_{2}$ of two monomials $m_{1}, m_{2}$ with corresponding multidegrees $e_{1}, e_{2} \in \mathbb{N}^{n}$ is the monomial corresponding to the multidegree $e_{1}+e_{2}$. For example, if $m_{1}=x_{1}^{2} x_{2} x_{3}^{3}$ and $m_{2}=x_{1} x_{2} x_{3}^{2}$ (so that $e_{1}=(2,1,3)$ and $e_{2}=(1,1,2)$ ), then $m_{1} \times m_{2}=x_{1}^{3} x_{2}^{2} x_{3}^{5}$ as $e_{1}+e_{2}=(3,2,5)$.

Definition 1.1.13 Let $R\left[x_{1}, x_{2}, \ldots, x_{n}\right]$ denote the set of all functions $f: \mathbb{N}^{n} \rightarrow R$ such that each function $f$ represents a polynomial in $n$ variables $x_{1}, \ldots, x_{n}$ with coefficients over a ring $R$. Given two functions $f, g \in R\left[x_{1}, x_{2}, \ldots, x_{n}\right]$, let us define the functions $f+g$ and $f \times g$ as follows.

$$
\begin{array}{ll}
(f+g)(\alpha)=f(\alpha)+g(\alpha) & \text { for all } \alpha \in \mathbb{N}^{n} \\
(f \times g)(\alpha)=\sum_{\beta+\gamma=\alpha} f(\beta) \times g(\gamma) & \text { for all } \alpha \in \mathbb{N}^{n}
\end{array}
$$

Then the set $R\left[x_{1}, x_{2}, \ldots, x_{n}\right]$ becomes a ring, known as the polynomial ring in $n$ variables over $R$, with the functions corresponding to the zero and constant polynomials being the respective zero and unit elements of the ring.

Remark 1.1.14 In $R\left[x_{1}, x_{2}, \ldots, x_{n}\right], R$ is known as the coefficient ring.

## Noncommutative Polynomial Rings

A nontrivial polynomial $p$ in $n$ noncommuting variables $x_{1}, \ldots, x_{n}$ is usually written as a sum

$$
\begin{equation*}
p=\sum_{i=1}^{k} a_{i} w_{i} \tag{1.3}
\end{equation*}
$$

where $k$ is a positive integer and each summand is a term made up of a nonzero coefficient $a_{i}$ from some ring $R$ and a monomial $w_{i}$ that is a word over the alphabet $X=\left\{x_{1}, x_{2}, \ldots, x_{n}\right\}$. We may think of a noncommutative polynomial as a function $f$ from the set of all words $X^{*}$ to the ring $R$.

Remark 1.1.15 The zero polynomial $p=0$ is the polynomial $0_{R} \varepsilon$, where $\varepsilon$ is the empty word in $X^{*}$. Similarly $1_{R} \varepsilon$ is the constant polynomial $p=1$.

Remark 1.1.16 The product $w_{1} \times w_{2}$ of two monomials $w_{1}, w_{2} \in X^{*}$ is given by concatenation. For example, if $X=\left\{x_{1}, x_{2}, x_{3}\right\}, w_{1}=x_{3}^{2} x_{2}$ and $w_{2}=x_{1}^{3} x_{3}$, then $w_{1} \times w_{2}=$ $x_{3}^{2} x_{2} x_{1}^{3} x_{3}$.

Definition 1.1.17 Let $R\left\langle x_{1}, x_{2}, \ldots, x_{n}\right\rangle$ denote the set of all functions $f: X^{*} \rightarrow R$ such that each function $f$ represents a polynomial in $n$ noncommuting variables with coefficients over a ring $R$. Given two functions $f, g \in R\left\langle x_{1}, x_{2}, \ldots, x_{n}\right\rangle$, let us define the functions $f+g$ and $f \times g$ as follows.

$$
\begin{array}{ll}
(f+g)(w)=f(w)+g(w) & \text { for all } w \in X^{*} \\
(f \times g)(w)=\sum_{u \times v=w} f(u) \times g(v) & \text { for all } w \in X^{*} .
\end{array}
$$

Then the set $R\left\langle x_{1}, x_{2}, \ldots, x_{n}\right\rangle$ becomes a ring, known as the noncommutative polynomial ring in $n$ variables over $R$, with the functions corresponding to the zero and constant polynomials being the respective zero and unit elements of the ring.

### 1.1.3 Ideals

Definition 1.1.18 Let $\mathcal{R}$ be an arbitrary commutative ring. An ideal $J$ in $\mathcal{R}$ is a subring of $\mathcal{R}$ satisfying the following additional condition: $j r \in J$ for all $j \in J, r \in \mathcal{R}$.

Remark 1.1.19 In the above definition, if $\mathcal{R}$ is a polynomial ring in $n$ variables over a ring $R\left(\mathcal{R}=R\left[x_{1}, \ldots, x_{n}\right]\right)$, the ideal $J$ is a polynomial ideal. We will only consider polynomial ideals in this thesis.

Definition 1.1.20 Let $\mathcal{R}$ be an arbitrary noncommutative ring.

- A left (right) ideal $J$ in $\mathcal{R}$ is a subring of $\mathcal{R}$ satisfying the following additional condition: $r j \in J(j r \in J)$ for all $j \in J, r \in \mathcal{R}$.
- A two-sided ideal $J$ in $\mathcal{R}$ is a subring of $\mathcal{R}$ satisfying the following additional condition: $r_{1} j r_{2} \in J$ for all $j \in J, r_{1}, r_{2} \in \mathcal{R}$.

Remark 1.1.21 Unless otherwise stated, all noncommutative ideals considered in this thesis will be two-sided ideals.

Definition 1.1.22 A set of polynomials $P=\left\{p_{1}, p_{2}, \ldots, p_{m}\right\}$ is a basis for an ideal $J$ of a noncommutative polynomial ring $\mathcal{R}$ if every polynomial $q \in J$ can be written as

$$
\begin{equation*}
q=\sum_{i=1}^{k} \ell_{i} p_{i} r_{i}\left(\ell_{i}, r_{i} \in \mathcal{R}, p_{i} \in P\right) \tag{1.4}
\end{equation*}
$$

We say that $P$ generates $J$, written $J=\langle P\rangle$.
Remark 1.1.23 The above definition has an obvious generalisation for left and right ideals of noncommutative polynomial rings and for ideals of commutative polynomial rings.

Example 1.1.24 Let $\mathcal{R}$ be the noncommutative polynomial ring $\mathbb{Q}\langle x, y\rangle$, and let $J=\langle P\rangle$ be an ideal in $\mathcal{R}$, where $P:=\left\{x^{2} y+y x-2, y x y-x+4 y\right\}$. Consider the polynomial $q:=2 x^{3} y+y x^{2} y+2 x y x-4 x^{2} y+x^{3}-2 x y-4 x$, and let us ask if $q$ is a member of the ideal. To answer this question, we have to find out if there is an expression for $q$ of the type shown in Equation (1.4). In this case, it turns out that $q$ is indeed a member of the ideal (because $\left.q=2 x\left(x^{2} y+y x-2\right)+\left(x^{2} y+y x-2\right) x y-x^{2}(y x y-x+4 y)\right)$, but how would we answer the question in general? This problem is known as the Ideal Membership Problem and is stated as follows.

Definition 1.1.25 (The Ideal Membership Problem) Given an ideal $J$ and a polynomial $q$, does $q \in J$ ?

As we shall see shortly, the Ideal Membership Problem can be solved by dividing a polynomial with respect to a Gröbner Basis for the ideal $J$. But before we can discuss this, we must first introduce the notion of polynomial division, for which we require a fixed ordering on the monomials in any given polynomial.

### 1.2 Monomial Orderings

A monomial ordering is a bivariate function O which tells us which monomial is the larger of any two given monomials $m_{1}$ and $m_{2}$. We will use the convention that $\mathrm{O}\left(m_{1}, m_{2}\right)=1$ if and only if $m_{1}<m_{2}$, and $\mathrm{O}\left(m_{1}, m_{2}\right)=0$ if and only if $m_{1} \geqslant m_{2}$. We can use a monomial ordering to order an arbitrary polynomial $p$ by inducing an order on the terms of $p$ from the order on the monomials associated with the terms.

Definition 1.2.1 A monomial ordering O is admissible if the following conditions are satisfied.
(a) $1<m$ for all monomials $m \neq 1$.
(b) $m_{1}<m_{2} \Rightarrow m_{\ell} m_{1} m_{r}<m_{\ell} m_{2} m_{r}$ for all monomials ${ }^{1} m_{1}, m_{2}, m_{\ell}, m_{r}$.

By convention, a polynomial is always written in descending order (with respect to a given monomial ordering), so that the leading term of the polynomial (with associated leading coefficient and leading monomial) always comes first.

Remark 1.2.2 For an arbitrary polynomial $p$, we will use $\operatorname{LT}(p), \operatorname{LM}(p)$ and $\operatorname{LC}(p)$ to denote the leading term, leading monomial and leading coefficient of $p$ respectively.

### 1.2.1 Commutative Monomial Orderings

A monomial ordering usually requires an ordering on the variables in our chosen polynomial ring. Given such a ring $R\left[x_{1}, x_{2}, \ldots, x_{n}\right]$, we will assume this order to be $x_{1}>x_{2}>$ $\cdots>x_{n}$.

We shall now consider the most frequently used monomial orderings, where throughout $m_{1}$ and $m_{2}$ will denote arbitrary monomials (with associated multidegrees $e_{1}=\left(e_{1}^{1}, e_{1}^{2}, \ldots, e_{1}^{n}\right)$

[^0]and $\left.e_{2}=\left(e_{2}^{1}, e_{2}^{2}, \ldots, e_{2}^{n}\right)\right)$, and $\operatorname{deg}\left(m_{i}\right)$ will denote the total degree of the monomial $m_{i}$ (for example $\operatorname{deg}\left(x^{2} y z\right)=4$ ). All orderings considered will be admissible.

## The Lexicographical Ordering (Lex)

Define $m_{1}<m_{2}$ if $e_{1}^{i}<e_{2}^{i}$ for some $1 \leqslant i \leqslant n$ and $e_{1}^{j}=e_{2}^{j}$ for all $1 \leqslant j<i$. In words, $m_{1}<m_{2}$ if the first variable with different exponents in $m_{1}$ and $m_{2}$ has lower exponent in $m_{1}$.

## The Inverse Lexicographical Ordering (InvLex)

Define $m_{1}<m_{2}$ if $e_{1}^{i}<e_{2}^{i}$ for some $1 \leqslant i \leqslant n$ and $e_{1}^{j}=e_{2}^{j}$ for all $i<j \leqslant n$. In words, $m_{1}<m_{2}$ if the last variable with different exponents in $m_{1}$ and $m_{2}$ has lower exponent in $m_{1}$.

## The Degree Lexicographical Ordering (DegLex)

Define $m_{1}<m_{2}$ if $\operatorname{deg}\left(m_{1}\right)<\operatorname{deg}\left(m_{2}\right)$ or if $\operatorname{deg}\left(m_{1}\right)=\operatorname{deg}\left(m_{2}\right)$ and $m_{1}<m_{2}$ in the Lexicographic Ordering.

Remark 1.2.3 The DegLex ordering is also known as the TLex ordering ( T for total degree).

## The Degree Inverse Lexicographical Ordering (DegInvLex)

Define $m_{1}<m_{2}$ if $\operatorname{deg}\left(m_{1}\right)<\operatorname{deg}\left(m_{2}\right)$ or if $\operatorname{deg}\left(m_{1}\right)=\operatorname{deg}\left(m_{2}\right)$ and $m_{1}<m_{2}$ in the Inverse Lexicographical Ordering.

## The Degree Reverse Lexicographical Ordering (DegRevLex)

Define $m_{1}<m_{2}$ if $\operatorname{deg}\left(m_{1}\right)<\operatorname{deg}\left(m_{2}\right)$ or if $\operatorname{deg}\left(m_{1}\right)=\operatorname{deg}\left(m_{2}\right)$ and $m_{1}<m_{2}$ in the Reverse Lexicographical Ordering, where $m_{1}<m_{2}$ if the last variable with different exponents in $m_{1}$ and $m_{2}$ has higher exponent in $m_{1}\left(e_{1}^{i}>e_{2}^{i}\right.$ for some $1 \leqslant i \leqslant n$ and $e_{1}^{j}=e_{2}^{j}$ for all $i<j \leqslant n$ ).

Remark 1.2.4 On its own, the Reverse Lexicographical Ordering (RevLex) is not admissible, as $1>m$ for any monomial $m \neq 1$.

Example 1.2.5 With $x>y>z$, consider the monomials $m_{1}:=x^{2} y z ; m_{2}:=x^{2}$ and $m_{3}:=x y z^{2}$, with corresponding multidegrees $e_{1}=(2,1,1) ; e_{2}=(2,0,0)$ and $e_{3}=(1,1,2)$. The following table shows the order placed on the monomials by the various monomial orderings defined above. The final column shows the order induced on the polynomial $p:=m_{1}+m_{2}+m_{3}$ by the chosen monomial ordering.

| Monomial Ordering O | $\mathrm{O}\left(m_{1}, m_{2}\right)$ | $\mathrm{O}\left(m_{1}, m_{3}\right)$ | $\mathrm{O}\left(m_{2}, m_{3}\right)$ | $p$ |
| :--- | :---: | :---: | :---: | :---: |
| Lex | 0 | 0 | 0 | $x^{2} y z+x^{2}+x y z^{2}$ |
| InvLex | 0 | 1 | 1 | $x y z^{2}+x^{2} y z+x^{2}$ |
| DegLex | 0 | 0 | 1 | $x^{2} y z+x y z^{2}+x^{2}$ |
| DegInvLex | 0 | 1 | 1 | $x y z^{2}+x^{2} y z+x^{2}$ |
| DegRevLex | 0 | 0 | 1 | $x^{2} y z+x y z^{2}+x^{2}$ |

### 1.2.2 Noncommutative Monomial Orderings

In the noncommutative case, because we use words and not multidegrees to represent monomials, our definitions for the lexicographically based orderings will have to be adapted slightly. All other definitions and conventions will stay the same.

## The Lexicographic Ordering (Lex)

Define $m_{1}<m_{2}$ if, working left-to-right, the first (say $i$-th) letter on which $m_{1}$ and $m_{2}$ differ is such that the $i$-th letter of $m_{1}$ is lexicographically less than the $i$-th letter of $m_{2}$ in the variable ordering. Note: this ordering is not admissible (counterexample: if $x>y$ is the variable ordering, then $x<x y$ but $\left.x^{2}>x y x\right)$.

Remark 1.2.6 When comparing two monomials $m_{1}$ and $m_{2}$ such that $m_{1}$ is a proper prefix of $m_{2}$ (for example $m_{1}:=x$ and $m_{2}:=x y$ as in the above counterexample), a problem arises with the above definition in that we eventually run out of letters in the shorter word to compare with (in the example, having seen that the first letter of both monomials match, what do we compare the second letter of $m_{2}$ with?). One answer is to introduce a padding symbol $\$$ to pad $m_{1}$ on the right to make sure it is the same length as $m_{2}$, with the convention that any letter is greater than the padding symbol (so that $m_{1}<m_{2}$ ). The padding symbol will not explicitly appear anywhere in the remainder of this thesis, but we will bear in mind that it can be introduced to deal with situations where prefixes and suffixes of monomials are involved.

Remark 1.2.7 The lexicographic ordering is also known as the dictionary ordering since the words in a dictionary (such as the Oxford English Dictionary) are ordered using the lexicographic ordering with variable (or alphabetical) ordering $a<b<c<\cdots$. Note however that while a dictionary orders words in increasing order, we will write polynomials in decreasing order.

## The Inverse Lexicographical Ordering (InvLex)

Define $m_{1}<m_{2}$ if, working left-to-right, the first (say $i$-th) letter on which $m_{1}$ and $m_{2}$ differ is such that the $i$-th letter of $m_{1}$ is lexicographically greater than the $i$-th letter of $m_{2}$. Note: this ordering (like Lex) is not admissible (counterexample: if $x>y$ is the variable ordering, then $x y<x$ but $x y x>x^{2}$ ).

## The Degree Reverse Lexicographical Ordering (DegRevLex)

Define $m_{1}<m_{2}$ if $\operatorname{deg}\left(m_{1}\right)<\operatorname{deg}\left(m_{2}\right)$ or if $\operatorname{deg}\left(m_{1}\right)=\operatorname{deg}\left(m_{2}\right)$ and $m_{1}<m_{2}$ in the Reverse Lexicographical Ordering, where $m_{1}<m_{2}$ if, working in reverse, or from right-to-left, the first (say $i$-th) letter on which $m_{1}$ and $m_{2}$ differ is such that the $i$-th letter of $m_{1}$ is lexicographically greater than the $i$-th letter of $m_{2}$.

Example 1.2.8 With $x>y>z$, consider the noncommutative monomials $m_{1}:=z x y x$; $m_{2}:=x z x$ and $m_{3}:=y^{2} z x$. The following table shows the order placed on the monomials by various noncommutative monomial orderings. As before, the final column shows the order induced on the polynomial $p:=m_{1}+m_{2}+m_{3}$ by the chosen monomial ordering.

| Monomial Ordering O | $\mathrm{O}\left(m_{1}, m_{2}\right)$ | $\mathrm{O}\left(m_{1}, m_{3}\right)$ | $\mathrm{O}\left(m_{2}, m_{3}\right)$ | $p$ |
| :--- | :---: | :---: | :---: | :---: |
| Lex | 1 | 1 | 0 | $x z x+y^{2} z x+z x y x$ |
| InvLex | 0 | 0 | 1 | $z x y x+y^{2} z x+x z x$ |
| DegLex | 0 | 1 | 1 | $y^{2} z x+z x y x+x z x$ |
| DegInvLex | 0 | 0 | 1 | $z x y x+y^{2} z x+x z x$ |
| DegRevLex | 0 | 1 | 1 | $y^{2} z x+z x y x+x z x$ |

### 1.2.3 Polynomial Division

Definition 1.2.9 Let $\mathcal{R}$ be a polynomial ring, and let $O$ be an arbitrary admissible monomial ordering. Given two nonzero polynomials $p_{1}, p_{2} \in \mathcal{R}$, we say that $p_{1}$ divides
$p_{2}$ (written $p_{1} \mid p_{2}$ ) if the lead monomial of $p_{1}$ divides some monomial $m$ (with coefficient c) in $p_{2}$. For a commutative polynomial ring, this means that $m=\mathrm{LM}\left(p_{1}\right) m^{\prime}$ for some monomial $m^{\prime}$; for a noncommutative polynomial ring, this means that $m=m_{\ell} \mathrm{LM}\left(p_{1}\right) m_{r}$ for some monomials $m_{\ell}$ and $m_{r}\left(\operatorname{LM}\left(p_{1}\right)\right.$ is a subword of $\left.m\right)$.

To perform the division, we take away an appropriate multiple of $p_{1}$ from $p_{2}$ in order to cancel off $\operatorname{LT}\left(p_{1}\right)$ with the term involving $m$ in $p_{2}$. In the commutative case, we do

$$
p_{2}-\left(c \mathrm{LC}\left(p_{1}\right)^{-1}\right) p_{1} m^{\prime} ;
$$

in the noncommutative case, we do

$$
p_{2}-\left(c \mathrm{LC}\left(p_{1}\right)^{-1}\right) m_{\ell} p_{1} m_{r}
$$

It is clear that the coefficient rings of our polynomial rings have to be division rings in order for the above expressions to be valid, and so we make the following assumption about the polynomial rings we will encounter in the remainder of this thesis.

Remark 1.2.10 From now on, all coefficient rings of polynomial rings will be fields unless otherwise stated.

Example 1.2.11 Let $p_{1}:=5 z^{2} x+2 y^{2}+x+4$ and $p_{2}:=3 x y x z^{2} x^{3}+2 x^{2}$ be two DegLex ordered polynomials over the noncommutative polynomial ring $\mathbb{Q}\langle x, y, z\rangle$. Because $\operatorname{LM}\left(p_{2}\right)=x y x\left(z^{2} x\right) x^{2}$, it is clear that $p_{1} \mid p_{2}$, with the quotient and the remainder of the division being

$$
q:=\left(\frac{3}{5}\right) x y x\left(5 z^{2} x+2 y^{2}+x+4\right) x^{2}
$$

and

$$
\begin{aligned}
r & :=3 x y x z^{2} x^{3}+2 x^{2}-\left(\frac{3}{5}\right) x y x\left(5 z^{2} x+2 y^{2}+x+4\right) x^{2} \\
& =3 x y x z^{2} x^{3}+2 x^{2}-3 x y x z^{2} x^{3}-\left(\frac{6}{5}\right) x y x y^{2} x^{2}-\left(\frac{3}{5}\right) x y x^{4}-\left(\frac{12}{5}\right) x y x^{3} \\
& =-\left(\frac{6}{5}\right) x y x y^{2} x^{2}-\left(\frac{3}{5}\right) x y x^{4}-\left(\frac{12}{5}\right) x y x^{3}+2 x^{2}
\end{aligned}
$$

respectively.

Now that we know how to divide one polynomial by another, what does it mean for a polynomial to be divided by a set of polynomials?

Definition 1.2.12 Let $\mathcal{R}$ be a polynomial ring, and let $O$ be an arbitrary admissible
monomial ordering. Given a nonzero polynomial $p \in \mathcal{R}$ and a set of nonzero polynomials $P=\left\{p_{1}, p_{2}, \ldots, p_{m}\right\}$, with $p_{i} \in \mathcal{R}$ for all $1 \leqslant i \leqslant m$, we divide $p$ by $P$ by working through $p$ term by term, testing to see if each term is divisible by any of the $p_{i}$ in turn. We recursively divide the remainder of each division using the same method until no more divisions are possible, in which case the remainder is either 0 or is irreducible.

Algorithms to divide a polynomial $p$ by a set of polynomials $P$ in the commutative and noncommutative cases are given below as Algorithms 1 and 2 respectively. Note that they take advantage of the fact that if the first $N$ terms of a polynomial $q$ are irreducible with respect to $P$, then the first $N$ terms of any reduction of $q$ will also be irreducible with respect to $P$.

```
Algorithm 1 The Commutative Division Algorithm
Input: A nonzero polynomial \(p\) and a set of nonzero polynomials \(P=\left\{p_{1}, \ldots, p_{m}\right\}\) over
    a polynomial ring \(R\left[x_{1}, \ldots x_{n}\right]\); an admissible monomial ordering O .
Output: \(\operatorname{Rem}(p, P):=r\), the remainder of \(p\) with respect to \(P\).
```

```
\(r=0 ;\)
```

$r=0 ;$
while $(p \neq 0)$ do
while $(p \neq 0)$ do
$u=\operatorname{LM}(p) ; c=\operatorname{LC}(p) ; j=1$; found $=$ false;
$u=\operatorname{LM}(p) ; c=\operatorname{LC}(p) ; j=1$; found $=$ false;
while $(j \leqslant m)$ and (found $==$ false) do
while $(j \leqslant m)$ and (found $==$ false) do
if $\left(\operatorname{LM}\left(p_{j}\right) \mid u\right)$ then
if $\left(\operatorname{LM}\left(p_{j}\right) \mid u\right)$ then
found $=$ true; $u^{\prime}=u / \mathrm{LM}\left(p_{j}\right) ; p=p-\left(c \mathrm{LC}\left(p_{j}\right)^{-1}\right) p_{j} u^{\prime}$;
found $=$ true; $u^{\prime}=u / \mathrm{LM}\left(p_{j}\right) ; p=p-\left(c \mathrm{LC}\left(p_{j}\right)^{-1}\right) p_{j} u^{\prime}$;
else
else
$j=j+1 ;$
$j=j+1 ;$
end if
end if
end while
end while
if (found $==$ false) then
if (found $==$ false) then
$r=r+\operatorname{LT}(p) ; p=p-\operatorname{LT}(p) ;$
$r=r+\operatorname{LT}(p) ; p=p-\operatorname{LT}(p) ;$
end if
end if
end while
end while
return $r$;

```
return \(r\);
```

Remark 1.2.13 All algorithms in this thesis use the conventions that ' $=$ ' denotes an assignment and ' $==$ ' denotes a test.

```
Algorithm 2 The Noncommutative Division Algorithm
\(R\left\langle x_{1}, \ldots, x_{n}\right\rangle\), we apply Algorithm 1 with the following changes. mutative polynomial ring \(R\left\langle x_{1}, \ldots, x_{n}\right\rangle\).
(b) Change the first if condition to read
```

```
if \(\left(\operatorname{LM}\left(p_{j}\right) \mid u\right)\) then
```

if $\left(\operatorname{LM}\left(p_{j}\right) \mid u\right)$ then
found $=$ true;
found $=$ true;
choose $u_{\ell}$ and $u_{r}$ such that $u=u_{\ell} \mathrm{LM}\left(p_{j}\right) u_{r}$;
choose $u_{\ell}$ and $u_{r}$ such that $u=u_{\ell} \mathrm{LM}\left(p_{j}\right) u_{r}$;
$p=p-\left(c \mathrm{LC}\left(p_{j}\right)^{-1}\right) u_{\ell} p_{j} u_{r} ;$
$p=p-\left(c \mathrm{LC}\left(p_{j}\right)^{-1}\right) u_{\ell} p_{j} u_{r} ;$
else
else
$j=j+1 ;$
$j=j+1 ;$
end if

```
    end if
```

To divide a nonzero polynomial $p$ with respect to a set of nonzero polynomials $P=$
$\left\{p_{1}, \ldots, p_{m}\right\}$, where $p$ and the $p_{i}$ are elements of a noncommutative polynomial ring
(a) In the inputs, replace the commutative polynomial ring $R\left[x_{1}, \ldots x_{n}\right]$ by the noncom-

Remark 1.2.14 In Algorithm 2, if there are several candidates for $u_{\ell}$ (and therefore for $u_{r}$ ) in the line 'choose $u_{\ell}$ and $u_{r}$ such that $u=u_{\ell} \mathrm{LM}\left(p_{j}\right) u_{r}$ ', the convention in this thesis will be to choose the $u_{\ell}$ with the smallest degree.

Example 1.2.15 To demonstrate that the process of dividing a polynomial by a set of polynomials does not necessarily give a unique result, consider the polynomial $p:=x y z+x$ and the set of polynomials $P:=\left\{p_{1}, p_{2}\right\}=\{x y-z, y z+2 x+z\}$, all polynomials being ordered by DegLex and originating from the polynomial ring $\mathbb{Q}[x, y, z]$. If we choose to divide $p$ by $p_{1}$ to begin with, we see that $p$ reduces to $x y z+x-(x y-z) z=z^{2}+x$, which is irreducible. But if we choose to divide $p$ by $p_{2}$ to begin with, we see that $p$ reduces to $x y z+x-(y z+2 x+z) x=-2 x^{2}-x z+x$, which is again irreducible. This gives rise to the question of which answer (if any!) is the correct one here? In Chapter 2, we will discover that one way of obtaining a unique answer to this question will be to calculate a Gröbner Basis for the dividing set $P$.

Definition 1.2.16 In order to describe how one polynomial is obtained from another through the process of division, we introduce the following notation.
(a) If the polynomial $r$ is obtained by dividing a polynomial $p$ by a polynomial $q$, then we will use the notation $p \rightarrow r$ or $p \rightarrow_{q} r$ (with the latter notation used if we wish to
show how $r$ is obtained from $p$ ).
(b) If the polynomial $r$ is obtained by dividing a polynomial $p$ by a sequence of polynomials $q_{1}, q_{2}, \ldots, q_{\alpha}$, then we will use the notation $p \xrightarrow{*} r$.
(c) If the polynomial $r$ is obtained by dividing a polynomial $p$ by a set of polynomials $Q$, then we will use the notation $p \rightarrow Q r$.

## Chapter 2

## Commutative Gröbner Bases

Given a basis $F$ generating an ideal $J$, the central idea in Gröbner Basis theory is to use $F$ to find a basis $G$ for $J$ with the property that the remainder of the division of any polynomial by $G$ is unique. Such a basis is known as a Gröbner Basis.

In particular, if a polynomial $p$ is a member of the ideal $J$, then the remainder of the division of $p$ by a Gröbner Basis $G$ for $J$ is always zero. This gives us a way to solve the Ideal Membership Problem for $J$ - if the remainder of the division of a polynomial $p$ by $G$ is zero, then $p \in J$ (otherwise $p \notin J$ ).

### 2.1 S-polynomials

How do we determine whether or not an arbitrary basis $F$ generating an ideal $J$ is a Gröbner Basis? Using the informal definition shown above, in order to show that a basis is not a Gröbner Basis, it is sufficient to find a polynomial $p$ whose remainder on division by $F$ is non-unique. Let us now construct an example in which this is the case, and let us analyse what can to be done to eliminate the non-uniqueness of the remainder.

Let $p_{1}=a_{1}+a_{2}+\cdots+a_{\alpha} ; p_{2}=b_{1}+b_{2}+\cdots+b_{\beta}$ and $p_{3}=c_{1}+c_{2}+\cdots+c_{\gamma}$ be three polynomials ordered with respect to some fixed admissible monomial ordering $O$ (the $a_{i}$, $b_{j}$ and $c_{k}$ are all nontrivial terms). Assume that $p_{1} \mid p_{3}$ and $p_{2} \mid p_{3}$, so that we are able to take away from $p_{3}$ multiples $s$ and $t$ of $p_{1}$ and $p_{2}$ respectively to obtain remainders $r_{1}$
and $r_{2}$.

$$
\begin{aligned}
r_{1} & =p_{3}-s p_{1} \\
& =c_{1}+c_{2}+\cdots+c_{\gamma}-s\left(a_{1}+a_{2}+\cdots+a_{\alpha}\right) \\
& =c_{2}+\cdots+c_{\gamma}-s a_{2}-\cdots-s a_{\alpha} ; \\
r_{2} & =p_{3}-t p_{2} \\
& =c_{2}+\cdots+c_{\gamma}-t b_{2}-\cdots-t b_{\beta} .
\end{aligned}
$$

If we assume that $r_{1}$ and $r_{2}$ are irreducible and that $r_{1} \neq r_{2}$, it is clear that the remainder of the division of the polynomial $p_{3}$ by the set of polynomials $P=\left\{p_{1}, p_{2}\right\}$ is non-unique, from which we deduce that $P$ is not a Gröbner Basis for the ideal that it generates. We must therefore change $P$ in some way in order for it to become a Gröbner Basis, but what changes are required and indeed allowed?

Consider that we want to add a polynomial to $P$. To avoid changing the ideal that is being generated by $P$, any polynomial added to $P$ must be a member of the ideal. It is clear that $r_{1}$ and $r_{2}$ are members of the ideal, as is the polynomial $p_{4}=r_{2}-r_{1}=-t p_{2}+s p_{1}$. Consider that we add $p_{4}$ to $P$, so that $P$ becomes the set

$$
\left\{a_{1}+a_{2}+\cdots+a_{\alpha}, b_{1}+b_{2}+\cdots+b_{\beta},-t b_{2}-t b_{3}-\cdots-t b_{\beta}+s a_{2}+s a_{3}+\cdots+s a_{\alpha}\right\} .
$$

If we now divide the polynomial $p_{3}$ by the enlarged set $P$, to begin with (as before) we can either divide $p_{3}$ by $p_{1}$ or $p_{2}$ to obtain remainders $r_{1}$ or $r_{2}$. Here however, if we assume (without loss of generality ${ }^{1}$ ) that $\operatorname{LT}\left(p_{4}\right)=-t b_{2}$, we can now divide $r_{2}$ by $p_{4}$ to obtain a new remainder

$$
\begin{aligned}
r_{3} & =r_{2}-p_{4} \\
& =c_{2}+\cdots+c_{\gamma}-t b_{2}-\cdots-t b_{\beta}-\left(-t b_{2}-t b_{3}-\cdots-t b_{\beta}+s a_{2}+s a_{3}+\cdots+s a_{\alpha}\right) \\
& =c_{2}+\cdots+c_{\gamma}-s a_{2}-\cdots-s a_{\alpha} \\
& =r_{1} .
\end{aligned}
$$

It follows that by adding $p_{4}$ to $P$, we have ensured that the remainder of the division of $p_{3}$ by $P$ is unique ${ }^{2}$ no matter which of the polynomials $p_{1}$ and $p_{2}$ we choose to divide

[^1]$p_{3}$ by first. This solves our original problem of non-unique remainders in this restricted situation.

At first glance, the polynomial added to $P$ to solve this problem is dependent upon the polynomial $p_{3}$. The reason for saying this is that the polynomial added to $P$ has the form $p_{4}=s p_{1}-t p_{2}$, where $s$ and $t$ are terms chosen to multiply the polynomials $p_{1}$ and $p_{2}$ so that the lead terms of $s p_{1}$ and $t p_{2}$ equal $\operatorname{LT}\left(p_{3}\right)$ (in fact $s=\frac{\operatorname{LT}\left(p_{3}\right)}{\operatorname{LT}\left(p_{1}\right)}$ and $t=\frac{\operatorname{LT}\left(p_{3}\right)}{\operatorname{LT}\left(p_{2}\right)}$ ).

However, by definition, $\operatorname{LM}\left(p_{3}\right)$ is a common multiple of $\operatorname{LM}\left(p_{1}\right)$ and $\operatorname{LM}\left(p_{2}\right)$. Because all such common multiples are multiples of the least common multiple of $\operatorname{LM}\left(p_{1}\right)$ and $\operatorname{LM}\left(p_{2}\right)$ (so that $\operatorname{LM}\left(p_{3}\right)=\mu\left(\operatorname{lcm}\left(\operatorname{LM}\left(p_{1}\right), \operatorname{LM}\left(p_{2}\right)\right)\right)$ for some monomial $\mu$ ), it follows that we can rewrite $p_{4}$ as

$$
p_{4}=\operatorname{LC}\left(p_{3}\right) \mu\left(\frac{\operatorname{lcm}\left(\operatorname{LM}\left(p_{1}\right), \operatorname{LM}\left(p_{2}\right)\right)}{\operatorname{LT}\left(p_{1}\right)} p_{1}-\frac{\operatorname{lcm}\left(\operatorname{LM}\left(p_{1}\right), \operatorname{LM}\left(p_{2}\right)\right)}{\operatorname{LT}\left(p_{2}\right)} p_{2}\right) .
$$

Consider now that we add the polynomial $p_{5}=\frac{p_{4}}{\mathrm{LC}\left(p_{3}\right) \mu}$ to $P$ instead of adding $p_{4}$ to $P$. It follows that even though this polynomial does not depend on the polynomial $p_{3}$, we can still obtain a unique remainder when dividing $p_{3}$ by $p_{1}$ and $p_{2}$, because we can do $r_{3}=r_{2}-\mathrm{LC}\left(p_{3}\right) \mu p_{5}$. Moreover, the polynomial $p_{5}$ solves the problem of non-unique remainders for any polynomial $p_{3}$ that is divisible by both $p_{1}$ and $p_{2}$ (all that changes is the multiple of $p_{5}$ used in the reduction of $r_{2}$ ); we call such a polynomial an $S$-polynomiaß ${ }^{3}$ for $p_{1}$ and $p_{2}$.

Definition 2.1.1 The $S$-polynomial of two distinct polynomials $p_{1}$ and $p_{2}$ is given by the expression

$$
\mathrm{S}-\operatorname{pol}\left(p_{1}, p_{2}\right)=\frac{\operatorname{lcm}\left(\mathrm{LM}\left(p_{1}\right), \mathrm{LM}\left(p_{2}\right)\right)}{\operatorname{LT}\left(p_{1}\right)} p_{1}-\frac{\operatorname{lcm}\left(\operatorname{LM}\left(p_{1}\right), \mathrm{LM}\left(p_{2}\right)\right)}{\operatorname{LT}\left(p_{2}\right)} p_{2} .
$$

Remark 2.1.2 The terms $\frac{\operatorname{lcm}\left(\operatorname{LM}\left(p_{1}\right) \operatorname{LM}\left(p_{2}\right)\right)}{\operatorname{LT}\left(p_{1}\right)}$ and $\frac{\operatorname{lcm}\left(\operatorname{LM}\left(p_{1}\right), \operatorname{LM}\left(p_{2}\right)\right)}{\operatorname{LT}\left(p_{2}\right)}$ can be thought of as the terms used to multiply the polynomials $p_{1}$ and $p_{2}$ so that the lead monomials of the multiples are equal to the monomial $\operatorname{lcm}\left(\operatorname{LM}\left(p_{1}\right), \operatorname{LM}\left(p_{2}\right)\right)$.

Let us now illustrate how adding an S-polynomial to a basis solves the problem of nonunique remainders in a particular example.

[^2]Example 2.1.3 Recall that in Example 1.2.15 we showed how dividing the polynomial $p:=x y z+x$ by the two polynomials in the set $P:=\left\{p_{1}, p_{2}\right\}=\{x y-z, y z+2 x+z\}$ gave two different remainders, $r_{1}:=z^{2}+x$ and $r_{2}:=-2 x^{2}-x z+x$ respectively. Consider now that we add $S-\operatorname{pol}\left(p_{1}, p_{2}\right)$ to $P$, where

$$
\begin{aligned}
\operatorname{S-pol}\left(p_{1}, p_{2}\right) & =\frac{x y z}{x y}(x y-z)-\frac{x y z}{y z}(y z+2 x+z) \\
& =\left(x y z-z^{2}\right)-\left(x y z+2 x^{2}+x z\right) \\
& =-2 x^{2}-x z-z^{2} .
\end{aligned}
$$

Dividing $p$ by the enlarged set, if we choose to divide $p$ by $p_{1}$ to begin with, we see that $p$ reduces (as before) to give $x y z+x-(x y-z) z=z^{2}+x$, which is irreducible. Similarly, dividing $p$ by $p_{2}$ to begin with, we obtain the remainder $x y z+x-(y z+2 x+z) x=$ $-2 x^{2}-x z+x$. However, whereas before this remainder was irreducible, now we can reduce it by the S-polynomial to give $-2 x^{2}-x z+x-\left(-2 x^{2}-x z-z^{2}\right)=z^{2}+x$, which is equal to the first remainder.

Let us now formally define a Gröbner Basis in terms of S-polynomials, noting that there are many other equivalent definitions (see for example [7], page 206).

Definition 2.1.4 Let $G=\left\{g_{1}, \ldots, g_{m}\right\}$ be a basis for an ideal $J$ over a commutative polynomial ring $\mathcal{R}=R\left[x_{1}, \ldots, x_{n}\right]$. If all the $S$-polynomials involving members of $G$ reduce to zero using $G\left(\mathrm{~S}-\mathrm{pol}\left(g_{i}, g_{j}\right) \rightarrow_{G} 0\right.$ for all $\left.i \neq j\right)$, then $G$ is a Gröbner Basis for $J$.

Theorem 2.1.5 Given any polynomial $p$ over a polynomial ring $\mathcal{R}=R\left[x_{1}, \ldots, x_{n}\right]$, the remainder of the division of $p$ by a basis $G$ for an ideal $J$ in $\mathcal{R}$ is unique if and only if $G$ is a Gröbner Basis.

Proof: $\quad(\Rightarrow)$ By Newman's Lemma (cf. [7], page 176), showing that the remainder of the division of $p$ by $G$ is unique is equivalent to showing that the division process is locally confluent, that is if there are polynomials $f, f_{1}, f_{2} \in \mathcal{R}$ with $f_{1}=f-t_{1} g_{1}$ and $f_{2}=f-t_{2} g_{2}$ for terms $t_{1}, t_{2}$ and $g_{1}, g_{2} \in G$, then there exists a polynomial $f_{3} \in \mathcal{R}$ such that both $f_{1}$ and $f_{2}$ reduce to $f_{3}$. By the Translation Lemma (cf. [7], page 200), this in turn is equivalent to showing that the polynomial $f_{2}-f_{1}=t_{1} g_{1}-t_{2} g_{2}$ reduces to zero, which is what we shall now do.

There are two cases to deal with, $\operatorname{LT}\left(t_{1} g_{1}\right) \neq \mathrm{LT}\left(t_{2} g_{2}\right)$ and $\operatorname{LT}\left(t_{1} g_{1}\right)=\mathrm{LT}\left(t_{2} g_{2}\right)$. In the first case, notice that the remainders $f_{1}$ and $f_{2}$ are obtained by cancelling off different terms of the original $f$ (the reductions of $f$ are disjoint), so it is possible, assuming (without loss of generality) that $\mathrm{LT}\left(t_{1} g_{1}\right)>\mathrm{LT}\left(t_{2} g_{2}\right)$, to directly reduce the polynomial $f_{2}-f_{1}=t_{1} g_{1}-t_{2} g_{2}$ in the following manner: $t_{1} g_{1}-t_{2} g_{2} \rightarrow_{g_{1}}-t_{2} g_{2} \rightarrow g_{2}$. In the second case, the reductions of $f$ are not disjoint (as the same term $t$ from $f$ is cancelled off during both reductions), so that the term $t$ does not appear in the polynomial $t_{1} g_{1}-$ $t_{2} g_{2}$. However, the term $t$ is a common multiple of $\operatorname{LT}\left(t_{1} g_{1}\right)$ and $\operatorname{LT}\left(t_{2} g_{2}\right)$, and thus the polynomial $t_{1} g_{1}-t_{2} g_{2}$ is a multiple of the S-polynomial S-pol $\left(g_{1}, g_{2}\right)$, say

$$
t_{1} g_{1}-t_{2} g_{2}=\mu\left(\mathrm{S}-\operatorname{pol}\left(g_{1}, g_{2}\right)\right)
$$

for some term $\mu$. Because $G$ is a Gröbner Basis, the S-polynomial S-pol $\left(g_{1}, g_{2}\right)$ reduces to zero, and hence by extension the polynomial $t_{1} g_{1}-t_{2} g_{2}$ also reduces to zero.
$(\Leftarrow)$ As all S-polynomials are members of the ideal $J$, to complete the proof it is sufficient to show that there is always a reduction path of an arbitrary member of the ideal that leads to a zero remainder (the uniqueness of remainders will then imply that members of the ideal will always reduce to zero). Let $f \in J=\langle G\rangle$. Then, by definition, there exist $g_{i} \in G$ and $f_{i} \in \mathcal{R}$ (where $1 \leqslant i \leqslant j$ ) such that

$$
f=\sum_{i=1}^{j} f_{i} g_{i}
$$

We proceed by induction on $j$. If $j=1$, then $f=f_{1} g_{1}$, and it is clear that we can use $g_{1}$ to reduce $f$ to give a zero remainder $\left(f \rightarrow f-f_{1} g_{1}=0\right)$. Assume that the result is true for $j=k$, and let us look at the case $j=k+1$, so that

$$
f=\left(\sum_{i=1}^{k} f_{i} g_{i}\right)+f_{k+1} g_{k+1}
$$

By the inductive hypothesis, $\sum_{i=1}^{k} f_{i} g_{i}$ is a member of the ideal that reduces to zero. The polynomial $f$ therefore reduces to the polynomial $f^{\prime}:=f_{k+1} g_{k+1}$, and we can now use $g_{k+1}$ to reduce $f^{\prime}$ to give a zero remainder ( $f^{\prime} \rightarrow f^{\prime}-f_{k+1} g_{k+1}=0$ ).

We are now in a position to be able to define an algorithm to compute a Gröbner Basis. However, to be able to prove that this algorithm always terminates, we must first prove
a result stating that all ideals over commutative polynomial rings are finitely generated. This proof takes place in two stages - first for monomial ideals (Dickson's Lemma) and then for polynomial ideals (Hilbert's Basis Theorem).

### 2.2 Dickson's Lemma and Hilbert's Basis Theorem

Definition 2.2.1 A monomial ideal is an ideal generated by a set of monomials.
Remark 2.2.2 Any polynomial $p$ that is a member of a monomial ideal is a sum of terms $p=\sum_{i} t_{i}$, where each $t_{i}$ is a member of the monomial ideal.

Lemma 2.2.3 (Dickson's Lemma) Every monomial ideal over the polynomial ring $\mathcal{R}=$ $R\left[x_{1}, \ldots, x_{n}\right]$ is finitely generated.

Proof (cf. [22], page 47): Let $J$ be a monomial ideal over $\mathcal{R}$ generated by a set of monomials $S$. We proceed by induction on $n$, our goal being to show that $S$ always has a finite subset $T$ generating $J$. For $n=1$, notice that all elements of $S$ will be of the form $x_{1}^{j}$ for some $j \geqslant 0$. Let $T$ be the singleton set containing the member of $S$ with the lowest degree (that is the $x_{1}^{j}$ with the lowest value of $j$ ). Clearly $T$ is finite, and because any element of $S$ is a multiple of the chosen $x_{1}^{j}$, it is also clear that $T$ generates the same ideal as $S$.

For the inductive step, assume that all monomial ideals over the polynomial ring $\mathcal{R}^{\prime}=$ $R\left[x_{1}, \ldots, x_{n-1}\right]$ are finitely generated. Let $C_{0} \subseteq C_{1} \subseteq C_{2} \subseteq \cdots$ be an ascending chain of monomial ideals over $\mathcal{R}^{\prime}$, where ${ }^{4}$

$$
C_{j}=\left\langle S_{j}\right\rangle \cap \mathcal{R}^{\prime}, S_{j}=\left\{\left.\frac{s}{\operatorname{gcd}\left(s, x_{n}^{j}\right)} \right\rvert\, s \in S\right\} .
$$

Let the monomial $m$ be an arbitrary member of the ideal $J$, expressed as $m=m^{\prime} x_{n}^{k}$, where $m^{\prime} \in \mathcal{R}^{\prime}$ and $k \geqslant 0$. By definition, $m^{\prime} \in C_{k}$, and so $m \in x_{n}^{k} C_{k}$. By the inductive hypothesis, each $C_{k}$ is finitely generated by a set $T_{k}$, and so $m \in x_{n}^{k}\left\langle T_{k}\right\rangle$. From this we can deduce that

$$
T=T_{0} \cup x_{n} T_{1} \cup x_{n}^{2} T_{2} \cup \cdots
$$

is a generating set for $J$.

[^3]Consider the ideal $C=\cup C_{j}$ for $j \geqslant 0$. This is another monomial ideal over $\mathcal{R}^{\prime}$, and so by the inductive hypothesis is finitely generated. It follows that the chain must stop as soon as the generators of $C$ are contained in some $C_{r}$, so that $C_{r}=C_{r+1}=\cdots$ (and hence $\left.T_{r}=T_{r+1}=\cdots\right)$. It follows that $T_{0} \cup x_{n} T_{1} \cup x_{n}^{2} T_{2} \cup \cdots \cup x_{n}^{r} T_{r}$ is a finite subset of $S$ generating $J$.

Example 2.2.4 Let $S=\left\{y^{4}, x y^{4}, x^{2} y^{3}, x^{3} y^{3}, x^{4} y, x^{k}\right\}$ be an infinite set of monomials generating an ideal $J$ over the polynomial ring $\mathbb{Q}[x, y]$, where $k$ is an integer such that $k \geqslant 5$. We can visualise $J$ by using the following monomial lattice, where a point $(a, b)$ in the lattice (for non-negative integers $a, b$ ) corresponds to the monomial $x^{a} y^{b}$, and the shaded region contains all monomials which are reducible by some member of $S$ (and hence belong to $J$ ).


To show that $J$ can be finitely generated, we need to construct the set $T$ as described in the proof of Dickson's Lemma. The first step in doing this is to construct the sequence of sets $S_{j}=\left\{\left.\frac{s}{\operatorname{gcd}\left(s, y^{j}\right)} \right\rvert\, s \in S\right\}$ for all $j \geqslant 0$.

$$
\begin{aligned}
S_{0} & =\left\{y^{4}, x y^{4}, x^{2} y^{3}, x^{3} y^{3}, x^{4} y, x^{k}\right\}=S \\
S_{1} & =\left\{y^{3}, x y^{3}, x^{2} y^{2}, x^{3} y^{2}, x^{4}, x^{k}\right\} \\
S_{2} & =\left\{y^{2}, x y^{2}, x^{2} y, x^{3} y, x^{4}, x^{k}\right\} \\
S_{3} & =\left\{y, x y, x^{2}, x^{3}, x^{4}, x^{k}\right\} \\
S_{4} & =\left\{y^{0}=1, x, x^{2}, x^{3}, x^{4}, x^{k}\right\} \\
S_{j+1} & =S_{j} \text { for all } j+1 \geqslant 5 .
\end{aligned}
$$

Each set $S_{j}$ gives rise to an ideal $C_{j}$ consisting of all monomials $m \in\left\langle S_{j}\right\rangle$ of the form $m=x^{i}$ for some $i \geqslant 0$. Because each of these ideals is an ideal over the polynomial ring $\mathbb{Q}[x]$, we can use an inductive hypothesis to give us a finite generating set $T_{j}$ for each $C_{j}$.

In this case, the first paragraph of the proof of Dickson's Lemma tells us how to apply the inductive hypothesis - each set $T_{j}$ is formed by choosing the monomial $m \in S_{j}$ of lowest degree such that $m=x^{i}$ for some $i \geqslant 0$.

$$
\begin{aligned}
T_{0} & =\left\{x^{5}\right\} \\
T_{1} & =\left\{x^{4}\right\} \\
T_{2} & =\left\{x^{4}\right\} \\
T_{3} & =\left\{x^{2}\right\} \\
T_{4} & =\left\{x^{0}=1\right\} \\
T_{j+1} & =T_{j} \text { for all } k+1 \geqslant 5 .
\end{aligned}
$$

We can now deduce that

$$
T=\left\{x^{5}\right\} \cup\left\{x^{4} y\right\} \cup\left\{x^{4} y^{2}\right\} \cup\left\{x^{2} y^{3}\right\} \cup\left\{y^{4}\right\} \cup\left\{y^{5}\right\} \cup \cdots
$$

is a generating set for $J$. Further, because $T_{j+1}=T_{j}$ for all $k+1 \geqslant 5$, we can also deduce that the set

$$
T^{\prime}=\left\{x^{5}, x^{4} y, x^{4} y^{2}, x^{2} y^{3}, y^{4}\right\}
$$

is a finite generating set for $J$ (a fact that can be verified by drawing a monomial lattice for $T^{\prime}$ and comparing it with the above monomial lattice for the set $S$ ).

Theorem 2.2.5 (Hilbert's Basis Theorem) Every ideal $J$ over a polynomial ring $\mathcal{R}=$ $R\left[x_{1}, \ldots, x_{n}\right]$ is finitely generated.

Proof: Let $O$ be a fixed arbitrary admissible monomial ordering, and define $\operatorname{LM}(J)=$ $\langle\mathrm{LM}(p) \mid p \in J\rangle$. Because $\mathrm{LM}(J)$ is a monomial ideal, by Dickson's Lemma it is finitely generated, say by the set of monomials $M=\left\{m_{1}, \ldots, m_{r}\right\}$. By definition, each $m_{i} \in M$ (for $1 \leqslant i \leqslant r$ ) has a corresponding $p_{i} \in J$ such that $\operatorname{LM}\left(p_{i}\right)=m_{i}$. We claim that $P=\left\{p_{1}, \ldots, p_{r}\right\}$ is a generating set for $J$. To prove the claim, notice that $\langle P\rangle \subseteq J$ so that $f \in\langle P\rangle \Rightarrow f \in J$. Conversely, given a polynomial $f \in J$, we know that $\mathrm{LM}(f) \in\langle M\rangle$ so that $\mathrm{LM}(f)=\alpha m_{j}$ for some monomial $\alpha$ and some $1 \leqslant j \leqslant r$. From this, if we define $\alpha^{\prime}=\frac{\mathrm{LC}(f)}{\mathrm{LC}\left(p_{j}\right)} \alpha$, we can deduce that $\mathrm{LM}\left(f-\alpha^{\prime} p_{j}\right)<\operatorname{LM}(f)$. Since $f-\alpha^{\prime} p_{j} \in J$, and because of the admissibility of $O$, by recursion on $f-\alpha^{\prime} p_{j}$ (define $f_{k+1}=f_{k}-\alpha_{k}^{\prime} p_{j_{k}}$ for $k \geqslant 1$, where $f_{1}-\alpha_{1}^{\prime} p_{j_{1}}:=f-\alpha^{\prime} p_{j}$ ), we can deduce that $f \in\langle P\rangle$ (in fact $f=\sum_{k=1}^{K} \alpha_{k}^{\prime} p_{j_{k}}$ for some finite $K$ ).

Corollary 2.2.6 (The Ascending Chain Condition) Every ascending sequence of ideals $J_{1} \subseteq J_{2} \subseteq \cdots$ over a polynomial ring $\mathcal{R}=R\left[x_{1}, \ldots, x_{n}\right]$ is eventually constant, so that there is an $i$ such that $J_{i}=J_{i+1}=\cdots$.

Proof: By Hilbert's Basis Theorem, each ideal $J_{k}$ (for $k \geqslant 1$ ) is finitely generated. Consider the ideal $J=\cup J_{k}$. This is another ideal over $\mathcal{R}$, and so by Hilbert's Basis Theorem is also finitely generated. From this we deduce that the chain must stop as soon as the generators of $J$ are contained in some $J_{i}$, so that $J_{i}=J_{i+1}=\cdots$.

### 2.3 Buchberger's Algorithm

The algorithm used to compute a Gröbner Basis is known as Buchberger's Algorithm. Bruno Buchberger was a student of Wolfgang Gröbner at the University of Innsbruck, Austria, and the publication of his PhD thesis in 1965 [11] marked the start of Gröbner Basis theory.

In Buchberger's algorithm, S-polynomials for pairs of elements from the current basis are computed and reduced using the current basis. If the S-polynomial does not reduce to zero, it is added to the current basis, and this process continues until all S-polynomials reduce to zero. The algorithm works on the principle that if an S-polynomial S-pol $\left(g_{i}, g_{j}\right)$ does not reduce to zero using a set of polynomials $G$, then it will certainly reduce to zero using the set of polynomials $G \cup\left\{\mathrm{~S}-\mathrm{pol}\left(g_{i}, g_{j}\right)\right\}$.

Theorem 2.3.1 Algorithm 3 always terminates with a Gröbner Basis for the ideal $J$.

Proof (cf. [7], page 213): Correctness. If the algorithm terminates, it does so with a set of polynomials $G$ with the property that all S-polynomials involving members of $G$ reduce to zero using $G\left(\mathrm{~S}-\operatorname{pol}\left(g_{i}, g_{j}\right) \rightarrow_{G} 0\right.$ for all $\left.i \neq j\right) . G$ is therefore a Gröbner Basis by Definition 2.1.4. Termination. If the algorithm does not terminate, then an endless sequence of polynomials must be added to the set $G$ so that the set $A$ never becomes empty. Let $G_{0} \subset G_{1} \subset G_{2} \subset \cdots$ be the successive values of $G$. If we consider the corresponding sequence $\operatorname{LM}\left(G_{0}\right) \subset \operatorname{LM}\left(G_{1}\right) \subset \operatorname{LM}\left(G_{2}\right) \subset \cdots$ of lead monomials, we note that these sets generate an ascending chain of ideals $J_{0} \subset J_{1} \subset J_{2} \subset \cdots$ because each time we add a monomial to a particular set $\operatorname{LM}\left(G_{k}\right)$ to form the set $\operatorname{LM}\left(G_{k+1}\right)$, the monomial we choose is irreducible with respect to $\operatorname{LM}\left(G_{k}\right)$, and hence does not belong to the ideal $J_{k}$. However the Ascending Chain Condition tells us that such a chain of ideals

```
Algorithm 3 A Basic Commutative Gröbner Basis Algorithm
Input: A Basis \(F=\left\{f_{1}, f_{2}, \ldots, f_{m}\right\}\) for an ideal \(J\) over a commutative polynomial ring
    \(R\left[x_{1}, \ldots x_{n}\right]\); an admissible monomial ordering O .
Output: A Gröbner Basis \(G=\left\{g_{1}, g_{2}, \ldots, g_{p}\right\}\) for \(J\).
    Let \(G=F\) and let \(A=\emptyset\);
    For each pair of polynomials \(\left(g_{i}, g_{j}\right)\) in \(G(i<j)\),
    add the S-polynomial S-pol \(\left(g_{i}, g_{j}\right)\) to \(A\);
    while ( \(A\) is not empty) do
        Remove the first entry \(s_{1}\) from \(A\);
        \(s_{1}^{\prime}=\operatorname{Rem}\left(s_{1}, G\right)\);
        if \(\left(s_{1}^{\prime} \neq 0\right)\) then
            Add \(s_{1}^{\prime}\) to \(G\) and add all the S-polynomials \(\mathrm{S}-\mathrm{pol}\left(g_{i}, s_{1}^{\prime}\right)\) to \(A\left(g_{i} \in G, g_{i} \neq s_{1}^{\prime}\right)\);
        end if
    end while
    return \(G\);
```

must eventually become constant, so there must be some $i \geqslant 0$ such that $J_{i}=J_{i+1}=\cdots$. It follows that the algorithm will terminate once the set $G_{i}$ has been constructed, as all of the S-polynomials left in $A$ will now reduce to zero (if not, some S-polynomial left in $A$ will reduce to a non-zero polynomial $s_{1}^{\prime}$ whose lead monomial is irreducible with respect to $\operatorname{LM}\left(G_{i}\right)$, allowing us to construct an ideal $J_{i+1}=\left\langle\operatorname{LM}\left(G_{i}\right) \cup\left\{\operatorname{LM}\left(s_{1}^{\prime}\right)\right\}\right\rangle \supset\left\langle\operatorname{LM}\left(G_{i}\right)\right\rangle=J_{i}$, contradicting the fact that $J_{i+1}=J_{i}$.)

Example 2.3.2 Let $F:=\left\{f_{1}, f_{2}\right\}=\left\{x^{2}-2 x y+3,2 x y+y^{2}+5\right\}$ generate an ideal over the commutative polynomial ring $\mathbb{Q}[x, y]$, and let the monomial ordering be DegLex. Running Algorithm 3 on $F$, there is only one S-polynomial to consider initially, namely S-pol $\left(f_{1}, f_{2}\right)=y\left(f_{1}\right)-\frac{1}{2} x\left(f_{2}\right)=-\frac{5}{2} x y^{2}-\frac{5}{2} x+3 y$. This polynomial reduces (using $f_{2}$ ) to give the irreducible polynomial $\frac{5}{4} y^{3}-\frac{5}{2} x+\frac{37}{4} y=: f_{3}$, which we add to our current basis. This produces two more S-polynomials to look at, $\mathrm{S}-\mathrm{pol}\left(f_{1}, f_{3}\right)=y^{3}\left(f_{1}\right)-\frac{4}{5} x^{2}\left(f_{3}\right)=$ $-2 x y^{4}+2 x^{3}-\frac{37}{5} x^{2} y+3 y^{3}$ and $\mathrm{S}-\operatorname{pol}\left(f_{2}, f_{3}\right)=\frac{1}{2} y^{2}\left(f_{2}\right)-\frac{4}{5} x\left(f_{3}\right)=\frac{1}{2} y^{4}+2 x^{2}-\frac{37}{5} x y+\frac{5}{2} y^{2}$, both of which reduce to zero. The algorithm therefore terminates with the set $\left\{x^{2}-2 x y+\right.$ 3, $\left.2 x y+y^{2}+5, \frac{5}{4} y^{3}-\frac{5}{2} x+\frac{37}{4} y\right\}$ as the output Gröbner Basis.

Here is a dry run for Algorithm 3 in this instance.


### 2.4 Reduced Gröbner Bases

Definition 2.4.1 Let $G=\left\{g_{1}, \ldots, g_{p}\right\}$ be a Gröbner Basis for an ideal over the polynomial ring $R\left[x_{1}, \ldots, x_{n}\right]$. $G$ is a reduced Gröbner Basis if the following conditions are satisfied.
(a) $\mathrm{LC}\left(g_{i}\right)=1_{R}$ for all $g_{i} \in G$.
(b) No term in any polynomial $g_{i} \in G$ is divisible by any $\operatorname{LT}\left(g_{j}\right), j \neq i$.

Theorem 2.4.2 Every ideal over a commutative polynomial ring has a unique reduced Gröbner Basis.

Proof: Existence. By Theorem 2.3.1, there exists a Gröbner Basis $G$ for every ideal over a commutative polynomial ring. We claim that the following procedure transforms $G$ into a reduced Gröbner Basis $G^{\prime}$.
(i) Multiply each $g_{i} \in G$ by $\operatorname{LC}\left(g_{i}\right)^{-1}$.
(ii) Reduce each $g_{i} \in G$ by $G \backslash\left\{g_{i}\right\}$, removing from $G$ all polynomials that reduce to zero.

It is clear that $G^{\prime}$ satisfies the conditions of Definition 2.4.1, so it remains to show that $G^{\prime}$ is a Gröbner Basis, which we shall do by showing that the application of each step of instruction (ii) above produces a basis which is still a Gröbner Basis.

Let $G=\left\{g_{1}, \ldots, g_{p}\right\}$ be a Gröbner Basis, and let $g_{i}^{\prime}$ be the reduction of an arbitrary
$g_{i} \in G$ with respect to $G \backslash\left\{g_{i}\right\}$, carried out as follows (the $t_{k}$ are terms).

$$
\begin{equation*}
g_{i}^{\prime}=g_{i}-\sum_{k=1}^{\kappa} t_{k} g_{j_{k}} \tag{2.1}
\end{equation*}
$$

Set $H=\left(G \backslash\left\{g_{i}\right\}\right) \cup\left\{g_{i}^{\prime}\right\}$ if $g_{i}^{\prime} \neq 0$, and set $H=G \backslash\left\{g_{i}\right\}$ if $g_{i}^{\prime}=0$. As $G$ is a Gröbner Basis, all S-polynomials involving elements of $G$ reduce to zero using $G$, so there are expressions

$$
\begin{equation*}
t_{a} g_{a}-t_{b} g_{b}-\sum_{u=1}^{\mu} t_{u} g_{c_{u}}=0 \tag{2.2}
\end{equation*}
$$

for every S-polynomial S-pol $\left(g_{a}, g_{b}\right)=t_{a} g_{a}-t_{b} g_{b}$, where $g_{a}, g_{b}, g_{c_{u}} \in G$. To show that $H$ is a Gröbner Basis, we must show that all S-polynomials involving elements of $H$ reduce to zero using $H$. For distinct polynomials $g_{a}, g_{b} \in H$ not equal to $g_{i}^{\prime}$, we can reduce the S-polynomial S-pol $\left(g_{a}, g_{b}\right)$ using the reduction shown in Equation (2.2), substituting for $g_{i}$ from Equation (2.1) if any of the $g_{c_{u}}$ in Equation (2.2) are equal to $g_{i}$. This gives a reduction to zero of S-pol $\left(g_{a}, g_{b}\right)$ in terms of elements of $H$.

If $g_{i}^{\prime}=0$, our proof is complete. Otherwise consider the S-polynomial S-pol $\left(g_{i}^{\prime}, g_{a}\right)$. We claim that $\mathrm{S}-\operatorname{pol}\left(g_{i}, g_{a}\right)=t_{1} g_{i}-t_{2} g_{a} \Rightarrow \mathrm{~S}-\operatorname{pol}\left(g_{i}^{\prime}, g_{a}\right)=t_{1} g_{i}^{\prime}-t_{2} g_{a}$. To prove this claim, it is sufficient to show that $\operatorname{LT}\left(g_{i}\right)=\operatorname{LT}\left(g_{i}^{\prime}\right)$. Assume for a contradiction that $\mathrm{LT}\left(g_{i}\right) \neq \mathrm{LT}\left(g_{i}^{\prime}\right)$. It follows that during the reduction of $g_{i}$ we were able to reduce its lead term, so that $\operatorname{LT}\left(g_{i}\right)=t \mathrm{LT}\left(g_{j}\right)$ for some term $t$ and some $g_{j} \in G$. By the admissibility of the chosen monomial ordering, the polynomial $g_{i}-t g_{j}$ reduces to zero without using $g_{i}$, leading to the conclusion that $g_{i}^{\prime}=0$, a contradiction.

It remains to show that $\operatorname{S-pol}\left(g_{i}^{\prime}, g_{a}\right) \rightarrow_{H} 0$. We know that $\operatorname{S-pol}\left(g_{i}, g_{a}\right)=t_{1} g_{i}-t_{2} g_{a} \rightarrow_{G} 0$, and Equation (2.2) tells us that $t_{1} g_{i}-t_{2} g_{a}-\sum_{u=1}^{\mu} t_{u} g_{c_{u}}=0$. Substituting for $g_{i}$ from Equation (2.1), we obtain ${ }^{5}$

$$
t_{1}\left(g_{i}^{\prime}+\sum_{k=1}^{\kappa} t_{k} g_{j_{k}}\right)-t_{2} g_{a}-\sum_{u=1}^{\mu} t_{u} g_{c_{u}}=0
$$

or

$$
t_{1} g_{i}^{\prime}-t_{2} g_{a}-\left(\sum_{u=1}^{\mu} t_{u} g_{c_{u}}-\sum_{k=1}^{\kappa} t_{1} t_{k} g_{j_{k}}\right)=0
$$

[^4]which implies that S-pol $\left(g_{i}^{\prime}, g_{a}\right) \rightarrow_{H} 0$.
Uniqueness. Assume for a contradiction that $G=\left\{g_{1}, \ldots, g_{p}\right\}$ and $H=\left\{h_{1}, \ldots, h_{q}\right\}$ are two reduced Gröbner Bases for an ideal $J$, with $G \neq H$. Let $g_{i}$ be an arbitrary element from $G$ (where $1 \leqslant i \leqslant p$ ). Because $g_{i}$ is a member of the ideal, then $g_{i}$ must reduce to zero using $H$ ( $H$ is a Gröbner Basis). This means that there must exist a polynomial $h_{j} \in H$ such that $\operatorname{LT}\left(h_{j}\right) \mid \operatorname{LT}\left(g_{i}\right)$. If $\operatorname{LT}\left(h_{j}\right) \neq \operatorname{LT}\left(g_{i}\right)$, then $\operatorname{LT}\left(h_{j}\right) \times m=\operatorname{LT}\left(g_{i}\right)$ for some nontrivial monomial $m$. But $h_{j}$ is also a member of the ideal, so it must reduce to zero using $G$. Therefore there exists a polynomial $g_{k} \in G$ such that $\operatorname{LT}\left(g_{k}\right) \mid \operatorname{LT}\left(h_{j}\right)$, which implies that $\operatorname{LT}\left(g_{k}\right) \mid \operatorname{LT}\left(g_{i}\right)$, with $k \neq i$. This contradicts condition (b) of Definition 2.4.1, so that $G$ cannot be a reduced Gröbner Basis for $J$ if $\operatorname{LT}\left(h_{j}\right) \neq \mathrm{LT}\left(g_{i}\right)$. From this we deduce that each $g_{i} \in G$ has a corresponding $h_{j} \in H$ such that $\operatorname{LT}\left(g_{i}\right)=\operatorname{LT}\left(h_{j}\right)$. Further, because $G$ and $H$ are assumed to be reduced Gröbner Bases, this is a one-to-one correspondence.

It remains to show that if $\operatorname{LT}\left(g_{i}\right)=\operatorname{LT}\left(h_{j}\right)$, then $g_{i}=h_{j}$. Assume for a contradiction that $g_{i} \neq h_{j}$, and consider the polynomial $g_{i}-h_{j}$. Without loss of generality, assume that $\operatorname{LM}\left(g_{i}-h_{j}\right)$ appears in $g_{i}$. Because $g_{i}-h_{j}$ is a member of the ideal, then there is a polynomial $g_{k} \in G$ such that $\operatorname{LT}\left(g_{k}\right) \mid \operatorname{LT}\left(g_{i}-h_{j}\right)$. But this again contradicts condition (b) of Definition 2.4.1, as we have shown that there is a term in $g_{i}$ that is divisible by $\mathrm{LT}\left(g_{k}\right)$ for some $k \neq i$. It follows that $G$ cannot be a reduced Gröbner Basis if $g_{i} \neq h_{j}$, which means that $G=H$ and therefore reduced Gröbner Bases are unique.

Given a Gröbner Basis $G$, we saw in the proof of Theorem 2.4.2 that if the lead term of any polynomial $g_{i} \in G$ is reducible by some polynomial $g_{j} \in G$ (where $g_{j} \neq g_{i}$ ), then $g_{i}$ reduces to zero. We can use this information to refine the procedure for finding a unique reduced Gröbner Basis (as given in the aforementioned proof) by allowing the removal of any polynomial $g_{i} \in G$ whose lead monomial is a multiple of some other lead monomial $\mathrm{LM}\left(g_{j}\right)$. This process, which if often referred to as minimising a Gröbner Basis (as in finding a Gröbner Basis with the minimal number of elements), is incorporated into our refined procedure, which we state as Algorithm 4.

### 2.5 Improvements to Buchberger's Algorithm

Nowadays, most general purpose symbolic computation systems possess an implementation of Buchberger's algorithm. These implementations often take advantage of the

```
Algorithm 4 The Commutative Unique Reduced Gröbner Basis Algorithm
Input: A Gröbner Basis \(G=\left\{g_{1}, g_{2}, \ldots, g_{m}\right\}\) for an ideal \(J\) over a commutative polyno-
    mial ring \(R\left[x_{1}, \ldots x_{n}\right]\); an admissible monomial ordering O .
Output: The unique reduced Gröbner Basis \(G^{\prime}=\left\{g_{1}^{\prime}, g_{2}^{\prime}, \ldots, g_{p}^{\prime}\right\}\) for \(J\).
    \(G^{\prime}=\emptyset ;\)
    for each \(g_{i} \in G\) do
        Multiply \(g_{i}\) by \(\operatorname{LC}\left(g_{i}\right)^{-1}\);
        if \(\left(\mathrm{LM}\left(g_{i}\right)=u \mathrm{LM}\left(g_{j}\right)\right.\) for some monomial \(u\) and some \(\left.g_{j} \in G\left(g_{j} \neq g_{i}\right)\right)\) then
            \(G=G \backslash\left\{g_{i}\right\} ;\)
        end if
    end for
    for each \(g_{i} \in G\) do
        \(g_{i}^{\prime}=\operatorname{Rem}\left(g_{i},\left(G \backslash\left\{g_{i}\right\}\right) \cup G^{\prime}\right) ;\)
        \(G=G \backslash\left\{g_{i}\right\} ; G^{\prime}=G^{\prime} \cup\left\{g_{i}^{\prime}\right\} ;\)
    end for
    return \(G^{\prime}\);
```

numerous improvements made to Buchberger's algorithm over the years, some of which we shall now describe.

### 2.5.1 Buchberger's Criteria

In 1979, Buchberger published a paper [10] which gave criteria that enable the a priori detection of S-polynomials that reduce to zero. This speeds up Algorithm 3 by drastically reducing the number of S-polynomials that must be reduced with respect to the current basis.

Proposition 2.5.1 (Buchberger's First Criterion) Let $f$ and $g$ be two polynomials over a commutative polynomial ring ordered with respect to some fixed admissible monomial ordering $O$. If the lead terms of $f$ and $g$ are disjoint (so that $1 \mathrm{~cm}(\mathrm{LM}(f), \mathrm{LM}(g))=$ $\mathrm{LM}(f) \mathrm{LM}(g))$, then $\mathrm{S}-\mathrm{pol}(f, g)$ reduces to zero using the set $\{f, g\}$.

Proof (Adapted from [7], Lemma 5.66): Assume that $f=\sum_{i=1}^{\alpha} s_{i}$ and $g=\sum_{j=1}^{\beta} t_{j}$, where the $s_{i}$ and the $t_{j}$ are terms. Because $s_{1}$ and $t_{1}$ are disjoint, it follows that

$$
\begin{align*}
\operatorname{S-pol}(f, g) & \equiv t_{1} f-s_{1} g \\
& =t_{1}\left(s_{2}+\cdots+s_{\alpha}\right)-s_{1}\left(t_{2}+\cdots+t_{\beta}\right) . \tag{2.3}
\end{align*}
$$

We claim that no two terms in Equation (2.3) are the same. Assume to the contrary that $t_{1} s_{i}=s_{1} t_{j}$ for some $2 \leqslant i \leqslant \alpha$ and $2 \leqslant j \leqslant \beta$. Then $t_{1} s_{i}$ is a multiple of both $t_{1}$ and $s_{1}$, which means that $t_{1} s_{i}$ is a multiple of $\operatorname{lcm}\left(t_{1}, s_{1}\right)=t_{1} s_{1}$. But then we must have $t_{1} s_{i} \geqslant t_{1} s_{1}$, which gives a contradiction (by definition $s_{1}>s_{i}$ ).

As every term in $t_{1}\left(s_{2}+\cdots+s_{\alpha}\right)$ is a multiple of $t_{1}$, we can use $g$ to eliminate each of the terms $t_{1} s_{\alpha}, t_{1} s_{\alpha-1}, \ldots, t_{1} s_{2}$ in Equation (2.3) in turn:

$$
\begin{align*}
& t_{1}\left(s_{2}+\cdots+s_{\alpha}\right)-s_{1}\left(t_{2}+\cdots+t_{\beta}\right) \\
\rightarrow & t_{1}\left(s_{2}+\cdots+s_{\alpha}\right)-s_{1}\left(t_{2}+\cdots+t_{\beta}\right)-s_{\alpha} g \\
= & t_{1}\left(s_{2}+\cdots+s_{\alpha-1}\right)-s_{1}\left(t_{2}+\cdots+t_{\beta}\right)-s_{\alpha}\left(t_{2}+\cdots+t_{\beta}\right) \\
\rightarrow & t_{1}\left(s_{2}+\cdots+s_{\alpha-2}\right)-\left(s_{1}+s_{\alpha-1}+s_{\alpha}\right)\left(t_{2}+\cdots+t_{\beta}\right) \\
\vdots & \\
\rightarrow & -\left(s_{1}+s_{2}+\cdots+s_{\alpha}\right)\left(t_{2}+\cdots+t_{\beta}\right) \\
= & -s_{1}\left(t_{2}+\cdots+t_{\beta}\right)-\cdots-s_{\alpha}\left(t_{2}+\cdots+t_{\beta}\right) . \tag{2.4}
\end{align*}
$$

We do this in reverse order because, having eliminated a term $t_{1} s_{\gamma}$ (where $3 \leqslant \gamma \leqslant \alpha$ ), to continue the term $t_{1} s_{\gamma-1}$ must appear in the reduced polynomial (which it does because $t_{1} s_{\gamma-1}>s_{\delta} t_{\eta}$ for all $\gamma \leqslant \delta \leqslant \alpha$ and $2 \leqslant \eta \leqslant \beta$ ).

We now use the same argument on $-s_{1}\left(t_{2}+\cdots+t_{\beta}\right)$, using $f$ to eliminate each of its terms in turn, giving the following reduction sequence.

$$
\begin{aligned}
& -s_{1}\left(t_{2}+\cdots+t_{\beta}\right)-\cdots-s_{\alpha}\left(t_{2}+\cdots+t_{\beta}\right) \\
\rightarrow & -s_{1}\left(t_{2}+\cdots+t_{\beta}\right)-\cdots-s_{\alpha}\left(t_{2}+\cdots+t_{\beta}\right)+t_{2} f \\
= & -s_{1}\left(t_{2}+\cdots+t_{\beta}\right)-\cdots-s_{\alpha}\left(t_{2}+\cdots+t_{\beta}\right)+t_{2}\left(s_{1}+\cdots+s_{\alpha}\right) \\
= & -s_{1}\left(t_{3}+\cdots+t_{\beta}\right)-\cdots-s_{\alpha}\left(t_{3}+\cdots+t_{\beta}\right) \\
\rightarrow & -s_{1}\left(t_{4}+\cdots+t_{\beta}\right)-\cdots-s_{\alpha}\left(t_{4}+\cdots+t_{\beta}\right) \\
\vdots & \\
\rightarrow & 0 .
\end{aligned}
$$

Technical point: If some term $s_{i} t_{j}$ (for $i, j \geqslant 2$ ) cancels the term $s_{1} t_{k}$ (for $k \geqslant 3$ ) in Equation (2.4), then as we must have $j<k$ in order to have $s_{i} t_{j}=s_{1} t_{k}$, the term $s_{1} t_{k}$ will reappear as $s_{i} t_{j}$ when the term $s_{1} t_{j}$ is eliminated, allowing us to continue the reduction as shown. This argument can be extended to the case where a combination of terms of the form $s_{i} t_{j}$ cancel the term $s_{1} t_{k}$, as the term $s_{1} t_{k}$ will reappear after all the terms $s_{1} t_{\kappa}$ (for $2 \leqslant \kappa<k$ ) have been eliminated.

Proposition 2.5.2 (Buchberger's Second Criterion) Let $f, g$ and $h$ be three members of a finite set of polynomials $P$ over a commutative polynomial ring satisfying the following conditions.
(a) $\mathrm{LM}(h) \mid \operatorname{lcm}(\operatorname{LM}(f), \operatorname{LM}(g))$.
(b) $\operatorname{S-pol}(f, h) \rightarrow_{P} 0$ and $\operatorname{S-pol}(g, h) \rightarrow_{P} 0$.

Then $\operatorname{S-pol}(f, g) \rightarrow_{P} 0$.

Proof: If $\operatorname{LM}(h) \mid \operatorname{lcm}(\operatorname{LM}(f), \operatorname{LM}(g))$, then $m_{h} \operatorname{LM}(h)=\operatorname{lcm}(\operatorname{LM}(f), \operatorname{LM}(g))$ for some monomial $m_{h}$. Assume that $\operatorname{lcm}(\mathrm{LM}(f), \mathrm{LM}(g))=m_{f} \mathrm{LM}(f)=m_{g} \mathrm{LM}(g)$ for some monomials $m_{f}$ and $m_{g}$. Then it is clear that $m_{f} \mathrm{LM}(f)=m_{h} \mathrm{LM}(h)$ is a common multiple of $\mathrm{LM}(f)$ and $\mathrm{LM}(h)$, and $m_{g} \mathrm{LM}(g)=m_{h} \mathrm{LM}(h)$ is a common multiple of $\mathrm{LM}(g)$ and $\mathrm{LM}(h)$. It follows that $\operatorname{lcm}(\mathrm{LM}(f), \mathrm{LM}(g))$ is a multiple of both $\operatorname{lcm}(\operatorname{LM}(f), \mathrm{LM}(h))$ and $1 \mathrm{~cm}(\mathrm{LM}(g), \operatorname{LM}(h))$, so that

$$
\begin{equation*}
\operatorname{lcm}(\operatorname{LM}(f), \operatorname{LM}(g))=m_{f h} \operatorname{lcm}(\operatorname{LM}(f), \mathrm{LM}(h))=m_{g h} \operatorname{lcm}(\mathrm{LM}(g), \mathrm{LM}(h)) \tag{2.5}
\end{equation*}
$$

for some monomials $m_{f h}$ and $m_{g h}$.
Because the S-polynomials S-pol $(f, h)$ and $\mathrm{S}-\operatorname{pol}(g, h)$ both reduce to zero using $P$, there are expressions

$$
\mathrm{S}-\operatorname{pol}(f, h)-\sum_{i=1}^{\alpha} s_{i} p_{i}=0
$$

and

$$
\operatorname{S-pol}(g, h)-\sum_{j=1}^{\beta} t_{j} p_{j}=0,
$$

where the $s_{i}$ and the $t_{j}$ are terms, and $p_{i}, p_{j} \in P$ for all $i$ and $j$. It follows that

$$
\begin{gathered}
m_{f h}\left(\mathrm{~S}-\operatorname{pol}(f, h)-\sum_{i=1}^{\alpha} s_{i} p_{i}\right)=m_{g h}\left(\mathrm{~S}-\operatorname{pol}(g, h)-\sum_{j=1}^{\beta} t_{j} p_{j}\right) ; \\
m_{f h}\left(\frac{\operatorname{lcm}(\mathrm{LM}(f), \mathrm{LM}(h))}{\mathrm{LT}(f)} f-\frac{\operatorname{lcm}(\mathrm{LM}(f), \mathrm{LM}(h))}{\mathrm{LT}(h)} h-\sum_{i=1}^{\alpha} s_{i} p_{i}\right)= \\
m_{g h}\left(\frac{\operatorname{lcm}(\mathrm{LM}(g), \mathrm{LM}(h))}{\mathrm{LT}(g)} g-\frac{\operatorname{lcm}(\mathrm{LM}(g), \mathrm{LM}(h))}{\mathrm{LT}(h)} h-\sum_{j=1}^{\beta} t_{j} p_{j}\right) ; \\
m_{f h}\left(\frac{\operatorname{lcm}(\mathrm{LM}(f), \mathrm{LM}(g))}{m_{f h} \mathrm{LT}(f)} f-\frac{\operatorname{lcm}(\mathrm{LM}(f), \mathrm{LM}(g))}{m_{f h} \mathrm{LT}(h)} h-\sum_{i=1}^{\alpha} s_{i} p_{i}\right)= \\
m_{g h}\left(\frac{\operatorname{lcm}(\mathrm{LM}(f), \mathrm{LM}(g))}{m_{g h} \mathrm{LT}(g)} g-\frac{\operatorname{lcm}(\mathrm{LM}(f), \mathrm{LM}(g))}{m_{g h} \mathrm{LT}(h)} h-\sum_{j=1}^{\beta} t_{j} p_{j}\right) ; \\
\frac{\operatorname{lcm}(\mathrm{LM}(f), \mathrm{LM}(g))}{\mathrm{LT}(f)} f-m_{f h} \sum_{i=1}^{\alpha} s_{i} p_{i}=\frac{\operatorname{lcm}(\mathrm{LM}(f), \mathrm{LM}(g))}{\mathrm{LT}(g)} g-m_{g h} \sum_{j=1}^{\beta} t_{j} p_{j} ; \\
\mathrm{S}-\operatorname{pol}(f, g)-\sum_{i=1}^{\alpha} m_{f h} s_{i} p_{i}+\sum_{j=1}^{\beta} m_{g h} t_{j} p_{j}=0 .
\end{gathered}
$$

To conclude that the S-polynomial S-pol $(f, g)$ reduces to zero using $P$, it remains to show that the algebraic expression $-\sum_{i=1}^{\alpha} m_{f h} s_{i} p_{i}+\sum_{j=1}^{\beta} m_{g h} t_{j} p_{j}$ corresponds to a valid reduction of $\operatorname{S-pol}(f, g)$. To do this, it is sufficient to show that no term in either of the summations is greater than $\operatorname{lcm}(\mathrm{LM}(f), \mathrm{LM}(g))$ (so that $\mathrm{LM}\left(m_{f h} s_{i} p_{i}\right)<\operatorname{lcm}(\mathrm{LM}(f), \mathrm{LM}(g))$ and $\mathrm{LM}\left(m_{g h} t_{j} p_{j}\right)<\operatorname{lcm}(\operatorname{LM}(f), \mathrm{LM}(g))$ for all $i$ and $\left.j\right)$. But this follows from Equation (2.5) and from the fact that the original reductions of $\mathrm{S}-\operatorname{pol}(f, h)$ and $\mathrm{S}-\mathrm{pol}(g, h)$ are valid, so that $\operatorname{LM}\left(s_{i} p_{i}\right)<\operatorname{lcm}(\operatorname{LM}(f), \operatorname{LM}(h))$ and $\operatorname{LM}\left(t_{j} p_{j}\right)<\operatorname{lcm}(\operatorname{LM}(g), \operatorname{LM}(h))$ for all $i$ and $j$.

### 2.5.2 Homogeneous Gröbner Bases

Definition 2.5.3 A polynomial is homogeneous if all its terms have the same degree. For example, the polynomial $x^{2} y+4 y z^{2}+3 z^{3}$ is homogeneous, but the polynomial $x^{3} y+4 x^{2}+45$ is not homogeneous.

Of the many systems available for computing commutative Gröbner Bases, some (such as Bergman [6]) only admit sets of homogeneous polynomials as input. This restriction leads to gains in efficiency as we can take advantage of some of the properties of homogeneous polynomial arithmetic. For example, the S-polynomial of two homogeneous polynomials is homogeneous, and the reduction of a homogeneous polynomial by a set of homogeneous polynomials yields another homogeneous polynomial. It follows that if $G$ is a Gröbner Basis for a set $F$ of homogeneous polynomials, then $G$ is another set of homogeneous polynomials.

At first glance, it seems that a system accepting only sets of homogeneous polynomials as input is not able to compute a Gröbner Basis for a set of polynomials containing one or more non-homogeneous polynomials. However, we can still use the system if we use an extendible monomial ordering and the processes of homogenisation and dehomogenisation.

Definition 2.5.4 Let $p=p_{0}+\cdots+p_{m}$ be a polynomial over the polynomial ring $R\left[x_{1}, \ldots, x_{n}\right]$, where each $p_{i}$ is the sum of the degree $i$ terms in $p$ (we assume that $p_{m} \neq 0$ ). The homogenisation of $p$ with respect to a new (homogenising) variable $y$ is the polynomial

$$
h(p):=p_{0} y^{m}+p_{1} y^{m-1}+\cdots+p_{m-1} y+p_{m},
$$

where $h(p)$ belongs to a polynomial ring determined by where $y$ is placed in the lexicographical ordering of the variables.

Definition 2.5.5 The dehomogenisation of a polynomial $p$ is the polynomial $d(p)$ given by substituting $y=1$ in $p$, where $y$ is the homogenising variable. For example, the dehomogenisation of the polynomial $x_{1}^{3}+x_{1} x_{2} y+x_{1} y^{2} \in \mathbb{Q}\left[x_{1}, x_{2}, y\right]$ is the polynomial $x_{1}^{3}+x_{1} x_{2}+x_{1} \in \mathbb{Q}\left[x_{1}, x_{2}\right]$.

Definition 2.5.6 A monomial ordering $O$ is extendible if, given any polynomial $p=$ $t_{1}+\cdots+t_{\alpha}$ ordered with respect to $O$ (where $t_{1}>\cdots>t_{\alpha}$ ), the homogenisation of $p$ preserves the order on the terms $\left(t_{i}^{\prime}>t_{i+1}^{\prime}\right.$ for all $1 \leqslant i \leqslant \alpha-1$, where the homogenisation
process maps the term $t_{i} \in p$ to the term $\left.t_{i}^{\prime} \in h(p)\right)$.

Of the monomial orderings defined in Section 1.2.1, two of them (Lex and DegRevLex) are extendible as long as we ensure that the new variable $y$ is lexicographically less than any of the variables $x_{1}, \ldots, x_{n}$; another (InvLex) is extendible as long as we ensure that the new variable $y$ is lexicographically greater than any of the variables $x_{1}, \ldots, x_{n}$.

The other monomial orderings are not extendible as, no matter where we place the new variable $y$ in the ordering of the variables, we can always find two monomials $m_{1}$ and $m_{2}$ such that, if $p=m_{1}+m_{2}$ ( with $m_{1}>m_{2}$ ), then in $h(p)=m_{1}^{\prime}+m_{2}^{\prime}$, we have $m_{1}^{\prime}<m_{2}^{\prime}$. For example, $m_{1}:=x_{1} x_{2}^{2}$ and $m_{2}:=x_{1}^{2}$ provides a counterexample for the DegLex monomial ordering.

Definition 2.5.7 Let $F=\left\{f_{1}, \ldots, f_{m}\right\}$ be a non-homogeneous set of polynomials. To compute a Gröbner Basis for $F$ using a program that only accepts sets of homogeneous polynomials as input, we proceed as follows.
(a) Construct a homogeneous set of polynomials $F^{\prime}=\left\{h\left(f_{1}\right), \ldots, h\left(f_{m}\right)\right\}$.
(b) Compute a Gröbner Basis $G^{\prime}$ for $F^{\prime}$.
(c) Dehomogenise each polynomial $g^{\prime} \in G^{\prime}$ to obtain a set of polynomials $G$.

As long as the chosen monomial ordering $O$ is extendible, $G$ will be a Gröbner Basis for $F$ with respect to $O$ [22, page 113]. A word of warning however - this process is not necessarily more efficient that the direct computation of a Gröbner Basis for $F$ using a program that does accept non-homogeneous sets of polynomials as input.

### 2.5.3 Selection Strategies

One of the most important factors when considering the efficiency of Buchberger's algorithm is the order in which S-polynomials are processed during the algorithm. A particular choice of a selection strategy to use can often cut down substantially the amount of work required in order to obtain a particular Gröbner Basis.

In 1979, Buchberger defined the normal strategy [10] that chooses to process an Spolynomial S-pol $(f, g)$ if the monomial $\operatorname{lcm}(\operatorname{LM}(f), \mathrm{LM}(g))$ is minimal (in the chosen
monomial ordering) amongst all such lowest common multiples. This strategy was refined in 1991 to give the sugar strategy [29], a strategy that chooses an S-polynomial to process if the sugar of the S-polynomial (a value associated to the S-polynomial) is minimal amongst all such values (the normal strategy is used in the event of a tie).

Motivation for the sugar strategy comes from the observation that the normal strategy performs well when used with a degree-based monomial ordering and a homogeneous basis; the sugar strategy was developed as a way to proceed based on what would happen when using the normal strategy in the computation of a Gröbner Basis for the corresponding homogenised input basis. We can therefore think of the sugar of an S-polynomial as representing the degree of the corresponding S-polynomial in the homogeneous computation.

The sugar of an S-polynomial is computed by using the following rules on the sugars of polynomials we encounter during the computation of a Gröbner Basis for the set of polynomials $F=\left\{f_{1}, \ldots, f_{m}\right\}$.
(1) The sugar $\operatorname{Sug}_{f_{i}}$ of a polynomial $f_{i} \in F$ is the total degree of the polynomial $f_{i}$ (which is the degree of the term of maximal degree in $f_{i}$ ).
(2) If $p$ is a polynomial and if $t$ is a term, then $\operatorname{Sug}_{t p}=\operatorname{deg}(t)+\operatorname{Sug}_{p}$.
(3) If $p=p_{1}+p_{2}$, then $\operatorname{Sug}_{p}=\max \left(\operatorname{Sug}_{p_{1}}, \operatorname{Sug}_{p_{2}}\right)$.

It follows that the sugar of the S-polynomial S-pol $(g, h)=\frac{\operatorname{lcm}(\operatorname{LM}(g), \mathrm{LM}(h))}{\operatorname{LT}(g)} g-\frac{\operatorname{lcm}(\operatorname{LM}(g), \mathrm{LM}(h))}{\operatorname{LT}(h)} h$ is given by the formula
$\operatorname{Sug}_{S-\operatorname{pol}(g, h)}=\max \left(\operatorname{Sug}_{g}-\operatorname{deg}(\operatorname{LM}(g)), \operatorname{Sug}_{h}-\operatorname{deg}(\operatorname{LM}(h))\right)+\operatorname{deg}(\operatorname{lcm}(\operatorname{LM}(g), \operatorname{LM}(h)))$.

Example 2.5.8 To illustrate how a selection strategy reduces the amount of work required to compute a Gröbner Basis, consider the ideal generated by the basis $\left\{x^{31}-x^{6}-\right.$ $\left.x-y, x^{8}-z, x^{10}-t\right\}$ over the polynomial ring $\mathbb{Q}[x, y, z, t]$. In our own implementation of Buchberger's algorithm, here is the number of S-polynomials processed during the algorithm when different selection strategies and different monomial orderings are used (the numbers quoted take into account the application of both of Buchberger's criteria).

| Selection Strategy | Lex | DegLex | DegRevLex |
| :---: | :---: | :---: | :---: |
| No strategy | 640 | 275 | 320 |
| Normal strategy | 123 | 63 | 61 |
| Sugar strategy | 96 | 55 | 54 |

### 2.5.4 Basis Conversion Algorithms

One factor which heavily influences the amount of time taken to compute a Gröbner Basis is the monomial ordering chosen. It is well known that some monomial orderings (such as Lex) are characterised as being 'slow', while other monomial orderings (such as DegRevLex) are said to be 'fast'. In practice what this means is that it usually takes far more time to calculate (say) a Lex Gröbner Basis than it does to calculate a DegRevLex Gröbner Basis for the same generating set of polynomials.

Because many of the useful applications of Gröbner Bases (such as solving systems of polynomial equations) depend on using 'slow' monomial orderings, a number of algorithms were developed in the 1990's that allow us to obtain a Gröbner Basis with respect to one monomial ordering from a Gröbner Basis with respect to another monomial ordering.

The idea is that the time it takes to compute a Gröbner Basis with respect to a 'fast' monomial ordering and then to convert it to a Gröbner Basis with respect to a 'slow' monomial ordering may be significantly less than the time it takes to compute a Gröbner Basis for the 'slow' monomial ordering directly. Although seemingly counterintuitive, the idea works well in practice.

One of the first conversion methods developed was the FGLM method, named after the four authors who published the paper [21] introducing it. The method relies on linear algebra to do the conversion, working with coefficient matrices and irreducible monomials. Its only drawback lies in the fact that it can only be used with zero-dimensional ideals, which are the ideals containing only a finite number of irreducible monomials (for each variable $x_{i}$ in the polynomial ring, a Gröbner Basis for a zero-dimensional ideal must contain a polynomial which has a power of $x_{i}$ as the leading monomial). This restriction does not apply in the case of the Gröbner Walk [18], a basis conversion method we shall study in further detail in Chapter 6.

### 2.5.5 Optimal Variable Orderings

In many cases, the ordering of the variables in a polynomial ring can have a significant effect on the time it takes to compute a Gröbner Basis for a particular ideal (an example can be found in [17]). This is worth bearing in mind if we are searching for any Gröbner Basis with respect to a certain ideal, so do not mind which variable ordering is being used. A heuristically optimal variable ordering is described in [34] (deriving from a discussion in [9]), where we order the variables so that the variable that occurs least often in the polynomials of the input basis is the largest variable; the second least common variable is the second largest variable; and so on (ties are broken randomly).

Example 2.5.9 Let $F:=\left\{y^{2} z^{2}+x^{2} y, x^{2} y^{4} z+x y^{2} z+y^{3}, y^{7}+x^{3} z\right\}$ generate an ideal over the polynomial ring $\mathbb{Q}[x, y, z]$. Because $x$ occurs 8 times in $F, y$ occurs 19 times and $z$ occurs 5 times, the heuristically optimal variable ordering is $z>x>y$. This is supported by the following table showing the times taken to compute a Lex Gröbner Basis for $F$ using all six possible variable orderings, where we see that the time for the heuristically optimal variable ordering is close to the time for the true optimal variable ordering.

| Variable Order | Time | Size of Gröbner Basis |
| :---: | :---: | :---: |
| $x>y>z$ | $1: 15.10$ | 6 |
| $x>z>y$ | $0: 02.85$ | 7 |
| $y>x>z$ | $2: 19.45$ | 7 |
| $y>z>x$ | $2: 16.09$ | 7 |
| $z>x>y$ | $0: 05.91$ | 8 |
| $z>y>x$ | $5: 44.38$ | 8 |

### 2.5.6 Logged Gröbner Bases

In some situations, such as in the algorithm for the Gröbner Walk, it is desirable to be able to express each member of a Gröbner Basis in terms of members of the original basis from which the Gröbner Basis was computed. When we have such representations, our Gröbner Basis is said to be a Logged Gröbner Basis.

Definition 2.5.10 Let $G=\left\{g_{1}, \ldots, g_{p}\right\}$ be a Gröbner Basis computed from an initial basis $F=\left\{f_{1}, \ldots, f_{m}\right\}$. We say that $G$ is a Logged Gröbner Basis if, for each $g_{i} \in G$, we
have an explicit expression of the form

$$
g_{i}=\sum_{\alpha=1}^{\beta} t_{\alpha} f_{k_{\alpha}},
$$

where the $t_{\alpha}$ are terms and $f_{k_{\alpha}} \in F$ for all $1 \leqslant \alpha \leqslant \beta$.
Proposition 2.5.11 Given a finite basis $F=\left\{f_{1}, \ldots, f_{m}\right\}$, it is always possible to compute a Logged Gröbner Basis for $F$.

Proof: We are required to prove that every polynomial added to the input basis $F=$ $\left\{f_{1}, \ldots, f_{m}\right\}$ during Buchberger's algorithm has a representation in terms of members of $F$. But any such polynomial must be a reduced S-polynomial, so it follows that the first polynomial $f_{m+1}$ added to $F$ will always have the form

$$
f_{m+1}=\operatorname{S-pol}\left(f_{i}, f_{j}\right)-\sum_{\alpha=1}^{\beta} t_{\alpha} f_{k_{\alpha}},
$$

where $f_{i}, f_{j}, f_{k_{\alpha}} \in F$ and the $t_{\alpha}$ are terms. This expression clearly gives a representation of our new polynomial in terms of members of $F$, and by induction (using substitution) it is also clear that each subsequent polynomial added to $F$ will also have a representation in terms of members of $F$.

Example 2.5.12 Let $F:=\left\{f_{1}, f_{2}, f_{3}\right\}=\{x y-z, 2 x+y z+z, x+y z\}$ generate an ideal over the polynomial ring $\mathbb{Q}[x, y, z]$, and let the monomial ordering be Lex. In obtaining a Gröbner Basis for $F$ using Buchberger's algorithm, three new polynomials are added to $F$, giving a Gröbner Basis $G:=\left\{g_{1}, g_{2}, g_{3}, g_{4}, g_{5}, g_{6}\right\}=\{x y-z, 2 x+y z+$ $\left.z, x+y z,-\frac{1}{2} y z+\frac{1}{2} z,-2 z^{2},-2 z\right\}$. These three new polynomials are obtained from the Spolynomials S-pol $(2 x+y z+z, x+y z), S-\operatorname{pol}\left(x y-z,-\frac{1}{2} y z+\frac{1}{2} z\right)$ and S-pol $(x y-z, 2 x+y z+z)$
respectively:

$$
\begin{aligned}
\mathrm{S}-\mathrm{pol}(2 x+y z+z, x+y z) & =\frac{1}{2}(2 x+y z+z)-(x+y z) \\
& =-\frac{1}{2} y z+\frac{1}{2} z ; \\
\text { S-pol }\left(x y-z,-\frac{1}{2} y z+\frac{1}{2} z\right) & =z(x y-z)+2 x\left(-\frac{1}{2} y z+\frac{1}{2} z\right) \\
& =x z-z^{2} \\
& \rightarrow_{f_{2}} \quad x z-z^{2}-\frac{1}{2} z(2 x+y z+z) \\
& =-\frac{1}{2} y z^{2}-\frac{3}{2} z^{2} \\
& \rightarrow_{g_{4}}-\frac{1}{2} y z^{2}-\frac{3}{2} z^{2}-z\left(-\frac{1}{2} y z+\frac{1}{2} z\right) \\
& =-2 z^{2} ; \\
& =(x y-z)-\frac{1}{2} y(2 x+y z+z) \\
& =-\frac{1}{2} y^{2} z-\frac{1}{2} y z-z \\
& \rightarrow_{g_{4}}-\frac{1}{2} y^{2} z-\frac{1}{2} y z-z-y\left(-\frac{1}{2} y z+\frac{1}{2} z\right) \\
& =-y z-z \\
& \rightarrow_{g_{4}}-y z-z-2\left(-\frac{1}{2} y z+\frac{1}{2} z\right) \\
& =-2 z .
\end{aligned}
$$

These reductions enable us to give the following Logged Gröbner Basis for $F$.

| Member of $G$ | Logged Representation |
| :--- | :--- |
| $g_{1}=x y-z$ | $f_{1}$ |
| $g_{2}=2 x+y z+z$ | $f_{2}$ |
| $g_{3}=x+y z$ | $f_{3}$ |
| $g_{4}=-\frac{1}{2} y z+\frac{1}{2} z$ | $\frac{1}{2} f_{2}-f_{3}$ |
| $g_{5}=-2 z^{2}$ | $z f_{1}+(x-z) f_{2}+(-2 x+z) f_{3}$ |
| $g_{6}=-z$ | $f_{1}+(-y-1) f_{2}+(y+2) f_{3}$ |

## Chapter 3

## Noncommutative Gröbner Bases

Once the potential of Gröbner Basis theory started to be realised in the 1970's, it was only natural to try to generalise the theory to related areas such as noncommutative polynomial rings. In 1986, Teo Mora published a paper [45] giving an algorithm for constructing a noncommutative Gröbner Basis. This work built upon the work of George Bergman; in particular his "diamond lemma for ring theory" [8].

In this chapter, we will describe Mora's algorithm and the theory behind it, in many ways giving a 'noncommutative version' of the previous chapter. This means that some material from the previous chapter will be duplicated; this however will be justified when the subtle differences between the cases becomes apparent, differences that are all too often overlooked when an 'easy generalisation' is made!

As in the previous chapter, we will consider the theory from the point of view of Spolynomials, in particular defining a noncommutative Gröbner Basis as a set of polynomials for which the S-polynomials all reduce to zero. At the end of the chapter, in order to give a flavour of a noncommutative Gröbner Basis program, we will give an extended example of the computation of a noncommutative Gröbner Basis, taking advantage of some of the improvements to Mora's algorithm such as Buchberger's criteria and selection strategies.

### 3.1 Overlaps

For a (two-sided) ideal $J$ over a noncommutative polynomial ring, the concept of a Gröbner Basis for $J$ remains the same: it is a set of polynomials $G$ generating $J$ such that remainders with respect to $G$ are unique. How we obtain that Gröbner Basis also remains the same (we add S-polynomials to an initial basis as required); the difference comes in the definition of an S-polynomial.

Recall (from Section 2.1) that the purpose of an S-polynomial S-pol $\left(p_{1}, p_{2}\right)$ is to ensure that any polynomial $p$ reducible by both $p_{1}$ and $p_{2}$ has a unique remainder when divided by a set of polynomials containing $p_{1}$ and $p_{2}$. In the commutative case, there is only one way to divide $p$ by $p_{1}$ or $p_{2}$ (giving reductions $p-t_{1} p_{1}$ or $p-t_{2} p_{2}$ respectively, where $t_{1}$ and $t_{2}$ are terms); this means that there is only one S-polynomial for each pair of polynomials. In the noncommutative case however, a polynomial may divide another polynomial in many different ways (for example the polynomial $x y x-z$ divides the polynomial $x y x y x+4 x^{2}$ in two different ways, giving reductions $z y x+4 x^{2}$ and $x y z+4 x^{2}$ ). For this reason, we do not have a fixed number of S-polynomials for each pair ( $p_{1}, p_{2}$ ) of polynomials in the noncommutative case - that number will depend on the number of overlaps between the lead monomials of $p_{1}$ and $p_{2}$.

In order to explain what an overlap is, we first need the following preliminary definitions allowing us to select a particular part of a noncommutative monomial.

Definition 3.1.1 Consider a monomial $m$ of degree $d$ over a noncommutative polynomial ring $\mathcal{R}$.

- Let Prefix $(m, i)$ denote the prefix of $m$ of degree $i$ (where $1 \leqslant i \leqslant d$ ). For example, $\operatorname{Prefix}\left(x^{2} y z, 3\right)=x^{2} y ; \operatorname{Prefix}\left(z y x^{2}, 1\right)=z$ and $\operatorname{Prefix}\left(y^{2} z x, 4\right)=y^{2} z x$.
- Let Suffix $(m, i)$ denote the suffix of $m$ of degree $i$ (where $1 \leqslant i \leqslant d$ ). For example, $\operatorname{Suffix}\left(x^{2} y z, 3\right)=x y z ; \operatorname{Suffix}\left(z y x^{2}, 1\right)=x$ and $\operatorname{Suffix}\left(y^{2} z x, 4\right)=y^{2} z x$.
- Let Subword $(m, i, j)$ denote the subword of $m$ starting at position $i$ and finishing at position $j$ (where $1 \leqslant i \leqslant j \leqslant d$ ). For example, $\operatorname{Subword}\left(z y x^{2}, 2,3\right)=y x$; $\operatorname{Subword}\left(z y x^{2}, 3,3\right)=x$ and $\operatorname{Subword}\left(y^{2} z x, 1,4\right)=y^{2} z x$.

Definition 3.1.2 Let $m_{1}$ and $m_{2}$ be two monomials over a noncommutative polynomial ring $\mathcal{R}$ with respective degrees $d_{1} \geqslant d_{2}$. We say that $m_{1}$ and $m_{2}$ overlap if any of the
following conditions are satisfied.
(a) $\operatorname{Prefix}\left(m_{1}, i\right)=\operatorname{Suffix}\left(m_{2}, i\right)\left(1 \leqslant i<d_{2}\right)$;
(b) Subword $\left(m_{1}, i, i+d_{2}-1\right)=m_{2}\left(1 \leqslant i \leqslant d_{1}-d_{2}+1\right)$;
(c) $\operatorname{Suffix}\left(m_{1}, i\right)=\operatorname{Prefix}\left(m_{2}, i\right)\left(1 \leqslant i<d_{2}\right)$.

We will refer to the above overlap types as being prefix, subword and suffix overlaps respectively; we can picture the overlap types as follows.


Remark 3.1.3 We have defined the cases where $m_{2}$ is a prefix or a suffix of $m_{1}$ to be subword overlaps.

Proposition 3.1.4 Let $p$ be a polynomial over a noncommutative polynomial ring $\mathcal{R}$ that is divisible by two polynomials $p_{1}, p_{2} \in \mathcal{R}$, so that $\ell_{1} \mathrm{LM}\left(p_{1}\right) r_{1}=\mathrm{LM}(p)=\ell_{2} \mathrm{LM}\left(p_{2}\right) r_{2}$ for some monomials $\ell_{1}, \ell_{2}, r_{1}, r_{2}$. As positioned in $\operatorname{LM}(p)$, if $\operatorname{LM}\left(p_{1}\right)$ and $\operatorname{LM}\left(p_{2}\right)$ do not overlap, then no matter which of the two reductions of $p$ we apply first, we can always obtain a common remainder.

Proof: We picture the situation as follows ( $u$ is a monomial).


We construct the common remainder by using $p_{2}$ to divide the remainder we obtain by dividing $p$ by $p_{1}$ (and vice versa).

Reduction by $p_{1}$ first

$$
\begin{aligned}
p & \rightarrow p-\left(\operatorname{LC}(p) \operatorname{LC}\left(p_{1}\right)^{-1}\right) \ell_{1} p_{1} r_{1} \\
& =(p-\operatorname{LT}(p))-\left(\operatorname{LC}(p) \operatorname{LC}\left(p_{1}\right)^{-1}\right) \ell_{1}\left(p_{1}-\operatorname{LT}\left(p_{1}\right)\right) r_{1} \\
& =(p-\operatorname{LT}(p))-\left(\mathrm{LC}(p) \operatorname{LC}\left(p_{1}\right)^{-1}\right) \ell_{1}\left(p_{1}-\operatorname{LT}\left(p_{1}\right)\right) u \operatorname{LM}\left(p_{2}\right) r_{2} \\
& \xrightarrow{*}(p-\operatorname{LT}(p))-\left(\mathrm{LC}(p) \operatorname{LC}\left(p_{1}\right)^{-1} \mathrm{LC}\left(p_{2}\right)^{-1}\right) \ell_{1}\left(p_{1}-\operatorname{LT}\left(p_{1}\right)\right) u\left(p_{2}-\operatorname{LT}\left(p_{2}\right)\right) r_{2}
\end{aligned}
$$

Reduction by $p_{2}$ first

$$
\begin{aligned}
p & \rightarrow p-\left(\operatorname{LC}(p) \operatorname{LC}\left(p_{2}\right)^{-1}\right) \ell_{2} p_{2} r_{2} \\
& =(p-\operatorname{LT}(p))-\left(\operatorname{LC}(p) \mathrm{LC}\left(p_{2}\right)^{-1}\right) \ell_{2}\left(p_{2}-\operatorname{LT}\left(p_{2}\right)\right) r_{2} \\
& =(p-\operatorname{LT}(p))-\left(\operatorname{LC}(p) \mathrm{LC}\left(p_{2}\right)^{-1}\right) \ell_{1} \operatorname{LM}\left(p_{1}\right) u\left(p_{2}-\operatorname{LT}\left(p_{2}\right)\right) r_{2} \\
& \xrightarrow{*}(p-\operatorname{LT}(p))-\left(\operatorname{LC}(p) \mathrm{LC}\left(p_{1}\right)^{-1} \mathrm{LC}\left(p_{2}\right)^{-1}\right) \ell_{1}\left(p_{1}-\operatorname{LT}\left(p_{1}\right)\right) u\left(p_{2}-\operatorname{LT}\left(p_{2}\right)\right) r_{2}
\end{aligned}
$$

Let $p, p_{1}, p_{2}, \ell_{1}, \ell_{2}, r_{1}$ and $r_{2}$ be as in Proposition 3.1.4. As positioned in $\operatorname{LM}(p)$, in general the lead monomials of $p_{1}$ and $p_{2}$ may or may not overlap, giving four different possibilities, each of which is illustrated by an example in the following table.

| $\mathrm{LM}(p)$ | $\ell_{1}$ | $\mathrm{LM}\left(p_{1}\right)$ | $r_{1}$ | $\ell_{2}$ | $\mathrm{LM}\left(p_{2}\right)$ | $r_{2}$ | Overlap? |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| $x^{2} y z x y^{3}$ | $x^{2} y z$ | $x y^{3}$ | 1 | $x^{2} y$ | $z x$ | $y^{3}$ | Prefix overlap |
| $x^{2} y z x y^{3}$ | $x$ | $x y z x y$ | $y^{2}$ | $x^{2}$ | $y z x$ | $y^{3}$ | Subword overlap |
| $x^{2} y z x y^{3}$ | $x$ | $x y z$ | $x y^{3}$ | $x^{2} y$ | $z x$ | $y^{3}$ | Suffix overlap |
| $x^{2} y z x y^{3}$ | $x^{2}$ | $y$ | $z x y^{3}$ | $x^{2} y z$ | $x y^{2}$ | $y$ | No overlap |

In the cases that $\mathrm{LM}\left(p_{1}\right)$ and $\mathrm{LM}\left(p_{2}\right)$ do overlap, we are not guaranteed to be able to obtain a common remainder when we divide $p$ by both $p_{1}$ and $p_{2}$. To counter this, we introduce (as in the commutative case) an S-polynomial into our dividing set to ensure a common remainder, requiring one S -polynomial for every possible way that $\operatorname{LM}\left(p_{1}\right)$ and $\operatorname{LM}\left(p_{2}\right)$ overlap, including self overlaps (where $p_{1}=p_{2}$, for example Prefix $(x y x, 1)=$ Suffix $(x y x, 1))$.

Definition 3.1.5 Let the lead monomials of two polynomials $p_{1}$ and $p_{2}$ overlap in such a way that $\ell_{1} \mathrm{LM}\left(p_{1}\right) r_{1}=\ell_{2} \mathrm{LM}\left(p_{2}\right) r_{2}$, where $\ell_{1}, \ell_{2}, r_{1}$ and $r_{2}$ are monomials chosen so that at least one of $\ell_{1}$ and $\ell_{2}$ and at least one of $r_{1}$ and $r_{2}$ is equal to the unit monomial. The $S$-polynomial associated with this overlap is given by the expression

$$
\operatorname{S-pol}\left(\ell_{1}, p_{1}, \ell_{2}, p_{2}\right)=c_{1} \ell_{1} p_{1} r_{1}-c_{2} \ell_{2} p_{2} r_{2},
$$

where $c_{1}=\operatorname{LC}\left(p_{2}\right)$ and $c_{2}=\operatorname{LC}\left(p_{1}\right)$.
Remark 3.1.6 The monomials $\ell_{1}$ and $\ell_{2}$ are included in the notation $\operatorname{S-pol}\left(\ell_{1}, p_{1}, \ell_{2}, p_{2}\right)$ in order to differentiate between distinct S-polynomials involving the two polynomials $p_{1}$ and $p_{2}$ (there is no need to include $r_{1}$ and $r_{2}$ in the notation because $r_{1}$ and $r_{2}$ are uniquely determined by $\ell_{1}$ and $\ell_{2}$ respectively).

Example 3.1.7 Consider the polynomial $p:=x y z+2 y$ and the set of polynomials $P:=$ $\left\{p_{1}, p_{2}\right\}=\{x y-z, y z-x\}$, all polynomials being ordered by DegLex and originating from the polynomial ring $\mathbb{Q}\langle x, y, z\rangle$. We see that $p$ is divisible (in one way) by both of the polynomials in $P$, giving remainders $z^{2}+2 y$ and $x^{2}+2 y$ respectively, both of which are irreducible by $P$. It follows that $p$ does not have a unique remainder with respect to $P$.

Because there is only one overlap involving the lead monomials of $p_{1}$ and $p_{2}$, namely $\operatorname{Suffix}(x y, 1)=\operatorname{Prefix}(y z, 1)$, there is only one S-polynomial for the set $P$, which is the polynomial $(x y-z) z-x(y z-x)=x^{2}-z^{2}$. When we add this polynomial to the set $P$, we see that the remainder of $p$ with respect to the enlarged $P$ is now unique, as the remainder of $p$ with respect to $p_{2}$ (the polynomial $x^{2}+2 y$ ) is now reducible by our new polynomial, giving a new remainder $z^{2}+2 y$ which agrees with the remainder of $p$ with respect to $p_{1}$.

Let us now give a definition of a noncommutative Gröbner Basis in terms of S-polynomials.
Definition 3.1.8 Let $G=\left\{g_{1}, \ldots, g_{m}\right\}$ be a basis for an ideal $J$ over a noncommutative polynomial ring $\mathcal{R}=R\left\langle x_{1}, \ldots, x_{n}\right\rangle$. If all the S -polynomials involving members of $G$ reduce to zero using $G$, then $G$ is a noncommutative Gröbner Basis for $J$.

Theorem 3.1.9 Given any polynomial $p$ over a polynomial ring $\mathcal{R}=R\left\langle x_{1}, \ldots, x_{n}\right\rangle$, the remainder of the division of $p$ by a basis $G$ for an ideal $J$ in $\mathcal{R}$ is unique if and only if $G$ is a Gröbner Basis.

Proof: $(\Rightarrow)$ Following the proof of Theorem 2.1.5, we need to show that the division process is locally confluent, that is if there are polynomials $f, f_{1}, f_{2} \in \mathcal{R}$ with $f_{1}=$ $f-\ell_{1} g_{1} r_{1}$ and $f_{2}=f-\ell_{2} g_{2} r_{2}$ for terms $\ell_{1}, \ell_{2}, r_{1}, r_{2}$ and $g_{1}, g_{2} \in G$, then there exists a polynomial $f_{3} \in \mathcal{R}$ such that both $f_{1}$ and $f_{2}$ reduce to $f_{3}$. As before, this is equivalent to showing that the polynomial $f_{2}-f_{1}=\ell_{1} g_{1} r_{1}-\ell_{2} g_{2} r_{2}$ reduces to zero.

If $\operatorname{LT}\left(\ell_{1} g_{1} r_{1}\right) \neq \operatorname{LT}\left(\ell_{2} g_{2} r_{2}\right)$, then the remainders $f_{1}$ and $f_{2}$ are obtained by cancelling off different terms of the original $f$ (the reductions of $f$ are disjoint), so it is possible, assuming (without loss of generality) that $\mathrm{LT}\left(\ell_{1} g_{1} r_{1}\right)>\operatorname{LT}\left(\ell_{2} g_{2} r_{2}\right)$, to directly reduce the polynomial $f_{2}-f_{1}=\ell_{1} g_{1} r_{1}-\ell_{2} g_{2} r_{2}$ in the following manner: $\ell_{1} g_{1} r_{1}-\ell_{2} g_{2} r_{1} \rightarrow g_{1}$ $-\ell_{2} g_{2} r_{2} \rightarrow g_{2} 0$.

On the other hand, if $\operatorname{LT}\left(\ell_{1} g_{1} r_{1}\right)=\operatorname{LT}\left(\ell_{2} g_{2} r_{2}\right)$, then the reductions of $f$ are not disjoint (as the same term $t$ from $f$ is cancelled off during both reductions), so that the term $t$ does not appear in the polynomial $\ell_{1} g_{1} r_{1}-\ell_{2} g_{2} r_{2}$. However, the monomial $\mathrm{LM}(t)$ must contain the monomials $\operatorname{LM}\left(g_{1}\right)$ and $\operatorname{LM}\left(g_{2}\right)$ as subwords if both $g_{1}$ and $g_{2}$ cancel off the term $t$, so it follows that $\mathrm{LM}\left(g_{1}\right)$ and $\mathrm{LM}\left(g_{2}\right)$ will either overlap or not overlap in $\mathrm{LM}(t)$. If they do not overlap, then we know from Proposition 3.1.4 that $f_{1}$ and $f_{2}$ will have a common remainder $\left(f_{1} \xrightarrow{*} f_{3}\right.$ and $\left.f_{2} \xrightarrow{*} f_{3}\right)$, so that $f_{2}-f_{1} \xrightarrow{*} f_{3}-f_{3}=0$. Otherwise, because of the overlap between $\mathrm{LM}\left(g_{1}\right)$ and $\mathrm{LM}\left(g_{2}\right)$, the polynomial $\ell_{1} g_{1} r_{1}-\ell_{2} g_{2} r_{2}$ will be a multiple of an S-polynomial, say $\ell_{1} g_{1} r_{1}-\ell_{2} g_{2} r_{2}=\ell_{3}\left(\mathrm{~S}-\mathrm{pol}\left(\ell_{1}^{\prime}, g_{1}, \ell_{2}^{\prime}, g_{2}\right)\right) r_{3}$ for some terms $\ell_{3}, r_{3}$ and some monomials $\ell_{1}^{\prime}, \ell_{2}^{\prime}$. But $G$ is a Gröbner Basis, so the S-polynomial S-pol $\left(\ell_{1}^{\prime}, g_{1}, \ell_{2}^{\prime}, g_{2}\right)$ will reduce to zero, and hence by extension the polynomial $\ell_{1} g_{1} r_{1}-\ell_{2} g_{2} r_{2}$ will also reduce to zero.
$(\Leftarrow)$ As all S-polynomials are members of the ideal $J$, to complete the proof it is sufficient to show that there is always a reduction path of an arbitrary member of the ideal that leads to a zero remainder (the uniqueness of remainders will then imply that members of the ideal always reduce to zero). Let $f \in J=\langle G\rangle$. Then, by definition, there exist $g_{i} \in G$ (not necessarily all different) and terms $\ell_{i}, r_{i} \in \mathcal{R}$ (where $1 \leqslant i \leqslant j$ ) such that

$$
f=\sum_{i=1}^{j} \ell_{i} g_{i} r_{i} .
$$

We proceed by induction on $j$. If $j=1$, then $f=\ell_{1} g_{1} r_{1}$, and it is clear that we can use $g_{1}$ to reduce $f$ to give a zero remainder $\left(f \rightarrow_{g_{1}} f-\ell_{1} g_{1} r_{1}=0\right)$. Assume that the result is true for $j=k$, and let us look at the case $j=k+1$, so that

$$
f=\left(\sum_{i=1}^{k} \ell_{i} g_{i} r_{i}\right)+\ell_{k+1} g_{k+1} r_{k+1} .
$$

By the inductive hypothesis, $\sum_{i=1}^{k} \ell_{i} g_{i} r_{i}$ is a member of the ideal that reduces to zero. The polynomial $f$ therefore reduces to the polynomial $f^{\prime}:=\ell_{k+1} g_{k+1} r_{k+1}$, and we can
now use $g_{k+1}$ to reduce $f^{\prime}$ to give a zero remainder $\left(f^{\prime} \rightarrow_{g_{k+1}} f^{\prime}-\ell_{k+1} g_{k+1} r_{k+1}=0\right)$.
Remark 3.1.10 The above Theorem forms part of Bergman's Diamond Lemma [8, Theorem 1.2].

### 3.2 Mora's Algorithm

Let us now consider the following pseudo code representing Mora's algorithm for computing noncommutative Gröbner Bases [45].

```
Algorithm 5 Mora's Noncommutative Gröbner Basis Algorithm
Input: A Basis \(F=\left\{f_{1}, f_{2}, \ldots, f_{m}\right\}\) for an ideal \(J\) over a noncommutative polynomial
    ring \(R\left\langle x_{1}, \ldots x_{n}\right\rangle\); an admissible monomial ordering O .
Output: A Gröbner Basis \(G=\left\{g_{1}, g_{2}, \ldots, g_{p}\right\}\) for \(J\) (in the case of termination).
    Let \(G=F\) and let \(A=\emptyset\);
    For each pair of polynomials \(\left(g_{i}, g_{j}\right)\) in \(G(i \leqslant j)\), add an S-polynomial S-pol \(\left(\ell_{1}, g_{i}, \ell_{2}, g_{j}\right)\)
    to \(A\) for each overlap \(\ell_{1} \mathrm{LM}\left(g_{i}\right) r_{1}=\ell_{2} \mathrm{LM}\left(g_{j}\right) r_{2}\) between the lead monomials of \(\mathrm{LM}\left(g_{i}\right)\)
    and \(\mathrm{LM}\left(g_{j}\right)\).
    while ( \(A\) is not empty) do
        Remove the first entry \(s_{1}\) from \(A\);
        \(s_{1}^{\prime}=\operatorname{Rem}\left(s_{1}, G\right)\);
        if \(\left(s_{1}^{\prime} \neq 0\right)\) then
            Add \(s_{1}^{\prime}\) to \(G\) and then (for all \(g_{i} \in G\) ) add all the S-polynomials of the form
            \(\mathrm{S}-\mathrm{pol}\left(\ell_{1}, g_{i}, \ell_{2}, s_{1}^{\prime}\right)\) to \(A\);
        end if
    end while
    return \(G\);
```

Structurally, Mora's algorithm is virtually identical to Buchberger's algorithm, in that we compute and reduce each S-polynomial in turn; we add a reduced S-polynomial to our basis if it does not reduce to zero; and we continue until all S-polynomials reduce to zero - exactly as in Algorithm 3. Despite this, there are major differences from an implementation standpoint, not least in the fact that noncommutative polynomials are much more difficult to handle on a computer; and noncommutative S-polynomials need more complicated data structures. This may explain why implementations of the noncommutative Gröbner Basis algorithm are currently sparser than those for the commutative algorithm;
and also why such implementations often impose restrictions on the problems that can be handled - Bergman [6] for instance only allows input bases which are homogeneous.

### 3.2.1 Termination

In the commutative case, Dickson's Lemma and Hilbert's Basis Theorem allow us to prove that Buchberger's algorithm always terminates for all possible inputs. It is a fact however that Mora's algorithm does not terminate for all possible inputs (so that an ideal may have an infinite Gröbner Basis in general) because there is no analogue of Dickson's Lemma for noncommutative monomial ideals.

Proposition 3.2.1 Not all noncommutative monomial ideals are finitely generated.

Proof: Assume to the contrary that all noncommutative monomial ideals are finitely generated, and consider an ascending chain of such ideals $J_{1} \subseteq J_{2} \subseteq \cdots$. By our assumption, the ideal $J=\cup J_{i}$ (for $i \geqslant 1$ ) will be finitely generated, which means that there must be some $k \geqslant 1$ such that $J_{k}=J_{k+1}=\cdots$. For a counterexample, let $\mathcal{R}=\mathbb{Q}\langle x, y\rangle$ be a noncommutative polynomial ring, and define $J_{i}$ (for $i \geqslant 1$ ) to be the ideal in $\mathcal{R}$ generated by the set of monomials $\left\{x y x, x y^{2} x, \ldots, x y^{i} x\right\}$. Because no member of this set is a multiple of any other member of the set, it is clear that there cannot be a $k \geqslant 1$ such that $J_{k}=J_{k+1}=\cdots$ because $x y^{k+1} x \in J_{k+1}$ and $x y^{k+1} x \notin J_{k}$ for all $k \geqslant 1$.

Another way of explaining why Mora's algorithm does not terminate comes from considering the link between noncommutative Gröbner Bases and the Knuth-Bendix Critical Pairs Completion Algorithm for monoid rewrite systems [39], an algorithm that attempts to find a complete rewrite system for any given monoid presentation. Because Mora's algorithm can be used to emulate the Knuth-Bendix algorithm (for the details, see for example [33]), if we assume that Mora's algorithm always terminates, then we have found a way to solve the word problem for monoids (so that we can determine whether any word in a given monoid is equal to the identity word); this however contradicts the fact that the word problem is actually an unsolvable problem (so that it is impossible to define an algorithm that can tell whether two words in a given monoid are identical).

### 3.3 Reduced Gröbner Bases

Definition 3.3.1 Let $G=\left\{g_{1}, \ldots, g_{p}\right\}$ be a Gröbner Basis for an ideal over a polynomial ring $R\left\langle x_{1}, \ldots, x_{n}\right\rangle . G$ is a reduced Gröbner Basis if the following conditions are satisfied.
(a) $\mathrm{LC}\left(g_{i}\right)=1_{R}$ for all $g_{i} \in G$.
(b) No term in any polynomial $g_{i} \in G$ is divisible by any $\operatorname{LT}\left(g_{j}\right), j \neq i$.

Theorem 3.3.2 If there exists a Gröbner Basis $G$ for an ideal $J$ over a noncommutative polynomial ring, then $J$ has a unique reduced Gröbner Basis.

Proof: Existence. We claim that the following procedure transforms $G$ into a reduced Gröbner Basis $G^{\prime}$.
(i) Multiply each $g_{i} \in G$ by $\operatorname{LC}\left(g_{i}\right)^{-1}$.
(ii) Reduce each $g_{i} \in G$ by $G \backslash\left\{g_{i}\right\}$, removing from $G$ all polynomials that reduce to zero.

It is clear that $G^{\prime}$ satisfies the conditions of Definition 3.3.1, so it remains to show that $G^{\prime}$ is a Gröbner Basis, which we shall do by showing that the application of each step of instruction (ii) above produces a basis which is still a Gröbner Basis.

Let $G=\left\{g_{1}, \ldots, g_{p}\right\}$ be a Gröbner Basis, and let $g_{i}^{\prime}$ be the reduction of an arbitrary $g_{i} \in G$ with respect to $G \backslash\left\{g_{i}\right\}$, carried out as follows (the $\ell_{k}$ and the $r_{k}$ are terms).

$$
\begin{equation*}
g_{i}^{\prime}=g_{i}-\sum_{k=1}^{\kappa} \ell_{k} g_{j_{k}} r_{k} \tag{3.1}
\end{equation*}
$$

Set $H=\left(G \backslash\left\{g_{i}\right\}\right) \cup\left\{g_{i}^{\prime}\right\}$ if $g_{i}^{\prime} \neq 0$, and set $H=G \backslash\left\{g_{i}\right\}$ if $g_{i}^{\prime}=0$. As $G$ is a Gröbner Basis, all S-polynomials involving elements of $G$ reduce to zero using $G$, so there are expressions

$$
\begin{equation*}
c_{b} \ell_{a} g_{a} r_{a}-c_{a} \ell_{b} g_{b} r_{b}-\sum_{u=1}^{\mu} \ell_{u} g_{c_{u}} r_{u}=0 \tag{3.2}
\end{equation*}
$$

for every S-polynomial S-pol $\left(\ell_{a}, g_{a}, \ell_{b}, g_{b}\right)=c_{b} \ell_{a} g_{a} r_{a}-c_{a} \ell_{b} g_{b} r_{b}$, where $c_{a}=\operatorname{LC}\left(g_{a}\right) ; c_{b}=$ $\mathrm{LC}\left(g_{b}\right)$; the $\ell_{u}$ and the $r_{u}$ are terms (for $1 \leqslant u \leqslant \mu$ ); and $g_{a}, g_{b}, g_{c_{u}} \in G$. To show that $H$ is
a Gröbner Basis, we must show that all S-polynomials involving elements of $H$ reduce to zero using $H$. For polynomials $g_{a}, g_{b} \in H$ not equal to $g_{i}^{\prime}$, we can reduce an S-polynomial of the form $\mathrm{S}-\mathrm{pol}\left(\ell_{a}, g_{a}, \ell_{b}, g_{b}\right)$ using the reduction shown in Equation (3.2), substituting for $g_{i}$ from Equation (3.1) if any of the $g_{c_{u}}$ in Equation (3.2) are equal to $g_{i}$. This gives a reduction to zero of S-pol $\left(\ell_{a}, g_{a}, \ell_{b}, g_{b}\right)$ in terms of elements of $H$.

If $g_{i}^{\prime}=0$, our proof is complete. Otherwise consider all S-polynomials $\operatorname{S-pol}\left(\ell_{i}^{\prime}, g_{i}^{\prime}, \ell_{b}, g_{b}\right)$ involving the pair of polynomials $\left(g_{i}^{\prime}, g_{b}\right)$, where $g_{b} \in G \backslash\left\{g_{i}\right\}$. We claim that there exists an S-polynomial S-pol $\left(\ell_{1}, g_{i}, \ell_{2}, g_{b}\right)=c_{b} \ell_{1} g_{i} r_{1}-c_{i} \ell_{2} g_{b} r_{2}$ such that $\operatorname{S-pol}\left(\ell_{i}^{\prime}, g_{i}^{\prime}, \ell_{b}, g_{b}\right)=$ $c_{b} \ell_{1} g_{i}^{\prime} r_{1}-c_{i} \ell_{2} g_{b} r_{2}$. To prove this claim, it is sufficient to show that $\operatorname{LT}\left(g_{i}\right)=\operatorname{LT}\left(g_{i}^{\prime}\right)$. Assume for a contradiction that $\mathrm{LT}\left(g_{i}\right) \neq \mathrm{LT}\left(g_{i}^{\prime}\right)$. It follows that during the reduction of $g_{i}$ we were able to reduce its lead term, so that $\operatorname{LT}\left(g_{i}\right)=\ell \operatorname{LT}\left(g_{j}\right) r$ for some terms $\ell$ and $r$ and some $g_{j} \in G$. Because $\operatorname{LM}\left(g_{i}-\ell g_{j} r\right)<\operatorname{LM}\left(g_{i}\right)$, the polynomial $g_{i}-\ell g_{j} r$ must reduce to zero without using $g_{i}$, so that $g_{i}^{\prime}=0$, giving a contradiction.

It remains to show that $\mathrm{S}-\mathrm{pol}\left(\ell_{i}^{\prime}, g_{i}^{\prime}, \ell_{b}, g_{b}\right) \rightarrow_{H} 0$. We know that $\mathrm{S}-\mathrm{pol}\left(\ell_{1}, g_{i}, \ell_{2}, g_{b}\right)=$ $c_{b} \ell_{1} g_{i} r_{1}-c_{i} \ell_{2} g_{b} r_{2} \rightarrow{ }_{G} 0$, and Equation (3.2) tells us that $c_{b} \ell_{1} g_{i} r_{1}-c_{i} \ell_{2} g_{b} r_{2}-\sum_{u=1}^{\mu} \ell_{u} g_{c_{u}} r_{u}=$ 0 . Substituting for $g_{i}$ from Equation (3.1), we obtain ${ }^{1}$

$$
c_{b} \ell_{1}\left(g_{i}^{\prime}+\sum_{k=1}^{\kappa} \ell_{k} g_{j_{k}} r_{k}\right) r_{1}-c_{i} \ell_{2} g_{b} r_{2}-\sum_{u=1}^{\mu} \ell_{u} g_{c_{u}} r_{u}=0
$$

or

$$
c_{b} \ell_{1} g_{i}^{\prime} r_{1}-c_{i} \ell_{2} g_{b} r_{2}-\left(\sum_{u=1}^{\mu} \ell_{u} g_{c_{u}} r_{u}-\sum_{k=1}^{\kappa} c_{b} \ell_{1} \ell_{k} g_{j_{k}} r_{k} r_{1}\right)=0,
$$

which implies that $\mathrm{S}-\mathrm{pol}\left(\ell_{i}^{\prime}, g_{i}^{\prime}, \ell_{b}, g_{b}\right) \rightarrow_{H} 0$. The only other case to consider is the case of an S-polynomial coming from a self overlap involving $\mathrm{LM}\left(g_{i}^{\prime}\right)$. But because we now know that $\operatorname{LT}\left(g_{i}^{\prime}\right)=\operatorname{LT}\left(g_{i}\right)$, we can use exactly the same argument as above to show that the S-polynomial S-pol $\left(\ell_{1}, g_{i}^{\prime}, \ell_{2}, g_{i}^{\prime}\right)$ reduces to zero using $H$ because an S-polynomial of the form $\operatorname{S-pol}\left(\ell_{1}, g_{i}, \ell_{2}, g_{i}\right)$ will exist.

Uniqueness. Assume for a contradiction that $G=\left\{g_{1}, \ldots, g_{p}\right\}$ and $H=\left\{h_{1}, \ldots, h_{q}\right\}$ are two reduced Gröbner Bases for an ideal $J$, with $G \neq H$. Let $g_{i}$ be an arbitrary element from $G$ (where $1 \leqslant i \leqslant p$ ). Because $g_{i}$ is a member of the ideal, then $g_{i}$ must reduce to zero using $H$ ( $H$ is a Gröbner Basis). This means that there must exist a polynomial

[^5]$h_{j} \in H$ such that $\operatorname{LT}\left(h_{j}\right) \mid \operatorname{LT}\left(g_{i}\right)$. If $\operatorname{LT}\left(h_{j}\right) \neq \operatorname{LT}\left(g_{i}\right)$, then $\ell \times \operatorname{LT}\left(h_{j}\right) \times r=\operatorname{LT}\left(g_{i}\right)$ for some monomials $\ell$ and $r$, at least one of which is not equal to the unit monomial. But $h_{j}$ is also a member of the ideal, so it must reduce to zero using $G$. Therefore there exists a polynomial $g_{k} \in G$ such that $\operatorname{LT}\left(g_{k}\right) \mid \operatorname{LT}\left(h_{j}\right)$, which implies that $\operatorname{LT}\left(g_{k}\right) \mid \operatorname{LT}\left(g_{i}\right)$, with $k \neq i$. This contradicts condition (b) of Definition 3.3.1 so that $G$ cannot be a reduced Gröbner Basis for $J$ if $\operatorname{LT}\left(h_{j}\right) \neq \operatorname{LT}\left(g_{i}\right)$. From this we deduce that each $g_{i} \in G$ has a corresponding $h_{j} \in H$ such that $\operatorname{LT}\left(g_{i}\right)=\operatorname{LT}\left(h_{j}\right)$. Further, because $G$ and $H$ are assumed to be reduced Gröbner Bases, this is a one-to-one correspondence.

It remains to show that if $\operatorname{LT}\left(g_{i}\right)=\operatorname{LT}\left(h_{j}\right)$, then $g_{i}=h_{j}$. Assume for a contradiction that $g_{i} \neq h_{j}$ and consider the polynomial $g_{i}-h_{j}$. Without loss of generality, assume that $\mathrm{LM}\left(g_{i}-h_{j}\right)$ appears in $g_{i}$. Because $g_{i}-h_{j}$ is a member of the ideal, then there is a polynomial $g_{k} \in G$ such that $\operatorname{LT}\left(g_{k}\right) \mid \operatorname{LT}\left(g_{i}-h_{j}\right)$. But this again contradicts condition (b) of Definition 3.3.1, as we have shown that there is a term in $g_{i}$ that is divisible by $\mathrm{LT}\left(g_{k}\right)$ for some $k \neq i$. It follows that $G$ cannot be a reduced Gröbner Basis if $g_{i} \neq h_{j}$, which means that $G=H$ and therefore reduced Gröbner Bases are unique.

As in the commutative case, we may refine the procedure for finding a unique reduced Gröbner Basis (as given in the proof of Theorem 3.3.2) by removing from the Gröbner Basis all polynomials whose lead monomials are multiples of the lead monomials of other Gröbner Basis elements. This leads to the definition of Algorithm 6.

### 3.4 Improvements to Mora's Algorithm

In Section 2.5, we surveyed some of the numerous improvements of Buchberger's algorithm. Let us now demonstrate that many of these improvements can also be applied in the noncommutative case.

### 3.4.1 Buchberger's Criteria

In the commutative case, Buchberger's first criterion states that we can ignore any Spolynomial S-pol $(f, g)$ in which $\operatorname{lcm}(\mathrm{LM}(f), \mathrm{LM}(g))=\mathrm{LM}(f) \mathrm{LM}(g)$. In the noncommutative case, this translates as saying that we can ignore any 'S-polynomial' S-pol ( $\left.\ell_{1}, f, \ell_{2}, g\right)=$ $\mathrm{LC}(g) \ell_{1} f r_{1}-\mathrm{LC}(f) \ell_{2} g r_{2}$ such that $\mathrm{LM}(f)$ and $\mathrm{LM}(g)$ do not overlap in the monomial $\ell_{1} \mathrm{LM}(f) r_{1}=\ell_{2} \mathrm{LM}(g) r_{2}$. We can certainly show that such an 'S-polynomial' will reduce to zero by utilising Proposition 3.1.4, but we will never be able to use this result as, by

```
Algorithm 6 The Noncommutative Unique Reduced Gröbner Basis Algorithm
Input: A Gröbner Basis \(G=\left\{g_{1}, g_{2}, \ldots, g_{m}\right\}\) for an ideal \(J\) over a noncommutative
    polynomial ring \(R\left\langle x_{1}, \ldots x_{n}\right\rangle\); an admissible monomial ordering O .
Output: The unique reduced Gröbner Basis \(G^{\prime}=\left\{g_{1}^{\prime}, g_{2}^{\prime}, \ldots, g_{p}^{\prime}\right\}\) for \(J\).
    \(G^{\prime}=\emptyset ;\)
    for each \(g_{i} \in G\) do
        Multiply \(g_{i}\) by \(\mathrm{LC}\left(g_{i}\right)^{-1}\);
        if \(\left(\operatorname{LM}\left(g_{i}\right)=\ell \operatorname{LM}\left(g_{j}\right) r\right.\) for some monomials \(\ell, r\) and some \(\left.g_{j} \in G\left(g_{j} \neq g_{i}\right)\right)\) then
            \(G=G \backslash\left\{g_{i}\right\} ;\)
        end if
    end for
    for each \(g_{i} \in G\) do
        \(g_{i}^{\prime}=\operatorname{Rem}\left(g_{i},\left(G \backslash\left\{g_{i}\right\}\right) \cup G^{\prime}\right) ;\)
        \(G=G \backslash\left\{g_{i}\right\} ; G^{\prime}=G^{\prime} \cup\left\{g_{i}^{\prime}\right\} ;\)
    end for
    return \(G^{\prime}\);
```

definition, an S-polynomial is only defined when we have an overlap between $\operatorname{LM}(f)$ and $\mathrm{LM}(g)$. It follows that an 'S-polynomial' of the above type will never occur in Mora's algorithm, and so Buchberger's first criterion is redundant in the noncommutative case. The same cannot be said of his second criterion however, which certainly does improve the efficiency of Mora's algorithm.

Proposition 3.4.1 (Buchberger's Second Criterion) Let $f, g$ and $h$ be three members of a finite set of polynomials $P$ over a noncommutative polynomial ring, and consider an $S$-polynomial of the form

$$
\begin{equation*}
\operatorname{S-pol}\left(\ell_{1}, f, \ell_{2}, g\right)=c_{2} \ell_{1} f r_{1}-c_{1} \ell_{2} g r_{2} . \tag{3.3}
\end{equation*}
$$

If $\operatorname{LM}(h) \mid \ell_{1} \operatorname{LM}(f) r_{1}$, so that

$$
\begin{equation*}
\ell_{1} \mathrm{LM}(f) r_{1}=\ell_{3} \mathrm{LM}(h) r_{3}=\ell_{2} \mathrm{LM}(g) r_{2} \tag{3.4}
\end{equation*}
$$

for some monomials $\ell_{3}, r_{3}$, then S -pol $\left(\ell_{1}, f, \ell_{2}, g\right) \rightarrow_{P} 0$ if all $S$-polynomials corresponding to overlaps (as placed in the monomial $\left.\ell_{1} \mathrm{LM}(f) r_{1}\right)$ between $\mathrm{LM}(h)$ and either $\mathrm{LM}(f)$ or $\mathrm{LM}(g)$ reduce to zero using $P$.

Proof (cf. [37], Appendix A): To be able to describe an S-polynomial corresponding to an overlap (as placed in the monomial $\left.\ell_{1} \operatorname{LM}(f) r_{1}\right)$ between $\operatorname{LM}(h)$ and either $\operatorname{LM}(f)$ or $\mathrm{LM}(g)$, we introduce the following notation.

- Let $\ell_{13}$ be the monomial corresponding to the common prefix of $\ell_{1}$ and $\ell_{3}$ of maximal degree, so that $\ell_{1}=\ell_{13} \ell_{1}^{\prime}$ and $\ell_{3}=\ell_{13} \ell_{3}^{\prime}$. (Here, and similarly below, if there is no common prefix of $\ell_{1}$ and $\ell_{3}$, then $\ell_{13}=1, \ell_{1}^{\prime}=\ell_{1}$ and $\ell_{3}^{\prime}=\ell_{3}$.)
- Let $\ell_{23}$ be the monomial corresponding to the common prefix of $\ell_{2}$ and $\ell_{3}$ of maximal degree, so that $\ell_{2}=\ell_{23} \ell_{2}^{\prime \prime}$ and $\ell_{3}=\ell_{23} \ell_{3}^{\prime \prime}$.
- Let $r_{13}$ be the monomial corresponding to the common suffix of $r_{1}$ and $r_{3}$ of maximal degree, so that $r_{1}=r_{1}^{\prime} r_{13}$ and $r_{3}=r_{3}^{\prime} r_{13}$.
- Let $r_{23}$ be the monomial corresponding to the common suffix of $r_{2}$ and $r_{3}$ of maximal degree, so that $r_{2}=r_{2}^{\prime \prime} r_{23}$ and $r_{3}=r_{3}^{\prime \prime} r_{23}$.

We can now manipulate Equation (3.3) as follows (where $c_{3}=\operatorname{LC}(h)$ ).

$$
\begin{aligned}
c_{3}\left(\mathrm{~S}-\mathrm{pol}\left(\ell_{1}, f, \ell_{2}, g\right)\right)= & c_{3} c_{2} \ell_{1} f r_{1}-c_{3} c_{1} \ell_{2} g r_{2} \\
= & c_{3} c_{2} \ell_{1} f r_{1}-c_{1} c_{2} \ell_{3} h r_{3}+c_{1} c_{2} \ell_{3} h r_{3}-c_{3} c_{1} \ell_{2} g r_{2} \\
= & c_{2}\left(c_{3} \ell_{1} f r_{1}-c_{1} \ell_{3} h r_{3}\right)-c_{1}\left(c_{3} \ell_{2} g r_{2}-c_{2} \ell_{3} h r_{3}\right) \\
= & c_{2}\left(c_{3} \ell_{13} \ell_{1}^{\prime} f r_{1}^{\prime} r_{13}-c_{1} \ell_{13} \ell_{3}^{\prime} h r_{3}^{\prime} r_{13}\right) \\
& -c_{1}\left(c_{3} \ell_{23} \ell_{2}^{\prime \prime} g r_{2}^{\prime \prime} r_{23}-c_{2} \ell_{23} \ell_{3}^{\prime \prime} h r_{3}^{\prime \prime} r_{23}\right) \\
= & c_{2} \ell_{13}\left(c_{3} \ell_{1}^{\prime} f r_{1}^{\prime}-c_{1} \ell_{3}^{\prime} h r_{3}^{\prime}\right) r_{13}-c_{1} \ell_{23}\left(c_{3} \ell_{2}^{\prime \prime} g r_{2}^{\prime \prime}-c_{2} \ell_{3}^{\prime \prime} h r_{3}^{\prime \prime}\right) r_{23} .
\end{aligned}
$$

As placed in $\ell_{1} \mathrm{LM}(f) r_{1}=\ell_{3} \mathrm{LM}(h) r_{3}$, if $\mathrm{LM}(f)$ and $\mathrm{LM}(h)$ overlap, then the S-polynomial corresponding to this overlap is ${ }^{2} S-\operatorname{pol}\left(\ell_{1}^{\prime}, f, \ell_{3}^{\prime}, h\right)$. Similarly, if $\operatorname{LM}(g)$ and $\operatorname{LM}(h)$ overlap as placed in $\ell_{2} \mathrm{LM}(g) r_{2}=\ell_{3} \mathrm{LM}(h) r_{3}$, then the S-polynomial corresponding to this overlap is $S$-pol $\left(\ell_{2}^{\prime \prime}, g, \ell_{3}^{\prime \prime}, h\right)$. By assumption, these $S$-polynomials reduce to zero using $P$, so there are expressions

$$
\begin{equation*}
c_{3} \ell_{1}^{\prime} f r_{1}^{\prime}-c_{1} \ell_{3}^{\prime} h r_{3}^{\prime}-\sum_{i=1}^{\alpha} u_{i} p_{i} v_{i}=0 \tag{3.5}
\end{equation*}
$$

[^6]and
\[

$$
\begin{equation*}
c_{3} \ell_{2}^{\prime \prime} g r_{2}^{\prime \prime}-c_{2} \ell_{3}^{\prime \prime} h r_{3}^{\prime \prime}-\sum_{j=1}^{\beta} u_{j} p_{j} v_{j}=0 \tag{3.6}
\end{equation*}
$$

\]

where the $u_{i}, v_{i}, u_{j}$ and $v_{j}$ are terms; and $p_{i}, p_{j} \in P$ for all $i$ and $j$. Using Proposition 3.1.4, we can state that these expressions will still exist even if $\operatorname{LM}(f)$ and $\mathrm{LM}(h)$ do not overlap as placed in $\ell_{1} \mathrm{LM}(f) r_{1}=\ell_{3} \mathrm{LM}(h) r_{3}$; and if $\mathrm{LM}(g)$ and $\mathrm{LM}(h)$ do not overlap as placed in $\ell_{2} \mathrm{LM}(g) r_{2}=\ell_{3} \mathrm{LM}(h) r_{3}$. It follows that

$$
\begin{aligned}
c_{3}\left(\mathrm{~S}-\mathrm{pol}\left(\ell_{1}, f, \ell_{2}, g\right)\right) & =c_{2} \ell_{13}\left(c_{3} \ell_{1}^{\prime} f r_{1}^{\prime}-c_{1} \ell_{3}^{\prime} h r_{3}^{\prime}\right) r_{13}-c_{1} \ell_{23}\left(c_{3} \ell_{2}^{\prime \prime} g r_{2}^{\prime \prime}-c_{2} \ell_{3}^{\prime \prime} h r_{3}^{\prime \prime}\right) r_{23} \\
& =c_{2} \ell_{13}\left(\sum_{i=1}^{\alpha} u_{i} p_{i} v_{i}\right) r_{13}-c_{1} \ell_{23}\left(\sum_{j=1}^{\beta} u_{j} p_{j} v_{j}\right) r_{23} \\
& =\sum_{i=1}^{\alpha} c_{2} \ell_{13} u_{i} p_{i} v_{i} r_{13}-\sum_{j=1}^{\beta} c_{1} \ell_{23} u_{j} p_{j} v_{j} r_{23} ; \\
\operatorname{S-pol}\left(\ell_{1}, f, \ell_{2}, g\right) & =\sum_{i=1}^{\alpha} c_{3}^{-1} c_{2} \ell_{13} u_{i} p_{i} v_{i} r_{13}-\sum_{j=1}^{\beta} c_{3}^{-1} c_{1} \ell_{23} u_{j} p_{j} v_{j} r_{23} .
\end{aligned}
$$

To conclude that the S-polynomial S -pol $\left(\ell_{1}, f, \ell_{2}, g\right)$ reduces to zero using $P$, it remains to show that the algebraic expression $-\sum_{i=1}^{\alpha} c_{3}^{-1} c_{2} \ell_{13} u_{i} p_{i} v_{i} r_{13}+\sum_{j=1}^{\beta} c_{3}^{-1} c_{1} \ell_{23} u_{j} p_{j} v_{j} r_{23}$ corresponds to a valid reduction of $\operatorname{S-pol}\left(\ell_{1}, f, \ell_{2}, g\right)$. To do this, it is sufficient to show that no term in either of the summations is greater than the term $\ell_{1} \operatorname{LM}(f) r_{1}$ (so that $\operatorname{LM}\left(\ell_{13} u_{i} p_{i} v_{i} r_{13}\right)<\ell_{1} \operatorname{LM}(f) r_{1}$ and $\operatorname{LM}\left(\ell_{23} u_{j} p_{j} v_{j} r_{23}\right)<\ell_{1} \operatorname{LM}(f) r_{1}$ for all $i$ and $\left.j\right)$. But this follows from Equation (3.4) and from the fact that the reductions of the expressions $c_{3} \ell_{1}^{\prime} f r_{1}^{\prime}-c_{1} \ell_{3}^{\prime} h r_{3}^{\prime}$ and $c_{3} \ell_{2}^{\prime \prime} g r_{2}^{\prime \prime}-c_{2} \ell_{3}^{\prime \prime} h r_{3}^{\prime \prime}$ in Equations (3.5) and (3.6) are valid, so that $\mathrm{LM}\left(u_{i} p_{i} v_{i}\right)<\mathrm{LM}\left(\ell_{1}^{\prime} f r_{1}^{\prime}\right)$ and $\operatorname{LM}\left(u_{j} p_{j} v_{j}\right)<\operatorname{LM}\left(\ell_{2}^{\prime \prime} g r_{2}^{\prime \prime}\right)$ for all $i$ and $j$.

Remark 3.4.2 The three polynomials $f, g$ and $h$ in the above proposition do not necessarily have to be distinct (indeed, $f=g=h$ is allowed) - the only restriction is that the S-polynomial S-pol $\left(\ell_{1}, f, \ell_{2}, g\right)$ has to be different from the S-polynomials S-pol $\left(\ell_{1}^{\prime}, f, \ell_{3}^{\prime}, h\right)$ and $\mathrm{S}-\mathrm{pol}\left(\ell_{2}^{\prime \prime}, g, \ell_{3}^{\prime \prime}, h\right)$; for example, if $f=h$, then we cannot have $\ell_{1}^{\prime}=\ell_{3}^{\prime}$.

### 3.4.2 Homogeneous Gröbner Bases

Because it is computationally more expensive to do noncommutative polynomial arithmetic than it is to do commutative polynomial arithmetic, gains in efficiency due to working with homogeneous bases are even more significant in the noncommutative case.

For this reason, some systems for computing noncommutative Gröbner Bases will only work with homogeneous input bases, although (as in the commutative case) it is still sometimes possible to use these systems on non-homogeneous input bases by using the concepts of homogenisation, dehomogenisation and extendible monomial orderings.

Definition 3.4.3 Let $p=p_{0}+\cdots+p_{m}$ be a polynomial over the polynomial ring $R\left\langle x_{1}, \ldots, x_{n}\right\rangle$, where each $p_{i}$ is the sum of the degree $i$ terms in $p$ (we assume that $p_{m} \neq 0$ ). The left homogenisation of $p$ with respect to a new (homogenising) variable $y$ is the polynomial

$$
h_{\ell}(p):=y^{m} p_{0}+y^{m-1} p_{1}+\cdots+y p_{m-1}+p_{m} ;
$$

and the right homogenisation of $p$ with respect to a new (homogenising) variable $y$ is the polynomial

$$
h_{r}(p):=p_{0} y^{m}+p_{1} y^{m-1}+\cdots+p_{m-1} y+p_{m} .
$$

Homogenised polynomials belong to polynomial rings determined by where $y$ is placed in the lexicographical ordering of the variables.

Definition 3.4.4 The dehomogenisation of a polynomial $p$ is the polynomial $d(p)$ given by substituting $y=1$ in $p$, where $y$ is the homogenising variable.

Definition 3.4.5 A monomial ordering $O$ is extendible if, given any polynomial $p=$ $t_{1}+\cdots+t_{\alpha}$ ordered with respect to $O$ (where $t_{1}>\cdots>t_{\alpha}$ ), the homogenisation of $p$ preserves the order on the terms $\left(t_{i}^{\prime}>t_{i+1}^{\prime}\right.$ for all $1 \leqslant i \leqslant \alpha-1$, where the homogenisation process maps the term $t_{i} \in p$ to the term $t_{i}^{\prime}$ ).

In the noncommutative case, an extendible monomial ordering must specify how to homogenise a polynomial (by multiplying with the homogenising variable on the left or on the right) as well as stating where the new variable $y$ appears in the ordering of the variables. Here are the conventions for those monomial orderings defined in Section 1.2.2 that are extendible, assuming that we start with a polynomial ring $R\left\langle x_{1}, \ldots, x_{n}\right\rangle$.

| Monomial Ordering | Type of Homogenisation | Position of the new variable $y$ <br> in the ordering of the variables |
| :---: | :---: | :---: |
| InvLex | Right | $y<x_{i}$ for all $x_{i}$ |
| DegLex | Left | $y<x_{i}$ for all $x_{i}$ |
| DegInvLex | Left | $y>x_{i}$ for all $x_{i}$ |
| DegRevLex | Right | $y>x_{i}$ for all $x_{i}$ |

Noncommutativity also provides the possibility of the new variable $y$ becoming 'trapped' in the middle of some monomial forming part of a polynomial computed during the course of Mora's algorithm. For example, working with DegRevLex, consider the homogenised polynomial $h_{r}\left(x_{1}^{2}+x_{1}\right)=x_{1}^{2}+x_{1} y$ and the S-polynomial

$$
\mathrm{S}-\operatorname{pol}\left(x_{1}, x_{1}^{2}+x_{1} y, 1, x_{1}^{2}+x_{1} y\right)=x_{1}\left(x_{1}^{2}+x_{1} y\right)-\left(x_{1}^{2}+x_{1} y\right) x_{1}=x_{1}^{2} y-x_{1} y x_{1}
$$

Because $y$ appears in the middle of the monomial $x_{1} y x_{1}$, the S-polynomial does not immediately reduce to zero as it does in the non-homogenised version of the S-polynomial,

$$
\mathrm{S}-\mathrm{pol}\left(x_{1}, x_{1}^{2}+x_{1}, 1, x_{1}^{2}+x_{1}\right)=x_{1}\left(x_{1}^{2}+x_{1}\right)-\left(x_{1}^{2}+x_{1}\right) x_{1}=0 .
$$

We must therefore make certain that $y$ only appears on one side of any given monomial by introducing the set of polynomials $H=\left\{h_{1}, h_{2}, \ldots, h_{n}\right\}=\left\{y x_{1}-x_{1} y, y x_{2}-\right.$ $\left.x_{2} y, \ldots, y x_{n}-x_{n} y\right\}$ into our initial homogenised basis, ensuring that $y$ commutes with all the other variables in the polynomial ring. This way, the first S-polynomial will reduce to zero as follows:

$$
x_{1}^{2} y-x_{1} y x_{1} \rightarrow_{h_{1}} x_{1}^{2} y-x_{1}^{2} y=0
$$

Which side $y$ will appear on will be determined by whether $\operatorname{LM}\left(y x_{i}-x_{i} y\right)=y x_{i}$ or $\mathrm{LM}\left(y x_{i}-x_{i} y\right)=x_{i} y$ in our chosen monomial ordering (pushing $y$ to the right or to the left respectively). This side must match the method of homogenisation, which explains why Lex is not an extendible monomial ordering - for Lex to be extendible, we must homogenise on the right and have $y<x_{i}$ for all $x_{i}$, but then because $\operatorname{LM}\left(y x_{i}-x_{i} y\right)=x_{i} y$ with respect to Lex, the variable $y$ will always in practice appear on the left.

Definition 3.4.6 Let $F=\left\{f_{1}, \ldots, f_{m}\right\}$ be a non-homogeneous set of polynomials over the polynomial ring $R\left\langle x_{1}, \ldots, x_{n}\right\rangle$. To compute a Gröbner Basis for $F$ using a program that only accepts sets of homogeneous polynomials as input, we use the following procedure (which will only work in conjunction with an extendible monomial ordering).
(a) Construct a homogeneous set of polynomials $F^{\prime}=\left\{h_{\ell}\left(f_{1}\right), \ldots, h_{\ell}\left(f_{m}\right)\right\}$ or $F^{\prime}=$ $\left\{h_{r}\left(f_{1}\right), \ldots, h_{r}\left(f_{m}\right)\right\}$ (dependent on the monomial ordering used).
(b) Compute a Gröbner Basis $G^{\prime}$ for the set $F^{\prime} \cup H$, where $H=\left\{y x_{1}-x_{1} y, y x_{2}-\right.$ $\left.x_{2} y, \ldots, y x_{n}-x_{n} y\right\}$.
(c) Dehomogenise each polynomial $g^{\prime} \in G^{\prime}$ to obtain a Gröbner Basis $G$ for $F$, noting
that no polynomial originating from $H$ will appear in $G\left(d\left(h_{i}\right)=0\right.$ for all $\left.h_{i} \in H\right)$.

### 3.4.3 Selection Strategies

As in the commutative case, the order in which S-polynomials are processed during Mora's algorithm has an important effect on the efficiency of the algorithm. Let us now generalise the selection strategies defined in Section 2.5.3 for use in the noncommutative setting, basing our decisions on the overlap words of S-polynomials.

Definition 3.4.7 The overlap word of an S-polynomial S-pol $\left(\ell_{1}, f, \ell_{2}, g\right)=\mathrm{LC}(g) \ell_{1} f r_{1}-$ $\mathrm{LC}(f) \ell_{2} g r_{2}$ is the monomial $\ell_{1} \mathrm{LM}(f) r_{1}\left(=\ell_{2} \mathrm{LM}(g) r_{2}\right)$.

Definition 3.4.8 In the noncommutative normal strategy, we choose an S-polynomial to process if its overlap word is minimal in the chosen monomial ordering amongst all such overlap words.

Definition 3.4.9 In the noncommutative sugar strategy, we choose an S-polynomial to process if its sugar (a value associated to the S-polynomial) is minimal amongst all such values (we use the normal strategy in the event of a tie).

The sugar of an S-polynomial is computed by using the following rules on the sugars of polynomials we encounter during the computation of a Gröbner Basis for the set of polynomials $F=\left\{f_{1}, \ldots, f_{m}\right\}$.
(1) The sugar $\operatorname{Sug}_{f_{i}}$ of a polynomial $f_{i} \in F$ is the total degree of the polynomial $f_{i}$ (which is the degree of the term of maximal degree in $f_{i}$ ).
(2) If $p$ is a polynomial and if $t_{1}$ and $t_{2}$ are terms, then $\operatorname{Sug}_{t_{1} p t_{2}}=\operatorname{deg}\left(t_{1}\right)+\operatorname{Sug}_{p}+\operatorname{deg}\left(t_{2}\right)$.
(3) If $p=p_{1}+p_{2}$, then $\operatorname{Sug}_{p}=\max \left(\operatorname{Sug}_{p_{1}}, \operatorname{Sug}_{p_{2}}\right)$.

It follows that the sugar of the S-polynomial S-pol $\left(\ell_{1}, g, \ell_{2}, h\right)=\mathrm{LC}(h) \ell_{1} g r_{1}-\mathrm{LC}(g) \ell_{2} h r_{2}$ is given by the formula

$$
\operatorname{Sug}_{s-\operatorname{pol}\left(\ell_{1}, g, \ell_{2}, h\right)}=\max \left(\operatorname{deg}\left(\ell_{1}\right)+\operatorname{Sug}_{g}+\operatorname{deg}\left(r_{1}\right), \operatorname{deg}\left(\ell_{2}\right)+\operatorname{Sug}_{h}+\operatorname{deg}\left(r_{2}\right)\right)
$$

### 3.4.4 Logged Gröbner Bases

Definition 3.4.10 Let $G=\left\{g_{1}, \ldots, g_{p}\right\}$ be a noncommutative Gröbner Basis computed from an initial basis $F=\left\{f_{1}, \ldots, f_{m}\right\}$. We say that $G$ is a Logged Gröbner Basis if, for each $g_{i} \in G$, we have an explicit expression of the form

$$
g_{i}=\sum_{\alpha=1}^{\beta} \ell_{\alpha} f_{k_{\alpha}} r_{\alpha}
$$

where the $\ell_{\alpha}$ and the $r_{\alpha}$ are terms and $f_{k_{\alpha}} \in F$ for all $1 \leqslant \alpha \leqslant \beta$.
Proposition 3.4.11 Let $F=\left\{f_{1}, \ldots, f_{m}\right\}$ be a finite basis over a noncommutative polynomial ring. If we can compute a Gröbner Basis for $F$, then it is always possible to compute a Logged Gröbner Basis for F.

Proof: We refer to the proof of Proposition 2.5.11, substituting

$$
\operatorname{S-pol}\left(\ell_{1}, f_{i}, \ell_{2}, f_{j}\right)-\sum_{\alpha=1}^{\beta} \ell_{\alpha} g_{k_{\alpha}} r_{\alpha}
$$

for $f_{m+1}$ (the $\ell_{\alpha}$ and the $r_{\alpha}$ are terms).

### 3.5 A Worked Example

To demonstrate Mora's algorithm in action, let us now calculate a Gröbner Basis for the ideal $J$ generated by the set of polynomials $F:=\left\{f_{1}, f_{2}, f_{3}\right\}=\{x y-z, y z+2 x+z, y z+x\}$ over the polynomial ring $\mathbb{Q}\langle x, y, z\rangle$. We shall use the DegLex monomial ordering (with $x>y>z$ ); use the normal selection strategy; calculate a Logged Gröbner Basis; and use Buchberger's criteria.

### 3.5.1 Initialisation

The first part of Mora's algorithm requires us to find all the overlaps between the lead monomials of the three polynomials in the initial basis $G:=\left\{g_{1}, g_{2}, g_{3}\right\}=\{x y-z, y z+$ $2 x+z, y z+x\}$. There are three overlaps in total, summarised by the following table.

|  | Overlap 1 | Overlap 2 | Overlap 3 |
| :---: | :---: | :---: | :---: |
| Overlap Word | $y z$ | $x y z$ | $x y z$ |
| Polynomial 1 | $y z+2 x+z$ | $x y-z$ | $x y-z$ |
| Polynomial 2 | $y z+x$ | $y z+2 x+z$ | $y z+x$ |
| $\ell_{1}$ | 1 | 1 | 1 |
| $r_{1}$ | 1 | $z$ | $z$ |
| $\ell_{2}$ | 1 | $x$ | $x$ |
| $r_{2}$ | 1 | 1 | 1 |
| Degree of Overlap Word | 2 | 3 | 3 |

Because we are using the normal selection strategy, it is clear that Overlap 1 will appear in the list $A$ first, but we are free to choose the order in which the other two overlaps appear (because their overlap words are identical). To eliminate this choice, we will use the following tie-breaking strategy to order any two S-polynomials whose overlap words are identical.

Definition 3.5.1 Let $s_{1}=\operatorname{S-pol}\left(\ell_{1}, g_{a}, \ell_{2}, g_{b}\right)$ and $s_{2}=\operatorname{S-pol}\left(\ell_{3}, g_{c}, \ell_{4}, g_{d}\right)$ be two Spolynomials with identical overlap words, where $g_{a}, g_{b}, g_{c}, g_{d} \in G=\left\{g_{1}, \ldots, g_{\alpha}\right\}$. Assuming (without loss of generality) that $a<b$ and $c<d$, the tie-breaking strategy places $s_{1}$ before $s_{2}$ in $A$ if $a<c$ or if $a=c$ and $b \leqslant d$; and later in $A$ otherwise.

Applying the tie-breaking strategy for Overlaps 2 and 3, it follows that Overlap $2=$ S-pol( $\left.1, g_{1}, x, g_{2}\right)$ will appear in $A$ before Overlap $3=\operatorname{S-pol}\left(1, g_{1}, x, g_{3}\right)$.

Before we start the main part of the algorithm, let us note that for the Logged Gröbner Basis, we begin the algorithm with trivial expressions for each of the three polynomials in the initial basis $G$ in terms of the polynomials of the input basis $F$ : $g_{1}=x y-z=f_{1}$; $g_{2}=y z+2 x+z=f_{2} ;$ and $g_{3}=y z+x=f_{3}$.

### 3.5.2 Calculating and Reducing S-polynomials

The first S-polynomial to analyse corresponds to Overlap 1 and is the polynomial

$$
1(y z+2 x+z) 1-1(y z+x) 1=2 x+z-x=x+z .
$$

This polynomial is irreducible with respect to $G$, and so we add it to $G$ to obtain a new basis $G=\{x y-z, y z+2 x+z, y z+x, x+z\}=\left\{g_{1}, g_{2}, g_{3}, g_{4}\right\}$. Looking for overlaps between
the lead monomial of $x+z$ and the lead monomials of the four elements of $G$, we see that there is one such overlap (with $g_{1}$ ) whose overlap word has degree 2, so this overlap is added to the beginning of the list $A$ to obtain $A=\{\mathrm{S}-\mathrm{pol}(1, x y-z, 1, x+z)$, S-pol( $1, x y-$ $z, x, y z+2 x+z), \mathrm{S}-\operatorname{pol}(1, x y-z, x, y z+x)\}$. As far as the Logged Gröbner Basis goes, $g_{4}=x+z=1(y z+2 x+z) 1-1(y z+x) 1=f_{2}-f_{3}$.

The next entry in $A$ produces the polynomial

$$
1(x y-z) 1-1(x+z) y=-z y-z .
$$

As before, this polynomial is irreducible with respect to $G$, so we add it to $G$ as the fifth element. There are also four overlaps between the lead monomial of $-z y-z$ and the lead monomials of the five polynomials in $G$ :

|  | Overlap 1 | Overlap 2 | Overlap 3 | Overlap 4 |
| :---: | :---: | :---: | :---: | :---: |
| Overlap Word | $z y z$ | $z y z$ | $y z y$ | $y z y$ |
| Polynomial 1 | $y z+2 x+z$ | $y z+x$ | $y z+2 x+z$ | $y z+x$ |
| Polynomial 2 | $-z y-z$ | $-z y-z$ | $-z y-z$ | $-z y-z$ |
| $\ell_{1}$ | $z$ | $z$ | 1 | 1 |
| $r_{1}$ | 1 | 1 | $y$ | $y$ |
| $\ell_{2}$ | 1 | 1 | $y$ | $y$ |
| $r_{2}$ | $z$ | $z$ | 1 | 1 |
| Degree of Overlap Word | 3 | 3 | 3 | 3 |

Inserting these overlaps into the list $A$, we obtain

$$
\left.\begin{array}{rl}
A=\{\quad & \mathrm{S}-\operatorname{pol}(z, y z+2 x+z, 1,-z y-z), \mathrm{S}-\operatorname{pol}(z, y z+x, 1,-z y-z), \\
& \mathrm{S}-\operatorname{pol}(1, y z+2 x+z, y,-z y-z), \mathrm{S}-\operatorname{pol}(1, y z+x, y,-z y-z), \\
& \mathrm{S}-\operatorname{pol}(1, x y-z, x, y z+2 x+z), \mathrm{S}-\operatorname{pol}(1, x y-z, x, y z+x)
\end{array}\right\} .
$$

The logged representation of the fifth basis element again comes straight from the Spolynomial (as no reduction was performed), and is as follows: $g_{5}=-z y-z=1$ ( $x y-$ $z) 1-1(x+z) y=1\left(f_{1}\right) 1-1\left(f_{2}-f_{3}\right) y=f_{1}-f_{2} y+f_{3} y$.

The next entry in $A$ yields the polynomial

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

This time, the fourth polynomial in our basis reduces the S-polynomial in question, giving a reduction $-2 z x \rightarrow_{g_{4}} 2 z^{2}$. When we add this polynomial to $G$ and add all five new overlaps to $A$, we are left with a six element basis $G=\{x y-z, y z+2 x+z, y z+x, x+$ $\left.z,-z y-z, 2 z^{2}\right\}$ and a list

$$
\begin{aligned}
A=\{ & \mathrm{S}-\operatorname{pol}\left(1,2 z^{2}, z, 2 z^{2}\right), \operatorname{S-pol}\left(z, 2 z^{2}, 1,2 z^{2}\right), \\
& \mathrm{S}-\operatorname{pol}\left(z,-z y-z, 1,2 z^{2}\right), \operatorname{S-pol}(z, y z+x, 1,-z y-z), \\
& \operatorname{S-pol}\left(1, y z+2 x+z, y, 2 z^{2}\right), \operatorname{S-pol}\left(1, y z+x, y, 2 z^{2}\right), \\
& \mathrm{S}-\operatorname{pol}(1, y z+2 x+z, y,-z y-z), \operatorname{S-pol}(1, y z+x, y,-z y-z), \\
& \operatorname{S-pol}(1, x y-z, x, y z+2 x+z), \operatorname{S-pol}(1, x y-z, x, y z+x) \quad\} .
\end{aligned}
$$

We obtain the logged version of the sixth basis element by working backwards through our calculations:

$$
\begin{aligned}
g_{6} & =2 z^{2} \\
& =-2 z x+2 z(x+z) \\
& =(-z(y z+2 x+z) 1-1(-z y-z) z)+2 z(x+z) \\
& =\left(-z\left(f_{2}\right)-\left(f_{1}-f_{2} y+f_{3} y\right) z\right)+2 z\left(f_{2}-f_{3}\right) \\
& =-f_{1} z+z f_{2}+f_{2} y z-2 z f_{3}-f_{3} y z .
\end{aligned}
$$

### 3.5.3 Applying Buchberger's Second Criterion

The next three entries in $A$ all yield S-polynomials that are either zero or reduce to zero (for example, the first entry corresponds to the polynomial $2\left(2 z^{2}\right) z-2 z\left(2 z^{2}\right) 1=$ $4 z^{3}-4 z^{3}=0$ ). The fourth entry in $A, \operatorname{S-pol}(z, y z+x, 1,-z y-z)$, then enables us (for the first time) to apply Buchberger's second criterion, allowing us to move on to look at the fifth entry of $A$. Before we do this however, let us explain why we can apply Buchberger's second criterion in this particular case.

Recall (from Proposition 3.4.1) that in order to apply Buchberger's second criterion for the S-polynomial S-pol $(z, y z+x, 1,-z y-z)$, we need to find a polynomial $g_{i} \in G$ such that $\mathrm{LM}\left(g_{i}\right)$ divides the overlap word of our S-polynomial, and any S-polynomials corresponding to overlaps (as positioned in the overlap word) between $\mathrm{LM}\left(g_{i}\right)$ and either $\mathrm{LM}(y z+x)$ or $\mathrm{LM}(-z y-z)$ reduce to zero using $G$ (which will be the case if those
particular S-polynomials have been processed earlier in the algorithm).
Consider the polynomial $g_{2}=y z+2 x+z$. The lead monomial of this polynomial divides the overlap word $z y z$ of our S-polynomial, which we illustrate as follows.


As positioned in the overlap word, we note that $\mathrm{LM}\left(g_{2}\right)$ overlaps with both $\mathrm{LM}\left(g_{3}\right)$ and $\mathrm{LM}\left(g_{5}\right)$, with the overlaps corresponding to the S -polynomials S -pol $\left(1, g_{2}, 1, g_{3}\right)=$ S-pol( $1, y z+2 x+z, 1, y z+x)$ and $\operatorname{S-pol}\left(z, g_{2}, 1, g_{5}\right)=\operatorname{S-pol}(z, y z+2 x+z, 1,-z y-z)$ respectively. But these S-polynomials have been processed earlier in the algorithm (they were the first and third S-polynomials to be processed); we can therefore apply Buchberger's second criterion in this instance.

There are now six S-polynomials left in $A$, all of whom either reduce to zero or are ignored due to Buchberger's second criterion. Here is a summary of what happens during the remainder of the algorithm.

| S-polynomial | Action |
| :--- | :--- |
| S-pol(1, $\left.y z+2 x+z, y, 2 z^{2}\right)$ | Reduces to zero using the division algorithm |
| S-pol(1, $\left.y z+x, y, 2 z^{2}\right)$ | Ignored due to Buchberger's second criterion |
| (using $y z+2 x+z)$ |  |
| S-pol(1, $y z+2 x+z, y,-z y-z)$ | Reduces to zero using the division algorithm |
| S-pol(1,yz+x,y,-zy-z) | Ignored due to Buchberger's second criterion <br> (using $y z+2 x+z)$ |
| S-pol(1, xy-z,x,yz+2x+z) | Ignored due to Buchberger's second criterion <br> (using $x+z)$ |
| S-pol(1, $x y-z, x, y z+x)$ | Ignored due to Buchberger's second criterion <br> (using $y z+2 x+z)$ |

As the list $A$ is now empty, the algorithm terminates with the following (Logged) Gröbner Basis.

| Input Basis $F$ | Gröbner Basis $G$ |
| :--- | :--- |
| $f_{1}=x y-z$ | $g_{1}=x y-z=f_{1}$ |
| $f_{2}=y z+2 x+z$ | $g_{2}=y z+2 x+z=f_{2}$ |
| $f_{3}=y z+x$ | $g_{3}=y z+x=f_{3}$ |
|  | $g_{4}=x+z=f_{2}-f_{3}$ |
|  | $g_{5}=-z y-z=f_{1}-f_{2} y+f_{3} y$ |
|  | $g_{6}=2 z^{2}=-f_{1} z+z f_{2}+f_{2} y z-2 z f_{3}-f_{3} y z$ |

### 3.5.4 Reduction

Now that we have constructed a Gröbner Basis for our ideal $J$, let us go on to find the unique reduced Gröbner Basis for $J$ by applying Algorithm 6 to $G$.

In the first half of the algorithm, we must multiply each polynomial by the inverse of its lead coefficient and remove from the basis each polynomial whose lead monomial is a multiple of the lead monomial of some other polynomial in the basis. For the Gröbner Basis in question, we multiply $g_{5}$ by -1 and $g_{6}$ by $\frac{1}{2}$; and we remove $g_{1}$ and $g_{2}$ from the basis (because $\mathrm{LM}\left(g_{1}\right)=\mathrm{LM}\left(g_{4}\right) \times y$ and $\left.\operatorname{LM}\left(g_{2}\right)=\mathrm{LM}\left(g_{3}\right)\right)$. This leaves us with the following (minimal) Gröbner Basis.

| Input Basis $F$ | Gröbner Basis $G$ |
| :--- | :--- |
| $f_{1}=x y-z$ | $g_{3}=y z+x=f_{3}$ |
| $f_{2}=y z+2 x+z$ | $g_{4}=x+z=f_{2}-f_{3}$ |
| $f_{3}=y z+x$ | $g_{5}=z y+z=-f_{1}+f_{2} y-f_{3} y$ |
|  | $g_{6}=z^{2}=-\frac{1}{2} f_{1} z+\frac{1}{2} z f_{2}+\frac{1}{2} f_{2} y z-z f_{3}-\frac{1}{2} f_{3} y z$ |

In the second half of the algorithm, we reduce each $g_{i} \in G$ with respect to $\left(G \backslash\left\{g_{i}\right\}\right) \cup G^{\prime}$, placing the remainder in the (initially empty) set $G^{\prime}$ and removing $g_{i}$ from $G$. For the Gröbner Basis in question, we summarise what happens in the following table, noting that the only reduction that takes place is the reduction $y z+x \rightarrow_{g_{4}} y z+x-(x+z)=y z-z$.

| $G$ | $G^{\prime}$ | $g_{i}$ | $g_{i}^{\prime}$ |
| :--- | :--- | :--- | :--- |
| $\left\{y z+x, x+z, z y+z, z^{2}\right\}$ | $\emptyset$ | $y z+x$ | $y z-z$ |
| $\left\{x+z, z y+z, z^{2}\right\}$ | $\{y z-z\}$ | $x+z$ | $x+z$ |
| $\left\{z y+z, z^{2}\right\}$ | $\{y z-z, x+z\}$ | $z y+z$ | $z y+z$ |
| $\left\{z^{2}\right\}$ | $\{y z-z, x+z, z y+z\}$ | $z^{2}$ | $z^{2}$ |
| $\emptyset$ | $\left\{y z-z, x+z, z y+z, z^{2}\right\}$ |  |  |

We can now give the unique reduced (Logged) Gröbner Basis for $J$.

| Input Basis $F$ | Unique Reduced Gröbner Basis $G^{\prime}$ |
| :--- | :--- |
| $f_{1}=x y-z$ | $y z-z=-f_{2}+2 f_{3}$ |
| $f_{2}=y z+2 x+z$ | $x+z=f_{2}-f_{3}$ |
| $f_{3}=y z+x$ | $z y+z=-f_{1}+f_{2} y-f_{3} y$ |
|  | $z^{2}=-\frac{1}{2} f_{1} z+\frac{1}{2} z f_{2}+\frac{1}{2} f_{2} y z-z f_{3}-\frac{1}{2} f_{3} y z$ |

## Chapter 4

## Commutative Involutive Bases

Given a Gröbner Basis $G$ for an ideal $J$ over a polynomial ring $\mathcal{R}$, we know that the remainder of any polynomial $p \in \mathcal{R}$ with respect to $G$ is unique. But although this remainder is unique, there may be many ways of obtaining the remainder, as it is possible that several polynomials in $G$ divide our polynomial $p$, giving several reduction paths for p.

Example 4.0.2 Consider the DegLex Gröbner Basis $G:=\left\{g_{1}, g_{2}, g_{3}\right\}=\left\{x^{2}-2 x y+\right.$ 3, $\left.2 x y+y^{2}+5, \frac{5}{4} y^{3}-\frac{5}{2} x+\frac{37}{4} y\right\}$ over the polynomial ring $\mathcal{R}:=\mathbb{Q}[x, y]$ from Example 2.3.2, and consider the polynomial $p:=x^{2} y+y^{3}+8 y \in \mathcal{R}$. The remainder of $p$ with respect to $G$ is 0 (so that $p$ is a member of the ideal $J$ generated by $G$ ), but there are two ways of obtaining this remainder, as shown in the following diagram.


An Involutive Basis is a Gröbner Basis $G$ for $J$ such that there is only one possible reduction path for any polynomial $p \in \mathcal{R}$. In order to find such a basis, we must restrict
which reductions or divisions may take place by requiring, for each potential reduction of a polynomial $p$ by a polynomial $g_{i} \in G$ (so that $\mathrm{LM}(p)=\mathrm{LM}\left(g_{i}\right) \times u$ for some monomial $u$ ), some extra conditions on the variables in $u$ to be satisfied, namely that all variables in $u$ have to be in a set of multiplicative variables for $g_{i}$, a set that is determined by a particular choice of an involutive division.

### 4.1 Involutive Divisions

In Definition 1.2.9, we saw that a commutative monomial $u_{1}$ is divisible by another monomial $u_{2}$ if there exists a third monomial $u_{3}$ such that $u_{1}=u_{2} u_{3}$; we also introduced the notation $u_{2} \mid u_{1}$ to denote that $u_{2}$ is a divisor of $u_{1}$, a divisor we shall now refer to as a conventional divisor of $u_{1}$. For a particular choice of an involutive division $I$, we say that $u_{2}$ is an involutive divisor of $u_{1}$, written $\left.u_{2}\right|_{I} u_{1}$, if, given a partitioning (by $I$ ) of the variables in the polynomial ring into sets of multiplicative and nonmultiplicative variables for $u_{2}$, all variables in $u_{3}$ are in the set of multiplicative variables for $u_{2}$.

Example 4.1.1 Let $u_{1}:=x y^{2} z^{2} ; u_{1}^{\prime}:=x^{2} y z$ and $u_{2}:=x z$ be three monomials over the polynomial ring $\mathcal{R}:=\mathbb{Q}[x, y, z]$, and let an involutive division $I$ partition the variables in $\mathcal{R}$ into the following two sets of variables for the monomial $u_{2}$ : multiplicative $=\{y, z\}$; nonmultiplicative $=\{x\}$. It is true that $u_{2}$ conventionally divides both monomials $u_{1}$ and $u_{1}^{\prime}$, but $u_{2}$ only involutively divides monomial $u_{1}$ as, defining $u_{3}:=y^{2} z$ and $u_{3}^{\prime}:=x y$ (so that $u_{1}=u_{2} u_{3}$ and $u_{1}^{\prime}=u_{2} u_{3}^{\prime}$ ), we observe that all variables in $u_{3}$ are in the set of multiplicative variables for $u_{2}$, but the variables in $u_{3}^{\prime}$ (in particular the variable $x$ ) are not all in the set of multiplicative variables for $u_{2}$.

More formally, an involutive division $I$ works with a set of monomials $U$ over a polynomial ring $R\left[x_{1}, \ldots, x_{n}\right]$ and assigns a set of multiplicative variables $\mathcal{M}_{I}(u, U) \subseteq\left\{x_{1}, \ldots, x_{n}\right\}$ to each element $u \in U$. It follows that, with respect to $U$, a monomial $w$ is divisible by a monomial $u \in U$ if $w=u v$ for some monomial $v$ and all the variables that appear in $v$ also appear in the set $\mathcal{M}_{I}(u, U)$.

Definition 4.1.2 Let $M$ denote the set of all monomials in the polynomial ring $\mathcal{R}=$ $R\left[x_{1}, \ldots, x_{n}\right]$, and let $U \subset M$. The involutive cone $\mathcal{C}_{I}(u, U)$ of any monomial $u \in U$ with respect to some involutive division $I$ is defined as follows.

$$
\mathcal{C}_{I}(u, U)=\left\{u v \text { such that } v \in M \text { and }\left.u\right|_{I} u v\right\} .
$$

Remark 4.1.3 We may think of an involutive cone of a particular monomial $u$ as containing all monomials that are involutively divisible by $u$.

Up to now, we have not mentioned any restriction on how we may assign multiplicative variables to a particular set of monomials. Let us now specify the rules that ensure that a particular scheme of assigning multiplicative variables may be referred to as an involutive division.

Definition 4.1.4 Let $M$ denote the set of all monomials in the polynomial ring $\mathcal{R}=$ $R\left[x_{1}, \ldots, x_{n}\right]$. An involutive division $I$ on $M$ is defined if, given any finite set of monomials $U \subset M$, we can assign a set of multiplicative variables $\mathcal{M}_{I}(u, U) \subseteq\left\{x_{1}, \ldots, x_{n}\right\}$ to any monomial $u \in U$ such that the following two conditions are satisfied.
(a) If there exist two monomials $u_{1}, u_{2} \in U$ such that $\mathcal{C}_{I}\left(u_{1}, U\right) \cap \mathcal{C}_{I}\left(u_{2}, U\right) \neq \emptyset$, then either $\mathcal{C}_{I}\left(u_{1}, U\right) \subset \mathcal{C}_{I}\left(u_{2}, U\right)$ or $\mathcal{C}_{I}\left(u_{2}, U\right) \subset \mathcal{C}_{I}\left(u_{1}, U\right)$.
(b) If $V \subset U$, then $\mathcal{M}_{I}(v, U) \subseteq \mathcal{M}_{I}(v, V)$ for all $v \in V$.

Remark 4.1.5 Informally, condition (a) above ensures that a monomial can only appear in two involutive cones $\mathcal{C}_{I}\left(u_{1}, U\right)$ and $\mathcal{C}_{I}\left(u_{2}, U\right)$ if $u_{1}$ is an involutive divisor of $u_{2}$ or vice-versa; while condition (b) ensures that the multiplicative variables of a polynomial $v \in V \subset U$ with respect to $U$ all appear in the set of multiplicative variables of $v$ with respect to $V$.

Definition 4.1.6 Given an involutive division $I$, the involutive span $\mathcal{C}_{I}(U)$ of a set of monomials $U$ with respect to $I$ is given by the expression

$$
\mathcal{C}_{I}(U)=\bigcup_{u \in U} \mathcal{C}_{I}(u, U) .
$$

Remark 4.1.7 The (conventional) span of a set of monomials $U$ is given by the expression

$$
\mathcal{C}(U)=\bigcup_{u \in U} \mathcal{C}(u, U),
$$

where $\mathcal{C}(u, U)=\{u v \mid v$ is a monomial $\}$ is the (conventional) cone of a monomial $u \in U$.
Definition 4.1.8 If an involutive division $I$ determines the multiplicative variables for a monomial $u \in U$ independent of the set $U$, then $I$ is a global division. Otherwise, $I$ is a local division.

Remark 4.1.9 The multiplicative variables for a set of polynomials $P$ (whose terms are ordered by a monomial ordering $O$ ) are determined by the multiplicative variables for the set of leading monomials $\mathrm{LM}(P)$.

### 4.1.1 Involutive Reduction

In Algorithm 7, we specify how to involutively divide a polynomial $p$ with respect to a set of polynomials $P$.

```
Algorithm 7 The Commutative Involutive Division Algorithm
Input: A nonzero polynomial \(p\) and a set of nonzero polynomials \(P=\left\{p_{1}, \ldots, p_{m}\right\}\) over a
    polynomial ring \(R\left[x_{1}, \ldots x_{n}\right]\); an admissible monomial ordering O ; an involutive division \(I\).
Output: \(\operatorname{Rem}_{I}(p, P):=r\), the involutive remainder of \(p\) with respect to \(P\).
\[
r=0 ;
\]
\[
\text { while }(p \neq 0) \text { do }
\]
\[
u=\operatorname{LM}(p) ; c=\mathrm{LC}(p) ; j=1 ; \text { found }=\text { false } ;
\]
\[
\text { while }(j \leqslant m) \text { and (found }==\text { false) do }
\]
\[
\text { if }\left(\left.\operatorname{LM}\left(p_{j}\right)\right|_{I} u\right) \text { then }
\]
\[
\text { found }=\text { true } ; u^{\prime}=u / \mathrm{LM}\left(p_{j}\right) ; p=p-\left(c \mathrm{LC}\left(p_{j}\right)^{-1}\right) p_{j} u^{\prime} ;
\]
else
\[
j=j+1
\]
end if
        end while
        if (found \(==\) false) then
            \(r=r+\operatorname{LT}(p) ; p=p-\operatorname{LT}(p) ;\)
        end if
    end while
    return \(r\);
```

Remark 4.1.10 The only difference between Algorithms 1 and 7 is that the line "if $\left(\operatorname{LM}\left(p_{j}\right) \mid u\right)$ then" in Algorithm 1 has been changed to the line "if $\left(\left.\operatorname{LM}\left(p_{j}\right)\right|_{I} u\right)$ then" in Algorithm 7.

Definition 4.1.11 If the polynomial $r$ is obtained by involutively dividing (with respect to some involutive division $I$ ) the polynomial $p$ by one of (a) a polynomial $q$; (b) a sequence
of polynomials $q_{1}, q_{2}, \ldots, q_{\alpha}$; or (c) a set of polynomials $Q$, we will use the notation $p \vec{I}_{q} r ; p \xrightarrow[I]{*} r$ and $p \xrightarrow[I]{ } r$ respectively (matching the notation introduced in Definition 1.2.16).

### 4.1.2 Thomas, Pommaret and Janet divisions

Let us now consider three different involutive divisions, all named after their creators in the theory of Partial Differential Equations (see [52], [47] and [35]).

Definition 4.1.12 (Thomas) Let $U=\left\{u_{1}, \ldots, u_{m}\right\}$ be a set of monomials over a polynomial ring $R\left[x_{1}, \ldots, x_{n}\right]$, where the monomial $u_{j} \in U$ (for $1 \leqslant j \leqslant m$ ) has corresponding multidegree ( $e_{j}^{1}, e_{j}^{2}, \ldots, e_{j}^{n}$ ). The Thomas involutive division $\mathcal{T}$ assigns multiplicative variables to elements of $U$ as follows: the variable $x_{i}$ is multiplicative for monomial $u_{j}$ (written $x_{i} \in \mathcal{M}_{\mathcal{T}}\left(u_{j}, U\right)$ ) if $e_{j}^{i}=\max _{k} e_{k}^{i}$ for all $1 \leqslant k \leqslant m$.

Definition 4.1.13 (Pommaret) Let $u$ be a monomial over a polynomial ring $R\left[x_{1}, \ldots, x_{n}\right]$ with multidegree ( $e^{1}, e^{2}, \ldots, e^{n}$ ). The Pommaret involutive division $\mathcal{P}$ assigns multiplicative variables to $u$ as follows: if $1 \leqslant i \leqslant n$ is the smallest integer such that $e^{i}>0$, then all variables $x_{1}, x_{2}, \ldots, x_{i}$ are multiplicative for $u$ (we have $x_{j} \in \mathcal{M}_{\mathcal{P}}(u)$ for all $1 \leqslant j \leqslant i$ ).

Definition 4.1.14 (Janet) Let $U=\left\{u_{1}, \ldots, u_{m}\right\}$ be a set of monomials over a polynomial ring $R\left[x_{1}, \ldots, x_{n}\right]$, where the monomial $u_{j} \in U$ (for $1 \leqslant j \leqslant m$ ) has corresponding multidegree $\left(e_{j}^{1}, e_{j}^{2}, \ldots, e_{j}^{n}\right)$. The Janet involutive division $\mathcal{J}$ assigns multiplicative variables to elements of $U$ as follows: the variable $x_{n}$ is multiplicative for monomial $u_{j}$ (written $x_{n} \in \mathcal{M}_{\mathcal{J}}\left(u_{j}, U\right)$ ) if $e_{j}^{n}=\max _{k} e_{k}^{n}$ for all $1 \leqslant k \leqslant m$; the variable $x_{i}$ (for $1 \leqslant i<n$ ) is multiplicative for monomial $u_{j}$ (written $x_{i} \in \mathcal{M}_{\mathcal{J}}\left(u_{j}, U\right)$ ) if $e_{j}^{i}=\max _{k} e_{k}^{i}$ for all monomials $u_{k} \in U$ such that $e_{j}^{l}=e_{k}^{l}$ for all $i<l \leqslant n$.

Remark 4.1.15 Thomas and Janet are local involutive divisions; Pommaret is a global involutive division.

Example 4.1.16 Let $U:=\left\{x^{5} y^{2} z, x^{4} y z^{2}, x^{2} y^{2} z, x y z^{3}, x z^{3}, y^{2} z, z\right\}$ be a set of monomials over the polynomial ring $\mathbb{Q}[x, y, z]$, with $x>y>z$. Here are the multiplicative variables for $U$ according to the three involutive divisions defined above.

| Monomial | Thomas | Pommaret | Janet |
| :---: | :---: | :---: | :---: |
| $x^{5} y^{2} z$ | $\{x, y\}$ | $\{x\}$ | $\{x, y\}$ |
| $x^{4} y z^{2}$ | $\emptyset$ | $\{x\}$ | $\{x, y\}$ |
| $x^{2} y^{2} z$ | $\{y\}$ | $\{x\}$ | $\{y\}$ |
| $x y z^{3}$ | $\{z\}$ | $\{x\}$ | $\{x, y, z\}$ |
| $x z^{3}$ | $\{z\}$ | $\{x\}$ | $\{x, z\}$ |
| $y^{2} z$ | $\{y\}$ | $\{x, y\}$ | $\{y\}$ |
| $z$ | $\emptyset$ | $\{x, y, z\}$ | $\{x\}$ |

Proposition 4.1.17 All three involutive divisions defined above satisfy the conditions of Definition 4.1.4.

Proof: Throughout, let $M$ denote the set of all monomials in the polynomial ring $\mathcal{R}=R\left[x_{1}, \ldots, x_{n}\right]$; let $U=\left\{u_{1}, \ldots, u_{m}\right\} \subset M$ be a set of monomials with corresponding multidegrees $\left(e_{k}^{1}, e_{k}^{2}, \ldots, e_{k}^{n}\right)$ (where $1 \leqslant k \leqslant m$ ); let $u_{i}, u_{j} \in U$ (where $1 \leqslant i, j \leqslant m, i \neq j$ ); and let $m_{1}, m_{2} \in M$ be two monomials with corresponding multidegrees $\left(f_{1}^{1}, f_{1}^{2}, \ldots, f_{1}^{n}\right)$ and $\left(f_{2}^{1}, f_{2}^{2}, \ldots, f_{2}^{n}\right)$. For condition (a), we need to show that if there exists a monomial $m \in M$ such that $m_{1} u_{i}=m=m_{2} u_{j}$ and all variables in $m_{1}$ and $m_{2}$ are multiplicative for $u_{i}$ and $u_{j}$ respectively, then either $u_{i}$ is an involutive divisor of $u_{j}$ or vice-versa. For condition (b), we need to show that all variables that are multiplicative for $u_{i} \in U$ are still multiplicative for $u_{i} \in V \subseteq U$.

Thomas. (a) It is sufficient to prove that $u_{i}=u_{j}$. Assume to the contrary that $u_{i} \neq u_{j}$, so that there is some $1 \leqslant k \leqslant n$ such that $e_{i}^{k} \neq e_{j}^{k}$. Without loss of generality, assume that $e_{i}^{k}<e_{j}^{k}$. Because $e_{i}^{k}+f_{1}^{k}=e_{j}^{k}+f_{2}^{k}$, it follows that $f_{1}^{k}>0$ so that the variable $x_{k}$ must be multiplicative for the monomial $u_{i}$. But this contradicts the fact that $x_{k}$ cannot be multiplicative for $u_{i}$ in the Thomas involutive division because $e_{j}^{k}>e_{i}^{k}$. We therefore have $u_{i}=u_{j}$.
(b) By definition, if $x_{j} \in \mathcal{M}_{\mathcal{T}}\left(u_{i}, U\right)$, then $e_{i}^{j}=\max _{k} e_{k}^{j}$ for all $u_{k} \in U$. Given a set $V \subseteq U$, it is clear that $e_{i}^{j}=\max _{k} e_{k}^{j}$ for all $u_{k} \in V$, so that $x_{j} \in \mathcal{M}_{\mathcal{T}}\left(u_{i}, V\right)$ as required.

Pommaret. (a) Let $\alpha$ and $\beta(1 \leqslant \alpha, \beta \leqslant n)$ be the smallest integers such that $e_{i}^{\alpha}>0$ and $e_{j}^{\beta}>0$ respectively, and assume (without loss of generality) that $\alpha \geqslant \beta$. By definition, we must have $f_{1}^{k}=f_{2}^{k}=0$ for all $\alpha<k \leqslant n$ because the $x_{k}$ are all nonmultiplicative for $u_{i}$ and $u_{j}$. It follows that $e_{i}^{k}=e_{j}^{k}$ for all $\alpha<k \leqslant n$. If $\alpha=\beta$, then it is clear that $u_{i}$ is an involutive divisor of $u_{j}$ if $e_{i}^{\alpha}<e_{j}^{\alpha}$, and $u_{j}$ is an involutive divisor of $u_{i}$ if $e_{i}^{\alpha}>e_{j}^{\alpha}$. If
$\alpha>\beta$, then $f_{2}^{\alpha}=0$ as variable $x_{\alpha}$ is nonmultiplicative for $u_{j}$, so it follows that $e_{i}^{\alpha} \leqslant e_{j}^{\alpha}$ and hence $u_{i}$ is an involutive divisor of $u_{j}$.
(b) Follows immediately because Pommaret is a global involutive division.

Janet. (a) We prove that $u_{i}=u_{j}$. Assume to the contrary that $u_{i} \neq u_{j}$, so there exists a maximal $1 \leqslant k \leqslant n$ such that $e_{i}^{k} \neq e_{j}^{k}$. Without loss of generality, assume that $e_{i}^{k}<e_{j}^{k}$. If $k=n$, we get an immediate contradiction because Janet is equivalent to Thomas for the final variable. If $k=n-1$, then because $e_{i}^{n-1}+f_{1}^{n-1}=e_{j}^{n-1}+f_{2}^{n-1}$, it follows that $f_{1}^{n-1}>0$ so that the variable $x_{n-1}$ must be multiplicative for the monomial $u_{i}$. But this contradicts the fact that $x_{n-1}$ cannot be multiplicative for $u_{i}$ in the Janet involutive division because $e_{j}^{n-1}>e_{i}^{n-1}$ and $e_{j}^{n}=e_{i}^{n}$. By induction on $k$, we can show that $e_{i}^{k}=e_{j}^{k}$ for all $1 \leqslant k \leqslant n$, so that $u_{i}=u_{j}$ as required.
(b) By definition, if $x_{j} \in \mathcal{M}_{\mathcal{J}}\left(u_{i}, U\right)$, then $e_{i}^{j}=\max _{k} e_{k}^{j}$ for all monomials $u_{k} \in U$ such that $e_{i}^{l}=e_{k}^{l}$ for all $i<l \leqslant n$. Given a set $V \subseteq U$, it is clear that $e_{i}^{j}=\max _{k} e_{k}^{j}$ for all $u_{k} \in V$ such that $e_{i}^{l}=e_{k}^{l}$ for all $i<l \leqslant n$, so that $x_{j} \in \mathcal{M}_{\mathcal{J}}\left(u_{i}, V\right)$ as required.

The conditions of Definition 4.1.4 ensure that any polynomial is involutively divisible by at most one polynomial in any Involutive Basis. One advantage of this important combinatorial property is that the Hilbert function of an ideal $J$ is easily computable with respect to an Involutive Basis (see [4]).

Example 4.1.18 Returning to Example 4.0.2, consider again the DegLex Gröbner Basis $G:=\left\{x^{2}-2 x y+3,2 x y+y^{2}+5, \frac{5}{4} y^{3}-\frac{5}{2} x+\frac{37}{4} y\right\}$ over the polynomial ring $\mathbb{Q}[x, y]$. A Pommaret Involutive Basis for $G$ is the set $P:=G \cup\left\{g_{4}:=-5 x y^{2}-5 x+6 y\right\}$, with the variable $x$ being multiplicative for all polynomials in $P$, and the variable $y$ being multiplicative for just $g_{3}$. We can illustrate the difference between the overlapping cones of $G$ and the non-overlapping involutive cones of $P$ by the following diagram.


The diagram also demonstrates that the polynomial $p:=x^{2} y+y^{3}+8 y$ is initially conventionally divisible by two members of the Gröbner Basis $G$ (as seen in Equation (4.1)), but is only involutively divisible by one member of the Involutive Basis $P$, starting the following unique involutive reduction path for $p$.


### 4.2 Prolongations and Autoreduction

Whereas Buchberger's algorithm constructs a Gröbner Basis by using S-polynomials, the involutive algorithm will construct an Involutive Basis by using prolongations and autoreduction.

Definition 4.2.1 Given a set of polynomials $P$, a prolongation of a polynomial $p \in P$ is a product $p x_{i}$, where $x_{i} \notin \mathcal{M}_{I}(\operatorname{LM}(p), \mathrm{LM}(P))$ with respect to some involutive division $I$.

Definition 4.2.2 A set of polynomials $P$ is said to be autoreduced if no polynomial $p \in P$ exists such that $p$ contains a term which is involutively divisible (with respect to $P$ ) by some polynomial $p^{\prime} \in P \backslash\{p\}$. Algorithm 8 provides a way of performing autoreduction,
and introduces the following notation: Let $\operatorname{Rem}_{I}(A, B, C)$ denote the involutive remainder of the polynomial $A$ with respect to the set of polynomials $B$, where reductions are only to be performed by elements of the set $C \subseteq B$.

Remark 4.2.3 The involutive cones associated to an autoreduced set of polynomials are always disjoint, meaning that a given monomial can only appear in at most one of the involutive cones.

```
        \(p_{i}^{\prime}=\operatorname{Rem}_{I}\left(p_{i}, P, P \backslash\left\{p_{i}\right\}\right)\);
        \(P=P \backslash\left\{p_{i}\right\} ;\)
        if \(\left(p_{i}^{\prime} \neq 0\right)\) then
            \(P=P \cup\left\{p_{i}^{\prime}\right\} ;\)
        end if
    end while
    \(Q=P\);
    return \(Q\);
```

Algorithm 8 The Commutative Autoreduction Algorithm
Input: A set of polynomials $P=\left\{p_{1}, p_{2}, \ldots, p_{\alpha}\right\}$; an involutive division $I$.
Output: An autoreduced set of polynomials $Q=\left\{q_{1}, q_{2}, \ldots, q_{\beta}\right\}$.
while $\left(\exists p_{i} \in P\right.$ such that $\left.\operatorname{Rem}_{I}\left(p_{i}, P, P \backslash\left\{p_{i}\right\}\right) \neq p_{i}\right)$ do

Proposition 4.2.4 Let $P$ be a set of polynomials over a polynomial ring $\mathcal{R}=R\left[x_{1}, \ldots, x_{n}\right]$, and let $f$ and $g$ be two polynomials also in $\mathcal{R}$. If $P$ is autoreduced with respect to an involutive division $I$, then $\operatorname{Rem}_{I}(f, P)+\operatorname{Rem}_{I}(g, P)=\operatorname{Rem}_{I}(f+g, P)$.

Proof: Let $f^{\prime}:=\operatorname{Rem}_{I}(f, P) ; g^{\prime}:=\operatorname{Rem}_{I}(g, P)$ and $h^{\prime}:=\operatorname{Rem}_{I}(h, P)$, where $h:=f+g$. Then, by the respective involutive reductions, we have expressions

$$
\begin{aligned}
& f^{\prime}=f-\sum_{a=1}^{A} p_{\alpha_{a}} t_{a} ; \\
& g^{\prime}=g-\sum_{b=1}^{B} p_{\beta_{b}} t_{b}
\end{aligned}
$$

and

$$
h^{\prime}=h-\sum_{c=1}^{C} p_{\gamma_{c}} t_{c},
$$

where $p_{\alpha_{a}}, p_{\beta_{b}}, p_{\gamma_{c}} \in P$ and $t_{a}, t_{b}, t_{c}$ are terms which are multiplicative (over $P$ ) for each $p_{\alpha_{a}}, p_{\beta_{b}}$ and $p_{\gamma_{c}}$ respectively.

Consider the polynomial $h^{\prime}-f^{\prime}-g^{\prime}$. By the above expressions, we can deduce ${ }^{1}$ that

$$
h^{\prime}-f^{\prime}-g^{\prime}=\sum_{a=1}^{A} p_{\alpha_{a}} t_{a}+\sum_{b=1}^{B} p_{\beta_{b}} t_{b}-\sum_{c=1}^{C} p_{\gamma_{c}} t_{c}=: \sum_{d=1}^{D} p_{\delta_{d}} t_{d} .
$$

Claim: $\operatorname{Rem}_{I}\left(h^{\prime}-f^{\prime}-g^{\prime}, P\right)=0$.
Proof of Claim: Let $t$ denote the leading term of the polynomial $\sum_{d=1}^{D} p_{\delta_{d}} t_{d}$. Then $\mathrm{LM}(t)=\mathrm{LM}\left(p_{\delta_{k}} t_{k}\right)$ for some $1 \leqslant k \leqslant D$ since, if not, there exists a monomial $\mathrm{LM}\left(p_{\delta_{k^{\prime}}} t_{k^{\prime}}\right)=$ $\operatorname{LM}\left(p_{\delta_{k^{\prime \prime}}} t_{k^{\prime \prime}}\right)=: u$ for some $1 \leqslant k^{\prime}, k^{\prime \prime} \leqslant D$ (with $\left.p_{\delta_{k^{\prime}}} \neq p_{\delta_{k^{\prime \prime}}}\right)$ such that $u$ is involutively divisible by the two polynomials $p_{\delta_{k^{\prime}}}$ and $p_{\delta_{k^{\prime \prime}}}$, contradicting Definition 4.1 .4 (recall that our set $P$ is autoreduced, so that the involutive cones of $P$ are disjoint). It follows that we can use $p_{\delta_{k}}$ to eliminate $t$ by involutively reducing $h^{\prime}-f^{\prime}-g^{\prime}$ as shown below.

$$
\begin{equation*}
\sum_{d=1}^{D} p_{\delta_{d}} t_{d} \underset{I p^{\delta_{k}}}{ } \sum_{d=1}^{k-1} p_{\delta_{d}} t_{d}+\sum_{d=k+1}^{D} p_{\delta_{d}} t_{d} \tag{4.2}
\end{equation*}
$$

By induction, we can apply a chain of involutive reductions to the right hand side of Equation (4.2) to obtain a zero remainder, so that $\operatorname{Rem}_{I}\left(h^{\prime}-f^{\prime}-g^{\prime}, P\right)=0$.

To complete the proof, we note that since $f^{\prime}, g^{\prime}$ and $h^{\prime}$ are all involutively irreducible, we must have $\operatorname{Rem}_{I}\left(h^{\prime}-f^{\prime}-g^{\prime}, P\right)=h^{\prime}-f^{\prime}-g^{\prime}$. It therefore follows that $h^{\prime}-f^{\prime}-g^{\prime}=0$, or $h^{\prime}=f^{\prime}+g^{\prime}$ as required.

Remark 4.2.5 The above proof is based on the proofs of Theorem 5.4 and Corollary 5.5 in [25].

Let us now give a definition of a Locally Involutive Basis in terms of prolongations. Later on in this chapter, we will discover that the Involutive Basis algorithm only constructs Locally Involutive Bases, and it is the extra properties of each involutive division used with the algorithm that ensures that any computed Locally Involutive Basis is an Involutive Basis.

Definition 4.2.6 Given an involutive division $I$ and an admissible monomial ordering

[^7]$O$, an autoreduced set of polynomials $P$ is a Locally Involutive Basis with respect to $I$ and $O$ if any prolongation of any polynomial $p_{i} \in P$ involutively reduces to zero using $P$.

Definition 4.2.7 Given an involutive division $I$ and an admissible monomial ordering $O$, an autoreduced set of polynomials $P$ is an Involutive Basis with respect to $I$ and $O$ if any multiple $p_{i} t$ of any polynomial $p_{i} \in P$ by any term $t$ involutively reduces to zero using $P$.

### 4.3 Continuity and Constructivity

In the theory of commutative Gröbner Bases, Buchberger's algorithm returns a Gröbner Basis as long as an admissible monomial ordering is used. In the theory of commutative Involutive Bases however, not only must an admissible monomial ordering be used, but the involutive division chosen must be continuous and constructive.

Definition 4.3.1 (Continuity) Let $I$ be an involutive division, and let $U$ be an arbitrary set of monomials over a polynomial ring $R\left[x_{1}, \ldots, x_{n}\right]$. We say that $I$ is continuous if, given any sequence of monomials $\left\{u_{1}, u_{2}, \ldots, u_{m}\right\}$ from $U$ such that for all $i<m$, we have $\left.u_{i+1}\right|_{I} u_{i} x_{j_{i}}$ for some variable $x_{j_{i}}$ that is nonmultiplicative for monomial $m_{i}$ (or $x_{j_{i}} \notin \mathcal{M}_{I}\left(u_{i}, U\right)$ ), no two monomials in the sequence are the same ( $u_{r} \neq u_{s}$ for all $r \neq s$, where $1 \leqslant r, s \leqslant m)$.

Proposition 4.3.2 The Thomas, Pommaret and Janet involutive divisions are all continuous.

Proof: Throughout, let the sequence of monomials $\left\{u_{1}, \ldots, u_{i}, \ldots, u_{m}\right\}$ have corresponding multidegrees $\left(e_{i}^{1}, e_{i}^{2}, \ldots, e_{i}^{n}\right)$ (where $1 \leqslant i \leqslant m$ ).

Thomas. If the variable $x_{j_{i}}$ is nonmultiplicative for monomial $u_{i}$, then, by definition, $e_{i}^{j_{i}} \neq \max _{t} e_{t}^{j_{i}}$ for all $u_{t} \in U$. Variable $x_{j_{i}}$ cannot therefore be multiplicative for monomial $u_{i+1}$ if $e_{i+1}^{j_{i}} \leqslant e_{i}^{j_{i}}$, so we must have $e_{i+1}^{j_{i}}=e_{i}^{j_{i}}+1$ in order to have $\left.u_{i+1}\right|_{\mathcal{T}} u_{i} x_{j_{i}}$. Further, for all $1 \leqslant k \leqslant n$ such that $k \neq j_{i}$, we must have $e_{i+1}^{k}=e_{i}^{k}$ as, if $e_{i+1}^{k}<e_{i}^{k}$, then $x_{k}$ cannot be multiplicative for monomial $u_{i+1}$ (which contradicts $\left.u_{i+1}\right|_{\tau} u_{i} x_{j_{i}}$ ). Thus $u_{i+1}=u_{i} x_{j_{i}}$, and so it is clear that the monomials in the sequence $\left\{u_{1}, u_{2}, \ldots, u_{m}\right\}$ are all different.

Pommaret. Let $\alpha_{i}\left(1 \leqslant \alpha_{i} \leqslant n\right)$ be the smallest integer such that $e_{i}^{\alpha_{i}}>0$ (where $1 \leqslant i \leqslant m$ ), so that $e_{i}^{k}=0$ for all $k<\alpha_{i}$. Because $\left.u_{i+1}\right|_{\mathcal{P}} u_{i} x_{j_{i}}$ for all $1 \leqslant i<m$,
and because (by definition) $j_{i}>\alpha_{i}$, it follows that we must have $e_{i+1}^{k}=0$ for all $k<\alpha_{i}$. Therefore $\alpha_{i+1} \geqslant \alpha_{i}$ for all $1 \leqslant i<n$. If $\alpha_{i+1}=\alpha_{i}$, we note that $e_{i+1}^{\alpha_{i}} \leqslant e_{i}^{\alpha_{i}}$ because variable $x_{\alpha_{i}}$ is multiplicative for monomial $u_{i+1}$. If then we have $e_{i+1}^{\alpha_{i}}=e_{i}^{\alpha_{i}}$, then because the variable $x_{j_{i}}$ is also nonmultiplicative for monomial $u_{i+1}$, we must have $e_{i+1}^{j_{i}}=e_{i}^{j_{i}}+1$.

It is now clear that the monomials in the sequence $\left\{u_{1}, u_{2}, \ldots, u_{m}\right\}$ are all different because (a) the values in the sequence $\alpha=\left\{\alpha_{1}, \alpha_{2}, \ldots, \alpha_{m}\right\}$ monotonically increase; (b) for consecutive values $\alpha_{s}, \alpha_{s+1}, \ldots, \alpha_{s+\sigma}$ in $\alpha$ that are identical $(1 \leqslant s<m, s+\sigma \leqslant m)$, the values in the corresponding sequence $E=\left\{e_{s}^{\alpha_{s}}, e_{s+1}^{\alpha_{s}}, \ldots, e_{s+\sigma}^{\alpha_{s}}\right\}$ monotonically decrease; (c) for consecutive values $e_{t}^{\alpha_{s}}, e_{t+1}^{\alpha_{s}}, \ldots, e_{t+\tau}^{\alpha_{s}}$ in $E$ that are identical $(s \leqslant t<s+\sigma, t+\tau \leqslant s+\sigma$ ), the degrees of the monomials $u_{t}, u_{t+1}, \ldots, u_{t+\tau}$ strictly increase.

Janet. Consider the monomials $u_{1}, u_{2}$ and the variable $x_{j_{1}}$ that is nonmultiplicative for $u_{1}$. We will first prove (by induction) that $e_{2}^{i}=e_{1}^{i}$ for all $j_{1}<i \leqslant n$. For the case $i=n$, we must have $e_{2}^{n}=e_{1}^{n}$ otherwise (by definition) variable $x_{n}$ is nonmultiplicative for monomial $u_{2}$ (we have $e_{2}^{n}<e_{1}^{n}$ ), contradicting that fact that $\left.u_{2}\right|_{\mathcal{J}} u_{1} x_{j_{1}}$. For the inductive step, assume that $e_{2}^{i}=e_{1}^{i}$ for all $k \leqslant i \leqslant n$, and let us look at the case $i=k-1$. If $e_{2}^{k-1}<e_{1}^{k-1}$, then (by definition) variable $x_{k-1}$ is nonmultiplicative for monomial $u_{2}$, again contradicting the fact that $\left.u_{2}\right|_{\mathcal{J}} u_{1} x_{j_{1}}$. It follows that we must have $e_{2}^{k-1}=e_{1}^{k-1}$.

Let us now prove that $e_{2}^{j_{1}}=e_{1}^{j_{1}}+1$. We can rule out the case $e_{2}^{j_{1}}<e_{1}^{j_{1}}$ immediately because this implies that the variable $x_{j_{1}}$ is nonmultiplicative for monomial $u_{2}$ (by definition), contradicting the fact that $\left.u_{2}\right|_{\mathcal{J}} u_{1} x_{j_{1}}$. The case $e_{2}^{j_{1}}=e_{1}^{j_{1}}$ can also be ruled out because we cannot have $e_{2}^{i}=e_{1}^{i}$ for all $j_{1} \leqslant i \leqslant n$ and variable $x_{j_{1}}$ being simultaneously nonmultiplicative for monomial $u_{1}$ and multiplicative for monomial $u_{2}$. Thus $e_{2}^{j_{1}}=e_{1}^{j_{1}}+1$. It follows that $u_{1}<u_{2}$ in the InvLex monomial ordering (see Section 1.2.1) and so, by induction, $u_{1}<u_{2}<\cdots<u_{m}$ in the InvLex monomial ordering. The monomials in the sequence $\left\{u_{1}, u_{2}, \ldots, u_{m}\right\}$ are therefore all different.

Proposition 4.3.3 If an involutive division I is continuous, and a given set of polynomials $P$ is a Locally Involutive Basis with respect to $I$ and some admissible monomial ordering $O$, then $P$ is an Involutive Basis with respect to $I$ and $O$.

Proof: Let $I$ be a continuous involutive division; let $O$ be an admissible monomial ordering; and let $P$ be a Locally Involutive Basis with respect to $I$ and $O$. Given any polynomial $p \in P$ and any term $t$, in order to show that $P$ is an Involutive Basis with respect to $I$ and $O$, we must show that $p t \vec{I}_{P} 0$.

If $\left.p\right|_{I} p t$ we are done, as we can use $p$ to involutively reduce $p t$ to obtain a zero remainder. Otherwise, $\exists y_{1} \notin \mathcal{M}_{I}(\operatorname{LM}(p), \operatorname{LM}(P))$ such that $t$ contains $y_{1}$. By Local Involutivity, the prolongation $p y_{1}$ involutively reduces to zero using $P$. Assuming that the first step of this involutive reduction involves the polynomial $p_{1} \in P$, we can write

$$
\begin{equation*}
p y_{1}=p_{1} t_{1}+\sum_{a=1}^{A} p_{\alpha_{a}} t_{\alpha_{a}}, \tag{4.3}
\end{equation*}
$$

where $p_{\alpha_{a}} \in P$ and $t_{1}, t_{\alpha_{a}}$ are terms which are multiplicative (over $P$ ) for $p_{1}$ and each $p_{\alpha_{a}}$ respectively. Multiplying both sides of Equation (4.3) by $\frac{t}{y_{1}}$, we obtain the equation

$$
\begin{equation*}
p t=p_{1} t_{1} \frac{t}{y_{1}}+\sum_{a=1}^{A} p_{\alpha_{a}} t_{\alpha_{a}} \frac{t}{y_{1}} . \tag{4.4}
\end{equation*}
$$

If $\left.p_{1}\right|_{I} p t$, it is clear that we can use $p_{1}$ to involutively reduce the polynomial $p t$ to obtain the polynomial $\sum_{a=1}^{A} p_{\alpha_{a}} t_{\alpha_{a}} \frac{t}{y_{1}}$. By Proposition 4.2.4, we can then continue to involutively reduce $p t$ by repeating this proof on each polynomial $p_{\alpha_{a}} t_{\alpha_{a}} \frac{t}{y_{1}}$ individually (where $1 \leqslant a \leqslant A$ ), noting that this process will terminate because of the admissibility of $O$ (we have $\mathrm{LM}\left(p_{\alpha_{a}} t_{\alpha_{a}} \frac{t}{y_{1}}\right)<\operatorname{LM}(p t)$ for all $1 \leqslant a \leqslant A$ ).

Otherwise, if $p_{1}$ does not involutively divide $p t$, there exists a variable $y_{2} \in \frac{t}{y_{1}}$ such that $y_{2} \notin \mathcal{M}_{I}\left(\operatorname{LM}\left(p_{1}\right), \mathrm{LM}(P)\right)$. By Local Involutivity, the prolongation $p_{1} y_{2}$ involutively reduces to zero using $P$. Assuming that the first step of this involutive reduction involves the polynomial $p_{2} \in P$, we can write

$$
\begin{equation*}
p_{1} y_{2}=p_{2} t_{2}+\sum_{b=1}^{B} p_{\beta_{b}} t_{\beta_{b}}, \tag{4.5}
\end{equation*}
$$

where $p_{\beta_{b}} \in P$ and $t_{2}, t_{\beta_{b}}$ are terms which are multiplicative (over $P$ ) for $p_{2}$ and each $p_{\beta_{b}}$ respectively. Multiplying both sides of Equation (4.5) by $\frac{t_{1} t}{y_{1} y_{2}}$, we obtain the equation

$$
\begin{equation*}
p_{1} t_{1} \frac{t}{y_{1}}=p_{2} t_{2} \frac{t_{1} t}{y_{1} y_{2}}+\sum_{b=1}^{B} p_{\beta_{b}} t_{\beta_{b}} \frac{t_{1} t}{y_{1} y_{2}} . \tag{4.6}
\end{equation*}
$$

Substituting for $p_{1} t_{1} \frac{t}{y_{1}}$ from Equation (4.6) into Equation (4.4), we obtain the equation

$$
\begin{equation*}
p t=p_{2} t_{2} \frac{t_{1} t}{y_{1} y_{2}}+\sum_{a=1}^{A} p_{\alpha_{a}} t_{\alpha_{a}} \frac{t}{y_{1}}+\sum_{b=1}^{B} p_{\beta_{b}} t_{\beta_{b}} \frac{t_{1} t}{y_{1} y_{2}} . \tag{4.7}
\end{equation*}
$$

If $\left.p_{2}\right|_{I} p t$, it is clear that we can use $p_{2}$ to involutively reduce the polynomial $p t$ to obtain the polynomial $\sum_{a=1}^{A} p_{\alpha_{a}} t_{\alpha_{a}} \frac{t}{y_{1}}+\sum_{b=1}^{B} p_{\beta_{b}} t_{\beta_{b}} \frac{t_{1} t}{y_{1} y_{2}}$. As before, we can then use Proposition 4.2.4 to continue the involutive reduction of $p t$ by repeating this proof on each summand individually.

Otherwise, if $p_{2}$ does not involutively divide $p t$, we continue by induction, obtaining a sequence $p, p_{1}, p_{2}, p_{3}, \ldots$ of elements in $P$. By construction, each element in the sequence divides $p t$. By continuity, each element in the sequence is different. Because $P$ is finite and because $p t$ has a finite number of distinct divisors, the sequence must be finite, terminating with an involutive divisor $p^{\prime} \in P$ of $p t$, which then allows us to finish the proof through use of Proposition 4.2.4 and the admissibility of $O$.

Remark 4.3.4 The above proof is a slightly clarified version of the proof of Theorem 6.5 in [25].

Definition 4.3.5 (Constructivity) Let $I$ be an involutive division, and let $U$ be an arbitrary set of monomials over a polynomial ring $R\left[x_{1}, \ldots, x_{n}\right]$. We say that $I$ is constructive if, given any monomial $u \in U$ and any nonmultiplicative variable $x_{i} \notin \mathcal{M}_{I}(u, U)$ satisfying the following two conditions, no monomial $w \in \mathcal{C}_{I}(U)$ exists such that $u x_{i} \in$ $\mathcal{C}_{I}(w, U \cup\{w\})$.
(a) $u x_{i} \notin \mathcal{C}_{I}(U)$.
(b) If there exists a monomial $v \in U$ and a nonmultiplicative variable $x_{j} \notin \mathcal{M}_{I}(v, U)$ such that $v x_{j} \mid u x_{i}$ but $v x_{j} \neq u x_{i}$, then $v x_{j} \in \mathcal{C}_{I}(U)$.

Remark 4.3.6 Constructivity allows us to consider only polynomials whose lead monomials lie outside the current involutive span as potential new Involutive Basis elements.

Proposition 4.3.7 The Thomas, Pommaret and Janet involutive divisions are all constructive.

Proof: Throughout, let the monomials $u, v$ and $w$ that appear in Definition 4.3.5 have corresponding multidegrees $\left(e_{u}^{1}, e_{u}^{2}, \ldots, e_{u}^{n}\right),\left(e_{v}^{1}, e_{v}^{2}, \ldots, e_{v}^{n}\right)$ and $\left(e_{w}^{1}, e_{w}^{2}, \ldots, e_{w}^{n}\right)$; and let the monomials $w_{1}, w_{2}, w_{3}$ and $\mu$ that appear in this proof have corresponding multidegrees $\left(e_{w_{1}}^{1}, e_{w_{1}}^{2}, \ldots, e_{w_{1}}^{n}\right),\left(e_{w_{2}}^{1}, e_{w_{2}}^{2}, \ldots, e_{w_{2}}^{n}\right),\left(e_{w_{3}}^{1}, e_{w_{3}}^{2}, \ldots, e_{w_{3}}^{n}\right)$ and $\left(e_{\mu}^{1}, e_{\mu}^{2}, \ldots, e_{\mu}^{n}\right)$.

To prove that a particular involutive division $I$ is constructive, we will assume that a monomial $w \in \mathcal{C}_{I}(U)$ exists such that $u x_{i} \in \mathcal{C}_{I}(w, U \cup\{w\})$. Then $w=\mu w_{1}$ for some
monomial $\mu \in U$ and some monomial $w_{1}$ that is multiplicative for $\mu$ over the set $U$ $\left(e_{w_{1}}^{k}>0 \Rightarrow x_{k} \in \mathcal{M}_{I}(\mu, U)\right.$ for all $\left.1 \leqslant k \leqslant n\right)$; and $u x_{i}=w w_{2}$ for some monomial $w_{2}$ that is multiplicative for $w$ over the set $U \cup\{w\}\left(e_{w_{2}}^{k}>0 \Rightarrow x_{k} \in \mathcal{M}_{I}(w, U \cup\{w\})\right.$ for all $1 \leqslant k \leqslant n$ ). It follows that $u x_{i}=\mu w_{1} w_{2}$. If we can show that all variables appearing in $w_{2}$ are multiplicative for $\mu$ over the set $U\left(e_{w_{2}}^{k}>0 \Rightarrow x_{k} \in \mathcal{M}_{I}(\mu, U)\right.$ for all $\left.1 \leqslant k \leqslant n\right)$, then $\mu$ is an involutive divisor of $u x_{i}$, contradicting the assumption $u x_{i} \notin \mathcal{C}_{I}(U)$.

Thomas. Let $x_{k}$ be an arbitrary variable $(1 \leqslant k \leqslant n)$ such that $e_{w_{2}}^{k}>0$. If $e_{w_{1}}^{k}>0$, then it is clear that $x_{k}$ is multiplicative for $\mu$. Otherwise $e_{w_{1}}^{k}=0$ so that $e_{w}^{k}=e_{\mu}^{k}$. By definition, this implies that $x_{k} \in \mathcal{M}_{\mathcal{T}}(\mu, U)$ as $x_{k} \in \mathcal{M}_{\mathcal{T}}(w, U \cup\{w\})$. Thus $x_{k} \in \mathcal{M}_{\mathcal{T}}(\mu, U)$.

Pommaret. Let $\alpha$ and $\beta(1 \leqslant \alpha, \beta \leqslant n)$ be the smallest integers such that $e_{\mu}^{\alpha}>0$ and $e_{w}^{\beta}>0$ respectively. By definition, $\beta \leqslant \alpha$ (because $w=\mu w_{1}$ ), so for an arbitrary $1 \leqslant k \leqslant n$, it follows that $e_{w_{2}}^{k}>0 \Rightarrow k \leqslant \beta \leqslant \alpha \Rightarrow x_{k} \in \mathcal{M}_{\mathcal{P}}(\mu, U)$ as required.

Janet. Here we proceed by searching for a monomial $\nu \in U$ such that $u x_{i} \in \mathcal{C}_{\mathcal{J}}(\nu, U)$, contradicting the assumption $u x_{i} \notin \mathcal{C}_{\mathcal{J}}(U)$. Let $\alpha$ and $\beta(1 \leqslant \alpha, \beta \leqslant n)$ be the largest integers such that $e_{w_{1}}^{\alpha}>0$ and $e_{w_{2}}^{\beta}>0$ respectively (such integers will exist because if $\operatorname{deg}\left(w_{1}\right)=0$ or $\operatorname{deg}\left(w_{2}\right)=0$, we obtain an immediate contradiction $\left.u x_{i} \in \mathcal{C}_{\mathcal{J}}(U)\right)$. We claim that $i>\max \{\alpha, \beta\}$.

- If $i<\beta$, then $e_{w}^{\beta}<e_{u}^{\beta}$ which contradicts $x_{\beta} \in \mathcal{M}_{\mathcal{J}}(w, U \cup\{w\})$ as $e_{w}^{\gamma}=e_{u}^{\gamma}$ for all $\gamma>\beta$. Thus $i \geqslant \beta$.
- If $i<\alpha$, then as $\beta \leqslant i$ we must have $e_{\mu}^{\gamma}=e_{u}^{\gamma}$ for all $\alpha<\gamma \leqslant n$. Therefore $e_{\mu}^{\alpha}<e_{u}^{\alpha} \Rightarrow x_{\alpha} \notin \mathcal{M}_{\mathcal{J}}(\mu, U)$, a contradiction; it follows that $i \geqslant \alpha$.
- If $i=\alpha$, then either $\beta<\alpha$ or $\beta=\alpha$. If $\beta=\alpha$, then as $e_{w_{1}}^{i}>0 ; e_{w_{2}}^{i}>0$ and $e_{u}^{i}+1=e_{\mu}^{i}+e_{w_{1}}^{i}+e_{w_{2}}^{i}$, we have $e_{u}^{i}>e_{\mu}^{i} \Rightarrow x_{\alpha} \notin \mathcal{M}_{\mathcal{J}}(\mu, U)$, a contradiction. If $\beta<\alpha$, then $e_{u}^{i}+1=e_{\mu}^{i}+e_{w_{1}}^{i}$. If $e_{w_{1}}^{i} \geqslant 2$, we get the same contradiction as before $\left(x_{\alpha} \notin \mathcal{M}_{\mathcal{J}}(\mu, U)\right)$. Otherwise $e_{w_{1}}^{i}=1$ so that $e_{u}^{\gamma}=e_{\mu}^{\gamma}$ for all $\alpha \leqslant \gamma \leqslant n$. If $w=\mu x_{i}$, then as $e_{w}^{\beta}<e_{u}^{\beta}$ we have $x_{\beta} \notin \mathcal{M}_{\mathcal{J}}(w, U \cup\{w\})$, a contradiction. Else let $\delta$ (where $1 \leqslant \delta<\alpha$ ) be the second greatest integer such that $e_{w_{1}}^{\delta}>0$. Then, as $e_{\mu}^{\delta}<e_{u}^{\delta}$ and $e_{\mu}^{\gamma}=e_{u}^{\gamma}$ for all $\delta<\gamma \leqslant n$, we have $x_{\delta} \notin \mathcal{M}_{\mathcal{J}}(\mu, U)$, another contradiction. It follows that $i>\max \{\alpha, \beta\}$, so that $e_{u}^{\gamma}=e_{\mu}^{\gamma}$ for all $i<\gamma \leqslant n$ and $e_{u}^{i}+1=e_{\mu}^{i}$.

If $u x_{i} \notin \mathcal{C}_{\mathcal{J}}(U)$, then there must exist a variable $x_{k}$ (where $1 \leqslant k<i$ ) such that $e_{w_{2}}^{k}>0$ and $x_{k} \notin \mathcal{M}_{\mathcal{J}}(\mu, U)$. Because $e_{w_{1}}^{\alpha}>0$, we can use condition (b) of Definition 4.3.5 to give
us a monomial $\mu_{1} \in U$ and a monomial $w_{3}$ multiplicative for $\mu_{1}$ over $U\left(e_{w_{3}}^{\gamma}>0 \Rightarrow x_{\gamma} \in\right.$ $\mathcal{M}_{\mathcal{J}}\left(\mu_{1}, U\right)$ for all $\left.1 \leqslant \gamma \leqslant n\right)$ such that

$$
\begin{aligned}
u x_{i} & =\mu w_{1} w_{2} \\
& =\mu x_{k} w_{1}\left(\frac{w_{2}}{x_{k}}\right) \\
& =\mu_{1} w_{3} w_{1}\left(\frac{w_{2}}{x_{k}}\right) .
\end{aligned}
$$

If $\left.\mu_{1}\right|_{\mathcal{J}} u x_{i}$, then the proof is complete, with $\nu=\mu_{1}$. Otherwise there must be a variable $x_{k^{\prime}}$ appearing in the monomial $w_{1}\left(\frac{w_{2}}{x_{k}}\right)$ such that $x_{k^{\prime}} \notin \mathcal{M}_{\mathcal{J}}\left(\mu_{1}, U\right)$. To use condition (b) of Definition 4.3.5 to yield a monomial $\mu_{2} \in U$ and a monomial $w_{4}$ multiplicative for $\mu_{2}$ over $U$ such that

$$
\mu_{1} w_{3} w_{1}\left(\frac{w_{2}}{x_{k}}\right)=\mu_{2} w_{4}\left(\frac{w_{1} w_{2}}{x_{k} x_{k^{\prime}}}\right) w_{3},
$$

it is sufficient to demonstrate that at least one variable appearing in the monomial $w_{3} w_{1}\left(\frac{w_{2}}{x_{k}}\right)$ is multiplicative for $\mu_{1}$ over the set $U$. We will do this by showing that $x_{\alpha} \in \mathcal{M}_{\mathcal{J}}\left(\mu_{1}, U\right)$ (recall that $e_{w_{1}}^{\alpha}>0$ ).

By the definition of the Janet involutive division,

$$
\begin{equation*}
e_{\mu_{1}}^{\gamma}=e_{\mu}^{\gamma} \text { for all } k<\gamma \leqslant n \tag{4.8}
\end{equation*}
$$

and

$$
\begin{equation*}
e_{\mu_{1}}^{k}=e_{\mu}^{k}+1 \tag{4.9}
\end{equation*}
$$

so that $\mu<\mu_{1}$ in the InvLex monomial ordering. If we can show that $\alpha>k$, then it is clear from Equation (4.8) and $x_{\alpha} \in \mathcal{M}_{\mathcal{J}}(\mu, U)$ that $x_{\alpha} \in \mathcal{M}_{\mathcal{J}}\left(\mu_{1}, U\right)$.

- If $\alpha>\beta$, then $\alpha>k$ because $k \leqslant \beta$ by definition.
- If $\alpha=\beta$, then $\alpha>k$ if $k<\beta$; otherwise $k=\beta$ in which case $x_{\alpha} \in \mathcal{M}_{\mathcal{J}}(\mu, U)$ is contradicted by Equations (4.8) and (4.9).
- If $\alpha<\beta$, then $e_{\mu}^{\gamma}=e_{w}^{\gamma}$ for all $\alpha<\gamma \leqslant n$. Thus $k \leqslant \alpha$ otherwise $x_{k} \in \mathcal{M}_{\mathcal{J}}(w, U \cup$ $\{w\}) \Rightarrow x_{k} \in \mathcal{M}_{\mathcal{J}}(\mu, U)$, a contradiction. Further, $k=\alpha$ is not allowed because $x_{\alpha} \in \mathcal{M}_{\mathcal{J}}(\mu, U)$ and $x_{k} \notin \mathcal{M}_{\mathcal{J}}(\mu, U)$ cannot both be true; therefore $\alpha>k$ again.

If $\left.\mu_{2}\right|_{\mathcal{J}} u x_{i}$, then the proof is complete, with $\nu=\mu_{2}$. Otherwise we proceed by induction to obtain the sequence shown below (Equation (4.10)), which is valid because $\mu_{\sigma-1}<\mu_{\sigma}$ (for $\sigma \geqslant 2$ ) in the InvLex monomial ordering allows us to prove that the variable $x_{\alpha}$ (that appears in the monomial $w_{1}$ ) is multiplicative (over $U$ ) for the monomial $\mu_{\sigma}$; this in turn enables us to construct the next entry in the sequence by using condition (b) of Definition 4.3.5.

$$
\begin{equation*}
\mu w_{1} w_{2}=\mu_{1} w_{3} w_{1}\left(\frac{w_{2}}{x_{k}}\right)=\mu_{2} w_{4}\left(\frac{w_{1} w_{2}}{x_{k} x_{k^{\prime}}}\right) w_{3}=\mu_{3} w_{5}\left(\frac{w_{1} w_{2} w_{3}}{x_{k} x_{k^{\prime}} x_{k^{\prime \prime}}}\right) w_{4}=\cdots \tag{4.10}
\end{equation*}
$$

Because $\mu<\mu_{1}<\mu_{2}<\cdots$ in the InvLex monomial ordering, elements of the sequence $\mu, \mu_{1}, \mu_{2}, \ldots$ are distinct. It follows that the sequence in Equation (4.10) is finite (terminating with the required $\nu$ ) because $\mu$ and the $\mu_{\sigma}$ (for $\sigma \geqslant 1$ ) are all divisors of the monomial $u x_{i}$, of which there are only a finite number of.

Remark 4.3.8 The above proof that Janet is a constructive involutive division does not use the property of Janet being a continuous involutive division, unlike the proofs found in [25] and [50].

### 4.4 The Involutive Basis Algorithm

To compute an Involutive Basis for an ideal $J$ with respect to some admissible monomial ordering $O$ and some involutive division $I$, it is sufficient to compute a Locally Involutive Basis for $J$ with respect to $I$ and $O$ if $I$ is continuous; and we can compute this Locally Involutive Basis by considering only prolongations whose lead monomials lie outside the current involutive span if $I$ is constructive. Let us now consider Algorithm 9, an algorithm to construct an Involutive Basis for $J$ (with respect to $I$ and $O$ ) in exactly this manner.

The algorithm starts by autoreducing the input basis $F$ using Algorithm 8. We then construct a set $S$ containing all the possible prolongations of elements of $F$, before recursively (a) picking a polynomial $s$ from $S$ such that $\mathrm{LM}(s)$ is minimal in the chosen monomial ordering; (b) removing $s$ from $S$; and (c) finding the involutive remainder $s^{\prime}$ of $s$ with respect to $F$.

If during this loop a remainder $s^{\prime}$ is found that is nonzero, we exit the loop and autoreduce the set $F \cup\left\{s^{\prime}\right\}$, continuing thereafter to construct a new set $S$ and repeating the above process on this new set. If however all the prolongations in $S$ involutively reduce to zero,

```
Algorithm 9 The Commutative Involutive Basis Algorithm
Input: A Basis \(F=\left\{f_{1}, f_{2}, \ldots, f_{m}\right\}\) for an ideal \(J\) over a commutative polynomial
    ring \(R\left[x_{1}, \ldots x_{n}\right]\); an admissible monomial ordering \(O\); a continuous and constructive
    involutive division \(I\).
Output: An Involutive Basis \(G=\left\{g_{1}, g_{2}, \ldots, g_{p}\right\}\) for \(J\) (in the case of termination).
    \(G=\emptyset ;\)
    \(F=\) Autoreduce \((F)\);
    while ( \(G==\emptyset\) ) do
        \(S=\left\{x_{i} f \mid f \in F, x_{i} \notin \mathcal{M}_{I}(f, F)\right\} ;\)
        \(s^{\prime}=0\);
        while \((S \neq \emptyset)\) and \(\left(s^{\prime}==0\right)\) do
            Let \(s\) be a polynomial in \(S\) whose lead monomial is minimal with respect to \(O\);
            \(S=S \backslash\{s\} ;\)
            \(s^{\prime}=\operatorname{Rem}_{I}(s, F)\);
        end while
        if \(\left(s^{\prime} \neq 0\right)\) then
            \(F=\) Autoreduce \(\left(F \cup\left\{s^{\prime}\right\}\right)\);
        else
            \(G=F ;\)
        end if
    end while
    return \(G\);
```

then by definition $F$ is a Locally Involutive Basis, and so we can exit the algorithm with this basis. The correctness of Algorithm 9 is therefore clear; termination however requires us to show that each involutive division used with the algorithm is Noetherian and stable.

Definition 4.4.1 An involutive division $I$ is Noetherian if, given any finite set of monomials $U$, there is a finite Involutive Basis $V \supseteq U$ with respect to $I$ and some arbitrary admissible monomial ordering $O$.

## Proposition 4.4.2 The Thomas and Janet divisions are Noetherian.

Proof: Let $U=\left\{u_{1}, \ldots, u_{m}\right\}$ be an arbitrary set of monomials over a polynomial ring $\mathcal{R}=R\left[x_{1}, \ldots, x_{n}\right]$ generating an ideal $J$. We will explicitly construct an Involutive Basis $V$ for $U$ with respect to some arbitrary admissible monomial ordering $O$.

Janet (Adapted from [50], Lemma 2.13). Let $\mu \in \mathcal{R}$ be the monomial with multidegree $\left(e_{\mu}^{1}, e_{\mu}^{2}, \ldots, e_{\mu}^{n}\right)$ defined as follows: $e_{\mu}^{i}=\max _{u \in U} e_{u}^{i}(1 \leqslant i \leqslant n)$. We claim that the set $V$ containing all monomials $v \in J$ such that $v \mid \mu$ is an Involutive Basis for $U$ with respect to the Janet involutive division and $O$. To prove the claim, first note that $V$ is a basis for $J$ because $U \subseteq V$ and $V \subset J$; to prove that $V$ is a Janet Involutive Basis for $J$ we have to show that all multiples of elements of $V$ involutively reduce to zero using $V$, which we shall do by showing that all members of the ideal involutively reduce to zero using $V$.

Let $p$ be an arbitrary element of $J$. If $p \in V$, then trivially $p \in \mathcal{C}_{\mathcal{J}}(V)$ and so $p$ involutively reduces to zero using $V$. Otherwise set $X=\left\{x_{i}\right.$ such that $\left.e_{\mathrm{LM}(p)}^{i}>e_{\mu}^{i}\right\}$, and define the monomial $p^{\prime}$ by $e_{p^{\prime}}^{i}=e_{\mathrm{LM}(p)}^{i}$ for $x_{i} \notin X$; and $e_{p^{\prime}}^{i}=e_{\mu}^{i}$ for $x_{i} \in X$ (so that $e_{p^{\prime}}^{i}=$ $\left.\min \left\{e_{\mathrm{LM}(p)}^{i}, e_{\mu}^{i}\right\}\right)$. By construction of the set $V$ and by the definition of $\mu$, it follows that $v^{\prime} \in V$ and $X \subseteq \mathcal{M}_{\mathcal{J}}\left(p^{\prime}, V\right)$. But this implies that $\operatorname{LM}(p) \in \mathcal{C}_{\mathcal{J}}\left(p^{\prime}, V\right)$, and thus $p \xrightarrow[\mathcal{J}^{\prime}]{ }(p-\mathrm{LM}(p))$. By induction and by the admissibility of $O, p \xrightarrow[\mathcal{J}^{\prime}]{ } 0$ and thus $V$ is a finite Janet Involutive Basis for $J$.

Thomas. We use the same proof as for Janet above, replacing "Janet" by "Thomas" and " $\mathcal{J}$ " by " $\mathcal{T}$ ".

Proposition 4.4.3 The Pommaret division is not Noetherian.

Proof: Let $J$ be the ideal generated by the monomial $u:=x y$ over the polynomial ring $\mathbb{Q}[x, y]$. For the Pommaret division, $\mathcal{M}_{\mathcal{P}}(u)=\{x\}$, and it is clear that $\mathcal{M}_{\mathcal{P}}(v)=\{x\}$ for
all $v \in J$ as $v \in J \Rightarrow v=(x y) p$ for some polynomial $p$. It follows that no finite Pommaret Involutive Basis exists for $J$ as no prolongation by the variable $y$ of any polynomial $p \in J$ is involutively divisible by some other polynomial $p^{\prime} \in J$; the Pommaret Involutive Basis for $J$ is therefore the infinite basis $\left\{x y, x y^{2}, x y^{3}, \ldots\right\}$.

Definition 4.4.4 Let $u$ and $v$ be two distinct monomials such that $u \mid v$. An involutive division $I$ is stable if $\operatorname{Rem}_{I}(v,\{u, v\},\{u\})=v$. In other words, $u$ is not an involutive divisor of $v$ with respect to $I$ when multiplicative variables are taken over the set $\{u, v\}$.

Proposition 4.4.5 The Thomas and Janet divisions are stable.

Proof: Let $u$ and $v$ have corresponding multidegrees $\left(e_{u}^{1}, \ldots, e_{u}^{n}\right)$ and $\left(e_{v}^{1}, \ldots, e_{v}^{n}\right)$. If $u \mid v$ and if $u$ and $v$ are different, then we must have $e_{u}^{i}<e_{v}^{i}$ for at least one $1 \leqslant i \leqslant n$.

Thomas. By definition, $x_{i} \notin \mathcal{M}_{\mathcal{T}}(u,\{u, v\})$, so that $\operatorname{Rem}_{\mathcal{T}}(v,\{u, v\},\{u\})=v$.
Janet. Let $j$ be the greatest integer such that $e_{u}^{j}<e_{v}^{j}$. Then, as $e_{u}^{k}=e_{v}^{k}$ for all $j<k \leqslant n$, it follows that $x_{j} \notin \mathcal{M}_{\mathcal{J}}(u,\{u, v\})$, and so $\operatorname{Rem}_{\mathcal{J}}(v,\{u, v\},\{u\})=v$.

Proposition 4.4.6 The Pommaret division is not stable.

Proof: Consider the two monomials $u:=x$ and $v:=x^{2}$ over the polynomial ring $\mathbb{Q}[x]$. Because $\mathcal{M}_{\mathcal{P}}(u,\{u, v\})=\{x\}$, it is clear that $\left.u\right|_{\mathcal{P}} v$, and so the Pommaret involutive division is not stable.

Remark 4.4.7 Stability ensures that any set of distinct monomials is autoreduced. In particular, if a set $U$ of monomials is autoreduced, and if we add a monomial $u \notin U$ to $U$, then the resultant set $U \cup\{u\}$ is also autoreduced. This contradicts a statement made on page 24 of [50], where it is claimed that if we add an involutively irreducible prolongation $u x_{i}$ of a monomial $u$ from an autoreduced set of monomials $U$ to that set, then the resultant set is also autoreduced regardless of whether or not the involutive division used is stable ${ }^{2}$. For a counterexample, consider the set of monomials $U:=\left\{u_{1}, u_{2}\right\}=\left\{x y, x^{2} y^{2}\right\}$ over the polynomial ring $\mathbb{Q}[x, y]$, and let the involutive division be Pommaret.

| $u$ | $\mathcal{M}_{\mathcal{P}}(u, U)$ |
| :---: | :---: |
| $x y$ | $\{x\}$ |
| $x^{2} y^{2}$ | $\{x\}$ |

[^8]Because the variable $y$ is nonmultiplicative for the monomial $x y$, it is clear that the set $U$ is autoreduced. Consider the prolongation $x y^{2}$ of the monomial $u_{1}$ by the variable $y$. This prolongation is involutively irreducible with respect to $U$, but if we add the prolongation to $U$ to obtain the set $V:=\left\{v_{1}, v_{2}, v_{3}\right\}=\left\{x y, x^{2} y^{2}, x y^{2}\right\}$, then $v_{3}$ will involutively reduce $v_{2}$, contradicting the claim that the set $V$ is autoreduced.

| $v$ | $\mathcal{M}_{\mathcal{P}}(v, V)$ |
| :---: | :---: |
| $x y$ | $\{x\}$ |
| $x^{2} y^{2}$ | $\{x\}$ |
| $x y^{2}$ | $\{x\}$ |

Proposition 4.4.8 Algorithm 9 always terminates when used with a Noetherian and stable involutive division.

Proof: Let $I$ be a Noetherian and stable involutive division, and consider the computation (using Algorithm 9) of an Involutive Basis for a set of polynomials $F$ with respect to $I$ and some admissible monomial ordering $O$. The algorithm begins by autoreducing $F$ to give a basis (which we shall denote by $F_{1}$ ) generating the same ideal $J$ as $F$. Each pass of the algorithm then produces a basis $F_{i+1}=$ Autoreduce $\left(F_{i} \cup\left\{s_{i}^{\prime}\right\}\right)$ generating $J(i \geqslant 1)$, where each $s_{i}^{\prime} \neq 0$ is an involutively reduced prolongation. Consider the monomial ideal $\left\langle\operatorname{LM}\left(F_{i}\right)\right\rangle$ generated by the lead monomials of the set $F_{i}$. Claim:

$$
\begin{equation*}
\left\langle\operatorname{LM}\left(F_{1}\right)\right\rangle \subseteq\left\langle\operatorname{LM}\left(F_{2}\right)\right\rangle \subseteq\left\langle\operatorname{LM}\left(F_{3}\right)\right\rangle \subseteq \cdots \tag{4.11}
\end{equation*}
$$

is an ascending chain of monomial ideals.
Proof of Claim: It is sufficient to show that if an arbitrary polynomial $f \in F_{i}$ does not appear in $F_{i+1}$, then there must be a polynomial $f^{\prime} \in F_{i+1}$ such that $\mathrm{LM}\left(f^{\prime}\right) \mid \mathrm{LM}(f)$. It is clear that such an $f^{\prime}$ will exist if the lead monomial of $f$ is not reduced during autoreduction; otherwise a polynomial $p$ reduces the lead monomial of $f$ during autoreduction, so that $\left.\mathrm{LM}(p)\right|_{I} \mathrm{LM}(f)$. If there exists a polynomial $p^{\prime} \in F_{i+1}$ such that $\operatorname{LM}\left(p^{\prime}\right)=\operatorname{LM}(p)$, we are done; otherwise we proceed by induction on $p$ to obtain a polynomial $q$ such that $\left.\operatorname{LM}(q)\right|_{I} \operatorname{LM}(p)$. Because $\operatorname{deg}(\operatorname{LM}(f))>\operatorname{deg}(\operatorname{LM}(p))>\operatorname{deg}(\operatorname{LM}(q))>\cdots$, this process is guaranteed to terminate with the required $f^{\prime}$.

By the Ascending Chain Condition (Corollary 2.2.6), the chain in Equation (4.11) must
eventually become constant, so there must be an integer $N(N \geqslant 1)$ such that

$$
\left\langle\operatorname{LM}\left(F_{N}\right)\right\rangle=\left\langle\operatorname{LM}\left(F_{N+1}\right)\right\rangle=\cdots
$$

Claim: If $F_{k+1}=$ Autoreduce $\left(F_{k} \cup\left\{s_{k}^{\prime}\right\}\right)$ for some $k \geqslant N$, then $\operatorname{LM}\left(s_{k}^{\prime}\right)=\operatorname{LM}\left(f x_{i}\right)$ for some polynomial $f \in F_{k}$ and some variable $x_{i} \notin \mathcal{M}_{I}\left(f, F_{k}\right)$ such that $s_{k}^{\prime}=\operatorname{Rem}_{I}\left(f x_{i}, F_{k}\right)$.

Proof of Claim: Assume to the contrary that $\operatorname{LM}\left(s_{k}^{\prime}\right) \neq \operatorname{LM}\left(f x_{i}\right)$. Then because $s_{k}^{\prime}=\operatorname{Rem}_{I}\left(f x_{i}, F_{k}\right)$, it follows that $\operatorname{LM}\left(s_{k}^{\prime}\right)<\operatorname{LM}\left(f x_{i}\right)$. But $\left\langle\operatorname{LM}\left(F_{k}\right)\right\rangle=\left\langle\operatorname{LM}\left(F_{k+1}\right)\right\rangle$, so that $\operatorname{LM}\left(s_{k}^{\prime}\right)=\operatorname{LM}\left(f^{\prime} u\right)$ for some $f^{\prime} \in F_{k}$ and some monomial $u$ containing at least one variable $x_{j} \notin \mathcal{M}_{I}\left(f^{\prime}, F_{k}\right)$ (otherwise $s_{k}^{\prime}$ can be involutively reduced with respect to $F_{k}$, a contradiction).

Because $O$ is admissible, $1 \leqslant \frac{u}{x_{j}}$ and therefore $x_{j} \leqslant u$, so that $\operatorname{LM}\left(f^{\prime} x_{j}\right) \leqslant \operatorname{LM}\left(f^{\prime} u\right)<$ $\mathrm{LM}\left(f x_{i}\right)$. But the prolongation $f x_{i}$ was chosen so that its lead monomial is minimal amongst the lead monomials of all prolongations of elements of $F_{k}$ that do not involutively reduce to zero; the prolongation $f^{\prime} x_{k}$ must therefore involutively reduce to zero, so that $\operatorname{LM}\left(f^{\prime} x_{j}\right)=\operatorname{LM}\left(f^{\prime \prime} u^{\prime}\right)$ for some polynomial $f^{\prime \prime} \in F_{k}$ and some monomial $u^{\prime}$ that is multiplicative for $f^{\prime \prime}$ over $F_{k}$. But $s_{k}^{\prime}$ is involutively irreducible with respect to $F_{k}$, so a variable $x_{j}^{\prime} \notin \mathcal{M}_{I}\left(f^{\prime \prime}, F_{k}\right)$ must appear in the monomial $\frac{u}{x_{j}}$.
It is now clear that we can construct a sequence $f^{\prime} x_{j}, f^{\prime \prime} x_{j}^{\prime}, \ldots$ of prolongations. But $I$ is continuous, so all elements in the corresponding sequence $\operatorname{LM}\left(f^{\prime}\right), \mathrm{LM}\left(f^{\prime \prime}\right), \ldots$ of monomials must be distinct. Because $F_{k}$ is finite, it follows that the sequence of prolongations will terminate with a prolongation that does not involutively reduce to zero and whose lead monomial is less than the monomial $\operatorname{LM}\left(f x_{i}\right)$, contradicting our assumptions. Thus $\mathrm{LM}\left(s_{k}^{\prime}\right)$ for $k \geqslant N$ is always equal to the lead monomial of some prolongation of some polynomial $f \in F_{k}$.

Consider now the set of monomials $\operatorname{LM}\left(F_{k+1}\right)$. Claim: $\operatorname{LM}\left(F_{k+1}\right)=\operatorname{LM}\left(F_{k}\right) \cup\left\{\operatorname{LM}\left(s_{k}^{\prime}\right)\right\}$ for all $k \geqslant N$, so that when autoreducing the set $F_{k} \cup\left\{s_{k}^{\prime}\right\}$, no leading monomial is involutively reducible.

Proof of Claim: Consider an arbitrary polynomial $p \in F_{k} \cup\left\{s_{k}^{\prime}\right\}$. If $p=s_{k}^{\prime}$, then (by definition) $p$ is irreducible with respect to the set $F_{k}$, and so (by condition (b) of Definition 4.1.4) $p$ will also be irreducible with respect to the set $F_{k} \cup\left\{s_{k}^{\prime}\right\}$. If $p \neq s_{k}^{\prime}$, then $p$ is irreducible with respect to the set $F_{k}$ (as the set $F_{k}$ is autoreduced), and so (again by condition (b) of Definition 4.1.4) the only polynomial in the set $F_{k} \cup\left\{s_{k}^{\prime}\right\}$
that can involutively reduce the polynomial $p$ is the polynomial $s_{k}^{\prime}$. But $I$ is stable, so that $s_{k}^{\prime}$ cannot involutively reduce $\operatorname{LM}(p)$. It follows that a polynomial $p^{\prime}$ will appear in the autoreduced set $F_{k+1}$ such that $\operatorname{LM}\left(p^{\prime}\right)=\operatorname{LM}(p)$, and thus $\operatorname{LM}\left(F_{k+1}\right)=\operatorname{LM}\left(F_{k}\right) \cup$ $\left\{\mathrm{LM}\left(s_{k}^{\prime}\right)\right\}$ as required.

For the final part of the proof, consider the basis $F_{N}$. Because $I$ is Noetherian, there exists a finite Involutive Basis $G_{N}$ for the ideal generated by the set of lead monomials $\operatorname{LM}\left(F_{N}\right)$, where $G_{N} \supseteq \operatorname{LM}\left(F_{N}\right)$. Let $f x_{i}$ be the prolongation chosen during the $N$-th iteration of Algorithm 9, so that $\operatorname{LM}\left(f x_{i}\right) \notin \mathcal{C}_{I}\left(F_{N}\right)$. Because $G_{N}$ is an Involutive Basis for $\operatorname{LM}\left(F_{N}\right)$, there must be a monomial $g \in G_{N}$ such that $\left.g\right|_{I} \operatorname{LM}\left(f x_{i}\right)$. Claim: $g=\operatorname{LM}\left(f x_{i}\right)$.

Proof of Claim: We proceed by showing that if $g \neq \operatorname{LM}\left(f x_{i}\right)$, then $g \in \mathcal{C}_{I}\left(\operatorname{LM}\left(F_{N}\right)\right)$ so that (because of condition (b) of Definition 4.1.4) $\mathrm{LM}\left(f x_{i}\right) \in \mathcal{C}_{I}\left(G_{N}\right) \Rightarrow \operatorname{LM}\left(f x_{i}\right) \in$ $\mathcal{C}_{I}\left(g, \operatorname{LM}\left(F_{N}\right) \cup\{g\}\right)$, contradicting the constructivity of $I$ (Definition 4.3.5).

Assume that $g \neq \operatorname{LM}\left(f x_{i}\right)$. Because $\left\langle G_{N}\right\rangle=\left\langle\operatorname{LM}\left(F_{N}\right)\right\rangle$, there exists a polynomial $f_{1} \in F_{N}$ such that $\operatorname{LM}\left(f_{1}\right) \mid g$. If $\left.\operatorname{LM}\left(f_{1}\right)\right|_{I} g$ with respect to $F_{N}$, then we are done. Otherwise $\operatorname{LM}(g)=\operatorname{LM}\left(f_{1}\right) u_{1}$ for some monomial $u_{1} \neq 1$ containing at least one variable $x_{j_{1}} \notin$ $\mathcal{M}_{I}\left(f_{1}, F_{N}\right)$. Because $\operatorname{deg}(g)<\operatorname{deg}\left(\operatorname{LM}\left(f x_{i}\right)\right)$ and $\operatorname{LM}\left(f_{1}\right) x_{j_{1}} \mid \operatorname{LM}\left(f x_{i}\right)$, we must have $\operatorname{LM}\left(f_{1}\right) x_{j_{1}}<\operatorname{LM}\left(f x_{i}\right)$ with respect to our chosen monomial ordering, so that $\operatorname{LM}\left(f_{1}\right) x_{j_{1}} \in$ $\mathcal{C}_{I}\left(F_{N}\right)$ by definition of how the prolongation $f x_{i}$ was chosen. It follows that there exists a polynomial $f_{2} \in F_{N}$ such that $\left.\operatorname{LM}\left(f_{2}\right)\right|_{I} \operatorname{LM}\left(f_{1}\right) x_{j_{1}}$ with respect to $F_{N}$. If $\left.\operatorname{LM}\left(f_{2}\right)\right|_{I} g$ with respect to $F_{N}$, then we are done. Otherwise we iterate $\left(\operatorname{LM}\left(f_{1}\right) x_{j_{1}}=\operatorname{LM}\left(f_{2}\right) u_{2}\right.$ for some monomial $u_{2}$ containing at least one variable $\left.x_{j_{2}} \notin \mathcal{M}_{I}\left(f_{2}, F_{N}\right) \ldots\right)$ to obtain the sequence ( $f_{1}, f_{2}, f_{3}, \ldots$ ) of polynomials, where the lead monomial of each element in the sequence divides $g$ and $\left.\operatorname{LM}\left(f_{k+1}\right)\right|_{I} \operatorname{LM}\left(f_{k}\right) x_{j_{k}}$ with respect to $F_{N}$ for all $k \geqslant 1$. Because $I$ is continuous, this sequence must be finite, terminating with a polynomial $f_{k} \in F_{N}$ (for some $k \geqslant 1$ ) such that $\left.f_{k}\right|_{I} g$ with respect to $F_{N}$.

It follows that during the $N$-th iteration of the algorithm, a polynomial is added to the current basis $F_{N}$ whose lead monomial is a member of the Involutive Basis $G_{N}$. By induction, every step of the algorithm after the $N$-th step also adds a polynomial to the current basis whose lead monomial is a member of $G_{N}$. Because $G_{N}$ is a finite set, after a finite number of steps the basis $\operatorname{LM}\left(F_{k}\right)$ (for some $k \geqslant N$ ) will contain all the elements of $G_{N}$. We can therefore deduce that $\operatorname{LM}\left(F_{k}\right)=G_{N}$; it follows that $\operatorname{LM}\left(F_{k}\right)$ is an Involutive Basis, and so $F_{k}$ is also an Involutive Basis.

Theorem 4.4.9 Every Involutive Basis is a Gröbner Basis.

Proof: Let $G=\left\{g_{1}, \ldots, g_{m}\right\}$ be an Involutive Basis with respect to some involutive division $I$ and some admissible monomial ordering $O$, where each $g_{i} \in G$ (for all $1 \leqslant i \leqslant$ $m$ ) is a member of the polynomial ring $R\left[x_{1}, \ldots, x_{n}\right]$. To prove that $G$ is a Gröbner Basis, we must show that all S-polynomials

$$
\mathrm{S}-\operatorname{pol}\left(g_{i}, g_{j}\right)=\frac{\operatorname{lcm}\left(\mathrm{LM}\left(g_{i}\right), \mathrm{LM}\left(g_{j}\right)\right)}{\operatorname{LT}\left(g_{i}\right)} g_{i}-\frac{\operatorname{lcm}\left(\mathrm{LM}\left(g_{i}\right), \mathrm{LM}\left(g_{j}\right)\right)}{\operatorname{LT}\left(g_{j}\right)} g_{j}
$$

conventionally reduce to zero using $G(1 \leqslant i, j \leqslant m, i \neq j)$. Because $G$ is an Involutive Basis, it is clear that $\frac{\operatorname{lcm}\left(\operatorname{LM}\left(g_{i}\right), \mathrm{LM}\left(g_{j}\right)\right)}{\operatorname{LT}\left(g_{i}\right)} g_{i}{ }_{I}{ }_{G} 0$ and $\frac{\operatorname{lcm}\left(\operatorname{LM}\left(g_{i}\right), \operatorname{LM}\left(g_{j}\right)\right)}{\operatorname{LT}\left(g_{j}\right)} g_{j}{ }_{I}{ }_{G} 0$. By Proposition 4.2.4, it follows that $\mathrm{S}-\mathrm{pol}\left(g_{i}, g_{j}\right) \vec{I}_{G} 0$. But every involutive reduction is a conventional reduction, so we can deduce that S-pol $\left(g_{i}, g_{j}\right) \rightarrow_{G} 0$ as required.

Lemma 4.4.10 Remainders are involutively unique with respect to Involutive Bases.

Proof: Given an Involutive Basis $G$ with respect to some involutive division $I$ and some admissible monomial ordering $O$, Theorem 4.4.9 tells us that $G$ is a Gröbner Basis with respect to $O$ and thus remainders are conventionally unique with respect to $G$. To prove that remainders are involutively unique with respect to $G$, we must show that the conventional and involutive remainders of an arbitrary polynomial $p$ with respect to $G$ are identical. For this it is sufficient to show that a polynomial $p$ is conventionally reducible by $G$ if and only if it is involutively reducible by $G .(\Rightarrow)$ Trivial as every involutive reduction is a conventional reduction. ( $\Leftarrow$ ) If a polynomial $p$ is conventionally reducible by a polynomial $g \in G$, it follows that $\mathrm{LM}(p)=\operatorname{LM}(g) u$ for some monomial $u$. But $G$ is an Involutive Basis, so there must exist a polynomial $g^{\prime} \in G$ such that $\mathrm{LM}(g) u=\mathrm{LM}\left(g^{\prime}\right) u^{\prime}$ for some monomial $u^{\prime}$ that is multiplicative (over $G$ ) for $g^{\prime}$. Thus $p$ is also involutively reducible by $G$.

Example 4.4.11 Let us return to our favourite example of an ideal $J$ generated by the set of polynomials $F:=\left\{f_{1}, f_{2}\right\}=\left\{x^{2}-2 x y+3,2 x y+y^{2}+5\right\}$ over the polynomial ring $\mathbb{Q}[x, y, z]$. To compute an Involutive Basis for $F$ with respect to the DegLex monomial ordering and the Janet involutive division $\mathcal{J}$, we apply Algorithm 9 to $F$, in which the first task is to autoreduce $F$. This produces the set $F=\left\{f_{2}, f_{3}\right\}=\left\{2 x y+y^{2}+5, x^{2}+y^{2}+8\right\}$ as output (because $f_{1}=x^{2}-2 x y+3{\underset{\mathcal{J}}{ } f_{2}}^{x^{2}}+y^{2}+8=: f_{3}$ and $f_{2}$ is involutively irreducible with respect to $f_{3}$ ), with multiplicative variables as shown below.

| Polynomial | $\mathcal{M}_{\mathcal{J}}\left(f_{i}, F\right)$ |
| :---: | :---: |
| $f_{2}=2 x y+y^{2}+5$ | $\{x, y\}$ |
| $f_{3}=x^{2}+y^{2}+8$ | $\{x\}$ |

The first set of prolongations of elements of $F$ is the set $S=\left\{f_{3} y\right\}=\left\{x^{2} y+y^{3}+8 y\right\}$. As this set only has one element, it is clear that on entering the second while loop of the algorithm, we must remove the polynomial $s=x^{2} y+y^{3}+8 y$ from $S$ and involutively reduce $s$ with respect to $F$ to give the polynomial $s^{\prime}=\frac{5}{4} y^{3}-\frac{5}{2} x+\frac{37}{4} y$ as follows.

$$
\begin{aligned}
s=x^{2} y+y^{3}+8 y & \overrightarrow{\mathcal{J}}_{f_{2}} \\
& =\quad-\frac{1}{2} x y^{2}+y^{3}-\frac{5}{2} x+8 y \\
& {\underset{\mathcal{J}}{f_{2}}} \quad-\frac{1}{2} x y^{2}+y^{3}-\frac{5}{2} x+8 y+\frac{1}{4} y\left(2 x y+y^{2}+5\right) \\
& =\frac{5}{4} y^{3}-\frac{5}{2} x+\frac{37}{4} y=s^{\prime}=: f_{4} .
\end{aligned}
$$

As the prolongation did not involutively reduce to zero, we exit from the second while loop of the algorithm and proceed by autoreducing the set $F \cup\left\{f_{4}\right\}=\left\{2 x y+y^{2}+5, x^{2}+\right.$ $\left.y^{2}+8, \frac{5}{4} y^{3}-\frac{5}{2} x+\frac{37}{4} y\right\}$. This process does not alter the set, so now we consider the prolongations of the three element set $F=\left\{f_{2}, f_{3}, f_{4}\right\}$.

| Polynomial | $\mathcal{M}_{\mathcal{J}}\left(f_{i}, F\right)$ |
| :---: | :---: |
| $f_{2}=2 x y+y^{2}+5$ | $\{x\}$ |
| $f_{3}=x^{2}+y^{2}+8$ | $\{x\}$ |
| $f_{4}=\frac{5}{4} y^{3}-\frac{5}{2} x+\frac{37}{4} y$ | $\{x, y\}$ |

We see that there are 2 prolongations to consider, so that $S=\left\{f_{2} y, f_{3} y\right\}=\left\{2 x y^{2}+y^{3}+\right.$ $\left.5 y, x^{2} y+y^{3}+8 y\right\}$. As $x y^{2}<x^{2} y$ in the DegLex monomial ordering, we must consider the prolongation $f_{2} y$ first.

$$
\begin{aligned}
f_{2} y=2 x y^{2}+y^{3}+5 y & {\underset{\mathcal{J}}{ }} f_{4} \\
& 2 x y^{2}+y^{3}+5 y-\frac{4}{5}\left(\frac{5}{4} y^{3}-\frac{5}{2} x+\frac{37}{4} y\right) \\
& =2 x y^{2}+2 x-\frac{12}{5} y=: f_{5} .
\end{aligned}
$$

As before, the prolongation did not involutively reduce to zero, so now we autoreduce the set $F \cup\left\{f_{5}\right\}=\left\{2 x y+y^{2}+5, x^{2}+y^{2}+8, \frac{5}{4} y^{3}-\frac{5}{2} x+\frac{37}{4} y, 2 x y^{2}+2 x-\frac{12}{5} y\right\}$. Again this leaves the set unchanged, so we proceed with the set $F=\left\{f_{2}, f_{3}, f_{4}, f_{5}\right\}$.

| Polynomial | $\mathcal{M}_{\mathcal{J}}\left(f_{i}, F\right)$ |
| :---: | :---: |
| $f_{2}=2 x y+y^{2}+5$ | $\{x\}$ |
| $f_{3}=x^{2}+y^{2}+8$ | $\{x\}$ |
| $f_{4}=\frac{5}{4} y^{3}-\frac{5}{2} x+\frac{37}{4} y$ | $\{x, y\}$ |
| $f_{5}=2 x y^{2}+2 x-\frac{12}{5} y$ | $\{x\}$ |

This time, $S=\left\{f_{2} y, f_{3} y, f_{5} y\right\}=\left\{2 x y^{2}+y^{3}+5 y, x^{2} y+y^{3}+8 y, 2 x y^{3}+2 x y-\frac{12}{5} y^{2}\right\}$, and we must consider the prolongation $f_{2} y$ first.

$$
\begin{aligned}
f_{2} y=2 x y^{2}+y^{3}+5 y & \overrightarrow{\mathcal{J}}_{f_{5}} \\
& =2 x y^{2}+y^{3}+5 y-\left(2 x y^{2}+2 x-\frac{12}{5} y\right) \\
& y^{3}-2 x+\frac{37}{5} y \\
& \overrightarrow{\mathcal{J}}_{f_{4}} y^{3}-2 x+\frac{37}{5} y-\frac{4}{5}\left(\frac{5}{4} y^{3}-\frac{5}{2} x+\frac{37}{4} y\right) \\
& =0 .
\end{aligned}
$$

Because the prolongation involutively reduced to zero, we move on to look at the next prolongation $f_{3} y$ (which comes from the revised set $S=\left\{f_{3} y, f_{5} y\right\}=\left\{x^{2} y+y^{3}+\right.$ $\left.\left.8 y, 2 x y^{3}+2 x y-\frac{12}{5} y^{2}\right\}\right)$.

$$
\begin{aligned}
f_{3} y=x^{2} y+y^{3}+8 y & \overrightarrow{\mathcal{J}}_{f_{2}} \\
& x^{2} y+y^{3}+8 y-\frac{1}{2} x\left(2 x y+y^{2}+5\right) \\
& =-\frac{1}{2} x y^{2}+y^{3}-\frac{5}{2} x+8 y \\
& \overrightarrow{\mathcal{J}}_{f_{5}} \\
& =-\frac{1}{2} x y^{2}+y^{3}-\frac{5}{2} x+8 y+\frac{1}{4}\left(2 x y^{2}+2 x-\frac{12}{5} y\right) \\
& y^{3}-2 x+\frac{37}{5} y \\
& =0 .
\end{aligned}
$$

Finally, we look at the prolongation $f_{5} y$ from the set $S=\left\{2 x y^{3}+2 x y-\frac{12}{5} y^{2}\right\}$.

$$
\begin{aligned}
f_{5} y=2 x y^{3}+2 x y-\frac{12}{5} y^{2} & {\underset{\mathcal{J}}{f_{4}}} \\
& =4 x^{2}-\frac{64}{5} x y-\frac{12}{5} y^{2} \\
& \overrightarrow{\mathcal{J}}_{f_{3}} \\
& =4 x^{2}-\frac{64}{5} x y-\frac{12}{5} y^{2}-4\left(x^{2}+y^{2}+8\right) \\
& -\frac{64}{5} x y-\frac{32}{5} y^{2}-32 \\
& \overrightarrow{\mathcal{J}}_{f_{2}} \\
& =0 . \frac{64}{5} x y-\frac{32}{5} y^{2}-32+\frac{52}{5}\left(2 x y+y^{2}+5\right)
\end{aligned}
$$

Because this prolongation also involutively reduced to zero using $F$, we are left with $S=\emptyset$, which means that the algorithm now terminates with the Janet Involutive Basis $G=\left\{2 x y+y^{2}+5, x^{2}+y^{2}+8, \frac{5}{4} y^{3}-\frac{5}{2} x+\frac{37}{4} y, 2 x y^{2}+2 x-\frac{12}{5} y\right\}$ as output.

### 4.5 Improvements to the Involutive Basis Algorithm

### 4.5.1 Improved Algorithms

In [58], Zharkov and Blinkov introduced an algorithm for computing an Involutive Basis and proved its termination for zero-dimensional ideals. This work led other researchers to produce improved versions of the algorithm (see for example [4], [13], [23], [26], [27] and [28]); improvements made to the algorithm include the introduction of selection strategies (which, as we have seen in the proof of Proposition 4.4.8, are crucial for proving the termination of the algorithm in general), and the introduction of criteria (analogous to Buchberger's criteria) allowing the a priori detection of prolongations that involutively reduce to zero.

### 4.5.2 Homogeneous Involutive Bases

When computing an Involutive Basis, a prolongation of a homogeneous polynomial is another homogeneous polynomial, and the involutive reduction of a homogeneous polynomial by a set of homogeneous polynomials yields another homogeneous polynomial. It would therefore be entirely feasible for a program computing Involutive Bases for ho-
mogeneous input bases to take advantage of the properties of homogeneous polynomial arithmetic.

It would also be desirable to be able to use such a program on input bases containing nonhomogeneous polynomials. The natural way to do this would be to modify the procedure outlined in Definition 2.5.7 by replacing every occurrence of the phrase "a Gröbner Basis" by the phrase "an Involutive Basis", thus creating the following definition.

Definition 4.5.1 Let $F=\left\{f_{1}, \ldots, f_{m}\right\}$ be a non-homogeneous set of polynomials. To compute an Involutive Basis for $F$ using a program that only accepts sets of homogeneous polynomials as input, we proceed as follows.
(a) Construct a homogeneous set of polynomials $F^{\prime}=\left\{h\left(f_{1}\right), \ldots, h\left(f_{m}\right)\right\}$.
(b) Compute an Involutive Basis $G^{\prime}$ for $F^{\prime}$.
(c) Dehomogenise each polynomial $g^{\prime} \in G^{\prime}$ to obtain a set of polynomials $G$.

Ideally, we would like to say that $G$ is always an Involutive Basis for $F$ as long as the monomial ordering used is extendible, mirroring the conclusion reached in Definition 2.5.7. However, we will only prove the validity of this statement in the case that the set $G$ is autoreduced, and also only for certain combinations of monomial orderings and involutive divisions - all combinations will not work, as the following example demonstrates.

Example 4.5.2 Let $F:=\left\{x_{1}^{2}+x_{2}^{3}, x_{1}+x_{3}^{3}\right\}$ be a basis generating an ideal $J$ over the polynomial ring $\mathbb{Q}\left[x_{1}, x_{2}, x_{3}\right]$, and let the monomial ordering be Lex. Computing an Involutive Basis for $F$ with respect to the Janet involutive division using Algorithm 9, we obtain the set $G:=\left\{x_{2}^{3}+x_{3}^{6}, x_{1} x_{2}^{2}+x_{2}^{2} x_{3}^{3}, x_{1} x_{2}+x_{2} x_{3}^{3}, x_{1}^{2}-x_{3}^{6}, x_{1}+x_{3}^{3}\right\}$.

Taking the homogeneous route, we can homogenise $F$ (with respect to Lex) to obtain the set $F^{\prime}:=\left\{x_{1}^{2} y+x_{2}^{3}, x_{1} y^{2}+x_{3}^{3}\right\}$ over the polynomial ring $\mathbb{Q}\left[x_{1}, x_{2}, x_{3}, y\right]$. Computing an Involutive Basis for $F^{\prime}$ with respect to the Janet involutive division, we obtain the set $G^{\prime}:=\left\{x_{2}^{3} y^{3}+x_{3}^{6}, x_{1} x_{2}^{2} y^{3}+x_{2}^{2} x_{3}^{3} y, x_{1} x_{2} y^{3}+x_{2} x_{3}^{3} y, x_{1} y^{3}+x_{3}^{3} y, x_{1} y^{2}+x_{3}^{3}, x_{1} x_{3}^{3} y-x_{2}^{3} y^{2}, x_{1}^{2} x_{3}^{2} y+\right.$ $\left.x_{2}^{3} x_{3}^{2}, x_{1}^{2} x_{3} y+x_{2}^{3} x_{3}, x_{1}^{2} y+x_{2}^{3}, x_{1} x_{3}^{3}-x_{2}^{3} y\right\}$. Finally, if we dehomogenise $G^{\prime}$, we obtain the set $H:=\left\{x_{2}^{3}+x_{3}^{6}, x_{1} x_{2}^{2}+x_{2}^{2} x_{3}^{3}, x_{1} x_{2}+x_{2} x_{3}^{3}, x_{1}+x_{3}^{3}, x_{1} x_{3}^{3}-x_{2}^{3}, x_{1}^{2} x_{3}^{2}+x_{2}^{3} x_{3}^{2}, x_{1}^{2} x_{3}+x_{2}^{3} x_{3}, x_{1}^{2}+x_{2}^{3}\right\} ;$ however this set is not a Janet Involutive Basis for $F$, as can be verified by checking that (with respect to $H$ ) the variable $x_{3}$ is nonmultiplicative for the polynomial $x_{2}^{3}+x_{3}^{6}$, and
the prolongation of the polynomial $x_{2}^{3}+x_{3}^{6}$ by the variable $x_{3}$ is involutively irreducible with respect to $H$.

The reason why $H$ is not an Involutive Basis for $J$ in the above example is that the Janet multiplicative variables for the set $G^{\prime}$ do not correspond to the Janet multiplicative variables for the set $H=d\left(G^{\prime}\right)$. This means that we cannot use the fact that all prolongations of elements of $G^{\prime}$ involutively reduce to zero using $G^{\prime}$ to deduce that all prolongations of elements of $H$ involutively reduce to zero using $H$. To do this, our involutive division must satisfy the following additional property, which ensures that the multiplicative variables of $G^{\prime}$ and $d\left(G^{\prime}\right)$ do correspond to each other.

Definition 4.5.3 Let $O$ be a fixed extendible monomial ordering. An involutive division $I$ is extendible with respect to $O$ if, given any set of polynomials $P$, we have

$$
\mathcal{M}_{I}(p, P) \backslash\{y\}=\mathcal{M}_{I}(d(p), d(P))
$$

for all $p \in P$, where $y$ is the homogenising variable.

In Section 2.5.2, we saw that of the monomial orderings defined in Section 1.2.1, only Lex, InvLex and DegRevLex are extendible. Let us now consider which involutive divisions are extendible with respect to these three monomial orderings.

Proposition 4.5.4 The Thomas involutive division is extendible with respect to Lex, InvLex and DegRevLex.

Proof: Let $P$ be an arbitrary set of polynomials over a polynomial ring containing variables $x_{1}, \ldots, x_{n}$ and a homogenising variable $y$. Because the Thomas involutive division decides whether a variable $x_{i}$ (for $1 \leqslant i \leqslant n$ ) is multiplicative for a polynomial $p \in P$ independent of the variable $y$, it is clear that $x_{i}$ is multiplicative for $p$ if and only if $x_{i}$ is multiplicative for $d(p)$ with respect to any of the monomial orderings Lex, InvLex and DegRevLex. It follows that $\mathcal{M}_{\mathcal{T}}(p, P) \backslash\{y\}=\mathcal{M}_{\mathcal{T}}(d(p), d(P))$ as required.

Proposition 4.5.5 The Pommaret involutive division is extendible with respect to Lex and DegRevLex.

Proof: Let $p$ be an arbitrary polynomial over a polynomial ring containing variables $x_{1}, \ldots, x_{n}$ and a homogenising variable $y$. Because we are using either the Lex or the

DegRevLex monomial orderings, the variable $y$ must be lexicographically less than any of the variables $x_{1}, \ldots, x_{n}$, and so we can state (without loss of generality) that $p$ belongs to the polynomial ring $R\left[x_{1}, \ldots, x_{n}, y\right]$. Let ( $e^{1}, e^{2}, \ldots, e^{n}, e^{n+1}$ ) be the multidegree corresponding to the monomial $\operatorname{LM}(p)$, and let $1 \leqslant i \leqslant n+1$ be the smallest integer such that $e^{i}>0$.

If $i=n+1$, then the variables $x_{1}, \ldots, x_{n}$ will all be multiplicative for $p$. But then $d(p)$ will be a constant, so that the variables $x_{1}, \ldots, x_{n}$ will also all be multiplicative for $d(p)$.

If $i \leqslant n$, then the variables $x_{1}, \ldots, x_{i}$ will all be multiplicative for $p$. But because $y$ is the smallest variable, it is clear that $i$ will also be the smallest integer such that $f^{i}>0$, where $\left(f^{1}, f^{2}, \ldots, f^{n}\right)$ is the multidegree corresponding to the monomial $\operatorname{LM}(d(p))$. It follows that the variables $x_{1}, \ldots, x_{i}$ will also all be multiplicative for $d(p)$, and so we can conclude that $\mathcal{M}_{\mathcal{P}}(p, P) \backslash\{y\}=\mathcal{M}_{\mathcal{P}}(d(p), d(P))$ as required.

Proposition 4.5.6 The Pommaret involutive division is not extendible with respect to InvLex.

Proof: Let $p:=y x_{2}+x_{1}^{2}$ be a polynomial over the polynomial ring $\mathbb{Q}\left[y, x_{1}, x_{2}\right]$, where $y$ is the homogenising variable (which must be greater than all other variables in order for InvLex to be extendible). As $\operatorname{LM}(p)=y x_{2}$ with respect to InvLex, it follows that $\mathcal{M}_{\mathcal{P}}(p)=\{y\}$. Further, as $\mathrm{LM}(d(p))=\mathrm{LM}\left(x_{2}+x_{1}^{2}\right)=x_{2}$ with respect to InvLex, it follows that $\mathcal{M}_{\mathcal{P}}(d(p))=\left\{x_{1}, x_{2}\right\}$. We can now deduce that the Pommaret involutive division is not extendible with respect to InvLex, as $\mathcal{M}_{\mathcal{P}}(p) \backslash\{y\} \neq \mathcal{M}_{\mathcal{P}}(d(p))$, or $\emptyset \neq\left\{x_{1}, x_{2}\right\}$.

Proposition 4.5.7 The Janet involutive division is extendible with respect to InvLex.

Proof: Let $P$ be an arbitrary set of polynomials over a polynomial ring containing variables $x_{1}, \ldots, x_{n}$ and a homogenising variable $y$. Because we are using the InvLex monomial ordering, the variable $y$ must be lexicographically greater than any of the variables $x_{1}, \ldots, x_{n}$, and so we can state (without loss of generality) that $p$ belongs to the polynomial ring $R\left[y, x_{1}, \ldots, x_{n}\right]$. But the Janet involutive division will then decide whether a variable $x_{i}$ (for $1 \leqslant i \leqslant n$ ) is multiplicative for a polynomial $p \in P$ independent of the variable $y$, so it is clear that $x_{i}$ is multiplicative for $p$ if and only if $x_{i}$ is multiplicative for $d(p)$, and so (with respect to InvLex) $\mathcal{M}_{\mathcal{J}}(p, P) \backslash\{y\}=\mathcal{M}_{\mathcal{J}}(d(p), d(P)$ ) as required.

Proposition 4.5.8 The Janet involutive division is not extendible with respect to Lex or DegRevLex.

Proof: Let $U:=\left\{x_{1}^{2} y, x_{1} y^{2}\right\}$ be a set of monomials over the polynomial ring $\mathbb{Q}\left[x_{1}, y\right]$, where $y$ is the homogenising variable (which must be less than $x_{1}$ in order for Lex and DegRevLex to be extendible). The Janet multiplicative variables for $U$ (with respect to Lex and DegRevLex) are shown in the table below.

| $u$ | $\mathcal{M}_{\mathcal{J}}(u, U)$ |
| :---: | :---: |
| $x_{1}^{2} y$ | $\left\{x_{1}\right\}$ |
| $x_{1} y^{2}$ | $\left\{x_{1}, y\right\}$ |

When we dehomogenise $U$ with respect to $y$, we obtain the set $d(U):=\left\{x_{1}^{2}, x_{1}\right\}$ with multiplicative variables as follows.

| $d(u)$ | $\mathcal{M}_{\mathcal{J}}(d(u), d(U))$ |
| :---: | :---: |
| $x_{1}^{2}$ | $\left\{x_{1}\right\}$ |
| $x_{1}$ | $\emptyset$ |

It is now clear that Janet is not an extendible involutive division with respect to Lex or DegRevLex, as $\mathcal{M}_{\mathcal{J}}\left(x_{1} y^{2}, U\right) \backslash\{y\} \neq \mathcal{M}_{\mathcal{J}}\left(x_{1}, d(U)\right)$, or $\left\{x_{1}\right\} \neq \emptyset$.

Proposition 4.5.9 Let $G^{\prime}$ be a set of polynomials over a polynomial ring containing variables $x_{1}, \ldots, x_{n}$ and a homogenising variable $y$. If (i) $G^{\prime}$ is an Involutive Basis with respect to some extendible monomial ordering $O$ and some involutive division I that is extendible with respect to $O$; and (ii) $d\left(G^{\prime}\right)$ is an autoreduced set, then $d\left(G^{\prime}\right)$ is an Involutive Basis with respect to $O$ and $I$.

Proof: By Definition 4.2.7, we can show that $d\left(G^{\prime}\right)$ is an Involutive Basis with respect to $O$ and $I$ by showing that any multiple $d\left(g^{\prime}\right) t$ of any polynomial $d\left(g^{\prime}\right) \in d\left(G^{\prime}\right)$ by any term $t$ involutively reduces to zero using $d\left(G^{\prime}\right)$. Because $G^{\prime}$ is an Involutive Basis with respect to $O$ and $I$, the polynomial $g^{\prime} t$ involutively reduces to zero using $G^{\prime}$ by the series of involutive reductions

$$
g^{\prime} t \underset{I g_{\alpha_{1}}^{\prime}}{\prime} h_{1} \xrightarrow[I g_{\alpha_{2}}^{\prime}]{ } h_{2} \underset{I g_{\alpha_{3}}^{\prime}}{\longrightarrow} \quad \underset{I g_{\alpha_{A}}^{\prime}}{\prime} 0,
$$

where $g_{\alpha_{i}}^{\prime} \in G^{\prime}$ for all $1 \leqslant i \leqslant A$.
Claim: The polynomial $d\left(g^{\prime}\right) t$ involutively reduces to zero using $d\left(G^{\prime}\right)$ by the series of involutive reductions

$$
d\left(g^{\prime}\right) t \xrightarrow[I d\left(g_{\alpha_{1}}^{\prime}\right)]{ } d\left(h_{1}\right) \xrightarrow[I d\left(g_{\alpha_{2}}^{\prime}\right)]{ } d\left(h_{2}\right) \xrightarrow[I d\left(g_{\alpha_{3}}^{\prime}\right)]{ } \cdots \xrightarrow[I d\left(g_{\alpha_{A}}^{\prime}\right)]{ } 0,
$$

where $d\left(g_{\alpha_{i}}^{\prime}\right) \in d\left(G^{\prime}\right)$ for all $1 \leqslant i \leqslant A$.
Proof of Claim: It is clear that if a polynomial $g_{j}^{\prime} \in G^{\prime}$ involutively reduces a polynomial $h$, then the polynomial $d\left(g_{j}^{\prime}\right) \in d\left(G^{\prime}\right)$ will always conventionally reduce the polynomial $d(h)$. Further, knowing that $I$ is extendible with respect to $O$, we can state that $d\left(g_{j}^{\prime}\right)$ will also always involutively reduce $d(h)$. The result now follows by noticing that $d\left(G^{\prime}\right)$ is autoreduced, so that $d\left(g_{j}^{\prime}\right)$ is the only possible involutive divisor of $d(h)$, and hence the above series of involutive reductions is the only possible way of involutively reducing the polynomial $d\left(g^{\prime}\right) t$.

Open Question 1 If the set $G$ returned by the procedure outlined in Definition 4.5.1 is not autoreduced, under what circumstances does autoreducing $G$ result in obtaining a set that is an Involutive Basis for the ideal generated by $F$ ?

Let us now consider two examples illustrating that the set $G$ returned by the procedure outlined in Definition 4.5.1 may or may not be autoreduced.

Example 4.5.10 Let $F:=\left\{2 x_{1} x_{2}+x_{1}^{2}+5, x_{2}^{2}+x_{1}+8\right\}$ be a basis generating an ideal $J$ over the polynomial ring $\mathbb{Q}\left[x_{1}, x_{2}\right]$, and let the monomial ordering be InvLex. Ordinarily, we can compute an Involutive Basis $G:=\left\{x_{2}^{2}+x_{1}+8,2 x_{1} x_{2}+x_{1}^{2}+5,10 x_{2}-x_{1}^{3}-4 x_{1}^{2}-\right.$ $\left.37 x_{1}, x_{1}^{4}+4 x_{1}^{3}+42 x_{1}^{2}+25\right\}$ for $F$ with respect to the Janet involutive division by using Algorithm 9.

Taking the homogeneous route (using Definition 4.5.1), we can homogenise $F$ to obtain the basis $F^{\prime}:=\left\{2 x_{1} x_{2}+x_{1}^{2}+5 y^{2}, x_{2}^{2}+y x_{1}+8 y^{2}\right\}$ over the polynomial ring $\mathbb{Q}\left[y, x_{1}, x_{2}\right]$, where $y$ is the homogenising variable (which must be greater than all other variables). Computing an Involutive Basis for the set $F^{\prime}$ with respect to the Janet involutive division using Algorithm 9 , we obtain the basis $G^{\prime}:=\left\{x_{2}^{2}+y x_{1}+8 y^{2}, 2 x_{1} x_{2}+x_{1}^{2}+5 y^{2}, 10 y^{2} x_{2}-x_{1}^{3}-\right.$ $\left.4 y x_{1}^{2}-37 y^{2} x_{1}, x_{1}^{4}+4 y x_{1}^{3}+42 y^{2} x_{1}^{2}+25 y^{4}\right\}$. When we dehomogenise this basis, we obtain the set $d\left(G^{\prime}\right):=\left\{x_{2}^{2}+x_{1}+8,2 x_{1} x_{2}+x_{1}^{2}+5,10 x_{2}-x_{1}^{3}-4 x_{1}^{2}-37 x_{1}, x_{1}^{4}+4 x_{1}^{3}+42 x_{1}^{2}+25\right\}$.

It is now clear that the set $d\left(G^{\prime}\right)$ is autoreduced (and hence $d\left(G^{\prime}\right)$ is an Involutive Basis for $J$ ) because $d\left(G^{\prime}\right)=G$.

Example 4.5.11 Let $F:=\left\{x_{2}^{2}+2 x_{1} x_{2}+5, x_{2}+x_{1}^{2}+8\right\}$ be a basis generating an ideal $J$ over the polynomial ring $\mathbb{Q}\left[x_{1}, x_{2}\right]$, and let the monomial ordering be InvLex. Ordinarily, we can compute an Involutive Basis $G:=\left\{x_{2}^{2}-2 x_{1}^{3}-16 x_{1}+5, x_{2}+x_{1}^{2}+8, x_{1}^{4}-2 x_{1}^{3}+\right.$ $\left.16 x_{1}^{2}-16 x_{1}+69\right\}$ for $F$ with respect to the Janet involutive division by using Algorithm 9 .

Taking the homogeneous route (using Definition 4.5.1), we can homogenise $F$ to obtain the basis $F^{\prime}:=\left\{x_{2}^{2}+2 x_{1} x_{2}+5 y^{2}, y x_{2}+x_{1}^{2}+8 y^{2}\right\}$ over the polynomial ring $\mathbb{Q}\left[y, x_{1}, x_{2}\right]$, where $y$ is the homogenising variable (which must be greater than all other variables). Computing an Involutive Basis for the set $F^{\prime}$ with respect to the Janet involutive division using Algorithm 9, we obtain the basis $G^{\prime}:=\left\{x_{2}^{2}+2 x_{1} x_{2}+5 y^{2}, x_{1}^{2} x_{2}+2 x_{1}^{3}-8 y x_{1}^{2}+\right.$ $\left.16 y^{2} x_{1}-69 y^{3}, y x_{1} x_{2}+x_{1}^{3}+8 y^{2} x_{1}, y x_{2}+x_{1}^{2}+8 y^{2}, x_{1}^{4}-2 y x_{1}^{3}+16 y^{2} x_{1}^{2}-16 y^{3} x_{1}+69 y^{4}\right\}$. When we dehomogenise this basis, we obtain the set $d\left(G^{\prime}\right):=\left\{x_{2}^{2}+2 x_{1} x_{2}+5, x_{1}^{2} x_{2}+\right.$ $\left.2 x_{1}^{3}-8 x_{1}^{2}+16 x_{1}-69, x_{1} x_{2}+x_{1}^{3}+8 x_{1}, x_{2}+x_{1}^{2}+8, x_{1}^{4}-2 x_{1}^{3}+16 x_{1}^{2}-16 x_{1}+69\right\}$. This time however, because the set $d\left(G^{\prime}\right)$ is not autoreduced (the polynomial $x_{1} x_{2}+x_{1}^{3}+8 x_{1} \in d\left(G^{\prime}\right)$ can involutively reduce the second term of the polynomial $x_{2}^{2}+2 x_{1} x_{2}+5 \in d\left(G^{\prime}\right)$ ), we cannot deduce that $d\left(G^{\prime}\right)$ is an Involutive Basis for $J$.

Remark 4.5.12 Although the set $G$ returned by the procedure outlined in Definition 4.5.1 may not always be an Involutive Basis for the ideal generated by $F$, because the set $G^{\prime}$ will always be an Involutive Basis (and hence also a Gröbner Basis), we can state that $G$ will always be a Gröbner Basis for the ideal generated by $F$ (cf. Definition 2.5.7).

### 4.5.3 Logged Involutive Bases

Just as a Logged Gröbner Basis expresses each member of the Gröbner Basis in terms of members of the original basis from which the Gröbner Basis was computed, a Logged Involutive Basis expresses each member of the Involutive Basis in terms of members of the original basis from which the Involutive Basis was computed.

Definition 4.5.13 Let $G=\left\{g_{1}, \ldots, g_{p}\right\}$ be an Involutive Basis computed from an initial basis $F=\left\{f_{1}, \ldots, f_{m}\right\}$. We say that $G$ is a Logged Involutive Basis if, for each $g_{i} \in G$,
we have an explicit expression of the form

$$
g_{i}=\sum_{\alpha=1}^{\beta} t_{\alpha} f_{k_{\alpha}},
$$

where the $t_{\alpha}$ are terms and $f_{k_{\alpha}} \in F$ for all $1 \leqslant \alpha \leqslant \beta$.
Proposition 4.5.14 Given a finite basis $F=\left\{f_{1}, \ldots, f_{m}\right\}$, it is always possible to compute a Logged Involutive Basis for F.

Proof: Let $G=\left\{g_{1}, \ldots, g_{p}\right\}$ be an Involutive Basis computed from the initial basis $F=\left\{f_{1}, \ldots, f_{m}\right\}$ using Algorithm 9 (where $f_{i} \in R\left[x_{1}, \ldots, x_{n}\right]$ for all $f_{i} \in F$ ). If an arbitrary $g_{i} \in G$ is not a member of the original basis $F$, then either $g_{i}$ is an involutively reduced prolongation, or $g_{i}$ is obtained through the process of autoreduction. In the former case, we can express $g_{i}$ in terms of members of $F$ by substitution because

$$
g_{i}=h x_{j}-\sum_{\alpha=1}^{\beta} t_{\alpha} h_{k_{\alpha}}
$$

for a variable $x_{j}$; terms $t_{\alpha}$ and polynomials $h$ and $h_{k_{\alpha}}$ which we already know how to express in terms of members of $F$. In the latter case,

$$
g_{i}=h-\sum_{\alpha=1}^{\beta} t_{\alpha} h_{k_{\alpha}}
$$

for terms $t_{\alpha}$ and polynomials $h$ and $h_{k_{\alpha}}$ which we already know how to express in terms of members of $F$, so it follows that we can again express $g_{i}$ in terms of members of $F$.

## Chapter 5

## Noncommutative Involutive Bases

In the previous chapter, we introduced the theory of commutative Involutive Bases and saw that such bases are always commutative Gröbner Bases with extra structure. In this chapter, we will follow a similar path, in that we will define an algorithm to compute a noncommutative Involutive Basis that will serve as an alternative method of obtaining a noncommutative Gröbner Basis, and the noncommutative Gröbner Bases we will obtain will also have some extra structure.

As illustrated by the diagram shown below, the theory of noncommutative Involutive Bases will draw upon all the theory that has come before in this thesis, and as a consequence will inherit many of the restrictions imposed by this theory. For example, our noncommutative Involutive Basis algorithm will not be guaranteed to terminate precisely because we are working in a noncommutative setting, and noncommutative involutive divisions will have properties that will influence the correctness and termination of the algorithm.


### 5.1 Noncommutative Involutive Reduction

Recall that in a commutative polynomial ring, a monomial $u_{2}$ is an involutive divisor of a monomial $u_{1}$ if $u_{1}=u_{2} u_{3}$ for some monomial $u_{3}$ and all variables in $u_{3}$ are multiplicative for $u_{2}$. In other words, we are able to form $u_{1}$ from $u_{2}$ by multiplying $u_{2}$ with multiplicative variables.

In a noncommutative polynomial ring, an involutive division will again induce a restricted form of division. However, because left and right multiplication are separate processes in noncommutative polynomial rings, we will require the notion of left and right multiplicative variables in order to determine whether a conventional divisor is an involutive divisor, so that (intuitively) a monomial $u_{2}$ will involutively divide a monomial $u_{1}$ if we are able to form $u_{1}$ from $u_{2}$ by multiplying $u_{2}$ on the left with left multiplicative variables and on the right by right multiplicative variables.

More formally, let $u_{1}$ and $u_{2}$ be two monomials over a noncommutative polynomial ring, and assume that $u_{1}$ is a conventional divisor of $u_{2}$, so that $u_{1}=u_{3} u_{2} u_{4}$ for some monomials $u_{3}$ and $u_{4}$. Assume that an arbitrary noncommutative involutive division $I$ partitions the variables in the polynomial ring into sets of left multiplicative and left nonmultiplicative variables for $u_{2}$, and also partitions the variables in the polynomial ring into sets of right multiplicative and right nonmultiplicative variables for $u_{2}$. Let us now define two methods of deciding whether $u_{2}$ is an involutive divisor of $u_{1}$ (written $\left.u_{2}\right|_{I} u_{1}$ ), the first of which will depend only on the first variable we multiply $u_{2}$ with on the left and on the right in order to form $u_{1}$, and the second of which will depend on all the variables we multiply $u_{2}$ with in order to form $u_{1}$.

Definition 5.1.1 Let $u_{1}=u_{3} u_{2} u_{4}$, and let $I$ be defined as in the previous paragraph.

- (Thin Divisor) $\left.u_{2}\right|_{I} u_{1}$ if the variable $\operatorname{Suffix}\left(u_{3}, 1\right)$ (if it exists) is in the set of left multiplicative variables for $u_{2}$, and the variable $\operatorname{Prefix}\left(u_{4}, 1\right)$ (again if it exists) is in the set of right multiplicative variables for $u_{2}$.
- (Thick Divisor) $\left.u_{2}\right|_{I} u_{1}$ if all the variables in $u_{3}$ are in the set of left multiplicative variables for $u_{2}$, and all the variables in $u_{4}$ are in the set of right multiplicative variables for $u_{2}$.

Remark 5.1.2 We introduce two methods for determining whether a conventional divisor is an involutive divisor because each of the methods has its own advantages and
disadvantages. From a theoretical standpoint, using thin divisors enables us to follow the path laid down in Chapter 4, in that we are able to show that a Locally Involutive Basis is an Involutive Basis by proving that the involutive division used is continuous, something that we cannot do if thick divisors are being used. On the other hand, once we have obtained our Locally Involutive Basis, involutive reduction with respect to thick divisors is more efficient than it is with respect to thin divisors, as less work is required in order to determine whether a monomial is involutively divisible by a set of monomials. For these reasons, we will use thin divisors when presenting the theory in this chapter (hence the following definition), and will only use thick divisors when, by doing so, we are able to gain some advantage.

Remark 5.1.3 Unless otherwise stated, from now on we will use thin divisors to determine whether a conventional divisor is an involutive divisor.

Example 5.1.4 Let $u_{1}:=x y z^{2} x ; u_{1}^{\prime}:=y z^{2} y$ and $u_{2}:=z^{2}$ be three monomials over the polynomial ring $\mathcal{R}=\mathbb{Q}\langle x, y, z\rangle$, and let an involutive division $I$ partition the variables in $\mathcal{R}$ into the following sets of variables for the monomial $u_{2}$ : left multiplicative $=\{x, y\}$; left nonmultiplicative $=\{z\}$; right multiplicative $=\{x, z\}$; right nonmultiplicative $=$ $\{y\}$. It is true that $u_{2}$ conventionally divides both monomials $u_{1}$ and $u_{1}^{\prime}$, but $u_{2}$ only involutively divides monomial $u_{1}$ as, defining $u_{3}:=x y ; u_{4}:=x ; u_{3}^{\prime}=y$ and $u_{4}^{\prime}=y$ (so that $u_{1}=u_{3} u_{2} u_{4}$ and $\left.u_{1}^{\prime}=u_{3}^{\prime} u_{2} u_{4}^{\prime}\right)$, we observe that the variable $\operatorname{Suffix}\left(u_{3}, 1\right)=y$ is in the set of left multiplicative variables for $u_{2}$; the variable $\operatorname{Prefix}\left(u_{4}, 1\right)=x$ is in the set of right multiplicative variables for $u_{2}$; but the variable $\operatorname{Prefix}\left(u_{4}^{\prime}, 1\right)=y$ is not in the set of right multiplicative variables for $u_{2}$.

Let us now formally define what is meant by a (noncommutative) involutive division.
Definition 5.1.5 Let $M$ denote the set of all monomials in a noncommutative polynomial ring $\mathcal{R}=R\left\langle x_{1}, \ldots, x_{n}\right\rangle$, and let $U \subset M$. The involutive cone $\mathcal{C}_{I}(u, U)$ of any monomial $u \in U$ with respect to some involutive division $I$ is defined as follows.

$$
\mathcal{C}_{I}(u, U)=\left\{v_{1} u v_{2} \text { such that } v_{1}, v_{2} \in M \text { and }\left.u\right|_{I} v_{1} u v_{2}\right\} .
$$

Definition 5.1.6 Let $M$ denote the set of all monomials in a noncommutative polynomial ring $\mathcal{R}=R\left\langle x_{1}, \ldots, x_{n}\right\rangle$. A strong involutive division $I$ is defined on $M$ if, given any finite set of monomials $U \subset M$, we can assign a set of left multiplicative variables $\mathcal{M}_{I}^{L}(u, U) \subseteq$
$\left\{x_{1}, \ldots, x_{n}\right\}$ and a set of right multiplicative variables $\mathcal{M}_{I}^{R}(u, U) \subseteq\left\{x_{1}, \ldots, x_{n}\right\}$ to any monomial $u \in U$ such that the following three conditions are satisfied.

- If there exist two elements $u_{1}, u_{2} \in U$ such that $\mathcal{C}_{I}\left(u_{1}, U\right) \cap \mathcal{C}_{I}\left(u_{2}, U\right) \neq \emptyset$, then either $\mathcal{C}_{I}\left(u_{1}, U\right) \subset \mathcal{C}_{I}\left(u_{2}, U\right)$ or $\mathcal{C}_{I}\left(u_{2}, U\right) \subset \mathcal{C}_{I}\left(u_{1}, U\right)$.
- Any monomial $v \in \mathcal{C}_{I}(u, U)$ is involutively divisible by $u$ in one way only, so that if $u$ appears as a subword of $v$ in more than one way, then only one of these ways allows us to deduce that $u$ is an involutive divisor of $v$.
- If $V \subset U$, then $\mathcal{M}_{I}^{L}(v, U) \subseteq \mathcal{M}_{I}^{L}(v, V)$ and $\mathcal{M}_{I}^{R}(v, U) \subseteq \mathcal{M}_{I}^{R}(v, V)$ for all $v \in V$.

If any of the above conditions are not satisfied, the involutive division is called a weak involutive division.

Remark 5.1.7 We shall refer to the three conditions of Definition 5.1.6 as (respectively) the Disjoint Cones condition, the Unique Divisor condition and the Subset condition.

Definition 5.1.8 Given an involutive division $I$, the involutive span $\mathcal{C}_{I}(U)$ of a set of noncommutative monomials $U$ with respect to $I$ is given by the expression

$$
\mathcal{C}_{I}(U)=\bigcup_{u \in U} \mathcal{C}_{I}(u, U)
$$

Remark 5.1.9 The (conventional) span of a set of noncommutative monomials $U$ is given by the expression

$$
\mathcal{C}(U)=\bigcup_{u \in U} \mathcal{C}(u, U)
$$

where $\mathcal{C}(u, U)=\left\{v_{1} u v_{2}\right.$ such that $v_{1}, v_{2}$ are monomials $\}$ is the (conventional) cone of a monomial $u \in U$.

Definition 5.1.10 If an involutive division $I$ determines the left and right multiplicative variables for a monomial $u \in U$ independent of the set $U$, then $I$ is a global division. Otherwise, $I$ is a local division.

Remark 5.1.11 The multiplicative variables for a set of polynomials $P$ (whose terms are ordered by a monomial ordering $O$ ) are determined by the multiplicative variables for the set of leading monomials $\mathrm{LM}(P)$.

In Algorithm 10, we specify how to involutively divide a polynomial $p$ with respect to a set of polynomials $P$ using thin divisors. Note that this algorithm combines the modifications made to Algorithm 1 in Algorithms 2 and 7.

```
Algorithm 10 The Noncommutative Involutive Division Algorithm
Input: A nonzero polynomial \(p\) and a set of nonzero polynomials \(P=\left\{p_{1}, \ldots, p_{m}\right\}\)
    over a polynomial ring \(R\left\langle x_{1}, \ldots x_{n}\right\rangle\); an admissible monomial ordering O ; an involutive
    division \(I\).
Output: \(\operatorname{Rem}_{I}(p, P):=r\), the involutive remainder of \(p\) with respect to \(P\).
    \(r=0\);
    while \((p \neq 0)\) do
        \(u=\operatorname{LM}(p) ; c=\mathrm{LC}(p) ; j=1\); found \(=\) false;
        while \((j \leqslant m)\) and (found \(==\) false) do
            if \(\left(\left.\operatorname{LM}\left(p_{j}\right)\right|_{I} u\right)\) then
            found \(=\) true;
            choose \(u_{\ell}\) and \(u_{r}\) such that \(u=u_{\ell} \mathrm{LM}\left(p_{j}\right) u_{r}\), the variable \(\operatorname{Suffix}\left(u_{\ell}, 1\right)\) (if it exists)
            is left multiplicative for \(p_{j}\), and the variable \(\operatorname{Prefix}\left(u_{r}, 1\right)\) (again if it exists) is
            right multiplicative for \(p_{j}\);
            \(p=p-\left(c \mathrm{LC}\left(p_{j}\right)^{-1}\right) u_{\ell} p_{j} u_{r} ;\)
            else
            \(j=j+1 ;\)
            end if
        end while
        if (found \(==\) false) then
            \(r=r+\mathrm{LT}(p) ; p=p-\mathrm{LT}(p) ;\)
        end if
    end while
    return \(r\);
```

Remark 5.1.12 Continuing the convention from Algorithm 2, we will always choose the $u_{\ell}$ with the smallest degree in the line 'choose $u_{\ell}$ and $u_{r}$ such that. ..' in Algorithm 10.

Example 5.1.13 Let $P:=\left\{x^{2}-2 y, x y-x, y^{3}+3\right\}$ be a set of polynomials over the polynomial ring $\mathbb{Q}\langle x, y\rangle$ ordered with respect to the DegLex monomial ordering, and assume that an involutive division $I$ assigns multiplicative variables to $P$ as follows.

| $p$ | $\mathcal{M}_{I}^{L}(\mathrm{LM}(p), \mathrm{LM}(P))$ | $\mathcal{M}_{I}^{R}(\mathrm{LM}(p), \mathrm{LM}(P))$ |
| :---: | :---: | :---: |
| $x^{2}-2 y$ | $\{x, y\}$ | $\{x\}$ |
| $x y-x$ | $\{y\}$ | $\{x, y\}$ |
| $y^{3}+3$ | $\{x\}$ | $\emptyset$ |

Here is a dry run for Algorithm 10 when we involutively divide the polynomial $p:=$ $2 x^{2} y^{3}+y x y$ with respect to $P$ to obtain the polynomial $y x-12 y$, where A; B; C and D refer to the tests $(p \neq 0)$ ?; $\left((j \leqslant 3)\right.$ and (found $==$ false))?; $\left(\left.\operatorname{LM}\left(p_{j}\right)\right|_{I} u\right)$ ? and (found $==$ false)? respectively.


### 5.2 Prolongations and Autoreduction

Just as in the commutative case, we will compute a (noncommutative) Locally Involutive Basis by using prolongations and autoreduction, but here we have to distinguish between left prolongations and right prolongations.

Definition 5.2.1 Given a set of polynomials $P$, a left prolongation of a polynomial $p \in$ $P$ is a product $x_{i} p$, where $x_{i} \notin \mathcal{M}_{I}^{L}(\mathrm{LM}(p), \mathrm{LM}(P))$ with respect to some involutive division $I$; and a right prolongation of a polynomial $p \in P$ is a product $p x_{i}$, where $x_{i} \notin \mathcal{M}_{I}^{R}(\mathrm{LM}(p), \mathrm{LM}(P))$ with respect to some involutive division $I$.

Definition 5.2.2 A set of polynomials $P$ is said to be autoreduced if no polynomial $p \in P$ exists such that $p$ contains a term which is involutively divisible (with respect to $P$ ) by some polynomial $p^{\prime} \in P \backslash\{p\}$.

```
Algorithm 11 The Noncommutative Autoreduction Algorithm
Input: A set of polynomials \(P=\left\{p_{1}, p_{2}, \ldots, p_{\alpha}\right\}\); an involutive division \(I\).
Output: An autoreduced set of polynomials \(Q=\left\{q_{1}, q_{2}, \ldots, q_{\beta}\right\}\).
    while \(\left(\exists p_{i} \in P\right.\) such that \(\left.\operatorname{Rem}_{I}\left(p_{i}, P, P \backslash\left\{p_{i}\right\}\right) \neq p_{i}\right)\) do
        \(p_{i}^{\prime}=\operatorname{Rem}_{I}\left(p_{i}, P, P \backslash\left\{p_{i}\right\}\right)\);
        \(P=P \backslash\left\{p_{i}\right\} ;\)
        if \(\left(p_{i}^{\prime} \neq 0\right)\) then
                \(P=P \cup\left\{p_{i}^{\prime}\right\} ;\)
        end if
    end while
    \(Q=P\);
    return \(Q\);
```

Remark 5.2.3 With respect to a strong involutive division, the involutive cones of an autoreduced set of polynomials are always disjoint.

Remark 5.2.4 The notation $\operatorname{Rem}_{I}\left(p_{i}, P, P \backslash\left\{p_{i}\right\}\right)$ used in Algorithm 11 has the same meaning as in Definition 4.2.2.

Proposition 5.2.5 Let $P$ be a set of polynomials over a noncommutative polynomial ring $\mathcal{R}=R\left\langle x_{1}, \ldots, x_{n}\right\rangle$, and let $f$ and $g$ be two polynomials also in $\mathcal{R}$. If $P$ is autoreduced with respect to a strong involutive division $I$, then $\operatorname{Rem}_{I}(f, P)+\operatorname{Rem}_{I}(g, P)=\operatorname{Rem}_{I}(f+g, P)$.

Proof: Let $f^{\prime}:=\operatorname{Rem}_{I}(f, P) ; g^{\prime}:=\operatorname{Rem}_{I}(g, P)$ and $h^{\prime}:=\operatorname{Rem}_{I}(h, P)$, where $h:=f+g$. Then, by the respective involutive reductions, we have expressions

$$
\begin{aligned}
& f^{\prime}=f-\sum_{a=1}^{A} u_{a} p_{\alpha_{a}} v_{a} \\
& g^{\prime}=g-\sum_{b=1}^{B} u_{b} p_{\beta_{b}} v_{b}
\end{aligned}
$$

and

$$
h^{\prime}=h-\sum_{c=1}^{C} u_{c} p_{\gamma_{c}} v_{c},
$$

where $p_{\alpha_{a}}, p_{\beta_{b}}, p_{\gamma_{c}} \in P$ and $u_{a}, v_{a}, u_{b}, v_{b}, u_{c}, v_{c}$ are terms such that each $p_{\alpha_{a}}, p_{\beta_{b}}$ and $p_{\gamma_{c}}$ involutively divides each $u_{a} p_{\alpha_{a}} v_{a}, u_{b} p_{\beta_{b}} v_{b}$ and $u_{c} p_{\gamma_{c}} v_{c}$ respectively.

Consider the polynomial $h^{\prime}-f^{\prime}-g^{\prime}$. By the above expressions, we can deduce ${ }^{1}$ that

$$
h^{\prime}-f^{\prime}-g^{\prime}=\sum_{a=1}^{A} u_{a} p_{\alpha_{a}} v_{a}+\sum_{b=1}^{B} u_{b} p_{\beta_{b}} v_{b}-\sum_{c=1}^{C} u_{c} p_{\gamma_{c}} v_{c}=: \sum_{d=1}^{D} u_{d} p_{\delta_{d}} v_{d} .
$$

Claim: $\operatorname{Rem}_{I}\left(h^{\prime}-f^{\prime}-g^{\prime}, P\right)=0$.
Proof of Claim: Let $t$ denote the leading term of the polynomial $\sum_{d=1}^{D} u_{d} p_{\delta_{d}} v_{d}$. Then $\mathrm{LM}(t)=\mathrm{LM}\left(u_{k} p_{\delta_{k}} v_{k}\right)$ for some $1 \leqslant k \leqslant D$ since, if not, there exists a monomial

$$
\operatorname{LM}\left(u_{k^{\prime}} p_{\delta_{k^{\prime}}} v_{k^{\prime}}\right)=\operatorname{LM}\left(u_{k^{\prime \prime}} p_{\delta_{k^{\prime \prime}}} v_{k^{\prime \prime}}\right)=: w
$$

for some $1 \leqslant k^{\prime}, k^{\prime \prime} \leqslant D$ (with $p_{\delta_{k^{\prime}}} \neq p_{\delta_{k^{\prime \prime}}}$ ) such that $w$ is involutively divisible by the two polynomials $p_{\delta_{k^{\prime}}}$ and $p_{\delta_{k^{\prime \prime}}}$, contradicting Definition 5.1 .6 (recall that $I$ is strong and $P$ is autoreduced, so that the involutive cones of $P$ are disjoint). It follows that we can use $p_{\delta_{k}}$ to eliminate $t$ by involutively reducing $h^{\prime}-f^{\prime}-g^{\prime}$ as shown below.

$$
\begin{equation*}
\sum_{d=1}^{D} u_{d} p_{\delta_{d}} v_{d} \underset{I p_{\delta_{k}}}{ } \sum_{d=1}^{k-1} u_{d} p_{\delta_{d}} v_{d}+\sum_{d=k+1}^{D} u_{d} p_{\delta_{d}} v_{d} \tag{5.1}
\end{equation*}
$$

By induction, we can apply a chain of involutive reductions to the right hand side of Equation (5.1) to obtain a zero remainder, so that $\operatorname{Rem}_{I}\left(h^{\prime}-f^{\prime}-g^{\prime}, P\right)=0$.

[^9]To complete the proof, we note that since $f^{\prime}, g^{\prime}$ and $h^{\prime}$ are all involutively irreducible, we must have $\operatorname{Rem}_{I}\left(h^{\prime}-f^{\prime}-g^{\prime}, P\right)=h^{\prime}-f^{\prime}-g^{\prime}$. It therefore follows that $h^{\prime}-f^{\prime}-g^{\prime}=0$, or $h^{\prime}=f^{\prime}+g^{\prime}$ as required.

Definition 5.2.6 Given an involutive division $I$ and an admissible monomial ordering $O$, an autoreduced set of noncommutative polynomials $P$ is a Locally Involutive Basis with respect to $I$ and $O$ if any (left or right) prolongation of any polynomial $p_{i} \in P$ involutively reduces to zero using $P$.

Definition 5.2.7 Given an involutive division $I$ and an admissible monomial ordering $O$, an autoreduced set of noncommutative polynomials $P$ is an Involutive Basis with respect to $I$ and $O$ if any multiple $u p_{i} v$ of any polynomial $p_{i} \in P$ by any terms $u$ and $v$ involutively reduces to zero using $P$.

### 5.3 The Noncommutative Involutive Basis Algorithm

To compute a (noncommutative) Locally Involutive Basis, we use Algorithm 12, an algorithm that is virtually identical to Algorithm 9, apart from the fact that at the beginning of the first while loop, the set $S$ is constructed in different ways.

### 5.4 Continuity and Conclusivity

In the commutative case, when we construct a Locally Involutive Basis using Algorithm 9, we know that the algorithm will always return a commutative Gröbner Basis as long as we use an admissible monomial ordering and the chosen involutive division possesses certain properties. In summary,
(a) Any Locally Involutive Basis returned by Algorithm 9 is an Involutive Basis if the involutive division used is continuous (Proposition 4.3.3);
(b) Algorithm 9 always terminates if (in addition) the involutive division used is constructive, Noetherian and stable (Proposition 4.4.8);
(c) Every Involutive Basis is a Gröbner Basis (Theorem 4.4.9).

In the noncommutative case, we cannot hope to produce a carbon copy of the above results because a finitely generated basis may have an infinite Gröbner Basis, leading to

```
Algorithm 12 The Noncommutative Involutive Basis Algorithm
Input: A Basis \(F=\left\{f_{1}, f_{2}, \ldots, f_{m}\right\}\) for an ideal \(J\) over a noncommutative polynomial
    ring \(R\left\langle x_{1}, \ldots x_{n}\right\rangle\); an admissible monomial ordering \(O\); an involutive division \(I\).
Output: A Locally Involutive Basis \(G=\left\{g_{1}, g_{2}, \ldots, g_{p}\right\}\) for \(J\) (in the case of termina-
    tion).
    \(G=\emptyset ;\)
    \(F=\) Autoreduce \((F)\);
    while ( \(G==\emptyset\) ) do
        \(S=\left\{x_{i} f \mid f \in F, x_{i} \notin \mathcal{M}_{I}^{L}(f, F)\right\} \cup\left\{f x_{i} \mid f \in F, x_{i} \notin \mathcal{M}_{I}^{R}(f, F)\right\} ;\)
        \(s^{\prime}=0\);
        while \((S \neq \emptyset)\) and ( \(s^{\prime}==0\) ) do
            Let \(s\) be a polynomial in \(S\) whose lead monomial is minimal with respect to \(O\);
            \(S=S \backslash\{s\} ;\)
            \(s^{\prime}=\operatorname{Rem}_{I}(s, F) ;\)
        end while
        if \(\left(s^{\prime} \neq 0\right)\) then
            \(F=\) Autoreduce \(\left(F \cup\left\{s^{\prime}\right\}\right)\);
        else
            \(G=F ;\)
        end if
    end while
    return \(G\);
```

the conclusion that Algorithm 12 does not always terminate. The best we can therefore hope for is if an ideal generated by a set of polynomials $F$ possesses a finite Gröbner Basis with respect to some admissible monomial ordering $O$, then $F$ also possesses a finite Involutive Basis with respect to $O$ and some involutive division $I$. We shall call any involutive division that possesses this property conclusive.

Definition 5.4.1 Let $F$ be an arbitrary basis generating an ideal over a noncommutative polynomial ring, and let $O$ be an arbitrary admissible monomial ordering. An involutive division $I$ is conclusive if Algorithm 12 terminates with $F, I$ and $O$ as input whenever Algorithm 5 terminates with $F$ and $O$ as input.

Of course it is easy enough to define the above property, but much harder to prove that a particular involutive division is conclusive. In fact, no involutive division defined in this thesis will be shown to be conclusive, and the existence of such divisions will be left as an open question.

### 5.4.1 Properties for Strong Involutive Divisions

Here is a summary of facts that can be deduced when using a strong involutive division.
(a) Any Locally Involutive Basis returned by Algorithm 12 is an Involutive Basis if the involutive division used is strong and continuous (Proposition 5.4.3);
(b) Algorithm 12 always terminates whenever Algorithm 5 terminates if (in addition) the involutive division used is conclusive;
(c) Every Involutive Basis with respect to a strong involutive division is a Gröbner Basis (Theorem 5.4.4).

Let us now prove the assertions made in parts (a) and (c) of the above list, beginning by defining what is meant by a continuous involutive division in the noncommutative case.

Definition 5.4.2 Let $I$ be a fixed involutive division; let $w$ be a fixed monomial; let $U$ be any set of monomials; and consider any sequence $\left(u_{1}, u_{2}, \ldots, u_{k}\right)$ of monomials from $U$ ( $u_{i} \in U$ for all $1 \leqslant i \leqslant k$ ), each of which is a conventional divisor of $w$ (so that $w=\ell_{i} u_{i} r_{i}$ for all $1 \leqslant i \leqslant k$, where the $\ell_{i}$ and the $r_{i}$ are monomials). For all $1 \leqslant i<k$, suppose that the monomial $u_{i+1}$ satisfies exactly one of the following conditions.
(a) $u_{i+1}$ involutively divides a left prolongation of $u_{i}$, so that $\operatorname{deg}\left(\ell_{i}\right) \geqslant 1$; $\operatorname{Suffix}\left(\ell_{i}, 1\right) \notin$ $\mathcal{M}_{I}^{L}\left(u_{i}, U\right)$; and $\left.u_{i+1}\right|_{I}\left(\operatorname{Suffix}\left(\ell_{i}, 1\right)\right) u_{i}$.
(b) $u_{i+1}$ involutively divides a right prolongation of $u_{i}$, so that $\operatorname{deg}\left(r_{i}\right) \geqslant 1$; $\operatorname{Prefix}\left(r_{i}, 1\right) \notin$ $\mathcal{M}_{I}^{R}\left(u_{i}, U\right)$; and $\left.u_{i+1}\right|_{I} u_{i}\left(\operatorname{Prefix}\left(r_{i}, 1\right)\right)$.

Then $I$ is continuous at $w$ if all the pairs $\left(\ell_{i}, r_{i}\right)$ are distinct $\left(\left(\ell_{i}, r_{i}\right) \neq\left(\ell_{j}, r_{j}\right)\right.$ for all $\left.i \neq j\right)$; $I$ is a continuous involutive division if $I$ is continuous for all possible $w$.

Proposition 5.4.3 If an involutive division $I$ is strong and continuous, and a given set of polynomials $P$ is a Locally Involutive Basis with respect to $I$ and some admissible monomial ordering $O$, then $P$ is an Involutive Basis with respect to $I$ and $O$.

Proof: Let $I$ be a strong and continuous involutive division; let $O$ be an admissible monomial ordering; and let $P$ be a Locally Involutive Basis with respect to $I$ and $O$. Given any polynomial $p \in P$ and any terms $u$ and $v$, in order to show that $P$ is an Involutive Basis with respect to $I$ and $O$, we must show that upv $\vec{I}_{P} 0$.

If $\left.p\right|_{I} u p v$ we are done, as we can use $p$ to involutively reduce $u p v$ to obtain a zero remainder. Otherwise, either $\exists y_{1} \notin \mathcal{M}_{I}^{L}(\operatorname{LM}(p), \operatorname{LM}(P))$ such that $y_{1}=\operatorname{Suffix}(u, 1)$, or $\exists y_{1} \notin \mathcal{M}_{I}^{R}(\operatorname{LM}(p), \mathrm{LM}(P))$ such that $y_{1}=\operatorname{Prefix}(v, 1)$. Without loss of generality, assume that the first case applies. By Local Involutivity, the prolongation $y_{1} p$ involutively reduces to zero using $P$. Assuming that the first step of this involutive reduction involves the polynomial $p_{1} \in P$, we can write

$$
\begin{equation*}
y_{1} p=u_{1} p_{1} v_{1}+\sum_{a=1}^{A} u_{\alpha_{a}} p_{\alpha_{a}} v_{\alpha_{a}}, \tag{5.2}
\end{equation*}
$$

where $p_{\alpha_{a}} \in P$ and $u_{1}, v_{1}, u_{\alpha_{a}}, v_{\alpha_{a}}$ are terms such that $p_{1}$ and each $p_{\alpha_{a}}$ involutively divide $u_{1} p_{1} v_{1}$ and each $u_{\alpha_{a}} p_{\alpha_{a}} v_{\alpha_{a}}$ respectively. Multiplying both sides of Equation (5.2) on the left by $u^{\prime}:=\operatorname{Prefix}(u, \operatorname{deg}(u)-1)$ and on the right by $v$, we obtain the equation

$$
\begin{equation*}
u p v=u^{\prime} u_{1} p_{1} v_{1} v+\sum_{a=1}^{A} u^{\prime} u_{\alpha_{a}} p_{\alpha_{a}} v_{\alpha_{a}} v . \tag{5.3}
\end{equation*}
$$

If $\left.p_{1}\right|_{I} u p v$, it is clear that we can use $p_{1}$ to involutively reduce the polynomial upv to obtain the polynomial $\sum_{a=1}^{A} u^{\prime} u_{\alpha_{a}} p_{\alpha_{a}} v_{\alpha_{a}} v$. By Proposition 5.2.5, we can then continue
to involutively reduce $u p v$ by repeating this proof on each polynomial $u^{\prime} u_{\alpha_{a}} p_{\alpha_{a}} v_{\alpha_{a}} v$ individually (where $1 \leqslant a \leqslant A$ ), noting that this process will terminate because of the admissibility of $O$ (we have $\operatorname{LM}\left(u^{\prime} u_{\alpha_{a}} p_{\alpha_{a}} v_{\alpha_{a}} v\right)<\operatorname{LM}(u p v)$ for all $1 \leqslant a \leqslant A$ ).

Otherwise, if $p_{1}$ does not involutively divide upv, either $\exists y_{2} \notin \mathcal{M}_{I}^{L}\left(\operatorname{LM}\left(p_{1}\right), \operatorname{LM}(P)\right)$ such that $y_{2}=\operatorname{Suffix}\left(u^{\prime} u_{1}, 1\right)$, or $\exists y_{2} \notin \mathcal{M}_{I}^{R}\left(\operatorname{LM}\left(p_{1}\right), \operatorname{LM}(P)\right)$ such that $y_{2}=\operatorname{Prefix}\left(v_{1} v, 1\right)$. This time (again without loss of generality), assume that the second case applies. By Local Involutivity, the prolongation $p_{1} y_{2}$ involutively reduces to zero using $P$. Assuming that the first step of this involutive reduction involves the polynomial $p_{2} \in P$, we can write

$$
\begin{equation*}
p_{1} y_{2}=u_{2} p_{2} v_{2}+\sum_{b=1}^{B} u_{\beta_{b}} p_{\beta_{b}} v_{\beta_{b}}, \tag{5.4}
\end{equation*}
$$

where $p_{\beta_{b}} \in P$ and $u_{2}, v_{2}, u_{\beta_{b}}, v_{\beta_{b}}$ are terms such that $p_{2}$ and each $p_{\beta_{b}}$ involutively divide $u_{2} p_{2} v_{2}$ and each $u_{\beta_{b}} p_{\beta_{b}} v_{\beta_{b}}$ respectively. Multiplying both sides of Equation (5.4) on the left by $u^{\prime} u_{1}$ and on the right by $v^{\prime}:=\operatorname{Suffix}\left(v_{1} v, \operatorname{deg}\left(v_{1} v\right)-1\right)$, we obtain the equation

$$
\begin{equation*}
u^{\prime} u_{1} p_{1} v_{1} v=u^{\prime} u_{1} u_{2} p_{2} v_{2} v^{\prime}+\sum_{b=1}^{B} u^{\prime} u_{1} u_{\beta_{b}} p_{\beta_{b}} v_{\beta_{b}} v^{\prime} . \tag{5.5}
\end{equation*}
$$

Substituting for $u^{\prime} u_{1} p_{1} v_{1} v$ from Equation (5.5) into Equation (5.3), we obtain the equation

$$
\begin{equation*}
u p v=u^{\prime} u_{1} u_{2} p_{2} v_{2} v^{\prime}+\sum_{a=1}^{A} u^{\prime} u_{\alpha_{a}} p_{\alpha_{a}} v_{\alpha_{a}} v+\sum_{b=1}^{B} u^{\prime} u_{1} u_{\beta_{b}} p_{\beta_{b}} v_{\beta_{b}} v^{\prime} . \tag{5.6}
\end{equation*}
$$

If $\left.p_{2}\right|_{I} u p v$, it is clear that we can use $p_{2}$ to involutively reduce the polynomial $u p v$ to obtain the polynomial $\sum_{a=1}^{A} u^{\prime} u_{\alpha_{a}} p_{\alpha_{a}} v_{\alpha_{a}} v+\sum_{b=1}^{B} u^{\prime} u_{1} u_{\beta_{b}} p_{\beta_{b}} v_{\beta_{b}} v^{\prime}$. As before, we can then use Proposition 5.2.5 to continue the involutive reduction of $u p v$ by repeating this proof on each summand individually.

Otherwise, if $p_{2}$ does not involutively divide $u p v$, we continue by induction, obtaining a sequence $p, p_{1}, p_{2}, p_{3}, \ldots$ of elements in $P$. By construction, each element in the sequence divides $u p v$. By continuity (at $\mathrm{LM}(u p v)$ ), no two elements in the sequence divide $u p v$ in the same way. Because upv has a finite number of subwords, the sequence must be finite, terminating with an involutive divisor $p^{\prime} \in P$ of $u p v$, which then allows us to finish the proof through use of Proposition 5.2.5 and the admissibility of $O$.

Theorem 5.4.4 An Involutive Basis with respect to a strong involutive division is a Gröbner Basis.

Proof: Let $G=\left\{g_{1}, \ldots, g_{m}\right\}$ be an Involutive Basis with respect to some strong involutive division $I$ and some admissible monomial ordering $O$, where each $g_{i} \in G$ (for all $1 \leqslant i \leqslant m$ ) is a member of the polynomial ring $R\left\langle x_{1}, \ldots, x_{n}\right\rangle$. To prove that $G$ is a Gröbner Basis, we must show that all S-polynomials involving elements of $G$ conventionally reduce to zero using $G$. Recall that each S-polynomial corresponds to an overlap between the lead monomials of two (not necessarily distinct) elements $g_{i}, g_{j} \in G$. Consider such an arbitrary overlap, with corresponding S-polynomial

$$
\operatorname{S-pol}\left(\ell_{i}, g_{i}, \ell_{j}, g_{j}\right)=c_{2} \ell_{i} g_{i} r_{i}-c_{1} \ell_{j} g_{j} r_{j}
$$

Because $G$ is an Involutive Basis, it is clear that $c_{2} \ell_{i} g_{i} r_{i} \vec{I}_{G} 0$ and $c_{1} \ell_{j} g_{j} r_{j} \vec{I}_{G} 0$. By Proposition 5.2.5, it follows that $\operatorname{S-pol}\left(\ell_{i}, g_{i}, \ell_{j}, g_{j}\right) \xrightarrow{{ }_{I}}{ }_{G} 0$. But every involutive reduction is a conventional reduction, so we can deduce that S-pol $\left(\ell_{i}, g_{i}, \ell_{j}, g_{j}\right) \rightarrow_{G} 0$ as required.

Lemma 5.4.5 Given an Involutive Basis $G$ with respect to a strong involutive division, remainders are involutively unique with respect to $G$.

Proof: Let $G$ be an Involutive Basis with respect to some strong involutive division $I$ and some admissible monomial ordering $O$. Theorem 5.4.4 tells us that $G$ is a Gröbner Basis with respect to $O$ and thus remainders are conventionally unique with respect to $G$. To prove that remainders are involutively unique with respect to $G$, we must show that the conventional and involutive remainders of an arbitrary polynomial $p$ with respect to $G$ are identical. For this it is sufficient to show that a polynomial $p$ is conventionally reducible by $G$ if and only if it is involutively reducible by $G .(\Rightarrow)$ Trivial as every involutive reduction is a conventional reduction. ( $\Leftarrow$ ) If a polynomial $p$ is conventionally reducible by a polynomial $g \in G$, it follows that $\mathrm{LM}(p)=u \mathrm{LM}(g) v$ for some monomials $u$ and $v$. But $G$ is an Involutive Basis, so there must exist a polynomial $g^{\prime} \in G$ such that $\left.\mathrm{LM}\left(g^{\prime}\right)\right|_{I} u \mathrm{LM}(g) v$. Thus $p$ is also involutively reducible by $G$.

### 5.4.2 Properties for Weak Involutive Divisions

While it is true that the previous three results (Proposition 5.4.3, Theorem 5.4.4 and Lemma 5.4.5) do not apply if a weak involutive division has been chosen, we will now show that corresponding results can be obtained for weak involutive divisions that are also Gröbner involutive divisions.

Definition 5.4.6 A weak involutive division $I$ is a Gröbner involutive division if every Locally Involutive Basis with respect to $I$ is a Gröbner Basis.

It is an easy consequence of Definition 5.4.6 that any Involutive Basis with respect to a weak and Gröbner involutive division is a Gröbner Basis; it therefore follows that we can also prove an analog of Lemma 5.4.5 for such divisions. To complete the mirroring of the results of Proposition 5.4.3, Theorem 5.4.4 and Lemma 5.4.5 for weak and Gröbner involutive divisions, it remains to show that a Locally Involutive Basis with respect to a weak; continuous and Gröbner involutive division is an Involutive Basis.

Proposition 5.4.7 If an involutive division I is weak; continuous and Gröbner, and if a given set of polynomials $P$ is a Locally Involutive Basis with respect to $I$ and some admissible monomial ordering $O$, then $P$ is an Involutive Basis with respect to $I$ and $O$.

Proof: Let $I$ be a weak; continuous and Gröbner involutive division; let $O$ be an admissible monomial ordering; and let $P$ be a Locally Involutive Basis with respect to $I$ and $O$. Given any polynomial $p \in P$ and any terms $u$ and $v$, in order to show that $P$ is an Involutive Basis with respect to $I$ and $O$, we must show that upv $\longrightarrow_{I} 0$.

For the first part of the proof, we proceed as in the proof of Proposition 5.4.3 to find an involutive divisor $p^{\prime} \in P$ of $u p v$ using the continuity of $I$ at $\mathrm{LM}(u p v)$. This then allows us to involutive reduce upv using $p^{\prime}$ to obtain a polynomial $q$ of the form

$$
\begin{equation*}
q=\sum_{a=1}^{A} u_{a} p_{\alpha_{a}} v_{a}, \tag{5.7}
\end{equation*}
$$

where $p_{\alpha_{a}} \in P$ and the $u_{a}$ and the $v_{a}$ are terms.
For the second part of the proof, we now use the fact that $P$ is a Gröbner Basis to find a polynomial $q^{\prime} \in P$ such that $q^{\prime}$ conventionally divides $q$ (such a polynomial will always exist because $q$ is clearly a member of the ideal generated by $P$ ). If $q^{\prime}$ is an involutive divisor of $q$, then we can use $q^{\prime}$ to involutively reduce $q$ to obtain a polynomial $r$ of the form shown in Equation (5.7). Otherwise, if $q^{\prime}$ is not an involutive divisor of $q$, we can use the fact that $I$ is continuous at $\operatorname{LM}(q)$ to find such an involutive divisor, which we can then use to involutive reduce $q$ to obtain a polynomial $r$, again of the form shown in Equation (5.7). In both cases, we now proceed by induction on $r$, noting that this process will terminate because of the admissibility of $O$ (we have $\mathrm{LM}(r)<\mathrm{LM}(q)$ ).

To summarise, here is the situation for weak and Gröbner involutive divisions.
(a) Any Locally Involutive Basis returned by Algorithm 12 is an Involutive Basis if the involutive division used is weak; continuous and Gröbner (Proposition 5.4.7);
(b) Algorithm 12 always terminates whenever Algorithm 5 terminates if (in addition) the involutive division used is conclusive;
(c) Every Involutive Basis with respect to a weak and Gröbner involutive division is a Gröbner Basis.

### 5.5 Noncommutative Involutive Divisions

Before we consider some examples of useful noncommutative involutive divisions, let us remark that it is possible to categorise any noncommutative involutive division somewhere between the following two extreme global divisions.

Definition 5.5.1 (The Empty Division) Given any monomial $u$, let $u$ have no (left or right) multiplicative variables.

Definition 5.5.2 (The Full Division) Given any monomial $u$, let $u$ have no (left or right) nonmultiplicative variables (in other words, all variables are left and right multiplicative for $u$ ).

Remark 5.5.3 It is clear that any set of polynomials $G$ will be an Involutive Basis with respect to the (weak) full division as any multiple of a polynomial $g \in G$ will be involutively reducible by $g$ (all conventional divisors are involutive divisors); in contrast it is impossible to find a finite Locally Involutive Basis for $G$ with respect to the (strong) empty division as there will always be a prolongation of an element of the current basis that is involutively irreducible.

### 5.5.1 Two Global Divisions

Whereas most of the theory seen so far in this chapter has closely mirrored the corresponding commutative theory from Chapter 4, the commutative involutive divisions (Thomas, Janet and Pommaret) seen in the previous chapter do not generalise to the noncommutative case, or at the very least do not yield noncommutative involutive divisions of any
value. Despite this, an essential property of these divisions is that they ensure that the least common multiple $\operatorname{lcm}\left(\operatorname{LM}\left(p_{1}\right), \operatorname{LM}\left(p_{2}\right)\right)$ associated with an S-polynomial S-pol $\left(p_{1}, p_{2}\right)$ is involutively irreducible by at least one of $p_{1}$ and $p_{2}$, ensuring that the S-polynomial $\mathrm{S}-\mathrm{pol}\left(p_{1}, p_{2}\right)$ is constructed and involutively reduced during the course of the Involutive Basis algorithm.

To ensure that the corresponding process occurs in the noncommutative Involutive Basis algorithm, we must ensure that all overlap words associated to the S-polynomials of a particular basis are involutively irreducible (as placed in the overlap word) by at least one of the polynomials associated to each overlap word. This obviously holds true for the empty division, but it will also hold true for the following two global involutive divisions, where all variables are either assigned to be left multiplicative and right nonmultiplicative, or left nonmultiplicative and right multiplicative.

Definition 5.5.4 (The Left Division) Given any monomial $u$, the left division $\triangleleft$ assigns no left nonmultiplicative variables to $u$, and assigns no right multiplicative variables to $u$ (in other words, all variables are left multiplicative and right nonmultiplicative for $u)$.

Definition 5.5.5 (The Right Division) Given any monomial $u$, the right division $\triangleright$ assigns no left multiplicative variables to $u$, and assigns no right nonmultiplicative variables to $u$ (in other words, all variables are left nonmultiplicative and right multiplicative for $u$ ).

Proposition 5.5.6 The left and right divisions are strong involutive divisions.

Proof: We will only give the proof for the left division - the proof for the right division will follow by symmetry (replacing 'left' by 'right', and so on).

To prove that the left division is a strong involutive division, we need to show that the three conditions of Definition 5.1.6 hold.

## - Disjoint Cones Condition

Consider two involutive cones $\mathcal{C}_{\triangleleft}\left(u_{1}\right)$ and $\mathcal{C}_{\triangleleft}\left(u_{2}\right)$ associated to two monomials $u_{1}, u_{2}$ over some noncommutative polynomial ring $\mathcal{R}$. If $\mathcal{C}_{\triangleleft}\left(u_{1}\right) \cap \mathcal{C}_{\triangleleft}\left(u_{2}\right) \neq \emptyset$, then there must be some monomial $v \in \mathcal{R}$ such that $v$ contains both monomials $u_{1}$ and $u_{2}$ as subwords, and (as placed in $v$ ) both $u_{1}$ and $u_{2}$ must be involutive divisors of $v$. By
definition of $\triangleleft$, both $u_{1}$ and $u_{2}$ must be suffices of $v$. Thus, assuming (without loss of generality) that $\operatorname{deg}\left(u_{1}\right)>\operatorname{deg}\left(u_{2}\right)$, we are able to draw the following diagram summarising the situation.


But now, assuming that $u_{1}=u_{3} u_{2}$ for some monomial $u_{3}$, it is clear that $\mathcal{C}_{\triangleleft}\left(u_{1}\right) \subset$ $\mathcal{C}_{\triangleleft}\left(u_{2}\right)$ because any monomial $w \in \mathcal{C}_{\triangleleft}\left(u_{1}\right)$ must be of the form $w=w^{\prime} u_{1}$ for some monomial $w^{\prime}$; this means that $w=w^{\prime} u_{3} u_{2} \in \mathcal{C}_{\triangleleft}\left(u_{2}\right)$.

## - Unique Divisor Condition

As a monomial $v$ is only involutively divisible by a monomial $u$ with respect to the left division if $u$ is a suffix of $v$, it is clear that $u$ can only involutively divide $v$ in at most one way.

## - Subset Condition

Follows immediately due to the left division being a global division.

Proposition 5.5.7 The left and right divisions are continuous.

Proof: Again we will only treat the case of the left division. Let $w$ be an arbitrary fixed monomial; let $U$ be any set of monomials; and consider any sequence ( $u_{1}, u_{2}, \ldots, u_{k}$ ) of monomials from $U\left(u_{i} \in U\right.$ for all $\left.1 \leqslant i \leqslant k\right)$, each of which is a conventional divisor of $w$ (so that $w=\ell_{i} u_{i} r_{i}$ for all $1 \leqslant i \leqslant k$, where the $\ell_{i}$ and the $r_{i}$ are monomials). For all $1 \leqslant i<k$, suppose that the monomial $u_{i+1}$ satisfies condition (b) of Definition 5.4.2 (condition (a) can never be satisfied because $\triangleleft$ never assigns any left nonmultiplicative variables). To show that $\triangleleft$ is continuous, we must show that no two pairs ( $\ell_{i}, r_{i}$ ) and $\left(\ell_{j}, r_{j}\right)$ are the same, where $i \neq j$.

Consider an arbitrary monomial $u_{i}$ from the sequence, where $1 \leqslant i<k$. Because $\triangleleft$ assigns no right multiplicative variables, the next monomial $u_{i+1}$ in the sequence must be a suffix of the prolongation $u_{i}\left(\operatorname{Prefix}\left(r_{i}, 1\right)\right)$ of $u_{i}$, so that $\operatorname{deg}\left(r_{i+1}\right)=\operatorname{deg}\left(r_{i}\right)-1$.

It is therefore clear that no two identical $(\ell, r)$ pairs can be found in the sequence, as $\operatorname{deg}\left(r_{1}\right)>\operatorname{deg}\left(r_{2}\right)>\cdots>\operatorname{deg}\left(r_{k}\right)$.

To illustrate the difference between the overlapping cones of a noncommutative Gröbner Basis and the disjoint cones of a noncommutative Involutive Basis with respect to the left division, consider the following example.

Example 5.5.8 Let $F:=\left\{2 x y+y^{2}+5, x^{2}+y^{2}+8\right\}$ be a basis over the polynomial ring $\mathbb{Q}\langle x, y\rangle$, and let the monomial ordering be DegLex. Applying Algorithm 5 to $F$, we obtain the Gröbner Basis $G:=\left\{2 x y+y^{2}+5, x^{2}+y^{2}+8,5 y^{3}-10 x+37 y, 2 y x+y^{2}+5\right\}$. Applying Algorithm 12 to $F$ with respect to the left involutive division, we obtain the Involutive Basis $H:=\left\{2 x y+y^{2}+5, x^{2}+y^{2}+8,5 y^{3}-10 x+37 y, 5 x y^{2}+5 x-6 y, 2 y x+y^{2}+5\right\}$.

To illustrate which monomials are reducible with respect to the Gröbner Basis, we can draw a monomial lattice, part of which is shown below. In the lattice, we draw a path from the (circled) lead monomial of any Gröbner Basis element to any multiple of that lead monomial, so that any monomial which lies on some path in the lattice is reducible by one or more Gröbner Basis elements. To distinguish between different Gröbner Basis elements we use different arrow types; we also arrange the lattice so that monomials of the same degree lie on the same level.


Notice that many of the monomials in the lattice are reducible by several of the Gröbner Basis elements. For example, the monomial $x^{2} y x$ is reducible by the Gröbner Basis elements $2 x y+y^{2}+5 ; x^{2}+y^{2}+8$ and $2 y x+y^{2}+5$. In contrast, any monomial in the corresponding lattice for the Involutive Basis may only be involutively reducible by at most one element in the Involutive Basis. We illustrate this by the following diagram, where we note that in the involutive lattice, a monomial only lies on a particular path if a member of the Involutive Basis is an involutive divisor of that monomial.

1
$x \quad y$


Comparing the two monomial lattices, we see that any monomial that is conventionally divisible by the Gröbner Basis is uniquely involutively divisible by the Involutive Basis. In other words, the involutive cones of the Involutive Basis form a disjoint cover of the conventional cones of the Gröbner Basis.

## Fast Reduction

In the commutative case, we can sometimes use the properties of an involutive division to speed up the process of involutively reducing a polynomial with respect to a set of polynomials. For example, the Janet tree [27, 28] enables us to quickly determine whether a polynomial is involutively reducible by a set of polynomials with respect to the Janet involutive division.

In the noncommutative case, we usually use Algorithm 10 to involutively reduce a polynomial $p$ with respect to a set of polynomials $P$. When this is done with respect to the left or right divisions however, we can improve Algorithm 10 by taking advantage of the fact that a monomial $u_{1}$ only involutively divides another monomial $u_{2}$ with respect to the left (right) division if $u_{1}$ is a suffix (prefix) of $u_{2}$.

For the left division, we can replace the code found in the first if loop of Algorithm 10 with the following code in order to obtain an improved algorithm.

```
if \(\left(\operatorname{LM}\left(p_{j}\right)\right.\) is a suffix of \(\left.u\right)\) then
    found \(=\) true;
    \(p=p-\left(c \mathrm{LC}\left(p_{j}\right)^{-1}\right) u_{\ell} p_{j}\), where \(u_{\ell}=\operatorname{Prefix}\left(p, \operatorname{deg}(p)-\operatorname{deg}\left(p_{j}\right)\right)\);
else
    \(j=j+1 ;\)
end if
```

We note that only one operation is required to determine whether the monomial $\operatorname{LM}\left(p_{j}\right)$ involutively divides the monomial $u$ here (test to see if $\operatorname{LM}\left(p_{j}\right)$ is a suffix of $u$ ); whereas in general there are many ways that $\operatorname{LM}\left(p_{j}\right)$ can conventionally divide $u$, each of which has to be tested to see whether it is an involutive reduction. This means that, with respect to the left or right divisions, we can determine whether a monomial $u$ is involutively irreducible with respect to a set of polynomials $P$ in linear time (linear in the number of elements in $P$ ); whereas in general we can only do this in quadratic time.

### 5.5.2 An Overlap-Based Local Division

Even though the left and right involutive divisions are strong and continuous (so that any Locally Involutive Basis returned by Algorithm 12 is a noncommutative Gröbner Basis), these divisions are not conclusive as the following example demonstrates.

Example 5.5.9 Let $F:=\left\{x y-z, x+z, y z-z, x z, z y+z, z^{2}\right\}$ be a basis over the polynomial ring $\mathbb{Q}\langle x, y, z\rangle$, and let the monomial ordering be DegLex. Applying Algorithm 5 to $F$, we discover that $F$ is a noncommutative Gröbner Basis ( $F$ is returned to us as the output of Algorithm 5). When we apply Algorithm 12 to $F$ with respect to the left involutive division however, we notice that the algorithm goes into an infinite loop, constructing the infinite basis $G:=F \cup\left\{z y^{n}-z, x y^{n}+z, z y^{m}+z, x y^{m}-z\right\}$, where $n \geqslant 2$, $n$ even and $m \geqslant 3, m$ odd.

The reason why Algorithm 12 goes into an infinite loop in the above example is that the right prolongations of the polynomials $x y-z$ and $z y+z$ by the variable $y$ do not involutively reduce to zero (they reduce to the polynomials $x y^{2}+z$ and $z y^{2}-z$ respectively). These prolongations are the only prolongations of elements of $F$ that do not involutively reduce to zero, and this is also true for all polynomials we subsequently add to $F$, thus allowing Algorithm 12 to construct the infinite set $G$.

Consider a modification of the left division where we assign the variable $y$ to be right multiplicative for the (lead) monomials $x y$ and $z y$. Then it is clear that $F$ will be a Locally Involutive Basis with respect to this modified division, but will it also be true that $F$ is an Involutive Basis and (had we not known so already) a Gröbner Basis?

Intuitively, for this particular example, it would seem that the answer to both of the above questions should be affirmative, because the modified division still ensures that all the overlap words associated with the S-polynomials of $F$ are involutively irreducible (as placed in the overlap word) by at least one of the polynomials associated to each S-polynomial. This leads to the following idea for a local involutive division, where we refine the left division by choosing right nonmultiplicative variables based on the overlap words of S-polynomials associated to a set of polynomials only (note that there will also be a similar local involutive division refining the right division called the right overlap division).

Definition 5.5.10 (The Left Overlap Division $\mathcal{O}$ ) Let $U=\left\{u_{1}, \ldots, u_{m}\right\}$ be a set of monomials, and assume that all variables are left and right multiplicative for all elements of $U$ to begin with.
(a) For all possible ways that a monomial $u_{j} \in U$ is a subword of a (different) monomial $u_{i} \in U$, so that

$$
\operatorname{Subword}\left(u_{i}, k, k+\operatorname{deg}\left(u_{j}\right)-1\right)=u_{j}
$$

for some integer $k$, if $u_{j}$ is not a suffix of $u_{i}$, assign the variable $\operatorname{Subword}\left(u_{i}, k+\right.$ $\left.\operatorname{deg}\left(u_{j}\right), k+\operatorname{deg}\left(u_{j}\right)\right)$ to be right nonmultiplicative for $u_{j}$.
(b) For all possible ways that a proper prefix of a monomial $u_{i} \in U$ is equal to a proper suffix of a (not necessarily different) monomial $u_{j} \in U$, so that

$$
\operatorname{Prefix}\left(u_{i}, k\right)=\operatorname{Suffix}\left(u_{j}, k\right)
$$

for some integer $k$ and $u_{i}$ is not a subword of $u_{j}$ or vice-versa, assign the variable Subword $\left(u_{i}, k+1, k+1\right)$ to be right nonmultiplicative for $u_{j}$.

Remark 5.5.11 One possible algorithm for the left overlap division is presented in Algorithm 13, where the reason for insisting that the input set of monomials is ordered with respect to DegRevLex is in order to minimise the number of operations needed to discover all the subword overlaps (a monomial of degree $d_{1}$ can never be a subword of a different monomial of degree $d_{2} \leqslant d_{1}$ ).

Example 5.5.12 Consider again the set of polynomials $F:=\{x y-z, x+z, y z-$ $\left.z, x z, z y+z, z^{2}\right\}$ from Example 5.5.9. Here are the left and right multiplicative variables for $\mathrm{LM}(F)$ with respect to the left overlap division $\mathcal{O}$.

| $u$ | $\mathcal{M}_{\mathcal{O}}^{L}(u, \mathrm{LM}(F))$ | $\mathcal{M}_{\mathcal{O}}^{R}(u, \mathrm{LM}(F))$ |
| :---: | :---: | :---: |
| $x y$ | $\{x, y, z\}$ | $\{x, y\}$ |
| $x$ | $\{x, y, z\}$ | $\{x\}$ |
| $y z$ | $\{x, y, z\}$ | $\{x\}$ |
| $x z$ | $\{x, y, z\}$ | $\{x\}$ |
| $z y$ | $\{x, y, z\}$ | $\{x, y\}$ |
| $z^{2}$ | $\{x, y, z\}$ | $\{x\}$ |

When we apply Algorithm 12 to $F$ with respect to the DegLex monomial ordering and the left overlap division, $F$ is returned to us as the output, an assertion that is easily verified by showing that the 10 right prolongations of elements of $F$ all involutively reduce to zero using $F$. This means that $F$ is a Locally Involutive Basis with respect to the left overlap division; to show that $F$ (and indeed any Locally Involutive Basis returned by Algorithm 12 with respect to the left overlap division) is also an Involutive Basis with respect to the left overlap division, we need to show that the left overlap division is continuous and either strong or Gröbner; we begin (after the following remark) by showing that the left overlap division is continuous.

Remark 5.5.13 In the above example, the table of multiplicative variables can be constructed from the table $T$ shown below, a table that is obtained by applying Algorithm 13 to $\operatorname{LM}(F)$.

```
Algorithm 13 The Left Overlap Division \(\mathcal{O}\)
    \(u_{m}\) ), where \(u_{i} \in R\left\langle x_{1}, \ldots, x_{n}\right\rangle\). entry of \(T\) is either 1 (multiplicative) or 0 (nonmultiplicative).
Create a table \(T\) of multiplicative variables as shown below:
\begin{tabular}{c|ccccccc} 
& \(x_{1}^{L}\) & \(x_{1}^{R}\) & \(x_{2}^{L}\) & \(x_{2}^{R}\) & \(\cdots\) & \(x_{n}^{L}\) & \(x_{n}^{R}\) \\
\hline\(u_{1}\) & 1 & 1 & 1 & 1 & \(\cdots\) & 1 & 1 \\
\(u_{2}\) & 1 & 1 & 1 & 1 & \(\cdots\) & 1 & 1 \\
\(\vdots\) & \(\vdots\) & \(\vdots\) & \(\vdots\) & \(\vdots\) & \(\ddots\) & \(\vdots\) & \(\vdots\) \\
\(u_{m}\) & 1 & 1 & 1 & 1 & \(\cdots\) & 1 & 1
\end{tabular}
for each monomial \(u_{i} \in U(1 \leqslant i \leqslant m)\) do
for each monomial \(u_{j} \in U(i \leqslant j \leqslant m)\) do
Let \(u_{i}=x_{i_{1}} x_{i_{2}} \ldots x_{i_{\alpha}}\) and \(u_{j}=x_{j_{1}} x_{j_{2}} \ldots x_{j_{\beta}}\);
if \((i \neq j)\) then
for each \(k(1 \leqslant k<\alpha-\beta+1)\) do
if \(\left(\operatorname{Subword}\left(u_{i}, k, k+\beta-1\right)==u_{j}\right)\) then
\[
T\left(u_{j}, x_{i_{k+\beta}}^{R}\right)=0 ;
\]
end if
end for
end if
for each \(k(1 \leqslant k \leqslant \beta-1)\) do
if \(\left(\operatorname{Prefix}\left(u_{i}, k\right)==\operatorname{Suffix}\left(u_{j}, k\right)\right)\) then
\(T\left(u_{j}, x_{i_{k+1}}^{R}\right)=0 ;\)
end if
if \(\left(\operatorname{Suffix}\left(u_{i}, k\right)==\operatorname{Prefix}\left(u_{j}, k\right)\right)\) then
\[
T\left(u_{i}, x_{j_{k+1}}^{R}\right)=0 ;
\]
end if
end for
end for
end for
return \(T\);
```

Input: A set of monomials $U=\left\{u_{1}, \ldots, u_{m}\right\}$ ordered by DegRevLex $\left(u_{1} \geqslant u_{2} \geqslant \cdots \geqslant\right.$

Output: A table $T$ of left and right multiplicative variables for all $u_{i} \in U$, where each

| Monomial | $x^{L}$ | $x^{R}$ | $y^{L}$ | $y^{R}$ | $z^{L}$ | $z^{R}$ |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| $x y$ | 1 | 1 | 1 | 1 | 1 | 0 |
| $x$ | 1 | 1 | 1 | 0 | 1 | 0 |
| $y z$ | 1 | 1 | 1 | 0 | 1 | 0 |
| $x z$ | 1 | 1 | 1 | 0 | 1 | 0 |
| $z y$ | 1 | 1 | 1 | 1 | 1 | 0 |
| $z^{2}$ | 1 | 1 | 1 | 0 | 1 | 0 |

The zero entries in $T$ correspond to the following overlaps between the elements of $\mathrm{LM}(F)$.

| Table Entry | Overlap |
| :---: | :---: |
| $T\left(x y, z^{R}\right)$ | $\operatorname{Suffix}(x y, 1)=\operatorname{Prefix}(y z, 1)$ |
| $T\left(x, y^{R}\right)$ | $\operatorname{Subword}(x y, 1,1)=x$ |
| $T\left(x, z^{R}\right)$ | $\operatorname{Subword}(x z, 1,1)=x$ |
| $T\left(y z, y^{R}\right)$ | $\operatorname{Suffix}(y z, 1)=\operatorname{Prefix}(z y, 1)$ |
| $T\left(y z, z^{R}\right)$ | $\operatorname{Suffix}(y z, 1)=\operatorname{Prefix}\left(z^{2}, 1\right)$ |
| $T\left(x z, y^{R}\right)$ | $\operatorname{Suffix}(x z, 1)=\operatorname{Prefix}(z y, 1)$ |
| $T\left(x z, z^{R}\right)$ | $\operatorname{Suffix}(x z, 1)=\operatorname{Prefix}\left(z^{2}, 1\right)$ |
| $T\left(z y, z^{R}\right)$ | $\operatorname{Suffix}(z y, 1)=\operatorname{Prefix}(y z, 1)$ |
| $T\left(z^{2}, y^{R}\right)$ | $\operatorname{Suffix}\left(z^{2}, 1\right)=\operatorname{Prefix}(z y, 1)$ |
| $T\left(z^{2}, z^{R}\right)$ | $\operatorname{Suffix}\left(z^{2}, 1\right)=\operatorname{Prefix}\left(z^{2}, 1\right)$ |

Proposition 5.5.14 The left overlap division $\mathcal{O}$ is continuous.

Proof: Let $w$ be an arbitrary fixed monomial; let $U$ be any set of monomials; and consider any sequence $\left(u_{1}, u_{2}, \ldots, u_{k}\right)$ of monomials from $U\left(u_{i} \in U\right.$ for all $\left.1 \leqslant i \leqslant k\right)$, each of which is a conventional divisor of $w$ (so that $w=\ell_{i} u_{i} r_{i}$ for all $1 \leqslant i \leqslant k$, where the $\ell_{i}$ and the $r_{i}$ are monomials). For all $1 \leqslant i<k$, suppose that the monomial $u_{i+1}$ satisfies condition (b) of Definition 5.4.2 (condition (a) can never be satisfied because $\mathcal{O}$ never assigns any left nonmultiplicative variables). To show that $\mathcal{O}$ is continuous, we must show that no two pairs $\left(\ell_{i}, r_{i}\right)$ and $\left(\ell_{j}, r_{j}\right)$ are the same, where $i \neq j$.

Consider an arbitrary monomial $u_{i}$ from the sequence, where $1 \leqslant i<k$. By definition of $\mathcal{O}$, the next monomial $u_{i+1}$ in the sequence cannot be either a prefix or a proper subword of $u_{i}$. This leaves two possibilities: (i) $u_{i+1}$ is a suffix of $u_{i}$ (in which case $\operatorname{deg}\left(u_{i+1}\right)<$ $\operatorname{deg}\left(u_{i}\right)$ ); or (ii) $u_{i+1}$ is a suffix of the prolongation $u_{i} v_{i}$ of $u_{i}$, where $v_{i}:=\operatorname{Prefix}\left(r_{i}, 1\right)$.

Example of possibility (i) Example of possibility (ii)


In both cases, it is clear that we have $\operatorname{deg}\left(r_{i+1}\right) \leqslant \operatorname{deg}\left(r_{i}\right)$, so that $\operatorname{deg}\left(r_{1}\right) \geqslant \operatorname{deg}\left(r_{2}\right) \geqslant$ $\cdots \geqslant \operatorname{deg}\left(r_{k}\right)$. It follows that no two ( $\left.\ell, r\right)$ pairs in the sequence can be the same, because for each subsequence $u_{a}, u_{a+1}, \ldots, u_{b}$ such that $\operatorname{deg}\left(r_{a}\right)=\operatorname{deg}\left(r_{a+1}\right)=\cdots=\operatorname{deg}\left(r_{b}\right)$, we must have $\operatorname{deg}\left(\ell_{a}\right)<\operatorname{deg}\left(\ell_{a+1}\right)<\cdots<\operatorname{deg}\left(\ell_{b}\right)$.

Having shown that the left overlap division is continuous, one way of showing that every Locally Involutive Basis with respect to the left overlap division is an Involutive Basis would be to show that the left overlap division is a strong involutive division. However, the left overlap division is only a weak involutive division, as the following counterexample demonstrates.

Proposition 5.5.15 The left overlap division is a weak involutive division.

Proof: Let $U:=\{y z, x y\}$ be a set of monomials over the polynomial ring $\mathbb{Q}\langle x, y, z\rangle$. Here are the multiplicative variables for $U$ with respect to the left overlap division $\mathcal{O}$.

| $u$ | $\mathcal{M}_{\mathcal{O}}^{L}(u, U)$ | $\mathcal{M}_{\mathcal{O}}^{R}(u, U)$ |
| :---: | :---: | :---: |
| $y z$ | $\{x, y, z\}$ | $\{x, y, z\}$ |
| $x y$ | $\{x, y, z\}$ | $\{x, y\}$ |

Because $y z x y \in \mathcal{C}_{\mathcal{O}}(y z, U)$ and $y z x y \in \mathcal{C}_{\mathcal{O}}(x y, U)$, one of the conditions $\mathcal{C}_{\mathcal{O}}(y z, U) \subset$ $\mathcal{C}_{\mathcal{O}}(x y, U)$ or $\mathcal{C}_{\mathcal{O}}(x y, U) \subset \mathcal{C}_{\mathcal{O}}(y z, U)$ must be satisfied in order for $\mathcal{O}$ to be a strong involutive division (this is the Disjoint Cones condition of Definition 5.1.6). But neither of these conditions can be satisfied when we consider that $x y \notin \mathcal{C}_{\mathcal{O}}(y z, U)$ and $y z \notin$ $\mathcal{C}_{\mathcal{O}}(x y, U)$, so $\mathcal{O}$ must be a weak involutive division.

The weakness of the left overlap division is the price we pay for refining the left division by allowing more right multiplicative variables. All is not lost however, as we can still show that every Locally Involutive Basis with respect to the left overlap division is an Involutive Basis by showing that the left overlap division is a Gröbner involutive division.

Proposition 5.5.16 The left overlap division $\mathcal{O}$ is a Gröbner involutive division.

Proof: We are required to show that if Algorithm 12 terminates with $\mathcal{O}$ and some arbitrary admissible monomial ordering $O$ as input, then the Locally Involutive Basis $G$ it returns is a noncommutative Gröbner Basis. By Definition 3.1.8, we can do this by showing that all S-polynomials involving elements of $G$ conventionally reduce to zero using $G$.

Assume that $G=\left\{g_{1}, \ldots, g_{p}\right\}$ is sorted (by lead monomial) with respect to the DegRevLex monomial ordering (greatest first), and let $U=\left\{u_{1}, \ldots, u_{p}\right\}:=\left\{\operatorname{LM}\left(g_{1}\right), \ldots, \mathrm{LM}\left(g_{p}\right)\right\}$ be the set of leading monomials. Let $T$ be the table obtained by applying Algorithm 13 to $U$. Because $G$ is a Locally Involutive Basis, every zero entry $T\left(u_{i}, x_{j}^{\Gamma}\right)(\Gamma \in\{L, R\})$ in the table corresponds to a prolongation $g_{i} x_{j}$ or $x_{j} g_{i}$ that involutively reduces to zero.

Let $S$ be the set of S-polynomials involving elements of $G$, where the $t$-th entry of $S$ $(1 \leqslant t \leqslant|S|)$ is the $S$-polynomial

$$
s_{t}=c_{t} \ell_{t} g_{i} r_{t}-c_{t}^{\prime} \ell_{t}^{\prime} g_{j} r_{t}^{\prime},
$$

with $\ell_{t} u_{i} r_{t}=\ell_{t}^{\prime} u_{j} r_{t}^{\prime}$ being the overlap word of the S-polynomial. We will prove that every S-polynomial in $S$ conventionally reduces to zero using $G$.

Recall (from Definition 3.1.2) that each S-polynomial in $S$ corresponds to a particular type of overlap - 'prefix', 'subword' or 'suffix'. For the purposes of this proof, let us now split the subword overlaps into three further types - 'left', 'middle' and 'right', corresponding to the cases where a monomial $m_{2}$ is a prefix, proper subword and suffix of a monomial $m_{1}$.


This classification provides us with five cases to deal with in total, which we shall process in the following order: right, middle, left, prefix, suffix.
(1) Consider an arbitrary entry $s_{t} \in S(1 \leqslant t \leqslant|S|)$ corresponding to a right overlap where the monomial $u_{j}$ is a suffix of the monomial $u_{i}$. Because $\mathcal{O}$ never assigns any left nonmultiplicative variables, $u_{j}$ must be an involutive divisor of $u_{i}$. But this contradicts the fact that the set $G$ is autoreduced; it follows that no $S$-polynomials corresponding to
right overlaps can appear in $S$.
(2) Consider an arbitrary entry $s_{t} \in S(1 \leqslant t \leqslant|S|)$ corresponding to a middle overlap where the monomial $u_{j}$ is a proper subword of the monomial $u_{i}$. This means that $s_{t}=$ $c_{t} g_{i}-c_{t}^{\prime} \ell_{t}^{\prime} g_{j} r_{t}^{\prime}$ for some $g_{i}, g_{j} \in G$, with overlap word $u_{i}=\ell_{t}^{\prime} u_{j} r_{t}^{\prime}$. Let $u_{i}=x_{i_{1}} \ldots x_{i_{\alpha}}$; let $u_{j}=x_{j_{1}} \ldots x_{j_{\beta}}$; and choose $D$ such that $x_{i_{D}}=x_{j_{\beta}}$.

$$
\begin{aligned}
& u_{i}= \\
& u_{j}= \\
& \overline{x_{i_{1}}}--\overline{x_{i_{D-\beta}}} \overline{\frac{x_{i_{D-\beta+1}}}{} \overline{x_{i_{D-\beta+2}}}---\overline{x_{i_{D-1}}} \frac{}{x_{i_{D}}} \overline{x_{i_{D+1}}}-\cdots-\overline{x_{i_{\alpha}}}} \begin{array}{c}
\frac{x_{j_{1}}}{x_{j_{2}}}--\frac{\overline{x_{j_{\beta-1}}} \frac{x_{j_{\beta}}}{x^{2}}}{}
\end{array}
\end{aligned}
$$

Because $u_{j}$ is a proper subword of $u_{i}$, it follows that $T\left(u_{j}, x_{i_{D+1}}^{R}\right)=0$. This gives rise to the prolongation $g_{j} x_{i_{D+1}}$ of $g_{j}$. But we know that all prolongations involutively reduce to zero ( $G$ is a Locally Involutive Basis), so Algorithm 10 must find a monomial $u_{k}=$ $x_{k_{1}} \ldots x_{k_{\gamma}} \in U$ such that $u_{k}$ involutively divides $u_{j} x_{i_{D+1}}$. Assuming that $x_{k_{\gamma}}=x_{i_{\kappa}}$, we can deduce that any candidate for $u_{k}$ must be a suffix of $u_{j} x_{i_{D+1}}$ (otherwise $T\left(u_{k}, x_{i_{k+1}}^{R}\right)=0$ because of the overlap between $u_{i}$ and $u_{k}$ ). This means that the degree of $u_{k}$ is in the range $1 \leqslant \gamma \leqslant \beta+1$; we shall illustrate this in the following diagram by using a squiggly line to indicate that the monomial $u_{k}$ can begin anywhere (or nowhere if $u_{k}=x_{i_{D+1}}$ ) on the squiggly line.

$$
\begin{aligned}
& u_{i}=\quad \overline{x_{i_{1}}}---\frac{}{x_{i_{D-\beta}}} \overline{x_{i_{D-\beta+1}}} \overline{x_{i_{D-\beta+2}}}---\frac{}{x_{i_{D-1}}} \frac{}{x_{i_{D}}} \overline{x_{i_{D+1}}}---\frac{}{x_{i_{\alpha}}} \\
& u_{j}= \\
& u_{k}= \\
& \frac{}{x_{j_{1}}} \frac{x_{j_{2}}}{x^{2}}--\frac{}{x_{j_{\beta-1}}} \frac{}{x_{j_{\beta}}} \xrightarrow{ } \frac{}{x_{k_{\gamma}}}
\end{aligned}
$$

We can now use the monomial $u_{k}$ together with Buchberger's Second Criterion to simplify our goal of showing that the S-polynomial $s_{t}$ reduces to zero. Notice that the monomial $u_{k}$ is a subword of the overlap word $u_{i}$ associated to $s_{t}$, and so in order to show that $s_{t}$ reduces to zero, all we have to do is to show that the two S-polynomials

$$
s_{u}=c_{u} g_{i}-c_{u}^{\prime}\left(x_{i_{1}} x_{i_{2}} \ldots x_{i_{D+1-\gamma}}\right) g_{k}\left(x_{i_{D+2}} \ldots x_{i_{\alpha}}\right)
$$

and $^{2}$

$$
s_{v}=c_{v}\left(x_{j_{1}} \ldots x_{i_{D+1-\gamma}}\right) g_{k}-c_{v}^{\prime} g_{j} x_{i_{D+1}}
$$

[^10]reduce to zero $(1 \leqslant u, v \leqslant|S|)$.
For the S-polynomial $s_{v}$, there are two cases to consider: $\gamma=1$, and $\gamma>1$. In the former case, because (as placed in $u_{i}$ ) the monomials $u_{j}$ and $u_{k}$ do not overlap, we can use Buchberger's First Criterion to say that the 'S-polynomial' $s_{v}$ reduces to zero (for further explanation, see the paragraph at the beginning of Section 3.4.1). In the latter case, we know that the first step of the involutive reduction of the prolongation $g_{j} x_{i_{D+1}}$ is to take away the multiple $\left(\frac{c_{v}}{c_{v}}\right)\left(x_{j_{1}} \ldots x_{i_{D+1-\gamma}}\right) g_{k}$ of $g_{k}$ from $g_{j} x_{i_{D+1}}$ to leave the polynomial $g_{j} x_{i_{D+1}}-\left(\frac{c_{y}}{c_{v}^{v}}\right)\left(x_{j_{1}} \ldots x_{i_{D+1-\gamma}}\right) g_{k}=-\left(\frac{1}{c_{v}^{v}}\right) s_{v}$. But as we know that all prolongations involutively reduce to zero, we can conclude that the S-polynomial $s_{v}$ conventionally reduces to zero.

For the S-polynomial $s_{u}$, we note that if $D=\alpha-1$, then $s_{u}$ corresponds to a right overlap. But we know from part (1) that right overlaps cannot appear in $S$, and so $s_{t}$ also cannot appear in $S$. Otherwise, we proceed by induction on the S-polynomial $s_{u}$ to produce a sequence $\left\{u_{q_{D+1}}, u_{q_{D+2}}, \ldots, u_{q_{\alpha}}\right\}$ of monomials, so that $s_{u}$ (and hence $s_{t}$ ) reduces to zero if the S -polynomial

$$
s_{\eta}=c_{\eta} g_{i}-c_{\eta}^{\prime}\left(x_{i_{1}} \ldots x_{i_{\alpha-\mu}}\right) g_{q_{\alpha}}
$$

reduces to zero $(1 \leqslant \eta \leqslant|S|)$, where $\mu=\operatorname{deg}\left(u_{q_{\alpha}}\right)$.

$$
\begin{aligned}
& u_{i}=\quad \overline{x_{i_{1}}}---\overline{x_{i_{D-\beta}}} \bar{x} \overline{i_{D-\beta+1}}---\frac{}{x_{i_{D}}} \overline{x_{i_{D+1}}} \overline{x_{i_{D+2}}}---\frac{}{x_{i_{\alpha-1}}} \frac{}{x_{i_{\alpha}}} \\
& u_{j}= \\
& u_{q_{D+1}}=u_{k}= \\
& u_{q_{D+2}}= \\
& u_{q_{\alpha}}=
\end{aligned}
$$

But $s_{\eta}$ always corresponds to a right overlap, so we must conclude that middle overlaps (as well as right overlaps) cannot appear in $S$.
(3) Consider an arbitrary entry $s_{t} \in S(1 \leqslant t \leqslant|S|)$ corresponding to a left overlap where the monomial $u_{j}$ is a prefix of the monomial $u_{i}$. This means that $s_{t}=c_{t} g_{i}-c_{t}^{\prime} g_{j} r_{t}^{\prime}$ for
some $g_{i}, g_{j} \in G$, with overlap word $u_{i}=u_{j} r_{t}^{\prime}$. Let $u_{i}=x_{i_{1}} \ldots x_{i_{\alpha}}$ and let $u_{j}=x_{j_{1}} \ldots x_{j_{\beta}}$.

$$
\begin{array}{ll}
u_{i}= & \frac{x_{i_{1}}}{x_{i_{2}}}---\frac{}{x_{i_{\beta-1}}} \frac{}{x_{i_{\beta}}} \frac{}{x_{i_{\beta+1}}}---\frac{}{x_{i_{\alpha-1}}} \frac{}{x_{i_{\alpha}}} \\
u_{j} & =\quad \frac{x_{j_{1}}}{x_{j_{\beta-1}}} \frac{}{x_{j_{\beta}}}
\end{array}
$$

Because $u_{j}$ is a prefix of $u_{i}$, it follows that $T\left(u_{j}, x_{i_{\beta+1}}^{R}\right)=0$. This gives rise to the prolongation $g_{j} x_{i_{\beta+1}}$ of $g_{j}$. But we know that all prolongations involutively reduce to zero, so there must exist a monomial $u_{k}=x_{k_{1}} \ldots x_{k_{\gamma}} \in U$ such that $u_{k}$ involutively divides $u_{j} x_{i_{\beta+1}}$. Assuming that $x_{k_{\gamma}}=x_{i_{\kappa}}$, any candidate for $u_{k}$ must be a suffix of $u_{j} x_{i_{\beta+1}}$ (otherwise $T\left(u_{k}, x_{i_{k+1}}^{R}\right)=0$ because of the overlap between $u_{i}$ and $\left.u_{k}\right)$. Further, any candidate for $u_{k}$ cannot be either a suffix or a proper subword of $u_{i}$ (because of parts (1) and (2) of this proof). This leaves only one possibility for $u_{k}$, namely $u_{k}=u_{j} x_{i_{\beta+1}}$.

$$
\begin{aligned}
& u_{i}=\quad \overline{x_{i_{1}}} \overline{x_{i_{2}}}---\overline{x_{i_{\beta-1}}} \overline{x_{i_{\beta}}} \overline{x_{i_{\beta+1}}}---\overline{x_{i_{\alpha-1}}} \overline{x_{i_{\alpha}}} \\
& u_{j}=\quad \overline{x_{j_{1}}} \frac{x_{j_{2}}}{}--\frac{}{x_{j_{\beta-1}}} \frac{}{x_{j_{\beta}}} \\
& u_{k}=\quad \overline{x_{k_{1}}} \overline{x_{k_{2}}}---\frac{x_{k_{\gamma-2}}}{\overline{x_{k_{\gamma-1}}}} \overline{x_{k_{\gamma}}}
\end{aligned}
$$

If $\alpha=\beta+1$, then it is clear that $u_{k}=u_{i}$, and so the first step in the involutive reduction of the prolongation $g_{j} x_{i_{\alpha}}$ is to take away the multiple $\left(\frac{c_{t}}{c_{t}}\right) g_{i}$ of $g_{i}$ from $g_{j} x_{i_{\alpha}}$ to leave the polynomial $g_{j} x_{i_{\alpha}}-\left(\frac{c_{t}}{c_{t}^{t}}\right) g_{i}=-\left(\frac{1}{c_{t}^{t}}\right) s_{t}$. But as we know that all prolongations involutively reduce to zero, we can conclude that the S-polynomial $s_{t}$ conventionally reduces to zero.

Otherwise, if $\alpha>\beta+1$, we can now use the monomial $u_{k}$ together with Buchberger's Second Criterion to simplify our goal of showing that the S-polynomial $s_{t}$ reduces to zero. Notice that the monomial $u_{k}$ is a subword of the overlap word $u_{i}$ associated to $s_{t}$, and so in order to show that $s_{t}$ reduces to zero, all we have to do is to show that the two S-polynomials

$$
s_{u}=c_{u} g_{i}-c_{u}^{\prime} g_{k}\left(x_{i_{\beta+2}} \ldots x_{i_{\alpha}}\right)
$$

and

$$
s_{v}=c_{v} g_{k}-c_{v}^{\prime} g_{j} x_{i_{\beta+1}}
$$

reduce to zero $(1 \leqslant u, v \leqslant|S|)$.
The S-polynomial $s_{v}$ reduces to zero by comparison with part (2). For the S-polynomial $s_{u}$, we proceed by induction (we have another left overlap), eventually coming across a left overlap of 'type $\alpha=\beta+1$ ' because we move one letter at a time to the right after
each inductive step.

(4 and 5) In Definition 3.1.2, we defined a prefix overlap to be an overlap where, given two monomials $m_{1}$ and $m_{2}$ such that $\operatorname{deg}\left(m_{1}\right) \geqslant \operatorname{deg}\left(m_{2}\right)$, a prefix of $m_{1}$ is equal to a suffix of $m_{2}$; suffix overlaps were defined similarly. If we drop the condition on the degrees of the monomials, it is clear that every suffix overlap can be treated as a prefix overlap (by swapping the roles of $m_{1}$ and $m_{2}$ ); this allows us to deal with the case of a prefix overlap only.

Consider an arbitrary entry $s_{t} \in S(1 \leqslant t \leqslant|S|)$ corresponding to a prefix overlap where a prefix of the monomial $u_{i}$ is equal to a suffix of the monomial $u_{j}$. This means that $s_{t}=c_{t} \ell_{t} g_{i}-c_{t}^{\prime} g_{j} r_{t}^{\prime}$ for some $g_{i}, g_{j} \in G$, with overlap word $\ell_{t} u_{i}=u_{j} r_{t}^{\prime}$. Let $u_{i}=x_{i_{1}} \ldots x_{i_{\alpha}}$; let $u_{j}=x_{j_{1}} \ldots x_{j_{\beta}}$; and choose $D$ such that $x_{i_{D}}=x_{j_{\beta}}$.

$$
\begin{array}{ll}
u_{i}= \\
u_{j} & =\quad \frac{}{x_{i_{1}}}-\cdots \frac{x_{i_{D}}}{x_{j_{1}}}---\frac{x_{i_{D+1}}}{x_{j_{\beta-D}}} \frac{\overline{x_{j_{\beta-D+1}}}-\cdots \frac{x_{i_{\alpha-1}}}{} \frac{x_{i_{\alpha}}}{x_{j_{\beta}}}}{}
\end{array}
$$

By definition of $\mathcal{O}$, we must have $T\left(u_{j}, x_{i_{D+1}}^{R}\right)=0$.
Because we know that the prolongation $g_{j} x_{i_{D+1}}$ involutively reduces to zero, there must exist a monomial $u_{k}=x_{k_{1}} \ldots x_{k_{\gamma}} \in U$ such that $u_{k}$ involutively divides $u_{j} x_{i_{D+1}}$. This $u_{k}$ must be a suffix of $u_{j} x_{i_{D+1}}$ (otherwise, assuming that $x_{k_{\gamma}}=x_{j_{k}}$, we have $T\left(u_{k}, x_{i_{D+1}}^{R}\right)=0$ if $\kappa=\beta$ (because of the overlap between $u_{i}$ and $u_{k}$ ); and $T\left(u_{k}, x_{j_{\kappa+1}}^{R}\right)=0$ if $\kappa<\beta$ (because of the overlap between $u_{j}$ and $\left.u_{k}\right)$ ).

$$
\begin{array}{ll}
u_{i} & =\frac{\overline{x_{i_{1}}}}{}--\frac{}{x_{i_{D}}} \frac{x_{i_{D+1}}}{}--\frac{x_{i_{\alpha-1}}}{\frac{x_{i_{\alpha}}}{x_{i_{\alpha}}}} \\
u_{j} & =--\frac{}{x_{j_{1}}}---\frac{x_{j_{\beta-D}}}{x_{j_{\beta-D+1}}} \\
u_{k} &
\end{array}
$$

Let us now use the monomial $u_{k}$ together with Buchberger's Second Criterion to simplify our goal of showing that the S-polynomial $s_{t}$ reduces to zero. Because $u_{k}$ is a subword of the overlap word $\ell_{t} u_{i}$ associated to $s_{t}$, in order to show that $s_{t}$ reduces to zero, all we have to do is to show that the two S-polynomials

$$
s_{u}= \begin{cases}c_{u}\left(x_{k_{1}} \ldots x_{j_{\beta-D}}\right) g_{i}-c_{u}^{\prime} g_{k}\left(x_{i_{D+2}} \ldots x_{i_{\alpha}}\right) & \text { if } \gamma>D+1 \\ c_{u} g_{i}-c_{u}^{\prime} \ell_{u}^{\prime} g_{k}\left(x_{i_{D+2}} \ldots x_{i_{\alpha}}\right) & \text { if } \gamma \leqslant D+1\end{cases}
$$

and

$$
s_{v}=c_{v} g_{j} x_{i_{D+1}}-c_{v}^{\prime}\left(x_{j_{1}} \ldots x_{j_{\beta+1-\gamma}}\right) g_{k}
$$

reduce to zero $(1 \leqslant u, v \leqslant|S|)$.
The S-polynomial $s_{v}$ reduces to zero by comparison with part (2). For the S-polynomial $s_{u}$, first note that if $\alpha=D+1$, then either $u_{k}$ is a suffix of $u_{i}, u_{i}$ is a suffix of $u_{k}$, or $u_{k}=u_{i}$; it follows that $s_{u}$ reduces to zero trivially if $u_{k}=u_{i}$, and (by part (1)) $s_{u}$ (and hence $s_{t}$ ) cannot appear in $S$ in the other two cases.

If however $\alpha \neq D+1$, then either $s_{u}$ is a middle overlap (if $\gamma<D+1$ ), a left overlap (if $\gamma=D+1$ ), or another prefix overlap. The first case leads us to conclude that $s_{t}$ cannot appear in $S$; the second case is handled by part (3) of this proof; and the final case is handled by induction, where we note that after each step of the induction, the value $\alpha+\beta-2 D$ strictly decreases, so we are guaranteed at some stage to find an overlap that is not a prefix overlap, enabling us either to verify that the S-polynomial $s_{t}$ conventionally reduces to zero, or to conclude that $s_{t}$ can not in fact appear in $S$.

### 5.5.3 A Strong Local Division

Thus far, we have encountered two global divisions that are strong and continuous, and one local division that is weak, continuous and Gröbner. Our next division can be considered to be a hybrid of these previous divisions, as it will be a local division that is continuous and (as long as thick divisors are being used) strong.

Definition 5.5.17 (The Strong Left Overlap Division $\mathcal{S}$ ) Let $U=\left\{u_{1}, \ldots, u_{m}\right\}$ be a set of monomials. Assign multiplicative variables to $U$ according to Algorithm 15, which (in words) performs the following two tasks.
(a) Assign multiplicative variables to $U$ according to the left overlap division.
(b) Using the recipe provided in Algorithm 14, ensure that at least one variable in every monomial $u_{j} \in U$ is right nonmultiplicative for each monomial $u_{i} \in U$.

Remark 5.5.18 As Algorithm 15 expects any input set to be ordered with respect to DegRevLex, we may sometimes have to reorder a set of monomials $U$ to satisfy this condition before we can assign multiplicative variables to $U$ according to the strong left overlap division.

```
Algorithm 14 'DisjointCones' Function for Algorithm 15
Output: \(T\).
    for each monomial \(u_{i} \in U(m \geqslant i \geqslant 1)\) do
        for each monomial \(u_{j} \in U(m \geqslant j \geqslant 1)\) do
            Let \(u_{i}=x_{i_{1}} x_{i_{2}} \ldots x_{i_{\alpha}}\) and \(u_{j}=x_{j_{1}} x_{j_{2}} \ldots x_{j_{\beta}}\);
            found \(=\) false;
            \(k=1 ;\)
            while \((k \leqslant \beta)\) do
                if \(\left(T\left(u_{i}, x_{j_{k}}^{R}\right)=0\right)\) then
                    found \(=\) true;
                    \(k=\beta+1 ;\)
            else
                    \(k=k+1 ;\)
            end if
            end while
            if (found \(==\) false) then
            \(T\left(u_{i}, x_{j_{1}}^{R}\right)=0 ;\)
            end if
        end for
    end for
    return \(T\);
```

Input: A set of monomials $U=\left\{u_{1}, \ldots, u_{m}\right\}$ ordered by DegRevLex $\left(u_{1} \geqslant u_{2} \geqslant \cdots \geqslant\right.$
$u_{m}$ ), where $u_{i} \in R\left\langle x_{1}, \ldots, x_{n}\right\rangle$; a table $T$ of left and right multiplicative variables for
all $u_{i} \in U$, where each entry of $T$ is either 1 (multiplicative) or 0 (nonmultiplicative).

```
Algorithm 15 The Strong Left Overlap Division \(\mathcal{S}\)
    \(u_{m}\) ), where \(u_{i} \in R\left\langle x_{1}, \ldots, x_{n}\right\rangle\). entry of \(T\) is either 1 (multiplicative) or 0 (nonmultiplicative).
Create a table \(T\) of multiplicative variables as shown below:
\begin{tabular}{c|ccccccc} 
& \(x_{1}^{L}\) & \(x_{1}^{R}\) & \(x_{2}^{L}\) & \(x_{2}^{R}\) & \(\cdots\) & \(x_{n}^{L}\) & \(x_{n}^{R}\) \\
\hline\(u_{1}\) & 1 & 1 & 1 & 1 & \(\cdots\) & 1 & 1 \\
\(u_{2}\) & 1 & 1 & 1 & 1 & \(\cdots\) & 1 & 1 \\
\(\vdots\) & \(\vdots\) & \(\vdots\) & \(\vdots\) & \(\vdots\) & \(\ddots\) & \(\vdots\) & \(\vdots\) \\
\(u_{m}\) & 1 & 1 & 1 & 1 & \(\cdots\) & 1 & 1
\end{tabular}
for each monomial \(u_{i} \in U(1 \leqslant i \leqslant m)\) do
for each monomial \(u_{j} \in U(i \leqslant j \leqslant m)\) do
Let \(u_{i}=x_{i_{1}} x_{i_{2}} \ldots x_{i_{\alpha}}\) and \(u_{j}=x_{j_{1}} x_{j_{2}} \ldots x_{j_{\beta}}\);
if \((i \neq j)\) then
for each \(k(1 \leqslant k<\alpha-\beta+1)\) do
\[
\text { if }\left(\operatorname{Subword}\left(u_{i}, k, k+\beta-1\right)==u_{j}\right) \text { then }
\]
\[
T\left(u_{j}, x_{i_{k+\beta}}^{R}\right)=0
\]
end if
end for
end if
for each \(k(1 \leqslant k \leqslant \beta-1)\) do
if \(\left(\operatorname{Prefix}\left(u_{i}, k\right)==\operatorname{Suffix}\left(u_{j}, k\right)\right)\) then
\(T\left(u_{j}, x_{i_{k+1}}^{R}\right)=0 ;\)
end if
if \(\left(\operatorname{Suffix}\left(u_{i}, k\right)==\operatorname{Prefix}\left(u_{j}, k\right)\right)\) then
\(T\left(u_{i}, x_{j_{k+1}}^{R}\right)=0 ;\)
end if
end for
end for
end for
\(T=\operatorname{DisjointCones}(U, T) ; \quad\) (Algorithm 14)
return \(T\);
```

Input: A set of monomials $U=\left\{u_{1}, \ldots, u_{m}\right\}$ ordered by DegRevLex $\left(u_{1} \geqslant u_{2} \geqslant \cdots \geqslant\right.$

Output: A table $T$ of left and right multiplicative variables for all $u_{i} \in U$, where each

Proposition 5.5.19 The strong left overlap division is continuous.

Proof: We refer to the proof of Proposition 5.5.14, replacing $\mathcal{O}$ by $\mathcal{S}$.
Proposition 5.5.20 The strong left overlap division is a Gröbner involutive division.

Proof: We refer to the proof of Proposition 5.5.16, replacing $\mathcal{O}$ by $\mathcal{S}$.
Remark 5.5.21 Propositions 5.5.19 and 5.5.20 apply either when using thin divisors or when using thick divisors.

Proposition 5.5.22 With respect to thick divisors, the strong left overlap division is a strong involutive division.

Proof: To prove that the strong left overlap division is a strong involutive division, we need to show that the three conditions of Definition 5.1.6 hold.

## - Disjoint Cones Condition

Let $\mathcal{C}_{S}\left(u_{1}, U\right)$ and $\mathcal{C}_{S}\left(u_{2}, U\right)$ be the involutive cones associated to the monomials $u_{1}$ and $u_{2}$ over some noncommutative polynomial ring $\mathcal{R}$, where $\left\{u_{1}, u_{2}\right\} \subset U \subset \mathcal{R}$. If $\mathcal{C}_{\mathcal{S}}\left(u_{1}, U\right) \cap \mathcal{C}_{\mathcal{S}}\left(u_{2}, U\right) \neq \emptyset$, then there must be some monomial $v \in \mathcal{R}$ such that $v$ contains both monomials $u_{1}$ and $u_{2}$ as subwords, and (as placed in $v$ ) both $u_{1}$ and $u_{2}$ must be involutive divisors of $v$. By definition of $\mathcal{S}$, both $u_{1}$ and $u_{2}$ must be suffices of $v$. Thus, assuming (without loss of generality) that $\operatorname{deg}\left(u_{1}\right)>\operatorname{deg}\left(u_{2}\right)$, we are able to draw the following diagram summarising the situation.


For $\mathcal{S}$ to be strong, we must have $\mathcal{C}_{\mathcal{S}}\left(u_{1}, U\right) \subset \mathcal{C}_{\mathcal{S}}\left(u_{2}, U\right)$ (it is clear that $\mathcal{C}_{\mathcal{S}}\left(u_{2}, U\right) \not \subset$ $\mathcal{C}_{\mathcal{S}}\left(u_{1}, U\right)$ because $\left.u_{2} \notin \mathcal{C}_{\mathcal{S}}\left(u_{1}, U\right)\right)$. This can be verified by proving that a variable is right nonmultiplicative for $u_{1}$ if and only if it is right nonmultiplicative for $u_{2}$.
$(\Rightarrow)$ If an arbitrary variable $x$ is right nonmultiplicative for $u_{2}$, then either some monomial $u \in U$ overlaps with $u_{2}$ in one of the ways shown below (where the variable immediately to the right of $u_{2}$ is the variable $x$ ), or $x$ was assigned right
nonmultiplicative for $u_{2}$ in order to ensure that some variable in some monomial $u \in U$ is right nonmultiplicative for $u_{2}$.

Overlap (i) Overlap (ii)


If the former case applies, then it is clear that for both overlap types there will be another overlap between $u_{1}$ and $u$ that will lead $\mathcal{S}$ to assign $x$ to be right nonmultiplicative for $u_{1}$. It follows that after we have assigned multiplicative variables to $U$ according to the left overlap division (which we recall is the first step of assigning multiplicative variables to $U$ according to $\mathcal{S}$ ), the right multiplicative variables of $u_{1}$ and $u_{2}$ will be identical. It therefore remains to show that if $x$ is assigned right nonmultiplicative for $u_{2}$ in the latter case (which will happen during the final step of assigning multiplicative variables to $U$ according to $\mathcal{S}$ ), then $x$ is also assigned right nonmultiplicative for $u_{1}$. But this is clear when we consider that Algorithm 14 is used to perform this final step, because for $u_{1}$ and $u_{2}$ in Algorithm 14, we will always analyse each monomial in $U$ in the same order.
$(\Leftarrow)$ Use the same argument as above, replacing $u_{1}$ by $u_{2}$ and vice-versa.

## - Unique Divisor Condition

Given a monomial $u$ belonging to a set of monomials $U, u$ may not involutively divide an arbitrary monomial $v$ in more than one way (and hence the Unique Divisor condition is satisfied) because (i) $\mathcal{S}$ ensures that no overlap word involving only $u$ is involutively divisible in more than one way by $u$; and (ii) $\mathcal{S}$ ensures that at least one variable in $u$ is right nonmultiplicative for $u$, so that if $u$ appears twice in $v$ as subwords that are disjoint from one another, then only the 'right-most' subword can potentially be an involutive divisor of $v$.

## - Subset Condition

Let $v$ be a monomial belonging to a set $V$ of monomials, where $V$ itself is a subset of a larger set $U$ of monomials. Because $\mathcal{S}$ assigns no left nonmultiplicative variables, it is clear that $\mathcal{M}_{\mathcal{S}}^{L}(v, U) \subseteq \mathcal{M}_{S}^{L}(v, V)$. To prove that $\mathcal{M}_{\mathcal{S}}^{R}(v, U) \subseteq \mathcal{M}_{S}^{R}(v, V)$, note that if a variable $x$ is right nonmultiplicative for $v$ with respect to $U$ and $\mathcal{S}$ (so that $x \notin \mathcal{M}_{S}^{R}(v, U)$ ), then (as in the proof for the Disjoint Cones Condition) either some monomial $u \in U$ overlaps with $v$ in one of the ways shown below (where the
variable immediately to the right of $v$ is the variable $x$ ), or $x$ was assigned right nonmultiplicative for $v$ in order to ensure that some variable in some monomial $u \in U$ is right nonmultiplicative for $v$.
Overlap (i)
Overlap (ii)


In both cases, it is clear that, with respect to the set $V$, the variable $x$ may not be assigned right nonmultiplicative for $v$ if $u \notin V$, so that $\mathcal{M}_{\mathcal{S}}^{R}(v, U) \subseteq \mathcal{M}_{\mathcal{S}}^{R}(v, V)$ as required.

Proposition 5.5.23 With respect to thin divisors, the strong left overlap division is a weak involutive division.

Proof: Let $U:=\{x y\}$ be a set of monomials over the polynomial ring $\mathbb{Q}\langle x, y\rangle$. Here are the multiplicative variables for $U$ with respect to the strong left overlap division $\mathcal{S}$.

| $u$ | $\mathcal{M}_{\mathcal{S}}^{L}(u, U)$ | $\mathcal{M}_{\mathcal{S}}^{R}(u, U)$ |
| :---: | :---: | :---: |
| $x y$ | $\{x, y\}$ | $\{y\}$ |

For $\mathcal{S}$ to be strong with respect to thin divisors, the monomial $x y^{2} x y$, which is conventionally divisible by $x y$ in two ways, must only be involutively divisible by $x y$ in one way (this is the Unique Divisor condition of Definition 5.1.6). However it is clear that $x y^{2} x y$ is involutively divisible by $x y$ in two ways with respect to thin divisors, so $\mathcal{S}$ must be a weak involutive division with respect to thin divisors.

Example 5.5.24 Continuing Examples 5.5.9 and 5.5.12, here are the multiplicative variables for the set $\mathrm{LM}(F)$ of monomials with respect to the strong left overlap division $\mathcal{S}$, where we recall that $F:=\left\{x y-z, x+z, y z-z, x z, z y+z, z^{2}\right\}$.

| $u$ | $\mathcal{M}_{S}^{L}(u, \operatorname{LM}(F))$ | $\mathcal{M}_{S}^{R}(u, \operatorname{LM}(F))$ |
| :---: | :---: | :---: |
| $x y$ | $\{x, y, z\}$ | $\{y\}$ |
| $x$ | $\{x, y, z\}$ | $\emptyset$ |
| $y z$ | $\{x, y, z\}$ | $\emptyset$ |
| $x z$ | $\{x, y, z\}$ | $\emptyset$ |
| $z y$ | $\{x, y, z\}$ | $\{y\}$ |
| $z^{2}$ | $\{x, y, z\}$ | $\emptyset$ |

When we apply Algorithm 12 to $F$ with respect to the DegLex monomial ordering, thick divisors and the strong left overlap division, $F$ (as in Example 5.5.12) is returned to us as the output Locally Involutive Basis.

Remark 5.5.25 In the above example, even though we know that $\mathcal{S}$ is continuous, we cannot deduce that the Locally Involutive Basis $F$ is an Involutive Basis because we are using thick divisors (Proposition 5.4.3 does not apply in the case of using thick divisors).

What this means is that the involutive cones of $F$ (and in general any Locally Involutive Basis with respect to $\mathcal{S}$ and thick divisors) will be disjoint (because $\mathcal{S}$ is strong), but will not necessarily completely cover the conventional cones of $F$, so that some monomials that are conventionally reducible by $F$ may not be involutively reducible by $F$. It follows that when involutively reducing a polynomial with respect to $F$, the reduction path will be unique but the correct remainder may not always be obtained (in the sense that some of the terms in our 'remainder' may still be conventionally reducible by members of $F$ ). One remedy to this problem would be to involutively reduce a polynomial $p$ with respect to $F$ to obtain a remainder $r$, and then to conventionally reduce $r$ with respect to $F$ to obtain a remainder $r^{\prime}$ which we can be sure contains no term that is conventionally reducible by $F$.

Let us now summarise (with respect to thin divisors) the properties of the involutive divisions we have encountered so far, where we note that any strong and continuous involutive division is by default a Gröbner involutive division.

| Division | Continuous | Strong | Gröbner |
| :--- | :---: | :---: | :---: |
| Left | Yes | Yes | Yes |
| Right | Yes | Yes | Yes |
| Left Overlap | Yes | No | Yes |
| Right Overlap | Yes | No | Yes |
| Strong Left Overlap | Yes | No | Yes |
| Strong Right Overlap | Yes | No | Yes |

There is a balance to be struck between choosing an involutive division with nice theoretical properties and an involutive division which is of practical use, which is to say that it is more likely to terminate compared to other divisions. To this end, one suggestion would be to try to compute an Involutive Basis with respect to the left or right divisions to begin with (as they are easily defined and involutive reduction with respect to these divisions is very efficient); otherwise to try one of the 'overlap' divisions, choosing a strong overlap division if it is important to obtain disjoint involutive cones.

It is also worth mentioning that for all the divisions we have encountered so far, if Algorithm 12 terminates then it does so with a noncommutative Gröbner Basis, which means that Algorithm 12 can be thought of as an alternative algorithm for computing noncommutative Gröbner Bases. Whether this method is more or less efficient than computing noncommutative Gröbner Bases using Algorithm 5 is a matter for further discussion.

### 5.5.4 Alternative Divisions

Having encountered three different types of involutive division so far (each of which has two variants - left and right), let us now consider if there are any other involutive divisions with some useful properties, starting by thinking of global divisions.

## Alternative Global Divisions

Open Question 2 Apart from the empty, left and right divisions, are there any other global involutive divisions of the following types:
(a) strong and continuous;
(b) weak, continuous and Gröbner?

Remark 5.5.26 It seems unlikely that a global division will exist that affirmatively answers Open Question 2 and does not either assign all variables to be left nonmultiplicative or all right nonmultiplicative (thus refining the right or left divisions respectively). The reason for saying this is because the moment you have one variable being left multiplicative and another variable being right multiplicative for the same monomial globally, then you risk not being able to prove that your division is strong; similarly the moment you have one variable being left nonmultiplicative and another variable being right nonmultiplicative for the same monomial globally, then you risk not being able to prove that your division is continuous.

## Alternative Local Divisions

So far, all the local divisions we have considered have assigned all variables to be multiplicative on one side, and have chosen certain variables to be nonmultiplicative on the other side. Let us now consider a local division that modifies the left overlap division by assigning some variables to be nonmultiplicative on both left and right hand sides.

Definition 5.5.27 (The Two-Sided Left Overlap Division $\mathcal{W}$ ) Consider a set $U=$ $\left\{u_{1}, \ldots, u_{m}\right\}$ of monomials, where all variables are assumed to be left and right multiplicative for all elements of $U$ to begin with. Assign multiplicative variables to $U$ according to Algorithm 16, which (in words) performs the following tasks.
(a) For all possible ways that a monomial $u_{j} \in U$ is a subword of a (different) monomial $u_{i} \in U$, so that

$$
\operatorname{Subword}\left(u_{i}, k, k+\operatorname{deg}\left(u_{j}\right)-1\right)=u_{j}
$$

for some integer $k$, assign the variable $\operatorname{Subword}\left(u_{i}, k-1, k-1\right)$ to be left nonmultiplicative for $u_{j}$ if $u_{j}$ is a suffix of $u_{i}$; and assign the variable $\operatorname{Subword}\left(u_{i}, k+\operatorname{deg}\left(u_{j}\right), k+\right.$ $\left.\operatorname{deg}\left(u_{j}\right)\right)$ to be right nonmultiplicative for $u_{j}$ if $u_{j}$ is not a suffix of $u_{i}$.
(b) For all possible ways that a proper prefix of a monomial $u_{i} \in U$ is equal to a proper suffix of a (not necessarily different) monomial $u_{j} \in U$, so that

$$
\operatorname{Prefix}\left(u_{i}, k\right)=\operatorname{Suffix}\left(u_{j}, k\right)
$$

for some integer $k$ and $u_{i}$ is not a subword of $u_{j}$ or vice-versa, use the recipe provided in the second half of Algorithm 16 to ensure that at least one of the following conditions
are satisfied: (i) the variable $\operatorname{Subword}\left(u_{i}, k+1, k+1\right)$ is right nonmultiplicative for $u_{j}$; (ii) the variable $\operatorname{Subword}\left(u_{j}, \operatorname{deg}\left(u_{j}\right)-k, \operatorname{deg}\left(u_{j}\right)-k\right)$ is left nonmultiplicative for $u_{i}$.

Remark 5.5.28 For task (b) above, Algorithm 16 gives preference to monomials which are greater in the DegRevLex monomial ordering (given the choice, it always assigns a nonmultiplicative variable to whichever monomial out of $u_{i}$ and $u_{j}$ is the smallest); it also attempts to minimise the number of variables made nonmultiplicative by only assigning a variable to be nonmultiplicative if both the variables $\operatorname{Subword}\left(u_{i}, k+1, k+1\right)$ and $\operatorname{Subword}\left(u_{j}, \operatorname{deg}\left(u_{j}\right)-k, \operatorname{deg}\left(u_{j}\right)-k\right)$ are respectively right multiplicative and left multiplicative. These refinements will become crucial when proving the continuity of the division.

Example 5.5.29 Consider the set of monomials $U:=\left\{z x^{2} y x y, y z x, x y\right\}$ over the polynomial ring $\mathbb{Q}\langle x, y, z\rangle$. Here are the left and right multiplicative variables for $U$ with respect to the two-sided left overlap division $\mathcal{W}$.

| $u$ | $\mathcal{M}_{\mathcal{W}}^{L}(u, U)$ | $\mathcal{M}_{\mathcal{W}}^{R}(u, U)$ |
| :---: | :---: | :---: |
| $z x^{2} y x y$ | $\{x, y, z\}$ | $\{x, y, z\}$ |
| $y z x$ | $\{y, z\}$ | $\{y, z\}$ |
| $x y$ | $\{x\}$ | $\{y, z\}$ |

The above table is constructed from the table $T$ shown below, a table which is obtained by applying Algorithm 16 to $U$.

| Monomial | $x^{L}$ | $x^{R}$ | $y^{L}$ | $y^{R}$ | $z^{L}$ | $z^{R}$ |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| $z x^{2} y x y$ | 1 | 1 | 1 | 1 | 1 | 1 |
| $y z x$ | 0 | 0 | 1 | 1 | 1 | 1 |
| $x y$ | 1 | 0 | 0 | 1 | 0 | 1 |

The zero entries in $T$ correspond to the following overlaps between the elements of $U$ (presented in the order in which Algorithm 16 encounters them).

```
Algorithm 16 The Two-Sided Left Overlap Division \(\mathcal{W}\)
Input: A set of monomials \(U=\left\{u_{1}, \ldots, u_{m}\right\}\) ordered by DegRevLex \(\left(u_{1} \geqslant u_{2} \geqslant \cdots \geqslant\right.\)
    \(u_{m}\) ), where \(u_{i} \in R\left\langle x_{1}, \ldots, x_{n}\right\rangle\).
Output: A table \(T\) of left and right multiplicative variables for all \(u_{i} \in U\), where each entry of \(T\) is either 1 (multiplicative) or 0 (nonmultiplicative).
Create a table \(T\) of multiplicative variables as shown below:
\begin{tabular}{c|ccccccc} 
& \(x_{1}^{L}\) & \(x_{1}^{R}\) & \(x_{2}^{L}\) & \(x_{2}^{R}\) & \(\cdots\) & \(x_{n}^{L}\) & \(x_{n}^{R}\) \\
\hline\(u_{1}\) & 1 & 1 & 1 & 1 & \(\cdots\) & 1 & 1 \\
\(\vdots\) & \(\vdots\) & \(\vdots\) & \(\vdots\) & \(\vdots\) & \(\ddots\) & \(\vdots\) & \(\vdots\) \\
\(u_{m}\) & 1 & 1 & 1 & 1 & \(\cdots\) & 1 & 1
\end{tabular}
for each monomial \(u_{i} \in U(1 \leqslant i \leqslant m)\) do
for each monomial \(u_{j} \in U(i \leqslant j \leqslant m)\) do
Let \(u_{i}=x_{i_{1}} x_{i_{2}} \ldots x_{i_{\alpha}}\) and \(u_{j}=x_{j_{1}} x_{j_{2}} \ldots x_{j_{\beta}}\);
if \((i \neq j)\) then
for each \(k(1 \leqslant k \leqslant \alpha-\beta+1)\) do
if \(\left(\operatorname{Subword}\left(u_{i}, k, k+\beta-1\right)==u_{j}\right)\) then
if \((k<\alpha-\beta+1)\) then \(T\left(u_{j}, x_{i_{k+\beta}}^{R}\right)=0\);
else \(T\left(u_{j}, x_{i_{k-1}}^{L}\right)=0\);
end if
end if
end for
end if
for each \(k(1 \leqslant k \leqslant \beta-1)\) do
if \(\left(\operatorname{Prefix}\left(u_{i}, k\right)==\operatorname{Suffix}\left(u_{j}, k\right)\right)\) then
if \(\left(T\left(u_{i}, x_{j_{\beta-k}}^{L}\right)+T\left(u_{j}, x_{i_{k+1}}^{R}\right)==2\right)\) then \(T\left(u_{j}, x_{i_{k+1}}^{R}\right)=0\);
end if
end if
if \(\left(\operatorname{Suffix}\left(u_{i}, k\right)==\operatorname{Prefix}\left(u_{j}, k\right)\right)\) then
if \(\left(T\left(u_{i}, x_{j_{k+1}}^{R}\right)+T\left(u_{j}, x_{i_{\alpha-k}}^{L}\right)==2\right)\) then \(T\left(u_{j}, x_{i_{\alpha-k}}^{L}\right)=0\);
end if
end if
end for
end for
end for
return \(T\);
```

| Table Entry | Overlap |
| :---: | :---: |
| $T\left(y z x, x^{R}\right)$ | $\operatorname{Prefix}\left(z x^{2} y x y, 2\right)=\operatorname{Suffix}(y z x, 2)$ |
| $T\left(y z x, x^{L}\right)$ | $\operatorname{Suffix}\left(z x^{2} y x y, 1\right)=\operatorname{Prefix}(y z x, 1)$ |
| $T\left(x y, x^{R}\right)$ | $\operatorname{Subword}\left(z x^{2} y x y, 3,4\right)=x y$ |
| $T\left(x y, y^{L}\right)$ | Subword $\left(z x^{2} y x y, 5,6\right)=x y$ |
| $T\left(x y, z^{L}\right)$ | $\operatorname{Suffix}(y z x, 1)=\operatorname{Prefix}(x y, 1)$ |

Notice that the overlap Prefix $(y z x, 1)=\operatorname{Suffix}(x y, 1)$ does not produce a zero entry for $T\left(x y, z^{R}\right)$, as by the time that we encounter this overlap in the algorithm, we have already assigned $T\left(y z x, x^{L}\right)=0$.

Proposition 5.5.30 The two-sided left overlap division $\mathcal{W}$ is a weak involutive division.

Proof: We refer to the proof of Proposition 5.5.15, making the obvious changes (for example replacing $\mathcal{O}$ by $\mathcal{W}$ ).

For the following two propositions, we defer their proofs to Appendix A due to their length and technical nature.

Proposition 5.5.31 The two-sided left overlap division $\mathcal{W}$ is continuous.

Proof: We refer to Appendix A.
Proposition 5.5.32 The two-sided left overlap division $\mathcal{W}$ is a Gröbner involutive division.

Proof: We refer to Appendix A, noting that the proof is similar to the proof of Proposition 5.5.16.

Remark 5.5.33 Because a variable is sometimes only assigned nonmultiplicative if two other variables are multiplicative in Algorithm 16, the subset condition of Definition 5.1.6 will not always be satisfied with respect to the two-sided left overlap division. This will still hold true even if we apply Algorithm 14 at the end of Algorithm 16, which means that the two-sided left overlap division cannot be converted to give a strong involutive division in the same way that we converted the left overlap division to give the strong left overlap division.

To finish this section, let us now consider some further variations of the left overlap division, variations that will allow us to assign more multiplicative variables than the left overlap division (and hence potentially have to deal with fewer prolongations when using Algorithm 12), but variations that cannot be modified to give strong involutive divisions in the same way that the left overlap division was modified to give the strong left overlap division (this is because there are other ways beside a monomial being a suffix of another monomial that two involutive cones can be non-disjoint with respect to these modified divisions).

Definition 5.5.34 (The Prefix-Only Left Overlap Division) Let $U=\left\{u_{1}, \ldots, u_{m}\right\}$ be a set of monomials, and assume that all variables are left and right multiplicative for all elements of $U$ to begin with.
(a) For all possible ways that a monomial $u_{j} \in U$ is a proper prefix of a monomial $u_{i} \in U$, assign the variable $\operatorname{Subword}\left(u_{i}, \operatorname{deg}\left(u_{j}\right)+1, \operatorname{deg}\left(u_{j}\right)+1\right)$ to be right nonmultiplicative for $u_{j}$.
(b) For all possible ways that a proper prefix of a monomial $u_{i} \in U$ is equal to a proper suffix of a (not necessarily different) monomial $u_{j} \in U$, so that

$$
\operatorname{Prefix}\left(u_{i}, k\right)=\operatorname{Suffix}\left(u_{j}, k\right)
$$

for some integer $k$ and $u_{i}$ is not a subword of $u_{j}$ or vice-versa, assign the variable Subword ( $u_{i}, k+1, k+1$ ) to be right nonmultiplicative for $u_{j}$.

Definition 5.5.35 (The Subword-Free Left Overlap Division) Consider a set $U=$ $\left\{u_{1}, \ldots, u_{m}\right\}$ of monomials, where all variables are assumed to be left and right multiplicative for all elements of $U$ to begin with.

For all possible ways that a proper prefix of a monomial $u_{i} \in U$ is equal to a proper suffix of a (not necessarily different) monomial $u_{j} \in U$, so that

$$
\operatorname{Prefix}\left(u_{i}, k\right)=\operatorname{Suffix}\left(u_{j}, k\right)
$$

for some integer $k$ and $u_{i}$ is not a subword of $u_{j}$ or vice-versa, assign the variable Subword ( $u_{i}, k+1, k+1$ ) to be right nonmultiplicative for $u_{j}$.

Proposition 5.5.36 Both the prefix-only left overlap and the subword-free left overlap divisions are continuous, weak and Gröbner.

Proof: We leave these proofs as exercises for the interested reader, noting that the proofs will be based on (and in some cases will be identical to) the proofs of Propositions 5.5.14, 5.5.15 and 5.5.16 respectively.

Remark 5.5.37 To help distinguish between the different types of overlap division we have encountered in this chapter, let us now give the following table showing which types of overlap each overlap division considers.


### 5.6 Termination

Given a basis $F$ generating an ideal over a noncommutative polynomial ring $\mathcal{R}$, does there exist a finite Involutive Basis for $F$ with respect to some admissible monomial ordering $O$ and some involutive division $I$ ? Unlike the commutative case, where the answer to the corresponding question (for certain divisions) is always 'Yes', the answer to this question can potentially be 'No', as if the noncommutative Gröbner Basis for $F$ with respect to $O$ is infinite, then the noncommutative Involutive Basis algorithm will not find a finite Involutive Basis for $F$ with respect to $I$ and $O$, as it will in effect be trying to compute the same infinite Gröbner Basis.

However, a valid follow-up question would be to ask whether the noncommutative Involutive Basis algorithm will terminate in the case that the noncommutative Gröbner Basis algorithm terminates. In Section 5.4, we defined a property of noncommutative involutive divisions (conclusivity) that ensures, when satisfied, that the answer to this secondary question is always 'Yes'. Despite this, we will not prove in this thesis that any of the divisions we have defined are conclusive. Instead, we leave the following open question for further investigation.

Open Question 3 Are there any conclusive noncommutative involutive divisions that are also continuous and either strong or Gröbner?

To obtain an affirmative answer to the above question, one approach may be to start by finding a proof for the following conjecture.

Conjecture 5.6.1 Let $O$ be an arbitrary admissible monomial ordering, and let $I$ be an arbitrary involutive division that is continuous and either strong or Gröbner. When computing an Involutive Basis for some basis $F$ with respect to $O$ and I using Algorithm 12, if $F$ possesses a finite unique reduced Gröbner Basis $G$ with respect to $O$, then after a finite number of steps of Algorithm 12, $\mathrm{LM}(G)$ appears as a subset of the set of leading monomials of the current basis.

To prove that a particular involutive division is conclusive, we would then need to show that once $\mathrm{LM}(G)$ appears as a subset of the set of leading monomials of the current basis, then the noncommutative Involutive Basis algorithm terminates (either immediately or in a finite number of steps), thus providing the required finite noncommutative Involutive Basis for $F$.

### 5.7 Examples

### 5.7.1 A Worked Example

Example 5.7.1 Let $F:=\left\{f_{1}, f_{2}\right\}=\left\{x^{2} y^{2}-2 x y^{2}+x^{2}, x^{2} y-2 x y\right\}$ be a basis for an ideal $J$ over the polynomial ring $\mathbb{Q}\langle x, y\rangle$, and let the monomial ordering be DegLex. Let us now compute a Locally Involutive Basis for $F$ with respect to the strong left overlap division $\mathcal{S}$ and thick divisors using Algorithm 12.

To begin with, we must autoreduce the input set $F$. This leaves the set unchanged, as we can verify by using the following table of multiplicative variables (obtained by using Algorithm 15), where $y$ is right nonmultiplicative for $f_{2}$ because of the overlap $\operatorname{LM}\left(f_{2}\right)=\operatorname{Subword}\left(\operatorname{LM}\left(f_{1}\right), 1,3\right)$; and $x$ is right nonmultiplicative for $f_{1}$ because we need to have a variable in $\operatorname{LM}\left(f_{2}\right)$ being right nonmultiplicative for $f_{1}$.

| Polynomial | $\mathcal{M}_{S}^{L}\left(f_{i}, F\right)$ | $\mathcal{M}_{S}^{R}\left(f_{i}, F\right)$ |
| :---: | :---: | :---: |
| $f_{1}=x^{2} y^{2}-2 x y^{2}+x^{2}$ | $\{x, y\}$ | $\{y\}$ |
| $f_{2}=x^{2} y-2 x y$ | $\{x, y\}$ | $\{x\}$ |

The above table also provides us with the set $S=\left\{f_{1} x, f_{2} y\right\}=\left\{x^{2} y^{2} x-2 x y^{2} x+x^{3}, x^{2} y^{2}-\right.$ $\left.2 x y^{2}\right\}$ of prolongations that is required for the next step of the algorithm. As $x^{2} y^{2}<x^{2} y^{2} x$ in the DegLex monomial ordering, we involutively reduce the element $f_{2} y \in S$ first.

$$
\begin{aligned}
f_{2} y=x^{2} y^{2}-2 x y^{2} & {\underset{s}{f}}_{f_{1}} \\
& x^{2} y^{2}-2 x y^{2}-\left(x^{2} y^{2}-2 x y^{2}+x^{2}\right) \\
& =-x^{2} .
\end{aligned}
$$

As the prolongation did not involutively reduce to zero, we now exit from the second while loop of Algorithm 12 and proceed by autoreducing the set $F \cup\left\{f_{3}:=-x^{2}\right\}=$ $\left\{x^{2} y^{2}-2 x y^{2}+x^{2}, x^{2} y-2 x y,-x^{2}\right\}$.

| Polynomial | $\mathcal{M}_{\mathcal{S}}^{L}\left(f_{i}, F\right)$ | $\mathcal{M}_{\mathcal{S}}^{R}\left(f_{i}, F\right)$ |
| :---: | :---: | :---: |
| $f_{1}=x^{2} y^{2}-2 x y^{2}+x^{2}$ | $\{x, y\}$ | $\{y\}$ |
| $f_{2}=x^{2} y-2 x y$ | $\{x, y\}$ | $\emptyset$ |
| $f_{3}=-x^{2}$ | $\{x, y\}$ | $\emptyset$ |

This process involutively reduces the third term of $f_{1}$ using $f_{3}$, leaving the new set $\left\{f_{4}:=x^{2} y^{2}-2 x y^{2}, f_{2}, f_{3}\right\}$ whose multiplicative variables are identical to the multiplicative variables of the set $\left\{f_{1}, f_{2}, f_{3}\right\}$ shown above.

Next, we construct the set $S=\left\{f_{4} x, f_{2} x, f_{2} y, f_{3} x, f_{3} y\right\}$ of prolongations, processing the element $f_{3} y$ first.

$$
\begin{array}{rll}
f_{3} y=-x^{2} y & {\underset{\mathcal{s}}{f_{2}}} & -x^{2} y+\left(x^{2} y-2 x y\right) \\
& = & -2 x y .
\end{array}
$$

Again the prolongation did not involutively reduce to zero, so we add the involutively reduced prolongation to our basis to obtain the set $\left\{f_{4}, f_{2}, f_{3}, f_{5}:=-2 x y\right\}$.

| Polynomial | $\mathcal{M}_{\mathcal{S}}^{L}\left(f_{i}, F\right)$ | $\mathcal{M}_{\mathcal{S}}^{R}\left(f_{i}, F\right)$ |
| :---: | :---: | :---: |
| $f_{4}=x^{2} y^{2}-2 x y^{2}$ | $\{x, y\}$ | $\{y\}$ |
| $f_{2}=x^{2} y-2 x y$ | $\{x, y\}$ | $\emptyset$ |
| $f_{3}=-x^{2}$ | $\{x, y\}$ | $\emptyset$ |
| $f_{5}=-2 x y$ | $\{x, y\}$ | $\emptyset$ |

This time during autoreduction, the polynomial $f_{2}$ involutively reduces to zero with respect to the set $\left\{f_{4}, f_{3}, f_{5}\right\}$ :

$$
\begin{array}{rll}
f_{2}=x^{2} y-2 x y & \mathcal{S}_{f_{5}} & x^{2} y-2 x y+\frac{1}{2} x(-2 x y) \\
& = & -2 x y \\
& {\underset{\mathcal{S}}{f_{5}}} & -2 x y-(-2 x y) \\
& = & 0 .
\end{array}
$$

This leaves us with the set $\left\{f_{4}, f_{3}, f_{5}\right\}$ after autoreduction is complete.

| Polynomial | $\mathcal{M}_{\mathcal{S}}^{L}\left(f_{i}, F\right)$ | $\mathcal{M}_{\mathcal{S}}^{R}\left(f_{i}, F\right)$ |
| :---: | :---: | :---: |
| $f_{4}=x^{2} y^{2}-2 x y^{2}$ | $\{x, y\}$ | $\{y\}$ |
| $f_{3}=-x^{2}$ | $\{x, y\}$ | $\emptyset$ |
| $f_{5}=-2 x y$ | $\{x, y\}$ | $\emptyset$ |

The next step is to construct the set $S=\left\{f_{4} x, f_{3} x, f_{3} y, f_{5} x, f_{5} y\right\}$ of prolongations, from which the element $f_{5} y$ is processed first.

$$
f_{5} y=-2 x y^{2}=: f_{6} .
$$

When the set $\left\{f_{4}, f_{3}, f_{5}, f_{6}\right\}$ is autoreduced, the polynomial $f_{4}$ now involutively reduces to zero, leaving us with the autoreduced set $\left\{f_{3}, f_{5}, f_{6}\right\}=\left\{-x^{2},-2 x y,-2 x y^{2}\right\}$.

| Polynomial | $\mathcal{M}_{\mathcal{S}}^{L}\left(f_{i}, F\right)$ | $\mathcal{M}_{\mathcal{S}}^{R}\left(f_{i}, F\right)$ |
| :---: | :---: | :---: |
| $f_{3}=-x^{2}$ | $\{x, y\}$ | $\emptyset$ |
| $f_{5}=-2 x y$ | $\{x, y\}$ | $\emptyset$ |
| $f_{6}=-2 x y^{2}$ | $\{x, y\}$ | $\{y\}$ |

Our next task is to process the elements of the set $S=\left\{f_{3} x, f_{3} y, f_{5} x, f_{5} y, f_{6} x\right\}$ of prolongations. The first element $f_{5} y$ we pick from $S$ involutively reduces to zero, but the second element $f_{5} x$ does not:

$$
\begin{array}{rll}
f_{5} y=-2 x y^{2} & \vec{s}_{f_{6}} & -2 x y^{2}-\left(-2 x y^{2}\right) \\
& = & 0 ; \\
f_{5} x=-2 x y x & =: & f_{7} .
\end{array}
$$

After constructing the set $\left\{f_{3}, f_{5}, f_{6}, f_{7}\right\}$, autoreduction does not alter the contents of the set, leaving us to construct our next set of prolongations from the following table of multiplicative variables.

| Polynomial | $\mathcal{M}_{\mathcal{S}}^{L}\left(f_{i}, F\right)$ | $\mathcal{M}_{\mathcal{S}}^{R}\left(f_{i}, F\right)$ |
| :---: | :---: | :---: |
| $f_{3}=-x^{2}$ | $\{x, y\}$ | $\emptyset$ |
| $f_{5}=-2 x y$ | $\{x, y\}$ | $\emptyset$ |
| $f_{6}=-2 x y^{2}$ | $\{x, y\}$ | $\{y\}$ |
| $f_{7}=-2 x y x$ | $\{x, y\}$ | $\emptyset$ |

Whilst processing this ( 7 element) set of prolongations, we add the involutively irreducible prolongation $f_{6} x=-2 x y^{2} x=: f_{8}$ to our basis to give a five element set which in unaffected by autoreduction.

| Polynomial | $\mathcal{M}_{\mathcal{S}}^{L}\left(f_{i}, F\right)$ | $\mathcal{M}_{\mathcal{S}}^{R}\left(f_{i}, F\right)$ |
| :---: | :---: | :---: |
| $f_{3}=-x^{2}$ | $\{x, y\}$ | $\emptyset$ |
| $f_{5}=-2 x y$ | $\{x, y\}$ | $\emptyset$ |
| $f_{6}=-2 x y^{2}$ | $\{x, y\}$ | $\{y\}$ |
| $f_{7}=-2 x y x$ | $\{x, y\}$ | $\emptyset$ |
| $f_{8}=-2 x y^{2} x$ | $\{x, y\}$ | $\emptyset$ |

To finish, we analyse the elements of the set

$$
S=\left\{f_{3} x, f_{3} y, f_{5} x, f_{5} y, f_{6} x, f_{7} x, f_{7} y, f_{8} x, f_{8} y\right\}
$$

of prolongations in the order $f_{5} y, f_{5} x, f_{3} y, f_{3} x, f_{6} x, f_{7} y, f_{7} x, f_{8} y, f_{8} x$.

$$
\begin{array}{rll}
f_{5} y=-2 x y^{2} & {\underset{\mathcal{O}}{f}} & -2 x y^{2}-\left(-2 x y^{2}\right) \\
& = & 0 ; \\
& \vdots & \\
f_{8} x=-2 x y^{2} x^{2} & {\underset{\mathcal{O}}{f}} & -2 x y^{2} x^{2}-2 x y^{2}\left(-x^{2}\right) \\
& =0 .
\end{array}
$$

Because all prolongations involutively reduce to zero (and hence $S=\emptyset$ ), the algorithm now terminates with the Involutive Basis

$$
G:=\left\{-x^{2},-2 x y,-2 x y^{2},-2 x y x,-2 x y^{2} x\right\}
$$

as output, a basis which can be visualised by looking at the following (partial) involutive monomial lattice for $G$.
$y$


For comparison, the (partial) monomial lattice of the reduced DegLex Gröbner Basis $H$
for $F$ is shown below, where $H:=\left\{x^{2}, x y\right\}$ is obtained by applying Algorithm 6 to $G$.
1


Looking at the lattices, we can verify that the involutive cones give a disjoint cover of the conventional cones up to monomials of degree 4. However, if we were to draw the next part of the lattices (monomials of degree 5), we would notice that the monomial $x y^{3} x$ is conventionally reducible by the Gröbner Basis, but is not involutively reducible by the Involutive Basis. This fact verifies that when thick divisors are being used, a Locally Involutive Basis is not necessarily an Involutive Basis, as for $G$ to be an Involutive Basis with respect to $\mathcal{S}$ and thick divisors, the monomial $x y^{3} x$ has to be involutively reducible with respect to $G$.

### 5.7.2 Involutive Rewrite Systems

Remark 5.7.2 In this section, we use terminology from the theory of term rewriting that has not yet been introduced in this thesis. For an elementary introduction to this theory, we refer to [5], [19] and [36].

Let $C=\langle A \mid B\rangle$ be a monoid rewrite system, where $A=\left\{a_{1}, \ldots, a_{n}\right\}$ is an alphabet and $B=\left\{b_{1}, \ldots, b_{m}\right\}$ is a set of rewrite rules of the form $b_{i}=\ell_{i} \rightarrow r_{i}(1 \leqslant i \leqslant m$; $\ell_{i}, r_{i} \in A^{*}$ ). Given a fixed admissible well-order on the words in $A$ compatible with $C$, the

Knuth-Bendix critical pairs completion algorithm [39] attempts to find a complete rewrite system $C^{\prime}$ for $C$ that is Noetherian and confluent, so that any word over the alphabet $A$ has a unique normal form with respect to $C^{\prime}$. The algorithm proceeds by considering overlaps of left hand sides of rules, forming new rules when two reductions of an overlap word result in two distinct normal forms.

It is well known (see for example [33]) that the Knuth-Bendix critical pairs completion algorithm is a special case of the noncommutative Gröbner Basis algorithm. To find a complete rewrite system for $C$ using Algorithm 5, we treat $C$ as a set of polynomials $F=\left\{\ell_{1}-r_{1}, \ell_{2}-r_{2}, \ldots, \ell_{m}-r_{m}\right\}$ generating a two-sided ideal over the noncommutative polynomial ring $\mathbb{Z}\left\langle a_{1}, \ldots, a_{n}\right\rangle$, and we compute a noncommutative Gröbner Basis $G$ for $F$ using a monomial ordering induced from the fixed admissible well-order on the words in $A$.

Because every noncommutative Involutive Basis (with respect to a strong or Gröbner involutive division) is a noncommutative Gröbner Basis, it is clear that a complete rewrite system for $C$ can now also be obtained by computing an Involutive Basis for $F$, a complete rewrite system we shall call an involutive complete rewrite system.

The advantage of involutive complete rewrite systems over conventional complete rewrite systems is that the unique normal form of any word over the alphabet $A$ can be obtained uniquely with respect to an involutive complete rewrite system (subject of course to certain conditions (such as working with a strong involutive division) being satisfied), a fact that will be illustrated in the following example.

Example 5.7.3 Let $C:=\langle Y, X, y, x| x^{3} \rightarrow \varepsilon, y^{2} \rightarrow \varepsilon,(x y)^{2} \rightarrow \varepsilon, X x \rightarrow \varepsilon, x X \rightarrow$ $\varepsilon, Y y \rightarrow \varepsilon, y Y \rightarrow \varepsilon\rangle$ be a monoid rewrite system for the group $S_{3}$, where $\varepsilon$ denotes the empty word, and $Y>X>y>x$ is the alphabet ordering. If we apply the Knuth-Bendix algorithm to $C$ with respect to the DegLex (word) ordering, we obtain the complete rewrite system

$$
\begin{gathered}
C^{\prime}:=\langle Y, X, y, x| x y x \rightarrow y, y x y \rightarrow X, x^{2} \rightarrow X, X x \rightarrow \varepsilon, y^{2} \rightarrow \varepsilon, X y \rightarrow y x, x X \rightarrow \\
\left.\varepsilon, y X \rightarrow x y, X^{2} \rightarrow x, Y \rightarrow y\right\rangle .
\end{gathered}
$$

With respect to the DegLex monomial ordering and the left division, if we apply Algorithm 12 to the basis $F:=\left\{x^{3}-1, y^{2}-1,(x y)^{2}-1, X x-1, x X-1, Y y-1, y Y-1\right\}$ corresponding to $C$, we obtain the following Involutive Basis for $F$ (which we have converted back to a
rewrite system to give an involutive complete rewrite system $C^{\prime \prime}$ for $C$ ).

$$
\begin{gathered}
C^{\prime \prime}:=\langle Y, X, y, x| y^{2} \rightarrow \varepsilon, X x \rightarrow \varepsilon, x X \rightarrow \varepsilon, Y y \rightarrow \varepsilon, y^{2} x \rightarrow x, Y \rightarrow y, Y x \rightarrow \\
y x, X x y \rightarrow y, Y y x \rightarrow x, x^{2} \rightarrow X, X^{2} \rightarrow x, x y x \rightarrow y, X y \rightarrow y x, X y x \rightarrow x y, x^{2} y \rightarrow \\
y x, y X \rightarrow x y, y x y \rightarrow X, Y x y \rightarrow X, Y X \rightarrow x y\rangle .
\end{gathered}
$$

With the involutive complete rewrite system, we are now able to uniquely reduce each word over the alphabet $\{Y, X, y, x\}$ to one of the six elements of $S_{3}$. To illustrate this, consider the word $y X Y x$. Using the 10 element complete rewrite system $C^{\prime}$ obtained by using the Knuth-Bendix algorithm, there are several reduction paths for this word, as illustrated by the following diagram.


However, by involutively reducing the word $y X Y x$ with respect to the 19 element involutive complete rewrite system $C^{\prime \prime}$, there is only one reduction path, namely

$$
\begin{aligned}
& y X Y x \\
& \mid Y x \rightarrow y x \\
& y X y x \\
& \mid X y x \rightarrow x y \\
& \downarrow y \\
& y x y \\
& \mid y x y \rightarrow X \\
& \vdots \\
& X
\end{aligned}
$$

### 5.7.3 Comparison of Divisions

Following on from the $S_{3}$ example above, consider the basis $F:=\left\{x^{4}-1, y^{3}-1,(x y)^{2}-\right.$ 1, $X x-1, x X-1, Y y-1, y Y-1\}$ over the polynomial ring $\mathbb{Q}\langle Y, X, y, x\rangle$ corresponding to a monoid rewrite system for the group $S_{4}$. With the monomial ordering being DegLex, below we present some data collected when, whilst using a prototype implementation of Algorithm 12 (as given in Appendix B), an Involutive Basis is computed for $F$ with respect to several different involutive divisions (the reduced DegLex Gröbner Basis for $F$ has 21 elements).

Remark 5.7.4 The program was run using FreeBSD 5.4 on an AMD Athlon XP 1800+ with 512 MB of memory.

| Key | Involutive Division | Key | Involutive Division |
| :---: | :--- | :---: | :--- |
| 1 | Left Division | 7 | Subword-Free Left Overlap Division |
| 2 | Right Division | 8 | Right Overlap Division |
| 3 | Left Overlap Division | 9 | Strong Right Overlap Division |
| 4 | Strong Left Overlap Division | 10 | Two-Sided Right Overlap Division |
| 5 | Two-Sided Left Overlap Division | 11 | Suffix-Only Right Overlap Division |
| 6 | Prefix-Only Left Overlap Division | 12 | Subword-Free Right Overlap Division |


| Division | Size of Basis | Number of <br> Prolongations | Number of <br> Involutive Reductions | Time |
| :---: | :---: | :---: | :---: | :---: |
| 1 | 73 | 104 | 15947 | 0.77 |
| 2 | 73 | 104 | 13874 | 0.74 |
| 3 | 65 | 64 | 10980 | 8.62 |
| 4 | 73 | 94 | 15226 | 23.14 |
| 5 | 77 | 70 | 12827 | 16.04 |
| 6 | 65 | 64 | 10980 | 8.97 |
| 7 | 65 | 64 | 10980 | 7.13 |
| 8 | 73 | 76 | 11046 | 13.27 |
| 9 | 73 | 95 | 13240 | 26.16 |
| 10 | 87 | 80 | 13005 | 24.53 |
| 11 | 73 | 76 | 11046 | 13.40 |
| 12 | 69 | 82 | 10458 | 9.52 |

We note that the algorithm completes quickest with respect to the global left or right divisions, as (i) we can take advantage of the efficient involutive reduction with respect to these divisions (see Section 5.5.1); and (ii) the multiplicative variables for a particular monomial with respect to these divisions is fixed (each time the basis changes when using one of the other local divisions, the multiplicative variables have to be recomputed). However, we also note that more prolongations are needed when using the left or right divisions, so that, in the long run, if we can devise an efficient way of finding the multiplicative variables for a set of monomials with respect to one of the local divisions, then the algorithm could (perhaps) terminate more quickly than for the two global divisions.

### 5.8 Improvements to the Noncommutative Involutive Basis Algorithm

### 5.8.1 Efficient Reduction

Conventionally, we divide a noncommutative polynomial $p$ with respect to a set of polynomials $P$ using Algorithm 2. In this algorithm, an important step is to find out if a polynomial in $P$ divides one of the monomials $u$ in the polynomial we are currently reducing, stated as the condition 'if $\left(\mathrm{LM}\left(p_{j}\right) \mid u\right)$ then' in Algorithm 2. One way of finding out if this condition is satisfied would be to execute the following piece of code, where $\alpha:=\operatorname{deg}(u) ; \beta:=\operatorname{deg}\left(\operatorname{LM}\left(p_{j}\right)\right)$; and we note that $\alpha-\beta+1$ operations are potentially needed to find out if the condition is satisfied.

```
\(i=1\);
while \((i \leqslant \alpha-\beta+1)\) do
    if \(\left(\operatorname{LM}\left(p_{j}\right)==\operatorname{Subword}(u, i, i+\beta-1)\right)\) then
        return true;
        else
            \(i=i+1 ;\)
        end if
end while
return false;
```

When involutively dividing a polynomial $p$ with respect to a set of polynomials $P$ and some involutive division $I$, the corresponding problem is to find out if some monomial
$\operatorname{LM}\left(p_{j}\right)$ is an involutive divisor of some monomial $u$. At first glance, this problem seems more difficult than the problem of finding out if $\operatorname{LM}\left(p_{j}\right)$ is a conventional divisor of $u$, as it is not just sufficient to discover one way that $\operatorname{LM}\left(p_{j}\right)$ divides $u$ (as in the code above) - we have to verify that if we find a conventional divisor of $u$, then it is also an involutive divisor of $u$. Naively, assuming that thin divisors are being used, we could solve the problem using the code shown below, code that is clearly less efficient than the code for the conventional case shown above.

```
\(i=1 ;\)
while \((i \leqslant \alpha-\beta+1)\) do
    if \(\left(\operatorname{LM}\left(p_{j}\right)==\operatorname{Subword}(u, i, i+\beta-1)\right)\) then
        if \(\left((i==1)\right.\) or \(\left((i>1)\right.\) and \(\left.\left(\operatorname{Subword}(u, i-1, i-1) \in \mathcal{M}_{I}^{L}\left(\operatorname{LM}\left(p_{j}\right), \operatorname{LM}(P)\right)\right)\right)\)
        then
                if \(((i==\alpha-\beta+1)\) or \(((i<\alpha-\beta+1)\) and \((\operatorname{Subword}(u, i+\beta, i+\beta) \in\)
                \(\left.\left.\mathcal{M}_{I}^{R}\left(\mathrm{LM}\left(p_{j}\right), \mathrm{LM}(P)\right)\right)\right)\) then
                return true;
                end if
        end if
    else
        \(i=i+1 ;\)
    end if
end while
return false;
```

However, for certain involutive divisions, it is possible to take advantage of some of the properties of these divisions in order to make it easier to discover whether $\operatorname{LM}\left(p_{j}\right)$ is an involutive divisor of $u$. We have already seen this in action in Section 5.5.1, where we saw that $\mathrm{LM}\left(p_{j}\right)$ can only involutively divide $u$ with respect to the left or right divisions if $\operatorname{LM}\left(p_{j}\right)$ is a suffix or prefix of $u$ respectively.

Let us now consider an improvement to be used whenever (i) an 'overlap' division that assigns all variables to be either left multiplicative or right multiplicative is used (ruling out any 'two-sided' overlap divisions); and (ii) thick divisors are being used. For the case of such an overlap division that assigns all variables to be left multiplicative (for example the left overlap division), the following piece of code can be used to discover whether or not $\mathrm{LM}\left(p_{j}\right)$ is an involutive divisor of $u$ (note that a similar piece of code can be given for the case of an overlap division assigning all variables to be right multiplicative).

```
\(k=\alpha\); skip \(=0 ;\)
while \((k \geqslant \beta+1)\) do
        if \(\left(\operatorname{Subword}(u, k, k) \notin \mathcal{M}_{I}^{R}\left(\operatorname{LM}\left(p_{j}\right), \operatorname{LM}(P)\right)\right)\) then
            skip \(=k ; k=\beta\);
        else
            \(k=k-1 ;\)
        end if
end while
if (skip \(==0\) ) then
        \(i=1\);
else
        \(i=\) skip \(-\beta+1 ;\)
end if
while \((i \leqslant \alpha-\beta+1)\) do
        if \(\left(\operatorname{LM}\left(p_{j}\right)==\operatorname{Subword}(u, i, i+\beta-1)\right)\) then
            return true;
        else
            \(i=i+1 ;\)
        end if
end while
return false;
```

We note that the final section of the code (from 'while ( $i \leqslant \alpha-\beta+1$ ) do' onwards) is identical to the code for conventional reduction; the code before this just chooses the initial value of $i$ (we rule out looking at certain subwords by analysing which variables in $u$ are right nonmultiplicative for $\left.\operatorname{LM}\left(p_{j}\right)\right)$. For example, if $u:=x y^{2} x y x y ; \operatorname{LM}\left(p_{j}\right):=x y x$; and only the variable $x$ is right nonmultiplicative for $p_{j}$, then in the conventional case we need 4 subword comparisons before we discover that $\operatorname{LM}\left(p_{j}\right)$ is a conventional divisor of $u$; but in the involutive case (using the code shown above) we only need 1 subword comparison before we discover that $\operatorname{LM}\left(p_{j}\right)$ is an involutive divisor of $u$ (this is because the variable $\operatorname{Subword}(u, 6,6)=x$ is right nonmultiplicative for $\operatorname{LM}\left(p_{j}\right)$, leaving just two subwords of $u$ that are potentially equal to $\operatorname{LM}\left(p_{j}\right)$ in such a way that $\operatorname{LM}\left(p_{j}\right)$ is an involutive divisor of $u$ ).

Conventional Reduction
$\left.\begin{array}{lllllllllllllll} & x & y & y & x & y & x & y & & x & y & y & x & y & x\end{array}\right) y$
$i=1 \begin{array}{llllllll} & x & y & x\end{array} \quad i=4 \quad x \quad y \quad x$
$i=2 \quad x \quad y \quad x$

Involutive Reduction
$\qquad$

$$
\begin{array}{lll}
x & 9 & x
\end{array}
$$

$$
i=3 \quad x \quad y \quad x
$$

$$
i=4 \quad x \quad y \quad x
$$

$\square$
$\square$


Of course our new algorithm will not always 'win' in every case (for example if $u:=$ $x y x^{2} y x y$ and $\left.\operatorname{LM}\left(p_{j}\right):=x y x\right)$, and we will always have the overhead from having to determine the initial value of $i$, but the impression should be that we have more freedom in the involutive case to try these sorts of tricks, tricks which may lead to involutive reduction being more efficient than conventional reduction.

### 5.8.2 Improved Algorithms

Just as Algorithm 9 was generalised to give an algorithm for computing noncommutative Involutive Bases in Algorithm 12, it is conceivable that other algorithms for computing commutative Involutive Bases (as seen for example in [24]) can be generalised to the noncommutative case. Indeed, in the source code given in Appendix B, a noncommutative version of an algorithm found in [23, Section 5] for computing commutative Involutive Bases is given; we present below data obtained by applying this new algorithm to our $S_{4}$ example from Section 5.7.3 (the data from Section 5.7.3 is given in brackets for comparison; we see that the new algorithm generally analyses more prolongations but performs less involutive reduction).

| Division | Size of Basis | Number of <br> Prolongations | Number of <br> Involutive Reductions | Time |
| :---: | :---: | :---: | :---: | :---: |
| 1 | $73(73)$ | $323(104)$ | $875(15947)$ | $0.72(0.77)$ |
| 2 | $73(73)$ | $327(104)$ | $929(13874)$ | $0.83(0.74)$ |
| 3 | $70(65)$ | $288(64)$ | $831(10980)$ | $5.94(8.62)$ |
| 4 | $73(73)$ | $318(94)$ | $863(15226)$ | $4.62(23.14)$ |
| 5 | $70(77)$ | $288(70)$ | $831(12827)$ | $5.79(16.04)$ |
| 6 | $70(65)$ | $288(64)$ | $831(10980)$ | $5.71(8.97)$ |
| 7 | $69(65)$ | $288(64)$ | $833(10980)$ | $5.33(7.13)$ |
| 8 | $68(73)$ | $358(76)$ | $1092(11046)$ | $28.51(13.27)$ |
| 9 | $73(73)$ | $322(95)$ | $917(13240)$ | $6.39(26.16)$ |
| 10 | $68(87)$ | $358(80)$ | $1092(13005)$ | $28.75(24.53)$ |
| 11 | $68(73)$ | $358(76)$ | $1092(11046)$ | $28.54(13.40)$ |
| 12 | $66(69)$ | $364(82)$ | $1127(10458)$ | $28.87(9.52)$ |

### 5.8.3 Logged Involutive Bases

A (noncommutative) Logged Involutive Basis expresses each member of an Involutive Basis in terms of members of the original basis from which the Involutive Basis was computed.

Definition 5.8.1 Let $G=\left\{g_{1}, \ldots, g_{p}\right\}$ be an Involutive Basis computed from an initial basis $F=\left\{f_{1}, \ldots, f_{m}\right\}$. We say that $G$ is a Logged Involutive Basis if, for each $g_{i} \in G$, we have an explicit expression of the form

$$
g_{i}=\sum_{\alpha=1}^{\beta} \ell_{\alpha} f_{k_{\alpha}} r_{\alpha},
$$

where the $\ell_{\alpha}$ and the $r_{\alpha}$ are terms and $f_{k_{\alpha}} \in F$ for all $1 \leqslant \alpha \leqslant \beta$.
Proposition 5.8.2 Let $F=\left\{f_{1}, \ldots, f_{m}\right\}$ be a finite basis over a noncommutative polynomial ring. If we can compute an Involutive Basis for $F$, then it is always possible to compute a Logged Involutive Basis for F.

Proof: Let $G=\left\{g_{1}, \ldots, g_{p}\right\}$ be an Involutive Basis computed from the initial basis $F=\left\{f_{1}, \ldots, f_{m}\right\}$ using Algorithm 12 (where $f_{i} \in R\left\langle x_{1}, \ldots, x_{n}\right\rangle$ for all $f_{i} \in F$ ). If an arbitrary $g_{i} \in G$ is not a member of the original basis $F$, then either $g_{i}$ is an involutively
reduced prolongation, or $g_{i}$ is obtained through the process of autoreduction. In the former case, we can express $g_{i}$ in terms of members of $F$ by substitution because either

$$
g_{i}=x_{j} h-\sum_{\alpha=1}^{\beta} \ell_{\alpha} h_{k_{\alpha}} r_{\alpha}
$$

or

$$
g_{i}=h x_{j}-\sum_{\alpha=1}^{\beta} \ell_{\alpha} h_{k_{\alpha}} r_{\alpha}
$$

for a variable $x_{j}$; terms $\ell_{\alpha}$ and $r_{\alpha}$; and polynomials $h$ and $h_{k_{\alpha}}$ which we already know how to express in terms of members of $F$. In the latter case,

$$
g_{i}=h-\sum_{\alpha=1}^{\beta} \ell_{\alpha} h_{k_{\alpha}} r_{\alpha}
$$

for terms $\ell_{\alpha}, r_{\alpha}$ and polynomials $h$ and $h_{k_{\alpha}}$ which we already know how to express in terms of members of $F$, so it follows that we can again express $g_{i}$ in terms of members of $F$.

Example 5.8.3 Let $F:=\left\{f_{1}, f_{2}\right\}=\left\{x^{3}+3 x y-y x, y^{2}+x\right\}$ generate an ideal over the polynomial ring $\mathbb{Q}\langle x, y\rangle$; let the monomial ordering be DegLex; and let the involutive division be the left division. In obtaining an Involutive Basis for $F$ using Algorithm 12, a polynomial is added to $F ; f_{1}$ is involutively reduced during autoreduction; and then four more polynomials are added to $F$, giving an Involutive Basis $G:=\left\{g_{1}, g_{2}, g_{3}, g_{4}, g_{5}, g_{6}, g_{7}\right\}=$ $\left\{x^{3}+2 y x, y^{2}+x, x y-y x, y^{2} x+x^{2}, x y x-y x^{2}, y^{2} x^{2}-2 y x, x y x^{2}-2 x^{2}\right\}$.

The five new polynomials were obtained by involutively reducing the prolongations $f_{2} y$,
$f_{2} x, g_{3} x, g_{4} x$ and $g_{5} x$ respectively.

\[

\]

These reductions (plus the reduction

$$
\begin{array}{rll}
f_{1} & \longrightarrow_{g_{3}} & x^{3}+3 x y-y x-3(x y-y x) \\
& = & x^{3}+2 y x
\end{array}
$$

of $f_{1}$ performed during autoreduction after $g_{3}$ is added to $F$ ) enable us to give the following Logged Involutive Basis for $F$.

| Member of $G$ | Logged Representation |
| :--- | :--- |
| $g_{1}=x^{3}+2 y x$ | $f_{1}-3 f_{2} y+3 y f_{2}$ |
| $g_{2}=y^{2}+x$ | $f_{2}$ |
| $g_{3}=x y-y x$ | $f_{2} y-y f_{2}$ |
| $g_{4}=y^{2} x+x^{2}$ | $f_{2} x$ |
| $g_{5}=x y x-y x^{2}$ | $f_{2} y x-y f_{2} x$ |
| $g_{6}=y^{2} x^{2}-2 y x$ | $-f_{1}+f_{2} x^{2}+3 f_{2} y-3 y f_{2}$ |
| $g_{7}=x y x^{2}-2 x^{2}$ | $y f_{1}+3 y^{2} f_{2}+f_{2} y x^{2}-2 f_{2} x-y f_{2} x^{2}-3 y f_{2} y$ |

## Chapter 6

## Gröbner Walks

When computing any Gröbner or Involutive Basis, the monomial ordering that has been chosen is a major factor in how long it will take for the algorithm to complete. For example, consider the ideal $J$ generated by the basis $F:=\left\{-2 x^{3} z+y^{4}+y^{3} z-x^{3}+\right.$ $\left.x^{2} y, 2 x y^{2} z+y z^{3}+2 y z^{2}, x^{3} y+2 y z^{3}-3 y z^{2}+2\right\}$ over the polynomial ring $\mathbb{Q}[x, y, z]$. Using our test implementation of Algorithm 3, it takes less than a tenth of a second to compute a Gröbner Basis for $F$ with respect to the DegRevLex monomial ordering, but 90 seconds to compute a Gröbner Basis for $F$ with respect to Lex.

The Gröbner Walk, introduced by Collart, Kalkbrener and Mall in [18], forms part of a family of basis conversion algorithms that can convert Gröbner Bases with respect to 'fast' monomial orderings to Gröbner Bases with respect to 'slow' monomial orderings (see Section 2.5.4 for a brief discussion of other basis conversion algorithms). This process is often quicker than computing a Gröbner Basis for the 'slow' monomial ordering directly, as can be demonstrated by stating that in our test implementation of the Gröbner Walk, it only takes half a second to compute a Lex Gröbner Basis for the basis $F$ defined above.

In this chapter, we will first recall the theory of the (commutative) Gröbner Walk, based on [18] and a paper [1] by Amrhein, Gloor and Küchlin; the reader is encouraged to read these papers in conjunction with this Chapter. We then describe two generalisations of the theory to give (i) a commutative Involutive Walk (due to Golubitsky [30]); and (ii) noncommutative Walks between harmonious monomial orderings.

### 6.1 Commutative Walks

To convert a Gröbner Basis with respect to one monomial ordering to a Gröbner Basis with respect to another monomial ordering, the Gröbner Walk works with the matrices associated to the orderings. Fortunately, [48] and [56] assert that any commutative monomial ordering has an associated matrix, allowing the Gröbner Walk to convert between any two monomial orderings.

### 6.1.1 Matrix Orderings

Definition 6.1.1 Let $m$ be a monomial over a polynomial ring $R\left[x_{1}, \ldots, x_{n}\right]$ with associated multidegree $\left(e^{1}, \ldots, e^{n}\right)$. If $\omega=\left(\omega^{1}, \ldots, \omega^{n}\right)$ is an $n$-dimensional weight vector (where $\omega^{i} \in \mathbb{Q}$ for all $1 \leqslant i \leqslant n$ ), we define the $\omega$-degree of $m$, written $\operatorname{deg}_{\omega}(m)$, to be the value

$$
\operatorname{deg}_{\omega}(m)=\left(e^{1} \times \omega^{1}\right)+\left(e^{2} \times \omega^{2}\right)+\cdots+\left(e^{n} \times \omega^{n}\right) .
$$

Remark 6.1.2 The $\omega$-degree of any term is equal to the $\omega$-degree of the term's associated monomial.

Definition 6.1.3 Let $m_{1}$ and $m_{2}$ be two monomials over a polynomial ring $R\left[x_{1}, \ldots, x_{n}\right]$ with associated multidegrees $e_{1}=\left(e_{1}^{1}, \ldots, e_{1}^{n}\right)$ and $e_{2}=\left(e_{2}^{1}, \ldots, e_{2}^{n}\right)$; and let $\Omega$ be an $N \times n$ matrix. If $\omega_{i}$ denotes the $n$-dimensional weight vector corresponding to the $i$-th row of $\Omega$, then $\Omega$ determines a monomial ordering as follows: $m_{1}<m_{2}$ if $\operatorname{deg}_{\omega_{i}}\left(m_{1}\right)<\operatorname{deg}_{\omega_{i}}\left(m_{2}\right)$ for some $1 \leqslant i \leqslant N$ and $\operatorname{deg}_{\omega_{j}}\left(m_{1}\right)=\operatorname{deg}_{\omega_{j}}\left(m_{2}\right)$ for all $1 \leqslant j<i$.

Definition 6.1.4 The corresponding matrices for the five monomial orderings defined in
Section 1.2.1 are

$$
\text { Lex }=\left(\begin{array}{cccccc}
1 & 0 & 0 & \ldots & 0 & 0 \\
0 & 1 & 0 & \ldots & 0 & 0 \\
0 & 0 & 1 & \ldots & 0 & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & 0 & \ldots & 1 & 0 \\
0 & 0 & 0 & \ldots & 0 & 1
\end{array}\right) ; \quad \operatorname{InvLex}=\left(\begin{array}{cccccc}
0 & 0 & 0 & \ldots & 0 & 1 \\
0 & 0 & 0 & \ldots & 1 & 0 \\
\vdots & \vdots & \vdots & . & \vdots & \vdots \\
0 & 0 & 1 & \ldots & 0 & 0 \\
0 & 1 & 0 & \ldots & 0 & 0 \\
1 & 0 & 0 & \ldots & 0 & 0
\end{array}\right) ;
$$

$$
\begin{aligned}
\operatorname{DegLex}= & \left(\begin{array}{cccccc}
1 & 1 & 1 & \ldots & 1 & 1 \\
1 & 0 & 0 & \ldots & 0 & 0 \\
0 & 1 & 0 & \ldots & 0 & 0 \\
0 & 0 & 1 & \ldots & 0 & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & 0 & \ldots & 1 & 0
\end{array}\right) ; \quad \operatorname{DegInvLex}=\left(\begin{array}{rrrrrr}
1 & 1 & 1 & \ldots & 1 & 1 \\
0 & 0 & 0 & \ldots & 0 & 1 \\
0 & 0 & 0 & \ldots & 1 & 0 \\
\vdots & \vdots & \vdots & . & \vdots & \vdots \\
0 & 0 & 1 & \ldots & 0 & 0 \\
0 & 1 & 0 & \ldots & 0 & 0
\end{array}\right) ; \\
& \text { DegRevLex }=\left(\begin{array}{rrrrrr}
1 & 1 & 1 & \ldots & 1 & 1 \\
0 & 0 & 0 & \ldots & 0 & -1 \\
0 & 0 & 0 & \ldots & -1 & 0 \\
\vdots & \vdots & \vdots & . & \vdots & \vdots \\
0 & 0 & -1 & \ldots & 0 & 0 \\
0 & -1 & 0 & \ldots & 0 & 0
\end{array}\right) .
\end{aligned}
$$

Example 6.1.5 Let $m_{1}:=x^{2} y^{2} z^{2}$ and $m_{2}:=x^{2} y^{3} z$ be two monomials over the polynomial ring $\mathcal{R}:=\mathbb{Q}[x, y, z]$. According to the matrix

$$
\left(\begin{array}{lll}
1 & 1 & 1 \\
1 & 0 & 0 \\
0 & 1 & 0
\end{array}\right)
$$

representing the DegLex monomial ordering with respect to $\mathcal{R}$, we can deduce that $m_{1}<$ $m_{2}$ because $\operatorname{deg}_{\omega_{1}}\left(m_{1}\right)=\operatorname{deg}_{\omega_{1}}\left(m_{2}\right)=6 ; \operatorname{deg}_{\omega_{2}}\left(m_{1}\right)=\operatorname{deg}_{\omega_{2}}\left(m_{2}\right)=2$; and $\operatorname{deg}_{\omega_{3}}\left(m_{1}\right)=$ $2<\operatorname{deg}_{\omega_{3}}\left(m_{2}\right)=3$.

Definition 6.1.6 Given a polynomial $p$ and a weight vector $\omega$, the initial of $p$ with respect to $\omega$, written $\mathrm{in}_{\omega}(p)$, is the sum of those terms in $p$ that have maximal $\omega$-degree. For example, if $\omega=(0,1,1)$ and $p=x^{4}+x y^{2} z+y^{3}+x z^{2}$, then $\operatorname{in}_{\omega}(p)=x y^{2} z+y^{3}$.

Definition 6.1.7 A weight vector $\omega$ is compatible with a monomial ordering $O$ if, given any polynomial $p=t_{1}+\cdots+t_{m}$ ordered in descending order with respect to $O, \operatorname{deg}_{\omega}\left(t_{1}\right) \geqslant$ $\operatorname{deg}_{\omega}\left(t_{i}\right)$ holds for all $1<i \leqslant m$.

### 6.1.2 The Commutative Gröbner Walk Algorithm

We present in Algorithm 17 an algorithm to perform the Gröbner Walk, modified from an algorithm given in [1].

```
Algorithm 17 The Commutative Gröbner Walk Algorithm
Input: A Gröbner Basis \(G=\left\{g_{1}, g_{2}, \ldots, g_{m}\right\}\) with respect to an admissible monomial or-
    dering \(O\) with an associated matrix \(A\), where \(G\) generates an ideal \(J\) over a commutative
    polynomial ring \(\mathcal{R}=R\left[x_{1}, \ldots, x_{n}\right]\).
Output: A Gröbner Basis \(H=\left\{h_{1}, h_{2}, \ldots, h_{p}\right\}\) for \(J\) with respect to an admissible
    monomial ordering \(O^{\prime}\) with an associated matrix \(B\).
    Let \(\omega\) and \(\tau\) be the weight vectors corresponding to the first rows of \(A\) and \(B\);
    Let \(C\) be the matrix whose first row is equal to \(\omega\) and whose remainder is equal to the
    whole of the matrix \(B\);
    \(t=0\); found \(=\) false;
    repeat
        Let \(G^{\prime}=\left\{\operatorname{in}_{\omega}\left(g_{i}\right)\right\}\) for all \(g_{i} \in G\);
        Compute a reduced Gröbner Basis \(H^{\prime}\) for \(G^{\prime}\) with respect to the monomial ordering
    defined by the matrix \(C\) (use Algorithms 3 and 4);
    \(H=\emptyset\);
    for each \(h^{\prime} \in H^{\prime}\) do
        Let \(\sum_{i=1}^{j} p_{i} g_{i}^{\prime}\) be the logged representation of \(h^{\prime}\) with respect to \(G^{\prime}\) (where \(g_{i}^{\prime} \in G^{\prime}\)
        and \(p_{i} \in \mathcal{R}\) ), obtained either through computing a Logged Gröbner Basis above or
        by dividing \(h^{\prime}\) with respect to \(G^{\prime}\);
        \(H=H \cup\left\{\sum_{i=1}^{j} p_{i} g_{i}\right\}\), where \(\mathrm{in}_{\omega}\left(g_{i}\right)=g_{i}^{\prime}\);
    end for
    Reduce \(H\) with respect to \(C\) (use Algorithm 4);
    if \((t==1)\) then
        found \(=\) true;
    else
        \(t=\min \left(\left\{s \mid \operatorname{deg}_{\omega(s)}\left(p_{1}\right)=\operatorname{deg}_{\omega(s)}\left(p_{i}\right), \operatorname{deg}_{\omega(0)}\left(p_{1}\right) \neq \operatorname{deg}_{\omega(0)}\left(p_{i}\right)\right.\right.\),
        \(\left.\left.h=p_{1}+\cdots+p_{k} \in H\right\} \cap(0,1]\right)\), where \(\omega(s):=\omega+s(\tau-\omega)\) for \(0 \leqslant s \leqslant 1\);
    end if
    if ( \(t\) is undefined) then
        found \(=\) true;
    else
        \(G=H ; \omega=(1-t) \omega+t \tau ;\)
    end if
    until (found \(=\) true)
    return \(H\);
```


## Some Remarks:

- In the first iteration of the repeat ... until loop, $G^{\prime}$ is a Gröbner Basis for the ideal ${ }^{1}$ $\mathrm{in}_{\omega}(J)$ with respect to the monomial ordering defined by $C$, as $\omega$ is compatible with C. During subsequent iterations of the same loop, $G^{\prime}$ is a Gröbner Basis for the ideal $\operatorname{in}_{\omega}(J)$ with respect to the monomial ordering used to compute $H$ during the previous iteration of the repeat ... until loop, as $\omega$ is compatible with this previous ordering.
- The fact that any set $H$ constructed by the for loop is a Gröbner Basis for $J$ with respect to the monomial ordering defined by $C$ is proved in both [1] and [18] (where you will also find proofs for the assertions made in the previous paragraph).
- The section of code where we determine the value of $t$ is where we determine the next step of the walk. We choose $t$ to be the minimum value of $s$ in the interval $(0,1]$ such that, for some polynomial $h \in H$, the $\omega$-degrees of $\operatorname{LT}(h)$ and some other term in $h$ differ, but the $\omega(s)$-degrees of the same two terms are identical. We say that this is the first point on the line segment between the two weight vectors $\omega$ and $\tau$ where the initial of one of the polynomials in $H$ degenerates.
- The success of the Gröbner Walk comes from the fact that it breaks down a Gröbner Basis computation into a series of smaller pieces, each of which computes a Gröbner Basis for a set of initials, a task that is usually quite simple. There are still cases however where this task is complicated and time-consuming, and this has led to the development of so-called path perturbation techniques that choose 'easier' paths on which to walk (see for example [1] and [53]).


### 6.1.3 A Worked Example

Example 6.1.8 Let $F:=\{x y-z, y z+2 x+z\}$ be a basis generating an ideal $J$ over the polynomial ring $\mathbb{Q}[x, y, z]$. Consider that we want to obtain the Lex Gröbner Basis $H:=\left\{2 x+y z+z, y^{2} z+y z+2 z\right\}$ for $J$ from the DegLex Gröbner Basis $G:=\{x y-$ $\left.z, y z+2 x+z, 2 x^{2}+x z+z^{2}\right\}$ for $J$ using the Gröbner Walk. Utilising Algorithm 17 to do this, we initialise the variables as follows.

[^11]\[

$$
\begin{gathered}
A=\left(\begin{array}{ccc}
1 & 1 & 1 \\
1 & 0 & 0 \\
0 & 1 & 0
\end{array}\right) ; B=\left(\begin{array}{ccc}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{array}\right) ; \omega=(1,1,1) ; \tau=(1,0,0) ; C=\left(\begin{array}{ccc}
1 & 1 & 1 \\
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{array}\right) ; \\
t=0 ; \text { found }=\text { false. }
\end{gathered}
$$
\]

Let us now describe what happens during each pass of the repeat... until loop of Algorithm 17, noting that as $A$ is equivalent to $C$ to begin with, nothing substantial will happen during the first pass through the loop.

## Pass 1

- Construct the set of initials: $G^{\prime}:=\left\{g_{1}^{\prime}, g_{2}^{\prime}, g_{3}^{\prime}\right\}=\left\{x y, y z, 2 x^{2}+x z+z^{2}\right\}$ (these are the terms in $G$ that have maximal ( $1,1,1$ )-degree).
- Compute the Gröbner Basis $H^{\prime}$ of $G^{\prime}$ with respect to $C$.

$$
\begin{aligned}
\mathrm{S}-\operatorname{pol}\left(g_{1}^{\prime}, g_{2}^{\prime}\right) & =\frac{x y z}{x y}(x y)-\frac{x y z}{y z}(y z) \\
& =0 ; \\
\text { S-pol }\left(g_{1}^{\prime}, g_{3}^{\prime}\right) & =\frac{x^{2} y}{x y}(x y)-\frac{x^{2} y}{2 x^{2}}\left(2 x^{2}+x z+z^{2}\right) \\
& =-\frac{1}{2} x y z-\frac{1}{2} y z^{2} \\
& \rightarrow_{g_{1}^{\prime}}-\frac{1}{2} y z^{2} \\
& \rightarrow_{g_{2}^{\prime}} \\
\text { S-pol }\left(g_{2}^{\prime}, g_{3}^{\prime}\right) & =0 \text { (by Buchberger's First Criterion). }
\end{aligned}
$$

It follows that $H^{\prime}=G^{\prime}$.

- As $H^{\prime}=G^{\prime}, H$ will also be equal to $G$, so that $H:=\left\{h_{1}, h_{2}, h_{3}\right\}=\{x y-z, y z+$ $\left.2 x+z, 2 x^{2}+x z+z^{2}\right\}$.
- Let

$$
\begin{aligned}
\omega(s) & :=\omega+s(\tau-\omega) \\
& =(1,1,1)+s((1,0,0)-(1,1,1)) \\
& =(1,1,1)+s(0,-1,-1) \\
& =(1,1-s, 1-s) .
\end{aligned}
$$

To find the next value of $t$, we must find the minimum value of $s$ such that the $\omega(s)$-degrees of the leading term of a polynomial in $H$ and some other term in the same polynomial agree where their $\omega$-degrees currently differ.
The $\omega$-degrees of the two terms in $h_{1}$ differ, so we can seek a value of $s$ such that

$$
\begin{aligned}
\operatorname{deg}_{\omega(s)}(x y) & =\operatorname{deg}_{\omega(s)}(z) \\
1+(1-s) & =(1-s) \\
1 & =0 \text { (inconsistent). }
\end{aligned}
$$

For $h_{2}$, we have two choices: either

$$
\begin{aligned}
\operatorname{deg}_{\omega(s)}(y z) & =\operatorname{deg}_{\omega(s)}(x) \\
(1-s)+(1-s) & =1 \\
2-2 s & =1 \\
s & =\frac{1}{2} ;
\end{aligned}
$$

or

$$
\begin{aligned}
\operatorname{deg}_{\omega(s)}(y z) & =\operatorname{deg}_{\omega(s)}(z) \\
(1-s)+(1-s) & =(1-s) \\
(1-s) & =0 \\
s & =1 .
\end{aligned}
$$

The $\omega$-degrees of all the terms in $h_{3}$ are the same, so we can ignore it.
It follows that the minimum value of $s$ (and hence the new value of $t$ ) is $\frac{1}{2}$. As this value appears in the interval $(0,1]$, we set $G=H$; set the new value of $\omega$ to be $\left(1-\frac{1}{2}\right)(1,1,1)+\frac{1}{2}(1,0,0)=\left(1, \frac{1}{2}, \frac{1}{2}\right)$ (and hence change $C$ to be the matrix $\left(\begin{array}{ccc}1 & \frac{1}{2} & \frac{1}{2} \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1\end{array}\right)$ ); and embark upon a second pass of the repeat. .. until loop.

## Pass 2

- Construct the set of initials: $G^{\prime}:=\left\{g_{1}^{\prime}, g_{2}^{\prime}, g_{3}^{\prime}\right\}=\left\{x y, 2 x+y z, 2 x^{2}\right\}$ (these are the
terms in $G$ that have maximal ( $1, \frac{1}{2}, \frac{1}{2}$ )-degree).
- Compute the Gröbner Basis of $G^{\prime}$ with respect to $C$.

$$
\begin{aligned}
\text { S-pol }\left(g_{1}^{\prime}, g_{2}^{\prime}\right) & =\frac{x y}{x y}(x y)-\frac{x y}{2 x}(2 x+y z) \\
& =-\frac{1}{2} y^{2} z=: g_{4}^{\prime} ; \\
\text { S-pol }\left(g_{1}^{\prime}, g_{3}^{\prime}\right) & =\frac{x^{2} y}{x y}(x y)-\frac{x^{2} y}{2 x^{2}}\left(2 x^{2}\right) \\
& =0 ; \\
\text { S-pol }\left(g_{2}^{\prime}, g_{3}^{\prime}\right) & =\frac{x^{2}}{2 x}(2 x+y z)-\frac{x^{2}}{2 x^{2}}\left(2 x^{2}\right) \\
& =\frac{1}{2} x y z \\
& \rightarrow 0 ; \\
g_{1}^{\prime} & \\
\text { S-pol }\left(g_{1}^{\prime}, g_{4}^{\prime}\right) & =\frac{x y^{2} z}{x y}(x y)-\frac{x y^{2} z}{-\frac{1}{2} y^{2} z}\left(-\frac{1}{2} y^{2} z\right) \\
& =0 ; \\
\text { S-pol }\left(g_{2}^{\prime}, g_{4}^{\prime}\right) & =0 \text { (by Buchberger's First Criterion); } \\
\text { S-pol }\left(g_{3}^{\prime}, g_{4}^{\prime}\right) & =0 \text { (by Buchberger's First Criterion). }
\end{aligned}
$$

It follows that $G^{\prime}=\left\{g_{1}^{\prime}, g_{2}^{\prime}, g_{3}^{\prime}, g_{4}^{\prime}\right\}=\left\{x y, 2 x+y z, 2 x^{2},-\frac{1}{2} y^{2} z\right\}$ is a Gröbner Basis for $\mathrm{in}_{\omega}(J)$ with respect to $C$.

Applying Algorithm 4 to $G^{\prime}$, we can remove $g_{1}^{\prime}$ and $g_{3}^{\prime}$ from $G^{\prime}$ (because $\operatorname{LM}\left(g_{1}^{\prime}\right)=$ $y \times \operatorname{LM}\left(g_{2}^{\prime}\right)$ and $\left.\operatorname{LM}\left(g_{3}^{\prime}\right)=x \times \operatorname{LM}\left(g_{2}^{\prime}\right)\right)$; we must also multiply $g_{2}^{\prime}$ and $g_{4}^{\prime}$ by $\frac{1}{2}$ and -2 respectively to obtain unit lead coefficients. This leaves us with the unique reduced Gröbner Basis $H^{\prime}:=\left\{h_{1}^{\prime}, h_{2}^{\prime}\right\}=\left\{x+\frac{1}{2} y z, y^{2} z\right\}$ for $\operatorname{in}_{\omega}(J)$ with respect to $C$.

- We must now express the two elements of $H^{\prime}$ in terms of members of $G^{\prime}$.

$$
\begin{aligned}
h_{1}^{\prime}=x+\frac{1}{2} y z & =\frac{1}{2} g_{2}^{\prime} \\
h_{2}^{\prime}=y^{2} z & =-2\left((x y)-\frac{1}{2} y(2 x+y z)\right) \text { (from the S-polynomial) } \\
& =-2\left(g_{1}^{\prime}-\frac{1}{2} y g_{2}^{\prime}\right) .
\end{aligned}
$$

Lifting to the full polynomials, $h_{1}^{\prime}$ lifts to give the polynomial $h_{1}:=x+\frac{1}{2} y z+\frac{1}{2} z$; $h_{2}^{\prime}$ lifts to give the polynomial $h_{2}:=-2\left((x y-z)-\frac{1}{2} y(2 x+y z+z)\right)=-2 x y+$
$2 z+2 x y+y^{2} z+y z=y^{2} z+y z+2 z$; and we are left with the Gröbner Basis $H:=\left\{h_{1}, h_{2}\right\}=\left\{x+\frac{1}{2} y z+\frac{1}{2} z, y^{2} z+y z+2 z\right\}$ for $J$ with respect to $C$.

- Let

$$
\begin{aligned}
\omega(s) & :=\omega+s(\tau-\omega) \\
& =\left(1, \frac{1}{2}, \frac{1}{2}\right)+s\left((1,0,0)-\left(1, \frac{1}{2}, \frac{1}{2}\right)\right) \\
& =\left(1, \frac{1}{2}, \frac{1}{2}\right)+s\left(0,-\frac{1}{2},-\frac{1}{2}\right) \\
& =\left(1, \frac{1}{2}(1-s), \frac{1}{2}(1-s)\right)
\end{aligned}
$$

Finding the minimum value of $s$, for $h_{1}$ we can have

$$
\begin{aligned}
\operatorname{deg}_{\omega(s)}(x) & =\operatorname{deg}_{\omega(s)}(z) \\
1 & =\frac{1}{2}(1-s) \\
s & =-1 \text { (undefined: we must have } s \in(0,1])
\end{aligned}
$$

Continuing with $h_{2}$, we can either have

$$
\begin{aligned}
\operatorname{deg}_{\omega(s)}\left(y^{2} z\right) & =\operatorname{deg}_{\omega(s)}(y z) \\
3\left(\frac{1}{2}(1-s)\right) & =2\left(\frac{1}{2}(1-s)\right) \\
\frac{1}{2}(1-s) & =0 \\
s & =1 ;
\end{aligned}
$$

or

$$
\begin{aligned}
\operatorname{deg}_{\omega(s)}\left(y^{2} z\right) & =\operatorname{deg}_{\omega(s)}(z) \\
3\left(\frac{1}{2}(1-s)\right) & =\frac{1}{2}(1-s) \\
1-s & =0 \\
s & =1 .
\end{aligned}
$$

It follows that the minimum value of $s$ (and hence the new value of $t$ ) is 1 . As this value appears in the interval $(0,1]$, we set $G=H$; set the new value of $\omega$
to be $(1-1)\left(1, \frac{1}{2}, \frac{1}{2}\right)+1(1,0,0)=(1,0,0)$ (and hence change $C$ to be the matrix $\left(\begin{array}{lll}1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1\end{array}\right) \equiv\left(\begin{array}{lll}1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1\end{array}\right)$ ); and embark upon a third (and final) pass of the repeat. . . until loop.

## Pass 3

- Construct the set of initials: $G^{\prime}:=\left\{g_{1}^{\prime}, g_{2}^{\prime}\right\}=\left\{x, y^{2} z+y z+2 z\right\}$ (these are the terms in $G$ that have maximal ( $1,0,0$ )-degree).
- Compute the Gröbner Basis $H^{\prime}$ of $G^{\prime}$ with respect to $C$.

$$
\mathrm{S}-\mathrm{pol}\left(g_{1}^{\prime}, g_{2}^{\prime}\right)=0 \text { (by Buchberger's First Criterion). }
$$

It follows that $H^{\prime}=G^{\prime}$.

- As $H^{\prime}=G^{\prime}, H$ will also be equal to $G$, so that $H:=\left\{h_{1}, h_{2}\right\}=\left\{x+\frac{1}{2} y z+\frac{1}{2} z, y^{2} z+\right.$ $y z+2 z\}$. Further, as $t$ is now equal to 1 , we have arrived at our target ordering (Lex) and can return $H$ as the output Gröbner Basis, a basis that is equivalent to the Lex Gröbner Basis given for $J$ at the beginning of this example.

We can summarise the path taken during the walk in the following diagram.


```
Algorithm 18 The Commutative Involutive Walk Algorithm
Input: An Involutive Basis \(G=\left\{g_{1}, g_{2}, \ldots, g_{m}\right\}\) with respect to an involutive division \(I\)
    and an admissible monomial ordering \(O\) with an associated matrix \(A\), where \(G\) generates
    an ideal \(J\) over a commutative polynomial ring \(\mathcal{R}=R\left[x_{1}, \ldots, x_{n}\right]\).
Output: An Involutive Basis \(H=\left\{h_{1}, h_{2}, \ldots, h_{p}\right\}\) for \(J\) with respect to \(I\) and an admis-
    sible monomial ordering \(O^{\prime}\) with an associated matrix \(B\).
    Let \(\omega\) and \(\tau\) be the weight vectors corresponding to the first rows of \(A\) and \(B\);
    Let \(C\) be the matrix whose first row is equal to \(\omega\) and whose remainder is equal to the
    whole of the matrix \(B\);
    \(t=0\); found \(=\) false;
    repeat
    Let \(G^{\prime}=\left\{\mathrm{in}_{\omega}\left(g_{i}\right)\right\}\) for all \(g_{i} \in G\);
    Compute an Involutive Basis \(H^{\prime}\) for \(G^{\prime}\) with respect to the monomial ordering defined
    by the matrix \(C\) (use Algorithm 9);
    \(H=\emptyset\);
    for each \(h^{\prime} \in H^{\prime}\) do
        Let \(\sum_{i=1}^{j} p_{i} g_{i}^{\prime}\) be the logged representation of \(h^{\prime}\) with respect to \(G^{\prime}\) (where \(g_{i}^{\prime} \in G^{\prime}\)
        and \(p_{i} \in \mathcal{R}\) ), obtained either through computing a Logged Involutive Basis above
        or by involutively dividing \(h^{\prime}\) with respect to \(G^{\prime}\);
        \(H=H \cup\left\{\sum_{i=1}^{j} p_{i} g_{i}\right\}\), where \(\operatorname{in}_{\omega}\left(g_{i}\right)=g_{i}^{\prime} ;\)
    end for
    if \((t==1)\) then
        found \(=\) true;
    else
        \(t=\min \left(\left\{s \mid \operatorname{deg}_{\omega(s)}\left(p_{1}\right)=\operatorname{deg}_{\omega(s)}\left(p_{i}\right), \operatorname{deg}_{\omega(0)}\left(p_{1}\right) \neq \operatorname{deg}_{\omega(0)}\left(p_{i}\right)\right.\right.\),
        \(\left.\left.h=p_{1}+\cdots+p_{k} \in H\right\} \cap(0,1]\right)\), where \(\omega(s):=\omega+s(\tau-\omega)\) for \(0 \leqslant s \leqslant 1\);
    end if
    if ( \(t\) is undefined) then
        found \(=\) true;
    else
        \(G=H ; \omega=(1-t) \omega+t \tau ;\)
        end if
    until (found \(=\) true)
    return \(H\);
```


### 6.1.4 The Commutative Involutive Walk Algorithm

In [30], Golubitsky generalised the Gröbner Walk technique to give a method for converting an Involutive Basis with respect to one monomial ordering to an Involutive Basis with respect to another monomial ordering. Algorithmically, the way in which we perform this Involutive Walk is virtually identical to the way we perform the Gröbner Walk, as can be seen by comparing Algorithms 17 and 18. The change however comes when proving the correctness of the algorithm, as we have to show that each $G^{\prime}$ is an Involutive Basis for $\operatorname{in}_{\omega}(J)$ and that each $H$ is an Involutive Basis for $J$ (see [30] for these proofs).

### 6.2 Noncommutative Walks

In the commutative case, any monomial ordering can be represented by a matrix that provides a decomposition of the ordering in terms of the rows of the matrix. This decomposition is then utilised in the Gröbner Walk algorithm when (for example) we use the first row of the matrix to provide a set of initials for a particular basis $G$ (cf. Definition 6.1.6).

In the noncommutative case, because monomials cannot be represented by multidegrees, monomial orderings cannot be represented by matrices. This seems to shut the door on any generalisation of the Gröbner Walk to the noncommutative case, as not only is there no first row of a matrix to provide a set of initials, but no notion of a walk between two matrices can be formulated either.

Despite this, we note that in the commutative case, if the first rows of the source and target matrices are the same, then the Gröbner Walk will complete in one pass of the algorithm, and all that is needed is the first row of the source matrix to provide a set of initials to work with.

Generalising to the noncommutative case, it is possible that if we can find a way to decompose a noncommutative monomial ordering to provide a set of initials to work with, then a noncommutative Gröbner Walk algorithm could complete in one pass if the source and target monomial orderings used the same method to compute sets of initials.

### 6.2.1 Functional Decompositions

Considering the monomial orderings defined in Section 1.2.2, we note that all the orderings are defined step-by-step. For example, the DegLex monomial ordering compares two monomials by degree first, then by the first letter of each monomial, then by the second letter, and so on. This provides us with an opportunity to decompose each monomial ordering into a series of steps or functions, a decomposition we shall term a functional decomposition.

Definition 6.2.1 An ordering function is a function

$$
\theta: M \longrightarrow \mathbb{Z}
$$

that assigns an integer to any monomial $m \in M$, where $M$ denotes the set of all monomials over a polynomial ring $R\left\langle x_{1}, \ldots, x_{n}\right\rangle$. We call the integer assigned by $\theta$ to $m$ the $\theta$-value of $m$.

Remark 6.2.2 The $\theta$-value of any term will be equal to the $\theta$-value of the term's associated monomial.

Definition 6.2.3 A functional decomposition $\Theta$ is a (possibly infinite) sequence of ordering functions, written $\Theta=\left\{\theta_{1}, \theta_{2}, \ldots\right\}$.

Definition 6.2.4 Let $O$ be a monomial ordering; let $m_{1}$ and $m_{2}$ be two arbitrary monomials such that $m_{1}<m_{2}$ with respect to $O$; and let $\Theta=\left\{\theta_{1}, \theta_{2}, \ldots\right\}$ be a functional decomposition. We say that $\Theta$ defines $O$ if and only if $\theta_{i}\left(m_{1}\right)<\theta_{i}\left(m_{2}\right)$ for some $i \geqslant 1$ and $\theta_{j}\left(m_{1}\right)=\theta_{j}\left(m_{2}\right)$ for all $1 \leqslant j<i$.

To describe the functional decompositions corresponding to the monomial orderings defined in Section 1.2.2, we first need the following definition.

Definition 6.2.5 Let $m$ be an arbitrary monomial over a polynomial ring $R\left\langle x_{1}, \ldots, x_{n}\right\rangle$. The $i$-th valuing function of $m$, written $^{\operatorname{val}}{ }_{i}(m)$, is an ordering function that assigns an integer to $m$ as follows.

$$
\operatorname{val}_{i}(m)= \begin{cases}j & \text { if } \operatorname{Subword}(m, i, i)=x_{j}(\text { where } 1 \leqslant j \leqslant n) . \\ n+1 & \text { if } \operatorname{Subword}(m, i, i) \text { is undefined. }\end{cases}
$$

Let us now describe the functional decompositions corresponding to those monomial orderings defined in Section 1.2.2 that are admissible.

Definition 6.2.6 The functional decomposition $\Theta=\left\{\theta_{1}, \theta_{2}, \ldots\right\}$ corresponding to the DegLex monomial ordering is defined (for an arbitrary monomial $m$ ) as follows.

$$
\theta_{i}(m)= \begin{cases}\operatorname{deg}(m) & \text { if } i=1 \\ n+1-\operatorname{val}_{i-1}(m) & \text { if } i>1\end{cases}
$$

Similarly, we can define DegInvLex by

$$
\theta_{i}(m)= \begin{cases}\operatorname{deg}(m) & \text { if } i=1 \\ \operatorname{val}_{i-1}(m) & \text { if } i>1\end{cases}
$$

and DegRevLex by

$$
\theta_{i}(m)= \begin{cases}\operatorname{deg}(m) & \text { if } i=1 \\ \operatorname{val}_{\operatorname{deg}(m)+2-i}(m) & \text { if } i>1\end{cases}
$$

Example 6.2.7 Let $m_{1}:=x y x z^{2}$ and $m_{2}:=x z y z^{2}$ be two monomials over the polynomial ring $\mathbb{Q}\langle x, y, z\rangle$. With respect to DegLex, we can work out that $x y x z^{2}>x z y z^{2}$, because $\theta_{1}\left(m_{1}\right)=\theta_{1}\left(m_{2}\right)\left(\operatorname{ordeg}\left(m_{1}\right)=\operatorname{deg}\left(m_{2}\right)\right) ; \theta_{2}\left(m_{1}\right)=\theta_{2}\left(m_{2}\right)\left(\right.$ or $n+1-\operatorname{val}_{1}\left(m_{1}\right)=$ $\left.n+1-\operatorname{val}_{1}\left(m_{2}\right), 3+1-1=3+1-1\right)$; and $\theta_{3}\left(m_{1}\right)>\theta_{3}\left(m_{2}\right)\left(\right.$ or $n+1-\operatorname{val}_{2}\left(m_{1}\right)>$ $\left.n+1-\operatorname{val}_{2}\left(m_{2}\right), 3+1-2>3+1-3\right)$. Similarly, with respect to DegInvLex, we can work out that $x y x z^{2}<x z y z^{2}$ (because $\theta_{3}\left(m_{1}\right)<\theta_{3}\left(m_{2}\right)$, or $2<3$ ); and with respect to DegRevLex, we can work out that $x y x z^{2}<x z y z^{2}$ (because $\theta_{4}\left(m_{1}\right)<\theta_{4}\left(m_{2}\right)$, or $1<2$ ).

Definition 6.2.8 Given a polynomial $p$ and an ordering function $\theta$, the initial of $p$ with respect to $\theta$, written $\operatorname{in}_{\theta}(p)$, is made up of those terms in $p$ that have maximal $\theta$-value. For example, if $\theta$ is the degree function and if $p=x^{4}+z x y^{2}+y^{3}+z^{2} x$, then $\mathrm{in}_{\theta}(p)=x^{4}+z x y^{2}$.

Definition 6.2.9 Given an ordering function $\theta$, a polynomial $p$ is said to be $\theta$-homogeneous if $p=\mathrm{in}_{\theta}(p)$.

Definition 6.2.10 An ordering function $\theta$ is compatible with a monomial ordering $O$ if, given any polynomial $p=t_{1}+\cdots+t_{m}$ ordered in descending order with respect to $O$, $\theta\left(t_{1}\right) \geqslant \theta\left(t_{i}\right)$ holds for all $1<i \leqslant m$.

Definition 6.2.11 An ordering function $\theta$ is extendible if, given any $\theta$-homogeneous polynomial $p$, any multiple $u p v$ of $p$ by terms $u$ and $v$ is also $\theta$-homogeneous.

Remark 6.2.12 Of the ordering functions encountered so far, only the degree function, $\operatorname{val}_{1}$ and $^{2} \operatorname{val}_{\operatorname{deg}(m)}$ (for any given monomial $m$ ) are extendible.

Definition 6.2.13 Two noncommutative monomial orderings $O_{1}$ and $O_{2}$ are said to be harmonious if (i) there exist functional decompositions $\Theta_{1}=\left\{\theta_{1_{1}}, \theta_{1_{2}}, \ldots\right\}$ and $\Theta_{2}=$ $\left\{\theta_{2_{1}}, \theta_{2_{2}}, \ldots\right\}$ defining $O_{1}$ and $O_{2}$ respectively; and (ii) the ordering functions $\theta_{1_{1}}$ and $\theta_{2_{1}}$ are identical and extendible.

Remark 6.2.14 The noncommutative monomial orderings DegLex, DegInvLex and DegRevLex are all (pairwise) harmonious.

### 6.2.2 The Noncommutative Gröbner Walk Algorithm for Harmonious Monomial Orderings

We present in Algorithm 19 an algorithm to perform a Gröbner Walk between two harmonious noncommutative monomial orderings.

Termination of Algorithm 19 depends on the termination of Algorithm 5 as used (in Algorithm 19) to compute a noncommutative Gröbner Basis for the set $G^{\prime}$. The correctness of Algorithm 19 is provided by the following two propositions.

Proposition 6.2.15 $G^{\prime}$ is always a Gröbner Basis for the ideaß ${ }^{\beta} \mathrm{in}_{\theta}(J)$ with respect to the monomial ordering $O$.

Proof: Because $\theta$ is compatible with $O$ (by definition), the S -polynomials involving members of $G$ will be in one-to-one correspondence with the S -polynomials involving members of $G^{\prime}$, with the same monomial being 'cancelled' in each pair of corresponding S-polynomials.

Let $p$ be an arbitrary S-polynomial involving members of $G$ (with corresponding Spolynomial $q$ involving members of $G^{\prime}$ ). Because $G$ is a Gröbner Basis for $J$ with respect

[^12]
## Algorithm 19 The Noncommutative Gröbner Walk Algorithm for Harmonious Monomial Orderings

Input: A Gröbner Basis $G=\left\{g_{1}, g_{2}, \ldots, g_{m}\right\}$ with respect to an admissible monomial ordering $O$ with an associated functional decomposition $A$, where $G$ generates an ideal $J$ over a noncommutative polynomial ring $\mathcal{R}=R\left\langle x_{1}, \ldots, x_{n}\right\rangle$.
Output: A Gröbner Basis $H=\left\{h_{1}, h_{2}, \ldots, h_{p}\right\}$ for $J$ with respect to an admissible monomial ordering $O^{\prime}$ with an associated functional decomposition $B$, where $O$ and $O^{\prime}$ are harmonious.

Let $\theta$ be the ordering function corresponding to the first ordering function of $A$;
Let $G^{\prime}=\left\{\mathrm{in}_{\theta}\left(g_{i}\right)\right\}$ for all $g_{i} \in G$;
Compute a reduced Gröbner Basis $H^{\prime}$ for $G^{\prime}$ with respect to the monomial ordering $O^{\prime}$ (use Algorithms 5 and 6);
$H=\emptyset$;
for each $h^{\prime} \in H^{\prime}$ do
Let $\sum_{i=1}^{j} \ell_{i} g_{i}^{\prime} r_{i}$ be the logged representation of $h^{\prime}$ with respect to $G^{\prime}$ (where $g_{i}^{\prime} \in G^{\prime}$ and the $\ell_{i}$ and the $r_{i}$ are terms), obtained either through computing a Logged Gröbner Basis above or by dividing $h^{\prime}$ with respect to $G^{\prime}$;
$H=H \cup\left\{\sum_{i=1}^{j} \ell_{i} g_{i} r_{i}\right\}$, where $\mathrm{in}_{\theta}\left(g_{i}\right)=g_{i}^{\prime}$;
end for
Reduce $H$ with respect to $O^{\prime}$ (use Algorithm 6);
return $H$;
to $O, p$ will reduce to zero using $G$ by the series of reductions

$$
p \rightarrow_{g_{i_{1}}} p_{1} \rightarrow_{g_{i_{2}}} p_{2} \rightarrow_{g_{i_{3}}} \cdots \rightarrow_{g_{i_{\alpha}}} 0
$$

where $g_{i_{j}} \in G$ for all $1 \leqslant j \leqslant \alpha$.
Claim: $q$ will reduce to zero using $G^{\prime}$ (and hence $G^{\prime}$ is a Gröbner Basis for $\mathrm{in}_{\theta}(J)$ with respect to $O$ by Definition 3.1.8) by the series of reductions

$$
q \rightarrow \operatorname{in}_{\theta}\left(g_{i_{1}}\right) q_{1} \rightarrow_{\mathrm{in}_{\theta}\left(g_{i_{2}}\right)} q_{2} \rightarrow_{\mathrm{in}_{\theta}\left(g_{i_{3}}\right)} \cdots \rightarrow_{\mathrm{in}_{\theta}\left(g_{i_{\beta}}\right)} 0
$$

where $0 \leqslant \beta \leqslant \alpha$.
Proof of Claim: Let $w$ be the overlap word associated to the S-polynomial $p$. If $\theta(\operatorname{LM}(p))<\theta(w)$, then because $\theta$ is extendible it is clear that $q=0$, and so the proof is complete. Otherwise, we must have $q=\operatorname{in}_{\theta}(p)$, and so by the compatibility of $\theta$ with $O$, we can use the polynomial $\operatorname{in}_{\theta}\left(g_{i_{1}}\right) \in G^{\prime}$ to reduce $q$ to give the polynomial $q_{1}$. We now proceed by induction (if $\theta\left(\operatorname{LM}\left(p_{1}\right)\right)<\theta(\operatorname{LM}(p))$ then $\left.q_{1}=0, \ldots\right)$, noting that the process will terminate because $\mathrm{in}_{\theta}\left(p_{\alpha}=0\right)=0$.

Proposition 6.2.16 The set $H$ constructed by the for loop of Algorithm 19 is a Gröbner Basis for $J$ with respect to the monomial ordering $O^{\prime}$.

Proof: By Definition 3.1.8, we can show that $H$ is a Gröbner Basis for $J$ by showing that all S-polynomials involving members of $H$ reduce to zero using $H$. Assume for a contradiction that an S-polynomial $p$ involving members of $H$ does not reduce to zero using $H$, and instead only reduces to a polynomial $q \neq 0$.

As all members of $H$ are members of the ideal $J$ (by the way $H$ was constructed as combinations of elements of $G$ ), it is clear that $q$ is also a member of the ideal $J$, as all we have done in constructing $q$ is to reduce a combination of two members of $H$ with respect to $H$. It follows that the polynomial $\mathrm{in}_{\theta}(q)$ is a member of the ideal $\mathrm{in}_{\theta}(J)$.

Because $H^{\prime}$ is a Gröbner Basis for the ideal $\mathrm{in}_{\theta}(J)$ with respect to $O^{\prime}$, there must be a polynomial $h^{\prime} \in H^{\prime}$ such that $h^{\prime} \mid \mathrm{in}_{\theta}(q)$. Let $\sum_{i=1}^{j} \ell_{i} g_{i}^{\prime} r_{i}$ be the logged representation of
$h^{\prime}$ with respect to $G^{\prime}$. Then it is clear that

$$
\sum_{i=1}^{j} \ell_{i} g_{i}^{\prime} r_{i} \mid \mathrm{in}_{\theta}(q)
$$

However $\theta$ is compatible with $O^{\prime}$, so that

$$
\sum_{i=1}^{j} \ell_{i} g_{i} r_{i} \mid q
$$

It follows that there exists a polynomial $h \in H$ dividing our polynomial $q$, contradicting our initial assumption.

### 6.2.3 A Worked Example

Example 6.2.17 Let $F:=\left\{x^{2}+y^{2}+8,2 x y+y^{2}+5\right\}$ be a basis generating an ideal $J$ over the polynomial ring $\mathbb{Q}\langle x, y\rangle$. Consider that we want to obtain the DegLex Gröbner Basis $H:=\left\{2 x y+y^{2}+5, x^{2}+y^{2}+8,5 y^{3}-10 x+37 y, 2 y x+y^{2}+5\right\}$ for $J$ from the DegRevLex Gröbner Basis $G:=\left\{2 x y-x^{2}-3, y^{2}+x^{2}+8,5 x^{3}+6 y+35 x, 2 y x-x^{2}-3\right\}$ for $J$ using the Gröbner Walk. Utilising Algorithm 19 to do this, we initialise $\theta$ to be the degree function and we proceed as follows.

- Construct the set of initials: $G^{\prime}:=\left\{g_{1}^{\prime}, g_{2}^{\prime}, g_{3}^{\prime}, g_{4}^{\prime}\right\}=\left\{-x^{2}+2 x y, x^{2}+y^{2}, 5 x^{3},-x^{2}+\right.$ $2 y x\}$ (these are the terms in $G$ that have maximal degree).
- Compute the Gröbner Basis of $G^{\prime}$ with respect to the DegLex monomial ordering (for simplicity, we will not provide details of those S-polynomials that reduce to zero
or can be ignored due to Buchberger's Second Criterion).

$$
\begin{aligned}
\operatorname{S-pol}\left(1, g_{1}^{\prime}, 1, g_{2}^{\prime}\right) & =\left(-x^{2}+2 x y\right)-(-1)\left(x^{2}+y^{2}\right) \\
& =2 x y+y^{2}=: g_{5}^{\prime} ; \\
\operatorname{S-pol}\left(1, g_{1}^{\prime}, 1, g_{4}^{\prime}\right) & =(-1)\left(-x^{2}+2 x y\right)-(-1)\left(-x^{2}+2 y x\right) \\
& =-2 x y+2 y x \\
& \rightarrow_{g_{5}^{\prime}}-2 x y+2 y x+\left(2 x y+y^{2}\right) \\
& =2 y x+y^{2}=: g_{6}^{\prime} ; \\
\operatorname{S-pol}\left(y, g_{1}^{\prime}, 1, g_{6}^{\prime}\right) & =2 y\left(-x^{2}+2 x y\right)-(-1)\left(2 y x+y^{2}\right) x \\
& =4 y x y+y^{2} x \\
& \rightarrow_{g_{5}^{\prime}} 4 y x y+y^{2} x-2 y\left(2 x y+y^{2}\right) \\
& =y^{2} x-2 y^{3} \\
& \rightarrow_{g_{6}^{\prime}} y^{2} x-2 y^{3}-\frac{1}{2} y\left(2 y x+y^{2}\right) \\
& =-\frac{5}{2} y^{3}=: g_{7}^{\prime} .
\end{aligned}
$$

After $g_{7}^{\prime}$ is added to the current basis, all S-polynomials now reduce to zero, leaving the Gröbner Basis $G^{\prime}=\left\{g_{1}^{\prime}, g_{2}^{\prime}, g_{3}^{\prime}, g_{4}^{\prime}, g_{5}^{\prime}, g_{6}^{\prime}, g_{7}^{\prime}\right\}=\left\{-x^{2}+2 x y, x^{2}+y^{2}, 5 x^{3},-x^{2}+\right.$ $\left.2 y x, 2 x y+y^{2}, 2 y x+y^{2},-\frac{5}{2} y^{3}\right\}$ for $\mathrm{in}_{\theta}(J)$ with respect to $O^{\prime}$.

Applying Algorithm 6 to $G^{\prime}$, we can remove $g_{1}^{\prime}, g_{2}^{\prime}$ and $g_{3}^{\prime}$ from $G^{\prime}$ (because their lead monomials are all multiplies of $\left.\operatorname{LM}\left(g_{4}^{\prime}\right)\right)$; we must multiply $g_{4}^{\prime}, g_{5}^{\prime}, g_{6}^{\prime}$ and $g_{7}^{\prime}$ by $-1, \frac{1}{2}, \frac{1}{2}$ and $-\frac{2}{5}$ respectively (to obtain unit lead coefficients); and the polynomial $g_{4}^{\prime}$ can (then) be further reduced as follows.

$$
\begin{aligned}
g_{4}^{\prime} & =x^{2}-2 y x \\
& \rightarrow g_{6}^{\prime}-2 y x+2\left(y x+\frac{1}{2} y^{2}\right) \\
& =x^{2}+y^{2} .
\end{aligned}
$$

This leaves us with the unique reduced Gröbner Basis $H^{\prime}:=\left\{h_{1}^{\prime}, h_{2}^{\prime}, h_{3}^{\prime}, h_{4}^{\prime}\right\}=$ $\left\{x^{2}+y^{2}, x y+\frac{1}{2} y^{2}, y x+\frac{1}{2} y^{2}, y^{3}\right\}$ for $\operatorname{in}_{\theta}(J)$ with respect to $O^{\prime}$.

- We must now express the four elements of $H^{\prime}$ in terms of members of $G^{\prime}$.

$$
\begin{aligned}
h_{1}^{\prime}=x^{2}+y^{2} & =g_{2}^{\prime} ; \\
h_{2}^{\prime}=x y+\frac{1}{2} y^{2} & =\frac{1}{2}\left(g_{1}^{\prime}+g_{2}^{\prime}\right)(\text { from the S-polynomial) } ; \\
h_{3}^{\prime}=y x+\frac{1}{2} y^{2} & =\frac{1}{2}\left(-g_{1}^{\prime}+g_{4}^{\prime}+\left(g_{1}^{\prime}+g_{2}^{\prime}\right)\right) \\
& =\frac{1}{2}\left(g_{2}^{\prime}+g_{4}^{\prime}\right) ; \\
h_{4}^{\prime}=y^{3} & =-\frac{2}{5}\left(2 y\left(g_{1}^{\prime}\right)+\left(g_{2}^{\prime}+g_{4}^{\prime}\right) x-2 y\left(g_{1}^{\prime}+g_{2}^{\prime}\right)-\frac{1}{2} y\left(g_{2}^{\prime}+g_{4}^{\prime}\right)\right) \\
& =-\frac{2}{5}\left(g_{2}^{\prime} x-\frac{5}{2} y g_{2}^{\prime}+g_{4}^{\prime} x-\frac{1}{2} y g_{4}^{\prime}\right) .
\end{aligned}
$$

Lifting to the full polynomials, we obtain the Gröbner Basis $H:=\left\{h_{1}, h_{2}, h_{3}, h_{4}\right\}$ as follows.

$$
\begin{aligned}
h_{1} & =g_{2} \\
& =x^{2}+y^{2}+8 ; \\
h_{2} & =\frac{1}{2}\left(g_{1}+g_{2}\right) \\
& =\frac{1}{2}\left(-x^{2}+2 x y-3+x^{2}+y^{2}+8\right) \\
& =x y+\frac{1}{2} y^{2}+\frac{5}{2} ; \\
h_{3} & =\frac{1}{2}\left(g_{2}+g_{4}\right) \\
& =\frac{1}{2}\left(x^{2}+y^{2}+8-x^{2}+2 y x-3\right) \\
& =y x+\frac{1}{2} y^{2}+\frac{5}{2} ; \\
h_{4} & =-\frac{2}{5}\left(g_{2} x-\frac{5}{2} y g_{2}+g_{4} x-\frac{1}{2} y g_{4}\right) \\
& =-\frac{2}{5}\left(x^{3}+y^{2} x+8 x-\frac{5}{2} y x^{2}-\frac{5}{2} y^{3}-20 y\right. \\
& \left.-x^{3}+2 y x^{2}-3 x+\frac{1}{2} y x^{2}-y^{2} x+\frac{3}{2} y\right) \\
& =y^{3}-2 x+\frac{37}{5} y .
\end{aligned}
$$

The set $H$ does not reduce any further, so we return the output DegLex Gröbner Basis $\left\{x^{2}+y^{2}+8, x y+\frac{1}{2} y^{2}+\frac{5}{2}, y x+\frac{1}{2} y^{2}+\frac{5}{2}, y^{3}-2 x+\frac{37}{5} y\right\}$ for $J$, a basis
that is equivalent to the DegLex Gröbner Basis given for $J$ at the beginning of this example.

### 6.2.4 The Noncommutative Involutive Walk Algorithm for Harmonious Monomial Orderings

We present in Algorithm 20 an algorithm to perform an Involutive Walk between two harmonious noncommutative monomial orderings.

## Algorithm 20 The Noncommutative Involutive Walk Algorithm for Harmonious Monomial Orderings

Input: An Involutive Basis $G=\left\{g_{1}, g_{2}, \ldots, g_{m}\right\}$ with respect to an involutive division $I$ and an admissible monomial ordering $O$ with an associated functional decomposition $A$, where $G$ generates an ideal $J$ over a noncommutative polynomial ring $\mathcal{R}=R\left\langle x_{1}, \ldots, x_{n}\right\rangle$.
Output: An Involutive Basis $H=\left\{h_{1}, h_{2}, \ldots, h_{p}\right\}$ for $J$ with respect to $I$ and an admissible monomial ordering $O^{\prime}$ with an associated functional decomposition $B$, where $O$ and $O^{\prime}$ are harmonious.

Let $\theta$ be the ordering function corresponding to the first ordering function of $A$;
Let $G^{\prime}=\left\{\operatorname{in}_{\theta}\left(g_{i}\right)\right\}$ for all $g_{i} \in G$;
Compute an Involutive Basis $H^{\prime}$ for $G^{\prime}$ with respect to $I$ and the monomial ordering $O^{\prime}$ (use Algorithm 12);
$H=\emptyset$;
for each $h^{\prime} \in H^{\prime}$ do
Let $\sum_{i=1}^{j} \ell_{i} g_{i}^{\prime} r_{i}$ be the logged representation of $h^{\prime}$ with respect to $G^{\prime}$ (where $g_{i}^{\prime} \in$ $G^{\prime}$ and the $\ell_{i}$ and the $r_{i}$ are terms), obtained either through computing a Logged Involutive Basis above or by involutively dividing $h^{\prime}$ with respect to $G^{\prime}$;
$H=H \cup\left\{\sum_{i=1}^{j} \ell_{i} g_{i} r_{i}\right\}$, where $\mathrm{in}_{\theta}\left(g_{i}\right)=g_{i}^{\prime} ;$
end for
return $H$;

Termination of Algorithm 20 depends on the termination of Algorithm 12 as used (in Algorithm 20) to compute a noncommutative Involutive Basis for the set $G^{\prime}$. The correctness of Algorithm 20 is provided by the following two propositions.

Proposition 6.2.18 $G^{\prime}$ is always an Involutive Basis for the ideal $\mathrm{in}_{\theta}(J)$ with respect to $I$ and the monomial ordering $O$.

Proof: Let $p:=u g v$ be an arbitrary multiple of a polynomial $g \in G$ by terms $u$ and $v$. Because $G$ is an Involutive Basis for $J$ with respect to $I$ and $O, p$ will involutively reduce to zero using $G$ by the series of involutive reductions

$$
p \xrightarrow[I g_{i_{1}}]{ } p_{1} \xrightarrow[I]{ } g_{i_{2}} p_{2} \xrightarrow[I]{g_{i_{3}}} \cdots{\underset{I}{ } g_{i_{\alpha}}} 0
$$

where $g_{i_{j}} \in G$ for all $1 \leqslant j \leqslant \alpha$.
Claim: The polynomial $q:=u \mathrm{in}_{\theta}(g) v$ will involutively reduce to zero using $G^{\prime}$ (and hence $G^{\prime}$ is an Involutive Basis for $\mathrm{in}_{\theta}(J)$ with respect to $I$ and $O$ by Definition 5.2.7) by the series of involutive reductions

$$
q \xrightarrow[{I \underset{\operatorname{in}_{\theta}\left(g_{i_{1}}\right)}{ } q_{1} \xrightarrow[I \operatorname{in}_{\theta}\left(g_{i_{2}}\right)]{ } q_{2} \xrightarrow[I \mathrm{in}_{\theta}\left(g_{i_{3}}\right)]{ } \cdots \xrightarrow[\operatorname{in}_{\theta}\left(g_{i_{\beta}}\right)]{ } 0, ~ \text {, }}]{ }=
$$

where $1 \leqslant \beta \leqslant \alpha$.
Proof of Claim: Because $\theta$ is extendible, it is clear that $q=\operatorname{in}_{\theta}(p)$. Further, because $\theta$ is compatible with $O$ (by definition), the multiplicative variables of $G$ and $G^{\prime}$ with respect to $I$ will be identical, and so it follows that because the polynomial $g_{i_{1}} \in G$ was used to involutively reduce $p$ to give the polynomial $p_{1}$, then the polynomial $\mathrm{in}_{\theta}\left(g_{i_{1}}\right) \in G^{\prime}$ can be used to involutively reduce $q$ to give the polynomial $q_{1}$.

If $\theta\left(\operatorname{LM}\left(p_{1}\right)\right)<\theta(\operatorname{LM}(p))$, then because $\theta$ is extendible it is clear that $q_{1}=0$, and so the proof is complete. Otherwise, we must have $q_{1}=\operatorname{in}_{\theta}\left(p_{1}\right)$, and so (again) by the compatibility of $\theta$ with $O$, we can use the polynomial $\operatorname{in}_{\theta}\left(g_{i_{2}}\right) \in G^{\prime}$ to involutively reduce $q_{1}$ to give the polynomial $q_{2}$. We now proceed by induction (if $\theta\left(\operatorname{LM}\left(p_{2}\right)\right)<\theta\left(\operatorname{LM}\left(p_{1}\right)\right)$ then $\left.q_{2}=0, \ldots\right)$, noting that the process will terminate because $\mathrm{in}_{\theta}\left(p_{\alpha}=0\right)=0$.

Proposition 6.2.19 The set $H$ constructed by the for loop of Algorithm 20 is an Involutive Basis for $J$ with respect to $I$ and the monomial ordering $O^{\prime}$.

Proof: By Definition 5.2.7, we can show that $H$ is an Involutive Basis for $J$ by showing that any multiple $u p v$ of any polynomial $p \in H$ by any terms $u$ and $v$ involutively reduces to zero using $H$. Assume for a contradiction that such a multiple does not involutively reduce to zero using $H$, and instead only involutively reduces to a polynomial $q \neq 0$.

As all members of $H$ are members of the ideal $J$ (by the way $H$ was constructed as combinations of elements of $G$ ), it is clear that $q$ is also a member of the ideal $J$, as all we have done in constructing $q$ is to reduce a multiple of a polynomial in $H$ with respect to $H$. It follows that the polynomial $\mathrm{in}_{\theta}(q)$ is a member of the ideal $\mathrm{in}_{\theta}(J)$.

Because $H^{\prime}$ is an Involutive Basis for the ideal $\mathrm{in}_{\theta}(J)$ with respect to $I$ and $O^{\prime}$, there must be a polynomial $h^{\prime} \in H^{\prime}$ such that $\left.h^{\prime}\right|_{I} \mathrm{in}_{\theta}(q)$. Let $\sum_{i=1}^{j} \ell_{i} g_{i}^{\prime} r_{i}$ be the logged representation of $h^{\prime}$ with respect to $G^{\prime}$. Then it is clear that

$$
\left.\sum_{i=1}^{j} \ell_{i} g_{i}^{\prime} r_{i}\right|_{I} \mathrm{in}_{\theta}(q)
$$

However $\theta$ is compatible with $O^{\prime}$ (in particular the multiplicative variables for $H^{\prime}$ and $H$ with respect to $I$ and $O^{\prime}$ will be identical), so that

$$
\left.\sum_{i=1}^{j} \ell_{i} g_{i} r_{i}\right|_{I} q
$$

It follows that there exists a polynomial $h \in H$ involutively dividing our polynomial $q$, contradicting our initial assumption.

### 6.2.5 A Worked Example

Example 6.2.20 Let $F:=\left\{x^{2}+y^{2}+8,2 x y+y^{2}+5\right\}$ be a basis generating an ideal $J$ over the polynomial ring $\mathbb{Q}\langle x, y\rangle$. Consider that we want to obtain the DegRevLex Involutive Basis $H:=\left\{2 x y-x^{2}-3, y^{2}+x^{2}+8,5 x^{3}+6 y+35 x, 5 y x^{2}+3 y+10 x, 2 y x-x^{2}-3\right\}$ for $J$ from the DegLex Involutive Basis $G:=\left\{2 x y+y^{2}+5, x^{2}+y^{2}+8,5 y^{3}-10 x+\right.$ $\left.37 y, 5 x y^{2}+5 x-6 y, 2 y x+y^{2}+5\right\}$ for $J$ using the Involutive Walk, where $H$ and $G$ are both Involutive Bases with respect to the left division $\triangleleft$. Utilising Algorithm 20 to do this, we initialise $\theta$ to be the degree function and we proceed as follows.

- Construct the set of initials:

$$
G^{\prime}:=\left\{g_{1}^{\prime}, g_{2}^{\prime}, g_{3}^{\prime}, g_{4}^{\prime}, g_{5}^{\prime}\right\}=\left\{y^{2}+2 x y, y^{2}+x^{2}, 5 y^{3}, 5 x y^{2}, y^{2}+2 y x\right\}
$$

(these are the terms in $G$ that have maximal degree).

- Compute the Involutive Basis of $G^{\prime}$ with respect to $\triangleleft$ and the DegRevLex monomial
ordering. Step 1: autoreduce the set $G^{\prime}$.

$$
\begin{aligned}
& g_{1}^{\prime}=y^{2}+2 x y \\
& \longrightarrow_{g_{2}^{\prime}} y^{2}+2 x y-\left(y^{2}+x^{2}\right) \\
& =2 x y-x^{2}=: g_{6}^{\prime} \text {; } \\
& G^{\prime}=\left(G^{\prime} \backslash\left\{g_{1}^{\prime}\right\}\right) \cup\left\{g_{6}^{\prime}\right\} ; \\
& g_{2}^{\prime}=y^{2}+x^{2} \\
& \longrightarrow_{g_{5}^{\prime}} y^{2}+x^{2}-\left(y^{2}+2 y x\right) \\
& =\quad-2 y x+x^{2}=: g_{7}^{\prime} \text {; } \\
& G^{\prime}=\left(G^{\prime} \backslash\left\{g_{2}^{\prime}\right\}\right) \cup\left\{g_{7}^{\prime}\right\} ; \\
& g_{3}^{\prime}=5 y^{3} \\
& \vec{\triangleleft}_{9_{5}^{\prime}} 5 y^{3}-5 y\left(y^{2}+2 y x\right) \\
& =-10 y^{2} x \\
& \overleftrightarrow{ه}_{g_{7}^{\prime}}-10 y^{2} x-5 y\left(-2 y x+x^{2}\right) \\
& =\quad-5 y x^{2}=: g_{8}^{\prime} ; \\
& G^{\prime}=\left(G^{\prime} \backslash\left\{g_{3}^{\prime}\right\}\right) \cup\left\{g_{8}^{\prime}\right\} ; \\
& g_{4}^{\prime}=5 x y^{2} \\
& \longrightarrow_{g_{5}^{\prime}} 5 x y^{2}-5 x\left(y^{2}+2 y x\right) \\
& =-10 x y x \\
& \overleftrightarrow{\triangleleft}_{g_{7}^{\prime}}-10 x y x-5 x\left(-2 y x+x^{2}\right) \\
& =\quad-5 x^{3}=: g_{9}^{\prime} \text {; } \\
& G^{\prime}=\left(G^{\prime} \backslash\left\{g_{4}^{\prime}\right\}\right) \cup\left\{g_{9}^{\prime}\right\} ; \\
& g_{5}^{\prime}=y^{2}+2 y x \\
& \overleftrightarrow{ه}_{g_{7}^{\prime}} y^{2}+2 y x+\left(-2 y x+x^{2}\right) \\
& =y^{2}+x^{2}=: g_{10}^{\prime} \text {; } \\
& G^{\prime}=\left(G^{\prime} \backslash\left\{g_{5}^{\prime}\right\}\right) \cup\left\{g_{10}^{\prime}\right\} \text {. }
\end{aligned}
$$

Step 2: process the prolongations of the set $G^{\prime}=\left\{g_{6}^{\prime}, g_{7}^{\prime}, g_{8}^{\prime}, g_{9}^{\prime}, g_{10}^{\prime}\right\}$. Because all ten of these prolongations involutively reduce to zero using $G^{\prime}$, we are left with the Involutive Basis $H^{\prime}:=\left\{h_{1}^{\prime}, h_{2}^{\prime}, h_{3}^{\prime}, h_{4}^{\prime}, h_{5}^{\prime}\right\}=\left\{2 x y-x^{2},-2 y x+x^{2},-5 y x^{2},-5 x^{3}, y^{2}+\right.$
$\left.x^{2}\right\}$ for $\mathrm{in}_{\theta}(J)$ with respect to $\triangleleft$ and $O^{\prime}$.

- We must now express the five elements of $H^{\prime}$ in terms of members of $G^{\prime}$.

$$
\begin{aligned}
h_{1}^{\prime}=2 x y-x^{2} & =g_{1}^{\prime}-g_{2}^{\prime} \text { (from autoreduction); } \\
h_{2}^{\prime}=-2 y x+x^{2} & =g_{2}^{\prime}-g_{5}^{\prime} ; \\
h_{3}^{\prime}=-5 y x^{2} & =g_{3}^{\prime}-5 y g_{5}^{\prime}-5 y\left(g_{2}^{\prime}-g_{5}^{\prime}\right) \\
& =-5 y g_{2}^{\prime}+g_{3}^{\prime} ; \\
h_{4}^{\prime}=-5 x^{3} & =g_{4}^{\prime}-5 x g_{5}^{\prime}-5 x\left(g_{2}^{\prime}-g_{5}^{\prime}\right) \\
& =-5 x g_{2}^{\prime}+g_{4}^{\prime} ; \\
h_{5}^{\prime}=y^{2}+x^{2} & =g_{5}^{\prime}+\left(g_{2}^{\prime}-g_{5}^{\prime}\right) \\
& =g_{2}^{\prime} .
\end{aligned}
$$

Lifting to the full polynomials, we obtain the Involutive Basis $H:=\left\{h_{1}, h_{2}, h_{3}, h_{4}, h_{5}\right\}$ as follows.

$$
\begin{aligned}
h_{1} & =g_{1}-g_{2} \\
& =\left(y^{2}+2 x y+5\right)-\left(y^{2}+x^{2}+8\right) \\
& =2 x y-x^{2}-3 ; \\
h_{2} & =g_{2}-g_{5} \\
& =\left(y^{2}+x^{2}+8\right)-\left(y^{2}+2 y x+5\right) \\
& =-2 y x+x^{2}+3 ; \\
h_{3} & =-5 y g_{2}+g_{3} \\
& =-5 y\left(y^{2}+x^{2}+8\right)+\left(5 y^{3}+37 y-10 x\right) \\
& =-5 y x^{2}-3 y-10 x ; \\
h_{4} & =-5 x g_{2}+g_{4} \\
& =-5 x\left(y^{2}+x^{2}+8\right)+\left(5 x y^{2}-6 y+5 x\right) \\
& =-5 x^{3}-6 y-35 x ; \\
h_{5} & =g_{2} \\
& =y^{2}+x^{2}+8 .
\end{aligned}
$$

We can now return the output DegRevLex Involutive Basis $H=\left\{2 x y-x^{2}-\right.$ 3, $\left.-2 y x+x^{2}+3,-5 y x^{2}-3 y-10 x,-5 x^{3}-6 y-35 x, y^{2}+x^{2}+8\right\}$ for $J$ with
respect to $\triangleleft$, a basis that is equivalent to the DegRevLex Involutive Basis given for $J$ at the beginning of this example.

### 6.2.6 Noncommutative Walks Between Any Two Monomial Orderings?

Thus far, we have only been able to define a noncommutative walk between two harmonious monomial orderings, where we recall that the first ordering functions of the functional decompositions of the two monomial orderings must be identical and extendible. For walks between two arbitrary monomial orderings, the first ordering functions need not be identical any more, but it is clear that they must still be extendible, so that (in an algorithm to perform such a walk) each basis $G^{\prime}$ is a Gröbner Basis for each ideal $\mathrm{in}_{\theta}(J)$ (compare with the proofs of Propositions 6.2.15 and 6.2.18). This condition will also apply to any 'intermediate' monomial ordering we will encounter during the walk, but the challenge will be in how to define these intermediate orderings, so that we generalise the commutative concept of choosing a weight vector $\omega_{i+1}$ on the line segment between two weight vectors $\omega_{i}$ and $\tau$.

Open Question 4 Is it possible to perform a noncommutative walk between two admissible and extendible monomial orderings that are not harmonious?

## Chapter 7

## Conclusions

### 7.1 Current State of Play

The goal of this thesis was to combine the theories of noncommutative Gröbner Bases and commutative Involutive Bases to give a theory of noncommutative Involutive Bases. To accomplish this, we started by surveying the background theory in Chapters 1 to 4, focusing our account on the various algorithms associated with the theory. In particular, we mentioned several improvements to the standard algorithms, including how to compute commutative Involutive Bases by homogeneous methods, which required the introduction of a new property (extendibility) of commutative involutive divisions.

The theory of noncommutative Involutive Bases was introduced in Chapter 5, where we described how to perform noncommutative involutive reduction (Definition 5.1.1 and Algorithm 10); introduced the notion of a noncommutative involutive division (Definition 5.1.6); described what is meant by a noncommutative Involutive Basis (Definition 5.2.7); and gave an algorithm to compute noncommutative Involutive Bases (Algorithm 12). Several noncommutative involutive divisions were also defined, each of which was shown to satisfy certain properties (such as continuity) allowing the deductions that all Locally Involutive Bases are Involutive Bases; and that all Involutive Bases are Gröbner Bases.

To finish, we partially generalised the theory of the Gröbner Walk to the noncommutative case in Chapter 6, yielding both Gröbner and Involutive Walks between harmonious noncommutative monomial orderings.

### 7.2 Future Directions

As well as answering a few questions, the work in this thesis gives rise to a number of new questions we would like the answers to. Some of these questions have already been posed as 'Open Questions' in previous chapters; we summarise below the content of these questions.

- Regarding the procedure outlined in Definition 4.5.1 for computing an Involutive Basis for a non-homogeneous basis by homogeneous methods, if the set $G$ returned by the procedure is not autoreduced, under what circumstances does autoreducing $G$ result in obtaining a set that is an Involutive Basis for the ideal generated by the input basis $F$ ?
- Apart from the empty, left and right divisions, are there any other global noncommutative involutive divisions of the following types:
(a) strong and continuous;
(b) weak, continuous and Gröbner?
- Are there any conclusive noncommutative involutive divisions that are also continuous and either strong or Gröbner?
- Is it possible to perform a noncommutative walk between two admissible and extendible monomial orderings that are not harmonious?

In addition to seeking answers to the above questions, there are a number of other directions we could take. One area to explore would be the development of the algorithms introduced in this thesis. For example, can the improvements made to the involutive algorithms in the commutative case, such as the a priori detection of prolongations that involutively reduce to zero (see [23]), be applied to the noncommutative case? Also, can we develop multiple-object versions of our algorithms, so that (for example) noncommutative Involutive Bases for path algebras can be computed?

Implementations of any new or improved algorithms would clearly build upon the code presented in Appendix B. We could also expand this code by implementing logged versions of our algorithms; implementing efficient methods for performing involutive reduction (as seen for example in Section 5.8.1); and implementing the algorithms from Chapter 6
for performing noncommutative walks. These improved algorithms and implementations could then be used (perhaps) to help judge the relative efficiency and complexity of the involutive methods versus the Gröbner methods.

## Applications

As every noncommutative Involutive Basis is a noncommutative Gröbner Basis (at least for the involutive divisions defined in this thesis), applications for noncommutative Involutive Bases will mirror those for noncommutative Gröbner Bases. Some areas in which noncommutative Gröbner Bases have already been used include operator theory; systems engineering and linear control theory [32]. Other areas in noncommutative algebra which could also benefit from the theory introduced in this thesis include term rewriting; Petri nets; linear logic; quantum groups and coherence problems.

Further applications may come if we can extend our algorithms to the multiple-object case. It would be interesting (for example) to compare a multiple-object algorithm to a (standard) one-object algorithm in cases where an Involutive Basis for a multiple-object example can be computed using the one-object algorithm by adding some extra relations. This would tie in nicely with the existing comparison between the commutative and noncommutative versions of the Gröbner Basis algorithm, where it has been noticed that although commutative examples can be computed using the noncommutative algorithm, taking this route may in fact be less efficient than when using the commutative algorithm to do the same computation.

## Appendix A

## Proof of Propositions 5.5.31 and 5.5.32

## A. 1 Proposition 5.5.31

(Proposition 5.5.31) The two-sided left overlap division $\mathcal{W}$ is continuous.
Proof: Let $w$ be an arbitrary fixed monomial; let $U$ be any set of monomials; and consider any sequence $\left(u_{1}, u_{2}, \ldots, u_{k}\right)$ of monomials from $U\left(u_{i} \in U\right.$ for all $\left.1 \leqslant i \leqslant k\right)$, each of which is a conventional divisor of $w$ (so that $w=\ell_{i} u_{i} r_{i}$ for all $1 \leqslant i \leqslant k$, where the $\ell_{i}$ and the $r_{i}$ are monomials). For all $1 \leqslant i<k$, suppose that the monomial $u_{i+1}$ satisfies exactly one of the conditions (a) and (b) from Definition 5.4 .2 (where multiplicative variables are taken with respect to $\mathcal{W}$ over the set $U$ ). To show that $\mathcal{W}$ is continuous, we must show that no two pairs $\left(\ell_{i}, r_{i}\right)$ and $\left(\ell_{j}, r_{j}\right)$ are the same, where $i \neq j$.

Assume to the contrary that there are at least two identical pairs in the sequence

$$
\left(\left(\ell_{1}, r_{1}\right),\left(\ell_{2}, r_{2}\right), \ldots,\left(\ell_{k}, r_{k}\right)\right)
$$

so that we can choose two separate pairs $\left(\ell_{a}, r_{a}\right)$ and $\left(\ell_{b}, r_{b}\right)$ from this sequence such that $\left(\ell_{a}, r_{a}\right)=\left(\ell_{b}, r_{b}\right)$ and all the pairs $\left(\ell_{c}, r_{c}\right)$ (for $a \leqslant c<b$ ) are different. We will now show that such a sequence $\left(\left(\ell_{a}, r_{a}\right), \ldots,\left(\ell_{b}, r_{b}\right)\right)$ cannot exist.

To begin with, notice that for each monomial $u_{i+1}$ in the sequence $\left(u_{1}, \ldots, u_{k}\right)$ of mono-
mials $(1 \leqslant i<k)$, if $u_{i+1}$ involutively divides a left prolongation of the monomial $u_{i}$ (so that $\left.\left.u_{i+1}\right|_{\mathcal{W}}\left(\operatorname{Suffix}\left(\ell_{i}, 1\right)\right) u_{i}\right)$, then $u_{i+1}$ must be a prefix of this prolongation; if $u_{i+1}$ involutively divides a right prolongation of the monomial $u_{i}$ (so that $\left.u_{i+1}\right|_{\mathcal{W}} u_{i}\left(\operatorname{Prefix}\left(r_{i}, 1\right)\right)$ ), then $u_{i+1}$ must be a suffix of this prolongation. This is because in all other cases, $u_{i+1}$ is either equal to $u_{i}$, in which case $u_{i+1}$ cannot involutively divide the (left or right) prolongation of $u_{i}$ trivially; or $u_{i+1}$ is a subword of $u_{i}$, in which case $u_{i+1}$ cannot involutively divide the (left or right) prolongation of $u_{i}$ by definition of $\mathcal{W}$.

Following on from the above, we can deduce that $u_{b}$ is either a suffix or a prefix of a prolongation of $u_{b-1}$, leaving the following four cases, where $x_{b-1}^{\ell}=\operatorname{Suffix}\left(\ell_{b-1}, 1\right)$ and $x_{b-1}^{r}=\operatorname{Prefix}\left(r_{b-1}, 1\right)$.


These four cases can all originate from one of the following two cases (starting with a left prolongation or a right prolongation), where $x_{a}^{\ell}=\operatorname{Suffix}\left(\ell_{a}, 1\right)$ and $x_{a}^{r}=\operatorname{Prefix}\left(r_{a}, 1\right)$.

Case 1


## Case 2



So there are eight cases to deal with in total, namely cases 1-A, 1-B, 1-C, 1-D, 2-A, 2-B, 2 -C and 2-D.

We can immediately rule out cases 1-C and 2-A because we can show that a particular variable is both multiplicative and nonmultiplicative for monomial $u_{a}=u_{b}$ with respect to $U$, a contradiction. In case 1-C, the variable is $x_{a}^{\ell}$ : it has to be left nonmultiplicative to provide a left prolongation for $u_{a}$, and left multiplicative so that $u_{b}$ is an involutive divisor of the right prolongation of $u_{b-1}$; in case $2-\mathrm{A}$, the variable is $x_{a}^{r}$ : it has to be right nonmultiplicative to provide a right prolongation for $u_{a}$, and right multiplicative
so that $u_{b}$ is an involutive divisor of the left prolongation of $u_{b-1}$. We illustrate this in the following diagrams by using a tick to denote a multiplicative variable and a cross to denote a nonmultiplicative variable.

Case 1-C


Case 2-A


For all the remaining cases, let us now consider how we may construct a sequence $\left(\left(\ell_{a}, r_{a}\right), \ldots,\left(\ell_{b}, r_{b}\right)=\left(\ell_{a}, r_{a}\right)\right)$. Because we know that each $u_{c+1}$ is a prefix (or suffix) of a left (or right) prolongation of $u_{c}$ (where $a \leqslant c<b$ ), it is clear that at some stage during the sequence, some $u_{c+1}$ must be a proper suffix (or prefix) of a prolongation, or else the degrees of the monomials in the sequence $\left(u_{a}, \ldots\right)$ will strictly increase, meaning that we can never encounter the same $(\ell, r)$ pair twice. Further, the direction in which prolongations are taken must change some time during the sequence, or else the degrees of the monomials in one of the sequences $\left(\ell_{a}, \ldots\right)$ and $\left(r_{a}, \ldots\right)$ will strictly decrease, again meaning that we can never encounter the same $(\ell, r)$ pair twice.

A change in direction can only occur if $u_{c+1}$ is equal to a prolongation of $u_{c}$, as illustrated below.

> Left Prolongation Turn


Right Prolongation Turn


However, if no proper prefixes or suffixes are taken during the sequence, it is clear that making left or right prolongation turns will not affect the fact that the degrees of the monomials in the sequence ( $u_{a}, \ldots$ ) will strictly increase, once again meaning that we can never encounter the same ( $\ell, r$ ) pair twice. It follows that our only course of action is to
make a (left or right) prolongation turn after a proper prefix or a suffix of a prolongation has been taken. We shall call such prolongation turns prefix or suffix turns.


Claim: It is impossible to perform a prefix turn when $\mathcal{W}$ has been used to assign multiplicative variables.

Proof of Claim: It is sufficient to show that $\mathcal{W}$ cannot assign multiplicative variables to $U$ as follows:

$$
\begin{equation*}
x_{c}^{\ell} \notin \mathcal{M}_{\mathcal{W}}^{L}\left(u_{c}, U\right) ; x_{c+2}^{r} \in \mathcal{M}_{\mathcal{W}}^{R}\left(u_{c+1}, U\right) ; x_{c+2}^{r} \notin \mathcal{M}_{\mathcal{W}}^{R}\left(u_{c+2}, U\right) . \tag{A.1}
\end{equation*}
$$

Consider how Algorithm 16 can assign the variable $x_{c+2}^{r}$ to be right nonmultiplicative for monomial $u_{c+2}$. As things are set up in the digram for the prefix turn, the only possibility is that it is assigned due to the shown overlap between $u_{c}$ and $u_{c+2}$. But this assumes that these two monomials actually overlap (which won't be the case if $\operatorname{deg}\left(u_{c+1}\right)=1$ ); that $u_{c}$ is greater than or equal to $u_{c+2}$ with respect to the DegRevLex monomial ordering (so any overlap assigns a nonmultiplicative variable to $u_{c+2}$, not to $u_{c}$ ); and that, by the time we come to consider the prefix overlap between $u_{c}$ and $u_{c+2}$ in Algorithm 16, the variable $x_{c}^{\ell}$ must be left multiplicative for monomial $u_{c}$. But this final condition ensures that Algorithm 16 will terminate with $x_{c}^{\ell}$ being left multiplicative for $u_{c}$, contradicting Equation (A.1). We therefore conclude that the variable $x_{c+2}^{r}$ must be assigned right nonmultiplicative for monomial $u_{c+2}$ via some other overlap.

There are three possibilities for this overlap: (i) there exists a monomial $u_{d} \in U$ such that $u_{c+2}$ is a prefix of $u_{d}$; (ii) there exists a monomial $u_{d} \in U$ such that $u_{c+2}$ is a subword of $u_{d}$; and (iii) there exists a monomial $u_{d} \in U$ such that some prefix of $u_{d}$ is equal to some suffix of $u_{c+2}$.


In cases (i) and (ii), the overlap shown between $u_{c+1}$ and $u_{d}$ ensures that Algorithm 16 will always assign $x_{c+2}^{r}$ to be right nonmultiplicative for monomial $u_{c+1}$, contradicting Equation (A.1). This leaves case (iii), which we break down into two further subcases, dependent upon whether $u_{c+1}$ is a prefix of $u_{d}$ or not. If $u_{c+1}$ is a prefix of $u_{d}$, then Algorithm 16 will again assign $x_{c+2}^{r}$ to be right nonmultiplicative for $u_{c+1}$, contradicting Equation (A.1). Otherwise, assuming that the shown overlap between $u_{c+2}$ and $u_{d}$ assigns $x_{c+2}^{r}$ to be right nonmultiplicative for $u_{c+2}$ (so that the variable immediately to the left of monomial $u_{d}$ must be left multiplicative), we must again come to the conclusion that variable $x_{c+2}^{r}$ is right nonmultiplicative for $u_{c+1}$ (due to the overlap between $u_{c+1}$ and $u_{d}$ ), once again contradicting Equation (A.1).

Technical Point: It is possible that several left prolongations may occur between the monomials $u_{c+1}$ and $u_{c+2}$ shown in the diagram for the prefix turn, but, as long as no proper prefixes are taken during this sequence (in which case we potentially start another prefix turn), we can apply the same proof as above (replacing $c+2$ by $c+c^{\prime}$ ) to show that we cannot perform an extended prefix turn (as shown below) with respect to $\mathcal{W}$.


Having ruled out prefix turns, we can now eliminate cases 1-D, 2-C and 2-D because they require (i) a proper prefix to be taken during the sequence (allowing $\operatorname{deg}\left(r_{b-1}\right)=$ $\operatorname{deg}\left(r_{b}\right)+1$ ); and (ii) the final prolongation to be a right prolongation, ensuring that a turn has to follow the proper prefix, and so an (extended) prefix turn is required.

For Cases 1-A and 1-B, we start by taking a left prolongation, which means that somewhere during the sequence a proper suffix must be taken. To do this, it follows that we must change the direction that prolongations are taken. Knowing that prefix turns are ruled out, we must therefore turn by using a left prolongation turn, which will happen after a finite number $a^{\prime} \geqslant 1$ of left prolongations.


Considering how Algorithm 16 assigns the variable $x_{a+a^{\prime}}^{r}$ to be right nonmultiplicative for monomial $u_{a+a^{\prime}}$, there are three possibilities: (i) there exists a monomial $u_{d} \in U$ such that $u_{a+a^{\prime}}$ is a prefix of $u_{d}$; (ii) there exists a monomial $u_{d} \in U$ such that $u_{a+a^{\prime}}$ is a subword of $u_{d}$; and (iii) there exists a monomial $u_{d} \in U$ such that some prefix of $u_{d}$ is equal to some suffix of $u_{a+a^{\prime}}$. In each of these cases, there will be an overlap between $u_{a}$ and $u_{d}$ that will ensure that Algorithm 16 also assigns the variable $x_{a+a^{\prime}}^{r}$ to be right nonmultiplicative for monomial $u_{a}$. This rules out Case 1-A, as variable $x_{a+a^{\prime}}^{r}$ must be right multiplicative for monomial $u_{b}=u_{a}$ in order to perform the final step of Case 1-A.

For Case 1-B, we must now make an (extended) suffix turn as we need to finish the sequence prolongating to the left. But, once we have done this, we must subsequently take a proper prefix in order to ensure that $u_{b-1}$ is a suffix of $u_{a}=u_{b}$. Pictorially, here is one way of accomplishing this, where we note that any number of prolongations may occur between any of the shown steps.


Once we have reached the stage where we are working with a suffix of $u_{a}$, we may continue prolongating to the left until we form the monomial $u_{b}=u_{a}$, seemingly providing a
counterexample to the proposition (we have managed to construct the same ( $\ell, r$ ) pair twice). However, starting with the monomial labelled $u_{a+a^{\prime \prime}}$ in the above diagram, if we follow the sequence from $u_{a+a^{\prime \prime}}$ via left prolongations to $u_{b}=u_{a}$, and then continue with the same sequence as we started off with, we notice that by the time we encounter the monomial $u_{a+a^{\prime}}$ again, an extended prefix turn has been made, in effect meaning that the first prolongation of $u_{a}$ we took right at the start of the sequence was invalid.


This leaves Case 2-B. Here we start by taking a right prolongation, meaning that somewhere during the sequence a proper prefix must be taken. To do this, it follows that we must change the direction that prolongations are taken. There are two ways of doing this: (i) by using an (extended) suffix turn; (ii) by using a right prolongation turn.

In case (i), after performing the (extended) suffix turn, we need to take a proper prefix so that the next monomial (say $u_{c}$ ) in the sequence is a suffix of $u_{a}$; we then continue by taking left prolongations until we form the monomial $u_{b}=u_{a}$. This provides an apparent counterexample to the proposition, but as for Case 1-B above, by taking the right prolongation of $u_{a}$ the second time around, we perform an extended prefix turn, rendering the first right prolongation of $u_{a}$ invalid.

Case (i)


In case (ii), after we make a right prolongation turn (which may itself occur after a finite number of right prolongations), we may now take the required proper prefix. But as we are then required to take a proper suffix (in order to ensure that we finish the sequence taking a left prolongation), we need to make a turn. But as this would entail making an (extended) prefix turn, we conclude that case (ii) is also invalid.


As we have now accounted for all eight possible sequences, we can conclude that $\mathcal{W}$ is continuous.

## A. 2 Proposition 5.5.32

(Proposition 5.5.32) The two-sided left overlap division $\mathcal{W}$ is a Gröbner involutive division.

Proof: We are required to show that if Algorithm 12 terminates with $\mathcal{W}$ and some arbitrary admissible monomial ordering $O$ as input, then the Locally Involutive Basis $G$ it returns is a noncommutative Gröbner Basis. By Definition 3.1.8, we can do this by showing that all S-polynomials involving elements of $G$ conventionally reduce to zero using $G$.

Assume that $G=\left\{g_{1}, \ldots, g_{p}\right\}$ is sorted (by lead monomial) with respect to the DegRevLex monomial ordering (greatest first), and let $U=\left\{u_{1}, \ldots, u_{p}\right\}:=\left\{\operatorname{LM}\left(g_{1}\right), \ldots, \mathrm{LM}\left(g_{p}\right)\right\}$ be the set of leading monomials. Let $T$ be the table obtained by applying Algorithm 16 to $U$. Because $G$ is a Locally Involutive Basis, every zero entry $T\left(u_{i}, x_{j}^{\Gamma}\right)(\Gamma \in\{L, R\})$ in the table corresponds to a prolongation $g_{i} x_{j}$ or $x_{j} g_{i}$ that involutively reduces to zero.

Let $S$ be the set of $S$-polynomials involving elements of $G$, where the $t$-th entry of $S$ $(1 \leqslant t \leqslant|S|)$ is the S -polynomial

$$
s_{t}=c_{t} \ell_{t} g_{i} r_{t}-c_{t}^{\prime} \ell_{t}^{\prime} g_{j} r_{t}^{\prime},
$$

with $\ell_{t} u_{i} r_{t}=\ell_{t}^{\prime} u_{j} r_{t}^{\prime}$ being the overlap word of the $S$-polynomial. We will prove that every S-polynomial in $S$ conventionally reduces to zero using $G$.

Recall (from Definition 3.1.2) that each S-polynomial in $S$ corresponds to a particular type of overlap - 'prefix', 'subword' or 'suffix'. For the purposes of this proof, let us now split the subword overlaps into three further types - 'left', 'middle' and 'right', corresponding to the cases where a monomial $m_{2}$ is a prefix, proper subword and suffix of a monomial $m_{1}$.


This classification provides us with five cases to deal with in total, which we shall process in the following order: right, middle, left, prefix, suffix.
(1) Consider an arbitrary entry $s_{t} \in S(1 \leqslant t \leqslant|S|)$ corresponding to a right overlap where the monomial $u_{j}$ is a suffix of the monomial $u_{i}$. This means that $s_{t}=c_{t} g_{i}-c_{t}^{\prime} \ell_{t}^{\prime} g_{j}$ for some $g_{i}, g_{j} \in G$, with overlap word $u_{i}=\ell_{t}^{\prime} u_{j}$. Let $u_{i}=x_{i_{1}} \ldots x_{i_{\alpha}}$; let $u_{j}=x_{j_{1}} \ldots x_{j_{\beta}}$; and let $D=\alpha-\beta$.

$$
\begin{aligned}
& u_{i}=\quad \overline{x_{i_{1}}} \overline{x_{i_{2}}}---\overline{x_{i_{D}}} \overline{x_{i_{D+1}}} \overline{x_{i_{D+2}}}---\overline{x_{i_{\alpha-1}}} \overline{x_{i_{\alpha}}} \\
& u_{j}= \\
& \overline{x_{j_{1}}} \frac{\overline{x_{j_{2}}}}{}--\overline{x_{j_{\beta-1}}} \overline{x_{j_{\beta}}}
\end{aligned}
$$

Because $u_{j}$ is a suffix of $u_{i}$, it follows that $T\left(u_{j}, x_{i_{D}}^{L}\right)=0$. This gives rise to the prolongation $x_{i_{D}} g_{j}$ of $g_{j}$. But we know that all prolongations involutively reduce to zero ( $G$ is a Locally Involutive Basis), so Algorithm 10 must find a monomial $u_{k}=x_{k_{1}} \ldots x_{k_{\gamma}} \in U$ such that $u_{k}$ involutively divides $x_{i_{D}} u_{j}$. Assuming that $x_{k_{\gamma}}=x_{i_{\kappa}}$, we can deduce that any candidate for $u_{k}$ must be a suffix of $x_{i_{D}} u_{j}$ (otherwise $T\left(u_{k}, x_{i_{k+1}}^{R}\right)=0$ because of the overlap between $u_{i}$ and $u_{k}$ ). But if $u_{k}$ is a suffix of $x_{i_{D}} u_{j}$, then we must have $u_{k}=x_{i_{D}} u_{j}$ (otherwise $T\left(u_{k}, x_{i_{\alpha-\gamma}}^{L}\right)=0$ again because of the overlap between $u_{i}$ and $u_{k}$ ). We have therefore shown that there exists a monomial $u_{k}=x_{k_{1}} \ldots x_{k_{\gamma}} \in U$ such that $u_{k}$ is a suffix of $u_{i}$ and $\gamma=\beta+1$.

$$
\begin{aligned}
& u_{i}=\quad \overline{x_{i_{1}}} \overline{x_{i_{2}}}---\frac{}{x_{i_{D}}} \overline{x_{i_{D+1}}} \overline{x_{i_{D+2}}}---\overline{x_{i_{\alpha-1}}} \overline{x_{i_{\alpha}}} \\
& u_{j}= \\
& u_{k}= \\
& \frac{\overline{x_{j_{1}}}}{x_{k_{1}}} \frac{\overline{x_{j_{2}}}}{x_{k_{2}}} \frac{--\overline{x_{j_{\beta-1}}}}{\frac{x_{j_{\beta}}}{x_{k_{3}}}}---\frac{}{x_{k_{\gamma-1}}} \frac{}{x_{k_{\gamma}}}
\end{aligned}
$$

In the case $D=1$, it is clear that $u_{k}=u_{i}$, and so the first step in the involutive reduction of the prolongation $x_{i_{1}} g_{j}$ of $g_{j}$ is to take away the multiple $\left(\frac{c_{t}}{c_{t}}\right) g_{i}$ of $g_{i}$ from $x_{i_{1}} g_{j}$ to leave the polynomial $x_{i_{1}} g_{j}-\left(\frac{c_{t}}{c_{t}}\right) g_{i}=-\left(\frac{1}{c_{t}}\right) s_{t}$. But as we know that all prolongations involutively reduce to zero, we can conclude that the S-polynomial $s_{t}$ conventionally reduces to zero.

For the case $D>1$, we can use the monomial $u_{k}$ together with Buchberger's Second Criterion to simplify our goal of showing that the S-polynomial $s_{t}$ reduces to zero. Notice that the monomial $u_{k}$ is a subword of the overlap word $u_{i}$ associated to $s_{t}$, and so in order to show that $s_{t}$ reduces to zero, all we have to do is to show that the two S-polynomials

$$
s_{u}=c_{u} g_{i}-c_{u}^{\prime}\left(x_{i_{1}} x_{i_{2}} \ldots x_{i_{D-1}}\right) g_{k}
$$

and

$$
s_{v}=c_{v} g_{k}-c_{v}^{\prime} x_{i_{D}} g_{j}
$$

reduce to zero $(1 \leqslant u, v \leqslant|S|)$. But $s_{v}$ is an S-polynomial corresponding to a right overlap of type $D=1$ (because $\gamma-\beta=1$ ), and so $s_{v}$ reduces to zero. It remains to show that the S-polynomial $s_{u}$ reduces to zero. But we can do this by using exactly the same argument as above - we can show that there exists a monomial $u_{\pi}=x_{\pi_{1}} \ldots x_{\pi_{\delta}} \in U$ such that $u_{\pi}$ is a suffix of $u_{i}$ and $\delta=\gamma+1$, and we can deduce that the S -polynomial $s_{u}$ reduces to zero (and hence $s_{t}$ reduces to 0 ) if the S -polynomial

$$
s_{w}=c_{w} g_{i}-c_{w}^{\prime}\left(x_{i_{1}} x_{i_{2}} \ldots x_{i_{D-2}}\right) g_{\pi}
$$

reduces to zero $(1 \leqslant w \leqslant|S|)$. By induction, there is a sequence $\left\{u_{q_{D}}, u_{q_{D-1}}, \ldots, u_{q_{2}}\right\}$ of monomials increasing uniformly in degree, so that $s_{t}$ reduces to zero if the S-polynomial

$$
s_{\eta}=c_{\eta} g_{i}-c_{\eta}^{\prime} x_{i_{1}} g_{q_{2}}
$$

reduces to zero $(1 \leqslant \eta \leqslant|S|)$.


But $s_{\eta}$ is always an S-polynomial corresponding to a right overlap of type $D=1$, and so $s_{\eta}$ reduces to zero - meaning we can conclude that $s_{t}$ reduces to zero as well.
(2) Consider an arbitrary entry $s_{t} \in S(1 \leqslant t \leqslant|S|)$ corresponding to a middle overlap where the monomial $u_{j}$ is a proper subword of the monomial $u_{i}$. This means that $s_{t}=$ $c_{t} g_{i}-c_{t}^{\prime} \ell_{t}^{\prime} g_{j} r_{t}^{\prime}$ for some $g_{i}, g_{j} \in G$, with overlap word $u_{i}=\ell_{t}^{\prime} u_{j} r_{t}^{\prime}$. Let $u_{i}=x_{i_{1}} \ldots x_{i_{\alpha}}$; let $u_{j}=x_{j_{1}} \ldots x_{j_{\beta}}$; and choose $D$ such that $x_{i_{D}}=x_{j_{\beta}}$.

$$
\begin{array}{cc}
u_{i}= & \overline{x_{i_{1}}}--\overline{x_{i_{D-\beta}}} \frac{\overline{x_{i_{D-\beta+1}}} \overline{x_{i_{D-\beta+2}}}--\frac{}{x_{i_{D-1}}} \frac{}{x_{i_{D}}} \overline{x_{i_{D+1}}}--\frac{x_{i_{\alpha}}}{x_{j_{1}}} \frac{x_{j_{2}}}{x_{j_{\beta-1}}} \frac{x_{j_{\beta}}}{x_{j}}}{}=-\frac{x_{j_{\beta}}}{}
\end{array}
$$

Because $u_{j}$ is a proper subword of $u_{i}$, it follows that $T\left(u_{j}, x_{i_{D+1}}^{R}\right)=0$. This gives rise to
the prolongation $g_{j} x_{i_{D+1}}$ of $g_{j}$. But we know that all prolongations involutively reduce to zero, so there must exist a monomial $u_{k}=x_{k_{1}} \ldots x_{k_{\gamma}} \in U$ such that $u_{k}$ involutively divides $u_{j} x_{i_{D+1}}$. Assuming that $x_{k_{\gamma}}=x_{i_{\kappa}}$, any candidate for $u_{k}$ must be a suffix of $u_{j} x_{i_{D+1}}$ (otherwise $T\left(u_{k}, x_{i_{\kappa+1}}^{R}\right)=0$ because of the overlap between $u_{i}$ and $u_{k}$ ). Unlike part (1) however, we cannot determine the degree of $u_{k}$ (so that $1 \leqslant \gamma \leqslant \beta+1$ ); we shall illustrate this in the following diagram by using a squiggly line to indicate that the monomial $u_{k}$ can begin anywhere (or nowhere if $u_{k}=x_{i_{D+1}}$ ) on the squiggly line.

$$
\begin{aligned}
& u_{i}=\quad \frac{}{x_{i_{1}}}---\frac{}{x_{i_{D-\beta}}} \overline{x_{i_{D-\beta+1}}} \overline{x_{i_{D-\beta+2}}}---\overline{x_{i_{D-1}}} \frac{}{x_{i_{D}}} \frac{}{x_{i_{D+1}}}---\frac{x_{i_{\alpha}}}{} \\
& u_{j}= \\
& u_{k}=
\end{aligned}
$$

We can now use the monomial $u_{k}$ together with Buchberger's Second Criterion to simplify our goal of showing that the S-polynomial $s_{t}$ reduces to zero. Notice that the monomial $u_{k}$ is a subword of the overlap word $u_{i}$ associated to $s_{t}$, and so in order to show that $s_{t}$ reduces to zero, all we have to do is to show that the two S-polynomials

$$
s_{u}=c_{u} g_{i}-c_{u}^{\prime}\left(x_{i_{1}} x_{i_{2}} \ldots x_{i_{D+1-\gamma}}\right) g_{k}\left(x_{i_{D+2}} \ldots x_{i_{\alpha}}\right)
$$

and $^{1}$

$$
s_{v}=c_{v}\left(x_{j_{1}} \ldots x_{i_{D+1-\gamma}}\right) g_{k}-c_{v}^{\prime} g_{j} x_{i_{D+1}}
$$

reduce to zero $(1 \leqslant u, v \leqslant|S|)$.
For the S-polynomial $s_{v}$, there are two cases to consider: $\gamma=1$, and $\gamma>1$. In the former case, because (as placed in $u_{i}$ ) the monomials $u_{j}$ and $u_{k}$ do not overlap, we can use Buchberger's First Criterion to say that the 'S-polynomial' $s_{v}$ reduces to zero (for further explanation, see the paragraph at the beginning of Section 3.4.1). In the latter case, note that $u_{k}$ is the only involutive divisor of the prolongation $u_{j} x_{i_{D+1}}$, as the existence of any suffix of $u_{j} x_{i_{D+1}}$ of higher degree than $u_{k}$ in $U$ will contradict the fact that $u_{k}$ is an involutive divisor of $u_{j} x_{i_{D+1}}$; and the existence of $u_{k}$ in $U$ ensures that any suffix of $u_{j} x_{i_{D+1}}$ that exists in $U$ with a lower degree than $u_{k}$ will not be an involutive divisor of $u_{j} x_{i_{D+1}}$. This means that the first step of the involutive reduction of $g_{j} x_{i_{D+1}}$ is to take away the multiple $\left(\frac{c_{v}}{c_{v}^{\prime}}\right)\left(x_{j_{1}} \ldots x_{i_{D+1-\gamma}}\right) g_{k}$ of $g_{k}$ from $g_{j} x_{i_{D+1}}$ to leave the polynomial $g_{j} x_{i_{D+1}}-$

[^13]$\left(\frac{c_{v}}{c_{v}}\right)\left(x_{j_{1}} \ldots x_{i_{D+1-\gamma}}\right) g_{k}=-\left(\frac{1}{c_{v}^{\prime}}\right) s_{v}$. But as we know that all prolongations involutively reduce to zero, we can conclude that the S-polynomial $s_{v}$ conventionally reduces to zero.

For the $S$-polynomial $s_{u}$, we note that if $D=\alpha-1$, then $s_{u}$ corresponds to a right overlap, and so we know from part (1) that $s_{u}$ conventionally reduces to zero. Otherwise, we proceed by induction on the S-polynomial $s_{u}$ to produce a sequence $\left\{u_{q_{D+1}}, u_{q_{D+2}}, \ldots, u_{q_{\alpha}}\right\}$ of monomials, so that $s_{u}$ (and hence $s_{t}$ ) reduces to zero if the S-polynomial

$$
s_{\eta}=c_{\eta} g_{i}-c_{\eta}^{\prime}\left(x_{i_{1}} \ldots x_{i_{\alpha-\mu}}\right) g_{q_{\alpha}}
$$

reduces to zero $(1 \leqslant \eta \leqslant|S|)$, where $\mu=\operatorname{deg}\left(u_{q_{\alpha}}\right)$.

$$
\begin{aligned}
& u_{i}= \\
& \overline{x_{i_{1}}}---\overline{x_{i_{D-\beta}}} \overline{x_{D-\beta+1}}---\frac{}{x_{i_{D}}} \overline{x_{i_{D+1}}} \overline{x_{i_{D+2}}}---\overline{x_{i_{\alpha-1}}} \frac{}{x_{i_{\alpha}}} \\
& u_{j}= \\
& u_{q_{D+1}}=u_{k}= \\
& u_{q_{D+2}}= \\
& u_{q_{\alpha}}=
\end{aligned}
$$

But $s_{\eta}$ always corresponds to a right overlap, and so $s_{\eta}$ reduces to zero - meaning we can conclude that $s_{t}$ reduces to zero as well.
(3) Consider an arbitrary entry $s_{t} \in S(1 \leqslant t \leqslant|S|)$ corresponding to a left overlap where the monomial $u_{j}$ is a prefix of the monomial $u_{i}$. This means that $s_{t}=c_{t} g_{i}-c_{t}^{\prime} g_{j} r_{t}^{\prime}$ for some $g_{i}, g_{j} \in G$, with overlap word $u_{i}=u_{j} r_{t}^{\prime}$. Let $u_{i}=x_{i_{1}} \ldots x_{i_{\alpha}}$ and let $u_{j}=x_{j_{1}} \ldots x_{j_{\beta}}$.

$$
\begin{aligned}
& u_{i}=\quad \overline{x_{i_{1}}} \overline{x_{i_{2}}}---\frac{}{x_{i_{\beta-1}}} \overline{x_{i_{\beta}}} \overline{x_{i_{\beta+1}}}---\overline{x_{i_{\alpha-1}}} \overline{x_{i_{\alpha}}} \\
& u_{j}= \\
& \overline{x_{j_{1}}} \overline{x_{j_{2}}}---\overline{x_{j_{\beta-1}}} \overline{x_{j_{\beta}}}
\end{aligned}
$$

Because $u_{j}$ is a prefix of $u_{i}$, it follows that $T\left(u_{j}, x_{i_{\beta+1}}^{R}\right)=0$. This gives rise to the prolongation $g_{j} x_{i_{\beta+1}}$ of $g_{j}$. But we know that all prolongations involutively reduce to zero, so there must exist a monomial $u_{k}=x_{k_{1}} \ldots x_{k_{\gamma}} \in U$ such that $u_{k}$ involutively divides $u_{j} x_{i_{\beta+1}}$. Assuming that $x_{k_{\gamma}}=x_{i_{\kappa}}$, any candidate for $u_{k}$ must be a suffix of $u_{j} x_{i_{\beta+1}}$
(otherwise $T\left(u_{k}, x_{i_{\kappa+1}}^{R}\right)=0$ because of the overlap between $u_{i}$ and $\left.u_{k}\right)$.

$$
\begin{array}{ll}
u_{i}= & \overline{x_{i_{1}}} \frac{}{x_{i_{2}}}---\overline{\frac{x_{i_{\beta-1}}}{} \frac{}{x_{i_{\beta}}} \frac{}{x_{i_{\beta+1}}}---\overline{x_{i_{\alpha-1}}} \frac{}{x_{i_{\alpha}}}} \begin{array}{l}
u_{j} \\
u_{k}
\end{array} \quad \overline{x_{j_{1}}} \overline{x_{j_{2}}}--\frac{x_{j_{\beta-1}}}{x_{j_{\beta}}} \\
\hline \sim \sim \sim & \\
x_{k_{\gamma}}
\end{array}
$$

If $\alpha=\gamma$, then it is clear that $u_{k}=u_{i}$, and so the first step in the involutive reduction of the prolongation $g_{j} x_{i_{\alpha}}$ is to take away the multiple $\left(\frac{c_{t}}{c_{t}}\right) g_{i}$ of $g_{i}$ from $g_{j} x_{i_{\alpha}}$ to leave the polynomial $g_{j} x_{i_{\alpha}}-\left(\frac{c_{t}}{c_{t}}\right) g_{i}=-\left(\frac{1}{c_{t}^{t}}\right) s_{t}$. But as we know that all prolongations involutively reduce to zero, we can conclude that the S-polynomial $s_{t}$ conventionally reduces to zero.

Otherwise, if $\alpha>\gamma$, we can now use the monomial $u_{k}$ together with Buchberger's Second Criterion to simplify our goal of showing that the S-polynomial $s_{t}$ reduces to zero. Notice that the monomial $u_{k}$ is a subword of the overlap word $u_{i}$ associated to $s_{t}$, and so in order to show that $s_{t}$ reduces to zero, all we have to do is to show that the two S -polynomials

$$
s_{u}=c_{u} g_{i}-c_{u}^{\prime}\left(x_{i_{1}} \ldots x_{i_{\beta+1-\gamma}}\right) g_{k}\left(x_{i_{\beta+2}} \ldots x_{i_{\alpha}}\right)
$$

and

$$
s_{v}=c_{v}\left(x_{i_{1}} \ldots x_{i_{\beta+1-\gamma}}\right) g_{k}-c_{v}^{\prime} g_{j} x_{i_{\beta+1}}
$$

reduce to zero $(1 \leqslant u, v \leqslant|S|)$.
The S-polynomial $s_{v}$ reduces to zero by comparison with part (2). For the S-polynomial $s_{u}$, first note that if $\alpha=\beta+1$, then $s_{u}$ corresponds to a right overlap, and so we know from part (1) that $s_{u}$ conventionally reduces to zero. Otherwise, if $\gamma \neq \beta+1$, then $s_{u}$ corresponds to a middle overlap, and so we know from part (2) that $s_{u}$ conventionally reduces to zero. This leaves the case where $s_{u}$ corresponds to another left overlap, in which case we proceed by induction on $s_{u}$, eventually coming across either a middle overlap or a right overlap because we move one letter at a time to the right after each inductive step.
(4 and 5) In Definition 3.1.2, we defined a prefix overlap to be an overlap where, given two monomials $m_{1}$ and $m_{2}$ such that $\operatorname{deg}\left(m_{1}\right) \geqslant \operatorname{deg}\left(m_{2}\right)$, a prefix of $m_{1}$ is equal to a suffix of $m_{2}$; suffix overlaps were defined similarly. If we drop the condition on the degrees of the monomials, it is clear that every suffix overlap can be treated as a prefix overlap (by swapping the roles of $m_{1}$ and $m_{2}$ ); this allows us to deal with the case of a prefix overlap only.

Consider an arbitrary entry $s_{t} \in S(1 \leqslant t \leqslant|S|)$ corresponding to a prefix overlap where a prefix of the monomial $u_{i}$ is equal to a suffix of the monomial $u_{j}$. This means that $s_{t}=c_{t} \ell_{t} g_{i}-c_{t}^{\prime} g_{j} r_{t}^{\prime}$ for some $g_{i}, g_{j} \in G$, with overlap word $\ell_{t} u_{i}=u_{j} r_{t}^{\prime}$. Let $u_{i}=x_{i_{1}} \ldots x_{i_{\alpha}}$; let $u_{j}=x_{j_{1}} \ldots x_{j_{\beta}}$; and choose $D$ such that $x_{i_{D}}=x_{j_{\beta}}$.


By definition of $\mathcal{W}$, at least one of $T\left(u_{i}, x_{j_{\beta-D}}^{L}\right)$ and $T\left(u_{j}, x_{i_{D+1}}^{R}\right)$ is equal to zero.

- Case $T\left(u_{j}, x_{i_{D+1}}^{R}\right)=0$.

Because we know that the prolongation $g_{j} x_{i_{D+1}}$ involutively reduces to zero, there must exist a monomial $u_{k}=x_{k_{1}} \ldots x_{k_{\gamma}} \in U$ such that $u_{k}$ involutively divides $u_{j} x_{i_{D+1}}$. This $u_{k}$ must be a suffix of $u_{j} x_{i_{D+1}}$ (otherwise, assuming that $x_{k_{\gamma}}=x_{j_{\kappa}}$, we have $T\left(u_{k}, x_{i_{D+1}}^{R}\right)=0$ if $\gamma=\beta$ (because of the overlap between $u_{i}$ and $\left.u_{k}\right) ; T\left(u_{k}, x_{j_{\beta-\gamma}}^{L}\right)=$ 0 if $\gamma<\beta$ and $\kappa=\beta$ (because of the overlap between $u_{j}$ and $u_{k}$ ); and $T\left(u_{k}, x_{j_{k+1}}^{R}\right)=$ 0 if $\gamma<\beta$ and $\kappa<\beta$ (again because of the overlap between $u_{j}$ and $u_{k}$ )).

$$
\begin{array}{ll}
u_{i}= \\
u_{j} & = \\
u_{k} & =\frac{}{x_{j_{1}}}---\frac{}{x_{i_{1}}}-\cdots \frac{x_{i_{D}}}{x_{j_{\beta-D}}} \overline{x_{j_{\beta-D+1}}}-\cdots \frac{x_{i_{D+1}}}{x_{j_{\beta}}} \\
\sim \sim
\end{array}
$$

Let us now use the monomial $u_{k}$ together with Buchberger's Second Criterion to simplify our goal of showing that the S-polynomial $s_{t}$ reduces to zero. Because $u_{k}$ is a subword of the overlap word $\ell_{t} u_{i}$ associated to $s_{t}$, in order to show that $s_{t}$ reduces to zero, all we have to do is to show that the two S-polynomials

$$
s_{u}= \begin{cases}c_{u}\left(x_{k_{1}} \ldots x_{j_{\beta-D}}\right) g_{i}-c_{u}^{\prime} g_{k}\left(x_{i_{D+2}} \ldots x_{i_{\alpha}}\right) & \text { if } \gamma>D+1 \\ c_{u} g_{i}-c_{u}^{\prime} \ell_{u}^{\prime} g_{k}\left(x_{i_{D+2}} \ldots x_{i_{\alpha}}\right) & \text { if } \gamma \leqslant D+1\end{cases}
$$

and

$$
s_{v}=c_{v} g_{j} x_{i_{D+1}}-c_{v}^{\prime}\left(x_{j_{1}} \ldots x_{j_{\beta+1-\gamma}}\right) g_{k}
$$

reduce to zero $(1 \leqslant u, v \leqslant|S|)$.
The S-polynomial $s_{v}$ reduces to zero by comparison with part (2). For the Spolynomial $s_{u}$, first note that if $\alpha=D+1$, then either $u_{k}$ is a suffix of $u_{i}, u_{i}$ is a suffix of $u_{k}$, or $u_{k}=u_{i}$; it follows that $s_{u}$ reduces to zero trivially if $u_{k}=u_{i}$, and $s_{u}$ reduces to zero by part (1) in the other two cases.

If however $\alpha \neq D+1$, then either $s_{u}$ is a middle overlap (if $\gamma<D+1$ ), a left overlap (if $\gamma=D+1$ ), or another prefix overlap. The first two cases can be handled by parts (2) and (3) respectively; the final case is handled by induction, where we note that after each step of the induction, the value $\alpha+\beta-2 D$ strictly decreases (regardless of which case $T\left(u_{j}, x_{i_{D+1}}^{R}\right)=0$ or $T\left(u_{i}, x_{j_{\beta-D}}^{L}\right)=0$ applies), so we are guaranteed at some stage to find an overlap that is not a prefix overlap, enabling us to verify that the S-polynomial $s_{t}$ conventionally reduces to zero.

- Case $T\left(u_{i}, x_{j_{\beta-D}}^{L}\right)=0$.

Because we know that the prolongation $x_{j_{\beta-D}} g_{i}$ involutively reduces to zero, there must exist a monomial $u_{k}=x_{k_{1}} \ldots x_{k_{\gamma}} \in U$ such that $u_{k}$ involutively divides $x_{j_{\beta-D}} u_{i}$. This $u_{k}$ must be a prefix of $x_{j_{\beta-D}} u_{i}$ (otherwise, assuming that $x_{k_{\gamma}}=x_{i_{\kappa}}$, we have $T\left(u_{k}, x_{j_{\beta-D}}^{L}\right)=0$ if $\gamma=\alpha$ (because of the overlap between $u_{j}$ and $\left.u_{k}\right) ; T\left(u_{k}, x_{i_{\kappa-\gamma}}^{L}\right)=$ 0 if $\gamma<\alpha$ and $\kappa=\alpha$ (because of the overlap between $u_{i}$ and $u_{k}$ ); and $T\left(u_{k}, x_{i_{\kappa+1}}^{R}\right)=0$ if $\gamma<\alpha$ and $\kappa<\alpha$ (again because of the overlap between $u_{i}$ and $u_{k}$ )).

$$
\begin{array}{ll}
u_{i} & =\frac{}{x_{i_{1}}}--\frac{x_{i_{D}}}{x_{i_{D+1}}}---\frac{}{x_{i_{\alpha-1}}} \frac{x_{i_{\alpha}}}{x_{i_{\alpha}}} \\
u_{j} & = \\
u_{k} & =-\frac{x_{j_{1}}}{x_{j_{\beta-D}}}---\frac{x_{j_{\beta}}}{x_{k_{\beta}-D+1}}
\end{array}
$$

Let us now use the monomial $u_{k}$ together with Buchberger's Second Criterion to simplify our goal of showing that the S-polynomial $s_{t}$ reduces to zero. Because $u_{k}$ is a subword of the overlap word $\ell_{t} u_{i}$ associated to $s_{t}$, in order to show that $s_{t}$ reduces to zero, all we have to do is to show that the two S-polynomials

$$
s_{u}=c_{u} x_{k_{1}} g_{i}-c_{u}^{\prime} g_{k}\left(x_{i_{\gamma}} \ldots x_{i_{\alpha}}\right)
$$

and

$$
s_{v}= \begin{cases}c_{v} g_{j}\left(x_{i_{D+1}} \ldots x_{k_{\gamma}}\right)-c_{v}^{\prime}\left(x_{j_{1}} \ldots x_{j_{\beta-D-1}}\right) g_{k} & \text { if } \gamma>D+1 \\ c_{v} g_{j}-c_{v}^{\prime}\left(x_{j_{1}} \ldots x_{j_{\beta-D-1}}\right) g_{k} r_{v}^{\prime} & \text { if } \gamma \leqslant D+1\end{cases}
$$

reduce to zero $(1 \leqslant u, v \leqslant|S|)$.
The S-polynomial $s_{u}$ reduces to zero by comparison with part (2). For the Spolynomial $s_{v}$, first note that if $\beta-D=1$, then either $u_{k}$ is a prefix of $u_{j}, u_{j}$ is a prefix of $u_{k}$, or $u_{k}=u_{j}$; it follows that $s_{v}$ reduces to zero trivially if $u_{k}=u_{j}$, and $s_{v}$ reduces to zero by part (3) in the other two cases.

If however $\beta-D \neq 1$, then either $s_{v}$ is a middle overlap (if $\gamma<D+1$ ), a right overlap (if $\gamma=D+1$ ), or another prefix overlap. The first two cases can be handled by parts (2) and (1) respectively; the final case is handled by induction, where we note that after each step of the induction, the value $\alpha+\beta-2 D$ strictly decreases (regardless of which case $T\left(u_{j}, x_{i_{D+1}}^{R}\right)=0$ or $T\left(u_{i}, x_{j_{\beta-D}}^{L}\right)=0$ applies), so we are guaranteed at some stage to find an overlap that is not a prefix overlap, enabling us to verify that the S-polynomial $s_{t}$ conventionally reduces to zero.

## Appendix B

## Source Code

In this Appendix, we will present ANSI C source code for an initial implementation of the noncommutative Involutive Basis algorithm (Algorithm 12), together with an introduction to AlgLib, a set of ANSI C libraries providing data types and functions that serve as building blocks for the source code.

## B. 1 Methodology

A problem facing anyone wanting to implement mathematical ideas is the choice of language or system in which to do the implementation. The decision depends on the task at hand. If all that is required is a convenient environment for prototyping ideas, a symbolic computation system such as Maple [55], Mathematica [57] or MuPAD [49] may suffice. Such systems have a large collection of mathematical data types, functions and algorithms already present; tools that will not be available in a standard programming language. There is however always a price to pay for convenience. These common systems are all interpreted and use a proprietary programming syntax, making it it difficult to use other programs or libraries within a session. It also makes such systems less efficient than the execution of compiled programs.

The AlgLib libraries can be said to provide the best of both worlds, as they provide data types, functions and algorithms to allow programmers to more easily implement certain mathematical algorithms (including the algorithms described in this thesis) in the ANSI C programming language. For example, AlgLib contains the FMon [41] and FAlg [40]
libraries, respectively containing data types and functions to perform computations in the free monoid on a set of symbols and the free associative algebra on a set of symbols. Besides the benefit of the efficiency of compiled programs, the strict adherence to ANSI C makes programs written using the libraries highly portable.

## B.1.1 MSSRC

AlgLib is supplied by MSSRC [46], a company whose Chief Scientist is Prof. Larry Lambe, an honorary professor at the University of Wales, Bangor. For an introduction to MSSRC, we quote the following passage from [42].

Multidisciplinary Software Systems Research Corporation (MSSRC) was conceived as a company devoted to furthering the long-term effective use of mathematics and mathematical computation. MSSRC researches, develops, and markets advanced mathematical tools for engineers, scientists, researchers, educators, students and other serious users of mathematics. These tools are based on providing levels of power, productivity and convenience far greater than existing tools while maintaining mathematical rigor at all times. The company also provides computer education and training.

MSSRC has several lines of ANSI C libraries for providing mathematical support for research and implementation of mathematical algorithms at various levels of complexity. No attempt is made to provide the user of these libraries with any form of Graphical User Interface (GUI). All components are compiled ANSI C functions which represent various mathematical operations from basic (adding, subtracting, multiplying polynomials, etc.) to advanced (operations in the free monoid on an arbitrary number of symbols and beyond). In order to use the libraries effectively, the user must be expert at ANSI C programming, e.g., in the style of Kernighan and Richie [38] and as such, they are not suited for the casual user. This does not imply in any way that excellent user interfaces for applications of the libraries cannot be supplied or are difficult to implement by well experienced programmers.

The use of MSSRC's libraries has been reported in a number of places such as [43], [14], [16], [15] and elsewhere.

## B.1.2 AlgLib

To give a taste of how AlgLib has been used to implement the algorithms considered in this thesis, consider one of the basic operations of these algorithms, the task of subtracting two polynomials to yield a third polynomial (an operation essential for computing an S-polynomial). In ordinary ANSI C, there is no data type for a polynomial, and certainly no function for subtracting two polynomials; AlgLib however does supply these data types and functions, both in the commutative and noncommutative cases. For example, the AlgLib data type for a noncommutative polynomial is an FAlg, and the AlgLib function for subtracting two such polynomials is the function fAlgMinus. It follows that we can write ANSI C code for subtracting two noncommutative polynomials, as illustrated below where we subtract the polynomial $2 b^{2}+a b+4 b$ from the polynomial $2 \times\left(b^{2}+b a+3 a\right)$.

## Source Code

```
# include <fralg.h>
int
main( argc, argv )
int argc;
char *argv[];
{
    // Define Variables
    FAlg p, q, r;
    QInteger two;
    // Set Monomial Ordering (DegLex)
    theOrdFun = fMonTLex;
    // Initialise Variables
```




```
    two = parseStrToQ("2");
    // Perform the calculation and display the result on screen
    r = fAlgMinus(fAlgScaTimes( two, p ), q );
    printf("2*(%s)
    return EXIT_SUCCESS;
}
```


## Program Output

```
ma6:mssrc-aux/thesis > fAlgMinusTest
2*(b^2 + b a + 3a) - (2 b^2 + a b + 4b) =2 b a - ab -4b+6a
ma6:mssrc-aux/thesis>
```


## B. 2 Listings

Our implementation of the noncommutative Involutive Basis algorithm is arranged as follows: involutive.c is the main program, dealing with all the input and output and calling the appropriate routines; the '-functions' files contain all the procedures and functions used by the program; and README describes how to use the program, including what format the input files should take and what the different options of the program are used for.

In more detail, arithmetic_functions.c contains functions for dividing a polynomial by its (coefficient) greatest common divisor and for converting user specified generators to ASCII generators (and vice-versa); file_functions.c contains all the functions needed to read and write polynomials and variables to and from disk; fralg_functions.c contains functions for monomial orderings, polynomial division and reduced Gröbner Bases computation; list_functions.c contains some extra functions needed to deal with displaying, sorting and manipulating lists; and ncinv_functions.c contains all the involutive routines, for example the Involutive Basis algorithm itself and associated functions for determining multiplicative variables and for performing autoreduction.

## Contents


B.2.2 arithmetic_functions.h ..................................................... 219
B.2.3 arithmetic_functions.c .................................................. 220
B.2.4 file_functions.h ............................................................ . . 226
B.2.5 file_functions.c ................................................................ 227
B.2.6 fralg_functions.h ...................................................... 238
B.2.7 fralg_functions.c ......................................................... . . 240
B.2.8 list_functions.h ............................................................. 268
B.2.9 list_functions.c ........................................................... . . 271
B.2.10 ncinv_functions.h ........................................................... 290
B.2.11 ncinv_functions.c ........................................................ . 291
B.2.12 involutive.c ..................................................................... 340

## B.2.1 README

* HOW TO USE THE INVOLUTIVE PROGRAM - QUICK GUIDE *

NAME
involutive - Computes Noncommutative Involutive Bases for ideals

## SYNOPSIS

involutive [OPTION]... [FILE]...

## DESCRIPTION

Here are the options for the program.

```
-a
```

e.g. $>$ involutive $-\mathrm{d}-$ a file. in

Optimises the lexicographical ordering according to the frequency of the variables in the input basis (most frequent $=$ lexicographically smallest). $-c(n)$
e. g. > involutive -c2 file .in

Chooses which involutive algorithm to use.
n is a required number between 1 and 2 .

1: *DEFAULT * Gerdt's Algorithm
2: Seiler 's Algorithm $-\mathrm{d}$
e. g. $>$ involutive - d file. in

Allows the user to calculate a DegLex
Involutive Basis for the basis in file .in. $-\mathrm{e}(\mathrm{n})$
e. g. $>$ involutive $-\mathrm{e} 2-\mathrm{s} 2$ file.in

Allows the user to select the type of Overlap
Division to use. n is a required number between
1 and 5 . Note: Must be used with either the
-s1 or -s2 options.

Left Overlap Division:

| A | B | C | D |
| :---: | :---: | :---: | :---: |
| x | -- |  |  |

1: * DEFAULT * A, B, C (weak, Gr $\backslash$ "obner)
2: A, B, C, Strong (strong if used with -m 2 )
3: A, B , C, D (weak, Gr \"obner)
4: A, B (weak, Gr\"obner)
5: A (weak, Gr\"obner)

Right Overlap Division:

| A | B | C | D |
| :---: | :---: | :---: | :---: |

1: * DEFAULT * A, B, C (weak, Gr \"obner)
2: A, B, C, Strong (strong if used with -m 2 )
3: A, B, C, D (weak, Gr $\$ "obner)
4: A, B (weak, Gr\"obner)
5: A (weak, Gr $\backslash$ "obner)
-f
e.g. $>$ involutive -f file .in

Removes any fractions from the input basis
e.g. $>$ involutive -1 file .in

Allows the user to calculate a Lex
Involutive Basis for the basis in file .in.
Warning: program may go into an infinite loop
(Lex is not an admissible monomial ordering).
$-m(n)$
e. g. $>$ involutive -m 2 file.in

Selects which method of deciding whether a monomial
involutively divides another monomial is used.
n is a required number between 1 and 2 .

1: * DEFAULT * 1st letters on left and right (thin divisor )
2: All letters on left and right (thick divisor)
-o(n)
e. g. $>$ involutive $-o 2$ file. in

Allows the user to select how the basis is sorted
during the algorithm. n is a required number between
1 and 3.

1: * DEFAULT * DegRevLex Sorted
2: No Sorting
3: Sorting by Main Ordering
$-p$
e. g. $>$ involutive $-1-\mathrm{p}$ file.in

An interactive Ideal Membership Problem Solver.
There are two ways the solver can be used:
either a file containing a list of polynomials
(e. g. $x * y-z$; $x^{\wedge} 2-z^{\wedge} 2+y^{\wedge} 2$; ) can be given, or the
polynomials can be input
manually (e.g. $x * y-z$ ). The solver tests to see
whether the Involutive Basis computed in the
algorithm reduces the polynomials given to zero.

```
-r * DEFAULT *
    e.g. > involutive -r file.in
    Allows the user to calculate a DegRevLex
    Involutive Basis for the basis in file .in
-s(n)
    e.g. > involutive -s2 file .in
    Allows the user to select the type of Involutive
    Basis to calculate. n is a required number between
    1 and 5. Note: If an 'Overlap' Division is selected,
    the type of Overlap Division can be chosen with
    the -e(n) option.
    1: Left Overlap Division (local, cts, see -e option)
    2: Right Overlap Division (local, cts, see -e option)
    : * DEFAULT * Left Division (global, cts, strong)
    : Right Division (global, cts, strong)
    : Empty Division (global, cts, strong)
-v(n)
    e.g. > involutive -v3 file.in
    Changes the amount of information given out by the
    program (i.e. the 'verbosity' of the program).
    n}\mathrm{ is a number between 0 and 9. Rough Guide:
    : Silent (no output given)
    * DEFAULT *
    Returns Number of Reductions Carried Out,
        Prints Out Every Polynomial Found
    3: More Autoreduction Information,
        Prolongation Information
    4: More Details of Steps Taken in Algorithm
    5: More Global Division Information
    6: Step-by-Step Reduction, Overlap Information
    7: Shows Multiplicative Grids
    8: More Overlap Division Information
    9: All Other Information
-w
    e.g. > involutive -w file.in
    Allows the user to calculate an Involutive Basis
    for the basis in file .in using the Wreath
    Product Monomial Ordering.
-x
    e.g. > involutive -x file.in
    Ignores any prolongations of degree greater than or
    equal to 2d, where d is a value determined by the degree
    of the largest degree lead monomial in the current minimal basis.
    Warning: May not return a valid Involutive Basis
    (only a valid Gr\"obner Basis)
```

FILE FORMATS

There is one file format for the input basis:

IDEALS:
$x ; y ; z ;$
$\mathrm{x} * \mathrm{y}-\mathrm{z}$;
$2 * \mathrm{x}+\mathrm{y} * \mathrm{z}+\mathrm{z}$;

First line $=$ List of variables in order. In the
above, $x ; y ; z$; represents $x>y>z$.
Remaining lines $=$ Polynomial generators (which must be terminated by semicolons).

## OUTPUT

As output, the program provides a reduced $\mathrm{Gr} \backslash$ "obner Basis and an Involutive Basis for the input ideal (if it can calculate it).

For the following, assume that our input basis was given as file .in.

* If a DegRevLex Gr\"obner Basis is calculated, it is stored as file . drl.
* If a DegLex Gr $\$ "obner Basis is calculated, it is stored as file .deg.
* If a Lex $\mathrm{Gr} \backslash$ "obner Basis is calculated, it is stored as file .lex.
* If a Wreath Product Gr $\$ "obner Basis is calculated, it is stored as file . wp.

The Involutive Basis is given as <Gr $\langle$ "obner Basis>.inv.
For example, if a DegLex Involutive Basis is calculated,
it is stored as file . deg.inv.

Note that the program has the ability to recognise the .in suffix and replace it with . drl, . deg, . lex or . wp as necessary.
If your input file does not have a .in suffix then the program will
simply append the appropriate suffix onto the end of the file name.
For example, using the command
$>$ involutive FILE
we obtain file . drl if FILE $=$ file.in
and obtain e.g. file . other. drl if FILE $=$ file.other.

## B.2.2 arithmetic_functions.h

```
/*
* File: arithmetic_functions.h
* Author: Gareth Evans
* Last Modified: 29th September 2004
*/
// Initialise file definition
& # ifndef ARITHMETIC_FUNCTIONS_HDR
# define ARITHMETIC_FUNCTIONS_HDR
```

```
// Include MSSRC Libraries
# include <fralg.h>
//
// Numerical Functions
//
// Returns the numerical value of a 3 letter word
ULong ASCIIVal( String );
// Returns the 3 letter word of a numerical value.
String ASCIIStr( ULong );
// Returns the monomial corresponding to the & letter word of a numerical value
FMon ASCIIMon(ULong);
//
// QInteger Functions
//
// Calculate Alternative LCM of 2 QIntegers
QInteger AltLCMQInteger( QInteger, QInteger );
//
// FAlg Functions
//
// Divides the input. FAlg by its common GCD
FAlg findGCD( FAlg );
// Returns maximal degree of leud term for the given FAlgList
ULong maxDegree( FAlgList );
// Returns the position of the smallest LM(g) in the given FAlgList
ULong fAlgListLowest( FAlgList );
# endif // ARITHMETIC_FUNCTIONS_HDR
```


## B.2.3 arithmetic_functions.c

```
/*
* File: arithmetic_functions.c
* Author: Gareth Evans
* Last Modified: 11th February 2005
*/
/*
* =====================
* Numerical Functions
* ======================
*/
**
* Function Name: ASCIIVal
*
* Overview: Returns the numerical value of a . 3 letter word
```

```
* Detail: Given a String containing % letters from the set
* {A,B,\ldots,Z}, this function returns the numerical
* value of the String according to the following rule:
* AAA = 1, AAB =2, .., AAZ = 26, ABA = 27, ABB=28,
* .., ABZ = 52, ACA = 53,\ldots
*
*/
ULong
ASCIIVal(word)
String word;
{
ULong back = 0;
// Add on 17576**value of 1st, letter ( }A=0,B=1,\ldots
back = back + 17576*((ULong)( (int)word[0] - (int)'A'));
// Add on 26*value of 2nd letter ( }A=0,B=1,\ldots
back = back + 26*((ULong)( (int)word[1] - (int)'A' ) );
// Add on the value of the 3rd letter ( }A=1,B=2,\ldots
back = back + (ULong)( (int)word[2] - (int)'A' + 1 );
return back;
}
/*
* Function Name: ASCIIStr
* Overview: Returns the 3 letter word of a numerical value
*
* Detail: Given a ULong, this function returns the
* 3 letter String corresponding to the following rule:
* 1 = AAA,2 = AAB, ., 26=AAZ, 27 = ABA,28=ABB,
* ...,52 = ABZ, 53 = ACA, ..
*
*/
String
ASCIIStr( number )
ULong number;
{
String back = strNew();
int i = 0, j = 0, k;
// Take away multiples of 20^2 to get the first letter
while( number > 17576 )
{
    i++;
    number = number - 17576;
}
// Take away multiples of 26 to get the second letter
while( number > 26 )
{
    j++;
```

```
    number = number - 26;
}
// We are now left. with the third letter
k = (int) number - 1;
// Convert the numbers to a String
sprintf( back, "%c%c%%c", (char)( (int)'A' + i ),
    (char)( (int)'A' + j ),
    (char)((int)'A' + k ) );
// Return the three letters
return back;
}
/*
* Function Name: ASCIIStr
*
* Overvicw: Returns the monomial corresponding to the
* 3 letter word of a numerical value
*
* Detail: Given a ULong, this function returns the
* monomial corresponding to the following rule:
* 1 = AAA,2 = AAB, ., 26=AAZ,27=ABA,28=ABB.
* ...,52 = ABZ, 53 = ACA, ..
*
*/
FMon
ASCIIMon( number )
ULong number;
{
// Obtain the String corresponding to the input.
// number and change it to an FMon
return parseStrToFMon( ASCIIStr( number ) );
}
* =====================
* QInteger Functions
* ===========================
*/
/*
* Function Name: AltLCMQInteger
*
* Overview: Calculates an 'alternative' LCM of 2 QIntegers
*
* Detail: Given two QIntegers a =an/ad and b = bn/bd,
* this function calculates the LCM given
* by alt_lcm(a,b) = (a*b)/(alt_gcd(a,b))
* = (an*bn*ad*bd)/(ad*bd*gcd(an, bn)*gcd(ad,bd))
* ==(an*bn)/(gcd(an,bn)*gcd(ad,bd)).
*
```

105
06 /*

```
*/
QInteger
AltLCMQInteger( a, b )
QInteger a, b;
{
Integer an =a }->>\mathrm{ num,
        ad =a -> den,
        bn = b -> num,
        bd = b }->\mathrm{ den;
    return qDivide( zToQ( zTimes(an, bn ) ),
                zToQ(zTimes(zGcd(an, bn ), zGcd(ad, bd ) ) ) );
}
/*
* ================
* FAlg Functions
* ===============
*/
/*
* Function Name: findGCD
*
* Overview: Divides the input FAlg by its common GCD
*
* Detail: Given an FAlg, this function divides the
* polynomial by its common GCD so that the output
* polynomial g cannot be unitten as g}=cg'\mathrm{ 'where
* g' is a polynomial and c is an integer, c>1.
*
*/
FAlg
findGCD(input )
FAlg input;
{
FAlg output = input, process = input;
QInteger coef;
Integer GCD = zOne, numerator, denominator;
Bool first = 0, allNeg = qLess( fAlgLeadCoef( input ), qZero() );
if((ULong) fAlgNumTerms( input ) == 1 ) // If poly has just 1 term.
{
    // Return that term with a unit coefficient
    return fAlgMonom(qOne(), fAlgLeadMonom( input ) );
}
else // Poly has more than 1 term
{
    while( process ) // Co through each term
    {
        coef =fAlgLeadCoef( process ); // Read the lead coefficient
        numerator = coef }->>\mathrm{ num; // Break the coefficient down
        denominator = coef }->>\mathrm{ den; // into a numerator and a denominator
        process = fAlgReductum(process );// Get ready to look at the next term
```

```
    if(zIsOne( denominator) != (Bool) 1) // If we encounter a fruction
    {
        return input; // We cannot divide through by a GCD so just retumn the input
    }
    else // The coefficient was an integer
    {
        if(first == 0) // If this is the first term
        {
            first = (Bool) 1;
            GCD = numerator; // Set the GCD to be the current numerntor
        }
        else // Recursively calculate the GCD
            GCD = zGcd(GCD, numerator );
        }
    }
    if( zLess(GCD, zZero )== (Bool) 1)// If the GCD is negative
        GCD = zNegate(GCD ); // Negate the GCD
    if(zLess(zOne, GCD ) == (Bool) 1)// If the GCD is > > 
        output = fAlgZScaDiv( output, GCD ); // Divide the poly by the GCD
    }
    if( allNeg == (Bool) 1) // If the original coefficient was negative
    return fAlgZScaTimes( zMinusOne, output ); // Return the ncgated polynomial
    else
    return output;
}
05 /*
* Function Name: maxDegree
*
* Overview: Returns maximal degree of lead term for the given FAlgList.
*
* Detail: Given an FAlgList, this function calculates the degree
* of the lead term for each element of the list and returns
* the largest value found.
*
*/
ULong
maxDegree( input )
FAlgList input;
{
ULong test, output = 0;
while( input ) // For each polynomial in the list
{
    // Calculate the degree of the lead monomial
        test = fMonLength(fAlgLeadMonom(input }->\mathrm{ first ) );
        if( test > output ) output = test;
        input = input }->\mathrm{ rest; // Advance the list
    }
```

204

```
APPENDIX B. SOURCE CODE
```

```
    // Return the maximal value
```

    // Return the maximal value
    return output;
    }
33/*

* Function Name: fAlgListLowest
* 
* Overview: Returns the position of the smallest LM(g) in the given FAlgList
* Detail: Given an FAlgList, this function looks at all the leading
* monomials of the elements in the list and returns the position of
* the smallest lead monomial with respect to the monomial ordering
* currently being used
* 

*/
ULong
fAlgListLowest( input )
FAlgList input;
ULong output =0, i, len = fAlgListLength( input );
FMon next, lowest;
if( input ) // Assume the 1st lead monomial is the smallest to begin with
{
lowest = fAlgLeadMonom( input }->\mathrm{ first );
output = 1;
}
for( }\textrm{i}=1;\textrm{i}<l=n;\textrm{i}++)// For the remaining polynomials
{
input = input }->\mathrm{ rest;
// Estract the nest lead monomial
next =fAlgLeadMonom(input }->\mathrm{ first );
// If this lead monomial is smaller than the current smallest
if( theOrdFun( next, lowest ) == (Bool) 1 )
{
// Make this lead monomial the smallest
output = i+1;
lowest = fAlgScaTimes( qOne(), next );
}
}
// Return position of smallest lead monomial
return output;
}
276 * ===========
277 * End of File
278 * ==============
279 */

```
232
274
275 /*

\section*{B.2.4 file_functions.h}
```

/*

* File: file_functions.h
* Author: Gareth Evans
* Last Modified: 14th July 2004
*/
// Initialise file definition


# ifndef FILE_FUNCTIONS_HDR

# define FILE_FUNCTIONS_HDR

// Include MSSRC: Libruries

# include <fralg.h>

// MAXLINE denotes the length of the longest allowable line in a file

# define MAXLINE }500

//
// Low Level File Handling Functions
//
// Read a line from a file; return length
int getLine( FILE *, char[], int );
// Pick an integer from a list such as "D, 5, 6,"
int intFromStr( char[], int, int *);
// Pick: a variable from a list such as "a;b;c;"
String variableFromStr( char[], int, int * );
// Pick an FMon from a list: such as "a; b; c;"
FMon fMonFromStr( char[], int, int *);
// Pick an FAlg from a string such as "x*y-z;"
FAlg fAlgFromStr( char[], int, int * );
//
// High Level File Reading Functions
//
// Routine to read an FMonList from the first line of a file
FMonList fMonListFromFile( FILE *);
// Routine to read an FAlgList from a file
FAlgList fAlgListFromFile( FILE * );
41 //
// High Level File Writing Functions
//
4 4
45 // Writes an FMon (in parse format) followed by a semicolon to a file
void fMonToFile( FILE *, FMon );
// Writes an FMonList to a file on a single line
void fMonListToFile( FILE *, FMonList );
50 //
51 // File Name Modification Functions

```
40
49
```

//
// Appends ".drl" onto a string (except in special case "*.in")
String appendDotDegRevLex( char[] );
// Appends ".deg" onto a string (ercept in special case "*.in")
String appendDotDegLex( char[] );
// Appends ".lex" onto a string (except in special case "*.in")
String appendDotLex( char[] );
// Appends ".up" onto a string (except in special case "*.in")
String appendDotWP( char[] );
// Calculates the length of an input string
int filenameLength(char[] );

# endif // FILE_FUNCTIONS_HDR

```

\section*{B.2.5 file_functions.c}
```

**

* File: fite_functions.c
* Author: Gareth Evans
* Last Modified: 16th August 2004
*/
/*

```

```

* Low Level File Handling Functions
* (Used in the high level functions)

```

```

*/
/*

* Function Name: getLine
* 
* Overview: Read a line from a file; return length
* 
* Detail: Given a file _infil, we read the first line
* of the file, placing the contents into the string _s_.
* The third parameter -lim_ determines the maximum length
* of any line to be returned (when we call the function
* this is usually MAXLINE); the retumned integer tells
* us the length of the line we have just read.
* 
* Known Issues: The length of a line is sometimes returned
* incorrectly when a file saved in.Windows is used
* on a UNIX machine. Resave your file in. UNIX.
*/
int
getLine( infil, s, lim )
FILE *infil;
char s[];
int lim;
{

```
```

int c, i;
/*
* Place characters in_s_ as long as (1) we do not erceed _lim_ number of
* charucters; (2) the end of the file is not encountered; (3) the end of the
* line is not encountered.
*/
for( }\textrm{i}=0;(\textrm{i}<\operatorname{lim}-1)\&\&(( c = fgetc(infil) ) != -1 ) \&\& ( c != (int)'\n' ); i++ )
{
s[i] = (char)c;
}
if(c== (int)'\n' ) // if the for loop was terminated due to reaching end of line
{
s[i] = (char)c; // add the newline character to our string
i++;
}
s[i] = '\0'; // \0' is the null character
return i-1; // The -1 is used to compensate for the null character
/*

* Function Name: intFromStr
* 
* Overview: Pick an integer from a list such as "2, 5, 6,"
* 
* Detail: Starting from position -j- in a string _s-,
* read in un integer and return it. Note that the integer
* in the string must be terminated urith a comma and that
* the sign of the integer is taken into account.
* Once the integer has been read, place the position we
* have reached in the string in the variable -pk..
*/
int
intFromStr( s, j, pk )
char s[];
int j, *pk;
char c;
int n = 0, sign = 1, k= j;
c=s[k];
// Traverse through any empty space
while( c == 'u')
{
k++;
c=s[k];
}
// If a sign is present, process it
if(c == '+')
{
k++;

```
5 \}
\{
```

    c=s[k];
    }
else if(c==,',}
{
sign = -1;
k++;
c}=\textrm{s}[\textrm{k}]
}
// Until a comma is encountered (signalling the
// end of the integer)
while( c != ',')
{
if(( c >= '0') \&\&( c <='' '))
{
n}=10*\textrm{n}+(\textrm{int})(\textrm{c}-\mp@subsup{}{}{\prime}\mp@subsup{0}{}{\prime});// the "-..'0'" is needed to get the correct integer
}
else
{
printf("Error:uIncorrectuInput
exit( EXIT_FAILURE );
}
k++;
c}=\textrm{s}[\textrm{k}]
}
*pk}=\textrm{k}+1;//veturn the finishing position
/*
* Note: In this function we returm *pk =k+1 and not *pk=k as
* in subsequent functions because this function has a slightly
* different structure due to having to denl with the + and -
* characters at the beginning of the string.
*/
return sign*n; // return the integer

* Function Name: variableFromStr
* 
* Overview: Pick a variable from a list such as "a; b; c;"
* 
* Detail: Starting from position _j- in a string _s--,
* read in a String and return it. Note that the String
* in the string must be terminated with a semicolon.
* Once the String has been read, place the position we
* have reached in the string in the variable _pk.-
*/
String
variableFromStr( s, j, pk )
char s[];
int j, *pk;

```
\}
25
26 /*
141 \{
```

char c = 'u';
int i=0,k=j
String back = strNew(), concat;
sprintf( back, "" ); // Initialise back
// Until a semicolon is encountered
while( c != ';')
{
c}=\textrm{s}[\textrm{k}];// Pick a character from the string
// If a semicolon was encountered
if(c == ';')
{
concat = strNew();
sprintf( concat, "%c", '\0' );
back = strConcat( back, concat ); // Finish with the null character
}
else if(c != 'u')
{
concat = strNew();
sprintf( concat, "%c", c );
// Transfer character to output String
if( i == 0 ) back = strCopy( concat );
else back = strConcat(back, concat );
i++;
}
k++;
}
*pk}=\textrm{k};// Place finish position in the variable _pk
return back; // Return the String
}

* Function Name: fMon FroniStr.
* 
* Overview: Pick an FMon from a list such as "a; l; c;"
* 
* Detail: Starting from position -j- in a string -s_-
* read in an FMon and return it. Note that the FMon.
* in the string must be terminated with a semicolon.
* Once the FMon has been read, place the position we
* have reached in the string in the variable _pk_.
*/
FMon
fMonFromStr( s, j, pk )
char s[];
int j, *pk;
{
char c = 'u', a[MAXLINE];
int i = 0, k= j;
FMon back;

```
175
76 /*
```

// Until a semicolon is encountered
while(c!= ';')
{
c}=\textrm{s}[\textrm{k}];// Pick a churacter from the string
// If we have found a semicolon
if( c == ';')
{
a[i] = '\0';// Finish the string with the null character
}
else
{
a[i] = c; // Continue to process...
i++;
}
k++;
}
*\textrm{pk}=\textrm{k};// Place the finish position in the variable .pk.
back = parseStrToFMon( a );// Convert the string to an FMon
return back; // Return the FMon

* Function Name: fAlgFromStr
* 
* Overview: Pick an FAlg from a string such as " }2*y=-\quadz;
* 
* Detail: Starting from position _j- in a string _s.,
* read in an FAlg and return it. Note that the FAly
* in the string must be terminated with a semicolon.
* Once the FAlg has been read, place the position we
* have reached in the string in the variable _pk..
*/
FAlg
fAlgFromStr( s, j, pk )
char s[];
int j, *pk;
{
char c = 'u', a[MAXLINE];
int i}=0,k=j
FAlg back;
// Until a semicolon is encountered
while(c!= ';')
{
c}=\textrm{s}[\textrm{k}];// Read a character from the string
// If a semicolon is encountered
if(c == ';')
{
a[i] = '\0'; // Finish with the null character

```
7 \}
21.8
219 /*
```

    }
    else
    {
        a[i] = c; // Continue to process...
        i++;
    }
    k++;
    }
*pk = k; // Place the finish position in the variable _pk_
back = parseStrToFAlg(a ); // Convert the string to an FAlg
return back; // Return the FAlg
263 * ====================================
264 * High Level File Reading Functions
265 * ======================================
269 * Function Name: fMonListFromFile
271 * Overview: Routine to read an FMonList from the first line of a file
273 * Detail: Given an input file, this function
274 * reads the first line of the file and returns
275 * the semicolon separated FMonList found on that line.
276 * For example, if the input is a list such as a; b; A; B;
277 * then the output is the F'MonList. (a,b, A, B).
FMonList
fMonListFromFile( infil)
FILE *infil;
FMon w;
FMonList words =fMonListNul;
char s[MAXLINE];
int j = 0, k= 0, len = 0;
// Get the first line of the file and its length
len = getLine(infil, s, MAXLINE );
// While there are more FMons to be found
while( j < len )
{
w = fMonFromStr( s, j, \&k ); // Obtain an FMon
j}=\textrm{k};//\mathrm{ Set the next starting position
words = fMonListPush( w, words );// Construct the list
}
// Returm the list - note that we must reverse the list
// because it has been read in reverse order.

```
\}
261
262/*
266 */
267
268 /*
270
272 *
278 */
\{
```

    return fMonListFXRev(words );
    }

* Function Name: fAlgListFromFile
* 
* Overview: Routine to read an FAlgList from a file
* 
* Detail: Given an input file, this function
* takes each line of the file in turn, pushing one FAlg from
* each line onto an FAlgList. This process is
* continued until there are no more lines in the file
* to process. For example, if the input is a list such as
* 
* 2*: }x\cdots\mp@code{4*
* 5*2*y;
*4+5*x+60*y;
* 
* then the output is the FAlgList
* (2x-4y,5xy, 4+5x+60y).
*/
FAlgList
fAlgListFromFile( infil )
FILE *infil;
{
FAlg entry;
FAlgList back = fAlgListNul;
char s[MAXLINE];
int j = 0, k=0, len;
// Get the first line of the file
len = getLine(infil, s, MAXLINE );
// While there are still lines to process
while( len > 0)
{
entry = fAlgFromStr( s, j, \&k ); // Obtain an FAlg from a line
back = fAlgListPush( entry, back ); // Push the FAlg onto the list
len = getLine( infil, s, MAXLINE ); // Get a new line
}
// Return the list - note that we must reverse the list
// because it has been read in reverse order.
return fAlgListFXRev( back );
}
* ====================================
* High Level File Writing Functions
* ====================================
*/

```
303
\(304 / *\)
346
347 /*
352
353 /*
```

354
355
356
357
358
359
360
6 1 ~ v o i d
fMonToFile( infil, w )
FILE *infil;
FMon w;
{
FMon wM;
ULong length;
// If the FMon is non-\cdotsempty
if ( fMonEqual( w, fMonOne()) != (Bool) 1 )
{
// While there are letters left in the FMon
while (w)
{
wM = fMonLeadPowFac( w ); // Obtain a factor
fprintf( infil, "%s", fMonToStr(wM ) );// Write the factor to file
length = fMonLength( wM );
w}= fMonSuffix( w, fMonLength( w ) - length );
if (fMonEqual( w, fMonOne() ) != (Bool) 1)
{
// In parse format, to separate variables we use an asterisk
fprintf( infil, "*" );
}
}
fprintf( infil, ";"); // At the end urite a semicolon to file
}
else // Just write a semicolon to file
{
fprintf(infl, ";" );
}
}
392
93 /*

* Function Name: fMonListToFile
* Overview: Writes an FMonList to a file on a single line
* 
* Detail: Given an input file and an FMonList, this function
* writes the list to file as 11; l2; 13; ...
*/
void
fMonListToFile( infil, L )
FILE *infil;
FMonList L;
{
ULong i, length = fMonListLength( L );

```
```

// For each element of the list
for( }\textrm{i}=1;\textrm{i}<=\mathrm{ length; i++ )
{
// Write an FMon to file
fMonToFile( infil, L }->\mathrm{ first );
// If there are more FMons left to look at
if( i < length )
{
fprintf( infil, "৬" ); // Provide a space between elements
}
else // else terminate the line
{
fprintf( infil, "\n" );
}
L}=\textrm{L}-> rest
}
}

```

```

* File Name Modification Functions

```

```

*/
/*

* Function Name: appendDotDegRevLeic
* 
* Overview: Appends ".drl" onto a string (except in special case "*.in")
* 
* Detail: Given an input character array, this function
* appends the String ".drl" onto the end of the character array.
* In the special case that the input ends with ".in", the function
* replaces the ".in" with ".drl".
*/
String
appendDotDegRevLex(input)
char input[];
int length = (int) strlen( input );
String back = strNew();
// First check for ,in at the end of the file name
if (input[length -1] == 'n' \& input[length-2] == 'i' \& input[length-3] == ',')
{
input[length-2] = 'd';
input[length -1] = 'r';
sprintf( back, "%s%s", input, "l" );
}
else // Just append with ".drl"
{
sprintf( back, "%s%s", input, ".drl" );

```
426
27 /*
\{
```

    }
    return back;
    }
5:5/*

* Function Name: appendDotDegLer
* Overview: Appends ".deg" onto a string (except in special case "*.in")
* 
* Detail: Given an input character array, this function
* appends the String ".deg" onto the end of the character array.
* In the special case that the input ends with ".in", the function
* replaces the ".in" with ".deg".
*/
String
appendDotDegLex(input )
char input[];
{
int length = (int) strlen( input );
String back = strNew();
// First check for in at the end of the file name
if (input[length-1] == 'n' \& input[length-2] == 'i' \& input[length-3] == '.')
{
input[length-2] = 'd';
input[length-1] = 'e';
sprintf( back, "%s%s", input, "g" );
}
else // Just append writh ".deg"
{
sprintf( back, "%s%/s", input, ".deg" );
}
return back;
}
/*
* Function Name: appendDotLe:c
* 
* Overview: Appends ".lex" onto a string (except in special case "*.in")
* 
* Detail: Given an input character array, this function
* appends the String ".lex" onto the end of the character array.
* In the special case that the input ends with ".in", the function
* replaces the ".in" with ".lex".
*/
String
appendDotLex(input)
char input[];
int length = (int) strlen( input );
String back = strNew();

```
464
\{
```

// First check for in at the end of the file name
if (input[length-1] == 'n' \& input[length-2] == ' i' \& input[length-3] == '.')
{
input[length-2] = '1';
input[length-1] = 'e';
sprintf( back, "%s%s", input, "x" );
}
else // Just append with ".lex"
{
sprintf( back, "%s%s", input, ".lex" );
}
return back;
}

* Frnction Name: appendDotWP
* Overview: Appends ".wp" onto a string (except in special case "*.in")
* Detail: Given an input character array, this function
* appends the String ".wp" onto the end of the character array.
* In the special case that the input ends with ".in", the function
* replaces the ".in" with ".wp".
*/
String
appendDotWP(input)
char input[];
{
int length = (int) strlen( input );
String back = strNew();
// First check for in at the end of the file name
if (input[length-1] == 'n' \& input[length-2] == ' i' \& input[length-3] == '.')
{
input[length-2] = 'W';
input[length-1] = ' }\textrm{p}\mathrm{ ';
sprintf( back, "%s", input );
}
else // Just append with ".wp"
{
sprintf(back, "%s%s", input, " . wp" );
}
return back;
}
* Function Name: filenameLength
* 
* Overview: Calculates the length of an input string
* 

```
528
29 /*
560
61 /*
```

* Detail:Given an input character array, this function
    * finds the length of that character array
*/
int
filenameLength(s )
char s[]
{
int i = 0;
while( s[i] != '\0' ) i++;
return i;
580 /*
581 * =============
582 * End of File
583 * ============
584 */

```
578 \}
579

\section*{B.2.6 fralg_functions.h}
```

/*

* File: fraly.functions.h
* Author: Gareth Evans
* Last Modified: 10th August 2005
*/
// Initialise filc definition


# ifndef FRALG_FUNCTIONS_HDR

# define FRALG_FUNCTIONS_HDR

// Include MSSRC Libraries

# include <fralg.h>

// Include System Libraries

# include <limits.h>

// Include *-functions Libraries

# include "list_functions.h"

# include "arithmetic_functions.h"

//
// External Variables Required
//
extern ULong nRed; // Stores how many reductions have been performed
extern int nOfGenerators, // Holds the number of generators
pl; // Holds the "Print Level"
//
// Functions Defined in fralg_functions.c

```
```

//
//
// Ordering Functions
//
// Returns 1 if 1st. arg <-{Lex} 2nd arg
Bool fMonLex(FMon, FMon );
// Returns 1 if 1st arg <-{InvLex} 2nd arg
Bool fMonInvLex(FMon, FMon );
// Returns 1 if 1st arg <-{DegRevLex} 2nd arg
Bool fMonDegRevLex(FMon, FMon );
// Returns 1 if 1st arg <-{WreathProduct} 2nd arg
Bool fMonWreathProd( FMon, FMon );
//
// Alphabet Manipulation. Functions
//
// Substitutes ASCII generators for original generators in a list of polynomials
FAlgList preProcess( FAlgList, FMonList );
// Substitutes original generators for ASCII generators in a given polynomial
String postProcess( FAlg, FMonList );
// As above but gives back its output in parse format
String postProcessParse( FAlg, FMonList );
// Aljusts the original generator order (1st arg) according to frequency of generators in 2nd arg
FMonList alphabetOptimise( FMonList, FAlgList );
//
// Polynomial Manipulation Functions
//
// Returns all possible ways that 2nd arg divides 1st arg: 3rd arg = is division possible?
FMonPairList fMonDiv( FMon, FMon, Short * );
// Returns the first way that 2nd arg divides 1st arg; 3rd arg = is division possible?
FMonPairList fMonDivFirst( FMon, FMon, Short *);
// Finds all possible overlaps of 2 FMons
FMonPairList fMonOverlaps( FMon, FMon );
// Returns the degrec---based initial of a polynomial
FAlg degInitial( FAlg );
// Reverses a monomial
FMon fMonReverse( FMon );
//
// Groelner Basis Functions
//
// Returns the normal form of a polynomial w.r.t. a list of polynomials
FAlg polyReduce( FAlg, FAlgList );
// Minimises a given Gioebner Busis
FAlgList minimalGB( FAlgList );
// Reduces each member of a Groebner Basis w.r.t. all other mernbers
FAlgList reducedGB( FAlgList );

```
```

APPENDIX B. SOURCE CODE

```
```

// Tests whether a given FAlg reduces to 0 using the given FAlgList

```
// Tests whether a given FAlg reduces to 0 using the given FAlgList
Bool idealMembershipProblem( FAlg, FAlgList );
Bool idealMembershipProblem( FAlg, FAlgList );
# endif // FRALG_FUNCTIONS_HDR
```


# endif // FRALG_FUNCTIONS_HDR

```

\section*{B.2.7 fralg_functions.c}
```

/*

* File: fralg_functions.c
* Author: Gareth Evans
* Last Modified: 10th. August 2005
*/
/*

```

```

* Global Variables for fralg-functions.c
* =========================================
*/
static int bigVar = 1; // Keeps track of iteration dept/s in WreathProd
/*
* =======================
* Ordering Functions

```

```

*/
/*

* Function Name: fMonLer
* Overview: Returns 1 if 1st arg <-{Lex} \&nd arg
* 
* Detail: Given two FMons a and y, this function.
* compares the two monomials using the lexicographic
* ordering, returning }1\mathrm{ if }x<y\mathrm{ and }0\mathrm{ if }x>=y\mathrm{ .
* 
* Description of the Lex ordering:
* 
* x<y iff (working left-to-right) the first (say ith)
* letter on which x and y differ is
* such that co_ i< y_i in the ordering of the variables.
* 
* External Variables Required: int pl;
* 
* Note: This code is based on L. Lambe's "fMonTLeac" code.
*/
Bool
fMonLex( x, y )
FMon x, y;
{ {
ULong lenx, leny, min, count = 1;
int j;

```
```

Bool back;
if(pl > 8 ) printf("Entered,toucompare|x
if( }\textrm{x}==(\textrm{FMOn})NULL ) // If :c is empty we only have to check that y is non--mpty
{

```

```

    return (Bool) ( y != (FMon) NULL );
    }
else if( }\textrm{y}==(\textrm{FMon})\mathrm{ NULL ) // If y is empty : cannot be less than it so just return 1)
{
if( pl > 8 ) printf("y山is|NULL|SOUreturningu0...\n");
return (Bool) 0;
}
else // Both non-empty
{
lenx = fMonLength( }\textrm{x}\mathrm{ );
leny = fMonLength(y);
if(lenx < leny ) // x ha,s minimum. length
{
min = lenx;
back =(Bool) 1; // If limit reached we know }x<y\mathrm{ so return 1
}
else // y has minimum length
{
min = leny;
back =(Bool) 0; // if limit reuched we know : }x>=y\mathrm{ so return. 0
}
while(count <= min ) // For each generator
{
if( pl>8)
{
printf("Comparingu%/suwithu%s\n", fMonLeadVar(fMonSubWordLen( x, count, 1 )),
fMonLeadVar(fMonSubWordLen( y, count, 1 )) );
}
// Compare generators
if(( }\textrm{j}=\operatorname{strcmp(fMonLeadVar(fMonSubWordLen( }\textrm{x},\mathrm{ count, 1 ) ),
fMonLeadVar( fMonSubWordLen( y, count,1)))) < 0)
{

```

```

            return (Bool) 1;
        }
        else if(j>0)
        {
            if( pl>8) printf("y山isulessuthan}|x\ldots...\\n")
            return (Bool) 0;
        }
        count++;
    }
    }
// Limit now reached; return previously agreed solution

```
```

if( pl > 8 ) printf("Returningь%%i...u\n", (int) back);
return back;
1 }
03 /*

* Function Name: fMonInvLex
* 
* Overview: Returns 1 if 1st arg <-{InvLex} 2nd arg
* 
* Detail: Given two FMons x and y, this function
* compares the two monomials using the inverse lexicographic
* ordering, returning 1 if }x<y\mathrm{ and 0 if }x>=y\mathrm{ .
* 
* Description of the InvLex ordering:
* 
* x<<y iff (working right--to-left) the first (say ith)
* letter on which s}\mathrm{ and y differ is
* such. that \mp@subsup{x}{-}{}i<\mp@subsup{y}{-}{}i\mathrm{ in the ordering of the variables.}
* 
* External Variables Required: int pl;
* 
* Note: This code is based on L. Lambe's "fMonTLex" code
*/
Bool
fMonInvLex( x, y )
FMon x, y;
{
ULong lenx, leny, min, count = 0;
int j;
Bool back;

```

```

if( }\textrm{x}==(\mathbf{FMOn})NULL )// If x is empty we only have to check that y is non-empty
{
if( pl > 8 ) printf("x (>is
return (Bool) ( y != (FMon) NULL );
}
else if( y ==(FMon) NULL ) // If y is empty x cannot be less than it so just return o
{

```

```

    return (Bool) 0;
    }
else // Both. non-empty
{
lenx = fMonLength( x );
leny = fMonLength( y );
if( lenx < leny ) // x has minimum length
{
min = lenx;
back = (Bool) 1; // If limit reached we know }x<y\mathrm{ so return 1
}
else // y has minimum length

```
02
```

    {
        min = leny;
        back}=(\mathrm{ Bool) 0;// if limit reached we know }x>=y\mathrm{ so Teturn 0
    }
    while( count < min )// For each generator
    {
        if( pl > 8)
        {
            printf("Comparing\llcorner\lrcorner%s⿱宀with
                    fMonLeadVar( fMonSubWordLen( y, leny-count, 1 ) ) ;
    }
    // Compare generators _in reverse_
    if(( j = strcmp(fMonLeadVar(fMonSubWordLen( x, lenx-count, 1)),
                                    fMonLeadVar( fMonSubWordLen( y, leny-count, 1 )))) < 0)
    {
    ```

```

        return (Bool) 1;
    }
    else if( j > 0)
    {
    ```

```

        return (Bool) 0;
    }
    count++;
    }
    }
// Limit now reached; return previously agreed solution
if( pl > 8 ) printf("Returningь%i...u\n", (int) back);
return back;

* Function Name: fMonDegRevLex
* Overview: Returns 1 if 1st arg<-{DegRevLer} 2nd arg
* 
* Detail: Given turo FMons x: and y, this function
* compares the two monomials using the degree reverse lexicographic
* ordering, returning 1 if }x<y\mathrm{ and 0 if }x>=y
* 
* Description of the DegRevLex ordering:
* 
* x<y iff deg(x)<\operatorname{deg}(y)\mathrm{ or deg(x)= deg(y)}\mp@code{|}\mathrm{ (t)}
* and : }<<{\mathrm{ RevLex} y. that is, working right to left,
* the first (say ith) letter on which x and y differ is
* such that cc_i> y_i in the ordering of the variables.
* 
* Erternal Variables Required: int pl;
* Note: This code is based on L. Lambe's "fMonTLex" code.
*/

```
\}
184
185 /*
```

Bool
fMonDegRevLex( x, y )
FMon x, y;
{
ULong lenx, leny, count;
int j;
if( pl > 8 ) printf("Entered
if( }\mathbf{x}==(\mathbf{FMOn})\mathrm{ NULL ) // If }x\mathrm{ is empty we only have to check that y is non-empty
{
if( pl > 8 ) printf("x (>is
return (Bool) ( y != (FMon) NULL );
}
else if( y == (FMon) NULL ) // If y is cmpty x cannot be less than it so just return 0
{

```

```

    return (Bool) 0;
    }
else // Both non-empty
{
lenx = fMonLength( x );
leny =fMonLength( y );
// In DeqRevLea, compare the degrees first...
if( lenx < leny )
{
if( pl > 8 ) printf("xisisuless_than
return (Bool) 1;
}
else if( leny < lenx )
{

```

```

        return (Bool) 0;
    }
    else // The degrees are the same, now use RevLer..
    {
        count = lenx; // lenic is arbitrury (because lenx = leny)
        while( count > 0) // Work in -reverse_
        {
            if( pl > 8)
            {
            printf("Comparingu%suwith」%s\n", fMonLeadVar(fMonSubWordLen( x, count, 1 )),
                                    fMonLeadVar( fMonSubWordLen( y, count, 1 )) );
            }
            if(( j = strcmp( fMonLeadVar( fMonSubWordLen( x, count, 1 )),
                    fMonLeadVar( fMonSubWordLen( y, count, 1 )) ) ) > 0 )
            {
            if( pl > 8 ) printf("x
            return (Bool) 1;
        }
        else if(j<0)
    ```
```

        {
    ```

```

            return (Bool) 0;
        }
            count--;
        }
    }
    }
// No differences found so monomials must be the same
if( pl > 8) printf("Same,ureturning\llcorner0...u\n");
return (Bool) 0;
273 * Function Nome: fMonWTeathProd
275 * Overview: Returns 1 if 1st arg<-{WreathProduct} %nd arg
277 * Detail: Given two FMons a and y, this function.
278 * compares the two monomials using the wreath product
279* ordering, returning 1 if }x<y\mathrm{ and 0 if }x>=y\mathrm{ .
280 * This function is recursive.
282 * Description of the Wreath Product Ordering:
284. * Let the alphabet have a total orler (e.g. a<b< ..)
285 * Count the number of occurrences of the highest weighted letter (e.g. z),
286 * the string with the most is bigger.
287 * If both. strings have the same number of those letters, they can
288 * be written uniquely:

* s1 =x0 z x1 z x2 ... z an
* sz = y0 z y.1 z y2 .. z yn
* 
* Then s1< < 22 if
* : x0< y0 or
* x0 = y0 and c1<y1, etc.
* (< = wreath product ordering 'on y'; iterate as needed)
* 
* Examples:
* a^100<aba^2 because 1<b
* aba^2< < a^2ba because b=b and a<a^2
* a`2ba< < '2a because b< b`2
* b`2a<bab because b^2 = '` 2 and 1<a (st=1b1ba and s2 = = 1bab1)

```

```

* 
* External Variables Required: int pl, nOfGenerators;
* Global Variables Used: int bigVar;
* 
* Note: This code is based on L. Lambe's "fMonTLer" code.
*/
Bool
fMonWreathProd( x, y )

```
) \}
271
\(272 / *\)
274 *
276 *
281
283 *
```

FMon x, y;
{
FMonList xList = fMonListNul, yList = fMonListNul;
FMon xPad =fMonOne(), yPad =fMonOne(), xLetter, yLetter, bigMon;
ULong xCount = 0, yCount = 0, i=0;
/*
* Note: the global variable 'brigVar' is used to keep
* track of the iteration depth. The algorithm is designed
* so that the value of bigVar is always returned to its
* original value (which is usually 1)
*/
if( pl > 8 ) printf("Entered
bigVar, fMonToStr( x ), fMonToStr(y ) );
// Fail safe check - cannot have more iterntions than generators;
// value 1 chosen by convention (in the case of equality)
if(!(nOfGenerators-bigVar > = 0) ) return (Bool) 1;
// Deal with special cases first
if( }\textrm{x}==(\mathbf{FMOn})NULL ) // If x is empty we only have to check that y is non-empty
{
if( pl > 8 ) printf("x (>is)
return (Bool) ( y != (FMon) NULL );
}
else if( }\textrm{y}==(\mathbf{FMOn})NULL )// If y is empty a cannot be less than it so just return 0
{
if( pl > 8) printf("y (yis
return (Bool) 0;
}
else if ( fMonEqual( x, y )==(Bool) 1) // If }x=== y just retum. 0
{
if( pl > 8 ) printf(" }\mp@subsup{x}{\cup}{\prime}=|\mp@subsup{y}{\cup}{\prime
return (Bool) 0;
}
else // Both non--empty and not equal
{
// Construct the generator for this iteration
bigMon = ASCIIMon( (ULong) nOfGenerators - (ULong) bigVar + 1 );
// Process x letter by letter, creating lists of intermediate terms
while(fMonIsOne(x) != (Bool) 1)
{
xLetter = fMonPrefix( x, 1);// Look at the first letter
if(fMonEqual(xLetter, bigMon ) == (Bool) 1 ) // if xLetter === bigMon
{
xCount++; // Increase the number of elements in the list.
xList = fMonListPush( xPad, xList );
xPad = fMonOne(); // Reset
}
else
{

```
```

        xPad = fMonTimes( xPad, xLetter ); // Build up next element.
    }
    x}=\textrm{fMonSuffix}(\textrm{x},\textrm{fMonLength}(\textrm{x})-1);// Look at nert letter.
    }
xList }=\textrm{fMonListPush}(\textrm{xPad},\textrm{xList});// Flush out the remainder
// Process y letter by letter
while(fMonIsOne( y )!= (Bool) 1)
{
yLetter = fMonPrefix( y, 1); // Look at the first letter
if(fMonEqual(yLetter, bigMon ) == (Bool) 1) // if yLetter === bigMon
{
yCount++; // Increase the number of elements in the list
yList = fMonListPush( yPad, yList );
yPad = fMonOne(); // Reset
}
else
{
yPad}=\textrm{fMonTimes}(\textrm{yPad},\textrm{yLetter});// Build up next element
}
y = fMonSuffix( y, fMonLength( y ) - 1 ); // Look at next letter
}
yList }=\textrm{fMOnListPush}(\textrm{yPad},\textrm{yList});// Flush out the remainder
*
* Assuming representutions

```

```

    * y = y0 z y1 z y2 .. z ym,
    *
    * We now have
    * xList = (xn, ..., xz, x1, x0),
    * yList =(ym, .., y2, y1, y0),
    * and xCount and yCount hold the number of
    * z's in }x\mathrm{ and y respectively.
    *
    */
    // If xCount != yCount then we have a result...
if( xCount < yCount )
{
if( pl > 8) printf("x (>has
return (Bool) 1;
}
else if( xCount > yCount)
{

```

```

    return (Bool) 0;
    }
else // ...otherwise we have to look at the intermadiate terms
{
// Reverse the lists to obtain
//xList =(x0,x1,x2,···,xn) and
// yList = (y0,y1,y2, .., yn)

```
```

APPENDIX B. SOURCE CODE

```
```

    xList = fMonListFXRev( xList );
    ```
    xList = fMonListFXRev( xList );
    yList = fMonListFXRev( yList );
    yList = fMonListFXRev( yList );
    // Increase the itcration value -----.- we will now compare the
    // Increase the itcration value -----.- we will now compare the
    // elements of the lists w.r.t. the ne:ct highest variable
    // elements of the lists w.r.t. the ne:ct highest variable
    bigVar++;
    bigVar++;
    while( xList )
    while( xList )
    {
    {
        i++;
        i++;
        if(fMonWreathProd( xList }->\mathrm{ first, yList }->\mathrm{ first ) ==(Bool) 1)
        if(fMonWreathProd( xList }->\mathrm{ first, yList }->\mathrm{ first ) ==(Bool) 1)
        {
        {
            if( pl > 8 ) printf("Onn
            if( pl > 8 ) printf("Onn
            bigVar--; // reset before return
            bigVar--; // reset before return
            return (Bool) 1;
            return (Bool) 1;
        }
        }
        else if( fMonWreathProd( yList }->\mathrm{ first, xList }->\mathrm{ first ) ==(Bool) 1)
        else if( fMonWreathProd( yList }->\mathrm{ first, xList }->\mathrm{ first ) ==(Bool) 1)
        {
```

        {
    ```


```

            bigVar--; // reset before return.
    ```
            bigVar--; // reset before return.
            return (Bool) 0;
            return (Bool) 0;
        }
        }
        else // (equal)
        else // (equal)
        {
        {
            // Look at the next values in the sequence
            // Look at the next values in the sequence
            xList = xList }->\mathrm{ rest;
            xList = xList }->\mathrm{ rest;
            yList = yList }->\mathrm{ rest;
            yList = yList }->\mathrm{ rest;
        }
        }
    }
    }
    /*
    /*
    * Note: we should never reach this part of the code
    * Note: we should never reach this part of the code
    * because we know that at least one list comparison
    * because we know that at least one list comparison
            * will return a result (not all list comparisons urill
            * will return a result (not all list comparisons urill
            * return 'equal' because we know by this stage that
            * return 'equal' because we know by this stage that
            * x is not equal to y). However we carry on for
            * x is not equal to y). However we carry on for
            * completion.
            * completion.
            */
            */
            bigVar--; // Reset
            bigVar--; // Reset
    }
    }
}
}
printf("ExecutingムUnreachable\Code\n");
printf("ExecutingムUnreachable\Code\n");
exit( EXIT_FAILURE );
exit( EXIT_FAILURE );
return (Bool) 0;
return (Bool) 0;
}
/*
```

/*

```


```

* Alphabet Manipulation Functions

```
* Alphabet Manipulation Functions
* ==================================
* ==================================
*/
*/
/*
/*
* Function Name: preProcess
```

* Function Name: preProcess

```

\section*{* Overview: Substitutes ASCII generators for original generators in a list of polynomials}
```

* 
* Detail: This function takes a list of polynomials _originalPolys_
* in a set of generators _originalGenerators_ and returns the
* same set of polynomials in ASCII generators, where the first
* element of _originalGenerators_ is replaced by 'AAA', the
* second element by 'AAB', etc.
* 
* For example, if originalGenerators $=(x, y, z)$ so that the
* generator order is $x<y<z$, and if_originalPolys_ $=\left(x * y-z, 4 * x^{*} 2-5 * z\right)$,
* the output list is ( $\left.A A B * A A B-A A C, 4 * A A A^{2} 2-5 * A A C\right)$.
*/
AlgList
preProcess( originalPolys, originalGenerators )
FAlgList originalPolys;
MonList originalGenerators;
FAlgList newPolys $=$ fAlgListNul;
FAlg oldPoly, newPoly, adder;
ULong i, oldPolySize, genLength, position;
FMon firstTermMon, newFirstTermMon, multiplier, gen;
QInteger firstTermCoef;
// Go through each polynomial in turn...
while( originalPolys )
\{
oldPoly $=$ originalPolys $->$ first; // Extract a polynomial
originalPolys $=$ originalPolys $\rightarrow>$ rest; oldPolySize $=($ ULong $)$ fAlgNumTerms ( oldPoly $) ; / /$ Obtain the number of terms newPoly $=\mathrm{fAlgZero}() ; / /$ Initialise the new polynomial
$\operatorname{for}(\mathrm{i}=1 ; \mathrm{i}<=$ oldPolySize $; \mathrm{i}++) / /$ For each term in the polynomial \{
firstTermMon $=$ fAlgLeadMonom (oldPoly $) ; / /$ Eatruct monomial firstTermCoef $=$ fAlgLeadCoef( oldPoly $) ; / /$ Ertruct coefficient oldPoly $=\mathrm{fAlgReductum}($ oldPoly $) ; / /$ Get ready to look at the next term newFirstTermMon $=\mathrm{fMonOn}() ; / /$ Initialise the new monomial
// Go through each term replacing generators as required while( fMonIsOne( firstTermMon) $!=($ Bool ) 1) \{
gen $=\mathrm{fMonPrefix}($ firstTermMon, 1 ); // Take the first letter ' $x$ ' position $=\mathrm{fMonListPosition}($ gen, originalGenerators ); // Find the position of the letter in the list multiplier $=$ ASCIIMon( position ); // Obtain the ASCII generator corresponding to $x$ genLength $=\mathrm{fMonLeadExp}($ firstTermMon $) ; / /$ Find the exponent ' $a$ ' as in $x^{\wedge} a$ $/ /$ Multiply new monomial by (ASCII) $x{ }^{\wedge} a$
newFirstTermMon $=\mathrm{fMonTimes}($ newFirstTermMon, fMonPow( multiplier, genLength ) ); // Lose x^a from original monomial firstTermMon $=\mathrm{fM}$ MonSuffix $($ firstTermMon, fMonLength $($ firstTermMon $)-$ genLength $)$; \}
}

```

470 -

473
```

APPENDIX B. SOURCE CODE

```
```

        adder = fAlgMonom( firstTermCoef, newFirstTermMon ); // Construct the new ASCII term
    ```
        adder = fAlgMonom( firstTermCoef, newFirstTermMon ); // Construct the new ASCII term
        newPoly = fAlgPlus(newPoly, adder ); // Add the new ASCII term to the output polynomial
        newPoly = fAlgPlus(newPoly, adder ); // Add the new ASCII term to the output polynomial
    }
    }
    newPolys = fAlgListPush( newPoly, newPolys ); // Push new polynomial onto output list
    newPolys = fAlgListPush( newPoly, newPolys ); // Push new polynomial onto output list
}
// Return the reversed list (it was read in reverse)
return fAlgListFXRev( newPolys );
}
3/*
* Function Name: postProcess
*
* Overview: Substitutes original gencrators for ASCII generators in a given polynomial
*
* Detail: This function takes a polynomial _oldPoly_ in ASCII generators
* and returns the same polynomial in a corresponding set of generators
* _originalGenerators_. The output is returned as a String
* in fAlgToStr( .. ) format.
*
* For example, if _originalGenerators. = = (x,y,z) so that the
* generator order is }x<y<z\mathrm{ , and if oldPoly- = A*B-C`2}\mathrm{ , then
* the output String is "xy-z`2".
*
*/
String
postProcess( oldPoly, originalGenerators )
FAlg oldPoly;
FMonList originalGenerators;
{
FAlg adder
Bool result;
FMon firstTermMon, gen, newFirstTermMon, multiplier;
QInteger firstTermCoef;
ULong i, match, oldPolySize, genLength;
String back = strNew();
sprintf( back, "" ); // lnitialise back
// Obtain the number of terms in the polynomial
oldPolySize = (ULong) fAlgNumTerms(oldPoly )
for( }\textrm{i}=1;\textrm{i}<=\mathrm{ oldPolySize; }\textrm{i}++)// For each term
{
    firstTermMon = fAlgLeadMonom( oldPoly ); // Obtain the lead monomial
    firstTermCoef = fAlgLeadCoef( oldPoly ); // Obtain the lead coefficient
    result = qLess( firstTermCoef, qZero() ); // Test if coefficient is -ve
    oldPoly = fAlgReductum( oldPoly ); // Get ready to look: at the next term
    newFirstTermMon = fMonOne(); // Initiulise the new monomial
    // Go through the term replacing generators as required.
    while(fMonIsOne(firstTermMon) != (Bool) 1 )
    {
```

532

```
    gen = fMonPrefix( firstTermMon, 1 ); // Obtain the first letter ' }x\mathrm{ '
    genLength = fMonLeadExp(firstTermMon ); // Obtain ' a' as, in x`a
    // Calculute the ASCII value ('AAA' = , 'AAB' = 2, ...)
    match = ASCIIVal(fMonToStr( gen ) );
    multiplier = fMonListNumber(match, originalGenerators ); // Find the original generator
    multiplier =fMonPow( multiplier, genLength );
    newFirstTermMon = fMonTimes( newFirstTermMon, multiplier ); // Multiply new monomial by the original }\mp@subsup{x}{}{\wedge
    // Remove ASCII r`a from original monomial
    firstTermMon = fMonSuffix( firstTermMon, fMonLength( firstTermMon ) - genLength );
    }
    // Now add the term to the output string
    if( i == 1) // First term
    back = strConcat( back, fAlgToStr(fAlgMonom(firstTermCoef, newFirstTermMon ) ) );
    else // Must insert the conrect sign (plus or minus)
    {
        if(result == 0) // Coefficient is +ve
        {
            adder = fAlgMonom( firstTermCoef, newFirstTermMon ); // Construct the neww term
            back = strConcat( back, "u+u" );
            back = strConcat( back, fAlgToStr( adder ) );
        }
        else // Coefficient is -ve
        {
            adder = fAlgMonom( qNegate( firstTermCoef ), newFirstTermMon ); // Construct the new term
            back = strConcat( back, "ь-ь" );
            back = strConcat( back, fAlgToStr( adder ) );
        }
    }
}
return back;
611 * Function Nome: postProcessParse
613 * Overview: As above but gives back its output in parse format
615 * Detail: This function takes a polynomial _oldPoly_ in ASCII generators
6 1 6 ~ * ~ a n d ~ r e t u r n s ~ t h e ~ s a m e ~ p o l y n o m i a l ~ i n ~ a ~ c o r r e s p o n d i n g ~ s e t ~ o f ~ g e n e r a t o r s ~
617 * _originalGenerators_. The output is returned os a String in.
618 * parse format (with asterisks).
620 * For example, if _originalGenerators_ = (x,y,z) so that the
6 2 1 ~ * ~ g e n e r a t o r ~ o r d e r ~ i s ~ x < y < z , ~ a n d ~ i f ~ - o l d P o l y - ~ = ~ A * B - C ` 2 , ~ t h e n ~
622 * the output String is "}x*y-\mp@subsup{z}{}{*}2"
626 postProcessParse( oldPoly, originalGenerators )
627 FAlg oldPoly;
628 FMonList originalGenerators;
```

\}
609
610 /*
612 *
614 *
619 *
623 *
624 */
625 String

```
29 {
Short first = 1, written;
FMon firstTermMon, gen, multiplier;
QInteger firstTermCoef;
ULong i, match, oldPolySize, genLength;
String back = strNew();
sprintf( back, "" ); // Initialise back
if( !oldPoly ) // If input: is NULL output the zero polynomial
{
    back = strConcat( back, "0" );
    return back;
}
// Obtain the number of terms in the polynomial
oldPolySize = (ULong) fAlgNumTerms( oldPoly );
for( i = 1; i <= oldPolySize; i++ )// For each term
{
    // Assume to begin with that nothing has been added to
    // the String regarding the term we are now looking at
    written = 0;
    // Break down a term of the polynomial into its pieces
    firstTermMon = fAlgLeadMonom(oldPoly ); // Obtain the lead monomial
    firstTermCoef = fAlgLeadCoef( oldPoly ); // Obtain the lead coefficient
    if(qLess( firstTermCoef, qZero()) == (Bool) 1 ) // If the coefficient is -ve
    {
        if( first == 1) // If this is the first term encountered
        {
            first = 0; // Set to avoid this loop in future
            // Note: there is no need for a space before the minus sign
        back = strConcat( back, "-" );
    }
    else // This is not the first term
    {
        // Separate two terms with a minus sign
        back = strConcat( back, "u-u" );
    }
    // Now that we have written the negative sign we can make
    // the coefficient positive
    firstTermCoef = qNegate( firstTermCoef );
    }
    else // The coefficient is +ve
    {
        if( first == 1) // If this is the first term encountered
        {
            first = 0;// Set to avoid this loop in future
```

    // Recall that there is no need to write out a plus
        // sign for the first term in a polynomial
    \}
    else // This is not the first term.
    \{
        // Separate two terms with a plus sign
        back \(=\) strConcat ( back, " \(\mathrm{u}+\mathrm{u}\) " );
    \}
    \}
if( $q$ IsOne( firstTermCoef ) $!=($ Bool ) 1 ) // If the coe/ficient is not one
\{
written $=1 ; / /$ Denote that we are going to write the coefficient to the String
if( fMonEqual( firstTermMon, fMonOne() ) $=(\mathrm{Bool}) 1$ ) // If the lead monomial is not 1
\{
// Provide an asterisk to denote that the coefficient is
// maltiplied by the monomial
back $=\operatorname{strConcat}($ back, qToStr( firstTermCoef ) );
back $=$ strConcat (back, "*" );
\}
else
\{
// As the monomial is I there is no need to write the
// monomial out and we can just write out the coefficient
back $=\operatorname{strConcat}($ back, qToStr( firstTermCoef ) );
\}
\}
// If the lead monomial is not one
if( fMonIsOne( firstTermMon ) $!=($ Bool $) 1$ )
\{
written $=1 ; / /$ Denote that we are going to urite the monomial to the String
// Go through the term replacing generators as required
while( firstTermMon )
\{
gen $=\mathrm{fMonPrefix}($ firstTermMon, 1$) ; / /$ Obtain the first letter ' $x$ '
genLength $=\mathrm{fMonLeadExp}($ firstTermMon $) ; / /$ Obtain ' $a$ ' as in $x^{\wedge} a$
$/ /$ Calculate the $A S C I I$ value ( $' A A A^{\prime}=1,{ }^{\prime} A A B^{\prime}=2, \ldots$ )
match $=$ ASCIIVal( fMonToStr (gen ) );
multiplier $=\mathrm{fMonListNumber}($ match, originalGenerators $) ; / /$ Find the original generator
multiplier $=\mathrm{fM}$ MonPow $($ multiplier, genLength $)$;
// Add multiplier onto the String
back $=$ strConcat $($ back, fMonToStr( multiplier ) );
// Move the monomial onwards
firstTermMon $=\mathrm{fMonSuffix}($ firstTermMon, $\mathrm{fMonLength}($ firstTermMon $)-$ genLength $) ; / /$ Remove ASCII $x^{\wedge} a$
if( firstTermMon ) back = strConcat( back, "*" );
\}
\}
\}

```
// If the coefficient is 1 and the monomial is 1 and nothing
// has yet been written about this term, write "1" to the String
// (This is to cutch the case where the term is --1)
if((qIsOne(firstTermCoef ) == (Bool) 1 )
        && (fMonIsOne(firstTermMon ) == (Bool) 1)
        && (written ==0))
    {
        back = strConcat( back, "1" );
    }
    oldPoly = fAlgReductum( oldPoly ); // Get ready to look at the next term
}
return back;
*
Function Name: alphabetOptimise
* Overview: Adjusts the original generator order (1st arg) according to
* frequency of generators in 2nd arg
* Detail: Given an FMonList _oldGens_ storing the given generator
* order, this function optimises this order according to the
* ficquency of the generators in the polynomial list _polys..
* More specifically, the most frequently occurning gencrator
* is set to be the smallest generator; the second most frequently
* occurling genciator is set to be the second smallest generator, ...
* For the reasoning behind this optimisation, see a paper called
* "A case where choosing a product order makes the
* calculations of a Groebner bnsis much faster" by
* Freyja Hreinsdottir (Journal of Symbolic Computation).
*
* Note:This function is designed to be used before the
* generators and polynomials are converted to ASCII order.
* External variables needed: int pl;
*/
FMonList
alphabetOptimise( oldGens, polys )
FMonList oldGens;
FAlgList polys;
ULong i, j, letterLength, size = fMonListLength(oldGens ), scores[size];
FMon monomial, letter, theLetters[size];
FAlg poly;
FMonList newGens = fMonListNul;
if( pl>0)
{
    printf("Old_Orderingu=ч");
    fMonListDisplayOrder( oldGens );
```

* 

```
APPENDIX B. SOURCE CODE
```

    printf("\n");
    ```
    printf("\n");
}
}
// Set up arrays
// Set up arrays
for( i = 0; i < size; i++)
for( i = 0; i < size; i++)
{
{
    theLetters[i] = oldGens }->\mathrm{ first; // Thansfer generator to array
    theLetters[i] = oldGens }->\mathrm{ first; // Thansfer generator to array
    oldGens = oldGens }->>\mathrm{ rest;
    oldGens = oldGens }->>\mathrm{ rest;
    scores[i] = 0; // Initialise scores
    scores[i] = 0; // Initialise scores
}
}
// Analyse the generators found in each polynomial
// Analyse the generators found in each polynomial
while(polys )
while(polys )
{
{
    poly = polys }->\mathrm{ first; // Eatract a polynomial
```

    poly = polys }->\mathrm{ first; // Eatract a polynomial
    ```


```

    polys = polys }->\mathrm{ rest;
    ```
    polys = polys }->\mathrm{ rest;
    while( poly ) // For each term in the polynomial
    while( poly ) // For each term in the polynomial
    {
    {
        monomial = fAlgLeadMonom( poly ); // Extract the lead monomial
        monomial = fAlgLeadMonom( poly ); // Extract the lead monomial
        poly = fAlgReductum( poly );
        poly = fAlgReductum( poly );
        while(fMonIsOne(monomial ) != (Bool) 1)
        while(fMonIsOne(monomial ) != (Bool) 1)
        {
        {
            letter = fMonPrefix( monomial, 1 );// Take the first letter : : 
            letter = fMonPrefix( monomial, 1 );// Take the first letter : : 
            letterLength = fMonLeadExp(monomial ); // Find the exponent. ' }a\mathrm{ ' as in x` a
            letterLength = fMonLeadExp(monomial ); // Find the exponent. ' }a\mathrm{ ' as in x` a
            j = 0;
            j = 0;
            while( }\textrm{j}<\mathrm{ size ) // Locate the letter in the generator array
            while( }\textrm{j}<\mathrm{ size ) // Locate the letter in the generator array
            {
            {
            if( fMonEqual( letter, theLetters[j] )==(Bool) 1)
            if( fMonEqual( letter, theLetters[j] )==(Bool) 1)
            {
            {
                    // Match found, increase scores appropriately
                    // Match found, increase scores appropriately
                    scores[j] = scores[j] + letterLength;
                    scores[j] = scores[j] + letterLength;
                    j = size; // Shortcut search
                    j = size; // Shortcut search
            }
            }
            else j++;
            else j++;
        }
        }
        monomial = fMonSuffix( monomial, fMonLength(monomial) - letterLength ); // Lose x}\mp@subsup{x}{}{`}\mathrm{ a from old monomial
        monomial = fMonSuffix( monomial, fMonLength(monomial) - letterLength ); // Lose x}\mp@subsup{x}{}{`}\mathrm{ a from old monomial
        }
        }
    }
    }
}
}
if( }\textrm{pl}>0)// Provide some information on screen
if( }\textrm{pl}>0)// Provide some information on screen
{
{
    printf("Frequencies
    printf("Frequencies
    for( i=0; i < size; i++)
    for( i=0; i < size; i++)
    {
    {
        printf("%u,山", scores[size-1-i] );
        printf("%u,山", scores[size-1-i] );
    }
    }
    printf("\n");
    printf("\n");
}
```

}

```
```

// Sort scores by a quicksort algorithm, adjusting the generators as we go along
alphabetArrayQuickSort( scores, theLetters, 0, size-1 );
// Build up new ulphabet
for( }\textrm{i}=1;\textrm{i}<=\mathrm{ size; i++ )
newGens = fMonListPush(theLetters[size-i], newGens );
if( pl>0)
{
printf("New\lrcornerOrdering\sqcup=ப");
fMonListDisplayOrder(newGens );
printf("\n");
}
// Return the sorted alphabet list
return newGens;
/*

* =============================================0
* Polynomial Manipulation Functions

```

```

*/
/*

* Function Name: fMonDiv
* 
* Overvicw: Returns all possible ways that 2nd arg divides 1.st arg;
* 3rd arg = is division possible?
* 
* Detail: Given two FMons _a_, and _b_, this function returns all possible
* ways that _b_ divides _a_- in the form of an FMonPairList. The third
* parameter flag- records whether or not (true/false) any divisions
* are possible. For example, if t}=abdababc and b=ab, then the
* output FMonPairList is ((abdab, c), (abd, abc), (1, dababc)) and we
* set _flag_ = truc.
* 
* External variables needel: int pl;
* 

*/
FMonPairList
fMonDiv( t, b, flag )
FMon t, b;
Short *flag;
{
ULong i, tl, bl, diff;
FMonPairList back = ( FMonPairList )theAllocFun( sizeof(*back ) );
back = fMonPairListNul; // Initialise the output list.
*flag = false; // Assume there are no possible divisions to begin with.
tl = fMonLength( t );
bl = fMonLength( b );

```
7 \}
```

APPENDIX B. SOURCE CODE

```
```

if $(\mathrm{tl}<\mathrm{bl}) / /$ There can be no possible divisions if $|t|<|b|$

```
if \((\mathrm{tl}<\mathrm{bl}) / /\) There can be no possible divisions if \(|t|<|b|\)
{
    return back;
}
else // Me mast now consider each possibility in turn
{
    diff = tl-bl;
    for( }\textrm{i}=0;\textrm{i}<=\mathrm{ diff; i++) // Working left to right
    {
        // Is the subword of t of length }|b|\mathrm{ starting at position i+1 equal to b?
        if(fMonEqual( b, fMonSubWordLen( t, i+1, bl ))==(Bool) 1)
        {
            // Match found; push the left und right factors onto the output list
            back = fMonPairListPush( fMonPrefix(t, i ), fMonSuffix(t, tl-bl-i ), back );
            if( pl > 6 ) printf("i
                                    fMonToStr( fMonPrefix( t, i ) ), fMonToStr( b ), fMonToStr( fMonSuffix( t, tl-bl-i ) ) );
        }
    }
}
// If we found some matches set _flag_ to be true
if( back) *flag = true;
return back; // Return the output list
}
920 /*
* Function Name: fMonDivFirst
*
* Overview: Return.s the first way that 2nd arg divides 1st arg;
* 3rd arg = is division possible?
*
* Detail: Given two FMons _ _ _ and _ _ __, this function returns the first
* way that _ __ divides _ a_ in the form of an FMonPairList. The third
* parameter _flag_ records whether or not (true/false) any divisions
* are possible. For e.cample, if t=abdababo and b=ab), then the
* output FMonPairList is ((1, dalabc)) und we
* set _fiag_ = true.
*
* External variables needed: int pl;
*
*/
FMonPairList
fMonDivFirst(t, b, flag )
FMon t, b;
Short *flag;
{
ULong i, tl, bl, diff;
FMonPairList back = ( FMonPairList )theAllocFun( sizeof(*\mathrm{ back ));}
back = fMonPairListNul; // Initialise the output list
*flag = false; // Assume there are no possible divisions to begin with.
```

91.9

```
tl = fMonLength(t );
bl = fMonLength( b );
if(tl<bl)// There can be no possible divisions if }|t|<|b
{
    return back;
}
else // Me must now consider each possibility in turn
{
    diff = tl-bl;
    for( }\textrm{i}=0;\textrm{i}<=\mathrm{ diff; i++ )// Working left to right
    {
        // Is the subword of t of length |b| starting at position i+1 equal to b?
        if(fMonEqual(b, fMonSubWordLen(t,i+1, bl ))==(Bool) 1)
        {
            // Match found; push the left andl right factors onto the output list und return it
            back = fMonPairListPush( fMonPrefix( t, i ), fMonSuffix( t, tl-bl-i ), back );
            if( pl > 6 ) printf("i
                    fMonToStr( fMonPrefix( t, i ) ), fMonToStr( b ), fMonToStr( fMonSuffix( t, tl-bl-i ) ) ;
            *flag = true; // Indicate that we have found a match
            return back;
        }
    }
}
    return back; // Return the empty output list -- no matches were found
976 * Function Name: fMonOverlaps
* Overview: Finds all possible overlaps of 2 FMons
*
* Detail: Given two FMons, this function returns all
* possible ways in which the two monomials overlap.
* For example, if _ a_ = abcabc and _ b_ = cab, then
* the output FMonPairList is
* ((1, 1), (ab, c), (c, 1), (1, cabc), (1, ab), (abcab, 1))
* as in
* 1*(abcabc)*1=ab*(cab)*c,
* c*(abcabc)*1=1*(cab)*cabc.
* 1*(abcabc)*ab = abcab*(cab)*1.
*
* External variables needed: int pl;
*
*/
FMonPairList
fMonOverlaps( a, b )
FMon a, b;
{
997 FMon still, move;
998 Short type;
999 ULong la, lb, ls, Im, i;
```

3 \}
974
975 / *
977 *

1000 1001 1002 1003 1004 1005 1006

```
FMonPairList back = ( FMonPairList )theAllocFun( sizeof( *back ));
back}=\textrm{fMonPairListNul; // Initialise the output list
la = fMonLength(a );
lb}=\textrm{fMonLength(b ;
// Check for the trivial monomial
if((la == 0)|(lb==0)) return back;
// Determine which monomial has the greater length
if(la<lb )
{
    still = b; ls = lb;
    move = a; lm = la;
    type =1;// Remember that }|a|<|b
}
else
{
    still = a; ls = la;
    move = b; lm = lb;
    type =2;// Remember that |a|>=| |b|
}
// First deal with prefix and suffic overlaps
for( }\textrm{i}=1;\textrm{i}<==\operatorname{lm}-1;\textrm{i}++
{
    // PREFIX overlap -..- is a prefix of still equal to a suffix: of move?
    if(fMonEqual(fMonPrefix( still, i ), fMonSuffix(move, i )) == (Bool) 1)
    {
        if( type == 1)// still = b, move =a.
        {
            // Need to multiply a on the right and b on the left to construct the overlap
            back = fMonPairListPush( fMonPrefix( a, la-i ), fMonOne(), back ); // b
            back = fMonPairListPush( fMonOne(), fMonSuffix( b, lb-i ), back ); // a
            if( pl > 5 ) printf("Left|Overlap\lrcornerFound
                                    fMonToStr( a ), fMonToStr( b ), fMonToStr( fMonOne() )
                    fMonToStr( fMonSuffix( b, lb-i ) ), fMonToStr( fMonPrefix( a, la-i )),
                    fMonToStr( fMonOne() ) );
    }
        else // still =a, move = b
        {
            // Need to multiply a on the left and b on the right to construct the overlap
            back = fMonPairListPush( fMonOne(), fMonSuffix( a, la-i ), back ); // b
            back = fMonPairListPush( fMonPrefix( b, lb-i ), fMonOne(), back );// a
```



```
                                    fMonToStr( a ), fMonToStr( b ), fMonToStr( fMonPrefix( b, lb-i ) ),
                                    fMonToStr( fMonOne() ), fMonToStr( fMonOne() ),
                                    fMonToStr( fMonSuffix( a, la-i ) ) );
        }
    }
    // SUFFLX overlan - is a suffix of still equal to a prefix of move?
```

```
    if(fMonEqual( fMonSuffix(still, i ), fMonPrefix( move, i )) == (Bool) 1)
    {
        if(type == 1)// still = b, move =a
    {
        // Need to multiply a on the left and b on the right to construct the overlap
        back =fMonPairListPush( fMonOne(), fMonSuffix( a, la-i ), back );//b
        back =fMonPairListPush( fMonPrefix( b, lb-i ), fMonOne(), back ); // a
        if( pl > 5 ) printf("Right」Overlap\lrcornerFound
                        fMonToStr( a ), fMonToStr( b ), fMonToStr( fMonPrefix( b, lb-i ) ),
                        fMonToStr( fMonOne() ), fMonToStr( fMonOne() ),
                fMonToStr( fMonSuffix( a, la-i ) ) );
    }
    else // still = = a, move = b
    {
        // Need to multiply a on the right and b on the left to construct the overlap
        back = fMonPairListPush( fMonPrefix( a, la-i ), fMonOne(), back ); // b
        back = fMonPairListPush( fMonOne(), fMonSuffix( b, lb-i ), back ); // a
        if( pl > 5 ) printf("Right_Overlap,Found
                        fMonToStr( a ), fMonToStr(b ), fMonToStr( fMonOne() ),
                fMonToStr( fMonSuffix( b, lb-i ) ), fMonToStr( fMonPrefix( a, la-i ) ),
                fMonToStr( fMonOne() ) );
        }
    }
}
// Subword overlaps
for( i = 1; i <= ls-lm+1; i++ )
{
    if(fMonEqual(move, fMonSubWordLen(still, i, lm ))==(Bool) 1)
    {
        if( type == 1) // still = b, move =a
        {
            // Need to multiply a on the left and right to construct the overlap
            back =fMonPairListPush( fMonOne(), fMonOne(), back ); // b
            back = fMonPairListPush( fMonPrefix( b, i-1 ), fMonSuffix( b, lb+1-i-lm ), back );// a
```



```
                fMonToStr( a ), fMonToStr( b ), fMonToStr( fMonPrefix( b, i-1 )),
                    fMonToStr( fMonSuffix( b, lb+1-i-lm ) ), fMonToStr( fMonOne() ),
                    fMonToStr( fMonOne() ) );
    }
    else // still =a, move = b
    {
            // Need to multiply b on the left and right to construct the overlap
        back = fMonPairListPush(fMonPrefix( a, i-1 ), fMonSuffix( a, la+1-i-lm ), back );//b
        back = fMonPairListPush( fMonOne(), fMonOne(), back ); // a
        if( pl > 5 ) printf("Middle⿺Overlap\lrcornerFound
                                    fMonToStr( a ), fMonToStr( b ), fMonToStr( fMonOne() ),
                                    fMonToStr( fMonOne() ), fMonToStr( fMonPrefix( a, i-1 )),
                                    fMonToStr( fMonSuffix(a, la+1-i-lm )) );
        }
    }
}
```

```
1106
1107 }
1108
1109 /*
1110 * Function Name: degInitial
1111 *
1 1 1 2 ~ * ~ O v e r v i e w : ~ R e t u r n s ~ t h e ~ d e g r e e - b a s e d ~ i n i t i a l ~ o f ~ a ~ g i v e n ~ p o l y n o m i a l .
1113 *
1 1 1 4 ~ * ~ D e t a i l : ~ G i v e n ~ a ~ p o l y n o m i a l ~ . i n p u t . , ~ t h i s ~ f u n c t i o n ~ r e t u r n s ~ t h e ~
1115 * initial of that polynomial w.r.t. degree. In other words,
1116 * all terms of highest degree are returned.
1117 *
1118 */
1119 FAlg
1120 degInitial( input)
1121 FAlg input;
1122 {
1123 FAlg output = fAlgZero();
1124 ULong max }=0\mathrm{ , next;
1125
1126 // If the input is trivial, the output is trivial
1127 if(!input ) return input;
1128
1129 // For each term in the input polynomial
1130 while( input )
1131 {
1132 // Find the degree of the neact term in the polynomial
1 1 3 3
1134
1135
1136
```

1159 * Function Name: fMonReverse
1160 *
1161 * Overview: Reverses a monomial
1162 *
1163 * Detail: Given a monomial m}=\mp@subsup{x}{-}{\prime}1\mp@subsup{c}{-}{\prime2}...t-n, this
1164 * function returns the monomial m}==\mp@code{m}n\mp@subsup{c}{-}{\prime}{n-1}···x-2x_1
1165 *
1166 */
116T FMon
1.68 fMonReverse( input)
1169 FMon input;
1170 {
1171. FMon output = fMonOne();
1.172
1193 * Overview: Returns the normal form of a polynomial w.r.t. a list of polynomials
1 1 9 4
195 * Detail: Given an FAlg and an FAlgList, this function
1196 * divides the FAlg w.r.t. the FAlgList, returning the
1197 * normal form of the input polynomial w.r.t. the list.
1198
199 * External Variables Required: int pl;
1200 * Global Variables Used: ULong nRed:
1 2 0 1
1202
1203 FAlg
204 polyReduce( poly, list )
1205 FAlg poly
206 FAlgList list;
1207 {
1208 ULong i, numRules = fAlgListLength( list );
1209 FAlg back = fAlgZero(), lead, upgrade, LHSA[numRules];
1210 FMon leadMonomial, leadLoopMonomial, LHSM[numRules];
1 2 1 1
FMonPairList factors;

```
```

QInteger leadQ, leadLoopQ, lcmQ, LHSQ[numRules];
Short flag, toggle;
// Convert the input list of polynomials to un urray and
// create arrays of lead monomials and lead coefficients
for( }\textrm{i}=0;\textrm{i}<\mathrm{ numRules; i++ )
{
if( pl > 5 ) printf("Polyu%%u
LHSA[i] = list }->>\mathrm{ first;
LHSM[i] = fAlgLeadMonom( list }->\mathrm{ first );
LHSQ[i] = fAlgLeadCoef( list }->\mathrm{ first );
list = list }->\mathrm{ rest;
}
// We will now recursively reduce every term in the polynomial
// until no more reductions are possible
while( fAlgIsZero(poly ) != (Bool) 1)
{
if( pl > 5 ) printf("Lookingцat_Lead
toggle = 1;// Assume no reductions are possible to begin with
lead = fAlgLeadTerm(poly );
leadMonomial = fAlgLeadMonom(lead );
leadQ = fAlgLeadCoef( lead );
i}=0
while( i < numRules ) // For cach polynomial in the list.
{
leadLoopMonomial = LHSM[i]; // Pick a test monomial
flag = false;
// Does the ith polynomial divide our polynomial?
factors = fMonDivFirst( leadMonomial, leadLoopMonomial, \&flag );
if(flag == true ) // i.e. leadMonomial = factors }->\mathrm{ lft * leadLoopMonomial * factors }->>r
{
if( pl > 1) nRed++; // Increase the number of reductions carried out

```

```

                        fMonToStr( factors }->> lft ), fMonToStr( leadLoopMonomial )
                        fMonToStr( factors -> rt ) );
            toggle = 0; // Indicate a reluction has been carricd out to e:cit the loop
            leadLoopQ = LHSQ[i]; // Pick the divisor's leading coe/ficient
            lcmQ = AltLCMQInteger( leadQ, leadLoopQ ); // Pick 'nice' cancelling coefficients
            // Construct poly #i*-1* coefficient to get lead terms the same
            upgrade = fAlgTimes( fAlgMonom(qOne(), factors }->>\mathrm{ lft ), LHSA[i] );
            upgrade = fAlgTimes( upgrade, fAlgMonom(qNegate( qDivide(lcmQ, leadLoopQ ) ), factors }->\mathrm{ rt ) );
            // Add in poly * coefficient to cancel off the lead terms
            upgrade = fAlgPlus( upgrade, fAlgScaTimes(qDivide(lcmQ, leadQ ), poly ) );
            // We must also now multiply the current discarded remainder by a factor
            back = fAlgScaTimes( qDivide( lcmQ, leadQ ), back );
            poly = upgrade; // In the nert iteration, we will be reducing the new polynomial upgrade
    ```

```

        }
            if( toggle == 1) // The ith polynomial did not divide poly
                i++;
            else // A reduction was carried out, exit the loop
            i = numRules;
    }
    if(toggle == 1 ) // No reductions were carried out: now look at the next term
    {
        // Add lead term to remainder and reduce the rest of the polynomial
        back = fAlgPlus( back, lead );
        poly = fAlgReductum(poly );
        if( pl > 5 ) printf("New
    }
    }
return back; // Return the reduced and simplified polynomial
1289 * Detail: Given an input Groelmer Busis, this function
1290 * will eliminate from the basis any polynomials whose
1291 * lead monomials are multiples of some other lead
1292 * monomial.
1294 * External variables required: int pl;
1297 FAlgList
1298 minimalGB( G )
*
*/
FAlgList G;
FAlgList G_Minimal = fAlgListNul, G_Copy = fAlgListCopy(G );
ULong i, p, length = fAlgListLength(G);
FMon checker[length];
FMonPairList sink;
Short flag, blackList[length];
// Create an array of lead monomials and initialise blackList
// which will store which monomials are to be deleted from the basis
for( }\textrm{i}=0;\textrm{i}<l\mathrm{ length; i++)
{
blackList[i] = 0;
checker[i] = fAlgLeadMonom( G_Copy }->\mathrm{ first );
G_Copy = G_Copy }->\mathrm{ rest;
}
// Test divisibility of each monomial w.r.t all other monomials
for( }\textrm{i}=0;\textrm{i}<l\mathrm{ length; i++ )

```
2 \}
1284 /*
1285
1287
1288
1293 *
1295
1296 *
1299
1300 \{
1301
```

{
p=0;
while( p < length )
{
// If p is different from i and p has not yet been 'deleted' from the basis
if(( p != i ) \&\& (blackList[p]!= 1))
{
flag = false;
sink = fMonDiv( checker[i], checker[p], \&flag );
if( flag == true ) // poly i's lead term is a multiple of poly p's lead term
{
blackList[i] = 1; // Ensure polynomial i is deleted later on
break; // Exit from the while loop
}
}
p++;
}
}
// Push. onto the output list elements not blacklisted
for(i = 0; i < length; i++ )
{
if(blackList[i] == 0)// Not to be deleted
{
G_Minimal = fAlgListPush( G }->\mathrm{ first, G_Minimal );
}
G = G -> rest; // Aduance the list
}
// As it was constructed in reverse, we must reverse G_Minimal before returning it
return fAlgListFXRev( G_Minimal );
1352 * Function Name: reducedGB
1354 * Overview: Reduces each member of a Groebner Basis w.r.t. all other membar's
1356 * Detail: Given a list of polynomials, this function takes each
1357 * member of the list in turn, relucing the polynomial u.r.t. all
1358 * other members of the basis.
1 3 6 0 ~ * ~ N o t e : ~ T h i s ~ f u n c t i o n ~ d o e s ~ n o t ~ c h e c k ~ w h e t h e r ~ a ~ p o l y n o m i n l ~ r e d u c e s ~ t o ~
1361. * zero or not (we usually want to delete polynomials that reduce to
1362 * zero from our basis) - it is assumed that no member of the basis will
1363 * reduce to zero (which will be the case if we start with a minimal Groebner
1.364 * Basis). Also, at the end of the algorithm, the total number of reductions
1365 * carried out during the *whole program* is reported if the print level
1366 * (pl) excecds }1
1368 * Extemal variables required: int pl

```
\}
1350
1.351 /*
1.353
1355 *
1359 *
1367 *
1369 *
1370 */
```

FAlgList
reducedGB( GBasis )
FAlgList GBasis;
{
FAlg poly;
FAlgList back = fAlgListNul, old_G, new_G;
ULong i, sizeOfInput = fAlgListLength( GBasis );
if( sizeOfInput > 1)// lf |GBasis }|>
{
i}=0;// i keeps track of which polynomial we are looking at
// Start by making a copy of G for processing
old_G = fAlgListCopy(GBasis );
while(old_G ) // For each polynomial
{
i++;
poly = old_G }->\mathrm{ first; // Extract a polynomial
old_G = old_G }->\mathrm{ rest; // Advance the list.
if( pl > 2 ) printf("\nLooking\&atuelement }\mp@subsup{|}{\cup}{\prime
// Construct basis without. 'poly' by appending
// the remaining polynomials to the reduced polynomials
new_G = fAlgListAppend( back, old_G );
poly = polyReduce( poly, new_G ); // Reduce poly w.r.t. new_G
poly = findGCD( poly ); // Divide out by the GCD
if( pl > 2 ) printf("Reduced龵to\lrcorner%s\n", fAlgToStr( poly ));
// Add the reduced polynomial to the list
back =fAlgListAppend( back, fAlgListSingle( poly ));
}
}
else // else |GBasis = = 1 and there is no point in doing any reduction
{
return GBasis;
}
// Report on the total number of reductions carried out during the *whole program*

```

```

    return back;
    1417 * Function Name: idealMembershipProblem
1419 * Overview: Tests whether a given FAlg reduces to 0) using the given FAlgList
1421 * Detail: Given a list of polynomials, this function tests whether
1422 * a given polynomial reduces to zero using this list. This is
1423 * done using a modified version of the function polyReduce in that

```
\(1414\}\)
1415
\(1416 / *\)
1418 *
1420 *
```

1424 * the moment an irreducible monomial is encountered, the algorithm
1425 * terminates with the knouledge that the polynomial will not
1426 * reduce to 0 .
1427 *
1.428 * External variables required: int pl;
1429 *
1430 */
1431 Bool
1432 idealMembershipProblem( poly, list )
1433 FAlg poly;
1434 FAlgList list
1435 \{
1436 ULong i, numRules $=$ fAlgListLength ( list );
1437 FAlg back $=$ fAlgZero(), lead, upgrade, LHSA[numRules];
1438 FMon leadMonomial, leadLoopMonomial, LHSM[numRules];
1434 FMonPairList factors;
1440 QInteger leadQ, leadLoopQ, lcmQ, LHSQ[numRules];
1441 Short flag, toggle;
1442
1443
1444
1446

```

```

144s LHSA[i] = list $\rightarrow$ first;
1449
1450
1451
1452
1453
// Convert the input list of polynomials to an array and
// create arrays of lead monomials and lead coefficients
for $(\mathrm{i}=0 ; \mathrm{i}<$ numRules; $\mathrm{i}++$ )
\{
LHSM $[\mathrm{i}]=\mathrm{fAlgLeadMonom}($ list $->$ first $)$;
LHSQ[i] $=$ fAlgLeadCoef( list $\rightarrow>$ first );
list $=$ list $\rightarrow$ rest;
\}
// We will now recursively reduce every term in the polynomial
// until an irreducible term is encountered or no more reductions are possible
while( fAlgIsZero( poly) $!=($ Bool $) 1$ )
\{

```

```

    toggle \(=1 ; / /\) Assume no reductions are possible to begin with
    lead \(=\) fAlgLeadTerm ( poly );
    leadMonomial \(=\) fAlgLeadMonom ( lead \()\);
    leadQ \(=\) fAlgLeadCoef( lead );
    \(\mathrm{i}=0\);
    while( i < numRules ) // For each polynomial in the list.
    \{
        leadLoopMonomial \(=\) LHSM[i]; // Pick a test monomial
        flag = false;
        // Does the ith polynomial divide our polynomial?
        factors \(=\mathrm{fMonDivFirst}(\) leadMonomial, leadLoopMonomial, \&flag \()\);
        if \((\) flag \(==\) true \() / /\) i.e. leadMonomial \(=\) factors \(\cdots>l f t *\) leadLoopMonomial \(*\) factors \(\cdots>r t\)
        \{
    ```

```

                                    fMonToStr ( factors \(\rightarrow>\mathrm{lft}\) ), fMonToStr( leadLoopMonomial ),
                                    fMonToStr( factors \(\rightarrow\) rt ) );
    ```
```

            toggle = 0; // Indicate a reduction has been carried out to exit the loop
            leadLoopQ = LHSQ[i]; // Pick the divisor's leading coefficient
            lcmQ = AltLCMQInteger( leadQ, leadLoopQ ); // Pick 'nice' cancelling coefficicnts
            // Construct poly #i * - -1 * coefficient to get lead terms the same
            upgrade = fAlgTimes( fAlgMonom( qOne(), factors }->\mathrm{ lft ), LHSA[i] );
            upgrade =fAlgTimes( upgrade, fAlgMonom(qNegate(qDivide( lcmQ, leadLoopQ ) ), factors }->\mathrm{ rt ) );
            // Add in poly * coefficient to cancel off the lead terms
            upgrade = fAlgPlus( upgrade, fAlgScaTimes(qDivide( lcmQ, leadQ ), poly ) );
            // We must also now multiply the current discarded remainder by a factor
            back = fAlgScaTimes( qDivide( lcmQ, leadQ ), back );
            poly = upgrade; // In the next iteration, we will be reducing the new polynomial upgrade
            if( pl > 5 ) printf("New_Word 
        }
        if( toggle == 1 ) // The ith polynomial did not divide poly
            i++;
            else // A reduction was carried out, exit the loop
            i= numRules;
    }
    /*
    * If toggle == 1, this means that no rule simplified the lead term of 'poly'
    * so that we have encountared an irreducible monomial. In this case. the polynomial
    * we are reducing will not reduce to vero, so we can now return 0.
    */
    if(toggle == 1)
        return (Bool) 0;
    }
    // If we reach here, the polynomial reduced to 0 so we return a positive result.
    return (Bool) 1;
    1513 * ==============
1514 * End of File
1515 * =============

```
\}
1511.
1512 /*
1516 */

\section*{B.2.8 list_functions.h}
```

* 
* File: list.-_unctions.h
* Author: Gareth Evans
* Last Modified: 9th August 2005
*/
// Initialise file definition


# ifndef LIST_FUNCTIONS_HDR

# define LIST_FUNCTIONS_HDR

```
```

// Include MSSRC Libraries

# include <fralg.h>

//
// External Variables Required
//
extern int pl; // Holds the "Print Level"
//
// Display Functions
//
// Displays an FMonList in the format l1\n l2\n 18\n...
void fMonListDisplay( FMonList );
// Displays an FMonList in the format ll>l2>l;_..
void fMonListDisplayOrder( FMonList );
// Displays an FMonPairList in the format (1,1,18)\n (13, 14)\n...
void fMonPairListMultDisplay( FMonPairList );
// Displays an FAlgList in the format p}<br>n p<br>n p<br>n..
void fAlgListDisplay( FAlgList );
//
// List Extraction Functions
//
// Returns the ith member of an FMonList (i=1st arg)
FMon fMonListNumber( ULong, FMonList );
// Returns the ith member of an FMonPairList ( }i=1=1\mathrm{ st arg)
FMonPair fMonPairListNumber( ULong, FMonPairList );
// Returns the ith member of an FAlgList (i = 1.st arg)
FAlg fAlgListNumber( ULong, FAlgList );
//
// List Membership Functions
//
// Does the FAlg appear in the FAlgList? ( }1=\mathrm{ yes)
Bool fAlgListIsMember( FAlg, FAlgList );
//
// List Position Functions
//
// Gives position of 1st appearance of FMon in FMonList
ULong fMonListPosition( FMon, FMonList );
// Gives position of 1st appearance of FAlg in FAlgList
ULong fAlgListPosition( FAlg, FAlgList );
//
// Sorting Functions
2 //

```
```

6 3
// Swaps 2 clements in arrays of ULongs and FMons
void alphabetArraySwap( ULong[], FMon[], ULong, ULong );
// Sorts an urray of ULongs (largest first) and applies the same changes to the FMon array
void alphabetArrayQuickSort( ULong[], FMon[], ULong, ULong );
// Swaps 2 elements in arrays of FAlgs and FMons
void fAlgArraySwap( FAlg[], FMon[], ULong, ULong );
// Sorts an array of FAlgs using DegRevLex (largest first)
void fAlgArrayQuickSortDRL( FAlg[], FMon[], ULong, ULong );
// Sorts an array of FAlgs using theOrdFun (largest first)
void fAlgArrayQuickSortOrd( FAlg[], FMon[], ULong, ULong );
// Sorts an FAlgList (largest first)
FAlgList fAlgListSort( FAlgList, int );
// Swaps 2 elements in arrays of FMons, ULongs and ULongs
void multiplicativeArraySwap( FMon[], ULong[], ULong[], ULong, ULong );
// Sorts input dat.a to OverlapDiv w.r.t. DegRevLe:r (largest first)
void multiplicativeQuickSort( FMon[], ULong[], ULong[], ULong, ULong );
//
// Insertion Sort Functions
//
// Insert into list according to DegRevLex
FAlgList fAlgListDegRevLexPush( FAlg, FAlgList );
// As above, but also returns the insertion position
FAlgList fAlgListDegRevLexPushPosition( FAlg, FAlgList, ULong * );
// Insert into list according to the curvent monomial ordering
FAlgList fAlgListNormalPush( FAlg, FAlgList );
// As above, but also returns the insertion position
FAlgList fAlgListNormalPushPosition( FAlg, FAlgList, ULong * );
//
// Deletion Functions
//
// Removes the (1st arg)-th element from the list
FMonList fMonListRemoveNumber( ULong, FMonList );
// Removes the (1st arg)--th element from the list.
FMonPairList fMonPairListRemoveNumber( ULong, FMonPairList );
// Removes the (1st arg)-th. element from the list
FAlgList fAlgListRemoveNumber( ULong, FAlgList );
//
// Normalising Functions
//
// Removes any fractions found in the FAlgList by scalar multiplication
FAlgList fAlgListRemoveFractions( FAlgList );
11.2
113 \# endif // LIST_FUNCTIONS_HDR

```

\section*{B.2.9 list_functions.c}
```

/*

* File: list_functions.c
* Author: Gareth Evans, Chris Wensley
* Last Modified: 9th August. 2005
*/
/*
* =========================
* Display Functions
* ===================
*/
**
* Function Name: fMonListDisplay
* 
* Overview: Displays an FMonList in the format l1\n l2\n l:3\n...
* 
* Detail: Given an FMonList, this function displays the
* elements of the list on screen in such a way that if the
* list is (for example) L=(l1, 12, l3, 14), the output is
* 
* l1
* 12
* l }
* l4
* 

*/
void
fMonListDisplay(L )
FMonList L;
{
while(L )
{
printf( "%s\n", fMonToStr( L -> first ) );
L}=\textrm{L}->> rest
}
}
/*

* Function Name: fMonListDisplayOrder
* 
* Overview: Displays an FMonList in the format l1<12< < % ...
* 
* Detail: Giver un FMonList, this function displays the
* elements of the list on screen in such a way that if the
* list is (for example) }L=(l1,12, l.3, l4), the output i
* 

*14>18>1.2>1.
*

* External variables required: int pl;
* 

```
```

*/
void
fMonListDisplayOrder(L )
FMonList L;
{
ULong i = 1, j = fMonListLength( L );
L=fMonListRev(L );
while( L )
{
if( pl >= 1)
printf( "%s", fMonToStr( L -> first ) );
if( pl>1)
{
printf( "u(%s)", ASCIIStr( j + 1 - i ));
i++;
}
L = L -> rest;
if( L ) // If there is unother element left provide a ">"
{
printf( "৬>>" );
}
}
}
/*

* Function Name: fMonPairListMultDisplay
* 
* Overview: Displays an F'MonPairList in the format (l, l2)\n (l,3,l4)\n...
* 
* Detail: Given an FMonPairList, this function displays the
* elements of the list on screen in such a way that. if the
* list is (for example) }L=((l1,12),(l.3,l4),(l5,16)), the output is
* 
* (l1, l2)
* (13,14)
* (l5,l6)
* 
* Remark: The "Mult" stands for multiplicative - this function is primarily
* used to display (Left, Right) multiplicative variables.
*/
void
fMonPairListMultDisplay( L )
FMonPairList L;
{
while( L )
{
printf( "(%s,, %s)\n", fMonToStr( L -> lft ), fMonToStr( L -> rt ) );
L}=\textrm{L}-> rest
}
}

```
104
```

/*

* Function Name: fAlgListDisplay
* 
* Overview: Displays an FAlgList in the format p<br>n p<br>n p3\n...
* 
* Detail: Given an FAlgList, this function displays the
* elements of the list on screen in such a way that if the
* list is (for example) L = (p1, p2, p.3, p4), the output is
* 
* p1
* p2
* p3
* p4
* 

*/
void
fAlgListDisplay(L )
FAlgList L;
{
while(L )
{
printf( "%s\n", fAlgToStr( L -> first ) );
L = L -> rest;
}
}
1 3 0
/*

* ===========================
* List. Extraction Functions
* ===============================
*/
/*
* Function Name: fMonListNumber
* 
* Overview: Returns the ith member of an FMonList (i }=1\textrm{st}\mathrm{ arg)
* 
* Detwil: Given un FMonList, this function returns the
* ith member of that list, where i is the first argument number_.
* 

*/
FMon
fMonListNumber( number, list )
ULong number;
FMonList list;
{
ULong i;
FMon back = newFMon();
for( }\textrm{i}=1;\textrm{i}<\mathrm{ number; i++ )
{
list = list }->\mathrm{ rest; // Traverse list
}

```
```

back = list -> first;
return back;
}

* Flunction Name: fMonPairListNumber
* 
* Overview: Returns the ith member of an FMonPairList ( }i=1\mathrm{ st arg)
* Detail: Civen an FMonPairList, this function returns the
* ith member of that list, where i is the first argument _number..
* 

*/
FMonPair
fMonPairListNumber( number, list)
ULong number;
FMonPairList list;
{
FMonPair back;
ULong i;
for(i=1; i < number; i++ )
{
list = list -> rest; // Traverse list
}
back.lft = list -> lft;
back.rt = list -> rt;
return back;
}

* Function Name: fAlgListNumber
* Overview: Returns the ith member of an. FAlgList (i=1st arg)
* 
* Detail: Given an FAlgList, this function returns the
* ith member of that list., where i is the first argument _number_.
* 

*/
FAlg
fAlgListNumber( number, list )
ULong number;
FAlgList list;
{
ULong i;
FAlg back = newFAlg();
for( }\textrm{i}=1;\textrm{i}<\mathrm{ < number; i++ )
{
list = list -> rest; // Traverse list

```
162
63 /*
190
1/*
```

}
back = list }->\mathrm{ first;
return back;
5}

* =============================
* List Membership Functions

```

```

*/
/*

* Function Name: fAlgListlsMember
* 
* Overvicu: Does the FAlg appear in the FAlgList? (1 = yes)
* 
* Detail: Given an FAlgList, this function tests whether
* a given FAlg appears in the list. This is done by
* moving through the list and checking each entry
* sequentially. Once a match is found, a positive result.
* is returned; otherwise once we have gone through the
* entire list, a negative result is retumed.
* 

*/
Bool
fAlgListIsMember( w, L )
FAlg w;
FAlgList L;
while(L )
{
if(fAlgEqual( w, L -> first ) ==(Bool) 1)
{
return (Bool) 1; // Match found
}
L}=\textrm{L}-> rest
}
return (Bool) 0; // No matches found
}

* =============================
* List Position Functions

```

```

*/
/*

* Function Name: fMonListPosition
* 
* Overview: Gives position of 1st appearance of FMon in FMonList
* 
* Detail: Given an. FMonList, this function returns the

```
216
217/*
\{
251
252 /*
```

* position of the first appearance of a given FMon in that
* list. If the FMon does not uppear in the list.
* 0 is returned.
* 

*/
ULong
fMonListPosition( w, L )
FMon w;
FMonList L;
{
ULong pos = 0; // Current position in list
if(fMonListLength(L ) == 0)
{
return (ULong) 0; // List is empty so no match
}
while( L ) // While there are still elements in the list
{
pos++;
if( fMonEqual( w, L -> first ) ==(Bool) 1)
{
return pos; // Match found; return position
}
L = L -> rest;
}
return (ULong) 0; // No match found in the list
}
/*

* Function Name: fAlgListPosition
* 
* Overview: Gives position of 1st appearance of FAlg in FAlgList
* 
* Detail: Given an FAlgList, this function returns the
* position of the first appearance of a given FAlg in that
* list. If the FAlg does not uppear in the list, 0 is returned.
* 

*/
ULong
fAlgListPosition( w, L )
FAlg w;
FAlgList L;
{
ULong pos=0;// Current position in list
if(fAlgListLength(L) == 0)
{
return (ULong) 0; // List is empty so no match
}
while( L ) // While there are still elements in the list
{
pos++;
if(fAlgEqual( w, L }->>\mathrm{ first ) ==(Bool) 1)

```
291
```

    {
        return pos; // Match found; retum position
    }
    L = L -> rest;
    }
return (ULong) 0; // No match. found in the list
}
/*

```

```

* Sorting Functions
* ============================0
*/
/*
* Function Name: alphabetArraySuap
* 
* Overview: Swaps 2 elements in arrays of ULongs and FMon.s
* 
* Detail: Given an array of ULongs and an array of FMons,
* this function swaps the ith and jth elements of both arrays.
* 

*/
void
alphabetArraySwap( array1, array2, i, j )
ULong array1[];
FMon array2[];
ULong i, j;
{
ULong swap1;
FMon swap2 = newFMon();
swap1 = array1[i];
swap2 = array2[i];
array1[i] = array1[j];
array2[i] = array2[j];
array1[j] = swap1;
array2[j] = swap2;
}

* Function Name: alphabetArrayQuickSort
* Overview: Sorts an array of ULongs (largest first) and
* applies the same changes to the array of FMons
* 
* Detail: Using a QuichSort algorithm, this function
* sorts un array of ULongs. The 3rd and 4th argurnents
* are used to facilitate the recursive behaviour of
* the function --.--.. the function should initially be called
* as alphabetArrayQuickSort( A, B, O, |A -1).
* It is assumed that }|A|=|B|\mathrm{ and the changes made to }
* during the algorithm are also applied to B.

```
324
356
357 /*
```

370
371 * Reference: "The C Programming Language"
372 * by Brian.W. Kernighan and Dennis M. Ritchie
373 * (Second Edition, 1988) Page 87.
374 *
75 */
7 6 ~ v o i d
alphabetArrayQuickSort( array1, array2, start, finish )
ULong array1[];
FMon array2[];
ULong start, finish;
{
ULong i, last;
if(start < finish )
{
alphabetArraySwap( array1, array2, start, (start + finish )/2 );// Move partition elem
last = start; // to array[0)
for( }\textrm{i}=\mathrm{ start }+1;\textrm{i}<=\mathrm{ finish; i++ )// Partition
{
if( array1[start] < array1[i] )
{
alphabetArraySwap( array1, array2, ++last, i );
}
}
alphabetArraySwap( array1, array2, start, last ); // Restore partition elem
if( last != 0)
{
if( start < last-1 ) alphabetArrayQuickSort( array1, array2, start, last-1 );
}
if( last+1 < finish ) alphabetArrayQuickSort( array1, array2, last+1, finish );
}
}
04
/*

* Function Name: fAlgAmaySwap
* 
* Overview: Swaps 2 elements in arrays of FAlgs and FMons
* 
* Detail: Given an array of FAlgs and an associated array
* of FMons, this function swaps the ith and jth elements
* of the arrays.
* 

*/
void
fAlgArraySwap( polynomials, monomials, i, j )
FAlg polynomials[];
FMon monomials[];
ULong i, j;
{
FAlg swapA = newFAlg();
FMon swapM = newFMon();

```
void
fAlgArrayQuickSortDRL( polynomials, monomials, start, finish )
FAlg polynomials[];
FMon monomials[]
ULong start, finish;
54 \{
ULong i, last;
if( start < finish )
\{
    fAlgArraySwap( polynomials, monomials, start, (start + finish )/2 ); // Move partition elem
    last \(=\) start; \(/ /\) to array \(/ 0\) )
    for \((\mathrm{i}=\operatorname{start}+1 ; \mathrm{i}<=\) finish \(; \mathrm{i}++) / /\) Partition
    \{
        if( fM MonDegRevLex \((\) monomials[start \(]\), monomials \([\mathrm{i}])==(\) Bool \() 1)\)
        \{
            fAlgArraySwap( polynomials, monomials, ++last, i );
        \}
    \}
    fAlgArraySwap( polynomials, monomials, start, last ); // Restore partition elem
    if( last != 0 )
    \{
        if( start < last-1 ) fAlgArrayQuickSortDRL( polynomials, monomials, start, last-1 )
    \}
    if( last+1 < finish ) fAlgArrayQuickSortDRL( polynomials, monomials, last+1, finish );
\}
```

}
478 / *

* Function Name: fAlgAriayQuickSortOrd
* Overview: Sorts an array of FAlgs using theOrdFun (largest first)
* 
* Detail: Using a QuickSort algorithm, this function
* sorts an array of FAlgs by sorting on the associated array
* of FMons which store the lead monomials of the polynomials.
* The 3rd and 4th arguments are used to facilitate the recursive
* behaviour of the function -- the function should initially be
* called as fAlgArrayQuickSortOrd( A, B, 0, |A|-1).
* 
* Reference: "The C Progrumming Language"
* by Brian. W. Kemighan and Dennis M. Ritchie
* (Second Edition, 1988) Page 87.
* 

*/
void
fAlgArrayQuickSortOrd( polynomials, monomials, start, finish )
FAlg polynomials[];
FMon monomials[];
ULong start, finish;
{
ULong i, last;
if( start < finish )
{
fAlgArraySwap( polynomials, monomials, start, (start + finish )/2 ); // Move partition elem
last = start; // to array[(0)
for( i = start +1; i <= finish; i++ )// Partition
{
if( theOrdFun(monomials[start], monomials[i])==(Bool) 1)
{
fAlgArraySwap( polynomials, monomials, ++last, i );
}
}
fAlgArraySwap( polynomials, monomials, start, last ); // Restore partition elem
if( last != 0)
{
if( start < last-1 ) fAlgArrayQuickSortOrd( polynomials, monomials, start, last-1 );
}
if( last+1 < finish ) fAlgArrayQuickSortOrd( polynomials, monomials, last+1, finish );
}
}
524/*
525 * Function Name: fAlgListSort
526 *
527 * Overview: Sorts an FAlgList (largest first)
528 *

```
477
523
```

* Detail: This function sorts an FAlgList by
* converting the list to an array, sorting the array
* with a QuickSort algorithm, and converting
* the array back to an FAlgList which is then returned.
* 

*/
FAlgList
fAlgListSort(L, type )
FAlgList L;
int type;
{
FAlgList back = fAlgListNul;
ULong length =fAlgListLength(L ), i;
FAlg polynomials[length];
FMon monomials[length];
// Check for empty list or singleton list
if( ( !L ) || (length == 1 ) ) return L;
// Transfer elements into array
for( i = 0; i < length; i++ )
{
polynomials[i] = L -> first;
monomials[i] = fAlgLeadMonom( L }->\mathrm{ first );
L = L -> rest;
}
// Sort the array (smallest .->> largest)
if(type == 1)// Sort by DegRevLer
fAlgArrayQuickSortDRL( polynomials, monomials, 0, length-1 );
else // Sort by theOrdFum,
fAlgArrayQuickSortOrd( polynomials, monomials, 0, length-1 );
// Transfer elements back *m reverse* onto an FAlgList
for( }\textrm{i}=\mathrm{ length; i >= 1; i-- )
{
back = fAlgListPush( polynomials[i-1], back );
}
// Return the sorted list
return back;
}

* Function Name: multiplicative.ArraySwap
* 
* Overview: Swaps 2 elements in arrays of FMons, ULongs and ULongs
* 
* Detail: Given an array of F'Mons and two associated arrays
* of ULLongs, this function swaps the ith and jth elements
* of the arrays.
* 

*/

```
571
572 /*
```

void
multiplicativeArraySwap( monomials, lengths, positions, i, j )
FMon monomials[];
ULong lengths[], positions[], i, j;
{
FMon swapM = newFMon();
ULong swapU1, swapU2;
swapM = monomials[i]
swapU1 = lengths[i];
swapU2 = positions[i];
monomials[i] = monomials[j];
lengths[i] = lengths[j];
positions[i] = positions[j];
monomials[j] = swapM;
lengths[j] = swapU1;
positions[j] = swapU2;
}
01 /*
2 * Function Name: multiplicativeQuickSort
603 *
604 * Overview: Sorts input data to OverlapDiv w.r.t. DegRevLex (largest first)
6 0 5
606
612
6 1 5
61.6 *
617 */
6 1 8 void
1 9 multiplicativeQuickSort( monomials, lengths, positions, start, finish )
20 FMon monomials[];
621 U
622 {
if(start < finish )
{
// Move partition elem to array[0]
multiplicativeArraySwap(monomials, lengths, positions, start, ( start + finish )/2 );
last = start;
for( }\textrm{i}=\mathrm{ start + 1; i <= finish; i++)// Partition.
{
if( fMonDegRevLex(monomials[start], monomials[i]) ==(Bool) 1)
{

```
600
607
608
609
610
611
613
61.4
623
624
```

            multiplicativeArraySwap( monomials, lengths, positions, ++last, i );
        }
    }
    multiplicativeArraySwap( monomials, lengths, positions, start, last ); // Restore purtition elem
    if( last != 0)
    {
    if( start < last-1 ) multiplicativeQuickSort( monomials, lengths, positions, start, last-1 );
    }
    if( last+1 < finish ) multiplicativeQuickSort(monomials, lengths, positions, last+1, finish );
    }
}

* =========================
* Insertion Sort Functions
* ==========================
*/
/*
* Function Name: fAlgListDegRevLexPush
* 
* Overview: Insert into list according to DegRevLex
* 
* Detail: This functions inserts the polynomial _poly-
* into the FAlgList _input_ so that the list remains
* sorted by DegRevLe:c (largest first).
* 

*/
FAlgList
fAlgListDegRevLexPush( poly, input )
FAlg poly;
FAlgList input;
FAlgList output = fAlgListNul; // Initialise the return list
FMon lead = fAlgLeadMonom(poly );
if(!input ) // If there is nothing in the input list
{
// Return a singleton list
return fAlgListSingle( poly );
}
else
{
// While the next element in the list is larger than .lead.
while(( fAlgListLength(input) >0)
\&\&(fMonDegRevLex(lead, fAlgLeadMonom(input }->\mathrm{ first )) == (Bool) 1 ))
{
// Push the list element onto the output list
output =fAlgListPush(input }->\mathrm{ first, output )
input = input }->\mathrm{ rest; // Alvance the list
}
// Now push the new element onto the list
output = fAlgListPush(poly, output );

```
646
47 /*
\{
```

    // Reverse the output list (it was constructed in reverse)
    output = fAlgListFXRev( output );
    // If there is anything left in the input list, tag it onto the output list.
    if( input ) output = fAlgListAppend( output, input );
    return output;
    }
    }

* Function Nume: fAlgListDegRevLexPushPosition
* 
* Overview: As above, but also returns the insertion position
* 
* Detail: Thris functions inserts the polynomial _poly_
* into the FAlgList _input_ so that the list remains
* sorted by DegRevLex (largest first). The position in
* which the insertion took place is placed in the
* variable _pos-.
* 

*/
FAlgList
fAlgListDegRevLexPushPosition( poly, input, pos )
FAlg poly;
FAlgList input;
ULong *pos;
{
FAlgList output = fAlgListNul; // Initialise the return list
FMon lead = fAlgLeadMonom(poly );
ULong position = 1;
if(!input ) // If there is mothing in the input list
{
*pos = 1; // Inserted into the first position
// Return a singleton list
return fAlgListSingle( poly );
}
else
{
// While the next element in the list is larger than -lead_
while((fAlgListLength(input) > 0 )
\&\& (fMonDegRevLex( lead, fAlgLeadMonom(input }->\mathrm{ first )) == (Bool) 1 ))
{
// Push the list element onto the output list
output = fAlgListPush( input }->\mathrm{ first, output );
input = input }->>\mathrm{ rest; // Advance the list
position++; // Increment the insertion position
}
// We now know the insertion position.
*pos = position;
// Push the new element onto the list
output = fAlgListPush( poly, output );
// Reverse the output list (it was constructed in reverse)

```
696
697 /*
```

    output = fAlgListFXRev(output );
    // If there is anything left in the input list, tag it onto the output list
    if( input) output = fAlgListAppend( output, input );
    return output;
    }
    }
/*

* Function Name: fAlgListNormalPush
* Overview: Insert into list according to the current monomial ordering
* 
* Detail: This functions inserts the polynomial _poly_
* into the FAlgList _input so that the list remains
* sorted by the current monomial ordering (largest first).
* 

*/
FAlgList
fAlgListNormalPush( poly, input )
FAlg poly;
FAlgList input
{
FAlgList output = fAlgListNul; // Initialise the return list
FMon lead = fAlgLeadMonom( poly );
if(!input ) // If there is nothing in the input list
{
// Return a singleton list
return fAlgListSingle( poly );
}
else
{
// While the next element in the list is larger than lead
while( (fAlgListLength(input ) >0)
\&\&(theOrdFun( lead, fAlgLeadMonom(input }->\mathrm{ first )) ==(Bool) 1))
{
// Push the list element onto the output list
output =fAlgListPush(input }->\mathrm{ first, output );
input = input }->\mathrm{ rest; // Advance the list
}
// Now push the new element onto the list
output = fAlgListPush(poly, output );
// Reverse the output list (it was constructed in reverse)
output = fAlgListFXRev(output );
// If there is anything left in the input list, tag it onto the output list
if( input ) output = fAlgListAppend( output, input );
return output;
}
}

```
792
793 /*
```

* Function Name: fAlgListNormalPushPosition
* 
* Overview: As above, but also returns the inscrtion position
* 
* Detuil: This functions inserts the polynomial _poly_
* into the FAlgList _input_ so that the list remains
* sorted by the current monomial ordering (largest first).
* The position in which the insertion took place is placed
* in the variable _pos_.
* 

*/
FAlgList
fAlgListNormalPushPosition( poly, input, pos )
FAlg poly;
FAlgList input;
ULong *pos;
{
FAlgList output = fAlgListNul; // Initialise the retarn list
FMon lead = fAlgLeadMonom( poly );
ULong position = 1;
if( !input ) // If there is nothing in the input list
{
*pos=1;// Inserted into the first position
// Return a singleton list
return fAlgListSingle( poly );
}
else
{
// While the next element in the list is larger than -lead.
while((fAlgListLength(input ) > 0 )
\&\& ( theOrdFun( lead, fAlgLeadMonom(input }->\mathrm{ first )) ==(Bool) 1))
{
// Push the list element onto the output list
output = fAlgListPush( input }->\mathrm{ first, output );
input = input }->\mathrm{ rest; // Advance the list
position++; // Increment the insertion position
}
// We now know the insertion position
*pos = position;
// Push the new element onto the list.
output = fAlgListPush( poly, output );
// Reverse the output list (it was constructed in reverse)
output = fAlgListFXRev( output );
// If there is anything left in the input list, tag it onto the output list
if( input) output = fAlgListAppend(output, input );
return output;
}
}
844
84.5 /*
8d6 * =======================

```
```

847 * Deletion Functions

* ===================
*/
850
851 /*
52 * Function Name: fMonListRemoveNumber
853
* Overview: Removes the (1st arg)-th element from the list
* 
* Detail: Given an FMonList _list_, this function removes
* from _list_ the element in position _number..
* 

FMonList
fMonListRemoveNumber( number, list )
ULong number;
FMonList list;
{
FMonList output = fMonListNul;
ULong i;
for(i = 1; i < number; i++ )
{
// Push the first (number-1) elements onto the list
output = fMonListPush( list }->\mathrm{ first, output );
list = list }->\mathrm{ rest;
}
// Delete the number-th element by skipping past it
list = list -> rest;
// Push the remaining elements onto the list
while( list )
{
output = fMonListPush( list -> first, output )
list = list }->\mathrm{ rest;
}
// Return the reversed list (it was constructed in reverse)
return fMonListFXRev( output );
}
888
889 /*
890 * Function Name: f'MonPairListRemoveNumber
891 *
892 * Overview; Removes the (1st arg)-th element from the list
893 *
8 9 4 ~ * ~ D e t a i l : ~ G i v e n ~ a n ~ F M o n P a i r L i s t ~ l i s t . , ~ t h i s ~ f u n c t i o n ~ r e m o v e s ~
895 * from _list_ the element in position _number_
896 *
897 */
FMonPairList
899 fMonPairListRemoveNumber( number, list )

```
```

ULong number
FMonPairList list;
{
FMonPairList output = fMonPairListNul;
ULong i;
for( }\textrm{i}=1;\textrm{i}<\mathrm{ number; i++ )
{
// Push the first (number-1) elements onto the list
output = fMonPairListPush( list }->\mathrm{ lft, list }-> rt, output )
list = list }->\mathrm{ rest;
}
// Delete the number-thelement by skipping past it
list = list -> rest;
// Push the remaining elements onto the list
while( list )
{
output = fMonPairListPush( list }->\mathrm{ lft, list }->\mathrm{ rt, output );
list = list }->\mathrm{ rest;
}
// Return the reversed list (it was constructed in reverse)
return fMonPairListFXRev( output );
}

* Function Name: fAlgListRemoveNumber
* 
* Overview: Removes the (1.st arg)-th element from the list
* 
* Detail: Given an FAlgList _list., this function removes
* from llist the element in position _number..
* 

*/
FAlgList
fAlgListRemoveNumber( number, list )
ULong number;
FAlgList list;
{
FAlgList output = fAlgListNul;
ULong i;
for( }\textrm{i}=1;\textrm{i}<\mathrm{ number; i++)
{
// Push the first (number-1) elements onto the list.
output = fAlgListPush( list }->\mathrm{ first, output );
list = list }->\mathrm{ rest;
}
// Delete the number-th element by skipping past it
list = list -> rest;

```
926
927 /k

953
,
\{
// Push the remaining elements onto the list
while( list )
\{
output \(=\mathrm{fAlg} \operatorname{ListPush}(\) list \(->\) first, output \()\);
list \(=\) list \(->\) rest;
\}
// Return the reversed list (it was constructed in reverse)
return fAlgListFXRev (output );
*
* \(=\) =二= \(===================\)
* Normalising Functions
*/
*
* Function Name: fAlgListRemoveFractions
* Overview: Removes any fractions found in the FAlgList by scalar multiplication
* Detail: Given a list of polynomials, this function analyses
* each polynomial in turn. multiplying a polynomial by an
* appropriate integer if a fractional coefficient is
* foumd for any term in the polynomial. For example, if one
* polynomial in the list is \((2 / 3) x y+(1 / 5) x+2 y\),
* then the polynomial is multiplied by \(3 * 5=15\) to remove
* the fractional coefficients, and the output polynomial
* is therefore \(10 x y+3 x+30 y\).
*/
FAlgList
fAlgListRemoveFractions( input )
FAlgList input;

FAlgList output \(=\mathrm{fAlgListNul}\);
FAlg \(\mathrm{p}, \mathrm{LTp}\), new;
Integer denominator;
while( input ) // For each polynomial in the list
\{
\(\mathrm{p}=\) input \(->\) first; // Extract a polynomial
input \(=\) input \(->\) rest; // Advance the list
new \(=\) fAlgZero(); // Initialise the new polynomial
while( p ) // For each term of the polynomial \(p\) \{

LTp \(=\) fAlgLeadTerm ( p ); // Extract the Lead term.
\(\mathrm{p}=\mathrm{fAlgReductum}(\mathrm{p})\); // Advance the polynomial
denominator \(=\mathrm{fAlgLeadCoef}(\operatorname{LTp})->\) den; // Estract the denominator
```

    if(zIsOne(denominator) == 0)// If the denominator is not I
    {
        // Multiply the whole polynomial by the denominator
        if(p) p = fAlgZScaTimes(denominator, p ); // Still to be looked at
        LTp = fAlgZScaTimes( denominator, LTp ); // Looking at,
        new = fAlgZScaTimes( denominator, new ); // Looked at
        }
        new =fAlgPlus(new, LTp );// Add the term to the output polynomial
    }
    output = fAlgListPush(new, output ); // Add the new polynomial to the output list.
    }
// The new list was read in reverse so we must reverse it before returning it
return fAlgListFXRev(output );
1023 * =============
1024 * End of File
1025 * =============

```
\(1020\}\)
1021
1022 /*
1026 */

\section*{B.2.10 ncinv_functions.h}
```

/*
*/
//
//
//

```
* File: ncinv_functions.h
* Author: Gareth Evans
* Last Modified: 6th July 2005
// Initialise file definition
\# ifndef NCINV_FUNCTIONS_HDR
\# define NCINV_FUNCTIONS_HDR
// Include MSSRC Libraries
\# include <fralg.h>
// External Variables Required
extern ULong nOfProlongations, // Stores the number of prolongations calculated
nRed; // Stores how many reductions have been performed
extern int degRestrict, // Determines whether of not prolongations are restricted by degree
    EType, // Stores the type of Overlap Division.
    IType, // Stores the involutive division used
    nOfGenerators, // Holds the number of generators
    pl, // Holds the "Print Level"
    SType, // Determines how the basis is sorted
    MType; // Determines involutive division method
```

// Functions Defined in ncinv_functions.c
//
//
// Overlap Functions
//
// Returns the union of (non-)multiplicative variables (1st arg) and a generator (2nd arg)
FMon multiplicativeUnion( FMon, FMon );
// Does the generator (1st arg) appear in the list of multiplicative variables (2nd arg)?
int fMonIsMultiplicative( FMon, FMon );
// Does the Ist arg appear as a subword in the 2nd arg (yes (1)/no (0))
int fMonIsSubword( FMon, FMon );
// Is the 1st arg a subword of the 2nd arg; if so, return start pos in 2nd arg
ULong fMonSubwordOf( FMon, FMon, ULong );
// Returns size of smallest overlap of type (suffix of 1st arg = prefix of 2nd arg)
ULong fMonPrefixOf( FMon, FMon, ULong, ULong );
// Returns size of smallest overlap of type (prefix of 1st arg = suffix of 2nd arg)
ULong fMonSuffixOf(FMon, FMon, ULong, ULong );
//
// Multiplicative Variables Functions
//
// Returns no ('empty') multiplicutive vuriables
void EMultVars( FMon, ULong *, ULong *);
// All variables left mult., no variables right mult.
void LMultVars( FMon, ULong *, ULong *);
// All variables right mult., no variables left mult.
void RMultVars( FMon, ULong *, ULong * );
// Returns local overlap-based multiplicative variables
FMonPairList OverlapDiv( FAlgList );
//
// Polynomial Reduction and Basis Completion Functions
//
// R.educes 1st arg w.r.t. 2nd arg (list) and 3rd arg (vurs)
FAlg IPolyReduce( FAlg, FAlgList, FMonPairList );
// Autoreduces an FAlgList recursively until no more reductions are possible
FAlgList IAutoreduceFull( FAlgList );
// Implements Seiler's original algorithm for computing locally involutive bases
FAlgList Seiler( FAlgList );
// Implements Gerdt's advanced algorithm for computing locally involutive bases
FAlgList Gerdt( FAlgList );

# endif // NCINV_FUNCTIONS_HDR.

```

\section*{B.2.11 ncinv_functions.c}
```

* File: ncinv_functions.c
* Author: Gureth Evans
* Last Modified: 10th August 2005
*/
/*
* ==============================================
* Global Variables for ncinv.functions.c
* ==============================================
*/
int headReduce = 0; // Controls type of polynomial reduction
ULong d, // Stores the bound on the restriction of prolongations
twod; // Stores 2*d for efficiency
/*
* =====================
* Overlap Functions
* ==================
*/
/*
* Function Name: multiplicativeUnion
* Overview: Returns the union of (non--) multiplicative variables
* (1st arg) and a generator (2nd arg)
* 
* Detail: This function inserts a generator into a monomial representing
* (non-)multiplicative variables so that the ASCII ordering of the
* monomial is preserved. For example, if _a_ = A*B*C*E*F and _b_ = D,
* then the output monomial is A*B*C*D*E*F.
* 

*/
FMon
multiplicativeUnion( a, b )
FMon a, b;
{
FMon output = fMonOne();
ULong test, insert = ASCIIVal( fMonLeadVar(b ) ),
len = fMonLength(a );
// If a is empty there is no problem - we just return b
if(!a) return b;
else
{
// Go through each generator in a
while( len > 0)
{
len--;
// Obtain the numerical value of the first generator
test = ASCIIVal(fMonLeadVar( a ) );
if( test < insert ) // We must skip past this generator

```
```

        output = fMonTimes(output, fMonPrefix( a, 1));
        else if( test == insert ) // b is already in a so we just return the _original a
            return fMonTimes( output, a );
        else // We insert b in this position and tag on the remainder
            return fMonTimes( output, fMonTimes( b, a ) );
        // Get ready to look at the next generator
        a = fMonTailFac( a );
    }
    }
// Deal with the case "insert > { everything in a}"
return fMonTimes( output, b );
/*

* Function Name: fMonIsMultiplicative
* 
* Overview: Does the generator _a_ appear in the list of multiplicative variables _ b_?
* 
* Detail: Given a generator _a.,, this function tests to see whether
* _ a.. appears in a list of multiplicative variables ..b_.
* 

*/
int
fMonIsMultiplicative( a, b )
FMon a, b;
ULong lenb = fMonLength( b ), i;
// For each possible overlap
for( }\textrm{i}=1;\textrm{i}<=l=l\mp@code{len; i++)
{
if(fMonEqual( a, fMonSubWordLen(b, i, 1 ))==(Bool) 1)
return 1; // Match found
}
return 0; // No match found
/*

* Function Name: fMonIsSubword
* 
* Overview: Does _ a_ appear as a subword in _b_ (yes (1)/no (0))
* 
* Detail: This function answers the question "Is _ _ _ a subword of _ b_?"
* The function returns 1 if _ a_ is a subword of _ b_ and 0 otherwise.
* 

*/
int
fMonIsSubword( a, b )
FMon a, b;

```
\} \(\}\)
\{
\}
4
107 \{
fMonSubwordOf( small, large, start )
FMon small, large;
ULong start;
\{
ULong \(\mathrm{i}=\) start, sLen \(=\mathrm{fM}\) onLength \((\) small \()\), LLen \(=f\) MonLength ( large \() ;\)
// While there are more sulwords to test for
while( i <= lLen-sLen+1)
\{
    // If small is equal to a subword of large
    if( \(\mathrm{fMonEqual}(\) small, \(\mathrm{fMonSubWordLen}(\) large, i , sLen \())=(\) Bool \() 1)\)
    \{
        return i; // Subword found
    \}
    i++;
\}
return 0; // No subwords found
\}
156
57 /*
* Function Name: fMonPrefixOf
*
* Overview: Returns size of smallest overlap of type (suffix of 1st ary = prefix of 2nd ary)
fMonPrefixOf( left, right, start, limit )
FMon left, right;
ULong start, limit;
\{
ULong \(\mathrm{i}=\) start;
while( \(\mathrm{i}<=\) limit \() / /\) For each overlap
\{
    if( fMonEqual( fMonSuffix( left, i ), fMonPrefix( right, i ) ) \(==(\) Bool \() 1\) )
    \{
            return i; // Prefix found
    \}
    i++;
\}
return 0; // No prefices found
\}
190
1 /*
* Function Name: fMonSuffixOf
*
* Overview: Returns size of smallest overlap of type (prefix of 1st arg \(=\) suffix of 2nd arg)
*
* Detail: This function can answer the question "Is _left... a suffix of _right_?"
* The function returns i if a prefix of _left_ is cqual to a suffix of _right,
* where \(i\) is the length of the smallest overlap, and returns 0 if no overlup exists.
* The lengths of the overtaps we look at are controlled by the 3nd and 4th
* arguments - we start by looking at the overlap of size _start_ and finish.
* by looking at the overlap of size _limit.. It is the user's responsibility
* to ensure that these bounds are correct - no checks are made by the function.
* To test all possibilities, the 3rd argument should be 1 ard the fourth
* argument should be min \((\mid\) efit \(|\),\(| right \mid)-1\).
*
*/
ULong
fMonSuffixOf( left, right, start, limit )
FMon left, right;
ULong start, limit;
\{
ULong \(\mathrm{i}=\) start;
213
```

while( i <= limit ) // For each overlap
{
if(fMonEqual(fMonPrefix( left, i ), fMonSuffix(right, i ) )== (Bool) 1 )
{
return i; // Sulfix found
}
i++;
}
return 0; // No suffixes found

```

```

* Multiplicative Variables Functions
* =======================================
*/
/*
* Function Name: EMultVars
* Overview: Returns no ('empty') multiplicative variables
* 
* Detail: Given a monomial, this function assigns
* no multiplicative variables.
* 
* External Variables Required: int nOfGenerators;
* 

*/
void
EMultVars(mon, max, min )
FMon mon;
ULong *max, *min;
// Nothing is right multiplicative
*max = (ULong)nOfGenerators + 1;
// Nothing is left multiplicative
*min =0;
/*

* Function Name: LMultVars
* 
* Overview: All variables left mult., no variables right mult.
* 
* Detail: Given a monomial, this function assigns
* all variables to be left multiplicative and all
* variables to be right nonmultiplicative.
* 
* External Variables Required: int nOfGenerators;
* 

*/
void
LMultVars(mon, max, min )

```
\}
224
5/*
\{
\}
2
```

FMon mon;
ULong *max, *min;
{
// Nothing is right multiplicative
*max = (ULong)nOfGenerators + 1;
// Everything is left multiplicative
*min = (ULong)nOfGenerators + 1;
}
76 /*
* Function Name: RMultVars
*

* Overview: All variables right mult., no variables left mult.
* 
* Detail: Given a monomial, this function assigns
    * all variables to be right multiplicative and all
    * variables to be left nonmultiplicative.
    * 
    * Extermal Variables Required: int nOfGenerators;
    * 

*/
void
RMultVars( mon, max, min )
FMon mon;
ULong *max, *min;
{
// Everything is right multiplicative
*max = 0;
// Nothing is left multiplicative
*min = 0;
}
299/*
* Function Name: OverlapDiv
* 
* Overview: Returns local overlap---based multiplicative variables
* 
* Detail: This function implements various algoritlms
* described in the thesis "Noncommutative Involutive Buses"
* for finding left and right multiplicative variables
* for a set of polynomials based on the overlaps
* between the leading monomials of the polynomials.
* 
* External Variables Required: int EType, IType, nOfGenerators, pl, SType;
* 

*/
FMonPairList
OverlapDiv( list)
FAlgList list;
{
FMonPairList output = fMonPairListNul;
FMon generator;
ULong listLen = fAlgListLength( list ),

```
275
298
```

    monLength[listLen], tracking[listLen],
    i, j, first, limit, result, len,
    letterVal1, letterVal2;
    FMon monomials[listLen], monExcl,
leftMult[listLen], rightMult[listLen];
short grid[listLen][(ULong)nOfGenerators * 2],
thresholdBroken, excludeL, excludeR;
// Give some initial information
if( pl > 3)
{
printf("OverlapDiv'suInput
fAlgListDisplay( list );
}
if( !list ) return output;
// Set up arrays
i = 0;
while( list ) // For each polynominl
{
monomials[i] = fAlgLeadMonom( list }->\mathrm{ first ); // Extract lead monomial
monLength[i] = fMonLength( monomials[i] );// Find monomial length
leftMult[ [] = fMonOne(); // Initialise left multiplicative variables
rightMult[i] = fMonOne(); // Initialise right multiplicative variables
for( j = 0; j < (ULong) nOfGenerators*2; j++ )
{
* Fill the maltiplicative grid with 1's
* where the columns of the grid are
* gen_1^L, gen_1^R, gen_2^L, gen_2`R, .., gen_{nOfGenerators} ^R
* and the rows of the grid are
* monomials[0], monomials[1], ..., monomials[listLen].
*/
grid[i][j] = 1;
}
// If SType > 1 we need to sort the basis first, keeping track of the changes made
if( SType > 1 ) tracking[i] = i;
i++;
list = list }->\mathrm{ rest; // Advance the list
}

```

```

// If STupe > 1 and there is more than one polynomial in the basis,
// we need to sort the basis w.r.t. DegRevLex (Grentest first) in order
// to be able to apply the algorithm.
if( ( SType > 1 ) \&\& (listLen > 1 ))
{
multiplicativeQuickSort( monomials, monLength, tracking, 0, listLen - 1);
if( pl > 6 )
{

```
```

        printf("Sorted
        for(i = 0; i < listLen; i++ ) printf("%s\n", fMonToStr( monomials[i] ) );
    }
    }
/*
* Now exclude multiplicative variables based on overlaps
*/
// For each monomial
for( }\textrm{i}=0;\textrm{i}<l\mathrm{ listLen; i++ )
{
thresholdBroken = 0;
for( }\textrm{j}=\textrm{i};\textrm{j}<l\mathrm{ listLen; j++ )// For each monomial less than or equal to monomial i in DRL
{
/*
* To look for subvoords, the length of monomial j has to
* be less than the length of monomial i. We use the variable
* thresholdBroken to store whether monomials of length less
* than the length of monomial i have been encountered yet.
* and obviously we must have j>i for this to be the case.
*/
if(( j > i ) \&\& ( thresholdBroken == 0))
{
if( monLength[j] < monLength[i] )
thresholdBroken = 1; // if deg(j) < deg(i) we can now start to consider subwords
}
if((thresholdBroken == 1) \&\& ( EType != 5 ) ) // Stage 1: Look for subwords
{
first = 1;
// There are monLength[i] - monLength[j] + 1 test subwords in all
limit = monLength[i] - monLength[j] + 1;
// Test whether monomial j is a subword of monomial i, starting with the first subword
result = fMonSubwordOf( monomials[j], monomials[i], first );
if( pl > 8 ) printf("fMonSubwordOf ( }%%\textrm{s},\stackrel{\iota}{~
fMonToStr( monomials[i] ), first, result );
while(result != 0) // While there are subwords to be processed
{
if( IType == 1)// Left Overlap Division
{
if( result < limit )
{
if((EType < 4)|(( EType == 4)\&\& (result == 1)))
{
/*
* Exclude right multiplicative variable - overlap of type 'B' or 'C'

```

```

                    * ............................. monomial[j] (space on the right)
                    * Note: the above diagram (and the following diagrams) may
                    * not appear correctly in Appendix B due to using, flexible columns.
                    * The correct diagrams (referenced by the letters 'A ' to 'D' can
                    * be found in the README file in. Appendix B.
    ```
```

            */
        generator = fMonSubWordLen( monomials[i], result + monLength[j], 1);
        letterVal1 = ASCIIVal( fMonLeadVar( generator ) ) - 1;
        grid[j][2*letterVal1+1] = 0; // Set right non multiplicative.
        }
    }
    else if( EType == 3)
    {
        /*
            * Exclude left. multiplicative variable - overlap of type 'D'
            * -------------- monomial[i]
            * x-------- monomial[j] (no space on the right)
            */
        generator = fMonSubWordLen( monomials[i], result-1, 1 );
        letterVal1 = ASCIIVal(fMonLeadVar( generator )) - 1;
        grid[j][2*letterVal1] = 0; // Set left non multiplicative
    }
    }
else // Right Overlap Division
{
if(result > 1 )
{
if((EType < 4)|(( EType == 4)\&\& (result == limit ) ))
{
/*
* Ecclude left multiplicative varialle --- overlap of type 'B' or 'C'

```

```

            * x- monomial[j] (space on the left)
            */
            generator = fMonSubWordLen( monomials[i], result-1, 1);
            letterVal1 = ASCIIVal( fMonLeadVar(generator )) - 1;
            grid[j][2*letterVal1] = 0; // Set left non multiplicative
        }
    }
    else if( EType == 3 )
    {
        /*
            * Exclude right multiplicative variable -- overlap of type 'D'
            * -..- -. - monomial[i]
            * ---------- monomial[i] (no space on the left)
            */
        generator = fMonSubWordLen( monomials[i], result + monLength[j], 1);
        letterVal1 = ASCIIVal(fMonLeadVar( generator ) ) - 1;
        grid[j][2*letterVal1+1] = 0; // Set right non multiplicative
    }
    }
// We will now look for the next availuble subworl
first = result + 1;
if( first <= limit ) // If the limit has not been excceded
{
result =fMonSubwordOf( monomials[j], monomials[i], first ); // Look for more suhnords
if(pl>8) printf("fMonSubwordOf (u%s,u%s, U%u)u=|%u\n", fMonToStr(monomials[j] ),

```
```

                    fMonToStr(monomials[i] ), first, result );
    }
    else // Otherwise exit from the loop
        result = 0;
    }
    }
// Stage 2: Look for prefixes
first = 1;
// There are monLength[ [j] - 1 test prefixes in all
limit = monLength[j] - 1;
// Test whether a suffix of monomial j is a prefix of monomial i, starting with the prefix of length I
result = fMonPrefixOf( monomials[j], monomials[i], first, limit );

```

```

                    fMonToStr( monomials[i] ), first, limit, result );
    while( result !=0) // While there are prefixes to be processed
{
* Possibly exclude right multiplicative variable - overlap of type 'A'
* 1------------ monomial [i]
* ---------2 monomial[j]
*/
generator = fMonSubWordLen( monomials[j], monLength[j] - result, 1 );
letterVal1 = ASCIIVal( fMonLeadVar( generator ) ) - 1;
generator = fMonSubWordLen( monomials[i], result + 1, 1);
letterVal2 = ASCIIVal( fMonLeadVar(generator ) ) - 1;
if( IType == 1) // Left Overlap Division.
{
if( EType != 3) // Assign right nonmaltiplicative.
{
grid[j][2*letterVal2+1] = 0;// Set j right non multiplicative for '2'
}
else // Assign nonmultiplicative only if both currently multiplicative
{
// If monomial i is left multiplicative for '1' and j right. multiplicative for '2:
if(}\operatorname{grid}[i][2*letterVal1] + grid[j][2*letterVal2+1] == 2)
grid[j][2*letterVal2+1] = 0;// Set j right non multiplicative for 'Q'
}
}
else // Right Overlap Division
{
if( EType != 3) // Assign left nonmultiplicative
{
grid[i][2*letterVal1] = 0; // Set i left non multiplicative for '1'
}
else // Assign nonmultiplicative only if both currently multiplicative
{
// If monomial i is left multiplicative for '1' and j right multiplicative for '2'
if(grid[i][2*letterVal1] + grid[j][2*letterVal2+1] == 2)
grid[i][2*letterVal1] = 0; // Set i left non multiplicative for ' }1\mathrm{ ',
}

```
\}
// We will now look for the neat available suffix:
first \(=\) result +1 ;
if ( first \(<=\) limit ) // If the limit has not been excceded
\{
        result \(=\mathrm{fMonPrefixOf}(\) monomials[ j\(]\), monomials[ i\(]\), first, limit \() ; / /\) Look for more prefixes

            fMonToStr( monomials[i] ), first, limit, result );
    \}
    else // Otherwise exit from the loop
        result \(=0\);
\}
// Stage 3: Look for suffixes
first \(=1\);
// There are monLength[ \([j]-1\) test suffixes in all
limit \(=\) monLength \([\mathrm{j}]-1\);
// Test whether a prefix of monomial \(j\) is a suffix of monomial \(i\), starting with the suffix of length. 1
result \(=\mathrm{fMonSuffixOf}(\) monomials \([\mathrm{j}]\), monomials[ \([\mathrm{i}]\), first, limit \()\);

                                    fMonToStr( monomials[i] ), first, limit, result );
while( result \(!=0\) ) // While there are suffixes to be processed
\{
    /*
    * Possibly exclude left multiplicative variable --.. overlap of type 'A'
    * - -n. ………
    * 2---------- monomial[ij]
    */
    generator \(=\) fMonSubWordLen \((\) monomials \([\mathrm{j}]\), result \(+1,1)\);
    letterVal1 \(=\) ASCIIVal( fMonLeadVar( generator \())-1\);
    generator \(=\mathrm{fMonSubWordLen}(\) monomials \([\mathrm{i}]\), monLength[ \([\mathrm{i}]-\) result, 1 );
    letterVal2 \(=\) ASCIIVal( fMonLeadVar( generator \())-1\);
    if( IType \(==1\) ) \(/ /\) Left Overlap Division
    \{
        if( EType \(!=3\) ) // Assign right nonmultiplicative.
        \{
            \(\operatorname{grid}[\mathrm{i}][2 *\) letterVal1 +1\(]=0 ; / /\) Set \(i\) right non multiplicative for ' 1 '
        \}
        else // Assign nonmultiplicative only if both currently multiplicative
        \{
            // If monomial \(i\) is right multiplicative for ' 1 ' and \(j\) left multiplicative for '2'
            if( \(\operatorname{grid}[\mathrm{i}][2 *\) letterVal1 +1\(]+\operatorname{grid}[\mathrm{j}][2 *\) letterVal2 2\(]==2)\)
            \(\operatorname{grid}[\mathrm{i}][2 *\) letterVal1 +1\(]=0 ; / /\) Set \(i\) right non multiplicative for \({ }^{\prime} 1\) ',
        \}
    \}
    else // Right Overlap Division
    \{
        if( EType \(!=3\) ) // Assign left nonmultiplicative
        \{
            \(\operatorname{grid}[j][2 *\) letterVal2 \(]=0 ; / /\) Set \(j\) left non multiplicative for \({ }^{\prime} 2^{\prime}\)
```

            }
            else // Assign nonmultiplicative only if both currently multiplicative
            {
                // If monomial i is right multiplicative for '1' and j left multiplicative for '2'
                    if(grid[i][2*letterVal1 +1] + grid[j][2*letterVal2] == 2)
                    grid[j][2*letterVal2] = 0; // Set j left non multiplicative for '2'
            }
        }
        // We will now look for the next available suffix
        first = result + 1;
        if( first <= limit ) // If the limit has not been exceeded
        {
            result =fMonSuffixOf( monomials[j], monomials[i], first, limit ); // Look for more suffixes
    ```

```

                        fMonToStr( monomials[i] ), first, limit, result );
        }
        else // Otherwise exit from the loop
            result = 0;
        }
    }
    }
if( EType == 2)
{
// Ensure all cones are disjoint
for( i = listLen; i > 0; i-- ) // For each monomial (working up)
{
for( }\textrm{j}=\mathrm{ listLen; j}>0;\textrm{j}--)// For each monomial
{
/*
* We will now make sure that some variable in monomial[j] is
* right (left) nonmultiplicative for monomial[i].
*/
// Assume to begin with that the above holds
if( IType == 1 )
{
first = 1; // Used to find the first variable
excludeL = 0
}
else excludeR = 0;
monExcl = monomials[j-1]; // Extract a monomial for processing
len =fMonLength( monExcl ); // Find the length of monExcl
while((len > 0) \&\& ((excludeL + excludeR )!=1))// For each variable in monomial[j]
{
len = len - fMonLeadExp(monExcl );
// Extract a variable
letterVal1 = ASCIIVal( fMonLeadVar( monExcl ) ) - 1;

```
```

            if( IType == 1)
            {
                if( first == 1)
                {
                    letterVal2 = letterVal1; // Store the first variable encounteved
                    first = 0;// To ensure this code only runs once
            }
            }
            if( IType == 1)// Left Overlap Division
            {
                // If this variable is right nonmultiplicative for monomialfil, change excludeL
                if( grid[i-1][2*letterVal1+1] == 0) excludeL = 1;
            }
            else // Right Overlap Division
            {
                // If this variable is left nonmultiplicative for monomial[i], change excludeR
                if(grid[i-1][2*letterVal1] == 0) excludeR = 1;
            }
            monExcl = fMonTailFac( monExcl );// Get, ready to look at the nert variable
        }
        if( IType == 1)// Left Overlap Division
        {
            // If no variable was right nonmultiplicative for monomial/i]...
            if(excludeL == 0)
                grid[i-1][2*letterVal2+1] = 0;// ...set the first variable encountered to be right nonmultiplicative
        }
        else // Right Overlap Division
        {
            // If no variable was left nonmultiplicative for monomial[i]...
            if( excludeR == 0)
                grid[i-1][2*letterVal1] = 0;// ...set the last variable encountered to be left nonmultiplicative
        }
    }
    }
    }
// Provide some intermediate output information
if( pl > 6 )
{
printf("Multiplicative_Grid:\n");
for( }\textrm{i}=0;\textrm{i}<l\mathrm{ listLen; }\textrm{i}++\mathrm{ )
{
printf("Monomialu%)
for( j = 0; j < (ULong) nOfGenerators * 2; j++ ) printf("%i,u", grid[i][j] );
printf("\n");
}
printf("\n");
}
/*
Convert the grid to 2 arrays of FMons, where

```
```

* each FMon stores a list of multiplicative variables
    * in increasing variable order.
*/
if(SType > 1) // Need to sort as well
{
// Convert the grid to monomial data
for( }\textrm{i}=0;\textrm{i}<listLen; i++ )// For each monomial
{
for( j = 0; j < (ULong) nOfGenerators; j++ ) // For each variable
{
if(grid[i][2*j] == 1) // LEFT Assigned
{
// Multiply on the left by a multiplicative variable
leftMult[tracking[i]] = fMonTimes( leftMult[tracking[i]], ASCIIMon( j+1 ));
}
if(\operatorname{grid}[i][2*j+1]==1)// RIGHT' Assigned
{
// Multiply on the left by a multiplicative variable
rightMult[tracking[i]] = fMonTimes(rightMult[tracking[i]], ASCIIMon( j+1 ));
}
}
}
}
else // No sorting required
{
// Convert the grid to monomial data
for( }\textrm{i}=0;\textrm{i}<listLen; i++) // For each monomial.
{
for( }\textrm{j}=0;\textrm{j}<(\mathrm{ ULong) nOfGenerators; j++ ) // For each variable
{
if(grid[i][2*j] == 1) // LEFT Assigned.
{
// Multiply on the left by a multiplicative variable
leftMult[i] = fMonTimes( leftMult[i],ASCIIMon( j+1 ));
}
if(grid[i][2*j+1] == 1)// RIGHT Assigned
{
// Multiply on the left by a multiplicative variable
rightMult[i] = fMonTimes(rightMult[i], ASCIIMon( j+1 ));
}
}
}
}
// Convert the two arrays of FMons to an FMonPairList
for( }\textrm{i}=0;\textrm{i}<l\mathrm{ listLen; }\textrm{i}++\mathrm{ )
output = fMonPairListPush( leftMult[i], rightMult[i], output );
// Provide some final output information
if( pl>3)
{
printf("OverlapDiv's_OOutput

```
```

    fMonPairListMultDisplay(fMonPairListRev( output ));
    }
// Return the reversed list. (it was constructed in reverse)
return fMonPairListFXRev( output );
, }
/*

```

```

* Polynomial Reduction and Basis Completion Functions

```

```

*/
**

* Function Name: IPolyReduce
* Overview: Reduces 1st arg w.r.t. 2nd arg (list) and 3rd arg (vars)
* 
* Detail: Given a polynomial _poly., this function involutively
* reduces the polynomial with respect to the given FAlgList list.
* with associated left and right multiplicative variables _vars_.
* The type of reduction. (head reduction / full reduction) is
* controlled by the global variable headReduce.
* If IType > 3, we can take advantage of fast global reduction
* 
* External Variables Required: ULong nRed;
* int IType, pl;
* Global Variables Used: int headReduce;
* 

*/
FAlg
IPolyReduce( poly, list, vars )
FAlg poly;
FAlgList list;
FMonPairList vars;
{
ULong i, numRules = fAlgListLength( list ), len,
cutoffL, cutoffR, value, lenOrig, lenSub;
FAlg LHSA[numRules], back = fAlgZero(), lead, upgrade;
FMonPairList factors = fMonPairListNul;
FMon LHSM[numRules], LHSVL[numRules], LHSVR[numRules],
leadMonomial, leadLoopMonomial, JLeft, JRight,
facLft, facRt, JMon;
QInteger LHSQ[numRules], leadQ, leadLoopQ, lcmQ;
short flag, toggle, M;
int appears;
// Catch special case list is empty
if( !list) return poly;
// Convert the input list of polynomials to an array and
// create arrays of lead monomials and lead coefficients
for( }\textrm{i}=0;\textrm{i}<\mathrm{ numRules; i++ )

```
```

{

```

```

    LHSA[i] = list }->\mathrm{ first;
    LHSM[i] = fAlgLeadMonom( list }->\mathrm{ first );
    LHSQ[i] = fAlgLeadCoef( list }->\mathrm{ first );
    if( IType < 3 ) // Using Local Division.
    {
        // Create array of multiplicative variables
        LHSVL[i] = vars }->\mathrm{ lft;
        LHSVR[i] = vars }->\mathrm{ rt;
        vars = vars }->\mathrm{ rest;
    }
    list = list -> rest;
    }
// We will now recursively reduce every term in the polynomial
// until no more reductions are possible
while(fAlgIsZero(poly ) == (Bool)0)
{
if( pl > 5 ) printf("Lookinguat_Lead_Term
toggle =1;// Assume no reductions are possible to begin with
lead = fAlgLeadTerm(poly );
leadMonomial = fAlgLeadMonom( lead );
leadQ = fAlgLeadCoef( lead );
i = 0;
while( i < numRules ) // For each polynomial in the list
{
if( IType >= 3 ) lenOrig = fMonLength( leadMonomial );
leadLoopMonomial = LHSM[i]; // Pick a test monomial
flag = 0;
if( IType < 3) // Local Division
{
// Does the ith polynomial divide our polynomial?
// If so, place all possible ways of doing this in factors
factors =fMonDiv( leadMonomial, leadLoopMonomial, \&flag );
}
else
{
if( IType == 5 )
factors }=\textrm{fMonPairListNul; // No divisors w.r.t. Empty Division
else
{
lenSub = fMonLength( leadLoopMonomial );
// Check if a prefix/sulfix is possible
if(lenSub <= lenOrig )
{
if( IType == 3)// Left Division; look for Suffix
{
if(fMonEqual( leadLoopMonomial, fMonSuffix(leadMonomial, lenSub ) ) == (Bool)1 )
{

```
```

                if( lenOrig == lenSub )
                    factors =fMonPairListSingle(fMonOne(), fMonOne() );
            else
                    factors = fMonPairListSingle( fMonPrefix(leadMonomial, lenOrig-lenSub ), fMonOne() );
            flag = 1;
        }
        }
        else if( IType == 4) // Right Division; Look for Prefic
        {
            if(fMonEqual( leadLoopMonomial, fMonPrefix(leadMonomial, lenSub )) == (Bool) 1 )
            {
                if( lenOrig == lenSub )
                    factors =fMonPairListSingle(fMonOne(), fMonOne() );
                else
                    factors = fMonPairListSingle( fMonOne(), fMonSuffix( leadMonomial, lenOrig-lenSub ) );
                flag = 1;
            }
        }
        }
    }
    }
if( flag == 1) // i.e. leadLoopMonomial divides leadMonomial
{
M = 0;// Assume that the first conventional division is not an involutive division
// While there are conventional divisions left to look at and
// while none of these have yet proved to be involutive divisions
while(( fMonPairListLength(factors ) > 0) \&\& ( M ==0 ))
{
// Assume that this conventional division is an involutive division
M = 1;
if( IType < 3) // Local Division
{
// Extract the ith left \& right multiplicative variables
JLeft = LHSVL[i];
JRight = LHSVR[i];
// Extract the left and right factors
facLft = factors }->\mathrm{ lft;
facRt = factors }-> rt
// Test all variables in facLft for left multiplicability in the ith monomial
len =fMonLength(facLft );
// Decide whether one/all variables in facLft are left multiplicative
if( MType == 1) // Right-most variable checked only
{
if(len > 0)
{
JMon = fMonSuffix( facLft, 1 );
appears =fMonIsMultiplicative( JMon, JLeft );
// If the generator doesn't appear this is not an involutive division

```
```

        if( appears ==0) M = 0;
    }
    }
else // All variables checked
{
while(len > 0)
{
len = len - fMonLeadExp( facLft );
// Extract a generator
JMon = fMonPrefix( facLft, 1);
// Test to see if the generator appears in the list of left multiplicative variables
appears = fMonIsMultiplicative( JMon, JLeft );
// If the generator doesn't appear this is not an involutive division
if( appears == 0)
{
M = 0;
break; // Esit from the while loop
}
facLft =fMonTailFac( facLft ); // Get ready to look at the next generator
}
}
// Test all variables in facRt for right multiplicability in the ith monominl
if( M == 1)
{
len = fMonLength(facRt );
// Decide whether one/all variables in facRt are left multiplicative
if(MType == 1 ) // Left-most. varinble checked only
{
if( len > 0)
{
JMon = fMonPrefix( facRt,1);
appears = fMonIsMultiplicative( JMon, JRight );
// If the generator doesn't appear this is not an involutive division.
if( appears ==0) M = 0;
}
}
else // All variables checked
{
while( len > 0)
{
len = len - fMonLeadExp(facRt );
// Extract a generator
JMon = fMonPrefix( facRt, 1 );
// Test to see if the generator appears in the list of right multiplicative variables
appears = fMonIsMultiplicative( JMon, JRight );
// If the generator doesn't appear this is not an involutive division
if( appears ==0)
{
M=0;
break; // Exit from the while loop
}

```
```

            facRt = fMonTailFac(facRt );
        }
    }
    }
}
else // Global division
{
M}=1;// Already potentially found an involutive divisor
// but include code below for reference
/*
// Obtain global cutoff positions
if(IType==3) LMultVars(leadLoopMonomial, \&cutoffL, \&cutoffR );
else if( IType == 4) RMultVars(leadLoopMonomial, EcutoffL, Bcutoffr. );
else EMultVars( LeadLoopMonomial, EcutoffL, EcutoffR );
if( pl> 4) printf("cutoff(%s)=(%u,%u)\u", fMonToStr(leadLoopMonomial ), cutoffL, cutoffR );
// Extract the left and right factors
facLft = fuctors }->>>1ft
facRt = factors }->rt
// Test all variables in facLft. for left multiplicability in the ith monomial
len = fMonLength( facL.ft );
// Decide whether one/all variables in faclft are left multiplicative
if(MType ==1)// Right-most variable checked only
{
if(len>0)
{
JMon = fMonSuffix(facLft. 1 );
value = ASCIIVal(fMonLeadVar( IMon ));
if(value > cutoffR )}M=0\mathrm{ ;
}
}
else // All variables checked
{
while(len>0)
{
len =len -- fMonLeadEap(facLft );
// Obtain the ASCII value of the next generator
value = ASCIIVal(fMonLeadVar(facLft:));
if(value > cutoffR ) // If the generator is not left multiplicative
{
M=0;
break; // Exit from the while loop
}
facL.ft = fMonTailFac(facLft: );
}
}
// Test all variables in. facRt for right multiplicability in the ith monomial
len =fMonLength(facRt );

```
```

    // Decide whether one/all variables in facRt are left multiplicative
    if(MType == 1) // Left-most variable checked only
    {
        if(len>0)
        {
            value = ASCIIVal( fMonLeadVar( facRt ) );
            if(value < cutoffL ) M=0;
        }
    }
        else // All variables checked
        {
        while(len>0)
        {
            len =len --fMonLeadEcp(fucRt);
            // Obtain the ASCII value of the neact generator
            value = ASCIIVal( fMonLeadVar(facR.t.) ),
            if(value < cutoffL ) // If the generator is not right multiplicative
            {
                M=0;
                break; // Exit from the while loop
            }
            facRt= fMonTailFac( facRt );
        }
        }
        */
    }
    // If this conventional division wasn't involutive, look at the neat division
    if( M == 0) factors = factors }->\mathrm{ rest;
    }
// If un involutive division was found
if( M == 1)
{
if( pl>1) nRed++; // Increase the number of reductions carried out

```

```

                    fMonToStr( factors -> lft ), fMonToStr( leadLoopMonomial ),
                    fMonToStr( factors -> rt ) );
    toggle = 0; // Indicate a reduction has been carried out to exit the loop
    leadLoopQ = LHSQ[i]; // Pick the divisor's leading coefficient
    lcmQ = AltLCMQInteger( leadQ, leadLoopQ );// Pick 'nice'cancelling coefficients
    // Construct poly #i*-1* coefficient to get lead terms the same
    upgrade = fAlgTimes(fAlgMonom(qOne(), factors }->\mathrm{ lft ), LHSA[i] );
    upgrade = fAlgTimes( upgrade, fAlgMonom(qNegate(qDivide(lcmQ, leadLoopQ ) ), factors }->> rt ) )
    // Add in poly * coefficient to cancel off the lead terms
    upgrade = fAlgPlus( upgrade, fAlgScaTimes(qDivide( lcmQ, leadQ ), poly ) );
    // We must also now multiply the current discarded remainder by a factor
    back = fAlgScaTimes(qDivide( lcmQ, leadQ ), back );
    poly = upgrade; // In the next iteration we will be reducing the new polynomial upgrade
    if( pl > 5 ) printf("New}\mp@subsup{|}{\cup}{\primeWord
    ```
```

        }
    }
    if( toggle == 1) // The ith polynomial did not involutivcly divide poly
        i++;
        else // A reduction was carried out, exit the loop
        i= numRules;
    }
    if(toggle == 1) // No reductions were carried out; now look at the next term
    {
        // If only head reduction is required, return reducer
        if( headReduce == 1 ) return poly;
        // Otherwise add lead term to remainder and simplify the rest
        lead = fAlgLeadTerm( poly );
        back = fAlgPlus( back, lead );
        poly = fAlgPlus(fAlgNegate( lead ), poly );
        if( pl > 5 ) printf("New}\mp@subsup{|}{\cup}{}Remainder = = %%s\n", fAlgToStr(poly ))
    }
    }
return back; // Return the reduced and simplified polynomial
1087 * Function Name: IAutoreduceFull
1089 * Overview: Autoreduces an FAlgList recursively until no more reductions are possible
1 0 9 1 ~ * ~ D e t a i l : ~ T h i s ~ f u n c t i o n ~ i n v o l u t i v e l y ~ r e d u c e s ~ e a c h
1092 * member of an FAlgList w.r.t. all the other members
1093 * of the list, removing the polynomial from the list
1094 * if it is involutively reduced to 0. This process is
1095 * iterated until no more such reductions are possible.
1097 * External Variables Required: int degRestrict, IType, pl, SType;
1098 * Global Variables Used: ULong d, tutod;
1102 IAutoreduceFull( input )
1105 FAlg oldPoly, newPoly;
1106 FAlgList new, old, oldCopy;
107 FMonPairList vars = fMonPairListNul;
*
FAlgList input;

```

```

ULong pos, pushPos, len = fAlgListLength(input );
// If the input basis has more than one element.
if(len > 1)
{
// Start by reducing the final element (working backwards means
// that less work has to be done calculating multiplicative variables)

```
\}
1086 /*
1088
1090
1096
1099 *
1100 */
1101 FAlgList
1103
1104 \{
1108
1109
1110
1111
1112
1113
1114

1062
1063 1064 1065 1066
```

pos = len;
// If we are using a local division and the basis is sorted by DegRevLer.,
// the last polynomial is irreducible so we do not have to consider it.
if(( IType < 3 ) \&\& (SType == 1 ) ) pos --;
// Make a copy of the input basis for traversal
old = fAlgListCopy(input );
while( pos > 0) // For each polynomial in old
{
// Extract the pos-th element of the basis
oldPoly = fAlgListNumber( pos, old );
if( pl > 2) printf("Lookinguatuelement
// Construct busis without 'poly'
oldCopy = fAlgListCopy(old ); // Make a copy of old
// Calculate Multiplicative Variables if using a local division
if( IType < 3)
{
vars = OverlapDiv( oldCopy );
vars =fMonPairListRemoveNumber( pos, vars );
}
new = fAlgListFXRem( old, oldPoly ); // Remove oldPoly from old
old = fAlgListCopy(oldCopy ); // Restore old
// To recap, _old_ is now unchanged whilst _new_ holds all
// the elements of _old_ except _oldPoly_.
// Involutively reduce the old polynomial w.r.t. the truncated list
newPoly = IPolyReduce( oldPoly, new, vars );
// If the polynomial did not reduce to O
if( fAlgIsZero( newPoly ) == (Bool) 0)
{
// Divide the polynomial through by its GCD
newPoly = findGCD( newPoly );
if( pl > 2 ) printf("Reduced\lrcornerP\iotato\lrcorner%s\n", fAlgToStr( newPoly ) );
// Check for trivinl ideal
if(fAlgIsOne( newPoly ) == (Bool) 1) return fAlgListSingle(fAlgOne() );
// If the old polynomial is equal to the new polynomial
// (no reduction took place)
if(fAlgEqual(oldPoly, newPoly ) == (Bool) 1)
{
pos--; // We may procked to look at the next polynomial
}
else // Otherwise some reduction took place so we have to start again
{
// If we are restricting prolongations bnsed on degree,...
if(degRestrict == 1)

```
```

    {
        // ...and if the degree of the lead term of the new
        // polynomial excceds the current bound...
        if(fMonLength(fAlgLeadMonom(newPoly ) ) > d )
        {
            // ...ve must adjust the bound accordingly
            d = fMonLength(fAlgLeadMonom(newPoly ));
            if( pl > 1) printf("New\cupvalue
            twod = 2*d;
        }
        }
        // Add the new polynomial onto the list
        if( IType < 3) // Local division.
        {
        if(SType == 1 ) // DegRevLex sorted
        {
            // Push the new polynomial onto the list
            old = fAlgListDegRevLexPushPosition( newPoly, new, &pushPos );
            // If it is inserted into the same position, we may continue and look at the next polynomial
            if(pushPos == pos ) pos--;
            // If it is inserted into a later position we continue from one position above
            else if(pushPos > pos) pos = pushPos - 1;
            // Note: the case pushPos < pos cannot occur
        }
        else if( SType == 2 ) // No sorting
        {
            // Push the new polynomial onto the end of the list
            old =fAlgListAppend( new, fAlgListSingle( newPoly ) );
            // Return to the end of the list minus one
            // (we know the last element is irreducible)
            pos = fAlgListLength(old ) - 1;
        }
        else // Sorted by main ordering
        {
            // Push the new polynomial onto the list
            old = fAlgListNormalPush(newPoly, new );
            // Return to the end of the list
            pos = fAlgListLength( old );
        }
        }
        else // Global division
        {
            // Push the new polynomial onto the end of the list
            old = fAlgListAppend( new, fAlgListSingle( newPoly ) );
            // Return to the end of the list minus one
            // (we know the last element is irreducible)
            pos = fAlgListLength(old ) - 1;
        }
        }
    }
else // The polynomial reduced to zero
{

```
```

        // Remove the polynomial from the list
        old = fAlgListCopy(new );
        // Continue to look at the neact element
        pos--;
        if( pl > 2) printf("Reduced
        }
    }
    }
else // The input basis is empty or consists of a single polynomial
return input;
// Return the fully autoreduced basis
return old;
}

* Function Name: Seiler
* Overview: Implements Seiler's original algorithm for computing locally involutive bases
* Detail: Given a list of polynomials, this algorithm computes a
* Locally Involutive Basis for the input basis by the following
* iterative method: find all prolongations, choose the 'lowest'
* one, autoreduce, find all prolongations, ...
* 
* External Variables Required: int degRestrict, IType, nOfGenerators, pl, SType;
* ULong nOfProlongations;
* Global Variables Used: ULong d, twod;
* 

*/
FAlgList
Seiler( FBasis )
FAlgList FBasis;
FAlgList H = fAlgListNul, HCopy = fAlgListNul, soFar = fAlgListNul, S;
FAlg g, gNew, h;
FMonPairList vars =fMonPairListNul, varsCopy,
factors = fMonPairListNul;
FMon all, LMh, Lmult, Rmult, nonMultiplicatives;
ULong precount, count, degTest, len, i, cutoffL, cutoffR;
short escape, degBound, flag, trip;
if( pl > 0) printf("\nComputing\sqcupan}|\mathrm{ Involutive
if( IType < 3) // Local division
{
// Create a monomial containing all generators
all = fMonOne();
for( }\textrm{i}=1;\textrm{i}<=(\mathrm{ (ULong) nOfGenerators; i++ )
all = fMonTimes( all, ASCIIMon( i ) );
}
// If prolongations are restricted by degree

```
1235
1236 /*
```

if(degRestrict == 1)
{
d=maxDegree(FBasis); // Initialise the value of d
if( pl > 1) printf("Initial|value
/*
* There is no point in looking at prolongations of length
* 2*d or more as these cannot possibly be associated with
* S-Polynomials - they are in effect 'disjoint overlaps'.
*/
twod = 2*d;
}
// Turn head reluction off
headReduce = 0;
// Remove duplicates from the input basis
FBasis = fAlgListRemDups(FBasis );
// If the basis should be kept sorted, do the initial sorting now
if(( IType < 3) \&\& ( SType != 2 ) ) FBasis = fAlgListSort(FBasis, SType );
// Now Autoreduce FBasis and place the result in.H
if( pl > 1) printf("Autoreducing...\n");
precount =fAlgListLength(FBasis );// Determine size of basis before autoreduction
H = IAutoreduceFull( FBasis ); // Fully cutoreduce the basis
count =fAlgListLength(H); // Deternine size of busis after autoreduction
if(( pl > 0) \&\& (count < precount ))

```

```

    // Check for trivial ideal
    if(( count == 1) & (fAlgIsOne( H -> first ) == (Bool) 1 ) )
    return fAlgListSingle(fAlgOne() );
    /*
* soFur will store all polynomials that will appear in H
* at any time so that we do not introduce duplicates into the set.
* To begin with, all we have encountered are the polynomials
* in the autoreduced input basis.
*/
soFar = fAlgListCopy(H);
escape = 1; // To enable the following white loop to begin
while( escape == 1)
{
if( IType < 3) // Calculate multiplicative variables for CBasis
{
vars = OverlapDiv( H );
varsCopy = fMonPairListCopy( vars ); // Make a copy for traversal
}
HCopy = fAlgListCopy(H); // Make a copy of H for traversal

```
```

// $S$ will hold all the possible prolongations
$\mathrm{S}=\mathrm{fAlgListNul}$;
while( HCopy ) // For each $\$ h \backslash$ in $H \$$
\{
$\mathrm{h}=\mathrm{HCopy} \rightarrow>$ first; // Extract a polynomial
$\mathrm{LMh}=\mathrm{fAlgLeadMonom}(\mathrm{h}) ; / /$ Find the lead monomial
if ( $\mathrm{pl}==3$ ) printf("Analysing $\% \mathrm{~s} \ldots \backslash \mathrm{n} "$, fMonToStr( LMh ) );
if( $\mathrm{pl}>3$ ) $\operatorname{printf("Analysing~} \% \mathrm{~s} . . . \backslash \mathrm{n} "$, fAlgToStr( h ) );
HCopy $=$ HCopy $\rightarrow$ rest; // Advance to the next polynomial
// Assume to begin with that any prolongations of this polynomial are OK
degBound $=0$;
if $($ degRestrict $==1) / /$ If we are restricting prolongations by degree...
\{
// ...and if the length of any prolongation of $g$ excecds the bound...
if( fMonLength(LMh ) $+1>=$ twod )
\{
// .ignore all prolongations involving this polynomial
$\operatorname{deg}$ Bound $=1$;
if $(\mathrm{pl}>2)$ printf("Degree ${ }_{\lrcorner} \circ f_{\lrcorner} l_{\text {lead }}^{\lrcorner}$term $_{\lrcorner}$exceeds $\left._{\llcorner } 2 * d-1 \backslash \mathrm{n} "\right)$;
if( IType < 3) // Local division - advance to the next polynomial
varsCopy $=$ varsCopy $\rightarrow>$ rest;
\}
\}
// Step 1 - find all prolongations
if $(($ IType $<3) \& \&(\operatorname{degBound}==0)) / /$ Local division
\{
// Extract the left and right multiplicative variables for this polynomial
Lmult $=$ varsCopy $\rightarrow>\mathrm{lft}$;
Rmult $=$ varsCopy $\rightarrow$ rt;
varsCopy $=$ varsCopy $->$ rest;
// LEFT PROLONGATIONS
// Construct the left nonmultiplicative vuriables
nonMultiplicatives $=$ all;
while( $\mathrm{fMonIsOne}(\mathrm{Lmult})!=(\mathrm{Bool}) 1) / /$ For each left multiplicative variable
\{
// Eliminate one multiplicative variable
factors $=\mathrm{fMonDivFirst}($ nonMultiplicatives, fM MonPrefix (Lmult, 1 ), \&flag );
nonMultiplicatives $=\mathrm{fMonTimes}($ factors $\rightarrow \mathrm{lft}$, factors $\rightarrow \mathrm{rt})$;
Lmult $=\mathrm{fMonRest}($ Lmult $)$;
\}
Lmult $=$ nonMultiplicatives;
// Find the number of left nonmultiplicative variables
len $=\mathrm{fMonLength}($ Lmult $) ;$
// For each variable $\$ x-i \$$ that is not Left Multiplicative for $\$ L M(g) \$$
for $(\mathrm{i}=1 ; \mathrm{i}<=\operatorname{len} ; \mathrm{i}++$ )
\{

```


```

    S = fAlgListPush(fAlgTimes(fAlgMonom(qOne(), fMonPrefix( Lmult, 1 ) ), h ), S );
    Lmult = fMonRest( Lmult );
    }
    // RIGHT PROLONGATIONS
    // Construct the right nonmultiplicative variables
    nonMultiplicatives = all;
    while( fMonIsOne( Rmult ) != (Bool) 1) // For each right multiplicative variable.
    {
        // Eliminate one multiplicative variable
        factors = fMonDivFirst( nonMultiplicatives, fMonPrefix( Rmult, 1), &flag );
        nonMultiplicatives = fMonTimes( factors }->>1ft, factors -> rt )
        Rmult = fMonRest( Rmult );
    }
    Rmult = nonMultiplicatives;
    // Find the number of right nonmultiplicative variables
    len =fMonLength( Rmult );
    // For each variable $x_i$ that is not Right Multiplicative for $LM(g)$
    for( }\textrm{i}=1;\textrm{i}<=l=l\mp@code{le}; i++
    {
        if( pl == 3) printf("AddingムRight
    ```

```

        S = fAlgListPush(fAlgTimes(h, fAlgMonom(qOne(), fMonPrefix( Rmult, 1)) ), S );
        Rmult = fMonRest( Rmult );
    }
    }
else if((IType >= 3) \&\& (degBound == 0)) // Global division
{
// Find the multiplicative variables for this monomial
if( IType == 3 ) LMultVars( LMh, \&cutoffL, \&cutoffR );
else if( IType == 4) RMultVars( LMh, \&cutoffL, \&cutoffR );
else EMultVars( LMh, \&cutoffL, \&cutoffR );
if( pl > 4) printf("cutoff(%s)
// LEFT PROLONGATIONS
// For each variable $x_i$ that is not Left Multiplicative for $LM(g)$
for( i = cutoffR; i < (ULong) nOfGenerators; i++ )
{
// Construct a nonmultiplicative variable
Lmult = ASCIIMon(i+1 );
if( pl == 3) printf("AddingムLeftt\Prolongation
if( pl > 3) printf("Adding}\mp@subsup{L}{Left}{|
S = fAlgListPush(fAlgTimes(fAlgMonom(qOne(), Lmult ), h ), S );
}
// RIGHT PROLONGATIONS

```
```

    // For each variable $x_i$ that is not Right Multiplicative for $LMM(g)$
    for( }\textrm{i}=1;\textrm{i}<\mathrm{ cutoffL; i++ )
    {
        // Construct a nonmultiplicative variable
        Rmult = ASCIIMon( i );
    ```

```

        if( pl > 3 ) printf("Adding\lrcornerRight
        S = fAlgListPush(fAlgTimes(h, fAlgMonom(qOne(), Rmult ) ), S );
        }
    }
    }
// Step 2 -- Find the lowest prolongation w.r.t. chosen monomial order
// Turn head reduction on when finding a suitable prolongation.
headReduce = 1;
// If there are no prolongations we may exit the loop
if( !S ) escape = 0;
else
{
// Sort the list of prolongations w.r.t. the chosen monomial order
S = fAlgListSort(S, 3 );
// Reverse the list so that the 'lowest' prolongation comes first
S = fAlgListFXRev(S );
// Obtain the first non-mero head--reduced element of the list
g = S }->>\mathrm{ first; // Extract a prolongation
trip = 0;
// While there are prolongations left to look at and while we have
// not yet found a non-zero head-reduced prolongation.
while((fAlgListLength(S ) >0) \&\& (trip == 0))
{
// Involutively head-reduce the prolongation
gNew = IPolyReduce( g, H, vars );
if(fAlgIsZero(gNew )== (Bool) 0) // If the prolongation did not reduce to zero
{
// Turn off head reduction
headReduce = 0;
// 'Fully' involutively reduce
gNew = IPolyReduce(gNew, H, vars );
gNew = findGCD(gNew ); // Divide through by the GCD
// Turn head reduction back on
headReduce = 1;
// If we have not encountered this polynomial before
if(fAlgListIsMember(gNew, soFar ) == (Bool) 0)
{
trip = 1;// We may exit the loop
headReduce = 0;// We do not need head reduction any more
}
else // Otherwise we go on to look at the next prolongation
{

```
```

        S = S }->\mathrm{ rest; // Advance the list:
        if(S ) g=S mirst; // If there are any more prolongations extract one
        }
    }
    else // Otherwise we go on to look at the neat prolongation
    {
        S = S -> rest; // Advance the list:
        if(S ) g = S }->>\mathrm{ first; // If there are any more prolongations extract one
    }
    }
// If no suitable prolongations were found we may exit the loop
if( !S ) escape = 0;
else
{
// Step 3 - Add the polynomial to the basis
if( pl > 2 ) printf("First |Non-Zerou_Reduced
if( pl > 2 ) printf("Prolongation
nOfProlongations++; // Increase the counter for the number of prolongations processed.
// Check for trivial ideal
if(fAlgIsOne(gNew ) == (Bool) 1 ) return fAlgListSingle(fAlgOne() );
// Adjust the prolongation degree bound if necessary
if( degRestrict == 1)
{
if(fAlgEqual(g,gNew )==(Bool) 0) // If the polynomial was reduced...
{
degTest = fMonLength( fAlgLeadMonom( gNew ) );
if(degTest > d ) // ...and if the degree of the new polynomial exceeds the bound...
{
// ...adjust the bound accordingly
d = degTest;
if( pl > 1) printf("New⿱value
twod = 2*d;
}
}
}
// Push the new polynomial onto the list
if( IType < 3 ) // Local division
{
if(SType == 1) H = fAlgListDegRevLexPush(gNew, H ); // DegRevLex sort
else if(SType == 2) H = fAlgListAppend(H,fAlgListSingle(gNew ) ); // No sorting - just append
else H=fAlgListNormalPush( gNew, H ); // Sort by monomial ordering
}
else H = fAlgListAppend( H, fAlgListSingle( gNew ) ); // Just append onto end
count++; // Increase the counter for the number of polynomials in the basis

```


```

    // Indicate that we have encountered a new polynomial for future reference
    ```
```

    soFar = fAlgListPush(gNew, soFar );
        // Step 4 -- Autoreduce
        precount = count; // Determine size of basis before autoreduction
        H = IAutoreduceFull( H ); // Fully autoreduce the basis
        count = fAlgListLength( H ); // Determine size of basis after autoreduction
        if(( pl>0)&& (count < precount ))
        printf("Autoreduction
        // Check for trivial ideal
        if( ( count == 1) && ( fAlgIsOne( H >> first ) == (Bool) 1))
        return fAlgListSingle( fAlgOne() );
        }
    }
    }
if( pl > 0 ) printf("...Involutive}\mp@subsup{|}{\cup}{\primeBasis
headReduce = 0; // Reset the value of herdReduce
return H;
}

* Function Name: Gerdt
* Overview: Implements Gerdt's advanced algorithm for computing locally involutive bases
* 
* Detail: Given a list of polynomials, this algorithm computes a
* Locally Involutive Basis for the input basis using the method
* outlined in the paper "Involutive Division Technique:
* Some generalisations and optimisations" by V.P. Gerdt.
* 
* External Variables Required: int deyRestrict, IType, nOfGenerators, pl, SType;
* ULong nOfProlongations:
* Global Variables Used: ULong d, twod;
* int headReduce;
* 

*/
FAlgList
Gerdt( FBasis )
FAlgList FBasis;
{
FAlgList GBasis = fAlgListNul, soFar = fAlgListNul,
Tp = fAlgListNul, Qp = fAlgListNul,
Tp2 = fAlgListNul, Qp2 = fAlgListNul;
FAlg f, g, h, gDotx, candidatePoly, testPoly;
FMonPairList Tv = fMonPairListNul, Qv = fMonPairListNul,
Tv2 = fMonPairListNul, vars =fMonPairListNul;
FMonList Tm = fMonListNul, Qm = fMonListNul,
Tm2 = fMonListNul;
FMonPair P, fVars, gVars, hVars;
FMon PL, PR, fVarsL, fVarsR, gVarsL, gVarsR, hVarsL, hVarsR,
LMf, LMg, LMh, all, DL, DR, gen, NML, NMR, u,

```
1560
\(1561 / *\)
* at any time so that we do not introduce duplicates into the set.
* To begin with, all we have encountered are the polynomials
* in the autoreduced input basis.
*/
soFar \(=\mathrm{fAlgListCopy}(\) FBasis \()\);
\(/ /\) Choose \(g \backslash\) in F with lowest LM(g) w.r.t. <
\(\mathrm{g}=\) fAlgListNumber( (fAlgListLowest( FBasis ) ), FBasis );
// Add entry ( \(g, L M(g)\), ( emptyset, \emptyset)) to \(T\)
\(\mathrm{Tp}=\mathrm{fAlgListPush}(\mathrm{g}, \mathrm{Tp})\);
\(\mathrm{Tm}=\mathrm{fMonListPush}(\) fAlgLeadMonom ( g\(), \mathrm{Tm}\) );
\(\mathrm{Tv}=\mathrm{fM}\) onPairListPush( fMonOne(), fMonOne(), Tv );
// Add entry to \(G\)
GBasis \(=\) fAlgListPush ( g , GBasis );
if ( pl > 1 ) printf("Adding \(\% \mathrm{~s}_{\cup} \operatorname{to}_{\lrcorner} \mathrm{G}_{\mathrm{\cup}}(\% \mathrm{u}) \ldots \backslash \mathrm{n} "\), fAlgToStr( g\()\), fAlgListLength( GBasis ) );

// For each \(f \backslash\) in FBasis \(\backslash\) setminus \(\{g\} \ldots\)
while( FBasis )
\{
    \(\mathrm{f}=\) FBasis \(->\) first;
    if (fAlgEqual (g, f) \(==(\) Bool \() 0\) )
    \{
        // Add entry ( \(f, L M(f)\). へemptyset. \emptyset)) to Q
        \(\mathrm{Qp}=\mathrm{fAlgListPush}(\mathrm{f}, \mathrm{Qp})\);
        \(\mathrm{Qm}=\mathrm{fMonListPush}(\mathrm{fAlgLeadMonom}(\mathrm{f}), \mathrm{Qm})\);
        \(\mathrm{Qv}=\mathrm{fMonPairListPush}(\mathrm{fMonOne}(), \mathrm{fMonOne}(), \mathrm{Qv})\);
    \}
    FBasis \(=\) FBasis \(\rightarrow\) rest;
\}

do // Repeat until \(Q\) is empty
\(\{\)
    \(\mathrm{h}=\mathrm{fAlg} \operatorname{Zero}()\);
    // While \(Q\) is not empty and \(h\) is not equal to 0
    while( \((\) fAlgListLength \((\mathrm{Qp})>0) \& \&(\operatorname{fAlgIsZero}(\mathrm{~h})==(\) Bool \() 1))\)
    \{
        // Choose the \(g\) in \((g, u(P L, P R)) \backslash\) in \(Q\) with lowest LM(g) w.r.t. \(<\)
        lowest \(=\mathrm{fAlgListLowest}(\mathrm{Qp})\);
    \(\mathrm{g}=\mathrm{fAlgListNumber}(\) lowest, Qp );
    \(\mathrm{u}=\mathrm{fM}\) onListNumber( lowest, Qm );
    \(\mathrm{P}=\mathrm{fMonPairListNumber}(\) lowest, Qv );
    if( \(\mathrm{pl}>2\) ) printf("Testing \(\lrcorner \mathrm{g} \mathrm{L}=\mathrm{\lrcorner} \% \mathrm{~s} \ldots \backslash \mathrm{n} "\), fAlgToStr \((\mathrm{g}))\);
    // Remove entry from \(Q\)
    \(\mathrm{Qp}=\mathrm{fAlgListRemoveNumber}(\) lowest, Qp\()\);
    \(\mathrm{Qm}=\mathrm{fM}\) onListRemoveNumber ( lowest, Qm );
    \(\mathrm{Qv}=\mathrm{fMonPairListRemoveNumber( }\) lowest, Qv );
```

    if( IType < 3) // Find Local Multiplicative Variables for GBasis
        vars \(=\) OverlapDiv( GBasis );
    // If the criterion is false... (to be implemented in the future...)
    \(/ /\) if (NCeriterion (g, u, Tp, Tm, GBasis, var's \()==0\) )
    \{
        // ...then find the normal form of \(g\) w.r.t. GBasis
        soFar \(=\) fAlgListPush ( g, soFar );
        \(\mathrm{h}=\) IPolyReduce( g, GBasis, vars ); // Find the involutive normal form.
        \(\mathrm{h}=\) findGCD \((\mathrm{h}) ; / /\) Divide through by the \(G C D\)
    ```

```

    \}
    \(/ /\) else if( \(p l>2\) ) printf("... Criterion used to discard \(g \ldots \backslash n\) ");
    \}
// If $h \backslash$ neq 0
if $($ fAlgIsZero $(\mathrm{h})==($ Bool $) 0)$
\{
// Add h to GBasis and recalculate multiplicative variables if necessary
if( IType < 3)
\{
pos $=1$;
if( SType ==1) GBasis = fAlgListDegRevLexPushPosition(h, GBasis, \&pos ); // DegRevLex sort
else if(SType ==2) GBasis = fAlgListAppend (GBasis, fAlgListSingle(h) ) // No sorting - just append
else GBasis $=$ fAlgListNormalPush $(\mathrm{h}$, GBasis $) ; / /$ Sort by monomial ovdering
vars $=$ OverlapDiv( GBasis $) ; / /$ Full recalculate
\}
else GBasis $=$ fAlgListAppend $($ GBasis, fAlgListSingle( h$)$ ); // Just append onto end

```


```

    \(\mathrm{LMh}=\mathrm{fAlgLeadMonom}(\mathrm{h}) ;\)
    if \((\) degRestrict \(=1) / /\) If we are restricting prolongations by degree...
    \{
        degTest \(=\mathrm{fMonLength}(\) LMh \() ;\)
        if \((\) degTest \(>\mathrm{d}) / /\)...and if the degree of the new polynomial exceeds the bound...
        \{
            // ...adjust the bound accordingly
            \(\mathrm{d}=\) degTest;
    ```

```

            twod \(=2 *\) d;
        \}
    \}
    //If \(L M(h)==L M(g)\)
    if( fMonEqual(fAlgLeadMonom (g ), LMh \()==(\) Bool \() 1)\)
    \{
        // Add entry to \(T\)
        \(\mathrm{Tp}=\mathrm{fAlgListPush}(\mathrm{h}, \mathrm{Tp})\);
        \(\mathrm{Tm}=\mathrm{fMonListPush}(\mathrm{u}, \mathrm{Tm})\);
    ```

```

// Find intersection of P and NM_I(h, G)
// (Note: NM_l(h, G) $=$ nonmultiplicative variables)
$\mathrm{PL}=\mathrm{P} . \mathrm{lft} ;$
$P R=P . r t ;$
if( IType < 3) // Local division.
\{
// Find NMI (h, GBasis)
pos $=$ fAlgListPosition ( $h$, GBasis );
hVars $=\mathrm{fMonPairListNumber}($ pos, vars $)$;
$\mathrm{hVarsL}=\mathrm{hVars} .1 \mathrm{ft}$;
hVarsR $=$ hVars.rt;
$\mathrm{NML}=\mathrm{fMonOne}()$;
$\mathrm{NMR}=\mathrm{fMonOne}()$;
$\mathrm{j}=1$;
// C'alculate the intersection
while( $\mathrm{j}<=$ (ULong) nOfGenerators )
\{
gen $=$ ASCIIMon( j );
// If gen appears in PL (nonmultiplicatives) but not in hVarsL (multiplicatives)
if $(($ fMonIsMultiplicative (gen, PL $)==1) \& \&($ fMonIsMultiplicative $($ gen, hVarsL $)=0))$
$\mathrm{NML}=\mathrm{fMonTimes}(\mathrm{NML}$, gen $) ; / /$ gen appears in the left intersection
// If gen appears in PR (nonmultiplicatives) but not in hVarsR (multiplicatives)
if $((\mathrm{fMonIsMultiplicative}($ gen, PR $)==1) \& \&($ fMonIsMultiplicative $($ gen, hVarsR $)==0))$
$\mathrm{NMR}=\mathrm{fMonTimes}(\mathrm{NMR}$, gen ); // gen appears in the right intersection
$\mathrm{j}++$; // Get ready to look at the next variable
\}
\}
else if( IType $>=3$ ) $/ /$ Global division
\{
// Find the multiplicative variables
if( IType ==3) LMultVars(LMh, \&cutoffL, \&cutoffR );
else if( IType ==4) RMultVars( LMh, \&cutoffL, \&cutoffR );
else EMultVars( LMh, \&cutoffL, \&cutoffR );
$\mathrm{NML}=\mathrm{fMonOne}()$;
$\mathrm{NMR}=\mathrm{fMonOne}()$;
// Calculate the left intersection
for $(\mathrm{j}=$ cutoffR $+1 ; \mathrm{j}<=$ (ULong) nOfGenerators; $\mathrm{j}++$ )
\{
gen $=$ ASCIIMon( j$) ; / /$ Obtain a nonmultiplicative variable
// If it appears in PL it appears in the intersection.
if( fMonIsMultiplicative( gen, PL ) $==1$ )
NML $=$ fMonTimes( NML, gen );
\}
// Calculate the right intersection

```
```

    for( j = 1; j < cutofl; j++ )
    {
        gen = ASCIIMon(j ); // Obtain a nonmultiplicative variable
        // If it appears in PR it appears in the intersection
        if(fMonIsMultiplicative(gen, PR ) == 1)
            NMR = fMonTimes( NMR, gen );
    }
    }
// Add an entry to Tv
Tv = fMonPairListPush( NML, NMR, Tv );
}
else // Add entry to T and adjust the lists
{
// Add entry to T
Tp = fAlgListPush( h, Tp );
Tm = fMonListPush(LMh, Tm );
Tv = fMonPairListPush(fMonOne(), fMonOne(), Tv );
if( pl>4) printf("Modifying|TU(size\&%u)...\n", fAlgListLength( Tp ) );
// Set up lists for neat operation
Tp2 = fAlgListNul;
Tm2 = fMonListNul;
Tv2 = fMonPairListNul;
// For each (f,v, (DL, DR)) \in T
if( pl > 4 ) printf("AdjustingムMultiplicative_Variables...\n");
while(Tp )
{
f=Tp -> first; // Extract a polynomial
LMf = fAlgLeadMonom(f);
if( pl > 4 ) printf("Testingu(%s, L%s)\n", fMonToStr( LMh ), fMonToStr( LMf ) );
// If LM(h)<LM(f)
if( theOrdFun( LMh, LMf ) == (Bool) 1)
{
// Add entry to Q
Qp = fAlgListPush( Tp -> first, Qp );
Qm = fMonListPush(Tm }->>\mathrm{ first, Qm );
Qv = fMonPairListPush( Tv >> lft, Tv >> rt, Qv );
// Discard f from GBrsis
GBasis = fAlgListFXRem(GBasis, f );

```

```

        else if ( }\textrm{pl}==1\mathrm{ ) printf("Discardedца⿱山Polynomialufrom
        }
        else
        {
            // Keep entry in T
            Tp2 = fAlgListPush(Tp -> first, Tp2 );
            Tm2 = fMonListPush( Tm -> first, Tm2 );
            Tv2 = fMonPairListPush(Tv -> lft, Tv -> rt, Tv2 );
    ```

1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1.901

1902
1903
1904
1905
1906
1907
1908
1909
\}
// Advance the lists to the next entry
\(\mathrm{Tp}=\mathrm{Tp} \rightarrow\) rest;
\(\mathrm{Tm}=\mathrm{Tm}->\) rest;
\(\mathrm{Tv}=\mathrm{Tv} \rightarrow\) rest;
\}
// Set up lists for nest operation.
\(\mathrm{Tp}=\mathrm{fAlgListNul} ;\)
\(\mathrm{Tm}=\mathrm{fM}\) onListNul;
\(\mathrm{Tv}=\mathrm{fMonPairListNul}\);
// Recalculate multiplicative variables
if( IType < 3) vars = OverlapDiv( GBasis );
// For each (f, v, (DL, DR)) \in T
while( Tp2)
\{
// Keep \(f\) and \(v\) as they are
\(\mathrm{f}=\mathrm{Tp} 2 \rightarrow\) first;
\(\mathrm{Tp}=\mathrm{fAlgListPush}(\mathrm{f}, \mathrm{Tp})\);
\(\mathrm{Tm}=\mathrm{fMonListPush}(\mathrm{Tm} 2 \rightarrow\) first, Tm \()\);
\(\mathrm{DL}=\mathrm{Tv} 2 \rightarrow \mathrm{lft}\);
\(\mathrm{DR}=\mathrm{Tv} 2 \rightarrow \mathrm{rt}\);
// Find intersection of D and NM_I(f, G)
if( IType < 3) // Local division
\{
// Find NM_I(f, GBasis)
pos \(=\mathrm{fAlgListPosition}(\mathrm{f}\), GBasis );
\(\mathrm{fVars}=\mathrm{fMonPairListNumber}(\) pos, vars \()\);
fV arsL \(=\mathrm{fV}\) ars.lft;
\(\mathrm{fVarsR}=\mathrm{fVars} . \mathrm{rt}\);
\(\mathrm{NML}=\mathrm{fMonOne}()\);
NMR \(=\) fMonOne();
\(\mathrm{j}=1\);
// Calculate the intersection
while( \(\mathrm{j}<=\) (ULong) nOfGenerators )
\{
gen \(=\) ASCIIMon ( j );
// If gen appears in DL (nonmultiplicatives) but not in fVarsL (multiplicatives)
if \(((\) fMonIsMultiplicative ( gen, DL \()==1) \& \&(f M o n I s M u l t i p l i c a t i v e(\) gen, fVarsL \()==0))\)
NML \(=\mathrm{fMonTimes}(\mathrm{NML}\), gen \() ; / /\) gen appears in the left intersection
// If gen appears in DR (nonmultiplicatives) but not in fVars \(R\) (multiplicatives)
if \(((\mathrm{fMonIsMultiplicative}(\) gen, DR\()==1) \& \&(\) fMonIsMultiplicative \((\) gen, fVarsR \()==0))\)
\(\mathrm{NMR}=\mathrm{fMonTimes}(\mathrm{NMR}\), gen \() ; / /\) gen appears in the right intersection
\(\mathrm{j}++\); // Get ready to look at the next variable
\}
\}
```

        else if (IType >= 3) // Global division
        {
            // Find the multiplicative variables
            if( IType == 3 ) LMultVars( fAlgLeadMonom(f), &cutofL, &cutoffR );
            else if( IType == 4 ) RMultVars( fAlgLeadMonom(f ), &cutoff, &cutoffR );
            else EMultVars(fAlgLeadMonom(f), &cutofL, &cutoff );
            NML = fMonOne();
            NMR = fMonOne();
            // Calculate the left intersection
            for( j = cutoffR+1; j <= (ULong) nOfGenerators; j++ )
            {
                gen = ASCIIMon(j); // Obtain a nonmultiplicative variable
            // If it appear's in DL it uppears in the intersection.
            if(fMonIsMultiplicative(gen, DL ) == 1)
                NML = fMonTimes( NML, gen );
            }
            // Calculate the right intersection
            for( j = 1; j < cutoflL; j++ )
            {
                    gen = ASCIIMon(j); // Obtain a nonmultiplicative variable
                    // If it appears in DR it appears in the intersection
            if(fMonIsMultiplicative( gen, DR ) == 1)
                NMR = fMonTimes( NMR, gen );
            }
        }
            // Add the nonmultiplicative variables to Tv
            Tv =fMonPairListPush( NML, NMR,Tv );
            // Advance the lists
            Tp2 = Tp2 -> rest;
            Tm2 = Tm2 -> rest;
            Tv2 = Tv2 -> rest;
        }
        }
    }
// Recalculate multiplicative variables
if( IType < 3 ) vars = OverlapDiv( GBasis );
// While exist (g, u, (PL, PR))\in T and }x<br>mathrm{ \in NM_I(g,GBasis \P and,
// if Q \neq \emptyset, s.t. LM(prolongation) < LM(f) for all f in
// (f,v,(DL,DR))\in Q do...
escape = 0;
while( escape ==0)
{
// Construct a candidate set for (g, u, (PL, PR)), c

```

```

    // Initialise variables
    ```
\(\mathrm{Tp} 2=\mathrm{fAlgListCopy}(\mathrm{Tp}) ;\)
Tm2 \(=\mathrm{fMonListCopy}(\mathrm{Tm})\);
\(\mathrm{Tv} 2=\mathrm{fMonPairListCopy}(\mathrm{Tv})\);
candidatePos \(=0\);
candidatePoly \(=\) fAlgZero();
candidateVariable \(=\mathrm{fMonOne}()\);
LorR \(=0\);
if( IType < 3 ) vars = OverlapDiv( GBasis );
// For each (g, u, (PL, PR)) in \(T\)
\(\mathrm{i}=1\);
while( Tp 2 )
\{
    // Extruct information about the first entry in \(T\)
    \(\mathrm{g}=\mathrm{Tp} 2->\) first;
    \(\mathrm{LMg}=\mathrm{fAlgLeadMonom}(\mathrm{g})\);
    \(\mathrm{PL}=\mathrm{Tv} 2 \rightarrow \mathrm{lft}\);
    \(\mathrm{PR}=\mathrm{Tv} 2 \rightarrow \mathrm{rt}\);
    // Advance the copy of \(T\)
    \(\mathrm{Tp} 2=\mathrm{Tp} 2->\) rest;
    \(\operatorname{Tm} 2=\operatorname{Tm} 2->\) rest;
    Tv2 \(=\) Tv2 \(->\) rest;
    if( IType < 3) // Local division
    \{
        pos \(=\) fAlgListPosition ( g, GBasis \()\);
        gVars \(=\mathrm{fMonPairListNumber}(\) pos, vars \()\);
        g VarsL \(=\mathrm{g}\) Vars.lft;
        gVarsR \(=\) gVars.rt;
        \(\mathrm{j}=1\);
        while( \(\mathrm{j}<=\) (ULong) nOfGenerators ) // For each generator
        \{
            gen \(=\) ASCIIMon( j );
            // LEFT PROLONGATIONS
        // Look for nonmultiplicative variables not in PL (unprocessed)
        if \(((\) fMonIsMultiplicative \((\) gen, PL \()==0) \& \&(\) fMonIsMultiplicative \((\) gen, \(g \operatorname{VarsL})==0))\)
        \{
            add \(=1 ; / /\) Candidate found
            mult \(=\mathrm{fMonTimes}(\) gen, fAlgLeadMonom( g ) ); // Construct. x.g
            // If \(Q\) is not empty
            if( Qp )
            \{
                \(/ /\) Make sure that \(L M(x . g)<L M(f)\) for all \(f\) in \((f, v, D) \backslash\) in \(Q\)
                \(\mathrm{Qp} 2=\mathrm{fAlgListCopy}(\mathrm{Qp}) ; / /\) Make a copy of \(Q\) for processing
                    while \(((\mathrm{fAlgListLength}(\mathrm{Qp} 2)>0) \& \&(\) add \(==1)) / /\) For all \(f\) in \((f, v, D) \backslash \operatorname{in} Q\)
                \{
                    // Extract a lead monomial
                    compare \(=\) fAlgLeadMonom( Qp2 \(\rightarrow>\) first \()\);
```

            Qp2 = Qp2 -> rest;
            // If LM(c.g) not less than LM(f) ignore this candidate
            if( theOrdFun( mult, compare ) == (Bool) 0) add = 0;
        }
    }
    if( add == 1) // Candidute found for (g, u,(PL,PR)),x
    {
        if(candidatePos > 0) // This is not the first candidate tried
            // Returns I if mult < fAlgLeadMonom(candidatePoly)
            balance = theOrdFun( mult, fAlgLeadMonom( candidatePoly ));
        // If we are restricting prolongations by degree
        if( degRestrict == 1)
        {
            // If the degree bound is not exceeded and the candidate is valid
            if((fMonLength(LMg ) + < twod ) && (( balance == (Bool) 1)|((candidatePos ==0)))
            {
                // Construct a candidate prolongation
                    testPoly = fAlgTimes( fAlgMonom( qOne(), gen ), g );
            // If we have not yet encountered this polynominl
            if(fAlgListIsMember(testPoly, soFar ) == (Bool) 0)
            {
                // We have found a new candidate
                    candidatePos = i;
                    candidatePoly = testPoly;
                    candidateVariable = gen;
                    LorR = 0; // Left prolongation.
            }
        }
        }
        // If we are not restricting prolongations by degree, proceed if
        // the candidate is valid (if this is the first candidate
        // encountered or LM(x.g) < LM(current candidate))
        else if}((\mathrm{ balance }==(\mathrm{ Bool ) 1) |( candidatePos ==0))
        {
            // Construct a candidate prolongation
            testPoly = fAlgTimes( fAlgMonom(qOne(), gen ), g );
            // If we have not yet encountered this polynomial
            if(fAlgListIsMember(testPoly, soFar ) == (Bool) 0)
            {
                // We have found a new candidate
                candidatePos = i;
                candidatePoly = testPoly;
                candidateVariable = gen;
                LorR = 0; // Left prolongation
            }
        }
    }
    }
// RIGHT PROLONGATIONS

```

2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081.

2082
2083
2084
2085
2086
2087
2088
```

// Look for nonmultiplicative variables not in PR (unprocessed)
if((fMonIsMultiplicative(gen, PR ) == 0) \&\& (fMonIsMultiplicative(gen, gVarsR ) == 0 ) )
{
add = 1; // Candidate found
mult =fMonTimes(fAlgLeadMonom(g ), gen ); // Construct g.x
// If Q is not empty
if( Qp )
{
// Make sure that LM }M(g.x)<LM(f)\mathrm{ for all fin (f,v,D)\in Q
Qp2 = fAlgListCopy( Qp ); // Make a copy of Q for processing
while((fAlgListLength(Qp2)>0) \&\& (add == 1))// For all f in (f,v,D)\in Q
{
// Extract a lead monomial
compare = fAlgLeadMonom( Qp2 -> first );
Qp2 = Qp2 -> rest;
// If LM(g.x) not less than LM(f) ignore this candidate
if(theOrdFun(mult, compare ) == (Bool) 0) add = 0;
}
}
if( add == 1) // Candidate found for (g, u, (PL, PR)), x
{
if(candidatePos > 0) // This is not the first candidate tried
// Returns 1 if mult < fAlgLeadMonom(candidatePoly)
balance = theOrdFun( mult, fAlgLeadMonom(candidatePoly ) );
// If we are restricting prolongations by degree
if( degRestrict == 1)
{
// If the degree bound is not exceeded and the candidate is valid
if((fMonLength( LMg ) + < < twod ) \&\& (( balance == (Bool)1)|( candidatePos == 0 ) ))
{
// Construct a candidate prolongation.
testPoly = fAlgTimes(g, fAlgMonom(qOne(), gen ) );
// If we have not yet encountered this polynomial
if(fAlgListIsMember(testPoly, soFar ) == (Bool)0)
{
// We have found a new candidate
candidatePos = i;
candidatePoly = testPoly;
candidateVariable = gen;
LorR = 1; // Right prolongation
}
}
}
// If ve are not restricting prolongations by degree, proceed if
// the candidate is valid (if this is the first candidate
// encountered or LM(g.x) < LM(current candidate))

```
```

        else if(( balance == (Bool)1)|(candidatePos==0))
            {
            // Construct a candidate prolongation
            testPoly = fAlgTimes(g, fAlgMonom(qOne(), gen ) );
            // If we have not yet encountered this polynomial
            if(fAlgListIsMember( testPoly, soFar ) == (Bool) 0)
            {
                // We have found a new candidute
                candidatePos = i;
                candidatePoly = testPoly;
                candidateVariable = gen;
                LorR = 1;// Right prolongation
            }
            }
        }
        }
        j++; // Move onto the next variable
    }
    }
else if( IType >= 3 ) // Global division
{
// Obtain the multiplicative variables for this polynomial
if( IType == 3 ) LMultVars(fAlgLeadMonom(g ), \&cutofL, \&cutoffR );
else if( IType == 4) RMultVars(fAlgLeadMonom(g), \&cutoflL, \&cutoffR );
else EMultVars( fAlgLeadMonom(g ), \&cutoflL, \&cutoffR );
// LEFT PROLONGATIONS
// For each left nonmultiplicative variable
for( }\textrm{j}=\mathrm{ cutoffR+1; j <= (ULong) nOfGenerators; j++ )
{
gen = ASCIIMon( j );
if(fMonIsMultiplicative(gen, PL ) == 0) // Not in P (unprocessed)
{
add = 1; // Candidate found
mult =fMonTimes(gen, fAlgLeadMonom(g ) ); // Construct x.g
// If Q is not empty
if( Qp )
{
// Make sure that LM(x.g)<LM(f) for all f in (f,v,D)\in Q
Qp2 = fAlgListCopy( Qp ); // Make a copy of Q for processing
while((fAlgListLength( Qp2) > 0) \&\& (add == 1)) // For all f in (f,v,D) \in Q
{
// Extract a lead monomial
compare = fAlgLeadMonom( Qp2 -> first );
Qp2 = Qp2 -> rest;
// If LM(x.g) not less than LM(f) ignore this candidate
if(theOrdFun(mult, compare ) == (Bool) 0) add = 0;
}

```
```

    }
    if(add == 1) // Candidate found for (g, u,(PL,PR)), x
    {
        if(candidatePos > 0) // This is not the first candidate tried
            // Returns 1 if mult < fAlgLeadMonom(candidatePoly)
            balance = theOrdFun(mult, fAlgLeadMonom(candidatePoly ) );
        // If we are restricting prolongations by degree
        if(}\mathrm{ degRestrict == 1)
        {
            // If the degree bound is not exceeded and the candidate is valid
            if((fMonLength( LMg ) + 1 < twod ) && (( balance == (Bool) 1)|( (candidatePos == 0)))
            {
                // Construct a candidate prolongation
                testPoly = fAlgTimes(fAlgMonom( qOne(), gen ),g );
                // If we have not yet encountered this polynomial
                if(fAlgListIsMember(testPoly, soFar ) == (Bool) 0)
            {
                // We have found a new candidate
                candidatePos = i;
                candidatePoly = testPoly;
                candidateVariable = gen;
                LorR = 0; // Left prolongation.
            }
            }
        }
        // If we are not restricting prolongutions. by degree, proceed if
        // the candidate is valid (if this is the first candidate
        // encountered or LM(x.g) < LM(current candidate))
        else if(( balance ==(Bool)1)|((candidatePos==0))
        {
            // Construct a candidate prolongation
            testPoly = fAlgTimes(fAlgMonom(qOne(), gen ),g );
            // If we have not yet encountered this polynomial
            if(fAlgListIsMember( testPoly, soFar ) == (Bool) 0)
            {
                // We have found a new candidate
                candidatePos = i;
                candidatePoly = testPoly;
                    candidateVariable = gen;
                    LorR = 0; // Lefl. prolongation
            }
        }
        }
    }
    }
// RIGHT PROLONGATIONS
// For each right nonmultiplicative variable
for( }\textrm{j}=1;\textrm{j}<\mathrm{ cutoflL; j++ )
{

```
```

gen = ASCIIMon(j );
mult =fMonTimes(fAlgLeadMonom(g ), gen ); // Construct g.x
if(fMonIsMultiplicative(gen, PR ) ==0)// Not in P (unprocessed)
{
add = 1; // Candidate found
// If Q is not empty
if( Qp )
{
// Make sure that LM(g.x)<LM(f) for all f in (f,v,D) \in Q
Qp2 = fAlgListCopy( Qp ); // Make a copy of Q for processing
while((fAlgListLength(Qp2 ) > 0) \&\& (add == 1)) // For all f in (f,v,D)\in Q
{
// Extract a lead monomial
compare = fAlgLeadMonom( Qp2 -> first );
Qp2 = Qp2 -> rest;
// If LM(g.x) not less than LM(f) ignore this candidate
if(theOrdFun(mult, compare ) == (Bool)0) add = 0;
}
}
if( add == 1) // Candidate found for (g, u, (PL, PR)), w
{
if(candidatePos > 0) // This is not the first candidate tried
// Returns 1 if mult < fAlgLeadMonom(candidatePoly)
balance = theOrdFun( mult, fAlgLeadMonom(candidatePoly ) );
// If we are restricting prolongations by degree
if( degRestrict == 1)
{
// If the degree bound is not exceeded and the candidate is valid
if((fMonLength( LMg ) + < < twod ) \&\& (( balance == (Bool) 1)|( candidatePos == 0)))
{
// Construct a candidate prolongation
testPoly = fAlgTimes(g, fAlgMonom( qOne(), gen ) );
// If we have not yet encountered this polynomial
if(fAlgListIsMember(testPoly, soFar ) == (Bool)0)
{
// We have found a new candidate
candidatePos = i;
candidatePoly = testPoly;
candidateVariable = gen;
LorR = 1; // Right prolongation
}
}
}
// If we are not restricting prolongations by degree, proceed if
// the candidate is valid (if this is the first candidate
// encountered or LM(g.x)<LM(current candidate))
else if(( balance == (Bool)1 )|( candidatePos == 0))

```
```

            {
                    // Construct a candidate prolongation
                    testPoly = fAlgTimes( g, fAlgMonom(qOne(), gen ) );
                    // If we have not yet encountered this polynomial
                    if(fAlgListIsMember( testPoly, soFar ) == (Bool)0)
                    {
                    // We have found a neur candidate
                    candidatePos = i;
                    candidatePoly = testPoly;
                    candidateVariable = gen;
                    LorR = 1; // Right prolongation
                    }
            }
            }
        }
        }
    }
    i++; // Move onto the next polynomial.
    }
if( pl > 3) printf(" . .Element
// If there is a candidate
if(candidatePos > 0)
{
// Construct the cundidute
g = fAlgListNumber( candidatePos,Tp );
u}=\textrm{fMonListNumber(candidatePos, Tm );
P}=\textrm{fMonPairListNumber(candidatePos,Tv );
if( pl>2)
{
if( LorR == 0)
printf("Analysingulefttuprolongation
fAlgToStr(g ), fMonToStr(candidateVariable ) );
else
printf("Analysing\&right \&prolongation
fAlgToStr(g ), fMonToStr(candidateVariable ) );
}
// Adjust T--.- Remove (g,u, P) from T and add ( }g,u,\mathrm{ (cnlarged P))
Tp = fAlgListRemoveNumber( candidatePos,Tp );
Tp = fAlgListPush( g, Tp );
Tm}=\textrm{fMonListRemoveNumber( candidatePos, Tm );
Tm}=\textrm{fMonListPush( u, Tm );
Tv = fMonPairListRemoveNumber( candidatePos, Tv );
if(LorR == 0) // Left prolongation
P.lft = multiplicativeUnion( P.lft, candidateVariable );
else // Right prolongation
P.rt = multiplicativeUnion( P.rt, candidateVariable );
Tv = fMonPairListPush( P.lft, P.rt, Tv );
// Construct the prolongation

```
```

if(LorR == 0 )
gDotx = fAlgTimes(fAlgMonom(qOne(), candidateVariable ), g );
else
gDotx = fAlgTimes(g, fAlgMonom( qOne(), candidateVariable ) );
// If the criterion is false...
// if( NCcriterion( gDotx, u, Tp, Tm, GBasis,vars ) ==0 0)
{
// ...then find the normal form of the prolongation w.r.t. GBasis
soFar = fAlgListPush(gDotx, soFar ); // Inticate we have encountered another polynomial
h = IPolyReduce( gDotx, GBasis, vars ); // Involutively reduce gDotx w.r.t. GBasis
h = findGCD( h ); // Divide through by the GCD
if( pl > 2 ) printf(" . . Reduced_prolongation
nOfProlongations++; // Increment the number of prolongations processed
// Check for trivial ideal
if(fAlgIsOne(h ) == (Bool) 1 ) return fAlgListSingle(fAlgOne() );
if(fAlgIsZero( }\textrm{h})==(\mathrm{ (Bool) 0) // If the prolongation did not reduce to 0
{
// Add h to GBasis and vecalculate multiplicative variables if necessary
if( IType < 3 )
{
pos=1;
if( SType == 1 ) GBasis = fAlgListDegRevLexPushPosition(h, GBasis, \&pos ); // DegRevLex sort
else if(SType == 2) GBasis = fAlgListAppend( GBasis, fAlgListSingle(h ) ); // Just uppend
else GBasis = fAlgListNormalPush( h, GBasis ); // Sort by monomial ordering
vars = OverlapDiv( GBasis ); // Full recalculate
}
else GBasis = fAlgListAppend( GBasis, fAlgListSingle( h ) ); // Just append onto end.
if( pl > 1) printf("Added
else if( pl == 1) printf("Added
LMh = fAlgLeadMonom( h );
if(degRestrict == 1)// If we are restricting protongations by degrec...
{
degTest = fMonLength( LMh );
if( degTest > d ) // ...and if the degree of the new polynomial exceeds the bound...
{
// ...adjust the bound accordingly
d = degTest;
if( pl > 2 ) printf("New
twod = 2*d;
}
}
// if LM(h) == LM(prolongation)
if(fMonEqual(fAlgLeadMonom(gDotx ), LMh ) == (Bool) 1)
{
// Add entry (h., u, \emptyset, \emptyset)) to T

```
```

    Tp = fAlgListPush( h, Tp );
    Tm = fMonListPush( u, Tm );
    Tv = fMonPairListPush(fMonOne(), fMonOne(), Tv );
    }
else // Add entry to T and adjust lists
{
// Add entry to T
Tp = fAlgListPush( h, Tp );
Tm = fMonListPush( LMh, Tm );
Tv = fMonPairListPush(fMonOne(), fMonOne(), Tv );
if( pl > 3 ) printf("Modifying山TU(size山|%u) ...\n", fAlgListLength( Tp ) );
// Set up lists for next operation
Tp2 = fAlgListNul;
Tm2 = fMonListNul;
Tv2 = fMonPairListNul;
// For each (f,v,(DL,DR))\in T
if( pl > 4) printf("Adjusting\Multiplicative_Variables...\n");
while(Tp )
{
f}=\textrm{Tp}->\mathrm{ first; // Extract a polynomial
LMf = fAlgLeadMonom(f);
if( pl>4) printf("Testingu(%s, L%s)\n", fMonToStr( LMh ), fMonToStr( LMf ) );
// If LM(h)<LM(f)
if( theOrdFun( LMh, LMf ) == (Bool) 1 )
{
// Add entry to Q
Qp = fAlgListPush(Tp m first, Qp );
Qm}=\textrm{fMonListPush( Tm }->\mathrm{ first, Qm );
Qv = fMonPairListPush( Tv >> lft, Tv >> rt, Qv );
// Discard f from GBasis
GBasis = fAlgListFXRem( GBasis, f ;

```

```

        else if( pl== 1) printf("Discarded_aцpolynomial_fromムG\cup(%u) ...\n", fAlgListLength(GBasis ) );
    }
    else
    {
        // Keep entry in T
        Tp2 = fAlgListPush( Tp -> first, Tp2 );
        Tm2 = fMonListPush(Tm }->\mathrm{ first, Tm2 );
        Tv2 = fMonPairListPush(Tv -> lft, Tv >> rt, Tv2 );
    }
    // Advance the lists to the next entry
    Tp = Tp >> rest;
    Tm}=\textrm{Tm}->> rest
    Tv = Tv -> rest;
    }
// Set up lists for neat operation

```
```

Tp = fAlgListNul;
Tm = fMonListNul;
Tv =fMonPairListNul;
// Recalculate multiplicative variables
if( IType < 3 ) vars = OverlapDiv( GBasis );
// For each (f.v, (DL, DR))\in T
while(Tp2)
{
// Keep f and v as they are
f = Tp2 -> first;
Tp = fAlgListPush( f, Tp );
Tm}=\textrm{fMonListPush(Tm2 }->>\mathrm{ first, Tm );
DL = Tv2 -> lft;
DR = Tv2 }->\textrm{rt}
// Find intersection of D and NM_I(f, GBasis)
if( IType < 3) // Local division
{
// Find NM_I(f,GBasis)
pos = fAlgListPosition(f, GBasis );
fVars =fMonPairListNumber( pos, vars );
fVarsL = fVars.lft;
fVarsR = fVars.rt;
NML = fMonOne();
NMR = fMonOne();
j = 1;
// Calculate the intersection
while( }\textrm{j}<=\mathrm{ (ULong) nOfGenerators )
{
gen = ASCIIMon(j);
// If gen appears in DL (nonmultiplicatives) but not in fVarsL (multiplicatives)
if (( fMonIsMultiplicative( gen, DL ) == 1)
\&\& (fMonIsMultiplicative(gen, fVarsL ) == 0))
NML =fMonTimes( NML, gen ); // gen appears in the left intersection
// If gen appears in. DR (nonmultiplicatives) but not in fVarsR (multiplicatives)
if ((fMonIsMultiplicative(gen, DR ) == 1)
\&\& (fMonIsMultiplicative(gen, fVarsR ) ==0))
NMR =fMonTimes( NMR, gen );// gen appears in the right intersection
j++; // Get ready to look at the next variable
}
}
else if ( IType >= 3 ) // Global division
{
// Find the multiplicative variables
if( IType == 3 ) LMultVars(fAlgLeadMonom(f), \&cutofL, \&cutoffR );
else if( IType == 4) RMultVars(fAlgLeadMonom(f), \&cutofLL, \&cutoffR );
else EMultVars(fAlgLeadMonom(f), \&cutoflL, \&cutoffR );

```
```

                    NML = fMonOne();
                    NMR = fMonOne();
                    // Calculate the left intersection
                    for( }\textrm{j}=\mathrm{ cutoffR +1; j <= (ULong) nOfGenerators; j++ )
                    {
                            gen = ASCIIMon(j ); // Obtain a nonmultiplicative variable
                                // If it appears in DL it appears in the intersection
                                if(fMonIsMultiplicative( gen, DL ) == 1 )
                        NML = fMonTimes( NML, gen );
                    }
                    // Calculate the right intersection
                    for( j = 1; j < cutoflL; j++ )
                    {
                    gen = ASCIIMon( }\textrm{j});// Obtain a nonmultiplicative variable:
                    // If it appears in DR it appears in the intersection
                        if(fMonIsMultiplicative(gen, DR ) == 1)
                    NMR = fMonTimes( NMR, gen );
                    }
                }
                    /( Add the nonmultiplicative variables to Tv
                    Tv = fMonPairListPush( NML, NMR, Tv );
                // Advunce the lists
                    Tp2 = Tp2 -> rest;
                    Tm2 = Tm2 -> rest;
                    Tv2 = Tv2 -> rest;
                    }
                }
            }
        }
        // else if( pl> 2 ) printf("...Criterion used to discard prolongation...\n");
        }
        else // erit from loop - no suitable prolongations found
        {
            escape = 1;
        }
    }
    }
while( Qp );
if( pl>0) printf("...Involutive
return GBasis;
25.42 * =============
2543 * End of File
2544 * ============

```
\}
2540
2541 /*
2545 */

\section*{B.2.12 involutive.c}
```

/*

* File: involutive.c (Noncommutative Involutive Basis Program)
* Author: Gareth Evans
* Last Modified: 10th August 2005
*/
// Include MSSRC Libraries


# include <fralg.h>

// Include *-functions Libraries

# include "file_functions.h"

# include "list_functions.h"

# include "fralg_functions.h"

# include "arithmetic_functions.h"

# include "ncinv_functions.h"

/*

```

```

* External Variables for ncinv_functions.c

```

```

*/
ULong nOfProlongations; // Stores the number of prolongations calculated
int degRestrict = 0,// Determines whether of not prolongations are restricted by degree
IType = 3,// Stores the involutive division used (1,2 = Left/Right Overlap, 3,4,= Left/Right, 5 = Empty)
EType = 0,// Stores the type of Overlap Division
SType =1,// Determines how the basis is sorted
MType = 1;// Determines method of involutive division
/*

```

```

* External Variables for fralg_functions.c AND ncinv_functions.c

```

```

*/
ULong nRed = 0;// Stores how many reductions have been carried out
int nOfGenerators, // Holds the number of generators
pl=1;// Holds the "Print Level"
/*

```

```

    * Global Variables for ncinv-functions.c
    * ============================================
*/
FMonList gens =fMonListNul; // Stores the generators for the basis
FMonPairList multVars = fMonPairListNul; // Stores multiplicative variables
FAlgList F = fAlgListNul, // Holds the input ba,sis
G = fAlgListNul, // Holds the Groebner Basis
G_Reduced = fAlgListNul, // Holds the Reduced Groebner Basis
IB = fAlgListNul, // Holds the Involutive Basis

```
```

            IMPChecker = fAlgListNul; // Stores a list of polynomials for the IMP
    FMon allVars; // Stores all the variables
int AlgType = 1,// Stores which involutive algorithm to use
order_switch =1;// Stores the monomial ordering u.sed
/*

* Remark: Here are the possible values of order_switch:
* 1: DegRevLex
* 2: DegLex
* 3: Lex
* 9: Wreath Product
*/
/*
* Function Nume: NormalBatch
* 
* Overview: Calculates an Involutive Basis and a
* Reduced Minimal Groebner Basis
* 
* Detail: Given an input basis, this function uses the
* functions in fralg_functions.c and ncinv_functions.c
* to calculate an Involutive Basis and a minimal
* reduced Groebner Basis for the input basis.
* 
* Extemal Variables Used: int pl;
* Global Variables Used: FAlgList F,G, G_Rciuced;
* 

*/
static void
NormalBatch()
{
FAlgList Display = fAlgListNul;
int plSwap = pl;
// Output some initial information to screen
if( pl>0)
{
printf("\nPolynomials
Display = fAlgListCopy(F );
while( Display )
{
// If pl===1, display the polynomial using the original generators
if( pl == 1) printf("%s,\n", postProcess(Display }->\mathrm{ first, gens ) );
// Otherwise, if pl>1, display the polynomial using ASCII generators
else if( pl>1) printf("%s,\n", fAlgToStr( Display —> first ) );
Display = Display }->\mathrm{ rest; // Advance the list.
}
printf("[%u\lrcornerPolynomials]\n", fAlgListLength(F ));
}
// Calculate an. Involutive Basis for F
if( AlgType == 1) G = Gerdt(F );
else G = Seiler( F );

```
```

// Display calculated basis
if( pl>0)
{

```

```

    if( IType < 3) // Local division.
    {
        printf("\nHere)
        IB = fAlgListCopy(G );
        Display = fAlgListCopy(G);
        // We will now calculate the multiplicative variables silently
        pl = 0; // Set silent print level
        if( IType < 3) multVars = OverlapDiv( G );
        pl = plSwap; // Restore original print level
        while( Display )
        {
            // If pl===1, display the polynomial using the original generators
            if( pl == 1 ) printf("%s, (% (% s, %s),\n", postProcess( Display }->\mathrm{ first, gens ),
                    postProcess(fAlgMonom(qOne(), fMonReverse(multVars }->\textrm{lft})),\mathrm{ gens ),
                    postProcess(fAlgMonom(qOne(), fMonReverse( multVars -> rt )), gens ));
            // Otherwise, if pl>1, display the polynomial using ASCII generators
            else if( pl > 1) printf("%s,u(%s,u%s),\n", fAlgToStr(Display }->>\mathrm{ first ),
                    fMonToStr( fMonReverse( multVars }->> lft ))
                    fMonToStr( fMonReverse(multVars -> rt ) ) ;
            Display = Display }->>\mathrm{ rest; // Advance the polynomial list
            multVars = multVars }->>\mathrm{ rest; // Advance the multiplicative variubles list
        }
        printf("[%u_Polynomials]\n", fAlgListLength(G ) );
    }
    else // Global division
    {
    ```

```

        IB = fAlgListCopy(G );
        Display = fAlgListCopy(G);
        while( Display )
        {
        if( IType == 3)// Left Division.
        {
            // If pl===1, display the polynomial using the original generators
            if( pl==1) printf("%s, ப(%s,山1),\n", postProcess( Display }->>\mathrm{ first, gens ), fMonToStr( allVars ) );
            // Otherwise, if pl>1, display the polynomial using ASCII generators
            else if( pl > 1) printf("%s, (all, цnone),\n", fAlgToStr( Display }->\mathrm{ first ));
        }
        else if( IType == 4) // Right Division
        {
            // If pl==1, display the polynomial using the original generators
            if( pl==1) printf("%s, (1, , %s),\n", postProcess(Display }->>\mathrm{ first, gens ), fMonToStr( allVars ) );
            // Otherwise, if pl>1, display the polynomial using ASCII genemators
            else if( pl > 1) printf("%s, (none, цall), \n", fAlgToStr( Display -> first ));
        }
        else if( IType == 5 ) // Empty Division.
    ```
```

            {
                // If pl== 1, display the polynomial using the original generators
                if( pl == 1 ) printf("%s, ц(1, ப1),\n", postProcess( Display -> first, gens ) );
                // Otherwise, if pl>1, display the polynomial using ASCII generators
                else if( pl > 1) printf("%s, (none, பnone),\n", fAlgToStr(Display }->>\mathrm{ first ));
            }
            Display = Display }->>\mathrm{ rest; // Advance the list
        }
        printf("[%u
    }
    }
// Calculate a reduced and minimal Groebner Basis
if( pl > 0) printf("\nComputing}\mp@subsup{\mathrm{ the }}{\llcorner}{
G = minimalGB(G);// Minimise the basis
G_Reduced = reducedGB(G); // Reduce the basis

```

```

// Display some information on screen
if( pl>0)
{
printf("\nHere
Display = fAlgListCopy( G_Reduced );
while(Display )
{
// If pl== 1, display the polynomial using the original generutors
if( pl == 1) printf("%s,\n", postProcess(Display }->\mathrm{ first, gens ) );
// Otherwise, if pl > 1, display the polynomial using ASCII generators
else if( pl > 1) printf("%s,\n", fAlgToStr( Display -> first ) );
Display = Display }->\mathrm{ rest;
}
printf("[%u_Polynomials]\n", fAlgListLength( G_Reduced ));
}

* Function Name: IMPSolver
* Overview: Solves the Ideal Membership Problem for polynomials
* sourced from disk or from user input.
* 
* Detail: Given a polynomial sourced from disk or from user
* input, this function solves the ideal membership problem.
* for that polynomial by reducing the polynomial w.r.t.
* a minimal reduced Groebner Basis (using a specially
* adapted function) and testing to see whether the
* polynomial reduces to zero or not.
* 
* Extemal Variables Used: FAlgList IMPChecker;
* FMonList gens;
* int. pl;
*/
static void

```
\}
192
93 /*
```

IMPSolver()
{
FAlgList polynomials = fAlgListNul;
FAlg polynomial;
int sink;
Short dk =2; // Convention: 1 = disk, 2 = keyboard.
Bool answer;
String inputChar = strNew(), inputStr = strNew(),
polyFileName = strNew(), outputString = strNew();
FILE *polyFile;
// Determine whether the input woll come from disk or from the keyloard
printf("***UIDEAL_MEMBERSHIP_PROBLEM_SOLVERU***\n\n");
printf("Source:\leftharpoonupDisk
sink = scanf( "%s", inputChar );
// If the user hasn't entered 'd' or ' }k\mathrm{ ', ask for another letter
while(( strEqual(inputChar, "d" ) == 0) \& (strEqual( inputChar, "k" ) == 0))
{

```

```

    sink = scanf( "%s", inputChar );
    }
printf("\n");
// If the polynomials are to be obtained from disk
if( strEqual( inputChar, "d") == (Bool) 1)
{
dk}=1;// Set input from dis
printf("Please\cupenter\iotathe
sink = scanf( "%s", polyFileName );
// Rend file from disk
if(( polyFile = fopen(polyFileName, "r" )) == NULL )
{

```

```

        exit( EXIT_FAILURE );
    }
        // Obtain the polynomials from the file
        polynomials = fAlgListFromFile( polyFile);
        polynomials = preProcess( polynomials, gens ); // Change to ASCII order
        sink = fclose( polyFile );
    }
    else // Else obtain the first polynomial from the keyboard
    {
        if( }\textrm{pl}<2)// Require polynomial using original generators
            printf("Please&enter_a\iotapolynomialu(e.g.\iotax*y`2-z)\n");
        else // Require polynomial using ASCII generators
            printf("Please
        printf("(A}\mp@subsup{A}{|}{
        sink = scanf( "%s", inputStr );
        if(( strEqual(inputStr, "" ) == (Bool) 1 )|( strEqual(inputStr, ";" ) == (Bool) 1 ) )
    ```
```

    polynomials =fAlgListNul; // No poly given, terminate program
    else
    {
        // Push the given polynomial onto the list
        polynomials = fAlgListPush( parseStrToFAlg( inputStr ), polynomials );
        if( pl < 2) // Need to convert to ASCII order
        polynomials = preProcess( polynomials, gens );
    }
    }
// For each polynomial in the list (for keyboard entry the list will have I element)
while(polynomials )
{
polynomial = polynomials }->\mathrm{ first; // Extract a polynomial to test
polynomials = polynomials }->\mathrm{ rest; // Advance the list.
// Solve the Ideal Membership Problem for the polynomial
// using the Groebner Basis stored in. IMPChecker
answer = idealMembershipProblem( polynomial, IMPChecker );
// Prepare to report the result correctly
if( pl < 2) outputString = postProcess( polynomial, gens );
else outputString = fAlgToStr( polynomial );
// Return the results
if(answer == (Bool) 0)

```

```

    else
    ```

```

    if( dk == 2) // Obtain another poly from keyboard.
    {
        if( pl < 2) // Require polynomial using original generators
    ```

```

        else // Require polynomial using ASCII generators
            printf("Please
        printf("(A}\mp@subsup{A}{\llcorner}{}\mathrm{ semicolon}\mp@subsup{|}{\bullet}{\primeterminates
        sink = scanf( "%s", inputStr );
        if((strEqual(inputStr, "")==(Bool) 1)|(strEqual(inputStr, ";")==(Bool) 1))
            polynomials = fAlgListNul; // No poly given, terminate program
        else
        {
            // Push the given polynomial onto the list
            polynomials = fAlgListPush( parseStrToFAlg( inputStr ), polynomials );
            if( pl < 2) // Need to convert to ASCII order
            polynomials = preProcess( polynomials, gens );
        }
        }
    }
    }

```
31.5
316 /*
333 main ( argc, argv )
334 int argc;
335. char *argv[]

336 \{
* Function Name: main
*
* FMonList gens;
int order_switch;
*/
int
int argc;
char *argv[];
if \((\operatorname{argc}<2)\)
\{
\} \{
\{
* Overview: A Noncommutative Involutive Basis Program
* Detail: This function deals with the inputs and outputs
* of the program. In particular, the command line arguments are
* processed, the input files are read, and once the Involutive
* Basis has been calculated, it is output to disk together with
* the reduced minimal Groebner Basis.
* External Variables Used: int n.OfGenerators, pl;
* Global Variables Used: FAlgList F;

String filename \(=\operatorname{strNew}(), / /\) Used to create the output file name filename \(2=\operatorname{strNew}() ; / /\) Used to create the involutive output file name
FAlg zeroOrOne; // Used to test for trivial basis elements
FMonList gens_copy \(=\mathrm{fMonListNul} ; / /\) Holds a copy of the generators
ULong k ; // Used as a counter
int \(\mathrm{i}, / /\) Used as a counter
length; // Used to store the length of a command line argument
Short alpha_switch \(=0, / /\) Do we optimise the generator order leaicographically? fractions \(=0, / /\) Do we eliminate fractions from the input basis?
IMP \(=0, / /\) At the end of the algorithm, do we solve the IMP? \(\mathrm{p} ; / /\) Used to navigate through the command line arguments
FILE *grobdata, // Stores the input file *outputdata; // Used to construct the output file
// Process Command Line Arguments

\(\operatorname{printf}\left(" \backslash\right.\) nSee \({ }_{\cup}\) README \(_{\lrcorner}\)for \(_{\sqcup}\) more \(_{\lrcorner}\)information. \(\left.\backslash n \backslash n "\right)\); exit( EXIT_FAILURE);
\(\mathrm{p}=1 ; / / \mathrm{p}\) will step through all the command line arguments
while \((\operatorname{argv}[\mathrm{p}][0]==\) ' - ) \(/ /\) While there is another command line argument.
length \(=(\) int \()\) strlen \((\operatorname{argv}[\mathrm{p}]) ; / /\) Determine length of argument

if( length \(==1) / /\) Just \(a\) "..." was given

 exit( EXIT_FAILURE);
```

}
// We will now deal with the different cllowable parameters
switch(argv[p][1] )
{
case 'a':
alpha_switch = 1;// Optimise the generator order lexicographically
break;
case 'c': // Choose the algorithm used to construct the involutive basis
if( length != 3 )
{
printf("\nInvalid|}\mp@subsup{I}{|}{\prime\prime
printf("\nSee}\mp@subsup{|}{}{README
exit( EXIT_FAILURE);
}
switch( argv[p][2] ) // Choose the algorithm type
{
case '1':
case '2':
AlgType = ((int) argv[p][2] ) - 48;
break;
default:
printf("\nInvalid
printf("\nSee
exit( EXIT_FAILURE );
break;
}
break;
case 'd':
order_switch = 2;// Use the DegLex Monomial Ordering
break;
case 'e': // C'hoose the Overlap Division type
if( length != 3 )
{
printf("\nInvalid
printf("\nSee}|\mathrm{ README\&for」more_information.\n\n");
exit(EXIT_FAILURE );
}
switch(argv[p][2]) // Assign the type
{
case '1':
case '2':
case '3':
case '4':
case '5':
EType = ( (int) argv[p][2] ) - 48;
break;
default:

```

```

                printf("\nSee
                exit(EXIT_FAILURE );
            break;
        }
    ```
```

    break;
    case 'f':
fractions=1;// Eliminate fractions from the input basis
break;
case 'l':
order_switch = 3;// Use the Lexicographic Monomial Ordering
break;
case 'm': // Choose method of involutive division
if( length != 3)
{
printf("\nInvalid
printf("\nSee
exit( EXIT_FAILURE );
}
switch(\operatorname{argv}[p][2]) // Choose the: method
{
case '1' :
case '2':
MType =((int) argv[p][2] ) - 48;
break;
default:
printf("\nInvalid

```

```

            exit( EXIT_FAILURE );
            break;
    }
    break;
    case '0': // Choose how the basis is stored
if( length != 3)
{
printf("\nInvalid
printf("\nSee
exit( EXIT_FAILURE );
}
switch(\operatorname{argv[p][2] ) // Choose the sorting method}<br>mp@code{L}
{
case '1' :
case '2' :
case '3' :
SType =( (int) argv[p][2]) - 48;
break;
default:
printf("\nInvalid
printf("\nSee
exit( EXIT_SUCCESS );
break;
}
break;
case 'p': // Calls the Interactive Ideal Membership Problem
IMP = 1;// Solver after the Groelner Busis has been found.
break;
case 'r': // Use the DegRevLex Monomial Ordering
break; // (we do nothing here - this is default option)

```
case 's': // Choose an involutive division
    if( length \(!=3\) )
    \{


        exit( EXIT_SUCCESS );
    \}
    switch( \(\operatorname{argv}[\mathrm{p}][2]) / /\) Assign the involutive division type
    \{
        case '1':
        case '2' :
        case ' 3 ':
        case '4':
        case '5' :
            IType \(=(\) (int \() \operatorname{argv}[\mathrm{p}][2])-48 ;\)
            break;
        default:

            printf(" \(\backslash n\) See \(_{\lrcorner}\)README \(_{\lrcorner}\)for more \(_{\sqcup}\) information. \(\backslash n \backslash n "\) );
            exit( EXIT_FAILURE);
            break;
    \}
    break;
case 'v': // Choose the amount of information given to screen
    if( length \(!=3\) )
    \{
        printf(" \(\backslash\) nInvalid Input \(_{\lrcorner}-\)incorrect \(_{\lrcorner}\)length \(_{\cup}\) on \(_{\lrcorner}\)verbose \(e_{\lrcorner}\)parameter.");
        printf(" \(\backslash\) nSee \({ }_{\cup}\) README \(_{\cup}\) for \(_{\llcorner }\)more \(_{\cup}\) information. \(\left.\backslash \mathrm{n} \backslash \mathrm{n} "\right)\);
        exit( EXIT_FAILURE );
    \}
    switch \((\operatorname{argv}[\mathrm{p}][2])\)
    \{
        case '0':
        case '1':
        case '2' :
        case '3':
        case '4' :
        case '5' :
        case '6' :
        case '7':
        case ' 8 ' :
        case ' 9 ':
            \(\mathrm{pl}=(\) (int \() \operatorname{argv}[\mathrm{p}][2])-48 ;\)
            break;
        default:


            exit( EXIT_FAILURE);
            break;
    \}
    break;
case 'w':
    order_switch \(=9 ; / /\) Use the Wreath Product Monomial Ordering
```

        break;
    case ' }\textrm{x}\mathrm{ ':
        degRestrict = 1;// Turns on restriction of prolongations by degree
        break;
    default:
        printf("\nInvalid_要arameter
        printf("\nSee}\mp@subsup{|}{\cup}{README}\mp@subsup{\mathrm{ forrumore}}{\lrcorner}{\prime}\mathrm{ information.\n\n");
        exit(EXIT_FAILURE);
        break;
    }
    p++; // Get ready to look at the next parameter
    }
p}=\textrm{p}-1;//p\mathrm{ now holds the numbcr of parameters processed
// Test overloading of suritches
if(filenameLength(\operatorname{argv}[1+p])>59)
{
printf("\nError:|The
printf("exceed
exit( EXIT_SUCCESS );
}
if(( EType > 0)\&\&( IType >= 3))
{
printf("\nError: : The

```

```

    exit( EXIT_SUCCESS );
    }
if(( EType == 2)\&\& ( MType == 1 ))
{
printf("\n***挰Warning: \The

```

```

}
// Open file specified on the command line
if((grobdata }=\mathrm{ fopen ( argv[1+p], "r")) == NULL )
{
printf("Error_
exit( EXIT_FAILURE );
}
/*

* The first line of the input file should contain the
    * generators in the format a;b;c;···
    * (representing a>b>c>···). We will now read the
    * generators from file and calculate the number of
    * generators obtained.
*/
gens =fMonListFromFile(grobdata );
/*

```
```

    * As the rest of the program assumes a generator order
    * a<b<c<\ldots (for ASCII comparison), we now reverse
    * the list of generators.
    */
gens = fMonListFXRev( gens );
k = fMonListLength( gens );
if( k >= (ULong) INT_MAX ) // Check limit
{
printf("Error:uINT_MAX_Exceeded_(in_main)\n");
exit(EXIT_FAILURE);
}
else nOfGenerators = (int) k;
// Check generator bound
if( nOfGenerators > 17576 )
{

```

```

    exit( EXIT_FAILURE);
    }
if( IType >= 3 ) // Global division
{
// Create a monomial storing all the generators in order
gens_copy = fMonListCopy( gens );
allVars = fMonOne();
while(gens_copy )
{
allVars =fMonTimes( allVars, gens_copy -> first );
gens_copy = gens_copy -> rest;
}
allVars = fMonReverse( allVars );
}
// Welcome
if( pl > 0)
{

```

```

    else printf("\n***UNONCOMMUTATIVE江NVOLUTIVE &BASIS\PROGRAMU(GLOBAL
    }
// We will now choose the monomial ordering to be used.
switch(order_switch )
{
case 1:
theOrdFun = fMonDegRevLex;
if( pl > 0 ) printf("\nUsing山the
break;
case 2:
theOrdFun = fMonTLex;
if( pl>0 ) print(" "\nUsing山the
break;
case 3:

```
```

    theOrdFun = fMonLex;
    if( pl >0 ) printf("\nUsing}\mp@subsup{|}{}{\mathrm{ the }
    break;
    case 9:
        theOrdFun = fMonWreathProd;
        if( pl > 0) printf("\nUsing
        break;
    default:
        break;
    }
// Output the generator order to screen...
if( pl > 0)
{
fMonListDisplayOrder( gens );
printf("\n");
}
// Now read the polynomials from disk
F = fAlgListFromFile( grobdata );
// If necessary, optimise the generator order
if( alpha_switch == 1 ) gens = alphabetOptimise( gens, F );
/*
* Now substitute original generators for ASCII generators in all
* busis polynomials. This is done because all the monomial
* orderings use ASCII string comparisons for efficiency.
* For example, if the original monomial ordering is }x>y>
* and a polynomial }x*y-2*z\mathrm{ is in the basis, then the polynomial
* we get after substituting for the ASCII order (AAC>AAB>AAA) is
* AAC*AAB-2*AAA.
*/
G = preProcess( F, gens ); // Note: placed in G for processing
F = fAlgListNul;
// If we are asked to remove all fractions from the input basis, do so now.
if(fractions == 1) G = fAlgListRemoveFractions(G );
// Test the list for special cases (trivial ideals)
while(G)
{
zeroOrOne = G -> first; // Extract a polynomial
if(fAlgIsZero(zeroOrOne )== (Bool) 0) // If the polynomial is not equal to 0...
F}=\textrm{fAlgListPush(zeroOrOne, F ); // ...add to the input list.
// Now divide by the leading coefficient to get a unit coefficient
zeroOrOne = fAlgScaDiv( zeroOrOne, fAlgLeadCoef(zeroOrOne ) );
if(fAlgIsOne(zeroOrOne)}==(\mathrm{ Bool) 1) // If the polynomial is equal to 1...
{
// ... we huve a trivial ideal
F = fAlgListSingle(fAlgOne() );
break;
}

```
```

    G = G -> rest; // Advance the list
    }
F=fAlgListFXRev(F); // Reverse the list (it was constructed in reverse)
G = fAlgListNul; // Reset for later use
// Calculate the number of polynomials in the input basis
k= fAlgListLength(F);
if( k >= (ULong) INT_MAX ) // Check limit
{
printf("Error:|INT_MAX EXxceeded
exit( EXIT_FAILURE);
}
// Calculate an Involutive Basis for F followed by a
// reduced and minimal Groebner Basis for F
NormalBatch();
// Write Reduced Groelner Basis to Disk
if( pl > 0) printf("\nWriting_Reduced_Groebner_Basis\iotato\llcornerDisk...ь");
// Choose the correct suffix for the filename (argv[1+p] is the original filename)
switch( order_switch )
{
case 1:
filename = appendDotDegRevLex( argv[1+p] );
break;
case 2:
filename = appendDotDegLex( argv[1+p] );
break;
case 3:
filename = appendDotLex(argv[1+p] );
break;
case 9:
filename = appendDotWP( argv[1+p] );
break;
default:
printf("\nERROR_DURING_SUFFIX }\mp@subsup{|}{U}{SELECTION\n\n");
exit( EXIT_FAILURE );
break;
}
filename2 = strConcat(filename, " .inv" );
// Now open the output file
if(( outputdata = fopen (filename, "w" )) == NULL )
{
printf("%s\n", "Error_०peningu/\iotacreating\iotathe_(first)_outputufile.");
exit(EXIT_FAILURE );
}
// Write the (reversed) generator order to disk
fMonListToFile( outputdata, fMonListRev( gens ) );

```
```

// Write Polynomials to disk
G = fAlgListNul;
// If we are required to solve the Ideal Membership Problem,
// let us make a copy of the output basis now
if( IMP == 1 ) IMPChecker = fAlgListCopy( G_Reduced );
// We will now convert all polynomials in the basis
// from. ASCTI order back to the user's order, writing
// the converted polynomials to file as we go.
while( G_Reduced )
{
fprintf( outputdata, "%s;\n", postProcessParse( G_Reduced -> first, gens ));
G_Reduced = G_Reduced }->\mathrm{ rest;
}
// Close off the output file
i}=\mathrm{ fclose( outputdata );
if( pl > 0) printf("Done.\nWriting山Involutive
// Now write the Involutive Basis to disk
if(( outputdata = fopen (filename2, "w" ) ) == NULL )
{

```

```

    exit( EXIT_FAILURE );
    }
// Write the (reversed) generator order to disk
fMonListToFile( outputdata, fMonListRev( gens ) );
// If we are using a local division we need to find the multiplicative variables now
if( IType < 3 ) multVars = OverlapDiv( IB );
while( IB )
{
fprintf( outputdata, "%s;u", postProcessParse( IB -> first, gens ));
if( IType < 3) // Overlap--mased Division
{
fprintf(outputdata, "(%s, s%s);\n",
postProcess(fAlgMonom(qOne(), fMonReverse(multVars -> lft )), gens ),
postProcess(fAlgMonom(qOne(), fMonReverse(multVars -> rt ) ), gens ));
}
else if( IType == 3)// Left Division
{
fprintf(outputdata, "(%s,,1);\n", fMonToStr(allVars ));
}
else if( IType == 4)// Right Division
{
fprintf(outputdata, "(1, L%s); \n", fMonToStr( allVars ) );
}
else if( IType == 5 ) // Empty Division
{

```
```

APPENDIX B. SOURCE CODE

```
```

    fprintf(outputdata, "(1, н1);\n" );
    ```
    fprintf(outputdata, "(1, н1);\n" );
    }
    }
    IB}=\textrm{IB}->> rest; // Advance the list of rule
    IB}=\textrm{IB}->> rest; // Advance the list of rule
    // If need be, advance the multiplicative variables list
    // If need be, advance the multiplicative variables list
    if( IType < 3 ) multVars = multVars }->\mathrm{ rest;
    if( IType < 3 ) multVars = multVars }->\mathrm{ rest;
}
}
// Close off the output file
// Close off the output file
i = fclose( outputdata );
i = fclose( outputdata );
if( pl > 0) printf("Done. \n\n");
if( pl > 0) printf("Done. \n\n");
// If the Ideal Membership Problem Solver is required, run it now.
// If the Ideal Membership Problem Solver is required, run it now.
if( IMP == 1 ) IMPSolver();
if( IMP == 1 ) IMPSolver();
return EXIT_SUCCESS; // Exit successfully
return EXIT_SUCCESS; // Exit successfully
1}
813 # include "file_functions.c"
813 # include "file_functions.c"
814 # include "list_functions.c"
814 # include "list_functions.c"
815 # include "fralg_functions.c"
815 # include "fralg_functions.c"
# include "arithmetic_functions.c"
# include "arithmetic_functions.c"
# include "ncinv_functions.c"
# include "ncinv_functions.c"
819 // End of File
```

819 // End of File

```
812
818

\section*{Appendix C}

\section*{Program Output}

In this Appendix, we provide sample sessions showing how the program given in Appendix B can be used to compute noncommutative Involutive Bases with respect to different involutive divisions and monomial orderings.

\section*{C. 1 Sample Sessions}

\section*{C.1.1 Session 1: Locally Involutive Bases}

Task: If \(F:=\left\{x^{2} y^{2}-2 x y^{2}+x^{2}, x^{2} y-2 x y\right\}\) generates an ideal \(J\) over the polynomial ring \(\mathbb{Q}\langle x, y\rangle\), compute a Locally Involutive Basis for \(F\) with respect to the strong left overlap division \(\mathcal{S}\); thick divisors; and the DegLex monomial ordering.

Origin of Example: Example 5.7.1.
Input File:
```

x; y;
x^2*y^2-2*x*y^2 + x^2;
x^2*y - 2*x*y;

```

Plan: Apply the program given in Appendix B to the above file, using the '-c2' option to select Algorithm 12; the '-d' option to select the DegLex monomial ordering; the '-m2' option to select thick divisors; and the '-e2' and '-s1' options to select the strong left overlap division.

\section*{Program Output:}
ma6:mssrc-aux/thesis \(>\) time involutive \(-\mathrm{c} 2-\mathrm{d}-\mathrm{e} 2-\mathrm{m} 2-\mathrm{s} 1\) thesis1.in
*** NONCOMMUTATIVE INVOLUTIVE BASIS PROGRAM (LOCAL DIVISION) ***
Using the DegLex Ordering with \(x>y\)

Polynomials in the input basis:
\(x^{\wedge} 2 y^{\wedge} 2-2 x y^{\wedge} 2+x^{\wedge} 2\),
\(x^{\wedge} 2 y-2 x y\),
[2 Polynomials]

Computing an Involutive Basis...
Added Polynomial \#3 to Basis...
Added Polynomial \#4 to Basis...
Autoreduction reduced the basis to size \(3 . .\).
Added Polynomial \#4 to Basis...
Autoreduction reduced the basis to size \(3 . .\).
Added Polynomial \#4 to Basis...
Added Polynomial \#5 to Basis...
...Involutive Basis Computed.
Here is the Involutive Basis
((Left, Right) Multiplicative Variables in Brackets):
\(\mathrm{x} \mathrm{y}^{\wedge} 2 \mathrm{x},(\mathrm{xy}, 1)\),
\(\mathrm{xy}^{\wedge} 2,(\mathrm{xy} y)\),
xy x, (xy, 1 ),
\(\mathrm{xy},(\mathrm{xy}, 1)\),
\(x^{\wedge} 2,(x y, 1)\),
[5 Polynomials]
Computing the Reduced Groebner Basis...
...Reduced Groebner Basis Computed.
Here is the Reduced Groebner Basis:
xy ,
\(\mathrm{x}^{\wedge} 2\),
[2 Polynomials]

Writing Reduced Groebner Basis to Disk... Done.
Writing Involutive Basis to Disk... Done.
\(0.000 \mathrm{u} 0.007 \mathrm{~s} 0: 00.150 .0 \% 0+0 \mathrm{k} 0+2\) io \(16 \mathrm{pf}+0 \mathrm{w}\)
ma6:mssrc-aux/thesis>

Output File:
```

x; y;
x*y^2*x; (x y, 1);
x*y^2; (x y, y);
x*y*x; (x y, 1);
x*y; (x y, 1);

```
```

x^2; (x y, 1);

```

\section*{C.1.2 Session 2: Involutive Complete Rewrite Systems}

Task: If \(F:=\left\{x^{3}-1, y^{2}-1,(x y)^{2}-1, X x-1, x X-1, Y y-1, y Y-1\right\}\) generates an ideal \(J\) over the polynomial ring \(\mathbb{Q}\langle Y, X, y, x\rangle\), compute an Involutive Basis for \(F\) with respect to the left division \(\triangleleft\) and the DegLex monomial ordering.

Origin of Example: Example 5.7.3 ( \(F\) corresponds to a monoid rewrite system for the group \(S_{3}\); we want to compute an involutive complete rewrite system for \(S_{3}\) ).

\section*{Input File:}
```

Y; X; y; x;
x^3-1;
y^2-1;
(x*y)^2-1;
X*x - 1;
x*X - 1;
Y*y - 1;
y*Y - 1;

```

Plan: Apply the program given in Appendix B to the above file, using the '-c2' option to select Algorithm 12 and the '-d' option to select the DegLex monomial ordering (the left division is selected by default).

Program Output:
```

ma6:mssrc-aux/thesis> time involutive -c2 -d thesis2.in
*** NONCOMMUTATIVE INVOLUTIVE BASIS PROGRAM (GLOBAL DIVISION) ***
Using the DegLex Ordering with Y > X > y > x
Polynomials in the input basis:
x^3-1,
y^2-1,
x y x y - 1,
X x - 1,
x X - 1,
Yy - 1,
y Y - 1,
[7 Polynomials]
Computing an Involutive Basis...
Added Polynomial \#8 to Basis...
Added Polynomial \#9 to Basis...

```

\footnotetext{
Added Polynomial \#10 to Basis...
Added Polynomial \#11 to Basis...
Added Polynomial \#12 to Basis...
Added Polynomial \#13 to Basis...
Autoreduction reduced the basis to size \(11 \ldots\)
Added Polynomial \#12 to Basis...
Added Polynomial \#13 to Basis...
Added Polynomial \#14 to Basis...
Added Polynomial \#15 to Basis...
Added Polynomial \#16 to Basis...
Added Polynomial \#17 to Basis...
Added Polynomial \#18 to Basis...
Added Polynomial \#19 to Basis...
Added Polynomial \#20 to Basis...
Added Polynomial \#21 to Basis...
Added Polynomial \#22 to Basis...
Added Polynomial \#23 to Basis...
Autoreduction reduced the basis to size \(19 \ldots\)
Added Polynomial \#20 to Basis...
Autoreduction reduced the basis to size 19...
Added Polynomial \#20 to Basis...
Added Polynomial \#21 to Basis...
Added Polynomial \#22 to Basis...
Added Polynomial \#23 to Basis...
Added Polynomial \#24 to Basis...
Added Polynomial \#25 to Basis...
Added Polynomial \#26 to Basis...
Added Polynomial \#27 to Basis...
Added Polynomial \#28 to Basis...
Added Polynomial \#29 to Basis...
Added Polynomial \#30 to Basis...
Added Polynomial \#31 to Basis...
Added Polynomial \#32 to Basis...
Added Polynomial \#33 to Basis...
Added Polynomial \#34 to Basis...
Added Polynomial \#35 to Basis...
Added Polynomial \#36 to Basis...
Added Polynomial \#37 to Basis...
Added Polynomial \#38 to Basis...
Added Polynomial \#39 to Basis...
Added Polynomial \#40 to Basis...
Autoreduction reduced the basis to size \(29 \ldots\) Added Polynomial \#30 to Basis...
Autoreduction reduced the basis to size 19...
...Involutive Basis Computed.
Here is the Involutive Basis
((Left, Right) Multiplicative Variables in Brackets):
\(\mathrm{y}^{\wedge} 2-1\), (Y X y x, 1 ),
X x -1 , (Y X y x, 1 ),
\(\mathrm{x} \mathrm{X}-1\), ( Y X y x, 1 ),
Yy-1, (Y X y x, 1),
\(\mathrm{y}^{\wedge} 2 \mathrm{x}-\mathrm{x}\), (Y X y x, 1 ),
}
```

Y - y, (Y X y x, 1),
Y x - y x, (Y X y x, 1),
X x y - y, (Y X y x, 1),
Y y x - x, (Y X y x, 1),
x^2 - X, (Y X y x, 1),
X^2 - x, (Y X y x, 1),
x y x - y, (Y X y x, 1),
X y - y x, (Y X y x, 1),
X y x - x y, (Y X y x, 1),
x^2 y - y x, (Y X y x, 1),
y X - x y, (Y X y x, 1),
y x y - X, (Y X y x, 1),
Y x y - X, (Y X y x, 1),
Y X - x y, (Y X y x, 1),
[19 Polynomials]

```

Computing the Reduced Groebner Basis...
...Reduced Groebner Basis Computed.

Here is the Reduced Groebner Basis:
\(y^{\wedge} 2-1\),
\(\mathrm{X} x-1\),
\(\mathrm{x} X-1\),
\(\mathrm{Y}-\mathrm{y}\),
\(x^{\wedge} 2-X\),
\(\mathrm{X}^{\wedge} 2-\mathrm{x}\),
\(x y x-y\),
\(X \mathrm{y}-\mathrm{y} \mathrm{x}\),
y \(X-x y\),
y x y - X,
[10 Polynomials]

Writing Reduced Groebner Basis to Disk... Done.
Writing Involutive Basis to Disk... Done.
\(0.105 \mathrm{u} 0.000 \mathrm{~s} 0: 00.1662 .5 \% 197+727 \mathrm{k} 0+2\) io \(0 \mathrm{pf}+0 \mathrm{w}\)
ma6:mssrc-aux/thesis>

\section*{Output File:}
```

Y; X; y; x;
y^2 - 1; (Y X y x, 1);
X*X - 1; (Y X y x, 1);
x*X - 1; (Y X y x, 1);
Y*y - 1; (Y X y x, 1);
y^2*x - x; (Y X y x, 1);
Y - y; (Y X y x, 1);
Y*x - y*x; (Y X y x, 1);
X*x*y - y; (Y X y x, 1);
Y*y*x - x; (Y X y x, 1);
x^2 - X; (Y X y x, 1);
X^2 - x; (Y X y x, 1);
x*y*x - y; (Y X y x, 1);

```
```

X*y - y*x; (Y X y x, 1);
X*y*x - x*y; (Y X y x, 1);
x^2*y - y*x; (Y X y x, 1);
y*X - x*y; (Y X y x, 1);
y*x*y - X; (Y X y x, 1);
Y*x*y - X; (Y X y x, 1);
Y*X - x*y; (Y X y x, 1);

```

\section*{C.1.3 Session 3: Noncommutative Involutive Walks}

Task: If \(G^{\prime}:=\left\{y^{2}+2 x y, y^{2}+x^{2}, 5 y^{3}, 5 x y^{2}, y^{2}+2 y x\right\}\) generates an ideal \(J\) over the polynomial ring \(\mathbb{Q}\langle x, y\rangle\), compute an Involutive Basis for \(G^{\prime}\) with respect to the left division \(\triangleleft\) and the DegRevLex monomial ordering.

Origin of Example: Example 6.2 .20 ( \(G^{\prime}\) corresponds to a set of initials in the noncommutative Involutive Walk algorithm; we want to compute an Involutive Basis \(H^{\prime}\) for \(\left.G^{\prime}\right)\).

Input File:
```

x; y;
y^2 + 2*x*y;
y^2+ x^2;
5*y^3;
5*x*y 2;
y^2+2*y*x;

```

Plan: Apply the program given in Appendix B to the above file, using the '-c2' option to select Algorithm 12 (the DegRevLex monomial ordering and the left division are selected by default).

\section*{Program Output:}
```

ma6:mssrc-aux/thesis> time involutive -c2 thesis3.in
*** NONCOMMUTATIVE INVOLUTIVE BASIS PROGRAM (GLOBAL DIVISION) ****
Using the DegRevLex Ordering with x > y
Polynomials in the input basis:
y^2+2xy,
y^2+ (*^2,
5 y^3,
5 x y^2,
y^2}+2yx
[5 Polynomials]

```

Computing an Involutive Basis...
...Involutive Basis Computed.
Here is the Involutive Basis
((Left, Right) Multiplicative Variables in Brackets):
\(2 \mathrm{yx}-\mathrm{x}^{\wedge} 2,(\mathrm{xy}, 1)\),
\(\mathrm{y} \mathrm{x}^{\wedge} 2,(\mathrm{xy}, 1)\),
\(x^{\wedge} 3,(x y, 1)\),
\(2 \mathrm{xy}-\mathrm{x}^{\wedge} 2,(\mathrm{xy}, 1)\),
\(y^{\wedge} 2+x^{\wedge} 2,(x y, 1)\),
[5 Polynomials]

Computing the Reduced Groebner Basis...
...Reduced Groebner Basis Computed.

Here is the Reduced Groebner Basis:
\(2 \mathrm{y} x-\mathrm{x}^{\wedge} 2\),
\(\mathrm{x}^{\wedge} 3\),
\(2 \mathrm{xy}-\mathrm{x}^{\wedge} 2\),
\(y^{\wedge} 2+x^{\wedge} 2\),
[4 Polynomials]

Writing Reduced Groebner Basis to Disk... Done.
Writing Involutive Basis to Disk... Done.
0.005u 0.000s 0:00.07 0.0\% 0+0k 0+2io 0pf +0 w
ma6:mssrc-aux/thesis>

More Verbose Program Output: (we select the '-v3' option to obtain more information about the autoreduction that occurs at the start of the algorithm).
```

ma6:mssrc-aux/thesis> time involutive -c2 -v3 thesis3.in
*** NONCOMMUTATIVE INVOLUTIVE BASIS PROGRAM (GLOBAL DIVISION) ***
Using the DegRevLex Ordering with }x(AAB)>y(AAA
Polynomials in the input basis:
AAA^2 + 2 AAB AAA,
AAA^2 + AAB^2,
5 AAA^3,
5 AAB AAA^2,
AAA^2 + 2 AAA AAB,
[5 Polynomials]
Computing an Involutive Basis...
Autoreducing...
Looking at element p = AAA^2 + 2 AAA AAB of basis
Reduced p to AAB AAA - AAA AAB
Looking at element p = 5 AAB AAA^2 of basis
Reduced p to AAB AAA AAB

```
```

Looking at element p = AAB AAA - AAA AAB of basis
Reduced p to AAB AAA - AAA AAB
Looking at element p = 5 AAA^ 3 of basis
Reduced p to AAA^2 AAB
Looking at element p = AAB AAA AAB of basis
Reduced p to AAB AAA AAB
Looking at element p = AAB AAA - AAA AAB of basis
Reduced p to AAB AAA - AAA AAB
Looking at element p = AAA^2 + AAB^2 of basis
Reduced p to 2 AAA AAB - AAB^2
Looking at element p = AAA^2 AAB of basis
Reduced p to AAA AAB^2
Looking at element p = 2 AAA AAB - AAB^2 of basis
Reduced p to 2 AAA AAB - AAB^2
Looking at element p = AAB AAA AAB of basis
Reduced p to AAB^3
Looking at element p = AAA AAB^2 of basis
Reduced p to AAA AAB^2
Looking at element p = 2 AAA AAB - AAB^2 of basis
Reduced p to 2 AAA AAB - AAB^2
Looking at element p = AAB AAA - AAA AAB of basis
Reduced p to 2 AAB AAA - AAB^2
Looking at element p = AAB^3 of basis
Reduced p to AAB^3
Looking at element p = AAA AAB^2 of basis
Reduced p to AAA AAB^2
Looking at element p = 2 AAA AAB - AAB^2 of basis
Reduced p to 2 AAA AAB - AAB^2
Looking at element p = AAA^2 +2 AAB AAA of basis
Reduced p to AAA^2 + AAB^2
Looking at element p = 2 AAB AAA - AAB^2 of basis
Reduced p to 2 AAB AAA - AAB^2
Looking at element p = AAB^3 of basis
Reduced p to AAB^3
Looking at element p = AAA AAB^2 of basis
Reduced p to AAA AAB^2
Looking at element p = 2 AAA AAB - AAB^2 of basis
Reduced p to 2 AAA AAB - AAB^2
Analysing AAA AAB...
Adding Right Prolongation by variable \#0 to S...
Adding Right Prolongation by variable \#1 to S...
Analysing AAA AAB^2...
Adding Right Prolongation by variable \#0 to S...
Adding Right Prolongation by variable \#1 to S...
Analysing AAB^3...
Adding Right Prolongation by variable \#0 to S...
Adding Right Prolongation by variable \#1 to S...
Analysing AAB AAA...
Adding Right Prolongation by variable \#0 to S...
Adding Right Prolongation by variable \#1 to S...
Analysing AAA^2...
Adding Right Prolongation by variable \#0 to S...
Adding Right Prolongation by variable \#1 to S...

```
```

...Involutive Basis Computed.
Number of Prolongations Considered =0
Here is the Involutive Basis
((Left, Right) Multiplicative Variables in Brackets):
2 AAA AAB - AAB^2, (all, none),
AAA AAB^2, (all, none),
AAB^3, (all, none),
2 AAB AAA - AAB^2, (all, none),
AAA^2 + AAB^2, (all, none),
[5 Polynomials]
Computing the Reduced Groebner Basis...
Looking at element p = 2 AAA AAB - AAB^2 of basis
Reduced p to 2 AAA AAB - AAB^2
Looking at element p = AAB^3 of basis
Reduced p to AAB^3
Looking at element p = 2 AAB AAA - AAB^2 of basis
Reduced p to 2 AAB AAA - AAB^2
Looking at element p = AAA^2 + AAB^2 of basis
Reduced p to AAA^2 + AAB^2
Number of Reductions Carried out = 34
...Reduced Groebner Basis Computed.
Here is the Reduced Groebner Basis:
2 AAA AAB - AAB^2,
AAB^3,
2 AAB AAA - AAB^2,
AAA^2 + AAB^2,
[4 Polynomials]
Writing Reduced Groebner Basis to Disk... Done.
Writing Involutive Basis to Disk... Done.
0.000u 0.005s 0:00.04 0.0% 0+0k 0+2io 0pf+0w
ma6:mssrc-aux/thesis>

```

Output File:
```

x; y;
2*y*x - x^2; (x y , 1);
y*x^2; (x y, 1);
x^3; (x y , 1);
2*x*y - x^2; (x y 1);
y^2 + x^2; (x y, 1);

```

\section*{C.1.4 Session 4: Ideal Membership}

Task: If \(F:=\left\{x+y+z-3, x^{2}+y^{2}+z^{2}-9, x^{3}+y^{3}+z^{3}-24\right\}\) generates an ideal \(J\) over the polynomial ring \(\mathbb{Q}\langle x, y, z\rangle\), are the polynomials \(x+y+z-3 ; x+y+z-2\); \(x z^{2}+y z^{2}-1 ; z y x+1\) and \(x^{10}\) members of \(J\) ?

Input File:
```

x; y; z;
x+y+z-3;
x^2+ (y^2 + (z^2-9;
x^3+ y^3+ z^ 3 - 24;

```

Plan: To solve the ideal membership problem for the five given polynomials, we first need to obtain a Gröbner or Involutive Basis for \(F\). We shall do this by applying the program given in Appendix B to compute an Involutive Basis for \(F\) with respect to the DegLex monomial ordering and the right division \(\triangleright\) (this requires the '-d' and '-s4' options respectively). Once the Involutive Basis has been computed (which then allows the program to compute the unique reduced Gröbner Basis \(G\) for \(F\) ), we can start an ideal membership problem solver (courtesy of the '-p' option) which allows us to type in a polynomial \(p\) and find out whether or not \(p\) is a member of \(J\) (the program reduces \(p\) with respect to \(G\), testing to see whether or not a zero remainder is obtained).

\section*{Program Output:}
```

ma6:mssrc-aux/thesis> involutive -c2 -d -p -s4 thesis4.in
*** NONCOMMUTATIVE INVOLUTIVE BASIS PROGRAM (GLOBAL DIVISION) ***
Using the DegLex Ordering with x>y>z
Polynomials in the input basis:
x+y+z-3,
x^2 + y^2 + z^2 - - ,
x^3+ y^3+ (*^3-24,
[3 Polynomials]
Computing an Involutive Basis...
Added Polynomial \#4 to Basis...
Added Polynomial \#5 to Basis...
Added Polynomial \#6 to Basis...
Added Polynomial \#7 to Basis...
Added Polynomial \#8 to Basis...
Added Polynomial \#9 to Basis...
Added Polynomial \#10 to Basis...
Added Polynomial \#11 to Basis...

```
```

Added Polynomial \#12 to Basis...
Added Polynomial \#13 to Basis...
Added Polynomial \#14 to Basis...
Added Polynomial \#15 to Basis...
Added Polynomial \#16 to Basis...
Added Polynomial \#17 to Basis...
Added Polynomial \#18 to Basis...
Added Polynomial \#19 to Basis...
Added Polynomial \#20 to Basis...
Added Polynomial \#21 to Basis...
Autoreduction reduced the basis to size 13...
...Involutive Basis Computed.
Here is the Involutive Basis
((Left, Right) Multiplicative Variables in Brackets):
$\mathrm{x}+\mathrm{y}+\mathrm{z}-3,(1, \mathrm{xyz})$,
$z x+z y+z^{\wedge} 2-3 z,(1, x y z)$,
$\mathrm{y} z-\mathrm{zy},(1, \mathrm{xyz})$,
$z^{\wedge} 3-3 z^{\wedge} 2+1,(1, x y z)$,
$z^{\wedge} 2 y^{\wedge} 2-y-z,(1, x y z)$,
$z^{\wedge} 2 y x+z,(1, x y z)$,
$z^{\wedge} 2 y z-3 z^{\wedge} 2 y+y,(1, x y z)$,
zyz-z^2y, (1, xyz),
$\mathrm{zy} x+1,(1, \mathrm{xyz})$,
z y^2 $2 z^{\wedge} 2 y-3 z y-1,(1, x y z)$,
$z^{\wedge} 2 x+z^{\wedge} 2 y-1,(1, x y z)$,
$y x-z^{\wedge} 2+3 z,(1, x y z)$,
$y^{\wedge} 2+z y+z^{\wedge} 2-3 y-3 z,(1, x y z)$,
[13 Polynomials]
Computing the Reduced Groebner Basis...
...Reduced Groebner Basis Computed.
Here is the Reduced Groebner Basis:
$x+y+z-3$,
$\mathrm{y} z-\mathrm{zy}$,
$z^{\wedge} 3-3 z^{\wedge} 2+1$,
$y^{\wedge} 2+z y+z^{\wedge} 2-3 y-3 z$,
[4 Polynomials]
Writing Reduced Groebner Basis to Disk... Done.
Writing Involutive Basis to Disk... Done.

```
*** IDEAL MEMBERSHIP PROBLEM SOLVER ***
Source: Disk (d) or Keyboard (k)? ...k
Please enter a polynomial (e.g. \(\mathrm{x} * \mathrm{y}{ }^{\wedge} 2-\mathrm{z}\) )
(A semicolon terminates the program) \(\ldots x+y+z-3\)
Polynomial \(x+y+z-3\) IS a member of the ideal.
Please enter a polynomial (e.g. \(x * y{ }^{\wedge} 2-z\) )
(A semicolon terminates the program) \(\ldots x+y+z-2\)
Polynomial \(y+2 \mathrm{z}-2\) is NOT a member of the ideal.

Please enter a polynomial (e.g. \(x * y{ }^{\wedge} 2-z\) )
(A semicolon terminates the program) \(\ldots x * z^{\wedge} 2+y * z^{\wedge} 2-1\)
Polynomial \(x z^{\wedge} 2+y z^{\wedge} 2-1\) IS a member of the ideal.
Please enter a polynomial (e.g. \(x * y{ }^{\wedge} 2-z\) )
(A semicolon terminates the program) \(\ldots z * y * x+1\)
Polynomial zyx+1 IS a member of the ideal.
Please enter a polynomial (e.g. \(x * y^{\wedge} 2-z\) )
(A semicolon terminates the program) \(\ldots x^{\wedge} 10\)
Polynomial \(x^{\wedge} 10\) is NOT a member of the ideal.
Please enter a polynomial (e.g. \(x * y^{\wedge} 2-z\) )
(A semicolon terminates the program)...;
ma6:mssrc-aux/thesis>

\section*{Output File:}
```

x; y; z;
x+y+z-3;(1,x y z);
z*x + z*y + z^2 - 3*z; (1,x y z);
y*z - z*y; (1, x y z);
z^3-3*z^2 + 1; (1, x y z);
\mp@subsup{z}{}{\wedge}2*y^2 - y - z; (1, x y z);
\mp@subsup{z}{}{\wedge}2*y*x + z; (1, x y z);
z^}2*y*z - 3*z^^2*y + y; (1,x y z)
z*y*z - z^ 2*y; (1, x y z);
z*y*x + 1; (1, x y z);
z*y^2+ z^2*y - 3*z*y - 1; (1, x y z);
z^}2*x+\mp@subsup{z}{}{\wedge}2*y-1; (1, x y z)
y*x - z^2 + 3*z; (1, x y z);
y^2+z*y+ z^2-3*y-3*z; (1, x y z);

```

\section*{Bibliography}
[1] B. Amrhein, O. Gloor and W. Küchlin. On the Walk. Theoret. Comput. Sci. 187 (12) (1997) \(179-202\).

URL http://dx.doi.org/10.1016/S0304-3975(97)00064-9
[2] J. Apel. A Gröbner Approach to Involutive Bases. J. Symbolic Comput. 19 (5) (1995) 441-457.

URL http://dx.doi.org/10.1006/jsco.1995.1026
[3] J. Apel. The Computation of Gröbner Bases Using an Alternative Algorithm. Progr. Comput. Sci. Appl. Logic 15 (1998) 35-45.
[4] J. Apel. The Theory of Involutive Divisions and an Application to Hilbert Function Computations. J. Symbolic Comput. 25 (6) (1998) 683-704. URL http://dx.doi.org/10.1006/jsco.1997.0194
[5] T. Baader and N. Tobias. Term Rewriting and All That. Cambridge University Press (1998).
[6] J. Backelin, S. Cojocaru and V. Ufnarovski. Bergman 1.0 User Manual (1998). URL http://servus.math.su.se/bergman/
[7] T. Becker and V. Weispfenning. Gröbner Bases: A Commutative Approach to Commutative Algebra. Springer-Verlag (1993).
[8] G. M. Bergman. The Diamond Lemma for Ring Theory. Adv. in Math. 29 (2) (1978) 178-218.
URL http://dx.doi.org/10.1016/0001-8708(78)90010-5
[9] W. Böge, R. Gebauer and H. Kredel. Some Examples for Solving Systems of Algebraic Equations by Calculating Groebner Bases. J. Symbolic Comput. 2 (1) (1986) 83-98.
[10] B. Buchberger. A Criterion for Detecting Unnecessary Reductions in the Construction of Gröbner Bases. In Symbolic and Algebraic Computation (EUROSAM '79, Int. Symp., Marseille, 1979), volume 72 of Lecture Notes in Comput. Sci., 3-21. Springer, Berlin (1979).
[11] B. Buchberger. An Algorithmic Criterion for the Solvability of a System of Algebraic Equations. Translation of PhD thesis by M. Abramson and R. Lumbert. In B. Buchberger and F. Winkler, editors, Gröbner Bases and Applications, volume 251 of Proc. London Math. Soc., 535-545. Cambridge University Press (1998).
[12] B. Buchberger and F. Winkler, editors. Gröbner Bases and Applications, volume 251 of Proc. London Math. Soc.. Cambridge University Press (1998).
[13] J. Calmet, M. Hausdorf and W. M. Seiler. A Constructive Introduction to Involution. In R. Akerkar, editor, Proc. Int. Symp. Applications of Computer Algebra - ISACA 2000, 33-50. Allied Publishers, New Delhi (2001).
[14] J. A. Camberos, L. A. Lambe and R. Luczak. Hybrid Symbolic-Numeric Methodology for Fluid Dynamic Simulations. In 34th AIAA Fluid Dynamics Conference and Exhibit, volume 2004-2330. Portland, OR (2004).
[15] J. A. Camberos, L. A. Lambe and R. Luczak. Computational Physics with Hybrid Symbolic-Numeric Methodology with Multidisciplinary Applications. Department of Defense HPC User's Group Meeting (2005).
[16] J. A. Camberos, L. A. Lambe and R. Luczak. Hybrid Symbolic-Numeric Methodology: Views and Visions. In 43rd AIAA Aerospace Sciences Meeting, volume 20050092. Reno, NV (2005).
[17] J. F. Carlson. Cohomology, Computations, and Commutative Algebra. Notices Amer. Math. Soc. 52 (4) (2005) 426-434.
[18] M. Collart, M. Kalkbrener and D. Mall. Converting Bases with the Gröbner Walk. J. Symbolic Comput. 24 (3-4) (1997) 465-469.

URL http://dx.doi.org/10.1006/jsco.1996.0145
[19] N. Dershowitz. A Taste of Rewrite Systems. In P. E. Lauer, editor, Functional Programming, Concurrency, Simulation and Automated Reasoning, volume 693 of Lecture Notes in Comput. Sci., 199-228. Springer (1993).
[20] G. A. Evans. Noncommutative Involutive Bases. In Q.-N. Tran, editor, Proc. 10th Int. Conf. Applications of Computer Algebra, 49-64. Beaumont, Texas, USA (2004).
[21] J. C. Faugère, P. Gianni, D. Lazard and T. Mora. Efficient Computation of Zerodimensional Gröbner Bases by Change of Ordering. J. Symbolic Comput. 16 (4) (1993) 329-344.

URL http://dx.doi.org/10.1006/jsco.1993.1051
[22] R. Fröberg. An Introduction to Gröbner Bases. John Wiley \& Sons (1998).
[23] V. P. Gerdt. Involutive Division Technique: Some Generalizations and Optimizations. J. Math. Sci. (N. Y.) 108 (6) (2002) 1034-1051. URL http://dx.doi.org/10.1023/A:1013596522989
[24] V. P. Gerdt. Involutive Algorithms for Computing Gröbner Bases. In S. Cojocaru, G. Pfister and V. Ufnarovski, editors, Computational Commutative and NonCommutative Algebraic Geometry, volume 196 of NATO Science Series: Computer and Systems Sciences, 199-225. IOS Press (2005).
[25] V. P. Gerdt and Yu. A. Blinkov. Involutive Bases of Polynomial Ideals. Math. Comput. Simulation 45 (5-6) (1998) 519-541.
URL http://dx.doi.org/10.1016/S0378-4754(97)00127-4
[26] V. P. Gerdt and Yu. A. Blinkov. Minimal Involutive Bases. Math. Comput. Simulation 45 (5-6) (1998) 543-560.
URL http://dx.doi.org/10.1016/S0378-4754(97)00128-6
[27] V. P. Gerdt, Yu. A. Blinkov and D. A. Yanovich. Construction of Janet Bases I. Monomial Bases. In V. G. Ghanzha, E. W. Mayr and E. V. Vorozhtsov, editors, Computer Algebra in Scientific Computing (CASC) 2001, 233-247. Springer-Verlag, Berlin (2001).
[28] V. P. Gerdt, Yu. A. Blinkov and D. A. Yanovich. Construction of Janet Bases II. Polynomial Bases. In V. G. Ghanzha, E. W. Mayr and E. V. Vorozhtsov, editors, Computer Algebra in Scientific Computing (CASC) 2001, 249-263. Springer-Verlag, Berlin (2001).
[29] A. Giovini, T. Mora, G. Niedi, L. Robbiano and C. Traverso. "One sugar cube, please" OR Selection Strategies in Buchberger Algorithm. In ISSAC '91: Proc. Int.

Symp. Symbolic and Algebraic Computation, 49-54. ACM Press, New York (1991). URL http://dx.doi.org/10.1145/120694.120701
[30] O. D. Golubitsky. Involutive Gröbner Walk. Fundam. Prikl. Mat. 7 (4) (2001) 993-1001.
[31] G. M. Greuel, G. Pfister and H. Schönemann. Singular 3.0.0. A Computer Algebra System for Polynomial Computations, Centre for Computer Algebra, University of Kaiserslautern (2005).
URL http://www.singular.uni-kl.de
[32] J. W. Helton and M. Stankus. NCGB 3.1: NonCommutative Gröbner Basis Package (2002).

URL http://math.ucsd.edu/~ncalg/
[33] A. Heyworth. Rewriting as a special case of Gröbner Basis Theory. In M. Atkinson, N. D. Gilbert, J. Howie, S. A. Linton and E. F. Robertson, editors, Computational and Geometric Aspects of Modern Algebra (Edinburgh, 1998), volume 275 of Proc. London Math. Soc., 101-105. Cambridge University Press (2000).
[34] F. Hreinsdóttir. A case where choosing a product order makes the calculation of a Gröbner Basis much faster. J. Symbolic Comput. 18 (4) (1994) 373-378.
URL http://dx.doi.org/10.1006/jsco.1994.1053
[35] M. Janet. Lecons sur les systémes d'équations aux deriveés partielles. GauthierVillars, Paris (1929).
[36] J.-P. Jouannaud. Rewrite Proofs and Computations. In H. Schwichtenberg, editor, Proof and Computation, volume 139 of Computer and Systems Sciences. SpringerVerlag (1995).
[37] B. J. Keller. Algorithms and Orders for Finding Noncommutative Gröbner Bases. PhD thesis, Virginia Tech (1997).
[38] B. W. Kernighan and D. M. Ritchie. The C Programming Language (2nd Edition). Prentice Hall (1988).
[39] D. E. Knuth and P. B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, Computational Problems in Abstract Algebra, 263-297. Pergamon Press (1970).
[40] L. A. Lambe. FAlg (An ANSI C Library for Free Associative Algebra Calculations): User Guide (2001).
[41] L. A. Lambe. FMon (An ANSI C Library for Free Monoid Calculations): User Guide (2001).
[42] L. A. Lambe. Letter of Support for Research at the University of Wales, Bangor (2005).
[43] L. A. Lambe, R. Luczak and J. Nehrbass. A new finite difference method for the Helmholtz equation using symbolic computation. Int. J. Comput. Eng. Sci. 4 (1) (2003) 121-144.

URL http://dx.doi.org/10.1142/S1465876303000739
[44] E. Mansfield. Differential Gröbner Bases. PhD thesis, University of Sydney (1991).
[45] T. Mora. Gröbner Bases for non-commutative polynomial rings. In J. Calmet, editor, AAECC-3: Proc. 3rd Int. Conf. on Algebraic Algorithms and Error-Correcting Codes (Grenoble, France, July 15-19, 1985), volume 223 of Lecture Notes in Comput. Sci., 353-362. Springer (1986).
[46] MSSRC (Multidisciplinary Software Systems Research Corporation). URL http://www.mssrc.com
[47] J. F. Pommaret. Systems of Partial Differential Equations and Lie Pseudogroups. Gordon and Breach Science Publishers (1978).
[48] L. Robbiano. Term Orderings on the Polynomial Ring. In EUROCAL '85: Research Contributions from the European Conference on Computer Algebra - Volume 2, volume 204 of Lecture Notes in Comput. Sci., 513-517. Springer (1985).
[49] SciFace Software. MuPAD Version 3.1. Paderborn, Germany (2005). URL http://www.mupad.com/
[50] W. M. Seiler. A Combinatorial Approach to Involution and \(\delta\)-Regularity I: Involutive Bases in Polynomial Algebras of Solvable Type (2002). Preprint Universitä Mannheim.
[51] W. M. Seiler. A Combinatorial Approach to Involution and \(\delta\)-Regularity II: Structure Analysis of Polynomial Modules with Pommaret Bases (2002). Preprint Universitä Mannheim.
[52] J. M. Thomas. Differential Systems. AMS, New York (1937).
[53] Q.-N. Tran. A Fast Algorithm for Gröbner Basis Conversion and its Applications. J. Symbolic Comput. 30 (4) (2000) 451-467. URL http://dx.doi.org/10.1006/jsco.1999.0416
[54] A. Wachowski and L. Wachowski. The Matrix. Film (Warner Bros.) (1999).
[55] Waterloo Maple Inc. Maple Version 10. Waterloo, Ontario, Canada. (2005). URL http://www.maplesoft.com
[56] V. Weispfenning. Admissible orders and linear forms. ACM Sigsam Bull. 21 (2) (1987) 16-18.

URL http://doi.acm.org/10.1145/24554.24557
[57] Wolfram Research, Inc. Mathematica Version 5.1. Champaign, Illinois (2004). URL http://www.wolfram.com/
[58] A. Yu. Zharkov and Yu. A. Blinkov. Involution Approach to Investigating Polynomial Systems. Math. Comput. Simulation 42 (4-6) (1996) 323-332.
URL http://dx.doi.org/10.1016/S0378-4754(96)00006-7

\section*{Index}
Symbols
\(<\). ..... 14
\(\triangleleft\). ..... 120
\(\triangleright\) ..... 120
\(\longrightarrow\) ..... 20
\(\xrightarrow{*}\) ..... 21
\(\xrightarrow[I]{ }\) ..... 74
\(\xrightarrow[I]{*}\) ..... 74
\(\mathcal{C}(U)\) ..... 72, 107
\(\mathcal{C}(u, U)\) ..... 72,107
\(\mathcal{C}_{I}(U)\) ..... 72, 107
\(\mathcal{C}_{I}(u, U)\) ..... 71, 106
\(\operatorname{deg}_{\omega}\) ..... 166
\(\mathrm{in}_{\omega}\) ..... 167
\(\mathrm{in}_{\theta}\) ..... 178
\(\mathcal{J}\) ..... 74
LC \((p)\) ..... 14
LM \((J)\) ..... 29
LM \((p)\) ..... 14
LT( \(p\) ) ..... 14
\(\mathcal{M}_{I}(u, U)\) ..... 72
\(\mathcal{M}_{I}^{L}(u, U)\) ..... 106
\(\mathcal{M}_{I}^{R}(u, U)\) ..... 107
\(\mathcal{O}\) ..... 125
\(\mathcal{P}\) ..... 74
\(R\left[x_{1}, x_{2}, \ldots, x_{n}\right]\) ..... 11
\(R\left\langle x_{1}, x_{2}, \ldots, x_{n}\right\rangle\) ..... 12
\(\mathcal{S}\) ..... 135
T ..... 74
\(\theta\)-homogeneous ..... 178
\(\mathrm{val}_{i}\) ..... 177
W ..... 143
A
abelian ..... 10
admissible monomial ordering ..... 14
ascending chain condition ..... 29
autoreduction ..... 77, 110
B
basis ..... 13
Gröbner ..... 22, 25, 50
involutive ..... 80, 112
Bergman, George ..... 46
binary operation ..... 9
Bruno Buchberger ..... 30
Buchberger's algorithm ..... 30
Buchberger's first criterion ..... 35, 56
Buchberger's second criterion ..... 37, 57
Buchberger, Bruno ..... 30
C
coefficient ..... 10, 12
leading ..... 14
coefficient ring ..... 12
commutative Gröbner basis ..... 25
commutative polynomial ring ..... 11
commutative ring ..... 10
compatible ordering function ..... 178
compatible weight vector ..... 167
complete rewrite system ..... 155
involutive ..... 155
conclusive involutive division ..... 114
cone
conventional ..... 72, 107
involutive ..... 71, 106
constructive involutive division ..... 83
continuous involutive division ..... 80, 115
conventional cone ..... 72, 107
conventional divisor ..... 71, 105
conventional span ..... 72, 107
criterion
Buchberger's first ..... 35, 56
Buchberger's second ..... 37, 57
D
degree ..... 15
degree inverse lexicographic ordering 15
degree lexicographic ordering ..... 15
degree reverse lex. ordering ..... 15, 17
dehomogenisation ..... 39, 60, 97
Dickson's lemma ..... 27
divisioninvolutive72, 106
polynomial ..... 17
division ring ..... 10
divisor
conventional ..... 71, 105
involutive ..... 71, 105
thick ..... 105
thin ..... 105
E
empty division ..... 119
extended prefix turn ..... 198
extendible involutive division ..... 98
extendible monomial ordering....39, 60extendible ordering function179
F
FGLM ..... 42
field ..... 10
full division ..... 119
functional decomposition ..... 177
G
George Bergman ..... 46
global involutive division ..... 72, 107
Gröbner basis ..... 22
commutative ..... 25
logged ..... 43, 63
minimal ..... 34, 57
noncommutative ..... 50
reduced ..... 32, 54
Gröbner involutive division ..... 118
Gröbner walk ..... 179
group ..... 9
H
harmonious monomial ordering ..... 179
Hilbert's basis theorem ..... 29
homogeneous ..... 39
homogenisation ..... 39, 60, 97
I
ideal ..... 12
left ..... 13
monomial ..... 27
right ..... 13
two-sided ..... 13
ideal membership problem ..... 13
initial ..... 167, 178
inverse lexicographic ordering ..... 15, 17
involutive basis ..... 80, 112
locally ..... 79, 112
logged ..... 102, 162
involutive complete rewrite system ..... 155
involutive cone ..... 71, 106
involutive division ..... 72, 106
conclusive ..... 114
constructive ..... 83
continuous. ..... 80, 115
empty ..... 119
extendible ..... 98
full ..... 119
global ..... 72, 107
Gröbner ..... 118
Janet ..... 74
left ..... 120
left overlap ..... 125
local ..... 72, 107
Noetherian ..... 88
Pommaret ..... 74
prefix-only left overlap ..... 147
right ..... 120
stable ..... 89
strong ..... 106
strong left overlap ..... 135
subword-free left overlap. ..... 147
Thomas ..... 74
two sided left overlap ..... 143
weak ..... 107
involutive divisions ..... 71, 119
involutive divisor ..... 71, 105
involutive span ..... 72, 107
involutive walk ..... 176, 185
J
Janet division ..... 74

\section*{K}
Knuth-Bendix algorithm ..... 53
L
leading coefficient ..... 14
leading monomial ..... 14
leading term ..... 14
left division ..... 120
left ideal ..... 13
left overlap division ..... 125
left prolongation ..... 110
left prolongation turn ..... 196
lexicographic ordering ..... 15, 16
local involutive division ..... 72, 107
locally confluent ..... 25, 50
locally involutive basis ..... 79, 112
logged Gröbner basis ..... 43, 63
logged involutive basis ..... 102, 162
M
minimal Gröbner basis ..... 34, 57
monomial. ..... 10, 12
leading ..... 14
monomial ideal ..... 27
monomial ordering ..... 14
admissible ..... 14
degree inverse lexicographic ..... 15
degree lexicographic ..... 15
degree reverse lexicographic ..... 15, 17
extendible ..... 39, 60
harmonious ..... 179
inverse lexicographic ..... 15, 17
lexicographic. ..... 15, 16
Mora's algorithm ..... 52
Mora, Teo ..... 46
multidegree ..... 11
multiplicative variables ..... 72
N
Noetherian involutive division ..... 88
noncommutative Gröbner basis ..... 50
noncommutative polynomial ring ..... 12
noncommutative ring ..... 10
normal strategy ..... 40, 62
Oorderingmonomial........................... 14
ordering function ..... 177
compatible ..... 178
extendible ..... 179
overlap ..... 47
self. ..... 49
overlap word ..... 62
P
padding symbol ..... 16
polynomial ..... 10, 12
polynomial division ..... 17
polynomial ring
commutative ..... 11
noncommutative ..... 12
Pommaret division ..... 74
prefix ..... 47
prefix turn ..... 197
extended ..... 198
prefix-only left overlap division ..... 147
prolongation ..... 77
left ..... 110
right ..... 110
prolongation turn
left ..... 196
right ..... 196
R
reduced Gröbner basis ..... 32, 54
remainder
unique ..... 25, 50, 93, 117
right division ..... 120
right ideal ..... 13
right prolongation ..... 110
right prolongation turn ..... 196
ring ..... 10
coefficient ..... 12
commutative ..... 10
division ..... 10
noncommutative ..... 10
sub- ..... 10
rng ..... 10
S
S-polynomial ..... 24, 49
selection strategies ..... 40, 62, 96
self overlap ..... 49
span
conventional ..... 72, 107
involutive ..... 72, 107
stable involutive division ..... 89
strategy
normal ..... 40, 62
sugar ..... 41, 62
tie-breaking ..... 64
strong involutive division ..... 106
strong left overlap division ..... 135
subring ..... 10
subword ..... 47
subword-free left overlap division ..... 147
suffix ..... 47
suffix turn ..... 197
sugar strategy ..... 41, 62
T
Teo Mora ..... 46
term ..... 10, 12
leading ..... 14
thick divisor ..... 105
thin divisor ..... 105
Thomas division ..... 74
tie-breaking strategy ..... 64
two sided left overlap division ..... 143
two-sided ideal ..... 13
U
unique remainder ..... 25, 50, 93, 117
W
walk
Gröbner ..... 42, 165, 179
involutive ..... 176, 185
weak involutive division ..... 107
word problem ..... 53
INDEX ..... 378```


[^0]:    ${ }^{1}$ For a commutative monomial ordering, we can ignore the monomial $m_{r}$.

[^1]:    ${ }^{1}$ The other possible case is $\operatorname{LT}\left(p_{4}\right)=s a_{2}$, in which case it is $r_{1}$ that reduces to $r_{2}$ and not $r_{2}$ to $r_{1}$.
    ${ }^{2}$ This may not strictly be true if $p_{3}$ is divisible by $p_{4}$; for the time being we shall assume that this is not the case, noting that the important concept here is of eliminating the non-uniqueness given by the

[^2]:    choice of dividing $p_{3}$ by $p_{1}$ or $p_{2}$ first.
    ${ }^{3}$ The S stands for Syzygy, as in a pair of connected objects.

[^3]:    ${ }^{4}$ Think of $C_{0}$ as the set of monomials $m \in J$ which are also members of $\mathcal{R}^{\prime}$; think of $C_{j}$ (for $j \geqslant 1$ ) as containing all the elements of $C_{j-1}$ plus the monomials $m \in J$ of the form $m=m^{\prime} x_{n}^{j}, m^{\prime} \in \mathcal{R}^{\prime}$.

[^4]:    ${ }^{5}$ Substitutions for $g_{i}$ may also occur in the summation $\sum_{u=1}^{\mu} t_{u} g_{c_{u}}$; these substitutions have not been considered in the displayed formulae.

[^5]:    ${ }^{1}$ Substitutions for $g_{i}$ may also occur in the summation $\sum_{u=1}^{\mu} \ell_{u} g_{c_{u}} r_{u}$; these substitutions have not been considered in the displayed formulae.

[^6]:    ${ }^{2}$ For completeness, we note that the S-polynomial corresponding to the overlap can also be of the form $\mathrm{S}-\mathrm{pol}\left(\ell_{3}^{\prime}, h, \ell_{1}^{\prime}, f\right)$; this (inconsequentially) swaps the first two terms of Equation (3.5).

[^7]:    ${ }^{1}$ For $1 \leqslant d \leqslant A, p_{\delta_{d}} t_{d}=p_{\alpha_{a}} t_{a}(1 \leqslant a \leqslant A)$; for $A+1 \leqslant d \leqslant A+B, p_{\delta_{d}} t_{d}=p_{\beta_{b}} t_{b}(1 \leqslant b \leqslant B)$; and for $A+B+1 \leqslant d \leqslant A+B+C=: D, p_{\delta_{d}} t_{d}=p_{\gamma_{c}} t_{c}(1 \leqslant c \leqslant C)$.

[^8]:    ${ }^{2}$ This claim is integral to the proof of Theorem 6.4 in [50], a theorem that states than an algorithm corresponding to Algorithm 9 in this thesis terminates.

[^9]:    ${ }^{1}$ For $1 \leqslant d \leqslant A, u_{d} p_{\delta_{d}} v_{d}=u_{a} p_{\alpha_{a}} v_{a}(1 \leqslant a \leqslant A)$; for $A+1 \leqslant d \leqslant A+B, u_{d} p_{\delta_{d}} v_{d}=u_{b} p_{\beta_{b}} v_{b}$ $(1 \leqslant b \leqslant B)$; and for $A+B+1 \leqslant d \leqslant A+B+C=: D, u_{d} p_{\delta_{d}} v_{d}=u_{c} p_{\gamma_{c}} v_{c}(1 \leqslant c \leqslant C)$.

[^10]:    ${ }^{2}$ Technical point: if $\gamma \neq \beta+1$, the S-polynomial $s_{v}$ could in fact appear as $s_{v}=c_{v} g_{j} x_{i_{D+1}}-$ $c_{v}^{\prime}\left(x_{j_{1}} \ldots x_{i_{D+1-\gamma}}\right) g_{k}$ and not as $s_{v}=c_{v}\left(x_{j_{1}} \ldots x_{i_{D+1-\gamma}}\right) g_{k}-c_{v}^{\prime} g_{j} x_{i_{D+1}}$; for simplicity we will treat both cases the same in the proof as all that changes is the notation and the signs.

[^11]:    ${ }^{1}$ The ideal $\operatorname{in}_{\omega}(J)$ is defined as follows: $p \in J$ if and only if $\mathrm{in}_{\omega}(p) \in \operatorname{in}_{\omega}(J)$.

[^12]:    ${ }^{2}$ Think of $\mathrm{val}_{\operatorname{deg}(m)}$ as finding the value of the final variable in $m$ (as opposed to $\mathrm{val}_{1}$ finding the value of the first variable in $m$ ).
    ${ }^{3}$ The ideal $\mathrm{in}_{\theta}(J)$ is defined as follows: $p \in J$ if and only if $\mathrm{in}_{\theta}(p) \in \mathrm{in}_{\theta}(J)$.

[^13]:    ${ }^{1}$ Technical point: if $\gamma \neq \beta+1$, the S-polynomial $s_{v}$ could in fact appear as $s_{v}=c_{v} g_{j} x_{i_{D+1}}-$ $c_{v}^{\prime}\left(x_{j_{1}} \ldots x_{i_{D+1-\gamma}}\right) g_{k}$ and not as $s_{v}=c_{v}\left(x_{j_{1}} \ldots x_{i_{D+1-\gamma}}\right) g_{k}-c_{v}^{\prime} g_{j} x_{i_{D+1}}$; for simplicity we will treat both cases the same in the proof as all that changes is the notation and the signs.

