## Contents

1 Introduction .................................................. 15
   1.1 History and Systems ........................................ 17
      1.1.1 The ‘calculus’ side ...................................... 17
      1.1.2 The ‘group theory’ side .................................. 18
      1.1.3 A synthesis? .............................................. 18
   1.2 Expansion and Simplification .............................. 18
      1.2.1 A Digression on “Functions” ......................... 20
      1.2.2 Expansion ................................................. 21
      1.2.3 Simplification ............................................ 22
      1.2.4 An example of simplification ...................... 25
   1.3 Algebraic Definitions ..................................... 25
      1.3.1 Algebraic Closures ...................................... 28
   1.4 Some Complexity Theory .................................... 28
      1.4.1 Probabilistic Algorithms ............................ 30
   1.5 Some Maple ................................................. 31
      1.5.1 Maple polynomials ...................................... 31
      1.5.2 The RootOf construct ................................. 32
      1.5.3 The simplify command .............................. 32

2 Polynomials .................................................. 33
   2.1 What are polynomials? .................................... 33
      2.1.1 How do we manipulate polynomials? .................. 35
      2.1.2 Polynomials in one variable ......................... 35
      2.1.3 A factored representation ........................... 39
      2.1.4 Polynomials in several variables ................... 41
      2.1.5 Other representations .................................. 42
      2.1.6 The Newton Representation .......................... 47
      2.1.7 Representations in Practice .......................... 47
      2.1.8 Comparative Sizes ...................................... 49
   2.2 Rational Functions ......................................... 50
      2.2.1 Canonical Rational Functions ........................ 50
      2.2.2 Candidness of rational functions .................... 52
   2.3 Greatest Common Divisors ................................. 53
      2.3.1 Polynomials in one variable ......................... 54
2.3.2 Subresultant sequences ............................................. 58
2.3.3 The Extended Euclidean Algorithm ......................... 60
2.3.4 Partial Fractions .................................................. 61
2.3.5 Polynomials in several variables ................................. 62
2.3.6 Square-free decomposition ..................................... 64
2.3.7 Sparse Complexity ................................................ 65
2.4 Non-commutative polynomials ..................................... 66

3 Polynomial Equations ................................................. 69
3.1 Equations in One Variable ........................................... 69
  3.1.1 Quadratic Equations ........................................... 69
  3.1.2 Cubic Equations ................................................ 70
  3.1.3 Quartic Equations .............................................. 72
  3.1.4 Higher Degree Equations .................................... 72
  3.1.5 Reducible defining polynomials ............................... 73
  3.1.6 Multiple Algebraic Numbers .................................. 74
  3.1.7 Solutions in Real Radicals .................................... 74
  3.1.8 Equations of curves ............................................ 75
  3.1.9 How many Real Roots? ........................................ 76
  3.1.10 Thom's Lemma .................................................. 79

3.2 Linear Equations in Several Variables ......................... 79
  3.2.1 Linear Equations and Matrices ............................... 80
  3.2.2 Representations of Matrices .................................. 80
  3.2.3 Matrix Inverses: not a good idea! .......................... 82
  3.2.4 Complexity ...................................................... 86
  3.2.5 Over/under-determined Systems .............................. 87

3.3 Nonlinear Multivariate Equations: Distributed ................ 88
  3.3.1 Gröbner Bases .................................................. 91
  3.3.2 How many Solutions? ......................................... 93
  3.3.3 Orderings ....................................................... 95
  3.3.4 Complexity of Gröbner Bases ................................. 98
  3.3.5 A Matrix Formulation ....................................... 101
  3.3.6 Example ........................................................ 103
  3.3.7 The Gianni–Kalkbrener Theorem .............................. 104
  3.3.8 The Faugère–Gianni–Lazard–Mora Algorithm .............. 107
  3.3.9 The Gröbner Walk .............................................. 111
  3.3.10 Factorization and Gröbner Bases ............................ 114
  3.3.11 The Shape Lemma ........................................... 114
  3.3.12 The Hilbert function ....................................... 116
  3.3.13 Coefficients other than fields .............................. 116
  3.3.14 Non-commutative Ideals .................................... 117

3.4 Nonlinear Multivariate Equations: Recursive .................. 118
  3.4.1 Triangular Sets and Regular Chains ....................... 118
  3.4.2 Zero Dimension ............................................... 119
  3.4.3 Positive Dimension ......................................... 120
  3.4.4 Conclusion .................................................... 123
3.5 Equations and Inequalities

3.5.1 Applications ............................................. 125
3.5.2 Real Radical ............................................ 126
3.5.3 Quantifier Elimination ................................. 126
3.5.4 Algebraic Decomposition .............................. 128
3.5.5 Cylindrical Algebraic Decomposition ................. 131
3.5.6 Computing Algebraic Decompositions ................. 134
3.5.7 Describing Solutions .................................. 135
3.5.8 Complexity ............................................. 139
3.5.9 Further Observations .................................. 140

3.6 Conclusions ................................................. 141

4 Modular Methods ............................................ 143

4.1 Matrices: a Simple Example ................................ 144
4.1.1 Matrices with integer coefficients: Determinants .... 145
4.1.2 Matrices with polynomial coefficients: Determinants . 146
4.1.3 Conclusion: Determinants .............................. 147
4.1.4 Linear Equations with integer coefficients ............. 147
4.1.5 Linear Equations with polynomial coefficients ....... 148
4.1.6 Conclusion: Linear Equations .......................... 148
4.1.7 Matrix Inverses ........................................ 149

4.2 Gcd in one variable ........................................ 149
4.2.1 Bounds on divisors ..................................... 150
4.2.2 The modular – integer relationship.................... 151
4.2.3 Computing the g.c.d.: one large prime ............... 153
4.2.4 Computing the g.c.d.: several small primes .......... 155
4.2.5 Computing the g.c.d.: early success ................... 157
4.2.6 An alternative correctness check ....................... 157
4.2.7 Conclusion ............................................. 158

4.3 Polynomials in two variables ............................. 160
4.3.1 Degree Growth in Coefficients ....................... 160
4.3.2 The evaluation–interpolation relationship ............. 162
4.3.3 G.c.d. in $\mathbb{Z}[x, y]$ ................................ 163
4.3.4 G.c.d. in $\mathbb{Z}[x, y]$ ................................ 164

4.4 Polynomials in several variables ......................... 167
4.4.1 A worked example .................................... 169
4.4.2 Converting this to an algorithm ....................... 170
4.4.3 Worked example continued ............................ 171
4.4.4 Conclusions ........................................... 176

4.5 Further Applications ...................................... 177
4.5.1 Resultants and Discriminants ......................... 177
4.5.2 Linear Systems ........................................ 178

4.6 Gröbner Bases .............................................. 180
4.6.1 General Considerations ............................... 182
4.6.2 The Hilbert Function and reduction ................... 182
# CONTENTS

4.6.3 The Modular Algorithm ........................................... 184
4.6.4 Conclusion ......................................................... 185
4.7 Conclusions .......................................................... 187

5 \textit{p}-adic Methods ..................................................... 189
5.1 Introduction to the factorization problem ......................... 189
5.2 Modular methods ..................................................... 191
5.2.1 The Musser test ................................................... 191
5.3 Factoring modulo a prime .......................................... 194
5.3.1 Berlekamp’s small \textit{p} method ................................ 194
5.3.2 The Cantor–Zassenhaus method ................................ 195
5.3.3 Berlekamp’s large \textit{p} method ................................ 197
5.4 From \mathbb{Z}_p to \mathbb{Z} ................................................. 197
5.5 Hensel Lifting ......................................................... 200
5.5.1 Linear Hensel Lifting .............................................. 200
5.5.2 Quadratic Hensel Lifting ........................................ 201
5.5.3 Hybrid Hensel Lifting ............................................ 203
5.6 The recombination problem ........................................ 205
5.7 Univariate Factoring Solved ......................................... 206
5.8 Multivariate Factoring ............................................... 208
5.8.1 A “Good Reduction” Complexity Result ..................... 208
5.8.2 A Sparsity Result ................................................ 210
5.8.3 The Leading Coefficient Problem ................................ 210
5.9 Other Applications ................................................... 212
5.9.1 \textit{p}-adic Greatest Common Divisors ......................... 212
5.9.2 \textit{p}-adic Gröbner Bases ......................................... 212
5.9.3 \textit{p}-adic determinants .......................................... 214
5.10 Conclusions .......................................................... 214

6 Algebraic Numbers and Functions .................................... 215
6.1 Representations of Algebraic Numbers ............................ 217
6.2 Factorisation with Algebraic Numbers ............................ 218
6.3 The D5 approach to algebraic numbers ............................ 219

7 Calculus ................................................................. 221
7.1 Introduction ............................................................ 221
7.2 Integration of Rational Expressions ............................... 223
7.2.1 Integration of Proper Rational Expressions .................. 223
7.2.2 Hermite’s Algorithm .............................................. 224
7.2.3 The Ostrogradski–Horowitz Algorithm ....................... 225
7.2.4 The Trager–Rothstein Algorithm ............................... 226
7.3 Theory: Liouville’s Theorem ....................................... 229
7.3.1 Liouville’s Principle .............................................. 231
7.3.2 Finding L .......................................................... 232
7.3.3 Risch Structure Theorem ........................................ 233
7.3.4 Overview of Integration ........................................ 233
<table>
<thead>
<tr>
<th>CONTENTS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>7.4 Integration of Logarithmic Expressions</td>
<td>235</td>
</tr>
<tr>
<td>7.4.1 The Polynomial Part</td>
<td>236</td>
</tr>
<tr>
<td>7.4.2 The Rational Expression Part</td>
<td>236</td>
</tr>
<tr>
<td>7.4.3 Conclusion of Logarithmic Integration</td>
<td>237</td>
</tr>
<tr>
<td>7.5 Integration of Exponential Expressions</td>
<td>239</td>
</tr>
<tr>
<td>7.5.1 The Polynomial Part</td>
<td>240</td>
</tr>
<tr>
<td>7.5.2 The Rational Expression Part</td>
<td>240</td>
</tr>
<tr>
<td>7.6 Integration of Algebraic Expressions</td>
<td>244</td>
</tr>
<tr>
<td>7.7 The Risch Differential Equation Problem</td>
<td>244</td>
</tr>
<tr>
<td>7.8 The Parallel Approach</td>
<td>247</td>
</tr>
<tr>
<td>7.8.1 The Parallel Approach: Algebraic Expressions</td>
<td>248</td>
</tr>
<tr>
<td>7.9 Definite Integration</td>
<td>248</td>
</tr>
<tr>
<td>7.10 Other Calculus Problems</td>
<td>249</td>
</tr>
<tr>
<td>7.10.1 Indefinite summation</td>
<td>249</td>
</tr>
<tr>
<td>7.10.2 Definite Symbolic Summation</td>
<td>249</td>
</tr>
<tr>
<td>7.10.3</td>
<td>250</td>
</tr>
<tr>
<td>7.10.4</td>
<td>250</td>
</tr>
<tr>
<td>8 Algebra versus Analysis</td>
<td>251</td>
</tr>
<tr>
<td>8.1 Functions and Formulae</td>
<td>251</td>
</tr>
<tr>
<td>8.2 Branch Cuts</td>
<td>253</td>
</tr>
<tr>
<td>8.2.1 Some Unpleasant Facts</td>
<td>253</td>
</tr>
<tr>
<td>8.2.2 The Problem with Square Roots</td>
<td>254</td>
</tr>
<tr>
<td>8.2.3 Possible Solutions</td>
<td>254</td>
</tr>
<tr>
<td>8.2.4 Removable Branch Cuts</td>
<td>257</td>
</tr>
<tr>
<td>8.3 Fundamental Theorem of Calculus Revisited</td>
<td>258</td>
</tr>
<tr>
<td>8.4 Constants Revisited</td>
<td>258</td>
</tr>
<tr>
<td>8.4.1 Constants can be useful</td>
<td>259</td>
</tr>
<tr>
<td>8.4.2 Constants are often troubling</td>
<td>259</td>
</tr>
<tr>
<td>8.5 Integrating ‘real’ Functions</td>
<td>259</td>
</tr>
<tr>
<td>8.6 Logarithms revisited</td>
<td>261</td>
</tr>
<tr>
<td>8.7 Other decision questions</td>
<td>261</td>
</tr>
<tr>
<td>8.8 Limits</td>
<td>264</td>
</tr>
<tr>
<td>8.8.1 A Definite Integral</td>
<td>264</td>
</tr>
<tr>
<td>A Algebraic Background</td>
<td>267</td>
</tr>
<tr>
<td>A.1 The resultant</td>
<td>267</td>
</tr>
<tr>
<td>A.2 Useful Estimates</td>
<td>270</td>
</tr>
<tr>
<td>A.2.1 Matrices</td>
<td>270</td>
</tr>
<tr>
<td>A.2.2 Coefficients of a polynomial</td>
<td>271</td>
</tr>
<tr>
<td>A.2.3 Roots of a polynomial</td>
<td>273</td>
</tr>
<tr>
<td>A.2.4 Root separation</td>
<td>274</td>
</tr>
<tr>
<td>A.2.5 Developments</td>
<td>275</td>
</tr>
<tr>
<td>A.3 Chinese Remainder Theorem</td>
<td>277</td>
</tr>
<tr>
<td>A.4 Chinese Remainder Theorem for Polynomials</td>
<td>278</td>
</tr>
<tr>
<td>A.5 Vandermonde Systems</td>
<td>280</td>
</tr>
</tbody>
</table>
A.6 Algebraic Structures ............................... 283

B Excursus .......................... 285
B.1 The Budan–Fourier Theorem ............................... 285
B.2 Equality of factored polynomials ............................... 286
B.3 Karatsuba’s method ........................................ 288
   B.3.1 Karatsuba’s method in practice ............................... 289
   B.3.2 Karatsuba’s method and sparse polynomials ............................... 289
   B.3.3 Karatsuba’s method and multivariate polynomials ............................... 290
   B.3.4 Faster still ............................... 290
   B.3.5 Faster division ............................... 291
   B.3.6 Faster g.c.d. computation ............................... 291
B.4 Strassen’s method ........................................ 291
   B.4.1 Strassen’s method in practice ............................... 293
   B.4.2 Further developments ............................... 293
   B.4.3 Matrix Inversion ............................... 294

C Systems .......................... 295
C.1 Axiom ........................................ 295
   C.1.1 Overview ........................................ 295
   C.1.2 History ........................................ 295
   C.1.3 Structure ........................................ 296
C.2 Macsyma ........................................ 296
   C.2.1 Overview ........................................ 296
   C.2.2 History ........................................ 298
C.3 Maple ........................................ 298
   C.3.1 Overview ........................................ 298
   C.3.2 History ........................................ 298
   C.3.3 Data structures ........................................ 298
   C.3.4 Heuristic GCD ........................................ 301
   C.3.5 Conclusion ........................................ 301
C.4 MuPAD ........................................ 302
   C.4.1 Overview ........................................ 302
   C.4.2 History ........................................ 302
C.5 Reduce ........................................ 303
   C.5.1 Overview ........................................ 303
   C.5.2 History ........................................ 303

D Index of Notation .............. 305
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Converting Monte Carlo to Las Vegas</td>
<td>31</td>
</tr>
<tr>
<td>1.2</td>
<td>An example of Maple’s RootOf construct</td>
<td>32</td>
</tr>
<tr>
<td>2.1</td>
<td>A polynomial SLP</td>
<td>44</td>
</tr>
<tr>
<td>2.2</td>
<td>Code fragment A — a graph</td>
<td>45</td>
</tr>
<tr>
<td>2.3</td>
<td>Code fragment B — a tree</td>
<td>46</td>
</tr>
<tr>
<td>2.4</td>
<td>DAG representation</td>
<td>46</td>
</tr>
<tr>
<td>2.5</td>
<td>Tree representation</td>
<td>46</td>
</tr>
<tr>
<td>2.6</td>
<td>Maple’s Original Polynomials</td>
<td>48</td>
</tr>
<tr>
<td>2.7</td>
<td>Maple’s New-Style Polynomials</td>
<td>49</td>
</tr>
<tr>
<td>2.8</td>
<td>Subresultant p.r.s. algorithm</td>
<td>59</td>
</tr>
<tr>
<td>3.1</td>
<td>Program for computing solutions to a cubic</td>
<td>71</td>
</tr>
<tr>
<td>3.2</td>
<td>Program for computing solutions to a quartic</td>
<td>72</td>
</tr>
<tr>
<td>3.3</td>
<td>$x^3 - x^2$ illustrating Thom’s Lemma</td>
<td>80</td>
</tr>
<tr>
<td>3.4</td>
<td>Gianni-Kalkbrener Algorithm</td>
<td>106</td>
</tr>
<tr>
<td>3.5</td>
<td>Algorithm 14</td>
<td>111</td>
</tr>
<tr>
<td>3.6</td>
<td>Body of Algorithm 15</td>
<td>113</td>
</tr>
<tr>
<td>3.7</td>
<td>Cylindrical Decomposition after Collins</td>
<td>134</td>
</tr>
<tr>
<td>3.8</td>
<td>$y^3 - 7y^2 + 14y - x - 8$: Thom’s Lemma</td>
<td>136</td>
</tr>
<tr>
<td>3.9</td>
<td>$y^3 - 7y^2 + 14y - x - 8$: indexing</td>
<td>137</td>
</tr>
<tr>
<td>4.1</td>
<td>Diagrammatic illustration of Modular Algorithms</td>
<td>143</td>
</tr>
<tr>
<td>4.2</td>
<td>Diagrammatic illustration of Algorithm 17</td>
<td>153</td>
</tr>
<tr>
<td>4.3</td>
<td>Algorithm 18</td>
<td>156</td>
</tr>
<tr>
<td>4.4</td>
<td>Diagrammatic illustration of Algorithm 18</td>
<td>157</td>
</tr>
<tr>
<td>4.5</td>
<td>“Early termination” g.c.d. code</td>
<td>158</td>
</tr>
<tr>
<td>4.6</td>
<td>Algorithm 19</td>
<td>159</td>
</tr>
<tr>
<td>4.7</td>
<td>Diagrammatic illustration of Algorithm 21</td>
<td>164</td>
</tr>
<tr>
<td>4.8</td>
<td>Algorithm 21</td>
<td>165</td>
</tr>
<tr>
<td>4.9</td>
<td>Diagrammatic illustration of g.c.d.s in $\mathbb{Z}[x, y]$ (1)</td>
<td>166</td>
</tr>
<tr>
<td>4.10</td>
<td>Diagrammatic illustration of g.c.d.s in $\mathbb{Z}[x, y]$ (2)</td>
<td>166</td>
</tr>
<tr>
<td>4.11</td>
<td>Diagrammatic illustration of sparse g.c.d.</td>
<td>172</td>
</tr>
<tr>
<td>4.12</td>
<td>Algorithm 22 Sparse g.c.d.</td>
<td>172</td>
</tr>
</tbody>
</table>
4.13 Algorithm 23: Inner sparse g.c.d. ........................................ 173
4.14 Algorithm 24: Sparse g.c.d. from skeleton ................................. 174
4.15 f from section 4.4.3 ................................................................ 174
4.16 g from section 4.4.3 ................................................................ 175
4.17 Algorithm 26 ............................................................................. 186
5.1 Diagrammatic illustration of Hensel Algorithms .............................. 189
5.2 Algorithm 27: Berlekamp for small p ........................................... 196
5.3 Algorithm 28: Distinct Degree Factorization ................................. 197
5.4 Algorithm 29: Split a Distinct Degree Factorization ....................... 198
5.5 Algorithm 30 .............................................................................. 202
5.6 Algorithm 31 .............................................................................. 202
5.7 Algorithm 32 .............................................................................. 203
5.8 Algorithm 33 .............................................................................. 204
5.9 Overview of Factoring Algorithm .................................................. 207
5.10 Algorithm 36 .............................................................................. 209
6.1 Non-candidness of algebras ............................................................ 216
6.2 Algebraic numbers in the denominator .......................................... 216
6.3 An evaluation of Maple’s RootOf construct .................................... 218
7.1 Algorithm 40: IntLog–Polynomial .................................................. 237
7.2 Algorithm 41: IntLog–Rational Expression .................................... 238
7.3 Algorithm 42: IntExp–Polynomial ................................................. 241
7.4 Algorithm 43: IntExp–Rational Expression .................................... 242
8.1 A Riemann surface example: log ................................................... 256
8.2 plot3d(C, x=-4..4, y=-4..4): C from (8.20) .................................. 260
8.3 Graph of apparent integral in (8.22) .............................................. 262
C.1 Axiom output .............................................................................. 296
C.2 Axiom type system ..................................................................... 297
C.3 Macsyma output ........................................................................ 297
C.4 Maple output ............................................................................. 299
C.5 Tree for A, B corresponding to table C.1 ....................................... 300
C.6 Tree for A, B corresponding to table C.2 ....................................... 301
C.7 MuPAD output .......................................................................... 302
C.8 Reduce output .......................................................................... 303
# List of Figures

### List of Algorithms

<table>
<thead>
<tr>
<th>Figure</th>
<th>Algorithm</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Miller–Rabin primality testing</td>
<td>31</td>
</tr>
<tr>
<td>2</td>
<td>Euclid</td>
<td>54</td>
</tr>
<tr>
<td>3</td>
<td>General g.c.d.</td>
<td>56</td>
</tr>
<tr>
<td>4</td>
<td>Subresultant p.r.s.</td>
<td>59</td>
</tr>
<tr>
<td>5</td>
<td>Extended Euclidean</td>
<td>60</td>
</tr>
<tr>
<td>6</td>
<td>General extended p.r.s.</td>
<td>61</td>
</tr>
<tr>
<td>7</td>
<td>Bivariate g.c.d.</td>
<td>62</td>
</tr>
<tr>
<td>8</td>
<td>Sturm Sequence evaluation</td>
<td>77</td>
</tr>
<tr>
<td>9</td>
<td>Buchberger</td>
<td>92</td>
</tr>
<tr>
<td>10</td>
<td>Gianni–Kalkbrener</td>
<td>106</td>
</tr>
<tr>
<td>11</td>
<td>Gianni–Kalkbrener Step</td>
<td>106</td>
</tr>
<tr>
<td>12</td>
<td>FGLM</td>
<td>108</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>110</td>
</tr>
<tr>
<td>14</td>
<td>Extended Buchberger</td>
<td>111</td>
</tr>
<tr>
<td>15</td>
<td>Gröbner Walk</td>
<td>111</td>
</tr>
<tr>
<td>16</td>
<td>Modular Linear Equations</td>
<td>147</td>
</tr>
<tr>
<td>17</td>
<td>Modular GCD (Large prime version)</td>
<td>153</td>
</tr>
<tr>
<td>18</td>
<td>Modular GCD (Small prime version)</td>
<td>156</td>
</tr>
<tr>
<td>19</td>
<td>Modular GCD (Alternative small prime version)</td>
<td>159</td>
</tr>
<tr>
<td>20</td>
<td>Content</td>
<td>164</td>
</tr>
<tr>
<td>21</td>
<td>Bivariate Modular GCD</td>
<td>165</td>
</tr>
<tr>
<td>22</td>
<td>Sparse g.c.d.</td>
<td>172</td>
</tr>
<tr>
<td>23</td>
<td>Inner sparse g.c.d.</td>
<td>173</td>
</tr>
<tr>
<td>24</td>
<td>Sparse g.c.d. from skeleton</td>
<td>174</td>
</tr>
<tr>
<td>25</td>
<td>Farey Reconstruction</td>
<td>179</td>
</tr>
<tr>
<td>26</td>
<td>Modular Gröbner base</td>
<td>186</td>
</tr>
<tr>
<td>27</td>
<td></td>
<td>196</td>
</tr>
<tr>
<td>28</td>
<td>Distinct Degree Factorization</td>
<td>197</td>
</tr>
<tr>
<td>29</td>
<td>Split a Distinct Degree Factorization</td>
<td>198</td>
</tr>
<tr>
<td>30</td>
<td>Hensel Lifting (Linear Two Factor version)</td>
<td>202</td>
</tr>
<tr>
<td>31</td>
<td>Univariate Hensel Lifting (Linear version)</td>
<td>202</td>
</tr>
<tr>
<td>32</td>
<td>Univariate Hensel Lifting (Quadratic Two Factor version)</td>
<td>203</td>
</tr>
<tr>
<td>33</td>
<td>Univariate Hensel Lifting (Quadratic version)</td>
<td>204</td>
</tr>
<tr>
<td>34</td>
<td>Combine Modular Factors</td>
<td>205</td>
</tr>
<tr>
<td>35</td>
<td>Factor over ( \mathbb{Z} )</td>
<td>207</td>
</tr>
<tr>
<td>36</td>
<td>Multivariate Hensel Lifting (Linear version)</td>
<td>209</td>
</tr>
<tr>
<td>37</td>
<td>Wang’s EEZ Hensel Lifting</td>
<td>211</td>
</tr>
<tr>
<td>38</td>
<td>Trager–Rothstein</td>
<td>227</td>
</tr>
<tr>
<td>39</td>
<td>Integration Paradigm</td>
<td>231</td>
</tr>
<tr>
<td>40</td>
<td>IntLog–Polynomial</td>
<td>237</td>
</tr>
<tr>
<td>41</td>
<td>IntLog–Rational Expression</td>
<td>238</td>
</tr>
<tr>
<td>42</td>
<td>IntExp–Polynomial</td>
<td>241</td>
</tr>
<tr>
<td>43</td>
<td>IntExp–Rational Expression</td>
<td>242</td>
</tr>
<tr>
<td>45</td>
<td>resultant</td>
<td>268</td>
</tr>
</tbody>
</table>
10

LIST OF FIGURES

46 Chinese Remainder .................................................. 278
47 Chinese Remainder (Polynomial form) ................................ 278
48 Chinese Remainder for Polynomials .................................. 279
49 Chinese Remainder (Multivariate) .................................. 279
50 Vandermonde solver ..................................................... 281
51 Vandermonde variant solver .......................................... 281
List of Open Problems

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Algebra of $O$</td>
<td>30</td>
</tr>
<tr>
<td>2</td>
<td>Sparse gcd (strong)</td>
<td>66</td>
</tr>
<tr>
<td>3</td>
<td>Sparse gcd (weak)</td>
<td>66</td>
</tr>
<tr>
<td>4</td>
<td>Roots of Sparse Polynomials</td>
<td>78</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>87</td>
</tr>
<tr>
<td>6</td>
<td>Sparse Gröbner Bases</td>
<td>98</td>
</tr>
<tr>
<td>7</td>
<td>Complexity of the FGLM Algorithm (I)</td>
<td>108</td>
</tr>
<tr>
<td>8</td>
<td>Complexity of the FGLM Algorithm (II)</td>
<td>108</td>
</tr>
<tr>
<td>9</td>
<td>Coefficient growth in the FGLM Algorithm</td>
<td>110</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>114</td>
</tr>
<tr>
<td>11</td>
<td>Better treatment of division</td>
<td>125</td>
</tr>
<tr>
<td>12</td>
<td>RAG Formulation I</td>
<td>133</td>
</tr>
<tr>
<td>13</td>
<td>Not all CADs are computed</td>
<td>141</td>
</tr>
<tr>
<td>14</td>
<td>Matrix Determinant costs</td>
<td>147</td>
</tr>
<tr>
<td>15</td>
<td>Improving Landau–Mignotte for g.c.d.</td>
<td>155</td>
</tr>
<tr>
<td>16</td>
<td>Alternative Route for Bivariate Polynomial g.c.d.</td>
<td>161</td>
</tr>
<tr>
<td>17</td>
<td>Which is the Better Route for Bivariate g.c.d.?</td>
<td>166</td>
</tr>
<tr>
<td>18</td>
<td>Bad reductions in Zippel’s algorithm</td>
<td>176</td>
</tr>
<tr>
<td>19</td>
<td>Contradictory Hilbert Functions</td>
<td>183</td>
</tr>
<tr>
<td>20</td>
<td>Bad Reduction for Gröbner Bases</td>
<td>184</td>
</tr>
<tr>
<td>21</td>
<td>Modular Gröbner Bases for Inhomogeneous Ideals</td>
<td>185</td>
</tr>
<tr>
<td>22</td>
<td>Reconstructed Bases might not be Gröbner</td>
<td>185</td>
</tr>
<tr>
<td>23</td>
<td>Evaluate $[vH02]$ against $[ASZ00]$</td>
<td>206</td>
</tr>
<tr>
<td>24</td>
<td>Better Choice of Best’ Prime</td>
<td>206</td>
</tr>
<tr>
<td>25</td>
<td>Low-degree Factorization</td>
<td>208</td>
</tr>
<tr>
<td>26</td>
<td>$p$-adic Gröbner bases</td>
<td>214</td>
</tr>
<tr>
<td>27</td>
<td>Algebraic Numbers Reviewed</td>
<td>218</td>
</tr>
<tr>
<td>28</td>
<td>Crossings of factored polynomials</td>
<td>287</td>
</tr>
</tbody>
</table>
Preface

This text is under active development, especially due to comments from colleagues, notably the Bath Mathematical Foundations seminar that listened to an explanation of section 4.4 and students on the course CM30070 — Computer Algebra at the University of Bath. David Wilson has been a most helpful recorder of errors during this, and he and Matthew England have signalled several problems. David Stoutemyer has been a very helpful reader. I am grateful to John Abbott (Genoa) for the polynomial \( f \) on page 190, and for the encouragement of people at CICM 2010 to write Chapter 8. I am grateful to Professor Vorobjov for the material in Excursus B.4.3. Professor Sergei Abramov (Russian Academy of Sciences) has been a helpful reader. As always when I write, I am grateful to David Carlisle for his \TeX\nical wisdom.

It is probably best cited as “J.H. Davenport, Computer Algebra. \url{http://staff.bath.ac.uk/masjhd/JHD-CA.pdf}” with a date.

Mathematical Prerequisites

This book has been used to support teaching at the University of Bath to both Mathematics and Computer Science Students. The main requirement is “mathematical maturity” rather than specific facts: some specific pre-requisites are listed in Table 1. If the students are not familiar with the \( O \) notation, it would be as well for the lecturer to explain this, at whatever level of sophistication is appropriate: this book is pretty unsophisticated in its use, and all that is needed is in section 1.4.

<table>
<thead>
<tr>
<th>Chapter(s)</th>
<th>Subjects</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2.1</td>
<td>Matrices</td>
</tr>
<tr>
<td>4.9</td>
<td>Modular arithmetic</td>
</tr>
<tr>
<td>7</td>
<td>Calculus</td>
</tr>
</tbody>
</table>

Table 1: Specific Prerequisites
Changes in Academic Year 2014–15

29.9.2014 Initial version published.

11.10.2014 Updated Section 1.1.1 (minor). The web site at [Ano07] had been trashed, so replaced by a reference into Way Back. Added section 1.5.1 and associated online resources. Improved ‘geobuckets’ text (page 39).

20/10/2014 Fixed description of Maple polynomials (section 2.1.7.3) following a student’s comment.

20/10/2014 Fixed $56$ to define $k$ (was $d$, but changed for consistency with multivariate case). Tidied (2.21) and (2.22). Strengthened Observation 2 to refer to coefficients growth as well.

31/10/2014 Added Proposition 25 (not examinable in 2014/15) nd Corollary 3 (stated in lectures).

5/11/2014 Make Algorithm 4 into a separate Figure 2.8 and describe as subresultant p.r.s. Also add index entries. Clarified Theorem 7.

15/11/2014 Make $\mathbb{Z} + \mathbb{Z}\sqrt{-5}$ example into a free-standing Example 2. Add Example 3. Added text at (new) Notation 24 et seq. Added new Section A.6 by popular demand.

22/11/2014 Added complexity analysis in Section 3.2.4. Added new text to Sections 3.3.8 (minor) and 3.3.9 (more substantial) and extra examples on the Web. Completed section 4.1.1 and 4.1.2, added 4.1.4. Appendix: added Proposition 93.


2/12/2014 Moved Observation 10 into section 4.2.7.

3/12/2014 Fixed an ambiguity in the $L := \text{part of Algorithm 11}$ thanks to student who reported it.

15/12/2014 Improved sections 3.5.5 and 5.3.2. Definitive version for Jan 2015 examination.

18/12/2014 Fixed bug in (3.24) and cite [AM01]. Knock-on effects for discussion after Proposition 61. Added Section 5.9.3.

25/12/2014 Substantial additions to Chapter 5. Extra text around Definition 18.

26/12/2014 Expanded section 3.1.10 added section 3.5.7. Also text at Proposition 9.

29/12/2014 More in section 3.5.7.
Chapter 1

Introduction

Computer algebra, the use of computers to do algebra rather than simply arithmetic, is almost as old as computing itself, with the first theses [Kah53, Nol53] dating back to 1953. Indeed it was anticipated from the time of Babbage, when Ada Augusta, Countess of Lovelace, wrote

WE MAY SAY MOST APTLY THAT THE ANALYTICAL ENGINE WEAVES ALGEBRAICAL PATTERNS JUST AS THE JACQUARD LOOM WEAVES FLOWERS AND LEAVES. [Ada43]

In fact, it is not even algebra for which we need software packages, computers by themselves can’t actually do arithmetic: only a limited subset of it. If we ask Excel\(^1\) to compute \(e^{\pi\sqrt{163}} - 262537412640768744\), we will be told that the answer is 256. More mysteriously, if we go back and look at the formula in the cell, we see that it is now \(e^{\pi\sqrt{163}} - 262537412640768800\). In fact, 262537412640768744 is too large a whole number (or integer, as mathematicians say) for Excel to handle, and it has converted it into floating-point (what Excel terms “scientific”) notation. Excel, or any other software using the IEEE standard [IEE85] representation for floating-point numbers, can only store them to a given accuracy, about 16 decimal places.\(^2\) In fact, it requires twice this precision to show that \(e^{\pi\sqrt{163}} \neq 262537412640768744\). Since \(e^{\pi\sqrt{163}} = (-1)^{\sqrt{-163}}\), it follows from deep results of transcendental number theory [Bak75], that not only is \(e^{\pi\sqrt{163}}\) not an integer, it is not a fraction (or rational number), nor even the solution of a polynomial equation with integer coefficients: essentially it is a ‘new’ number.

\(^1\)Or any similar software package.

\(^2\)We say ‘about’ since the internal representation is binary, rather than decimal.

\(^3\)In fact, Excel is more complicated even than this, as the calculations in this table show.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>...</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>...</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>10^i</td>
<td>10</td>
<td>100</td>
<td>1000</td>
<td>1...0</td>
<td>1...0</td>
<td>10^11</td>
<td>10^12</td>
<td>...</td>
<td>10^15</td>
<td>10^16</td>
</tr>
<tr>
<td>b</td>
<td>a-1</td>
<td>9</td>
<td>99</td>
<td>999</td>
<td>9999</td>
<td>9...9</td>
<td>10^11</td>
<td>10^12</td>
<td>...</td>
<td>10^15</td>
<td>10^16</td>
</tr>
<tr>
<td>c</td>
<td>a-b</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>...</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

We can see that the printing changes at 12 decimal digits, but that actual accuracy is not lost until we subtract 1 from 10^16.
Definition 1 A number $n$, or more generally any algebraic object, is said to be transcendental over a ring $R$ if there is no non-zero polynomial $p$ with coefficients in $R$ such that $p(n) = 0$.

With this definition, we can say that $e^{\pi \sqrt{163}}$ is transcendental over the integers. Transcendence is a deep mathematical question with implications for the decidability of computer algebra, see page 263.

We will see throughout this book (for an early example, see page 58) that innocent-seeming problems can give rise to numbers far greater than one would expect. Hence there is a requirement to deal with numbers larger, or to greater precision, than is provided by the hardware manufacturers whose chips underlie our computers. Practically every computer algebra system, therefore, has a package for manipulating arbitrary-sized integers (so-called bignums) or real numbers (bigfloats). These arbitrary-sized objects, and the fact that mathematical objects in general are unpredictably sized, means that computer algebra systems generally need sophisticated memory management, generally garbage collection (see [JHM11] for a general discussion of this topic).

But the fact that the systems can deal with large numbers does not mean that we should let numbers increase without doing anything. If we have two numbers with $n$ digits, adding them requires a time proportional to $n$, or in more formal language (see section 1.4) a time $O(n)$. Multiplying them requires a time $O(n^2)$, and calculating a g.c.d., which is fundamental in the calculation of rational numbers, requires $O(n^3)$, or $O(n^2)$ with a bit of care. This implies that if the numbers become 10 times longer, the time is multiplied by 10, or by 100, or by 1000. So it is always worth reducing the size of these integers. We will see later (an early example is on page 143) that much ingenuity has been well-spent in devising algorithms to compute “obvious” quantities by “non-obvious” ways which avoid, or reduce, the use of large numbers. The phrase intermediate expression swell is used to denote the phenomenon where intermediate quantities are much larger than the input to, or outputs from, a calculation.

Notation 1 We write algorithms in an Algol-like notation, with the Rutishauser symbol := to indicate assignment, and = (as opposed to C’s ==) to indicate the equality predicate. We use indentation to indicate grouping, rather than clutter the text with begin . . . end. Comments are introduced by the # character, running to the end of the line.

---

4In principle, $O(n \log n \log \log n)$ is enough [AHU74, Chapter 8], but no computer algebra system routinely uses this, for it is more like $20n \log n \log \log n$. However, most systems will use ‘Karatsuba arithmetic’ (see [KO63] and section B.3), which takes $O(n^{\log_2 3} \approx n^{1.585})$, once the numbers reach an appropriate length, often 16 words [SGV94].

5In principle, $O(n \log^2 n \log \log n)$ [AHU74, Chapter 8], but again no system uses it routinely, but this combined with Karatsuba gives $O(n^{\log_2 3} \log n)$ (section B.3.6), and this is commonly used.

6An idea which was tried in Axiom [JS92], but which turns out to be better in books than in real life.
1.1 History and Systems

The first recorded use of computers to do computations of the sort we envisage was in 1951 [MW51], where \( (2^{127} - 1)^2 - 1 \), a 79-digit number, was shown to be prime. In 1952 the great mathematician Emil Artin had the equally great computer scientist John von Neumann perform an extensive calculation relating to elliptic curves on the MANIAC computer [ST92, p. 119]. In 1953, two theses [Kah53, Nol53] kicked off the ‘calculus’ side of computer algebra with programs to differentiate expressions. In the same year, [Has53] showed that algorithms in group theory could be implemented on computers.

1.1.1 The ‘calculus’ side

The initial work [Kah53, Nol53] consisted of programs to do one thing, but the focus soon moved on to ‘systems’, capable of doing a variety of tasks. One early one was Collins’ system SAC [Col71], written in Fortran. Its descendants continue in SAC-2 [Col85] and QEPCAD [Bro03]. Of course, computer algebra systems can be written in any computer language, even Cobol [fHN76]. However, many of the early systems were written in LISP, largely because of its support for garbage collection and large integers. The group at M.I.T., very influential in the 1960s and 70s, developed Macsyma [MF71, PW85]. This system now exists in several versions [Ano07], even on mobile ‘phones’. At about the same time, Hearn developed Reduce [Hea05], and shortly after a group at IBM Yorktown Heights produced SCRATCHPAD [GJY75]. This group then produced Axiom [JS92], a system that attempted to match the generality of mathematics with ‘generic programming’ to allow algorithms to be programmed in the generality in which they are conventionally (as in this book) stated, e.g. “polynomials over a ring” as in Definition 19.

These were, on the whole, very large software systems, and attempts were made to produce smaller ones. muMATH [RS79] and its successor Derive [RS92] were extremely successful systems on the early PC, and paved the way for the computer algebra facilities of many high-end calculators. Much of the work on “compact computer algebra” is described in [Sto11b].

Maple [CGGG83] pioneered a ‘kernel+library’ design.

The basic Maple system, the *kernel*, is a relatively small collection of compiled C code. When a Maple session is started, the entire kernel is loaded. It contains the essential facilities required to run Maple and perform basic mathematical operations. These components include the Maple programming language interpreter, arithmetic and simplification routines, print routines, memory management facilities, and a collection of fundamental functions. [MGH+03, p. 6]
1.1.2 The ‘group theory’ side

Meanwhile, those interested in computation group theory, and related topics, had not been idle. One major system developed during the 1970s/80s was CAYLEY [BC90]. This team later looked at Axiom, and built the system MAGMA [BCM94], again with a strong emphasis on genericity. Another popular system is GAP [BL98], whose ‘kernel+library’ design was consciously [Neu95] inspired by Maple.

The reader may well ask ‘why two different schools of thought?’ The author has often asked himself the same question. The difference seems one of mathematical attitude, if one can call it that. The designer of a calculus system envisages it being used to compute an integral, factor a polynomial, multiply two matrices, or otherwise operate on a mathematical datum. The designer of a group theory system, while he will permit the user to multiply, say, permutations or matrices, does not regard this as the object of the system: rather the object is to manipulate whole groups (etc.) of permutations (or matrices, or ...), i.e. a mathematical structure.

1.1.3 A synthesis?

While it is too early to say that the division has been erased, it can be seen that MAGMA, for example, while firmly rooted in the group-theory tradition, has many more ‘calculus like’ features. Conversely, the interest in polynomial ideals, as in Proposition 33, means that systems specialising in this direction, such as SINGULAR [Sch03b] or COCOA [GN90], use polynomial algorithms, but the items of interest are the mathematical structures such as ideals rather than the individual polynomials.

1.2 Expansion and Simplification

These two words, or the corresponding verbs ‘expand’ and ‘simplify’ are much used (abused?) in computer algebra, and a preliminary explanation may be in order. Computers of course, do not deal in mathematical objects, but, ultimately, in certain bit patterns which are representations of mathematical objects. Just as in ink on paper, a mathematical object may have many representations, just as most of us would say that \( x + y \) and \( y + x \) are the same mathematical object.

Definition 2 A correspondence \( f \) between a class \( O \) of objects and a class \( R \) of representations is a representation of \( O \) by \( R \) if each element of \( O \) corresponds to one or more elements of \( R \) (otherwise it is not represented) and each element of \( R \) corresponds to one and only one element of \( O \) (otherwise we do not know which element of \( O \) is represented). In other words “is represented by”, is the inverse of a surjective function \( f \) or “represents” from a subset of \( R \) (the “legal representations”) to \( O \).
1.2. EXPANSION AND SIMPLIFICATION

**Notation 2** When discussing the difference between abstract objects and computer representations, we will use mathematical fonts, such as $x$, for the abstract objects and typewriter font, such as $x$, for the representations.

Hence we could represent any mathematical objects, such as polynomials, by well-formed strings of indeterminates, numbers and the symbols $+,-,\times,(,)$. The condition “well-formed” is necessary to prevent nonsense such as $)$x+++1y$, and would typically be defined by some finite grammar [ALSU07, Section 2.2]. With no simplification rules, such a representation would regard $x-x$ as just that, rather than as zero.

**Definition 3** A representation of a monoid (i.e. a set with a 0, and an addition operation) is said to be normal if the only representation of the object 0 is 0.

If we have a normal representation $f$, then we can tell if two mathematical objects $a$ and $b$, represented by $a$ and $b$, are equal by computing $a-b$: if this is zero (i.e. 0), then $a$ and $b$ are equal, while if this is not zero, they must be unequal. However, this is an inefficient method, to put it mildly.

**Observation 1** Normal representations are very important in practice, since many algorithms contain tests for zero/non-zero of elements. Sometimes these are explicit, as in Gaussian elimination of a matrix, but often they are implicit, as in Euclid’s Algorithm (Algorithm 2, page 54), where we take the remainder after dividing one polynomial by another, which in turn requires that we know the leading non-zero coefficient of the divisor.

**Definition 4** A representation is said to be canonical if every object has only one representation, i.e. $f$ is a bijection, or 1–1 mapping.

With a canonical representation, we can say that two objects “are the same if, and only if, they look the same”. For example, we cannot have both $(x+1)^2$ and $x^2+2x+1$, since $(x+1)^2 = x^2 + 2x + 1$.

It is possible to build any normal representation into a locally canonical one, using an idea due to [Bro69]. We store every computed (top-level) expression $e_1,e_2,...$, and for a new expression $E$, we compute the normal form of every $E-e_i$. If this is zero, then $E = e_i$, so we return $e_i$, otherwise $E$ is a new expression to be stored. This has various objections, both practical (the storage and computation time) and conceptual (the answer printed depends on the past history of the session), but nevertheless is worth noting.

**Definition 5 ([Sto11a, Definition 3])** A candid expression is one that is not equivalent to an expression that visibly manifests a simpler expression class.

In particular, if the “simpler class” is $\{0\}$, “candid” means the same as normal, but the concept is much more general, and useful. In particular, if a candid expressions contains a variable $v$, then it really depends on $v$. Untyped systems such as Maple and Macsyma could be candid in this sense⁹, but typed systems like

⁹See section 2.2.2 for other aspects of candidness.
Chapter 1. Introduction

Axiom have more problems in this area, since subtracting two polynomials in \( v \) will produce a polynomial in \( v \), even if the result in a particular case is, say, \( v - (v - 1) = 1 \), which doesn’t in fact depend on \( v \).

Candidness depends on the class of expressions considered as ‘simpler’. Consider

\[ F : x \mapsto x - \sqrt{x^2} \quad (1.1) \]

(viewed as a function \( \mathbb{R} \to \mathbb{R} \)). It might be tempting to consider \( F \) to be zero, but in fact \( F(-1) = -2 \). In fact, \( F \) can also be written as \( x \mapsto x - |x| \) or \( x \mapsto \begin{cases} 0 & x \geq 0 \\ 2x & x \leq 0 \end{cases} \). Is (1.1) therefore candid? It depends on the definition of “simpler”.

Since practically everyone would agree that \( \{0\} \) is a simple class, candid expressions are automatically normal, but need not be canonical (see (1.1)), so this definition provides a useful intermediate ground, which often coincides with naïve concepts of “simpler”. We refer the reader to [Sto11a] for further, very illustrative, examples, to section 2.2.2 for a discussion of candidness for rational functions, and to sections 7.3.2 and 7.3.3 for a discussion of candidness in integration theory.

We should note that ‘candidness’ is not always achievable. One possible requirement would be “if the final value is real, then no complex numbers should be involved”, but this is not generally possible: at (3.10) we see that the three real roots of \( x^3 - x \) can only be computed via complex numbers. In this case, they have simple expressions (0, +1, and −1), but that is not true in general: consider \( x^3 - x - \frac{1}{1000} \), whose roots are

\[
\begin{align*}
&\frac{1}{20} \left( \sqrt[3]{108 + 12i\sqrt{11999919}} + 20 \frac{1}{\sqrt[3]{108 + 12i\sqrt{11999919}}} \right), \\
&-\frac{1}{20} \left( \sqrt[3]{108 + 12i\sqrt{11999919}} - 10 \frac{1}{\sqrt[3]{108 + 12i\sqrt{11999919}}} \right), \\
&\frac{i\sqrt{3}}{20} \left( \frac{1}{\sqrt[3]{108 + 12i\sqrt{11999919}}} - 200 \frac{1}{\sqrt[3]{108 + 12i\sqrt{11999919}}} \right), \\
&-\frac{1}{20} \left( \sqrt[3]{108 + 12i\sqrt{11999919}} - 10 \frac{1}{\sqrt[3]{108 + 12i\sqrt{11999919}}} \right), \\
&\frac{i\sqrt{3}}{20} \left( \frac{1}{\sqrt[3]{108 + 12i\sqrt{11999919}}} - 200 \frac{1}{\sqrt[3]{108 + 12i\sqrt{11999919}}} \right).
\end{align*}
\]

1.2.1 A Digression on “Functions”

In colloquial mathematics, the word “function” has many uses and meanings, and one of the distinctive features of computer algebra is that it plays on these various uses.

In principle, (pure) mathematics is clear on the meaning of “function”.

On dit qu’un graphe \( F \) est un graphe fonctionnel si, pour tout \( x \), il existe au plus un objet correspondant à \( x \) par \( F \) (I, p. 40). On dit qu’une correspondance \( f = (F,A,B) \) est une fonction si son graphe \( F \) est un graphe fonctionnel, et si son ensemble de départ \( A \) est égal à son ensemble de définition \( \text{pr}_1 F \) [pr1] is “projection on the first
1.2. EXPANSION AND SIMPLIFICATION

component”].

The present author’s loose translation is as follows.

We say that a graph (i.e. a subset of \( A \times B \)) \( F \) is a functional graph if, for every \( x \), there is at most one pair \((x, y)\) \( \in F \) [alternatively \((x, y_1), (x, y_2) \in F \) implies \( y_1 = y_2 \)]. We then say that a correspondence \( f = (F, A, B) \) is a function if its graph \( F \) is a functional graph and the starting set \( A \) is equal to \( \text{pr}_1 F = \{x \mid \exists y (x, y) \in F\} \).

So for Bourbaki a function includes the definition of the domain and codomain, and is total and single-valued.

**Notation 3** We will write \((F, A, B)_B \) for such a function definition. If \( C \subset A \), we will write \((F, A, B)_B \mid C \), “\((F, A, B)_B \) restricted to \( C \)”, for \((G, C, B)_B \) where \( G = \{(x, y) \in F \mid x \in C\} \).

Hence the function \( \sin \) is, formally, \( \{(x, \sin x) : x \in \mathbb{R}\}, \mathbb{R}, \mathbb{R}\}_B \), or, equally possibly, \( \{(x, \sin x) : x \in \mathbb{C}\}, \mathbb{C}, \mathbb{C}\}_B \). But what about \( x + 1 \)? We might be thinking of it as a function, e.g. \( \{(x, x+1) : x \in \mathbb{R}\}, \mathbb{R}, \mathbb{R}\}_B \), but equally we might just be thinking of it as an abstract polynomial, a member of \( \mathbb{Q}[x] \). In fact, the abstract view can be pursued with \( \sin \) as well, as seen in Chapter 7, especially Observation 17, where \( \sin \) is viewed as \( \frac{1}{2i} (\theta - 1/\theta) \in \mathbb{Q}(i, x, \theta|\theta' = i\theta) \). The relationship between the abstract view and the Bourbakist view is pursued further in Chapter 8.

1.2.2 Expansion

This word is relatively easy to define, as application of the distributive law, as seen in the first bullet point of Maple’s description.

- The expand command distributes products over sums. This is done for all polynomials. For quotients of polynomials, only sums in the numerator are expanded; products and powers are left alone.
- The expand command also expands most mathematical functions, including . . . .

In any given system, the precise meaning of expansion depends on the underlying polynomial representation used (recursive/distributed — see page 41, so Maple, which is essentially distributed, would expand \( x(y+1) \) into \( xy + x \), while Reduce, which is recursive, would not, but would expand \( y(x+1) \) into \( xy + y \), since its default ordering is ‘\( x \) before \( y \)’.

Expansion can, of course, cause exponential blow-up in the size of the expression: consider \((a + b)(c + d)(e + f) \ldots \) or \(\sin(a + b + c + \ldots)\). The second bullet point of Maple’s description can lead to even more impressive expansion, as in
22

CHAPTER 1. INTRODUCTION

expand(BesselJ(4,t)^3);
(just where did the number 165888 come from?) or
expand(WeierstrassP(x+y+z,2,3));

1.2.3 Simplification

This word is much used in algebra, particularly at the school level, and has
been taken over by computer algebra, which has thereby committed the sin of
importing into a precise subject a word without a precise meaning.

Looking at the standard textbooks on Computer Algebra Systems
(CAS) leaves one even more perplexed: it is not even possible to find
a proper definition of the problem of simplification [Car04].

Let us first consider a few examples.

1. Does \(x^2 - 1\) simplify to \(x + 1\)? For most people, the answer would be
   ‘yes’, but some would query “what happens when \(x = 1\)”, i.e. would
   ask whether we are dealing with abstract formulae, or representations of
   functions. This is discussed further for rational functions on pages 51 and
   252 and in item 5 below.

2. Assuming the answer to the previous question is ‘yes’, does \(x^{1000} - 1\)
   simplify to \(x^{999} + \cdots + 1\)? Here the fraction is much shorter than the explicit
   polynomial, and we have misled the reader by writing \(\cdots\) her\textsuperscript{10}.

3. Does \(\sqrt{1 - x} + x\) simplify to \(\sqrt{1 - x^2}\)? Assuming the mathematical
   validity, which is not trivial [BD02], then the answer is almost certainly
   yes, since the second operations involves fewer square roots than the first.

4. Does \(\sqrt{x - 1} + \sqrt{x + 1}\) simplify to \(\sqrt{x^2 - 1}\)? This might be thought to be
   equivalent to the previous question, but consider what happens when \(x =
   -2\). The first one simplifies to \(\sqrt{-3} - 1 = i\sqrt{3} - i = -\sqrt{3}\), while the
   second simplifies to \(\sqrt{3}\). Note that evaluations result in real numbers,
   though they proceed via complex ones. Distinguishing this case from
   the previous one is a subject of active research [BBDP07, CDJW00].

5. Assume we are working modulo a prime \(p\), i.e. in the field \(\mathbb{F}_p\). Does
   \(x^p - x\) simplify to 0? As polynomials, the answer is no, but as functions
   \(\mathbb{F}_p \rightarrow \mathbb{F}_p\), the answer is yes, by Fermat’s Little Theorem. Note that, as
   functions \(\mathbb{F}_{p^2} \rightarrow \mathbb{F}_{p^2}\), say, the answer is no.

In terms of Notation 3 we can say that

\[
\{(x, x^p - x) : x \in \mathbb{F}_p\}, \mathbb{F}_p, \mathbb{F}_p\}_{B} = \{(x, 0) : x \in \mathbb{F}_p\}, \mathbb{F}_p, \mathbb{F}_p\}_{B},
\]

\textsuperscript{10}The construct \(\cdots\) is very common in written mathematics, but has had almost (but see
SS06) no analysis in the computer algebra literature.
but
\[
\left\{ (x, x^p - x) : x \in \mathbb{F}_{p^2}, \mathbb{F}_{p^2}, \mathbb{F}_{p^2} \right\} \neq \left\{ (x, 0) : x \in \mathbb{F}_{p^2}, \mathbb{F}_{p^2}, \mathbb{F}_{p^2} \right\}.
\]

Now we give a few illustrations of what simplification means to different audiences.

**Teachers** At a course on computer algebra systems for teachers of the French ‘concours’, among the most competitive mathematics examinations in the western world, there was a round table on the meaning of simplification. Everyone agreed that the result should be ‘mathematically equivalent’, though it was less clear, prompted by example 1, exactly what this meant. The response to example 2 was along the lines of “well, you wouldn’t ask such a question”. The author wishes he had had examples 3 and 4 to hand at the time!

The general consensus was that ‘simplification’ meant ‘give me the answer I want’. This answer is not effective, in the sense that it cannot be converted into a set of rules.

Whether this is an appropriate approach to pedagogy is outside the scope of this book, but we refer the reader to [BDS09, BDS10].

**Moses** [Mos71] This is a seminal paper, but describes approaches to simplification rather than defining it. Inasmuch as it does define it, it talks about ‘utility for further operations’, which again is not effective as a definition for top-level simplification, though it’s relevant for intermediate operations. However, the principle is important, since a request to factor would find the expression \(x^{999} + \cdots + 1\) appropriate\(^{11}\), whereas \(x^{1000} - 1 = x - 1\) is in a field, and factorisation is not a relevant question.

**Carette** [Car04] He essentially defines simplification in terms of the length of the result, again insisting on mathematical equivalence. This would regard examples 1 (assuming \(Q(x)\), so that the expressions were mathematically equivalent) and 3 as simplifications, but not 2 since the expression becomes longer, or 4 since we don’t have mathematical equivalence.

**Stoutemyer** [Sto11a] sets out 10 goals for simplification, one of which (Goal 6) is that

Default simplification should produce candid [Definition 5] results for rational expressions and for as many other classes as is practical. Default simplification should try hard even for classes where candidness cannot be guaranteed for all examples.

Candid expressions tend be the shorter than others (\(\frac{x^{10} - 1}{x - 1}\) etc. being an obvious family of counter-examples), so this view is relatively close to Carette’s, but not identical (see also page 52, item 2).

\(^{11}\)In fact, knowing the expression came from that quotient would be relevant [BD89] to factorisation algorithms, but that’s beside the point here.
Numerical Analysts  A numerical analyst would be shocked at the idea that $x^2 - y^2$ was ‘simpler’ than $(x + y)(x - y)$. He would instantly quote an example such as the following [Ham07].

For simplicity, assume decimal arithmetic, with perfect rounding and four decimal places. Let $x = 543.2$ and $y = 543.1$. Then $x^2$ evaluates to 295100 and $y^2$ evaluates to 295000, so $x^2 - y^2$ becomes 100, while $(x + y)(x - y)$ evaluates to 108.6, a perfect rounding of the true answer 108.63.

Furthermore, if we take $x = 913.2$ and $y = 913.1$, $x^2 - y^2$ is still 100, while the true result is 182.63.

This is part of the whole area of numerical stability: fascinating, but outside the scope of this text.

One principle that can be extracted from the above is “if the expression is zero, please tell me”: this would certainly meet both the teachers’ and Carette’s views. This can be seen as a call for simplification to return a normal form where possible [Joh71, Ric97].

Maple’s description of the ‘simplify’ command is as follows.

- The `simplify` command is used to apply simplification rules to an expression.
- The `simplify/expr` calling sequence searches the expression, `expr`, for function calls, square roots, radicals, and powers. It then invokes the appropriate simplification procedures.
- `symbolic` Specifies that formal symbolic manipulation of expressions is allowed without regard to the analytical issue of branches for multi-valued functions. For example, the expression `sqrt(x^2)` simplifies to `x` under the symbolic option. Without this option, the simplified result must take into account the different possible values of the (complex) sign of `x`.

Maple does its best to return a normal form, but can be fooled: for example

$$\text{RootOf} \left( x^4 + b x^2 + d \right) - \frac{1}{2} \sqrt{-2 b + 2 \sqrt{b^2 - 4 d}},$$

which is actually zero (applying figure 3.2), does not simplify to zero under Maple 11.

Because simplification may often require expansion, e.g. to take $(x-1)(x+1)$ to $x^2 - 1$, the two are often confused, and indeed both Macsyma and Reduce (internally) used `ratsimp` and `*simp` (respectively) to denote what we have called expansion.
1.2.4 An example of simplification

This section is inspired by an example in [Sto13]. Consider

\[
\begin{align*}
(\cos(x))^3 \sin(x) + \frac{1}{2} (\cos(x))^3 \sin(x) + 2 (\cos(x))^3 \cos(2x) \sin(x) + \\
\frac{1}{2} (\cos(x))^3 \cos(4x) \sin(x) - \frac{3}{2} \cos(x) (\sin(x))^3 - \\
2 \cos(x) \cos(2x) (\sin(x))^3 - \frac{1}{2} \cos(x) \cos(4x) (\sin(x))^3.
\end{align*}
\]

(1.2)

Typing this into Maple simply collects terms, giving

\[
\begin{align*}
\frac{3}{8} (\cos(x))^3 \sin(x) + 2 (\cos(x))^3 \cos(2x) \sin(x) + \\
\frac{1}{2} (\cos(x))^3 \cos(4x) \sin(x) - \frac{3}{2} \cos(x) (\sin(x))^3 - \\
2 \cos(x) \cos(2x) (\sin(x))^3 - \frac{1}{2} \cos(x) \cos(4x) (\sin(x))^3.
\end{align*}
\]

(1.3)

\text{combine}(\%, \text{trig}), \text{i.e. using the multiple angle formulae to replace trigonometric powers by sin/cos of multiples of the angles, gives}

\[
\frac{3}{8} \sin(4x) + \frac{1}{4} \sin(2x) + \frac{1}{4} \sin(6x) + \frac{1}{16} \sin(8x).
\]

(1.4)

\text{expand}(\%, \text{trig}) (\text{i.e. using the multiple angle formulae in the other direction})
gives

\[
4 \sin(x) (\cos(x))^7 - 4 (\cos(x))^5 (\sin(x))^3,
\]

(1.5)

which is also given by Maple’s \text{simplify} applied to (1.3) (\text{simplify} applied to (1.4) leaves it unchanged). Mathematica’s \text{FullSimplify} gives

\[
2 (\sin(3x) - \sin(x)) (\cos(x))^5.
\]

(1.6)

However, an even simpler form is found by Eureqa\footnote{http://creativemachines.cornell.edu/eureqa}:

\[
(\cos(x))^4 \sin(4x).
\]

(1.7)

We note that (1.4) and (1.5) are the results of algorithmic procedures, pushing identities in one direction or the other, applied to (1.3), while (1.6) and (1.7) are half-way positions, which happen to be shorter than the algorithmic results.

1.3 Algebraic Definitions

In this section we give some classic definitions from algebra, which we will return to throughout this book. Other concepts are defined as they occur, but these ones are assumed.

\textbf{Definition 6} A set \( R \) is said to be a ring if it has two binary operations \( + \) and \( * \), a unary operation \( - \) and a distinguished element \( 0 \), such that, for all \( a, b \) and \( c \) in \( R \):

\footnote{However, we should note that Maple (resp. Mathematica) has proved that (1.5) (resp. (1.6)) is equivalent to (1.2), whereas Eureqa merely claims that (1.7) fits the same data points as a finite sample of (1.2). Nevertheless, Eureqa’s capability to find a short equivalent in essentially Carette’s sense [Car04] is impressive, and in any case, knowing what to prove, any algebra system can prove equivalence.}
CHAPTER 1. INTRODUCTION

1. \( a + (b + c) = (a + b) + c \) (associativity of +);
2. \( a \ast (b \ast c) = (a \ast b) \ast c \) (associativity of \( \ast \));
3. \( a + b = b + a \) (commutativity of +);
4. \( a + (-a) = 0 \);
5. \( a + 0 = a \);
6. \( a \ast (b + c) = (a \ast b) + (a \ast c) \) (distributivity of \( \ast \) over +);
6' \( (b + c) \ast a = b \ast a + c \ast a \) (right-distributivity);
7. \( a \ast b = b \ast a \) (commutativity of \( \ast \)).

Not every text includes the last clause, and they would call a ‘commutative ring’ what we have called simply a ‘ring’. In the absence of the last clause, we will refer to a ‘non-commutative ring’. 6' is unnecessary if we have commutativity, of course.

**Definition 7** If \( R \) is a (possibly non-commutative) ring and \( \emptyset \neq I \subseteq R \), then we say that \( I \) is a (left-)ideal of \( R \), written \( I \triangleleft R \), if the following two conditions are satisfied:

(i) \( \forall f, g \in I, f - g \in I \),

(ii) \( \forall f \in R, g \in I, fg \in I \).

**Notation 4** If \( S \subseteq R \), we write \( (S) = \bigcup_{n \in \mathbb{N}} \{ \sum_{i=1}^{n} f_i g_i \mid f_i, g_i \in R, f_i \in S \} \): the set of all linear combinations (over \( R \)) of the elements of \( S \). We tend to abuse notation and write \( (a_1, \ldots, a_k) \) instead of \( \{(a_1, \ldots, a_k)\} \). This is called the (left-) ideal generated by \( S \), and clearly is a left-ideal.

There are also concepts of right ideal and two-sided ideal, but all concepts agree in the case of commutative rings. Non-trivial ideals (the trivial ideals are \( \{0\} \) and \( R \) itself) exist in most rings: for example, the set of multiples of \( m \) is an ideal in \( \mathbb{Z} \).

**Proposition 1** If \( I \) and \( J \) are ideals, then \( I + J = \{ f + g : f \in I, g \in J \} \) and \( IJ = \{ fg : f \in I, g \in J \} \) are themselves ideals.

**Definition 8** A ring is said to be noetherian, or to satisfy the ascending chain condition if every ascending chain \( I_1 \subseteq I_2 \subseteq \cdots \) of ideals is finite.

**Theorem 1 (Noether)** If \( R \) is a commutative noetherian ring, then so is \( R[x] \) (Notation 10, page 34).

**Corollary 1** If \( R \) is a commutative noetherian ring, then so is \( R[x_1, \ldots, x_n] \).

---
13 We write \( f - g \in I \) since then \( 0 = f - f \in I \), and then \( f + g = f - (0 - g) \in I \).
1.3. ALGEBRAIC DEFINITIONS

Definition 9 A ring $R$ is said to be an integral domain if, in addition to the conditions above, there is a neutral element 1 such that $1 \ast a = a$ and, whenever $a \ast b = 0$, at least one of $a$ and $b$ is zero.

Another way of stating the last is to say that $R$ has no zero-divisors (meaning none other than zero itself).

Definition 10 An element $u$ of a ring $R$ is said to be a unit of $R$ if there is an element $u^{-1} \in R$ such that $u \ast u^{-1} = 1$. $u^{-1}$ is called the inverse of $u$. Note that the context $R$ matters: 2 is a unit in $\mathbb{Q}$ (with inverse $1/2$), but not in $\mathbb{Z}$.

Definition 11 If $a = u \ast b$ where $u$ is a unit, we say that $a$ and $b$ are associates. Proposition 2 If $a$ and $b$ are associates, and $b$ and $c$ are associates, then $a$ and $c$ are associates. In other words, being associates is an equivalence relation.

For the integers, $n$ and $-n$ are associates, whereas for the rational numbers, any two non-zero numbers are associates.

Definition 12 An ideal $I$ of the form $(f)$, i.e. such that every $h \in I$ is $gf$ for some $g \in R$, is called principal. If $R$ is an integral domain such that every ideal $I$ is principal, then $R$ is called a principal ideal domain, or P.I.D.

Classic P.I.D.s are the integers $\mathbb{Z}$, and polynomials in one variable over a field. Inside a principal ideal domain, we have the standard concept of a greatest common divisor (formally defined in Definition 27).

Proposition 3 Let $R$ be a P.I.D., $a, b \in R$ and $(a, b) = (g)$. Then $g$ is a greatest common divisor of $a$ and $b$, in the sense that any other common divisor divides $g$, and $g = ca + db$ for $c, d \in R$.

It is possible to have g.c.d.s without being a P.I.D.: common examples are $\mathbb{Z}[x]$ (where the ideal $(2, x)$ is not principal) and $\mathbb{Q}[x, y]$ (where the ideal $(x, y)$ is not principal).

Definition 13 If $F$ is a ring in which every non-zero element is a unit, $F$ is said to be a field.

The “language of fields” therefore consists of two constants (0 and 1), four binary operations (+, −, ∗ and ÷) and two unary operations (− and $-1$), which can be replaced by the binary operations combined with the constants. The rational numbers and real numbers are fields, but the integers are not. For any $m$, the integers modulo $m$ are a ring, but only if $m$ is prime do they form a field. The only ideals in a field are the trivial ones.

Definition 14 If $R$ is an integral domain, we can always form a field from it, the so called field of fractions, consisting of all formal fractions $\frac{a}{b}$ : $a, b \in R, b \neq 0$, where $a/b$ is zero if and only if $a$ is zero. Addition is defined by $\frac{a}{b} + \frac{c}{d} = \frac{ad + bc}{bd}$, and multiplication by $\frac{a}{b} \ast \frac{c}{d} = \frac{ac}{bd}$. So $\frac{a}{b} = \frac{c}{d}$ if, and only if, $ad - bc = 0$.

\footnote{Strictly speaking, it consists of equivalence classes of formal fractions, under the equality we are about to define.}
In particular, the rational numbers are the field of fractions of the integers.

Definition 15 If \( F \) is a field, the characteristic of \( F \), written \( \text{char}(F) \), is the least positive number \( n \) such that \( 1 + \cdots + 1 = 0 \). If there is no such \( n \), we say that the characteristic is zero.

So the rational numbers have characteristic 0, while the integers modulo \( p \) have characteristic \( p \), as do, for example, the rational functions whose coefficients are integers modulo \( p \).

Proposition 4 The characteristic of a field, if non-zero, is always a prime.

1.3.1 Algebraic Closures

Some polynomial equations have solutions in a given ring/field, and some do not. For example, \( x^2 - 1 = 0 \) always has two solutions: \( x = -1 \) and \( x = 1 \). The reader may protest that, over a field of characteristic two, there is only one root, since \( 1 = -1 \). However, over a field of characteristic two, \( x^2 - 1 = (x - 1)^2 \), so \( x = 1 \) is a root with multiplicity two.

Definition 16 A field \( F \) is said to be algebraically closed if every polynomial in one variable over \( F \) has a root in \( F \).

Proposition 5 If \( F \) is algebraically closed, then every polynomial of degree \( n \) with coefficients in \( F \) has, with multiplicity, \( n \) roots in \( F \).

Theorem 2 ("Fundamental Theorem of Algebra") \( \mathbb{C} \), the set of complex numbers, is algebraically closed.

Definition 17 If \( F \) is a field, the algebraically closure of \( F \), denoted \( \overline{F} \) is the field generated by adjoining to \( F \) the roots of all polynomials over \( F \).

It follows from proposition \[79\] that the algebraic closure is in fact algebraically closed.

1.4 Some Complexity Theory

As is usual in computing we will use the so-called "Landau notation"\[16\] to describe the computing time (or space) of operations.

Notation 5 ("Landau") Let \( N \) be some measure of the size of a problem, generally the size of the input to a program, and \( f \) some function. If \( t(N) \) is

\[15\] The title is in quotation marks, since \( \mathbb{C} \) (and \( \mathbb{R} \)) are constructs of analysis, rather than algebra.

\[16\] Though apparently first introduced by [Bac94, p. 401]. See [Knu74].
1.4. SOME COMPLEXITY THEORY

the time taken, on a given hardware/software configuration, for a particular program to solve the hardest problem of size \( N \), we say that

\[
t(N) = O(f(N))
\]  

(1.8)

if \( \exists C \in \mathbb{R}, M : \forall N > M \quad t(N) < Cf(N) \).

Hardware tends to change in speed, but generally linearly, so \( O \)-notation is independent of particular hardware choices. If the program implements a particular algorithm, we will say that the algorithm has this \( O \) behaviour.

We will also use “soft \( O \)” notation.

**Notation 6 (“soft \( O \)”)** Let \( N \) be some measure of the size of a problem, and \( f \) some function. If \( t(N) \) is the time taken, on a given hardware/software configuration, for a particular program to solve the hardest problem of size \( N \), we say that

\[
t(N) = \tilde{O}(f(N))
\]  

(1.9)

if \( t(N) \) grows “almost no faster” than \( f(N) \), i.e. slower than \( f(N)^{1+\varepsilon} \) for any positive \( \varepsilon \): in symbols \( \forall \varepsilon > 0 \exists C \in \mathbb{R}, M \in \mathbb{N} : \forall N > M \quad t(N) < Cf(N)^{1+\varepsilon} \).

We should note that “\( = O \)” and “\( = \tilde{O} \)” should really be written with “\( \in \)” rather than “\( = \)” and this use of “\( \in \)” is not reflexive, symmetric or transitive. Also, many authors use \( \tilde{O} \) to mean “up to logarithmic factors”, which is included in our, more general, definition. [DL08].

The key results of complexity theory for elementary algorithms are that it is possible to multiply two \( N \)-bit integers in time \( \tilde{O}(N) \), and two degree-\( N \) polynomials with coefficients of bit-length at most \( \tau \) in time \( \tilde{O}(N\tau) \). [AHU83]

For matrix multiplication, the situation is more complicated: see Notation 40

**Definition 18 (Following [BFSS06])** We say that an algorithm producing output of size \( N \) is optimal if its running time is \( O(N) \), and almost optimal if its running time is \( \tilde{O}(N) \).

Addition of numbers and polynomials is therefore optimal, and multiplication almost optimal. Matrix multiplication is not known to be either.

However, the reader should be warned that \( \tilde{O} \) expressions are often far from the reality experienced in computer algebra, where data are small enough that the limiting processes in equations (1.8) and (1.9) have not really taken hold (see note B), or are quantised (in practice integer lengths are measured in words, not bits, for example). We therefore often use the phrase “classical arithmetic” to mean \( O(n^2) \) integer/polynomial multiplication, and \( O(n^3) \) matrix multiplication.

When it comes to measuring the intrinsic difficulty of a problem, rather than the efficiency of a particular algorithm, we need lower bounds rather than the upper bounds implied in (1.8) and (1.9).
CHAPTER 1. INTRODUCTION

Notation 7 (Lower bounds) Consider a problem \( P \), and a given encoding, e.g. “dense polynomials (Definition 23) with coefficients in binary”. Let \( N \) be the size of a problem instance, and \( C \) a particular computing paradigm, and way of counting operations. If we can prove that there is a \( c \) such that any algorithm solving this problem must take at least \( cf(N) \) operations on at least one problem instance of size \( N \), then we say that this problem has cost at least of the order of \( f(N) \), written

\[
P_C = \Omega(f(N)) \text{ or loosely } P = \Omega(f(N)). \quad (1.10)
\]

Again “=” really ought to be “\( \in \)”. In some instances (sorting is one of these) we can match upper and lower bounds.

Notation 8 (\( \Theta \)) If \( P_C = \Omega(f(N)) \) and \( P_C = O(f(N)) \), then we say that \( P_C \) is of order exactly \( f(N) \), and write \( P_C = \Theta(f(N)) \).

For example if \( C \) is the paradigm in which we only count comparison operations, sorting \( N \) objects is \( \Theta(N \log N) \).

Notation 9 (Further Abuse) We will sometimes abuse these notations further, and write, say, \( f(N) = 2^{O(N)} \), which can be understood as either of the equivalent forms \( \log_2 f(N) = O(N) \) or \( \exists C \in \mathbb{R}, M : \forall N > M f(N) < 2^{CN} \). Note that \( 2^{O(N)} \) and \( O(2^N) \) are very different things. \( 4^N = 2^{2N} \) is \( 2^{O(N)} \) but not \( O(2^N) \).

Open Problem 1 (Algebra of \( O \)) Manipulating such \( O \)-expressions, especially when they depend on several variables, can be very tedious as seen in [Col75, pp. 160–163]. Write a computer algebra package to simplify such expressions, so that, for example,

```
0simplify(O(N^3)+O(N^2));
```

would yield \( O(N^3) \).

1.4.1 Probabilistic Algorithms

The traditional definition of an algorithm is “a definite sequence of operations which terminates and produces the desired result” or words to that effect, so the title of this section is, taken literally, an oxymoron. Nevertheless, the concept is very useful, and can be made precise by inserting into that definition “(which may include calls to random number generators)”. We can then distinguish various kinds of probabilistic algorithms (where “fast” tends to mean “polynomial time”). In each case, “probably” refers to probability across the possible outputs of the random number generators.

Monte Carlo ("always fast/probably correct") The classic example here is the Miller–Rabin primality testing algorithm [Rab80].
Algorithm 1 (Miller–Rabin primality testing)

**Input:** a number $N$ of $n$-bits

**Output:** Either “$N$ is definitely composite” or “$N$ is probably prime”.

The algorithm picks a random $a \in [2, N - 1]$ and computes $a^{N-1}$ carefully. The running time, with classical arithmetic, is therefore $O(n^3)$. If $N$ is prime, the algorithm always outputs “$N$ is probably prime”. If $N$ is composite, the algorithm outputs “$N$ is probably prime” for at most $1/4$ of $a$-values.

**Las Vegas** (“always correct/probably fast”) If Algorithm $A$ is a Monte Carlo algorithm for problem $P$, and Algorithm $B$ is a fast verifier for correctness, then the process in Figure 1.1 will give a Las Vegas algorithm for problem $P$. It is the absence of such a verifier that means that Miller–Rabin primality testing (as opposed to the deterministic, but much more expensive AKS [AKS04] test) is only Monte Carlo.

**Atlantic City** (“probably fast/probably correct”) Again, the existence of a fast verifier for correctness would let the process in Figure 1.1 convert this to a Las Vegas algorithm. One example of an Atlantic City algorithm is given in Theorem 3 where again we lack a fast verifier, so the algorithm is “only” Atlantic City.

1.5 Some Maple

1.5.1 Maple polynomials

From Maple 17 onwards, Maple has two possible representations for polynomials.

**Original Expressions** An expression, whether or not it had been through `expand`, was an $n$-ary sum of $n$-ary power products with numerical coefficients. Uniqueness of power products (and hence collection of like terms [S]) on page 34 was enforced through hashing, rather than sorting, and hence summands and multiplicands could, and did, appear in any order.

**Sparse distributed** [MP12] This is as on page 41 and the order is total degree lexicographic (Section 3.3.3).
there’s a Maple worksheet demonstrating this at \url{http://staff.bath.ac.uk/masjhd/JHD-CA/MaplePoly.html}.

### 1.5.2 The RootOf construct

Maple uses this construct to indicate a solution of an (univariate) equation. It’s generally seen in the context of polynomial equations, as in Figure 1.2, where essentially nothing more can be said about these numbers other than they are the roots of the polynomial $z^5 - 5z^3 + 4z - 1$. Note that Maple indexes the result of \texttt{RootOf} according to the rules at \url{http://www.maplesoft.com/support/help/Maple/view.aspx?path=RootOf/indexed}.

### 1.5.3 The simplify command

This Maple command is been discussed elsewhere, especially in section 1.2.4. It is worth noting that simplification in this sense does not commute with substitution, and it can be argued [Zim07] that it is a ‘user-oriented’ command that should not be used inside Maple programs, where well-specified commands such as \texttt{expand} (Section 1.2.2), or more specific ones such as \texttt{combine(%, trig)} are very appropriate.
Chapter 2

Polynomials

Polynomials are fundamental to much of mathematics, even when the objects under discussion are apparently not polynomials, such as differential equations. Equally, polynomials underpin much of computer algebra. But what, precisely, are they?

2.1 What are polynomials?

There are numerous definitions. From our point of view, computer algebra, we will adopt the following definition for commutative polynomials, leaving non-commutative polynomials to be discussed in section 2.4.

Definition 19 (Polynomials) A (commutative) polynomial is built up from coefficients, which are assumed to form a ring (definition 6), and certain indeterminates (often called variables), by the algebraic operations of addition, subtraction and multiplication. These are subject to the following laws, where $a, b, c$ are polynomials, $m, n$ coefficients, and 0 and 1 certain distinguished coefficients.

1. $a + b = b + a$;
2. $(a + b) + c = a + (b + c)$;
3. $a + 0 = a$
4. $a + (-a) = 0$;
5. $a \ast b = b \ast a$;
6. $a \ast (b \ast c) = (a \ast b) \ast c$;

"Commutative" meaning $a \ast b = b \ast a$. Strictly speaking, we should also worry whether addition is commutative, i.e. whether $a + b = b + a$, but we will always assume that addition is commutative.
7. \( a \times 1 = a; \)
8. \( a \times (b + c) = (a \times b) + (a \times c); \)
9. \( m + n = m \oplus n; \)
10. \( m \times n = m \otimes n; \)

where we have used \( \oplus \) and \( \otimes \) to denote the operations of addition and multiplication on coefficients, which are assumed to be given to us.

The reader can think of the coefficients as being numbers, though they need not be, and may include other indeterminates that are not the “certain indeterminates” of the definition. However, we will use the usual ‘shorthand’ notation of 2 for \( 1 \oplus 1 \) etc. The associative laws (2 and 6 above) mean that addition and multiplication can be regarded as \( n \)-ary operations. A particular consequence of these rules is

\[
\text{8'} \quad m \times a + n \times a = (m \oplus n) \times a
\]

which we can think of as ‘collection of like terms’.

**Proposition 6** Polynomials over a ring form a ring themselves.

**Definition 20 (Free Algebra)** If it is the case that a polynomial is only zero if it can be deduced to be zero by rules 1–10 above, and the properties of \( \oplus \) and \( \otimes \), then we say that we have a free polynomial algebra.

Free algebras are common, but by no means the only one encountered in computer algebra. For examples, trigonometry is often encoded by regarding \( \sin \theta \) and \( \cos \theta \) as indeterminates, but subject to \( \sin^2 \theta + \cos^2 \theta = 1 \) [Sto77].

Notice what we have not mentioned: division and exponentiation.

**Definition 21 ((Exact) Division)** If \( a = b \times c \), then we say that \( b \) divides \( a \), and we write \( b \div a \).

Note that, for the moment, division is only defined in this context. We note that, if \( c \) is not a zero-divisor, \( b \) is unique.

**Definition 22 (Exponentiation)** If \( n \) is a natural number and \( a \) is a polynomial, then we define \( a^n \) inductively by:

- \( a^0 = 1; \)
- \( a^{n+1} = a \times a^n. \)

**Notation 10** If \( K \) is a set of coefficients, and \( V \) a set of variables, we write \( K[V] \) for the set of polynomials with coefficients in \( K \) and variables in \( V \). We write \( K[x] \) instead of \( K[\{x\}] \) etc.
2.1. WHAT ARE POLYNOMIALS?

2.1.1 How do we manipulate polynomials?

We have defined the abstract, almost Platonic, concept of polynomials as mathematical objects, and polynomial algebra as rules for these objects. What do we mean by the representation of these objects in a computer system?

One option would be for a computer algebra system essentially to do nothing, simply recording the computations requested by the user, so that \( a+b \) would become simply \( a+b \). However, we would not be very happy with a calculator which computed \( 1+1 \) as “1+1”, as we would rather see “2”. In particular, if the answer is 0, we would like to see that shown, i.e. we would like the representation to be normal (definition 3).

2.1.2 Polynomials in one variable

We will first consider polynomials in one variable, say \( x \). If the coefficients come from a domain \( K \), the polynomials in \( x \) over \( K \) are denoted by \( K[x] \) (Notation 10). One obvious route to a canonical representation (definition 4) is to insist that polynomials be expanded, i.e. that multiplication is only applied to coefficients and variables, not to general polynomials. This is achieved by applying distributivity, rule 8 from definition 19, where necessary, ensuring that multiplication is not applied to additions, and rule 8’ to collect terms. Once this is done, the polynomial is of the form \( \sum_{i=0}^{n} a_i x^i \), where the \( a_i \) are coefficients.

Notation 11 indexDegree!of polynomial We assume that \( a_n \neq 0 \), which is easy if the coefficients are represented normally (Definition 3). In this case, \( n \) is called the degree of the polynomial, denoted \( \deg(f) \), or \( \deg_x(f) \) if we wish to make it clear which variable is being considered. \( a_n \) is called the leading coefficient of \( f \), and denoted \( \text{lc}(f) \) or \( \text{lc}_x(f) \). If \( \text{lc}(f) = 1 \), we say that \( f \) is monic. \( f - \text{lc}(f)x^{\deg_x(f)} \), i.e. \( f \) minus its leading term, is known as the reductum of \( f \), \( \text{red}(f) \). The set \( \{ \text{red}(f), \text{red}(\text{red}(f)), \ldots \} \) is known as the iterated reducta of \( f \).

There is then an important distinction, which does not really occur when doing algebra by hand: does one represent the zero coefficients, or not?

Definition 23 A representation\footnote{In the current case, we are dealing with polynomials. But the concept is more general — see section 3.2.2 for sparse matrices, for example.} is said to be dense if every coefficient, zero or not, is represented, while it is sparse if zero coefficients are not stored.

Hence the polynomial \( x^2+0x-1 \), normally written as \( x^2-1 \), would be stored as \( <1,0,-1> \) in a dense representation, but \( <2,1>,<0,-1> \) in a sparse representation. As is implicit in the previous sentence, the normal “human” representation is sparse. Those systems that automatically expand, e.g. Reduce \[Hca05\], use a sparse representation, since a system would look fairly silly if it was unable to represent \( x^{1000000000}+1 \) since it could not store the 999,999,999 zeros. However, dense representations are often used internally in some algorithms.
CHAPTER 2. POLYNOMIALS

Proposition 7 Both the dense and the sparse expanded representation are canonical (definition [3]), provided that:

• the coefficients themselves are canonical (otherwise polynomials of degree 0 would not be canonical)
• leading (in the dense case) or all (in the sparse case) zeros are suppressed;
• (in the sparse case) the individual terms are stored in a specified order, generally sorted.

Addition is fairly straight-forward in either representation. In Lisp, we can do addition in a sparse representation as follows. We use a representation in which the CAR of a polynomial is a term, whilst the CDR is another polynomial: the initial polynomial minus the term defined by the CAR, i.e. the reductum (Notation [11]). A term is a CONS, where the CAR is the exponent and the CDR is the coefficient. Thus the LISP structure of the polynomial \(3x^2+1\) is \(((2 \cdot 3) (0 \cdot 1))\), and the list NIL represents the polynomial 0, which has no non-zero coefficients, and thus nothing to store. In this representation, we must note that the number 1 does not have the same representation as the polynomial 1 (which is \(((0 \cdot 1))\)), and that the polynomial 0 is represented differently from the other numerical polynomials.

(DE ADD-POLY (A B)
  (COND ((NULL A) B)
          ((NULL B) A)
          ((GREATERP (CAAR A) (CAAR B))
            (CONS (CAR A) (ADD-POLY (CDR A) B)))
          ((GREATERP (CAAR B) (CAAR A))
            (CONS (CAR B) (ADD-POLY A (CDR B))))
          ((ZEROP (PLUS (CDAR A) (CDAR B)))
            ; We must not construct a zero term
            (ADD-POLY (CDR A) (CDR B))))
    (T (CONS (CONS (CAAR A) (PLUS (CDAR A) (CDAR B)))
            (ADD-POLY (CDR A) (CDR B)))))))

(DE MULTIPLY-POLY (A B)
  (COND ((OR (NULL A) (NULL B)) NIL)
        ; If \(a = a0+a1\) and \(b = b0+b1\), then \(ab = a0b0 + a0b1 + a1b\)
        (T (CONS (CONS (PLUS (CAAR A) (CAAR B))
                    (TIMES (CDAR A) (CDAR B)))
            (ADD-POLY (MULTIPLY-POLY (LIST (CAAR A))
                                   (CDR B))
                      (MULTIPLY-POLY (CDR A) B)))))))

\[\text{But we should observe that Maple, for example, which uses a hash-based representation, is still canonical, even though it may not seem so to the human eye.}\]
2.1. WHAT ARE POLYNOMIALS?

If \( A \) has \( m \) terms and \( B \) has \( n \) terms, the calculating time (i.e. the number of LISP operations) for \textsc{add-poly} is bounded by \( O(m + n) \), and that for \textsc{multiply-poly} by \( O(m^2 n) \) \((m(m + 3)/2 - 1)n\) to be exact.\(^4\)

There is a technical, but occasionally important, difficulty with this procedure, reported in [ABD88]. We explain this difficulty in order to illustrate the problems which can arise in the translation of mathematical formulae into computer algebra systems. In \textsc{multiply-poly}, we add \( a_0 b_1 \) to \( a_1 b \). The order in which these two objects are calculated is actually important. Why, since this can affect neither the results nor the time taken? The order can dramatically affect the maximum memory space used during the calculations. If \( a \) and \( b \) are dense of degree \( n \), the order which first calculates \( a_0 b_1 \) should store all these intermediate results before the recursion finishes. Therefore the memory space needed is \( O(n^2) \) words, for there are \( n \) results of length between 1 and \( n \). The other order, \( a_1 b \) calculated before \( a_0 b_1 \), is clearly more efficient, for the space used at any moment does not exceed \( O(n) \). This is not a purely theoretical remark: [ABD88] were able to factorise \( x^{1155} - 1 \) with REDUCE in 2 megabytes of memory\(^5\), but they could not remultiply the factors without running out of memory, which appears absurd.

There are multiplication algorithms which are more efficient than this one: roughly speaking, we ought to sort the terms of the product so that they appear in decreasing order, and the use of \textsc{add-poly} corresponds to an insertion sort. We know that the number of coefficient multiplications in such a ‘classical’ method is \( mn \), so this emphasises that the extra cost is a function of the exponent operations (essentially, comparison) and list manipulation. Of course, the use of a better sorting method (such as “quicksort” or “mergesort”) offers a more efficient multiplication algorithm, say \( O(m n \log m) \) in terms of time. Naive construction of all the terms followed by sorting would take \( O(m n) \) space to build the unsorted list, and we can do better with ”heapsort” [Joh74]. But most systems have tended to use an algorithm similar to the procedure given above: [MP08] shows the substantial improvements that can be made using a better algorithm.

In general, the product of a sparse polynomial with \( m \) terms by one with \( n \) terms will have \( mn \) terms, so a \( O(m n \log m) \) has “nearly optimal” (optimal up to logarithmic factors) complexity in terms of the worst-case output size. However, this worst-case output size is often not achieved in practice. For example, the polynomials may actually be dense in \( x^{1000} \), and then the number of terms is \( m + n - 1 \) rather than \( mn \).

\textbf{Notation 12} Define the support of a polynomial \( f \), \( \text{supp}(f) \), to be the set of exponents \( n \) such that \( x^n \) occurs in \( f \) with a non-zero coefficient. The sparsity of \( f \) is the size of \( \text{supp}(f) \). Given two polynomials \( f \) and \( g \), the possible exponent set of \( f \cdot g \), \( \text{poss}(f, g) \) is \( \{ f_0 + e_g : e_f \in \text{supp}(f), e_g \in \text{supp}(g) \} \).

\(^4\)It is worth noting the asymmetry in the computing time of what is fundamentally a symmetric operation. Hence we ought to choose \( A \) as the one with the lower terms. If this involves counting the number of terms, many implementors ‘cheat’ and take \( A \) as the one of lower degree, hoping for a similar effect.

\(^5\)A large machine for the time!
Hence supp($f \cdot g$) \subseteq poss($f, g$), with strict inequality occurring when cancellation of coefficients gives us an “unexpected” zero.

**Theorem 3 ([AR15, Theorem 1.1])** Given $f, g \in \mathbb{Z}[x]$ with degree $D \geq \deg(f) + \deg(g)$ and height $C \geq ||f||_\infty + ||g||_\infty$, and constant $\mu \in (0, 1)$, Algorithm SparseMultZZ of [AR15] correctly computes the product $h = fg$ with probability exceeding $1 - \mu$, using expected $\tilde{O}((\# poss(f, g) \log D + \# supp(f \cdot g) \log C)$ bit operations.

We can simplify the complexity to $\tilde{O}((\# poss(f, g)(\log D + \log C))$, which shows that this algorithm, which is based on interpolation, is nearly optimal in the expected sparsity of the output. However, it is probabilistic, which limits its use in practice.

Maple’s original representation (see section 1.5.1), and methods, are rather different. The terms in a Maple sum might appear to be unordered, so we might ask what prevents $2x^2$ from appearing at one point, and $-2x^2$ at another. Maple uses a hash-based representation [CGGG83], so that insertion in the hash table takes amortised constant time, and the multiplication is $O(mn)$.

In a dense representation, we can use radically different, and more efficient, methods based on Karatsuba’s method [KO63] and section 3.3, which takes time $O \left( \max(m, n) \min(m, n)^{0.57} \right)$, or the Fast Fourier Transform [AHU74, chapter 8], where the running time is $O(\max(m, n) \log \min(m, n))$. Since the number of terms in a dense product is $m + n - 1 = O(\max(m, n))$, these algorithms are nearly optimal.

Division is fairly straight-forward: to divide $f$ by $g$, we keep subtracting appropriate (meaning $c_i = (\text{lc}(f) / \text{lc}(g)) x^{\deg(f) - \deg(g)}$) multiples of $g$ from $f$ until the degree of $f$ is less than the degree of $g$. If the remaining term (the remainder) is zero, then $g$ divides $f$, and the quotient can be computed by summing the $c_i$. This is essentially the process known to schoolchildren as “long division”.

However, the complexity of sparse polynomial division is not so straightforward [DC10]. We cannot bound the complexity in terms of just the number of terms in the input, because of the example of

$$\frac{x^n - 1}{x - 1} = x^{n-1} + \cdots + x + 1,$$

where two two-term inputs give rise to an $n$-term output. So imagine that we are dividing $f$ by $g$, to give a quotient of $q$ and a remainder of $r$.

**Notation 13** Let a polynomial $f$ have degree $d_f$, and $t_f$ non-zero terms.

Since each step in the division algorithm above generates a term of the quotient, there are $t_q$ steps. Each step generates $t_g$ terms, so there are $O(t_g)$ arithmetic operations.

---

\(^6\)This is an Atlantic City (“probably fast/probably correct”) algorithm: see section 1.4.1.

\(^7\)Occasionally the hash table may need to grow, so an individual insertion may be expensive, but the average time is constant.
2.1. WHAT ARE POLYNOMIALS?

operations. But there are also the comparison operations required to organise the subtraction, and it is hard to find a better bound than \( d_f \) for the number of these. Hence the total cost is \( O(t_q(d_f + t_g)) \) operations. Since we don’t know \( t_q \) until we have done the division, the best estimate is \( O(d_f^2 + d_ft_g) \). Again, ideas based on Heapsort can improve this, and the best result is \( O(t_f + t_q t_g \log \min(t_q, t_g)) = O(t_f + (d_f - d_g)t_g \log \min(d_f - d_g, t_g)) \) [MP11].

Example 1 (Bad Mergesort) The reason we need to use Heapsort rather than, say, Mergesort is seen is a case like the following

\[
\frac{x^n}{x^{n/2} + x^{n/2 - 1} + x^{n/4} + 1},
\]

where we perform \( n/2 \) merges of the divisor (4 terms) with polynomials with up to \( n/2 \) terms, i.e. \( O(n^2) \) work. The problem arises because the merges in this case are asymmetrical, and a merge is only efficient when merging things of roughly the same size.

An alternative strategy, known as “geobuckets” for “geometrically increasing buckets”, was devised by Yan [Yan98]. Let \( d \) be a fixed growth factor (his experiments suggested \( d = 4 \)). The application was Buchberger’s Algorithm [9], where we are repeatedly computing \( f := f - c_i g_i \), where \( f \) typically is a large polynomial and the \( c_i g_i \) are polynomials which may be large or small. His intermediate representation for a polynomial \( f \) was

\[
f := f_1 \oplus f_2 \oplus \cdots \oplus f_l,
\]

where \( \oplus \) signifies a lazy summation that we have yet to perform, and each “bucket” \( f_i \) consists of at most \( d^i \) terms, sorted normally. Then, when we have to subtract \( c_i g_i \) from \( f \), we subtract it from \( f_k \), where \( k \) is minimal with \( d^k \geq t_g \). It is conceivable that this will cause \( f_k \) to ‘overflow’, i.e. have more than \( d^k \) terms, in which case we add \( f_k \) to \( f_{k+1} \), and then set \( f_k \) to 0. This has the consequence that we (almost) never add (i.e. merge) polynomials of greatly unequal sizes, and also, compared with a heap, the storage requirements are reduced: if \( d \geq 2 \) the redundancy is bound to be less than 50\%, even if every monomial in every other \( f_i \) is a duplicate of a monomial in \( f_l \). At the end, we actually perform the summations implicit in (2.2) to get a polynomial in the usual form, taking care to perform them as

\[
(f_1 \oplus f_2 \oplus \cdots) \oplus f_l
\]

so as to preserve the balanced nature of the merges.

2.1.3 A factored representation

Instead of insisting that multiplication not be applied to addition, we could insist that addition not be applied to multiplication. This would mean that a
polynomial was represented as a product of polynomials, each the sum of simple terms:

\[ f = \prod_i f_i = \prod_i \left( \sum_{j=0}^{n_i} a_{i,j} x^j \right). \]  \hspace{1cm} (2.3)

In practice, repeated factors are stored explicitly, as in the following format:

\[ f = \prod_i f_i^{d_i} = \prod_i \left( \sum_{j=0}^{n_i} a_{i,j} x^j \right)^{d_i}. \]  \hspace{1cm} (2.4)

We have a choice of using sparse or dense representations for the \( f_i \), but usually sparse is chosen. It is common to insist that the \( f_i \) are square-free\(^8\) and relatively prime\(^9\) (both of which necessitate only g.c.d. computations\(^10\) — lemma 3), but not necessarily\(^11\) irreducible. Hence this representation is generally known as \textit{partially factored}. In this format, the representation is not canonical, since the polynomial \( x^2 - 1 \) could be stored either as that (with 2 terms), or as \( (x-1)(x+1) \) (with 4 terms); however, it is normal in the sense of definition \[3\]. For equality testing, see excursus \[B.2\].

Multiplication is relatively straight-forward, we check (via g.c.d. computations\(^12\)) for duplicated factors between the two multiplicands, and then combine the multiplicands. Addition can be extremely expensive, and the result of an addition can be exponentially larger than the inputs: consider

\[ (x + 1)(x^2 + 1) \cdots (x^{2^k} + 1) + (x + 2)(x^2 + 2) \cdots (x^{2^k} + 2), \]

where the input has \( 4(k+1) \) non-zero coefficients, and the output has \( 2^k + 1 \) (somewhat larger) ones.

This representation is not much discussed in the general literature, but is used in Redlog \[DS97\] and Qepcad \[CH91\], both of which implement cylindrical algebraic decomposition (see section \[3.5\]), where a great deal of use can be made of corollaries \[20\] and \[21\].

---

\( ^8 \)Which almost certainly improves compactness, but see \[CD91\], where a dense polynomial of degree 12 was produced (13 terms), whose square had only 12 nonzero terms, and the process can be generalised. Twelve is minimal \[Abb02\].

\( ^9 \)Which often improves compactness, but consider \((x^p - 1)(x^q - 1)\) where \( p \) and \( q \) are distinct primes, which would have to be represented as \((x - 1)^2(x^{p-1} + \cdots + 1)(x^{q-1} + \cdots + 1)\).

\( ^10 \)These g.c.d. computations, if carried out by modular or \( p \)-adic methods (pages \[143\] and \[189\]), \textit{should} be cheap if the answer is “no simplification”, and otherwise should, at least in non-pathological cases, lead to greater efficiency later.

\( ^11 \)If we were to insist on irreducibility, we would need to store \( x^{p-1} \) as \((x - 1)(x^{p-1} + \cdots + 1)\), with \( p + 2 \) terms rather than with 2. Furthermore, irreducibility can be expensive to prove \[ASZ00\].

\( ^12 \)Again, these should be cheap if there is no factor to detect, and otherwise lead to reductions in size.
2.1. WHAT ARE POLYNOMIALS?

2.1.4 Polynomials in several variables

Here the first choice is between factored and expanded. The arguments for, and algorithms handling, factored polynomials are much the same as in the case of one variable. The individual factors of a factored form can be stored in any of the ways described below for expanded polynomials, but recursive is more common since it is more suited to g.c.d. computations (chapters 4 and 5), which as we saw above are crucial to manipulating factored representations.

If we choose an expanded form, we have one further choice to make, which we explain in the case of two variables, \(x\) and \(y\), and illustrate the choices with \(x^2y + x^2 + xy^2 - xy + x - y^2\).

**Notation 14** The total degree of a monomial is the sum of the degrees of all the variables. The total degree of a polynomial is the greatest total degree of any monomial in it.

Note that this definition implicitly assumes that we have performed any cancellation, since the total degree of \(x^2y^3 - x - x^2y^3\) is 1, not 5. Put another way, we require the representation to be ***candid*** (Definition 5), at least in this respect.

**recursive \(\rightarrow C[x][y]\).** We regard the polynomials as polynomials in \(y\), whose coefficients are polynomials in \(x\). Then the sample polynomial would be \((x - 1)y^2 + (x^2 - x)y + (x^2 + x)y^0\). We have made the \(y^0\) term explicit here: in practice detailed representations in different systems differ on this point.

**recursive \(\rightarrow C[y][x]\).** We regard the polynomials as polynomials in \(x\), whose coefficients are polynomials in \(y\). Then the sample polynomial would be \((y + 1)x^2 + (y^2 - y + 1)x + (-y^2)x^0\).

**distributed \(\rightarrow C[x, y]\).** We regard the polynomials as polynomials in \(x\) and \(y\), whose coefficients are numbers. With 6 terms (as in this example), there are 6! = 720 possible orders. It is usual to impose two additional constraints on the order on terms, or more accurately on the monomials\(^{14}\), i.e. ignoring the coefficients, which we will denote\(^{15}\) as >.

**Definition 24** An ordering is said to be an admissible ordering if it satisfies the following conditions.

- **Compatibility with multiplication:** if \(a > b\) then, for all monomials \(c\), \(ac > bc\).

\(^{13}\)In one variable, the space requirements for a typical dense polynomial and its factors are comparable, e.g. 11 terms for a polynomial of degree 10, and 6 each for two factors of degree 5. For multivariates, this is no longer the case. Even for bivariates, we would have 121 terms for a dense polynomial of degree 10, and 36 each for two factors of degree 5. The gain is greater as the number of variables increases.

\(^{14}\)We use the word monomial to mean a product of (possibly repeated) variables, as in \(xyz\) or \(x^2y\), without any coefficient. Usage on this point differs.

\(^{15}\)We are not making any numerical evaluation of the monomials, merely saying which order we put the monomials in.
• **Well-foundedness:** for all non-trivial monomials \( a, a > 1 \).

These requirements greatly reduce the available orders for our sample polynomial. One possibility would be to sort by total degree (i.e. the sum of the degrees in each variable), using degree in \( x \) as a tie-breaker. This would give us \( x^2y + xy^2 + x^2 - xy - y^2 + x \). There is a fuller discussion of such orderings in Section 3.3.3. However, we should note one important property of admissible orders here.

**Theorem 4 (Descending Chain Condition; Dickson’s Lemma)**

Any decreasing sequence (with respect to an admissible ordering) of monomials is finite. \[Dic13\]

In general, if there are \( n \) variables, there are \( n! \) possible recursive representations, but an infinite number of possible distributed representations, though clearly only finitely many different ones for any one given polynomial or finite set of polynomials.

In both cases, we use sparse, rather than dense, representations, since any reasonable multivariate polynomial had better be sparse: degree 6 in each of 6 variables means \( 7^6 = 117649 \) terms. The same results as for univariate polynomials apply.

**Proposition 8** For a fixed ordering, both recursive and distributed representations are canonical (definition 3). Partially factored representations are normal, but not canonical.

It makes no sense to compare polynomials in different representations, or the same representation but different orderings. We have spoken about ‘representations’, but in fact the division between recursive and distributed goes deeper. While characterisations 3 and 2 of a Gröbner base (theorem 16) can make sense in either view, characterisations 1 and 4 (the only effective one) only make sense in a distributed view. Conversely, while the abstract definitions of factorisation and greatest common divisors (definition 20) make sense whatever the view, the only known algorithms for computing them (algorithm 2 or the advanced ones in chapters 4 and 5) are inherently recursive.\[16\]

### 2.1.5 Other representations

Sparse representations take up little space if the polynomial is sparse. But shifting the origin from \( x = 0 \) to \( x = 1 \), say, will destroy this sparsity, as might many other operations. The following example, adapted from [CGH+03], illustrates this. Let \( \Phi(Y,T) = \exists X_1 \ldots \exists X_n (X_1 = T + 1) \land (X_2 = X_1^2) \land \cdots \land (X_n = X_{n-1}^2) \land (Y = X_n^2). \) (2.5)

\[16\] At least for commutative polynomials. Factorisation of non-commutative polynomials is best done in a distributed form.
2.1. WHAT ARE POLYNOMIALS?

The technology described in section 3.5.3 will convert this to a polynomial equation

\[ \Psi(Y, T) : Y = (1 + T)^{2^n}. \]  

(2.6)

Dense or sparse representations have problems with this, in the sense that expression (2.5) has length \( O(n) \), but expression (2.6) has length \( O(2^n) \) or more. A factored representation could handle the right-hand side, \textit{assuming} that we are not representing the equations as polynomial \( = 0 \). But changing the last conjunct of \( \Phi \) to \( Y = (X_n + 1)^2 \) changes \( \Psi \) to

\[ Y = \left(1 + (1 + T^{2^{n-1}}) \right)^2, \]  

(2.7)

whose factored representation now has length \( O(2^n) \).

Factored representations display a certain amount of internal structure, but at the cost of an expensive, and possibly data-expanding, process of addition. Are there representations which do not have these ‘defects’? Yes, though they may have other ‘defects’.

**Expression tree** This representation “solves” the cost of addition in the factored representation, by storing addition as such, just as the factored representation stored multiplication as such. Hence \((x + 1)^3 - 1)^2 \) would be legal, and represented as such. Equation (2.7) would also be stored compactly \textit{provided} exponentiation is stored as such, e.g. \( Z^2 \) requiring one copy of \( Z \), rather than two as in \( Z \cdot Z \). This system is not canonical, or even normal: consider \((x + 1)(x - 1) = (x^2 - 1) \). This would be described by Moses \[ Mos71 \] as a “liberal” system, and generally comes with some kind of \textit{expand} command to convert to a canonical representation. Assuming now that the leaf nodes are constants and variables, and the tree’s internal nodes are (binary, i.e. with two arguments) addition, subtraction and multiplication, then a tree with maximal depth \( p \) can represent a polynomial with maximum total degree \( 2^p \). It would need to have \( 2^p - 1 \) internal nodes (all multiplication), and \( 2^p \) leaf nodes. The degree is easy to bound, by means of a tree-walk, but harder to compute, especially if cancellation actually occurs. Similarly, the leading coefficient can be computed via a simple tree-walk \textit{if} no cancellation occurs.

**Expression DAG** also known as \textit{Straight-Line Program (SLP)} \[ IL80 \].

This is essentially\[^{17}\] the representation used by Maple — it looks like the previous representation, but the use of hashing in fact makes it a directed acyclic graph (DAG). Again, a straight-line program of length \( l \) (i.e. a DAG of depth \( l - 1 \)) can store a polynomial of degree \( 2^{l-1} \). The difference with the expression tree representation above is that we only need \( l \) nodes, since the nodes can be reused.

This format is essentially immune to the “change of origin” problem mentioned above, since we need merely replace the \( x \) node by a tree to compute

\[^{17}\] Maple uses \( n \)-ary addition and multiplication, rather than binary, as described in section \[ C.3 \].
x + 1, thus adding two nodes, and possibly increasing the depth by one, irrespective of the size of the polynomial. The general ‘straight-line’ formalism has advantages where multi-valued functions such as square root are concerned: see the discussions around figures 3.1 and 3.2.

However, there is one important caveat about straight-line programs: we must be clear what operations are allowed. If the only operations are +, − and ×, then evidently a straight-line program computes a polynomial. Equally, if division is allowed, the program might not compute a polynomial. But might it? If we look at figure 2.1 we see that $p = x^2 - 1$ and $q = x - 1$, so the result is $\frac{p}{q} = \frac{x^2 - 1}{x - 1} = x + 1$. Or is it? If we feed in $x = 1$, we in fact get $\frac{0}{0}$, rather than 2. This is a singularity of the kind known as a removable singularity, because $\lim_{x \to 1} \frac{p(x)}{q(x)} = 2$. In fact [IL80] Theorem 3], deciding if two straight-line programs are equivalent is undecidable if division is allowed.

![Figure 2.1: A polynomial SLP](image)

We said earlier that the only known algorithms for computing greatest common divisors were recursive. This is essentially true, and means that the computation of greatest common divisors of straight-line programs is not a straight-forward process [Kal88].

Reconstructing one of the more explicit representations (usually a sparse one!) from a straight-line program representation is not straightforward. The current state of the art over general finite fields is the following

**Proposition 9 ([AGR14a, Theorem 1])** Let $F \in \mathbb{F}_q[z_1, \ldots, z_n]$, and suppose we are given a division-free straight-line program $S_F$ of length $L$ which evaluates $F$, an upper bound $D = \max_j \deg_{z_j}(F)$, and an upper bound $T$ on the number of nonzero terms $t$ of $F$. There exists a probabilistic algorithm which interpolates $F$ with probability at least $3/4$. The algorithm requires

$$\bar{O}(L n \log D + n)(\log D + \log q) \log D + n^{\omega - 1} T \log D + n^\omega \log D)$$

bit operations, where $\omega$ is the matrix multiplication exponent.

---

18The author is grateful to Dan Roche for explanations here, and for [Roc14].
2.1. WHAT ARE POLYNOMIALS?

Figure 2.2: Code fragment A — a graph

\[ p := x + 1; \]
\[ q := p; \]
\[ r := p \cdot q; \]

We note that this is linear in \( T \), which is as good as we could hope for.

If we can choose the finite field, to be \( \mathbb{Z}_p \) where \( p \) is smooth (has only small prime divisors) then we can do much better: \( \tilde{O}(LT \log D + n^2 T \log^2 D) \) [Kal10] for the algorithm, [AGR14a] for the complexity. Such fields appear to be common [HB78], and this gives us an algorithm for interpolating \( F \) over the integers with time \( \tilde{O} \left( (n^2 T \log D + nLT)(n \log D + \log H) \right) \) [Kal10] for the algorithm, [Roc14] for the complexity. There are further improvements to be had in [AGR14b].

Additive Complexity This [Ris85] [Ris88] is similar to a straight-line program, except that we only count the number of (binary) addition/subtraction nodes, i.e. multiplication and exponentiation are ‘free’. Hence the degree is unbounded in terms of the additive complexity, but for a given expression (tree/DAG) can be bounded by a tree-walk. A univariate polynomial of additive complexity \( a \) has at most \( C a^2 \) real roots for some absolute constant \( C \); conjecturally this can be reduced to \( 3a \). These bounds trivially translate to the straight-line program and expression tree cases.

Specialist There are many possible highly specialist representations of polynomials. One of the most impressive (where it’s applicable) is the graph representation, used in [AIR14] to represent a polynomial with 317,881,154 monomials in a half-page graph.

“Additive complexity” is more of a measure of the ‘difficulty’ of a polynomial than an actual representation. Of the others, the first was used in Macsyma for its “general expression”, and the second is used in Maple\(^{19}\). In fact, Macsyma would\(^{20}\) allow general DAGs, but would not force them. Consider the two code fragments in figures 2.2 and 2.3. In the case of figure 2.2 both systems would produce the structure in figure 2.4. For figure 2.3 Macsyma would produce the structure\(^{21}\) in figure 2.5. Maple would still produce the structure of figure 2.4 since the hashing mechanism would recognise that the two \( x + 1 \) were identical.

\(^{19}\)Until such time as operations such as expand are used!

\(^{20}\)Not explicitly, but rather as a side-effect of the fact that Macsyma is implemented in Lisp, which cares little for the difference. The basic function \texttt{EQUAL} does not distinguish between acyclic and cyclic structures

\(^{21}\)The \( x \) is shown as shared since the Lisp implementation will store symbols uniquely.
Figure 2.3: Code fragment B — a tree

\[
p := x + 1; \\
q := x + 1; \\
r := p \times q;
\]

Figure 2.4: DAG representation

\[
\begin{align*}
1 & \searrow x \\
p & \rightarrow + \leftarrow q \\
r & \rightarrow * 
\end{align*}
\]

Figure 2.5: Tree representation

\[
\begin{align*}
1 & \searrow x & 1 \\
p & \rightarrow + & + & \leftarrow q \\
r & \rightarrow * 
\end{align*}
\]
2.1. WHAT ARE POLYNOMIALS?

2.1.6 The Newton Representation

For simplicity, in this subsection we will only consider the case of characteristic 0: finite characteristic has some serious technical difficulties, and we refer the reader to [BFSS06]. We will also only consider monic polynomials.

Notation 15 Let \( p = x^n + \sum_{i=0}^{n-1} a_i x^i = \prod_{i=0}^{n} (x - \alpha_i) \) be a polynomial of degree \( n \). Let \( \beta_s = \sum_{i=0}^{n} \alpha_i^s \), and define the Newton series of \( p \) to be \( \text{Newton}(p) = \sum_{s \geq 0} \beta_s T^s \).

It is well-known that the \( a_i \) and \( \alpha_i \) are related:

\[
\begin{align*}
a_{n-1} &= - \sum_{i=0}^{n} \alpha_i \\
a_{n-2} &= \sum_{i=0}^{n} \sum_{j=i+1}^{n} \alpha_i \alpha_j \\
& \vdots \\
a_0 &= (-1)^n \prod_{i=0}^{n} \alpha_i.
\end{align*}
\]

These are then related to the \( \beta_i \):

\[
\begin{align*}
\beta_1 &= -a_{n-1} \\
\beta_1^2 &= \beta_2 + 2a_{n-2} \\
& \vdots \\
& \vdots
\end{align*}
\]

Hence, in characteristic 0, the \( \beta_i \) (\( i \leq n \)) form an alternative to the \( a_i \), a fact known since 1840 [LV40]. But how do we convert rapidly between these representations?

Proposition 10 ([Sch82], see also [BFSS06, Lemma 1]) \( \text{Newton}(p) = \frac{\text{rev}(p')}{\text{rev}(p)} \) as a power series about \( x = 0 \), where \( \text{rev}(p) = \sum_{i=0}^{n} a_{n-i} x^i \).

2.1.7 Representations in Practice

General-purpose (calculus-side) computer algebra systems have to deal with many expressions other than polynomials, but tend to regard polynomials as the basic construct.

2.1.7.1 Representations in Reduce

This is probably one of the most straightforward. The basic Reduce object is a standard quotient, i.e. a pair (literally a CONS cell in Lisp) of polynomials, or standard forms. A standard form is a sparse recursive multivariate polynomial, where the “variables” (known as kernels in Reduce) may be variables such
as \( x \), but equally functions and expressions such as \((\cos x)\) — a Lisp form representing \( \cos(x) \).

If the kernels are genuinely indeterminates, i.e. we have a free algebra (Definition 20), then this is a normal form, and it is canonical subject to the restrictions in Proposition 14. Initially Reduce did not, by default, compute gcds (clause 3), but this changed as more efficient gcd algorithms were implemented.

### 2.1.7.2 Representations in Macsyma

Macsyma’s “general representation” is essentially an expression tree one. There is a special form, known asCanonical Rational Expression\(^{22}\) which again is a ratio of sparse recursive polynomials.

### 2.1.7.3 Representations in Maple

Maple’s original internal representation\(^{23}\) was an expression tree whose fundamental form was an \( n \)-ary sum of \( n \)-ary power products, as in the representation of \(9xy^3z - 4y^3z^2 - 6xy^2z - 8x^3 - 5\) as in Figure 2.6. The “variables” might in fact be other Maple expressions, so that \( x^2 - (x - 1)(x + 1) - 1 \) would be a sum of three terms, one of which was the product of two elements each of which were themselves sums. The elements in \textsc{sum} and \textsc{prod} expressions were stored in hash-code order, which greatly facilitated combining like terms. The advantages and disadvantages of this are described in \textsc{MP12, MP13}.

As described there, for genuine polynomials (which would exclude \( x^2 - (x - 1)(x+1)-1 \) and expressions involving \textsc{RootOf} etc.), an alternative data structure

---

\(^{22}\)See \url{http://www.ma.utexas.edu/maxima/maxima_11.html}.

\(^{23}\)The author is grateful to Michael Monagan of Simon Fraser University for much of this information, and for permission to reproduce the images. Note that, in the standard format for a \textsc{sum}, all the coefficients are the odd-numbered elements (counting the \textsc{sum} as number 1) \textit{except} for the constant term. This is apparently a historical design decision.
2.1. WHAT ARE POLYNOMIALS?

was introduced at Maple 17: the POLY data structure, which is a packed sparse distributed representation: in this case as in Figure 2.7 where the numbers 5131

![Figure 2.7: Maple’s New-Style Polynomials](image)

POLY12 means a poly data structure of header word, a pointer to the variables, and five exponent/coefficients pairs.

```plaintext
SEQ 4 x y z
-4 -6 -8 -59 5032 4121 3300 0000 5131
```

etc. are in fact numbers base $2^{15}$, so “5131” = $5 \cdot 2^{45} + 2^{30} + 3 \cdot 2^{15} + 1$, meaning ‘total degree 5’ and then the exponents of the individual variables. This form is only used if the packed exponent field will fit into a 64-bit integer (or a 32-bit integer on 32-bit Maple. Note that a 64-bit Maple integer actually goes up to $2^{62} - 1$, as given by `kernelopts(maximmediate)`). In this case, we had four ‘variables’ (three real ones and the slot for total degree), and 16 = $\lceil 62/4 \rceil$ — with four real variables we would have a base $2^{12}$ expression as $12 = \lceil 62/5 \rceil$, and so on. The fact that total degree is stored, and that the items in a POLY data structure are stored in decreasing order of this packed exponent, means that we have a ‘graded lexicographic’ (see p. 96) ordering.

The difference should be invisible to the casual Maple user, except for performance, but can be seen via the `dismantle` command.

### 2.1.8 Comparative Sizes

This section is largely of theoretical interest, since practical systems employ a variety of techniques for storing polynomials, and are also constrained by the actual size of machine words etc. Hence an analysis of a practical system will tend to contain $4\lceil \log_2(n) \rceil$ for the size in bytes, rather than $\log_2(n)$ for the size in bits.

**Notation 16** We assume our polynomials have integer coefficients and are in $N$ variables, each of which $\leq 24$ occurs to degree at most $D$, and the coefficients are at most $C$ in absolute value. Furthermore, suppose there are at most $T$ non-zero terms. Let $n = \log_2 N$, $d = \log_2 D$ and $c = 1 + \log_2 C$ (the “1+” allows for a sign. We ignore the space needed to store $N$, $D$ etc. themselves.

**Dense** At least in principle, dense storage, whether recursive or distributed, just stores the coefficients and no additional structural information. There are $(D+1)^N$ terms, and hence the storage needed is $s_{\text{dense}} = c(D+1)^N$.

\[ \text{Dense} \]

\[ s_{\text{dense}} = c(D+1)^N \]

\[ ^{24} \text{Many authors prefer to bound the total degree by } D. \text{ The difference is not great in practice.} \]
Sparse (Distributed) There are $T$ terms, each needing a coefficient and $N$ exponents, hence $s_{\text{sparse}} = T(c + Nd)$. If $T$ is maximal (i.e. the polynomial is completely dense), $s_{\text{sparse}} = s_{\text{dense}} + (D + 1)^N Nd$.

Sparse (Recursive) This is harder to describe, partly because it depends on the variable order, as $x_1^D x_2^D \cdots x_{N-1}^D (x_N^D + \cdots + x_N^0)$ stores $N + D$ exponents, but if $x_N$ is the main variable, it stores $(D + 1)N$ exponents. The coefficient storage is the same as for $s_{\text{sparse}}$, though.

Expression Tree Since this representation is not canonical, we need to discuss the size of a particular representation, not just of an abstract polynomial. Call this $s_{\text{tree}}$. Note that $x^N$ requires $N - 1$ multiplications, whether we use repeated squaring, as in $x^N = (x \ast x) \ast (x \ast x)$ or iteration as in $x^4 = x \ast (x \ast (x \ast x))$.

Expression DAG Again this is not canonical. $x^N$ can now be computed by recursive squaring with reuse, so $x^{2^n}$ only needs $n$ multiplications. Call this $s_{\text{DAG}}$. Since every tree is a DAG, we have $s_{\text{DAG}} \leq s_{\text{tree}}$.

Though there are unusual counter-examples, if we assume that $c \gg d \gg n$, then

$$s_{\text{dense}} \gg s_{\text{sparse}} \gg s_{\text{tree}} \gg s_{\text{DAG}}.$$  \hfill (2.8)

Proposition 11 Each of the gaps in (2.8) can be exponentially big.

2.2 Rational Functions

Of course, we want to manipulate a wider class of expressions, and even $\frac{1}{x}$ is not a polynomial.

Definition 25 A rational function is built up from coefficients, which are assumed to form an integral domain (definition 9), and certain indeterminates, by the algebraic operations of addition, subtraction, multiplication and division (except that division by zero is not permitted). In addition to the laws in definition 19 (but with $a$, $b$ and $c$ interpreted as rational functions), the following law is obeyed.

11. $a \ast (1/a) = 1$.

2.2.1 Canonical Rational Functions

Proposition 12 Any rational function $f$ can be put over a common denominator, i.e. written as $n/d$ where $n$ and $d$ are polynomials, known as the numerator and denominator respectively. We write num$(f)$ and den$(f)$ respectively, noting that in fact these are only defined up to units.
Proposition 13  *In common denominator format, \( \frac{a}{b} = \frac{c}{d} \) if, and only if, \( ad - bc = 0 \).*

We can in fact characterise three simple forms of equality.

**common coefficients**  An example of this would be.

\[
\frac{x^2 - 2x + 1}{x^2 - 1} \quad \text{versus} \quad \frac{2x^2 - 4x + 2}{2x^2 - 2}.
\]

Here we need to remove the g.c.d. of the contents (definition 30) of the two polynomials.

**“up to sign”**  An example of this would be.

\[
\frac{-x^2 + 2x - 1}{x^2 - 1} \quad \text{versus} \quad \frac{x^2 - 2x + 1}{-x^2 + 1}.
\]

These are ‘clearly equal’ but “computers don’t do clearly”. We need a convention, and the common one\(^{25}\) is ‘leading coefficient positive’ in the denominator. However, this does not generalise so easily to other domains of coefficients [DT90].

**common factors**  An example of this would be

\[
\frac{x^2 - 2x + 1}{x^2 - 1} \quad \text{versus} \quad \frac{x - 1}{x + 1}.
\]

If we put the difference between the two over a common denominator, we get \( \frac{0}{x^2 - 1} = 0 \). The reader may complain that \( \frac{x^2 - 2x + 1}{x^2 - 1} \) “is undefined when \( x = 1 \)”, whereas \( \frac{x - 1}{x + 1} \) “has the value 0”. However, we have not defined what we mean by such substitutions, and for the purposes of this chapter, we are concerned with *algebraic equality* in the sense of proposition 13.

**Proposition 14 ([DT90])**  *A representation \( n/d \) where \( n \) and \( d \) are polynomials is canonical if the following conditions are satisfied:*

1. \( n \) and \( d \) are polynomials from a free (Definition 20) polynomial algebra;
2. these polynomials are themselves represented canonically;
3. *Any greatest common divisor, whether polynomial or content, is removed from \( n \) and \( d \);
4. \( n/d \) is canonical with respect to units, typically by insisting that \( d \) have a canonical-up-to-associates (e.g. positive) leading coefficient.*

\(^{25}\) By no means the only possible one: ‘leading coefficient negative’ would be equally valid, as would ‘trailing coefficient positive’.
CHAPTER 2. POLYNOMIALS

The reader might think that condition 1 was unnecessary in view of condition 2. This is not so is shown by

$$\sqrt{2} - 1 = \frac{1}{\sqrt{2} + 1} \quad (2.9)$$

where each of \( \sqrt{2} - 1 \) and \( \sqrt{2} + 1 \) are represented canonically, but nevertheless we have an equality here, which wouldn’t occur for any value of the “indeterminate” except \( \sqrt{2} \).

2.2.2 Candidness of rational functions

We have already given (Definition 5) an abstract definition of candidness, which can also be described as “what you see is what you’ve got” mathematically. What would this mean for rational functions (and therefore for polynomials)? [Sto11a, p.869] gives the following as a sufficient set of conditions:

1. there are no compound ratios such as

$$x + \frac{x + 1}{1 + \frac{1}{x}} \quad (2.10)$$

(note that this is 2x, and therefore “only” a polynomial, so violates the general definition of candidness),

2. all ratios that occur are reduced (therefore preferring \( \frac{x^{999} + \cdots + x + 1}{x^{1000} - 1} \) to \( \frac{x - 1}{x - 1} \), so disagreeing with Carette’s definition of ‘simplification’ on page 23),

3. the factors and terms are ordered in an easily discerned traditional way, such as lexically by descending degree,

4. all manifestly similar factors and terms are collected,

5. for each variable, the actual degree of every variable in a reduced ratio of an expanded numerator and denominator would be no less than what a user would predict assuming no cancellations. For example, assuming no cancellations, we can predict that at most the degree of \( x \) will be 3 in the denominator and 6 in the numerator when

$$x^3 + \frac{1}{x^2 - 1} + \frac{1}{x + 2} \quad (2.11)$$

is reduced over a common denominator. Those are the resulting degrees, so (2.11) is a candid representation, even though it’s probably not one of a class of canonical representations. Conversely, (2.10) violates this condition, since we would predict it to be the ratio of a degree 2 and a degree 1 polynomial.

---

26He also remarks “There can be other candid forms for rational expressions, including [appropriately reduced, ruling out (2.10)] continued fractions. However, the complexity of implementing a candid simplifier increases with the permissiveness of the allowed result forms.”

27It is this clause that Maple’s sometimes disconcerting hash-based output breaks.
In particular, clause 2 (or 5) implies that any common factors are cancelled, which poses the question, answered in the next section: how do we compute common factors? Given \( \frac{f}{g} \), is there an \( h \) dividing both \( f \) and \( g \) that should be cancelled?

### 2.3 Greatest Common Divisors

The following definition is valid whenever we have a concept of division.

**Definition 26** \( h \) is said to be a greatest common divisor, or g.c.d., of \( f \) and \( g \) if, and only if:

1. \( h \) divides both \( f \) and \( g \);
2. if \( h' \) divides both \( f \) and \( g \), then \( h' \) divides \( h \).

This definition clearly extends to any number of arguments. The g.c.d. is normally written \( \gcd(f, g) \).

Note that we have defined a g.c.d, whereas it is more common to talk of the g.c.d. However, ‘a’ is correct. We normally say that 2 is the g.c.d. of 4 and 6, but in fact \(-2\) is equally a g.c.d. of 4 and 6.

**Proposition 15** If \( h \) and \( h' \) are greatest common divisors of \( a \) and \( b \), they are associates (definition 11).

**Example 2 (Greatest common divisors need not exist)** Consider the set of all integers with \( \sqrt{-5} \). 2 clearly divides both 6 and \( 2 + 2\sqrt{-5} \). However, so does \( 1 + \sqrt{-5} \) (since \( 6 = (1 + \sqrt{-5})(1 - \sqrt{-5}) \)), yet there is no multiple of both 2 and \( 1 + \sqrt{-5} \) which divides both.

**Definition 27** An integral domain (definition 9) in which any two elements have a greatest common divisor is known\(^{28}\) as a g.c.d. domain.

If \( R \) is a g.c.d. domain, then the elements of the field of fractions (definition 14) can be simplified by cancelling a g.c.d. between numerator and denominator, often called “reducing to lowest terms”. While this simplifies fractions, it does not guarantee that they are normal or canonical. One might think that \( \frac{0}{1} \) was the unique representation of zero required for normality, but what of \( \frac{-1}{-1} \)? Equally \( \frac{-1}{2} = \frac{1}{2} \), and in general we have to remove the ambiguity caused by units. In the case of rational numbers, we do this automatically by making the denominator positive, but the general case is more difficult [DT90].

**Definition 28** \( h \) is said to be a least common multiple, or l.c.m., of \( f \) and \( g \) if, and only if:

\(^{28}\) Normally known as a *unique factorisation domain*, but, while the existence of greatest common divisors is equivalent to the existence of unique factorisation, the ability to *compute* greatest common divisors is not equivalent to the ability to *compute* unique factorisations [FS56, DCT91], and hence we wish to distinguish the two.
1. both \( f \) and \( g \) divide \( h \);
2. if both \( f \) and \( g \) divide \( h' \), then \( h \) divides \( h' \).

This definition clearly extends to any number of arguments. The l.c.m. is normally written \( \text{lcm}(f, g) \).

**Proposition 16** If \( \gcd(f, g) \) exists, then \( \frac{fg}{\gcd(f, g)} \) is a least common multiple of \( f \) and \( g \).

This result is normally written as \( fg = \gcd(f, g) \text{lcm}(f, g) \), but this is only true up to associates. We should also note that this result does not extend to any number of arguments: in general \( fgh \neq \gcd(f, g, h) \text{lcm}(f, g, h) \).

### 2.3.1 Polynomials in one variable

For univariate polynomials over a field, we can define a more extended version of division.

**Definition 29** If \( a \) and \( b \neq 0 \) are polynomials in \( K[x] \), \( K \) a field, and \( a = qb + r \) with \( \deg(r) < \deg(b) \), then we say that \( b \) divides \( a \) with quotient \( q \) and remainder \( r \), and \( q \) and \( r \) are denoted \( \text{quo}(a, b) \) and \( \text{rem}(a, b) \).

It is clear that \( q \) and \( r \) exist, and are unique. Division in the previous sense then corresponds to the case \( r = 0 \).

**Theorem 5 (Euclid)** If \( K \) is a field, the univariate polynomials \( K[x] \) form a g.c.d. domain.

**Algorithm 2 (Euclid)**

**Input:** \( f, g \in K[x] \).

**Output:** \( h \in K[x] \) a greatest common divisor of \( f \) and \( g \)

\[
i := 1; \\
\text{if } \deg(f) < \deg(g) \\
\quad \text{then } a_0 := g; a_1 := f; \\
\quad \text{else } a_0 := f; a_1 := g; \\
\text{while } a_i \neq 0 \text{ do} \\
\quad a_{i+1} = \text{rem}(a_{i-1}, a_i); \\
\quad \#q_i := \text{the corresponding quotient: } a_{i+1} = a_{i-1} - q_i a_i \\
\quad i := i + 1; \\
\text{return } a_{i-1};
\]

**Proof.** We must first show that this is an algorithm, i.e. that the potentially infinite loop actually terminates. But \( \deg(a_i) \) is a non-negative integer, strictly decreasing each time round the loop, and therefore the loop must terminate. So \( a_i = 0 \), but \( a_i = \text{rem}(a_{i-2}, a_{i-1}) \), so \( a_{i-1} \) divides \( a_{i-2} \). In fact, \( a_{i-2} = q_{i-1} a_{i-1} \). Now \( a_{i-1} = a_{i-3} - q_{i-2} a_{i-2} \), so \( a_{i-3} = a_{i-1}(1 + q_{i-2} q_{i-1}) \), and so on, until we deduce that \( a_{i-1} \) divides \( a_0 \) and \( a_1 \), i.e. \( f \) and \( g \) in some order. Hence the result
of this algorithm is a common divisor. To prove that it is a greatest common divisor, we must prove that any other common divisor, say $d$, of $f$ and $g$ divides $a_{i-1}$. $d$ divides $a_0$ and $a_1$. Hence it divides $a_2 = a_0 - q_1a_1$. Hence it divides $a_3 = a_1 - q_2a_2$, and so on until it divides $a_{i-1}$.

We should note that our algorithm is asymmetric in $f$ and $g$: if they have the same degree, it is not generally the case that $\gcd(f,g) = \gcd(g,f)$, merely that they are associates.

**Lemma 1** In these circumstances, the result of Euclid’s algorithm is a linear combination of $f$ and $g$, i.e. $a_{i-1} = \lambda_{i-1}f + \mu_{i-1}g$: $\lambda_{i-1}, \mu_{i-1} \in K[x]$.

**Proof.** $a_0$ and $a_1$ are certainly such combinations: $a_0 = 1 \cdot f + 0 \cdot g$ or $1 \cdot g + 0 \cdot f$ and similarly for $a_1$. Then $a_2 = a_0 - q_1a_1$ is also such a combination, and so on until $a_{i-1}$, which is the result.

The above theory, and algorithm, are all very well, but we would like to compute (assuming they exist!) greatest common divisors of polynomials with integer coefficients, polynomials in several variables, etc. So now let $R$ be any g.c.d. domain.

**Definition 30** If $f = \sum_{i=0}^n a_ix^i \in R[x]$, define the content of $f$, written $\text{cont}(f)$, or $\text{cont}_x(f)$ if we wish to make it clear that $x$ is the variable, as $\gcd(a_0, \ldots, a_n)$. Technically speaking, we should talk of a content, but in the theory we tend to abuse language, and talk of the content. Similarly, the primitive part, written $\text{pp}(f)$ or $\text{pp}_x(f)$, is $f / \text{cont}(f)$. $f$ is said to be primitive if $\text{cont}(f)$ is a unit.

**Proposition 17** If $f$ divides $g$, then $\text{cont}(f)$ divides $\text{cont}(g)$ and $\text{pp}(f)$ divides $\text{pp}(g)$. In particular, any divisor of a primitive polynomial is primitive.

The following result is in some sense a converse of the previous sentence.

**Lemma 2 (Gauss)** The product of two primitive polynomials is primitive.

**Proof.** Let $f = \sum_{i=0}^n a_ix^i$ and $g = \sum_{j=0}^m b_jx^j$ be two primitive polynomials, and $h = \sum_{i=0}^{m+n} c_ix^i$ their product. Suppose, for contradiction, that $h$ is not primitive, and $p$ is a prime dividing $\text{cont}(h)$. Suppose that $p$ divides all the coefficients of $f$ up to, but not including, $a_k$, and similarly for $g$ up to but not including $b_l$. Now consider

$$c_{k+l} = a_kb_l + \sum_{i=0}^{k-1} a_ib_{k+l-i} + \sum_{i=k+1}^{k+l} a_ib_{k+l-i} \tag{2.12}$$

(where any indices out of range are deemed to correspond to zero coefficients).
CHAPTER 2. POLYNOMIALS

Since \( p \) divides \( \text{cont}(h) \), \( p \) divides \( c_{k+l} \). By the definition of \( k \), \( p \) divides every \( a_i \) in \( \sum_{i=0}^{k-1} a_i b_{k+l-i} \), and hence the whole sum. Similarly, by definition of \( l \), \( p \) divides every \( b_{k+l-i} \) in \( \sum_{i=0}^{k+l} a_i b_{k+l-i} \), and hence the whole sum. Hence \( p \) divides every term in equation (2.12) except \( a_k b_l \). But, by definition of \( \text{divides} \), it does not divide either \( a_k \) or \( b_l \), and hence cannot divide the product. Hence the hypothesis, that \( \text{cont}(h) \) could be divisible by a prime, is false.

**Corollary 6** \( \text{cont}(fg) = \text{cont}(f)\text{cont}(g) \).

**Theorem 6 ("Gauss' Lemma")** If \( R \) is a g.c.d. domain, and \( f, g \in R[x] \), then \( \text{gcd}(f, g) \) exists, and is \( \text{gcd}(\text{cont}(f), \text{cont}(g)) \text{gcd}(\text{pp}(f), \text{pp}(g)) \).

**Proof.** Since \( R \) is an integral domain, its field of fractions, say \( K \) is a field. Hence, in \( K[x] \) where theorem 5 is applicable, \( \text{pp}(f) \) and \( \text{pp}(g) \) have a greatest common divisor, say \( h \). If \( c \) is any non-zero element of \( R \), then \( ch \) is also a greatest common divisor of \( \text{pp}(f) \) and \( \text{pp}(g) \). Hence we can assume that \( h \) is in \( R[x] \) and, as a polynomial of \( R[x] \), is primitive. In \( K[x] \), \( \text{pp}(f) \) is a multiple of \( h \), say \( \text{pp}(f) = hk \) for \( k \in K[x] \). We can write \( k = dk' \), where \( k' \in R[x] \) and is primitive. Then \( d^{-1}\text{pp}(f) = hk' \). But \( h \) and \( k' \) are primitive, so, by the Lemma, their product is primitive, and \( d \) is a unit. Hence \( h \) is, in \( R[x] \), a common divisor of \( \text{pp}(f) \) and \( \text{pp}(g) \).

But, if \( \bar{h} \) is a common divisor of \( \text{pp}(f) \) and \( \text{pp}(g) \) in \( R[x] \), it is certainly a common divisor of \( f \) and \( g \) in \( K[x] \), hence divides \( h \) in \( K[x] \), and so \( \text{pp}(\bar{h}) \) divides \( h \) in \( R[x] \). Hence \( h \) is a greatest common divisor of \( \text{pp}(f) \) and \( \text{pp}(g) \) in \( R[x] \), and the rest of the theorem is obvious.

This gives us one obvious means of computing g.c.d.s in \( R[x] \), which can be described as "compute in \( K[x] \) and sort out the contents afterwards". More formally it would be Algorithm 3.

**Algorithm 3 (General g.c.d.)**

**Input:** \( f, g \in R[x] \).

**Output:** \( h \in R[x] \) a greatest common divisor of \( f \) and \( g \)

1. \( f_c := \text{cont}_x(f); \ f_p := f/f_c; \ g_c := \text{cont}_x(g); \ g_p := g/g_c. \)
2. \( h := \text{gcd}(f_p, g_p) \) computed in \( K[x] \) by Algorithm 2.
3. \( h_p := \text{pp}(h \times (\text{enough to remove denominators})) \)
4. **return** \( \text{gcd}(f_c, g_c) \times h_p \)

# Correct by reasoning above.

Certainly this is an algorithm, but is it a good one? Let \( k = \max(\deg_x(f), \deg_x(g)) \). In terms of the number of coefficient operations, and in the dense representations, we do \( O(k^2) \) operations. But how big do these coefficients get?

Consider the computation of the g.c.d. of the following two polynomials (this analysis is mostly taken from [Bro71a, Bro71b], but with one change):\(^30\)

\(^30\)Originally an error, but it makes the point better.
−21 instead of +21 for the trailing coefficient of $B$):

$$A(x) = x^8 + x^6 - 3x^4 - 3x^3 + 8x^2 + 2x - 5;$$
$$B(x) = 3x^6 + 5x^4 - 4x^2 - 9x - 21.$$

The first elimination gives $A - \left(\frac{x^2}{3} - \frac{2}{3}\right)B$, that is

$$\frac{-5}{9} \cdot x^4 + \frac{-127}{9} \cdot x^2 - \frac{29}{3},$$

and the subsequent eliminations give

$$\frac{50157}{25} \cdot x^2 - \frac{35847}{25},$$
$$\frac{93060801700}{1557792607653} \cdot x + \frac{23315940650}{173088067517},$$

and, finally,

$$\frac{76103000733847895048691}{86603128130467228900}.$$

Since this is a number, it follows that no polynomial can divide both $A$ and $B$, i.e. that $\text{gcd}(A, B) = 1$.

It is obvious that these calculations on polynomials with rational coefficients require several g.c.d. calculations on integers, and that the integers in these calculations are not always small.

We can eliminate these g.c.d. calculations by working all the time with polynomials with integer coefficients, and this gives a generalisation of the $a_i$ of algorithm 2, known as polynomial remainder sequences or p.r.s., by extending the definition of division.

**Definition 31** Instead of dividing $f$ by $g$ in $K[x]$, we can multiply $f$ by a suitable power of the leading coefficient of $g$, so that the divisions stay in $R$. The pseudo-remainder of dividing $f$ by $g$, written $\text{prem}(f, g)$, is the remainder when one divides $\text{lcm}(g)^{\text{deg}(f) - \text{deg}(g) + 1}f$ by $g$, conceptually in $K[x]$, but in fact all the calculations can be performed in $R$, i.e. all divisions are exact in $R$. This is denoted by $\text{prem}(f, g)$.

In some applications (section 3.1.9) it is necessary to keep track of the signs: we define a signed polynomial remainder sequence or s.p.r.s. of $f_0 = f$ and $f_1 = g$ to have $f_i$ proportional by a positive constant to $-\text{rem}(f_{i-2}, f_{i-1})$.

This gives us a pseudo-euclidean algorithm, analogous to algorithm 2, where we replace rem by prem, and fix up the contents afterwards. In the above example, we deduce the following sequence:

$$-15x^4 + 381x^2 - 261,$$

31This definition agrees with Maple, but not with all software systems, which often use $\text{prem}$ to denote what Maple calls $\text{sprem}$, i.e. only raising $\text{lcm}(g)$ to the smallest power necessary.
6771195 \ x^2 - 30375 \ x - 4839345,  
50074529585202821250 \ x + 1129134141014747231250 
and 
7436622422540486538114177255855890572956445312500. 

Again, this is a number, so \( \gcd(A, B) = 1 \). We have eliminated the fractions, 
but at a cost of even larger numbers. Can we do better?

### 2.3.2 Subresultant sequences

One option would be to make the \( a_i \) primitive at each step, since we are going to 
fix up the content terms later: giving the so-called \textit{primitive p.r.s.} algorithm, 
which in this case would give 

\[
-5x^4 + 127x^2 - 87; 5573x^2 - 25x - 3983; -4196868317x - 1861216034; 1
\]

This is a perfectly reasonable algorithm when it is a question of polynomials in 
one variable, and is essentially equivalent to calculating with rational numbers, 
but over a common denominator. However, if we come to polynomials in several 
variables, every step of the g.c.d. for polynomials in \( n \) variables would involve 
the g.c.d. of several polynomials in \( n - 1 \) variables, each step of each of which 
would involve the g.c.d. of several polynomials in \( n - 2 \) variables, and so on.

The following, slightly mysterious\(^{32}\), algorithm will do the trick. By the 
Subresultant Theorem \[Loo82\], all divisions involved are exact, i.e. we always 
stay in \( \mathbb{R}[x] \). Furthermore, the factors \( \beta_i \) that are cancelled are \textit{generically as large as possible}, where by “generically” we mean that, if the coefficients of \( f \) and \( g \) were all independent, nothing more could be cancelled\(^{33}\). In the same 
example as before, we get the following:

\[
\begin{align*}
a_2 &= 15x^4 - 381x^2 + 261, \\
a_3 &= -27865x^2 + 125x + 19915, \\
a_4 &= -3722432068x - 8393738634, \\
a_5 &= 1954124052188. 
\end{align*}
\]

Here the numbers are much smaller, and indeed it can be proved that the 
coefficient growth is only linear in the step number. \( a_2 \) has a content of 3, 
which the primitive p.r.s. would eliminate, but this content in fact disappears 
later. Similarly \( a_3 \) has a content of 5, which again disappears later. Hence we
have the following result.

\[^{32}\text{Some of the mystery is explained by corollary 4 on page 85. In particular the various }-\text{signs, which are irrelevant as far as a strict g.c.d. algorithm is concerned, come from corollary 4.}\]

\[^{33}\text{The reader may comment that the example, repeated below with this algorithm, shows a consistent factor of 3 in } a_2, \text{ and this is true however the non-zero coefficients are perturbed. Indeed, if the leading coefficient of } a_1 \text{ is changed to, say, 4, we get a consistent factor of 4. However, if the coefficient of } x^7 \text{ in } a_0 \text{ is made non-zero, then the common factor will generally go away, and that is what we mean by “generically”.}\]
Algorithm 4 (Subresultant p.r.s.)

Input: \( f, g \in K[x] \).

Output: \( h \in K[x] \) a greatest common divisor of \( \text{pp}(f) \) and \( \text{pp}(g) \).

Comment: If \( f, g \in R[x] \), where \( R \) is an integral domain and \( K \) is the field of fractions of \( R \), then all computations are exact in \( R[x] \). This can therefore fulfil the rôle of steps 2–3 of Algorithm 3.

\[
i := 1;
\]

if \( \deg(f) < \deg(g) \)

then \( a_0 := \text{pp}(g); a_1 := \text{pp}(f); \)

else \( a_0 := \text{pp}(f); a_1 := \text{pp}(g); \)

\( \delta_0 := \deg(a_0) - \deg(a_1); \)

\( \beta_2 := (-1)^{\delta_0+1}; \)

\( \psi_2 := -1; \)

while \( a_i \neq 0 \) do

\( a_{i+1} := \text{prem}(a_{i-1}, a_i)/\beta_{i+1}; \)

\( \#q_i := \text{the corresponding quotient}; \)

\( a_{i+1} = \text{lc}(a_i)^{\delta_i-1} a_{i-1} - q_i a_i \)

\( \delta_i := \deg(a_i) - \deg(a_{i+1}); \)

\( i := i + 1; \)

\( \psi_{i+1} := (-\text{lc}(a_{i-1}))^{\delta_i-2} \psi_i^{1-\delta_i-2}; \)

\( \beta_{i+1} := -\text{lc}(a_{i-1}) \psi_i^{\delta_i-1}; \)

return \( \text{pp}(a_{i-1}); \)

The \( a_i \) are referred to as a subresultant polynomial remainder sequence.
Theorem 7 Let $R$ be a g.c.d. domain. Then there is an algorithm to calculate the g.c.d. of polynomials in $R[x]$. If the original coefficients have length bounded by $B$, the length at the $i$-th step is bounded by $iB$. Strictly speaking, this theorem is true for polynomials in several variables, where “length” is replaced by “degree in variables other than $x$”. Over the integers, we need to add $\log(2^i!)$ to allow for the fact that the sum of two integers can be larger than either.

This algorithm is the best method known for calculating the g.c.d., of all those based on Euclid’s algorithm applied to polynomials with integer coefficients. In chapter 4 we shall see that if we go beyond these limits, it is possible to find better algorithms for this calculation.

2.3.3 The Extended Euclidean Algorithm

We can in fact do more with the Euclidean algorithm. Consider the following variant of Algorithm 2, where we have added a few extra lines, marked (*), manipulating $(a, b)$, $(c, d)$ and $(e, e')$.

Algorithm 5 (Extended Euclidean)

Input: $f, g \in K[x]$.  
Output: $h \in K[x]$ a greatest common divisor of $f$ and $g$, and $c, d \in K[x]$ such that $cf + dg = h$.

\[
i := 1; \\
\text{if } \deg(f) < \deg(g) \text{ then } a_0 := g; \quad a_1 := f; \\
\text{else } a_0 := f; \quad a_1 := g; \\
\text{while } a_i \neq 0 \text{ do} \\
\quad \text{#Loop invariant: } a_i = af + bg; \quad a_{i-1} = cf + dg; \\
\quad a_{i+1} := \text{rem}(a_{i-1}, a_i); \\
\quad q_i := \text{the corresponding quotient: } \#a_{i+1} = a_{i-1} - q_ia_i \\
\quad e := c - q_ia; \quad e' := d - q_ib; \\
\quad i := i + 1; \\
\text{return } (a_i, c, d); \\
\text{(*3)} \quad (a, b) := (a, b); \\
\text{(*3)} \quad (a, b) := (e, e')
\]

The comments essentially form the proof of correctness. In particular, if $f$ and $g$ are relatively prime, there exist $c$ and $d$ such that

\[cf + dg = 1: \quad (2.13)\]

a result often called Bezout’s identity.

For the sake of further developments (section 3.3.6), we can express the marked lines as

\[\begin{pmatrix} a & b \\ c & d \end{pmatrix} := \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \text{ or } \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad (2.14)\]
2.3. GREATEST COMMON DIVISORS

\[(*)_2 \quad \left( \begin{array}{c} a_i \\ a_{i+1} \end{array} \right) = \left( \begin{array}{cc} a & b \\ c & d \end{array} \right) \left( \begin{array}{c} f \\ g \end{array} \right) \] (2.15)

\[(*)_3 \quad \left( \begin{array}{cc} a & b \\ c & d \end{array} \right) := \left( \begin{array}{cc} 0 & 1 \\ 1 & -q_i \end{array} \right) \left( \begin{array}{cc} a & b \\ c & d \end{array} \right) \] (2.16)

It is possible to make similar modifications to algorithm [3] and the same theory that shows the division by \( \beta_{i+1} \) is exact shows that we can perform the same division of \((e, e')\). However, at the end, we return \( \text{pp}(a_{i-1}) \), and the division by \( \text{cont}(a_{i-1}) \) is not guaranteed to be exact when applied to \((a, b)\).

Algorithm 6 (General extended p.r.s.)

Input: \( f, g \in K[x] \).
Output: \( h \in K[x] \) a greatest common divisor of \( \text{pp}(f) \) and \( \text{pp}(g) \), \( h' \in K \) and \( c, d \in K[x] \) such that \( cf + dg = h'h \).

Comment: If \( f, g \in R[x] \), where \( R \) is an integral domain and \( K \) is the field of fractions of \( R \), then all computations are exact in \( R[x] \), and \( h' \in R, c, d \in R[x] \).

\[ i := 1; \]
\[ \text{if } \text{deg}(f) < \text{deg}(g) \]
\[ \quad \text{then } a_0 := \text{pp}(g); a_1 := \text{pp}(f); \]
\[ \quad a := 1; d := 1; b := c := 0 \]
\[ \text{else } a_0 := \text{pp}(f); a_1 := \text{pp}(g); \]
\[ c := 1; b := 1; a := d := 0 \]
\[ \delta_0 := \text{deg}(a_0) - \text{deg}(a_1); \]
\[ \beta_2 := (-1)^{b+1}; \]
\[ \psi_2 := 1; \]
\[ \text{while } a_i \neq 0 \text{ do} \]
\[ \quad \text{#Loop invariant: } a_i = af + bg; a_{i-1} = cf + dg; \]
\[ \quad a_{i+1} = \text{prem}(a_{i-1}, a_i)/\beta_{i+1}; \]
\[ \quad \text{#q}_i := \text{the corresponding quotient: } a_{i+1} = \text{lc}(a_i)^{\delta_{i-1}}a_{i-1} - q_ia_i \]
\[ \quad \delta_i := \text{deg}(a_i) - \text{deg}(a_{i+1}); \]
\[ \quad e := (c - q_ia)/\beta_{i+1}; \]
\[ \quad e' := (d - q_ib)/\beta_{i+1}; \quad \#a_{i+1} = ef + e'g \]
\[ \quad i := i + 1; \]
\[ \quad (c, d) = (a, b); \]
\[ \quad (a, b) = (e, e') \]
\[ \quad \psi_{i+1} := -\text{lc}(a_{i-1})^{\delta_{i-2}}\psi_i^{1-\delta_{i-2}}; \]
\[ \beta_{i+1} := -\text{lc}(a_{i-1})\psi_{i+1}^{1-\delta_{i-1}}; \]
\[ \text{return } (\text{pp}(a_{i-1}), \text{cont}(a_{i-1}), c, d); \]

2.3.4 Partial Fractions

Bezout’s Identity [2.13] has a useful consequence. Suppose we have a fraction \( p/q \), and \( q = fg \) with \( \text{gcd}(f, g) = 1 \). Then

\[ \frac{p}{gf} = \frac{p(cf + dg)}{fg} = \frac{pc}{g} + \frac{pd}{f}. \] (2.17)
Even if $p/q$ is proper, i.e. $\deg p < \deg q$, the same may not be true of $pc/g$ or $pd/f$. However, if the left-hand side of (2.17) is proper, so must the right-hand side be when collected over a common denominator. Hence the extents to which $pc/g$ and $pd/f$ are improper must cancel. So

$$\frac{p}{q} = \frac{pc \text{ rem } g}{g} + \frac{pd \text{ rem } f}{f}. \quad (2.18)$$

A decomposition of the form of the right-hand side of (2.18) is called a partial fraction decomposition. This can clearly be extended to any number of factors of the denominator, i.e.

$$\frac{p}{\prod_{i=1}^{n} f_i} = \sum_{i=1}^{n} \frac{p_i}{f_i}. \quad (2.19)$$

There is one important caution: even if $p,q \in \mathbb{Z}[x]$, this need not be the case for the partial fraction decomposition, for example

$$\frac{1}{x^2 - 1} = \frac{1}{2x - 1} + \frac{-1}{2x + 1}.$$

### 2.3.5 Polynomials in several variables

Here it is best to regard the polynomials as recursive, so that $R[x,y]$ is regarded as $R[y][x]$. In this case, we now know how to compute the greatest common divisor of two bivariate polynomials.

**Algorithm 7 (Bivariate g.c.d.)**

**Input:** $f, g \in R[y][x]$.

**Output:** $h \in R[y][x]$ a greatest common divisor of $f$ and $g$.

$h_c := \text{g.c.d. of } \text{cont}_x(f) \text{ and } \text{cont}_x(g)$  

# this is a g.c.d. computation in $R[y]$.  
$h_p := \text{algorithm} 7(pp_x(f), pp_x(g))$  

# replacing $R$ by $R[y]$, which we know, by theorem 7, is a g.c.d. domain.  
return $h_h_p$  

# which by theorem 6 is a g.c.d. of $f$ and $g$.

This process generalises.

**Theorem 8** If $R$ is a g.c.d. domain, then $R[x_1, \ldots, x_n]$ is also a g.c.d. domain.

**Proof.** Induction on $n$, with theorem 7 as the building block.

What can we say about the complexity of this process? It is easier to analyse if we split up the division process which computes rem$(a_{i-1}, a_i)$ into a series of repeated subtractions of shifted scaled copies of $a_i$ from $a_{i-1}$. Each such subtraction reduces deg$(a_{i-1})$, in general by 1. For simplicity, we shall assume that the reduction is precisely by 1, and that $\deg(a_{i+1}) = \deg(a_i) - 1$. It also

\[\beta_i = \pm l_c(a_{i-2})^2.\]

In fact, the sub-resultant algorithm was first developed for normal p.r.s., where it can be seen as a consequence of the Dodgson–Bareiss Theorem (theorem 15).
turns out that the polynomial manipulation in \( x \) is the major cost (this is not the case for the primitive \( \text{p.r.s} \), where the recursive costs of the content computations dominates), so we will skip all the other operations (the proof of this is more tedious than enlightening). Let us assume that \( \deg_x(f) + \deg_x(g) = k \), and that the coefficients have maximum degree \( d \). Then the first subtraction will reduce \( k \) by 1, and replace \( d \) by \( 2d \), and involve \( k \) operations on the coefficients. The next step will involve \( k - 1 \) operations on coefficients of size \( 2d \). The next step combines one of the original polynomials, with coefficients of degree \( d \), with this polynomial with coefficients of degree \( 2d \), giving a polynomial with coefficients of degree \( 3d \). Combining this with a polynomial with coefficients of degree \( 2d \) ought to give us coefficients of degree \( 5d \), but in fact we divide by the previous leading coefficient (degree \( d \)) so the answer is a polynomial with coefficients of degree \( 4d \), and so on, giving degree \( id \) at the \( i \)-th step (for a matrix view of this, see Corollary 6 and the discussion after it), and a total cost of \( \sum_{i=0}^{k} (k-i)F(id) \), where \( F(d) \) is the cost of operating on coefficients of degree \( d \). Let us suppose that there are \( v \) variables in all: \( x \) itself and \( v-1 \) variables in the coefficients with respect to \( x \).

\( v = 2 \) Here the coefficients are univariate polynomials. If we assume classic multiplication on dense polynomials, \( F(d) = cd^2 + O(d) \). We are then looking at

\[
\sum_{i=0}^{k} (k-i)F(id) \leq c \sum_{i=0}^{k} (k-i)i^2d^2 + \sum_{i=0}^{k} kO(id)
\]

\[
\leq ck \sum_{i=0}^{k} i^2d^2 - c \sum_{i=0}^{k} i^3d^2 + k^3O(d)
\]

\[
= c \left( \frac{1}{3} k^4 + \frac{1}{2} k^3 + \frac{1}{6} k^2 \right) d^2 - c \left( \frac{1}{4} k^4 + \frac{1}{2} k^3 + \frac{1}{4} k^2 \right) d^2 + k^3O(d)
\]

\[
= c \left( \frac{1}{12} k^4 - \frac{1}{12} k^2 \right) d^2 + k^3O(d)
\]

which we can write as \( O(k^4d^2) \). We should note the asymmetry here: this means that we should choose the principal variable (i.e. the \( x \) in algorithm \( \Box \)) to be whichever of \( x \) and \( y \) minimises

\[
\min(\max(\deg_x(f), \deg_x(g)), \max(\deg_y(f), \deg_y(g))).
\]

\( v = 3 \) Here the coefficients are bivariate polynomials. If we assume classic multiplication on dense polynomials, \( F(d) = cd^4 + O(d^3) \). We are then looking at

\[
\sum_{i=0}^{k} (k-i)F(id) \leq c \sum_{i=0}^{k} (k-i)i^4d^4 + \sum_{i=0}^{k} kO(i^3d^3)
\]

\[
\leq ck \sum_{i=0}^{k} i^4d^4 - c \sum_{i=0}^{k} i^5d^4 + k^5O(d^3)
\]
which we can write as $O(k^6d^4)$. The asymmetry is again obvious.

general $v$ The same analysis produces $O(k^{2v}d^{2v-2})$.

We see that the cost is exponential in $v$, even though it is polynomial in $d$ and $k$. This is not a purely theoretical observation: any experiment with several variables will bear this out, even when the inputs (being sparse) are quite small: the reader need merely use his favourite algebra system on

\[ a_0 := ax^4 + bx^3 + cx^2 + dx + e; \quad a_1 := fx^4 + gx^3 + hx^2 + ix + j, \]

treating $x$ as the main variable (which of course one would not do in practice), to see the enormous growth of the coefficients involved.

### 2.3.6 Square-free decomposition

Let us revert to the case of polynomials in one variable, $x$, over a field $K$, and let us assume that $\text{char}(K) = 0$ (see definition [15] — the case of characteristic non-zero is more complicated [DT81], and we really ought to talk about ‘separable decomposition’ [Lec08]).

**Definition 32** The formal derivative of $f(x) = \sum_{i=0}^{n} a_i x^i$ is written $f'(x)$ and computed as $f'(x) = \sum_{i=1}^{n} ia_i x^{i-1}$.

This is what is usually referred to as the derivative of a polynomial in calculus texts, but we are making no appeal to the theory of differentiation here: merely defining a new polynomial whose coefficients are the old ones (except that $a_0$ disappears) multiplied by the exponents, and where the exponents are decreased by 1.

**Proposition 18** The formal derivative satisfies the usual laws:

\[ (f + g)' = f' + g' \quad (fg)' = f'g + fg'. \]

**Proof.** By algebra from the definition. This is taken up in more generality in Proposition [67].

Let us consider the case $f = g^n h$, where $g$ and $h$ have no common factors. Then $f' = g^n h' + ng^{n-1}g'h$ and is clearly divisible by $g^{n-1}$. $n$ is not zero in $K$ (by the assumption on $\text{char}(K)$), so $g$ does not divide $f'/g^{n-1} = gh' + ng'h$. Hence $\gcd(f, f')$ is divisible by $g^{n-1}$ but not by $g^n$. These considerations lead to the following result.
2.3. GREATEST COMMON DIVISORS

Proposition 19 Let \( f = \prod_{i=1}^{k} f_i^{n_i} \) where the \( f_i \) are relatively prime and have no repeated factors. Then

\[
gcd(f, f') = \prod_{i=1}^{k} f_i^{n_i - 1}.
\]

Definition 33 The square-free decomposition of a polynomial \( f \) is an expression

\[
f = \prod_{i=1}^{n} f_i^i
\]

where the \( f_i \) are relatively prime and have no repeated factors. \( f \) is said to be square-free if \( n = 1 \).

Let that some of the \( f_i \) may be 1.

Lemma 3 Such a decomposition exists for any non-zero \( f \), and can be calculated by means of gcd computations and divisions.

Proof. Let \( g = \gcd(f, f') = \prod_{i=1}^{n} f_i^{i-1} \) by the previous proposition. Then \( f/g = \prod_{i=1}^{n} f_i \) and \( \gcd(g, f/g) = \prod_{i=2}^{n} f_i \). Hence

\[
\frac{f/g}{\gcd(g, f/g)} = f_1.
\]

Applying the same process to \( g \) will compute \( f_2 \), and so on.

This is not in fact the most efficient way of computing such a decomposition: a better method was given by Yun [Yun76].

2.3.7 Sparse Complexity

So far we have, implicitly, considered dense polynomials. What if the polynomials are sparse? There are then various embarrassing possibilities that

1. the gcd \( g \) might be much denser than the inputs \( A \) and \( B \)

2. even if the gcd \( g \) is not much denser than the inputs \( A \) and \( B \), the cofactors \( A/g \) and \( B/g \) computed as part of the verification might be much denser

3. even if neither of these happens, various intermediate results might be much denser than \( A \) and \( B \).

Problem 2 is easy to demonstrate: consider the cofactors in \( \gcd(x^p - 1, x^q - 1) \), where \( p \) and \( q \) are distinct primes. Problem 4 is demonstrated by the following elegant example of [Sch03a]

\[
gcd(x^{pq} - 1, x^{p+q} - x^p - x^q + 1) = \frac{(x^p-1)(x^q-1)}{x^{p+q-1} + x^{p+q-2} + \cdots + 1},
\]

and indeed just knowing whether two polynomials have a non-trivial gcd is hard, by the following result.
Theorem 9 ([Pla77]) It is NP-hard to determine whether two sparse polynomials (in the standard encoding) have a non-trivial common divisor.

This theorem, like the examples above, relies on the factorisation of \(x^p - 1\), and it is an open question [DC10, Challenge 3] whether this is the only obstacle. More precisely, we have the following equivalent of the problem solved for division.

**Open Problem 2 (Sparse gcd (strong))** Find an algorithm for computing \(h = \gcd(f, g)\) which is polynomial-time in \(t_f, t_g\) and \(t_h\). See [DC10, Challenge 5].

For multivariate polynomials, we seem to have a solution in practice (page 177) in the sense that its running time depends on the number of actual terms in the multivariate g.c.d., rather than the potential number of terms.

A weaker problem, but still unsolved, is

**Open Problem 3 (Sparse gcd (weak))** Find an algorithm for computing \(h = \gcd(f, g)\) which is polynomial-time in \(t_f, t_g, t_h, t_{f/h}, t_{g/h}\).

**Observation 2** It follows from (2.21) that square-free decompositions are also hard in the sense that the number of terms in the output is unbounded in the number of terms in the input:

\[
\left(\frac{x^{pq} + x^p + x^q + x^p + x^q + x^p + x^q - 1}{8 \text{ terms}}\right) (x^{pq} - 1) (x^{pq} - x^p - x^q + 1) = (x - 1) (x^{pq} - 2x^p + 2x^q - 3 + \cdots + 1)^2 (\ldots)
\]

Furthermore, the largest coefficient in the marked term of multiplicity 2 is \(\min(p, q)\), quashing any hopes that the coefficients of a square-free decomposition might be bounded in terms only of the coefficients of the input.

## 2.4 Non-commutative polynomials

The label “non-commutative polynomials” in fact covers three cases.

1. The indeterminates commute, but the coefficients do not. For definiteness, we will refer to this case as polynomials with non-commuting coefficients.

In this case, rule 5 of definition 19 has to be replaced by

\[ x * y = y * x; \]

where \(x\) and \(y\) are indeterminates, not general polynomials. This means that some of the traditional laws of algebra cease to operate: for example

\[(ax + b)(ax - b) = a^2x^2 - b^2\]

becomes

\[(ax + b)(ax - b) = a^2x^2 + (-a * b + b * a)x - b^2\]
2. The coefficients commute, but the indeterminates do not. For definiteness, we will refer to this case as *polynomials with non-commuting indeterminates*. In this case, rule 5 of definition 19 has to be replaced by the assumption

- \( m \otimes n = n \otimes m \).

At this point, many of the traditional laws of algebra cease to operate: even the Binomial Theorem in the form

\[
(x + y)^2 = x^2 + 2xy + y^2
\]

has to be replaced by

\[
(x + y)^2 = x^2 + (xy + yx) + y^2.
\]

A common case of non-commuting indeterminates is in differential algebra, where the variable \( x \) and the differentiation operator \( \frac{d}{dx} \) do not commute, but rather satisfy the equation

\[
\frac{d}{dx} (xa) = x \frac{da}{dx} + a. \tag{2.23}
\]

3. Neither can be assumed to commute, in which case rule 5 of definition 19 is just deleted, with no replacement.

**Notation 17** If the variables do not commute, it is usual to use the notation \( R\langle x_1, \ldots, x_n \rangle \) for the ring of polynomials with coefficients in \( R \) and the non-commuting variables \( x_1, \ldots, x_n \).
Chapter 3

Polynomial Equations

In the first parts of this chapter, we will deal with polynomial equations, either singly or as sets of equations. A preliminary remark is in order. Any polynomial equation

\[ A = B, \quad (3.1) \]

where \( A \) and \( B \) are polynomial equations, can be reduced to one whose right-hand side is zero, i.e.

\[ A - B = 0. \quad (3.2) \]

**Notation 18** Henceforth, all polynomial equations will be assumed to be in the form of (3.2).

### 3.1 Equations in One Variable

We may as well assume that the unknown variable is \( x \). If the equation is linear in \( x \) then, by the notation above, it takes the form

\[ ax + b = 0. \quad (3.3) \]

The solution is then obvious: \( x = -b/a \).

#### 3.1.1 Quadratic Equations

Again, by the notation above, our equation takes the form

\[ ax^2 + bx + c = 0. \quad (3.4) \]

The solutions are well-known to most schoolchildren: there are two of them, of the form

\[ x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}. \quad (3.5) \]
However, if $b^2 - 4ac = 0$, i.e. $c = b^2/4a$ then there is only one solution: $x = -b/2a$. In this case, the equation becomes $ax^2 + bx + b^2/4a = 0$, which can be re-written as $a(x + b/2a)^2 = 0$, making it more obvious that there is a repeated root, and that the polynomial is not square-free (definition 33).

Mathematicians dislike the sort of anomaly in “this equation has two solutions except when $c = b^2/4a$”, especially as there are two roots as $c$ tends to the value $b^2/4a$. We therefore say that, in this special case, $x = -b/2a$ is a double root of the equation. This can be generalised, and made more formal.

**Definition 34** If, in the equation $f = 0$, $f$ has a square-free decomposition $f = \prod_{i=1}^n f_i$, and $x = \alpha$ is a root of $f_i$, we say that $x = \alpha$ is a root of $f$ of multiplicity $i$. When we say we are counting the roots of $f$ with multiplicity, we mean that $x = \alpha$ should be counted $i$ times.

**Proposition 20** The number of roots of a polynomial equation over the complex numbers, counted with multiplicity, is equal to the degree of the polynomial.

**Proof.** $\deg(f) = \sum i\deg(f_i)$, and each root of $f_i$ is to be counted $i$ times as a root of $f$. That $f_i$ has $i$ roots is the so-called Fundamental Theorem of Algebra.

In this case, the two roots are given by the two possible signs of the square root, and $\sqrt{0}$ is assumed to have both positive and negative signs.

### 3.1.2 Cubic Equations

There is a formula for the solutions of the cubic equation

$$x^3 + ax^2 + bx + c,$$  

(3.6)

albeit less well-known to schoolchildren:

$$\frac{1}{6}\sqrt[3]{36ba - 108c - 8a^3 + 12\sqrt{12b^3 - 3b^2a^2 - 54bac + 81c^2 + 12ca^3} - \frac{2b - \frac{2}{3}a^2}{\sqrt[3]{36ba - 108c - 8a^3 + 12\sqrt{12b^3 - 3b^2a^2 - 54bac + 81c^2 + 12ca^3}}} - 1 \frac{1}{3}} a. \tag{3.8}$$

We can simplify this by making a transformation\(^1\) to equation (3.6), replacing $x$ by $x - \frac{a}{3}$. This transforms it into an equation

$$x^3 + bx + c$$  

(3.7)

(where $b$ and $c$ have changed). This has solutions of the form

$$\frac{1}{6}\sqrt[3]{-108c + 12\sqrt{12b^3 + 81c^2}} - \frac{2b}{\sqrt[3]{-108c + 12\sqrt{12b^3 + 81c^2}}} \tag{3.8}$$

\(^1\)This is the simplest case of the Tschirnhaus transformation\(^{[T38]}\), which can always eliminate the $x^{n-1}$ term in a polynomial of degree $n$. 

---

1. Tschirnhaus transformation: A method in algebra for transforming a polynomial equation into another equation of the same degree with fewer terms. It was introduced by Fermat and later developed by Tschirnhaus. The transformation aims to eliminate the highest degree term of the equation, making it easier to solve the equation.

2. [T38]: Reference to a possible source or textbook on advanced algebra, possibly “Treatise on Algebra” by Fermat, where the Tschirnhaus transformation is discussed.
3.1. EQUATIONS IN ONE VARIABLE

\[
S := \sqrt{12b^3 + 81c^2};
\]
\[
T := \sqrt[3]{-108c + 12S};
\]
\[
\text{return } \frac{1}{6}T - \frac{2b}{T};
\]

Figure 3.1: Program for computing solutions to a cubic

Now a cubic is meant to have three roots, but a naïve look as equation (3.8) shows two cube roots, each with three values, and two square roots, each with two values, apparently giving a total of \(3 \times 3 \times 2 \times 2 = 36\) values. Even if we decide that the two occurrences of the square root should have the same sign, and similarly the cube root should have the same value, i.e. we effectively execute the program in figure 3.1, we would still seem to have six possibilities. In fact, however, the choice in the first line is only apparent, since

\[
\frac{1}{6} \sqrt[3]{-108c - 12 \sqrt{12b^3 + 81c^2}} = \frac{2b}{\sqrt[3]{-108c + 12 \sqrt{12b^3 + 81c^2}}}. \tag{3.9}
\]

In the case of the quadratic with real coefficients, there were two real solutions if \(b^2 - 4ac > 0\), and complex solutions otherwise. However, the case of the cubic is more challenging. If we consider \(x^3 - 1 = 0\), we compute (in figure 3.1)

\[
S := 9; \quad T := 6; \quad \text{return } 1;
\]
(or either of the complex cube roots of unity if we choose different values of \(T\)).

If we consider \(x^3 + 1 = 0\), we get

\[
S := 9; \quad T := 0; \quad \text{return } "0";
\]
but we can (and must!) take advantage of equation (3.9) and compute

\[
S := -9; \quad T := -6; \quad \text{return } -1;
\]
(or either of the complex variants).

For \(x^3 + x\), we compute

\[
S := \sqrt[3]{12}; \quad T := \sqrt[3]{12}; \quad \text{return } 0;
\]
and the two complex roots come from choosing the complex roots in the computation of \(T\), which is really \(\sqrt[3]{12 \sqrt[3]{12}}\).

\(x^3 - x\) is more challenging: we compute

\[
S := \sqrt[3]{-12}; \quad T := \sqrt[3]{-12}; \quad \text{return } \{ -1, 0, 1 \}; \tag{3.10}
\]
i.e. three real roots which can only be computed (at least via this formula) by means of complex numbers. In fact it is clear that any other formula must have the same problem, since the only choices of ambiguity lie in the square and cube roots, and with the cube root, the ambiguity involves complex cube roots of unity.
3.1.3 Quartic Equations

Here the equation would be \( x^4 + ax^3 + bx^2 + cx + d \), but after the Tschirnhaus transformation \( x \to x - \frac{a}{4} \), analogous to that which took equation (3.6) to (3.7), we can assume that \( a = 0 \). A truly marvellous solution then looks as follows (but the page is too small to contain it!).

\[
\sqrt[6]{\frac{-4b\sqrt{-288db + 108c^2 + 8b^3 + 12\sqrt{-768d^3 + 384d^2b^2 - 48db^4 - 432dcb^2 + 81c^4 + 12c^2b^5}}}{12}} \\
\sqrt[3]{-288db + 108c^2 + 8b^3 + 12\sqrt{-768d^3 + 384d^2b^2 - 48db^4 - 432dcb^2 + 81c^4 + 12c^2b^5}} + (-48db^4 - 432dcb^2 + 81c^4 + 12c^2b^5)
\]

We can adopt the same formulation as in Figure 3.1 as shown in figure 3.2. Here

\[
S := \sqrt{-768d^3 + 384d^2b^2 - 48db^4 - 432dcb^2 + 81c^4 + 12c^2b^5} \\
T := \sqrt[3]{-288db + 108c^2 + 8b^3 + 12S} \\
U := \sqrt[3]{-4bT + T^2 + 48d + 4b^2} \\
\text{return} \ \frac{\sqrt[6]{6}}{12}U + \frac{\sqrt[6]{6}}{12}\sqrt[3]{-\frac{8bTU + UT^2 + 48Ud + 4Ub^2 + 12c\sqrt[3]{6}}{TU}}
\]

Figure 3.2: Program for computing solutions to a quartic

the problem of multiple choices is even more apparent, but in this formulation it turns out that choices cancel, much as in the case of the cubic. We have the same problem as in the case of the cubic, that real solutions can arise from complex intermediates, but also that the answer apparently involves \( \sqrt[6]{6} \), even though it clearly need not do so in reality. For example, with \( x^4 - 5x^2 + 4 \), whose solutions are \( \pm 1, \pm 2 \), we can evaluate

\[
S := 72\sqrt{-3}; \quad T := 17 + \sqrt{-3}; \quad U := 3\sqrt{6}; \quad \text{return} \ 2; \quad (3.12)
\]

taking the other square root at the end gives 1, and taking the other square root when computing \( U \) gives \(-1 \) or \(-2 \). We should also note that \( T \) was evaluated as \( \sqrt[3]{4760 + 864\sqrt{-3}} \): not entirely obvious.

3.1.4 Higher Degree Equations

When it comes to higher degree equations, the situation is very different.

**Theorem 10 (Abel, Galois \([\text{Gal79}]\))** The general polynomial equation of degree 5 or more is not soluble in radicals (i.e. in terms of \( k \)-th roots).
3.1. EQUATIONS IN ONE VARIABLE

In fact, if we choose such a polynomial “at random”, the probability of its having a solution that can be expressed in terms of radicals is zero. Of course, any particular quintic, or higher degree equation, may have solutions expressible in radicals, such as \( x^5 - 2 \), whose solutions are \( \sqrt[5]{2} \), but this is the exception rather than the rule.

Hence algebra systems, if they handle such concepts, can only regard the roots of such equations as being defined by the polynomial of which they are a root. A Maple example is given in figure 1.2, where the Maple operator \texttt{RootOf} is generated. It is normal to insist that the argument to \texttt{RootOf} (or its equivalent) is square-free: then differently-indexed roots are genuinely different. Then \( \alpha \), the first root of \( f(x) \), satisfies \( f(\alpha) = 0 \), the second root \( \beta \) satisfies \( f(x)/(x - \alpha) = 0 \), and so on. Even if \( f \) is irreducible, these later polynomials may not be, but determining the factorisations if they exist is a piece of Galois theory which would take us too far out of our way [FM89]. It is, however, comparatively easy to determine the Monte Carlo question: “such factorisations definitely do not exist”/“they probably do exist” [DS00].

3.1.5 Reducible defining polynomials

It should be noted that handling such constructs when the defining polynomial is not irreducible can give rise to unexpected results. For example, in Maple, if \( \alpha \) is \texttt{RootOf}(\( x^2-1 \),\( x \)), then \( 1/\alpha \) returns that, but attempting to evaluate this numerically gives infinity, which is right if \( \alpha = 1 \), but wrong if \( \alpha = -1 \), the other, equally valid, root of \( x^2 - 1 \). In this case, the mathematical answer to “is \( \alpha - 1 \) zero?” is neither ‘yes’ nor ‘no’, but rather ‘it depends which \( \alpha \) you mean’, and Maple is choosing the 1 value (as we can see from \( 1/\alpha + 1 \), which evaluates to 0.5). However, the ability to use polynomials not guaranteed to be irreducible can be useful in some cases — see section 3.3.7. In particular, algorithm 11 asks if certain expressions are invertible, and a ‘no’ answer here entails a splitting into cases, just as asking “is \( \alpha - 1 \) zero?” entails a splitting of \texttt{RootOf}(\( x^2-1 \),\( x \)).

In general, suppose we are asking if \( g(\alpha) \) is invertible, where \( \alpha = \texttt{RootOf}(f(x), x) \), i.e. we are asking for \( d(\alpha) \) such that \( d(\alpha)g(\alpha) = 1 \) after taking account of the fact that \( \alpha = \texttt{RootOf}(f(x), x) \). This is tantamount to asking for \( d(x) \) such that \( d(x)g(x) = 1 \) modulo \( f(x) = 0 \), i.e. \( d(x)g(x) + c(x)f(x) = 1 \) for some \( c(x) \). But applying the Extended Euclidean Algorithm (Algorithm 5) to \( f \) and \( g \) gives us \( c \) and \( d \) such that \( cf + dg = \gcd(f, g) \). Hence if the \( \gcd \) is in fact 1, \( g(\alpha) \) is invertible, and we have found the inverse.

If in fact the \( \gcd \) is not 1, say some \( h(x) \neq 1 \), then we have split \( f \) as \( f = h\hat{h} \),

---

\footnote{The precise statement is as follows. For all \( n \geq 5 \), the fraction of polynomials of degree \( n \) and coefficients at most \( H \) which have a root expressible in radicals tends to zero as \( H \) tends to infinity.}

\footnote{By default, Maple will also use this formulation for roots of most quartics, and the expression in figure 3.2 is obtained by \texttt{convert}(%\,\texttt{,radical}) and then locating the sub-expressions by hand. This can be seen as an application of Carette’s view of simplification (page 23), though historically Carette’s paper is a retrospective justification.}
where $\hat{h} = f/h$. Now
\[
\alpha = \text{RootOf}(f(x), x) \iff \alpha = \text{RootOf}(h(x), x) \vee \alpha = \text{RootOf}(\hat{h}(x), x),
\]
and in the first case $g(\alpha)$ is definitely zero, and the second case requires us to consider $\gcd(g, \hat{h})$, and $\hat{h}$ has lower degree than $f$, so this splitting process terminates.

### 3.1.6 Multiple Algebraic Numbers

The situation gets more complicated if we have several such algebraic numbers in play. By hand, such situations tend to be bypassed almost instinctively: if there are $\sqrt{2}$ around, we replace $\sqrt{8}$ by $2\sqrt{2}$, and if $\sqrt{2}$ and $\sqrt{3}$ are around, we replace $\sqrt{6}$ by $\sqrt{2}\sqrt{3}$. For positive radicals, such an approach is good enough if correctly formalised.

**Proposition 21** Let the $N_i$ be positive numbers, and the $M_j$ be a square-free basis for the $N_i$, i.e. the $M_j$ are relatively prime and have no repeated factors, and each $N_i = \prod_j M_j^{n_{i,j}}$. Then the $k$-th roots of the $M_j$ form a multiplicative basis for the $k$-th roots of the $N_i$, and the only relations are $\left(M_j^{1/k}\right)^k = M_j$.

If we allow negative numbers, we have to deal with the catch that $4\sqrt{-4} = 1 + i$, which corresponds to the fact that $x^4 + 4 = (x^2 + 2x + 2)(x^2 - 2x + 2)$, but this is all [Sch00a, Theorem 19].

In terms of the `RootOf` construct, we see that $\sqrt{2}$ is actually $\alpha = \text{RootOf}(x^2 - 2, x)$ and $\sqrt{8}$ is actually $\beta = \text{RootOf}(x^2 - 8, x)$. Now both $x^2 - 2$ and $x^2 - 8$ are irreducible polynomials over the integers. But, in the presence of $\alpha$, $x^2 - 8$ factors as $(x - 2\alpha)(x + 2\alpha)$. Hence in general we are led to the complications of factorisation in the presence of algebraic numbers (Section 6.2).

For non-nested radicals, the situation is fortunately simpler. \[RS13\] TO BE COMPLETED

### 3.1.7 Solutions in Real Radicals

We have seen above, both in the case of the cubic, equation (3.10), and the quartic, equation (3.12), that real roots may need to be expressed via complex radicals, even if all the root are real. Indeed, in the case of the cubic, this is necessary. However, the quartic $x^4 + 4x^3 + x^2 - 6x + 2$, whose roots are
\[
\left\{-1 + \sqrt{3}, -1 - \sqrt{3}, -1 + \sqrt{2}, -1 - \sqrt{2}\right\}
\]
shows that polynomials can have real roots expressible in terms of real radicals, and a slightly less obvious example is given by $x^4 + 4x^3 - 44x^2 - 96x + 552$, whose roots are
\[
\left\{-1 - \sqrt{25 + 2\sqrt{6}}, -1 + \sqrt{25 + 2\sqrt{6}}, -1 - \sqrt{25 - 2\sqrt{6}}, -1 + \sqrt{25 - 2\sqrt{6}}\right\}.
\]
There is a little-known theorem in this area.
Theorem 11 (Isa85) Suppose that all the roots of an irreducible polynomial \( f(x) \) over \( \mathbb{Q} \) are real. Then if any root of the polynomial is expressible in radicals, the degree of the polynomial must be a power of two.

3.1.8 Equations of curves

For a fuller description of this topic, see [Ful69]. In particular, we only consider the affine case, whereas the projective case (i.e. allowing for “points at infinity”) is in many ways more general.

Definition 35 An (affine) algebraic curve \( C(x_1, \ldots, x_n) \) in \( n \) dimensions over a field \( K \) is the set of solutions \( (x_1, \ldots, x_n) \) to \( n-1 \) independent algebraic equations, i.e. polynomials \( g_i(x_1, \ldots, x_n) = 0 \).

If \( n = 2 \) we say that we have a plane algebraic curve.

Of course, the precise curve and equations are often not very interesting: for instance we would like to think that the parabola \( x_1^2 - x_2 \) was “the same” as \( y_1 - y_2^2 \), and so on.

Definition 36 Two curves \( C(x_1, \ldots, x_n) \) and \( C'(y_1, \ldots, y_m) \) are said to be birationally equivalent if there are two families of rational functions

\[
F = (f_1(x_1, \ldots, x_n), \ldots, f_m(x_1, \ldots, x_n))
\]

and

\[
G = (g_1(y_1, \ldots, y_m), \ldots, g_n(y_1, \ldots, y_m))
\]

such that:

1. for almost all \( (x_1, \ldots, x_n) \in C \), \( (f_1(x_1, \ldots, x_n), \ldots, f_m(x_1, \ldots, x_n)) \) is defined and \( \in C' \);
2. for almost all \( (y_1, \ldots, y_m) \in C' \), \( g_1(y_1, \ldots, y_m), \ldots, g_n(y_1, \ldots, y_m) \) is defined and \( \in C \);
3. almost everywhere, \( F \) and \( G \) are mutually inverse, i.e.

\[
f_i(g_1(y_1, \ldots, y_m), \ldots, g_n(y_1, \ldots, y_m)) = y_i
\]

and

\[
g_j(f_1(x_1, \ldots, x_n), \ldots, f_m(x_1, \ldots, x_n)) = x_j.
\]

“Almost everywhere” means “on a non-empty Zariski open set” [Ful69], and can be thought of as “except where we get \( 0/0 \) behaviour”.

Theorem 12 Every algebraic curve is birationally equivalent to a plane curve.

Proof. If there are more than two variables, there is more than one equation, and we can use resultants to eliminate one variable and one equation.

We then have the concept [Sen08] of a curve being soluble by radicals. In this case, the generic curve of degree greater than six is not soluble by radicals [Zar26]. However, many “interesting” curves are soluble by radicals.
CHAPTER 3. POLYNOMIAL EQUATIONS

Proposition 22 [Sen08, Corollary 3.2] Every irreducible plane curve of degree at most five is soluble by radicals.

Proposition 23 [Sen08, Corollary 3.3] Every irreducible singular plane curve of degree at most six is soluble by radicals.

Algorithms to compute these expressions are given in [Har11, SS11].

It is also the case that the offset, i.e. the curve defined as the set of points a fixed distance \( d \) from the original curve, to a curve soluble by radicals is also soluble by radicals.

3.1.9 How many Real Roots?

While a polynomial of degree \( n \) has \( n \) complex roots, it generally has fewer real ones, though how many, on average, is an interesting question, depending on the definition of “on average”. The ‘obvious’ definitions would be ‘normally distributed’ or ‘uniformly distributed in some range’, and for these Kac [Kac43] shows that the average number is \( \frac{2}{\pi} \log n \). A definition with better geometric invariance properties gives \( \sqrt{\frac{n(n+2)}{3}} \)

We have seen that it is not obvious how many real roots a polynomial has: can we answer that question, or more formally the following?

Problem 1 Given a square-free polynomial \( f \), determine how many real roots \( f \) has, and describe each real root sufficiently precisely to distinguish it from the others. Many authors have asked the same question about non-square-free polynomials, and have laboured to produce better theoretical complexity bounds, since the square-free part of a polynomial may have larger coefficients than the original polynomial. However, in practice it is always better to compute the square-free part first.

The usual description (but see Section 3.1.10) is to enclose the root in an interval, within which it is the only root.

Definition 31 introduced the concept of a signed polynomial remainder sequence, also called a Sturm–Habicht sequence: \( f_i \) is proportional by a positive constant to \(-\text{rem}(f_i-2,f_i-1)\). The positive constant is normally chosen to keep the coefficients integral and as small as possible.

Definition 37 If \( f \) is a square-free polynomial, let \( V_f(a) \) denote the number of sign changes in the sequence \( f_0(a), f_1(a), \ldots, f_n(a) \), where \( f_0, \ldots, f_n \) is the Sturm–Habicht sequence of \( f \) and \( f' \), also known as the Sturm sequence of \( f \).

Unpublished. Prof. Sendra has supplied this proof for plane curves.

Let \((R_1, R_2)\) be a square parametrization of the curve and \( K_0 = \mathbb{C}(t) \subset K_1 \subset \ldots \subset K_s \) be a (radical) field tower such that \( R_1, R_2 \in K_s \). Considering the formal derivation with respect to \( t \), one can deduce (for instance by induction on \( s \)) that if \( R \in K_s \) then its derivative \( R' \) is also in \( K_s \).

Now consider \( a = (R_1')^2 + (R_2')^2 \in K_s \) and \( K_{s+1} = K_s(\sqrt{a}) \), then \((O_1, O_2) = (R_1, R_2) \pm d\sqrt{a}/(R_2')', (R_1')' \in (K_{s+1})^2\). So \((O_1, O_2)\) is radical with the tower \( K_0 \subset \ldots \subset K_s \subset K_{s+1} \).
If \( f \) is not square-free, we need more careful definitions \cite{BPR06}, and to be clear whether we are counting with multiplicity or not.

**Theorem 13 (Sturm)** If \( a < b \) are not zeros of \( f \), and \( f \) is square-free, then \( V_f(a) - V_f(b) \) is the number of zeros of \( f \) in \((a, b)\).

\( V_f(\infty) \) (which can be regarded as \( \lim_{a \to \infty} V_f(a) \)) can be computed as the number of sign changes in the sequence of leading coefficients of the Sturm sequence. Similarly, \( V_f(-\infty) \) is the number of sign changes in the sequence of leading coefficients of the Sturm sequence with the signs of the odd-degree terms reversed. Hence \( V_f(-\infty) - V_f(\infty) \), the total number of real roots, is easily computed from the Sturm sequence.

**Theorem 14** If \( f \) is a square-free polynomial of degree \( d \) and coefficients less than \( 2^L \), then the number of subdivisions of \((-\infty, \infty)\) required is \( O(d(L + \log d)) \) \cite{Dav85a}. If \( L > \log d \), then the number of subdivisions required can be \( \Omega(d(L + \log d)) \) \cite{ESY06}. Hence if \( L > \log d \), the number of subdivisions required is \( \Theta(d(L + \log d)) \).

While the obvious way of computing \( V_f(a) \) is by the definition, i.e. evaluating \( f_0(a), \ldots, \), this turns out not to be the most efficient. Rather, while computing the Sturm sequence \( f_0, \ldots \), we should also store the quotients \( q_i \), so that \( f_i(x) = -(f_{i-2}(x) - q_i(x)f_{i-1}(x)) \). We then compute as follows.

**Algorithm 8 (Sturm Sequence evaluation)**

\[ a: \quad \text{A number} \]

**Input:**

- \( f_n(x) \): Last non-zero element of Sturm sequence of \( f \)
- \( q_i(x) \): Quotient sequence from Sturm sequence of \( f \)

**Output:**

Sequence \( L \) of \( f_n(a), f_{n-1}(a), \ldots, f_0(a) \).

\[
\begin{align*}
L[n] & := f_n(a); \\
L[n-1] & := q_{n+1}(a)L[n]; \\
\text{for } i &= n \ldots 2 \\
L[i-2] & := q_i(a)L[i-1] - L[i];
\end{align*}
\]

return \( L \)

If \( f \) has degree \( n \), coefficients of bit-length at most \( \tau \), \( a \) has numerator and denominator of bit-length \( \sigma \), this algorithm has asymptotic complexity \( \tilde{O}(d^2 \max(\sigma, \tau)) \) \cite{LR01}.

Since it is possible to say how big the roots of a polynomial can be (propositions \[\ref{prop:bound_root} \] and \[\ref{prop:bound_root2} \]), we can determine, as precisely as we wish, the location of the real roots of a univariate polynomial: every time the Sturm sequence says that there are more than one root in an interval, we divide the interval in two, and re-compute \( V(a) - V(b) \) for each half.

This is far from the only way of counting and locating real roots, i.e. solving problem \[\ref{prob:real_root} \]; other methods are based on Descartes’ rule of signs (Theorem \[\ref{thm:descartes_rule} \]).

\[\footnote{This rule is always called after Descartes, though the proof actually seems to be due to Gauss \cite{BP93}.} \]
CHAPTER 3. POLYNOMIAL EQUATIONS

the number of roots of \( f \) in \((0, \infty)\) is less than or equal to, by an even number, the number of sign changes in the coefficients of \( f \) \cite{CA76}, its generalisation the Budan–Fourier theorem \cite{Hur12} (Corollaries 33 and 34: the number of roots of \( f \) in \([a,b]\) is less than or equal to, by an even number, the number of sign changes in the derivatives of \( f \) evaluated at \( a \) (i.e. \( f(a), f'(a), f''(a) \ldots \)) less the same evaluated at \( b \)), on continued fractions \cite{TE07}, or on numerical methods \cite{Pan02}.

All such algorithms take time polynomial in \( d \), the degree of the polynomial whose real roots are being counted/determined, i.e. they are algorithms for the dense model. The best results are in \cite{Bur13}, and are \( O(d^4L^2) \) for coefficients \( \leq 2L \).

In the sparse model, we have theorems, but are short on algorithms.

**Proposition 24** A polynomial with \( t \) non-zero terms has at most \( 2t - 1 \) real roots.

Since the number of sign changes in a sequence of \( t \) terms is at most \( t - 1 \), there are at most \( 2t - 1 \) real roots (\( t - 1 \) positive, \( t - 1 \) negative and zero: consider \( x^3 - x \)) for a polynomial with \( t \) non-zero terms, irrespective of \( d \).

**Open Problem 4 (Roots of Sparse Polynomials)** Find algorithms for counting the number of real roots which depend polynomially on \( t \) alone, or \( t \) and \( \log d \). There is recent progress described in \cite{BHPR11}: notably a probabilistic algorithm when \( t = 4 \). See also \cite{Sag14}.

In fact, we can say more than Proposition 24.

**Proposition 25** \textit{(KPT12, Theorem 9)} Let \( g \) be a polynomial of of the form

\[
g(x) = \sum_{i=1}^{k} a_i \prod_{j=1}^{m} f_j^{\alpha_{i,j}}
\]

where \( a_i \in \mathbb{R}, \alpha_{i,j} \in \mathbb{N} \) and the \( f_j \) have at most \( t \) non-zero terms. Then the number of real roots of \( g \) (not counted with multiplicity) is at most \( 2k^2t^2m^2/2 + 2kmt = 2O(k^2m \log t) \), and in particular is polynomial in \( t \).

If all the \( \alpha_{i,j} \in \{0, 1\} \) the result would be trivial, as \( g \) would have at most \( kt^m \) non-zero terms. If \( k = 1 \) the result is also trivial. The point is that we are allowing a slightly wider range of polynomials.

**Problem 2** Having solved Problem 1 we may actually wish to know the roots to a given accuracy, say \( L \) bits after the point. Again, many authors have asked the same question about non-square-free polynomials, and have laboured to produce better theoretical complexity bounds, since the square-free part of a polynomial may have larger coefficients than the original polynomial. However, in practice it is always better to compute the square-free part first.

The best published solution to this problem currently is that in \cite{KS11}. TO BE COMPLETED

\textsuperscript{6}We assume neither \( a \) nor \( b \) are roots.
3.1.10 Thom's Lemma

An alternative approach to the description of real roots is provided by Thom's Lemma, and elaborated in [CR88].

Notation 19 A sign condition $\epsilon_i$ is any of the symbols "$> 0$", "$< 0$", "$\leq 0$", or "$\geq 0$". A generalised sign condition is any of these or "$\geq 0$", "$\leq 0$". If $\epsilon = (\epsilon_0, \ldots, \epsilon_{n-1})$ is any $n$-tuple of generalised sign conditions, we denote by $\epsilon$ the relaxation of $\epsilon$ obtained by replacing $< 0$ by $\leq 0$ and $> 0$ by $\geq 0$.

Lemma 4 (Thom's Lemma [CR88 Proposition 1.2]) Let $p$ be a polynomial of degree $n$, and $\epsilon$ an $n$-tuple of sign conditions. Let

$$A(\epsilon) = \{ x \in \mathbb{R}| \forall i p^{(i)} \epsilon_i \},$$

where $p^{(i)}$ denotes the $i$-th formal derivative (Definition 32) of $p$. Then:

(a) $A(\epsilon)$ is either empty or connected, i.e. an interval;

(b) If $A(\epsilon)$ is an interval, then $A(\bar{\epsilon})$ is the closure of $A(\epsilon)$, obtained by replacing $< r$ by $\leq r$ and $> l$ by $\geq l$.

The proof is by induction on $\deg(p)$. Item (b) may seem obvious, but is in fact rather subtle. Consider $p = x^3 - x^2$, depicted in Figure 3.3. While $\{x|p(x) > 0\}$ is the interval $(1, \infty)$, $\{x|p(x) \geq 0\}$ is not the interval $[1, \infty)$, but rather $\{0\} \cup [1, \infty)$. Thom’s Lemma actually talks about $A(\epsilon)$ with all the signs fixed, and the sign of $p''$ distinguishes $\{0\}$ from $[1, \infty)$.

Corollary 3 A root $x_0$ of $p$, hence a point where $p^{(0)}(x_0) = 0$, is uniquely determined by the signs of all the derivatives of $p$ there.

It is possible to compute with real algebraic numbers defined this way, and answer questions such as “how many roots does $q(x, y)$ have, when $y$ is the (unique) root of $p(y) = 0$ with the following sign conditions: $p'(y) < 0, p''(y) > 0$ ots?”: again for details see [CR88].

3.2 Linear Equations in Several Variables

We now consider the case of several polynomial equations in several (not necessarily the same number) of variables.

Notation 20 The variables will be called $x_1, \ldots, x_n$, though in specific examples we may use $x, y$ or $x, y, z$ etc.
3.2.1 Linear Equations and Matrices

A typical set of 3-by-3 linear equations might look like the following.

\[ \begin{align*}
2x + 3y - 4z &= a; \\
3x - 2y + 2z &= b; \\
4x - 3y - 2z &= c.
\end{align*} \]

If we denote by \( M \) the matrix
\[
\begin{pmatrix}
2 & 3 & -4 \\
3 & -2 & 2 \\
4 & -3 & -1
\end{pmatrix},
\]
and \( \mathbf{x} \) the (column) vector \((x, y, z)\) and \( \mathbf{a} \) the (column) vector \((a, b, c)\), then this becomes the single matrix equation
\[ M \mathbf{x} = \mathbf{a}, \tag{3.13} \]
which has, assuming \( M \) is invertible, the well-known solution
\[ \mathbf{x} = M^{-1} \mathbf{a}. \tag{3.14} \]

This poses two questions: how do we store matrices, and how do we compute inverses?

3.2.2 Representations of Matrices

The first question that comes to mind here is “dense or sparse?”, as in definition 23 (page 35). For a dense representation of matrices, the solution is obvious:
we store a two-dimensional array (or one-dimensional array of one-dimensional arrays if our language does not support two-dimensional arrays) containing the values $m_{i,j}$ of the elements of the matrix. The algorithms for adding and multiplying dense matrices are pretty obvious, though in fact it is possible to multiply two $2 \times 2$ matrices with seven multiplications of entries rather than the obvious eight: this leads to being able to multiply two $n \times n$ matrices with $O(n^{\log_2 7} \approx 2.807)$ element multiplications rather than $O(n^3)$: see Excursus B.4.

For a sparse representation of matrices, we have more choices.

**row-sparse** Here $M$ is stored as a one-dimensional array of rows: the $i$-th row consisting of a list of pairs $(j, m_{i,j})$. This representation is equivalent to that of the sparse polynomial $\sum_j m_{i,j} x^j$, and this technique has been used in practice and has a useful analogue in the case of non-linear equations (see section 3.3.5).

**column-sparse** Here $M$ is stored as a one-dimensional array of columns: the $j$-th column consisting of a list of pairs $(i, m_{i,j})$.

**totally sparse** Here $M$ is stored as a list of triples $(i, j, m_{i,j})$.

**structured** There are a large variety of special structures of matrices familiar to numerical analysts, such as ‘banded’, ‘Toeplitz’ etc. Each of these can be stored efficiently according to their special form. For example, circulant (a special form of Toeplitz) matrices, of the form

\[
\begin{pmatrix}
  a_1 & a_2 & a_3 & \ldots & a_{n-1} & a_n \\
  a_n & a_1 & a_2 & \ldots & a_{n-1} & a_n \\
  \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\
  a_2 & a_3 & \ldots & a_{n-1} & a_n & a_1
\end{pmatrix},
\]

are, strictly speaking, dense, but only have $n$ distinct entries, so are “information-sparse”. Recognizing these structures is not just a matter of storage economy: there are also faster algorithms, e.g. [KMS13] can solve an $n \times n$ Toeplitz (3.13) in time $O(n \log^2 n)$, rather than the ‘obvious’ $O(n^3)$ or better $O(n^2 \ldots)$ (see Section B.4).

Clearly, if the matrix is structured, we should use the corresponding representation. For randomly sparse matrices, the choice depends on what we are doing with the matrix: if it is row operations, then row-sparse is best, and so on. One big issue with sparse matrices is known as fill-in — the tendency for operations on sparse matrices to yield less sparse matrices. For example, if we multiply two $n \times n$ matrices, each with $e$ non-zero elements per row, with $n \gg e$, we would expect, assuming the non-zero elements are scattered at random, the resulting matrix to have $e^2$ non-zero elements per row. This has some apparently paradoxical consequences. Suppose $M$ and $N$ are two such matrices, and we wish to compute $MNv$ for many vectors $v$. Clearly, we compute $MN$ once and for all, and multiply this by $v$, and for dense $M$ and $N$, this is right if there are more
than \( n \) such vectors \( v \). But, if \( n \gg e > 2 \), this is not optimal, since, once \( MN \) is computed, computing \((MN)v\) requires \( ne^2 \) operations, while computing \( Nv \) requires \( ne \), as does computing \( M(Nv) \), totalling \( 2ne < ne^2 \).

3.2.3 Matrix Inverses: not a good idea!

The first response to the question “how do we compute matrix inverses” ought to be “are you sure you want to?” Solving (as opposed to thinking about the solution of) equation (3.13) via equation (3.14) is often not the best way to proceed. Gaussian elimination (possibly using some of the techniques described later in this section) directly on equation (3.13) is generally the best way. This is particularly true if \( M \) is sparse, since \( M^{-1} \) is generally not sparse — an extreme example of fill-in. Indeed, special techniques are generally used for the solution of large sparse systems, particularly those arising in integer factorisation or other cryptographic applications [HD03].

The usual method of solving linear equations, or computing the inverse of a matrix, is via Gaussian elimination, i.e. transforming equation (3.13) into one in which \( M \) is upper triangular, and then back-substituting. This transformation is done by row operations, which amount to adding/subtracting multiples of one row from another, since

\[
P = Q \quad \text{&} \quad R = S \quad \text{implies} \quad P + \lambda R = Q + \lambda S. \tag{3.15}
\]

If we try this on the above example, we deduce successively that 

\[
z = a - 18b + 13c, \\
y = a - 14b + 10c \quad \text{and} \quad x = a - 15b + 11c.
\]

Emboldened by this we might try a larger matrix:

\[
M = \begin{pmatrix}
a & b & c & d \\
e & f & g & h \\
i & j & k & l \\
m & n & o & p
\end{pmatrix}. \tag{3.16}
\]

After clearing out the first column, we get the matrix

\[
\begin{pmatrix}
a & b & c & d \\
0 & -\frac{eb}{a} + f & -\frac{ec}{a} + g & -\frac{ed}{a} + h \\
0 & -\frac{ib}{a} + j & -\frac{ic}{a} + k & -\frac{id}{a} + l \\
0 & -\frac{mb}{a} + n & -\frac{mc}{a} + o & -\frac{md}{a} + p
\end{pmatrix}.
\]

Clearing the second column gives us

\[
\begin{pmatrix}
a & b & c & d \\
0 & -\frac{eb}{a} + f & -\frac{ec}{a} + g & -\frac{ed}{a} + h \\
0 & -\frac{ib}{a} + j & -\frac{ic}{a} + k & -\frac{id}{a} + l \\
0 & -\frac{mb}{a} + n & -\frac{mc}{a} + o & -\frac{md}{a} + p
\end{pmatrix},
\]

\[
\begin{pmatrix}
a & b & c & d \\
0 & -\left(\frac{-ib}{a} + j\right)\left(\frac{-ic}{a} + k\right) & -\frac{ic}{a} + k & -\frac{-ib}{a} + j\left(\frac{-id}{a} + l\right) - \frac{id}{a} + l \\
0 & -\left(\frac{-mb}{a} + n\right)\left(\frac{-mc}{a} + o\right) & -\frac{mc}{a} + o & -\frac{-mb}{a} + n\left(\frac{-md}{a} + p\right) - \frac{md}{a} + p
\end{pmatrix}.
\]
which we can “simplify” to
\[
\begin{pmatrix}
 a & b & c & d \\
 0 & -eb+af & -ec+ag & -ed+ah \\
 0 & 0 & -eb+af & -ec+ag \\
 0 & 0 & -eb+af & -ec+ag \\
\end{pmatrix}
\]
\begin{equation}
(3.17)
\end{equation}

After clearing the third column, the last element of the matrix is
\[
\begin{pmatrix}
 a & b & c & d \\
 0 & -eb+af & -ec+ag & -ed+ah \\
 0 & aj - ib & ak - ic & al - id \\
 0 & -mb + an & ao - mc & ap - md \\
\end{pmatrix}
\]
\begin{equation}
M_2 :=
\end{equation}
\begin{equation}
(3.19)
\end{equation}

After clearing column two, we get
\[
M_3 :=
\begin{pmatrix}
 a & b & c & d \\
 0 & -eb + af & -ec + ag & -ed + ah \\
 0 & 0 & -eb + af & -ec + ag \\
 0 & 0 & -eb + af & -ec + ag \\
\end{pmatrix}
\]
\begin{equation}
(3.20)
\end{equation}
The result of the next step is better contemplated than printed!

However, if we do contemplate the result printed above, we see that rows 3 and 4 contain polynomials of degree four, whereas in the “simplified” form (3.17) we only have polynomials of degree three in the numerators. Indeed, if we were to expand the matrix above, we would observe that rows three and four each had a common factor of $a$. Similarly, if we were to (or were to get a computer algebra system to) expand and then factor the last step, we would get $a^2(af - eb)[M]$, as in equation (3.18). Such common factors are not a coincidence (indeed, they cannot be, since $M$ is the most general $4 \times 4$ matrix possible).

**Theorem 15 (Dodgson–Bareiss [Bar68, Dod66])** Consider a matrix with entries $m_{i,j}$. Let $m_{i,j}^{(k)}$ be the determinant

$$
\begin{vmatrix}
 m_{1,1} & m_{1,2} & \cdots & m_{1,k} & m_{1,j} \\
 m_{2,1} & m_{2,2} & \cdots & m_{2,k} & m_{2,j} \\
 \vdots & \vdots & \ddots & \vdots & \vdots \\
 m_{k,1} & m_{k,2} & \cdots & m_{k,k} & m_{k,j} \\
 m_{i,1} & m_{i,2} & \cdots & m_{i,k} & m_{i,j}
\end{vmatrix},
$$

i.e. that of rows $1 \ldots k$ and $i$, with columns $1 \ldots k$ and $j$. In particular, the determinant of the matrix of size $n$ whose elements are $(m_{i,j})$ is $m_{n,n}^{(n-1)}$ and $m_{i,j}^{(0)} = m_{i,j}^{(0)}$. Then (assuming $m_{0,0}^{(-1)} = 1$):

$$
m_{i,j}^{(k)} = \frac{1}{m_{k-1,k-1}^{(k-2)}} \begin{vmatrix}
 m_{k,k}^{(k-1)} & m_{k,j}^{(k-1)} \\
 m_{i,k}^{(k-1)} & m_{i,j}^{(k-1)}
\end{vmatrix}.
$$

**Proof.** By fairly tedious induction on $k$.

How does this relate to what we have just seen? If we do fraction-free elimination on the matrix $M$ of (3.16), we get (3.19), which we can rewrite as

$$
M_2 = \begin{pmatrix}
 a & b & c & d \\
 0 & a & b & a & c & a & d \\
 0 & e & f & e & g & e & h \\
 0 & a & b & a & c & a & d \\
 0 & i & j & i & k & i & l \\
 0 & m & n & m & o & m & p
\end{pmatrix},
$$

or, in the terminology of Theorem [15]

$$
M_2 = \begin{pmatrix}
 m_{1,1}^{(0)} & m_{1,2}^{(0)} & m_{1,3}^{(0)} & m_{1,4}^{(0)} \\
 0 & m_{2,2}^{(1)} & m_{2,3}^{(1)} & m_{2,4}^{(1)} \\
 0 & m_{3,2}^{(1)} & m_{3,3}^{(1)} & m_{3,4}^{(1)} \\
 0 & m_{4,2}^{(1)} & m_{4,3}^{(1)} & m_{4,4}^{(1)}
\end{pmatrix},
$$

7The Oxford logician Charles Dodgson was better known as Lewis Carroll. Much of this seems to have been known earlier [Chi53].
The next elimination step is

\[
M_3 = \begin{pmatrix}
    m_{1,1}^{(0)} & m_{1,2}^{(0)} & m_{1,3}^{(0)} & m_{1,4}^{(0)} \\
    0 & m_{2,2}^{(1)} & m_{2,3}^{(1)} & m_{2,4}^{(1)} \\
    0 & 0 & m_{3,2}^{(1)} & m_{3,3}^{(1)} \\
    0 & 0 & m_{4,2}^{(1)} & m_{4,3}^{(1)}
\end{pmatrix},
\]

and Theorem 15 guarantees that \( m_{1,1}^{(0)} \) (i.e. \( a \)) divides the determinants in rows 3 and 4, so that we have

\[
\begin{pmatrix}
    1 & 0 & 0 & 0 \\
    0 & 1 & 0 & 0 \\
    0 & 0 & 1/a & 0 \\
    0 & 0 & 0 & 1/a
\end{pmatrix} M_3 = \begin{pmatrix}
    m_{1,1}^{(0)} & m_{1,2}^{(0)} & m_{1,3}^{(0)} & m_{1,4}^{(0)} \\
    0 & m_{2,2}^{(1)} & m_{2,3}^{(1)} & m_{2,4}^{(1)} \\
    0 & 0 & m_{3,3}^{(2)} & m_{3,4}^{(2)} \\
    0 & 0 & m_{4,4}^{(2)}
\end{pmatrix}.
\]

This is a general result.

**Corollary 4 (Bareiss’ algorithm)** When doing fraction-free Gaussian elimination, after clearing column \( k \), every element of rows \( k + 1 \ldots n \) is divisible by \( m_{k-2,k-1} \).

This is actually the ‘one-step’ variant of Bareiss [Bar68]: there are other variants with more advanced look-ahead, but they do not (and can not) cancel any more in general. This result accounts for the factor of \( a \) observed in rows 3 and 4 above, and for the factors of \( a^2 \) and \( af - eb \) in the last step. Cancelling the \( a \) in rows 3 and 4 would in fact automatically prevent the \( a^2 \) from even being generated — far better than generating it and then cancelling it!

We normally apply Gaussian elimination (fraction-free or not) to solve systems such as (3.13), by transforming them to

\[
U \cdot x = a',
\]

where \( U' \) is upper-triangular: performing the same operations on the rows of \( a \) to get \( a' \) as we do on \( M \) to get \( U \). This can be viewed as elimination on the augmented matrix \( \overline{M} \) — \( M \) with \( a \) as an extra column.

**Corollary 5** When doing fraction-free Gaussian elimination in the augmented matrix \( \overline{M} \), after clearing column \( k \), every element of rows \( k + 1 \ldots n \), including the “right-hand side”, is divisible by \( m_{k-1,k-1}^{(k-2)} \).

The Bareiss–Dodgson calculation pre-supposes that we are indeed doing Gaussian elimination precisely, and building all the sub-determinants specified. It is tempting to assume that we can take “short cuts”, and skip zero elements,
as after all “they are already zero, and so don’t need to be cleared”. This is a mistake, as we may miss factors we should cancel. Unfortunately, demonstrating this is not trivial, so there’s a Maple worksheet demonstrating this at http://staff.bath.ac.uk/masjhd/JHD-CA/BDwarning.html.

Corollary 6 If the initial entries are integers of length \( l \) (resp. polynomials of degree \( l \)), then after \( k \) steps, the entries will have length (resp. degree) \( O(kl) \).

This is to be contrasted with the \( O(2^k l) \) of the naïve fraction-free approach. We note that a similar approach can be applied to the “LU decomposition” of matrices [Jef10].

It is possible to view Euclid’s algorithm for polynomials as Gaussian elimination in a matrix (Sylvester’s matrix — definition [99]) of coefficients, and the factors \( \beta_i \) that are cancelled by the sub-resultant variant for normal polynomial remainder sequences (footnote 34 on page 62) are those predicted by Corollary 4 above.

3.2.4 Complexity

Let the elements of an \( n \times n \) matrix have size \( d \) (degree for polynomials, or bit-length for integers, ignoring\(^8\) the fact that \( a + b \) may have greater bit-length than either \( a \) or \( n \)), and let the cost of multiplying two elements of size \( d \), or dividing an element of size \( 2d \) by an element of size \( \leq d \), be \( M(d) \). We will ignore the cost of addition/subtraction. Then the cost of clearing columns is as follows.

**column 1** For \( n - 1 \) rows, we do \( n - 1 \) calculations of new elements, which are \( 2 \times 2 \) determinants: cost \( 2M(d) \) each, making a total of \( 2(n - 1)2M(d) \).

**column 2** For \( n - 2 \) rows, we do \( n - 2 \) calculations of new elements, which are \( 2 \times 2 \) determinants followed by a cancellation: cost \( 3M(2d) \) each, making a total of \( 3(n - 2)^2M(2d) \).

**column \( k \)** For \( n - k \) rows, we do \( n - k \) calculations of new elements, which are \( 2 \times 2 \) determinants followed by a cancellation: cost \( 3M(kd) \) each, making a total of \( 3(n - k)^2M(kd) \).

If we call the first term \( 3(n - 1)^2M(d) \) rather than \( 2(n - 1)^2M(d) \) (i.e. ignore the fact that there is no cancellation for the first column) we get

\[
3 \sum_{k=1}^{n-1} (n - k)^2M(kd). \tag{3.23}
\]

The value of this depends on \( M \) (we’ll ignore constant factors, so there is an implied \( O(\cdots) \), or \( \tilde{O}(\cdots) \), round everything in this analysis).

\(^8\)If we wanted to take it into account, we would use the Hadamard bounds (Propositions 91 and 92) to bound the size of the intermediate objects, which are \( k \times k \) determinants by Theorem 45 and get an extra factor of \( \log d \) in the length.
3.2. LINEAR EQUATIONS IN SEVERAL VARIABLES

\[ M(kd) = (kd)^2 \] (classical arithmetic on integers, or univariate polynomials with fixed-length coefficients) we get

\[
\frac{1}{30} d^2 n^5 - \frac{1}{30} d^2 n = O(d^2 n^5).
\] (3.24)

Allowing for coefficient growth in the integer case, we would have

\[ O(d^2 n^5 \log^2 n). \] (3.25)

\[ M(kd) = kd \] (the dominant effect in FFT-based arithmetic on integers, or univariate polynomials with fixed-length coefficients) we get

\[
\frac{1}{12} d n^4 - \frac{1}{12} n^2 d = \tilde{O}(dn^4).
\]

Suppose we have an \( n \times n \) matrix, and the elements are polynomials of degree \( d \) in \( s \) variables. We have two options for computing the determinant:

1. Fraction-free Gaussian elimination, as described above;

2. Expansion by minors, where we compute all \( \frac{n(n-1)}{2} \)

\[
\begin{vmatrix}
  a_{1,i} & a_{1,j} \\
  a_{2,i} & a_{2,j} \\
  a_{3,i} & a_{3,j}
\end{vmatrix}
\]

\[
\begin{vmatrix}
  a_{1,i} & a_{1,j} & a_{1,k} \\
  a_{2,i} & a_{2,j} & a_{2,k} \\
  a_{3,i} & a_{3,j} & a_{3,k}
\end{vmatrix}
\]

and so on.

The first involves \( O(n^3) \) operations on polynomials, and the second \( O(n2^n) \) such operations. The comparison seems clear-cut, we should use the Bareiss-Dodgson Fraction-free method (Corollary 4). Furthermore the intermediate results of Corollary 4 are, apart from the pre-cancellation

\[
\begin{vmatrix}
  m_{3,2}^{(1)} & m_{3,3}^{(1)} \\
  m_{4,2}^{(1)} & m_{4,3}^{(1)}
\end{vmatrix}
\]

etc., sub-minors of the same size as would be computed in the minors method: see (3.21) for an example.

The experimental evidence [Smi76, Smi79] is rather different, and much depends in practice on the sparsity of the matrix. This is analysed in [GJ76], whose conclusions also depend on the sparsity of the polynomials.

**Open Problem 5** The computations in [Smi76, Smi79] should be repeated on modern systems/computers, and the scaling should be re-examined now that larger systems are feasible.

### 3.2.5 Over/under-determined Systems

So far we have implicitly assumed that there are as many equations as there are unknowns, and that the equations determine the unknowns precisely (in other words, that the determinant of the corresponding matrix is non-zero). What happens if these assumptions do not hold? There are several cases to be distinguished.
Over-determined and consistent Here the ‘extra’ equations are consistent with those that determine the solution. A trivial example in one variable would be the pair \(2x = 4, 3x = 6\).

Over-determined and inconsistent Here the ‘extra’ equations are not consistent with those that determine the solution. A trivial example in one variable would be the pair \(2x = 4, 3x = 9\), where the first implies that \(x = 2\), but the second that \(x = 3\).

Spuriously over-determined This is a generalisation of “over-determined and consistent” when, after deleting the ‘extra’ equations that convey no new information, we are left with an under-determined system.

Under-determined and consistent Here there are not enough equations (possibly after deleting spurious ones) to determine all the variables. An example would be \(x + y = 3\). Here \(x\) can be anything, but, once \(x\) is chosen, \(y\) is fixed as \(3 - x\). Equally, we could say that \(y\) can be anything, but, once \(y\) is chosen, \(x\) is fixed as \(3 - y\). The solutions form a \(k\)-dimensional hyper-plane, where \(k\) is the number of variables minus the number of (non-spurious) equations.

Under-determined yet inconsistent Here the equations (possibly after deleting spurious ones) are still inconsistent. One example would be \(x + 2y + 3z = 1, 2x + 4y + 6z = 3\).

We are then left with three possibilities for the solutions, which can be categorised in terms of the dimension (‘dim’).

\[\dim = -1\] This is the conventional ‘dimension’ assigned when there are no solutions, i.e. the equations are inconsistent.

\[\dim = 0\] Precisely one solution.

\[\dim > 0\] An infinite number of solutions, forming a hyperplane of dimension \(\dim\).

3.3 Nonlinear Multivariate Equations: Distributed

Most of the section has its origin in the pioneering work of Buchberger \[Buc70\]. Some good modern texts are \[AL94, BW93, CLO06\].

If the equations are nonlinear, equation (3.15) is still available to us. So, given the three equations

\[x^2 - y = 0 \quad x^2 - z = 0 \quad y + z = 0,\]

we can subtract the first from the second to get \(y - z = 0\), hence \(y = 0\) and \(z = 0\), and we are left with \(x^2 = 0\), so \(x = 0\), albeit with multiplicity 2 (definition 34). However, we can do more than this. Given the two equations

\[x^2 - 1 = 0 \quad xy - 1 = 0, \quad (3.26)\]
3.3. NONLINEAR MULTIVARIATE EQUATIONS: DISTRIBUTED

there might seem to be no row operation available. But in fact we can subtract
x times the second equation from y times the first, to get x − y = 0. Hence the
solutions are x = ±1, y = x.

We can generalise equation (3.15) to read as follows: for all polynomials f
and g,

\[ P = Q \quad \& \quad R = S \quad \text{implies} \quad fP + gR = fQ + gS. \quad (3.27) \]

Lemma 5 In equation (3.27), it suffices to consider terms (monomials with
leading coefficients) for f and g rather than general polynomials.

Proof. Let f be \( \sum a_i m_i \) and g be \( \sum b_i m_i \), where the \( m_i \) are monomials and
the \( a_i \) and \( b_i \) coefficients (possibly zero, but for a given \( i \), both \( a_i \) and \( b_i \) should
not be zero, since then \( m_i \) would be redundant). Then for each \( i \), the monomial
version of equation (3.27) gives

\[ P = Q \quad \& \quad R = S \quad \text{implies} \quad a_i m_i P + b_i m_i R = a_i m_i Q + b_i m_i S. \]

Then we can use equation (3.15) repeatedly, with \( \lambda = 1 \), to add these together
to get the general form of equation (3.27).

Because of equation (3.2), we can regard equations as synonymous with
polynomials. Equation (3.27) then motivates the following definition.

Definition 38 Let \( S \) be a set of polynomials in the variables \( x_1, \ldots, x_n \), with
coefficients from \( R \). The ideal generated by \( S \), denoted \( (S) \), is the set of all
finite sums \( \sum f_i s_i : s_i \in S, f_i \in R[x_1, \ldots, x_n] \). If \( S \) generates \( I \), we say that \( S \)
is a basis for \( I \).

Proposition 26 This is indeed an ideal in the sense of definition 7.

Strictly speaking, what we have defined here is the left ideal: there are also
concepts of right ideal and two-sided ideal, but all concepts agree in the case of
commutative polynomials, which we will assume until section 3.3.14.

Proposition 27 \( ((S)) = (S) \).

Definition 39 Two sets of polynomial equations are equivalent if the polyno-
mials defining the left-hand sides generate the same ideal. We will see how to
test this in corollary 7.

Just as an upper triangular matrix is a nice formulation of a set of linear equa-
tions, allowing us to “read off”, the solutions, so we would like a similarly ‘nice’
basis for an ideal generated by non-linear equations. In order to do this, we
will regard our polynomials in a distributed format, with the terms sorted in
some admissible (page 41) ordering \( > \). Note that we are not requiring that
the polynomials are stored this way in an algebra system, though in fact most
algebra systems specialising in this area will do so: we are merely discussing
the mathematics of such polynomials. Having fixed such an ordering \( > \), we can
define the following concepts.
Definition 40 If \( f \) is a non-zero polynomial, the leading term of \( f \), denoted \( \text{lt}(f) \), is that term greatest with respect to \( > \). The corresponding monomial is called the leading monomial of \( f \), \( \text{lm}(f) \). We will sometimes apply \( \text{lm} \) to sets, where \( \text{lm}(S) = \{ \text{lm}(s) | s \in S \} \).

“Monomial algebra” is a particularly simple form of polynomial algebra: in particular

\[
\gcd \left( \prod_{i=1}^{n} x_i^{a_i}, \prod_{i=1}^{n} x_i^{b_i} \right) = \prod_{i=1}^{n} x_i^{\min(a_i, b_i)},
\]

\[
\text{lcm} \left( \prod_{i=1}^{n} x_i^{a_i}, \prod_{i=1}^{n} x_i^{b_i} \right) = \prod_{i=1}^{n} x_i^{\max(a_i, b_i)}.
\]

Definition 41 If \( \text{lm}(g) \) divides \( \text{lm}(f) \), then we say that \( g \) reduces \( f \) to \( h = \text{lcm}(f) - (\text{lt}(f)/\text{lm}(g))g \), written \( f \rightarrow^g h \). Otherwise we say that \( f \) is reduced with respect to \( g \). The Maple user should note that Maple’s Reduce command actually implements complete reduction — see Definition 42.

If \( R \) is a field, division is possible, and so it is more usual to reduce \( f \) to \( f - (\text{lt}(f)/\text{lm}(g))g \). In the construction of \( h \), the leading terms of both \( \text{lcm}(f) \) and \( (\text{lt}(f)/\text{lm}(g))g \) are \( \text{lcm}(f)\text{lm}(f) \), and so cancel. Hence \( \text{lm}(h) \leq \text{lm}(f) \).

This observation and theorem 4 give us the following result.

Proposition 28 Any chain \( f_1 \rightarrow^g f_2 \rightarrow^g f_3 \cdots \) is finite, i.e. terminates in a polynomial \( h \) reduced with respect to \( g \). We write \( f_1 \rightarrow^g \star h \).

These concepts and results extend to reduction by a set \( G \) of polynomials, where \( f \rightarrow^G h \) means \( \exists g \in G : f \rightarrow^g h \). We must note that a polynomial can have several reductions with respect to \( G \) (one for each element of \( G \) whose leading monomial divides the leading monomial of \( f \)). For example, let \( G = \{ g_1 = x - 1, g_2 = y - 2 \} \) and \( f = xy \). Then there are two possible reductions of \( f \): \( f \rightarrow^{g_1} h_1 = f - yg_1 = y, \) and \( f \rightarrow^{g_2} h_2 = f - xg_2 = 2x \). In this case \( h_1 \rightarrow^{g_1} 2 \) and \( h_2 \rightarrow^{g_2} 2 \), so that \( f \rightarrow^G 2 \) uniquely, but even this need not always be the case. If we let \( G = \{ g_1 = x - 1, g_2 = x^2 \} \) and \( f = x^2 - 1 \), then \( f \rightarrow^{g_2} h_2 = f - g_2 = -1 \), whereas \( f \rightarrow^{g_1} f - xg_1 = x - 1 \rightarrow^{g_1} 0 \); so \( f \rightarrow^G 0 \) or \(-1 \).

This definition deals with reduction of the leading monomial of \( f \) by \( g \), but it might be that other monomials are reducible. For simplicity we consider the case when \( R \) is a field.

Definition 42 If any term \( cm \) of \( f \) is reducible by \( g \), i.e. the leading monomial of \( g \) divides \( m \), we say that \( g \) part-reduces \( f \), and write \( f \Rightarrow^g f - (cm/\text{lt}(g))g \).

We can continue this process (only finitely often, by repeated application of theorem 4), until no monomial of \( f \) is reducible by \( g \), when we write \( f \Rightarrow^g s \), and
say that $f$ is completely reduced by $g$ to $h$. Again, this extends to reduction by a set of polynomials.

Reduction is conceptually fairly easy, but can be expensive if implemented na"ively. Yan [Yan98] observed this, and invented the “geobucket” data structure (see (2.2)), which ensured that we were not repeatedly subtracting polynomials with few terms (as tends to be the case in $G$) from polynomials with many terms (as $f$ often is). In particular, he observed a factor of over 32 in one large computation: 43 hours instead of 8 weeks!

In section 3.2.1 we performed row operations: subtracting a multiple of one row from another, which is essentially what reduction does, except that the ‘multiple’ can include a monomial factor. It turns out that we require a more general concept, given in the next definition.

**Definition 43** Let $f, g \in R[x_1, \ldots, x_n]$. The $S$-polynomial of $f$ and $g$, written $S(f, g)$ is defined as

$$S(f, g) = \frac{\text{lt}(g)}{\gcd(\text{lm}(f), \text{lm}(g))} f - \frac{\text{lt}(f)}{\gcd(\text{lm}(f), \text{lm}(g))} g. \quad (3.28)$$

We note that the divisions concerned are exact, and that this generalises reduction in the sense that, if $\text{lm}(g)$ divides $\text{lm}(f)$, then $f \rightarrow S(f, g)$. As with reduction, the leading monomials in the two components on the righthand side of equation (3.28) cancel. Another way of thinking of the $S$-polynomial (when $R$ is a field) is that it is the difference between what you get by reducing $\text{lcm}(\text{lm}(f), \text{lm}(g))$ by $f$ and by $g$.

**Proposition 29** $S(f, g) = -S(g, f)$.

**Proposition 30** $S(f, g) \in (\{f, g\})$.

### 3.3.1 Gröbner Bases

From now until section 3.3.13 we will assume that $R$ is a field. However, we will continue to use $R$, and not gratuitously make polynomials monic, since this can be expensive.

**Theorem 16** [BW93, Proposition 5.38, Theorem 5.48] The following conditions on a set $G \subseteq R[x_1, \ldots, x_n]$, with a fixed ordering $>$ on monomials, are equivalent.

1. $\forall f, g \in G, S(f, g) \rightarrow G 0$.

2. If $f \rightarrow G g_1$ and $f \rightarrow G g_2$, then $g_1$ and $g_2$ differ at most by a multiple in $R$, i.e. $\rightarrow G$ is essentially well-defined.

3. $\forall f \in (G), f \rightarrow G 0$.

4. $(\text{lm}(G)) = (\text{lm}(G))$, i.e. the leading monomials of $G$ generate the same ideal as the leading monomials of the whole of $(G)$.
If $G$ satisfies these conditions, $G$ is called a Gröbner base (or standard basis).

These are very different kinds of conditions, and the strength of Gröbner theory lies in their interplay. Condition 2 underpins the others: \( \rightarrow^G \) is well-defined. Condition 1 looks technical, but has the great advantage that, for finite $G$, it is finitely checkable: if $G$ has $k$ elements, we take the $k(k-1)/2$ unordered (by proposition 29) pairs from $G$, compute the $S$-polynomials, and check that they reduce to zero. This gives us either a proof or an explicit counter-example (which is the key to algorithm 9). Since $f \rightarrow^G 0$ means that $f \in (G)$, condition 3 means that ideal membership is testable if we have a Gröbner base for the ideal. Condition 4 can be seen as a generalisation of “upper triangular” — see section 3.3.5.

Now let $G$ and $H$ be Gröbner bases, possibly with respect to different orderings.

**Proposition 31** If $\forall g \in G, g \rightarrow^H 0$, then $(G) \subseteq (H)$.

**Proof.** Let $f \in (G)$. Then $f \rightarrow^G 0$, so $f = \sum c_i g_i$. But $g_i \rightarrow^H 0$, so $g_i = \sum_j d_{ij} h_j$. Therefore $f = \sum_j (\sum_i c_i d_{ij}) h_j$, and so $f \in (H)$.

**Corollary 7** If $\forall g \in G, g \rightarrow^H 0$, and $\forall h \in H, h \rightarrow^G 0$, then $(G) = (H)$.

Over a field, a particularly useful Gröbner base is a completely reduced Gröbner base (abbreviated crGb) $G$, i.e. one where every element is completely reduced with respect to all the others: in symbols

$$\forall g \in G \quad g \rightarrow^{G \setminus \{g\}} g.$$

For a consistent set of linear polynomials, the crGb would be a set of linear polynomials in one variable each, e.g. \{\(x-1, y-2, z-3\}\}, effectively the solution. In general, a monic crGb is a canonical (definition 3) form for an ideal: two ideals are equal if, and only if, they have the same crGb (with respect to the same ordering, of course).

**Theorem 17 (Buchberger)** Every polynomial ideal has a Gröbner base: we will show this constructively for finitely-generated ideals over noetherian (definition 8) rings.

**Algorithm 9 (Buchberger)**

**Input:** finite $G_0 \subset R[x_1, \ldots, x_n]$; monomial ordering $>$.  
**Output:** $G$ a Gröbner base for $(G_0)$ with respect to $>$.  

---

9In fact, every polynomial ideal over a noetherian ring is finitely generated. However, it is possible to encode undecidability results in infinite descriptions of ideals, hence we say “finitely generated” to avoid this paradox.
3.3. NONLINEAR MULTIVARIATE EQUATIONS: DISTRIBUTED

\[ G := G_0; \ n := |G|; \]
\[ \# \text{ we consider } G \text{ as } \{g_1, \ldots, g_n\} \]
\[ P := \{(i, j) : 1 \leq i < j \leq n\} \]

while \( P \neq \emptyset \) do
  \begin{align*}
  \text{Pick } (i, j) & \in P; \\
  P & := P \setminus \{(i, j)\}; \\
  \text{Let } S(g_i, g_j) & \rightarrow^G h \\
  \text{If } h \neq 0 \text{ then} & \\
  \# \text{ lm}(h) & \notin (\text{lm}(G)) \\
  g_{n+1} & := h; \ G := G \cup \{h\}; \\
  P & := P \cup \{(i, n+1) : 1 \leq i \leq n\}; \\
  n & := n + 1;
  \end{align*}

Optionally \( G := \text{Interreduce}(G) \)

**Proof.** The polynomials added to \( G \) are reductions of S-polynomials of members of \( G \), and hence are in the same ideal as \( G \), and therefore of \( G_0 \). If this process terminates, then the result satisfies condition \([\text{I}]\) and so is a Gröbner base for some ideal, and therefore the ideal of \( G \). By proposition \([\text{II}]\) and the properties of \( \rightarrow^G \), \( h \in (G) \), so \( (G) \) is constant throughout this process and \( G \) has to be a Gröbner base for \( (G_0) \). Is it possible for the process of adding new \( h \) to \( G \), which implies increasing \( (\text{lm}(G)) \), to go on for ever? No: corollary \([\text{I}]\) says that the increasing chain of \( (\text{lm}(G)) \) is finite, so at some point we cannot increase \( (\text{lm}(G)) \) any further, i.e. we cannot add a new \( h \).

**Proposition 32** Every finitely generated polynomial ideal over a field \( K \) has a completely reduced Gröbner base with respect to any given ordering, and this is unique up to order of elements and multiplication by elements of \( K^* \).

Hence, for a fixed ordering, a monic crGb is a “fingerprint” of an ideal, uniquely identifying it. This makes definition \([\text{III}]\) algorithmic. It also allows ideal arithmetic.

**Proposition 33** Let \( G_1 \) and \( G_2 \) be Gröbner bases of the ideals \( I_1 \) and \( I_2 \) with respect to a fixed ordering. Then:

1. \( I_1 \triangleleft I_2 \iff \forall g \in G_1 \quad g \rightarrow^{G_2} 0; \)
2. \( I_1 + I_2 = (G_1 \cup G_2); \)
3. \( I_1 I_2 = \{(g_1g_2 \mid g_1 \in G_1, g_2 \in G_2)\}. \)

Furthermore, all these processes are algorithmic.

### 3.3.2 How many Solutions?

Here we will try to give an analysis of the various possibilities for the number of solutions of a set of polynomial equations. We will assume that a crGb for the polynomials has been computed, which therefore cannot be over-determined.
in the sense of having redundant equations. However, we may still need more equations than variables — see the examples at the start of section [3.3.7]

Unlike section [3.2.5] however, we have to ask ourselves “in which domain are the solutions?” We saw in Theorem 10 that, even for an equation in one variable, the ‘solutions’ may have no simpler formulation than ‘this is a root of \( p(x) \).’ Fortunately, this is all that we need. We will assume that \( K \) is the algebraic closure (definition 17) of (the field of fractions of) \( R \).

**Definition 44** The set of solutions over \( K \) of an ideal \( I \) is called the variety of \( I \), written \( V(I) \). If \( S \) is a set of polynomials which generates \( I \), so \( I = \langle S \rangle \), we will write \( V(S) \) as shorthand for \( V(\langle S \rangle) \).

We should note that two different ideals can have the same variety, e.g. \( (x) \) and \( (x^2) \) both have the variety \( x = 0 \), but the solution has different multiplicity. However, the two ideals \( (x^2 + y^2) \) and \( (x, y) \) both have only the solution \( x = y = 0 \) over the reals, but over the complexes the first has the solutions \( x = \pm iy \), and hence the varieties are different. See Section 3.5.2.

**Proposition 34** \( V(I_1 \cdot I_2) = V(I_1) \cup V(I_2) \).

**Proposition 35** \( V(I_1 \cup I_2) = V(I_1) \cap V(I_2) \).

**Definition 45** The radical of an ideal \( I \), denoted \( \sqrt{I} \), is defined as

\[
\sqrt{I} = \{ p | \forall x \in V(I), p(x) = 0 \}.
\]

An equivalent definition is

\[
\sqrt{I} = \{ p | \exists m : p^m \in I \}.
\]

If \( I \) is generated by a single polynomial \( p \), \( \sqrt{I} \) is generated by the square-free part of \( p \).

**Example 3** The ideal \( (x^2 + y^2) \) and the ideal \( (xy) \) are both radical. However, the ideal \( I = (x^2 + y^2, xy) \) is not radical, and in fact \( \sqrt{I} = (x, y) \).

Let us see how this happens. \( x^2 + y^2 \in I \) and \( xy \in I \), therefore \( x^2 + y^2 + 2xy \in I \). But this polynomial is \( (x + y)^2 \), therefore \( x + y \in \sqrt{I} \). Since \( xy \in \sqrt{I} \) and \( x + y \in \sqrt{I} \), we have \( xy - y(x + y) \in \sqrt{I} \). But this is just \(-y^2\), so \( y^2 \in \sqrt{I} \), and hence \( y \in \sqrt{I} \). Similarly \( x \in \sqrt{I} \).

In terms of varieties, \( V(I) \) is \( \{ x = 0, y = 0 \} \) (which in fact is with multiplicity two, since it is a point of multiplicity 2 in both \( (x^2 + y^2) \) and \( (xy) \)). A simpler (indeed the simplest) ideal with this variety is \( (x, y) \).

**Proposition 36** \( \sqrt{I} \) is itself an ideal, and \( \sqrt{I} \supset I \).

**Definition 46** The dimension of an ideal \( I \) in \( S = k[x_1, \ldots, x_n] \) is the maximum number of algebraically independent, over \( k \), elements of the quotient \( S/I \). We can also talk about the dimension of a variety.
3.3. NONLINEAR MULTIVARIATE EQUATIONS: DISTRIBUTED

No solutions in \( K \) Here the crGb will be \( \{1\} \), or more generally \( \{c\} \) for some non-zero constant \( c \). The existence of a solution would imply that this constant was zero, so there are no solutions. The dimension is undefined, but normally written as \( -1 \).

A finite number of solutions in \( K \) There is a neat generalisation of the result that a polynomial of degree \( n \) has \( n \) roots.

**Proposition 37** The number (with multiplicity) of solutions of a system with Gröbner basis \( G \) is equal to the number of monomials which are not reducible by \( G \).

It follows from this that, if (and only if) there are finitely many solutions, every variable \( x_i \) must appear alone, to some power, as the leading monomial of some element of \( G \). In this case, the dimension is zero. We return to this case in section 3.3.7.

An infinite number of solutions in \( K \) Then some variables do not occur alone, to some power, as the leading monomial of any element of \( G \). In this case, the dimension is greater than zero.

While ‘dimension’, as defined above, is a convenient generalisation of the linear case, many more things can happen in the non-linear case. If the dimension of the ideal is \( d \), there must be at least \( d \) variables which do not occur alone, to some power, as the leading monomial of any element of \( G \). However, if \( d > 0 \), there may be more. Consider the ideal \( (xy - 1)^{k[x,y]} \). \( \{xy - 1\} \) is already a Gröbner base, and neither \( x \) nor \( y \) occur alone, to any power, in a leading term (the only leading term is \( xy \)). However, the dimension is 1, not 2, because fixing \( x \) determines \( y \), and vice versa, so there is only one independent variable. In the case of a triangular set (definition 58), we can do much better, as in Proposition 45.

There are other phenomena that occur with nonlinear equations that cannot occur with linear equations. Consider the ideal

\[
\langle (x + 1 - y)(x - 6 + y), (x + 1 - y)(y - 3) \rangle
\]

(where the generators we have quoted do in fact form a Gröbner base, at least for \( \text{plex}(x,y) \) and \( \text{tdeg}(x,y) \), and the leading monomials are \( x^2 \) and \( xy \)). \( x \) occurs alone, but \( y \) does not, so in fact this ideal has dimension greater than 0 but at most 1, i.e. dimension 1. But the solutions are \( x = y - 1 \) (a straight line) and the point \( (3,3) \). Such ideals are said to be of mixed dimension, and are often quite tedious to work with [Laz09].

3.3.3 Orderings

In section 2.1.4 we defined an admissible ordering on monomials, and the theory so far is valid for all orderings. What sort of orderings are admissible? We first need an ordering on the variables themselves, which we will also denote \( > \), and
we will assume that $x_1 > \cdots > x_n$ (in examples, $x > y > z$). Suppose the two monomials to be compared are $A = x_1^{a_1} \cdots x_n^{a_n}$ and $B = x_1^{b_1} \cdots x_n^{b_n}$. These monomials have total degree $a = \sum_{i=1}^n a_i$ and $b = \sum_{i=1}^n b_i$.

purely lexicographic — plex in Maple We first compare $a_1$ and $b_1$. If they differ, this tells us whether $A > B$ ($a_1 > b_1$) or $A < B$ ($a_1 < b_1$). If they are the same, we go on to look at $a_2$ versus $b_2$ and so on. The order is similar to looking up words in a dictionary/lexicon — we look at the first letter, and after finding this, look at the second letter, and so on. In this order $x^2$ is more important than $xy^{10}$.

total degree, then lexicographic — grlex in Maple We first look at the total degrees: if $a > b$, then $A > B$, and $a < b$ means $A < B$. If $a = b$, then we look at lexicographic comparison. In this order $xy^{10}$ is more important than $x^2$, and $x^2y$ more important than $xy^2$.

total degree, then reverse lexicographic — tdeg in Maple This order is the same as the previous, except that, if the total degrees are equal, we look lexicographically, then take the opposite. Many systems, in particular Maple and Mathematica\footnote{http://reference.wolfram.com/mathematica/tutorial/PolynomialOrderings.html} reverse the order of the variables first. The reader may ask “if the order of the variables is reversed, and we then reverse the sense of the answer, what’s the difference?” Indeed, for two variables, there is no difference. However, with more variables it does indeed make a difference. For three variables, the monomials of degree three are ordered as

$$x^3 > x^2y > x^2z > xy^2 > xz^2 > y^3 > y^2z > yz^2 > z^3$$

under grlex, but as

$$x^3 > x^2y > xy^2 > y^3 > x^2z > xyz > y^2z > xz^2 > yz^2 > z^3$$

under tdeg. One way of seeing the difference is to say that grlex with $x > y > z$ discriminates in favour of $x$, whereas tdeg with $z > y > x$ discriminates against $z$. This metaphor reinforces the fact that there is no difference with two variables.

It seems that tdeg is, in general, the most efficient order.

$k$-elimination Here we choose any order $>^\prime$ on $x_1, \ldots, x_k$, and use that. If this cannot decide, we then use a second order $>^\prime\prime$ on $x_{k+1}, \ldots, x_n$. Since $>^\prime$ is admissible, the least monomial is $x_1^0 \cdots x_k^0$, so this order will eliminate $x_1, \ldots, x_k$ as far as possible, in the sense that the polynomials in only $x_{k+1}, \ldots, x_n$ in a Gröbner base computed with such an order are all that can be deduced about these variables. It is common, but by no means required, to use tdeg for both $>^\prime$ and $>^\prime\prime$. Note that this is not the same as simply using tdeg, since the exponents of $x_{k+1}, \ldots, x_n$ are not considered unless $x_1, \ldots, x_k$ gives a tie.
3.3. NONLINEAR MULTIVARIATE EQUATIONS: DISTRIBUTED

**weighted orderings** Here we compute the total degree with a weighting factor, e.g. we may weight \( x \) twice as much as \( y \), so that the total degree of \( x^i y^j \) would be \( 2i + j \). This can come in lexicographic or reverse lexicographic variants.

**matrix orderings** These are in fact the most general form of orderings \([\text{Rob}85]\). Let \( M \) be a fixed \( n \times n \) matrix of reals, and regard the exponents of \( A \) as an \( n \)-vector \( a \). Then we compare \( A \) and \( B \) by computing the two vectors \( M.a \) and \( M.b \), and comparing these lexicographically.

**lexicographic** \( M \) is the identity matrix.

\[
M = \begin{pmatrix}
1 & 1 & \ldots & 1 & 1 \\
1 & 0 & \ldots & 0 & 0 \\
0 & \ddots & 0 & \ldots & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
0 & \ldots & 0 & 1 & 0
\end{pmatrix}
\]

**tdeg** It would be tempting to say, by analogy with \( \text{grlex} \), that the matrix

\[
\begin{pmatrix}
1 & 1 & \ldots & 1 & 1 \\
0 & 0 & \ldots & 0 & 1 \\
0 & \ldots & 0 & 1 & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
0 & \ldots & 0 & 1 & 0
\end{pmatrix}
\]

is lexicographic \( M \) with the variable order reversed, not genuine reverse lexicographic. To get that, we need the matrix

\[
\begin{pmatrix}
1 & 1 & \ldots & 1 & 1 \\
-1 & 0 & \ldots & 0 & 0 \\
0 & \ddots & 0 & \ldots & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
0 & \ldots & 0 & -1 & 0
\end{pmatrix}
\]

adopting the Maple convention of reversing the variables as well.

**k-elimination** If the matrices are \( M_k \) for \( >' \) and \( M_{n-k} \) for \( >'' \), then

\[
M = \begin{pmatrix}
M_k & 0 \\
0 & M_{n-k}
\end{pmatrix}
\]

**weighted orderings** Here the first row of \( M \) corresponds to the weights, instead of being uniformly 1.

Most “serious” Gröbner systems\(^1\) implement matrix orderings, but have special case implementations for the more common ones listed above, often storing the (weighted) total degree as well as the individual degrees to minimise recomputation.

\(^1\)Such as SINGULAR \([\text{Sch}03b]\), CoCoA \([\text{Abb}04]\) or Macauley \([\text{BS}86]\).
3.3.4 Complexity of Gröbner Bases

We have proved nothing about the running time of Buchberger’s algorithm. Indeed, “algorithm” is almost an over-statement: we have not specified the choice of \((i, j) \in P\) at all. It turns out that the complexity, though not the correctness, of the algorithm is strongly dependent on this choice.

**Observation 3** There have been many improvements and alternative suggestions for computing Gröbner bases. Some improvements are given below. As of writing probably the best algorithm is the \(F_5\) algorithm [Fau02, HA10]. Section 4.6 discusses ways of reducing the coefficient growth in these computations.

**Observation 4** However many improvements have been, or might be, made, computing Gröbner bases can never be a trivial task. A classic result [MM82] shows that the output degree can be doubly exponential in the number of variables, and this is also the worst case [Dub90].

**Theorem 18** ([MR10, Theorem 4.1]) For any \(d\) and \(k\), let \(n = 14(k + 1)\). There is a set \(J_{n,d}\) of polynomials in \(k[x_1, \ldots, x_n]\) of degree at most \(d\) such that any Gröbner base of \(J_{n,d}\) with respect to a total degree ordering contains a polynomial of degree at least \(\frac{1}{2} d^2 + 4\).

Therefore this is \(O \left( d^{2^{n/14}} \right)\). We can improve this to “nearly” \(n/2\), at the cost of being less explicit.

**Theorem 19** ([?]) Fix an admissible monomial ordering. Then there is a family of ideals \(I_n \subseteq k[x_1, \ldots, x_n]\) for \(n \in \mathbb{N}\) generated by \(O(n)\) polynomials of degree \(\leq d\), such that any Gröbner base \(G_n\) of \(I_n\) has degree \(\geq d^n(1/2 - \epsilon)^n\) for any \(\epsilon > 0\) and sufficiently large \(d, n\).

**Theorem 20** ([Dub90]) Whatever the ordering, polynomials of total degree \(\leq d\) in \(k[x_1, \ldots, x_n]\) have a reduced Gröbner base with polynomials of degree \(\leq 2 \left( \frac{d^2}{2} + d \right)\).

**Open Problem 6** (Sparse Gröbner Bases) Even when \(k = 1\), we are very far away from being able in practice to compute Gröbner bases in 28 variables. Also, of course, these polynomials could be very sparse, so this result does not necessarily prove that the complexity of computing Gröbner bases is doubly exponential in a sparse encoding. What, then, are the more practical implications of Theorems 18, 19?

We note that the degree of a Gröbner base depends strongly on the dimension (Definition 46) of the ideal — see section 3.3.8 for the zero-dimensional case, and more generally the following result.

**Theorem 21** ([MR10, Corollary 3.21]) Let \(I\) be an \(r\)-dimensional ideal in \(k[x_1, \ldots, x_n]\) generated by \(s\) polynomials of total degree \(d_1 \geq \ldots \geq d_s\). Then, for any admissible ordering, the maximum total degree required in a Gröbner base of \(I\) is at most \(2 \left( \frac{1}{2} d_1 \cdots d_{n-r} + d_1 \right)^2\).
These ideals are very far from being radical (Definition 45), and indeed the complexity comes from this property. For radical ideals, we have a much better degree bound.

**Theorem 22 TO BE COMPLETED**?

However, this doesn’t mean that the Gröbner bases are small, as shown by this example.

**Example 4 (van Hoeij [vH15])** Consider 3n variables \( V := \{x_1, \ldots, x_n, y_1, \ldots, y_n, z_1, \ldots, z_n\} \). Let us work modulo 2, and also with the equations \( S := \{c^2 - c : \forall c \in V\} \), so effectively this is the Boolean ring. Let \( \mathcal{L} := \{x_i y_i - x_i - y_i : i \in \{1, \ldots, n\}\} \) and \( \mathcal{H} := S \cup \mathcal{L} \cup \{\prod_{i=1}^n z_i\} \). Intuitively, the last equation in \( \mathcal{H} \) says that (at least) one of the \( z_i \) is zero, and if \( z_i = 0 \), the \( i \)-th equation in \( \mathcal{L} \) says that \( x_i = y_i = 0 \). If one is familiar with algebraic geometry, it is easy to see that \( (\mathcal{H}) \) is a radical ideal.

Let \( \mathcal{T} = \{x_i z_i - x_i : i \in \{1, \ldots, n\}\} \cup \{y_i z_i - y_i : i \in \{1, \ldots, n\}\} \) and \( \mathcal{P} = \{\prod_{i=1}^n c_i : c_i \in \{x_1, y_1, z_1\}, \ldots, c_n \in \{x_n, y_n, z_n\}\} \). Note that, while all the other sets have \( O(n) \) elements, \( \mathcal{P} \) has \( 3^n \) elements. van Hoeij shows that, with respect to any total degree ordering, the reduced Gröbner base of \( \mathcal{H} \) is \( \mathcal{G} := S \cup \mathcal{L} \cup \mathcal{T} \cup \mathcal{P} \), i.e. with exponentially more elements.

The author has also observed that, with the lexicographic order and the \( z_i \) coming first, the Gröbner base only has those elements of \( \mathcal{P} \) containing no \( z_i \), i.e. \( 2^n \) of them. So van Hoeij’s example is also one where any total degree order has exponentially more polynomials than a good lexicographic order.

Having got, as it were, the depressing news out of the way, let’s look at some useful results in practice.

**Proposition 38 (Buchberger’s gcd (or First) Criterion [Buc79])** If

\[
\gcd(\text{lcm}(f), \text{lcm}(g)) = 1,
\]

then \( S(f, g) \xrightarrow{\ast} 0 \).

In practice, this is implemented by not even adding to \( P \), either in the initial construction or when augmenting it due to a new \( h \), pairs for which equation (3.29) is satisfied. This proposition also explains why the more general construct of an \( S \)-polynomial is not relevant to linear equations: when \( f \) and \( g \) are linear, if they have the same leading variable, one can reduce the other, and if they do not, then the \( S \)-polynomial reduces to zero.

**Proposition 39 (Buchberger’s lcm (or Third) Criterion [Buc79])** If \( I = (B) \) contains \( f \), \( g \), \( h \) and the reductions under \( \xrightarrow{\ast} B \) of \( S(f, g) \) and \( S(f, h) \), and if both \( \text{lcm}(\text{lcm}(f), \text{lcm}(g)) \) and \( \text{lcm}(\text{lcm}(f), \text{lcm}(h)) \) divide \( \text{lcm}(\text{lcm}(g), \text{lcm}(h)) \), then \( S(g, h) \xrightarrow{\ast} 0 \), and hence need not be computed.
This has been generalised to a chain of polynomials $f_i$ connecting $g$ and $h$: see [BF91].

Propositions 38 and 39 are therefore sufficient to say that we need not compute an $S$-polynomial: the question of whether they are necessary is discussed by [HP07]. Terminology varies in this area, and some refer to Buchberger’s Second Criterion as well. The more descriptive gcd/lcm terminology is taken from [Per09].

Whereas applying the gcd Criterion (Proposition 38) to $S(f,g)$ depends only on $f$ and $g$, applying the lcm Criterion (Proposition 39 and its generalisations) to $S(g,h)$ depends on the whole past history of the computation. It might be that the Criterion is not applicable now, but might become applicable in the future. Hence we can ask which way of picking elements from $P$ in Algorithm 9 will maximise the effectiveness of the lcm Criterion?

A partial answer was given in [Buc79].

**Definition 47** We say that an implementation of Algorithm 9 follows a normal selection strategy if, at each stage, we pick a pair $(i,j)$ such that $\text{lcm}(\text{lcm}(g_i), \text{lcm}(g_j))$ is minimal with respect to the ordering in use.

This does not quite specify the selection completely: given a tie between $(i,j)$ and $(i',j')$ (with $i < j$, $i' < j'$), we choose the pair $(i,j)$ if $j < j'$, otherwise $(i',j')$ [GMN+91].

For the sake of simplicity, let us assume that we are dealing with a total degree ordering, or a lexicographic ordering. The case of weighted orderings, or so-called multi-graded orderings (e.g. an elimination ordering each of whose components is a total degree ordering) is discussed in [BCR11].

**Definition 48** A polynomial is said to be homogeneous if every term has the same total degree. A set of polynomials is said to be homogeneous if each of them separately is homogeneous. Note that we are not insisting that all terms in the set have the same degree, merely that within each polynomial they have the same total degree.

**Definition 49** If $f = \sum c_i \prod_{j=1}^{n} x_j^{a_{i,j}} \in K[x_1,\ldots,x_n]$ is not homogeneous, and has total degree $d$, we can define its homogenisation to be $f_0 \in K[x_0, x_1,\ldots,x_n]$ as

$$f_0 = \sum c_i x_0^{d - \sum_{j=1}^{n} a_{i,j}} \prod_{j=1}^{n} x_j^{a_{i,j}}.$$

**Proposition 40** If $f$ and $g$ are homogeneous, so is $S(f,g)$ and $h$ where $f \rightarrow^g h$.

**Corollary 8** If the input to Algorithm 9 is a set of homogeneous polynomials, then the entire computation is carried out with homogeneous polynomials.

---

12This can be extended to weighted homogeneous orderings, and most of the advantages carry through [FSEDT14].
3.3. NONLINEAR MULTIVARIATE EQUATIONS: DISTRIBUTED

The normal selection strategy is observed to work well with homogeneous polynomials, but can sometimes be very poor on non-homogeneous polynomials. Hence [GMN+91] introduced the following concept.

**Definition 50** The ‘sugar’ $S_f$ of a polynomial $f$ in Algorithm 9 is defined inductively as follows:

1. For an input $f \in G_0$, $S_f$ is the total degree of $f$ (even if we are working in a lexicographic ordering)
2. If $t$ is a term, $S_{ft} = \text{deg}(t) + S_f$;
3. $S_{f+g} = \max(S_f, S_g)$.

We define the sugar of a pair of polynomials to be the sugar of their $S$-polynomial, i.e. (the notation is not optimal here!) $S_{(f,g)} = S_{S(f,g)}$.

The sugar of a polynomial is then the degree it would have had we homogenised all the polynomials before starting Algorithm 9.

**Definition 51** We say that an implementation of Algorithm 9 follows a sugar selection strategy if, at each stage, we pick a pair $(i,j)$ such that $S_{(g_i,g_j)}$ is minimal.

This does not completely specify what to do, and it is usual to break ties with the normal selection strategy (Definition 47), and “sugar then normal” is generally just referred to as “sugar”.

### 3.3.5 A Matrix Formulation

Equation (3.13) showed how a family of linear equations can be represented as a matrix equation. We can do the same with nonlinear equations: (3.26) can be written as

$$
\begin{pmatrix}
1 & 0 & 0 & 0 & -1 \\
0 & 1 & 0 & 0 & -1 \\
\end{pmatrix}
\begin{pmatrix}
x^2 \\
xy \\
x \\
y \\
1 \\
\end{pmatrix} = 0
$$

(3.30)

However, this does not give us an obvious solution. Rather, we need to extend the system, allowing not just the original equations, but also $y$ times the first and $x$ times the second, to give the following.

$$
\begin{pmatrix}
1 & 0 & 0 & 0 & -1 & 0 \\
0 & 1 & 0 & 0 & 0 & -1 \\
1 & 0 & 0 & -1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & -1 \\
\end{pmatrix}
\begin{pmatrix}
x^2y \\
x^2 \\
xy \\
x \\
y \\
\end{pmatrix} = 0.
$$

(3.31)
Elimination in this gives us

$$\begin{pmatrix} 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} x^2 y \\ x^2 \\ xy \\ x \\ y \\ 1 \end{pmatrix} = 0, \quad (3.32)$$

which produces, as the third row, the equation $y - x$, as we do (up to a change of sign) after (3.26). In pure linear algebra, we can do no further, since we really require $y$ times this equation. This means considering

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} x^2 y^2 \\ x^2y \\ x^2 \\ xy^2 \\ xy \\ x \\ y^2 \\ y \\ 1 \end{pmatrix} = 0. \quad (3.33)$$

Eliminating here (using row 1 to kill the leading term in row 4, and the same with row 2 against row 5) gives

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} x^2 y^2 \\ x^2y \\ x^2 \\ xy^2 \\ xy \\ x \\ y^2 \\ y \\ 1 \end{pmatrix} = 0. \quad (3.34)$$

and now row 4 can kill the leading term in row 6, to give

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} x^2 y^2 \\ x^2y \\ x^2 \\ xy^2 \\ xy \\ x \\ y^2 \\ y \\ 1 \end{pmatrix} = 0. \quad (3.35)$$

The last line of this corresponds to $y^2 - 1$. To use this to deduce an equation for $x$, we would need to consider $x$ times this equation, which would mean adding further rows and columns to the matrix.
In this formulation, the statement “$G$ is a Gröbner base for $(F)$” corresponds to the existence (not the uniqueness, as reduction is not unique) of matrices $X$, $Y$ and $R$ such that

\[
\begin{align*}
X.F &= G \\
Y.G &= F \\
R.G &= 0
\end{align*}
\]

(3.36)

where $F$ and $G$ are the matrix versions of $F$ and $G$: the first two lines say that $F$ and $G$ generate the same ideal, and the last that $G$ is a Gröbner basis.

No-one would actually suggest doing this in practice, any more than any-one would compute a g.c.d. in practice by building the Sylvester matrix (which is actually the univariate case of this process), but the fact that it exists can be useful in theory, as we will find that the Sylvester matrix formulation of g.c.d. computation is useful in chapter 4. See section 5.9.2.

### 3.3.6 Example

Consider the three polynomials below.

\[
\begin{align*}
g_1 &= x^3yz - xz^2, \\
g_2 &= xy^2z - xyz, \\
g_3 &= x^2y^2 - z.
\end{align*}
\]

The S-polynomials to be considered are $S(g_1, g_2)$, $S(g_1, g_3)$ and $S(g_2, g_3)$. We use a purely lexicographical ordering with $x > y > z$. The leading terms of $g_2 = xy^2z - xyz$ and $g_3 = x^2y^2 - z$ are $xy^2z$ and $x^2y^2$, whose l.c.m. is $x^2y^2z$. Therefore

\[
S(g_2, g_3) = xg_2 - zg_3 = (x^2y^2z - x^2yz) - (x^2y^2z - z^2) = -x^2yz + z^2.
\]

This polynomial is non-zero and reduced with respect to $G$, and therefore $G$ is not a Gröbner basis. Therefore we can add this polynomial (or, to make the calculations more readable, its negative) to $G$ — call it $g_4$. This means that the S-polynomials to be considered are $S(g_1, g_2)$, $S(g_1, g_3)$, $S(g_1, g_4)$, $S(g_2, g_4)$ and $S(g_3, g_4)$.

Fortunately, we can make a simplification, by observing that $g_1 = xg_4$, and therefore the ideal generated by $G$ does not change if we suppress $g_1$. This simplification leaves us with two S-polynomials to consider: $S(g_2, g_4)$ and $S(g_3, g_4)$.

\[
S(g_2, g_4) = xg_2 - yg_4 = -x^2yz + yz^2,
\]

and this last polynomial can be reduced (by adding $g_4$), which gives us $yz^2 - z^2$. As it is not zero, the basis is not Gröbner, and we must enlarge $G$ by adding this new generator, which we call $g_5$. The S-polynomials to be considered are $S(g_3, g_4)$, $S(g_2, g_5)$, $S(g_3, g_5)$ and $S(g_4, g_5)$.

\[
S(g_3, g_4) = zg_3 - yg_4 = -z^2 + yz^2,
\]
and this can be reduced to zero (by adding $g_5$). In fact, this reduction follows from Buchberger’s lcm (third) criterion, proposition 39, as we have already computed $S(g_2, g_3)$ and $S(g_2, g_4)$.

$$S(g_2, g_5) = zg_2 - xyg_5 = -xyz^2 + xyz^2 = 0,$$
$$S(g_4, g_5) = zg_4 - x^2g_5 = -z^3 + x^2z^2 = x^2z^2 - z^3,$$

where the last rewriting arranges the monomials in decreasing order (with respect to $<\!$). This polynomial is already reduced with respect to $G$, $G$ is therefore not a Gröbner basis, and we must add this new polynomial to $G$ — let us call it $g_6$. The S-polynomials to be considered are $S(g_3, g_5)$, $S(g_2, g_6)$, $S(g_3, g_6)$, $S(g_4, g_6)$ and $S(g_5, g_6)$. The reader can check that $G$ reduces all these S-polynomials to zero, and that $G$ is therefore a Gröbner basis of the ideal, viz.

$$g_2 = xy^2z - yz,$$
$$g_3 = x^2y^2 - z,$$
$$g_4 = x^2yz - z^2,$$
$$g_5 = yz^2 - z^2,$$
$$g_6 = x^2z^2 - z^3.$$

No power of $x$, $y$ or $z$ occurs alone, so we see that the variety is certainly not zero-dimensional, even though we started with three equations in three variables, and $z$ is undetermined. If $z \neq 0$, then $g_5$ can be divided by $z^2$ to give $y = 1$ and then $g_3$ becomes $x^2 - z$, hence this part of the solution variety is a parabola. But if $z = 0$, all equations except $g_3$ collapse, and we have $x^2y^2 = 0$. Hence this part of the solution variety is two straight lines $x = z = 0$ and $y = z = 0$, each in fact of multiplicity four. Hence the solution is in fact of dimension one, a fact that was not evident when we started.

### 3.3.7 The Gianni–Kalkbrener Theorem

In this section, we will consider the case of dimension 0, i.e. finitely many solutions over $K$. We first remark that the situation can be distinctly more challenging than in the case of linear equations, which we illustrate by means of two examples.

1. $G = \{x^2 - 1, y^2 - 1\}$. This is a Gröbner base with respect to any ordering. There are four irreducible monomials $\{1, x, y, xy\}$, and hence four solutions, $x = \pm 1, y = \pm 1$.

2. $G = \{x^2 - 1, y^2 - 1, (x - 1)(y - 1)\}$. This is also a Gröbner base with respect to any ordering. There are three irreducible monomials $\{1, x, y\}$, and hence three solutions. There are $x = 1, y = \pm 1$, but when $x = -1$, we only have $y = 1$. The additional polynomial $(x - 1)(y - 1)$, which rules out the monomial $xy$, rules out the solution $x = y = -1$. Another way of looking at this is that, when $x = 1$, the polynomial $(x - 1)(y - 1)$ vanishes, but when $x = -1$, it adds an extra constraint.
Can we generalise this? The answer is ‘yes’, at least for purely lexicographical Gröbner bases of zero-dimensional ideals. If the order is \( x_n < x_{n-1} < \cdots < x_1 \) then such a Gröbner base \( G \) must have the form

\[
p_n(x_n) \\
p_{n-1,1}(x_{n-1}, x_n), \ldots, p_{n-1,k_{n-1}}(x_{n-1}, x_n), \\
p_{n-2,1}(x_{n-2}, x_{n-1}, x_n), \ldots, p_{n-2,k_{n-2}}(x_{n-2}, x_{n-1}, x_n), \\
\ldots
\]

\[
p_{i,1}(x_1, \ldots, x_{n-1}, x_n), \ldots, p_{1,k_i}(x_1, \ldots, x_{n-1}, x_n),
\]

where\(^{13}\)

\[
\deg_{x_k}(p_{i,j}) \leq \deg_{x_k}(p_{i,j+1}) \tag{3.37}
\]

and \( p_{i,k_i} \) is monic in \( x_i \). Let \( G_k = G \cap k[x_k, \ldots, x_n] \), i.e. those polynomials in \( x_k, \ldots, x_n \) only.

**Theorem 23 (Gianni–Kalkbrenner [Gia89, Kal89])** Let \( \alpha \) be a solution of \( G_{k+1} \). Then if \( \text{lcm}(p_{k,i}) \) vanishes at \( \alpha \), then \( (p_{k,i}) \) vanishes at \( \alpha \). Furthermore, the lowest degree (in \( x_k \)) polynomial of the \( p_{k,i} \) not to vanish at \( \alpha \), say \( p_{k,m_{\alpha}} \), divides all of the other \( p_{k,j} \) at \( \alpha \). Hence we can extend \( \alpha \) to solutions of \( G_k \) by adding \( x_k = \text{RootOf}(p_{k,m_{\alpha}}) \).

This gives us an algorithm (Figure 3.4) to describe the solutions of a zero-dimensional ideal from such a Gröbner base \( G \). This is essentially a generalisation of back-substitution into triangularised linear equations, except that there may be more than one solution, since the equations are non-linear, and possibly more than one equation to substitute into.

In practice, particularly if we are interested in keeping track of multiplicities of solutions, it may be more efficient to perform a square-free decomposition (Definition 33) of \( p_n \), and initialise \( S \) to a list of the RootOf of each of its square-free factors, and also associate the multiplicity to each solution.

In case 2 above, the three equations are \( G = \{ x^2 - 1, y^2 - 1, (x - 1)(y - 1) \} \). Taking \( x_n = x \), we start off with \( S = \{ x = \text{RootOf}(x^2 - 1) \} \), and we call GKstep on this. The initial value of \( L \) is \( \text{RootOf}(x^2 - 1) - 1 \), and we ask whether this is invertible. Adopting a common-sense (i.e. heuristic) approach for the moment, we see that this depends on which root we take: for +1 it is not invertible, and for −1 it is. Hence GKstep makes two recursive calls to itself, on \( x = 1 \) and \( x = -1 \).

**GKstep(G, 1, \{ x = 1 \})** Here \( L := \text{lcm}_x(p_{1,1}(x = 1)) \) is 0, so we consider \( p_{1,2} \), whose leading coefficient is 1, so \( y = \text{RootOf}(y^2 - 1) \).

**GKstep(G, 1, \{ x = -1 \})** Here \( L := \text{lcm}_x(p_{1,1}(x = -1)) \) is −2, and \( y = 1 \).

\(^{13}\)It is tempting to write < rather than ≤ in (3.37). However, this is not always possible, even though, by the time we come to use the \( p_{k,j} \) in Algorithm 11, the non-zero \( p_{k,j}(\alpha) \) will indeed be in a strict order of \( \deg_{x_k} \). We can, of course sort the \( p_{k,j} \) by the purely lexicographic order.
Algorithm 10 (Gianni–Kalkbrener) \( GK(G, n) \)

**Input:** A Gröbner base \( G \) for a zero-dimensional ideal \( I \) in \( n \) variables with respect to lexicographic order.

**Output:** A list of solutions of \( G \).

\[
S := \{x_n = \text{RootOf}(p_n)\}
\]

for \( k = n - 1, \ldots, 1 \) do

\[
S := \text{GKstep}(G, k, S)
\]

return \( S \)

Algorithm 11 (Gianni–Kalkbrener Step) \( \text{GKstep}(G, k, A) \)

**Input:** A Gröbner base \( G \) for a zero-dimensional ideal \( I \) with respect to lexicographic order, an integer \( k \), and \( A \) a list of solutions of \( G_{k+1} \).

**Output:** A list of solutions of \( G_k \).

\[
B := \emptyset
\]

for each \( \alpha \in A \)

\[
i := 1
\]

while \( (L := (\text{lc}_{x_k}(p_{k,i})))(\alpha)) = 0 \) do \( i := i + 1 \)

if \( L \) is invertible with respect to \( \alpha \)

\# see section 3.1.5

then \( B := B \cup \{\alpha \cup \{x_k = \text{RootOf}(p_{k,i}(\alpha))\}\} \)

else \ # \( \alpha \) is split as \( \alpha_1 \cup \alpha_2 \)

\[
B := B \cup \text{GKstep}(G, k, \{\alpha_1\}) \cup \text{GKstep}(G, k, \{\alpha_2\})
\]

return \( B \)
There is a larger worked example of this later, at equation (3.47), and a generalisation of the above at http://staff.bath.ac.uk/masjhd/JHD-CA/WorkedGK.html.

Theorem 23 relies on a special case of a more general result.

**Theorem 24 (Elimination Theorem)** Suppose \( \{u_1, \ldots, u_k\} \subset \{x_1, \ldots, x_n\} \) and \( \prec \) is an elimination ordering (page 96), only considering the \( u_i \) if the ordering on the rest of the variables is a tie. If \( G \) is a Gröbner basis for an ideal \( I \) with respect to \( \prec \), then \( G \cap k[u_1, \ldots, u_k] \) is a Gröbner basis for \( I \cap k[u_1, \ldots, u_k] \) \[BW93, Proposition 6.15\]. \( I \cap k[u_1, \ldots, u_k] \) is called an elimination ideal for \( I \) with respect to the variables \( \{u_1, \ldots, u_k\} \).

Theorem 23 can be generalised in several ways to non-zero-dimensional ideals, but not completely \[FGT01, Examples 3.6, 3.11\]. The first is particularly instructive for us.

**Example 5** Let \( I = \langle ax^2 + x + y, bx + y \rangle \) with the order \( a \prec b \prec y \prec x \). The Gröbner base is \( B_1 \cup B_2 \) and there are no polynomials in \( (a, b) \) only (so the ideal is at least two-dimensional), in \( (a, b, y) \) we have \( B_2 := \{ay^2 + b^2y - by\} \), and in all variables \( B_1 := \{ax^2 + x + y, axy - by + y, bx + y\} \). We then have the following situation for values of \( a \) and \( b \).

normally \( B_2 \) determines \( y \) (generally as the solution of a quadratic), then \( x = -y/b \) except when \( b = 0 \), when \( ay^2 = 0 \) so \( y = 0 \), and \( ax^2 + x = 0 \), so \( x = 0 \) or \( x = -1/a \).

\( a = b = 0 \) \( B_2 \) vanishes, so we would be tempted, by analogy with Theorem 23, to deduce that \( y \) is undetermined. But in fact \( B_1|_{a=b=0} = \{x + y, y, y\} \), so \( y = 0 \) (and then \( x = 0 \)).

\( a = 0, b = 1 \) Again \( B_2 \) vanishes. This time, \( B_1|_{a=0, b=1} = \{x + y, 0, x + y\} \), and \( y \) is undetermined, with \( x = -y \).

This example is taken up again as Example 7.

**3.3.8 The Faugère–Gianni–Lazard–Mora Algorithm**

We have seen in the previous section that, for a zero-dimensional ideal, a purely lexicographical Gröbner base is a very useful concept. But these are generally the most expensive to compute, with a worst-case complexity of \( O(d^n) \) for polynomials of degree \( d \) in \( n \) variables \[CGH88\]. A total degree, reverse lexicographic Gröbner base, on the other hand, has complexity \( O(d^n) \), or \( O(d^n) \) if the number of solutions at infinity is also finite \[Laz83\]. If one prefers practical evidence, we can look at (4.26), which has a simple Gröbner base of (4.27), but 120-digit numbers occur in the intermediate calculations if we compute the Gröbner base in a total degree order, and numbers with tens of thousands of digits when computed in a lexicographic order. The computing time \[14\] is 1.02

\[14\] Axiom 3.4 on a 3GHz Intel P4.
CHAPTER 3. POLYNOMIAL EQUATIONS

or 2.04 seconds in total degree (depending on the order of variables), but 40126 seconds (over 11 hours) in lexicographic.

Hence the following algorithm [FGLM93] can be very useful, with $>'$ being total degree, reverse lexicographic and $''$ being purely lexicographical, though it does have uses in other settings as well.

Algorithm 12 (FGLM)

Input: A Gröbner base $G$ for a zero-dimensional ideal $I$ with respect to $>'$; an ordering $''$.

Output: A Gröbner base $H$ for $I$ with respect to $''$.

1. $H := \emptyset$; $i := j := 0$
2. Enumerate the monomials irreducible under $H$ in increasing order for $''$
   # When $H$ is a Gröbner base, this is finite by proposition 37
   # This enumeration needs to be done lazily — see (*)
3. for each such $m$:
   Let $m \rightarrow G^v$
   if $v = \sum_{k=1}^j c_k v_k$
   then $h_{i+1} := m - \sum_{k=1}^j c_k m_k$
   $H := H \cup \{h_{i+1}\}; i := i + 1$
   # Changes “irreducible under $H$” (*)
   else $j := j + 1; m_j := m; v_j := v$

return $H$

# It is not totally trivial that $H$ is a Gröbner base, but it is [FGLM93].

Since this algorithm is basically doing linear algebra in the space spanned by the irreducible monomials under $G$, whose dimension $D$ is the number of solutions (proposition 37), it is not surprising that the running time seems to be $O(D^3)$, whose worst case is $O(d^3 n)$. Strictly speaking, this analysis refers to the number of arithmetic operations, ignoring any growth in coefficient sizes.

Open Problem 7 (Complexity of the FGLM Algorithm (I)) The complexity of the FGLM algorithm (Algorithm 12) is $O(D^3)$ where $D$ is the number of solutions. Can faster matrix algorithms such as Strassen–Winograd [Str69, Win71] (see Notation 70) speed this up? We note that this is not trivial, since the rows are “arriving one at a time” rather than being presented all at once. See [FGHR13, FSEDT14] for recent progress: in particular the latter claims $O(nD^{2\omega})$.

Open Problem 8 (Complexity of the FGLM Algorithm (II)) The complexity of the FGLM algorithm (Algorithm 12) is $O(D^3)$ where $D$ is the number of solutions. Can we do any better in practice if the linear algebra is sparse? Some progress in this direction has been made in [FML3].

As an example of the FGLM algorithm, we take the system $\text{Aux}$ from their
paper\textsuperscript{15} with three polynomials
\[
abc + a^2bc + ab^2c + abc^2 + ab + ac + bc
\]
\[
a^2bc + a^2b^2c + b^2c^2a + abc + a + c + bc
\]
\[
a^2b^2c + a^2b^2c^2 + ab^2c + ac + 1 + c + abc
\]
The total degree Gröbner basis has fifteen polynomials, whose leading monomials are
\[
c^4, bc^3, ac^3, b^2c^2, abc^2, a^2c^2, b^3c, ab^2c, a^2bc, ac^3, b^4, a^2b^2, a^3b, a^4.
\]
This defines a zero-dimensional ideal (\(c^4, b^4\) and \(a^4\) occur in this list), and we can see that the irreducible monomials are
\[
1, c, c^2, c^3, b, bc, b^2, b^3, c, a, ac, c^2, ab, abc, a^2c, a^2b, a^3b, a^4.
\]
twenty in number (as opposed to the 64 we would have if the basis only had the polynomials \(a^4 + \cdots, b^4 + \cdots, c^4 + \cdots\)). If we wanted a purely lexicographic base to which to apply Gianni-Kalkbrener, we would enumerate the monomials in lexicographic order as
\[
1 \quad \text{(irreducible)}
\]
\[
c \quad \text{(irreducible)}
\]
\[
c^2 \quad \text{(irreducible)}
\]
\[
c^3 \quad \text{(irreducible)}
\]
\[
c^4 \text{ which reduces to } -\frac{185}{11} \frac{abc}{ac} + \frac{293}{7} a^3 - \frac{1153}{7} a^2b + \frac{509}{7} ab^2 - \frac{323}{11} b^3 - \frac{2035}{44} a^2c - \frac{821}{21} abc + \frac{173}{21} b^2c - \frac{526}{21} b^2c^2 + \frac{31}{44} c^3 - \frac{1449}{11} a^2 + \frac{1415}{11} ab - \frac{772}{21} b^2 + \frac{21}{11} ac - \frac{42}{7} bc + \frac{184}{7} c^2 - \frac{407}{6} a - \frac{261}{22} b - \frac{4799}{44} c
\]
\[
\vdots
\]
\[
c^{20} \text{ which reduces to } -\frac{1564732090555876438}{2415043982608847426} \frac{bc^2}{21} - \frac{24534744738888607327}{21} a^2c - \frac{455363768951017165}{21} b - \frac{521009308753678597}{21} c + \frac{1264966801336921700}{21} \frac{ab}{abc} - \frac{99597734526585822}{21} \frac{bc}{abc} + \frac{1369597713438958550}{21} \frac{bc}{abc} + \frac{1118956342568748374}{21} \frac{ac}{abc} - \frac{629351722358787789}{21} \frac{b^2c}{abc} + \frac{77412092228216564}{21} \frac{ac^2}{abc} - \frac{1416003666265496227}{21} a^2b + \frac{1196637152769448957}{21} \frac{a^2b}{abc} - \frac{7065265798247671}{21} \frac{a^2c}{abc} + \frac{1536916645521260147}{21} b^2c - \frac{41787128515094524}{21} \frac{a^3}{abc} - \frac{35628865966988974}{21} b^3 + \frac{37361952754752163}{21} c^3,\]
which can be expressed in terms of the previous ones as \(p = -1 + 6c + 41 c^2 - 71 c^3 + 41 c^4 - 197 c^6 + 106 c^{16} + 6 c^{19} - 106 c^4 - 71 c^{17} - 92 c^5 - 197 c^6 - 145 c^7 - 257 c^8 - 278 c^9 - 201 c^{10} - 278 c^{11} - 257 c^{12} - 145 c^{13} - 92 c^{15}.
\]
The polynomial \(c^{20} - p\) gets added to \(H\): all higher powers of \(c\) are therefore expressible, and need not be enumerated.
\textsuperscript{15} The system is obtained as they describe, except that the substitutions are \(x_5 = 1/c, x_7 = 1/a.\)
Open Problem 9 (Coefficient growth in the FGLM Algorithm) We observe above that the coefficient growth in the expression of $c^{20}$ in terms of the $>^t$ monomials is far greater than its expression in terms of the $>^u$ monomials (powers of $c$). Could fraction-free methods as in Theorem 15 do better?

$b$ which can be expressed in terms of the previous ones as

$$b = \frac{974153}{464537} c^{18} + \frac{14067876}{1645371} c^{17} + \frac{2335702}{1645371} c^{16} + \frac{13420192}{182819} c^{14} + \frac{79900378}{1645371} c^{16} + \frac{1389370}{11193} c^{12} + \frac{3378002}{42189} c^{4} - \frac{358660781}{1645371} c^{10} + \frac{1395703}{548457} c^{18} + \frac{1868291}{4459} c^{5} + \frac{1389370}{11193} c^{6} + \frac{337505020}{1645371} c^{7} + \frac{118784873}{548457} c^{8} + \frac{271667666}{1645371} c^{9}.$$

$q^b$ is added to $H$ and all multiples of $b$ are therefore expressible, and need not be enumerated.

$a$ which can be expressed in terms of the previous ones as $r = \frac{487915}{705159} c^{18} - \frac{301514}{705159} c^{17} - \frac{2038824}{705159} c^{16} - \frac{1053359}{235053} c^{15} + \frac{1987863}{235053} c^{14} - \frac{4113489}{200720} c^{13} + \frac{5090095}{100737} c^{12} - \frac{6527}{705159} c^{11} - \frac{705159}{18081} c^{10} - \frac{1395703}{548457} c^{9} + \frac{1868291}{4459} c^{5} + \frac{1389370}{11193} c^{6} + \frac{337505020}{1645371} c^{7} + \frac{118784873}{548457} c^{8} + \frac{271667666}{1645371} c^{9}.$

These last three give us the Gröbner base in a purely lexicographical order, which looks like $\{c^{20} + \ldots, b + \ldots, a + \ldots\}$. As there are twenty solutions in reasonably general position (the polynomial in $c$ alone does factor, but is square-free), we only need one polynomial per variable, as is often the case. The complete version of this, and another worked example, are given at [http://staff.bath.ac.uk/masjhd/JHD-CA/FGLMexample.html](http://staff.bath.ac.uk/masjhd/JHD-CA/FGLMexample.html).

The existence of this algorithm leads to the following process for ‘solving’ a zero-dimensional set of polynomial equations.

**Algorithm 13**

**Input:** A set $S$ of polynomials

**Output:** A ‘description of solutions’

$$G := \text{Buchberger}(S, >^\text{tdeg})$$

if $G$ is not zero-dimensional (Proposition 37)

then return “not zero-dimensional”

else $H := \text{FGLM}(G, >^\text{plex})$

Use Gianni–Kalkbrener to solve $H$

In the author’s experience, describing the solutions of a set of polynomial equations when the dimension is not zero is still rather an art form, but much aided by the computation of Gröbner bases: see the description at the end of Section 3.3.6.

Other ways of computing Gröbner bases over $\mathbb{Q}$ will be looked at in section 4.6.
3.3. NONLINEAR MULTIVARIATE EQUATIONS: DISTRIBUTED

Algorithm 14 (Extended Buchberger)

**Input:** finite \( G_0 = \{g^{(0)}_1, \ldots, g^{(0)}_m\} \subset R[x_1, \ldots, x_n]; \) monomial ordering >.

**Output:** \( G = \{g_1, \ldots, g_k\} \) a Gröbner base for \( (G_0) \) with respect to >.

Matrix \( M = (m_{i,j}) \) such that \( g_i = \sum m_{i,j}g_j^{(0)} \)

\[ G := G_0; \ n := |G|; \ M := n \times n \] Identity

\# we consider \( G \) as \( \{g_1, \ldots, g_n\} \)

\[ P := \{(i, j) : 1 \leq i < j \leq n\} \]

\[ \text{while } P \neq \emptyset \text{ do} \]

\[ \text{Pick } (i, j) \in P; \]

\[ P := P \setminus \{(i, j)\}; \]

\[ \text{Let } S(g_i, g_j) \rightarrow^G h \text{ } \# \text{ tracking } M \]

\[ \text{If } h \neq 0 \text{ then} \]

\[ \text{# } \text{lm}(h) \notin (\text{lm}(G)) \]

\[ g_{n+1} := h; \ G := G \cup \{h\}; \]

\[ P := P \cup \{(i, n+1) : 1 \leq i \leq n\}; \]

\[ n := n + 1; \]

\[ \text{Add new row to } M \]

\[ G := \text{Interreduce}(G) \]

We note that we have to keep track of the \( m_{i,j} \) during the reduction process as well as during the \( S \)-polynomial computation. We use the version that does the inter-reduction of the final basis, noting that this has also to update the \( m_{i,j} \).

3.3.9 The Gröbner Walk

Though the Gianni–Kalkbrener algorithm only works in dimension 0, we may still want lexicographical-order Gröbner bases in higher dimensions, again balk at the cost of computing them directly, and look for an alternative. The FGLM algorithm intrinsically relies on the dimension being zero. Another method, which does not, is the Gröbner walk, due originally to [CKM97], see also [AGK97, Tra00, FJLT07]. The fundamental observation is in [MR88, Lemma 2.6], that a given ideal only has finitely many possible (across all possible orderings) leading monomial ideals, and hence for this ideal there are only finitely many essentially different orderings. The space of orderings is then partitioned into polyhedral sets: a partition known as the Gröbner Fan.

We first need a variant of Algorithm 9, given in Figure 3.5, which is to Algorithm 9 as Algorithm 5 is to Algorithm 2.

Algorithm 15 (Gröbner Walk)

**Input:** A Gröbner base \( G \) for an ideal \( I \) with respect to >'; an ordering >".

**Output:** A Gröbner base \( H \) for \( I \) with respect to >".

The idea is to construct (generally incrementally) a sequence of orders >_{i}=>'.
CHAPTER 3. POLYNOMIAL EQUATIONS

>2, >3, ... >k=>

and corresponding Gröbner bases \( G_1 = G, \ldots, G_k = H \).

We use the matrix representation of orderings, so that \( >_i \) is given by the matrix \( M_i \) — see page 97. \( M_2 \) is special — its first row is the first row of \( M_1 \), and the remaining rows are the corresponding rows of \( M_k \). All subsequent \( M_i \) have the same rows two onwards, and their first rows are a sequence of 'hybrids' between the first row of \( M_2 \) and the first row of \( M_k \). Let \( \omega_i \) be the first row of \( M_i \). The reader who is familiar with homotopy methods may care to view this algorithm as a homotopy method between \( \omega_1 = \omega_2 \) and \( \omega_k \). There are then two open questions.

1. Which hybrids, i.e., which intermediate orderings, do we need? The answer, at least for the simple version of the Gröbner walk, is to let \( \omega_i = (1 - t_i)\omega_1 + t_i\omega_k \), so \( t_1 = t_2 = 0 \) and \( t_k = 1 \). The \( t_i \) are then computed by NextCritical.

2. How do we compute \( G_i \), given \( G_{i-1} \)? Naïvely, we could use Algorithm 9 for \( >_i \), ignoring the fact that \( G_{i-1} \) is a Gröbner base for \( >_{i-1} \), an ordering "fairly similar" to \( >_i \). We will see that we can do better.

**Definition 52** Let \( \omega = (\omega_1, \ldots, \omega_n) \) be a vector of (non-negative) rational numbers. The \( \omega \)-degree of a monomial \( \prod x_i^{a_i} \) is the sum \( \sum \omega_i a_i \). A polynomial is said to be \( \omega \)-homogeneous if every term has the same \( \omega \)-degree. A set of polynomials is said to be \( \omega \)-homogeneous if each of them separately is \( \omega \)-homogeneous.\(^{16}\) The \( \omega \)-initial form of a polynomial \( p \), denoted \( \text{init}_\omega(p) \), is the sum of all terms of maximal \( \omega \)-degree. If \( > \) is a monomial ordering the first row of whose matrix is \( \omega \), we write \( \text{init}_> \) as well as \( \text{init}_\omega \).

With Algorithm 14, we can express step 2 in the Gröbner Walk as "when the initials change, run Buchberger’s algorithm \( \{14\} \) on the new initials, then use the results to update the whole Gröbner base", as described in Figure 3.6. This depends crucially on the fact that we are only making minimal changes to the order, and the initials: see results in [AGK97].

The theoretical complexity of the Gröbner walk has not been analysed. The experimental results in [AGK97] show that, on zero dimensional ideals, it can be ten or more times faster than their implementation of FGML. Their implementation of the walk contains several improvements over the one we have outlined. There are two main sources of inefficiency in the algorithm as we have outlined it.

1. If many new terms suddenly appear in \( \text{init}_\omega(G) \), the computation of Algorithm 14 can become quite close to a full Gröbner base computation by Algorithm 9. This happens when our walk passes through the intersection of several polyhedral cones in the Gröbner fan, and can be avoided by perturbing the walk [AGK97 §3].

\(^{16}\)Note that we are not insisting that all terms in the set have the same \( \omega \)-degree, merely that within each polynomial they have the same \( \omega \)-degree.
Figure 3.6: Body of Algorithm 15

\[ i := 2; \quad \omega := \text{first row of } M_\succ; \]
\[ \tau := \text{first row of } M_{\succ''}; \]
\[ \textbf{while } (\succ_i \neq \succ'') \]
\[ \quad G' := \text{init}_\omega(G) \]
\[ \quad G'', M := \text{Algorithm 14}(G', \succ_i) \]
\[ \quad H := \text{Transform}(G, M, \succ_i) \]
\[ \quad t := \text{NextCritical}(H, \omega, \tau) \]
\[ \quad G := H; \quad i := i + 1; \quad \omega := \omega + t(\tau - \omega) \]
\[ \quad >_i := \text{Ordering with matrix as } >'' \text{ but first row } \omega \]

where

- Algorithm 14 is applied to an \( \omega \)-homogeneous set of polynomials, many of which will, generically, be monomials. This allows or some efficiency improvements, see [AGK97].
- \( \text{Transform}(G, M, >) \) computes \( h_i = \sum_j m_{i,j}g_j \) in order \( > \);
- \( \text{NextCritical}(H, \omega, \tau) \) computes the least \( t > 0 \) such that \( \text{init}_\omega(H) \neq \text{init}_{\omega + t(\tau - \omega)}(H) \).
2. In general, the \( t \) computed by NextCritical, especially if we perturb to walk to avoid the previous problem, can become quite complicated rational numbers. We “solve” this by clearing denominators, but then have to deal with large integers, as seen in the examples.

The worked versions of the examples of section 3.3.8 for this algorithm are given at [http://staff.bath.ac.uk/masjhd/JHD-CA/GWalkexample.html](http://staff.bath.ac.uk/masjhd/JHD-CA/GWalkexample.html). We note that, at least in Maple, for examples of dimension zero the FGLM process seems ten times faster than the Gröbner walk, which rather contradicts the experimental results in \[AGK97\].

**Open Problem 10** Perform more extensive experiments comparing the FGLM and Gröbner Walk examples.

### 3.3.10 Factorization and Gröbner Bases

It may happen, either initially or during the computation of a Gröbner basis, that we observe that a polynomial \( f_i \) factors as \( f_{i,1}f_{i,2} \) (or more, but for simplicity we consider the case of two factors. Since \( f_i(x_1, \ldots, x_n) = 0 \) is and only if one of \( f_{i,1}(x_1, \ldots, x_n) \) or \( f_{i,2}(x_1, \ldots, x_n) \) is zero, we can reduce the problem to two, hopefully simpler, ones. For “random” problems, this is unlikely\(^{17}\) to happen, but people do not generally ask “random” questions.

This observation\(^{18}\) can be powerful, but is not as simple to implement as might be expected. One ‘clearly’ modifies algorithm \([9]\) so that, if \( h = h_1h_2 \), instead of adding \( h \) to \( G \), one forks two copies of the algorithm, one with \( G \cup \{h_1\} \) and one with \( G \cup \{h_2\} \).

**TO BE COMPLETED**

### 3.3.11 The Shape Lemma

Let us look again at example 2 of section 3.3.7. Here we needed three equations to define an ideal in two variables. We note that interchanging the roles of \( x \) and \( y \) does not help (in this case, it might in others). However, using other coordinates than \( x \) and \( y \) definitely does. If we write the equations in terms of \( u = x + y, v = x - y \) instead, we get the basis

\[
[-4v + v^3, v^2 - 4 + 2u] : \tag{3.38}
\]

three values for \( v \) (0, 2 and \(-2\)), each with one value of \( u \) (2, 0 and 0 respectively), from which the solutions in \( x, y \) can be read off. Note that ordering \( v \) before \( u \) would give the basis

\[
[u^2 - 2u, uv, v^2 - 4 + 2u], \tag{3.39}
\]

\(^{17}\)A “random” zero-dimensional system will have a ‘shape basis’ (Definition \([53]\) and the nonlinear polynomial is a “random” polynomial, and therefore factors with zero probability.

\(^{18}\)Another case of (near)-simultaneous discovery: \([Dav87]\) (who could solve in 96 seconds problems that could not be solved in two hours), \([MNN89, NM92]\), and \([Hei92, Hei93]\) (solving Quantum Yang-Baxter equations).
which is not of this form: it has three polynomials in two variables.

This kind of operation is called, for obvious reasons, a rotation. Almost all rotations will place the equations “in general position”: and many theoretical approaches to these problems assume a “generic rotation” has been performed. In practice, this is a disaster, since sparsity is lost.

**Definition 53 ([BMMT94])** A basis for a zero-dimensional ideal is a shape basis if it is of the form

\[ \{ g_1(x_1), x_2 - g_2(x_1), \ldots, x_n - g_n(x_1) \}. \]

This is a Gröbner basis for any ordering in which ‘degree in \( x_1 \)’ is the first criterion: in the terminology of matrix orderings (page 97), any ordering where the first row of the matrix is \((\lambda, 0, \ldots, 0)\).

For a shape basis, the Gianni–Kalkbrener process is particularly simple: “determine \( x_1 \) and the rest follows”. Almost all zero-dimensional ideals have shape bases. The precise criterion (●) in the theorem below is somewhat technical, but is satisfied if there are no repeated components.

**Theorem 25 (Shape lemma) ([BMMT94], Corollary 3)** After a generic rotation, a zero-dimensional ideal has a shape basis if, and only if,

- each primary component is simple or of local dimension 1.

Furthermore ([BMMT94], Lemma 2], such a rotation need only be 1-generic, i.e. have matrix

\[
\begin{pmatrix}
1 & v \\
0 & I
\end{pmatrix}
\]

for some generic vector \( v \).

Their paper generalises this to ideals of higher dimension, but the complexity in notation is not worth it for our purposes.

A word of warning is in order here. The Shape Lemma is a powerful theoretical tool, but its application can be costly. Consider example 2 (page 104): \( G = \{ x^2 - 1, y^2 - 1, (x - 1)(y - 1) \} \). This is certainly not a shape basis, since it has more polynomials than indeterminates. This is inevitable, since the variety is not equiprojectable (see Definition 60 below) onto either \( x \) or \( y \). If we write \( s = x + y, t = x - y \), then the basis becomes \( \{ -4 + t^3, -4 + t^2 + 2s \} \) for the ordering \( s > t \), which is a shape basis. However, consider the similar basis \( G' = \{ x^{2n} - 1, y^{2n} - 1, (x^n - 1)(y^n - 1) \} \). Similar rotations will work, but \( t \) is now the root of a polynomial of degree \( 3n^2 \) with at least \( 3n + 1 \) nonzero coefficients, and quite large ones at that, e.g. for \( n = 3 \)

\[
\{ 8469703983104 - 1328571568128 t^3 + 56109155544 t^6 - 3387236203 t^9 + 149161506 t^{12} - 11557977 t^{15} + 279604 t^{18} - 1053 t^{21} - 78 t^{24} + t^{27},
-586877251095044672 t + 11229793345003520 t^4 - 363020550569195 t^7 + 24557528419410 t^{10} - 33283244425 t^{13} + 88786830300 t^{16} - 417476125 t^{19} - 23303630 t^{22} + 307217 t^{25} + 259387804304663680 s \}.
\]

But not for \( t > s \), since \( s \) defines the other line, besides the \( x \) and \( y \) axes, going through two of the points.
3.3.12 The Hilbert function

Let $I$ be any ideal (other than the whole ring) of $k[x_1,\ldots,x_n]$. Let $A$ be the algebra $k[x_1,\ldots,x_n]/I$, i.e. the set of all polynomials under the equivalence relation $f \equiv g$ if there is an $h \in I$ with $f = g + h$.

**Proposition 41** If $G$ is a Gröbner base for $I$, then $A$ is generated, as a $k$-vector space, by $M := \{ m \text{ monomial } \in k[x_1,\ldots,x_n] | m \rightarrow G \}$, i.e. the set of irreducible monomials.

We have already seen (Proposition 37) that the variety corresponding to $I$ is zero-dimensional if, and only if, $M$ is finite. In this case, $|M|$ is the number of solutions (counted with multiplicity).

The Hilbert function is a way of measuring $M$ when it is infinite.

**Definition 54** Let $A_l$ be the subset of $A$ where there is a representative polynomial of total degree $\leq l$. $A_l$ is a finite-dimension vector space over $k$. Let $H_I$ be the function $N \rightarrow N$ defined by

$$H_I(l) = \dim_k A_l.$$ (3.40)

**Proposition 42** If $G$ is a Gröbner base for $I$, then $A_l$ is generated, as a $k$-vector space, by $M := \{ m \text{ monomial } \in k[x_1,\ldots,x_n] | tdeg(m) \leq l \land m \rightarrow G \}$, i.e. the set of irreducible monomials of total degree at most $l$.

Note that, while $M$ itself will depend on $G$ (and therefore on the ordering used), Definition 54 defines an intrinsic property of $A$, and hence $|M|$ is independent of the order chosen.

**Theorem 26** ([BW93, Lemma 9.21]) \[ \left( \frac{l + d}{d} \right) \leq H_I(l) \leq \left( \frac{l + n}{m} \right). \]

**Theorem 27** ([BW93, Part of Theorem 9.27]) Let $G$ be a Gröbner base for $I \subseteq k[x_1,\ldots,x_n]$ under a total degree order. Let $I$ have dimension $d$, and let

$$N = \max \{ \deg_{x_i}(\text{lm}(g)) \mid g \in G; 1 \leq i \leq n \}.$$ 

Then there is a unique polynomial $h_I \in \mathbb{Q}[X]$ such that $H_I(l) = h_I(l)$ for all $l \geq nN$. $h_I$ is called the Hilbert polynomial of $I$.

3.3.13 Coefficients other than fields

Most of the theory of this section, notably theorem 16, goes over to the case when $R$ is a P.I.D. (definition 12) rather than a field, as described in [BW93, section 10.1]. However, when it comes to computation, things are not quite so obvious. What is the Gröbner base of $\{2x, 3y\}$? There are two possible answers to the question.

- $\{2x, 3y\}$ [Tri78].
3.3. NONLINEAR MULTIVARIATE EQUATIONS: DISTRIBUTED

- \{2x, 3y, xy\} (where \(xy\) is computed as \(x(3y) - y(2x)\)) [Buc84].

We note that \(xy = x(3y) - y(2x) \in (2x, 3y)\), so we need \(xy\) to reduce to zero. We therefore modify definition 43 as follows

**Definition 55** Let \(f, g \in R[x_1, \ldots, x_n]\). Suppose the leading coefficients of \(f\) and \(g\) are \(a_f\) and \(a_g\), and the leading monomials \(m_f\) and \(m_g\). Let \(a\) be a least common multiple of \(a_f\) and \(a_g\), and write \(a = a_f b_f = a_g b_g\). Let \(m\) be the least common multiple of \(m_f\) and \(m_g\). The \(S\)-polynomial of \(f\) and \(g\), written \(S(f, g)\), is defined as

\[
S(f, g) = b_f \frac{m}{m_f} f - b_g \frac{m}{m_g} g. \tag{3.41}
\]

Let \(c_f a_f + c_g a_g = \gcd(a_f, a_g)\), and define the \(G\)-polynomial of \(f\) and \(g\), written \(G(f, g)\), as

\[
G(f, g) = c_f \frac{m}{m_f} f + c_g \frac{m}{m_g} g. \tag{3.42}
\]

Note that (3.41) is the same as (3.28) up to a factor of \(\gcd(a_f, a_g)\). The \(S\)-polynomial is defined up to unit factors, whereas the \(G\)-polynomial is much less well-defined, but it turns out not to matter.

For the example quoted above, \(S(2x, 3y) = 0\) (which follows from Proposition 38), while \(G(2x, 3y) = xy\). Algorithm 9 goes over to this setting, except that we have to add \(G\)-polynomials as well as \(S\)-polynomials, and some care has to be taken to eliminate \(G\)-polynomials first — see [BW93, table 10.1].

3.3.14 Non-commutative Ideals

Much of the general mechanism of ideals generalises to the case of non-commutative ideals, provided we are careful to distinguish left, right or two-sided ideals. However, the theory is notably weaker. In particular we have the following opposite of theorem and its corollary.

**Proposition 43** \(K\langle x_1, \ldots, x_n \rangle\) is not noetherian for \(n \geq 2\).

Hence Buchberger’s algorithm might not terminate, and in general it does not [Mor86].

In fact, not only does this approach not work, but no approach can, as demonstrated by this result.

**Proposition 44 ([KRW90])** Ideal membership is insoluble in \(Q\langle x_1, x_2 \rangle\).

One case of great interest is when \(R\) is some field of (expressions representing) functions, and the “indeterminates” are differential or difference operators.

**Example 6** \(R\) is \(Q(x, y)\) and the indeterminates are \(\frac{\partial}{\partial x}\) and \(\frac{\partial}{\partial y}\), so that we are working in \(R[\frac{\partial}{\partial x}, \frac{\partial}{\partial y}]\). Here the “indeterminates” commute with each other, but not with \(R\), since \(\frac{\partial}{\partial x}(xf) = f + x \frac{\partial}{\partial x} f\), i.e. \(\frac{\partial}{\partial x} x = 1 + x \frac{\partial}{\partial x}\).
CHAPTER 3. POLYNOMIAL EQUATIONS

We should note that the result of multiplying a term by an indeterminate is not necessarily a term, e.g. \( \frac{\partial}{\partial x} \left( x \frac{\partial}{\partial x} \right) = \frac{\partial}{\partial x} + x \frac{2\partial^2}{\partial x^2} \). This makes characterising a Gröbner base harder, but the following definition is an appropriate generalisation of the last clause of theorem \([16]\) in the setting where the indeterminates commute with each other.

**Definition 56** [Pau07, Definition 4] A finite subset \( G \) of \( I \setminus \{0\} \), where \( I \) is a left-ideal, is a Gröbner basis of \( I \) iff, for all monomials \( m \), the \( R \)-ideal \( (\text{lc}(f) | f \in I \land \text{lm}(f) = m) \) is generated by \( \{\text{lc}(g) | g \in G \land \text{lm}(g) \text{ divides } m\} \).

If \( R \) is a principal ideal domain, it is possible to define \( S \)-polynomials and \( G \)-polynomials as in the previous section, but in general we need to consider more complicated (but still finitely many) combinations [Pau07]. This leads to an effective test for a Gröbner base in this setting, i.e. we need to check that finitely many combinations reduce to zero. We also get a generalisation of Buchberger’s algorithm [Pau07, Proposition 10]: if the combination does not reduce to zero, add it. Termination is non-trivial, however.

### 3.4 Nonlinear Multivariate Equations: Recursive

Whereas the previous section looked at polynomials as living in \( k[x_1, \ldots, x_n] \) thought of (irrespective of implementation) as a distributed way (page 41), with a certain order on the monomials, it is equally possible to think of them in a recursive way (page 41), but now with an order on the variables, rather than the monomials.

#### 3.4.1 Triangular Sets and Regular Chains

An alternative approach to polynomial equation solving is that of characteristic [Rit32, Wu86] or triangular [Laz91] sets, or regular chains [20]. See [ALM99] for a reconciliation of the various theories, and [AM99] for a practical comparison. We can regard regular chains as an approach based on recursive views of polynomials, while Gröbner bases are based on a distributed view. We assume that the variables \( x_1, \ldots, x_n \) are ordered as \( x_1 < \ldots < x_n \), so that \( x_n \) is the most important variable.

**Definition 57** Let \( p \) be a polynomial. The main variable of \( p \), denoted \( \text{mvar}(p) \), is the most important variable of \( p \). The initial of \( p \), written \( \text{init}(p) \), is its leading coefficient, when regarded as a univariate polynomial in \( \text{mvar}(p) \).

If \( S = \{p_1, \ldots, p_k\} \) is a finite set of polynomials, we write

\[
\text{init}(S) = \text{lcm}_{1 \leq i \leq k} \text{init}(p_i).
\]

[20] Terminology in this area has been confused: we are following a recent reconciliation of terminology.
3.4. NONLINEAR MULTIVARIATE EQUATIONS: RECURSIVE

It should be noted that many authors define $\text{init}(S)$ as $\prod_{1\leq i\leq k} \text{init}(p_i)$. Since we are normally concerned with the zeros of $\text{init}(S)$, the two definitions have the same consequences, and ours leads to smaller polynomials.

**Definition 58** A set $T$ of polynomials is said to be triangular if different polynomials have different main variables.

Example 2 of section 3.3.7 shows\(^{21}\) that there may not always be a triangular set generating a particular ideal. If we have a triangular set, then the structure of the ideal, and the variety, is relatively obvious.

**Definition 59** Let $T$ be a triangular set generating an ideal $I$ in $k[x_1,\ldots,x_n]$. Then every variable $x_i$ which occurs as a main variable is called algebraic, and the set of such variables is denoted $\text{AlgVar}(T)$.

**Proposition 45** For a triangular set $T$, the dimension of $I(T)$ is $n - |\text{AlgVar}(T)|$.

### 3.4.2 Zero Dimension

Much of the theory applies to positive dimension as well, but we will only consider in this section the case of zero-dimensional ideals/varieties. Let $V$ be a zero-dimensional variety, and $V_k$ be its projection onto $x_1,\ldots,x_k$, i.e.

$$V_k = \{(a_1,\ldots,a_k) : \exists (a_1,\ldots,a_n) \in V\}.$$ 

**Definition 60** A zero-dimensional variety $V$ is equiprojectable iff, for all $k$, the projection $V_k \rightarrow V_{k-1}$ is an $n_k : 1$ mapping for some fixed $n_k$. Note that this definition depends on the order of the $x_i$: a variety might be equiprojectable with respect to one order, but not another, as in (3.38) versus (3.39).

Such an equiprojectable variety will have $\prod n_k$ points (i.e. solutions, not counting multiplicity, to the equations).

The variety $V$ of Example 2 of section 3.3.7 is $\{(x = -1, y = 1), (x = 1, y = \pm 1)\}$ and is not equiprojectable. In fact, its equations can be written as $\{(x^2 - 1), (x - 1)(y - 1) + (x + 1)(y^2 - 1)\}$, which is a triangular set with $y$ more important than $x$ (main variables $x$ and $y$ respectively). However, the second polynomial sometimes has degree 1 in $y$ (if $x = -1$), and sometimes degree 2. Hence we need a stronger definition.

**Definition 61** A list, or chain, of polynomials $f_1,\ldots,f_k$ is a regular chain if:

1. whenever $i < j$, $\text{mvar}(f_i) \prec \text{mvar}(f_j)$ (therefore the chain is triangular);
2. $\text{init}(f_j)$ is invertible modulo the ideal ($\text{init}(f_j) : j < i$).

---

\(^{21}\)It actually only shows that this set of generators is not triangular. $\{x^2 - 1, (x + 1)(y - 1) + (x - 1)(y^2 - 1)\}$ is triangular [Ron11], but is not a Gröbner basis.
Proposition 46 Every equiprojectable variety corresponds to a zero-dimensional regular chain, and vice versa.

However, $V$ of Example 2 of section 3.3.7 can be written as $V = V_1 \cup V_2$ where $V_1 = \{(x = -1, y = 1)\}$ and $V_2 = \{(x = 1, y = \pm 1)\}$, each of which is equiprojectable. The corresponding regular chains are $T_1 = \{x + 1, y - 1\}$ and $T_2 = \{x - 1, y^2 - 1\}$.

Theorem 28 (Gianni–Kalkbrener (triangular variant)) Every zero-dimensional variety can be written as a union of disjoint equiprojectable varieties — an equiprojectable decomposition.

In fact, each solution description in Algorithm 10 is a description of an equiprojectable variety.

This theorem can be, and was, proved independently, and the decomposition into regular chains (the union of whose varieties is the original variety) can be computed directly. This gives us an alternative to algorithm 13: compute the regular chains corresponding to the equiprojectable decomposition, and solve each one separately \cite{Laz92}.

It appears that the triangular decomposition approach is more suitable to modular methods (chapter 4, especially section 4.6) than the Gröbner-base approach, but both aspects are areas of active research.

3.4.3 Positive Dimension

Here we consider the case of solution sets of positive dimension (over the algebraic closure, e.g. over the complexes). As in Theorem 28 the ultimate aim is to express a variety as a union (preferably a disjoint union) of “nicer” varieties, or other sets.

Definition 62 Quasi-algebraic System If $P$ and $Q$ are two (finite) sets of polynomials, we call the ordered pair $(P, Q)$ a quasi-algebraic system, and we write $Z(P, Q)$, the zeros of the quasi-algebraic system, for $V(P) \setminus V(\prod Q)$, with the convention that if $Q$ is empty, $\prod Q = 1$, so $V(Q) = \emptyset$.

$$Z(P, Q) = \{x \in K^n | (\forall p \in P \ p(x) = 0) \land (\forall q \in Q \ q(x) \neq 0)\}.$$  

We say that $(P, Q)$ is consistent if $Z(P, Q) \neq \emptyset$.

In Definition 44 we defined the variety of a set of polynomials, but we need some more concepts, all of which depend on having fixed an order of the variables.

Definition 63 If $T$ is a triangular system, we define the pseudo-remainder of $p$ by $T$ to be the pseudo-remainder of dividing $p$ by each $q_i \in T$ in turn (turn defined by decreasing order of $\text{mvar}(q_i)$), regarded as univariate polynomials in $\text{mvar}(q_i)$.

This is a generalization of Definition 31 (page 57).
3.4. NONLINEAR MULTIVARIATE EQUATIONS: RECURSIVE

**Definition 64** Let $S$ be a finite set of polynomials. The set of regular zeros of $S$, written $W(S)$, is $Z(S, \{\mathrm{init}(S)\}) = V(S) \setminus V(\{\mathrm{init}(S)\})$. For $(a_1, \ldots, a_n)$ to be in $W(S)$, where $S = \{p_1, \ldots, p_k\}$, we are insisting that all of the $p_i$ vanish at this point, but none of the $\mathrm{init}(p_i)$.

For Example 2 of section 3.3.7, the variety is $\{(x = -1, y = 1), (x = 1, y = \pm 1)\}$. If we take $y > x$, then the initial of the set of polynomials is $\lcm(1, x - 1, 1) = x - 1$, so only the zero with $x = -1, y = 1$ is regular. Conversely, if we take $x > y$, the initial is $y - 1$ and only the zero with $y = -1, x = 1$ is regular. This emphasises that $W$ depends on the variable ordering. It is also a property of the precise set $S$, not just the ideal $\langle S \rangle$.

In this case, $W(S)$ was in fact a variety (as always happens in dimension 0). In general, this is not guaranteed to happen: consider the (trivial) triangular system $S = \{(x - 1)y - x + 1\}$ with $y > x$. Since this polynomial is $(x-1)(y-1)$, $V(S)$ is the two lines $x = 1$ and $y = 1$. However, $W(S)$ is the line $y = 1$ except for the point $(1, 1)$. In fact this is the only direct description we can give, though we could say that $W(S)$ is “almost” the line $y = 1$. This “almost” is made precise as follows.

**Definition 65** If $W$ is any subset of $K^n$, the Zariski closure of $W$, written $\overline{W}$, is the smallest variety containing it:

$$\overline{W} = \bigcap \{V(F) \mid W \subseteq V(F)\},$$

which is itself a variety by Proposition [35].

In the example above, $\overline{W}(S) = V(y - 1)$.

### 3.4.3.1 An example

This example is from [AM99, p. 126]. Suppose we have, in two dimensions, a manipulator consisting of an arm of length 1 fixed at the origin, and with another arm, also of length 1, at its other end. We wish the far end of the manipulator to reach the point $(a, b)$ in the plane. Let $\theta_1$ be the angle that the first arm makes with the $x$ axis, and write $c_1 = \cos \theta_1$, $s_1 = \sin \theta_1$. Let $\theta_2$ be the angle that the second arm makes with the first. Then we have the following equations

$$c_1 + \cos(\theta_1 + \theta_2) = a \quad (3.43)$$

$$s_1 + \sin(\theta_1 + \theta_2) = b \quad (3.44)$$

$$s_1^2 + c_1^2 = 1 \quad (3.45)$$

$$s_2^2 + c_2^2 = 1, \quad (3.46)$$

where the last two equations state that the arms have length 1. We can apply the addition formulae for trigonometric functions to (3.43) and (3.44) to get

$$c_1 + c_1c_2 - s_1s_2 = a \quad (3.43),$$

---

Note that we use the same notation for algebraic closure and Zariski closure.

The author is grateful to Russell Bradford for explaining the geometric context.
\[ s_1 + c_1 s_2 + c_2 s_1 = b \]

Rewriting these equations as polynomials, assumed to be zero, and using the order

\[ c_2 > s_2 > c_1 > s_1 > b > a, \]

we get

\[ S = \{ c_2 c_1 - s_2 s_1 + c_1 - a, c_2 s_1 + s_2 c_1 + s_1 - b, c_1^2 + s_1^2 - 1, c_2 + s_2^2 - 1 \}, \]

which is not triangular since \( c_2 \) is the main variable of three different equations. AM99 implement the method of Laz91 to express \( V(S) \) as a (disjoint) union

\[ W(T_1) \cup W(T_2) \cup W(T_3), \]

where

\[ T_1 = \{ (b^2 + a^2)(4s_1^2 - 4bs_1 + b^2 + a^2) - 4a^2, 2ac_1 + 2bs_1 - b^2 - a^2, 2as_2 + 2(b^2 + a^2)s_1 - b^2 - a^2 b, 2c_2 - b^2 - a^2 + 2 \}, \]

\[ T_2 = \{ a, 2s_1 - b, 4c_1^2 + b^2 - 4, s_2 - bc_1, 2c_2 - b^2 + 2 \}, \]

\[ T_3 = \{ a, b, c_1^2 + s_1^2 - 1, s_2, c_2 + 1 \}. \]

3.4.3.2 Another Example

This was also considered as Example 5 (page 107).

Example 7 Let \( I = \langle ax^2 + x + y, bx + y \rangle \) with the order \( a \prec b \prec y \prec x \). The full triangular decomposition (obtained from Maple’s RegularChains package with option=lazard) is

\[ \{ bx + y, ay + b^2 - b, [x, y], [ax + 1, y, b], [x + y, a, b - 1] \}. \]

The first two components correspond to two two-dimensional surfaces (in fact the second one is a plane), whereas the second two correspond to one-dimensional solutions (hyperbola and straight line).

3.4.3.3 Regular Zeros and Saturated Ideals

Definition 66 If \( T \) is a triangular system, define the saturated ideal of \( T \) to be

\[
\text{sat}(T) = \{ p \in K[x_1, \ldots, x_n] | \exists n \in \mathbb{N} \text{ init}(T)^n p \in (T) \} = \{ p \in K[x_1, \ldots, x_n] | \text{prem}(p, T) = 0 \}.
\]

In other words it is the set of polynomials which can be reduced to zero by \( T \) after multiplying by enough of the initials of \( T \) so that division works. In terms of the more general concept of saturation \( I : S^\infty \) of an ideal (Bou61 p. 90), this is \( (T : (\text{init}(T))^\infty) \).
3.4. NONLINEAR MULTIVARIATE EQUATIONS: RECURSIVE

Theorem 29 ([ALM99, Theorem 2.1]) For any non-empty triangular set $T$,

$$W(T) = V(\text{sat}(T)).$$

In the example motivating Definition 65, sat$(S)$ is generated by $y - 1$, and indeed $W(S) = V(\text{sat}(S))$.

3.4.4 Conclusion

Whether we follow the Gianni–Kalkbrener approach directly (algorithm 13) or go via triangular sets, the solutions to a zero-dimensional family of polynomial equations can be expressed as a union of (equiprojectable) sets, each of which can be expressed as a generalised RootOf construct. For example, if we take the ideal

$$\{-3x - 6 + x^2 - y^2 + 2x^3 + x^4, -x^3 + x^2y + 2x - 2y, -6 + 2x^2 - 2y^2 + x^3 + y^3\},$$

its Gröbner basis (purely lexicographic, $y > x$) is

$$[6 - 3x^2 - 2x^3 + x^5, -x^3 + x^2y + 2x - 2y, 3x + 6 - x^2 + y^2 - 2x^3 - x^4].$$  (3.47)

There are seven irreducible monomials: 1, $x$, $x^2$, $x^3$, $x^4$, $y$ and $xy$. We know that $x$ satisfies a quintic, and $y$ then satisfies $(x^2 - 2)y - x^3 + 2x$. When $x^2 = 2$, this vanishes, so our quintic for $x$ decomposes into $(x^2 - 2)(x^3 - 3)$, and the whole solution reduces to

$$\langle x^2 - 2, y^2 - x \rangle \cup \langle x^3 - 3, y - x \rangle.$$  (3.48)

Unfortunately, we do not have a convenient syntax to express this other than via the language of ideals. We are also very liable to fall into the ‘too many solutions’ trap, as in equation (3.8): Maple resolves the first component (in radical form) to

$$\left\{ y = \sqrt[3]{2}, x = \sqrt[3]{2} \right\},$$  (3.49)

and the second one to

$$\left\{ y = \sqrt[3]{3}, x = \sqrt[3]{3} \right\},$$  (3.50)

both of which lose the connections between $x$ and $y$ ($x = y^2$ in the first case, $x = y$ in the second).

We are also dependent on the choice of order, since with $x > y$ the Gröbner basis is

$$[6 - 3y^4 - 2y^3 + y^7, 18 - 69y^2 - 9y^4 - 46y + 23y^5 - 2y^6 + 73x],$$  (3.51)

and no simplification comes to mind, short of factoring the degree seven polynomial in $y$, which of course is $(y^3 - 3)(y^4 - 2)$, and using the choice here to simplify the equation for $x$ into either $x - y$ or $x - y^2$. 
Maple’s RegularChains package, using the technology of section 3.4.1, produces essentially equation (3.48) for the order \( y > x \), and for \( x > y \) produces

\[
\begin{align*}
(2y^3 + 4y^2 + 2) x - 8 - 2y^2 - 2y^3 - 2y - 2y - 2y^4 - 2, \\
(5y + 3 + 4y^2) x - 12 - 5y^2 - 3y - 3y^4 - 3 - 3 + y^3
\end{align*}
\]

essentially the factored form of (3.51).

3.4.5 Regular Decomposition

TO BE COMPLETED

3.5 Equations and Inequalities

While it is possible to work in more general settings (real closed fields), we will restrict our attention to solving systems over \( \mathbb{R} \). Consider the two equations

\[
\begin{align*}
x^2 + y^2 &= 1 \quad \text{(3.52)} \\
x^2 + y^2 &= -1 \quad \text{(3.53)}
\end{align*}
\]

Over the complexes, there is little to choose between these two equations, both define a one-dimensional variety. Over \( \mathbb{R} \), the situation is very different: (3.52) still defines a one-dimensional variety (a circle), while (3.53) defines the empty set, even though we have only one equation in two variables.

**Definition 67 ([ARS+13])** We say that a complex (hyper)-surface \( V := \{(x_1, \ldots, x_n)|p_1(x_1, \ldots, x_n) = \cdots = p_k(x_1, \ldots, x_n) = 0 \} \) is real if every complex polynomial vanishing over \( V_\mathbb{R} := V \cap \mathbb{R}^n \) also vanishes over \( V \). Algebraic Geometers would say that \( V \) is the Zariski closure of \( V_\mathbb{R} \).

With the definition, (3.52) defines a real surface (in fact a curve), but (3.53) does not, since 1 vanishes over \( V \cap \mathbb{R}^n \) (which is the empty set), but not over \( V \).

The above example shows that we can essentially introduce the constraint \( x \geq 0 \) by adding a new variable \( y \) and the equation \( y^2 - x = 0 \). We can also introduce the constraint \( x \neq 0 \) by adding a new variable \( z \) and \( xz - 1 = 0 \) (essentially insisting that \( x \) be invertible). Hence \( x > 0 \) can be introduced. Having seen that \( \geq \) and \( > \) can creep in through the back door, we might as well admit them properly, and deal with the language of real closed fields, i.e. the language of fields (definition 13) augmented with the binary predicate \( > \) and the additional laws:

1. Precisely one of \( a = b \), \( a > b \) and \( b > a \) holds;
2. \( a > b \) and \( b > c \) imply \( a > c \);
3. \( a > b \) implies \( a + c > b + c \);
4. \( a > b \) and \( c > 0 \) imply \( ac > bc \).

This is the domain of real algebraic geometry, a lesser-known, but very important, variant of classical algebraic geometry. Suitable texts on the subject are [BPR06, BCR98]. However, we will reserve the word ‘algebraic’ to mean a set defined by equalities only, and reserve semi-algebraic for the case when inequalities (or inequations) are in use. More formally:

**Definition 68** An algebraic proposition is one built up from expressions of the form \( p_i(x_1, \ldots, x_n) = 0 \), where the \( p_i \) are polynomials with integer coefficients, by the logical connectives \( \neg \) (not), \( \land \) (and) and \( \lor \) (or). A semi-algebraic proposition is the same, except that the building blocks are expressions of the form \( p_i(x_1, \ldots, x_n) \sigma 0 \) where \( \sigma \) is one of \( =, \neq, >, \geq, <, \leq \).

This language is in fact redundant, since \( \neq, \geq, \leq \) can be replaced with the help of \( \neg \), but corresponds more closely to natural usage. The reader will also notice that it is not quite the language of real closed fields described above, since we do not allow division. This is partly for ease of subsequent development, but also allows us to sidestep “division by zero” questions, as raised in problem 1 of section 1.2.3. Hence the proposition \( \frac{p}{q} > 0 \) has to be translated as

\[
(q > 0 \land p > 0) \lor (q < 0 \land p < 0),
\]

which is not true when \( q = 0 \). If this is not what we mean, e.g. when \( p \) and \( q \) have a common factor, we need to say so.

**Open Problem 11 (Better treatment of division)** (3.54) is equivalent to \( pq > 0 \). However, \( \frac{p}{q} \geq 0 \) is not equivalent to \( pq \geq 0 \), but rather to \( pq \geq 0 \land q \neq 0 \). In general, the polynomial theorists tend to dismiss the problem as above. The logician tends to worry much more about the problem: consider [AP10, Appendix, lines 1–12] for an example of rational function manipulation.

### 3.5.1 Applications

It turns out that many of the problems one wishes to apply computer algebra to can be expressed in terms of real semi-algebraic geometry. This is not totally surprising, since after all, the “real world” is largely real in the sense of \( \mathbb{R} \). Furthermore, even if problems are posed purely in terms of equations, there may well be implicit inequalities as well. For example, it may be implicit that quantities are non-negative, or that concentrations in biochemistry lie in the range \([0, 1] \).

Robot motion planning … **TO BE COMPLETED**

It is also often important to prove unsatisfiability, i.e. that a semi-algebraic formula has no solutions. [Mon09] gives several examples, ranging from program

---

24Everyone agrees that an equation \( a = b \) is an equality. \( a > b \) and its variants are traditionally referred to as inequalities. This only leaves the less familiar inequation for \( a \neq b \). Some treatments ignore inequations, since \( "a \neq b" = "a > b \lor a < b" \), but in practice it is useful to regard inequations as first-class objects.
proving to biological systems. The program proving one is as follows. One wishes to prove that $I$ is an invariant (i.e. if it was true at the start, it is true at the end) of a program which moves from one state to another by a transition relation $\tau$. More formally, one wishes to prove that there do not exist two states $s, s'$ such that $s \in I$, $s' \notin I$, but $s \rightarrow_{\tau} s'$. Such a pair $(s, s')$ would be where “the program breaks down”, so a proof of unsatisfiability becomes a proof of program correctness. This places stress on the concept of ‘proof’ — “I can prove that there are no bad cases” is much better than “I couldn’t find any bad cases”.

### 3.5.2 Real Radical

. We recall the second part of Definition 45: the radical of an ideal $I$, denoted $\sqrt{I}$, is defined as

\[ \sqrt{I} = \{ p|m : p^m \in I \} . \]

**Definition 69** Let $A = \mathbb{R}[x_1, \ldots, x_k]$ (it is possible to be more general, and talk about real closed fields). The real radical of an ideal $I \subset A$, denoted $\sqrt[\mathbb{R}]{I}$, is defined as

\[ \sqrt[\mathbb{R}]{I} = \left\{ p|m, k \in \mathbb{N}, r_i \in \mathbb{R}^+, g_i \in A : p^{2m} + \sum_{i=1}^{k} r_i g_i^2 \in I \right\} . \]

### 3.5.3 Quantifier Elimination

A fundamental result of algebraic geometry is the following, which follows from the existence of resultants (section A.1).

**Theorem 30** A projection of an algebraic set is itself an algebraic set.

For example, the projection of the set defined by

\[ \left\{ (x - 1)^2 + (y - 1)^2 + (z - 1)^2 - 4, x^2 + y^2 + z^2 = 4 \right\} \]  

on the $x, y$-plane is the ellipse

\[ 8 x^2 + 8 y^2 - 7 - 12 x + 8 xy - 12 y. \]

We can regard equation (3.55) as defining the set

\[ \exists z \left( (x - 1)^2 + (y - 1)^2 + (z - 1)^2 = 4 \land x^2 + y^2 + z^2 = 4 \right) \]

and equation (3.56) as the quantifier-free equivalent

\[ 8 x^2 + 8 y^2 - 12 x + 8 xy - 12 y = 7. \]
Is the same true in real algebraic geometry? If \( P \) is a projection operator, and \( \Re \) denotes the real part, then clearly

\[
P(\Re(U) \cap \Re(V)) \subseteq \Re(P(U \cap V)). \tag{3.59}
\]

However, the following example shows that the inclusion can be strict. Consider

\[
\{ (x - 3)^2 + (y - 1)^2 + z^2 - 1, x^2 + y^2 + z^2 - 1 \}
\]

Its projection is \((10 - 6x - 2y)^2\), i.e. a straight line (with multiplicity 2). If we substitute in the equation for \( y \) in terms of \( x \), we get

\[
z = \sqrt{-10x^2 + 30x - 24},
\]

which is never real for real \( x \). In fact \( \Re(U) \cap \Re(V) = \emptyset \), as is obvious from the geometric interpretation of two spheres of radius 1 centred at \((0,0,0)\) and \((3,1,0)\). Hence the methods we used for (complex) algebraic geometry will not translate immediately to real algebraic geometry.

The example of \( y^2 - x \), whose projection is \( x \geq 0 \), shows that the projection of an algebraic set need not be an algebraic set, but might be a semi-algebraic set. Is even this guaranteed? What about the projection of a semi-algebraic set? In the language of quantified propositions, we are asking whether, when \( F \) is an algebraic or semi-algebraic proposition, the proposition

\[
\exists y_1 \ldots \exists y_m F(y_1, \ldots, y_m, x_1, \ldots, x_n) \tag{3.60}
\]

has a quantifier-free equivalent \( G(x_1, \ldots, x_n) \), where \( G \) is a semi-algebraic proposition. We can generalise this.

**Problem 3 (Quantifier Elimination)** *Given a quantified proposition*\(^{25}\)

\[
Q_1 y_1 \ldots Q_m y_m F(y_1, \ldots, y_m, x_1, \ldots, x_n), \tag{3.61}
\]

*where \( F \) is a semi-algebraic proposition and the \( Q_i \) are each either \( \exists \) or \( \forall \), does there exist a quantifier-free equivalent semi-algebraic proposition \( G(x_1, \ldots, x_n) \)? If so, can we compute it?*

The fact that there is a quantifier-free equivalent is known as the Tarski–Seidenberg Principle \([\text{Sei54, Tar51}]\). The first constructive answer to the question was given by Tarski \([\text{Tar51}]\), but the complexity of his solution was indescribable\(^{26}\). A better (but nevertheless doubly exponential) solution had to await the concept of cylindrical algebraic decomposition (CAD) \([\text{Coh75}]\) described in the next section.

**Notation 21** *Since \( \exists x \exists y \) is equivalent to \( \exists y \exists x \), and similarly for \( \forall \), we extend \( \exists \) and \( \forall \) to operate on blocks of variables, so that, if \( x = (x_1, \ldots, x_n) \), \( \exists x \) is equivalent to \( \exists x_1 \ldots \exists x_n \). If we use this notation to rewrite equation 3.61 with*  

\(^{25}\)Any proposition with quantified variables can be converted into one in this form, so-called prenex normal form — see any standard logic text.  
\(^{26}\)In the formal sense, that there was no elementary function which could describe it, i.e. no tower of exponentials of fixed height would suffice!
the fewest number of quantifiers, the quantifiers then have to alternate, so the formula is (where the $y_i$ are sets of variables)

$$\forall y_1 \exists y_2 \forall y_3 \ldots F(y_1, y_2, \ldots, x_1, \ldots, x_n),$$

or

$$\exists y_1 \forall y_2 \exists y_3 \ldots F(y_1, y_2, \ldots, x_1, \ldots, x_n).$$

In either form, the number of quantifiers is one more than the number of alternations.

### 3.5.4 Algebraic Decomposition

**Definition 70** An algebraic decomposition of $\mathbb{R}^n$ is an expression of $\mathbb{R}^n$ as the disjoint union of non-empty connected sets, known as cells, each defined as

$$p_1(x_1, \ldots, x_n)\sigma_0 \land \cdots \land p_m(x_1, \ldots, x_n)\sigma_0,$$

where the $\sigma$ are one of $=, >, <$.

Equation (3.64) is known as the defining formula of the cell $C$, and denoted $\text{Def}(C)$.

These should properly be called semi-algebraic decompositions, but this terminology has stuck. Note that (3.64) need not define a non-empty connected set — external information is required to show this. We should note that these definitions are a very restricted form of definition 68. Here are some examples.

1. $\mathbb{R}^1$ can be decomposed as $\{x < 0\} \cup \{x = 0\} \cup \{x > 0\}$.

2. $\mathbb{R}^1$ cannot be decomposed as $\{x^2 = 0\} \land \{x^2 > 0\}$, as the second set is not connected. Rather, we need the previous decomposition.

3. $\mathbb{R}^1$ cannot be decomposed as

$$\{(x^2 - 3)^2 - 2 = 0\} \land \{(x^2 - 3)^2 - 2 > 0\} \land \{(x^2 - 3)^2 - 2 < 0\},$$

as the sets are not connected. Rather, we need the decomposition (writing $(x^2 - 3)^2 - 2$ as $f$)

$$\{f > 0 \land x < -2\} \cup \{f = 0 \land x < -2\} \cup \{f < 0 \land x < 0\} \cup \{f = 0 \land x > -2 \land x < 0\} \cup \{f > 0 \land x > -2 \land x < 2\} \cup \{f = 0 \land x > 0 \land x < 2\} \cup \{f < 0 \land x > 0\} \cup \{f = 0 \land x > 2\} \cup \{f > 0 \land x > 2\}.$$

4. $\mathbb{R}^2$ can be decomposed as $\{(x^2 + y^2) < 0\} \cup \{(x^2 + y^2) = 0\} \cup \{(x^2 + y^2) > 0\}$.

5. $\mathbb{R}^2$ cannot be decomposed as $\{xy < 1\} \cup \{xy = 1\} \cup \{xy > 1\}$, as the last two sets are not connected. Rather, we need the more complicated

$$\{xy < 1\} \cup \{xy = 1 \land x > 0\} \cup \{xy = 1 \land x < 0\} \cup \{xy > 1 \land x < 0\} \cup \{xy > 1 \land x > 0\}.$$
6. $\mathbb{R}^2$ cannot be decomposed as $\{ f < 0 \} \cup \{ f = 0 \} \cup \{ f > 0 \}$, where $f = (x^2 + y^2 - 1) \left( (x-3)^2 + y^2 - 1 \right)$, as the first two sets are not connected. Rather, we need the more complicated

$$\{ f < 0 \land x < \frac{3}{2} \} \cup \{ f < 0 \land x > \frac{3}{2} \} \cup \{ f = 0 \land x < \frac{3}{2} \} \cup \{ f = 0 \land x > \frac{3}{2} \} \cup \{ f > 0 \}$$

The reader may complain that example 3 is overly complex: can’t we just write

$$\{ f > 0 \land x < -2 \} \cup \{ x = -\sqrt{3 + \sqrt{2}} \} \cup \{ f < 0 \land x < 0 \} \cup$$

$$\{ x = -\sqrt{3 - \sqrt{2}} \} \cup \{ f < 0 \land x > 0 \} \cup \{ x = \frac{1}{2} \}$$

In this case we could, but in general theorem 10 means that we cannot use RootOf constructs, and the question then is “which root of...”. In example 3 we chose to use numeric inequalities (and we were lucky that they could be chosen with integer end-points). It is also possible to describe the roots in terms of the signs of the derivatives of $f$, i.e.

$$\{ f > 0 \land x < -2 \} \cup \{ f = 0 \land f' < 0 \land f'' < 0 \} \cup \{ f < 0 \land x < 0 \} \cup$$

$$\{ f = 0 \land f' > 0 \land f'' < 0 \} \cup \{ f > 0 \land x > -2 \} \cup$$

$$\{ f = 0 \land f' < 0 \land f'' > 0 \} \cup \{ f < 0 \land x > 0 \} \cup$$

$$\{ f = 0 \land f' > 0 \land f'' > 0 \} \cup \{ f > 0 \land x > 2 \}$$

(as it happens, the sign of $f''$ is irrelevant here). This methodology can also be applied to the one-dimensional regions, e.g. the first can also be defined as $\{ f > 0 \land f' > 0 \land f'' < 0 \land f''' < 0 \}$. We may ask how we know that we have a decomposition, and where these extra constraints (such as $x > 0$ in example 5 or $x < \frac{3}{2}$ in example 6) come from. This will be addressed in the next section, but the brief answers are:

- we know something is a decomposition because we have constructed it that way;
- $x = 0$ came from the leading coefficient (with respect to $y$) of $xy - 1$, whereas $\frac{3}{2}$ in example 6 is a root of $\text{Disc}_y(f)$.

We stated in definition 70 that the cells must be non-empty. How do we know this? For the zero-dimensional cells $\{ f = 0 \land x > a \land x < b \}$, we can rely on the fact that if $f$ changes sign between $a$ and $b$, there must be at least one zero, and if $f'$ does not vanish there cannot be more than one: such an interval can be called an isolating interval. In general, we are interested in the following concept.

---

27 And equation (3.11) demonstrates that we probably wouldn’t want to even when we could!

28 Which will involve looking at $f''$ and so on.
Definition 71 A sampled algebraic decomposition of $\mathbb{R}^n$ is an algebraic decomposition together with, for each cell $C$, an explicit point $\text{Sample}(C)$ in that cell.

By ‘explicit point’ we mean a point each of whose coordinates is either a rational number, or a precise algebraic number: i.e. a defining polynomial together with an indication of which root is meant, an isolating interval, a sufficiently exact numerical approximation or a Thom’s Lemma list of signs of derivatives.

Definition 72 A decomposition $D$ of $\mathbb{R}^n$ is said to be sign-invariant for a polynomial $p(x_1, \ldots, x_n)$ if and only if, for each cell $C \in D$, precisely one of the following is true:

1. $\forall x \in C \quad p(x) > 0$;
2. $\forall x \in C \quad p(x) < 0$;
3. $\forall x \in C \quad p(x) = 0$;

It is sign-invariant for a set of polynomials if, and only if, for each polynomial, one of the above conditions is true for each cell.

It therefore follows that, for a sampled decomposition, the sign throughout the cell is that at the sample point. A stronger concept is provided by the following definition.

Definition 73 A decomposition $D$ of $\mathbb{R}^n$ is said to be order-invariant for a polynomial $p(x_1, \ldots, x_n)$ if and only if, for each cell $C \in D$, precisely one of the following is true:

1. $\forall x \in C \quad p(x) > 0$; and
2. $\forall x \in C \quad p(x) < 0$;
3. $\exists k \in \mathbb{N}$ such that:

(a) $\forall x \in C$ all derivatives of $p$ of order at most $k$ vanish at $x$; and

(b) $\forall x \in C$ there exists a derivative $\frac{\partial^{k+1} p}{\partial x_1 \cdots \partial x_{k+1}}$ which does not vanish at $x$.

- note that it may be different order $k+1$-derivatives at different points of $C$.

It is order-invariant for a set of polynomials if, and only if, for each polynomial, one of the above conditions is true for each cell. Order-invariance is a strictly stronger concept than sign-invariance.

---

29 Not necessarily irreducible, though it is normal to insist that it be square-free.

30 By this, we mean an approximation such that the root cannot be confused with any other, which generally means at least an approximation close enough that Newton’s iteration will converge to the indicated root. Maple’s $\text{RootOf}$ supports such a concept.
3.5. EQUATIONS AND INEQUALITIES

3.5.5 Cylindrical Algebraic Decomposition

The idea of Cylindrical Algebraic Decomposition is due to [Col75]. The presentation here is more general (see Observation 5), and largely unpublished.

**Notation 22** Let \( n > m \) be positive natural numbers, and let \( \mathbb{R}^n \) have coordinates \( x_1, \ldots, x_n \), with \( \mathbb{R}^m \) having coordinates \( x_1, \ldots, x_m \).

**Definition 74** An algebraic decomposition \( D \) of \( \mathbb{R}^n \) is said to be cylindrical over a decomposition \( D' \) of \( \mathbb{R}^m \) if the projection onto \( \mathbb{R}^m \) of every cell of \( D \) is a cell of \( D' \). The cells of \( D \) which project to \( C \in D' \) are said to form the cylinder over \( C \), denoted \( \text{Cyl}(C) \). For a sampled algebraic decomposition, we also insist that the sample point in \( C \) be the projection of the sample points of all the cells in the cylinder over \( C \). This definition is usually stated when \( m = n - 1 \), but the greater generality is theoretically worth having, even though we only currently know how to compute these when \( m = n - 1 \).

Cylindricity is by no means trivial.

**Example 8** Consider the decomposition of \( \mathbb{R}^2 = S_1 \cup S_2 \cup S_3 \) where

\[
S_1 = \{(x, y) \mid x^2 + y^2 - 1 > 0\}, \quad S_2 = \{(x, y) \mid x^2 + y^2 - 1 < 0\}, \quad S_3 = \{(x, y) \mid x^2 + y^2 - 1 = 0\}.
\]

This is an algebraic decomposition, and is sign-invariant for \( x^2 + y^2 - 1 \). However, it is not cylindrical over any decomposition of the \( x \)-axis \( \mathbb{R}^1 \). The projection of \( S_2 \) is \((-1, 1)\), so we need to decompose \( \mathbb{R}^1 \) as

\[
(-\infty, -1) \cup \{-1\} \cup (-1, 1) \cup \{1\} \cup (1, \infty).
\] (3.65)

\( S_3 \) projects onto \([-1, 1]\), which is the union of three sets in (3.65). We have to decompose \( S_3 \) into four sets:

\[
S_{3,1} = \{(-1,0)\}, \quad S_{3,2} = \{(1,0)\}, \\
S_{3,3} = \{(x,y) \mid x^2 + y^2 - 1 = 0\, \land\, y > 0\}, \\
S_{3,4} = \{(x,y) \mid x^2 + y^2 - 1 = 0\, \land\, y < 0\}.
\]

\( S_1 \) splits into eight sets, one above each of \((-\infty, -1)\) and \((1, \infty)\) and two above each of the other components of (3.65). It is obvious that this is the minimal refinement of the original decomposition to possess a cylindric decomposition. Furthermore in this case no linear transformation of the axes can reduce this.

If we wanted a sampled decomposition, we could choose \( x \)-coordinates of \(-2, -1, 0, 1\) and \( 2 \), and \( y \)-coordinates to match, from \( \{0, \pm1, \pm2\} \).

Cylindricity is fundamental to solving problem 3 via the following two propositions.
CHAPTER 3. POLYNOMIAL EQUATIONS

Proposition 47 Let
\[ \exists x_n \ldots \exists x_{m+1} P(x_1, \ldots, x_n) \]  
be an existentially quantified formula, \( D \) be a sampled algebraic decomposition of \( \mathbb{R}^n \) which is sign-invariant for all the polynomials occurring in \( P \), and \( D' \) be a sampled algebraic decomposition of \( \mathbb{R}^m \) such that \( D \) is cylindrical over \( D' \). Then a quantifier-free form of (3.66) is
\[ \bigvee_{C' \in D'} \text{Def}(C'). \]  
(3.67)

Proposition 48 Let
\[ \forall x_n \ldots \forall x_{m+1} P(x_1, \ldots, x_n) \]  
be a universally quantified formula, \( D \) be a sampled algebraic decomposition of \( \mathbb{R}^n \) which is sign-invariant for all the polynomials occurring in \( P \), and \( D' \) be a sampled algebraic decomposition of \( \mathbb{R}^m \) such that \( D \) is cylindrical over \( D' \). Then a quantifier-free form of (3.68) is
\[ \bigvee_{C' \in D'} \text{Def}(C'). \]  
(3.69)

These two propositions lead to a solution of problem 3.

Theorem 31 (Col75) Let \( x_0, \ldots, x_k \) be sets of variables, with \( x_i = (x_{i,1}, \ldots, x_{i,n_i}) \), and let \( N_i = \sum_{j=0}^{n_i} n_j \). Let \( P(x_0, \ldots, x_k) \) be a semi-algebraic proposition, \( D_i \) be an algebraic decomposition of \( \mathbb{R}^{N_i} \) such that each \( D_i \) is cylindric over \( D_{i-1} \) and \( D_k \) is sign-invariant for all the polynomials in \( P \). Then a quantifier-free form of
\[ Q_k x_k \ldots Q_1 x_1 P(x_0, \ldots, x_k) \]  
(where the \( Q_i \) are \( \forall \) or \( \exists \)) is
\[ \bigvee_{C' \in D_k \forall \exists C \in \text{Cyl}(C')} \text{Def}(C'), \]  
(3.70)

where by \( \forall \exists \) we mean that we are quantifying across the coordinates of \( \text{Sample}(C) \) according to the quantifiers in (3.70).

We can use the (sampled) cylindrical algebraic decomposition in example 8 to answer various questions.

Example 9 \( \forall y \quad x^2 + y^2 - 1 > 0 \). For the sampled cells \( \langle (-\infty, -1), (x = -2, y = 0) \rangle \) and \( \langle (1, \infty), (x = 2, y = 0) \rangle \), the proposition is true at the sample points, hence true everywhere in the cell. For all the other cells in (3.65), there is a sample point for which it is false (in fact, \( y = 0 \) always works). So the answer is \( (-\infty, -1) \cup (1, \infty) \).
3.5. EQUATIONS AND INEQUALITIES

Example 10  \( \exists y \ x^2 + y^2 - 1 > 0 \). For every cell in (3.65), there is a sample point above it for which the proposition is true, hence we deduce that the answer is (3.65), which can be simplified to true.

We should note (and this is both one of the strengths and weaknesses of this approach) that the same cylindrical algebraic decomposition can be used to answer all questions of this form with the same order of (blocks of) quantified variables, irrespective of what the quantifiers actually are.

Example 11  \(( \exists y \ x^2 + y^2 - 1 > 0 \) \( \land \) \( \exists y \ x^2 + y^2 - 1 < 0 \)) This formula is not directly amenable to this approach, since it is not in prenex form. In prenex form, it is \( \exists y_1 \exists y_2 ((x^2 + y_1^2 - 1 > 0) \land (x^2 + y_2^2 - 1 < 0)) \) and we need an analogous decomposition of \( \mathbb{R}^3 \) cylindric over \( \mathbb{R}^3 \). Fortunately, (3.65) suffices for our decomposition of \( \mathbb{R}^3 \), and the answer is \((-1 < x < 1)\), shown by the sample point \((x = 0, y_1 = 2, y_2 = 0)\), and by the fact that at other sample points of \( \mathbb{R}^3 \), we do not have \( y_1, y_2 \) satisfying the conditions.

We should note that it is not legitimate to reduce the formula to \( \exists y ((x^2 + y^2 - 1 > 0) \land (x^2 + y^2 - 1 < 0)) \), since that is trivially false. It is, however, legitimate to solve two separate problems, \( \exists y \ x^2 + y^2 - 1 > 0 \) (which is true, with \( y = 2 \)) and \( \exists y \ x^2 + y^2 - 1 < 0 \) (which is \(-1 < x < 1)\), and combine to get \(-1 < x < 1\).

Open Problem 12 (RAG Formulation 1) It is pretty obviously simpler to solve two two-dimensional problem than one three-dimensional one, but in general the correct translation of an arbitrary statement such as that of Example 11 into the most efficient problem formulation is a hard one.

However, it could be argued that all we have done is reduce problem 3 to the following one.

Problem 4 Given a quantified semi-algebraic proposition as in theorem 31 produce a sign-invariant decomposition \( D_k \) cylindrical over the appropriate \( D_i \) such that theorem 31 is applicable. Furthermore, since theorem 31 only talks about “a” quantifier-free form, we would like the simplest possible such \( D_k \) (see [11]).

There is no common term for such a decomposition: we will call it block-cylindrical.

Observation 5 Collins’ original presentation of Cylindrical Algebraic Decomposition insisted that every decomposition of \( \mathbb{R}^m \), generated by \( x_1, \ldots, x_m \) was cylindrical, in the sense of Definition 14 over every \( \mathbb{R}^k \) generated by \( x_1, \ldots, x_k \).

Currently, this is the only sort of decomposition we know how to compute, but, since the weaker form of block-cylindrical is all we need for quantifier elimination, we have given the more general definition.  

\footnote{Easier said than done. Above \( x = -1 \) we have nine cells: \( \{y_1 < 0, y_1 = 0, y_1 > 0\} \times \{y_2 < 0, y_2 = 0, y_2 > 0\} \), and the same for \( x = 1 \), whereas above \((-1, 1)\) we have 25, totalling 45.
Figure 3.7: Cylindrical Decomposition after Collins

\[ S_n \subset \mathbb{R}[x_1, \ldots, x_n] \quad \mathbb{R}^n \quad \mathbb{R}^n \text{ decomposed by } D_n \]
\[ S_{n-1} \subset \mathbb{R}[x_1, \ldots, x_{n-1}] \quad \mathbb{R}^{n-1} \quad \mathbb{R}^{n-1} \text{ decomposed by } D_{n-1} \]
\[ \ldots \]
\[ S_2 \subset \mathbb{R}[x_1, x_2] \quad \mathbb{R}^2 \quad \mathbb{R}^2 \text{ decomposed by } D_2 \]
\[ S_1 \subset \mathbb{R}[x_1] \quad \mathbb{R}^1 \quad \mathbb{R}^1 \text{ decomposed by } D_1. \]

3.5.6 Computing Algebraic Decompositions

Though many improvements have been made to it since, the basic strategy for computing algebraic decompositions is still generally\footnote{But \cite{CMXY09} have an alternative strategy based on triangular decompositions, as in section 3.4, which may well turn out to have advantages.} that due to Collins \cite{Col75}, and is to compute them cylindrically, as illustrated in the Figure 3.7. From the original proposition, we extract the set of polynomials \( S_n \). We then project this set into \( S_{n-1} \) in \( n-1 \) variables, and so on, until we have a set of univariates \( S_1 \). We then isolate, or otherwise describe, the roots of these polynomials, as described in problem \footnote{Notation 23} to produce a decomposition \( D_1 \) of \( \mathbb{R}^1 \), and then successively lift this to a decomposition \( D_2 \) of \( \mathbb{R}^2 \) and so on, each \( D_i \) being sign-invariant for \( S_i \) and cylindrical over \( D_{i-1} \).

Note that the projection from \( S_{i+1} \) to \( S_i \) must be such that a decomposition \( D_i \) sign-invariant for \( S_i \) can be lifted to a decomposition \( D_{i+1} \) sign-invariant for \( S_{i+1} \). Note also that the decomposition thus produced will be block-cylindrical for every possible blocking of the variables, since it is block-cylindric for the finest such.

Projection turns out to be a trickier problem than might be expected. One’s immediate thought is that one needs the discriminants (with respect to the variable being projected) of all the polynomials in \( S_{i+1} \), since this will give all the critical points. Then one sees that one needs the resultants of all pairs of such polynomials. Example \footnote{Example 5 (page 128)} shows that one might need leading coefficients. Then there are issues of what happens when leading coefficients vanish. This led Collins \cite{Col75} to consider the following projection operation \( \text{Proj}_C \) for a set \( \mathcal{A} \) of polynomials in \( x_1, \ldots, x_n \), where \( x_n \) is the variable being projected.

\textbf{Notation 23} Define the following sets, where we assume \( \mathcal{A} \) has \( k \) polynomials of degree \( d \).

\( B \) is the set of non-constant (with respect to \( x_n \)) iterated reducta (Notation\footnote{Notation 11})
of all elements of $A$: $\leq kd$ elements.

$\mathcal{L}$ is the set of all leading coefficients of $B$, which is the same as saying all the coefficients of non-constant terms of $A$: $\leq kd$ elements.

$S_1$ is the set of all principal subresultant coefficients (Definition 100) $\text{psc}_j(g, \frac{\partial g}{\partial x^n})$ for all $g \in B$: $\leq kd$ elements.

$S_2$ is the set of all principal subresultant coefficients $\text{psc}_j(g_1, g_2)$ for all $g_1, g_2 \in B : g_1 \neq g_2$: $\leq k^2d^3$ elements.

$\text{Proj}_C(A)$ is $\mathcal{L} \cup S_1 \cup S_2$: $O(k^2d^3)$ elements.

[Col75] essentially Theorem 5] showed that this projection operator is sufficient for Figure 3.7 to be valid.

TO BE COMPLETED

3.5.7 Describing Solutions

If describing the roots of a general polynomial in one variable was tricky, it is more so in two or more dimensions: not that the mathematics doesn’t exist, but rather that we are, in general, unused to dealing with it. Consider the example $f := y^3 - 7y^2 + 14y - x - 8$ in Figure 3.8. Above each value of $x$, $y$ is given by a univariate polynomial, whose roots can be described in terms of the signs of the derivatives, by Thom’s Lemma (Lemma 3). The top and bottom branches of the curve are defined by $f = 0$, $f' > 0$, but are distinguished by the signs of $f''$. Similarly, the two inflection points are defined by $f = f' = 0$, but are again distinguished by the signs of $f''$. However, the middle branch, $f = 0$, $f' < 0$, is unhelpfully split by $f'' = 0$ into two parts. We could (though doing so algorithmically has never, to the author’s knowledge, been solved) remove this distinction and just define the branch as $f = 0$, $f' < 0$. Since the Collins projection $\text{Proj}_C$ includes all the derivatives, it is capable of expressing the solutions in the Thom’s Lemma manner.

Conversely, and perhaps more naturally, we can try counting the branches, just as we might count the roots of a univariate polynomial (starting from $\infty$). This is shown in Figure 3.9. Here the top branch is split: it starts off being the third root of $f$, but as we cross the ($x$-value of the) inflection point of the other branch, it becomes the first root (passing momentarily through the pont at which it is the second root). Unlike the previous description, this split cannot be removed. Note also that “the first root” is not continuous, jumping from the bottom branch to the top branch at this $x$-value.
Figure 3.8: $y^3 - 7y^2 + 14y - x - 8$: Thom' Lemma
Figure 3.9: $y^3 - 7y^2 + 14y - x - 8$: indexing
CHAPTER 3. POLYNOMIAL EQUATIONS

Here is an edited version of the output from Maple on this problem.

\[
\begin{align*}
&\{ [RC, [[-4, -4], [-1, -1]]] : y < R(f_1, i_1) \} \\
&\{ [RC, [[-4, -4], [1/4, 1/2]]] : y = R(f_1, i_1) \} \\
&\{ [RC, [[-4, -4], [2, 2]]] : R(f_1, i_1) < y \} \\
&\{ [RC, [[-27/128, -135/128], [-1, -1]]] : y < R(f_2, i_1) \} \\
&\{ [RC, [[-27/128, -135/128], [1/2, 3/4]]] : y = R(f_2, i_1) \} \\
&\{ [RC, [[-27/128, -135/128], [13/8, 15/8]]] \land \left( \begin{array}{c} R(f_2, i_1) < y \\ y < R(f_2, i_2) \end{array} \right) \} : x = R(f_3, i_1) \\
&\{ [RC, [[-95/128, -95/128], [25/8, 25/8]]] : y = R(f_2, i_2) \} \\
&\{ [RC, [[-95/128, -95/128], [3/4, 5/4]]] : y = R(f_1, i_1) \} \\
&\{ [RC, [[-95/128, -95/128], [13/8, 13/8]]] \land \left( \begin{array}{c} R(f_1, i_1) < y \\ y < R(f_1, i_2) \end{array} \right) \} \} \\
&\{ [RC, [[5/8, 81/128], [11/8, 3/2]]] : y = R(f_2, i_1) \} \\
&\{ [RC, [[5/8, 81/128], [11/8, 11/4]]] \land \left( \begin{array}{c} R(f_2, i_1) < y \\ y < R(f_2, i_2) \end{array} \right) \} : x = R(f_3, i_2) \\
&\{ [RC, [[5/8, 81/128], [4, 4/3]]] : y = R(f_2, i_2) \} \\
&\{ [RC, [[5/8, 81/128], [6, 6]]] : R(f_2, i_2) < y \} \\
&\{ [RC, [[2, 2], [3, 3]]] : y < R(f_1, i_1) \} \\
&\{ [RC, [[2, 2], [17/4, 9/2]]] : y = R(f_1, i_1) \} \\
&\{ [RC, [[2, 2], [6, 6]]] : R(f_1, i_1) < y \} \\
\end{align*}
\]

This splits \( \mathbb{R}^2 \) into 23 cells, which is the minimum for any cylindrical decomposition with \( y \) projected first. In the other direction, we just have three cells: “below the curve”, “on the curve” and “above the curve”. This sort of variation depending on projection order is not unusual: \textsuperscript{BD07} have examples in \( n \) variables where the number of cells is constant for some projections, and \( 2^{2^\Theta(n)} \) for others. The variation is not universal though: there are also examples with \( 2^{2^\Theta(n)} \) cells for all projection orders.

The drawback of this representation is that “the \( i \)th branch of” is not semi-algebraic in the sense of Definition \textsuperscript{BD07} and we need to convert to the other in order to stay within the semi-algebraic language.

\textsuperscript{34}Common polynomials have been pulled out, and various abbreviations, such as \( R(f, i_1) \) for \( \text{RootOf}(f, index = \text{real}_1) \), have been used.
3.5.8 Complexity

Let us suppose that there are $s$ polynomials involved in the input formula (3.61), of maximal degree $d$. Then such a cylindrical algebraic decomposition can be computed in time $O\left((sd)^{O(k)}\right)$.

There are examples [BD07, DH88], which shows that this behaviour is best-possible, indeed the projection onto $\mathbb{R}^1$ might have a number of components doubly-exponential in $k$. This is true even for [BD07], where the polynomials are “only” linear.

While this behaviour is intrinsic to cylindrical algebraic decomposition, it is not necessarily intrinsic to quantifier elimination as such. If $a$ is the number of alternations of quantifiers (Notation 21) in the problem (so $a < k$), then there are algorithms [Bas99, for example] whose behaviour is singly-exponential in $k$ but doubly-exponential in $a$; typically $O\left((sd)^{O(k^2)}\right)$. The construction of [BD07, DH88] makes extensive use of $\lor$ symbols, and this is essential, as [RESW14] have shown that, with no $\lor$ symbols in (3.61), the linear problem can be solved in polynomial time, essentially by a series of reductions to linear programming, which can be solved in polynomial time [Kha79, Kar84].

One particular special case is that of no alternations. Hence, using the fact that $\exists x\left(P(x) \lor Q(x)\right)$ is equivalent to $\left(\exists x P(x)\right) \lor \left(\exists x Q(x)\right)$, an existential problem is equivalent to a set of problems of the form

$$
\exists x \left(\bigwedge_{f_i(x) \geq 0} f_i(x) \geq 0 \right) \land \left(\bigwedge_{g_i(x) = 0} g_i(x) = 0 \right) \land \left(\bigwedge_{h_i(x) \neq 0} h_i(x) \neq 0 \right).
$$

(3.72)

This is generally referred to as the existential theory of the reals. Since the truth of a universal problem is equivalent to the falsity of an existential problem ($\forall x P(x) \iff \neg \exists x \neg P(x)$), this is all we need to consider.

Given a problem (3.72), cylindrical algebraic decomposition will yield such an $x$, if one exists, and failure to yield one is a proof that no such $x$ exists. However, this is a somewhat unsatisfactory state of affairs in practice, since, computationally, we are relying not just on the correctness of the theory of cylindrical algebraic decomposition, but also on the absence of bugs in the implementation.

An alternative is provided by the Positivstellensatz approach [Ste74].

**Theorem 32 ([PQR09, Theorem 3])** The set of solutions to (3.72) is empty if, and only if, there are:

$s \in \text{con}(F)$ where $\text{con}(F)$ is the smallest set generated by $F$ and the set of squares of all elements of $\mathbb{R}[x]$ which is closed under multiplication and addition;

$g \in \langle G \rangle$ the ideal generated by $G$;

34There may be singly-exponential blow-up here as we convert into disjunctive normal form, but this is small compared to the other exponential issues in play!
$m \in \text{mon}(H)$ where $\text{mon}(H)$ is the set of all products (including $1$ = the empty product) of elements of $H$;

such that $s + g + m^2 = 0$. Furthermore, there is an algorithm to find such $s$, $g$ and $m$ (if they exist) in $\mathbb{Q}[x]$ provided $F$, $G$ and $H \subset \mathbb{Q}[x]$.

**Partial Proof.** If $s + g + m^2 = 0$ but $x$ is a solution to (3.72), then $s(x) + g(x) + m(x)^2$ is of the form “non-negative + zero + strictly positive”, so cannot be zero.

We can think of $(s, g, m)$ as a witness to the emptiness of the set of solutions to (3.72). Again, failure to find such an $(s, g, m)$ is a proof of the existence of solutions provided we trust the correctness of Theorem 32 and the correctness of the implementation.

### 3.5.9 Further Observations

1. The methodology outlined in figure 3.7, and indeed that of 

   has the pragmatic drawback that the decomposition computed, as well as solving the given problem (3.61), solves all other problems of the same form with the same polynomials and the variables in the same order. For example, a decomposition which allows us to write down a quantifier-free equivalent of

   $\forall x_4 \exists x_3 p(x_1, x_2, x_3, x_4) > 0 \land q(x_1, x_2, x_3, x_4) < 0$  \hspace{1cm} (3.73)

   will also solve

   $\forall x_4 \exists x_3 p(x_1, x_2, x_3, x_4) > 0 \land q(x_1, x_2, x_3, x_4) < 0$  \hspace{1cm} (3.74)

   and even

   $\exists x_4 \exists x_3 \forall x_2 p(x_1, x_2, x_3, x_4) < 0 \land q(x_1, x_2, x_3, x_4) \geq 0$  \hspace{1cm} (3.75)

   The process of Partial Cylindrical Algebraic Decomposition [CH91] can make the lifting process (right hand side $\uparrow$ in Figure 3.7) more efficient, but still doesn’t take full account of the structure of the incoming quantified formula.

2. If $F$ is of the form $p(y_1, \ldots, y_n, x_1, \ldots, x_n) \land \hat{\phi}(y_1, \ldots, y_m, x_1, \ldots, x_n)$, or can be transformed into this form, then, as observed in [Co98] we are only concerned in Theorem 31 with those cells on which $p$ is zero, and the polynomials in $\hat{\phi}$ do not need to be sign-invariant elsewhere. This was formalised in [McC99], where $p$ was described as an *equational constraint*.

   This idea has been further generalised in [BDE+13] to cases where every part of $F$ has an equational constraint, but not necessarily the same one.

3. It is tempting to think that “the problem is the data structure”, and maybe if used Straight-Line Programs (page 43), or some other data structure, we could do better. However, this hope is destroyed by [CGH+03] who state:
In this paper we are going to argue that the non-polynomial complexity character of the known symbolic geometric elimination procedures is not a special feature of a particular data structure (like the dense, sparse or arithmetic circuit encoding of polynomials), but rather a consequence of the information encoded by the respective data structure . . . and their Theorem 4 (which is too complicated to state here in detail), says that any universal elimination procedure for the theory of algebraically closed fields of characteristic zero must have non-polynomial complexity.

3.6 Conclusions

1. The RootOf construct is inevitable (theorem 10), so should be used, as described in footnote 3 (page 73). Such a notation can avoid the “too many solutions” trap — see equations (3.49) and (3.50). We should find a way of extending it to situations such as equation (3.48).

2. While matrix inversion is a valuable concept, it should generally be avoided in practice.

3. Real algebraic geometry is not simply “algebraic geometry writ real”: it has different problems and needs different techniques.

Open Problem 13 (Not all CADs are computed) So the set is (say)

\[(x = -1 & y \in (-1, 1)) \cup (-1 < x < 1 & y = 0) \cup (x = 1 & y \in (-1, 1))\].

Then indeed \(x \neq -1\): one cell \(x = -1\): five cells \((y \neq -1, -1 \leq y \leq 1, y - 1)\) \(-1 \leq x \leq 1\): three cells \(x = 1\): five cells \(x \neq 1\): one cell is a cylindrical algebraic decomposition, but not one I know how to compute. Indeed, I am not sure I even know how to state it formally to QEPCAD or Maple. Nice example of the difference between arbitrary CADs and ones we compute!
Chapter 4

Modular Methods

In chapter 2, describing the subresultant method of computing greatest common divisors, we said the following.

This algorithm is the best method known for calculating the g.c.d., of all those based on Euclid’s algorithm applied to polynomials with integer coefficients. In chapter 4 we shall see that if we go beyond these limits, it is possible to find better algorithms for this calculation.

Now is the time to fulfil that promise, which we do by describing the historically-first “advanced” algorithm, first with a simple example (section 4.1), and then its greatest success, g.c.d. calculation, first in one variable (section 4.2), then in two (section 4.3) and several (section 4.4) variables. We will then look at other applications (section 4.5), and finally at Gröbner bases (section 4.6).

The basic idea behind these algorithms is shown in Figure 4.1: instead of doing a calculation in some (large) domain \(R\), we do it in several smaller domains \(R_i\), possibly discard some of these, piece the result together to \(R_1' \ldots k\), and finally regard this combination as over a subset of \(R_1, \ldots, R_k\).

![Figure 4.1: Diagrammatic illustration of Modular Algorithms](image)

\(R_1' \ldots k\) indicates that some of the \(R_i\) may have been rejected by the compatibility checks, so the combination is over a subset of \(R_1, \ldots, R_k\).
as being in $R$ and check that it is indeed the right result. The key questions are then the following.

1. Are there domains $R_i$ for which the behaviour of the computation in $R_i$ is sufficiently close to that in $R$ for us to make deductions about the computation in $R$ from that in $R_i$? Such $R_i$ will generally be called “good”.

2. Can we tell, either immediately or with hindsight, whether an $R_i$ is “good”? It will often turn out that we can’t, but may be able to say that, given $R_i$ and $R_j$, one of them is definitely “bad”, and preferably which.

3. How many reductions should we take? In practice we will only count “good” reductions, so this question is bound up with the previous one.

4. How do we combine the results from the various $R_i$? The answer to this will often turn out to be a variant of the Chinese Remainder Theorem.

5. How do we check the result? Can we be absolutely certain that this result is in fact the answer to the original question? In category speak, does Figure 4.1 commute?

A common choice for the $R_i$ is given by the following.

**Notation 24** Let $n$ be a positive number. By $\mathbb{Z}_n$ we mean the integers considered modulo $n$.

**Proposition 49** $\mathbb{Z}_n$ is a commutative ring with a multiplicative identity (the number 1 itself).

**Example 12** Note that we can’t say that $\mathbb{Z}_n$ is an integral domain (Definition 9), since, for example, in $\mathbb{Z}_6$ $2 \times 3 = 0$, even though neither 2 nor 3 are zero in $\mathbb{Z}_6$.

**Proposition 50** Let $p$ be a prime number. Then $\mathbb{Z}_p$ is a field.

**Proof.** Since $p$ is a prime, the problem of Example 12 can’t happen, and $\mathbb{Z}_p$ is certainly an integral domain. Let $n \in \mathbb{Z}_p$ be nonzero, and apply the Extended Euclidean Algorithm (in $\mathbb{Z}$) to the integers $n$ and $p$. Since $\gcd(n,p) = 1$, we find $a, b$ such that $an + bp = 1$. Then $an \equiv 1 \pmod{p}$, so $a = n^{-1}$ and hence inverses exist.

### 4.1 Matrices: a Simple Example

Suppose we wish to calculate the determinant, or possibly the inverse, $D$ of an $n \times n$ matrix $M$, and for simplicity we will assume that $n$ is even. If the size of the entries (degree for polynomials, or number of bits/words for integers) is $s$, then the determinant will have size at most $\binom{n}{1} ns$, since it is the sum of $n!$ products of $n$ entries from the matrix.

\footnote{Not quite for integers, since $a + b$ can be bigger than either $a$ or $b$, but we’ll ignore this complication, which does not affect the general point.}
4.1. MATRICES: A SIMPLE EXAMPLE

4.1.1 Matrices with integer coefficients: Determinants

If we use the fraction-free methods of Theorem 15, then, just before clearing column \( k \), the entries should have size roughly \( \frac{7}{12} ks \), and at the end, the last entry, which is the determinant, is (roughly) bounded by \( ns \). But consider the half-way state, after we have eliminated \( \frac{n}{2} \) columns. The entries are:

row 1 \( n \) entries of size \( s \): total \( ns \);
row 2 \( n - 1 \) entries of size \( 2s \): total \( 2(n - 1)s \);
row 3 \( n - 2 \) entries of size \( 3s \): total \( 3(n - 3)s \);
...
row \( \frac{n}{2} \) \( \frac{n}{2} \) entries of size \( \left(\frac{n}{2}\right)s \): total \( \left(\frac{n}{2}\right)^2 s \);

Total \( \sum_{i=1}^{\frac{n}{2}} i(n + 1 - i)s = \left(\frac{n^3}{12} + \frac{n^2}{4} + \frac{n}{6}\right)s \).

next \( n/2 \) rows \( n/2 \) elements of size \( \left(\frac{n}{2}\right)s \) in each row, totalling \( \frac{n^3}{8} s \)

Grand total \( \frac{5}{24}n^3 s + O(n^2 s) \).

Since the final result has size \( ns \), we are storing, and manipulating, \( \frac{5}{24}n^2 \) times as much data as are needed — a quantifiable case of intermediate expression swell. Can we do better?

Let the \( p_i \) be a family of distinct primes \( \leq 3 \), chosen so that arithmetic modulo \( p \) is fast, typically 31-bit or 63-bit primes. Since \( D \) is the sum of products of entries of the matrix, we can evaluate \( D \) (mod \( p \)) (which we will write as \( D_p \)) either by computing \( D \), and then reducing it modulo \( p \), or by computing (modulo \( p \)) the determinant of the matrix \( M_p \); in symbols

\[
\det(M) = \det(M_p). \tag{4.1}
\]

The latter takes \( O(n^3) \) operations, by classical arithmetic, to which we should add \( O(n^2 s) \) for the initial reduction modulo \( p \).

The Hadamard bounds (Propositions 81 and 82; see Chapter 10) for an analysis of the extent to which they are over-estimates — roughly 0.22\( n \) decimal digits) will give us a number \( D_0 \) such that \( |D| \leq D_0, \log_2 D_0 \leq \frac{7}{8} s(\log_2 n) \). If we know \( D_p \), for enough primes such that \( N := \prod p_i > 2D_0 \), we can use the Chinese Remainder Theorem (Theorem 37 and Algorithm 46) to deduce \( D_N \). If we choose it in the range \( \frac{N}{2} < D_N < \frac{N}{2} \), then \( D_N = D \), and we are done.

The number of primes needed is \( O(\log D_0) = O(ns \log n) \). Hence the cost of all the determinants is \( O((n^3 + n^2 s)ns \log n) = O(n^3 s \log n + n^3 s^2 \log n) \). The

\footnote{Again, we need “roughly” in the integer case, since there is again the question of carries, as in note 4.}

\footnote{For this example, we do not actually need them to be primes, merely relatively prime, but the easiest way to ensure this is for them to be primes.}
cost of the Chinese Remainder Algorithm is, by Proposition 92, $O(\log^2 D_0) = O(n^2 s^2 \log^2 n)$, which is dominated by the second term in the determinant cost. For large $n$, the first term dominates, but for large $s$, the second term might dominate. Hence we have proved the following, slightly clumsy, statement.

Proposition 51 The cost of computing the determinant of an $n \times n$ matrix with integer entries bounded by $2^s$ is $O(\max(n^4 s \log n, n^3 s^2 \log n))$. If we attempt to simplify this, we can write $O(n^4 s^2 \log n)$, which doesn’t look much different from $(3.25)$’s $O(n^5 s^2 \log^2 n)$, and this is true in terms of worst-case complexity with respect to each of $n$ and $s$ considered separately. But if, for example, $s$ and $n$ are both $k$, then $(3.25)$ is $O(k^7 \log^2 k)$ and Proposition 51 is $O(k^5 \log k)$. An even better method is mentioned in Section 5.9.3.

4.1.2 Matrices with polynomial coefficients: Determinants

Assume our matrix entries are polynomials in $x$, of degree at most $d$, over a (sufficiently large) ring $R$ which supports the polynomial Chinese Remainder Theorem (Appendix A.4) — in practice $\mathbb{Q}$ or $\mathbb{Q}(y_1, \ldots, y_m)$. One method is the fraction-free computations of Theorem 15, then the same analysis as at the start of the previous section shows that we have intermediate expression swell. Can we do better?

Suppose we know that the degree (in $x$) of the determinant is less than $N$, and let $v_1, \ldots, v_N$ be $N$ distinct values in $R$. Then, using the notation $Z_{x - v}$ to mean “$Z$, but replacing $x$ by the value $v$” (which for polynomial $Z$ is the same as the remainder on dividing $Z$ by $x - v$), we have

$$\det(M)_{x - v} = \det(M_{x - v}).$$

(4.2)

Proposition 52 If $M$ is a $n \times n$ matrix whose entries are polynomials in $x$ of degree at most $d$, then the degree of $\det(M)$ is at most $nd$.

If we assume no expression growth in the elements of $R$ (unrealistic, but simplifying), the cost of computing $M_{x - v}$ is $O(n^2 d)$ operations in $K$, and the cost of evaluating the determinant is $O(n^3)$. The cost of the Chinese Remainder Algorithm is, by Proposition 93, $O(N^2) = O((nd + 1)^2)$.

Proposition 53 The cost of computing the determinant of an $n \times n$ matrix with polynomial entries of degree bounded by $d$ is $O(\max(n^4 d, n^3 d^2))$ coefficient operations.

In practice, if we had polynomial entries with integer coefficients, we would reduce the polynomial determinant calculation to many integer ones, and do these by the method of section 4.1.1. The analysis becomes rather tedious, but it is still possible to show that the complexity is always better than fraction-free methods.
Open Problem 14 (Matrix Determinant costs) In Open Problem 5, we saw that the obvious calculation of costs did not seem to have been borne out by (admittedly old) experiments. What is the practical validity of these calculations?

4.1.3 Conclusion: Determinants

Hence we have simple answer to the questions on page 144.

1. Are there good reductions from R?: yes, every reduction.

2. How can we tell if \( R_i \) is good? — always.

3. How many reductions should we take? This depends on bounding the size of \( \det(M) \). In the polynomial case, this is easy from the polynomial form of the determinant: Proposition 52. In the integer case, the fact that we are adding \( n! \) products makes the estimate more subtle: see Proposition 81.

4. How do we combine the results — one of Algorithms 46 (integer) or 47 (polynomial).

5. How do we check the result: irrelevant.

4.1.4 Linear Equations with integer coefficients

Now suppose that we wish to actually solve a linear system \( M.x = a \) with coefficients in \( \mathbb{Z} \), rather than simply compute the determinant (which we will assume is nonzero). As before, one option is the fraction-free method (Corollary 4) followed by back-substitution, and it can be shown that the cost of the back-substitution is asymptotically the same as that of the fraction-free triangularization.

Let \( p \) be a prime not dividing \( \det(M) \). We know (3.14) that \( x = M^{-1}.a \), and hence \( x_p = (M^{-1}.a)_p = (M^{-1})_p.a_p = M_p^{-1}.a_p \). If we knew the \( x_p \) for enough \( p \), we ought to be able to recover \( x \). However, the entries of \( x \) are rational numbers, not integers. There is a general answer to this problem in section 4.5.2.3 but in fact we can do better here. We know that the denominators of \( x \) all divide \( \det(M) \), so we shall solve the related problem

\[
y = (\det(M))M^{-1}.a, \tag{4.3}
\]

whose solution \( y \) is a vector of integers, again bounded by the Hadamard bounds (Propositions 81 and 82).

Algorithm 16 (Modular Linear Equations)
Input: \( M \) an \( n \times n \) non-singular matrix, \( a \) an \( n \)-vector over \( \mathbb{Z} \)
Output: \( x \) an \( n \)-vector over \( \mathbb{Q} \) with \( M.x = a \)
1. for enough \((N)\) primes \(p_i\)  
2. reduce \(M \mod p_i\) to get \(M_i\), and \(a\) to get \(a_i\)  
3. triangularize \(M_i||a_i\)  
4. if \(M_i\) is non-singular  
5. Backsubstitute to solve \(y_i = (\det(M_i))M_i^{-1}.a_i\).  
6. Reconstruct \(\det(M)\) from the \(\det(M_i)\).  
7. Reconstruct \(y\) from the \(y_i\).  
8. Return \(y/\det(M)\) (after cancellation).

The cost of steps 2 and 3 are \(O(n^2s)\) and \(O(n^3)\), as in section 4.1.1. Step 5 costs \(O(n^2)\) operations. Step 6 costs, by Proposition 92 \(O(\log^2(\det(M))) = O(n^2s^2 \log^2 n)\) operations, and step 7 costs \(n\) times as much. Step 8 as the same order of complexity as step 7, as we are computing \(n\) gcds of numbers of size bounded by the Hadamard bounds. Hence the total cost is

\[
O \left( N(n^2s + n^3) + n(n^2s^2 \log^2 n) \right) 
\]

operations. The number of primes we actually need is given by the Hadamard bound\(^4\), i.e. \(ns \log n\), and the number of primes that we have to discard is (at most) the same, as these must divide the determinant. The total cost is therefore

\[
O(n^3s^2 \log n + n^4s \log n + n^3s^2 \log^2 n) = O(n^4s \log n + n^3s^2 \log^2 n) \quad (4.5)
\]

(where we have annotated the causes of the various summands): almost the same as Proposition 51 and the extra factor of \(\log n\) multiplying \(n^3s^2\) is caused by the fact that the Chinese Remainder Theorem is no longer negligible, as we have to reconstruct \(n\) numbers.

4.1.5 Linear Equations with polynomial coefficients
This is sufficiently similar that we will skip the details.

4.1.6 Conclusion: Linear Equations
Hence we have simple answer to the questions on page 144.

1. Are there good reductions from \(R\)? yes, all reductions except those that annihilate the determinant — a finite number.

2. How can we tell if \(R_i\) is good? — if the determinant is nonzero.

\(^4\) Though it doesn’t change the asymptotics, these logarithms are \(\log_B\), where \(B\) is the size of our working primes. Hence if \(B = 2^{31}\), we have \(\log_{2^{31}}\) rather than \(\log_e\), saving a factor of 21.
3. How many reductions should we take? This depends on bounding the size of \( \text{det}(M) \). In the polynomial case, this is easy from the polynomial form of the determinant: Proposition 52. In the integer case, the fact that we are adding \( n! \) products makes the estimate more subtle: see Proposition 81.

4. How do we combine the results — one of Algorithms 46 (integer) or 47 (polynomial).

5. How do we check the result: irrelevant.

### 4.1.7 Matrix Inverses

The same techniques as linear equations apply: indeed we could (though probably should not) compute the columns of the inverse as the solutions of \( M \mathbf{x} = \mathbf{e}_i \), where \( \mathbf{e}_i \) has a 1 in the \( i \)th position, and zero elsewhere.

### 4.2 Gcd in one variable

Let us consider Brown’s example (from which page 57 was modified):

\[
A(x) = x^8 + x^6 - 3x^4 - 3x^3 + 8x^2 + 2x - 5; \quad (4.6)
\]
\[
B(x) = 3x^6 + 5x^4 - 4x^2 - 9x + 21. \quad (4.7)
\]

Let us suppose that these two polynomials have a common factor, that is a polynomial \( P \) (of non-zero degree) which divides \( A \) and \( B \). Then there is a polynomial \( Q \) such that \( A = PQ \). This equation still holds if we take each coefficient as an integer modulo 5. If we write \( P_5 \) to signify the polynomial \( P \) considered as a polynomial with coefficients modulo 5, this equation implies that \( P_5 \) divides \( A_5 \) and \( B_5 \), and therefore it is a common factor of \( A_5 \) and \( B_5 \). But calculating the g.c.d. of \( A_5 \) and \( B_5 \) is fairly easy:

\[
A_5(x) = x^8 + x^6 + 2x^4 + 3x^2 + 2x;
\]
\[
B_5(x) = 3x^6 + x^2 + x + 1;
\]
\[
C_5(x) = \text{remainder}(A_5(x), B_5(x)) = A_5(x) + 3(x^2 + 1)B_5(x) = 4x^2 + 3;
\]
\[
D_5(x) = \text{remainder}(B_5(x), C_5(x)) = B_5(x) + (x^4 + 4x^2 + 3)C_5(x) = x;
\]
\[
E_5(x) = \text{remainder}(C_5(x), D_5(x)) = C_5(x) + xD_5(x) = 3.
\]

Thus \( A_5 \) and \( B_5 \) are relatively prime, which implies that \( P_5 = 1 \). As the leading coefficient of \( P \) has to be one, we deduce that \( P = 1 \).

The concept of modular methods is inspired by this calculation, where there is no possibility of intermediate expression swell, for the integers modulo 5 are bounded (by 4). Obviously, there is no need to use the integers modulo 5: any prime number \( p \) will suffice (we chose 5 because the calculation does not work

\footnote{Note that we cannot deduce that \( P_5 = \gcd(A_5, B_5) \): a counter-example is \( A = x - 3, B = x + 2 \), where \( P = 1 \), but \( A_5 = B_5 = x + 2 \), and so \( \gcd(A_5, B_5) = x + 2 \), whereas \( P_5 = 1 \).}
modulo 2, for reasons to be described later, and 3 divides one of the leading coefficients). In this example, the result was that the polynomials are relatively prime. This raises several questions about generalising this calculation to an algorithm capable of calculating the g.c.d. of any pair of polynomials:

1. how do we calculate a non-trivial g.c.d.?

2. what do we do if the modular g.c.d. is not the modular image of the g.c.d. (as in the example in the footnote [5])?

3. how much does this method cost?

4.2.1 Bounds on divisors

Before we can answer these questions, we have to be able to bound the coefficients of the g.c.d. of two polynomials.

**Theorem 33 (Landau–Mignotte Inequality)**\(^{[Lan05, Mig74, Mig82]}\) Let \(Q = \sum_{i=0}^{\alpha} b_i x^i\) be a divisor of the polynomial \(P = \sum_{i=0}^{p} a_i x^i\) (where \(a_i\) and \(b_i\) are integers). Then

\[
\max_{i=0}^{q} |b_i| \leq \sum_{i=0}^{q} |b_i| \leq 2^q \frac{|b_q|}{a_p} \sqrt{\sum_{i=0}^{p} a_i^2}.
\]

These results are corollaries of statements in Appendix A.2.2.

If we regard \(P\) as known and \(Q\) as unknown, this formulation does not quite tell us about the unknowns in terms of the knowns, since there is some dependence on \(Q\) on the right, but we can use a weaker form:

\[
\sum_{i=0}^{q} |b_i| \leq 2^p \sqrt{\sum_{i=0}^{p} a_i^2}.
\]

When it comes to greatest common divisors, we have the following result.

**Corollary 9** Every coefficient of the g.c.d. of \(A = \sum_{i=0}^{\alpha} a_i x^i\) and \(B = \sum_{i=0}^{\beta} b_i x^i\) (with \(a_i\) and \(b_i\) integers) is bounded by

\[
2^{\min(\alpha, \beta)} \gcd(a_\alpha, b_\beta) \min \left( \frac{1}{|a_\alpha|}, \frac{1}{|b_\beta|} \right) \sqrt{\frac{1}{\sum_{i=0}^{\alpha} a_i^2}, \frac{1}{\sum_{i=0}^{\beta} b_i^2}}.
\]

**Proof.** The g.c.d. is a factor of \(A\) and of \(B\), the degree of which is, at most, the minimum of the degrees of the two polynomials. Moreover, the leading coefficient of the g.c.d. has to divide the two leading coefficients of \(A\) and \(B\), and therefore has to divide their g.c.d.

A slight variation of this corollary is provided by the following result.
Corollary 10 Every coefficient of the g.c.d. of \( A = \sum_{i=0}^{\alpha} a_i x^i \) and \( B = \sum_{i=0}^{\beta} b_i x^i \) (where \( a_i, b_i \) are integers) is bounded by

\[
2^{\min(\alpha, \beta)} \gcd(a_0, b_0) \min\left( \frac{1}{|a_0|} \sqrt[\alpha]{\sum_{i=0}^{\alpha} a_i^2}, \frac{1}{|b_0|} \sqrt[\beta]{\sum_{i=0}^{\beta} b_i^2} \right).
\]

Proof. If \( C = \sum_{i=0}^{\gamma} c_i x^i \) is a divisor of \( A \), then \( \hat{C} = \sum_{i=0}^{\gamma} c_{\gamma-i} x^i \) is a divisor of \( \hat{A} = \sum_{i=0}^{\alpha} a_{\alpha-i} x^i \), and conversely. Therefore, the last corollary can be applied to \( \hat{A} \) and \( \hat{B} \), and this yields the bound stated.

It may seem strange that the coefficients of a g.c.d. of two polynomials can be greater than the coefficients of the polynomials themselves. One example which shows this is the following (due to Davenport and Trager):

\[
A = x^3 + x^2 - x - 1 = (x + 1)^2(x - 1); \\
B = x^4 + x^3 + x + 1 = (x + 1)^2(x^2 - x + 1); \\
gcd(A, B) = x^2 + 2x + 1 = (x + 1)^2.
\]

This example can be generalised, as say

\[
A = x^5 + 3x^4 + 2x^3 - 2x^2 - 3x - 1 = (x + 1)^4(x - 1); \\
B = x^6 + 3x^5 + 3x^4 + 2x^3 + 3x^2 + 3x + 1 = (x + 1)^4(x^2 - x + 1); \\
gcd(A, B) = x^4 + 4x^3 + 6x^2 + 4x + 1 = (x + 1)^4.
\]

In fact, Mignotte [Mig81] has shown that the number 2 in corollaries 9 and 10 is asymptotically the best possible, i.e. it cannot be replaced by any smaller \( c \).

4.2.2 The modular – integer relationship

In this sub-section, we answer the question raised above: what do we do if the modular g.c.d. is not the modular image of the g.c.d. calculated over the integers?

Lemma 6 If \( p \) does not divide the leading coefficient of \( \gcd(A, B) \), the degree of \( \gcd(A_p, B_p) \) is greater than or equal to that of \( \gcd(A, B) \).

Proof. Since \( \gcd(A, B) \) divides \( A \), then \( (\gcd(A, B))_p \) divides \( A_p \). Similarly, it divides \( B_p \), and therefore it divides \( \gcd(A_p, B_p) \). This implies that the degree of \( \gcd(A_p, B_p) \) is greater than or equal to that of \( \gcd(A, B)_p \). But the degree of \( \gcd(A, B)_p \) is equal to that of \( \gcd(A, B) \), for the leading coefficient of \( \gcd(A, B) \) does not cancel when it is reduced modulo \( p \).

This lemma is not very easy to use on its own, for it supposes that we know the g.c.d. (or at least its leading coefficient) before we are able to check whether the modular reduction has the same degree. But this leading coefficient has to divide the two leading coefficients of \( A \) and \( B \), and this gives a formulation which is easier to use.
**Corollary 11** If $p$ does not divide the leading coefficients of $A$ and of $B$ (it may divide one, but not both), then the degree of $\gcd(A_p, B_p)$ is greater than or equal to that of $\gcd(A, B)$.

As the g.c.d. is the only polynomial (to within an integer multiple) of its degree which divides $A$ and $B$, we can test the correctness of our calculations of the g.c.d.: if the result has the degree of $\gcd(A_p, B_p)$ (where $p$ satisfies the hypothesis of this corollary) and if it divides $A$ and $B$, then it is the g.c.d. (to within an integer multiple).

It is quite possible that we could find a $\gcd(A_p, B_p)$ of too high a degree. For example, in the case cited above, $\gcd(A_2, B_2) = x + 1$ (it is obvious that $x + 1$ divides the two polynomials modulo 2, because the sum of the coefficients of each polynomial is even). The following lemma shows that this possibility can only arise for a finite number of $p$.

**Lemma 7** Let $C = \gcd(A, B)$. If $p$ satisfies the condition of the corollary above, and if $p$ does not divide $\text{Res}_x(A/C, B/C)$, then $\gcd(A_p, B_p) = C_p$.

**Proof.** $A/C$ and $B/C$ are relatively prime, for otherwise $C$ would not be the g.c.d. of $A$ and $B$. By the corollary, $C_p$ does not vanish. Therefore

$$\gcd(A_p, B_p) = C_p \gcd(A_p/C_p, B_p/C_p).$$

For the lemma to be false, the last g.c.d. has to be non-trivial. This implies that the resultant $\text{Res}_x(A_p/C_p, B_p/C_p)$ vanishes, by proposition 73 of the Appendix. This resultant is the determinant of a Sylvester matrix, and $|M_p| = (|M|)_p$, for the determinant is only a sum of products of the coefficients. In the present case, this amounts to saying that $\text{Res}_x(A/C, B/C)_p$ vanishes, that is that $p$ divides $\text{Res}_x(A/C, B/C)$. But the hypotheses of the lemma exclude this possibility.

**Definition 75** If $\gcd(A_p, B_p) = \gcd(A, B)_p$, we say that the reduction of this problem modulo $p$ is good, or that $p$ is of good reduction. If not, we say that $p$ is of bad reduction.

This lemma implies, in particular, that there are only a finite number of values of $p$ such that $\gcd(A_p, B_p)$ does not have the same degree as that of $\gcd(A, B)$, that is the $p$ which divide the g.c.d. of the leading coefficients and the $p$ which divide the resultant of the lemma (the resultant is non-zero, and therefore has only a finite number of divisors). In particular, if $A$ and $B$ are relatively prime, we can always find a $p$ such that $A_p$ and $B_p$ are relatively prime.

**Observation 6** It would be tempting to conclude “the probability that $p$ is bad is the probability that $p$ divides this resultant, i.e. $1/p$. However, a given resultant either is, or is not, divisible by $p$: there is no probability involved. If we consider the space of all $(A, B)$ pairs, then we have to allow for $p$ dividing the leading coefficients. This argument would also assume that the distribution of resultants

---

6Technically, the limit as $H \to \infty$ of all pairs with coefficients at most $H$. 
4.2. GCD IN ONE VARIABLE

is the same as the distribution of integers, and this seems not to be the case: very few resultants are actually prime, for example. Nevertheless, it is an empirical observation that the probability of \( p \) being bad does seem to be proportional to \( 1/p \).

An alternative proof that there are only finitely primes of bad reduction can be deduced from Lemma 10. We can summarize this section in the following.

**Theorem 34 (Good Reduction Theorem (Z))** If \( p \) does not divide \( \gcd(a_\alpha, b_\beta) \) (which can be checked for in advance) or \( \text{Res}_x(A/C, B/C) \), then \( p \) is of good reduction. Furthermore, if \( p \) divides \( \text{Res}_x(A/C, B/C) \) but not \( \gcd(a_\alpha, b_\beta) \), then the gcd computed modulo \( p \) has a larger degree than the true result.

### 4.2.3 Computing the g.c.d.: one large prime

In this section we answer the question posed earlier: how do we calculate a non-trivial g.c.d.? One obvious method is to use the Landau-Mignotte inequality, which can determine an \( M \) such that all the coefficients of the g.c.d. are bounded by \( M \), and to calculate modulo a prime number greater than \( 2M \). This method translates into Algorithm 17/Figure 4.2 (where Landau_Mignotte_bound\((A, B)\) applies corollary 9 and/or corollary 10 and find_large_prime\((2M)\) produces a different prime \( > 2M \) each time it is called within a given invocation of the algorithm). We restrict ourselves to monic polynomials, and assume modular_gcd gives a monic result, to avoid the problem that a modular g.c.d. is only defined up to a constant multiple.

**Algorithm 17 (Modular GCD (Large prime version))**

**Input:** \( A, B \) monic polynomials in \( \mathbb{Z}[x] \).

**Output:** \( \gcd(A, B) \)

\[
M := \text{Landau}\_\text{Mignotte}\_\text{bound}(A, B); \\
\text{do } p := \text{find}\_\text{large}\_\text{prime}(2M); \\
\quad \text{if } p \text{ does not divide } \gcd(\text{lc}(A), \text{lc}(B)) \\
\quad \quad \text{then } C := \text{modular}_\text{gcd}(A, B, p); \\
\quad \quad \quad \text{if } C \text{ divides } A \text{ and } C \text{ divides } B \\
\quad \quad \quad \quad \text{then return } C \\
\quad \text{forever} \quad \#\text{Lemma 10 guarantees termination}
\]

We can think of the use of modular_gcd as a Monte Carlo Algorithm (Section 1.4.1 and Algorithm 17 as an instance of Figure 11 converting a Monte Carlo algorithm into a Las Vegas (“always correct/probably fast”) one.)
If the inputs are not monic, the answer might not be monic. For example, $x + 4$ divides both $2x^2 + x$ and $2x^2 - x - 1$ modulo 7, but the true common divisor over the integers is $2x + 1$, which is $2(x + 4) \pmod{7}$. We do know that the leading coefficient of the g.c.d. divides each leading coefficient $\text{lc}(A)$ and $\text{lc}(B)$, and therefore their g.c.d. $g = \gcd(\text{lc}(A), \text{lc}(B))$. We therefore compute

$$C := \underbrace{\text{pp}(g \times \text{modular \ gcd}(A, B))}_{\text{computed modulo } M}, \text{ then interpreted in } \mathbb{Z}$$

(4.8)

instead, where the pp is needed in case the leading coefficient of the g.c.d. is a proper factor of $g$.

It is tempting to argue that this algorithm will only handle numbers of the size of twice the Landau–Mignotte bound, but this belief has two flaws.

- While we have proved that there are only finitely many bad primes, we have said nothing about how many there are. The arguments can in fact be made effective, but the results tend to be unduly pessimistic, since it is extremely likely that all the bad primes would be clustered just above $2M$.

- In theory, and indeed very often in practice, the division tests could yield very large numbers if done as tests of the remainder being zero: for example the remainder on dividing $x^{100}$ by $x - 10$ is $10^{100}$. This can be solved by a technique known as “early abort” trial division.

**Proposition 54** If $h$, of degree $m$, is a factor of $f$ of degree $n$, the coefficient of $x^{n-m-i}$ in the quotient is bounded by $(n-m-1)^{\frac{1}{\text{lc}(h)}}|f|$.

This is basically Corollary 25. Hence, as we are doing the trial division, we can give up as soon as we find a coefficient in the quotient that exceeds this bound, which is closely related to $M$ (the difference relates to the leading coefficient terms).

For example, if we take $p = 7$ in the example at the start of this chapter, we find that the g.c.d. of $A_7$ and $B_7$ is $x + 10$ (it could equally well be $x + 3$, but $x + 10$ makes the point better). Does $x + 10$ divide $B$? We note that $|B| \approx 23.92$. Successive terms in the quotient are $3x^0$ (and $3$ is a permissible coefficient), $-30x^4$ (and $30 < \binom{5}{4} \times 23.92$) and $305x^3$, at which point we observe that $305 > \binom{5}{4} \times 23.92 = 239.2$, so this cannot be a divisor of $B$. Hence 7 was definitely unlucky.

With this refinement, it is possible to state that the numbers dealt with in this algorithm are “not much larger” than $2M$, though considerable ingenuity is needed to make this statement more precise.

If we apply this algorithm to the polynomials at the start of this section, we deduce that $\sqrt{\sum_{i=0}^{8} a_i^2} = \sqrt{113}$, $\sqrt{\sum_{i=0}^{6} b_i^2} = 2\sqrt{143}$, and hence corollary 9
4.2. GCD IN ONE VARIABLE

gives a bound of
\[ 2^6 \min \left( \sqrt{113}, \frac{2}{3} \sqrt{143} \right) \approx 510.2, \]  
so our first prime would be 1021, which is indeed of good reduction. In this case, corollary 10 gives a bound of
\[ 2^6 \min \left( \frac{1}{5} \sqrt{113}, \frac{2}{21} \sqrt{143} \right) \approx 72.8, \]
so our first prime would be 149. In general, we cannot tell which gives us the best bound, and it is normal to take the minimum.

Open Problem 15 (Improving Landau–Mignotte for g.c.d.) A significant factor in the Landau–Mignotte bound here, whether (4.9) or (4.10), was the \( 2^{\min(8,6)} \) contribution from the degree of the putative g.c.d. But in fact the exponent is at most 4, not 6, since the g.c.d. cannot have leading coefficient divisible by 3 (since A does not). Hence the g.c.d. must have at most the degree of the g.c.d. modulo 3, and modulo 3 B has degree 4, so the g.c.d. must have degree at most 4.

Can this be generalised, in particular can we update our estimate “on the fly” as upper bounds on the degree of the g.c.d change, and is it worth it? In view of the ‘early success’ strategies discussed later, the answer to the last part is probably negative.

4.2.4 Computing the g.c.d.: several small primes

While algorithm 17 does give us some control on the size of the numbers being considered, we are still often using numbers larger than those which hindsight would show to be necessary. For example, in (4.6), (4.7) we could deduce co-primeness using the prime 5, rather than 1021 from (4.9) or 149 from (4.10). If instead we consider \((x-1)A\) and \((x-1)B\), the norms change, giving 812.35 in (4.9) (a prime of 1627) and 116.05 in (4.10) (a prime of 239). Yet primes such as 5, 11, 13 etc. will easily show that the result is \(x-1\). Before we leap ahead and use such primes, though, we should reflect that, had we taken \((x-10)A\) and \((x-10)B\), 5 would have suggested \(x\) as the gcd, 11 would have suggested \(x+1\), 13 would have suggested \(x+3\) and so on.

The answer to this comes in observing that the smallest polynomial (in terms of coefficient size) which is congruent to \(x\) modulo 5 and to \(x+1\) modulo 11 is \(x-10\) (it could be computed by algorithm 47). More generally, we can apply the Chinese Remainder Theorem (Theorem 47) to enough primes of good reduction, as follows. We assume that find_prime \(g\) returns a prime not dividing \(g\), a different one each time. The algorithm is given in Figure 4.3, with a diagram in Figure 4.4.

Observation 7 The reader may think that Algorithm 18 is faulty: line (*) in Figure 4.3 iterates until \(N \geq 2M\), which would be fine if we were actually computing the g.c.d. But we have forced the leading coefficient to be \(g\), which
Algorithm 18 (Modular GCD (Small prime version))

**Input:** $A, B$ polynomials in $\mathbb{Z}[x]$.  

**Output:** $\gcd(A, B)$

\[
M := \text{Landau-Mignotte bound}(A, B); \\
g := \gcd(\text{lc}(A), \text{lc}(B)); \\
p := \text{find prime}(g); \\
D := g \modular_{\gcd}(A, B, p); \\
\text{if } \deg(D) = 0 \text{ then return 1} \\
\]

$N := p; \quad \# N$ is the modulus we will be constructing

\[
\text{while } N < 2M \text{ repeat } \quad (*) \\
\quad p := \text{find prime}(g); \\
\quad C := g \modular_{\gcd}(A, B, p); \\
\quad \text{if } \deg(C) = \deg(D) \\
\quad \quad \text{then } D := \text{Algorithm } 47(C, D, p, N); \\
\quad \quad N := pN; \\
\quad \text{else if } \deg(C) < \deg(D) \\
\quad \quad \# C$ proves that $D$ is based on primes of bad reduction \\
\quad \quad \text{if } \deg(C) = 0 \text{ then return 1} \\
\quad \quad D := C; \\
\quad \quad N := p; \\
\quad \text{else } \# D$ proves that $p$ is of bad reduction, so we ignore it \\
\]

$D := \text{pp}(D); \quad \# \text{In case multiplying by } g \text{ was overkill}$

Check that $D$ divides $A$ and $B$, and return it

If not, all primes must have been bad, and we start again

The “early abort” of Proposition\[^{54}\] is needed for these divisibility checks if we are to maintain a “numbers not much larger than $2M$” guarantee.
4.2. GCD IN ONE VARIABLE

Figure 4.4: Diagrammatic illustration of Algorithm 18

\[ Z[x] \xrightarrow{k \times \text{reduce}} Z[x] \]

\[ \xrightarrow{\text{interpret and check}} \]

\[ \begin{array}{c}
\mathbb{Z}_p[x] \\
\vdots \\
\mathbb{Z}_{p_k}[x]
\end{array} \xrightarrow{\text{gcd}}
\begin{array}{c}
\mathbb{Z}_{p_1}[x] \\
\vdots \\
\mathbb{Z}_{p_k}[x]
\end{array} \]

\[ \xrightarrow{\text{C.R.T.}} \]

\[ \mathbb{Z}'_{p_1 \cdots p_k}[x] \]

\( Z'_p[p] \) indicates that some of the \( p_i \) may have been rejected by the compatibility checks, so the product is over a subset of \( p_1 \cdots p_k \).

may be overkill. Hence aren’t we in danger of trying to recover \( g \) times the true g.c.d., whose coefficients may be greater than \( 2M \)?

In fact there is not a problem. The proof of Corollary 9 relies on estimating the leading coefficient of \( \gcd(A, B) \) by \( g \), and so the bound is in fact a bound for the coefficients after this leading coefficient has been imposed.

Having said that, we can’t “mix and match”. If we decide that Corollary 10 provides a better lower bound than Corollary 9, then we must go for “imposed trailing coefficients” rather than “imposed leading coefficients”, or, and this is the way the author has tended to implement it, compute the g.c.d. of \( \hat{A} \) and \( \hat{B} \), and reverse that.

We should note the heavy reliance on Corollary 11 to detect bad reduction. We impose \( g \) as the leading coefficient throughout, and make the result primitive at the end as in the large prime variant.

4.2.5 Computing the g.c.d.: early success

While Algorithm 18 will detect a g.c.d. of 1 early, it will otherwise compute as far as the Landau–Mignotte bound if the g.c.d. is not 1. While this may be necessary, it would be desirable to terminate earlier if we have already found the g.c.d. This is easily done by replacing the line

then \( D := \text{Algorithm 17}(C, D, p, N); \)

by the code in Figure 4.5. We should note that we return an \( E \) which divides the inputs, and is derived from modular images, and therefore has to be the greatest common divisor by Corollary 11.

4.2.6 An alternative correctness check

So far we have suggested computing the putative g.c.d. \( G \), then checking that it really divides both, and relying on Corollary 11 to say that \( G \) is therefore a
then $D' := D$

$D := \text{Algorithm 17}(C, D, p, N)$;

if $D = D'$  # We may have found the answer

then $E := \text{pp}(D);$ 

if $E$ divides $A$ and $B$

then return $E$; 

# Otherwise this was a false alert, and we continue as normal.

Figure 4.5: “Early termination” g.c.d. code

greatest common divisor. An alternative approach is to compute the co-factors, i.e. $A'$ such that $A = A'G$ and $B'$ such that $B = B'G$ at the same time, and use these as the check. So let us assume that \texttt{modular\_gcd\_cofactors} returns a triple $[G, A', B']$ modulo $p$. The Algorithm (19) is given in Figure 4.6 and the diagram in Figure 4.4 is still relevant.

**Observation 8** It is tempting to conjecture that we do not need to make both the multiplication checks at the end, but this is false: consider $A = H$, $B = H + 3p_1 \cdots p_k$, when the algorithm will find $H$ as the putative g.c.d., since the Landau–Mignotte bound will ignore the large extra term in $B$, and only the multiplication check for $B$ will detect this.

**Observation 9** Early termination can perfectly well be applied to this variant: at any time $gA = DA'$ and $gB = DB'$ over the integers, we can finish.

### 4.2.7 Conclusion

**Observation 10** We have presented this material as if there were a choice between one large prime (Algorithm 17) and several small ones (Algorithms 18, 19). In practice, of course, a computer regards all numbers less than 32 bits (and increasingly 64 bits) as ‘small’, so an implementation would generally use the largest ‘small’ primes it could in Algorithms 18, 19, and thus often one prime will suffice, and we have the same effect as Algorithm 17.

Let us see how we have answered the questions on page 144.

1. Are there “good” reductions from $\mathbb{Z}$ to $\mathbb{Z}_p$? Yes — all primes except those that divide both leading coefficients (Lemma 6) and do not divide a certain resultant (Lemma 7). The technique of Lemma 10 (page 182) will show that there are only finitely many bad primes, but does not give a bound.

2. How can we tell if $p$ is good? We can’t, but given two different primes $p$ and $q$ which give different results, we can tell which is definitely bad: Corollary 11.
Algorithm 19 (Modular GCD (Alternative small prime version))

**Input:** $A, B$ polynomials in $\mathbb{Z}[x]$.

**Output:** $G := \gcd(A, B), A', B'$ with $A = A'G$, $B = B'G$.

\[ M := \text{Landau_Mignotte_bound}(A, B); \]
\[ g := \gcd(\text{lc}(A), \text{lc}(B)); \]
\[ p := \text{find_prime}(g); \]
\[ [D, A', B'] := \text{modular_gcd_cofactors}(A, B, p); \]
\[ \text{if } \deg(D) = 0 \text{ then return } [1, A, B]; \]
\[ D := gD \quad \# \text{ is our guess at the leading coefficient of the g.c.d.} \]
\[ N := p; \quad \# \text{ is the modulus we will be constructing} \]
\[ \text{while } N < 2M \text{ repeat} \]
\[ p := \text{find_prime}(g); \]
\[ [C, A_1, B_1] := \text{modular_gcd_cofactors}(A, B, p); \]
\[ \text{if } \deg(C) = \deg(D) \]
\[ \text{ then } D := \text{Algorithm 47}(C, D, p, N); \]
\[ A' := \text{Algorithm 47}(A_1, A', p, N); \]
\[ B' := \text{Algorithm 47}(B_1, B', p, N); \]
\[ N := pN; \]
\[ \text{else if } \deg(C) < \deg(D) \]
\[ \# \text{ } C \text{ proves that } D \text{ is based on primes of bad reduction} \]
\[ \text{if } \deg(C) = 0 \text{ then return } [1, A, B]; \]
\[ D := C; A' = A_1; B' = B_1; \]
\[ N := p; \]
\[ \text{else } \# D \text{ proves that } p \text{ is of bad reduction, so we ignore it} \]
\[ \text{if } gA = DA' \text{ and } gB = DB' \]
\[ \text{ then } G := \text{pp}(D) \gcd(\text{cont}(A), \text{cont}(B)); \quad \# \text{ Theorem 6} \]
\[ A' := A' / \text{lc}(G); B' := B' / \text{lc}(G); \quad \# \text{ Fix leading coefficients} \]
\[ \text{return } [G, A', B'] \]
\[ \text{else all primes must have been bad, and we start again} \]
CHAPTER 4. MODULAR METHODS

3. How many reductions should we take? We want the apparently good primes to have a product greater than twice the Landau–Mignotte bound (Corollaries 9 and 10). Alternatively we can use early termination as in Figure 4.5.


5. How do we check the result? If it divides both the inputs, then it is a common divisor, and hence (Corollary 11) has to be the greatest.

All these algorithms use modular computations as Monte Carlo (“always fast/probably correct”) algorithms (Section 1.4.1), converted into a Las Vegas (“always correct/probably fast”) one in the style of Figure 1.1 because we have correctness checks. In fact, because we can bound the number of bad primes (as opposed to just the probability of a prime being bad) we are actually guaranteed polynomial running time, so these Las Vegas algorithms are in fact perfectly normal algorithms, though the upper limits on the running times so obtained are almost always gross over-estimates.

4.3 Polynomials in two variables

The same techniques can be used to compute the greatest common divisor of polynomials in several variables. This is even more important than in the case of univariate polynomials, since the coefficient growth observed on page 57 becomes degree growth in the other variables.

4.3.1 Degree Growth in Coefficients

As a trivial example of this, we can consider

\[ A = (y^2 - y - 1)x^2 - (y^2 - 2)x + (2y^2 + y + 1); \]
\[ B = (y^2 - y + 1)x^2 - (y^2 + 2)x + (y^2 + y + 2). \]

The first elimination gives

\[ C = (2y^2 - 4y)x + (y^4 - y^3 + 2y^2 + 3y + 3), \]

and the second gives

\[ D = -y^{10} + 3y^9 - 10y^8 + 11y^7 - 23y^6 + 22y^5 - 37y^4 + 29y^3 - 32y^2 + 15y - 9. \]

Since this is a polynomial in \( y \) only, the greatest common divisor does not depend on \( x \), i.e. \( pp_x(\gcd(A, B)) = 1 \). Since \( \operatorname{cont}_x(A) = 1 \), \( \operatorname{cont}_x(\gcd(A, B)) = 1 \), and hence \( \gcd(A, B) = 1 \), but we had to go via a polynomial of degree 10 to show this. Space does not allow us to show bigger examples in full, but it can be shown that, even using the subresultant algorithm (Algorithm 4), computing
the g.c.d. of polynomials of degree $d$ in $x$ and $y$ can lead to intermediate\footnote{We stress this word. Unlike the integer case, where the coefficients of a g.c.d. can be larger than those of the original polynomials, the degree in $y$ of the final g.c.d. cannot be greater than the (minimum of) the degrees (in $y$) of the inputs.} polynomials of degree $O(d^2)$.

Suppose $A$ and $B$ both have degree $d_x$ in $x$ and $d_y$ in $y$, with $A = \sum a_i x^i$ and $B = \sum b_i x^i$. After the first division (which is in fact a scaled subtraction $C := a_x B - b_x A$), the coefficients have, in general, degree 2$d_y$ in $y$. If we assume\footnote{This is the "normal p.r.s. assumption: see footnote 34 (page 62).} that each division reduces the degree by 1, then the next result would be $\lambda B - (\mu x + \nu)C$ where $\mu = b_c$ has degree $d_y$ in $y$ and $\lambda$ and $\nu$ have degree 3$d_y$. This result has degree 5$d_y$ in $y$, but the subresultant algorithm will divide through by $b_c$, to leave a result $D$ of degree $d_x - 2$ in $x$ and 4$d_y$ in $y$. The next result would be $\lambda' C - (\mu' x + \nu')D$ where $\mu' = \text{lcm}(C)$ has degree 2$d_y$ in $y$ and $\lambda'$ and $\nu'$ have degree 6$d_y$. This result has degree 10$d_y$ in $y$, but the subresultant algorithm will divide by a factor of degree 4, leaving an $E$ of degree 6$d_y$ in $y$. The next time round, we will have degree (after subresultant removal) 8$d_y$ in $y$, and ultimately degree 2$d_x d_y$ in $y$ when it has degree 0 in $x$.

If this is not frightening enough, consider what happens if, rather than being in $x$ and $y$, our polynomials were in $n + 1$ variables $x$ and $y_1, \ldots, y_n$. Then the coefficients (with respect to $x$) of the initial polynomials would have degree $d_y$, and hence (at most) $(1+d_y)^n$, whereas the result would have $(1+2d_x d_y)^n$ terms, roughly $(2d_x)^n$, or exponentially many, times as many terms as the inputs.

If we wish to consider taking greatest common divisors of polynomials in several variables, we clearly have to do better. Fortunately there are indeed better algorithms. The historically first such algorithm is based on Algorithm 18, except that evaluating a variable at a value replaces working modulo a prime.

**Open Problem 16 (Alternative Route for Bivariate Polynomial g.c.d.)**

Is there any reasonable hope of basing an algorithm on Algorithm 17? Intuition says not, and that, just as Algorithm 18 is preferred to Algorithm 17 for the univariate problem, so should it be here, but to the best of the author’s knowledge the question has never been explored.

The reader will observe that the treatment here is very similar to that of the univariate case, and may well ask “can the treatments be unified?”. Indeed they can, and this was done in \cite{Lau82}, but the unification requires rather more algebraic machinery than we have at our disposal.

**Notation 25** From now until section 4.4, we will consider the bivariate case, $\gcd(A, B)$ with $A, B \in R[x, y] \equiv R[y][x]$, and we will be considering evaluation maps replacing $y$ by $v \in R$, writing $A_{y \rightarrow v}$ for the result of this evaluation.

This notation is analogous to the previous section, where $A_p$ was the remainder on dividing $A$ by $p$. 

Observation 11  Clearly $R[x, y] \equiv R[x]/[y]$ also, and the definition of g.c.d. is independent of this choice. Algorithmically, though, it seems as if we must make such a choice. Some systems may already have imposed a default choice, but if we have a free hand it is usual to choose as the main variable ($x$ in Notation 25) the one which minimises $\min(\deg(A), \deg(B))$.

4.3.2 The evaluation–interpolation relationship

In this sub-section, we answer a question analogous to that in section 4.2.2: what do we do if the g.c.d. of the evaluations is not the image under evaluation of the g.c.d. calculated before evaluation?

Lemma 8  If $y - v$ does not divide the leading coefficient of $\gcd(A, B)$, the degree of $\gcd(A_{y-v}, B_{y-v})$ is greater than or equal to that of $\gcd(A, B)$.

Proof.  Since $\gcd(A, B)$ divides $A$, then $(\gcd(A, B))_{y-v}$ divides $A_{y-v}$. Similarly, it divides $B_{y-v}$, and therefore it divides $\gcd(A_{y-v}, B_{y-v})$. This implies that the degree of $\gcd(A_{y-v}, B_{y-v})$ is greater than or equal to that of $\gcd(A, B)_{y-v}$. But the degree of $\gcd(A, B)_{y-v}$ is equal to that of $\gcd(A, B)$, for the leading coefficient of $\gcd(A, B)$ does not cancel when it is evaluated at $v$.

This lemma is not very easy to use on its own, for it supposes that we know the g.c.d. (or at least its leading coefficient) before we are able to check whether the modular reduction has the same degree. But this leading coefficient has to divide the two leading coefficients of $A$ and $B$, and this gives a formulation which is easier to use.

Corollary 12  If $y - v$ does not divide the leading coefficients of $A$ and of $B$ (it may divide one, but not both), then the degree of $\gcd(A_{y-v}, B_{y-v})$ is greater than or equal to that of $\gcd(A, B)$.

As the g.c.d. is the only polynomial (to within a multiple from $R[y]$) of its degree (in $x$) which divides $A$ and $B$, we can test the correctness of our calculations of the g.c.d.: if the result has the degree of $\gcd(A_{y-v}, B_{y-v})$ (where $v$ satisfies the hypothesis of this corollary) and if it divides $A$ and $B$, then it is the g.c.d. (to within a multiple from $R[y]$).

As in section 4.2.2 it is quite possible that we could find a $\gcd(A_{y-v}, B_{y-v})$ of too high a degree: consider $A = x - 1, B = x - y$ and the evaluation $y \mapsto 1$. The following lemma shows that this possibility can only arise for a finite number of $v$.

Lemma 9  Let $C = \gcd(A, B)$. If $v$ satisfies the condition of the corollary above, and if $y - v$ does not divide $\operatorname{Res}_x(A/C, B/C)$, then $\gcd(A_{y-v}, B_{y-v}) = C_{y-v}$.

Proof.  $A/C$ and $B/C$ are relatively prime, for otherwise $C$ would not be the g.c.d. of $A$ and $B$. By the corollary, $C_{y-v}$ does not vanish. Therefore

$$\gcd(A_{y-v}, B_{y-v}) = C_{y-v} \gcd(A_{y-v}/C_{y-v}, B_{y-v}/C_{y-v}).$$
For the lemma to be false, the last g.c.d. has to be non-trivial. This implies that the resultant \( \text{Res}_x(A_{y-v}/C_{y-v}, B_{y-v}/C_{y-v}) \) vanishes, by proposition 73 of the Appendix. This resultant is the determinant of a Sylvester matrix\(^9\) and \(|M_{y-v}| = (|M|)_{y-v} \), for the determinant is only a sum of products of the coefficients. In the present case, this amounts to saying that \( \text{Res}_x(A/C, B/C) \) vanishes, that is that \( y - v \) divides \( \text{Res}_x(A/C, B/C) \). But the hypotheses of the lemma exclude this possibility.

**Definition 76** If \( \gcd(A_{y-v}, B_{y-v}) = \gcd(A, B)_{y-v} \), we say that the evaluation of this problem at \( v \) is good, or that \( y - v \) is of good reduction. If not, we say that \( y - v \) is of bad reduction.

This lemma implies, in particular, that there are only a finite number of values \( v \) such that \( \gcd(A_{y-v}, B_{y-v}) \) does not have the same degree as that of \( \gcd(A, B) \), that is the \( y - v \) which divide the g.c.d. of the leading coefficients and the \( y - v \) which divide the resultant of the lemma (the resultant is non-zero, and therefore has only a finite number of divisors). In particular, if \( A \) and \( B \) are relatively prime, we can always find a \( v \) such that \( A_{y-v} \) and \( B_{y-v} \) are relatively prime.

We can summarize this section as follows.

**Theorem 35 (Good Reduction Theorem (polynomial))** If \( y - v \) does not divide \( \gcd(a_\alpha, b_\beta) \) (which can be checked for in advance) or \( \text{Res}_x(A/C, B/C) \), then \( v \) is of good reduction. Furthermore, if \( y - v \) divides \( \text{Res}_x(A/C, B/C) \) but not \( \gcd(a_\alpha, b_\beta) \), then the \( \gcd \) computed modulo \( y - v \) has a larger degree than the true result.

### 4.3.3 G.c.d. in \( \mathbb{Z}_p[x, y] \)

By Gauss’ Lemma (Theorem 6),

\[
\gcd(A, B) = \gcd(\text{cont}_x(A), \text{cont}_x(B)) \gcd(\text{pp}_x(A), \text{pp}_x(B)),
\]

and the real problem is the second factor.

**Observation 12** While the content of \( A = \sum_{i=0}^{m} a_i x^i \) can be computed as

\[
\gcd(a_m, \gcd(a_m-1, \gcd(a_m-2, \ldots, a_0)) \ldots),
\]

the following process is more efficient in practice (asymptotically, it’s only worth a constant factor on what is asymptotically the cheaper operation, but in practice it is worthwhile), and is valid over any g.c.d. domain \( R \) with a concept of ‘size’ equivalent to degree.

\(^9\)There’s a subtle point here. The resultant of polynomials of degrees \( m \) and \( n \) is the determinant of an \( (m + n)^2 \) matrix. Hence if \( y - v \) divides neither leading coefficient, the Sylvester matrix of \( A_{y-v} \) and \( B_{y-v} \) is indeed the reduction of the Sylvester matrix of \( A \) and \( B \). If \( y - v \) divides one leading coefficient, but not the other, the Sylvester matrix of \( A_{y-v} \) and \( B_{y-v} \) is smaller, and the reader should check that this only makes a difference of a product of that leading coefficient which doesn’t vanish when \( v \) is substituted for \( y \).
Figure 4.7: Diagrammatic illustration of Algorithm 21

\[
\begin{align*}
Z_p[y][x] & \rightarrow Z_p[y][x] \\
k \times \text{reduce} & \downarrow \\
Z_p[y_{v_1}] & \rightarrow \text{gcd} Z_p[y_{v_1}] \\
\vdots & \vdots \\
Z_p[y_{v_k}] & \rightarrow \text{gcd} Z_p[y_{v_k}]
\end{align*}
\]

\[
\text{C.R.T.} \rightarrow \prod (y - v_1) \cdots (y - v_k)
\]

\(\prod\) indicates that some of the \(v_i\) may have been rejected by the compatibility checks, so the product is over a subset of \((y - v_1) \cdots (y - v_k)\).

Algorithm 20 (Content)

Input: \(A = \sum_{i=0}^m a_i x^i \in R[x]\).

Output: \(\text{cont}_x(A)\)

\[
S := \{a_i\} \quad \# \text{Really a set, as duplicates don’t matter}
\]

\[g := \text{minimal degree element of } S; S := S \setminus \{g\}\]

if \(g\) is a unit

then return 1

\[
g := \text{gcd}(g, \sum_{h_i \in S} \lambda_i h_i) \quad \# \lambda_i \text{ random}
\]

if \(g\) is a unit

then return 1

for \(h \in S\)

if \(g\) does not divide \(h\)

then \(g := \text{gcd}(g, h)\)

if \(g\) is a unit

then return 1

return \(g\)

We present an Algorithm 21, analogous to Algorithm 18, for the g.c.d. of primitive polynomials. It would equally be possible to start from Algorithm 19. There is also an ‘early termination’ version, analogous to the code in Figure 4.5.

We should note the caveat on \texttt{find_value} in Figure 4.8 although we know there are only a finite number of bad values \(v\), since \(Z_p\) is finite it is possible for too many, indeed all, values in \(Z_p\) to be bad, and we may need to move to an algebraic extension of \(Z_p\) to have enough good values.

4.3.4 G.c.d. in \(Z[x, y]\)

We know how to compute g.c.d.s in \(Z_p[x]\), and wish to compute g.c.d.s in \(Z[x, y]\). We have seen the building blocks in the previous sections, diagrammatically in figures 4.3 and 4.7. We have a choice of ways of combining these constructions, though, depending on what we choose as the intermediate domain.
Algorithm 21 (Bivariate Modular GCD)

**Input:** $A, B$ primitive polynomials in $\mathbb{Z}_p[y][x]$.

**Output:** $\gcd(A, B)$

1. $g := \gcd(\text{lcm}(A), \text{lcm}(B))$
2. $v := \text{find}\_\text{value}(g)$
3. $D := g\text{modular}\_\text{gcd}(A_{y-v}, B_{y-v}, p)$
4. If $\deg(D) = 0$ then **return** $1$
5. $N := y - v$; # $N$ is the modulus we will be constructing
6. While $\deg_y(N) \leq \min(\deg_y(A), \deg_y(B))$
   - $v := \text{find}\_\text{value}(g)$
   - $C := g\text{modular}\_\text{gcd}(A_{y-v}, B_{y-v}, p)$
   - If $\deg(C) = \deg(D)$ then
     - $D := \text{Algorithm 49}(C, D, y - v, N)$
     - $N := (y - v)N$
   - Else if $\deg(C) < \deg(D)$
     - # $C$ proves that $D$ is based on values of bad reduction
     - If $\deg(C) = 0$ then **return** $1$
     - $D := C$
     - $N := y - v$
   - Else
     - # $D$ proves that $v$ is of bad reduction, so we ignore it
     - $D := pp(D)$; # In case multiplying by $g$ was overkill
8. Check that $D$ divides $A$ and $B$, and return it

If not, all values must have been bad, and we start again

$\text{find}\_\text{value}(g)$ finds a new value $v$ each time, such that $g_{y - v}$ does not vanish.

It is conceivable that we will exhaust $\mathbb{Z}_p$, in which case we have to move to choosing $v$ from an algebraic extension of $\mathbb{Z}_p$. 

Figure 4.8: Algorithm 21
CHAPTER 4. MODULAR METHODS

Figure 4.9: Diagrammatic illustration of g.c.d.s in $\mathbb{Z}[x, y]$ (1)

$\mathbb{Z}[y][x] \quad \longrightarrow \quad \mathbb{Z}[y][x]$  

$k \times \text{reduce} \downarrow$  

$\mathbb{Z}[y_{y-v_1}[x] \xrightarrow{\gcd} \mathbb{Z}[y_{y-v_1}][x]$  
$\vdots \quad \vdots \quad \vdots$  

$\mathbb{Z}[y_{y-v_k}[x] \xrightarrow{\gcd} \mathbb{Z}[y_{y-v_k}][x]$  

\[ \longrightarrow \quad \mathbb{Z}[y][x] \xrightarrow{\text{C.R.T.}} \prod(y-v_1) \cdots (y-v_k) \]

using Algorithm 18/19

Z[x] Here we use an analogue of Figure 4.7 to reduce us to the univariate case. The corresponding diagram then looks like Figure 19 There is one complication we should note, though. We are now applying the Chinese Remainder Theorem to polynomials in $\mathbb{Z}[y]$, rather than $\mathbb{Z}_p[y]$, and $\mathbb{Z}$ is not a field. Indeed, the Theorem does not always apply over $\mathbb{Z}$, since, for example, the polynomial satisfying $f(0) = 0$ and $f(2) = 1$ is $f = \frac{1}{2}y$, which is in $\mathbb{Q}[y]$ but not $\mathbb{Z}[y]$.

Should this happen, we actually know that all reductions so far are wrong, because they are compatible in degree, but cannot be right. Hence we start again.

$\mathbb{Z}_p[y][x]$ Here we use an analogue of Figure 4.2 to reduce us to the case of Figure 4.7 The overall structure is shown in Figure 4.10

Open Problem 17 (Which is the Better Route for Bivariate g.c.d.?) As far as the author can tell, the question of which route to follow has not been systematically explored. The initial literature [Bro71b, Algorithm M] and the more recent survey [Lau82] assume the route in Figure 4.10. [Bro71b] explicitly assumes that $p$ is large enough that we never run out of values, i.e. that the
algebraic extension at the end of Figure 4.8 is never needed.

Implementations the author has seen tend to follow the route in Figure 4.9. This is probably for pragmatic reasons: as one is writing a system one first wishes for univariate g.c.d.s, so implements Algorithm 18 (or 19). Once one has this, Figure 4.9 is less work.

There is a natural tendency to believe that Figure 4.10 is better, as all numbers involved are bounded by \( p \) until the very last Chinese Remainder calculations. On the other hand, if the g.c.d. is actually 1, but the first prime chosen is unlucky, one will reconstruct all the way back to a bivariate before realising the problem.

Let us look at the questions on page 144.

1. Are there "good" reductions from \( R \)? Yes, since there are only finitely many bad reductions, but it is possible that we will need to go to an algebraic extension if we are working over a finite field: see the caveat on \texttt{find-value} in Figure 4.8.

2. How can we tell if an evaluation \( v \) is good? We can't, but given two different evaluations \( v \) and \( w \) which give different results, we can tell which is definitely bad: Corollary 12.

3. How many reductions should we take? If \( d \) bounds the degree in \( y \) of the g.c.d., then we only need \( d + 1 \) good reductions. We don't have the same problem as in the univariate case, when our bounds were almost certainly pessimistic, but it is nevertheless common to use early termination as in Figure 4.5.

4. How do we combine the results — Algorithm 49.

5. How do we check the result? If it divides both the inputs, then it is a common divisor, and hence (Corollary 12) has to be the greatest.

As on page 160, these algorithms use modular/evaluation computations as Monte Carlo ("always fast/probably correct") algorithms (Section 1.4.1), converted into a Las Vegas ("always correct/probably fast") one in the style of Figure 1.1 because we have correctness checks. In fact, because we can bound the number of bad evaluations and bad primes (as opposed to just the probability of an evaluation/prime being bad) we are actually guaranteed polynomial running time. It is even truer here that the upper limits on the running times so obtained are almost always gross over-estimates.

4.4 Polynomials in several variables

There is no conceptual difficulty in generalising the work of the previous section to \( n \) variables. In principle, there are \( n \) possible diagrams corresponding to Figures 4.9, 4.10 but in practice the choice is between 'modular last' (Figure
and ‘modular first’ (Figure 4.10), and the author has never seen a hybrid algorithm which reduces, say,

\[ Z[y][z][x] \rightarrow Z[y][z][\pm x] \rightarrow Z_p[y][z][\pm x] \rightarrow Z_p[y][y'][z][\pm x] \]

Theorem 36 ([Bro71b, (95)]) Suppose \( f, g \in \mathbb{Z}[x_1, \ldots, x_n] \). Let \( l \) bound the lengths of the integer coefficients of \( f \) and \( g \), and \( d \) bound their degree in each of the \( x_i \). Then under the following assumptions:

1. classical \( O(N^2) \) multiplication and division of \( N \)-word numbers or \( N \)-term polynomials;
2. no bad values are found, either for primes or for evaluation points;
3. that the answer, and corresponding quotients, have coefficient lengths at most \( l \);
4. that we can use single-word primes;

the running time of the ‘modular first’ (Figure 4.10) algorithm is

\[ O \left( l^2 (d+1)^n + (d+1)^{n+1} \right) \]

This contrasts with the subresultant algorithm’s bound [op. cit. (80)] (assuming the p.r.s. are normal) of

\[ O \left( l^2 (d+1)^{4n} 2^{2n^2} 3^n \right) , \]

so the dependence on \( d^n \) has essentially gone from quartic to linear.

The real challenge comes with the potential sparsity of the polynomials. The factor \( (d+1)^n \) in the running time comes simply from the fact that a dense polynomial of degree \( d \) in \( n \) variables has that many terms, and we will need \((d+1)^{n-1}\) univariate g.c.d.s (therefore \( (d+1)^n \) coefficients) to deduce these potential terms. Furthermore it is possible for sparse polynomials to have dense g.c.d.s: the following elegant univariate example is due to [Sch03a]:

\[
gcd \left( \underbrace{x^p q - 1}_{f(x)} , \underbrace{x^p q + x - x^q + 1}_{g(x)} \right) = \frac{(x^p - 1)(x^q - 1)}{x - 1} = x^{p+q-1} - x^{p+q-2} + \cdots - 1, \tag{4.11} \]

Example 13 (Dense g.c.d.s) Therefore

\[
gcd(f(x_1)f(x_2) \cdots f(x_k), g(x_1)g(x_2) \cdots g(x_k)) = h(x_1)h(x_2) \cdots h(x_k), \tag{4.12} \]

and the righthand side has \((p + q)^k\) terms, whereas the arguments to gcd have \(2^k\) and \(4^k\) terms.
4.4. POLYNOMIALS IN SEVERAL VARIABLES

Problem 5 (Sparse g.c.d.) Produce an algorithm for sparse multivariate g.c.d. whose running time is polynomial, ideally linear, in the number of terms in the inputs and output. If we can’t find a deterministic algorithm, maybe we can find a Las Vegas (section 1.4.1) one.

The ‘modular first’ (Figure 4.10) algorithm does not solve this problem as it will attempt to interpolate \( \prod(d_i + 1) \) terms if the g.c.d. has degree \( d_i \) in \( x_i \). The problem was first addressed, in Las Vegas style, in Zip79a, Zip79b.

4.4.1 A worked example

Let \( f \) be

\[
x^5y^7z^4 + x^4y^5z^4 + x^4y^7z^4 + x^2y^8z^4 + x^2y^5z^5 + x^4y^3z^5 +
\]

\[
x^3y^5z^6 + x^2y^7z^4 + x^2y^9z^6 - x^5y^7 + x^5y^2 - x^4y^6 + x^4y^3z^5 +
\]

\[
x^4y^3z^5 - 2x^4y^7z^4 + 2x^4y^5z^4 + 2x^3y^8z^6 + 2x^4y^6z^2 - x^4y^2z^5 -
\]

\[
x^4y^3z^5 - 2x^4y^7z^4 + 2x^4y^5z^4 + 2x^3y^8z^6 + 2x^4y^6z^2 - x^4y^2z^5 +
\]

\[
x^4y^3z^5 + 2x^4z^7z^2 + x^4y^4z^2 + 3x^2y^7 + x^4y^3z + 4x^4y^3 + 3x^2y^3 +
\]

\[
x^4y^2z^2 + x^3yz + 3x^4y + x^4z - x^4 - x^2z + 2x^2 - 2z + 3
\]

and \( g \) be

\[
x^6y^7z^4 - x^6y^7z^4 + x^6y^7z^5 + x^4y^7z^3 + x^4y^4z^6 - x^6y^7 + x^6y^2 + x^4y^7z^2 -
\]

\[
x^4y^2z^5 - 2x^2y^7z^4 + 2x^4y^5z^4 + 2x^3y^8z^6 + 2x^4y^6z^2 - x^4y^2z^5 +
\]

\[
x^4y^3z^5 + 2x^4z^7z^2 + x^4y^4z^2 + 3x^2y^7 + x^4y^3z + 4x^4y^3 + 3x^2y^3 +
\]

\[
x^4y^2z^2 + x^3yz + 3x^4y + x^4z - x^4 - x^2z + 2x^2 - 2z + 3
\]

polynomials with 42 and 39 terms respectively (as against the 378 and 392 they would have if they were dense of the same degree).

Let us regard \( x \) as the variable to be preserved throughout. If we compute \( \gcd(f|_{z=2}, g|_{z=2}) \) (which would require at least\(^{10}9 \) 8 y values) we get (assuming that \( z = 2 \) is a good reduction)

\[
\gcd(f, g)|_{z=2} = (15y^7 + 31y^3 + 63y) x^4 + (15y^7 + 32y^4 + 1) x^2 + 1.
\]

We note that each coefficient (with respect to \( x \)) has at most three terms. A dense algorithm would compute five more equivalents of \( \text{(4.15)} \), at different \( z \) values, and then interpolate \( z \) polynomials. Each such computation would require 8 y values. Instead, if we believe\(^{11} \) that \( \text{(4.15)} \) describes accurately the dependence of the g.c.d. on \( y \), we should be able to deduce these equivalents with only three \( y \) values. Consider

\[
\begin{align*}
\gcd(f|_{z=3, y=1}, g|_{z=3, y=1}) &= 525x^4 + 284x^2 + 1 \\
\gcd(f|_{z=3, y=-1}, g|_{z=3, y=-1}) &= -525x^4 + 204x^2 + 1 \\
\gcd(f|_{z=3, y=2}, g|_{z=3, y=2}) &= 6816x^3 + 9009x^2 + 1
\end{align*}
\]

\(^{10}\)In fact, with \( y = 0 \) both \( f \) and \( g \) drop in \( x \)-degree, whereas with \( y = -2 \) we get a g.c.d. with \( x \)-degree 5, as \( y = -2 \) is a root of the resultant in Lemma 3.

\(^{11}\)We refer to this as the Zippel assumption, after its introduction in Zip79a, Zip79b, and formalize it in Definition 78.
We should interpolate the coefficients of \( x^4 \) to fit the template \( ay^7 + by^3 + cy \), and those of \( x^2 \) to fit the template \( a'y^7 + b'y^4 + c \), while those of \( x^0 \) should fit the template \( a''y^0 \), i.e. be constant. Considering the coefficients of \( x^4 \), we have to solve the equations

\[
\begin{align*}
  a + b + c &= 525 \\
  -a - b - c &= -525 \\
  2^7a + 2^3b + 2c &= 6816 \\
\end{align*}
\]  

(4.17)

Unfortunately, these equations are under-determined (the second is minus twice the first), so we have to add another equation to (4.16), e.g.

\[ \gcd(f|_{z=3,y=3}, g|_{z=3,y=3}) = 91839x^4 + 107164x^2 + 1, \]

which adds \( 3^7a + 3^3b + 3c = 91839 \) to (4.17) and the augmented system is now solvable, as \( a = 40, \ b = 121, \ c = 364 \).

This process gives us an assumed (two assumptions are now in play here): that \( z = 3 \) is a good reduction, and that (4.15) describes the sparsity structure \( \gcd(f, g) \) accurately value of

\[ \gcd(f, g)|_{z=3} = \left( 40y^7 + 121y^3 + 364y \right) x^4 + \left( 40y^7 + 243y^4 + 1 \right) x^2 + 1 \]  

(4.18)

Similarly we can deduce that

\[
\begin{align*}
  \gcd(f, g)|_{z=4} &= \left( 85y^7 + 341y^3 + 1365y \right) x^4 + \left( 85y^7 + 1024y^4 + 1 \right) x^2 + 1 \\
  \gcd(f, g)|_{z=5} &= \left( 156y^7 + 781y^3 + 3906y \right) x^4 + \left( 156y^7 + 3125y^4 + 1 \right) x^2 + 1 \\
  \gcd(f, g)|_{z=6} &= \left( 259y^7 + 1555y^3 + 9331y \right) x^4 + \left( 259y^7 + 7776y^4 + 1 \right) x^2 + 1 \\
  \gcd(f, g)|_{z=7} &= \left( 400y^7 + 2801y^3 + 19608y \right) x^4 + \left( 400y^7 + 16807y^4 + 1 \right) x^2 + 1 \\
  \gcd(f, g)|_{z=8} &= \left( 585y^7 + 4681y^3 + 37449y \right) x^4 + \left( 585y^7 + 32768y^4 + 1 \right) x^2 + 1 \\
\end{align*}
\]  

(4.19)

(each requiring, at least, three \( y \) evaluations, but typically no more).

If we now examine the coefficients of \( x^4y^7 \), and assume that in \( \gcd(f, g) \) there is a corresponding term \( x^4y^7p(z) \), we see that \( p(2) = 15 \) (from (4.15)), \( p(3) = 40 \) (from (4.18)), and similarly \( p(4), \ldots, p(8) \) from (4.19). Using Algorithm 48 we deduce that \( p(z) = z^3 + z^2 + z + 1 \). We can do the same for all the other (presumed) \( x^iy^j \) terms in the g.c.d., to get the following final result:

\[
x^4\left( y^7 \left( z^3 + z^2 + z + 1 \right) + y^3 \left( z^4 + z^3 + z^2 + z + 1 \right) + \right.
\left. y \left( z^5 + z^4 + z^3 + z^2 + z + 1 \right) \right) + x^2 \left( y^7 \left( z^3 + z^2 + z + 1 \right) + y^4z^5 + 1 \right) + 1.
\]  

(4.20)

We still need to check that it does divide \( g \) and \( g \), but, once we have done so, we are assured by Corollary 12 that it is the greatest common divisor.

### 4.4.2 Converting this to an algorithm

There are, however, several obstacles to converting this to an algorithm: most centring around the linear systems such as (4.17). First, however, we must formalize the assumption made on page 169.
Definition 77. We define the skeleton of a polynomial \( f \in \mathbb{R}[x_1, \ldots, x_n] \), denoted \( \text{Sk}(f) \), to be the set of monomials in a (notional) distributed representation of \( f \).

Definition 78. We say that an evaluation \( \phi : \mathbb{R}[x_1, \ldots, x_n] \to \mathbb{R}[x_i : i \notin S] \), which associates a value in \( \mathbb{R} \) to every \( x_i : i \notin S \), satisfies the Zippel assumption for \( f \) if \( \psi(\text{Sk}(f)) = \text{Sk}(\phi(f)) \), where \( \psi \) deletes the powers of \( x_i : i \notin S \) from monomials.

For example, if \( f = (y^2 - 1)x^2 - x \), \( \text{Sk}(f) = \{x^2y^2, x^2, x\} \). If \( \phi : y \mapsto 1 \), then \( \phi(f) = -x \), so \( \text{Sk}(\phi(f)) = \{x\} \), but \( \psi(\text{Sk}(f)) = \{x^2, x\} \), so this \( \phi \) does not satisfy the Zippel assumption. However, if \( \phi : y \mapsto 2 \), \( \phi(f) = 3x^2 - x \) and \( \text{Sk}(\phi(f)) = \{x^2, x\} \), so this \( \phi \) satisfies the Zippel assumption.

Hence the assumption on page 169 was that \( z \mapsto 2 \) satisfies the Zippel assumption. Let us now look at the problems surrounding (4.17). In fact, there are three of them.

1. The equations might be under-determined, as we saw there.

2. We need to solve a \( t \times t \) system, where \( t \) is the number of terms being interpolated. This will normally take \( O(t^3) \) coefficient operations, and, as we know from section 3.2.3, the coefficients may also grow during these operations.

3. The third problem really goes back to (4.16). We stated that
   \[
   \gcd(f|_{z=3,y=-1}, g|_{z=3,y=-1}) = -525x^4 + 204x^2 + 1,
   \]
   but we could equally well have said that it was \( 525x^4 - 204x^2 - 1 \), at which point we would have deduced that the equivalent of (4.17) was inconsistent. In fact, Definition 26 merely defines a greatest common divisor, for the reasons outlined there. We therefore, as in Algorithm 18, impose \( \gcd(lc(f), lc(g)) \), as the leading coefficient of the greatest common divisor.

To solve the first two problems, we will use the theory of section A.5 and in particular Corollary 28 which guarantees that a Vandermonde matrix is invertible. In fact, we will need to solve a modified Vandermonde system of form (A.11), and rely on Corollary 29. If, instead of random values for \( y \), we had used powers of a given value, we would get such a system in place of (4.17).

We now give the interlocking set of algorithms, in Figures 4.12–4.14. We present them in the context of the diagram in Figure 4.11.

4.4.3 Worked example continued

We now consider a larger example in four variables. Let \( f \) and \( g \) be the polynomials in Figures 4.15 and 4.16, with 104 and 83 terms respectively (as against the 3024 and 3136 they would have if they were dense of the same degree).
CHAPTER 4. MODULAR METHODS

Figure 4.11: Diagrammatic illustration of sparse g.c.d.

\[ R[y][x] \rightarrow R[y][x] \]
\[ \downarrow k \times \text{reduce} \]
\[ R[y]_{y_n-v_1}[x] \xrightarrow{\text{gcd}} R[y]_{y_n-v_1}[x] \]
\[ \sqrt{\text{Sk}} \]
\[ R[y]_{y_n-v_2}[x] \xrightarrow{\text{gcd'}} R[y]_{y_n-v_2}[x] \]
\[ \vdots \]
\[ R[y]_{y_n-v_k}[x] \xrightarrow{\text{gcd'}} R[y]_{y_n-v_k}[x] \]
\[ \prod' \]
\[ \text{C.R.T.} \]
\[ \frac{R[y][x]}{\prod(y_n-v_1) \cdots (y_n-v_k)} \]

\[ \prod' \] indicates that some of the \( v_i \) may have been rejected by the compatibility checks, so the product is over a subset of \( (y_n-v_1) \cdots (y_n-v_k) \).

\( \text{gcd} \) is a recursive call to this algorithm, while \( \text{gcd'} \) indicates a g.c.d. computation to a prescribed skeleton, indicated by Sk: Algorithm 24.

Figure 4.12: Algorithm Sparse g.c.d.

Algorithm 22 (Sparse g.c.d.)

Input: \( A, B \) polynomials in \( R[y][x] \).

Output: \( \text{gcd}(A,B) \)

\[ A_c := \text{cont}_x(A); A := A/A_c; \quad \# \text{Using Algorithm 20 which} \]
\[ B_c := \text{cont}_x(B); B := B/B_c; \quad \# \text{calls this algorithm recursively} \]
\[ g := \text{Algorithm 22}(\text{lc}_x(A), \text{lc}_x(B)); \quad \# \text{one fewer variable} \]
\[ G := \text{failed} \]
while \( G = \text{failed} \) \# Expect only one iteration
\[ G := \text{Algorithm 23}(g, gA, gB); \]
return \( \text{pp}_x(G) \times \text{Algorithm 22}(A_c, B_c); \quad \# \text{one fewer variable} \]
Figure 4.13: Algorithm 23 Inner sparse g.c.d.

Algorithm 23 (Inner sparse g.c.d.)

**Input:** $g$ leading coefficient; $A,B$ polynomials in $R[y][x]$.

**Output:** $\text{gcd}(A,B)$, but with leading coefficient $g$
or failed if we don’t have the Zippel assumption

\[ d_n := \min(\deg_y(A), \deg_y(B)) \]  
\[ v_0 := \text{random}(\text{lc}(A), \text{lc}(B)); \]  # (over)estimate for $\deg_y(\text{gcd}(A,B))$

\[ P_0 := \text{Algorithm 23}(g|_{y_n-v_0}, A|_{y_n-v_0}, B|_{y_n-v_0}) \]

if $P_0 =$failed
  return failed  # Don’t try again

\[ d_x := \deg_x(P_0) \]
\[ i := 0 \]

while $i < d_x$
  \[ v := \text{random}(\text{lc}(A), \text{lc}(B)); \]  # assumed good
  \[ P_{i+1} := \text{Algorithm 24}(\text{Sk}(P_i), g|_{y_n-v_0}, A|_{y_n-v_0}, B|_{y_n-v_0}) \]
  if $\deg_x(P_{i+1}) > d_x$  # Either $v^j$ in Algorithm 24 or continue round the loop
    # $y_n-v$ was not a good evaluation
  else
    \[ i := i + 1; v_i := v \]  # store $v$ and the corresponding $P$
    \[ C := \text{Algorithm 49}(\{P_i\}, \{v_i\}) \]  # Reconstruct
    if $C$ divides both $A$ and $B$
      return $C$
    else return failed

random(lc(A), lc(B)) chooses a value not annihilating both leading coefficients
Figure 4.14: Algorithm 24 Sparse g.c.d. from skeleton

Algorithm 24 (Sparse g.c.d. from skeleton)

Input: Sk skeleton, g leading coefficient; A, B polynomials in R[y][x].
Output: gcd(A, B) assumed to fit Sk, but with leading coefficient g 
or the univariate gcd if this doesn’t fit Sk

Note that y is \((y_1, \ldots, y_{n-1})\) since \(y_n\) has been evaluated

\[
d_x := \deg_x(Sk) \\
t := \max_{0 \le i \le d_x} \text{term count}(Sk, x^i) \\
v = (v_1, \ldots, v_{n-1}) := \text{values in } R \\
\text{for } j := 1 \ldots t \\
\quad C := \gcd(A_y=v_j, B_y=v_j) \\
\quad \text{if } \deg_x(C) \neq d_x \\
\quad \quad \text{return } C \quad \# \text{ We return the univariate} \\
\quad P_j := \frac{\deg_x(C)}{\deg_x(C)} C \quad \# \text{ impose leading coefficient} \\
G := 0 \\
\text{for } i := 0 \ldots d_x \\
\quad S := [y^n : x^i y^n \in Sk] \\
\quad R := [m|y=v_j : m \in S] \\
\quad V := [\text{coeff}(x^i, P_j) : j = 1 \ldots \text{length}(S)] \\
\quad W := \text{Algorithm } 51(R, V) \\
\quad \text{for } j := 1 \ldots \text{length}(S) \\
\quad \quad G := G + W_j S_j x^i \\
\text{return } G
\]
4.4. POLYNOMIALS IN SEVERAL VARIABLES

Let us consider computing \( \gcd(f, g) \) where \( f \) and \( g \) are polynomials of (4.13) and (4.14), whose g.c.d. we have already computed to be (4.20).

We note that the coefficients (with respect to \( x \)) have fifteen, six and one term respectively, and \( f \) and \( g \) both have degree seven in \( w \). We need to compute seven more equivalents of (4.20), at different \( w \) values, and then interpolate \( w \) polynomials. Let us consider computing \( \gcd(f, g) \) under the assumptions that 1 and 2 are good values of \( w \), and that (4.20) correctly describes the sparsity structure of \( \gcd(f, g) \).

We take evaluations with \( y = 2^i \) and \( z = 3^j \). For simplicity, let us consider the coefficient of \( x^2 \), which, we know from (4.20), is assumed to have the shape

\[
y^7 (a_1 z^3 + a_2 z^2 + a_3 z + a_4) + a_5 y^4 z^5 + a_6.
\]

(4.21)

Hence, taking the coefficients of \( x^2 \) from Table 4.1.

<table>
<thead>
<tr>
<th>( x )</th>
<th>( y )</th>
<th>( z )</th>
<th>( \gcd(f, g) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>2</td>
<td>\gcd(f, g)</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>2</td>
<td>19612 ( x^4 + 9009 x^2 + 127 )</td>
</tr>
<tr>
<td>8</td>
<td>27</td>
<td>2</td>
<td>43407439216 ( x^4 + 101638909953 x^2 + 127 )</td>
</tr>
<tr>
<td>16</td>
<td>81</td>
<td>2</td>
<td>1446930041136768 ( x^4 + 372950726410241 x^2 + 127 )</td>
</tr>
<tr>
<td>32</td>
<td>243</td>
<td>2</td>
<td>495206093484836032 ( x^4 + 1383508483289120769 x^2 + 127 )</td>
</tr>
<tr>
<td>64</td>
<td>729</td>
<td>2</td>
<td>1706321451043380811520 ( x^4 + 516051383842297503825 x^2 + 127 )</td>
</tr>
<tr>
<td>128</td>
<td>215</td>
<td>2</td>
<td>19841169176522147209214289933300473966170786477572096 ( x^4 + 821125029267868660742578644189298158324777001819308033 x^2 + 127 )</td>
</tr>
<tr>
<td>256</td>
<td>315</td>
<td>2</td>
<td>\vdots</td>
</tr>
</tbody>
</table>
2^7 3^4 a_1 + 2^7 3^2 a_2 + 2^7 3 a_3 + 2^7 a_4 + 2^4 3^5 a_5 + a_6 = 9009
(2^7 3^3)^2 a_1 + (2^7 3^2)^2 a_2 + (2^7 3)^2 a_3 + (2^7)^2 a_4 + (2^4 3^3)^2 a_5 + a_6 = 28551425
(2^7 3^3)^3 a_1 + (2^7 3^2)^3 a_2 + (2^7 3)^3 a_3 + (2^7)^3 a_4 + (2^4 3^3)^3 a_5 + a_6 = 10 \ldots
(2^7 3^3)^4 a_1 + (2^7 3^2)^4 a_2 + (2^7 3)^4 a_3 + (2^7)^4 a_4 + (2^4 3^3)^4 a_5 + a_6 = 37 \ldots
(2^7 3^3)^5 a_1 + (2^7 3^2)^5 a_2 + (2^7 3)^5 a_3 + (2^7)^5 a_4 + (2^4 3^3)^5 a_5 + a_6 = 13 \ldots
(2^7 3^3)^6 a_1 + (2^7 3^2)^6 a_2 + (2^7 3)^6 a_3 + (2^7)^6 a_4 + (2^4 3^3)^6 a_5 + a_6 = 51 \ldots
(4.22)

This is indeed a system of linear equations

4.4.4 Conclusions

Let us look at the questions on page 144.

1. Are there evaluations which are not merely good in the sense of the previous section, but also satisfy the Zippel assumption? The answer is yes. For the evaluation \( y_n = v \) to be good, \( v \) must not annihilate both leading coefficients (checked for in random) and must not be a root of the relevant resultant (Corollary 12). For the Zippel assumption to be violated, a certain coefficient must vanish under the \( \psi \) of Definition 78. There are only finitely many such, and each such only has finitely many roots, hence in all there are only finitely many bad values for each \( x_i \).

2. How can we tell if an evaluation should be used? The tests in Algorithm 23 apply.

3. How many reductions should we take? The number of good reductions at each stage \( x_i \) is \( 1 + \min(\deg_{x_i}(f), \deg_{x_i}(g)) \), by the usual argument on the degree of a g.c.d.

Open Problem 18 (Bad reductions in Zippel’s algorithm) An interesting question is how many bad reductions there can be. If the degree in each variable is at most \( d \), then the number of values that annihilate both leading coefficients is at most \( d \), and the number that divide the relevant resultant is at most \( 2d^2 \), the degree of the resultant. The challenge is to bound the number that violate the Zippel assumption, which we showed was finite in the previous point.

It seems to the author, though he has not seen this in the literature, that, as a polynomial with \( t \) terms can have at most \( 2t - 1 \) roots (Proposition 24), the number of Zippel-bad values is bounded by twice the number of terms in the g.c.d. being computed. Though we have no good a priori bound on this (Example 13), it means that the number of Zippel-bad reductions is linear in the size of the output.

Conjecture 1 Let \( d_f, d_g \) be the degrees of \( f \) in the main variable \( x \), \( d \) be \( \max_i \min(\deg_{y_i}(f), \deg_{y_i}(g)) \), \( d' \) be \( \max_i(\deg_{y_i}(f) + \deg_{y_i}(g)) \) and \( t \) the
4.5. FURTHER APPLICATIONS

number of terms in $\gcd(f, g)$. Then the running time of Zippel’s algorithm is bounded by

$$\left( d + (d_f + d_g)d' + 2t \right) \frac{\min(d_e, d_g)dt}{\text{cost}}$$

where we multiply the possible number of failed evaluations by the cost of a successful calculation.

4. How do we combine? This is the main subtlety of this method, using the theory of section A.5.

5. How do we check the result? We have used the “If $C$ divides both $A$ and $B$” test, as in Algorithm 17 et seq. We could consider using a “reconstruction of cofactors” argument, as in Algorithm 19. To the best of the author’s knowledge, this route has not been explored experimentally. However, there is an argument against it: it would require the evaluations to satisfy the Zippel assumption for the cofactors as well as for the g.c.d., and intuitively this seems like asking for trouble.

A variant on this algorithm is presented in [HM13], which uses an interpolation technique from [GLL09], and this seems to be the current state of the art, though there is no formal complexity analysis.

In practice these methods seem to solve Open Problem 2 a g.c.d. algorithm whose running time is polynomial in the number of terms in the input and outputs, even though (4.23) is $O(t^2)$, rather than the $O(t)$ we would need for the algorithm to be optimal.

4.5 Further Applications

4.5.1 Resultants and Discriminants

The resultant of two polynomials $f = \sum_{i=0}^n a_i x^i$ and $g = \sum_{i=0}^m b_i x^i$ is defined (Definition 100) as the determinant of a certain $(m+n) \times (m+n)$ matrix $Syl(f, g)$. Hence if $Syl(f, g)|_p = Syl(f|_p, g|_p)$ (or the equivalent for $|_{y-v}$), the theory of Section 4.1 holds. But the only way this can fail to hold is if $Syl(f|_p, g|_p)$ is no longer an $(m+n) \times (m+n)$ matrix, i.e. if $a_n$ or $b_m$ evaluate to zero. Hence again we have simple answer to the questions on page 144.

1. Are there "good" reductions from R: yes — all evaluations which do not reduce $a_nb_m$ to zero, and this can be tested a priori.

2. How can we tell if $R_i$ is good?: in advance!

3–5 As in Section 4.1.
4.5.2 Linear Systems

We consider the matrix equation

\[ Mx = a, \]  

and the case of \( M \) and \( a \) having integer entries, and reduction modulo a prime: the polynomial version is very similar. Conceptually, this has the well-known solution

\[ x = M^{-1}a. \]  

(3.14)

This formulation shows the fundamental problem: \( M^{-1} \), and \( x \) itself, might not have integer entries. In fact, if \( p \) divides \( \det(M) \), \( (M|_p)^{-1} \) does not exist. There are two solutions to this problem.

4.5.2.1 Clear Fractions

If we compute \( \det(M) \) first, by the method of section 4.1, we can clear denominators in (3.14), and get

\[ \tilde{x} := \det(M)x = \det(M)M^{-1}a, \]  

(4.24)

i.e.

\[ M\tilde{x} = \det(M)a. \]  

(4.25)

If we avoid primes dividing \( \det(M) \), we can solve (4.25) for enough primes (suitable bounds are given in Corollary 24), reconstruct integer entries in \( \tilde{x} \), and then divide through by \( \det(M) \).

4.5.2.2 Solve with Fractions

If (3.13) is soluble modulo \( p \), its solution \( x_p \) is indeed congruent to \( x \) when evaluated at \( p \), i.e. \( x|_p = x_p \). If we use many primes \( p_i \) (discarding those for which (3.13) is not soluble), and apply Algorithm 46 to the vectors \( x_{p_i} \), we get a vector \( x_N \) such that \( x_N \equiv x \pmod{N} \), where \( N = \prod p_i \). However, the entries of \( x \) are rationals, with numerator and denominator bounded, say, by \( B \) (see Corollary 24), rather than integers. How do we find the entries of \( x \) from \( x_N \)? This problem has a long history in number theory, generally under the name Farey fractions, but was first considered in computer algebra in [Wan81]. Since we will have occasion to use this solution elsewhere, we consider it in more generality in the next section.

If we assume this problem solved, we then have the following answer to the questions on page 144.

1. Are there good primes?: yes — all that do not divide \( \det(M) \)

2. How can we tell if a prime \( p \) is bad? Equation (3.13) is not soluble modulo \( p \).
3. How many reductions should we take? Enough such that the product of the good primes is greater than \(2B^2\).


5. How do we check the result? If we use the bound from Corollary 24, we do not need to check. However, there are “early success” variations, analogous to section 4.2.5, where we do need to check, which can be done by checking that \(Mx = a\); an \(O(n^2)\) arithmetic operations process rather than the \(O(n^3)\) of solving.

4.5.2.3 Farey reconstruction

In this section, we consider the problem of reconstructing an unknown fraction \(x = n/d\), with \(|n|,|d| < B\), given that we know \(x \equiv y \pmod{N}\), i.e. \(n \equiv yd \pmod{N}\), where \(N > 2B^2\). We first observe that this representation is unique, for if \(n'/d'\) (similarly bounded) is also congruent to \(y\), then \(nd' \equiv ydd' \equiv n'd\), so \(nd' - n'd \equiv 0 \pmod{N}\), and the only solution satisfying the bounds is \(nd' - n'd = 0\), i.e. \(n/d = n'/d'\).

Actually finding \(n\) and \(d\) is done with the Extended Euclidean Algorithm (see Algorithm 5).

Algorithm 25 (Farey Reconstruction)

Input: \(y, N \in \mathbb{N}\)
Output: \(n, d \in \mathbb{Z}\) such that \(|n|,|d| < \sqrt{N}/2\) and \(n/d \equiv y \pmod{N}\), or failed if none such exist.

\[
i := 1; \quad a_0 := N; \quad a_1 := y; \quad a := 1; \quad d := 1; \quad b := e := 0
\]

\[
\text{#Loop invariant: } a_i = ay + bN; \quad a_{i-1} = cy + dN;
\]

while \(a_i > \sqrt{N}/2\) do

\[
a_{i+1} = \text{rem}(a_{i-1}, a_i);
\]

\[
q_i := \text{the corresponding quotient}; \quad # a_{i+1} = a_{i-1} - q_ia_i
\]

\[
e := c - q_ia; \quad e' := d - q_ib; \quad # a_{i+1} = ef + e'g
\]

\[
i := i + 1;
\]

\[
(a, b) = (a, b);
\]

\[
(a, b) = (e, e')
\]

\[
\text{if } |a| < \sqrt{N}/2 \text{ and } \gcd(b, N) = 1 \]

\[\text{then return } (a_i, a)\]

\[\text{else return } \text{failed}\]

Correctness of this algorithm, i.e. the fact that the first \(a_i < \sqrt{N}/2\) corresponds to the solution if it exists, is proved in [WGD82], using [HW79] Theorem 184. The condition gcd\((b, N) = 1\) was stressed by [CE95], without which we may return meaningless results, such as \((-1, 2)\), when trying to reconstruct 5 (mod 12).
4.6 Gröbner Bases

If coefficient growth is a major problem in g.c.d. computations, it can be even more apparent in Gröbner basis computations. \[Arn03\] gives the example of

\[
\begin{align*}
8x^2y^2 + 5xy^3 + 3x^3z + x^2yz, & \quad x^5 + 2y^3z^2 + 13y^2z^3 + 5yz^4, \\
8x^3 + 12y^3 + xz^2 + 3, & \quad 7x^2y^4 + 18xy^3z^2 + y^3z^3,
\end{align*}
\]

whose Gröbner base, computed with practically any order, is

\[
\left\{x, y^3 + \frac{1}{4}, z^2\right\}.
\]

Both (4.26) and (4.27) involve very manageable numbers, but computing with total degree orders gives intermediate terms such as

\[
807916413783107531643530794245367453679\ldots
\]

(4.26)

and a lexicographic order gives coefficients with tens of thousands of digits. Can the same ideas allow us to bypass such huge intermediate expressions?

**Observation 13** [Tra88] proposed a very simple efficiency improvement, known as the Gröbner trace idea. We should only compute and reduce an \(S\)-polynomial over \(\mathbb{Q}\) if the equivalent computation modulo \(p\) yields a non-zero polynomial. This isn’t guaranteed correct, so a final check, either using Theorem 16(1) or on the lines of Theorem 39, is necessary, but can be a great time-saver in practice.

We should note that there can be various differences between the Gröbner base computed over the rationals and that computed modulo \(p\).

**Example 14** These examples are taken from [Win88].

1. (From [Ebe83]). Let \(F = \{xy^2 - 2y, x^2y + 3x\}\). Then (under any order) \(S(F_1, F_2) = 5xy\). Hence if this is non-zero, it reduces the other elements of \(F\) to \(\{x, y\}\), which is a Gröbner base. But if this is zero (i.e. if we are working modulo 5), then \(F\) is already a Gröbner base (Theorem 16 clause 1).

2. Let \(F = \{7xy + y + 4x, y + 2\}\) Then (under any order) \(S(F_1, F_2) = -10x + y\).

\[p = 2\] This is therefore \(y\), which is \(F_2\). Hence a reduced Gröbner base is \(\{y\}\): infinitely many solutions (\(x\) unconstrained).

\[p = 5\] This is therefore \(y\), which reduces \(F_2\) to 2, hence the Gröbner base is \(\{1\}\): no solutions.

**Otherwise** We get a Gröbner base of \(\{y + 2, 5x + 1\}\), with one solution.
3. Let $F = \{4 xy^2 + 16 x^2 - 4 z + 1, 2 y^2 z + 4 x + 1, -2 x^2 z + 2 y^2 + x\}$ with a purely lexicographic ordering $z > y > x$. Then the Gröbner base over the integers is
\[
\begin{align*}
32 & x^7 + 232 x^6 - 34 x^4 - 44 x^3 + x^2 + 30 x + 8, \\
2745 & y^2 + 112 x^6 - 756 x^5 - 11376 x^4 - 65 x^3 + 756 x^2 + 1772 x + 2, \\
10980 & z - 6272 x^6 - 45504 x^5 + 216 x^4 + 3640 x^3 - 36846 x^2 - 412 x - 2857. \\
\end{align*}
(4.28)
\]

$p = 2$ Here $F$ is immediately $\{1, x^2 + 1, 1\}$ and there are no solutions, which is a classic “$p$ divides the leading coefficient” issue.

$p = 7$ Though there might not seem to be a problem (none of the leading coefficients above are divisible by 7), in fact $S(S(F_1, F_3), S(F_1, F_2))$ reduces $S(F_1, F_2)$ to $14 x^2 y^2 + y^2 - 8 x^4 - 2 x^3 + 8 x + 2$, and the leading monomial here vanishes modulo 7. In fact, though the computation follows a different route, we get out a Gröbner basis of the same shape as (4.28).

If we reverse the order, though, events take a somewhat different course. The Gröbner base over the integers is
\[
\begin{align*}
16 & z^7 - 8 z^6 + z^5 + 52 z^4 + 75 z^3 - 342 z^2 + 266 z - 60, \\
1988 & y^2 - 76752 z^6 + 1272 z^5 - 4197 z^4 - 251555 z^3 - 481837 z^2 \\
&+ 1407741 z - 595666, \\
3976 & x + 37104 z^6 - 600 z^5 + 2111 z^4 + 122062 z^3 + 232833 z^2 \\
&- 680336 z + 288814. \\
\end{align*}
(4.29)
\]

Here 1988 is divisible by 7, and we might expect a different result modulo 7, which indeed we get:
\[
\begin{align*}
z^6 + 4 z^5 + z^4 + 6 z^3 + 5 z^2 + 2 z + 2, \\
y^2 z^2 + 3 z^5 + 6 z^4 + 5 z^3 + 6 y^2 + 3 z + 4, \\
y^2 z + 5 y^2 + 6 z^5 + 2 z^4 + 4 z^3 + 5 y^2 + 4 z^2 + 6 z + 5, \\
x + 2 z^5 + 4 z^4 + z^3 + 4 y^2 + 2 z. \\
\end{align*}
(4.30)
\]

This example shows that the badness of the prime might depend, not just on the input polynomials, but also on the ordering chosen. Both (4.29) and (4.30) have 14 solutions (as does (4.28), since this does not depend on the order), but the bases have different leading monomial ideals.

The rest of this section is substantially based on \cite{Arno03}: a $p$-adic approach to Gröbner bases is described in section 5.9.2. As in that paper, we assume that our ideal is homogeneous, i.e. that it can be generated by a set

\textsuperscript{13} A somewhat different approach is taken in \cite{PS11}: they distinguish good/bad primes by majority voting — their algorithm deleteUnluckyPrimesSS. They also use the Gröbner trace idea: subsequent primes do not try an $S$-polynomial which the first prime reduced to 0.

\textsuperscript{14} Note that there may well be sets of generators which are not homogeneous.
of generators each of which is homogeneous (Definition 48). If we wish to handle non-homogeneous ideals with respect to a total degree ordering\textsuperscript{15} we can homogenize, compute the Gröbner basis and dehomogenize, as in [MM84]. This may not be optimal, see [GMN+91, p. 49] and Open Problem 21 later. [IPS11] have shown that the assumption of homogeneity is not strictly necessary.

4.6.1 General Considerations

Given the blow-up in coefficients that can arise when computing Gröbner bases, we might wish to compute them by a modular method.

**Definition 79** Given a specific computation $C$ (by which we mean specifying not just the ordering, but all the choices in Algorithm 9 or any other algorithm for computing a Gröbner base) of a Gröbner base $G$ from an input $S$ over $\mathbb{Q}$, denoted $G := C(S)$, we say that the prime $p$ is of good reduction if $\text{lt}(G) = \text{lt}(C(S_p))$, i.e. we get the same leading term ideal when we apply $C$ modulo $p$ as when we apply it over $\mathbb{Q}$. If not, we say that $p$ is of bad reduction.

**Lemma 10** For a given $S$ and $C$, there are only finitely many primes of bad reduction.

**Proof.** If we compare $C(S)$ and $C(S_p)$, they can only start differing when a leading coefficient in the computation of $C(S)$ vanishes when considered modulo $p$, because the computation of $S$-polynomials, and reduction, is entirely driven by leading terms. But there are only finitely many such coefficients, and hence only finitely many bad primes: the set of divisors of these coefficients.

We should note that this proof is far less effective than the proof of Lemma 7, but it is sufficient for our purposes. Another proof is given in [Win88, Theorem 1], which in fact shows that the set of bad primes depends only on the input and the ordering, not on the computation.

Note that we cannot follow the plan of section 4.2.3 as we do not know a bound on all the integers appearing (and anyway, the idea is to compute with smaller integers). If we try to follow the plan of section 4.2.4 we hit a snag. Suppose that two computations modulo different primes (say $p$ and $q$) give us different leading term ideals. Then one is certainly bad, but which? We do not have a simple degree comparison as in the g.c.d. case.

4.6.2 The Hilbert Function and reduction

We recall the Hilbert function from section 3.3.12 which will turn out to be a key tool in comparing Gröbner bases, as earlier we have used degrees for comparing polynomials.

\textsuperscript{15}And we generally wish to do so, before, if necessary, using the FGLM Algorithm 12 to compute a lexicographical base.
Theorem 37 ([Arn03, Theorem 5.3]) Let \((f_1, \ldots, f_k)\) generate the ideal \(I\) over \(\mathbb{Q}\), and the ideal \(I_p\) modulo \(p\). Then
\[
\forall n \in \mathbb{N} : H_I(n) \leq H_{I_p}(n).
\] (4.31)

Definition 80 \(p\) is said to be Hilbert-good if, and only if, we have equality in (4.31).

Observation 14 Note that we do not have a test for Hilbert-goodness as such, but we do have one for Hilbert-badness: If \(H_{I_p}(n) < H_{I_q}(n)\) then \(q\) is definitely Hilbert-bad.

Open Problem 19 (Contradictory Hilbert Functions) It is conceivable to have \(H_{I_p}(n) < H_{I_q}(n)\) but \(H_{I_p}(m) > H_{I_q}(m)\), in which case both \(p\) and \(q\) must be bad. Can we give any examples of this?

Proposition 55 If \(p\) is of good reduction for \(C(S)\), then it is Hilbert-good for the ideal generated by \(S\).

Theorem 38 ([Arn03, Theorem 5.6]) Let \((g_1, \ldots, g_t)\) be a Gr"obner base under \(<\) for the ideal generated by \((f_1, \ldots, f_s)\) over \(\mathbb{Q}\), and \((g'_1, \ldots, g'_t)\) be a Gr"obner base under \(<\) for the ideal generated by \((f_1, \ldots, f_s)\) modulo \(p\). In both cases these bases are to be ordered by increasing order under \(<\). Suppose that \(p\) is Hilbert-good for this ideal. Then:

1. \(\text{lt}(g'_i) \leq \text{lt}(g_i)\);
2. If \(\text{lt}(g'_j) = \text{lt}(g_j)\) for \(1 \leq j \leq i\), then \(\text{lt}(g'_{i+1}) \leq \text{lt}(g_{i+1})\).

This result means that, for Hilbert-good primes, we can compare the sequence of \(\text{lt}(g_i)\) for relative luckiness as we compared degrees in the case of g.c.d.s: at the first point \(i\) where they differ, the prime with the lesser \(\text{lt}(g_i)\) is definitely of bad reduction.

Example 15 ([Arn03, Example 5.7]: importance of caveat above) Let \(I = \langle 3y^2x - 5yx^2 + 2x^3, -7y^3x + 5y^2x^2, 7y^6 - 2y^3x^3 + yx^5 \rangle\). We use the ordering ‘total degree then lexicographic, with \(y > x\)’, and consider the primes 5 and 2.

\(I_5\) has Gröbner base
\[
\{ 3y^2x + 2x^3, 29yx^3, x^5, y^6 \},
\] (4.32)
whereas \(I_2\) has Gröbner base
\[
\{ y^2x + yx^2, y^6 + yx^5 \}.
\] (4.33)

We can tabulate the Hilbert functions as in Table 4.2, though in fact we know they are different, since \(I_5\) has dimension zero and \(I_2\) does not, and hence the Hilbert polynomials have different degree. Either way, Hilbert functions tell us that 2 is definitely not good.
Table 4.2: Hilbert functions for example [15]

<table>
<thead>
<tr>
<th>l</th>
<th>( H_{I_5}(l) )</th>
<th>( H_{I_2}(l) )</th>
<th>( M_l )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>{1}</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>3</td>
<td>{1, x, y}</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>6</td>
<td>{1, x, y, x^2, xy, y^2}</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>9</td>
<td>{1, x, y, x^2, xy, y^2, x^3, x^2y, y^3}</td>
</tr>
<tr>
<td>4</td>
<td>12</td>
<td>11</td>
<td>{1, x, y, x^2, xy, y^2, x^3, x^2y, y^3, x^4, x^3y, y^4}</td>
</tr>
</tbody>
</table>

If we just looked at the leading terms, we would note that \( xy^2 \), the least leading term, is the same for both \( I_2 \) and \( I_5 \), but that \( x^3y < y^6 \) (the next leading terms), leading us, incorrectly, to infer that \( 5 \) was of bad reduction.

In fact, \( 5 \) is of good reduction, and the true Gröbner base for \( I \) itself (as determined by Chinese remainder with 7 and 11) is

\[
\{3y^2x - 5yx^2 + 2x^3, 29yx^3 - 20x^4, x^5, y^6\}\].

(4.34)

Open Problem 20 (Bad Reduction for Gröbner Bases) Is there a corresponding example where the dimension does not differ, i.e. when we actually have to compare values of the Hilbert functions?

4.6.3 The Modular Algorithm

We are now most of the way towards a Chinese Remainder Theorem solution to computing Gröbner Bases for an ideal \( I(f_1, \ldots, f_k) \). We still have to solve three issues.

leading coefficients In the case of g.c.d. computations, the modular answer is only defined up to multiplication by a constant, and the same is true here: each element of the Gröbner base modulo \( p \) can be multiplied, independently, by any non-zero constant. For g.c.d.s, we knew a multiple of the leading coefficient of the answer (i.e. the g.c.d. of the leading coefficients of the inputs), which we imposed in [4.8] and thereafter. Here we have no such prior knowledge. We therefore reconstruct a monic Gröbner base, with rational number coefficients, using Algorithm 25 above.

When do we stop? In the g.c.d. case we had the Landau–Mignotte bound. For Gröbner bases, while some bounds are known (e.g. [Dah09]), they are not directly relevant. In practice we take a leaf from Figure 4.5 and say that, if the Gröbner base doesn’t change when we take a new prime, we are likely to have terminated.

How do we know correctness? The first issue is that our reconstructed object \( G \), while a Gröbner base modulo each prime we used, may not be a
Gröbner base over the rationals. This can be tested by Theorem 16 clause (1), using, as appropriate, the optimizations from Propositions 38 and 39. But, even if $G$ is a Gröbner base, is it one for $I$?

Once we know $G$ is a Gröbner base, we can check that $I \subseteq \langle G \rangle$ simply by checking that $f_i \rightarrow^G 0$ for $i = 1, \ldots, k$. Inclusion the other way round comes from the following theorem.

**Theorem 39 ([Arn03, Theorem 7.1])** If $G$ is a Gröbner base, $I \subseteq \langle G \rangle$ and $\text{lm}(G) = \text{lm}(G_p)$ for some $G_p$ a Gröbner base obtained from the generators of $I$ modulo $p$, then $I = \langle G \rangle$.

It is worth noting that we have produced no complexity bound for this algorithm, just as we have stated none for the original Algorithm 9. In practice it seems to be especially efficient in cases where the final Gröbner basis has small coefficients, despite the intermediate expression swell.

**Open Problem 21 (Modular Gröbner Bases for Inhomogeneous Ideals)** This algorithm as described is limited, as is [Arn03], to homogeneous ideals with respect to a total degree ordering. Can it be generalized? [IPS11, Remark 2.5] claims that it can be, at the cost of either

- being content with a probabilistic algorithm, which may return a $G$ such that $\langle f_1, \ldots, f_k \rangle \subset \langle G \rangle$; or

- homogenizing first — however they say “experiments showed that this is usually not efficient since the standard basis of the homogenized input often has many more elements than the standard basis of the ideal that we started with”.

**Open Problem 22 (Reconstructed Bases might not be Gröbner)** We stated that it was possible that the reconstruction of modular Gröbner bases might not be a Gröbner base. This certainly seems to be theoretically possible, though we have given no examples of this.

1. Give examples where this happens. This is probably trivial, but what might be harder is to give ones where the primes were nevertheless good, and the problem is that we did not take enough of them.

2. Or show that this cannot in fact happen when the primes are of good reduction.

**4.6.4 Conclusion**

Let us see how we have answered the questions on page 144.

1. Are there "good" reductions $p$? Yes, by Lemma 10 there are only finitely many primes of bad reduction, though we have no bounds on the number.
Algorithm 26 (Modular Gröbner base)

**Input:** $S = \{f_1, \ldots, f_k\}$ homogeneous polynomials in $\mathbb{Z}[x_1, \ldots, x_n]$.

**Output:** $G$ a Gröbner base for $\langle S \rangle$ over $\mathbb{Q}$.

1. $p := \text{find\_prime}(S)$;
2. $G := \text{modular\_GB}(S, p)$;
3. $G_{\mathbb{Q}} := \text{Algorithm 25}(G, p)$;
4. $N := p$;  # $N$ is the modulus we will be constructing

while true repeat

1. $p := \text{find\_prime}(S)$;
2. $G' := \text{modular\_GB}(S, p)$;
3. if $H_G < H_{G'}$ then # Do nothing: $p$ is of bad reduction
4. else if $H_G > H_{G'}$ (*)
5. then $G := G'$; $N := p$;  previous primes were bad
6. $G_{\mathbb{Q}} := \text{Algorithm 25}(G, p)$
7. else if $\text{lm}(G) < \text{lm}(G')$ # Comparison in the sense of Theorem 38
8. then # Do nothing: $p$ is of bad reduction
9. else if $\text{lm}(G) > \text{lm}(G')$
10. $G := G'$; $N := p$;  previous primes were bad
11. $G_{\mathbb{Q}} := \text{Algorithm 25}(G, p)$
12. else if $G' \equiv G_{\mathbb{Q}} \pmod{p}$ # Stabilization as in Figure 4.5
13. and $G_{\mathbb{Q}}$ is a Gröbner base
14. and $\forall i f_i \rightarrow G_{\mathbb{Q}}$
15. return $G_{\mathbb{Q}}$ # Correct by Theorem 39
16. else $G := \text{Algorithm 47}(G', G, p, N)$;
17. $N := pN$;
18. $G_{\mathbb{Q}} := \text{Algorithm 25}(G, N)$

**find\_prime** finds a prime that does not divide any $\text{lc}(f_i)$ (otherwise we’re bound to be following a different computation, and almost certain to have bad reduction).

**modular\_GB** computes a *monic* modular Gröbner base, either via Algorithm 9 or any other such (Observation 3).

(*) Note that in principle the Hilbert functions might be incomparable (see Open Problem 19). In this case every reduction we have seen so far is bad and we have to start again.
2. How can we tell if $p$ is good? Following [Arn03], we have a two-stage process for comparing $p$ and $q$ when $G_p$ and $G_q$ differ: we first compare Hilbert functions (Theorem 37), then leading monomial ideals (Theorem 38).

3. How many reductions should we take? We have no bound, but rely on stabilization, as in Figure 4.5.


5. How do we check the result? Theorem 39 [IPS10] propose doing a further modular check, but this works for them as it is essentially the equivalent of waiting for stabilization in Algorithm 26.

4.7 Conclusions

1.

2.

3.

4.

5.
Chapter 5

\(p\)-adic Methods

In this chapter, we wish to consider a different problem, that of factoring polynomials. We will see that this cannot reasonably be solved by the methods of the previous chapter, and we need a new technique for solving problems in (large) domains via small ones. The technique is based on the mathematical concept of \(p\)-adic numbers, and its fundamental result, Hensel’s Lemma. The basic idea behind these algorithms is shown in Figure 5.1: instead of doing a calculation in some (large) domain \(R\), we do it in several smaller domains \(R_i\), pick one of these (say \(R_l\)) as the best one, grow the solution to some larger domain \(\hat{R}_l\), regard this as being in \(R\) and check that it is indeed the right result.

5.1 Introduction to the factorization problem

For simplicity, we will begin with the case of factoring a univariate polynomial over \(\mathbb{Z}\). More precisely, we consider the following.

**Problem 6** Given \(f \in \mathbb{Z}[x]\), compute polynomials \(f_i \in \mathbb{Z}[x]\) (1 ≤ \(i\) ≤ \(k\)) such that:

![Figure 5.1: Diagrammatic illustration of Hensel Algorithms](image-url)
1. \( f = \prod_{i=1}^{k} f_i \);  
2. each \( f_i \) is irreducible in \( \mathbb{Z}[x] \), i.e. any polynomial \( g \) that divides \( f_i \) is either an integer or has the same degree as \( f_i \).

We might wonder whether we wouldn’t be better off considering \( f \in \mathbb{Q}[x] \), but in fact the answers are the same.

**Proposition 56** Any factorization over \( \mathbb{Q}[x] \) of a polynomial \( f \in \mathbb{Z}[x] \) is (up to rational multiples) a factorization over \( \mathbb{Z}[x] \).

**Proof.** Let \( f = \prod_{i=1}^{k} f_i \) with \( f_i \in \mathbb{Q}[x] \). By clearing denominators and removing contents, we can write \( f \) as \( \left( \prod_{i=1}^{k} c_i \right) \left( \prod_{i=1}^{k} g_i \right) \), and, since the product of primitive polynomials is primitive (Lemma 2), \( \prod_{i=1}^{k} c_i \) is an integer, and can be absorbed into, say, \( g_1 \).

Even knowing that we have only to consider integer coefficients does not seem to help much — we still seem to have an infinite number of possibilities to consider. In fact this is not quite the case.

**Notation 26** Let the polynomial \( f = \sum_{i=0}^{n} a_i x^i \) to be factored have degree \( n \), and coefficients bounded by \( H \). Let us suppose we are looking for factors of degree at most \( d \).

**Corollary 13 (to Theorem 33)** It is sufficient to look for factors of degree \( d \leq n/2 \), whose coefficients are bounded by \( 2^d H \).

One might have hoped that it was sufficient to look for factors whose coefficients are bounded by \( H \), but this is not the case. [Abb09] gives the example of

\[
\begin{align*}
    f &= x^{80} - 2x^{78} + x^{76} + 2x^{74} + 2x^{70} + x^{68} + 2x^{66} + x^{64} + x^{62} + 2x^{60} + 2x^{58} \\
    &\quad - 2x^{54} + 2x^{52} + 2x^{50} + 2x^{48} - x^{44} - x^{42} + 2x^{38} + 2x^{36} + 2x^{34} + 2x^{32} + 2x^{30} + 2x^{28} - 2x^{26} \\
    &\quad + 2x^{22} + 2x^{20} + x^{18} + x^{16} + 2x^{14} + x^{12} + 2x^{10} + 2x^{6} + x^{4} - 2x^{2} + 1
\end{align*}
\]

whose factors have coefficients as large as 36, i.e. 18 times as large as the coefficients of \( f \). Non-squarefree polynomials are even worse: [Abb09] p. 18 gives the example of

\[
\begin{align*}
    f &= -1 - x + x^2 - x^3 + x^4 + x^5 + x^6 + x^8 + x^{10} - x^{11} \\
    &\quad - x^{12} - x^{13} + x^{14} - x^{15} - x^{17} - x^{18} + x^{20} + x^{21} = \\
    &\quad (1 + 4x + 8x^2 + 14x^3 + 21x^4 + 28x^5 + 34x^6 + 39x^7 + 42x^8 + 43x^9 + 41x^{10} + 37x^{11} + 32x^{12} + 27x^{13} + 21x^{14} + 15x^{15} + 9x^{16} + 4x^{17} + x^{18}) (x - 1)^3
\end{align*}
\]

Although some caution is in order, his table appears to show coefficient growth behaving like \( 0.7 \times 1.22^d \), where \( d \) is the degree of a polynomial with coefficients at most \( \pm 1 \).

**Corollary 14** To detect all irreducible factors of \( f \) (except possibly for the last one, which is \( f \) divided by all the factors of degree \( \leq n/2 \)), it suffices to consider \( (2^d H)^{d+1} \) polynomials.
We can in fact do better, since the leading coefficient of the factor must divide $a_n$, and similarly the trailing coefficient must divide $a_0$, so we get $2^{d(d-1)}H^{d+1}$, and in practice such “brute force” methods$^1$ can easily factor low-degree polynomials, but the asymptotics are still very poor.

5.2 Modular methods

Hence we might want to use modular methods. Assuming always that $p$ does not divide $a_n$, we know that, if $f$ is irreducible modulo $p$, it is irreducible over the integers. Since, modulo $p$, we can assume our polynomials are monic, there are only $p^d$ polynomials of degree $d$, this gives us a bound that is exponential in $d$ rather than $d^2$. In fact, we can do better by comparing results of factorizations modulo different primes.

Example 16 Suppose $f$ is quartic, and factors modulo $p$ into a linear and a cubic, and modulo $q$ into two quadratics (all such factors being irreducible), we can deduce that it is irreducible over the integers, since no factorization over the integers is compatible with both pieces of information.

Definition 81 For a square-free polynomial $f$, define its factorization shape to be the multiset of the degrees of all the irreducible factors in a given factorization of $f$ into irreducibles. Roughly speaking, this is the “degrees of all the irreducible factors”, but we have to be slightly careful, since if $(x-1)$ is an irreducible factor, so is $(-x+1)$, and we don’t want to count both. Note that the factorization shape is independent of which factorization we take.

If $f$ is not known to be square-free, then we have to take a multiset of (degree, multiplicity) pairs. This case is not useful in practice, but because a square-free decomposition can be larger than the starting point (Observation 2, page 66), it can be useful in theory: see section 5.8.1.

In this language, we are saying that the factorization shape modulo $p$ has to be a splitting of the factorization shape over the integers.

5.2.1 The Musser test

Definition 82 The sumset of a factorisation shape is the set of all sums of subsets of the factorisation shape.

The sumset of a factorisation shape modulo $p$ is therefore the set of all possible degrees of factors over the integers. Example 16 has two factorisation shapes $\{1,3\}$ and $\{2,2\}$, whose sumsets are $\{0,1,3,4\}$ and $\{0,2,4\}$. The intersection of these sumsets is $\{0,4\}$, so no proper factorisation over $\mathbb{Z}$ is possible.

$^1$Sometimes known as Newton’s algorithm.

$^2$That is, a set but allowing repetitions, so $\{2,2,1\}$ and $\{2,1,1,1\}$ would be different multisets, but of course are both the set $\{2,1\}$. 
It is an old experimental observation [Mus78] that, if a polynomial can be proved irreducible by intersecting sumsets of factorisation shapes, five shapes, i.e. five primes, are nearly always sufficient to prove it.

This test was formally analysed in [PPRL], and very substantial experimental evidence is presented in their Figure 1. The bottom line is that we should probably take seven primes. The rest of this subsection is somewhat more technical, and can be skipped by the reader not familiar with Galois Theory and permutation groups.

**Definition 83** The Galois group of a polynomial \( f = a_n \prod_{i=1}^{n} (x - \alpha_i) \in \mathbb{Z}[x] \) is the set of all automorphisms of \( \mathbb{Q}(\alpha_1, \ldots, \alpha_n) \) which leaves \( \mathbb{Q} \) fixed. The Galois group is determined by its action on the \( \alpha_i \), and is therefore a subgroup of the symmetric group \( S_n \).

**Proposition 57** \( f \) is irreducible if, and only if, its Galois group is transitive (for all roots \( \alpha \) and \( \beta \), there is a permutation of the Galois group taking \( \alpha \) to \( \beta \)).

**Example 17** Various Galois groups are as follows.

\( x^2 - 2 \) Here the roots are \( \alpha = \sqrt{2} \approx 1.4142 \) and \( \beta = -\sqrt{2} \approx -1.4142 \). Apart from the identity permutation, the only option is to exchange \( \alpha \) and \( \beta \). Hence the Galois Group is \( C_2 = \langle (\alpha, \beta) \rangle \). The cycle types are \((1,1)\) and \((2,)\).

\((x^2 - 2)(x^2 - 3)\) Here the roots are \( \alpha = \sqrt{2} \approx 1.4142 \), \( \beta = -\sqrt{2} \approx -1.4142 \), \( \gamma = \sqrt{3} \approx 1.732 \) and \( \delta = -\sqrt{3} \approx -1.732 \). Apart from the identity permutation, the only options are to exchange \( \alpha \) and \( \beta \), exchange \( \gamma \) and \( \delta \), or possibly both. Sending \( \alpha \) to \( \gamma \), say, is not possible as \( \alpha^2 = 2 \) but \( \gamma^2 = 3 \). Hence the Galois Group is \( C_2 \times C_2 = \langle (\alpha, \beta), (\gamma, \delta) \rangle \). The cycle types are \((1,1,1,1), (1,1,2), (2,2)\). Note that this group is not transitive.

\( x^4 + 1 \) Here the roots are \( \alpha = \frac{1 + i}{\sqrt{2}}, \beta = \frac{1 - i}{\sqrt{2}} = \frac{1}{\alpha} \), \( \gamma = -\alpha \) and \( \delta = -\beta \). One option is to send \( \alpha \) to \( \beta \), which replaces \( i \) by \( -i \), and is therefore the permutation \( (\alpha, \beta)(\gamma, \delta) \), and another is to replace \( \sqrt{2} \) by \( -\sqrt{2} \), which is the permutation \( (\alpha, \gamma)(\beta, \delta) \). Hence the Galois group contains, and is in fact generated by, these two permutations: it is often known as the Klein 4-group, or \( V \). The cycle types are \((1,1,1,1), (1,1,2), (2,2)\). This group is transitive, but every cycle shape in this group is a valid cycle shape in \( C_2 \times C_2 \) from the previous example, which is not transitive.

\( x^8 - 2 \) Here the roots are \( \alpha_k = \sqrt[4]{2}e^{2\pi ik/8}, k = 0..7 \). One option is to replace \( \sqrt[4]{2} \) by \( \sqrt[4]{2}e^{2\pi i/4} \), which cyclically permutes the roots, i.e. \( (\alpha_0, \alpha_1, \ldots, \alpha_7) \). Another option is to replace \( i \) by \( -i \), which corresponds to the permutation \( (\alpha_1, \alpha_7), (\alpha_2, \alpha_6), (\alpha_3, \alpha_5) \). In fact the Galois group is generated by these two, and is generally called the dihedral group, with 16 elements.

\( x^7 - 2 \) Here the roots are \( \alpha_k = \sqrt[7]{2}e^{2\pi ik/7}, k = 0..6 \). One option is to replace \( \sqrt[7]{2} \) by \( \sqrt[7]{2}e^{2\pi i/7} \), which cyclically permutes the roots, i.e. \( \pi_1 := \alpha_k \).
Another option is to replace \( i \) by \(-i\), which is the permutation \( \pi_2 := (\alpha_1, \alpha_6, (\alpha_2, \alpha_5), (\alpha_3, \alpha_4)) \). However, \( \pi_3 := (\alpha_1, \alpha_3, \alpha_2, \alpha_6, \alpha_6, \alpha_5) \) is in fact also a legal permutation, and the group is generated by \( \pi_1 \) and \( \pi_3 \) (since \( \pi_2 = \pi_3^3 \)) and has 42 elements.

**Proposition 58 ([vdW34])** For any \( n \), almost all polynomials of degree \( n \) have Galois group \( S_n \), or, more precisely, the probability of a polynomial having Galois group \( S_n \) is 1, i.e.

\[
\lim_{H \to \infty} \frac{|\{\text{polynomials of degree } n \text{ with coefficients } \leq H \text{ and group } S_n\}|}{|\{\text{polynomials of degree } n \text{ with coefficients } \leq H\}|} = 1.
\]

**Corollary 15** For any \( n \), almost all polynomials of degree \( n \) are irreducible, since \( S_n \) is transitive.

Hence, to make a statement about “almost all polynomials”, it suffices to consider those with Galois group \( S_n \).

The next question is how the factorisation shapes mod \( p \) relate to the Galois group, which is settled by the following result.

**Proposition 59 (Frobenius Density Theorem)** The density of prime numbers \( q \) for which \( f(x) \mod q \) has factorisation shape \( \{d_1, \ldots, d_k\} \) is equal to the density in the Galois group \( G \leq S_n \) of \( f(x) \) of elements of \( S_n \) with cycle type \( (d_1, \ldots, d_k) \).

Hence, if we assume the Galois group of \( f \) is \( S_n \), the factorisation shape of \( f(x) \mod q \) is distributed the same way as random permutations from \( S_n \), and the concept of subset generalises.

**Definition 84 (See Definition 82)** The sumset \( I(\sigma) \) of a permutation \( \sigma \) is the set of all sums of subsets of the cycle shape of the permutation.

**Proposition 60 ([PPR14, Theorem 1.5])** There is a constant \( b_4 \), independent of \( n \), such that

\[
\Pr_{S_n} (I(\sigma_1) \cap I(\sigma_2) \cap I(\sigma_3) \cap I(\sigma_4) = \{0, n\}) > b_4,
\]

where the probability is taken uniformly over quadruples of permutations. This means that, if the Galois group is \( S_n \), four primes have a nonzero probability of detecting transitivity (i.e. irreducibility) and in fact of detecting that the group is \( S_n \) ([DS00], [PPR14], Theorem 1.3 and discussion).

Empirically [PPR14, Figure 1], \( b_4 \approx 0.5 \). Similarly \( b_5 \) seems to be about 0.7 (for small \( n \), the probability is larger, which is presumably why [Mus78] recommends five primes), and \( b_7 > 0.9 \).

**Observation 15** Since determining that a polynomial is irreducible if one thinks it factors is much more expensive than modulo \( p \) factorisation, we should probably take seven primes, rather than the five of [Mus78].
5.3 Factoring modulo a prime

Throughout this section, we let $p$ be an odd prime, and $f$ a polynomial of degree $n$ which is square-free modulo $p$. There are three common methods for factoring $f$: two due to Berlekamp and one to Cantor & Zassenhaus, and various subsequent improvements, largely in terms of asymptotic complexity. A survey of this field is [vzGP01]. Most implementers of systems use (variants of) the Cantor–Zassenhaus algorithm, for reasons explained at the start of Section 5.7.

5.3.1 Berlekamp’s small $p$ method

This is due to [Ber67]. We first need to state some facts about arithmetic modulo a prime $p$.

Proposition 61 If $a$ and $b$ are two integers modulo $p$, then $(a + b)^p \equiv a^p + b^p$.

If we expand $(a + b)^p$ by the Binomial Theorem, all intermediate coefficients are divisible by $p$.

Proposition 62 $a^p \equiv a$ modulo $p$.

Corollary 16 Every integer modulo $p$ is a root of $x^p - x$, and therefore

$$x^p - x = (x - 0)(x - 1)\ldots(x - (p - 1)).$$

These facts extend to polynomials.

Proposition 63 Let $a(x)$ be a polynomial, then $a(x)^p \equiv a(x^p)$ modulo $p$.

Let us now suppose that $f$ factorises into $r$ irreducible polynomials:

$$f(x) = f_1(x)f_2(x)\ldots f_r(x)$$

($r$ is unknown for the present). Since $f$ has no multiple factors, the $f_i$ are relatively prime. Let $s_1, \ldots, s_r$ be integers modulo $p$. By the Chinese Remainder Theorem (Algorithm 47) there is a polynomial $v$ such that

$$v \equiv s_i \pmod{p, f_i(x)}, \quad (5.1)$$

and moreover, the degree of $v$ is less than that of the product of the $f_i$, that is $f$. Such a polynomial $v$ is useful, for if $s_i \neq s_j$, then $\gcd(f, v - s_i)$ is divisible by $f_i$, but not by $f_j$, and therefore leads to a decomposition of $f$. We have the following relation:

$$v(x)^p \equiv s_j^p \equiv s_j \equiv v(x) \pmod{f_j(x), p},$$

\footnote{It is possible to generalise these methods to the case $p = 2$, but the cost, combined with the unlikelihood of 2 being a good prime, means that computer algebraists (unlike cryptographers) rarely do so.}
and, by the Chinese remainder theorem,
\[ v(x)^p \equiv v(x) \pmod{f(x), p}. \]  \hspace{.5cm} (5.2)

But, on replacing \( x \) by \( v(x) \) in Corollary 16
\[ v(x)^p - v(x) \equiv (v(x) - 0)(v(x) - 1)\ldots(v(x) - (p - 1)) \pmod{p}. \]  \hspace{.5cm} (5.3)

Thus, if \( v(x) \) satisfies (5.2), \( f(x) \) divides the left hand side of (5.3), and each of its irreducible factors, the \( f_i \), divides one of the polynomials on the right hand side of (5.3). But this implies that \( v \) is equivalent to an integer modulo \( f_i \), that is that \( v \) satisfies (5.1). We have proved (by Knuth’s method \([\text{Knu81, p. 422}]\)) the following result.

**Theorem 40** The solutions \( v \) of (5.1) are precisely the solutions of (5.2).

We have already said that the solutions of (5.1) provide information about the factorisation of \( f \), but we still have the problem of finding them. Berlekamp’s basic idea is to note that (5.2) is a linear equation for the coefficients of \( v \). This remark may seem strange, but it is a consequence of Proposition 63. In fact, if \( n \) is the degree of \( f \), let us consider the matrix

\[
Q = \begin{pmatrix}
q_0,0 & q_0,1 & \cdots & q_0,n-1 \\
q_1,0 & q_1,1 & \cdots & q_1,n-1 \\
& & \ddots & \\
q_{n-1,0} & q_{n-1,1} & \cdots & q_{n-1,n-1}
\end{pmatrix},
\]

where
\[ x^{pk} \equiv q_{k,n-1}x^{n-1} + \cdots + q_{k,1}x + q_{k,0} \pmod{f(x), p}. \]

If we consider a polynomial as a vector of its coefficients, multiplication by \( Q \) corresponds to the calculation of the \( p \)-th power of the polynomial. The solutions of (5.2) are thus the eigenvectors of the matrix \( Q \pmod{p} \) for the eigenvalue 1. Hence the algorithm in Figure 5.2. Determining \( r \), the number of irreducible factors, takes time \( O(n^3) \) (with classical linear algebra, and assuming that \( p \) is single-word size). The running time of the full algorithm is \( O(n^3 + prn^2) \), and the average value of \( r \) is \( \ln n \). This is very fast if \( p \) is small, but may be expensive if \( p \) is not small. We note that the algorithm is completely deterministic.

### 5.3.2 The Cantor–Zassenhaus method

This method, a combination of Algorithms 28 and 29, is generally attributed to \([\text{CZ81}]\). It is based on the following generalization of Fermat’s (Little) Theorem/Corollary 16.

\footnote{Though \([\text{Zip93}]\) credits algorithm 28 to \([\text{Arw18}]\), and \([\text{vzGP01}]\) to Gauß.}
Algorithm 27

Input: Prime $p$ and $f$ a squarefree (mod $p$) univariate polynomial

Output: The irreducible factors of $f$

Calculate $Q$
Calculate (a basis for) the eigenvectors of $Q$ for the eigenvalue 1,
  i.e. (a basis for) the solutions $v$ of $(Q - I)v = 0$.

$\#(1,0,\ldots,0)^T$ is the trivial eigenvector, as integers solve (5.2)

if $r = 1$
  then return $\{f\}$
$S = \emptyset$
for $v$ a non-trivial eigenvector
  Let $g$ be the polynomial whose coefficients are $v$
  for $s \in \{0,\ldots,p-1\}$
    if $(h := \gcd(g-s,f)) \neq 1$
      then $S := S \cup \{h\}$
    if $|S| = r$
      then return $S$

Proposition 64 ([LN97, Theorem 3.20]) All irreducible polynomials of degree $d$ with coefficients modulo $p$ divide $x^{p^d} - x$, and in fact

$$x^{p^d} - x = \prod_{e|d} \prod_{f : \deg(f) = e}^* f,$$

where $\prod^*$ means that we only take irreducible polynomials.

Corollary 17 All irreducible polynomials of degree $d$ with coefficients modulo $p$ divide $x^{p^d} - 1$, except for $x$ itself in the case $d = 1$. Furthermore, no irreducible polynomials of degree more than $d$ divide $x^{p^d} - 1$.

Corollary 18 Half of the irreducible polynomials of degree $d$ (except for $x$ itself in the case $d = 1$) with coefficients modulo $p$ divide $(x-a)^{(p^d-1)/2} - 1$, for any $a$ (but a different 50%, depending on $a$).

Hence we deduce Algorithm 28 which splits a square-free polynomial $f$ as $\prod f_i$, where each $f_i$ is the product of irreducibles of degree $i$.

If $f_i$ has degree $i$, then it is clearly irreducible: otherwise we have to split it. This is the purpose of Algorithm 29 which relies on a generalization of Corollary 18.

Proposition 65 ([CZ81, p. 589]) Let $f$ be a product of $r > 1$ irreducible polynomials of degree $d$ modulo $p$, and $g$ a random (non-constant) polynomial
5.4. FROM $\mathbb{Z}_p$ TO $\mathbb{Z}$?

Figure 5.3: Algorithm 28 Distinct Degree Factorization

Algorithm 28 (Distinct Degree Factorization)
Input: $f(x)$ a square-free polynomial modulo $p$, not divisible by $x$; a prime $p$
Output: A decomposition $f = \prod f_i$, where each $f_i$ is the product of irreducibles of degree $i$.

$i := 1$
while $2i \leq \deg(f)$
    
    $g := x^{p^i - 1} \pmod{f}$
    
    $f_i := \gcd(g - 1, f)$
    
    $f := f/f_i$
    
    $i := i + 1$

if $f \neq 1$
    then $f_{\deg(f)} := f$

Note that the computation in line (*) should be done by the repeated squaring method, reducing modulo $f$ at each stage. We can save time in practice by reusing the previous $g$. Various improvements (at least in asymptotic complexity) are given in [vzGS92].

of degree $< d$. Then the probability that $\gcd(g^{(p^d - 1)/2} - 1, f)$ is either 1 or $f$ is at most $2^{1-r}$.

Proposition 66 In classical arithmetic, the running time of the Cantor–Zassenhaus Algorithm (i.e. Algorithm 28 followed by Algorithm 29) is $O(d^3 \log p)$, where $d$ is the degree of the polynomial being factored.

We do $O(d \log p)$ operations on polynomials of degree $d$, where the factor $\log p$ comes from the $x^{p^i} \pmod{f}$ computations. We note that Algorithm 28 is deterministic, but Algorithm 29 is probabilistic.

5.3.3 Berlekamp’s large $p$ method
This method is due to [Ber70]. It has an alternative, matrix-based, form of Algorithm 28 again deterministic, and a probabilistic equivalent of Algorithm 29. We do not go further into it here.

5.4 From $\mathbb{Z}_p$ to $\mathbb{Z}$?

Now that we know that factoring over the integers modulo $p$ is possible, the obvious strategy for factoring polynomials over the integers would seem to be to follow one of algorithms 17 or 18. This would depend on having ‘good’ reduction, which one would naturally define as follows.
Algorithm 29 (Split a Distinct Degree Factorization)

**Input:** A prime $p$, a degree $d$ and a polynomial $f(x) \pmod{p}$ known to be the product of irreducibles of degree $d$ (and not divisible by $x$)

**Output:** The factorization of $f$ (modulo $p$) into irreducibles of degree $d$.

```plaintext
if $d = \deg(f)$
    then return $f$  # degree $d$ so is irreducible
W := \{f\}
ans := \emptyset
while W \neq \emptyset  # factors of degree $d$ found
    h := RandomPoly(p,d)
    h := h^{(p^d-1)/2} \pmod{g}  (*)
    V := \emptyset  # list of polynomials to be processed
    for $g \in W$
        $h_1 := \gcd(h-1,g)$
        if $\deg(h_1) = 0 \lor \deg(h_1) = \deg(g)$
            then V := V \cup \{g\}  # we made no progress
        else process($h_1$)
            process($g/h_1$)
    W := V
return ans
```

Here `RandomPoly(p,d)` returns a random non-constant polynomial modulo $p$ of degree less than $d$, and the sub-function `process(g1)` takes a polynomial $g_1$ which is a result of splitting $g$ by $h_1$, and adds it to `ans` if it has degree $d$, otherwise adds it to `V`. Again (*) should be done by repeated squaring, reducing modulo $g$ every time.

**Definition 85 (Optimistic)** We say that $p$ is of good reduction for the factorization of the polynomial $f$ if the degrees of the irreducible factors of $f$ (modulo $p$) are the same as those of the factors of $f$.

And indeed, if we can find $p$ of good reduction, then we could factorize $f$. Unfortunately these are rare, and possibly non-existent.

**Corollary 19 (of Proposition 59)** If $f$ is a generic (formally speaking, with Galois group $S_n$) polynomial of degree $n$, the probability of its remaining irreducible modulo $p$ is $1/n$.

Nevertheless, we could hope that we can piece together results from several primes to get information. For example, $x^5 + x + 3$ factors into irreducibles as

$$(x^3 + 7x^2 + 3x + 7)(x^2 + 4x + 2) \pmod{11}$$
(and therefore does not have linear factors over \( \mathbb{Z} \)) and as
\[
(x^4 + 5x^3 + 12x^2 + 8x + 2) (x + 8) \mod 13
\]
(therefore any factorization has a linear factor) and hence must be irreducible
over the integers. This test is described in Section 5.2.1. However, there are
irreducible polynomials which can never be proved so this way.

**Example 18** The polynomial \( x^4 + 1 \), which is irreducible over the integers,
factors into two quadratics (and possibly further) modulo every prime \( p = 2 \) Then \( x^4 + 1 = (x + 1)^4 \).

\( p = 4k + 1 \) In this case, \(-1\) is always a square, say \(-1 = q^2\). This gives us the
factorisation \( x^4 + 1 = (x^2 - q)(x^2 + q) \).

\( p = 8k \pm 1 \) In this case, \( 2 \) is always a square, say \( 2 = q^2 \). This gives us the
factorisation \( x^4 + 1 = (x^2 - (2/q)x + 1)(x^2 + (2/q)x + 1) \). In the case \( p = 8k + 1 \), we have this factorisation
and the factorisation given in the previous case. As these two factorisations are not equal, we can calculate
the g.c.d.s of the factors, in order to find a factorisation as the product of
four linear factors.

\( p = 8k + 3 \) In this case, \(-2\) is always a square, say \(-2 = q^2\). This is a result
of the fact that \(-1\) and \( 2 \) are not squares, and so their product must
be a square. This property of \(-2\) gives us the factorisation \( x^4 + 1 =
(x^2 - (2/q)x - 1)(x^2 + (2/q)x - 1) \).

This polynomial is not an isolated oddity: [SD70] and [KMS83] proved that there
are whole families of polynomials with this property of being irreducible, but of
factorising compatibly modulo every prime, and indeed compatibly into many
quadratics. Several people have said that these polynomials are, nevertheless,
“quite rare”, which is true if one takes polynomials at random, but [ABD85]
showed that they can often occur in the manipulation of algebraic numbers.

Even if we have factorizations modulo several primes, a further problem
arises, which we will illustrate with the example of \( x^4 + 3 \). This factors as
\[
x^4 + 3 = (x^2 + 2) (x + 4) (x + 3) \mod 7
\]
and
\[
x^4 + 3 = (x^2 + x + 6) (x^2 + 10x + 6) \mod 11. \tag{5.5}
\]
In view of the second factorization, the first has too much decomposition, and
we need only consider the split
\[
x^4 + 3 = (x^2 + 2) (x^2 + 5) \mod 7, \tag{5.6}
\]
obtained by combining the two linear factors.

\[\text{Footnote 5: The Galois group of this polynomial is discussed on page 192}\]
When we come to combine results modulo these two primes by Chinese Remainder Theorem (Theorem 47) to deduce a congruence modulo 77, we have a dilemma: do we pair \((x^2 + x + 6)\) with \((x^2 + 2)\) or \((x^2 + 5)\)? Both seem feasible.

In fact, both are feasible. The first pairing gives

\[
x^4 + 3 = (x^2 + 56x + 72)(x^2 - 56x - 16) \mod 77,
\]

and the second gives

\[
x^4 + 3 = (x^2 + 56x + 61)(x^2 - 56x - 5) \mod 77:
\]

both of which are correct. The difficulty in this case, as in general, is that, while polynomials over \(\mathbb{Z}_{7}\) have unique factorization, as do those over \(\mathbb{Z}_{11}\) (and indeed modulo any prime), polynomials over \(\mathbb{Z}_{77}\) (or any product of primes) do not, as (5.7) and (5.8) demonstrate.

### 5.5 Hensel Lifting

Our attempts to use the Chinese Remainder Theorem seem doomed: we need a different solution, which is provided by what mathematicians call \(p\)-adic methods, and computer algebraists call Hensel Lifting. This is the topic of the next three subsections.

#### 5.5.1 Linear Hensel Lifting

This is the simplest implementation of the phase described as ‘grow’ in Figure 5.1 we grow incrementally \(\mathbb{Z}_p \rightarrow \mathbb{Z}_{p^2} \rightarrow \mathbb{Z}_{p^3} \rightarrow \cdots \rightarrow \mathbb{Z}_{p^m}\).

For simplicity, we consider first the case of a monic polynomial \(f\), which factorizes modulo \(p\) as \(f = gh\), where \(g\) and \(h\) are relatively prime (which implies that \(f \mod p\) is square-free, that is, that \(p\) does not divide the resultant of \(f\) and \(f'\)). We use parenthesized superscripts, as in \(g^{(1)}\), to indicate the power of \(p\) modulo which an object has been calculated. Thus our factorization can be written \(f^{(1)} = g^{(1)}h^{(1)} \mod p\) and our aim is to calculate a corresponding factorization \(f^{(k)} = g^{(k)}h^{(k)} \mod p^k\) such that \(p^k\) is sufficiently large.

Obviously, \(g^{(2)} \equiv g^{(1)} \mod p\), and therefore we can write \(g^{(2)} = g^{(1)} + p\hat{g}^{(2)}\) where \(\hat{g}^{(2)}\) is a measure of the difference between \(g^{(1)}\) and \(g^{(2)}\). The same holds for \(f\) and \(h\), so that \(f^{(2)} = g^{(2)}h^{(2)} \mod p^2\) becomes

\[
f^{(1)} + p\hat{f}^{(2)} = (g^{(1)} + p\hat{g}^{(2)})(h^{(1)} + p\hat{h}^{(2)}) \mod p^2.
\]

Since \(f^{(1)} = g^{(1)}h^{(1)} \mod p\), this equation can be rewritten in the form

\[
\frac{f^{(1)} - g^{(1)}h^{(1)}}{p} + \hat{f}^{(2)} = \hat{g}^{(2)}h^{(1)} + \hat{h}^{(2)}g^{(1)} \mod p.
\]

The left hand side of this equation is known, whereas the right hand side depends linearly on the unknowns \(\hat{g}^{(2)}\) and \(\hat{h}^{(2)}\). Applying the extended Euclidean
Algorithm (5) to \(g^{(1)}\) and \(h^{(1)}\), which are relatively prime, we can find polynomials \(\hat{g}^{(2)}\) and \(\hat{h}^{(2)}\) of degree less than \(g^{(1)}\) and \(h^{(1)}\) respectively, which satisfy this equation modulo \(p\). The restrictions on the degrees of \(\hat{g}^{(2)}\) and \(\hat{h}^{(2)}\) are valid in the present case, for the leading coefficients of \(g^{(k)}\) and \(h^{(k)}\) have to be 1. Thus we can determine \(g^{(2)}\) and \(h^{(2)}\).

Similarly, \(g^{(3)} \equiv g^{(2)} \pmod{p^2}\), and we can therefore write \(g^{(3)} = g^{(2)} + p^2\hat{g}^{(3)}\) where \(\hat{g}^{(3)}\) is a measure of the difference between \(g^{(2)}\) and \(g^{(3)}\). The same is true for \(f\) and \(h\), so that \(f^{(3)} = g^{(3)}h^{(3)} \pmod{p^3}\) becomes

\[
\frac{f^{(2)} - g^{(2)}h^{(2)}}{p^2} + \hat{f}^{(3)} = \hat{g}^{(3)}h^{(2)} + \hat{h}^{(3)}g^{(2)} \pmod{p},
\]

(5.10)

Moreover, \(g^{(2)} \equiv g^{(1)} \pmod{p}\), so this equation simplifies to

\[
\frac{f^{(2)} - g^{(2)}h^{(2)}}{p^2} + \hat{f}^{(3)} = \hat{g}^{(3)}h^{(1)} + \hat{h}^{(3)}g^{(1)} \pmod{p}.
\]

The left hand side of this equation is known, whilst the right hand side depends linearly on the unknowns \(\hat{g}^{(3)}\) and \(\hat{h}^{(3)}\). Applying the extended Euclidean algorithm to \(g^{(1)}\) and \(h^{(1)}\), which are relatively prime, we can find the polynomials \(\hat{g}^{(3)}\) and \(\hat{h}^{(3)}\) of degrees less than those of \(g^{(1)}\) and \(h^{(1)}\) respectively, which satisfy this equation modulo \(p\). Thus we determine \(g^{(3)}\) and \(h^{(3)}\) starting from \(g^{(2)}\) and \(h^{(2)}\), and we can continue these deductions in the same way for every power \(p^k\) of \(p\) until \(p^k\) is sufficiently large.

We should note that Euclid’s algorithm is always applied to the same polynomials, and therefore it suffices to perform it once. In fact, we can state the algorithm in Figure 5.5.

To solve the more general problem of lifting a factorization of a non-monic polynomial, we adopt the same solution as in the g.c.d. case: we impose the leading coefficient in each factor to be the leading coefficient of the original polynomial (which we may as well assume to be primitive). We also address the problem of lifting \(n\) (rather than just 2) factors in Algorithm [31]

5.5.2 Quadratic Hensel Lifting

This is an alternative implementation of the phase described as ‘grow’ in Figure 5.1, we grow incrementally \(\mathbb{Z}_p \rightarrow \mathbb{Z}_{p^2} \rightarrow \mathbb{Z}_{p^4} \rightarrow \cdots \rightarrow \mathbb{Z}_{p^{2m}}\), squaring the modulus each time.

As in the previous section, we first consider the case of a monic polynomial with two factors. The lifting from \(p\) to \(p^2\) proceeds exactly as in the previous section, and equation (5.9) is still valid. The lifting from \(p^2\) to \(p^4\) is similar to the lifting from \(p^2\) to \(p^3\) in the previous section, and the analogue of equation
Algorithm 30 (Hensel Lifting (Linear Two Factor version))

Input: \( f, g^{(1)}, h^{(1)}, p, k \) with \( f \) monic and \( \equiv g^{(1)} h^{(1)} \pmod{p} \)

Output: \( g^{(k)}, h^{(k)} \) with \( f \equiv g^{(k)} h^{(k)} \pmod{p^k} \)

\[
\begin{align*}
g & := g^{(1)} \\
h & := h^{(1)} \\
g^{(r)}, h^{(r)} & := \text{Algorithm}\ 30(g^{(1)}, h^{(1)})\ \text{in } \mathbb{Z}_p[x] \\
\text{for } i := 2 \ldots k & \\
\quad \Delta & := f - g h \pmod{p^i} \\
\quad g^{(c)} & := \Delta \ast h^{(r)} \pmod{(p, g^{(1)})} \\
\quad h^{(c)} & := \Delta \ast g^{(r)} \pmod{(p, h^{(1)})} \\
\quad g^{(c)} & := g + p^{i-1} g^{(c)} \\
\quad h^{(c)} & := h + p^{i-1} h^{(c)} \\
\text{return } (g, h)
\end{align*}
\]

Algorithm 31 (Univariate Hensel Lifting (Linear version))

Input: \( f, g^{(1)}, \ldots, g^{(1)}_n, p, k \) with \( f \in \mathbb{Z}[x] \) primitive and \( \equiv \prod g^{(1)}_i \pmod{p} \)

Output: \( g^{(k)}_1, \ldots, g^{(k)}_n \) with \( f \equiv \prod g^{(k)}_i \pmod{p^k} \)

\[
\begin{align*}
\text{for } j := 1 \ldots n & \\
\quad g^{(1)}_j & := \frac{\text{lc}(f)}{\text{lc}(g^{(1)}_j)} g^{(1)}_j \\
F & := \text{lc}(f)^{n-1} f \quad \# \text{leading coefficients imposed} \\
\text{for } j := 1 \ldots n & \\
\quad g^{(r)}_j, h^{(r)}_j & := \text{Algorithm}\ 30(g^{(1)}_j, \prod_{i \neq j} g^{(1)}_i)\ \text{in } \mathbb{Z}_p[x] \\
\text{for } i := 2 \ldots k & \\
\quad \text{if } F = \prod g_j & \\
\quad \quad \text{then break} \quad \# \text{True factorization discovered} \\
\quad \Delta & := F - \prod g_j \pmod{p^i} \\
\quad \text{for } j := 1 \ldots n & \\
\quad \quad g^{(c)}_j & := \Delta \ast h^{(r)}_j \pmod{(p, g^{(1)}_j)} \\
\quad \quad g_j & := g_j + p^{i-1} g^{(c)}_j \\
\text{for } j := 1 \ldots n & \\
\quad g_j & := \text{pp}(g_j) \quad \# \text{undo the imposed leading coefficients} \\
\text{return } (g_1, \ldots, g_n) &
\end{align*}
\]
Algorithm 32 (Univariate Hensel Lifting (Quadratic Two Factor version))

Input: \(f, g^{(1)}, h^{(1)}, p, k\) with \(f\) monic and \(\equiv g^{(1)}h^{(1)} \pmod p\)

Output: \(g, h\) with \(f \equiv gh \pmod {p^2}\)

\[
\begin{align*}
g &:= g^{(1)} \\
h &:= h^{(1)} \\
\text{for } i := 1 \ldots k & \quad \#f \equiv gh \pmod {p^{2^i-1}} \\
g^{(r)}, h^{(r)} &:= \text{Algorithm 5}(g, h) \text{ in } \mathbb{Z}_{p^{2^i-1}}[x] \\
\Delta &:= f - gh \pmod {p^{2^i-1}} \\
g^{(c)} &:= \Delta \ast h^{(r)} \pmod {p^{2^i-1}, g^{(i)}} \\
h^{(c)} &:= \Delta \ast g^{(r)} \pmod {p^{2^i-1}, h^{(i)}} \\
g &:= g + p^{2^i-1}g^{(c)} \\
h &:= h + p^{2^i-1}h^{(c)} \\
\#f \equiv gh \pmod {p^{2^i}} \\
\text{return } (g, h)
\end{align*}
\]

(5.10) is the following

\[
\frac{f^{(2)} - g^{(2)}h^{(2)}}{p^2} + f^{(3)}h^{(2)} + g^{(3)}h^{(2)} \equiv \tilde{f}^{(3)}h^{(2)} + \tilde{g}^{(3)}h^{(2)} \pmod {p^2}. \tag{5.11}
\]

The difference is that this equation is modulo \(p^2\) rather than modulo \(p\), and hence the inverses have to be recomputed, rather than being the same, as (5.10) can re-use the inverses from (5.9). We give the corresponding Algorithm 32 in Figure 5.7, an analogue of Algorithm 30, except that the call to Algorithm 5 is inside the loop, rather than preceding it.

Equally, we can give the general Algorithm 33 in Figure 5.8 as an equivalent of Algorithm 31. Again, the Extended Euclidean Algorithm is in the innermost loop, and practical experience confirms that this is indeed the most expensive step. It is also the case that we are repeating work, inasmuch as the calculations are the same as before, except that the inputs are correct to a higher power of \(p\) than before, and the results are required to a higher power of \(p\) than before.

5.5.3 Hybrid Hensel Lifting

Further details, and a “balanced factor tree” approach, are given in \textit{vzGG99} \S 15.5.
Algorithm 33 (Univariate Hensel Lifting (Quadratic version))

Input: $f, g^{(1)}_1, \ldots, g^{(1)}_n, p, k$ with $f$ primitive and $\equiv \prod g^{(1)}_i \pmod{p}$

Output: $g_1, \ldots, g_n$ with $f \equiv \prod g_i \pmod{p^k}$

for $j := 1 \ldots n$
\[ g^{(1)}_j := \frac{\text{lcm}(f)}{\text{lcm}(g^{(1)}_j)} g^{(1)}_j \]
\[ F := \text{lcm}(f)^{n-1} f \quad \# \text{leading coefficients imposed} \]
for $i := 1 \ldots k$
\[ \Delta := \frac{F - \prod g_j \pmod{p^{2^{i-1}}}}{p^{2^{i-1}}} \]
for $j := 1 \ldots n$
\[ g^{(r)}_j, h^{(r)}_j := \text{Algorithm 33}(g_j, \prod_{i \neq j} g_i) \text{ in } \mathbb{Z}_{p^{2^{i-1}}} [x] \]
\[ g^{(c)}_j := \Delta \ast h^{(r)}_j \pmod{(p^{2^{i-1}}, g_j)} \]
\[ g_j := g_j + p^{2^{i-1}} g^{(c)}_j \]
\[ \#F = \prod g_j \pmod{p^2} \]
for $j := 1 \ldots n$
\[ g_j := \text{pp}(g_j) \quad \# \text{undo the imposed leading coefficients} \]
return $(g_1, \ldots, g_n)$
5.6. THE RECOMBINATION PROBLEM

However, as pointed out in section 5.4, the fact that we have a factorization of \( f \) modulo \( p^k \), where \( p^k > 2M \), \( M \) being the Landau–Mignotte bound (Theorem 33), or some alternative (see page 272) on the size of any coefficients in any factor of \( f \), does not solve our factorization problem. It is perfectly possible that \( f \) is irreducible, or that \( f \) does factor, but less than it does modulo \( p \).

Assuming always that \( p \) does not divide \( \text{lcm}(f) \), all that can happen when we reduce \( f \) modulo \( p \) is that a polynomial that was irreducible over the integers now factors modulo \( p \) (and hence modulo \( p^k \)). This gives us the following algorithm, first suggested in [Zas69].

**Algorithm 34 (Combine Modular Factors)**

**Input:** A prime power \( p^k \), a monic polynomial \( f(x) \) over the integers, a factorisation \( f = \prod_{i=1}^{r} f_i \) modulo \( p^k \)

**Output:** The factorization of \( f \) into monic irreducible polynomials over the integers.

\[
\begin{align*}
\text{ans} &:= \emptyset \\
M &:= \text{LandauMignotteBound}(f) \\
S &:= \{f_1, \ldots, f_r\} \\
\text{for subsets } T \text{ of } S \\
& \quad h := \prod_{g \in T} g \pmod{p^k} \\
& \quad \text{if } |h| < M \text{ and } h \text{ divides } f \\
& \quad \quad \text{then } \text{ans} := \text{ans} \cup \{h\} \\
& \quad \quad f := f/h \\
& \quad \quad S := S \setminus T \\
& \quad \quad M := \min(M, \text{LandauMignotteBound}(f))
\end{align*}
\]

To guarantee irreducibility of the factors found, the loop must try all subsets of \( T \) before trying \( T \) itself, but this still leaves a lot of choices for the loop: see [1] below.

The running time of this algorithm is, in the worst case, exponential in \( r \) since \( 2^{r-1} \) subsets of \( S \) have to be considered (\( 2^{r-1} \) since considering \( T \) effectively also considers \( S \setminus T \)). Let \( n \) be the degree of \( f \), and \( H \) a bound on the coefficients of \( f \), so the Landau–Mignotte bound is at most \( 2^n(n+1)H \), and \( k \leq \log_p(2^{n+1}(n+1)H) \).

Many improvements to, or alternatives to, this basic algorithm have been suggested since. In essentially chronological order, the most significant ones are as follows.

1. [Col79] pointed out that there were two obvious ways to code the “for subsets \( T \) of \( S \)” loop: increasing cardinality of \( T \) and increasing degree of \( \prod_{g \in T} g \). He showed that, subject to two very plausible conjectures, the average number of products actually formed with the cardinality ordering was \( O(n^2) \), thus the average running time would be polynomial in \( n \).
2. [LLJS82] had a completely different approach to algorithm 34. They asked, for each \( d < n \), “given \( f_1 \in S \), what is the polynomial \( g \) of degree \( d \) which divides \( f \) over the integers and is divisible by \( f_1 \) modulo \( p^k \)?”. Unfortunately, answering this question needed a \( k \) far larger than that implied by the Landau–Mignotte bound, and the complexity, while polynomial in \( n \), was \( O(n^{12}) \), at least while using classical arithmetic. This paper introduced the ‘LLL’ lattice reduction algorithm, which has many applications in computer algebra and far beyond.

3. [ABDS85] showed that, by a combination of simple divisibility tests and “early abort” trial division (Proposition 54) it was possible to make dramatic reductions, at the time up to four orders of magnitude, in the constant implied in the statement “exponential in \( r \”).

4. [ASZ00] much improved this, and the authors were able to eliminate whole swathes of possible \( T \) at one go.

5. [vH02] reduces the problem of finding \( T \) to a ‘knapsack’ problem, which, as in method 2, is solved by LLL, but the lattices involved are much smaller — of dimension \( r \) rather than \( n \). At the time of writing, this seems to be the best known method. His paper quoted a polynomial of degree \( n = 180 \), with \( r = 36 \) factors of degree 5 modulo \( p = 19 \), but factoring as two polynomials of degree 90 over the integers. This took 152 seconds to factor.

Open Problem 23 (Evaluate [vH02] against [ASZ00]) How does the factorization algorithm of [vH02] perform on the large factorizations successfully solved by [ASZ00]? Clearly the algorithm of [vH02] is asymptotically faster, but where is the cut-off point? Note also that [vH02]’s example of \( x^{128} - x^{112} + x^{80} - x^{64} + x^{48} - x^{16} + 1 \) is in fact a disguised cyclotomic polynomial (as shown by the methods of [BD89]), being

\[
\frac{(x^{240} - 1)}{(x^{80} - 1)(x^{48} - 1)} = \prod_{1 \leq k < 15, \gcd(k, 15) = 1} \left( x^{16} - e^{2\pi i k/15} \right).
\]

5.7 Univariate Factoring Solved

We can put together the components we have seen to deduce an algorithm (Figure 5.9) for factoring square-free polynomials over \( \mathbb{Z}[x] \). We have chosen to use the Cantor–Zassenhaus method (Section 5.3.2) for the modular factorisation, as most implementers today do, simply because it gives the information needed for the Musser test.

Open Problem 24 (Better Choice of ‘Best’ Prime) In algorithm 35 we picked the ‘best’ prime and the corresponding factorization. In principle, it is
5.7. UNIVARIATE FACTORING SOLVED

Figure 5.9: Overview of Factoring Algorithm

Algorithm 35 (Factor over \( \mathbb{Z} \))

**Input:** A primitive square-free \( f(x) \in \mathbb{Z}[x] \)

**Output:** The factorization of \( f \) into irreducible polynomials over the integers.

\[
p_1 := \text{find\_prime}(f);
F_1 := \text{Corollary}[\text{17}](f, p_1)
\]

if \( F_1 \) is a singleton

\[
\text{return } f
\]

\[
S := \text{AllowableDegrees}(F_1)
\]

for \( i := 2, \ldots, 7 \)  \#7 from Observation[\text{15}]

\[
p_i := \text{find\_prime}(f);
F_i := \text{Corollary}[\text{17}](f, p_i)
S := S \cap \text{AllowableDegrees}(F_i)
\]

if \( S = \emptyset \)

\[
\text{return } f
\]

\[
(p, F) := \text{best}(\{(p_i, F_i)\}); \quad \#'\text{best'} \text{ in terms of fewest factors}
\]

Complete factorization modulo \( p \) by Algorithm[\text{29}]

\[
F := \text{Algorithm}[\text{31}](f, F, p, \log_p(2\text{LM}(f)))
\]

**Return** Algorithm[\text{34}](p^{\log_p(2\text{LM}(f))}, f, F)

\text{find\_prime}(f) \text{ returns a prime } p \text{ such that } f \text{ remains square-free modulo } p.

\text{AllowableDegrees}(F) \text{ returns the set of allowable proper factor degrees from a distinct degree (not needing a complete) modular factorization.}

Algorithm[\text{34}] \text{ can be replaced by any of improvements [1,5] on pages 205–206.}
possible to do better, as in the following example, but the author has seen no systematic treatment of this.

**Example 19** Suppose $f_p$ factors as polynomials of degree 2, 5 and 5, and $f_q$ factors as polynomials of degree 3, 4 and 5. The factorization modulo $p$ implies that the allowable degrees of (proper) factors are 2, 5, 7 and 10. The factorization modulo $q$ implies that the allowable degrees of (proper) factors are 3, 4, 5, 7, 8 and 9. Hence the only possible degrees of proper factors over the integers are 5 and 7. The factorization modulo $p$ can yield this in two ways, but the factorization modulo $q$ can only yield this by combining the factors of degrees 3 and 4. Hence we should do this, and lift the factorization of $f_q$.

**Open Problem 25 (Low-degree Factorization)** Although no system to the author’s knowledge implements it, it would be possible to implement an efficient procedure to find all factors of limited total degree $d$. This would be efficient for three reasons:

- the Cantor–Zassenhaus algorithm (section 5.3.2) need only run up to maximum degree $d$;
- it should be possible to use smaller bounds on the lifting
- The potentially-exponential recombination process need only run up to total degree $d$. In particular, if $d = 1$, no recombination is needed.

At the moment, the user who only wants low-degree factors has to either program such a search, or rely on the system’s `factor`, which may waste large amounts of time looking for high-degree factors, which the user does not want.

### 5.8 Multivariate Factoring

Just as in section 4.3, we can use “evaluating $y$ at the value $v$”, i.e. working modulo $(y - v)$, as an analogue of working modulo $p$. Just as in section 4.3, having worked modulo $(y - v)$, we can lift this to $(y - v)^2$ and so on. There is a fairly obvious generalisation of Algorithm 31, given in Algorithm 36, where we are given $f(x, y, x_1, \ldots, x_m) \in \mathbb{Z}[x_1, \ldots, x_m, y, x] = R[y, x]$, a value $v \in \mathbb{Z}$ and the factorization of $f(x, v, x_1, \ldots, x_m) \in R[x]$, and we wish to lift this to a factorization modulo $(y - v)^k$. In practice, $k = 1 + \deg_y(f)$, for then this lifted factorization should be the true factorization in $R[y, x]$.

This is linear lifting: there are also analogies of quadratic lifting (Algorithm 33) and hybrid lifting (section 5.5.3), but in practice these are rarely implemented, as in the multivariate setting most of the cost is in the last lift, as the polynomials have more and more terms as we lift.

#### 5.8.1 A “Good Reduction” Complexity Result

Of course, it is possible that $v$ is a “bad reduction” for $f$, in that $f(x, v, x_1, \ldots, x_n)$ factors more than $f(x, y, x_1, \ldots, x_n)$ does. Unlike the modulo $p$ case, where we
Algorithm 36 (Multiivariate Hensel Lifting (Linear version))

Input: \( f, g_1^{(1)}, \ldots, g_n^{(1)}, v, k \) with \( f \in R[y][x] \) primitive and \( \equiv \prod g_i^{(1)} \pmod{(y-v)} \)

Output: \( g_1^{(k)}, \ldots, g_n^{(k)} \) with \( f \equiv \prod g_i^{(k)} \pmod{(y-v)^k} \)

for \( j := 1 \ldots n \)
\[ g_j^{(1)} := \frac{\text{lc}(f)}{\text{lc}(g_j^{(1)})} g_j^{(1)} \]
\[ F := \text{lc}(f)^{n-1} f \quad \# \text{leading coefficients imposed} \]
for \( j := 1 \ldots n \)
\[ g_j^{(r)}, h_j^{(r)} := \text{Algorithm 5} \left(g_j^{(1)}, \prod_{i \neq j} g_i^{(1)} \right) \text{in } R[x] \]
for \( i := 2 \ldots k \)
\[ \text{if } F = \prod_j g_j \]
\[ \text{then break} \quad \# \text{True factorization discovered} \]
\[ \Delta := \frac{F - \prod_j g_j}{(y-v)^{i-1}} \pmod{(y-v)^i} \]
for \( j := 1 \ldots n \)
\[ g_j^{(c)} := \Delta \ast h_j^{(r)} \pmod{((y-v), g_j^{(1)})} \]
\[ g_j := g_j + (y-v)^{i-1} g_j^{(c)} \]
for \( j := 1 \ldots n \)
\[ g_j := \text{pp}(g_j) \quad \# \text{undo the imposed leading coefficients} \]
\[ \text{if } f \neq \prod_j g_j \]
\[ \text{then return "bad evaluation"} \]
return \( (g_1, \ldots, g_n) \)
saw in Example 18 that this might always happen, in practice these cases are rare. Indeed, if \( m \geq 1 \), i.e. we are factoring trivariates or beyond, we can efficiently pick a good reduction, by the following result.

**Theorem 41 ([vzG85, Theorem 4.5])** Let \( f \) be a representation of a multivariate polynomial. Then there is a reduction from the problem of finding the factorisation shape of \( f \) (Definition 81) to that of finding the factorisation shape of bivariates, which is polynomial in the size and degree of \( f \), for any of the sizes in (2.8).

**5.8.2 A Sparsity Result**

We have already seen (section 2.3.7) that the complexity questions associated to sparse polynomials are challenging, and indeed

\[
x^p - 1 = (x - 1)(x^{p-1} + x^{p-2} + \cdots + x + 1)
\]

shows that a sparse univariate polynomial can have completely dense factors. There are two possible generalisations of (5.12) to multivariates:

\[
(x_1 \ldots x_n)^p - 1 = (x_1 \ldots x_n - 1) \left( (x_1 \ldots x_n)^{p-1} + (x_1 \ldots x_n)^{p-2} + \cdots + x_1 \ldots x_n + 1 \right);
\]

\[
\prod_{i=1}^{n} (x_i^p - 1) = \left( \prod_{i=1}^{n} (x_i - 1) \right) \left( \prod_{i=1}^{n} (x_i^{p-1} + x_i^{p-2} + \cdots + x_i + 1) \right).
\]

If we write these equations as \( f = gh \), and if we write \( t_f \) for the number of monomials in the expanded representation of \( f \), then (5.13) has \( t_f = 2 \) and \( t_h = p \), as in the univariate case, and (5.14) has \( t_f = 2^n \) and \( t_h = p^n \) — in particular if \( n = p \), \( t_h \) is not polynomial in \( t_f \), being \( t_f^{\log_2 \log_2 t_f} \). Might there be still worse examples, say where \( t_f \) was polynomial in \( n \) and the degree, but \( t_h \) was exponential. It has recently been shown that not.

**Theorem 42 ([DdO14, Theorem 1])** If \( f \), a polynomial in \( n \) variables, of degree at most \( d \) in each variable, factors as \( f = gh \) then

\[
t_g = O \left( \max \left( t_f^{O(\log t_f \log \log t_f)}, d^{O(\log d)} \right) \right).
\]

**5.8.3 The Leading Coefficient Problem**

It seems too good to be true, that the multivariate generalization of Hensel lifting needed no new concepts. While this is true as it stands, there is a major snag in practice with Algorithm 36 and that is the cost of imposing leading coefficients. In the univariate case, all this meant was that we had to lift very slightly further, but in the multivariate case we are lifting factors of \( F \), which may well have many more terms than \( f \). Solving this problem was (one of) the

---

6The theory and implementations of this chapter have all been in terms of recursive representations, but the theory appears to be better expressed in terms of expanded representations.
5.8. MULTIVARIATE FACTORING

major contributions of the EEZ algorithm proposed in [Wan78], which can best be described as “discovered leading coefficients” rather than “imposed leading coefficients”.

Algorithm 37 (Wang’s EEZ Hensel Lifting)

Input: \( f \in \mathbb{Z}[x_1, \ldots, x_n] \) squarefree and primitive (no integer common factor)

Output: Factorization of \( f \) over \( \mathbb{Z} \)

Write \( f = \sum_{i=0}^{d} c_i(x_1, \ldots, x_n) x^i \)

Recursively, factor \( c_d(x_1, \ldots, x_n) \) as \( \Omega F_1^{e_1} \cdots F_k^{e_k} \)

Choose integers \( a_1, \ldots, a_n \) such that (see [Wan78, p. 1218] for feasibility)

1. \( c_d(a_1, \ldots, a_n) \neq 0 \)
2. \( f^{(0)}(x) := f(a_1, \ldots, a_n, x) \) is square-free
3. Each \( F_i(a_1, \ldots, a_n) \) is divisible by a prime \( p_i \), not dividing any \( F_j(a_1, \ldots, a_n) : j < i, \Omega \) or \( \text{cont}(f^{(0)}) \).

Factor \( f^{(0)}(x) = c \prod_{i=1}^{m} f_i^{(0)}(x) \)

[In practice, repeat the previous steps a few times, to ensure \( m \) minimal, and hence reduce the possibility of bad reduction]

Then \( f = \prod_{i=1}^{n} f_i \) where \( f_i(a_1, \ldots, a_n, x) = f_i^{(0)}(x) \).

Use the \( p_i \) to distribute the \( F_i \) among the leading coefficients of the \( f_i \), to get \( h_i \) but with the correct leading coefficient

For \( k := 1 \ldots n \)

\[ f(x_1, \ldots, x_k, a_{k+1}, \ldots, a_n, x) = \prod h_i(x_1, \ldots, x_k, a_{k+1}, \ldots, a_n, x) \]

We know what the leading coefficients are

Use a variant of Algorithm 36 to lift this to

\[ f(x_1, \ldots, x_k, a_{k+1}, \ldots, a_n, x) = \prod h_i(x_1, \ldots, x_k, a_{k+1}, \ldots, a_n, x) \]

Example 20 ([Wan78, p. 1218]) Consider \footnote{There is a typographical error: see http://staff.bath.ac.uk/masjhd/JHD-CA/Wangp1220.html} \( f(x, y, z) = (4z^2y^4 + 4z^3y^3 - 4z^4y^2 - 4z^5y)x^6 + \cdots. \)

The leading coefficient factors as \( 4yz^2(y + z)^2(y - z) \). A suitable evaluation is \( (y = -14, z = 3) \) when

\[ F_1 = y \rightarrow \frac{14}{p_1=7}, F_2 = z \rightarrow \frac{3}{p_2=3}, F_3 = y + z \rightarrow \frac{11}{p_3=11}, F_4 = y - z \rightarrow \frac{17}{p_4=17}. \]

\[ f(x, -14, 3) = f_1^{(0)}(x) f_2^{(0)}(x) f_3^{(0)}(x) = (187x^2 - 23)(44x^2 + 42x + 1)(126x^2 - 9x + 28), \]

so \( f_1^{(0)} \) has \( p_3 p_4 \), \( f_2^{(0)} \) has \( 4p_3 \) and \( f_3^{(0)} \) has \( p_1 p_2^2 \). Hence the leading coefficients, are
in fact \((y+z)(y-z), -4(y+z)\) and \(-yz^2\). The lift, with these leading coefficients, rather than the whole of \(4z^2y^4 + 4z^3y^3 - 4z^4y^2 - 4z^5y\), imposed, works fine. The polynomial \(f\) has 88 terms when expanded, whereas if we imposed leading coefficients in the style of algorithm 36, we would be lifting against an \(F\) with 530 terms.

5.9 Other Applications

5.9.1 \(p\)-adic Greatest Common Divisors

The entire \(p\)-adic/Hensel construction is applicable to the computation of greatest common divisors as well.

5.9.1.1 Univariate Greatest Common Divisors

Since it is practically as cheap to use the largest possible single word prime as it is to use a small one, and the probability that \(p\) is bad seems to be proportional to \(1/p\) (Observation 6), we take one such large \(p\) (possibly a second just to check that we don’t have bad reduction), and compute \(h_p = \gcd(f_p, g_p)\). This gives us a factorization \(f_p = h_p f_p\). Since we are not sure of the leading coefficient, we will impose \(\gcd(\text{lc}(f), \text{lc}(g))\) as the leading coefficient of \(h_p\) and \(\text{lc}(f)\) as that of \(f_p\). We then lift this (generally Algorithm 36) to a factorization of \(\gcd(\text{lc}(f), \text{lc}(g))f = h_p(f_p)^{(k)}\) for a suitable \(k\) (as in section 4.2.1), interpret \(h_p^{(k)}\) as a polynomial over the integers, make it primitive, and check that it divides \(f\) and \(g\).

5.9.1.2 Multivariate Greatest Common Divisors

Assume we have two polynomials \(f, g \in R[y][x]\) (where typically \(R = \mathbb{Z}[x_1, \ldots, x_n]\)) and we wish to compute their greatest common divisor. We may as well (Theorem 6) assume that they are primitive with respect to \(x\). We take a value \(v\) (normally in \(\mathbb{Z}\) or the base ring of \(R\)) for \(y\) (possibly a second just to check that we don’t have bad reduction), and compute \(h_v = \gcd(f_v, g_v)\). This gives us a factorization \(f_v = h_v f_v\). Since we are not sure of the leading coefficient, we will impose \(\gcd(\text{lc}(f), \text{lc}(g))\) as the leading coefficient of \(h_v\) and \(\text{lc}(f)\) as that of \(f_v\). We then lift this (generally Algorithm 36) to a factorization of \(\gcd(\text{lc}(f), \text{lc}(g))f = h_v^{(k)}(f_v)^{(k)}\) for a suitable \(k\), interpret \(h_v^{(k)}\) as a polynomial in \(R[y][x]\), make it primitive, and check that it divides \(f\) and \(g\).

5.9.2 \(p\)-adic Gröbner Bases

This section is largely taken from [Win88]. The importance of, and difficulties with, moving from computations over \(\mathbb{Q}\) to finite domains are described in section 4.6. In particular we note the definition of bad reduction and the fact (Lemma 10) that there are only finitely many primes of bad reduction.
5.9. OTHER APPLICATIONS

Suppose we are given $F$ as input, and wish to compute its Gröbner base $G$. We can choose a prime $p$ (it would certainly be a good idea to choose a $p$ not dividing any leading coefficient of $F$), and compute a Gröbner base $G^{(1)}$ modulo $p = p^1$. In the matrix formulation of Gröbner base theory (section 3.3.5), it follows that there exist matrices $X, Y$ and $R$ such that

\begin{align*}
X^{(1)}F &= G^{(1)} \\
Y^{(1)}G^{(1)} &= F \pmod{p} \quad (3.36) \\
R^{(1)}G^{(1)} &= 0
\end{align*}

where $F$ and $G^{(1)}$ are the matrix versions of $F$ and $G^{(1)}$. We might then be tempted to lift these equations, by analogy with (5.9), but there is a snag.

Since $G^{(1)}$ is a Gröbner base, it reduces $F$ to zero, and hence the monomials needed to write $Y^{(1)}G^{(1)} = F$ are at most those of $F$ (or possibly lower ones that don’t actually occur in a sparse representation of $F$). Similarly $R^{(1)}G^{(1)} = 0$ is the statement that all S-polynomials reduce to 0, and hence the highest degree that any $x_i$ can need is its highest degree in $G^{(1)}$. However, there is no such tidy bound for $X^{(1)}$. Fortunately [Win88, Theorem 2] shows that we only need to lift the other two, whose degrees are bounded.

So we consider

\begin{align*}
Y^{(2)}G^{(2)} &= F \pmod{p^2} \quad (5.16) \\
R^{(2)}G^{(2)} &= 0
\end{align*}

write $G^{(2)} = G^{(1)} + p\hat{G}^{(2)}$ etc. as in (5.9), divide through by $p$ and get

\begin{align*}
Y^{(1)}\hat{G}^{(2)} + \hat{Y}^{(2)}G^{(1)} &= \frac{F - Y^{(1)}G^{(1)}}{p} \pmod{p} \quad (5.17)
\end{align*}

Just as (3.36) did not guarantee uniqueness, these equations don’t necessarily have a unique solution. Indeed $G$ is not unique, since we can multiply by any rational (without a $p$ part), so we need to impose that $G$ and every $G^{(k)}$ is monic. Then the $\hat{G}^{(2)}$ part is unique [Win88, Theorem 4] as long as $p$ is lucky.

This lifting process can be continued until we reach a sufficiently high power of $p$, at which point we can convert the coefficients from being modulo $p^k$ to $Q$ by the techniques of Algorithm 25. But what is a sufficiently high power? This is somewhat of a conundrum, unsolved in [Win88]. We can try taking a leaf out of section 4.2.5 and say that, if the solution stabilises, we will verify if it is correct.

**Stabilisation** This is harder to check than in section 4.2.5, since we are looking for rational not integer coefficients. Hence it is not correct to check that $G^{(k)} = 0$; generally it never will be. Instead, we have to check that the reconstitutions by Algorithm 25 are equal. Doing this efficiently, rather than reconstructing completely and discovering that the two are not equal, is an interesting programming task.

**Verification** We need to check two things: $F \rightarrow^* G$ and that $G$ is a Gröbner base, i.e. that every $S(g_i, g_j) \rightarrow^* G$.0.
Unluckiness of the prime $p$ can manifest itself in one of three ways.

1. Failure to lift. \cite{Win88} Theorem 4 only guarantees a lift (unique for every $G^{(k)}$) if $p$ is lucky. In some sense, this is the best case, since we at least know that $p$ is unlucky.

2. Failure to stabilise — we just compute forever.

3. Failure to verify — if we terminate in a genuine Gröbner base $G$ over the rationals, but $F$ does not reduce to 0, then indeed $p$ must have been unlucky. However, if the reconstructed $G$ is not a Gröbner base over the rationals, we do not know whether $p$ was unlucky, or the stabilisation was an accident and computing further will give the right result.

**Open Problem 26 (p-adic Gröbner bases)** Convert the above, and \cite{Win88}, into a genuine algorithm for computing Gröbner bases $p$-adically. In particular, is it possible to use the “majority voting” idea of \cite{IPS11} (footnote 13 on page 181) to improve the chances of having good reduction.

### 5.9.3 $p$-adic determinants

\cite{ABM99} describes a method for computing determinants which is “largely $p$-adic”, in the sense that “most of” $\det(M)$ is computed via a $p$-adic solution to some linear systems, the l.c.m. of whose denominators is typically $\det(M)$, and generally a large factor $D$ of $\det(M)$, and then computing $\det(M)/D$ by a Chinese Remainder approach. The expected complexity is

$$O(n^4 + n^3(\log n + s)^2),$$

where $s$ is the length of the entries.

### 5.10 Conclusions

In this and the previous chapter, we have seen two different methods for computing in “large” domains via “small” ones: the modular method (Figure 4.1) and the $p$-adic/Hensel method (Figure 5.1). Both suffer from “bad reduction” in that the calculation in the small domain may not be a faithful representation of the calculation in the large domain. Although the $p$-adic/Hensel method only needs one small domain, we may compute more than one to minimise the probability of bad reduction. How do the two fare on various examples.

**Linear Algebra** Section 4.1 presented the modular approach for computing the determinant and other tasks of linear algebra. \cite{Dix82} presents a $p$-adic approach for solving $Ax = b$ which he claims to be somewhat faster — $O(n^3 \log^2 n)$ rather than $O(n^4)$. However he admits that the modular approach is probably better for determinants themselves: a belief contradicted by \cite{ABM99}. 
Chapter 6

Algebraic Numbers and Functions

Definition 86 An algebraic number is a root of a polynomial with integer coefficients, i.e. \( \alpha \) such that \( f(\alpha) = 0 : f \in \mathbb{Z}[t] \). The set of all algebraic numbers is denoted by \( A \). An \( \alpha \in A \setminus \mathbb{Q} \) will be referred to as a non-trivial algebraic number. If \( f \) is monic, we say that \( \alpha \) is an algebraic integer.

Allowing \( f \in \mathbb{Q}[t] \) gives us no advantage, since we can clear denominators without affecting whether \( f(\alpha) = 0 \). Of course, “monic” only makes sense over \( \mathbb{Z}[t] \).

Notation 27 Let \( A \) stand for the set of all algebraic numbers, so that \( \mathbb{Q} \subset A \subset \mathbb{C} \).

Definition 87 An algebraic function is a root of a polynomial with polynomial coefficients, i.e. \( \alpha \) such that \( f(\alpha) = 0 : f \in \mathbb{Z}[x_1, \ldots, x_n][t] \). If \( \alpha \notin \mathbb{Q}(x_1, \ldots, x_n) \), we will say that \( \alpha \) is a non-trivial algebraic function.

As above, allowing \( f \in \mathbb{Q}(x_1, \ldots, x_n)[t] \) gives us no advantage, since we can clear denominators without affecting whether \( f(\alpha) = 0 \).

In either case, if \( f \) is irreducible, there is no algebraic way of distinguishing one root of \( f \) from another. For example, \( \sqrt{2} \approx 1.4142 \) and \(-\sqrt{2} \approx -1.4142 \) are both roots of \( t^2 - 2 = 0 \), but distinguishing them involves operations outside the language of fields, e.g. by saying that \( \sqrt{2} \in [1, 2] \) but \(-\sqrt{2} \in [-2, -1] \).

Notation 28 Let \( f(t) \) be the polynomial defining \( \alpha \), and write \( f = \sum_{i=0}^{n} a_i t^i \) with \( a_n \neq 0 \). Until Section 6.3, we will assume that \( f \) is irreducible.

It is perfectly possible to work with \( \alpha \) as if it were another variable, but replacing \( \alpha^n \) by \( \left( \sum_{i=0}^{n-1} a_i \alpha^i \right) / a_n \). A simple example of this is shown in the Maple session in Figure 6.1, which also demonstrates the lack of candidness (i.e. the expression appears to contain \( \sqrt{2} \), but doesn’t really) if we don’t expand.
CHAPTER 6. ALGEBRAIC NUMBERS AND FUNCTIONS

Figure 6.1: Non-candidness of algebraics

\[
> (\sqrt{2}+1)\cdot(\sqrt{2}-1);
\]
\[
\frac{1}{2} \quad \frac{1}{2}
\]
\[
(2 - 1) (2 + 1)
\]

\[
> \text{expand}(%);
\]
\[
1
\]

Observation 16 Introduction of denominators of \(a_n\) can be very tedious in practice, so several systems will work internally with monic \(f\). For example, Maple itself does not, but its high-performance algorithms [vHM02, vHM04] so.

However, mere use of a polynomial-style \texttt{expand} command, which gave us canonical forms in the case of polynomials, is insufficient to give us even normal forms when we allow algebraic numbers, as seen in Figure 6.2. Here, \texttt{simplify} has to “clear denominators”. This process, which may have seemed quite obscure when it was introduced at school, is, from our present point of view, quite easy to understand. Consider an expression

\[
E := \frac{p(\alpha)}{q(\alpha)}
\]  

(6.1)

where \(p\) and \(q\) are polynomials in \(\alpha\), of degree less than \(n\). If we apply the extended Euclidean algorithm (Algorithm 5) to \(q(t)\) and \(f(t)\), we get \(h(t) = \gcd(f, q)\) and polynomials \(c, d\) such that \(cf + dq = h\). Assuming \(h = 1\) (as is
bound to be the case if \( f \) is irreducible), we see that
\[
E = \frac{p(\alpha)}{q(\alpha)} = \frac{d(\alpha)p(\alpha)}{d(\alpha)q(\alpha)} = \frac{d(\alpha)p(\alpha)}{c(\alpha)f(\alpha) + d(\alpha)q(\alpha)}
\]
(since \( f(\alpha) = 0 \))
\[
= d(\alpha)p(\alpha),
\]
and this is purely a polynomial in \( \alpha \).

In the case of Figure 6.2, \( f(t) = t^2 - 1 \) and \( p(t) = 1, q(t) = t - 1 \). The gcd is indeed 1, with \( f = -1, d = t + 1 \). Hence \( \sqrt{2} - 1 = \frac{\sqrt{2} - 1}{(\sqrt{2} + 1)(\sqrt{2} - 1)} = \sqrt{2} + 1 \). Such is the simplicity of this approach that it is usual to insist that \( f \) is irreducible, however in section 6.3 we will explain an alternative approach.

**Definition 88** If \( f \) in Definition 86 or 87 is irreducible and primitive, we say that \( f \) is the minimal polynomial of \( \alpha \). Strictly speaking, we have only defined \( f \) up to associates, but this is usually ignored in theory, though it can be tedious in practice: see the description of the canonicalUnitNormal property in DT90 for a pragmatic solution.

### 6.1 Representations of Algebraic Numbers

The calculations in Figure 6.2 would have been valid whether \( \sqrt{2} = 1.4142\ldots \) or \( \sqrt{2} = -1.4142\ldots \), but often we care about which specific root of a polynomial we mean. For quadratics with real roots there is no problem, we (generally implicitly) use \( \sqrt{n} \) to mean \( \alpha \) such that \( \alpha^2 - n \) and \( \alpha \geq 0 \). Once we move beyond quadratics, life is more complicated: as pointed out at (3.10), the three real roots of \( x^3 - x \) can only be computed via complex numbers. A slight perturbation makes things worse: the roots of \( x^3 - x - \frac{1}{1000} \), numerically
\[
1.000499625, -0.001000001000, -.9994996245
\]
are given algebraically by
\[
\left\{ \begin{array}{l}
\frac{1}{60} \sqrt{108 + 12i\sqrt{11999919}} + 20 \frac{1}{\sqrt{108+12i\sqrt{11999919}}} \\
- \frac{1}{120} \sqrt{108 + 12i\sqrt{11999919}} - 10 \frac{1}{\sqrt{108+12i\sqrt{11999919}}} \\
\frac{i\sqrt{3}}{20} \left( \frac{1}{6} \sqrt{108 + 12i\sqrt{11999919}} - 200 \frac{1}{\sqrt{108+12i\sqrt{11999919}}} \right) \\
- \frac{1}{120} \sqrt{108 + 12i\sqrt{11999919}} - 10 \frac{1}{\sqrt{108+12i\sqrt{11999919}}} \\
\frac{i\sqrt{3}}{20} \left( \frac{1}{6} \sqrt{108 + 12i\sqrt{11999919}} - 200 \frac{1}{\sqrt{108+12i\sqrt{11999919}}} \right) \end{array} \right\},
\]
(6.3)
and these cannot be expressed in terms of real radicals, i.e. the \( i \) is intrinsic. We can avoid this by means of a representation in terms of RootOf constructs:

\[
\text{RootOf}(f(z), \text{index}=1), \ldots, \text{RootOf}(f(z), \text{index}=3),
\]

or by saying “near” which value these roots are, as well as the polynomial:

\[
\text{RootOf}(f(z), 1.0), \ldots, \text{RootOf}(f(z), -1.0).
\]

We should be careful of the difference between (6.2) and (6.5): the first specifies three floating-point numbers, but the second specifies three precise algebraic numbers, which can in theory (and in practice) be evaluated to arbitrary precision, as shown in Figure 6.3.

### 6.2 Factorisation with Algebraic Numbers

If we insist, as we do in this section, that the \( f \) in Definition 86 or 87 be irreducible, the obvious question is “how do we ensure this?” If \( f \in \mathbb{Z}[x_1,\ldots,x_n,t] \), then the theory of Chapter 5 will factor \( f \). However, suppose we have defined \( \alpha \) by \( f \), and then wish to define \( \beta \) by \( g \in \mathbb{Z}[x_1,\ldots,x_n,\alpha,t] \). How do we ensure \( g \) is irreducible? Indeed, even if \( g \) does not involve \( \alpha \) as such, how do we ensure that \( g \) is irreducible in \( \mathbb{Z}[x_1,\ldots,x_n,\alpha,t] \), even if it is irreducible in \( \mathbb{Z}[x_1,\ldots,x_n,t] \)?

This is not just an abstract conundrum. Consider \( f = t^2 - 2 \), \( g = t^2 - 8 \). Both are irreducible in \( \mathbb{Z}[t] \). But if \( \alpha \) is defined to be a root of \( f \), then \( g \) factors as \((t - 2\alpha)(t + 2\alpha)\), corresponding to the fact that \( \sqrt{8} = 2\sqrt{2} \).

For the rest of this section we will consider the case of factoring over algebraic number fields, i.e. polynomials in \( \mathbb{Q}[\alpha_1,\ldots,\alpha_m,x_1,\ldots,x_n] \), where each \( \alpha_i \) is defined by a minimal polynomial in \( \mathbb{Z}[\alpha_1,\ldots,\alpha_{i-1}][t] \). The more general case of algebraic function fields is discussed in [DT81].

The first remark is that the methods of Section 5.3 extend to factoring over algebraic extensions of the integers modulo \( p \).

**Open Problem 27 (Algebraic Numbers Reviewed)** Reconsider the standard approach to factoring polynomials with algebraic number/function coefficients, as described above, in the light of recent progress in factoring polynomials with integer coefficients, notably [vH02].
6.3 The D5 approach to algebraic numbers

The techniques of the previous section, while they do guarantee irreducibility of the defining polynomials, are extremely expensive, and often overkill.

In fact, we have already seen a technique that does not require factoring, and that is the Gianni–Kalkbrener algorithm (Algorithms 10, 11). If we consider again the example at equation (3.47), we see that \( x \) satisfies \( 6 - 3x^2 - 2x^3 + x^5 = 0 \). Over \( \mathbb{Z} \), this already factors as \( (x^2 - 2)(x^3 - 3) \), and \( x^3 - 3 \) is irreducible over \( \mathbb{Z}[\sqrt{2}] \), so the five possible values of \( x \) are \( \alpha \) with \( \alpha^2 - 2 = 0 \), \( -\alpha \), \( \beta \) with \( \beta^3 - 3 = 0 \), \( \gamma \) with \( \gamma^2 + \beta \gamma + \beta^2 = 0 \) and \( -\gamma - \beta \). For each of these there is corresponding value of \( y \), but it is hard to believe that this is simpler than

\[
\langle x^2 - 2, y^2 - x \rangle \cup \langle x^3 - 3, y - x \rangle. \tag{3.48}
\]

\[\text{[DDDD85]}
\]

TO BE COMPLETED

---

1This is not the historical development of the subject, for which see [DDDD85], but is more pedagogic from our current point of view.

2Given that \( x^3 - 3 \) is irreducible over \( \mathbb{Z} \), its irreducibility over \( \mathbb{Z}[\sqrt{2}] \) follows from the fact that the exponent, 2 and 3, are relatively prime. But this is a very ad hoc argument.
Chapter 7

Calculus

Throughout this chapter we shall assume that we are in characteristic zero, and therefore all rings contains \( \mathbb{Z} \), and all fields contain \( \mathbb{Q} \). The emphasis in this chapter will be on algorithms for integration. Historically, the earliest attempts at integration in computer algebra \([\text{Sla61}]\) were based on rules, and attempts to “do it like a human would”. These were rapidly replaced by algorithmic methods, based on the systematisation, in \([\text{Ris69}]\), of work going back to Liouville. Recently, attempts to get ‘neat’ answers have revived interest in rule-based approaches, which can be surprisingly powerful on known integrals — see \([\text{JR10}]\) for one recent approach. In general, though, only the algorithmic approach is capable of proving that an expression in unintegrable.

7.1 Introduction

We defined (Definition \([\text{32}]\)) the formal derivative of a polynomial purely algebraically, and observed (Proposition \([\text{2.3.6}]\)) that it satisfied the sum and product laws. We can actually make the whole theory of differentiation algebraic as follows.

**Definition 89** A differential ring is a ring (Definition \([\text{0}]\)) equipped with an additional unary operation, referred to as differentiation and normally written with a postfix \( ' \), which satisfies two additional laws:

1. \((f + g)' = f' + g'\);
2. \((fg)' = fg' + f'g\).

A differential ring which is also a field (Definition \([\text{13}]\)) is referred to as a differential field.

Definition \([\text{32}]\) and Proposition \([\text{2.3.6}]\) can then be restated as the following result.
**Proposition 67** If \( R \) is any ring, we can make \( R[x] \) into a differential ring by defining \( r' = 0 \) \( \forall r \in R \) and \( x' = 1 \).

**Definition 90** In any differential ring, an element \( \alpha \) with \( \alpha' = 0 \) is referred to as a constant. For any ring \( R \), we write \( R_{\text{const}} \) for the constants of \( R \), i.e.
\[
R_{\text{const}} = \{ r \in R \mid r' = 0 \}.
\]

We will use \( \mathbb{C} \) to stand for any appropriate field of constants.

We will revisit this definition in section 8.4.

**Proposition 68** In any differential field \((f g)' = f' g - f g' g^2 \). (7.1)

The proof is by differentiating \( f = g \left(\frac{f}{g}\right) \) to get
\[
f' = g' \left(\frac{f}{g}\right) + g \left(\frac{f}{g}\right)'
\]
and solving for \( \left(\frac{f}{g}\right)' \). (7.1) is generally referred to as the quotient rule and can be given a fairly tedious analytic proof in terms of \( \epsilon/\delta \), but from our present point of view it is an algebraic corollary of the product rule. It therefore follows that \( K(x) \) can be made into a differential field.

**Notation 29** (Fundamental Theorem of Calculus) (Indefinite) integration is the inverse of differentiation, i.e.
\[
F = \int f \Leftrightarrow F' = f.
\] (7.2)

The reader may be surprised to see a “Fundamental Theorem” reduced to the status of a piece of notation, but from the present point of view, that is what it is. We shall return to this point in section 8.3. The reader may also wonder “where did \( dx \) go?” but \( x \) is, from this point of view, merely that object such that \( x' = 1 \), i.e. \( x = \int 1 \).

We should also note that we have no choice over the derivative of algebraic expressions\footnote{It would be more normal to write “algebraic functions” instead of “algebraic expressions”, but for reasons described in section 8.1 we reserve ‘function’ for specific mappings (e.g. \( \mathbb{C} \rightarrow \mathbb{C} \)), and the proposition is a property of differentiation applied to formulae.}.

**Proposition 69** Let \( K \) be a differential field, and \( \theta \) be algebraic over \( K \) with \( p(\theta) = 0 \) for some polynomial \( p = \sum_{i=0}^{n} a_i z^i \in K[z] \). Then \( K(\theta) \) can be made into a differential field in only one way: by defining
\[
\theta' = -\frac{\sum_{i=0}^{n} a_i' \theta^i}{\sum_{i=0}^{n} ia_i \theta^{i-1}}.
\] (7.3)

In particular, if the coefficients of \( p \) are all constants, so is \( \theta \).

The proof is by formal differentiation of \( p(\theta) = 0 \).
7.2 Integration of Rational Expressions

The integration of polynomials is trivial:

\[ \int \sum_{i=0}^{n} a_i x^i = \sum_{i=0}^{n} \frac{1}{i+1} a_i x^{i+1}. \quad (7.4) \]

Since any rational expression \( f(x) \in K(x) \) can be written as

\[ f = p + \frac{q}{r} \text{ with } \left\{ \begin{array}{l} p, q, r \in K[x] \\ \deg(q) < \deg(r) \end{array} \right. \]

and \( p \) is always integrable by (7.4), we have proved the following (trivial) result: we will see later that its generalisations, Lemmas 11 and 12, are not quite so trivial.

Proposition 70 (Decomposition Lemma (rational expressions)) In the notation of (7.5), \( f \) is integrable if, and only if, \( q/r \) is.

\( q/r \) with \( \deg(q) < \deg(r) \) is generally termed a proper rational function, but, since we are concerned with the algebraic form of expressions in this chapter, we will say “proper rational expression.”

7.2.1 Integration of Proper Rational Expressions

In fact, the integration of proper rational expressions is conceptually trivial (we may as well assume \( r \) is monic, absorbing any constant factor in \( q \)):

1. perform a square-free decomposition (Definition 33) of \( r = \prod_{i=1}^{n} r_i^i \);
2. factorize each \( r_i \) completely, as \( r_i(x) = \prod_{j=1}^{n_i} (x - \alpha_{i,j}) \);
3. perform a partial fraction decomposition (Section 2.3.4) of \( q/r \) as

\[ \frac{q}{r} = \frac{q}{\prod_{i=1}^{n} r_i^i} = \sum_{i=1}^{n} \frac{q_i}{r_i^i} = \sum_{i=1}^{n} \sum_{j=1}^{n_i} \sum_{k=1}^{i} \frac{\beta_{i,j,k}}{(x - \alpha_{i,j})^k}; \quad (7.6) \]

4. integrate this term-by-term, obtaining

\[ \int \frac{q}{r} = \sum_{i=1}^{n} \sum_{j=1}^{n_i} \sum_{k=2}^{i} \frac{-\beta_{i,j,k}}{(k-1)(x - \alpha_{i,j})^{k-1}} + \sum_{i=1}^{n} \sum_{j=1}^{n_i} \beta_{i,j,1} \log(x - \alpha_{i,j}). \quad (7.7) \]

From a practical point of view, this approach has several snags:

1. we have to factor \( r \), and even the best algorithms from the previous chapter can be expensive;
2. we have to factor each \( r_i \) into linear factors, which might necessitate the introduction of algebraic numbers to represent the roots of polynomials;
3. These steps might result in a complicated expression of what is otherwise a simple answer.

To illustrate these points, consider the following examples.

\[
\int \frac{5x^4 + 60x^3 + 255x^2 + 450x + 274}{x^5 + 15x^4 + 85x^3 + 225x^2 + 274x + 120} \, dx
= \log(x^5 + 15x^4 + 85x^3 + 225x^2 + 274x + 120)
= \log(x + 1) + \log(x + 2) + \log(x + 3) + \log(x + 4) + \log(x + 5)
\] (7.8)

is pretty straightforward, but adding 1 to the numerator gives

\[
\int \frac{5x^4 + 60x^3 + 255x^2 + 450x + 275}{x^5 + 15x^4 + 85x^3 + 225x^2 + 274x + 120} \, dx
= \frac{5}{27} \log(x^5 + 72x^4 + \cdots + 102643200000x + 9331200000)
= \frac{5}{27} \log(x + 1) + \frac{5}{27} \log(x + 2) + \frac{5}{27} \log(x + 3) + \frac{5}{27} \log(x + 4) + \frac{5}{27} \log(x + 5)
\] (7.9)

Adding 1 to the denominator is pretty straightforward,

\[
\int \frac{5x^4 + 60x^3 + 255x^2 + 450x + 274}{x^5 + 15x^4 + 85x^3 + 225x^2 + 274x + 121} \, dx
= \log(x^5 + 15x^4 + 85x^3 + 225x^2 + 274x + 121),
\] (7.10)

but adding 1 to both gives

\[
\int \frac{5x^4 + 60x^3 + 255x^2 + 450x + 275}{x^5 + 15x^4 + 85x^3 + 225x^2 + 274x + 121} \, dx
= 5 \sum_\alpha \log(x + 2632025698 + 289 \alpha^4 - 2086891452 + 289 \alpha^3 + 608708804 + 17 \alpha + 3632420 \alpha^2)
\] (7.11)

where

\[
\alpha = \text{RootOf}(38569 z^5 - 38569 z^4 + 15251 z^3 - 2981 z^2 + 288 z - 11).
\] (7.12)

Hence the challenge is to produce an algorithm that achieves (7.8) and (7.10) simply, preferably gives us the second form of the answer in (7.9), but is still capable of solving (7.11). We might also wonder where (7.11) came from: the answer is at (7.26).

### 7.2.2 Hermite’s Algorithm

The key to the method (usually attributed to Hermite [Her72], though informally known earlier) is to rewrite equation (7.7) as

\[
\int \frac{q}{r} = \frac{s_1}{t_1} + \int \frac{s_2}{t_2},
\] (7.13)

where the integral on the right-hand resolves itself as \textit{purely} a sum of logarithms, i.e. is the \( \sum_{i=1}^{n} \beta_{i,j} \log(x - \alpha_{i,j}) \) term. Then a standard argument from

\[
\sum_{i=1}^{n} \beta_{i,j} \log(x - \alpha_{i,j})
\]
Galois theory shows that \( s_1, t_1, s_2 \) and \( t_2 \) do not involve any of the \( \alpha_i \), i.e. that the decomposition (7.13) can be written without introducing any algebraic numbers. If we could actually obtain this decomposition without introducing these algebraic numbers, we would have gone a long way to solving objection 2 above.

We can perform a square-free decomposition (Definition 33) of \( r \) as \( \prod r_i \), and then a partial fraction decomposition to write

\[
\frac{q}{r} = \sum \frac{q_i}{r_i^2} \tag{7.14}
\]

and, since each term is a rational expression and therefore integrable, it suffices to integrate (7.14) term-by-term.

Now \( r_i \) and \( r_i' \) are relatively prime, so, by Bezout’s Identity (2.13), there are polynomials \( a \) and \( b \) satisfying \( ar_i + br_i' = 1 \). Therefore

\[
\int \frac{q_i}{r_i^2} = \int \frac{q_i(\alpha_i + \beta_i)}{r_i^{i-1}} \tag{7.15}
\]

\[
= \int \frac{q_i \alpha}{r_i^{i-1}} + \int \frac{q_i \beta_i'}{r_i^{i-1}} \tag{7.16}
\]

\[
= \int \frac{q_i \alpha}{r_i^{i-1}} + \int \frac{(q_i b/(i - 1))'}{r_i^{i-1}} - \left( \frac{q_i b/(i - 1)}{r_i^{i-1}} \right)' \tag{7.17}
\]

\[
= - \left( \frac{q_i b/(i - 1)}{r_i^{i-1}} \right) + \int \frac{q_i a + (q_i b/(i - 1))'}{r_i^{i-1}} \tag{7.18}
\]

and we have reduced the exponent of \( r_i \) by one. When programming this method one may need to take care of the fact that, while \( \frac{q_i}{r_i^2} \) is a proper rational expression, \( \frac{q_i b}{r_i^{i-1}} \) may not be, but the excess is precisely compensated for by the other term in (7.18).

Hence, at the cost of a square-free decomposition and a partial fraction decomposition, but not a factorization, we have found the rational part of the integral, i.e. performed the decomposition of (7.13). In fact, we have done somewhat better, since the integral term will have been split into summands corresponding to the different \( r_i \).

### 7.2.3 The Ostrogradski–Horowitz Algorithm

Although quite simple, Hermite’s method still needs square-free decomposition and partial fractions. Horowitz [Hor69, Hor71] therefore proposed to computer algebraists the following method, which was in fact already known [Ost45], but largely forgotten in the west. It follows from (7.18) that, in the notation of (7.14) \( t_1 = \prod r_i^{i-1} \). Furthermore every factor of \( t_2 \) arises from the \( r_i \), and is not
repeated. Hence we can choose
\[ t_1 = \gcd(r, r') \text{ and } t_2 = r/t_1. \] (7.19)

Having done this, we can solve for the coefficients in \( s_1 \) and \( s_2 \), and the resulting equations are linear in the unknown coefficients. More precisely, the equations become
\[ q = s'_1 \frac{r}{t_i} - s_1 \frac{t'_1 t_2}{t_1} + s_2 t_1, \] (7.20)
where the polynomial divisions are exact, and the linearity is now obvious. The programmer should note that \( s_1/t_1 \) is guaranteed to be in lowest terms, but \( s_2/t_2 \) is not (and indeed will be 0 if there is no logarithmic term).

### 7.2.4 The Trager–Rothstein Algorithm

Whether we use the method of section 7.2.2 or 7.2.3, we have to integrate the logarithmic part. (7.8)–(7.11) shows that this may, but need not, require algebraic numbers. How do we tell? The answer is provided by the following observation\(^3\)\(^4\): if we write the integral of the logarithmic part as \( \sum c_i \log v_i \), we can determine the equation satisfied by the \( c_i \), i.e. the analogue of (7.12), by purely rational computations.

So write
\[ \int \frac{s_2}{t_2} = \sum c_i \log v_i, \] (7.21)
where we can assume\(^4\)

1. \( s_2/t_2 \) is in lowest terms;
2. the \( v_i \) are polynomials (using \( \log \frac{f}{g} = \log f - \log g \));
3. the \( v_i \) are square-free (using \( \log \prod f_i^i = \sum i \log f_i \));
4. the \( v_i \) are relatively prime (using \( c \log pq + d \log pr = (c+d) \log p + c \log q + d \log r \));
5. the \( c_i \) are all different (using \( c \log p + c \log q = c \log pq \));
6. the \( c_i \) generate the smallest possible extension of the original field of coefficients.

(7.21) can be rewritten as
\[ \frac{s_2}{t_2} = \sum c_i \frac{v'_i}{v_i}. \] (7.22)

\(^3\) As happens surprisingly often in computer algebra, this was a case of simultaneous discovery.

\(^4\) We are using “standard” properties of the log operator here without explicit justification; they are justified in Section 7.3 and revisited in Section 8.6.
Hence $t_2 = \prod v_i$ and, writing $u_j = \prod_{i \neq j} v_i$, we can write (7.22) as

$$s_2 = \sum c_i v'_i u_i.$$  (7.23)

Furthermore, since $t_2 = \prod v_i$, $t'_2 = \sum v'_i u_i$. Hence

$$v_k = \gcd(0, v_k) = \gcd(s_2 - \sum c_i v'_i u_i, v_k) = \gcd(s_2 - c_k v'_k u_k, v_k)$$

since all the other $u_i$ are divisible by $v_k$

$$= \gcd(s_2 - c_k \sum v'_i u_i, v_k)$$

for the same reason

$$= \gcd(s_2 - c_k t'_2, v_k).$$  (7.24)

But if $l \neq k$,

$$\gcd(s_2 - c_k t'_2, v_l) = \gcd\left(\sum c_i v'_i u_i - c_k \sum v'_i u_i, v_l\right)$$

$$= \gcd(c_l v'_l u_l - c_k v'_l u_l, v_l)$$

since all the other $u_i$ are divisible by $v_l$

$$= 1.$$

Since $t_2 = \prod v_i$, we can put these together to deduce that

$$v_k = \gcd(s_2 - c_k t'_2, t_2).$$  (7.24)

Given $c_k$, this will tell us $v_k$. But we can deduce more from this: the $c_k$ are precisely those numbers $\lambda$ such that $\gcd(s_2 - \lambda t'_2, t_2)$ is non-trivial. Hence we can appeal to Proposition 73 (page 268), and say that $\lambda$ must be such that

$$P(\lambda) := \text{Res}_x(s_2 - \lambda t'_2, t_2) = 0.$$  (7.25)

If $t_2$ has degree $n$, $P(\lambda)$ is the determinant of a $2n - 1$ square matrix, $n$ of whose rows depend linearly on $\lambda$, and thus is a polynomial of degree $n$ in $\lambda$ (strictly speaking, this argument only shows that it has degree at most $n$, but the coefficient of $\lambda^n$ is $\text{Res}(t'_2, t_2) \neq 0$).

**Algorithm 38 (Trager–Rothstein)**

**Input:** $s_2, t_2 \in K[x]$ relatively prime with $\deg_x(s_2) < \deg_x(t_2)$, $t_2$ squarefree

**Output:** A candidate expression for $\int s_2/t_2 \, dx$

5In the sense of Definition 5, every algebraic extension occurring in the expression is necessary.
\[ P(\lambda) := \text{Res}(s^2 - \lambda t', t_2, x) \]

Write \[ P(\lambda) = \prod_i Q_i(\lambda)^{v_i} \] (square-free decomposition)

\[ R := 0 \]

for \( i \) in 1... such that \( Q_i \neq 1 \)

\[ v_i := \gcd(s^2 - \lambda t', t_2) \text{ where } Q_i(\lambda) = 0 \quad \# \text{deg}(v_i) = i \]

\[ R := R + \sum_{\lambda \text{ a root of } Q_i} \lambda \log v_i(x) \]

return \( R \)

An alternative formulation is given in [LR90], with an important clarification in [Mul97]. In practice, roots of a \( Q_i \) which are rational, or possibly even quadratic over the rationals, are normally made explicit rather than represented as RootOf constructs. This accounts for answers such as the following:

\[
\int \frac{3 x^3 - 2 x^2}{(x^2 - 2)(x^3 - x - 3)} \, dx = - \ln (x^2 - 2) + \sum_{r = \text{RootOf}(z^3 - z - 3)} \frac{r (3 + 2 r) \ln (x - r)}{3 r^2 - 1}.
\]

The integrands in (7.8–7.11) are all special cases of

\[
\int \frac{5 x^4 - 60 x^3 + 255 x^2 - 450 x + a}{x^5 - 15 x^4 + 85 x^3 - 225 x^2 + 274 x - b} \, dx \quad (7.26)
\]

whose integral is, by Algorithm 38

\[
\sum_{\alpha} \frac{(5 \alpha^4 - 60 \alpha^3 + 255 \alpha^2 - 450 \alpha + a) \ln (x - \alpha)}{5 \alpha^4 - 60 \alpha^3 + 255 \alpha^2 - 450 \alpha + 274} \quad (7.27)
\]

where \( \alpha = \text{RootOf}(z^5 - 15 z^4 + 85 z^3 - 225 z^2 + 274 z - b) \). (7.11) is of this form, but (7.8–7.10) show how Algorithm 38 can produce more candid expressions where available, and indeed guarantee not to involve any unnecessary RootOf constructs.

The same process also leads to

\[
\int \frac{1}{x^2 + 1} \, dx = \frac{i}{2} (\ln (1 - ix) - \ln (1 + ix)), \quad (7.28)
\]

at which point the reader might complain “I asked to integrate a real function, but the answer is coming back in terms of complex numbers”. The answer is, of course, formally correct: differentiating the right-hand side of (7.28) yields

\[
\frac{i}{2} \left( \frac{-i}{1 - ix} - \frac{i}{1 + ix} \right) = \frac{i}{2} \left( \frac{-i(1 + ix)}{1 + x^2} - \frac{i(1 - ix)}{1 + x^2} \right) = \frac{1}{1 + x^2}:
\]

the issue is that the reader, interpreting the symbols log etc. as the usual functions of calculus, is surprised. This question of interpretation as functions \( R \rightarrow R \) or \( C \rightarrow C \), will be taken up in section 8.5. In the next section we will give a totally algebraic definition of log etc.
7.3. Theory: Liouville’s Theorem

Definition 91 Let $K$ be a field of expressions. The expression $\theta$ is an elementary generator over $K$ if one of the following is satisfied:

(a) $\theta$ is algebraic over $K$, i.e. $\theta$ satisfies a polynomial equation with coefficients in $K$;

(b) $\theta$ (assumed to be nonzero) is an exponential over $K$, i.e. there is an $\eta$ in $K$ such that $\theta' = \eta' \theta$, which is only an algebraic way of saying that $\theta = \exp \eta$;

(c) $\theta$ is a logarithm over $K$, i.e. there is an $\eta$ in $K$ such that $\theta' = \eta'/\eta$, which is only an algebraic way of saying that $\theta = \log \eta$.

In the light of this definition, (7.28) would be interpreted as saying

$$\int \frac{1}{x^2 + 1} \, dx = \frac{i}{2} (\theta_1 - \theta_2),$$

where $\theta_1' = \frac{i}{1+x}$ and $\theta_2' = \frac{i}{1-x}$.

We should note that, if $\theta$ is a logarithm of $\eta$, then so is $\theta + c$ for any constant (Definition 90) $c$. Similarly, if $\theta$ is an exponential of $\eta$, so is $c \theta$ for any constant $c$, including the case $c = 0$, which explains the stipulation of nonzeroness in Definition 91(b).

A consequence of these definitions is that log and exp satisfy the usual laws “up to constants”.

log Suppose $\theta_i$ is a logarithm of $\eta_i$. Then

$$(\theta_1 + \theta_2)' = \theta_1' + \theta_2' = \frac{\eta_1' + \eta_2'}{\eta_1 \eta_2} = \frac{\eta_1' \eta_2 + \eta_1 \eta_2'}{\eta_1 \eta_2} = \frac{(\eta_1 \eta_2)'}{\eta_1 \eta_2},$$

and hence $\theta_1 + \theta_2$ is a logarithm of $\eta_1 \eta_2$, a rule normally expressed as (but see the discussion on page 253 for what happens when we try to interpret log as a function $\mathbb{C} \to \mathbb{C}$)

$$\log \eta_1 + \log \eta_2 = \log(\eta_1 \eta_2).$$

(7.29)

Similarly $\theta_1 - \theta_2$ is a logarithm of $\eta_1/\eta_2$ and $n \theta_1$ is a logarithm of $\eta_1^n$ (for $n \in \mathbb{Z}$: we have attached no algebraic meaning to arbitrary powers).

---

This clause is not normally stated, but is important in practice: see the statement about “up to a constant” later.
Suppose now that \( \theta_i \) is an exponential of \( \eta_i \). Then
\[
(\theta_1 \theta_2)' = \theta'_1 \theta_2 + \theta_1 \theta'_2 \\
= \eta'_1 \theta_1 \theta_2 + \theta_1 \eta'_2 \theta_2 \\
= (\eta_1 + \eta_2)' (\theta_1 \theta_2)
\]
and hence \( \theta_1 \theta_2 \) is an exponential of \( \eta_1 + \eta_2 \), a rule normally expressed as
\[
\exp \eta_1 \exp \eta_2 = \exp(\eta_1 + \eta_2). \tag{7.30}
\]
Similarly \( \theta_1 / \theta_2 \) is an exponential of \( \eta_1 - \eta_2 \) and \( \theta^n \) is an exponential of \( n \eta \) (for \( n \in \mathbb{Z} \): we have attached no algebraic meaning to arbitrary powers).

(1) Suppose \( \theta \) is a logarithm of \( \eta \), and \( \phi \) is an exponential of \( \theta \). Then
\[
\phi' = \theta' \phi = \frac{\eta'}{\eta}, \text{ so }
\]
\[
\frac{\phi'}{\phi} = \frac{\eta'}{\eta} = \theta',
\]
and \( \theta \) is a logarithm of \( \phi \), as well as \( \phi \) being an exponential of \( \theta \).

(2) Suppose now that \( \theta \) is an exponential of \( \eta \), and \( \phi \) is a logarithm of \( \theta \). Then
\[
\phi' = \frac{\theta'}{\theta} \\
= \frac{\eta' \theta}{\theta} = \eta',
\]
so \( \eta \) and \( \phi \) differ by a constant. But \( \phi \), being a logarithm, is only defined up to a constant.

(1)+(2) These can be summarised by saying that, up to constants, log and exp are inverses of each other.

**Definition 92** Let \( K \) be a field of expressions. An overfield \( K(\theta_1, \ldots, \theta_n) \) of \( K \) is called a field of elementary expressions over \( K \) if every \( \theta_i \) is an elementary generator over \( K(\theta_1, \ldots, \theta_{i-1}) \). A expression is elementary over \( K \) if it belongs to a field of elementary expressions over \( K \).

If \( K \) is omitted, we understand \( \mathbb{C}(x) \): the field of rational expressions.

For example, the expression \( \exp(\exp x) \) can be written as elementary over \( K = \mathbb{Q}(x) \) by writing it as \( \theta_2 \in K(\theta_1, \theta_2) \) where \( \theta'_1 = \theta_1 \), so \( \theta_1 \) is elementary over \( K \), and \( \theta'_2 = \theta'_1 \theta_2 \), and so \( \theta_2 \) is elementary over \( K(\theta_1) \).

**Observation 17** Other functions can be written this way as well. For example, if \( \theta' = i \theta \) (where \( i^2 = -1 \)), then \( \phi = \frac{1}{2i} (\theta - 1/\theta) \) is a suitable model for \( \sin(x) \), as in the traditional \( \sin x = \frac{1}{2i} (e^{ix} - e^{-ix}) \). Note that \( \phi'' = -\phi \), as we would hope.
From this point of view, the problem of integration, at least of elementary expressions, can be seen as an exercise in the following paradigm.

**Algorithm 39 (Integration Paradigm)**

**Input:** an elementary expression $f$ in $x$

**Output:** An elementary $g$ with $g' = f$, or failure

Find fields $C$ of constants, $L$ of elementary expressions over $C(x)$ with $f \in L$

if this fails

then error "integral not elementary"

Find an elementary overfield $M$ of $L$, and $g \in M$ with $g' = f$

if this fails

then error "integral not elementary"

else return $g$

This looks very open-ended, and much of the rest of this chapter will be devoted to turning this paradigm into an algorithm: one that when it returns "integral not elementary" has actually proved this fact. We will see later on that the poly-logarithm function and the erf function (defined as $\text{erf} x = \int e^{-x^2}$ up to a constant multiple) are not elementary, and can be proved so this way. Other examples of non-elementary functions are:

- $W$ the Lambert $W$ function, the solution of $W(z) \exp(W(z)) = z$ [CGH+96], proved non-elementary in [BCDJ08];
- $\omega$ the Wright $\omega$ function, the solution of $\omega(z) + \ln \omega(z) = z$ [CJ02], proved non-elementary in [BCDJ08].

### 7.3.1 Liouville’s Principle

The first question that might come to mind is that the search for $M$ looks pretty open-ended. Here we are helped by the following result.

**Theorem 43 (Liouville’s Principle)** Let $f$ be a expression from some expression field $L$. If $f$ has an elementary integral over $L$, it has an integral of the following form:

$$
\int f = v_0 + \sum_{i=1}^{n} c_i \log v_i,
$$

(7.31)

where $v_0$ belongs to $L$, the $v_i$ belong to $\hat{L}$, an extension of $L$ by a finite number of constants algebraic over $L_{\text{const}}$, and the $c_i$ belong to $\hat{L}$ and are constant.

The proof of this theorem (see, for example, [Rit48]), while quite subtle in places, is basically a statement that the only new expression in $g$ which can disappear on differentiation is a logarithm with a constant coefficient.

In terms of the integration paradigm (Algorithm 39), this says that we can restrict our search for $M$ to $M$ of the form $L(c_1, \ldots, c_k, v_1, \ldots, v_k)$ where the $c_i$ are algebraic over $L_{\text{const}}$ and the $v_i$ are logarithmic over $L(c_1, \ldots, c_k)$. 
Another way of putting this is to say that, if \( f \) has an elementary integral over \( L \), then \( f \) has the following form:

\[
f = v_0' + \sum_{i=1}^{n} \frac{c_i v_i'}{v_i},
\]

(7.32)

### 7.3.2 Finding \( L \)

A question that might seem trivial is the opening line of Algorithm 39: “Find a field \( C \) of constants, and a field \( L \) of elementary expressions over \( C(x) \) with \( f \in L \).” From an algorithmic point of view, this is not so trivial, and indeed is where many of the theoretical difficulties lie. We can distinguish three major difficulties.

1. Hidden constants. It is possible to start from a field \( C \), make various elementary extensions (Definition 91) of \( C(x) \) to create \( L \), but have \( L_{\text{const}} \) be strictly larger than \( C \). Consider, for example, \( L = \mathbb{Q}(x, \theta_1, \theta_2, \theta_3) \) where \( \theta_1' = \theta_1, \theta_2' = 2x\theta_2 \), and \( \theta_3' = (2x + 1)\theta_3 \). Then

\[
\left( \frac{\theta_1' \theta_2'}{\theta_3'} \right)' = \frac{\theta_3' (\theta_1' \theta_2')' - \theta_1' \theta_2' \theta_3'}{\theta_3'^2} = \frac{\theta_3' \theta_1' \theta_2' + \theta_3' \theta_1' \theta_2' - \theta_1' \theta_2' \theta_3'}{\theta_3'^2} = \frac{\theta_1' \theta_2' \theta_3' + 2x\theta_1' \theta_2' \theta_3' - (2x + 1)\theta_1' \theta_2' \theta_3'}{\theta_3'^2} = 0,
\]

so \( \frac{\theta_1' \theta_2'}{\theta_3'} \) is, unexpectedly, a constant \( c \), and we should be considering \( \mathbb{Q}(c)(x, \theta_1, \theta_2) \), with \( \theta_3 = c\theta_1 \theta_2 \). This is perhaps not so surprising if we give the \( \theta_i \) their conventional meanings as \( \exp(x) \) etc., and write \( L = \mathbb{Q}(x, \exp(x), \exp(x^2), \exp(x^2 + x)) \), where we can clearly write \( \exp(x^2 + x) = \exp(x^2) \exp(x) \). Of course, \( \theta_1 \) might equally well be \( 100 \exp(x) \), etc., so all we can deduce (in the language of differential algebra) is that the ratio is a constant \( c \), not necessarily that \( c = 1 \).

Equally, we could consider \( L = \mathbb{Q}(x, \theta_1, \theta_2, \theta_3) \) where \( \theta_1' = \frac{1}{x-1}, \theta_2' = \frac{1}{x-1} \) and \( \theta_3' = \frac{2x}{x^2 - 1} \). Then

\[
(\theta_1 + \theta_2 - \theta_3)' = \theta_1' + \theta_2' - \theta_3' = \frac{1}{x-1} + \frac{1}{x-1} - \frac{2x}{x^2 - 1} = 0,
\]

and again we have a hidden constant \( c = \theta_1 + \theta_2 - \theta_3 \), and we should be considering \( \mathbb{Q}(c)(x, \theta_1, \theta_2) \), with \( \theta_3 = \theta_1 + \theta_2 - c \). Again, this is not so surprising if we give the \( \theta_i \) their conventional meanings as \( \log(x - 1) \) etc.,
7.3. THEORY: LIOUVILLE’S THEOREM

where we can clearly write \( \log(x^2 - 1) = \log(x - 1) + \log(x + 1) \). Of course, \( \theta_1 \) might equally well be \( 100 + \log(x - 1) \), etc., so all we can deduce (in the language of differential algebra) is that \( \theta_1 + \theta_2 - \theta_3 \) is a constant \( c \), not necessarily that \( c = 0 \).

2. Hidden algebraics. It is possible to start from a field \( \mathbb{C} \), make \( k \) exponential and logarithmic extensions (Definition 91) of \( \mathbb{C}(x) \) to create \( L \), but have the transcendence degree of \( L \) over \( \mathbb{C}(x) \) be less than \( k \), i.e. for there to be unexpected algebraic elements of \( L \), where we had thought they were transcendental. The obvious example is that \( \sqrt{x} = \exp(\frac{1}{2} \log x) \), but there are more subtle ones, such as the following variant of the exponential example from the previous item. Consider \( L = \mathbb{Q}(x, \theta_1, \theta_2, \theta_3) \) where \( \theta_1' = \theta_1, \theta_2' = 2x\theta_2 \) and \( \theta_3' = (2x + \frac{1}{2})\theta_3 \). Then

\[ \left( \frac{\theta_1 \theta_2^2}{\theta_3^2} \right)' = \frac{\theta_3 \left( \theta_1 \theta_2^2 \right)' - 2\theta_1 \theta_2^3 \theta_3'}{\theta_3^3} = \frac{\theta_3 \theta_1' \theta_2^2 + \theta_3 \theta_1 \theta_2' \theta_2 - 2\theta_1 \theta_2 \theta_3'}{\theta_3^3} = \frac{\theta_1 \theta_2^2 \theta_3 + 4x \theta_1 \theta_2^3 \theta_3 - 2(2x + \frac{1}{2}) \theta_1 \theta_2^2 \theta_3}{\theta_3^3} = 0, \]

so again we have an unexpected constant \( c \). The correct rewriting now is \( \mathbb{Q}(c)(x, \theta_1, \theta_2, \theta_3) \), with \( \theta_3 = \sqrt{c \theta_1 \theta_2^2} \).

3. Ineffective constants. The previous two difficulties, have led to the introduction of “new” constants: what are these? Their values arise from the translation of the language of functions \( \mathbb{R} \to \mathbb{R} \) (or \( \mathbb{C} \to \mathbb{C} \)) to the language of differential algebra. We deduced a constant that might be 1, or might be 100, but equally it might be \( e \), as when the user has both \( e^x \) and \( e^{x+1} \) in an expression, when we have to transform \( e^{x+1} \to e \cdot e^x \) (or \( e^x \to \frac{1}{e} \cdot e^{x+1} \)). This doesn’t seem to be a problem: \( e \) is well-known to be transcendental, so we can effectively regard it as a new indeterminate.

[Rich68] [Ax71] TO BE COMPLETED

The last difficulty is inherent in mathematics, while the first two can be seen as failures of candidness (Definition 5), and are addressed in the next section.

7.3.3 Risch Structure Theorem

[Ris79] TO BE COMPLETED

7.3.4 Overview of Integration

Notation 30 Throughout sections 7.3–7.7, we assume that we are given an integrand \( f \in L = \mathbb{C}(x, \theta_1, \ldots, \theta_n) \), where \( L_{\text{const}} = \mathbb{C} \) is an effective field of constants, each \( \theta_i \) is an elementary generator (Definition 91) over \( \mathbb{C}(x, \theta_1, \ldots, \theta_{i-1}) \).
and, if \( \theta_i \) is given as an exponential or logarithmic generator, then \( \theta_i \) is actually transcendental over \( \mathcal{C}(x, \theta_1, \ldots, \theta_{i-1}) \).

Ideally, the solution of the integration problem would then proceed by induction on \( n \): we assume we can integrate in \( K = \mathcal{C}(x, \theta_1, \ldots, \theta_{n-1}) \), and reduce integration in \( L \) to problems of integration in \( K \). It was the genius of Risch [Ris69] to realise that a more sophisticated approach is necessary.

**Definition 93** For a differential field \( L \), the elementary integration problem for \( L \) is to find an algorithm which, given an element \( f \) of \( L \), finds an elementary overfield \( M \) of \( L \), and \( g \in M \) with \( g' = f \), or proves that such \( M \) and \( g \) do not exist.

**Definition 94** For a differential field \( L \), the elementary Risch differential equation problem for \( L \) is to find an algorithm which, given elements \( f \) and \( g \) of \( L \) (with \( f \) such that \( \exp(\int f) \) is a genuinely new expression, i.e., for any non-zero \( F \) with \( F' = f F \), \( M = L(F) \) is transcendental over \( L \) and has \( M_{\text{const}} = L_{\text{const}} \)), finds \( y \in L \) with \( y' + f y = g \), or proves that such a \( y \) does not exist. We write \( \text{RDE}(f, g) \) for the solution to this problem.

The reader might object “but I can solve this by integrating factors!”. Indeed, if we write \( y = z \exp(\int -f) \), we get

\[
\left( z \exp(\int -f) \right)' + fz \exp(\int -f) = g
\]

\[
z' \exp(\int -f) - fz \exp(\int -f) + fz \exp(\int -f) = g
\]

which simplifies to

\[
z' \exp(\int -f) = g
\]

and hence \( z' = g \exp(\int f) \), \( z = \int g \exp(\int f) \) and

\[
y = \exp(\int -f) \int \ast g \exp(\int f) . \quad (7.33)
\]

However, if we come to apply the theory of section 7.7 to the principal integral (marked \( \int \) of (7.33), we find the integration problem reduces to solving the original \( y' + f y = g \). Hence this problem must be attacked in its own right, which we do in section 7.7 for the base case \( f, g \in C(x) \).

**Theorem 44 (Risch Integration Theorem)** Let \( L = \mathcal{C}(x, \theta_1, \ldots, \theta_n) \), where \( L_{\text{const}} = \mathcal{C} \) is an effective field of constants, each \( \theta_i \) is an elementary generator (Definition 97) over \( \mathcal{C}(x, \theta_1, \ldots, \theta_{i-1}) \) and, if \( \theta_i \) is given as an exponential or logarithmic generator, then \( \theta_i \) is actually transcendental over \( \mathcal{C}(x, \theta_1, \ldots, \theta_{i-1}) \). Then:

(a) we can solve the elementary integration problem for \( L \);
(b) we can solve the elementary Risch differential equation problem for \(L\).

Here the proof genuinely is by induction on \(n\), and, when \(n \neq 0\), the case satisfied by \(\theta_n\) in Definition 91.

(a) \(n = 0\) This was treated in section 7.2.

(b) \(n = 0\) This will be treated in section 7.7.

The Risch induction hypothesis then is that both parts hold for \(C(x, \theta_1, \ldots, \theta_{n-1})\), and we prove them for \(C(x, \theta_1, \ldots, \theta_n)\).

(a) \(\theta_n\) logarithmic This will be treated in section 7.4.

(b) \(\theta_n\) logarithmic See \cite{Dav85a}.

(a) \(\theta_n\) exponential This will be treated in section 7.5.

(b) \(\theta_n\) exponential See \cite{Dav85a}.

(a) \(\theta_n\) algebraic This will be treated in section 7.6.

(b) \(\theta_n\) algebraic See \cite{Bro90, Bro91}. The case \(n = 1\), i.e. algebraic expressions in \(C(x, y)\) with \(y\) algebraic over \(C(x)\), was solved in \cite{Dav84}.

7.4 Integration of Logarithmic Expressions

Let \(\theta = \theta_n\) be a (transcendental) logarithm over \(K = C(x, \theta_1, \ldots, \theta_{n-1})\).

Lemma 11 (Decomposition Lemma (logarithmic)) \(f \in K(\theta)\) can be written uniquely as \(p + q/r\), where \(p\), \(q\) and \(r\) are polynomials of \(K[\theta]\), \(q\) and \(r\) are relatively prime, and the degree of \(q\) is less than that of \(r\). If \(f\) has an elementary integral over \(K\), then \(p\) and \(q/r\) each possess an elementary integral over \(K\).

Proof. By Liouville’s Principle (Theorem 43), if \(f\) is integrable, it is of the form

\[
f = v_0' + \sum_{i=1}^{n} \frac{c_i v_i'}{v_i},
\]

(7.32 bis)

where \(v_0 \in K(\theta)\), \(c_i \in \mathbb{C}\), and \(v_i \in K(c_1, \ldots, c_n)[\theta]\). Write \(v_0 = p_0 + \frac{q_0}{r_0}\), where \(p_0, q_0, r_0 \in K[\theta]\) with \(\deg(q_0) < \deg(r_0)\), and re-arrange the \(v_i\) such that \(v_1, \ldots, v_k \in K(c_1, \ldots, c_n)\), but \(v_{k+1}, \ldots, v_n\) genuinely involve \(\theta\), and are monic.

Then we can re-arrange (7.32 bis) as

\[
p + \frac{q}{r} = \underbrace{p_0 + \sum_{i=1}^{k} \frac{c_i v_i'}{v_i}}_{\text{in } K(c_1, \ldots, c_n)[\theta]} + \underbrace{\left(\frac{q_0}{r_0}\right)'}_{\text{proper rational expression}} + \underbrace{\sum_{i=k+1}^{n} \frac{c_i v_i'}{v_i}},
\]

(7.34)

and the decomposition of the right-hand side proves the result.
This means that it is sufficient to integrate the polynomial part (which we will do in Algorithm 40) and the rational part (which we will do in Algorithm 41) separately, and that failure in either part indicates that the whole expression does not have an elementary integral. In other words, there is no cross-cancellation between these parts.

7.4.1 The Polynomial Part

Let us turn first to the polynomial part. Assume that \( p = \sum_{i=0}^{n} a_i \theta^i \) and \( p_0 = \sum_{i=0}^{m} b_i \theta^i \). The polynomial parts of equation (7.34) then say that

\[
\sum_{i=0}^{n} a_i \theta^i = \sum_{i=0}^{m} b_i' \theta^i + \sum_{i=0}^{m} i b_i \theta^{i-1} + \sum_{i=1}^{k} \frac{c_i v_i'}{v_i} \theta^i \tag{7.35}
\]

independent of \( \theta \).

Hence \( n = m \), except for the special case \( n = m - 1 \) and \( b_m \) constant. If we consider coefficients of \( \theta^n \) (assuming \( n > 0 \)) we have

\[ a_n = b_n' + (n + 1)b_{n+1} \theta'. \]

We can integrate this formally (recalling that \( b_{n+1} \) is constant) to get

\[
\int a_n = b_n + (n + 1)b_{n+1} \theta. \tag{7.36}
\]

But \( a_n \in K \), and, by the Risch induction hypothesis (page 235), we have an integration algorithm for \( K \). In fact, not any integral will do: we want an answer in \( K \) itself, apart possibly from a new logarithmic term of \( \theta \), which will determine \( b_{n+1} \). If \( \int a_n \) contained any other logarithms, then multiplying them by \( \theta^n \) would give us a new logarithm with a non-constant coefficient, which is not allowed by Liouville’s Principle (Theorem 43).

Hence the contribution to \( \int p \) is \( b_n \theta^n + b_{n+1} \theta^{n+1} \). However,

\[
(b_n \theta^n + b_{n+1} \theta^{n+1})' = a_n \theta^n + nb_n \theta^n - 1, \tag{7.37}
\]

so we should subtract \( nb_n \theta^i \) from \( a_{n-1} \). Of course, \( b_n \) is only determined “up to a constant of integration \( \tilde{b}_n \theta^n \). When we come to integrate \( a_{n-1} \), we may get a term \( nb_n \theta^i \), which determines this. The process proceeds until we come to integrate \( a_0 \), when any new logarithms are allowed, and the constant of integration here is that of the whole integration. The corresponding algorithm is given in Figure 7.1.

7.4.2 The Rational Expression Part

Now for the rational expression part, where we have to integrate a proper rational expression, and the integral will be a proper rational expression plus a sum.
7.4. INTEGRATION OF LOGARITHMIC EXPRESSIONS

Algorithm 40 (IntLog–Polynomial)

Input: \( p = \sum_{i=0}^{n} a_i \theta^i \in K[\theta] \).

Output: An expression for \( \int p \, dx \), or failed if not elementary

\[
\text{Ans} := 0 \\
\text{for } i := n, n-1, \ldots, 1 \text{ do} \\
\quad c_i := \int a_i \, dx \quad \# \text{integration in } K: (7.36) \\
\quad \text{if } c_i = \text{failed} \text{ or } c_i \notin K[\theta] \quad \text{return failed} \\
\quad \text{Write } c_i = b_i + (i+1)b_{i+1} \theta: b_i \in K \text{ and } b_{i+1} \text{ constant} \\
\quad \text{Ans} := \text{Ans} + b_i \theta^i + b_{i+1} \theta^{i+1} \\
\quad a_{i-1} := a_{i-1} - i\theta b_i \quad \# \text{correction from } (7.37) \\
\quad c_0 := \int a_0 \, dx \quad \# \text{integration in } K \\
\quad \text{if } c_0 = \text{failed} \quad \text{return failed} \\
\quad \text{Ans} := \text{Ans} + c_0
\]

of logarithms with constant coefficients — see (*) in Algorithm 41. The proper rational expression part is determined by an analogue of Hermite’s algorithm, and (7.18) is still valid.

The Trager–Rothstein algorithm is still valid, with that additional clause that the roots of \( P(\lambda) \), which don’t depend on \( \theta \) since \( P(\lambda) \) is a resultant, actually be constants.

Example 21 To see why this is necessary, consider \( \int \frac{1}{\log x} \, dx \). Here \( q_1 = 1 \), \( r_1 = \theta \), and \( P(\lambda) = \text{Res}(1 - \lambda/x, \theta, \theta) = 1 - \lambda/x \) (made monic, i.e. \( \lambda - x \)). This would suggest a contribution to the integral of \( x \log \log x \), which indeed gives \( \log x \) as one term on differentiation, but also a term of \( \log \log x \), which is not allowed, since it is not present in the integrand.

Note that \( \int \frac{1}{x \log x} \, dx \) gives \( P = \lambda - 1 \) and an integral of \( \log \log x \), which is correct.

7.4.3 Conclusion of Logarithmic Integration

There are essentially three ways in which an expression whose “main variable” is a logarithmic \( \theta \), i.e. which is being written in \( K(\theta) \), can fail to have an elementary integral.

1. In Algorithm 41 some \( P(\lambda) \), whose roots should be the coefficients of the logarithms, turns out to have non-constant roots. Again, this would violate Liouville’s principle. Example 21 is a classic case of this.
Algorithm 41 (IntLog–Rational Expression)

**Input:** \( s_2, t_2 \in K[\theta] \) relatively prime, \( \deg s_2 < \deg t_2 \).

**Output:** An expression for \( \int \frac{s_2}{t_2} \, dx \), or **failed** if not elementary

\[ \text{Ans} := 0 \]

Square-free decompose \( t_2 \) as \( \prod_{i=1}^{k} r_i \)

Write \( \frac{s_2}{t_2} = \sum_{i=1}^{k} \frac{q_i}{r_i} \)

for \( i := 1, \ldots, k \) do

Find \( a_i, b_i \) such that \( a_i r_i + b_i r_i' = 1 \)

for \( j := i, i-1, \ldots, 2 \) do

\[ \text{Ans} := \text{Ans} - q_i b_i \left( \frac{1}{(j-1)r_i^j} \right) \quad \#(7.18) \]

\[ q_i := q_i a_i + \left( q_i b_i / (j-1) \right)' \]

\[ P(\lambda) := \text{Res}(q_i - \lambda r_i', r_i, \theta) \text{ (made monic)} \]

if \( P \) has non-constant coefficients (*)

return **failed**

Write \( P(\lambda) = \prod_j Q_j(\lambda)^j \) (square-free decomposition)

for \( j \) in \( 1 \ldots \) such that \( Q_j \neq 1 \)

\[ v_j := \gcd(q_i - \lambda r_i', r_i) \text{ where } Q_j(\lambda) = 0 \quad \# \deg(v_j) = j \]

\[ \text{Ans} := \text{Ans} + \sum_{\lambda \text{ a root of } Q_j} \lambda \log v_j(x) \]
7.5. INTEGRATION OF EXPONENTIAL EXPRESSIONS

Notation 31 It is usual to define $\int \frac{1}{\log x}$ to be the logarithmic integral function $\text{li}(x)$, but this is not an elementary function in the sense of Definition [92].

2. In Algorithm 40, some $a_i \in K \ (i > 0)$ may have an elementary integral, but one that involves ‘new’ logarithms other than $\theta$, which would violate Liouville’s Principle.

Example 22 Consider $\int \frac{1}{x} \log(x+1)$ with $\theta$ being $\log(x+1)$. Then $a_1 = \frac{1}{x}$, and $\int a_1 = \log(x)$, which is not in the original field. If we allow this, and suggest that the integral has a term of $\log(x) \log(x+1)$, its derivative also has a term of $\frac{1}{x+1} \log(x)$, and we seem to be trapped in a vicious cycle.

3. In Algorithm 40 some $a_i \in K$ may itself fail to have an elementary integral: this failure cannot be compensated for elsewhere, by Lemma [11].

Example 23 Consider $\int \frac{1}{\log x} \log(x+1)$ with $\theta$ being $\log(x+1)$. Then $a_1 = \frac{1}{\log x}$, and, as we have seen in Example [21], this does not have an elementary integral.

Even if we use the logarithmic integral $\text{li}(x)$ here, and claim that $\int \frac{1}{\log x} = \text{li}(x)$, we get a term in the integral of $\text{li}(x) \log(x+1)$, whose derivative also has a term of $\frac{1}{x+1} \log(x)$, and we now need a theory of integrating $\text{li}$-involving functions, which is beyond the scope of this book and is, as far as the author knows, unsolved (note that [Che86] only considers elementary integrands).

### 7.5 Integration of Exponential Expressions

Throughout this section, we let $\theta = \theta_n$ be a (transcendental) exponential over $K = \mathcal{C}(x, \theta_1, \ldots, \theta_{n-1})$, so that $\theta' = \eta \theta$. We should note that this choice is somewhat arbitrary, since $\theta^{-1} = \overline{\theta}$ satisfies $\overline{\theta}' = -\eta \overline{\theta}$ and $K(\overline{\theta}) \equiv K(\theta)$. Hence negative powers of $\theta$ are just as legitimate as positive powers, and this translates into a difference in the next result: rather than writing expressions as “polynomial + proper rational expressions”, we will make use of the following concept.

Definition 95 A generalised (or Laurent) polynomial in $\theta$ over $K$ is a sum $\sum_{i=-m}^{n} a_i \theta^i$ with $a_i \in K$.

Lemma 12 (Decomposition Lemma (exponential)) $f \in K(\theta)$ can be written uniquely as $p+q/r$, where $p$ is a Laurent polynomial, $q$ and $r$ are polynomials of $K[\theta]$ such that $\theta$ does not divide $r$, $q$ and $r$ are relatively prime, and the degree of $q$ is less than that of $r$. If $f$ has an elementary integral over $K$, then each of the terms of $p$, and also $q/r$, have an elementary integral over $K$. 

Proof. By Liouville’s Principle (Theorem 43), if \( f \) is integrable, it is of the form

\[
  f = v_0' + \sum_{i=1}^n c_i v_i' / v_i,
\]

(7.32 ter)

where \( v_0 \in K(\theta) \), \( c_i \in \mathbb{C} \), and \( v_i \in K(c_1, \ldots, c_n)[\theta] \). Write \( v_0 = p_0 + \frac{q_0}{r_0} \), where \( p_0 \in K[\theta, 1/\theta] \) is a Laurent polynomial, \( q_0, r_0 \in K[\theta] \) with \( \deg(q_0) < \deg(r_0) \) and \( \theta \not| q_0 \), and re-arrange the \( v_i \) such that \( v_1, \ldots, v_k \in K(c_1, \ldots, c_n) \), but \( v_{k+1}, \ldots, v_n \) genuinely involve \( \theta \), and are monic. Furthermore, we can suppose that \( \theta \) does not divide any of these \( v_i \), since \( \log \theta = \eta \) (up to constants). Unlike Lemma 11, though, it is no longer the case that \( (\log v_i)' \) (\( i > k \)) is a proper rational expression. Let \( n_i \) be the degree (in \( \theta \)) of \( v_i \), then, recalling that we have supposed that the \( v_i \) are monic, \( v_i' = n_i \eta' \theta^{n_i} + \) lower terms, and \( (v_i' - n_i \eta' v_i)/v_i \) is a proper rational expression.

Then we can re-arrange (7.32 ter) as

\[
  p + \frac{q}{r} = p_0' + \sum_{i=1}^k \frac{c_i v_i'}{v_i} + \sum_{i=k+1}^n c_i n_i \eta' + \left( \frac{q_0}{r_0} \right)' + \sum_{i=k+1}^n \left( \frac{c_i (v_i' - n_i \eta' v_i)}{v_i} \right),
\]

(7.38)

and the decomposition of the right-hand side proves the result.

This means that it is sufficient to integrate the polynomial part (which we will do in Algorithm 42) and the rational part (which we will do in Algorithm 43) separately, and that failure in either part indicates that the whole expression does not have an elementary integral. In other words, there is no cross-cancellation between these parts.

7.5.1 The Polynomial Part

In fact, this is simpler than the logarithmic case, since Lemma 12 says that each summand \( a_i \theta^i \) has to be integrable separately. The case \( i = 0 \) is just integration in \( K \), and all the others are cases of the Risch differential equation problem (Definition 94).

This translates straightforwardly into Algorithm 12.

7.5.2 The Rational Expression Part

The last equation (7.38) of Lemma 12 says that

\[
  \int \frac{q}{r} = \left( \sum_{i=k+1}^n c_i n_i \right) \eta + \left( \frac{q_0}{r_0} \right)' + \sum_{i=k+1}^n c_i (\log v_i - n_i \eta),
\]

(7.39)

where the \( v_i \) are monic polynomials (not divisible by \( \theta \)) of degree \( n_i \). The proper rational expression part is determined by an analogue of Hermite’s algorithm,
7.5. INTEGRATION OF EXPONENTIAL EXPRESSIONS

Figure 7.3: Algorithm [42] IntExp–Polynomial

Algorithm 42 (IntExp–Polynomial)

Input: \( p = \sum_{i=-m}^{n} a_i \theta^i \in K[\theta], \) where \( \theta = \exp \eta. \)

Output: An expression for \( \int p \, dx, \) or failed if not elementary

Ans:=0
for \( i := -m, \ldots, -1, 1, \ldots, n \) do
    \( b_i := \text{RDE}(i\eta', a_i) \)
    if \( b_i = \text{failed} \)
        return failed
    else Ans:=Ans+\( b_i \theta^i \)
\( c_0 := \int a_0 \, dx \) # integration in \( K \)
if \( c_0 = \text{failed} \)
    return failed
\( \text{Ans} := \text{Ans} + c_0 \)

and (7.18) is still valid, though we should point out that the justification involved stating that \( \gcd(r_i, r'_i) = 1, \) where the \( r_i \) were the square-free factors of \( r_0. \) Since \( \theta' = \eta \theta, \) this is no longer true if \( \theta | r_i, \) but this is excluded since such factors were moved into the Laurent polynomial (Definition [95]) part.

Hence the Hermite part of Algorithm [43] is identical to the rational and logarithmic cases. The Trager–Rothstein part is slightly more complicated, since \( v'_i/v_i \) is no longer a proper rational expression, which is the cause of the term marked ‘correction” in (7.39). Suppose we have a term \( s_2/t_2 \) (in lowest terms: the programmer must not forget this check) left after Hermite’s algorithm, and write

\[
\int \frac{s_2}{t_2} = \sum c_i (\log v_i - n_i \eta),
\]

which can be differentiated to

\[
\frac{s_2}{t_2} = \sum c_i \frac{v'_i - n_i \eta' v_i}{v_i}.
\]

Hence \( t_2 = \prod v_i, \) which means \( N := \deg t_2 = \sum n_i, \) and, writing \( u_j = \prod_{i \neq j} v_i, \) we can write as

\[
s_2 = \sum c_i (v'_i - n_i \eta' v_i) u_i.
\]

Furthermore, since \( t_2 = \prod v_i, t'_2 = \sum v'_i u_i. \) Hence

\[
v_k = \gcd(0, v_k) = \gcd \left( s_2 - \sum c_i (v'_i - n_i \eta' v_i) u_i, v_k \right) = \gcd \left( s_2 - c_k (v'_k - n_k \eta' v_k) u_k, v_k \right)
\]
since all the other \( u_i \) are divisible by \( v_k \)
Algorithm 43 (IntExp–Rational Expression)

Input: \( q, r \in K[\theta] \) relatively prime, \( \deg q < \deg r \).

Output: An expression for \( \int \frac{q}{r} \, dx \), or failed if not elementary

\[
\text{Ans} := 0
\]

Square-free decompose \( r \) as \( \prod_{i=1}^{k} r_i^j \)

Write \( \frac{q}{r} = \sum_{i=1}^{k} \frac{q_i}{r_i} \)

for \( i := 1, \ldots, k \) do

Find \( a_i, b_i \) such that \( a_i r_i + b_i r_i' = 1 \)

for \( j := i, i-1, \ldots, 2 \) do

\[
\text{Ans} := \text{Ans} - \frac{q_i b_i}{(j-1) r_i'} \quad \#(7.18)
\]

Cancel any common factor between \( q_i \) and \( r_i \)

\( P(\lambda) := \text{Res}(q_i - \lambda (r_i'^2 - (\deg r_i') \eta / r_i), r_i, \theta) \) (made monic)

if \( P \) has non-constant coefficients \(*\)

return failed

Write \( P(\lambda) = \prod_j Q_j(\lambda)^j \) (square-free decomposition)

for \( j \) in \( 1 \ldots \) such that \( Q_j \neq 1 \)

\( v_j := \gcd(q_i - \lambda r_i', r_i) \) where \( Q_j(\lambda) = 0 \quad \#\text{deg}(v_j) = j \)

\[
\text{Ans} := \text{Ans} + \sum_{\lambda \text{ a root of } Q_j} \lambda \log v_j(x)
\]
7.5. INTEGRATION OF EXPONENTIAL EXPRESSIONS

\[ \begin{align*}
7.5.1 & \quad \gcd \left( s_2 - c_k \sum (v'_i - n_i \eta' v_i) u_i, v_k \right) \\
    & = \gcd (s_2 - c_k (t'_2 - t_2 N \eta'), v_k) \text{ for the same reason} \\
    & = \gcd (s_2 - c_k (t'_2 - t_2 N \eta'), v_k) \\
    & \text{since } t_2 = v_i u_i \text{ and } N = \sum n_i
\end{align*} \]

But if \( l \neq k \),

\[ \gcd (s_2 - c_k (t'_2 - t_2 N \eta'), v_l) = \gcd \left( \sum c_i v'_i u_i - c_k \sum v'_i u_i, v_l \right) \]

\[ t_2 N \eta' \text{ disappears as } v_i | t_2 \]

\[ = \gcd (c_l v'_l u_l - c_k v'_l u_l, v_l) \]

\[ \text{since all the other } u_i \text{ are divisible by } v_l \]

\[ = 1. \]

Since \( t_2 = \prod v_i \), we can put these together to deduce that

\[ v_k = \gcd (s_2 - c_k (t'_2 - N \eta' t_2), t_2). \quad (7.43) \]

Given \( c_k \), this will tell us \( v_k \). But we can deduce more from this: the \( c_k \) are precisely those numbers \( \lambda \) such that \( \gcd (s_2 - \lambda (t'_2 - N \eta' t_2), t_2) \) is non-trivial. Hence we can appeal to Proposition 7.3 (page 268), and say that \( \lambda \) must be such that

\[ P(\lambda) := \text{Res}_\theta (s_2 - \lambda (t'_2 - N \eta' t_2), t_2) = 0. \quad (7.44) \]

As \( t_2 \) has degree \( N \) and \( t'_2 - N \eta' t_2 \) has degree \( N - 1 \), \( P(\lambda) \) is the determinant of a \( 2N - 1 \) square matrix, \( N \) of whose rows depend linearly on \( \lambda \), and thus is a polynomial of degree \( N \) in \( \lambda \) (strictly speaking, this argument only shows that it has degree at most \( N \), but the coefficient of \( \lambda^N \) is \( \text{Res}(t'_2 - N \eta' t_2, t_2) \neq 0) \).

Hence, while we needed to re-prove the result, the application of Trager–Rothstein is little different from the logarithmic, and indeed rational expression, case. As in the logarithmic case, we have the caveat (*) that the roots of \( P \) must be constant. An analogous example to that of section 7.4.2 is that of \( \int x \frac{1}{1 + e^x} \).

Here \( s_2 = x \), \( t_2 = 1 + \theta \) and

\[ \text{Res}_\theta (s_2 - \lambda (t'_2 - N \eta' t_2), t_2) = \text{Res}_\theta (x - \lambda (\theta - (1 + \theta)), 1 + \theta) = x + \lambda. \]

This would suggest a contribution of \(-x \log(1+e^x)\), but this also leaves \(-\log(1+e^x)\) on differentiating, which contradicts Liouville’s Principle. Computer algebra systems will give answers such as Maple’s

\[ \int \frac{x}{1 + e^x} \, dx = \frac{1}{2} x^2 - x \ln (1 + e^x) - \text{polylog} (2, -e^x), \quad (7.45) \]

but we have, essentially, just proved that \( \text{polylog} (2, -e^x) \) is not elementary.
7.6 Integration of Algebraic Expressions

The integration of algebraic expressions is normally taught as a bunch of, apparently ad hoc tricks. A typical calculation would be

\[ \int \frac{1}{\sqrt{1-x^2}} \, dx = \int \frac{1}{\sqrt{1-\sin^2 t}} \, dt \, \frac{dt}{\cos t} = \int \frac{\cos t}{\cos t} \, dt \]

substituting \( x = \sin t \)

which we can write as \( 1/2 \pi + i \log(\sqrt{1-x^2} + ix) \) to show that it really is elementary, and furthermore does not violate Liouville’s Principle, since the only new expression is now seem to be a logarithm with a constant coefficient. The \( \pi \) is, of course, just a particular choice of constant of integration. This leaves numerous questions.

1. Why \( x = \sin t \)? \( x = \cos t \) seems to work, but gives a different result.
2. How do I know which substitutions will work?
3. How can I tell when no substitutions work?
4. What about cases when it only partially works, such as

\[ \int \frac{1}{\sqrt{1-x^2} \sqrt{1-2x^2}} \, dx = \int \frac{1}{\sqrt{1-2\sin^2 t}} \, dt =? \]

5. These inverse functions have branch cuts — what does that mean for the result?

The last question will be taken up in Chapter 8. The others are material for this section, and indeed seem difficult, to the point where the great 20th century number theorist Hardy [Har16] even conjectured that there was no general method. Integration theory in the 1960s, typified by the SAINT program [Sla61] concentrated on “doing it like a human”, i.e. by guessing and substitution.

Risch [Ris70] observed that this wasn’t necessary, and [Dav81] [Tra84] converted his observations into algorithms, and implementations.

TO BE COMPLETED

7.7 The Risch Differential Equation Problem

In this section we solve case (b), \( n = 0 \) of Theorem 44, viz.

\footnote{Another case of simultaneous discovery.}
to find an algorithm which, given elements $f$ and $g$ of $C(x)$ (with $f$ such that $\exp(\langle f \rangle)$ is a genuinely new expression, i.e., for any non-zero $F$ with $F' = fF$, $M = C(x, F)$ is transcendental over $C(x)$ and has $M_{\text{const}} = C$), finds $y \in C(x)$ with $y' + fy = g$, or proves that such a $y$ does not exist. This is written as $\text{RDE}(f, g)$.

These conditions on $f$ mean that it is not a constant, and its integral is not purely a sum of logarithms with rational number coefficients.

We will first consider the denominator of $y \in C(x)$. We could assume that $C$ is algebraically closed, so that all polynomials factor into lines, but in fact we need not. We will assume that we can factor polynomials, though we will see afterwards that this is algorithmically unnecessary.

Let $p$ be an irreducible polynomial. Let $\alpha$ be the largest integer such that $p^\alpha$ divides the denominator of $y$, which we can write as $p^\alpha \parallel \text{den}(y)$. Let $\beta$ and $\gamma$ be such that $p^\beta \parallel \text{den}(f)$ and $p^\gamma \parallel \text{den}(g)$. So we can calculate the powers of $p$ which divide the terms of the equation to be solved:

$$y' + fy = g.$$  \hspace{1cm} (7.49)

There are then three possibilities.

1. $\beta > 1$. In this case the terms in $p^{\alpha + \beta}$ and $p^\gamma$ have to cancel, that is we must have $\alpha = \gamma - \beta$.

2. $\beta < 1$ (in other words, $\beta = 0$). In this case the terms in $p^{\alpha + 1}$ and $p^\gamma$ must cancel, that is, we must have $\alpha = \gamma - 1$.

3. $\beta = 1$. In this case, it is possible that the terms on the left-hand side cancel and that the power of $p$ which divides the denominator of $y' + fy$ is less than $\alpha + 1$. If there is no cancellation, the result is indeed $\alpha = \gamma - \beta$.

So let us suppose that there is a cancellation. We express $f$ and $y$ in partial fractions with respect to $p$: $f = F/p^\beta + \hat{f}$ and $y = Y/p^\alpha + \hat{y}$, where the powers of $p$ which divide the denominators of $f$ and $y$ are at most $\beta - 1 = 0$ and $\alpha - 1$, and $F$ and $Y$ have degree less than that of $p$.

$$y' + fy = -\frac{\alpha Y'}{p^{\alpha + 1}} + \frac{Y'}{p^\alpha} + \frac{FY}{p^{\alpha + 1}} + \frac{\hat{f}Y}{p^\alpha} + \frac{\hat{f}y}{p} + \hat{f}y.$$  \hspace{1cm} (7.49)

For there to be a cancellation in this equation, $p$ must divide $-\alpha p'Y + FY$.

But $p$ is irreducible and $Y$ is of degree less than that of $p$, therefore $p$ and $Y$ are relatively prime. This implies that $p$ divides $\alpha p' - F$. But $p'$ and $F$ are of degree less than that of $p$, and the only polynomial of degree less than that of $p$ and divisible by $p$ is zero. Therefore $\alpha = F/p'$.

Putting these together proves the following result.

**Lemma 13 (Ris69)** $\alpha \leq \max(\min(\gamma - 1, \gamma - \beta), F/p')$, where the last term only applies when $\beta = 1$, and when it gives rise to a positive integer.
In fact, it is not necessary to factorise the denominators into irreducible polynomials. It is enough to find square-free polynomials \( p_i \), relatively prime in pairs, and non-negative integers \( \beta_i \) and \( \gamma_i \) such that \( \text{den}(f) = \prod p_i^{\beta_i} \) and \( \text{den}(g) = \prod p_i^{\gamma_i} \). When \( \beta = 1 \), we have, in theory, to factorise \( p \) completely, but it is enough to find the integral roots of \( \text{Res}_x(F - zp', p) \), by an argument similar to Trager’s algorithm for calculating the logarithmic part of the integral of a rational expression.

We have, therefore, been able to bound the denominator of \( y \) by \( D = \prod p_i^{\alpha_i} \), so that \( y = Y/D \) with \( Y \) polynomial. So it is possible to suppress the denominators in our equation, and to find an equation

\[
RY' + SY = T. \tag{7.50}
\]

Let \( \alpha, \beta, \gamma \) and \( \delta \) be the degrees of \( Y, R, S \) and \( T \). Then (7.50) becomes

\[
\frac{RY'}{\alpha + \beta - 1} + \frac{SY}{\alpha + \gamma} = \frac{T}{\delta}. \tag{7.51}
\]

There are again three possibilities\(^8\):

1. \( \beta - 1 > \gamma \). In this case, the terms of degree \( \alpha + \beta - 1 \) must cancel out the terms of degree \( \delta \), therefore \( \alpha = \delta + 1 - \beta \).

2. \( \beta - 1 < \gamma \). In this case, the terms of degree \( \alpha + \gamma \) must cancel out the terms of degree \( \delta \), therefore \( \alpha = \delta - \gamma \).

3. \( \beta - 1 = \gamma \). In this case, the terms of degree \( \alpha + \beta - 1 \) on the left may cancel. If not, the previous analysis still holds, and \( \alpha = \delta + 1 - \beta \). To analyse the cancellation, we write \( Y = \sum_{i=0}^{\alpha} y_i x^i \), \( R = \sum_{i=0}^{\beta} r_i x^i \) and \( S = \sum_{i=0}^{\gamma} s_i x^i \). The coefficients of the terms of degree \( \alpha + \beta - 1 \) are \( \alpha r_\beta y_\alpha \) and \( s_\gamma y_\alpha \). The cancellation is equivalent to \( \alpha = -s_\gamma/r_\beta \).

**Lemma 14** ([Ris69]) \( \alpha \leq \max(\min(\delta - \gamma, \delta + 1 - \beta), -s_\gamma/r_\beta) \), where the last term is included only when \( \beta = \gamma + 1 \), and only when it gives rise to a positive integer.

Determining the coefficients \( y_i \) of \( Y \) is a problem of linear algebra. In fact, the system of equations is triangular, and is easily solved.

**Example 24** Let us consider \( y' - 2xy = 1 \), i.e. \( f = -2x \), \( g = 1 \). Since neither \( f \) nor \( g \) have any denominator, \( y \) does not, i.e. it is purely polynomial, of degree \( \alpha \), and \( R = 1 \), \( S = -2x \) and \( T = 1 \) in (7.50). Comparing degrees gives us

\[
\frac{y'}{\alpha} + \frac{-2xy}{\alpha + 1} = \frac{1}{0}, \tag{7.52}
\]

i.e. \( \alpha = -1 \), as predicted by Lemma 14. But this is impossible.

This is the Risch Differential equation that arises from integrating \( \exp(-x^2) \), whose integral has to be \( y \exp(-(x^2)) \). Hence we have proved the folk-lore result:

\[^8\text{This is not an accidental similarity, but we refer to [Dav84] for a unifying exposition.}\]
e\(^{-x^2}\) has no [elementary] integral.

We can introduce a new, non-elementary, operator erf, with the property\(^9\) that \((\text{erf}(\eta))' = \eta' \exp(-\eta^2)\), and then \(\int \exp(-x^2) = \text{erf}(x)\).

The case when \(K\) is an algebraic extension of \(C(x)\) is treated in \([\text{Dav84}]\), and the more general cases in \([\text{Ris69, Dav85b}]\). The principles are the same, but the treatment of the cancellation case gets more tedious, both in theory and in practice.

### 7.8 The Parallel Approach

The methods described so far this chapter have essentially taken a recursive approach to the problem: if the integrand lives in \(K(x, \theta_1, \ldots, \theta_n)\), we have regarded this as \(K(x)\langle\theta_1\rangle(\ldots)\langle\theta_n\rangle\), and the results of the previous sections all justify an "induction on \(n\)" algorithm. In this section we will adopt a different approach, regarding \(K(x, \theta_1, \ldots, \theta_n)\) as the field of fractions of \(K[x, \theta_1, \ldots, \theta_n]\), and taking a distributed (page \[41\]), also called parallel as we are treating all the \(\theta_i\) in parallel, view of \(K[x = \theta_0, \theta_1, \ldots, \theta_n]\).

We note that Liouville’s Principle (Theorem \[43\]) can be applied to this setting.

**Theorem 45 (Liouville’s Principle, Parallel version)** Let \(f\) be a expression from some expression field \(L = K(x, \theta_1, \ldots, \theta_n)\). If \(f\) has an elementary integral over \(L\), it has an integral of the following form:

\[
\int f = v_0 + \sum_{i=1}^{m} c_i \log v_i, \tag{7.53}
\]

where \(v_0\) belongs to \(L\), the \(v_i\) belong to \(\hat{K}[x, \theta_1, \ldots, \theta_n]\), an extension of \(K[x, \theta_1, \ldots, \theta_n]\) by a finite number of constants algebraic over \(K\), and the \(c_i\) belong to \(\hat{K}\).

Other that being more explicit about the domains objects belong to, we are asserting that the \(v_i\) are polynomial, which can be assumed since \(\log(p/q) = \log(p) - \log(q)\), or in the language of differential algebra,

\[
\frac{(p/q)'}{p/q} = \frac{p'}{p} - \frac{q'}{q}.
\]

**Notation 32** In \([7.53]\), we write \(v_0 = \frac{p}{q}\), where \(p, q \in K[x, \theta_1, \ldots, \theta_n]\), but are not necessarily relatively prime.

\(^9\)The reader may object that \(\int e^{-x^2} = \frac{\sqrt{\pi}}{2} \text{erf}(x)\). This statement is only relevant after one has attached precise numeric meanings \(e^{-x^2}\) to the differential-algebraic concept \(\exp(-x^2)\) — see Chapter \[8\]
From now on, until section 7.8.1 we will assume that the \( \theta_i \) are transcendental. We can then infer the following algorithm for integrating expressions in this setting.

**Pseudo-Algorithm 44 (Parallel Risch [NM77, Bro07])**

**Input:** \( L = K(x, \theta_1, \ldots, \theta_n) \) a purely transcendental differential field with constants \( K, f \in L \)

**Output:** \( g \) an elementary integral of \( f \), or a proof that there is no such one satisfying the assumptions of steps (1)–(3).

1. Decide candidate \( v_1, \ldots, v_m \) (we may have too many)
2. Decide a candidate \( q \) (which may be a multiple of the true value)
3. Decide degree bounds for \( p \) (which may be too large), i.e. \( n_0, \ldots, n_n \) such that
   \[
   p = \sum_{i_0=0}^{n_0} \sum_{i_1=0}^{n_1} \cdots \sum_{i_n=0}^{n_n} c_{i_0, i_1, \ldots, i_n} x^{i_0} \prod_{j=1}^{n} \theta_j^{i_j}
   \]
4. Clear denominators in the derivative of (7.53)
5. Solve the resulting linear equations for the \( c_i \) and \( c_{i_0, i_1, \ldots, i_n} \)
   \( (c_0, \ldots, 0 \text{ is the the constant of integration, and is never determined}) \)
6. **if** there’s a solution **then** reduce \( p/q \) to lowest terms and return the result
   **else** “integral not found”

As explained in [Bro07], it is the decisions taken in steps (1)–(3) which mean that this is not a guaranteed “integrate or prove unintegrable” algorithm. The work of the previous sections allows partial solutions:

1. Those \( v_1, \ldots, v_m \) which depend on \( \theta_n \)
2. A candidate \( q \) (which may be a multiple of the true value)
   But the multiple is in \( K(x, \theta_1, \ldots, \theta_{n-1})[\theta_n] \), not in \( K[x, \theta_1, \ldots, \theta_n] \)
3. A degree bound \( n_n \) for \( p \) as a polynomial in \( \theta_n \)

### 7.8.1 The Parallel Approach: Algebraic Expressions

### 7.9 Definite Integration

We have shown (Example 24) that \( \int \exp(-x^2) \) has no elementary expression, i.e. that it is a new expression, which we called erf. However, \( \int_{-\infty}^{\infty} e^{-x^2} \) (where we have attached a precise numerical meaning \( e^x \) to \( \exp(x) \)) is well-known to be \( \pi \), which is essentially another way of saying that erf(\( \pm \infty \)) = \( \pm 1 \), and is therefore a matter of giving numerical values to functions — see Chapter 8.

TO BE COMPLETED: Meijer etc.
7.10 Other Calculus Problems

7.10.1 Indefinite summation

The initial paper in this area was [Gos78]. Although not expressed that way in this paper, the key idea [Kar81] is the following, akin to Definition 89.

**Definition 96** A difference ring is a ring (Definition 6) equipped with an additional unary operation, referred to as differentiation and written \( \delta \), which satisfies three additional laws:

1. \( \delta(f + g) = \delta f + \delta g \);
2. \( \delta(fg) = f(\delta g) + (\delta f)g + (\delta f)(\delta g) \);
3. \( \delta(1) = 0 \).

A difference ring which is also a field (Definition 13) is referred to as a difference field.

One thinks of \( \delta(f) \) as being \( f(n+1) - f(n) \). It is worth remarking the differences with Definition 89: clause 3 is necessary, whereas previously it could be inferred from the others, and clause 2 is different, essentially because \( (x^2)' = 2x \) but \( \delta(n^2) = (n + 1)^2 - n^2 = 2n + 1 \).

The process of formal (indefinite) summation \( \sum \) is then that of inverting \( \delta \), as \( \int \) is the process of inverting \( ' \). Just as the rôle of \( dx \) in integration theory is explained by the fact that \( x' = 1 \), i.e. \( \int 1 = x \), so the traditional rôle of \( n \) in summation theory is explained by the fact that \( \delta n = 1 \), equivalently that \( \sum 1 = n \). Note that the two procedures of integration and summation are more similar than normal notation would suggest: we are happy with \( \int x = \frac{1}{2}x^2 \), but less so with \( \sum n = \frac{1}{2}n(n + 1) \), yet the two are essentially equivalent.

To further the analogy, we know that \( \int \frac{1}{x} \) cannot be expressed as a rational function of \( x \), but is a new expression, known conventionally as \( \log x \). Similarly, \( \sum \frac{1}{n} \) cannot be expressed as a rational function of \( n \) [Kar81, Example 16], but is a new expression, known conventionally as \( H_n \), the \( n \)-th harmonic number. Again, \( \int \log x = x\log x - x \), and \( \sum H_n = nH_n - n \).

7.10.2 Definite Symbolic Summation

Definite summation might be thought to be related to indefinite summation in the way that definite integration (Section 7.9) is to indefinite. There is certainly the same problem of evaluating expressions from difference algebra at numerical points. However, in practice we are more concerned with a subtly different class of problems, where the limits of summation also figure in the summand.

**Example 25** ([GKP94, Exercise 6.69], [Sch00b])

\[
\sum_{k=1}^{n} k^2H_{n+k} = \frac{1}{3}n\left(n + \frac{1}{2}\right)(n + 1)(2H_{2n} - H_n) - \frac{1}{36}n(10n^2 + 9n - 1). \quad (7.54)
\]

\[10\text{In this text: notations differ.}\]
What are the key references here? \[\text{Sch04}\]?

7.10.3
7.10.4
Chapter 8

Algebra versus Analysis

We have seen in the previous chapter how we can construct an algebraic theory of mathematical objects such as ‘exp’ and ‘log’, and possibly others. From an algebraic point of view, they seem to behave like the mathematical objects we are familiar with from analysis. Are they the same? If not, what are the differences? This is perhaps one of the less-discussed topics in computer algebra, and indeed possibly in mathematics more generally.

Notation 33 Throughout this chapter, we use the notation \( \equiv \) to denote an equation that might or might not be true, or partially true, depending on the interpretations, from algebra or from analysis, that one places on the symbols either side of \( \equiv \).

8.1 Functions and Formulae

This question turns out to be related to the difference between functions and formulae (which we have also called expressions). Consider the two-dimensional formula \( x^2 - 1 \), or \( (x^2-1)/(x-1) \) if we prefer one-dimensional expressions. It has potentially many roles.

formula There are several options here: strings of characters, or a parse tree. Whichever we choose, \( (x^2-1)/(x-1) \) is a different formula from \( x+1 \).

\( \in \mathbb{Q}(x) \) (see section 2.2) This is therefore mathematically equivalent to \( x + 1 \), and an algebra system may or may not transform one into the other: a system that aims for candidness in this context (section 2.2.2) should transform \( (x^2-1)/(x-1) \) into \( x+1 \).

\( \in K(x) \) Of course, it is only convention that chooses \( \mathbb{Q} \) for the ground field. It could be any extension of \( \mathbb{Q} \), or, more challengingly, a finite field of positive characteristic.

\footnote{But see [Dav10].}
chapter 8. algebra versus analysis

**rule** This is what a computer scientist would think of as \( \lambda x. \frac{x^2-1}{x-1} \): that rule which, given an \( x \), computes the corresponding value of the formula

None of these are, as such, a function in the sense of Notation 3, though the last is the nearest. However, trying to express \( \frac{x^2-1}{x-1} \) in this notation exposes our problems.

\[
\left\{ \left( x, \frac{x^2-1}{x-1} \right) \mid x \in \mathbb{Q} \right\}_B
\]

is illegal, because of the case \( x = 1 \). Ruling this out gives us

\[
\left\{ \left( x, \frac{x^2-1}{x-1} \right) \mid x \in \mathbb{Q} \setminus \{1\} \right\}_B.
\]

If we regard \( 0 \) as \( \perp \), then we can have

\[
\left\{ \left( x, \frac{x^2-1}{x-1} \right) \mid x \in \mathbb{Q} \right\}_B, \mathbb{Q}, \mathbb{Q} \cup \{\perp\}
\]

Making the \( \perp \) explicit gives us

\[
\left\{ \left( x, \frac{x^2-1}{x-1} \right) \mid x \in \mathbb{Q} \right\}_B, \mathbb{Q}, \mathbb{Q} \cup \{\perp\}
\]

If we’re going to single out a special case, we might as well (since \( 2 = \lim_{x \to 1} \frac{x^2-1}{x-1} \), i.e. this is a *removable singularity*, in the sense that we can give our function a value at the apparently singular point which makes it continuous) write

\[
\left\{ \left( x, \frac{x^2-1}{x-1} \right) \mid x \in \mathbb{Q} \setminus \{1\} \right\}_B \cup \{(1,2)\}, \mathbb{Q}, \mathbb{Q} \cup \{\perp\}
\]

dropping the \( \perp \), and this is equal to

\[
\left\{ \left( x, x+1 \right) \mid x \in \mathbb{Q} \right\}_B.
\]

The case of polynomials is simpler (we restrict consideration to one variable, but this isn’t necessary). Over a ring \( R \) of characteristic zero\(^2\), equality of abstract polynomials is the same as equality of the corresponding functions:

\[
p = q \text{ in } R[x] \Leftrightarrow \left( \{ (x, p(x)) \mid x \in R \}, R, R \right)_B = \left( \{ (x, q(x)) \mid x \in R \}, R, R \right)_B
\]

This is a consequence of the fact that a non-zero polynomial \( p - q \) has only a finite number of zeros.

If we attach a meaning to elements of \( R(x) \) by analogy with \( [8.2] \), omitting dubious points in the domain, then equality in the sense of Definition 14 is related to the equality of Bourbakist functions in the following way

\[
f = g \text{ in } R(x) \Leftrightarrow \left( \{ (x, f(x)) \mid x \in S \}, S, R \right)_B = \left( \{ (x, g(x)) \mid x \in S \}, S, R \right)_B
\]

where \( S \) is the intersection of the domains of \( f \) and \( g \).

\(^2\)See example 5 on page 22 to explain this limitation.
8.2 Branch Cuts

In the previous section, we observed that there was a difference between “expressions”, whether concrete formulae or abstract expressions in $K(x)$, and functions $\mathbb{R} \to \mathbb{R}$ or $\mathbb{C} \to \mathbb{C}$. If we go beyond $K(x)$, the problems get more serious.

**Example 26** Consider the formula $\sqrt{z}$, or $\sqrt{z}$, which we can formalise as $\theta \in K(z, \theta | \theta^2 = z)$. What happens if we try to interpret $\theta$ as a function $\theta(z) : \mathbb{C} \to \mathbb{C}$? Presumably we choose $\theta(1) = 1$, and, by continuity, $\theta(1 + \epsilon) = 1 + \frac{1}{2} \epsilon - \frac{1}{8} \epsilon^2 + \cdots \approx 1 + \frac{1}{2} \epsilon$. Similarly, $\theta(1 + i \epsilon) \approx 1 + \frac{1}{2} i \epsilon$, and so on. If we continue to track $\theta(z)$ around the unit circle $\{ z = x + iy : x^2 + y^2 = 1 \}$, we see that $\theta(i) = \frac{1+i}{\sqrt{2}}$, and $\theta(1) = i$. Continuing, $\theta(-i) = i \pi$, and ultimately $\theta(1) = 2 \pi i$.

It would be tempting to dismiss this as “the usual square root ambiguity”, but in fact the problem is not so easily dismissed.

**Example 27** Similarly, consider $\log(z)$, which we can formalise as $\theta \in K(z, \theta | \exp(\theta(z)) = z)$. What happens if we try to interpret $\theta$ as a function $\theta(z) : \mathbb{C} \to \mathbb{C}$? We chose $\theta(1) = 0$, and, since $\exp(i \epsilon) \approx i \epsilon$, $\theta(i \epsilon) \approx i \epsilon$. As we track $\theta(z)$ around the unit circle $\{ z = x + iy : x^2 + y^2 = 1 \}$, we see that $\theta(i) = \frac{i \pi}{2}$, $\theta(-1) = i \pi$, and ultimately $\theta(1) = 2 \pi i$.

8.2.1 Some Unpleasant Facts

The blunt facts are the following (and many analogues).

**Proposition 71 (No square root function)** There is no continuous function $f : \mathbb{C} \to \mathbb{C}$ (or $\mathbb{C} \setminus \{0\} \to \mathbb{C} \setminus \{0\}$) with the property that $\forall z : f(z)^2 = z$.

**Proposition 72 (No logarithm function)** There is no continuous function $f : \mathbb{C} \setminus \{0\} \to \mathbb{C} \setminus \{0\}$ with the property that $\forall z : \exp(f(z)) = z$ (note that $f(\exp(z)) = z$ is clearly impossible as $\exp$ is many:one).

**Observation 18** This statement, about actual functions $\mathbb{C} \to \mathbb{C}$, might seem to be in contradiction with the statement labelled “(1)+(2)” on page 230. In fact, it is not so much a contradiction as a statement that the world of functions $\mathbb{C} \to \mathbb{C}$ is not as neat as the algebraic world of the previous chapter.

In particular, the common statements

\[
\log z_1 + \log z_2 = \log z_1 z_2. \tag{8.9}
\]

\[
\log 1/z = - \log z. \tag{8.10}
\]

are false for (any interpretation of) the function $\log : \mathbb{C} \setminus \{0\} \to \mathbb{C} \setminus \{0\}$, even though they are true for the usual $\log : \mathbb{R}^+ \to \mathbb{R}$. On page 229 we proved a

---

3 “No continuous argument”, from which the others follow, is proved in [Pri03, §9.2].

4 Consider $z_1 = z_2 = -1$ and $z = -1$ for counter-examples.
result which we said was “normally expressed as” (8.9). What we in fact proved was that \( \phi := \log \eta_1 + \log \eta_2 - \log(\eta_1 \eta_2) \) was a constant, in the sense that it differentiated to 0: in fact it is

\[
\phi = \begin{cases} 
2\pi i & \arg \eta_1 + \arg \eta_2 > \pi \\
0 & -\pi < \arg \eta_1 + \arg \eta_2 \leq \pi \\
-2\pi i & \arg \eta_1 + \arg \eta_2 \leq -\pi 
\end{cases}
\]  

(8.11)

Similarly the discrepancy in (8.10), i.e. \( \psi := \log(1/z) + \log z \), is a differential constant whose value is

\[
\psi = \begin{cases} 
2\pi i & z = x + iy : x < 0 \land y = 0 \\
0 & \text{otherwise} 
\end{cases}
\]  

(8.12)

### 8.2.2 The Problem with Square Roots

These difficulties arise even with the square root function, and without needing to consider differential algebra.

**Example 28 ([BCD\textsuperscript{+}02])** Consider two apparently similar statements:

\[
\sqrt{1-z} \sqrt{1+z} \overset{?}{=} \sqrt{1-z^2} \]  

(8.13)

\[
\sqrt{z-1} \sqrt{z+1} \overset{?}{=} \sqrt{z^2-1} 
\]  

(8.14)

Naïvely, we might consider both to be true: square both sides and they reduce to \((1-z)(1+z) = 1-z^2 \) and \( (z-1)(z+1) = z^2-1 \), both of which are true. But in fact, while (8.13) is true ([BCD\textsuperscript{+}02 Lemma 2]), (8.14) is false, as evaluation at \( z = -2 \) gives \( \sqrt{-3} \sqrt{-1} \overset{?}{=} \sqrt{3} \), and in fact the left-hand side is \(-\sqrt{3} \), rather than \( \sqrt{3} \).

### 8.2.3 Possible Solutions

These facts have, of course, been known since the beginnings of complex analysis, and there are various approaches to the problem, discussed in [Dav10].

**Multivalued Functions** (taking values not on \( C \), but in the set of subsets of \( C \), i.e. \( P(C) \)). Here we think of the logarithm of 1 as being, not 0, but any of \( \{..., -4\pi i, -2\pi i, 0, 2\pi i, 4\pi i, ...\} \). Our exemplar problem (8.9) is then treated as follows.

The equation merely states that the sum of one of the (infinitely many) logarithms of \( z_1 \) and one of the (infinitely many) logarithms of \( z_2 \) can be found among the (infinitely many) logarithms of \( z_1 z_2 \), and conversely every logarithm of \( z_1 z_2 \) can be represented as a sum of this kind (with a suitable choice of log \( z_1 \) and log \( z_2 \)).

[Car58] pp. 259–260 (our notation)
8.2. BRANCH CUTS

Since log is to be the inverse of exp, we essentially transpose the graph and have \((\text{graph(exp)}^T, C, P(C))_B\) as the Bourbaki formulation of logarithm.

**Notation 34** We will use function names beginning with capitals, e.g. \(\text{Log}\) or \(\text{Sqrt}\) (so \(\text{Sqrt}(4) = \{-2, 2\}\)) for such multivalued functions.

(8.13) and (8.14), then both become true, when rewritten as

\[
\text{Sqrt}(1 - z) \text{Sqrt}(1 + z) = \text{Sqrt}(1 - z^2)
\]

(8.13)

\[
\text{Sqrt}(z - 1) \text{Sqrt}(z + 1) = \text{Sqrt}(z^2 - 1).
\]

(8.14)

Note, however, that what appears to be a special case of (8.9), viz.

\[
2 \log z = \log z^2,
\]

is not true in this setting, since when \(z = 1\), \(\text{Log} z^2 = \text{Log} 1 = \{2k\pi i\}\), while \(2 \text{Log} z = 2 \text{Log} 1 = \{4k\pi i\}\). The problem is, in fact, that \(\text{Log} z + \text{Log} z \neq 2 \text{Log} z\).

**Riemann Surfaces** Instead of saying that \(\text{Log} 1\) has multiple values, we say that there are multiple versions of ‘1’, each of which has a well-defined logarithm. In terms of Example 26 the ‘1’ that we reach after going round the circle once is different from the ‘1’ we started from. In terms of Notation 3, we have \((\text{graph(exp)}^T, \mathcal{R}_{\log z}, C)_B\) where \(\mathcal{R}_{\log z}\) signifies the Riemann surface corresponding to the function \(\log z\), shown in Figure 8.1. The Riemann surface view is discussed in [BCD’02, Section 2.4], which concludes

Riemann surfaces are a beautiful conceptual scheme, but at the moment they are not computational schemes.

Note that the standard counterexample to (8.10), viz. \(z = -1\), so that \(\log(1/-1) = \pi i \neq -\log(-1)\) is now solved by the fact that \(1/-1\) is a \(-1\) on a different branch, and in (8.9) \(z_1 + z_2\) is on the appropriate branch to make (8.9) valid.

* Both the previous solutions have preserved continuity, and the identities, at the cost of not having a function \(C \rightarrow C\).

**Branches** Of course, we could also consider subsets of \(C\) which do not contain the whole of the troublesome circle of Example 26. If we do that, then we can have a continuous version of, say, \(\log z\) as in

\(\log z\) has a branch in any simply connected open set which does not contain 0. [Car73, p. 61]
So any given branch would be \((G, D, I)_B\), where \(D\) is a simply connected open set which does not contain 0, \(G\) is a graph obtained from one element of the graph (i.e. a pair \((z, \log(z))\) for some \(z \in D\)) by analytic continuation, and \(I\) is the relevant image set. While this approach preserves continuity, it has three drawbacks.

1. We now have as many definitions of \(\log\), say \(\log_D\), as we have choices of \(D\), and these do not necessarily agree, even if we insist that \(1 \in D\) and \(\log_D 1 = 0\). For example, if \(D_1 = \mathbb{C} \setminus \{iy : y \in [0, \infty)\}\) and \(D_2 = \mathbb{C} \setminus \{-iy : y \in [0, \infty)\}\), then \(\log_{D_1}(-1) = -i\pi\) (since \(\log(1-\epsilon i) \approx -\epsilon i\) and we get from 1 to \(-1\) within \(D_1\) by going clockwise round the unit circle from 1 via \(-i\), whose logarithm is \(-i\pi/2\)) but \(\log_{D_2}(-1) = i\pi\), since now we go anticlockwise round the unit circle, from 1 to \(-1\) via \(i\).

2. \(\log_D\) is not defined outside \(D\).

3. Identities such as (8.9) are not valid, even if \(z_1, z_2\) and \(z_1z_2\) are all in \(D\): consider \(D_1\) as above, with \(z_1 = z_2 = -1\). This is inevitable: no assignment of a nonzero value to \(\log(-1)\) can satisfy (8.10), for example.

This solution preserves continuity within \(D\), at the cost of losing identities, and the uncertainty caused by the multiple options for \(D\).
8.2. BRANCH CUTS

Branch Cuts Here we associate with each potentially multi-valued operator $f$ the following.

- An “initial value” $(z_0, f(z_0))$.
- The “branch cut”, which is a curve, or set of curves, $B$ in $\mathbb{C}$ joining the singularities of $f$, such that $\mathbb{C} \setminus B$ is simply-connected. $f(z)$ for any $z \notin B$ is then obtained by analytic continuation from $(z_0, f(z_0))$ by analytic continuation along any path not meeting $B$.
- A rule for computing $f(z)$ on $B$, which is generally arranged to make $f(z)$ continuous with one side or the other of $B$.

A typical example would be $\log$, where it is usual these days to have:

- Initial value $(1, 0)$;
- Branch cut $\{x + 0i : x \in (-\infty, 0]\}$;
- Rule for $\log x : x < 0$: $\log(x + 0i) = \lim_{\epsilon \to 0^+} \log(x + \epsilon i) = (\pi + \log |x|)$.

We should note that there are many possible choices: the author was in fact initially taught

- Initial value $(1, 0)$;
- Branch cut $\{x + 0i : x \in [0, \infty)\}$;
- Rule for $\log x : x > 0$: $\log(x + 0i) = \lim_{\epsilon \to 0^+} \log(x + \epsilon i)$.

Despite the apparent arbitrariness, the world has tended to converge on the branch cuts as defined in [AS64, Nat10].

8.2.4 Removable Branch Cuts

This term is introduced in [DF94, §4.2], presumably by analogy with “removable singularity”. The example they give is illustrative.

Example 29 ([DF94, §4.2], our phrasing) Consider $f(z) = g(z) - h(z)$, where $g(z) = \log(z + 1)$ and $h(z) = \log(z - 1)$. Then $g$ has a branch cut along $(-\infty, -1]$ and $h$ has one along $(-\infty, 1]$, and hence we would expect a branch cut for $f$ along $(-\infty, 1]$. In fact, the contributions from $g$ and $h$ exactly cancel along $(-\infty, -1)$, and the actual branch cut is only $[-1, 1]$. We should note that this is only true of $g - h$, and $(1 + \epsilon)g - h$ genuinely has a branch cut along $(-\infty, 1]$ for all $\epsilon \neq 0$.

This is, in fact, analogous to the “removable singularity” behaviour we saw at (8.1): $\frac{x^2 - 1 - \epsilon}{x - 1}$ has a genuine singularity at $x = 1$ for all $\epsilon \neq 0$, but when $\epsilon = 0$ the singularity is in fact removable.

The careful reader should note that one should use printing 9 or later of [AS64]; the branch cut for $\text{arccot}$ moved!


8.3 Fundamental Theorem of Calculus Revisited

In the previous chapter we reduced the Fundamental Theorem of Calculus to the status of Notation 29, saying that integration is the inverse of differentiation. From the algebraic point of view, that is correct. From the analytic point of view, where the following definitions hold, there is indeed something to prove.

Definition 97 (Analysis) Given a function \( f : \mathbb{R} \to \mathbb{R} \), we define, where the right-hand sides exist,

\[
F(x) = \int_a^x f(x) \, dx = \lim_{h \to 0} \frac{f(x+h)-f(x)}{h} = \lim_{|\Delta| \to 0} S^\Delta(f) = \lim_{|\Delta| \to 0} S_{\Delta}(f)
\]

(8.16)

provided both limits exist and are equal

where \( \Delta = [x_0 = a < x_1 < \cdots < x_n = x] \) is a dissection of \([a, x]\), \(|\Delta| = \max(x_i - x_{i-1})\), and \( S^\Delta(f) = \sum_{i=1}^{n}(x_i - x_{i-1}) \max_{y \in [x_{i-1}, x_i]} f(y) \) and \( S_{\Delta}(f) = \sum_{i=1}^{n}(x_i - x_{i-1}) \min_{y \in [x_{i-1}, x_i]} f(y) \) and the upper and lower approximations of “the area under the curve” of \( f \).

Theorem 46 (Fundamental Theorem of Calculus [Apo67, §5.3]) Let \( f \) and \( F \) be functions defined on a closed interval \([a, b]\) such that \( F' = f \). Then, if \( f \) is Riemann-integrable on \([a, b]\), then

\[
\int_a^b f(x) \, dx = F(b) - F(a).
\]

Though this is the classical statement, we must emphasise that \( F' = f \) must hold throughout \([a, b]\), and therefore \( F \) is differentiable, hence continuous, throughout this interval.

8.4 Constants Revisited

In the previous chapter we defined (Definition 90) a constant as being an element whose derivative was zero. How well does this compare with our usual intuition, which is of a constant function, i.e. one whose value is independent of the argument?

It is a truism of calculus that a constant function has to have derivative zero, and a theorem of analysis that a function whose derivative is defined and zero everywhere is indeed constant, but that italicised phrase is important. The classic example is the Heaviside function:

\[
H(x) = \begin{cases} 
0 & x \leq 0 \\
1 & x > 0 
\end{cases}
\]

which is clearly not constant, but whose derivative is zero except at \( x = 0 \), where it is undefined.
8.4.1 Constants can be useful

We can make constructive use of the concept of a differential constant, however. Mathematica, for example, writes $\sqrt{x^2}$ in the form $x \text{sign}(x)$, and then treats $\text{sign}(x)$ as a differential constant internally, replacing it by $\frac{1}{x} \sqrt{x^2}$ at the end.

8.4.2 Constants are often troubling

One sometimes sees\(^6\) (but stated as an equality, rather than with our $\equiv$
\begin{equation}
\arctan(x) + \arctan(y) \equiv \arctan\left(\frac{x + y}{1 - xy}\right). \tag{8.17}
\end{equation}

If we let
\begin{equation}
C = \arctan(x) + \arctan(y) - \arctan\left(\frac{x + y}{1 - xy}\right), \tag{8.18}
\end{equation}

then
\begin{equation}
\frac{\partial C}{\partial x} = \frac{1}{1 + x^2} - \frac{(1 - xy) + y(x + y)}{(1 + xy)^2} = \frac{1}{1 + x^2} - \frac{(1 - xy) + y(x + y)}{(1 - xy)^2 + (x + y)^2} = 0, \tag{8.19}
\end{equation}

and similarly $\frac{\partial C}{\partial y} = 0$, so $C$ would seem to be a constant. Differentially, it is, but in fact
\begin{equation}
C(x, y) = \begin{cases} 
\pi & xy > 1; x < 0 \\
0 & < 1 \\
\pi & xy > 1; x > 0
\end{cases} \tag{8.20}
\end{equation}

see Figure 8.2.

8.5 Integrating ‘real’ Functions

In the previous chapter we saw
\begin{equation}
\int \frac{1}{x^2 + 1} \, dx = \frac{i}{2} (\ln (1 - i x) - \ln (1 + i x)), \tag{7.28bis}
\end{equation}

and said that the reader might complain “I asked to integrate a real function, but the answer is coming back in terms of complex numbers”.

A calculus text would normally write
\begin{equation}
\int \frac{1}{x^2 + 1} \, dx = \arctan(x), \tag{7.28er}
\end{equation}

\(^6\)For an example, see [http://www.mathamazement.com/Lessons/Pre-Calculus/05_Analytic-Trigonometry/sum-and-difference-formulas.html](http://www.mathamazement.com/Lessons/Pre-Calculus/05_Analytic-Trigonometry/sum-and-difference-formulas.html)
where arctan is the inverse of the tan function, defined by

\[
\tan(x) = \frac{\sin(x)}{\cos(x)} = \frac{1}{i} \frac{e^{ix} - e^{-ix}}{e^{ix} + e^{-ix}} = \frac{1}{i} \frac{\theta - \theta^{-1}}{\theta + \theta^{-1}} = \frac{1}{i} \frac{\theta^{2} - 1}{\theta^{2} + 1},
\]

(8.21)

where \( \theta' = i\theta \) is an exponential in the sense of Definition 91.

If we write \( \phi = \frac{\theta^{2} - 1}{\theta^{2} + 1} \), we can deduce that \( \phi' = \phi^{2} + 1 \), and treat this as a *definition* of “arctangents” in the same way that Definition 91 is a definition of “exponentials”, by adding a clause

(d) \( \theta \) (assumed to be nonzero) is a *tangent* over \( K \), i.e. there is an \( \eta \) in \( K \) such that \( \theta' = \eta' (\theta^{2} + 1) \): \( \theta \) is a tan of \( \eta \).

Similarly, we could add

(e) \( \theta \) is an *inverse tangent* over \( K \), i.e. there is an \( \eta \) in \( K \) such that \( \theta' = \eta'/(\eta^{2} + 1) \): \( \theta \) is an arctan of \( \eta \).

We could then deduce an equivalent of statement “(1)+(2)” on page 230, that tan are (differential) inverses of each other.

**Lemma 15** ((8.17) restated) *If \( \theta \) is an arctan of \( \alpha \), \( \phi \) is an arctan of \( \beta \) and \( \psi \) is an arctan of \( \frac{\alpha + \beta}{1 - \alpha \beta} \), then \( \theta + \phi - \psi \) is a constant, in the sense of Definition 90.*

---

7It is common to write tan\(^{-1}(x)\) rather than arctan\((x)\), but the author finds this open to confusion with the other abuse of notation that writes tan\(^{2}(x)\) for \((\tan(x))^{2}\). What would tan\(^{-2}(x)\) mean?
8.6. LOGARITHMS REVISITED

The proof is equivalent to (8.19):

\[
(\theta + \phi - \psi)' = \frac{\alpha'}{1 + \alpha^2} + \frac{\beta'}{1 + \beta^2} - \frac{(\frac{\alpha + \beta}{1 - \alpha \beta})'}{1 + \left(\frac{\alpha + \beta}{1 - \alpha \beta}\right)^2}
\]

\[= 0.
\]

However, the fact that this “constant” is given by (8.20) can trip us up: consider

\[
\int_1^2 \frac{1}{x^2 + 1} + \frac{2}{x^2 + 4} \, dx = \arctan\left(\frac{3x}{2 - x^2}\right).
\]

This is differentially valid, but apparently violates numerical evaluation

\[
\int_1^2 \frac{1}{x^2 + 1} + \frac{2}{x^2 + 4} \, dx \approx \arctan\left(\frac{3x}{2 - x^2}\right)
\]

\[= -2 \arctan(3) \approx -2.498,
\]

and we have an integral of a positive function apparently being negative. In fact the correct answer is \(\pi - 2 \arctan(3) \approx 0.644\).

One way of explaining this problem to someone not familiar with the area is to point out that an integral should always be continuous, whereas \(\arctan\left(\frac{3x}{2 - x^2}\right)\) is not continuous in the interval \([1, 2]\), since there’s a discontinuity at \(x = \sqrt{2}\) (see Figure 8.3)

\[
\lim_{x \to \sqrt{2}^-} \arctan\left(\frac{3x}{2 - x^2}\right) = \arctan(\infty) = \frac{\pi}{2}
\]

but

\[
\lim_{x \to \sqrt{2}^+} \arctan\left(\frac{3x}{2 - x^2}\right) = \arctan(-\infty) = -\frac{\pi}{2},
\]

and it is this discontinuity of \(\pi\) that needs to be accounted for.

8.6 Logarithms revisited

8.7 Other decision questions

We saw the theory of polynomial quantifier elimination in section 3.5.3. There, given a formula

\[
Q_1x_1, \ldots, Q_kx_k \phi(x_1, \ldots, x_n)
\]

(where each \(Q_i\) is either \(\exists\) or \(\forall\), and \(\phi\) is a Boolean combination of equalities and inequalities between polynomials), this produces an equivalent formula \(\psi(x_{k+1}, \ldots, x_n)\). In particular, if \(k = n\), we get either ‘true’ or ‘false’. Expressions like \(\sqrt{x^2 - 1}\) can be handled by writing them as \(y\) and adding \(\exists y : y^2 = x^2 - 1 \land \) in the appropriate place, for example. How can we add transcendental functions? This, of course, assumes that we know about the
Figure 8.3: Graph of apparent integral in (8.22)

plot(arctan(3/(2-x^2)), x = 1 .. 2)
numerical values of the functions as well as their differential properties. It is usual to assume Schanuel’s conjectures [Ax71], which roughly speaking state that there are no unexpected identities in exponentials and logarithms of complex numbers.

**Conjecture 2 (Schanuel [Ax71])** Given any \( n \) complex numbers \( z_1, \ldots, z_n \) which are linearly independent over the rational numbers \( \mathbb{Q} \), the extension field \( \mathbb{Q}(z_1, \ldots, z_n, \exp(z_1), \ldots, \exp(z_n)) \) has transcendence degree of at least \( n \) over \( \mathbb{Q} \).

[AW00] take a different approach.

**Definition 98** A real or complex valued function \( f \) defined in some open domain of \( \mathbb{R} \) or \( \mathbb{C} \), respectively, is called strongly transcendental (with exceptional point \( \xi \)) if for all numbers \( x \) in the domain of \( f \) excluding \( \xi \) not both \( x \) and \( f(x) \) are algebraic.

It follows from Lindemann’s Theorem that \( \exp \) is transcendental with exceptional point 0, and \( \log \) is transcendental with exceptional point 1. Similarly \( \sin, \cos \) and \( \tan \) and their inverses, with exceptional points 0 except for \( \arccos \), which has 1.

[MW12] first consider problems of the form

\[ Q_1 x_1 \Phi(x_1, \text{trans}(x_1)), \]

where \( \text{trans} \) is a strongly transcendental function with

\[ \text{trans}'(x) = \frac{a(x) + b(x) \text{trans}(x)}{d(x)} : a, b, d \in \mathbb{Z}[x]. \]

log, \( \exp \) and \( \arctan \) all satisfy these requirements. Here they produce an unconditional algorithm for reducing this to ‘true’ or ‘false’, where ‘unconditional’ means not relying on Conjecture 2 or equivalent hard problems in number theory, and it is this unconditionality that is the most surprising part of the work. This will, for example, decide

\[ \forall x_1 (\exp(x_1)(1 - x_1) \leq 1 \lor x_1 \geq 1), \]

and in fact does so in less than 1 second [Ach06].

They then take problems of the form

\[ Q_1 x_1 Q_2 x_2 \ldots, Q_n x_n \phi(x_1, \text{trans}(x_1), x_2, \ldots, x_n), \]

apply the Collins-style quantifier elimination to \( Q_2 x_2 \ldots, Q_n x_n \phi(x_1, y, x_2, \ldots, x_n) \) to get \( \Phi(x_1, y) \), and apply their previous method to \( Q_1 x_1 \Phi(x, \text{trans}(x_1)) \). Again, the process is unconditional. They also show how various other problems, e.g.

\[ \forall x_1 x_1 > 7 \Rightarrow \cosh(x_1) > x_1^3 - 4x_1, \]

can be transformed into this form, and decided (17 seconds).

While this is a significant step forward, we should note that we are limited to one occurrence of \( \text{trans} \), and this has to correspond to the first quantifier. Also, this is purely a decision procedure, and will not do more general quantifier elimination.
8.8 Limits

While we can, and have, algebraised differentiation and integration, the process of computing limits is more fundamentally analytic. The first serious exploration of limits was in Macsyma [Wan71b, Wan71a]. This process essentially added four “constants” to Macsyma’s language:

- `infinity` which is the infinity of the (one-point compactification of the) complex plane;
- `inf` which is the positive one of the two-point compactification of the reals (“plus infinity”);
- `minf` which is the negative one of the two-point compactification of the reals (“minus infinity”)
- `ind` which is “indeterminate”.

Computer algebra systems are in general schizophrenic about whether they are computing over the reals or the complexes, but we should note the confusing fact that the reals are a subset of the complexes, but the usual (two-point, \(+\infty\) and \(-\infty\)) compactification of the reals is not a subset of the compactification of the complexes.

8.8.1 A Definite Integral

With the conventional choice of branch cuts, we can write the fractional part of \(x\) as

\[
\text{fra}(x) = x - |x| = \frac{1}{2} + \frac{i}{2\pi} \log(-\exp(-2\pi ix)).
\]

(8.24)

If we ask Maple to simplify the above, nothing happens, but with the symbolic option (i.e. ignoring the branch cuts of log)\(^8\) we just get \(x\). We note that \(\text{fra}'(x) = 1\) since \(\text{fra}(x)\) and \(x\) “differ by a constant”, that (differential) constant being \(|x|\).

If we ask for

\[
\int_{1/2}^{1} \text{fra}(1/x) \, dx = \int_{1/2}^{1} \left( \frac{1}{x} - 1 \right) \, dx = \ln 2 - \frac{1}{2} \approx 0.193,
\]

(8.25)

many systems\(^9\) will get this right. The fun comes when we try, say, \(\int_{1/3}^{1} \text{fra}(1/x) \, dx\). Maple returns \(\ln 3 - \frac{1}{4} = 0.765\), which is absurd for integrating a function whose value is between 0 and 1 on an interval of length 2/3. It gets

\[
\int_{1/3}^{1/2} \text{fra}(1/x) \, dx = \int_{1/3}^{1/2} \left( \frac{1}{x} - 2 \right) \, dx = \ln 3 - \ln 2 - \frac{2}{6} \approx 0.072
\]

(8.26)

\(^8\)Sage’s full `simplify` apparently does the same.
\(^9\)Sage apparently returns `log2` — [https://groups.google.com/forum/#!topic/sage-support/0UkERaidET0](https://groups.google.com/forum/#!topic/sage-support/0UkERaidET0)
correct, but has failed to spot the singularity in the middle of the integrand in
\[
\int_{1/3}^{1} \text{frac}(1/x) \, dx = \int_{1/3}^{1} \left( \frac{1}{x} - \left\{ \begin{array}{ll} 2 & x < 1/2 \\ 1 & x \geq 1/2 \end{array} \right. \right) \, dx = \ln 3 - \frac{1}{2} - \frac{2}{6} \approx 0.265. \quad (8.27)
\]

If we split the integral by hand, as in \( \int_{1/3}^{1/2} \text{frac}(1/x) \, dx + \int_{1/2}^{1} \text{frac}(1/x) \, dx \), we get the right answer, as we do when we use Maple’s preferred notation, \texttt{frac}(1/x)\) rather than \[8.24\].
Appendix A

Algebraic Background

A.1 The resultant

It quite often happens that we have to consider whether two polynomials, which are usually relatively prime, can have a common factor in certain special cases. The basic algebraic tool for solving this problem is called the resultant. In this section we shall define this object and we shall give some properties of it. We take the case of two polynomials \( f \) and \( g \) in one variable \( x \) and with coefficients in a ring \( R \).

We write \( f = \sum_{i=0}^{n} a_i x^i \) and \( g = \sum_{i=0}^{m} b_i x^i \).

**Definition 99** The **Sylvester matrix** of \( f \) and \( g \) is the matrix

\[
\text{Syl}(f, g) = \begin{pmatrix}
a_n & a_{n-1} & \ldots & a_1 & a_0 & 0 & 0 & \ldots & 0 \\
0 & a_n & a_{n-1} & \ldots & a_1 & a_0 & 0 & \ldots & 0 \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\
0 & \ldots & 0 & a_n & a_{n-1} & \ldots & a_1 & a_0 & 0 \\
0 & \ldots & 0 & 0 & a_n & a_{n-1} & \ldots & a_1 & a_0 \\
b_m & b_{m-1} & \ldots & b_1 & b_0 & 0 & 0 & \ldots & 0 \\
0 & b_m & b_{m-1} & \ldots & b_1 & b_0 & 0 & \ldots & 0 \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\
0 & \ldots & 0 & b_m & b_{m-1} & \ldots & b_1 & b_0 & 0 \\
0 & \ldots & 0 & 0 & b_m & b_{m-1} & \ldots & b_1 & b_0
\end{pmatrix}
\]

where there are \( m \) lines constructed with the \( a_i \), \( n \) lines constructed with the \( b_i \).

**Definition 100** The resultant of \( f \) and \( g \), written \( \text{Res}(f, g) \), or \( \text{Res}_x(f, g) \) if there is doubt about the variable, is the determinant of this matrix. The \( j \)-th principal subresultant coefficient, \( \text{psc}_j(f, g) \) is the determinant of the matrix obtained by deleting the last \( j \) rows of \( f \) coefficients, the last \( j \) rows of \( g \) coefficients and the last \( 2j \) columns.
Well-known properties of determinants imply that the resultant belongs to $R$, and that $\text{Res}(f, g)$ and $\text{Res}(g, f)$ are equal, to within a sign. We must note that, although the resultant is defined by a determinant, this is not the best way of calculating it. Because of the special structure of the Sylvester matrix, we can consider Euclid’s algorithm as Gaussian elimination in this matrix (hence the connection between the resultant and the g.c.d.). One can also consider the sub-resultant method as an application of the Sylvester identity (theorem 15) to this elimination. It is not very difficult to adapt advanced methods (such as the method of sub-resultants described in section 2.3.2, or the modular and $p$-adic methods described in chapters 4 and 5) to the calculation of the resultant. Collins [1971] and Loos [1982] discuss this problem. We now give a version of Euclid’s algorithm for calculating the resultant. We denote by $\text{lc}(p)$ the leading coefficient of the polynomial $p(x)$, by $\text{degree}(p)$ its degree, and by $\text{remainder}(p, q)$ the remainder from the division of $p(x)$ by $q(x)$. We give the algorithm in a recursive form.

Algorithm 45 (resultant)
Input: $f, g$;
Output: $r = \text{Res}(f, g)$.

\[
n := \text{degree}(f);
m := \text{degree}(g);
\]

if $n > m$ then $r := (-1)^{nm}\text{resultant}(g, f)$
else $a_n := \text{lc}(f)$;

if $n = 0$ then $r := a_m^n$
else $h := \text{remainder}(g, f)$;

if $h = 0$ then $r := 0$
else $p := \text{degree}(h)$;

$r := a_m^{n-p}\text{resultant}(f, h)$;

return $r$;

It is worth noting that this returns a partially factored form of the resultant, if the underlying arithmetic supports this (e.g. the factored representation of section 2.1.3).

We write $h = \sum_{i=0}^{p} c_i x^i$ and $c_i = 0$ for $i > p$. This algorithm does indeed give the resultant of $f$ and $g$ for, when $n \leq m$ and $n \neq 0$, the polynomials $x^i g - x^i h$ (for $0 \leq i < n$) are linear combinations of the $x^j f$ (for $0 \leq j < m$), and therefore we are not changing the determinant of the Sylvester matrix of $f$ and $g$ by replacing $b_i$ by $c_i$ for $0 \leq i < m$. Now this new matrix has the form

\[
\begin{pmatrix}
A & \ast \\
0 & B
\end{pmatrix}
\]

where $A$ is a triangular matrix with determinant $a_n^{m-p}$ and $B$ is the Sylvester matrix of $f$ and $h$. From this algorithm we immediately get

**Proposition 73** $\text{Res}(f, g) = 0$ if and only if $f$ and $g$ have a factor in common.

\footnote{The modular method is described in section 4.5.1.}
It is now easy to prove the following propositions:

**Proposition 74** If the $\alpha_i$ are the roots of $f$, then

$$\text{Res}(f, g) = a_n^m \prod_{i=1}^n g(\alpha_i).$$

**Proposition 75** If the $\beta_i$ are the roots of $g$, then

$$\text{Res}(f, g) = (-1)^{mn} b_m^m \prod_{i=1}^m f(\beta_i).$$

**Proposition 76** $\text{Res}(f, g) = a_n^m b_m^m \prod_{i=1}^n \prod_{j=1}^m (\alpha_i - \beta_j).$

**Proof** [Duva87]. We write the right hand sides of the three propositions as

\[
\begin{align*}
R_2(f, g) &= a_n^m \prod_{i=1}^n g(\alpha_i), \\
R_3(f, g) &= (-1)^{mn} b_m^m \prod_{i=1}^m f(\beta_i), \\
R_4(f, g) &= a_n^m b_m^m \prod_{i=1}^n \prod_{j=1}^m (\alpha_i - \beta_j).
\end{align*}
\]

It is clear that $R_2(f, g)$ and $R_3(f, g)$ are equal to $R_4(f, g)$. The three propositions are proved simultaneously, by induction on the integer $\min(n, m)$. If $f$ and $g$ are swapped, their resultant is multiplied by $(-1)^{nm}$, and gives $R_4(f, g) = (-1)^{mn} R_4(g, f)$, while $R_2(f, g) = (-1)^{nm} R_3(g, f)$. We can therefore suppose that $n \leq m$. Moreover $R_2(f, g)$ is equal to $a_n^n$ when $n = 0$, as is the resultant of $f$ and $g$, and $R_4(f, g)$ is zero when $n > 0$ and $h = 0$, as is the resultant. It only remains to consider the case when $m \geq n > 0$ and $h \neq 0$. But then $R_2(f, g) = a_n^{n-p} R_2(f, h)$ for $g(\alpha_i) = h(\alpha_i)$ for each root $\alpha_i$ of $f$, and the algorithm shows that $\text{Res}(f, g) = a_n^{n-p} \text{Res}(f, h)$, from which we get the desired result.

**Corollary 20** $\text{Res}(fg, h) = \text{Res}(f, h) \text{Res}(g, h)$.

**Definition 101** The discriminant of $f$, $\text{Disc}(f)$ or $\text{Disc}_x(f)$, is

$$a_n^{2n-2} \prod_{i=1}^n \prod_{j=1, j \neq i}^n (\alpha_i - \alpha_j).$$

**Proposition 77** $\text{Disc}(f) = 0$ if, and only if, $f$ has a repeated root, i.e. is not square-free.
Proposition 78  \( \text{Res}(f, f') = a_n \text{Disc}(f) \). Moreover \( \text{Disc}(f) \in R \).

Corollary 21  \( \text{Disc}(fg) = \text{Disc}(f) \text{Disc}(g) \text{Res}(f, g)^2 \).

Whichever way they are calculated, the resultants are often quite large. For example, if the \( a_i \) and \( b_i \) are integers, bounded by \( A \) and \( B \) respectively, the resultant is less than \((n + 1)^\frac{m}{2}(m + 1)^\frac{n}{2}A^m B^n\), but it is very often of this order of magnitude (see section A.2). Similarly, if the \( a_i \) and \( b_i \) are polynomials of degree \( \alpha \) and \( \beta \) respectively, the degree of the resultant is bounded by \( m\alpha + n\beta \). A case in which this swell often matters is the use of resultants to calculate primitive elements, which uses the following result.

Proposition 79  If \( \alpha \) is a root of \( p(x) = 0 \), and \( \beta \) is a root of \( q(x, \alpha) = 0 \), then \( \beta \) is a root of \( \text{Res}_y(y - p(x), q(x, y)) \).

A.2 Useful Estimates

Estimates of the sizes of various things crop up throughout computer algebra. They can be essentially of three kinds.

- **Upper bounds:** \( X \leq B \).
- **Lower bounds:** \( X \geq B \).
- **Average sizes:** \( X \) “is typically” \( B \). This may be a definitive average result (but even here we have to be careful what we are averaging over: the average number of factors of a polynomial is one, for example, but this does not mean we can ignore factorisation), or some heuristic “typically we find”.

Estimates are used throughout complexity theory, of course. “Worst case” complexity is driven by upper bound results, while “average case” complexity is driven by average results. They are also used in algorithms: most algorithms of the ‘Chinese Remainder’ variety (section 4) rely on upper bounds to tell when they have used enough primes/evaluation points. In this case, a better upper bound generally translates into a faster algorithm in practice.

A.2.1 Matrices

How big is the determinant \( |M| \) of an \( n \times n \) matrix \( M \)?

**Notation 35** If \( \mathbf{v} \) is a vector, then \( |\mathbf{v}| \) denotes the Euclidean norm of \( \mathbf{v} \), \( \sqrt{\sum |v_i|^2} \). If \( f \) is a polynomial, \( |f| \) denotes the Euclidean norm of its vector of coefficients.

**Proposition 80** If \( M \) is an \( n \times n \) matrix with entries \( \leq B \), \( |M| \leq n!B^n \).

This bound is frequently used in algorithm analysis, but we can do better.
A.2. USEFUL ESTIMATES

Proposition 81 [Hadamard bound \(H_r\)] If \(M\) is an \(n \times n\) matrix whose rows are the vectors \(v_i\), then \(|M| \leq H_r = \prod |v_i|\), which in turn is \(\leq n^{n/2}B^n\).

Corollary 22 If \(f\) and \(g\) are polynomials of degrees \(n\) and \(m\) respectively, then \(\text{Res}(f, g) \leq (m + n)^{(m+n)/2}|f|^m|g|^n\).

Corollary 23 If \(f\) is a polynomial of degree \(n\), then \(\text{Disc}(f) \leq n^{n-1}|f|^{2n-1}\).

In practice, especially if \(f\) is not monic, it is worth taking rather more care over this estimation.

Proposition 82 (Hadamard bound (columns)) If \(M\) is an \(n \times n\) matrix whose columns are the vectors \(v_i\), then \(|M| \leq H_c = \prod |v_i| \leq n^{n/2}B^n\).

In practice, for general matrices one computes \(\min(H_r, H_c)\). While there are clearly bad cases (e.g. matrices of determinant 0), the Hadamard bounds are “not too bad”. As pointed out in [AM01], \(\log(\min(H_r, H_c)/|M|)\) is a measure of the “wasted effort” in a modular algorithm for computing the determinant, and “on average” this is \(O(n)\), with a variance of \(O(\log n)\). It is worth noting that this is independent of the size of the entries.

Corollary 24 If \(x\) is the solution to \(M.x = a\), where \(M\) and \(a\) have integer entries bounded by \(B\) and \(A\) respectively, then the denominators of \(x\) are bounded by \(\min(H_r, H_c)\) and the numerators of \(x\) are bounded by \(n^{n/2}AB^{n-1}\).

Proof. This follows from the linear algebra result that \(x = \frac{1}{|M|}\text{adj}(M).a\), where \(\text{adj}(M)\), the adjoint of \(M\), is the matrix whose \((i, j)\)th entry is the determinant of the matrix obtained by striking row \(i\) and column \(j\) from \(M\). The \(i\)th entry of \(\text{adj}(M).a\) is then the determinant of the matrix obtained by replacing the \(i\)th column of \(M\) by \(a\), and we can apply Proposition 82 to this matrix.

A.2.2 Coefficients of a polynomial

Here we are working implicitly with polynomials with complex coefficients, though the bounds will be most useful in the case of integer coefficients.

Notation 36 Let

\[f(x) = \sum_{i=0}^{n} a_i x^i = a_n \prod_{i=1}^{n} (x - \alpha_i)\]

be a polynomial of degree \(n\) (i.e. \(a_n \neq 0\)). Define the following measures of the size of the polynomial \(f\):

- \(H(f)\) (often written \(||f||_\infty\)), the height or 1-norm, is \(\max_{i=0}^{n} |a_i|\);
- \(||f||\) (often written \(||f||_2\)), the 2-norm, is \(\sqrt{\sum_{i=0}^{n} |a_i|^2}\);
- \(L(f)\) (often written \(||f||_1\)), the length or 1-norm, is \(\sum_{i=0}^{n} |a_i|\);
APPENDIX A. ALGEBRAIC BACKGROUND

$M(f)$, the Mahler measure of $f$, is $|a_n| \prod_{|\alpha_i| > 1} |\alpha_i|.$

**Proposition 83**

$$H(f) \leq ||f|| \leq L(f) \leq (n + 1)H(f),$$

where the first inequality is strict unless $f$ is a monomial, the second is strict unless all the $a_i$ are equal or zero, and the third is strict unless all the $a_i$ are equal.

We should note that the last inequality could be replaced by $\leq cH(f)$, where $c$ is the number of nonzero monomials in $f$, but this seems not to be much exploited.

**Proposition 84 (Landau’s Inequality [Lan05, Mig89, §4.3]).**

$$M(f) \leq ||f||$$

and the inequality is strict unless $f$ is a monomial.

**Corollary 25** $|a_{n-i}| \leq \binom{n}{i} M(f) \leq \binom{n}{i} ||f||$.

The first part of this follows from the fact that $a_{n-i}$ is $\pm a_n$ times a sum of $\binom{n}{i}$ products of roots, and products of roots are bounded by Proposition 84. For some applications, e.g. Theorem 33, we often bound $\binom{n}{i}$ by $2^n$, but for others, such as Proposition 54, the more precise value is needed. $2^n$ might seem like overkill, but in fact, both in general and in the application to factoring [Mig81], 2 cannot be replaced by any smaller number.

It should be noted that the Landau–Mignotte bound is not the only way to bound the coefficients of a polynomial: [Abb09] gives four methods that depend on knowing the degree of the factor being searched for, and two others that will bound the least height of a factor. Depressingly, he gives a family of examples that shows that no bound is superior to any other, and indeed [Abb09, 3.3.5] it may be necessary to “mix-and-match” using different bounds for different coefficients.

These results can be generalised to polynomials in several variables [Mig89, Chapter 4.4]. The definition of $M(f)$ is more complicated.

**Definition 102** The Mahler measure of $f \in \mathbb{C}[x_1, \ldots, x_n]$ is defined inductively on $n$, using Notation 36 for $n = 1$ and more generally

$$\log M(f) = \int_0^1 \log (M(f(e^{2\pi it}, x_2, \ldots, x_n))) dt.$$  

**Proposition 85 ([Mig89, Proposition 4.8]).** Let

$$f(x_1, \ldots, x_n) = \sum_{i_1=0}^{d_1} \sum_{i_2=0}^{d_2} \cdots \sum_{i_n=0}^{d_n} a_{i_1, \ldots, i_n} x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n}.$$
A.2. USEFUL ESTIMATES

Then

\[ |a_{i_1, \ldots, i_n}| \leq \left( \frac{d_{i_1}}{i_1} \right) \left( \frac{d_{i_2}}{i_2} \right) \cdots \left( \frac{d_{i_n}}{i_n} \right) M(f). \]

Proposition 84 is still valid.

Corollary 26 Hence

\[ |a_{i_1, \ldots, i_n}| \leq \left( \frac{d_{i_1}}{i_1} \right) \left( \frac{d_{i_2}}{i_2} \right) \cdots \left( \frac{d_{i_n}}{i_n} \right) ||f||. \]

A.2.3 Roots of a polynomial

Several questions can be asked about the roots of a univariate polynomial. The most obvious ones are how large/small can they be, but one is often interested in how close they can be. These questions are often asked of the real roots (section 3.5.5), but we actually need to study all roots, real and complex. The distinction between real and complex roots is pretty fine, as shown in the following example.

Example 30 (Wilkinson Polynomial [Wil59]) Let \( W_{20} \) have roots at \(-1, -2, \ldots, -20, \) so that \( W_{20} = (x+1)(x+2)\cdots(x+20) = x^{20} + 210x^{19} + \cdots + 20! \). Consider now the polynomial \( W_{20}(x) + 2^{-23}x^{19} \). One might expect this to have twenty real roots close to the original ones, but in fact it has ten real roots, at approximately \(-1, -2, \ldots, -7, -8.007, -8.917 \) and \(-20.847, -16.731 \pm 2.813i \) and \(-19.502 \pm 1.940i \).

The discriminant of \( W_{20} \) is \( 2.74 \times 10^{275} \), which would seem well away from zero. However, the largest coefficient of \( W_{20} \) is \( 1.38 \times 10^{19} \), and of \( W'_{20} \) is \(-3.86 \times 10^{19} \). The norms of \( W_{20} \) and \( W'_{20} \) are \( 2.27 \times 10^{19} \) and \( 6.11 \times 10^{19} \), so corollary 23 gives a bound of \( 4.43 \times 10^{779} \) for \( \text{Disc}(W_{20}) \), and a direct application of corollary 22 gives \( 3.31 \times 10^{763} \). Hence the discriminant of \( W_{20} \) is “much smaller than it ought to be”, and \( W_{20} \) is “nearly not square-free”. Put another way, the Sylvester matrix for the discriminant is very ill-conditioned (this was in fact Wilkinson’s original motivation for constructing \( W_{20} \)): the discrepancy between the actual determinant and corollary 22 is 489 decimal digits, whereas [AM01] would lead us to expect about 17.

Notation 37 Let \( f \in C[x] = \sum_{i=0}^n a_i x^i \), and let the roots of \( f \) be \( \alpha_1, \ldots, \alpha_n \), and define

\[ \text{rb}(f) = \max_{1 \leq i \leq n} |\alpha_i|, \]

\[ \text{sep}(f) = \min_{1 \leq i < j \leq n} |\alpha_i - \alpha_j|. \]

\( \text{sep}(f) \) is zero if, and only if, \( f \) has a repeated factor.

Proposition 86 [Cau29, p. 122] \( \text{rb}(f) \leq 1 + \max(|a_i|)/|a_n|. \)
Corollary 27 If the polynomial $f$ does not take the value 0 at $x = 0$, then every root of $f$ has absolute value at least $|a_0|/(|a_0| + \max(|a_i|))$.

Proposition 87 [Cau29, p. 123]

$$\text{rb}(f) \leq \max \left( \frac{n|a_{n-1}}{a_n}, \sqrt{\frac{n|a_{n-2}}{a_n}}, \ldots, \frac{n|a_1}{a_n}, \frac{n|a_0|}{a_n} \right).$$

Proposition 88 [Knu98, 4.6.2 exercise 20]

$$\text{rb}(f) \leq B = 2 \max \left( \frac{|a_{n-1}}{a_n}, \sqrt{\frac{|a_{n-2}}{a_n}}, \ldots, \frac{|a_1|}{a_n}, \frac{|a_0|}{a_n} \right).$$

Furthermore, there is at least one root of absolute value $\geq B/(2n)$.

Applied to $W_{20}$, these propositions give respectively $1.38 \times 10^{15}, 4200$ and 420. If we centre the roots, to be $-9.5, \ldots, 9.5$, the three propositions give respectively $2.17 \times 10^{12}, 409.5$ and 49.5. While the advantages of centrality are most prominent in the case of proposition 86, they are present for all of them. There are in fact improved bounds available in this ($a_{n-1} = 0$) case [Mig00].

If instead we re-balance $W_{20}$ so that the leading and trailing coefficients are both 1, by replacing $x$ by $\frac{x}{\sqrt{20!}}$, then the bounds for this new polynomial are 6961419, 505.76 and 50.58 (and hence the roots of $W_{20}$ are bounded by 838284.7, 4200 and 420).

Proposition 89 ([Gra37]) If $p(x) = p_e(x^2) + x p_o(x^2)$, i.e. $p_e(x^2)$ is the even terms of $p(x)$, then the roots of $q(x) = p_e^2(x) - x p_o^2(x)$ are the squares of the roots of $p$.\footnote{For the history of the attribution to Graeffe, see [Hou59].}

Applying this process to $W_{20}$, then computing the three bounds, and square-rooting the results, gives us bounds of $3.07 \times 10^{18}, 239.58$ and 75.76. Repeating the process gives $2.48 \times 10^{18}, 61.66$ and 34.67. On the centred polynomial, we get $2.73 \times 10^{12}, 117.05$ and 37.01, and a second application gives $1.83 \times 10^{18}, 30.57$ and 17.19 (and hence root bounds for $W_{20}$ as $1.83 \times 10^{18}, 40.07$ and 26.69). This last figure is reasonably close to the true value of 20 [DM90].

A.2.4 Root separation

The key result is the following.

Proposition 90 ([Mah64]) $\text{sep}(f) > \sqrt{3|\text{Disc}(f)| n^{-(n+2)/2} |f|^{1-n}}$. 
We note that the bound is zero if, and only if, the discriminant is zero, as it should be, and this bound is unchanged if we multiply the polynomial by a constant. The bound for $W_{20}$ is $7.27 \times 10^{-245}$, but for the centred variant it becomes $1.38 \times 10^{-113}$. Should we ever need a root separation bound in practice, centring the polynomial first is almost always a good idea. Similarly re-balancing changes the separation bound to $5.42 \times 10^{-112}$, which means $6.52 \times 10^{-113}$ for the original polynomial.

These bounds seem very far away from reality (i.e. 1 for $W_n$), but in fact are almost optimal.

**Proposition 91 ([ESY06, Theorem 3.6])** Let $a \geq 3$ be an $L$-bit integer, and $n \geq 4$ be an even integer. Then $Q(x) = (x^n - 2(ax - 1)^2)(x^n - (ax - 1)^2)$ has degree $2n$ and coefficients of size $O(L)$, but any sub-division method requires $\Omega(nL)$ subdivisions to isolate the roots, of which there are three in $(a^{-1} - h, a^{-1} + h)$ with $h = a^{-n/2} - 1$.

### A.2.5 Developments

The monic polynomials of degree 7 with maximum root separation and all roots in the unit circle are:

- **six complex** $x^7 - 1$, with discriminant $-823543$, norm $\sqrt{2}$ and root bounds $[2.0, 1.383087554, 2.0]$

  and root separation bound (proposition 90) $\sqrt[3]{\frac{27}{56}} \approx 3.09 \times 10^{-2}$ (true value 0.868);

- **four complex** $x^7 - x$, with discriminant $6^6 = 46656$, norm $\sqrt{2}$ and root bounds $[2.0, 1.383087554, 2.0]$

  and root separation bound (proposition 90) $\frac{27\sqrt{2}}{18607} \approx 7.361786385 \times 10^{-3}$ (true value 1);

- **two complex** $x^7 - \frac{1}{4} x^5 - x^3 + \frac{1}{4} x$, with discriminant $-\frac{50635}{4096} \approx -12.36$, norm $\frac{1}{4} \sqrt{34} \approx 1.45737974$ and root bounds $[2, 1.626576562, 2.0]$

  and root separation bound (proposition 90) $\frac{1880\sqrt{2}}{82572791} \approx 9.99 \times 10^{-5}$ (true value 1/2);

- **all real** $x^7 - \frac{12}{9} x^5 + \frac{40}{81} x^3 - \frac{4}{81} x$, with discriminant $3$

  $10485760000 \approx 5.66 \times 10^{-6}$, \hspace{1cm} \text{(A.1)}$

\[ ^3 \text{We note how the constraint that all the roots be real forces the discriminant to be small.} \]
norm $\frac{17}{81} \sqrt{86} \approx 1.946314993$ and root bounds
$$[\frac{23}{9}, 3.299831645, 2.494438258]$$

and root separation bound (proposition 90)
$$\frac{83980800}{32254409474403781} \approx 1.19 \times 10^{-8}$$ (true value 1/3).

If there are $n + 1$ equally-spaced real roots (call the spacing 1 for the time being), then the first root contributes $n!$ to $\prod_{i} \prod_{j \neq i} (\alpha_i - \alpha_j)$, the second root $1! \cdot \frac{n}{(n-1)!}$ and so on, so we have a total product of
$$\prod_{i=0}^{n} i! = \prod_{i=0}^{n} i^2 = \left( \prod_{i=0}^{n} i! \right)^2.$$  \hspace{1cm} (A.2)

This is sequence A055209 [Slo07], which is the square of A000178.

If we now assume that the roots are equally-spaced in $[-1, 1]$, then the spacing is $\frac{2}{n}$, we need to correct equation (A.2) by dividing by $\left( \frac{n}{2} \right)^n (n-1)!$: call the result $C_n$. $C$ is initially greater than one, with $C_3 = \frac{12288}{16807} \approx 0.73$, the optimum. The norms are respectively $\sqrt{\frac{305853}{625}} \approx 1.77$ and $\sqrt{\frac{47321}{125}} \approx 2.07$.

Equally spaced roots, at $\pm \frac{1}{2}$ and 0, give a discriminant of $\frac{51298814505517056}{37252902984619140625} \approx 0.00138$. Unconstrained solving for the maximum of the discriminant, using Maple’s Groebner, Solve, starts becoming expensive, but if we assume symmetry, we are led to choose roots at $\pm \sqrt{\frac{1471+42\sqrt{7}}{21}}$, with a discriminant of $\frac{47108864}{1690796065} \approx 0.0041$. The norms are respectively $\sqrt{\frac{105857}{625}} \approx 1.77$ and $\sqrt{\frac{207}{21}} \approx 2.07$. 

degree 7 Equally spaced roots, at $\pm \frac{2}{3}$, $\pm \frac{1}{3}$ and 0, give a discriminant of $\frac{209715200000}{5615789612636313} \approx 5.66 \cdot 10^{-6}$. Again assuming symmetry, we are led to choose roots at $\pm \sqrt{\frac{17\sqrt{86} + 55}{81}}$, and 0, which gives $\frac{209715200000}{5615789612636313} \approx 3.73 \cdot 10^{-5}$. The norms are respectively $\sqrt{\frac{117+35\sqrt{86}}{81}} \approx 1.95$ and $\sqrt{\frac{117-35\sqrt{86}}{81}} \approx 2.41$.

degree 8 Equally spaced roots, at $\pm \frac{5}{3}$, $\pm \frac{3}{3}$ and $\pm \frac{1}{3}$, give a discriminant of $\approx 5.37 \cdot 10^{-9}$. Assuming symmetry, we get roots at $\pm 0.87$, $\pm 0.59$.
and $\pm \approx 0.21$, with a discriminant of $\approx 9.65 \cdot 10^{-8}$. The norms are respectively $\approx 2.15$ and $\sqrt{2 \sqrt{727171}} \approx 2.81$.

**degree 9** Equally spaced roots, at $\pm \frac{3}{4}, \pm \frac{1}{2}, \pm \frac{1}{4}$ and 0, give a discriminant of $1.15 \cdot 10^{-12}$. Assuming symmetry, we get roots at $\pm 0.8998, \pm 0.677, \pm 0.363$ and zero, with a discriminant of $\approx 7.03 \cdot 10^{-11}$. The norms are respectively $\approx 2.15$ and $\sqrt{2 \sqrt{727171}} \approx 2.81$.

If we now consider the case with two complex roots, which may as well be at $x = \pm i$, we have the following behaviour.

**degree 4** The maximal polynomial is $x^4 - 1$, with discriminant $-256$ and norm $\sqrt{2}$. The bound is $\frac{\sqrt{2}}{216} \approx 0.153$.

**degree 5** The maximal polynomial is $x^5 - x$, with discriminant $-256$ and norm $\sqrt{2}$. The bound is $\frac{\sqrt{2}}{625} \approx 0.0248$.

**degree 6** Equally spaced roots, at $\pm \frac{1}{3}$, gives a discriminant of $-108.26$ and a norm of $2 \sqrt{31}$. The bound is $\frac{400 \sqrt{172}}{1800497} \approx 2.38 \cdot 10^{-3}$. The maximal discriminant is attained with roots at $\pm \frac{1}{3}$, with discriminant $-4194304 \approx -213.09$ and norm $2 \sqrt{31}$. The bound is $\frac{4 \sqrt{31}}{3375} \approx 2.65 \cdot 10^{-3}$.

**degree 7** Equally spaced roots, at $\pm \frac{1}{2}$ and 0, gives a discriminant of $\approx -12.36$ and norm of $\sqrt{17} \approx 1.46$. The bound is $\frac{\sqrt{172}}{82572791} \approx 9.99 \cdot 10^{-5}$. The maximal discriminant is attained with roots at $\pm \sqrt{\frac{17}{4}}$, with discriminant $\approx 40.8$ and norm of $\sqrt{31} \approx 1.596$. The bound is $\approx 1.06 \cdot 10^{-4}$.

### A.3 Chinese Remainder Theorem

In this section we review the result of the title, which is key to the methods in section 4.2.4, and hence to much of computer algebra.

**Theorem 47 (Chinese Remainder Theorem (coprime form))** Two congruences

\[
X \equiv a \pmod{M} \tag{A.3}
\]

and

\[
X \equiv b \pmod{N}, \tag{A.4}
\]

where $M$ and $N$ are relatively prime, are precisely equivalent to one congruence

\[
X \equiv c \pmod{MN}. \tag{A.5}
\]

By this we mean that, given any $a$, $b$, $M$ and $N$, we can find such a $c$ that satisfaction of (A.3) and (A.4) is precisely equivalent to satisfying (A.5). The converse direction, finding (A.3) and (A.4) given (A.5), is trivial: one takes $a$ to be $c$ (mod $M$) and $b$ to be $d$ (mod $N$).
Algorithm 46 (Chinese Remainder)

Input: $a, b, M$ and $N$ (with $\gcd(M,N) = 1$).

Output: $c$ satisfying Theorem 47.

Compute $\lambda, \mu$ such that $\lambda M + \mu N = 1$;

$\#$The process is analogous to Lemma 1 (page 55)

$c := a + \lambda M (b - a)$;

Clearly $c \equiv a \pmod{M}$, so satisfying (A.5) means that (A.3) is satisfied. What about (A.4)?

$$c = a + \lambda M (b - a) = a + (1 - \mu N) (b - a) \equiv a + (b - a) \pmod{N}$$

so, despite the apparent asymmetry of the construction, $c \equiv b \pmod{N}$ as well.

In fact, we need not restrict ourselves to $X$ being an integer: $X$, $a$ and $b$ may as well be polynomials (but $M$ and $N$ are still integers).

Algorithm 47 (Chinese Remainder (Polynomial form))

Input: Polynomials $a = \sum_{i=0}^{n} a_i x^i$, $b = \sum_{i=0}^{n} b_i x^i$, and integers $M$ and $N$ (with $\gcd(M,N) = 1$).

Output: A polynomial $\sum_{i=0}^{n} c_i x^i$ satisfying Theorem 47.

Compute $\lambda, \mu$ such that $\lambda M + \mu N = 1$;

$\#$The process is analogous to Lemma 1 (page 55)

for $i := 0$ to $n$ do

$c_i := a_i + \lambda M (b_i - a_i)$;

In some applications, e.g. Section 4.1, we will want to apply Algorithm 46 repeatedly to many primes $p_i$ known in advance, to deduce a value $c$ modulo $N := \prod p_i$ from many values $a_i \pmod{p_i}$. While the obvious way to do this is “one prime at a time”, applying Algorithm 46 to $p_1$ and $p_2$, then to $p_1 p_2$ and $p_3$, and so on, the correct way to do this is “balanced combination”, first combining the $p_i$ in pairs, then the pairs to form quadruples and so on.

Proposition 92

In this case, the dominant cost is that of the last step, and with classical arithmetic this is $O((\log N)^2)$.

A.4 Chinese Remainder Theorem for Polynomials

The theory of the previous section has an obvious generalisation if we replace $\mathbb{Z}$ by $K[y]$ for a field $K$, and an equivalent application to sections 4.3–4.4.
A.4. CHINESE REMAINDER THEOREM FOR POLYNOMIALS

Theorem 48 (Chinese Remainder Theorem for polynomials) Two congruences

\[ X \equiv a \pmod{M} \quad (A.6) \]

and

\[ X \equiv b \pmod{N}, \quad (A.7) \]

where \( M \) and \( N \) are relatively prime polynomials in \( K[y] \), are precisely equivalent to one congruence

\[ X \equiv c \pmod{MN}. \quad (A.8) \]

By this we mean that, given any \( a, b, M \) and \( N \), we can find such a \( c \) that satisfaction of (A.6) and (A.7) is precisely equivalent to satisfying (A.8). The converse direction, finding (A.6) and (A.7) given (A.8), is trivial: one takes \( a \) to be \( c \pmod{M} \) and \( b \) to be \( d \pmod{N} \).

Algorithm 48 (Chinese Remainder for Polynomials)
Input: \( a, b, M \) and \( N \in K[y] \) (with \( \gcd(M,N) = 1 \)).
Output: \( c \) satisfying Theorem 48

Compute \( \lambda, \mu \) such that \( \lambda M + \mu N = 1 \);
#As in Lemma 1 (page 55). Note \( \mu \) isn’t needed in practice.
\( c := a + \lambda M(b - a) \);

Clearly \( c \equiv a \pmod{M} \), so satisfying (A.8) means that (A.6) is satisfied. What about (A.7)?

\[
\begin{align*}
    c &= a + \lambda M(b - a) \\
    &= a + (1 - \mu N)(b - a) \\
    &\equiv a + (b - a) \pmod{N}
\end{align*}
\]

so, despite the apparent asymmetry of the construction, \( c \equiv b \pmod{N} \) as well.

As in proposition 92, the correct way to combine values modulo many (say \( d \)) evaluations is “balanced combination”, first combining the \( x - v_i \) in pairs, then the pairs to form quadruples and so on.

Proposition 93 In this case, the dominant cost is that of the last step, and with classical arithmetic this is \( O(d^2) \).

As in Algorithm 47, \( X, a \) and \( b \) may as well be polynomials in \( x \) whose coefficients are polynomials in \( y \) (but \( M \) and \( N \) are still in \( y \) only).

Algorithm 49 (Chinese Remainder (Multivariate))
Input: Polynomials \( a = \sum_{i=0}^{n} a_i x^i, b = \sum_{i=0}^{n} b_i x^i \in K[y][x], \) and \( M \) and \( N \in K[y] \) (with \( \gcd(M,N) = 1 \)).
Output: A polynomial \( = \sum_{i=0}^{n} c_i x^i \) satisfying Theorem 48
Compute \( \lambda, \mu \) such that \( \lambda M + \mu N = 1 \);
# As in Lemma \[\text{III} \] (page \[\text{55} \]). Note \( \mu \) isn’t needed in practice.

for \( i := 0 \) to \( n \)
\( c_i := a_i + \lambda M(b_i - a_i) \);

It is even possible for \( x \) (and \( i \)) to represent numerous indeterminates, as we are basically just doing coefficient-by-coefficient reconstruction.

**Observation 19** We have explicitly considered \( K[y] \) (e.g. \( Q[y] \)), but in practice we will often wish to consider \( Z[y] \). Even if all the initial values are in \( Z[x] \), \( \lambda \) and \( \mu \) may not be, as in \( M = (y - 1) \) and \( N = (y + 1) \), when \( \lambda = \frac{-1}{2} \) and \( \mu = \frac{1}{2} \). This may not be an obstacle to such reconstruction from values (often called interpolation, by analogy with interpolation over \( R \)). Interpolation over \( Z[y] \) may still be possible: consider reconstructing \( X \) with \( X \equiv 1 \pmod{M} \) and \( X \equiv -1 \pmod{N} \), which is \( 1 + \frac{-1}{2} M(-1 - 1) = y \). But interpolation over \( Z[y] \) is not *always* possible: consider reconstructing \( X \) with \( X \equiv 1 \pmod{M} \) and \( X \equiv 0 \pmod{N} \), which gives \( \frac{1}{2}(y - 1) \).

### A.5 Vandermonde Systems

**Definition 103** The Vandermonde matrix\(^4\) generated by \( k_1, \ldots, k_n \) is

\[
V(k_1, \ldots, k_n) = \begin{pmatrix}
1 & k_1 & k_1^2 & \ldots & k_1^{n-1} \\
1 & k_2 & k_2^2 & \ldots & k_2^{n-1} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & k_n & k_n^2 & \ldots & k_n^{n-1}
\end{pmatrix}.
\]

**Notation 38** In the context of \( V(k_1, \ldots, k_n) \), let \( P(z) = \prod_{i=1}^{n} (z-k_i) \) and \( P_j(z) = \prod_{i \neq j} (z-k_i) \).

**Proposition 94** The inverse of a Vandermonde matrix has a particularly simple form: \( V(k_1, \ldots, k_n)^{-1} = (m_{i,j}) \) with

\[
m_{i,j} = \frac{\text{coeff}(P_j(z), z^i)}{\prod_{k \neq j} (k_j - k_k)} = \frac{\text{coeff}(P_j(z), z^i)}{P_j(k_j)}.
\]

For example

\[
V(k_1, \ldots, k_3)^{-1} = \begin{pmatrix}
k_1 k_3 & -k_1 k_3 & k_1 k_3 \\
(-k_2 + k_1)(-k_3 + k_3) & (-k_2 + k_1)(k_2 - k_3) & (k_2 - k_3)(-k_3 + k_3) \\
(-k_2 + k_3)(-k_3 + k_3) & (-k_2 + k_1)(k_2 - k_3) & (k_2 - k_3)(-k_3 + k_3) \\
(1) & (1) & (1)
\end{pmatrix}
\]

\(^4\)This section is based on [Zip93 §13.1]. Our algorithm [50] is his SolveVanderMonde, and our algorithms [51] is the one at the top of his p. 214.
Corollary 28  If all the $k_i$ are distinct, then $V(k_1,\ldots,k_n)$ is invertible.

Equation (A.9) and the fact that the $P_j$ can be rapidly computed from $p$ suggest a rapid way of computing the elements of the inverse of a $n \times n$ Vandermonde matrix in $O(n^2)$ time. In fact, we can solve a system of Vandermonde linear equations in $O(n^2)$ time and $O(n)$ space.

Algorithm 50 (Vandermonde solver)
Input: Vandermonde matrix $V(k_1,\ldots,k_n)$, right-hand side $w$.
Output: Solution $x$ to $V(k_1,\ldots,k_n)x = w$

\[
x := 0 \quad P := \prod_{i=1}^n (z - k_i) \quad \#O(n^2)
\]

for $i := 1$ to $n$
\[
Q := P/(z - k_i) \quad \#nO(n)
\]
\[
D := P(k_i) \quad \#nO(n) \text{ by Horner’s rule}
\]

for $j := 1$ to $n$
\[
x_j := x_j + w_i \text{coeff}(Q,z^{-1})
\]

In section 4.4.2 we will want to solve a slightly different system. Algorithm 50 solves a system of the form

\[
x_1 + k_1 x_2 + k_1^2 x_3 + \cdots + k_1^{n-1} x_n = w_1 \\
x_1 + k_2 x_2 + k_2^2 x_3 + \cdots + k_2^{n-1} x_n = w_1 \\
\vdots \\
x_1 + k_n x_2 + k_n^2 x_3 + \cdots + k_n^{n-1} x_n = w_1
\]

(A.10)

whereas we need to solve a system of the form

\[
k_1 x_1 + k_2 x_2 + \cdots + k_n x_n = w_1 \\
k_1^2 x_1 + k_2^2 x_2 + \cdots + k_n^2 x_n = w_2 \\
\vdots \\
k_1^n x_1 + k_2^n x_2 + \cdots + k_n^n x_n = w_n
\]

(A.11)

By comparison with (A.10), we have transposed the matrix (which is not a problem, since the inverse of the transpose is the transpose of the inverse), and multiplied column $i$ by an extra factor of $k_i$. From Corollary 28, we can deduce the criteria for this system to be soluble.

Corollary 29  If all the $k_i$ are distinct and non-zero, then the system (A.11) is soluble.

The following variant of Algorithm 50 will solve the system.

Algorithm 51 (Vandermonde variant solver)
Input: Vandermonde style data $(k_1,\ldots,k_n)$, right-hand side $w$.
Output: Solution $x$ to (A.11)
\[ x := 0 \]
\[ P := \prod_{i=1}^{n} (z - k_i) \quad \#O(n^2) \]
for \( i := 1 \to n \)
\[ Q := P/(z - k_i) \quad \#nO(n) \]
\[ D := k_i P(k_i) \quad \#nO(n) \text{ by Horner’s rule} \]
for \( j := 1 \to n \)
\[ x_i := x_i + w_j \frac{\text{coeff}(Q, z^{j-1})}{D} \]
A.6 Algebraic Structures

In this section we gather together many of the definitions we have seen throughout the book in one table. The definitive references, as far as computer algebra (rather than abstract algebra) is concerned are [DT90, DGT91], which explain why we say “g.c.d. domain” rather than “unique factorisation domain”: the argument is summarised in note 28 (page 53).

Table A.1: Algebraic Structures

<table>
<thead>
<tr>
<th>Name</th>
<th>Reference</th>
<th>Example(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ring (+, −, 0, ∗)</td>
<td>Definition 6</td>
<td>2 × 2 matrices</td>
</tr>
<tr>
<td>Commutative ring</td>
<td></td>
<td>{0, 2, 4, 6, 8, 10} (mod 12)</td>
</tr>
<tr>
<td>Commutative ring with 1</td>
<td></td>
<td>{0, 1, 2, ..., 11} (mod 12) = ( \mathbb{Z}_{12} )</td>
</tr>
<tr>
<td>Integral Domain</td>
<td>Definition 9</td>
<td>( { a + b\sqrt{-5} } ) (Example 2)</td>
</tr>
<tr>
<td>g.c.d. domain</td>
<td>Definition 27</td>
<td>( \mathbb{Z}[x, y] )</td>
</tr>
<tr>
<td>principal ideal domain</td>
<td>Definition 12</td>
<td>( \mathbb{Z}; \mathbb{Q}[x] )</td>
</tr>
<tr>
<td>field</td>
<td>Definition 13</td>
<td>( \mathbb{Q}; \mathbb{Z}_p ) (see Proposition 50)</td>
</tr>
<tr>
<td>Algebraically closed field</td>
<td>Definition 16</td>
<td>( \mathbb{C} )</td>
</tr>
</tbody>
</table>

Note that any integral domain can be extended into its field of fractions (Definition 14), and any field can be extended into its algebraic closure (Definition 17).
Appendix B

Excursus

This appendix includes various topics on computer algebra that do not seem to be well-treated in the literature.

B.1 The Budan–Fourier Theorem

Definition 104 Given a sequence $A$ of non-zero numbers $a_0, \ldots, a_n$, the number of sign variations of $A$, written $V(A)$, is the number of times two consecutive elements have different signs, i.e. $a_i a_{i+1} < 0$. If $A$ does contain zeros, we erase them before doing this computation, or equivalent we count the number of $(i, j)$ with $a_i a_j < 0$ and all intermediate $a_{i+1}, \ldots, a_{j-1} = 0$. If $f$ is the polynomial $a_n x^n + \cdots + a_0$, we write $V(f)$ rather than $V(A)$ where $A$ is the sequence of coefficients of $f$.

Proposition 95 $V(f) = V(f(0), f'(0), \ldots, f^{(n)}(0))$ for a polynomial $f$ of degree $n$.

The reader should note that the definition of variation is numerically unstable. $V(1) = 0$, and therefore (by the erasure rule) $V(1, 0) = 0$. For positive $\epsilon$, $V(1, \epsilon) = 0$, but $V(1, -\epsilon) = 1$. This is related to the fact that $x + \epsilon$ has no positive real roots, but $x - \epsilon$ has one, as seen in the following result.

Theorem 49 (Descartes’ rule of signs [CA76]) (the number of roots of $f$ in $(0, \infty)$ is less than or equal to, by an even number, $V(f)$.

Corollary 30 The number of roots of $f$ in $(a, \infty)$ is less than or equal to, by an even number, $V(f(x - a))$.

Corollary 31 The number of roots of $f$ in $(a, \infty)$ is less than or equal to, by an even number, $V((f(a), f'(a), \ldots, f^{(n)}(a))$.

For dense $f$, there is not a great deal to choose between these two formulations, but, since the derivatives of a sparse polynomial are sparse but its translate is not, corollary 31 is greatly to be preferred to corollary 30 in the sparse case.
We can also deduce some results about the number of roots of sparse polynomials. If \( f \) has \( n \) non-zero terms, \( V(f) \leq n - 1 \). We note that \( V(ax^k) = 0 \), and this polynomial indeed has no roots in \((0, \infty)\).

**Corollary 32** A polynomial in \( x \) with \( n \) terms, not divisible by \( x \), has at most \( 2(n - 1) \) roots in \( \mathbb{R} \). If it is divisible by \( x \), then the answer is at most \( 2n - 1 \).

The example of \( x^3 - x \), which has three real roots \((\pm 1, 0)\) shows that the special case is necessary.

For the sake of simplicity, we will consider only square-free polynomials in the rest of this section: the results generalise fairly easily to non square-free ones.

Let us fix \( f \), and consider \( V(y) := V(f(x - y)) \) and \( N(y) := |\{ x > y : f(x) = 0 \}| \) as functions of \( y \). For large enough \( y \), both are zero. As \( y \) decreases, \( N(y) \) increases monotonically, by 1 at each root of \( f \). In fact, the same monotonic behaviour is true of \( V(y) \), increasing by 1 at roots of \( f \) and by 2 at certain other points. This allows us to compute the number of roots in an interval, a result known as the Budan–Fourier Theorem.

**Corollary 33 (Budin–Fourier Theorem)** The number of roots of \( f \) in \((a, b)\) is less than or equal to, by an even number, \( V(f(x - a)) - V(f(x - b)) \).

**Corollary 34 (Budin–Fourier Theorem [Hur12])** The number of roots of \( f \) in \((a, b)\) is less than or equal to, by an even number, \( V((f(a), f'(a), \ldots, f^n(a)) - V((f(b), f'(b), \ldots, f^n(b)) \).

For the same reasons as above, corollary 34 is to be preferred in the case of sparse polynomials.

An almost complete generalisation of Corollary 32 in the sense that the bounds depend on the number of monomials rather than the degrees, to sparse polynomials in \( n \) variables is given in [BHNS15, §2].

### B.2 Equality of factored polynomials

This section treats the following problem.

**Problem 7** Given two polynomials in factored form (section 2.1.3), are they equal? More precisely, if

\[
 f = \prod_{i=1}^{n} f_i^{a_i}, \quad g = \prod_{j=1}^{m} g_j^{b_j},
\]

with the \( f_i \) and \( g_j \) square-free and relatively prime, i.e.: \( \gcd(f_i, f_i') = 1 \), \( \gcd(g_j, g_j') = 1 \), is \( f \equiv g \).

---

1See [BdB22, Fou31]. The question of precedence was hotly disputed at the time: see [Akr82] and [http://www-history.mcs.st-andrews.ac.uk/Biographies/Budan_de_Boislaurent.html](http://www-history.mcs.st-andrews.ac.uk/Biographies/Budan_de_Boislaurent.html).
The obvious solution is to expand $f$ and $g$, and check that the expanded forms (which are canonical) are equal. Can we do better?

One important preliminary remark is that the square-free representation of a polynomial is unique. This leads to the following result.

**Proposition 96** If $f = g$, then every $a_i$ has to be a $b_j$, and vice versa. For each such occurring value $k$, we must verify

$$f_k = \prod_{i=1}^{n} a_i^{\frac{f_i}{k}} = \prod_{j=1}^{m} g_j^{\frac{g_j}{k}},$$

In particular, $f_k$ and $g_k$ must have the same degree, i.e.

$$\sum_{i=1}^{n} \deg(f_i) = \sum_{j=1}^{m} \deg(g_j).$$

Again, we could check $f_k = g_k$ by expansion, but there is a better way.

**Example 31** Let $f = x^{2l} - 1$ and $g = (x-1)(x+1)(x^2+1) \cdots (x^{2l-1}+1)$, where both are square-free, so proposition 96 does not help. $f$ is already expanded, but expansion of $g$ can give rise to $x^{2l-1} + x^{2l-2} + \cdots + x + 1$, which has length $O(2^l)$, whereas $g$ has length $O(l)$.

From now on we will assume that we are working over a domain that includes the integers.

**Lemma 16** If $f$ and $g$ have degree at most $N$, and agree at $N + 1$ different values, then $f = g$.

**Proof.** $f - g$ is a polynomial of degree at most $N$, but has $N + 1$ zeros, which contradicts proposition [3] if it is non-zero.

Hence it suffices to evaluate $f$ and $g$ at $N + 1$ points $x_i$ and check that the values agree. It is not necessary to construct any polynomial, and the integers involved are bounded (if we choose $|x_i| < N$) by $BN^N$, where $B$ is a function of the coefficients of the $f_i$, $g_j$. Furthermore, we can evaluate at all these points in parallel. However, it does seem that we need to evaluate at $N + 1$ points, or very nearly so, even if $f$ and $g$ are very small.

**Open Problem 28 (Crossings of factored polynomials)** Produce some non-trivial bounds on the maximum number of zeros of $f - g$, where $f$ and $g$ have small factored representations. See [RR90].

The best we can say is as follows. Suppose, in the notation of (B.1), each $f_i$ has $k_i$ non-zero terms, and $g_j$ has $l_j$ non-zero terms, and no $f_i$ or $g_j$ is $x$ (if either was, then trivially $f \neq g$, since a common factor of $x$ would have been detected). Then, by Corollary 32, $f - g$, if it is not identically zero, has at most $2 \left( \sum_{i=1}^{n} k_i + \sum_{j=1}^{m} l_j - 1 \right)$ roots in $\mathbb{R}$, and hence, if it is zero when evaluated at more than this number of integers, is identically zero. The factor of 2 can be dropped if we use only positive evaluation points, and rely on Theorem 40.
B.3 Karatsuba’s method

This method was originally introduced in [KO63] for multiplying large integers; however, it is easier to explain in the (dense) polynomial context, where issues of carrying do not arise. Consider the product of two linear polynomials

\[(ax + b)(cx + d) = acx^2 + (ad + bc)x + bd.\]  

This method so patently requires four multiplications of the coefficients that the question of its optimality was never posed. However, [KO63] rewrote it as follows:

\[(ax + b)(cx + d) = acx^2 + [(a + b)(c + d) - ac - bd]x + bd,\]  

which only requires three distinct multiplications, \(ac\) and \(bd\) each being used twice. However, it requires four coefficients additions rather than one, so one might question the practical utility of it. For future reference, we will also express equation (B.4) as

\[(ax + b)(cx + d) = ac(x^2 - X) + (a + b)(c + d)x + bd(1 - X),\]  

which makes the three coefficient multiplications explicit.

However, it can be cascaded. Consider a product of two polynomials of degree three (four terms):

\[(a_3y^3 + a_2y^2 + a_1y + a_0)(b_3y^3 + b_2y^2 + b_1y + b_0).\]  

If we write \(X = y^2\), \(a = a_3 + a_2\) etc., this product looks like the left-hand side of equation (B.4), and so can be computed with three multiplications of linear polynomials, each of which can be done in three multiplications of coefficients, thus making nine such multiplications in all, rather than the classic method’s 16.

If the multiplicands have \(2^k\) terms, then this method requires \(3^k = (2^k)^{\log_2 3}\) multiplications rather than the classic \(2^k\) coefficient multiplications. For arbitrary numbers of terms, not necessarily a power of two, the cost of “rounding up” to a power of two is subsumed in the \(O\) notation, and we see a cost of \(O(n^{\log_2 3})\) rather than the classic \(O(n^2)\) coefficient multiplications. We note that \(\log_2 3 \approx 1.585\), and the number of extra coefficient additions required is also \(O(n^{\log_2 3})\), being three extra additions at each step. While the “rounding up” is not important in \(O\)-theory, it matters in practice, and [Mon05] shows various other formulae, e.g.

\[(ax^2 + bx + c)(dx^2 + ex + f) = \]
\[cf(1 - X) + be(-X + 2X^2 - X^3) + ad(-X^3 + X^4) + (b + c)(e + f)(X - X^2) + (a + b)(d + e)(X^3 - X^2) + (a + b + c)(d + e + f)X^2,\]

\(^2\)This formulation is due to [Mon05].
B.3. Karatsuba’s Method

requiring six coefficient multiplications rather than the obvious nine, or eight if we write it as

\[(aX^2 + (bX + c))(dX^2 + (eX + f)) =
\]

\[adX^4 + aX^2(eX + f) + dX^2(bX + c) + (bX + c)(eX + f)\]

and use (B.4) on the last summand (asymptotics would predict \(3\log_2 3 \approx 5.7\), so we are much nearer the asymptotic behaviour). Cascading this formula rather than (B.4) gives \(O(n \log^6 3)\), which as \(\log_3 6 \approx 1.63\) is not as favorable asymptotically. His most impressive formula describes the product of two six-term polynomials in 17 coefficient multiplications, and \(\log_3 17 \approx 1.581\), a slight improvement. We refer the reader to the table in [Mon05], which shows that his armoury of formulae can get us very close to the asymptotic costs.

Theorem 50 We can multiply two (dense) polynomials with \(m\) and \(n\) terms respectively in \(O(\max(m,n) \min(m,n)^{(\log_2 3)-1})\) coefficient operations.

Let the polynomials be

\(f = a_{m-1}Y^{m-1} + \cdots\) and \(f = b_{n-1}Y^{n-1} + \cdots\) Without loss of generality, we may assume \(m \geq n\) (so we have to prove \(O(mn^{\log_2 3} - 1)\), and write \(k = [\frac{m}{n}]\). We can then divide \(f\) into blocks with \(n\) terms (possibly fewer for the most significant one) each: \(f = \sum_{i=1}^{k-1} f_iY^i\). Then

\(fg = \sum_{i=1}^{k-1} (f_ig)Y^i\).

Each \(f_ig\) can be computed in time \(O(n^{\log_2 3})\), and the addition merely takes time \(O(m - n)\) (since there is one addition to be performed for each power of \(Y\) where overlap occurs). Hence the total time is \(O(kn^{\log_2 3})\), and the constant factor implied by the \(O\)-notation allows us to replace \(k = [\frac{m}{n}]\) by \(\frac{m}{n}\), which gives the desired result.

B.3.1 Karatsuba’s method in practice

When it comes to multiplying numbers of \(n\) digits (in whatever base we are actually using, generally a power of 2), the received wisdom is to use \(O(n^2)\) methods for small \(n\) (say \(n < 16\)), Karatsuba-style methods for intermediate \(n\) (say \(16 \leq n < 4096\) and Fast Fourier Transform methods (section B.3.4 or [SS71]) for larger \(n\). However, it is possible for the Fast Fourier Transform to require too much memory, and [Tak10] was forced to use these methods on numbers one quarter the required size, and then nested Karatsuba to combine the results.

B.3.2 Karatsuba’s method and sparse polynomials

The use of the Karatsuba formula and its variants for sparse polynomials is less obvious. One preliminary remark is in order: in the dense case we split the multiplicands in equation (B.6) or its equivalent in two (the same point for
each), and we were multiplying components of half the size. This is no longer the case for sparse polynomials, e.g. every splitting point of

\[(a_7x^7 + a_6x^6 + a_5x^5 + a_0)(b_7 + b_2x^2 + b_1x + b_0)\] (B.7)
gives a 3–1 split of one or the other: indeed possibly both, as when we use \(x^4\) as the splitting point.

Worse, in equation (B.5), the component multiplications were on ‘smaller’ polynomials, whereas, measured in number of terms, this is no longer the case. If we split equation (B.7) at \(x^4\), the sub-problem corresponding to \((a + b)*(c + d)\) in equation (B.5) is

\[(a_7x^3 + a_6x^2 + a_5x^1 + a_0)(b_3 + b_2x^2 + b_1x + b_0)\]

which has as many terms as the original (B.7). This difficulty led [Fat03] to conclude that Karatsuba-based methods did not apply to sparse polynomials. It is clear that they cannot always apply, since the product of a polynomial with \(m\) terms by one with \(n\) terms may have \(mn\) terms, but it is probably the difficulty of deciding when they apply that has led system designers to shun them.

B.3.3 Karatsuba’s method and multivariate polynomials

TO BE COMPLETED

B.3.4 Faster still

It is possible to do still better than \(3^k = \left(2^k\right)^{\log_2 3}\) for multiplying dense polynomials \(f\) and \(g\) with \(n = 2^k\) coefficients. Let \(\alpha_0, \ldots, \alpha_{2n-1}\) be distinct values. If \(h = fg\) is the desired product, then \(h(\alpha_i) = f(\alpha_i)g(\alpha_i)\). Hence if we write \(\tilde{f}\) for the vector \(f(\alpha_0), f(\alpha_1), \ldots, f(\alpha_{2n-1})\), then \(\tilde{h} = \tilde{f} \ast \tilde{g}\), where \(\ast\) denoted element-by-element multiplication of vectors, an \(O(n)\) operation on vectors of length \(2n\). This gives rise to three questions.

1. What should the \(\alpha_i\) be?
2. How do we efficiently compute \(\tilde{f}\) from \(f\) (and \(\tilde{g}\) from \(g\))?
3. How do we efficiently compute \(h\) from \(\tilde{h}\)?

The answer to the first question is that the \(\alpha_i\) should be the \(2n\)-th roots of unity in order, i.e. over \(\mathbb{C}\) we would have \(\alpha_i = e^{2\pi ij/2n}\). In practice, though, we work over a finite field in which these roots of unity exist. If we do this, then the answers to the next two questions are given by the Fast Fourier Transform (FFT) [SS71], and we can compute \(\tilde{f}\) from \(f\) (or \(h\) from \(\tilde{h}\)) in time \(O(k2^k) = O(n \log n)\) (note that we are still assuming \(n\) is a power of 2). Putting these together gives

\[O(n \log n) + O(n \log n) + O(n) + O(n \log n) = O(n \log n).\] (B.8)

\[\tilde{f}\text{ from }f\quad \tilde{g}\text{ from }g\quad \tilde{h}\text{ from }\tilde{f}, \tilde{g}\quad h\text{ from }\tilde{h}\]
While this method works fine for multiplying polynomials over a finite field \( K \) with an appropriate root of unity, or over \( \mathbb{C} \), it requires substantial attention to detail to make it work over the integers. Nevertheless it is practicable, and the point at which this FFT-based method is faster than ones based on Theorem 50 depends greatly on the coefficient domain.

**Notation 39** We will denote \( M_Z(n) \) the asymptotic \( O(M_Z(n)) \) time taken to multiply two integers of length \( n \), and \( M_K[x](n) \) the number of operations in \( K \) or its associated structures needed to multiply two polynomials with \( n \) terms (or of degree \( n \), since in \( O \)-speak the two are the same). Most authors omit the subscripts, and speak of \( M(n) \), when the context is clear.

For integers, the point at which this FFT-based method is faster than ones based on Theorem 50 is typically when the integers are of length a few thousand words.

### B.3.5 Faster division

TO BE COMPLETED

### B.3.6 Faster g.c.d. computation

In order to use this method to compute g.c.d.s faster, we need to solve a slightly more general problem: the extended Euclidean one. If we look at Algorithm 5, the key updating operation is (2.16), which updates the matrix \( \begin{pmatrix} a & b \\ c & d \end{pmatrix} \) based only on \( q_i \), which depends only on the leading entries of \( a_i \) and \( a_{i-1} \). Call this matrix, immediately after \( q_i \) has been computed and used, \( R_{i,0} \). Then (2.14) defines \( R_{0,0} \) and (2.16) becomes

\[
R_{i,0} = \begin{pmatrix} 0 & 1 \\ 1 & -q_i \end{pmatrix} R_{i-1,0}.
\]

(B.9)

TO BE COMPLETED

### B.4 Strassen’s method

Just as Karatsuba’s method (Excursus B.3) lets us multiply dense polynomials of degree \( n - 1 \) (\( n \) terms) in \( O(n^{\log_2 3} \approx n^{1.585}) \) operations rather than \( O(n^2) \), so Strassen’s method, introduced in \[Str69\], allows us to multiply dense \( n \times n \) matrices in \( O(n^{\log_2 7} \approx n^{2.807}) \) operations rather than \( O(n^3) \) operations. Again like Karatsuba’s method, it is based on “divide and conquer” and an ingenious base case for \( n = 2 \), analogous to (B.4).

\[
\begin{pmatrix} c_{1,1} & c_{1,2} \\ c_{2,1} & c_{2,2} \end{pmatrix} = \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix} \begin{pmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{pmatrix}
\]

(B.10)
is usually computed as
\[
\begin{pmatrix}
c_{1,1} & c_{1,2} \\
c_{2,1} & c_{2,2}
\end{pmatrix}
= 
\begin{pmatrix}
a_{1,1}b_{1,1} + a_{1,2}b_{2,1} & a_{1,1}b_{1,2} + a_{1,2}b_{2,2} \\
a_{2,1}b_{1,1} + a_{2,2}b_{2,1} & a_{2,1}b_{1,2} + a_{2,2}b_{2,2}
\end{pmatrix}
\] (B.11)

This method so patently requires eight multiplications of the coefficients that the question of its optimality was never posed. However, [Str69] rewrote it as follows:
\[
\begin{align*}
M_1 & := (a_{1,1} + a_{2,2})(b_{1,1} + b_{2,2}) \\
M_2 & := (a_{2,1} + a_{2,2})b_{1,1} \\
M_3 & := a_{1,1}(b_{1,2} - b_{2,2}) \\
M_4 & := a_{2,2}(b_{2,1} - b_{1,1}) \\
M_5 & := (a_{1,1} + a_{1,2})b_{2,2} \\
M_6 & := (a_{2,1} - a_{1,1})(b_{1,1} + b_{1,2}) \\
M_7 & := (a_{1,2} - a_{2,2})(b_{2,1} + b_{2,2}) \\
c_{1,1} & := M_1 + M_4 - M_5 + M_7 \\
c_{1,2} & := M_3 + M_5 \\
c_{2,1} & := M_2 + M_4 \\
c_{2,2} & := M_1 - M_2 + M_3 + M_6
\end{align*}
\] (B.12)

requiring seven multiplications (rather than eight), though eighteen additions rather than four, so one might question the practical utility of it.

However, it can be cascaded. We first note that we do not require the multiplication operation to be commutative\footnote{A point often glossed over. For example, we can multiply 3 \times 3 matrices in 22 multiplications [Mak86], but this assumes commutativity and hence cannot be cascaded. The best non-commutative results are 23 multiplications [Lad76, CBH11]. Asymptically, both are in any case worse than Strassen, since \log_2 7 \approx 2.807, but \log_3 23 \approx 2.854 and \log_3 22 \approx 2.813.} Hence if we have two 4 \times 4 matrices to multiply, say
\[
\begin{pmatrix}
a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\
a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\
a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\
a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4}
\end{pmatrix}
\begin{pmatrix}
b_{1,1} & b_{1,2} & b_{1,3} & b_{1,4} \\
b_{2,1} & b_{2,2} & b_{2,3} & b_{2,4} \\
b_{3,1} & b_{3,2} & b_{3,3} & b_{3,4} \\
b_{4,1} & b_{4,2} & b_{4,3} & b_{4,4}
\end{pmatrix}
\] (B.13)

we can write this as
\[
\begin{pmatrix}
A_{1,1} & A_{1,2} \\
A_{2,1} & A_{2,2}
\end{pmatrix}
\begin{pmatrix}
B_{1,1} & B_{1,2} \\
B_{2,1} & B_{2,2}
\end{pmatrix}
\] (B.14)

where \( A_{1,1} = \) \begin{pmatrix}
a_{1,1} & a_{1,2} \\
a_{2,1} & a_{2,2}
\end{pmatrix} \) etc., and apply \[ \boxed{B.12} \] both to the multiplications in (B.14), needing seven multiplications (and 18 additions) of 2 \times 2 matrices, and to those multiplications themselves, thus needing 49 = 7 \times 7 multiplications of entries, at the price of 198 = 7 \times 18 + 18 \times 4 additions, rather than 64 recursion 2 \times 2 adds multiplications and 48 additions.
B.4. STRASSEN’S METHOD

Similarly, multiplying $8 \times 8$ matrices can be regarded as (B.14), where now

$$A_{1,1} = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} \end{pmatrix}$$

etc., needing seven multiplications (and 18 additions) of $4 \times 4$ matrices, hence $343 = 7 \times 49$ multiplications of entries, at the price of $1674 = 7 \times 198$ recursion + 18 $4 \times 4$ adds.

If the matrices have $2^k$ rows/columns, then this method requires $7^k = (2^k)^{\log_2 7}$ multiplications rather than the classic $(2^k)^3$. For arbitrary sizes $n$, not necessarily a power of two, the cost of “rounding up” to a power of two is subsumed in the $O$ notation and we see a cost of $O(n^{\log_2 7})$ rather than the classic $O(n^3)$ coefficient multiplications.

We note that $\log_2 7 \approx 2.8074$, and the number of extra coefficient additions required is also $O(n^{\log_2 7})$, being $18(n/2)^2$ additions at each step.

**Theorem 51 (Strassen)** We can multiply two (dense) $n \times n$ matrices in

$$O(n^{\log_2 7}) \approx n^{2.8074}$$

multiplications, and the same order of additions, of matrix entries.

B.4.1 Strassen’s method in practice

Strassen’s method, with floating-point numbers, has break-even points between 400 and 2000 rows (160,000 to 4 million elements) [DN07]. This is achieved with one or two (occasionally three) levels of (B.12), followed by classical $O(n^3)$ multiplication from then on. The pragmatist also notes that modern chips have instructions for manipulating four-vectors of floating-point numbers in a single instruction, so there is no advantage in applying (B.12) all the way.

B.4.2 Further developments

Seven is in fact minimal for $2 \times 2$ matrices [Lan06]. $3 \times 3$ matrices can be multiplied in 23 multiplications rather than the obvious 27 (but $\log_3 23 \approx 2.854 > \log_2 7$), and there is an approximation algorithm with 21 multiplications [Sch71], giving a $O(n^{\log_3 21\approx 2.77})$ general method, but for $3 \times 3$ matrices the best known lower bound is 15 [Lic84, LO11]. In general the best known lower bound is $3n^2 + O(n^{3/2})$ [Lan12, MR12].

In theory one can do better than Theorem 51 $O(n^{2.376})$, [CW90], but these methods require unfeasibly large $n$ to be cost-effective. The complexity of matrix multiplication has to be at least $O(n^3)$ since there are that many entries.

---

5In principle: substantial ingenuity is required to get good performance in practice.

6Recently improved to $O(n^{2.373})$ [Wil12]. This reference actually claimed 2.2327, but this has been corrected to 2.2329 in [Wil14]. The most recent seems to be 2.3728639[LG14 §6.3].
Notation 40  It is customary to assume that the cost of \( n \times n \) matrix multiplication is \( O(n^\omega) \), where \( 2 \leq \omega \leq 3 \).

B.4.3  Matrix Inversion

Lemma 17 (Frobenius)  Let \( A \) be a square matrix divided into blocks \( \begin{pmatrix} P & Q \\ R & S \end{pmatrix} \), with \( P \) square and nonsingular. Assume that \( U = S - R(P^{-1}Q) \) is also nonsingular. Then

\[
A^{-1} = \begin{pmatrix}
P^{-1} + (P^{-1}Q)(U^{-1}RP^{-1}) & -(P^{-1}Q)U^{-1} \\
(U^{-1}RP^{-1}) & U^{-1}
\end{pmatrix}.
\]  (B.15)

The proof is by direct verification. Note that the computation of the right-hand side of (B.15) requires two matrix inverses (\( P \) and \( U \)) and six matrix multiplications (viz. \( P^{-1}Q \), \( R(P^{-1}Q) \), \( RP^{-1} \), \( U(RP^{-1}) \), \( (P^{-1}Q)(U^{-1}RP^{-1}) \) and \( (P^{-1}Q)(U^{-1}) \)). If the conditions of Lemma 17 were always satisfied, and we took \( P \) to be precisely one quarter the size of \( A \) (half as many rows and half as many columns), this would show that the complexity of matrix inverse is also \( O(n^\omega) \) if \( \omega > 2 \) (if \( \omega = 2 \) we get an extra factor of \( \log n \)).

Lemma 18  If \( B \) is nonsingular then \( A := B^TB \) is symmetric and positive definite (for any non-zero \( x \), \( x^TBAx > 0 \)).

Lemma 19  If \( A \) is symmetric and positive definite then, in the notation of Lemma 17, so are \( P \) and \( U \).

Theorem 52  If \( B \) is an invertible square matrix, \( B^{-1} \) can be computed in time \( O(n^\omega) \) if \( \omega > 2 \) (if \( \omega = 2 \) we get an extra factor of \( \log n \)).

Proof.  We compute \( A := B^TB \) in time \( O(n^\omega) \), its inverse by the process after Lemma 17 and then \( B^{-1} := A^{-1}A^T \) (another \( O(n^\omega) \)).
Appendix C

Systems

This appendix discusses various computer algebra systems, especially from the point of view of their internal data structures and algorithms, and how this relates to the ideas expressed in the body of this book. We do not discuss the user interfaces as such, nor is this intended to replace any manuals, or specialist books.

Moses [Mos71] described systems as falling into various categories, depending on the extent of automatic transformation. While most systems today are what Moses would call ‘new left’, the extent of automatic transformation still varies dramatically.

C.1 Axiom

C.1.1 Overview

See Figure C.1.

We note that the polynomials have been expanded and the greatest common divisor cancelled (in the terminology of Definition 4 we have canonical forms for this data type). We are told the data type of the result, Fraction Polynomial Integer, i.e. the field of fractions of $\mathbb{Z}[\text{variables}]$. In fact, this answer happens to have denominator 1, so lies in Polynomial Integer, but the system does not automatically check for such retractions.

C.1.2 History

This system [JS92], with which the author has been heavily involved, can be seen as the first of the ‘Bourbakist’ systems — since followed, in particular, by Magma [BCM94] and SAGE. It was developed at IBM Yorktown Heights,
Figure C.1: Axiom output

(1) -> (x^2-1)^10/(x-1)^10

(1)

10 9 8 7 6 5 4 3 2
x + 10x + 45x + 120x + 210x + 252x + 210x + 120x + 45x + 10x + 1

Type: Fraction Polynomial Integer

(2) -> %^2

(2)

20 19 18 17 16 15 14 13
x + 20x + 190x + 1140x + 4845x + 15504x + 38760x + 77520x +

12 11 10 9 8 7 6
125970x + 167960x + 184756x + 167960x + 125970x + 77520x + 38760x +

5 4 3 2
15504x + 4845x + 1140x + 190x + 20x + 1

Type: Fraction Polynomial Integer

originally under name of Scratchpad II, with its implementation language known as Modlisp [Jen79, DJ80].

It was commercialised under the name Axiom by NAG Ltd in the 1990s, but never achieved the necessary commercial success, and is now open-source — [www.axiom-developer.org](http://www.axiom-developer.org).

### C.1.3 Structure

All Axiom objects are typed, as seen in Figures C.1 and C.2. In the second one, we see that a and b are objects of appropriate, but different, types. The system ‘knows’ (for details, see Doy99) that an appropriate common type is Polynomial Integer, which is what c becomes.

### C.2 Macsyma

#### C.2.1 Overview

See Figure C.3. We note that the g.c.d. is only cancelled on demand (radcan stands for ‘radical canonicalizer’). The equation at line 3 is treated as such, but if we force the system (line 4) to regard it as a Boolean, the result is false, despite the fact that the comparands are mathematically equal.
C.2. MACSYMA

Figure C.2: Axiom type system

(1) -> a:=1

(1) 1

Type: PositiveInteger

(2) -> b:=x

(2) x

Type: Variable x

(3) -> c:=a+b

(3) x + 1

Type: Polynomial Integer

Figure C.3: Macsyma output

(%i1) (x^2-1)^10/(x+1)^10;

\[
\frac{2^{10}(x - 1)}{10(x + 1)}
\]

(%o1)

(%i2) radcan(%);

\[
x - 10x + 45x - 120x + 210x - 252x + 210x - 120x + 45x - 10x + 1
\]

(%o2) 

(%i3) %= (x^2-1)^10/(x+1)^10;

\[
x - 10x + 45x - 120x + 210x - 252x + 210x - 120x + 45x - 10x + 1
\]

(%o3) 

(%i5) if % then 1 else 2;

(%o5) 2
APPENDIX C. SYSTEMS

C.2.2 History

C.3 Maple

C.3.1 Overview

See Figure C.4. Note that cancelling the g.c.d., and expansion, did not take place until asked for. Also, while the exponents in the expanded form of \((x + 1)^{10}\) were in order, the same was not true of its square.

C.3.2 History

This system started in the early 1980s, a period when computer power, and particularly memory, were much less than they are today. It was designed to support multiple users, particularly classes of students, on what were, by the standards of the time, relatively modest university resources. Two important early references are [CGGG83, CFG84]. These circumstances led to three design principles.

1. The system had to be small — early hosts systems had limitations of, for example, 110K words of memory. In particular, users must not pay, in terms of memory occupancy, for features they were not using, which led to a ‘kernel plus loadable modules’ design, where the kernel knew basic algebraic features, and the rest of the system was in loadable modules written in the Maple language itself, and therefore interpreted rather than compiled. The precise definition of ‘basic’ has changed over time — see section C.3.4.

2. The system had to be portable — early versions ran on both 32-bit VAX computers and 36-bit Honeywell computers. Its kernel, originally some 5500 lines of code, was macro-processed into ‘languages of the BCPL family’, of which the most successful, and the one used today, is C.

3. Memory was scarce, and hence re-use had to be a priority.

C.3.3 Data structures

These considerations led to an “expression DAG” design (see page 43). The internal form of an expression is a node of a certain type followed by an arbitrary number (hence we talk about an n-ary tree) of operands. If \(A\) is a Maple expression, the construct \(\text{op}(0,A)\) gives the type of the node, and \(\text{op}(i,A)\) gives the \(i\)-th operand. This is demonstrated by the session in table C.1 which builds the DAG shown in figure C.5. It might be assumed from table C.1 that Maple had some ‘preferred’ order which \(A\) matched but \(B\) did not. However, if we look at table Maple:code2 (run in a fresh session of Maple), we see that \(B\) is now the preferred instance. The point is that, since Maple’s internalising process...
Figure C.4: Maple output

```maple
> (x^2-1)^10/(x+1)^10;

2 10
(x - 1) --------------------
10
(x + 1)

> simplify(%);

10
(x - 1)

> expand(%);

10 9 8 7 6 5 4 3
x - 10 x + 45 x - 120 x + 210 x - 252 x + 210 x - 120 x

2
+ 45 x - 10 x + 1

> %^2;

10 9 8 7 6 5 4 3
(x - 10 x + 45 x - 120 x + 210 x - 252 x + 210 x - 120 x

2
+ 45 x - 10 x + 1)

> expand(%);

2 3 5 6 7
1 + 190 x - 20 x - 1140 x - 15504 x + 38760 x - 77520 x

8 9 10 20 19
+ 125970 x - 167960 x + 184756 x + x - 20 x

18 17 16 15 14
+ 190 x - 1140 x + 4845 x - 15504 x + 38760 x

13 12 11 4
- 77520 x + 125970 x - 167960 x + 4845 x
```
Table C.1: A small Maple session

> A := x + y + a*b;
> op(0, A);
> op(3, A);
> op(0, op(3, A));
> B := y + b*a + x;

Figure C.5: Tree for A, B corresponding to table C.1

```
+   *
\ y  \ a/b
   \  \\
   \  \\
  \  \\
 \ x
```

structures these trees, in which + nodes cannot be children of + nodes, or + nodes of * nodes (hence implicitly enforcing associativity of these operators), and the children are unordered (hence implicitly enforcing commutativity\(^3\)), once the subtree corresponding to a*b has been built, as in the first line of table C.1, an equivalent tree, such as that corresponding to b*a, is stored as the same tree. If it is fundamentally unordered, which way does it print? The answer is given in [CFG+84, p. 7]

if the expression is found, the one in the table is used, and the [new] one is discarded.

Hence in table C.1 the presence of a*b means that b*a automatically becomes a*b. The converse behaviour occurs in table C.2

In terms of the \(10\) algebraic rules on pp. 33–34, this structure automatically follows all except (8), which is implemented only in the weaker form (8').

---

2Referred to as the ‘simplifier’ in [CGGG+83, CFG+84], but we do not use that term to avoid confusion with Maple’s `simplify` command.

3We have found an example where this is not the case, but this is explicitly described as a bug by Maple’s Senior Architect.

Two simpl’d PROD DAGs containing the same entries but in a different order is a kernel bug by definition. [Vor10]
The Maple command \texttt{expand} implements (8) fully, therefore producing, for polynomials, what we referred to (page 41) as a distributed representation\footnote{But we should note that there are no guaranteed mathematical properties of the ordering. Good properties are provided by the \texttt{MonomialOrders} of the \texttt{Groebner} package.}. This is therefore canonical (definition 4), but in a limited sense: it is canonical within a given Maple session, but may vary between sessions. This means that operations such as \texttt{op}(i,A) (i \neq 0) are not necessarily consistent between sessions.

\subsection*{C.3.4 Heuristic GCD} \cite{CGG84,CGG89}

\subsection*{C.3.5 Conclusion}

There are many books written on Maple, particularly in the educational context. A comprehensive list would be out-of-date before it was printed, but we should mention \cite{Cor02}.

\begin{table}[h]
\centering
\begin{tabular}{|c|}
\hline
Table C.2: Another small Maple session  \\
\hline
\end{tabular}
\caption{Another small Maple session}
\end{table}

\begin{verbatim}
> B:=y+b*a+x;  B := y + b \ a + x  \\
> op(0,B);  \\
+  \\
> op(2,B);  \\
b \ a  \\
> op(0,op(2,B));  \\
\ast  \\
> A:=x+y+a*b;  A := y + b \ a + x
\end{verbatim}

\begin{figure}[h]
\centering
\begin{tikzpicture}
\node (y) at (0,0) {$y$};
\node (x) at (1,0) {$x$};
\node (a) at (1,-1) {$a$};
\node (b) at (0,-1) {$b$};
\node (plus) at (0.5,0.5) {$+$};
\node (times) at (0.5,-0.5) {$\ast$};
\draw (y) -- (plus);  \\
\draw (plus) -- (times);  \\
\draw (times) -- (a);  \\
\draw (times) -- (b);
\end{tikzpicture}
\caption{Tree for $A, B$ corresponding to table C.2}
\end{figure}
Figure C.7: MuPAD output

```matlab
>> (x^2+1)^10/(x-1)^10
ans =
(x^2 + 1)^10/(x - 1)^10
```

```matlab
>> (x^2-1)^10/(x+1)^10
ans =
(x^2 - 1)^10/(x + 1)^10
```

```matlab
>> simplify(ans)
ans =
(x - 1)^10
```

```matlab
>> expand(ans)
ans =
x^10 - 10*x^9 + 45*x^8 - 120*x^7 + 210*x^6 - 252*x^5 + 210*x^4 - 120*x^3 + 45*x^2 - 10*x + 1
```

```matlab
>> ans == (x^2 + 1)^10/(x - 1)^10
ans =
0
```

C.4 MuPAD

C.4.1 Overview

See Figure C.7 We note that the g.c.d. is only cancelled on demand, and that the result of the equality test is 0, i.e. false.

C.4.2 History

This system [FGK+94] could easily have been written off as “yet another computer algebra system”, until MathWorks Inc. bought the company, and made MuPAD into the “symbolic toolbox” of MatLab (replacing Maple). The session
Figure C.8: Reduce output

1: \((x^2-1)^{10}/(x+1)^{10};\)

\[
x^{10} - 10x^9 + 45x^8 - 120x^7 + 210x^6 - 252x^5 + 210x^4 - 120x^3 + 45x^2 - 10x + 1
\]

2: off exp;
3: \((x^2-1)^{10}/(x+1)^{10};\)

\[
(x-1)^{10}
\]

4: off gcd;
5: \((x^2-1)^{10}/(x+1)^{10};\)

\[
(x-1)^{10}
\]

6: \((x^2-1)^{10}/(x^2+2*x+1)^{10};\)

\[
\frac{(x^2 - 1)^{10}}{(x^2 + 2x + 1)^{10}}
\]

7: on gcd;
8: ws;

\[
\frac{(x - 1)^{10}}{(x + 1)^{10}}
\]

above is taken from MatLab.

C.5 Reduce

C.5.1 Overview

Note that Reduce produces output that cuts/pastes as TeX, as in Figure C.8. By default, the output is canonical (Definition [4]), but this can be changed via the \texttt{exp} (short for \texttt{expand}) and \texttt{gcd} (compute them) switches. Note that exact division tests are done even when \texttt{gcd} is off, as in line 5, and we need a more challenging example, as in line 6.

C.5.2 History
## Appendix D

### Index of Notation

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
<th>Reference</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(S)$</td>
<td>The ideal generated by $S$</td>
<td>Notation 4</td>
<td>26</td>
</tr>
<tr>
<td>$(F, A, B)_B$</td>
<td>A Bourbakist function definition</td>
<td>Notation 5</td>
<td>21</td>
</tr>
<tr>
<td>$(F, A, B)_B</td>
<td>C$</td>
<td>$(F, A, B)_B$ restricted to $C$</td>
<td>Notation 5</td>
</tr>
<tr>
<td>$? =$</td>
<td>An equation whose validity depends on interpretations</td>
<td>Notation 33</td>
<td>251</td>
</tr>
<tr>
<td>$|f|_k$</td>
<td>The $k$-norm of $f(x)$</td>
<td>Notation 36</td>
<td>271</td>
</tr>
<tr>
<td>$p \land q$</td>
<td>The logical and of $p$ and $q$</td>
<td>Definition 68</td>
<td>125</td>
</tr>
<tr>
<td>$p \lor q$</td>
<td>The logical (inclusive) or of $p$ and $q$</td>
<td>Definition 68</td>
<td>125</td>
</tr>
<tr>
<td>$\neg p$</td>
<td>The logical negation of $p$</td>
<td>Definition 68</td>
<td>125</td>
</tr>
<tr>
<td>$A$</td>
<td>The algebraic numbers</td>
<td>Notation 27</td>
<td>215</td>
</tr>
<tr>
<td>$C$</td>
<td>The complex numbers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\mathcal{C}$</td>
<td>Any field of constants</td>
<td>Definition 90</td>
<td>222</td>
</tr>
<tr>
<td>$\text{con}(F)$</td>
<td>The cone of $F$</td>
<td>Theorem 32</td>
<td>139</td>
</tr>
<tr>
<td>$R_{\text{const}}$</td>
<td>The constants of $R$</td>
<td>Definition 90</td>
<td>222</td>
</tr>
<tr>
<td>$\text{den}(f)$</td>
<td>The denominator of $f$</td>
<td>Proposition 12</td>
<td>50</td>
</tr>
<tr>
<td>$\epsilon$</td>
<td>The relaxation of $\epsilon$</td>
<td>Notation 19</td>
<td>79</td>
</tr>
<tr>
<td>$H(f)$</td>
<td>The height of $f(x)$, $</td>
<td></td>
<td>f</td>
</tr>
<tr>
<td>$L(f)$</td>
<td>The length of $f(x)$, $</td>
<td></td>
<td>f</td>
</tr>
<tr>
<td>$\text{li}$</td>
<td>The logarithmic integral</td>
<td>Notation 31</td>
<td>237</td>
</tr>
<tr>
<td>$\text{Ln}$</td>
<td>The multivalued logarithm</td>
<td>Notation 34</td>
<td>255</td>
</tr>
<tr>
<td>Notation</td>
<td>Meaning</td>
<td>Reference</td>
<td>Page</td>
</tr>
<tr>
<td>----------</td>
<td>----------</td>
<td>-----------</td>
<td>------</td>
</tr>
<tr>
<td>$M(f)$</td>
<td>The Mahler measure of $f(x)$</td>
<td>Notation 36</td>
<td>271</td>
</tr>
<tr>
<td>$M(f)$</td>
<td>The Mahler measure of $f(x_1, \ldots, x_n)$</td>
<td>Definition 102</td>
<td>272</td>
</tr>
<tr>
<td>$M(n)$</td>
<td>The time needed to multiply two objects of length $n$</td>
<td>Notation 39</td>
<td>291</td>
</tr>
<tr>
<td>$\text{mon}(H)$</td>
<td>The (multiplicative) monoid of $H$</td>
<td>Theorem 32</td>
<td>139</td>
</tr>
<tr>
<td>$\mathbb{N}$</td>
<td>The nonnegative integers (natural numbers)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Newton($p$)</td>
<td>The Newton series</td>
<td>Notation 15</td>
<td>47</td>
</tr>
<tr>
<td>$\text{num}(f)$</td>
<td>The numerator of $f$</td>
<td>Proposition 12</td>
<td>50</td>
</tr>
<tr>
<td>$\mathbb{Q}$</td>
<td>The rational numbers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\mathbb{R}$</td>
<td>The real numbers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$S(f,g)$</td>
<td>The $S$-polynomial of $f$ and $g$</td>
<td>Definition 43</td>
<td>91</td>
</tr>
<tr>
<td>$S_f$</td>
<td>The “sugar’ of the polynomial $f$</td>
<td>Definition 50</td>
<td>101</td>
</tr>
<tr>
<td>$S_{(f,g)}$</td>
<td>The “sugar’ of the polynomial pair $(f,g)$</td>
<td>Definition 50</td>
<td>101</td>
</tr>
<tr>
<td>$\text{Sk}(f)$</td>
<td>The skeleton of the polynomial $f$</td>
<td>Definition 77</td>
<td>171</td>
</tr>
<tr>
<td>Sqrt</td>
<td>The multivalued square root</td>
<td>Notation 34</td>
<td>255</td>
</tr>
<tr>
<td>$W(z)$</td>
<td>The Lambert $W$ function</td>
<td></td>
<td>231</td>
</tr>
<tr>
<td>$\mathbb{Z}$</td>
<td>The integers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\mathbb{Z}_N$</td>
<td>The integers modulo $N$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\omega$</td>
<td>The exponent of matrix multiplication</td>
<td>Notation 40</td>
<td>294</td>
</tr>
<tr>
<td>$\omega(z)$</td>
<td>The Wright $\omega$ function</td>
<td></td>
<td>231</td>
</tr>
</tbody>
</table>
Bibliography


BIBLIOGRAPHY


<table>
<thead>
<tr>
<th>Reference</th>
<th>Author(s)</th>
<th>Title</th>
<th>Source</th>
</tr>
</thead>
</table>


<table>
<thead>
<tr>
<th>Bibliography Reference</th>
<th>Title and Details</th>
</tr>
</thead>
</table>


Index

Abel’s theorem, 72
Additive complexity, 45
Admissible orderings, 41
Algebra
  Free, 34
  Algebraic
    closure, 28
    curve, 13
    decomposition, 128
      block-cylindrical, 133
      cylindrical, 131
      sampled, 130
    function, 215
    integer, 215
    number, 215
    proposition, 125
    variable, 119
Algorithm
  Atlantic City, 31
  Bareiss, 85
  Buchberger, 92
    Extended, 111
  Cantor–Zassenhaus, 197
  Chinese Remainder, 278
    for Polynomials, 279
    Multivariate, 279
    Polynomial form, 278
  Euclid’s, 54
  Extended Euclid, 60
  Extended Subresultant, 61
  Faugère–Gianni–Lazard–Mora, 107
  Hermite’s, 223
  IntExp–Polynomial, 241
  IntExp–Rational Expression, 242
  IntLog–Polynomial, 237
  IntLog–Rational Expression, 238
  Las Vegas, 31
  Monte Carlo, 30
  Ostrogradski–Horowitz, 225
  Parallel Risch, 248
  Primitive p.r.s., 58
  Sturm sequence evaluation, 77
  Subresultant, 59
  Trager–Rothstein, 227
  Vandermonde solver, 281
  variant, 281
Alternations
  of quantifiers, 128
Arithmetic
  Classical, 29
  Ascending Chain Condition, 26
  Associates, 27
  Associativity, 25
  Assumption
    Zippel, 171
  Axiom, 295
Bad reduction, 152, 163, 182
Banded Matrix, 81
Bareiss algorithm, 85
Basis
  completely reduced, 92
  Gröbner, 91
  completely reduced, 92
  shape, 115
Bezout’s Identity, 60
Birational equivalence, 75
Block-cylindrical
  decomposition, 133
Bound, 270
  Cauchy, 273
  Hadamard, 270
  Knuth, 274
  Mahler, 274

332
<table>
<thead>
<tr>
<th>Term</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branch Cut</td>
<td>257</td>
</tr>
<tr>
<td>Removable</td>
<td></td>
</tr>
<tr>
<td>Buchberger</td>
<td>92, 111</td>
</tr>
<tr>
<td>Algorithm</td>
<td></td>
</tr>
<tr>
<td>Extended</td>
<td></td>
</tr>
<tr>
<td>Criterion</td>
<td>99, 111</td>
</tr>
<tr>
<td>First</td>
<td></td>
</tr>
<tr>
<td>gcd</td>
<td></td>
</tr>
<tr>
<td>lcm</td>
<td></td>
</tr>
<tr>
<td>Third</td>
<td></td>
</tr>
<tr>
<td>Theorem</td>
<td>92</td>
</tr>
<tr>
<td>Budan–Fourier theorem</td>
<td>78, 286</td>
</tr>
<tr>
<td>Calculus</td>
<td></td>
</tr>
<tr>
<td>Fundamental Theorem of</td>
<td>222, 258</td>
</tr>
<tr>
<td>Candid representation</td>
<td>19</td>
</tr>
<tr>
<td>Canonical representation</td>
<td>19</td>
</tr>
<tr>
<td>Cantor–Zassenhaus Algorithm</td>
<td>197</td>
</tr>
<tr>
<td>Cauchy bound</td>
<td>273</td>
</tr>
<tr>
<td>Cell</td>
<td>128</td>
</tr>
<tr>
<td>Chain</td>
<td></td>
</tr>
<tr>
<td>Regular</td>
<td>119</td>
</tr>
<tr>
<td>Chain Condition</td>
<td></td>
</tr>
<tr>
<td>Ascending</td>
<td>26</td>
</tr>
<tr>
<td>Descending</td>
<td>42</td>
</tr>
<tr>
<td>Characteristic</td>
<td>28</td>
</tr>
<tr>
<td>set</td>
<td>119</td>
</tr>
<tr>
<td>Chinese Remainder Theorem</td>
<td>277</td>
</tr>
<tr>
<td>(Polynomial)</td>
<td></td>
</tr>
<tr>
<td>Circulant Matrix</td>
<td>81</td>
</tr>
<tr>
<td>Classical</td>
<td></td>
</tr>
<tr>
<td>arithmetic</td>
<td>29</td>
</tr>
<tr>
<td>Closure</td>
<td></td>
</tr>
<tr>
<td>Zariski</td>
<td>121</td>
</tr>
<tr>
<td>Coefficient</td>
<td></td>
</tr>
<tr>
<td>leading</td>
<td>35</td>
</tr>
<tr>
<td>of a polynomial</td>
<td>33</td>
</tr>
<tr>
<td>Commutativity</td>
<td>25</td>
</tr>
<tr>
<td>Complexity</td>
<td></td>
</tr>
<tr>
<td>additive</td>
<td>45</td>
</tr>
<tr>
<td>Conjecture</td>
<td></td>
</tr>
<tr>
<td>Schamel’s</td>
<td>263</td>
</tr>
<tr>
<td>Constant</td>
<td></td>
</tr>
<tr>
<td>definition of</td>
<td>222</td>
</tr>
<tr>
<td>Constraint</td>
<td></td>
</tr>
<tr>
<td>equational</td>
<td>140</td>
</tr>
<tr>
<td>Content (of polynomials)</td>
<td>55</td>
</tr>
<tr>
<td>Cylinder</td>
<td>131</td>
</tr>
<tr>
<td>Cylindrical algebraic decomposition</td>
<td>131</td>
</tr>
<tr>
<td>Partial</td>
<td>140</td>
</tr>
<tr>
<td>Decomposition</td>
<td></td>
</tr>
<tr>
<td>algebraic</td>
<td>128</td>
</tr>
<tr>
<td>block-cylindrical</td>
<td>133</td>
</tr>
<tr>
<td>cylindrical</td>
<td>131</td>
</tr>
<tr>
<td>partial</td>
<td>140</td>
</tr>
<tr>
<td>equiprojectable</td>
<td>120</td>
</tr>
<tr>
<td>Lemma (exponential)</td>
<td>239</td>
</tr>
<tr>
<td>Lemma (logarithmic)</td>
<td>235</td>
</tr>
<tr>
<td>Lemma (rational expressions)</td>
<td>223</td>
</tr>
<tr>
<td>order-invariant</td>
<td>130</td>
</tr>
<tr>
<td>Partial Fraction</td>
<td>62</td>
</tr>
<tr>
<td>sign-invariant</td>
<td>130</td>
</tr>
<tr>
<td>square-free</td>
<td>70</td>
</tr>
<tr>
<td>Defining formula</td>
<td>128</td>
</tr>
<tr>
<td>Degree</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>11</td>
</tr>
<tr>
<td>Denominator</td>
<td></td>
</tr>
<tr>
<td>common</td>
<td>50</td>
</tr>
<tr>
<td>Dense</td>
<td></td>
</tr>
<tr>
<td>matrix</td>
<td>80</td>
</tr>
<tr>
<td>polynomial</td>
<td>35</td>
</tr>
<tr>
<td>Density Theorem</td>
<td></td>
</tr>
<tr>
<td>Frobenius</td>
<td>193, 198</td>
</tr>
<tr>
<td>Descartes rules of signs</td>
<td>78, 285</td>
</tr>
<tr>
<td>Descending Chain Condition</td>
<td>42</td>
</tr>
<tr>
<td>Difference</td>
<td></td>
</tr>
<tr>
<td>field</td>
<td>249</td>
</tr>
<tr>
<td>ring</td>
<td>249</td>
</tr>
<tr>
<td>Differential</td>
<td></td>
</tr>
<tr>
<td>field</td>
<td>221</td>
</tr>
<tr>
<td>ring</td>
<td>221</td>
</tr>
<tr>
<td>Dimension</td>
<td></td>
</tr>
<tr>
<td>ideal</td>
<td>94</td>
</tr>
<tr>
<td>linear space</td>
<td>88</td>
</tr>
<tr>
<td>mixed</td>
<td>95</td>
</tr>
<tr>
<td>triangular set</td>
<td>119</td>
</tr>
<tr>
<td>Directed Acyclic Graph</td>
<td>43</td>
</tr>
<tr>
<td>Discriminant</td>
<td>269</td>
</tr>
<tr>
<td>Distributed representation</td>
<td>11</td>
</tr>
<tr>
<td>Distributivity</td>
<td>26</td>
</tr>
<tr>
<td>Division</td>
<td>54</td>
</tr>
</tbody>
</table>
Dodgson–Bareiss theorem, 84

Domain
- g.c.d., 53
- integral, 27

Elementary
- expression, 230
- generator, 229

Elimination
- Gauss, 82
- fraction-free, 85
- ideal, 107
- ordering, 96

Equality
- algebraic, 51
- Equational constraint, 140

Eqiprojectable
- decomposition, 120
- variety, 119

Equivalent, 89

Euclid’s
- Algorithm, 54
- Algorithm (Extended), 60, 179
- Theorem, 54

Excel, 15

Existential theory of the reals, 139

Exponentiation
- polynomial, 34

Expression
- DAG representation, 43
- tree representation, 43

Factorization
- shape, 191

Farey
- fractions, 178
- reconstruction, 179

Farey Reconstruction, 179

Faugère–Gianni–Lazard–Mora
- algorithm, 107

Fermat
- Little Theorem of, 194

Field
- difference, 249
- differential, 221
- of fractions, 27

real closed, 124

Fill-in (loss of sparsity), 81

Formula
- defining, 128

Free Algebra, 34

Frobenius
- Lemma, 294
- Frobenius Density Theorem, 193, 198

Function
- Algebraic, 215
- Hilbert, 116

Fundamental
- Theorem of Calculus, 222, 268

Galois
- group, 192

Gauss
- elimination, 82
- Lemma, 53

Generalised
- Polynomial, 239
- Geobuckets, 39
- Gianni–Kalkbrener
- algorithm, 106
- theorem, 105

Good reduction, 152, 163, 182

Graeffe method, 274

Greatest common divisor, 27, 53
- domain, 55
- Gröbner base, 91, 118
  - completely reduced, 92
- Gröbner fan, 111
- Gröbner trace idea, 180
- Gröbner walk, 111

Hadamard bound, 270

Hensel
- Algorithm
  - General, 189
  - Lemma, 189
- Lifting
  - Hybrid, 203
  - Linear, 200, 201, 208
  - Multivariate, 211
  - Quadratic, 201

Hermite
Algorithm, 224
Hilbert function, 116
polynomial, 116
History of Computer Algebra, 15
Ideal, 26
elimination, 107
polynomial, 89
Principal, 27
saturated, 122
Identity
Bezout’s, 60
Inequality
Landau, 272
Landau–Mignotte, 150
Initial, 118
Integer
Algebraic, 215
Integral domain, 27
Integration indefinte, 222
Intermediate Expression Swell, 16
Example, 180
IntExp–Polynomial
Algorithm, 241
IntExp–Rational Expression
Algorithm, 242
IntLog–Polynomial
Algorithm, 237
IntLog–Rational Expression
Algorithm, 238
Inverse
matrix, 82
Knuth
bound, 274
Landau Inequality, 272
Landau notation, 28
Landau–Mignotte Inequality, 150
Laurent
Polynomial, 239
Least common multiple, 53
Lemma
Frobenius, 294
Hensel’s, 189
Thom’s, 79
Liouville’s Principle, 231
Parallel Version, 247
Macsyma, 296
Mahler
bound, 274
measure, 272
Main variable, 118
Maple, 298
Matrix
Banded, 81
Circulant, 81
inverse, 82
Sylvester, 267
Toeplitz, 81
measure
Mahler, 272
Minimal Polynomial, 217
Monic polynomial, 35
Monomial, 41
leading, 90
Multiplicity of a solution, 70
MuPAD, 302
Newton
series, 47
Noetherian ring, 26
Normal representation, 19
Normal Selection Strategy, 100
Number
Algebraic, 215
Numerator, 50
Ordering
admissible, 41
elimination, 96
matrix, 97
purely lexicographic, 96
total degree, then lexicographic, 96
total degree, then reverse lexicographic, 96
weighted, 97
Ostrogradski–Horowitz
Algorithm, 225
INDEX

p-adic numbers, 189
p.r.s., 57
primitive, 58
subresultant, 59
Parallel Risch Algorithm, 248
Partial
Cylindrical algebraic decomposition, 140
Fraction Decomposition, 62
Polynomial
definition, 33
factored representation, 39
generalised, 239
height of, 271
Hilbert, 116
Laurent, 239
length of, 271
minimal, 217
remainder sequence, 57
signed, 57
Wilkinson, 273
Positivstellensatz, 139
Prenex normal form, 127
Primitive
(of polynomials), 55
p.r.s., 58
part, 55
Principal
ideal, 27
domain, 27
Principle
Liouville’s, 231
(Parallel Version), 247
Tarski–Seidenberg, 127
Problem
Elementary integration, 234
Elementary Risch differential equation, 234
Projection, 134
Proposition
algebraic, 125
semi-algebraic, 125
Pseudo-euclidean algorithm, 57
Pseudo-remainder, 57 120
Quantifier
alternation, 128
elimination, 127
Quantifier-free, 126
Quotient (polynomial), 54
Quotient rule, 222
Radical
(i.e. n-th root), 72
(of an ideal), 94
Real, 126
Real closed fields, 124
existential theory of, 139
Recursive representation, 41
Reduce, 303
Reduction
bad, 152 163 182
good, 152 163 182
Reductum, 35
Regular
Chain, 119
Remainder (polynomial), 54
Removable
Branch Cut, 257
Singularity, 252
Representation
distributed (of polynomials), 41
expression DAG, 43
expression tree, 43
factored (of polynomials), 39
of objects, 18
candid, 19
canonical, 19
normal, 19
recursive (of polynomials), 41
Straight-Line Program, 43
Resultant, 207 269
modular calculation, 177
Ring, 25
difference, 249
differential, 221
Risch
differential equation problem, 234
induction hypothesis, 235
Integration Theorem, 234
Parallel Algorithm, 248
RootOf, 32 73
INDEX

Rule
- quotient, 222

Saturated ideal, 122
Schanuel’s Conjecture, 263
Series
- Newton, 47
Set
- triangular, 119
Shape
- factorization, 191
Shape basis, 115
sign
- variations, 285
Signed polynomial remainder sequence, 57
Singularity
- Removable, 252
Sparse
- matrix, 81
- polynomial, 35
Sparsity
- of a polynomial, 37
S-polynomial, 91 117
Square-free, 65
- decomposition, 65
Stability
- numerical, 24
Straight-Line Program Representation, 43
Strassen’s Theorem, 293
Strategy
- Normal Selection, 100
- Sugar, 101
Sturm
- Habicht sequence, 76
- sequence, 76
Subresultant algorithm, 59
Sugar, 101
Sugar Selection Strategy, 101
Support
- of a polynomial, 37
Sylvester matrix, 267
System
- quasi-algebraic, 120
Systems
- Axiom, 295
- Macsyma, 296
- Maple, 298
- MuPAD, 302
- Reduce, 303
- Tarski–Seidenberg principle, 127
- Term
- leading, 90
Theorem
- Buchberger, 92
- Chinese Remainder, 277
- Chinese Remainder (Polynomial), 279
- Risch Integration, 234
- Thom’s Lemma, 79
- Toeplitz Matrix, 81
- Total degree, 41
- Trager–Rothstein Algorithm, 227
- Transcendental, 16
- Strongly, 263
- Triangular set, 119
- Tschirnhaus transformation, 70
- Unique factorisation domain, 58
- Unit, 27
- Vandermonde systems, 280
- Variable
- algebraic, 119
- main, 118
- Wilkinson Polynomial, 273
- Zariski closure, 121
- Zero-divisors, 27
- Zippel
- Assumption, 171
The total-degree Gröbner base from page\textsuperscript{109}
\[(42c^6 + 555 + 293a^7 + 1153ac^5b + 3695ab + 323b^3 + 2035a^2c + 1642abc - 1211b^2c + 2252c^2 - 1252bc^2 - 31c^2 + 1347ac^2 - 3495ab + 1544b^2 + 1100ac + 2574bc - 368c^2 + 2849a + 281b + 4799c, 21bc^3 - 57 - 10a^3 - 41a^2b + 183ab^2 - 25b^3 - 104a^2c - 65abc + 91b^2c - 129ac^2 - 59bc^2 - 16c^3 - 33a^2 + 225ab - 142b^2 + 67ac - 174bc - 5c^2 - 154a - 46b - 310c, 21ac^3 - 75 - 29a^3 - 121a^2b + 369ab^2 - 41b^3 - 226a^2c - 178abc + 161bc^2 + 267ac^2 + 148bc^2 + 4c^3 - 123a^2 + 411ab - 206b^2 + 146ac - 324bc + 38c^2 - 329a - 62b - 584c, 14b^2c^2 + 5 + 5a^3 + 9a^2b + 2ab^2 - b^3 + 23a^2c + 10abc - 21b^2c + 15ac^2 - 8a^2c^2 + 3c^2 + 3a^2 - 19ab - 4b^2 - 6ac - 26bc - 10c - 7a + 6b + 3c, 21abc^2 + 30 - a^3 - 2a^2b + 51ab^2 + 8b^3 + 40a^2c + 25abc - 56bc^2 + 69ac^2 - 13bc^2 + 11c^3 - 18a^2 - 72ab + 53b^2 + 8ac + 54bc + 10c^3 + 56a + 29b + 116c, 14ac^2 + 11c^3 + 2a^3 + 31a^2b - 74ab^2 + 9b^3 + 45a^2c + 22abc - 35b^2c + 61ac^2 - 40bc^2 + c^3 + 13a^2c - 95ab + 50b^2 - 44ac + 66bc + 6c^2 + 63a + 23b + 127c, 21b^2c - 6 - 4a^3 + 13ac^2b + 48ab^2 - 10b^3 + 8a^2c - 5abc - 14b^2c + 3ac^2 - 10bc^2 + 2c^3 - 30a^2b + 27ab - 40b^2 + 10ac - 57bc - 2c^3 - 7a - 10b - 40c, 6ab^2c - 3 - a^3 - 5a^2b + 12ab^2 - 11ac^2 - 2abc + 7b^2c - 9ac^2 + 2bc^2 - c^3 - 3a^2 + 15ab - 4b^2 + 4ac - 6bc - 2c^2 - 13a - b - 19c, 14a^2bc - 13 + 3a^3 + 13a^2b + 6ab^2 - 3b^3 - a^2c + 2abc + 21b^2c - 25ac^2 + 4bc^2 - 5c^3 + 19a^2 + 27ab - 26b^2 + 10ac - 8bc - 2c^2 - 7a - 17b - 33c, 7a^3c + 3 - 5a^3 - 24a^2b + 25ab^2 + 2b^2 - 17a^2 - 8abc + 7b^2c - 19ac^2 + 12bc^2 - c^3 - 6a^2 + 32ab - 8b^2 + 23ac - 17b - 6c^2 - 21a - 2b - 36c, 42b^3 - 3 - 121a^3 - 557a^2b + 570ab^2 + 275b^3 - 515a^2c - 608abc - 77b^2c - 555ac^2 + 2bc^2 - 55c^3 - 645a^2b + 635ab - 16b^2 + 82ac - 690c - 302c^2 - 679a + 65b - 1147c, 42ab^3 + 15 - 11a^3 - 85a^2b + 6ab^2 + 25bc^3 - 43a^2c - 40abc - 49b^2c - 39ac^2 + 4bc^2 + 5c^3 - 51a^2b + 58b^2 - 4ac + 6bc - 16c^2 - 35a + 25b + 5c, 21a^2b^2 + 3 + 2a^3 + 25a^2b - 45ab^2 + 5b^3 + 25a^2c - 29abc - 14b^2c + 9abc - 16b^2c - c^3 - 6a^2 - 45ab + 20b^2 + 47ac + 18bc + c^2 + 14a + 5b + 41c, 21a^3b + 18 - 16a^2c - 74a^2b + 24ab^2 + 2b^3 - 53a^2c + abc - 35bc^2c + 12ac^2 + 23bc^2 + 8c^3 - 16a^2c + 29ab + 11ac + 3bc - 8c^2 - 28a + 23b - 13c, 42a^4 + 57c^3 + 431a^2 + 757a^2b - 804ab^2 + 59b^3 + 799a^2c - 2abc - 119b^2c + 417ac^3 - 340bc^2 + 5c^3 + 303a^2b - 1203ab + 194b^2 - 752ac + 246bc + 184c^2 + 581a - 67b + 1013c).

The equivalent lexicographic base.
\{1 - 6c - 41c^2 + 71c^3 - 41c^{18} + 197c^{14} + 106c^{16} - 6c^{19} + 106c^4 + 71c^{17} + c^{20} + 92c^3 + 197c^6 + 257c^8 + 278c^9 + 201c^{10} + 278c^{11} + 257c^{12} + 145c^{13} + 92c^{15}, 9741532 + 39671190c - 96977172c^2 - 140671876c^3 + 7007106c^{18} - 120781728c^{14} - 79900378c^{16} - 1184459c^{19} - 131742078c^7 + 49142070c^7 - 253979379c^9 - 204237390c^9 - 33750520c^9 - 356354619c^9 - 271667666c^9 - 358606781c^{11} - 32381024c^{11} - 193581378c^{12} - 244307826c^{13} - 131861787c^{15} + 1665371b - 487915c^{18} + 705159a + 4406102c + 16292173c^{14} + 17206178c^9 + 3830961c^{16} + 91729c^{19} - 377534 + 21640797c^3 + 26686318c^{14} - 4114333c^{17} + 34893715c^5 + 37340389c^6 + 47961810c^7 + 46227230c^8 + 42839310c^9 + 46349985c^{10} + 37678046c^{11} + 28185846c^{12} + 26536060c^{13} + 13243117c^{15}\}.