CoCoA 5.1.2 Manual

August 19, 2015
Contents

I  Alphabetical List of Commands  21
I-0  Special Characters  23
   I-0.1  operators, shortcuts  ........................................... 23
I-1  A  25
   I-1.1  abs  ................................................................. 25
   I-1.2  adj  ................................................................. 25
   I-1.3  AffHilbert [OBSOLESCENT]  ................................. 26
   I-1.4  AffHilbertFu  .................................................. 26
   I-1.5  AffHilbertSeries  ............................................. 26
   I-1.6  AffPoincare [OBSOLESCENT]  ......................... 27
   I-1.7  alias  .............................................................. 27
   I-1.8  aliases  ............................................................ 28
   I-1.9  AlmostQR  ......................................................... 28
   I-1.10 and  ............................................................... 29
   I-1.11 append  .............................................................. 29
   I-1.12 apply  .............................................................. 30
   I-1.13 ApproxSolve  .................................................. 30
   I-1.14 ascii  .............................................................. 31
   I-1.15 AsINT  ............................................................ 31
   I-1.16 AsRAT  ............................................................. 32
I-2  B  33
   I-2.1  BaseRing  .......................................................... 33
   I-2.2  BBasis5  .......................................................... 33
   I-2.3  BettiDiagram  .................................................. 34
   I-2.4  BettiMatrix  .................................................... 34
   I-2.5  binomial  .......................................................... 35
   I-2.6  BinomialRepr, BinomialReprShift  .................... 35
   I-2.7  block  ............................................................... 36
   I-2.8  BlockMat  .......................................................... 37
   I-2.9  BlockMat2x2  .................................................... 37
   I-2.10  Bool01  ............................................................ 38
   I-2.11  break  ............................................................... 38
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I-2.12</td>
<td>BringIn</td>
</tr>
<tr>
<td>I-3 C</td>
<td>Call [OBSCOLETE]</td>
</tr>
<tr>
<td>I-3.1</td>
<td>CanonicalHom</td>
</tr>
<tr>
<td>I-3.2</td>
<td>CanonicalRepr</td>
</tr>
<tr>
<td>I-3.3</td>
<td>CartesianProduct, CartesianProductList</td>
</tr>
<tr>
<td>I-3.4</td>
<td>Cast [OBSCOLETE]</td>
</tr>
<tr>
<td>I-3.5</td>
<td>ceil</td>
</tr>
<tr>
<td>I-3.6</td>
<td>CFAprox</td>
</tr>
<tr>
<td>I-3.7</td>
<td>CFAproximants</td>
</tr>
<tr>
<td>I-3.8</td>
<td>characteristic</td>
</tr>
<tr>
<td>I-3.9</td>
<td>CharPoly</td>
</tr>
<tr>
<td>I-3.10</td>
<td>CheckArgTypes</td>
</tr>
<tr>
<td>I-3.11</td>
<td>ciao</td>
</tr>
<tr>
<td>I-3.12</td>
<td>ClearDenom</td>
</tr>
<tr>
<td>I-3.13</td>
<td>close</td>
</tr>
<tr>
<td>I-3.14</td>
<td>CloseLog</td>
</tr>
<tr>
<td>I-3.15</td>
<td>CoCoA-4 mode</td>
</tr>
<tr>
<td>I-3.16</td>
<td>CocoaLimits</td>
</tr>
<tr>
<td>I-3.17</td>
<td>CocoaPackagePath</td>
</tr>
<tr>
<td>I-3.18</td>
<td>codomain</td>
</tr>
<tr>
<td>I-3.19</td>
<td>CoeffEmbeddingHom</td>
</tr>
<tr>
<td>I-3.20</td>
<td>coefficients</td>
</tr>
<tr>
<td>I-3.21</td>
<td>CoefficientsWRT</td>
</tr>
<tr>
<td>I-3.22</td>
<td>CoeffListWRT</td>
</tr>
<tr>
<td>I-3.23</td>
<td>CoefOfTerm</td>
</tr>
<tr>
<td>I-3.24</td>
<td>CoeffRing</td>
</tr>
<tr>
<td>I-3.25</td>
<td>ColMat</td>
</tr>
<tr>
<td>I-3.26</td>
<td>colon</td>
</tr>
<tr>
<td>I-3.27</td>
<td>ColumnVectors [OBSCOLETE]</td>
</tr>
<tr>
<td>I-3.28</td>
<td>Comp [OBSCOLETE]</td>
</tr>
<tr>
<td>I-3.29</td>
<td>Comparison Operators</td>
</tr>
<tr>
<td>I-3.30</td>
<td>CompleteToOrd</td>
</tr>
<tr>
<td>I-3.31</td>
<td>compts</td>
</tr>
<tr>
<td>I-3.32</td>
<td>CONCATAntiDiag</td>
</tr>
<tr>
<td>I-3.33</td>
<td>CONCATDiag</td>
</tr>
<tr>
<td>I-3.34</td>
<td>CONCATHor</td>
</tr>
<tr>
<td>I-3.35</td>
<td>CONCATHorList</td>
</tr>
<tr>
<td>I-3.36</td>
<td>CONCATLists</td>
</tr>
<tr>
<td>I-3.37</td>
<td>CONCATVer</td>
</tr>
<tr>
<td>I-3.38</td>
<td>CONCATVerList</td>
</tr>
<tr>
<td>I-3.41</td>
<td>content .................................................................</td>
</tr>
<tr>
<td>I-3.42</td>
<td>ContentFreeFactor ....................................................</td>
</tr>
<tr>
<td>I-3.43</td>
<td>ContentWRT ....................................................................</td>
</tr>
<tr>
<td>I-3.44</td>
<td>ContFrac .........................................................................</td>
</tr>
<tr>
<td>I-3.45</td>
<td>ContFracToRat ..................................................................</td>
</tr>
<tr>
<td>I-3.46</td>
<td>count ...............................................................................</td>
</tr>
<tr>
<td>I-3.47</td>
<td>CpuTime ...........................................................................</td>
</tr>
<tr>
<td>I-3.48</td>
<td>CRT ..................................................................................</td>
</tr>
<tr>
<td>I-3.49</td>
<td>CurrentRing ......................................................................</td>
</tr>
<tr>
<td>I-3.50</td>
<td>CurrentTypes ....................................................................</td>
</tr>
<tr>
<td>I-3.51</td>
<td>cyclotomic ........................................................................</td>
</tr>
<tr>
<td>I-4</td>
<td>D .....................................................................................</td>
</tr>
<tr>
<td>I-4.1</td>
<td>dashes ...............................................................................</td>
</tr>
<tr>
<td>I-4.2</td>
<td>date ..................................................................................</td>
</tr>
<tr>
<td>I-4.3</td>
<td>DecimalStr .......................................................................</td>
</tr>
<tr>
<td>I-4.4</td>
<td>define ...............................................................................</td>
</tr>
<tr>
<td>I-4.5</td>
<td>DefiningIdeal .....................................................................</td>
</tr>
<tr>
<td>I-4.6</td>
<td>deg ....................................................................................</td>
</tr>
<tr>
<td>I-4.7</td>
<td>den .....................................................................................</td>
</tr>
<tr>
<td>I-4.8</td>
<td>DensePoly ..........................................................................</td>
</tr>
<tr>
<td>I-4.9</td>
<td>Depth .................................................................................</td>
</tr>
<tr>
<td>I-4.10</td>
<td>deriv ..................................................................................</td>
</tr>
<tr>
<td>I-4.11</td>
<td>DerivationAction ................................................................</td>
</tr>
<tr>
<td>I-4.12</td>
<td>describe ............................................................................</td>
</tr>
<tr>
<td>I-4.13</td>
<td>det ......................................................................................</td>
</tr>
<tr>
<td>I-4.14</td>
<td>DF ......................................................................................</td>
</tr>
<tr>
<td>I-4.15</td>
<td>DiagMat .............................................................................</td>
</tr>
<tr>
<td>I-4.16</td>
<td>diff ....................................................................................</td>
</tr>
<tr>
<td>I-4.17</td>
<td>dim ......................................................................................</td>
</tr>
<tr>
<td>I-4.18</td>
<td>discriminant ......................................................................</td>
</tr>
<tr>
<td>I-4.19</td>
<td>distrib ...............................................................................</td>
</tr>
<tr>
<td>I-4.20</td>
<td>div ......................................................................................</td>
</tr>
<tr>
<td>I-4.21</td>
<td>DivAlg ...............................................................................</td>
</tr>
<tr>
<td>I-4.22</td>
<td>domain ...............................................................................</td>
</tr>
<tr>
<td>I-5</td>
<td>E .......................................................................................</td>
</tr>
<tr>
<td>I-5.1</td>
<td>E_ [OBSOLETE] ...................................................................</td>
</tr>
<tr>
<td>I-5.2</td>
<td>eigenvectors .....................................................................</td>
</tr>
<tr>
<td>I-5.3</td>
<td>elim .....................................................................................</td>
</tr>
<tr>
<td>I-5.4</td>
<td>ElimMat ..............................................................................</td>
</tr>
<tr>
<td>I-5.5</td>
<td>EmbeddingHom ....................................................................</td>
</tr>
<tr>
<td>I-5.6</td>
<td>EqSet ..................................................................................</td>
</tr>
</tbody>
</table>
I-5.7 Equality Test ......................................................... 78
I-5.8 EquiIsoDec ......................................................... 78
I-5.9 error ................................................................. 79
I-5.10 eval ................................................................. 79
I-5.11 EvalHilbertFn ..................................................... 80
I-5.12 Ext ................................................................. 80
I-5.13 ExternalLibs ...................................................... 82

I-6 F 83
I-6.1 factor ............................................................... 83
I-6.2 factorial ............................................................ 83
I-6.3 FactorMultiplicity ................................................ 84
I-6.4 FGLM5 ............................................................... 84
I-6.5 fields ................................................................. 85
I-6.6 first ................................................................. 85
I-6.7 FirstNonZero ...................................................... 85
I-6.8 FirstNonZeroPosn ................................................ 86
I-6.9 flatten .............................................................. 86
I-6.10 FloatApprox ...................................................... 87
I-6.11 FloatStr ........................................................... 87
I-6.12 floor ............................................................... 88
I-6.13 for ................................................................. 88
I-6.14 foreach ............................................................ 89
I-6.15 format .............................................................. 90
I-6.16 FrbAlexanderDual .............................................. 90
I-6.17 FrbAssociatedPrimes ......................................... 91
I-6.18 FrbIrreducibleDecomposition ................................. 91
I-6.19 FrbMaximalStandardMonomials .............................. 92
I-6.20 FrbPrimaryDecomposition .................................... 92
I-6.21 func ............................................................... 92
I-6.22 Function [OBsolete] ............................................. 93
I-6.23 functions [OBsolete] ........................................... 93
I-6.24 FVector .......................................................... 93

I-7 G 95
I-7.1 GBasis .............................................................. 95
I-7.2 GBasisTimeout .................................................... 95
I-7.3 GBM ................................................................. 96
I-7.4 gcd ................................................................. 96
I-7.5 GCDFreeBasis ..................................................... 97
I-7.6 GenericPoints .................................................... 97
I-7.7 GenRepr ........................................................... 98
I-7.8 gens ............................................................... 98
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I-7.9</td>
<td>GensAsCols, GensAsRows</td>
<td>99</td>
</tr>
<tr>
<td>I-7.10</td>
<td>Get</td>
<td>100</td>
</tr>
<tr>
<td>I-7.11</td>
<td>GetCol</td>
<td>100</td>
</tr>
<tr>
<td>I-7.12</td>
<td>GetCols</td>
<td>101</td>
</tr>
<tr>
<td>I-7.13</td>
<td>GetEnv</td>
<td>101</td>
</tr>
<tr>
<td>I-7.14</td>
<td>GetErrMsg</td>
<td>101</td>
</tr>
<tr>
<td>I-7.15</td>
<td>GetRow</td>
<td>102</td>
</tr>
<tr>
<td>I-7.16</td>
<td>GetRows</td>
<td>102</td>
</tr>
<tr>
<td>I-7.17</td>
<td>gin</td>
<td>102</td>
</tr>
<tr>
<td>I-7.18</td>
<td>GradingDim</td>
<td>103</td>
</tr>
<tr>
<td>I-7.19</td>
<td>GradingMat</td>
<td>103</td>
</tr>
<tr>
<td>I-8</td>
<td>H</td>
<td>105</td>
</tr>
<tr>
<td>I-8.1</td>
<td>HColon</td>
<td>105</td>
</tr>
<tr>
<td>I-8.2</td>
<td>HGBM</td>
<td>105</td>
</tr>
<tr>
<td>I-8.3</td>
<td>hilbert [OBSOLETE]</td>
<td>106</td>
</tr>
<tr>
<td>I-8.4</td>
<td>HilbertBasisKer</td>
<td>106</td>
</tr>
<tr>
<td>I-8.5</td>
<td>HilbertFn</td>
<td>107</td>
</tr>
<tr>
<td>I-8.6</td>
<td>HilbertPoly</td>
<td>107</td>
</tr>
<tr>
<td>I-8.7</td>
<td>HilbertSeries</td>
<td>108</td>
</tr>
<tr>
<td>I-8.8</td>
<td>HilbertSeriesMultiDeg</td>
<td>109</td>
</tr>
<tr>
<td>I-8.9</td>
<td>HilbertSeriesShifts</td>
<td>110</td>
</tr>
<tr>
<td>I-8.10</td>
<td>homog</td>
<td>110</td>
</tr>
<tr>
<td>I-8.11</td>
<td>HomogElimMat</td>
<td>111</td>
</tr>
<tr>
<td>I-8.12</td>
<td>HSaturation</td>
<td>111</td>
</tr>
<tr>
<td>I-8.13</td>
<td>HVector</td>
<td>112</td>
</tr>
<tr>
<td>I-9</td>
<td>I</td>
<td>113</td>
</tr>
<tr>
<td>I-9.1</td>
<td>ID [OBSOLET]</td>
<td>113</td>
</tr>
<tr>
<td>I-9.2</td>
<td>ideal</td>
<td>113</td>
</tr>
<tr>
<td>I-9.3</td>
<td>IdealAndSeparatorsOfPoints</td>
<td>114</td>
</tr>
<tr>
<td>I-9.4</td>
<td>IdealAndSeparatorsOfProjectivePoints</td>
<td>115</td>
</tr>
<tr>
<td>I-9.5</td>
<td>IdealOfPoints</td>
<td>116</td>
</tr>
<tr>
<td>I-9.6</td>
<td>IdealOfProjectivePoints</td>
<td>117</td>
</tr>
<tr>
<td>I-9.7</td>
<td>IdentityMat</td>
<td>118</td>
</tr>
<tr>
<td>I-9.8</td>
<td>if</td>
<td>118</td>
</tr>
<tr>
<td>I-9.9</td>
<td>ILogBase</td>
<td>119</td>
</tr>
<tr>
<td>I-9.10</td>
<td>image [OBSOLET]</td>
<td>119</td>
</tr>
<tr>
<td>I-9.11</td>
<td>implicit</td>
<td>120</td>
</tr>
<tr>
<td>I-9.12</td>
<td>ImplicitPlot</td>
<td>120</td>
</tr>
<tr>
<td>I-9.13</td>
<td>ImplicitPlotOn</td>
<td>121</td>
</tr>
<tr>
<td>I-9.14</td>
<td>ImportByRef, ImportByValue</td>
<td>121</td>
</tr>
<tr>
<td>I-9.15</td>
<td>in</td>
<td>122</td>
</tr>
<tr>
<td>I-9.16</td>
<td>incr, decr</td>
<td>122</td>
</tr>
<tr>
<td>I-9.17</td>
<td>indent</td>
<td>123</td>
</tr>
<tr>
<td>I-9.18</td>
<td>indet</td>
<td>123</td>
</tr>
<tr>
<td>I-9.19</td>
<td>IndetIndex</td>
<td>124</td>
</tr>
<tr>
<td>I-9.20</td>
<td>IndetName</td>
<td>124</td>
</tr>
<tr>
<td>I-9.21</td>
<td>indets</td>
<td>125</td>
</tr>
<tr>
<td>I-9.22</td>
<td>IndetSubscripts</td>
<td>126</td>
</tr>
<tr>
<td>I-9.23</td>
<td>IndetSymbols</td>
<td>126</td>
</tr>
<tr>
<td>I-9.24</td>
<td>InducedHom</td>
<td>127</td>
</tr>
<tr>
<td>I-9.25</td>
<td>InitialIdeal</td>
<td>127</td>
</tr>
<tr>
<td>I-9.26</td>
<td>insert [OBSOLESCENT]</td>
<td>128</td>
</tr>
<tr>
<td>I-9.27</td>
<td>Interpolate</td>
<td>128</td>
</tr>
<tr>
<td>I-9.28</td>
<td>interreduce, interreduced</td>
<td>129</td>
</tr>
<tr>
<td>I-9.29</td>
<td>intersection</td>
<td>129</td>
</tr>
<tr>
<td>I-9.30</td>
<td>IntersectList</td>
<td>130</td>
</tr>
<tr>
<td>I-9.31</td>
<td>inverse</td>
<td>130</td>
</tr>
<tr>
<td>I-9.32</td>
<td>InverseSystem</td>
<td>131</td>
</tr>
<tr>
<td>I-9.33</td>
<td>IO.SprintTrunc</td>
<td>131</td>
</tr>
<tr>
<td>I-9.34</td>
<td>iroot</td>
<td>131</td>
</tr>
<tr>
<td>I-9.35</td>
<td>IsAntiSymmetric</td>
<td>132</td>
</tr>
<tr>
<td>I-9.36</td>
<td>IsConstant</td>
<td>132</td>
</tr>
<tr>
<td>I-9.37</td>
<td>IsContained</td>
<td>132</td>
</tr>
<tr>
<td>I-9.38</td>
<td>IsDefined</td>
<td>133</td>
</tr>
<tr>
<td>I-9.39</td>
<td>IsDiagonal</td>
<td>133</td>
</tr>
<tr>
<td>I-9.40</td>
<td>IsDivisible</td>
<td>133</td>
</tr>
<tr>
<td>I-9.41</td>
<td>IsElem</td>
<td>134</td>
</tr>
<tr>
<td>I-9.42</td>
<td>IsEven, IsOdd</td>
<td>134</td>
</tr>
<tr>
<td>I-9.43</td>
<td>IsFactorClosed</td>
<td>134</td>
</tr>
<tr>
<td>I-9.44</td>
<td>IsField</td>
<td>135</td>
</tr>
<tr>
<td>I-9.45</td>
<td>IsFiniteField</td>
<td>135</td>
</tr>
<tr>
<td>I-9.46</td>
<td>IsHomog</td>
<td>136</td>
</tr>
<tr>
<td>I-9.47</td>
<td>IsIn</td>
<td>136</td>
</tr>
<tr>
<td>I-9.48</td>
<td>IsIndet</td>
<td>137</td>
</tr>
<tr>
<td>I-9.49</td>
<td>IsInjective</td>
<td>137</td>
</tr>
<tr>
<td>I-9.50</td>
<td>IsInRadical</td>
<td>138</td>
</tr>
<tr>
<td>I-9.51</td>
<td>IsInSubalgebra [OBSOLETE]</td>
<td>138</td>
</tr>
<tr>
<td>I-9.52</td>
<td>IsLexSegment</td>
<td>138</td>
</tr>
<tr>
<td>I-9.53</td>
<td>IsNumber [OBSOLETE]</td>
<td>139</td>
</tr>
<tr>
<td>I-9.54</td>
<td>IsOne</td>
<td>139</td>
</tr>
<tr>
<td>I-9.55</td>
<td>IsPositiveGrading</td>
<td>139</td>
</tr>
<tr>
<td>I-9.56</td>
<td>IsPrime</td>
<td>140</td>
</tr>
<tr>
<td>I-9.57</td>
<td>IsProbPrime</td>
<td>140</td>
</tr>
<tr>
<td>I-9.58</td>
<td>IsPthPower</td>
<td>141</td>
</tr>
</tbody>
</table>
I-9.59  IsQQ ................................................................. 141
I-9.60  isqrt ................................................................. 141
I-9.61  IsQuotientRing .................................................. 142
I-9.62  IsStable ............................................................. 142
I-9.63  IsStdGraded ....................................................... 142
I-9.64  IsStronglyStable .................................................. 143
I-9.65  IsSubset ............................................................. 143
I-9.66  IsSurjective ........................................................ 144
I-9.67  IsSymmetric ....................................................... 144
I-9.68  IsTerm ............................................................... 144
I-9.69  IsTermOrdering .................................................... 145
I-9.70  IsTree5 .............................................................. 145
I-9.71  IsTrueGCDDomain ................................................ 146
I-9.72  IsZero ............................................................... 146
I-9.73  IsZeroCol, IsZeroRow ......................................... 147
I-9.74  IsZeroDim .......................................................... 147
I-9.75  IsZeroDivisor ..................................................... 147
I-9.76  IsZZ ................................................................. 148
I-9.77  It .................................................................... 148

I-10  J ................................................................. 151
I-10.1  jacobian ............................................................ 151
I-10.2  JanetBasis .......................................................... 151

I-11  K ................................................................. 153
I-11.1  ker ................................................................. 153

I-12  L ................................................................. 155
I-12.1  last ................................................................. 155
I-12.2  LaTeX .............................................................. 155
I-12.3  LC ................................................................. 156
I-12.4  lcm ................................................................. 156
I-12.5  len ................................................................. 157
I-12.6  LexMat ............................................................. 157
I-12.7  LexSegmentIdeal ................................................. 158
I-12.8  LF ................................................................. 158
I-12.9  LinearSimplify ................................................... 159
I-12.10 LinKer ............................................................ 159
I-12.11 LinKerBasis ....................................................... 160
I-12.12 LinKerModP [OBSOLETE] ..................................... 160
I-12.13 LinSol [OBSOLETE] ............................................. 161
I-12.14 LinSolve .......................................................... 161
I-12.15 LM ................................................................. 162
I-12.16 log ................................................................. 162
I-12.17 LogCardinality .................................................... 162
I-12.18 LogToTerm ...................................................... 163
I-12.19 LPosn ............................................................ 163
I-12.20 LPP .............................................................. 164
I-12.21 LT ................................................................. 164

I-13 M ........................................................................ 167
I-13.1 MakeCheck .......................................................... 167
I-13.2 MakeMatByRows, MakeMatByCols ............................. 167
I-13.3 MakeSet ............................................................ 168
I-13.4 MantissaAndExponent10 ......................................... 168
I-13.5 MantissaAndExponent2 .......................................... 168
I-13.6 Manual .............................................................. 169
I-13.7 MapDown [OBSOLETE] .......................................... 170
I-13.8 matrix ............................................................... 170
I-13.9 max ................................................................. 171
I-13.10 MayerVietorisTreeN1 ........................................... 171
I-13.11 min ................................................................. 171
I-13.12 MinGens .......................................................... 172
I-13.13 MinGensGeneral [OBSOLESCENT] ......................... 172
I-13.14 minimalize ......................................................... 173
I-13.15 minimalized ....................................................... 173
I-13.16 MinimalPresentation ........................................... 173
I-13.17 minors ............................................................ 174
I-13.18 MinPoly .......................................................... 174
I-13.19 MinPowerInIdeal ................................................ 175
I-13.20 MinSubsetOfGens ................................................ 175
I-13.21 mod ............................................................... 175
I-13.22 Mod2Rat [OBSOLETE] ......................................... 176
I-13.23 ModuleElem ...................................................... 176
I-13.24 ModuleOf ........................................................ 176
I-13.25 monic ............................................................. 177
I-13.26 monomials ......................................................... 177
I-13.27 MonsInIdeal ...................................................... 178
I-13.28 multiplicity ...................................................... 178

I-14 N ........................................................................ 179
I-14.1 NewFractionField .................................................. 179
I-14.2 NewFreeModule .................................................... 179
I-14.3 NewId [OBSOLETE] .............................................. 180
I-14.4 NewLine [OBSOLESCENT] ................................. 180
I-14.5 NewList ........................................................... 180
I-14.6 NewMat ............................................................ 181
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I-16.1</td>
<td>Packages</td>
<td>201</td>
</tr>
<tr>
<td>I-16.2</td>
<td>panel [OBSOLETE]</td>
<td>201</td>
</tr>
<tr>
<td>I-16.3</td>
<td>panels [OBSOLETE]</td>
<td>201</td>
</tr>
<tr>
<td>I-16.4</td>
<td>partitions</td>
<td>202</td>
</tr>
<tr>
<td>I-16.5</td>
<td>permutations</td>
<td>202</td>
</tr>
<tr>
<td>I-16.6</td>
<td>PerpIdealOfForm</td>
<td>202</td>
</tr>
<tr>
<td>I-16.7</td>
<td>pfaffian</td>
<td>203</td>
</tr>
<tr>
<td>I-16.8</td>
<td>PkgName</td>
<td>203</td>
</tr>
<tr>
<td>I-16.9</td>
<td>PlotPoints</td>
<td>204</td>
</tr>
<tr>
<td>I-16.10</td>
<td>PlotPointsOn</td>
<td>204</td>
</tr>
<tr>
<td>I-16.11</td>
<td>poincare [OBSOLESCENT]</td>
<td>205</td>
</tr>
<tr>
<td>I-16.12</td>
<td>PoincareMultiDeg [OBSOLETE]</td>
<td>205</td>
</tr>
<tr>
<td>I-16.13</td>
<td>PoincareShifts [OBSOLETE]</td>
<td>205</td>
</tr>
<tr>
<td>I-16.14</td>
<td>PolyAlgebraHom</td>
<td>205</td>
</tr>
<tr>
<td>I-16.15</td>
<td>PolyRingHom</td>
<td>206</td>
</tr>
<tr>
<td>I-16.16</td>
<td>PowerMod</td>
<td>206</td>
</tr>
<tr>
<td>I-16.17</td>
<td>PreImage</td>
<td>207</td>
</tr>
<tr>
<td>I-16.18</td>
<td>PreprocessPts</td>
<td>207</td>
</tr>
<tr>
<td>I-16.19</td>
<td>PrimaryDecomposition</td>
<td>208</td>
</tr>
<tr>
<td>I-16.20</td>
<td>PrimaryDecomposition0</td>
<td>209</td>
</tr>
<tr>
<td>I-16.21</td>
<td>PrimaryDecompositionGTZ0</td>
<td>209</td>
</tr>
<tr>
<td>I-16.22</td>
<td>PrimaryHilbertSeries</td>
<td>209</td>
</tr>
<tr>
<td>I-16.23</td>
<td>PrimaryPoincare [OBSOLESCENT]</td>
<td>211</td>
</tr>
<tr>
<td>I-16.24</td>
<td>PrimitiveRoot</td>
<td>211</td>
</tr>
<tr>
<td>I-16.25</td>
<td>print</td>
<td>211</td>
</tr>
<tr>
<td>I-16.26</td>
<td>print on</td>
<td>212</td>
</tr>
<tr>
<td>I-16.27</td>
<td>PrintBettiDiagram</td>
<td>212</td>
</tr>
<tr>
<td>I-16.28</td>
<td>PrintBettiMatrix</td>
<td>213</td>
</tr>
<tr>
<td>I-16.29</td>
<td>println</td>
<td>213</td>
</tr>
<tr>
<td>I-16.30</td>
<td>PrintRes</td>
<td>213</td>
</tr>
<tr>
<td>I-16.31</td>
<td>product</td>
<td>214</td>
</tr>
<tr>
<td>I-16.32</td>
<td>protect</td>
<td>214</td>
</tr>
<tr>
<td>I-16.33</td>
<td>PthRoot</td>
<td>215</td>
</tr>
<tr>
<td>I-17</td>
<td>Q</td>
<td>217</td>
</tr>
<tr>
<td>I-17.1</td>
<td>QQ</td>
<td>217</td>
</tr>
<tr>
<td>I-17.2</td>
<td>quit</td>
<td>217</td>
</tr>
<tr>
<td>I-17.3</td>
<td>QuotientBasis</td>
<td>218</td>
</tr>
<tr>
<td>I-17.4</td>
<td>QZP</td>
<td>218</td>
</tr>
<tr>
<td>I-18</td>
<td>R</td>
<td>221</td>
</tr>
<tr>
<td>I-18.1</td>
<td>radical</td>
<td>221</td>
</tr>
<tr>
<td>I-18.2</td>
<td>RadicalOfUnmixed</td>
<td>221</td>
</tr>
</tbody>
</table>
I-18.46 RingZZ ................................................................. 242
I-18.47 rk ................................................................. 242
I-18.48 RMap [OBSOLESCENT] ........................................ 243
I-18.49 RootBound ...................................................... 243
I-18.50 round .......................................................... 243
I-18.51 RowMat .......................................................... 243

I-19 S ................................................................. 245
I-19.1 saturate .......................................................... 245
I-19.2 ScalarProduct ................................................ 245
I-19.3 ScientificStr .................................................. 246
I-19.4 seed ............................................................ 246
I-19.5 SeparatorsOfPoints .......................................... 247
I-19.6 SeparatorsOfProjectivePoints .............................. 248
I-19.7 SetRow .......................................................... 249
I-19.8 SetStackSize ................................................... 249
I-19.9 shape ............................................................ 249
I-19.10 sign ............................................................. 250
I-19.11 SimplestRatBetween ....................................... 251
I-19.12 SimplexInfo .................................................. 251
I-19.13 SimplicialHomology ........................................ 251
I-19.14 size [OBSOLETE] ............................................. 252
I-19.15 skip ............................................................ 252
I-19.16 SmoothFactor ................................................ 253
I-19.17 sort ............................................................ 253
I-19.18 SortBy .......................................................... 254
I-19.19 sorted .......................................................... 254
I-19.20 SortedBy ....................................................... 255
I-19.21 source .......................................................... 256
I-19.22 SourceRegion ................................................ 256
I-19.23 spaces .......................................................... 256
I-19.24 sprint ........................................................... 257
I-19.25 SqFreeFactor .................................................. 257
I-19.26 StableBBasis5 ............................................... 258
I-19.27 StableIdeal .................................................... 258
I-19.28 StarPrint, StarSprint ...................................... 259
I-19.29 starting ......................................................... 260
I-19.30 StdDegLexMat ............................................... 260
I-19.31 StdDegRevLexMat ......................................... 260
I-19.32 StronglyStableIdeal ....................................... 261
I-19.33 SubalgebraMap [OBSOLETE] .............................. 261
I-19.34 SubalgebraRepr ............................................. 261
I-19.35 submat ........................................................ 262
II-7 CoCoA Packages

II-7.1 Introduction to Packages .................................................. 313
II-7.2 First Example of a Package ................................................. 313
II-7.3 Package Essentials .......................................................... 314
II-7.4 Global Aliases ..................................................................... 314
II-7.5 Sharing Your Package .......................................................... 314
II-7.6 Commands and Functions for Packages ................................. 314
II-7.7 Supported Packages ............................................................ 315
II-7.8 Galois Package .................................................................... 315
II-7.9 Integer Programming ........................................................... 315
II-7.10 Algebra of Invariants .......................................................... 315
II-7.11 Special Varieties ................................................................. 315
II-7.12 Statistics ............................................................................. 316
II-7.13 Geometrical Theorem-Proving ............................................. 316
II-7.14 Typevectors ....................................................................... 316
II-7.15 Conductor ........................................................................... 316
II-7.16 Matrix Normal Form ........................................................... 317
II-7.17 CantStop ............................................................................. 317
II-7.18 Control ............................................................................... 317

II-8 Linked libraries ........................................................................ 319

II-8.1 CoCoALib ............................................................................ 319
II-8.2 GMP .................................................................................... 319
II-8.3 GSL ...................................................................................... 319
II-8.4 Frobby ................................................................................. 319
II-8.5 Normaliz ................................................................................ 319

II-9 Migrating from CoCoA-4 and keeping up-to-date ..................... 321

II-9.1 Changes in the CoCoA language ............................................ 321
II-9.2 Recent changes in the CoCoA-5 language ............................... 322
II-9.3 Obsolete and obsolescent functions ...................................... 322

III CoCoA datatypes ........................................................................ 325

III-1 BOOL ...................................................................................... 327

III-1.1 Introduction to BOOL ............................................................ 327
III-1.2 Commands and Functions for BOOL ................................. 327
III-1.3 Commands and Functions returning BOOL ....................... 327

III-2 INT ......................................................................................... 329

III-2.1 Introduction to INT ............................................................... 329
III-2.2 Commands and Functions for INT ..................................... 329
### Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>III-2.3</td>
<td>Commands and Functions returning INT</td>
<td>331</td>
</tr>
<tr>
<td>III-3</td>
<td>RAT</td>
<td></td>
</tr>
<tr>
<td>III-3.1</td>
<td>Introduction to RAT</td>
<td>333</td>
</tr>
<tr>
<td>III-3.2</td>
<td>Commands and Functions for RAT</td>
<td>333</td>
</tr>
<tr>
<td>III-3.3</td>
<td>Commands and Functions returning RAT</td>
<td>334</td>
</tr>
<tr>
<td>III-4</td>
<td>STRING</td>
<td></td>
</tr>
<tr>
<td>III-4.1</td>
<td>String Literals</td>
<td>335</td>
</tr>
<tr>
<td>III-4.2</td>
<td>String Operations</td>
<td>335</td>
</tr>
<tr>
<td>III-4.3</td>
<td>Commands and Functions for STRING</td>
<td>336</td>
</tr>
<tr>
<td>III-4.4</td>
<td>Commands and Functions returning STRING</td>
<td>336</td>
</tr>
<tr>
<td>III-5</td>
<td>LIST</td>
<td></td>
</tr>
<tr>
<td>III-5.1</td>
<td>Introduction to LIST</td>
<td>339</td>
</tr>
<tr>
<td>III-5.2</td>
<td>List Constructors</td>
<td>340</td>
</tr>
<tr>
<td>III-5.3</td>
<td>Commands and Functions for LIST</td>
<td>340</td>
</tr>
<tr>
<td>III-5.4</td>
<td>Commands and Functions returning LIST</td>
<td>342</td>
</tr>
<tr>
<td>III-6</td>
<td>RECORD</td>
<td></td>
</tr>
<tr>
<td>III-6.1</td>
<td>Introduction to RECORD</td>
<td>345</td>
</tr>
<tr>
<td>III-6.2</td>
<td>Commands and Functions for RECORD</td>
<td>346</td>
</tr>
<tr>
<td>III-6.3</td>
<td>Commands and Functions returning RECORD</td>
<td>346</td>
</tr>
<tr>
<td>III-7</td>
<td>FUNCTION</td>
<td></td>
</tr>
<tr>
<td>III-7.1</td>
<td>Introduction to FUNCTION</td>
<td>349</td>
</tr>
<tr>
<td>III-7.2</td>
<td>FUNCTIONs are first class objects</td>
<td>349</td>
</tr>
<tr>
<td>III-7.3</td>
<td>Commands and Functions for FUNCTION</td>
<td>349</td>
</tr>
<tr>
<td>III-7.4</td>
<td>Commands and Functions returning FUNCTION</td>
<td>350</td>
</tr>
<tr>
<td>III-8</td>
<td>TYPE</td>
<td></td>
</tr>
<tr>
<td>III-8.1</td>
<td>Commands and Functions for TYPE</td>
<td>351</td>
</tr>
<tr>
<td>III-8.2</td>
<td>Commands and Functions returning TYPE</td>
<td>351</td>
</tr>
<tr>
<td>III-9</td>
<td>RING</td>
<td></td>
</tr>
<tr>
<td>III-9.1</td>
<td>Introduction to RING</td>
<td>353</td>
</tr>
<tr>
<td>III-9.2</td>
<td>Polynomial Rings</td>
<td>353</td>
</tr>
<tr>
<td>III-9.3</td>
<td>Coefficient Rings</td>
<td>354</td>
</tr>
<tr>
<td>III-9.4</td>
<td>Indeterminates</td>
<td>354</td>
</tr>
<tr>
<td>III-9.5</td>
<td>Orderings</td>
<td>354</td>
</tr>
<tr>
<td>III-9.6</td>
<td>Module Orderings</td>
<td>355</td>
</tr>
<tr>
<td>III-9.7</td>
<td>Quotient Rings</td>
<td>355</td>
</tr>
<tr>
<td>III-9.8</td>
<td>Commands and Functions for RING</td>
<td>356</td>
</tr>
<tr>
<td>III-9.9</td>
<td>Commands and Functions returning RING</td>
<td>357</td>
</tr>
<tr>
<td>III-10</td>
<td>RINGHOM</td>
<td></td>
</tr>
<tr>
<td>III-10.1</td>
<td>Commands and Functions returning RINGHOM</td>
<td>359</td>
</tr>
</tbody>
</table>
Part I

Alphabetical List of Commands
Chapter I-0

Special Characters

I-0.1 operators, shortcuts

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>A := B</code></td>
<td><code>A:=variable, B:OBJECT</code> compute ( B ) then assign the result to ( A )</td>
</tr>
<tr>
<td><code>A = B</code></td>
<td>( A,B:OBJECT ) test whether ( A ) and ( B ) are equal</td>
</tr>
<tr>
<td><code>A &lt;&gt; B</code></td>
<td>( A,B:OBJECT ) test whether ( A ) and ( B ) are not equal</td>
</tr>
<tr>
<td><code>[...]</code></td>
<td>LIST build a new list (see “List Constructors” (III-5.2 pg.340))</td>
</tr>
<tr>
<td><code>[...[,]</code></td>
<td>LIST build a new list (see “List Constructors” (III-5.2 pg.340))</td>
</tr>
<tr>
<td><code>L[N]</code></td>
<td>( L: LIST ), ( N: INT ) access ( N )-th entry of list ( L )</td>
</tr>
<tr>
<td><code>A..B</code></td>
<td>( A,B: INT ) or ( A,B:\text{indeterminates} ) is the “Range Operator”</td>
</tr>
<tr>
<td><code>R.F</code></td>
<td>( R:\text{RECORD} ) and ( F ) field name</td>
</tr>
<tr>
<td><code>R ::= ...</code></td>
<td>( R:\text{variable} ) for the special ring syntax (“NewPolyRing” (I-14.8 pg.182))</td>
</tr>
<tr>
<td><code>A &gt;&lt; B</code></td>
<td>equivalent to ( \text{CartesianProduct}(A, B) ) ( A,B:LIST )</td>
</tr>
<tr>
<td><code>M : N</code></td>
<td>( M, N: \text{MODULE} ) or ( IDEAL )</td>
</tr>
<tr>
<td><code>R/I</code></td>
<td>( R:\text{RING} ), ( I: IDEAL )</td>
</tr>
<tr>
<td><code>S[N]</code></td>
<td>( S: STRING ), ( N: INT ) access ( N )-th char of string ( S )</td>
</tr>
<tr>
<td><code>***E***</code></td>
<td>( E:\text{expression} ) interpret ( E ) in “CoCoA-4 mode” (I-3.16 pg.46)</td>
</tr>
<tr>
<td><code>?S</code></td>
<td>( S: STRING ) prints the manual page for ( S ) or a list of matching</td>
</tr>
<tr>
<td><code>&lt;S</code></td>
<td>( S: STRING ) manual pages</td>
</tr>
</tbody>
</table>

See Also: colon(I-3.27 pg.52), List Constructors(III-5.2 pg.340), CartesianProduct, CartesianProductList(I-3.4 pg.42), NewPolyRing(I-14.8 pg.182), NewQuotientRing(I-14.9 pg.182), record field selector(I-18.21 pg.230), Range Operator(II-3.5 pg.302), source(I-19.21 pg.256), Manual(I-13.6 pg.169), Character Set and Special Symbols(II-2.1 pg.299), CoCoA Operators(II-3.1 pg.301)
Chapter I-1

A

I-1.1 abs

Syntax

\[
\begin{align*}
\text{abs}(N: \text{INT}): \text{INT} \\
\text{abs}(N: \text{RAT}): \text{RAT} \\
\text{abs}(N: \text{RINGELEM}): \text{RINGELEM}
\end{align*}
\]

Description

This function returns the absolute value of \( N \). If \( N \) is a \( \text{RINGELEM} \) then it must belong to an ordered ring.

Example

```cpp
/**/ abs(-3);
3
/**/ abs(-2/3);
2/3
```

I-1.2 adj

Syntax

\[
\text{adj}(M: \text{MAT}): \text{MAT}
\]

Description

This function returns the classical adjoint of the square matrix \( M \).

Example

```cpp
/**/ Use R ::= QQ[t,x,y,z];
/**/ adj(mat([[x,y,z],[t,y,x],[x,x^2,x*y]]));
matrix([[-x^3 +x*y^2, -x*y^2 +x^2*z, x*y -y*z],
[-t*x*y +x^2, x^2*y -x*z, -x^2 +t*z],
[t*x^2 -x*y, -x^3 +x*y, -t*y +x*y]
])
/**/ Z5 := NewRingFp(5);
/**/ adj(matrix(Z5, [[1,2],[3,1]]));
matrix( /*FFp(5)*/
```

25
See Also: inverse(I-9.31 pg.130)

I-1.3 AffHilbert [OBSOLESCENT]

**syntax**

[OBSOLESCENT]

**Description**

Renamed “AffHilbertFn” (I-1.4 pg.26).

I-1.4 AffHilbertFn

**syntax**

AffHilbertFn(R: RING): TAGGED("hp.Hilbert")
AffHilbertFn(R: RING, N: INT): INT

**Description**

The first form of this function computes the affine Hilbert function for R. The second form computes the N-

th value of the affine Hilbert function. The weights of the indeterminates of R must all be 1. For repeated
evaluations of the Hilbert function, use “EvalHilbertFn” (I-5.11 pg.80) instead of “AffHilbertFn(R, N)” in
order to speed up execution.

The coefficient ring must be a field.

**example**

```/**/ Use R ::= QQ[x,y,z];
/**/ AffHilbertFn(R/ideal(z^4-1, x*z^4-y-3));
H(0) = 1
H(1) = 3
H(t) = 4t - 2 for t >= 2```

See Also: AffHilbertSeries(I-1.5 pg.26), EvalHilbertFn(I-5.11 pg.80), HilbertPoly(I-8.6 pg.107), HVector(I-

8.13 pg.112), HilbertSeries(I-8.7 pg.108)

I-1.5 AffHilbertSeries

**syntax**

AffHilbertSeries(TAGGED("Quotient")):TAGGED("$hp.PSeries")

**Description**

This function computes the affine Hilbert-Poincare series of M. The grading must be a positive \(Z^1\)-grading (i.e.

“GradingMat” (I-7.19 pg.103) must have a single row with positive entries), and the ordering must be degree
compatible. In the standard case, i.e. the weights of all indeterminates are 1, the result is simplified so that the
power appearing in the denominator is the dimension of M + 1.

It used to be called “AffPoincare [OBSOLESCENT]” (I-1.6 pg.27).
NOTES:
(i) the coefficient ring must be a field.
(ii) these functions produce tagged objects: they cannot safely be (non-)equality to other values.
For further details on affine Hilbert functions see the book: Kreuzer, Robbiano “Computer Commutative Algebra II”, Section 5.6.

```plaintext
/**/ Use R ::= QQ[x,y,z];
/**/ AffHilbertSeries(R/ideal(z^4-1, x*z^4-y-3));
(1 +x +x^2 +x^3) / (1-x)^2
```

See Also: AffHilbertFn(I-1.4 pg.26), HilbertSeries(I-8.7 pg.108)

**I-1.6  AffPoincare [OBSOLESCENT]**

**Syntax**

```
[OBSOLESCENT]
```

**Description**

Renamed “AffHilbertSeries” (I-1.5 pg.26).

**I-1.7  alias**

**Syntax**

```
alias B_1,...,B_r
```

where each B_i is a ‘‘{\it binding}’’ of the form: Identifier := $PackageName

**Description**

This function is for declaring both global and local aliases for package names. Recall that package names are meant to be long in order to avoid conflicts between the names of functions that are read into a CoCoA session. However, it is inconvenient to have to type out the long package name when referencing a function. So the user chooses an alias to take the place of the package name; the alias is just a means to avoid typing.

1. Global aliases. To avoid typing the full package name as a prefix to package functions, one may declare a short global alias during a CoCoA session. A list of the global aliases is produced by the function “aliases” (I-1.8 pg.28). For examples, see the chapter on packages in the manual, in particular the section, “Global Aliases” (II-7.4 pg.314). Online, enter “?global aliases”.

2. Local aliases. A local alias has the same syntax as a global alias, however it appears inside a package definition. The local aliases work only inside the package and do not conflict with any global aliases already defined. In fact, in order to avoid conflicts, global aliases are not recognized within a package. For examples, again look in the chapter for packages.

```plaintext
/**/ alias LL := $abcd;
/**/ aliases();
Coclib = $coclib
Approx = $approx
(...)
```
TP = $contrib/thmproving
TV = $contrib/typevectors
LL = $abcd

See Also: aliases(I-1.8 pg.28), Introduction to Packages(II-7.1 pg.313)

I-1.8 aliases

syntax

aliases():TAGGED("aliases")

Description

This function prints a list of global aliases for packages. Aliases are formed with the function “alias” (I-1.7 pg.27).

example

/**/ alias LL := $abcd;
/**/ aliases();

Coclib = $coclib
Approx = $approx
(....)
TP = $contrib/thmproving
TV = $contrib/typevectors
LL = $abcd

See Also: alias(I-1.7 pg.27), Introduction to Packages(II-7.1 pg.313)

I-1.9 AlmostQR

syntax

AlmostQR(M: MAT): RECORD

Description

This function computes the decomposition of the matrix into an orthogonal and an upper triangular matrix with 1 on the diagonal. [“orthogonal” meaning that $Q^T * Q$ is a diagonal matrix]

The auxiliary (possibly slow!) function “Mat.SimplifySquareFactorsInAQR” modifies $Q$ and $R$ in the decomposition so that the entries of the diagonal matrix $Q^T * Q$ are squarefree rational.

example

/**/ M := matrix(QQ, [[4, -2, 3], [3, 2, -2], [0, 0, 3]]);
/**/ Dec := AlmostQR(M);
/**/ Dec;
record[Q := matrix(QQ,
[[4, -42/25, 0],
[3, 56/25, 0],
[0, 0, 3]])
, R := matrix(QQ,
[[1, -2/25, 6/25],
[0, 1, -17/14],
[0, 0, 1]]
]
/**/ A := -1;
/**/ A >= 0 and isqrt(A) < 10; --> calls isqrt only if A >= 0
false
See Also: or(I-15.9 pg.199), not(I-14.27 pg.190)

I-1.11 append

append(ref L: LIST, E: OBJECT)

Append the object “E” to the list “L”; this call returns nothing!

NOTE: the old CoCoA-4 syntax “Append(L, E)” is still allowed, but produces a warning; replace the call by “append(ref L, E)”.

/**/ Use R := QQ[t,x,y,z];
/**/ L := [1,2,3];
/**/ append(ref L, 4);
/**/ L;
[1, 2, 3, 4]

See Also: ref(I-18.23 pg.231), concat(I-3.33 pg.54), ConcatLists(I-3.38 pg.57), remove(I-18.29 pg.234)
I-1.12 apply

syntax
apply(phi: RINGHOM, X: RINGELEM): RINGELEM
apply(phi: RINGHOM, X: LIST): LIST
apply(phi: RINGHOM, X: MAT): MAT

Description
Apply homomorphism “phi” to all elements in second argument “X” (“RINGELEM”, “LIST”, or “MAT”)
When “X” is of type “RINGELEM” this is equivalent to the natural syntax “phi(X)”.

example
/**/ Use R ::= QQ[x,y,z];
/**/ S ::= QQ[x[1..3]];  
/**/ phi := PolyAlgebraHom(R, S, indets(S));
/**/ apply(phi, [x^2-y, z-2]);
/**/ apply(phi, x^2-y); -- same as phi(x^2-y)
x[1]^2 -x[2]
/**/ phi(x^2-y);
x[1]^2 -x[2]

See Also: PolyAlgebraHom(I-16.14 pg.205), CanonicalHom(I-3.2 pg.41)

I-1.13 ApproxSolve

syntax
ApproxSolve(L: LIST of RINGELEM): LIST of LIST of RAT

Description
This function returns the list of real solutions (points) of a 0-dimensional polynomial system “L”. Works only if with rational coefficients. Approximate coordinates are given for non-rational solutions.
See also “RationalSolve” (I-18.13 pg.226)

example
/**/ Use QQ[x,y,z];
/**/ L := [x^3-y^2+z-1, x-2, (y-3)*(y+2)];
/**/ RationalSolve(L);
[[2, -2, -3], [2, 3, 2]]
/**/ ApproxSolve(L);
[[2, -2, -3], [2, 3, 2]]
/**/ L := [x^3-y^2+1, (y-3)*(y+2), z];
/**/ indent(ApproxSolve(L));
[ [167001090947516369641767378634802431634869700965461961120334511774287062707365/11579208923731619542357098500868790785326... 0], [2, 3, 0] ]
/**/ L := [x^3-y^2+z-1, x^2-2, (y-3)*(y+2)];
/**/ Pts := ApproxSolve(L);
--> [[17564737135690137373...
/**/ indent([ [ DecimalStr(coord,10) | coord in pt ] | pt in Pts]);

[  
  ["1.4142135624", "-2.0000000000", "2.1715728753"],
  ["1.4142135624", "3.0000000000", "7.1715728753"],
  ["-1.4142135624", "-2.0000000000", "7.8284271247"],
  ["-1.4142135624", "3.0000000000", "12.8284271247"]
]

-- Verify we have an approximate answer:
/**/ indent([ [ FloatStr(eval(f, pt)) | f In L ] | pt In Pts]);

[  
  ["-3.2567*10^(-76)", "-6.2932*10^(-77)", "2.3668*10^(-76)"],
  ["-1.3971*10^(-77)", "8.1808*10^(-78)", "2.5541*10^(-77)"],
  ["-3.7110*10^(-77)", "8.1808*10^(-78)", "2.5541*10^(-77)"],
  ["7.7208*10^(-77)", "3.2902*10^(-77)", "-1.2374*10^(-76)"]
]

See Also: LinSolve(I-12.14 pg.161), RationalSolve(I-18.13 pg.226)

I-1.14 ascii

<table>
<thead>
<tr>
<th><strong>syntax</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>ascii(N: INT): STRING</td>
</tr>
<tr>
<td>ascii(L: LIST of INT): STRING</td>
</tr>
<tr>
<td>ascii(S: STRING): LIST of INT</td>
</tr>
</tbody>
</table>

**Description**

In the first form, “ascii” returns the character whose ASCII code is “N”.
In the second form, “ascii” returns the string whose characters, in order, have the ASCII codes listed in “L”.
The third form is the inverse of the second: it returns the ASCII codes of the characters in “S”.

<table>
<thead>
<tr>
<th><strong>example</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>/**/ ascii(97);</td>
</tr>
<tr>
<td>a</td>
</tr>
<tr>
<td>/**/ C := ascii(&quot;hello world&quot;);</td>
</tr>
<tr>
<td>/**/ C;</td>
</tr>
<tr>
<td>/**/ ascii(C);</td>
</tr>
<tr>
<td>hello world</td>
</tr>
</tbody>
</table>

I-1.15 AsINT

<table>
<thead>
<tr>
<th><strong>syntax</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>AsINT(N: INT): INT</td>
</tr>
<tr>
<td>AsINT(N: RAT): INT</td>
</tr>
<tr>
<td>AsINT(N: RINGELEM): INT</td>
</tr>
</tbody>
</table>

**Description**

If the argument is an integer value this function returns this value as an INT, otherwise it throws an error.
/**/ Use P ::= QQ[x,y];
/**/ type(LC(3*x-y));
RINGELEM
/**/ type(AsINT(LC(3*x-y)));
INT
-- /**/ type(AsINT(LC((3/2)*x-y))); --> !!! ERROR !!!

See Also: AsRAT(I-1.16 pg.32)

I-1.16 AsRAT

syntax

AsRAT(N: INT): RAT
AsRAT(N: RAT): RAT
AsRAT(N: RINGELEM): RAT

Description

If the argument is a rational value this function returns this value as a RAT, otherwise it throws an error. Note that if the argument is actually an integer the result is nevertheless a RAT (with denominator 1).

/**/ Use P ::= QQ[x,y];
/**/ type(LC(3*x-y));
RINGELEM
/**/ type(AsRAT(LC(3*x-y)));
RAT

See Also: AsINT(I-1.15 pg.31)
Chapter I-2

B

I-2.1 BaseRing

**syntax**

BaseRing(RmodI: (Quotient)RING: RING
BaseRing(K: (Fraction Field)RING): RING

**Description**

This function gives the "base ring" of a given ring; e.g. if "K" was constructed as the fraction field of "R" then "BaseRing(K)" produces "R". All rings in CoCoA are derived from "ZZ" via various steps; "BaseRing" gives the ring which is one step closer to "ZZ".

**example**

/**/ Fpx ::= ZZ/(7)[x];
/**/ Fp := BaseRing(Fpx); --> ZZ/(7)
/**/ BaseRing(Fp) = ZZ;
true

See Also: NewFractionField(I-14.1 pg.179), NewQuotientRing(I-14.9 pg.182), NewPolyRing(I-14.8 pg.182)

I-2.2 BBasis5

**syntax**

BBasis5(I: IDEAL): LIST

**Description**

***** NOT YET IMPLEMENTED *****

This function is implemented in ApCoCoALib by Stefan Kaspar.

The function "BBasis5" calls the CoCoAServer to compute a Border Basis of zero dimensional ideal I.

**example**

Use QQ[x, y], DegLex;
I := ideal([x^2, xy + y2]);
BBasis := BBasis5(I);
### I-2.3 BettiDiagram

**syntax**

\[
\text{BettiDiagram}(X: \text{IDEAL or (quotient)RING or MODULE})
\]

**Description**

This function computes the ("Macaulay-style") Betti diagram for "M".

**example**

```plaintext
/**/ Use R ::= QQ[t,x,y,z];
/**/ I := ideal(x^2-y*t, x*y-z*t, x*y);
/**/ RES := res(I);
/**/ PrintRes(RES);
0 --> R(-5)^2 --> R(-4)^4 --> R(-2)^3
/**/ B := BettiDiagram(RES); indent(B);
record[
  Diagram := matrix(ZZ,
  [[3, 0, 0],
   [0, 4, 2]]),
  FirstShift := 2
]
/**/ PrintBettiDiagram(B);

0 1 2
---
2: 3 - -
3: - 4 2
---
Tot: 3 4 2
```

**See Also:** BettiMatrix(I-2.4 pg.34), PrintRes(I-16.30 pg.213), PrintBettiDiagram(I-16.27 pg.212), PrintBettiMatrix(I-16.28 pg.213)

### I-2.4 BettiMatrix

**syntax**

\[
\text{BettiMatrix}(M: \text{IDEAL|MODULE|LISTResolution})
\]

**Description**

This function returns the Betti matrix for "M".

**example**

```plaintext
/**/ Use R ::= QQ[t,x,y,z];
/**/ I := ideal(x^2-y*t, x*y-z*t, x*y);
/**/ PrintRes(I);
0 --> R^2(-5) --> R^4(-4) --> R^3(-2)
/**/ BettiMatrix(I);

matrix(ZZ,
[[0, 0, 0],
 [3, 0, 0],
 [0, 0, 0],
 [0, 4, 0],
 [0, 0, 2]])
```
/**/ PrintBettiMatrix(I);
-- --> --
0 0 0
0 0 3
0 0 0
0 4 0
2 0 0
-- --> --

See Also: PrintRes(I-16.30 pg.213), PrintBettiDiagram(I-16.27 pg.212)

I-2.5  binomial

    syntax

binomial(N: INT, K: INT): INT
binomial(N: RINGELEM, K: INT): RINGELEM

Description
This function computes the binomial coefficient, "N choose K" according to the formula \((N)(N-1)(N-2)...(N-K+1)/K!\).
The same formula is used if N is a polynomial. The integer K cannot be negative.

    example

/**/ binomial(4,2);
6
/**/ binomial(-4,3);
-20
/**/ binomial(x^2+2*y,3);
(1/6)*x^6 +x^4*y +(1/2)*x^4 +2*x^2*y^2 -2*x^2*y +(4/3)*y^3 +(1/3)*x^2 -2*y^2 +(2/3)*y
/**/ It = ***(x^2+2*y)*(x^2+2*y-1)*(x^2+2*y-2)/6***;
true

See Also: BinomialRepr, BinomialReprShift(I-2.6 pg.35)

I-2.6  BinomialRepr, BinomialReprShift

    syntax

BinomialRepr(N: INT, K: INT): LIST of INT

where N and K are positive.

Description
The function "BinomialRepr" computes the "K"-binomial representation of "N", also called Macaulay representation, i.e. the unique expression

\[ N = \text{binomial}(N(K),K) + \text{binomial}(N(K-1),K-1) + ... + \text{binomial}(N(L),L) \]
where $N(K) > \ldots > N(L) \geq 1$, for some $L$. The value returned is the list \[ \{N(t) \mid t \in 1..K\} \] where $N(t)=0$ for all $t < L$.

The function call “BinomialReprShift(N,K,up,down)” computes the integer

\[
\binomial(N(K) +up, K+down) + \\
\binomial(N(K-1)+up,(K-1)+down) + \\
\ldots + \\
\binomial(N(L) +up, L+down)
\]

It is useful in generalizations of Macaulay’s theorem characterizing Hilbert functions.

```plaintext
/**/ BinRep := BinomialRepr(13,4);
/**/ BinRep;
[1, 3, 4, 5]
/**/ BinomialReprShift(13,4,1,1);
16
```

See Also: binomial(I-2.5 pg.35)

### I-2.7 block

**syntax**

```
block C_1; \ldots ; C_n EndBlock;
```

where each $C_i$ is a command.

**Description**

The “block” command executes the commands as if they were one command. What this means in practice is that CoCoA will not print a string of dashes after executing each “$C_i$”. Thus, “Block” is used on-the-fly and not inside user-defined functions. (It has nothing to do with declaration of local variables, for instance, as one might infer from some other computer languages.) The following example should make the use of “Block” clear:

```plaintext
/**/ Print "hello "; Print "world";
hello world
---------------------
/**/ Block
/**/ Print "hello ";
/**/ Print "world";
/**/ EndBlock;
hello world
---------------------
/**/ Block
/**/ PrintLn GCD([12, 24, 96]);
/**/ PrintLn LCM([12, 24, 96]);
/**/ PrintLn GCD([x+y, x^2-y^2]);
/**/ Print LCM([x+y, x^2-y^2]);
/**/ EndBlock;
12
96
```
I-2.8 BlockMat

**syntax**

BlockMat(LIST of LIST of MAT): MAT

**Description**

This function creates a block matrix from a LIST of rows of matrices.

The following restrictions on the sizes of the matrices apply: in each row of matrices “NumRows(\(M\))” must be constant, and for all rows of matrices the total number of columns must be the same.

The function “BlockMat2x2” (I-2.9 pg.37) has a simpler syntax for a 2x2 block matrix.

**example**

```plaintext
/**/ A := RowMat([1,2,3,4]); B := RowMat([0,0]);
-- /**/ BlockMat2x2(A, B, B, A); --> !!! ERROR !!! as expected
/**/ BlockMat([[A,B], [B,A]]);
matrix(QQ,
[1, 2, 3, 4, 0, 0],
[0, 0, 1, 2, 3, 4])
```

**See Also:** ConcatHor(I-3.36 pg.56), ConcatVer(I-3.39 pg.57), ConcatHorList(I-3.37 pg.56), ConcatVerList(I-3.40 pg.58), ConcatDiag(I-3.35 pg.55), ConcatAntiDiag(I-3.34 pg.55), BlockMat2x2(I-2.9 pg.37)

I-2.9 BlockMat2x2

**syntax**

BlockMat2x2(A: MAT, B: MAT, C: MAT, D: MAT): MAT

**Description**

This function creates a block matrix. Each entry is a matrix. Given A, B, C, D matrices, then “BlockMat(A, B, C, D)” returns the matrix

\[
\begin{bmatrix}
A & B \\
C & D
\end{bmatrix}
\]

The obvious restrictions on the sizes of the matrices apply:

“NumRows(A) = NumRows(B)” and “NumRows(C) = NumRows(D)” and “NumCols(A) = NumCols(C)” and “NumCols(B) = NumCols(D)”.

The function “BlockMat” (I-2.8 pg.37) offers more flexibility, but with a heavier syntax.

**example**

```plaintext
/**/ A := matrix([[1,2,3], [4,5,6]]);
/**/ B := matrix([[1,2], [3,4]]);
/**/ C := matrix([[1,1,1], [2,2,2], [3,3,3]]);
/**/ D := matrix([[4,4], [5,5], [6,6]]);
/**/ BlockMat2x2(A, B, C, D);
matrix(QQ,
[1, 2, 3, 1, 2],
[4, 5, 6, 5, 6])
```
[4, 5, 6, 3, 4],
[1, 1, 1, 4, 4],
[2, 2, 2, 5, 5],
[3, 3, 3, 6, 6])

See Also: ConcatHor(I-3.36 pg.56), ConcatVer(I-3.39 pg.57), ConcatDiag(I-3.35 pg.55), ConcatAntiDiag(I-3.34 pg.55), BlockMat(I-2.8 pg.37)

I-2.10  Bool01

syntax

Bool01(B: BOOL): INT

Description

This function converts a boolean to an integer using the convention: "false" becomes 0, and "true" becomes 1.

example

/**/ Id4 := matrix([[Bool01(i=j) | i in 1..4] | j in 1..4]);
/**/ Id4;
matrix(QQ,
    [[1, 0, 0, 0],
     [0, 1, 0, 0],
     [0, 0, 1, 0],
     [0, 0, 0, 1]])

I-2.11  break

syntax

break

Description

This command must be used inside a loop statement ("for", "foreach", "repeat", or "while"). When executed, the current loop statement is terminated and control passes to the command following the loop statement. Thus, in the case of nested loops "break" does "not" break out of all loops back to the "top level" (see " Return ").

gamma

example

/**/ For I := 5 To 1 Step -1 Do
/**/     For J := 1 To 100 Do
/**/         Print J, ",
/**/             If J = I Then PrintLn; Break; EndIf;
/**/         EndFor;
/**/     EndFor;
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1

See Also: return(I-18.35 pg.236)
I-2.12 BringIn

Syntax

`BringIn(E: OBJECT): OBJECT`

Description

This function maps a polynomial (or a list, matrix of these) into the current ring, preserving the names of the indeterminates.

This function is not implemented on ideals because might be misleading: one might expect that bringing an ideal from \("\mathbb{R}[x,y]\) into \("\mathbb{R}[x]\)" means eliminating \("y\)”, while others might expect the ideal generated by mapping the generators. For example in the first case \((x - y, x + y)\) returns the ideal \((x)\), in the second case returns an error. So, if you want to map the generators of the ideal type \("\text{ideal(BringIn(gens(I)))}\)\).  

– Changing characteristic from non-0 to 0 is NOT YET IMPLEMENTED in CoCoA-5 When mapping from a ring of finite characteristic to one of zero characteristic then consistent choices of image for the coefficients are made (i.e. if two coefficients are equal mod p then their images will be equal).

Example

```plaintext
/**/ RR ::= QQ[x[1..4],z,y];
/**/ SS ::= ZZ[z,y,x[1..2]];
/**/ Use RR;
/**/ F := (x[1]-y-z)^2; F;
x[1]^2 -2*x[1]*z +z^2 -2*x[1]*y +2*z*y +y^2
/**/ Use SS;
/**/ BringIn(F);
z^2 +2*z*y +y^2 -2*z*x[1] -2*y*x[1] +x[1]^2
/**/ Use R ::= QQ[x,y,z];
/**/ F := (1/2)*x^3 + (34/567)*x*y*z - 890; -- poly with rational coefficients
/**/ Use S ::= ZZ/(101)[x,y,z];
/**/ BringIn(F);
-50*x^3 -19*x*y*z +19
```

See Also: PolyAlgebraHom(I-16.14 pg.205), apply(I-1.12 pg.30), QZP(I-17.4 pg.218), ZPQ(I-25.3 pg.292)
Chapter I-3

C

I-3.1 Call [OBsolete]

[OBsolete]

Description

OBsolete: in CoCoA-5 functions can be used directly. See “FUNCTIONs are first class objects” (III-7.2 pg.349).

I-3.2 CanonicalHom

CanonicalHom(R: RING, S: RING): RINGHOM

Description

CanonicalHom(R, S) – where R and S are rings, gives the canonical homomorphism from R to S. Currently it works only on the most natural constructions:

\[ \begin{align*}
\mathbb{Z} &\to S \\
\mathbb{Q} &\to S \\
R &\to R/I \\
R &\to \text{FractionField}(R) \\
R &\to R[x[1..N]] \\
\end{align*} \]

example

/**/ Use R ::= QQ[x,y];
/**/ RmodI := NewQuotientRing(R, ideal(x^2-1));

/**/ phi := CanonicalHom(R, RmodI);
/**/ phi(x^3*y);
(x*y)
/**/ RingOf(It) = RmodI;
true

/**/ RingElem(RmodI, x^3*y); -- same as phi(x^3*y)
-- internally computes CanonicalHom

See Also: NewFractionField(I-14.1 pg.179), NewQuotientRing(I-14.9 pg.182), NewPolyRing(I-14.8 pg.182), CanonicalHom(I-3.2 pg.41), PolyAlgebraHom(I-16.14 pg.205), PolyRingHom(I-16.15 pg.206)
I-3.3 CanonicalRepr

**syntax**

```plaintext
CanonicalRepr(f: RINGELEM): RINGELEM
```

**Description**

Given an element "f" in a quotient ring "R/I" this function returns a representative of "f" in "R".

```plaintext
/**/ Use R ::= QQ[a];
/**/ RmodI := R/ideal(a^2-2);
/**/ Use RmodI;
/**/ a^3;
(2*a)
/**/ RingOf(a^3);
RingWithID(9, "RingWithID(7)/ideal(a^2 -2)"
/**/ CanonicalRepr(a^3);
2*a
/**/ RingOf(CanonicalRepr(a^3));
RingWithID(7, "QQ[a]"
```

**See Also:** NewQuotientRing(I-14.9 pg.182), DefiningIdeal(I-4.5 pg.66)

I-3.4 CartesianProduct, CartesianProductList

**syntax**

```plaintext
CartesianProduct(L1: LIST, L2: LIST, L3: LIST, ..): LIST
CartesianProductList(L: LIST of LIST): LIST
L1 >< L2
L1 >< L2 >< ... >< Ln
where each Li is a LIST
```

**Description**

This command returns the list whose elements form the Cartesian product of \( L_1, ..., L_n \).

For the N-fold product of a list with itself, one may use "tuples" (I-20.14 pg.278).

```plaintext
/**/ L1 := [1,2,3];
/**/ L2 := ["a","b"];
/**/ L1 >< L2 >> [5]; -- same as
/**/ CartesianProduct(L1, L2, [5]); -- same as
/**/ CartesianProductList([L1, L2, [5]]); -- this takes a list of lists
[[1, "a", 5], [1, "b", 5], [2, "a", 5], [2, "b", 5], [3, "a", 5], [3, "b", 5]]
-------------------------------
/**/ ChessBoard := (1..8)><(1..8); -- Need brackets around 1..8 otherwise
-- we get a parse error.
```

Note that only \("\neq\)" is used for "not equal" in CoCoA.

**See Also:** CoCoA Operators(II-3.1 pg.301), operators, shortcuts(I-0.1 pg.23), tuples(I-20.14 pg.278)
I-3.5  Cast [OBSOLETE]

[OBSOLETE] syntax

Description

[OBSOLETE] To cast INT, RAT, STRING to a polynomial (and more in general to a RINGELEM) use “RingElem” (I-18.38 pg.238).
To cast RINGELEM to INT, RAT use “AsINT” (I-1.15 pg.31), “AsRAT” (I-1.16 pg.32).
To cast LIST to MAT use “matrix” (I-13.8 pg.170). To cast MAT to LIST use “GetRows” (I-7.16 pg.102), “GetCols” (I-7.12 pg.101).
To cast a MODULEELEM to LIST use “compts” (I-3.32 pg.54). To cast a MODULE to MAT use “GensAsCols, GensAsRows” (I-7.9 pg.99). To cast a MAT to MODULE use “SubmoduleCols, SubmoduleRows” (I-19.37 pg.263).

See Also:  AsINT(I-1.15 pg.31), AsRAT(I-1.16 pg.32), gens(I-7.8 pg.98), GensAsCols, GensAsRows(I-7.9 pg.99), GetCols(I-7.12 pg.101), GetRows(I-7.16 pg.102), ideal(I-9.2 pg.113), matrix(I-13.8 pg.170), ModuleElem(I-13.23 pg.176), RingElem(I-18.38 pg.238), SubmoduleCols, SubmoduleRows(I-19.37 pg.263)

I-3.6  ceil

ceil(X: RAT): INT

Description

This function returns the least integer greater than or equal to “X”.

example

/**/  ceil(0.99);  1
/**/  ceil(0.01);  1
/**/  ceil(1);  1
/**/  ceil(-0.99);  0

See Also:  floor(I-6.12 pg.88), round(I-18.50 pg.243), num(I-14.29 pg.191), den(I-4.7 pg.67)

I-3.7  CFApprox

CFApprox(X: RAT, MaxRelErr: RAT): RAT

Description

“CFApprox” finds the “simplest” continued fraction approximant to “X” which is within the maximum specified “relative error”.
Chapter I-3. C

/**/ CFApprox(1.414213, 10^(-2));
17/12

See Also: CFApproximants(I-3.8 pg.44), ContFrac(I-3.44 pg.59), SimplestRatBetween(I-19.11 pg.251)

I-3.8 CFApproximants

CFApproximants(X: RAT): LIST of RAT

Description

“CFApproximants” returns a list of all continued fraction approximants to the rational “X”.

/**/ CFApproximants(1.414213);
[1, 3/2, 7/5, 17/12, 41/29, 99/70, 239/169, 577/408, 816/577, 1393/985, 6388/4517, 7781/5502, 14169/10019, 21950/15521, 36119/25540, 58069/41061, 152257/107662, 210326/148723, 1414213/1000000]

See Also: CFApprox(I-3.7 pg.43), ContFrac(I-3.44 pg.59)

I-3.9 characteristic

characteristic(R: RING): INT

Description

This function returns the characteristic of the current ring, in the first case, or of the ring R, in the second.

/**/ Use R ::= ZZ/(3)[t];
/**/ S ::= QQ[x,y];
/**/ characteristic(CurrentRing);
3
/**/ characteristic(S);
0

See Also: IsFiniteField(I-9.45 pg.135), LogCardinality(I-12.17 pg.162)

I-3.10 CharPoly

CharPoly(M: MAT, X: RINGELEM): RINGELEM

Description

This function returns the characteristic polynomial of “M”, square matrix, in the indeterminate “X”.

See also “MinPoly” (I-13.18 pg.174).
I-3.11 CheckArgTypes

### Syntax

```
CheckArgTypes(Ltype: LIST of TYPE, Larg: LIST)
```

### Description

This function provides a basic type checking for user defined functions: it checks whether the types of the elements in the third argument, a list, correspond to the types in the second list. If so, it returns nothing, otherwise returns an error.

```
/**/ -- the following returns an error for the 2nd argument (INT)
/**/ -- CheckArgTypes([RAT, RINGELEM, MAT], [2/3, 20, LexMat(3)]); --> ERROR
--> ERROR: Arg 2 is INT but must be RINGELEM

/**/ -- the following returns nothing
/**/ CheckArgTypes([RAT, [INT,RAT,RINGELEM], MAT], [2/3, 20, LexMat(3)]);

/**/ -- an example of use for type checking
/**/ Define Pow(F, N)
/**/   CheckArgTypes([[INT,RAT,RINGELEM,IDEAL,MAT], INT], [F, N]);
/**/   Return F^N;
/**/ EndDefine; -- Pow
/**/ Pow(x, 3);
x^3
/**/ -- Pow(2, x); --> ERROR
--> ERROR: Arg 2 is RINGELEM but must be INT
```

### See Also:

MinPoly(I-13.18 pg.174)

I-3.12 ciao

### Syntax

```
ciao
```

### Description

This command is used to quit CoCoA. It may be used only at top level.

### See Also:

quit(I-17.2 pg.217)

I-3.13 ClearDenom

### Syntax

```
ClearDenom(F: RINGELEM): RINGELEM
```
Chapter I-3. C

### Description

This function clears the denominators of the coefficients in a polynomial over QQ. It simply multiplies by the least common multiple of the denominators.

```plaintext
/**/ Use QQ[x,y];
/**/ F := (2/3)*x + (4/5)*y;
/**/ ClearDenom(F);
10*x +12*y
```

### I-3.14 close

#### Syntax

```
close(D: DEVICE)
```

#### Description

This function closes the device D.

```plaintext
D := OpenOFile("my-test"); -- open file for output from CoCoA
Print "test" On D; -- write to my-file
Close(D); -- close the file
Close(DEV.STDIN); -- close the standard input device
-- Bye

(Close(DEV.OUT) suppresses all output to the CoCoA window.)
```

**See Also:** Introduction to IO (II-6.1 pg.309)

### I-3.15 CloseLog

#### Syntax

```
CloseLog(D: DEVICE)
```

#### Description

***** NOT YET IMPLEMENTED *****

This function “OpenLog” (I-15.4 pg.196) opens the output device D and starts to record the output from a CoCoA session on D.

This function closes the device D and stops recording the CoCoA session on D.

**See Also:** OpenLog (I-15.4 pg.196)

### I-3.16 CoCoA-4 mode

#### Syntax

```
*** E ***
```

where ‘\verb+\verb+E\verb+\verb+’ is a CoCoA-4 expression.
**Description**

CoCoA-5 is not fully backward compatible with CoCoA-4, i.e. some CoCoA-4 programs will be rejected by CoCoA-5. CoCoA-4 mode helps ease the transition to CoCoA-5.

In CoCoA-4 it was not necessary to write explicitly the product between two indeterminates; in CoCoA-5 this is obligatory.

The expression “E” may also contain function calls, but only if the function names begin with a capital letter.

```plaintext
/**/ Use QQ[x,y,z];
/**/ f := 2*x^2*y - 3*x*y*z + 5*y*z^2 + 6*z^3;
/**/ g := 2*x^2*y - 3*x*y*z + 5*y*z^2 + 6*z^3; -- C4 mode, more compact!
/**/ f = g;
true
```

See Also:  Migrating from CoCoA-4 and keeping up-to-date(II-9 pg.321), not(I-14.27 pg.190), and(I-1.10 pg.29), or(I-15.9 pg.199)

---

**I-3.17  CocoaLimits**

**syntax**

```plaintext
CocoaLimits(): RECORD
```

**Description**

***** NOT YET IMPLEMENTED *****

This function returns the maximum allowable characteristic of a CoCoA ring and the maximum allowable exponent in a CoCoA expression. These numbers may vary depending on the platform on which CoCoA is run.

```plaintext
CocoaLimits();
record[MaxChar := 32767, MaxExp := 2147483647]
```

---

**I-3.18  CocoaPackagePath**

**syntax**

```plaintext
CocoaPackagePath(): STRING
```

**Description**

This function returns the path name of the directory containing the CoCoA libraries. It is platform dependent.

```plaintext
/**/ CocoaPackagePath();
/Applications/CoCoA-5/packages
```

---

**I-3.19  codomain**

**syntax**

```plaintext
codomain(\text{phi}: \text{RINGHOM}): \text{RING}
```
**Description**

This function returns the domain of the homomorphism “phi”

```plaintext
/**/ P := NewPolyRing(RingQQ(), "alpha,beta");
/**/ phi := CanonicalHom(RingZZ(), P);
/**/ codomain(phi);
RingWithID(4, "QQ[alpha,beta]")
/**/ psi := CoeffEmbeddingHom(P);
/**/ codomain(psi);
RingWithID(4, "QQ[alpha,beta]")
```

**See Also:** codomain(I-3.19 pg.47), Commands and Functions for RINGHOM(III-10.3 pg.360), Commands and Functions returning RINGHOM(III-10.4 pg.360)

---

**I-3.20 CoeffEmbeddingHom**

**syntax**

```
CoeffEmbeddingHom(P: RING): RINGHOM
```

**Description**

This function returns the coefficient embedding homomorphism of the polynomial ring “P”.

```plaintext
/**/ Use P ::= QQ[x,y];
/**/ phi := CoeffEmbeddingHom(P); -- phi: QQ -> P
/**/ f := 2*x+3*y;
/**/ f/phi(LC(f));
x + (3/2)*y
```

**See Also:** CanonicalHom(I-3.2 pg.41)

---

**I-3.21 coefficients**

**syntax**

```
coefficients(F: RINGELEM): LIST
coefficients(F: RINGELEM, S: LIST): LIST
```

**Description**

This function returns a list of coefficients of “F” in “CoeffRing(RingOf(F))”.

Called with one argument “F” it returns the list of all non-zero coefficients; the order being decreasing on the terms in “F” as determined by the term-ordering of “RingOf(F)”.

Called with two arguments “F,S” it returns the coefficients of the list of specified terms “S”; their order is determined by the list “S”. If a terms does not appear in “F” then the corresponding coefficient is 0.

The old form (CoCoA-4) “Coefficients(F,x)” for the coefficients of F w.r.t an indeterminate x is now implemented as “CoefficientsWRT” (I-3.22 pg.49) and “CoeffListWRT” (I-3.23 pg.50).

```plaintext
/**/ Use R ::= QQ[x,y,z];
/**/ F := 3*x^2*y + 5*y^2 - x*y;
/**/ Coeffs := coefficients(F); Coeffs; -- with one argument
```
/**/ phi := CoeffEmbeddingHom(RingOf(F));
/**/ F = ScalarProduct(apply(phi, Coeffs), support(F));
true

/**/ Skeleton := [1, x, y, z, x^2, x*y, y^2, y*z, z^2];
/**/ Coefs := coefficients(F, Skeleton); Coefs; -- with two arguments
[0, 0, 0, 0, -1, 5, 0, 0]
/**/ ScalarProduct(apply(phi, Coeffs), Skeleton); -x*y +5*y^2

/**/ L := CoefficientsWRT(F,[x,y,z]); indent(L); -- similar function
[
  record[PP := y^3, coeff := 5],
  record[PP := x^2*y, coeff := 3],
  record[PP := x*y^5, coeff := -1]
]
/**/ F = sum([X.coeff * X.PP | X In L]);
true

/**/ L := CoeffListWRT(F, y); L; -- similar function
[0, 3*x^2 -x, 5]
/**/ F = sum([L[d+1]*y^d | d in 0..(len(L)-1)]);
true

/**/ R3 := NewFreeModule(R,3);
/**/ V := ModuleElem(R3, ***[3x^2+y, x-5z^3, x+2y]***);
/**/ ConcatLists([coefficients(V[i]) | i In 1..NumCompts(V)]);
[3, 1, -5, 1, 1, 2]

See Also: Coefficient Rings(III-9.3 pg.354), CoefficientsWRT(I-3.22 pg.49), CoeffListWRT(I-3.23 pg.50), LC(I-12.3 pg.156), monomials(I-13.26 pg.177), support(I-19.41 pg.265)

I-3.22 CoefficientsWRT

** syntax 
CoefficientsWRT(F: RINGELEM, X: RINGELEM): LIST of RECORD
CoefficientsWRT(F: RINGELEM, S: LIST of RINGELEM): LIST of RECORD

** Description 

The first function returns the list of the coefficients and PPs of “F” seen as a polynomial in “X”; the second function does the same but viewing “F” as a polynomial in all the indeterminates in the set “S”.

Note that coefficients in the result are RINGELEM belonging to “RingOf(F)”.

** example 
/**/ Use R := QQ[x,y,z];
/**/ f := x^3*z+x*y+x*z+y+2*z;
/**/ Cx := CoefficientsWRT(f, x); -- same as...
/**/ Cx := CoefficientsWRT(f, [x]);
/**/ indent(Cx);
[ record[PP := x^-3, coeff := z],
  record[PP := x, coeff := y +z],
  record[PP := 1, coeff := y +2*z]
Chapter I-3. C

```plaintext
/**/ f = sum([M.coeff * M.PP | M In Cx]);
true
/**/ Foreach M In Cx Do Print " +(" , M.coeff, ")*", M.PP; EndForeach;
+(y +2*z)*1 +(y +z)*x +(z)*x^3
/**/ Cxz := CoefficientsWRT(f, [x,z]);
/**/ indent(Cxz);
[ record[PP := x^3*z, coeff := 1],
  record[PP := x*z, coeff := 1],
  record[PP := x, coeff := y],
  record[PP := z, coeff := 2],
  record[PP := 1, coeff := y]
]
```

See Also: Coefficient Rings(III-9.3 pg.354), CoeffListWRT(1-3.23 pg.50), coefficients(I-3.21 pg.48), CoeffOfTerm(I-3.24 pg.50), ContentWRT(I-3.43 pg.59), LC(I-12.3 pg.156), monomials(I-13.26 pg.177), support(I-19.41 pg.265)

### I-3.23 CoeffListWRT

**syntax**

```
CoeffListWRT(F: RINGELEM, X: RINGELEM): LIST of RINGELEM
```

**Description**

This function returns the list of the coefficients of “F” seen as a univariate polynomial in “X”, an indeterminate or a list of indeterminates. All entries in the returned list are RingElems belonging to “RingOf(F)”.

Note that the returned list is “reversed” from the CoCoA-4 analogue “Coefficients(F,X)” thus to re-use old code you should call “reversed(CoeffListWRT(F,X))”.

**example**

```plaintext
/**/ Use R ::= QQ[x,y,z];
/**/ F := 5*y^2 + (3*x^2-x)*y;
/**/ L := CoeffListWRT(F, y); Print L;
[0, 3*x^2 -x, 5]
/**/ F = sum([L[d+1]*y^d | d in 0..(len(L)-1)]);
true
```

See Also: coefficients(I-3.21 pg.48), CoeffListWRT(I-3.22 pg.49)

### I-3.24 CoeffOfTerm

**syntax**

```
CoeffOfTerm(F: RINGELEM, T: RINGELEM): RINGELEM
```

**Description**

This function returns the coefficient of the term “T” occurring in “F”. NOTE: In CoCoA 4 the order of the arguments was different.

**example**

```plaintext
/**/ Use R ::= QQ[x,y,z];
/**/ F := 5*x*y^2 - 3*z^3;
```
/**/ CoeffOfTerm(F, x*y^2); 5
/**/ CoeffOfTerm(F, x^3); 0
/**/ CoeffOfTerm(F, z^3); -3

See Also: coefficients(I-3.21 pg.48), LC(I-12.3 pg.156), log(I-12.16 pg.162), LogToTerm(I-12.18 pg.163), monomials(I-13.26 pg.177), support(I-19.41 pg.265)

I-3.25 CoeffRing

**syntax**

CoeffRing(R: RING): RING

**Description**

This function returns the ring of coefficients of a polynomial ring.

**example**

/**/ Use R ::= QQ[x,y,z];
/**/ S ::= ZZ/(2)[a,b,c];
/**/ CoeffRing(R);
QQ
/**/ CoeffRing(S);
FFp(2)

See Also: characteristic(I-3.9 pg.44), coefficients(I-3.21 pg.48), CurrentRing(I-3.49 pg.61), indets(I-9.21 pg.125)

I-3.26 ColMat

**syntax**

ColMat(L: LIST): MAT
ColMat(R: RING, L: LIST): MAT

**Description**

This function returns the matrix whose only column consists of the elements of the list L.

**example**

/**/ ColMat([3,4,5]);
matrix(QQ,
    [[3],
     [4],
     [5]])
/**/ RingOf(It); -- default ring is QQ
QQ
/**/ ColMat(ZZ,[3,4,5]);
matrix(ZZ,
    [[3],
     [4],
     [5]])
See Also: matrix(I-13.8 pg.170), BlockMat(I-2.8 pg.37), DiagMat(I-4.15 pg.71), RowMat(I-18.51 pg.243), GensAsCols, GensAsRows(I-7.9 pg.99)

I-3.27 colon

**syntax**

colon(M: IDEAL, N: IDEAL): IDEAL
colon(M: MODULE, N: MODULE): IDEAL

**Description**

This function returns the quotient of M by N: the ideal of all polynomials F such that F*G is in M for all G in N. The command “M : N” is a shortcut for “colon(M, N)”.

See also “HColon” (I-8.1 pg.105) for non-homogeneous input.

**example**

```plaintext
/**/ Use R ::= QQ[x,y];
/**/ ideal(x*y, x^2) : ideal(x);
ideal(y, x)
/**/ colon(ideal(x^2, x*y), ideal(x, x-y^2));
ideal(x)
```

See Also: saturate(I-19.1 pg.245), HSaturation(I-8.12 pg.111), HColon(I-8.1 pg.105)

I-3.28 ColumnVectors [OBSOLETE]

**syntax**

([OBSOLETE])

**Description**

[OBSOLETE] Essentially replaced by “GensAsCols, GensAsRows” (I-7.9 pg.99) and “SubmoduleCols, SubmoduleRows” (I-19.37 pg.263) See Also: GensAsCols, GensAsRows(I-7.9 pg.99), SubmoduleCols, SubmoduleRows(I-19.37 pg.263)

I-3.29 Comp [OBSOLETE]

**syntax**

([OBSOLETE])

**Description**

[OBSOLETE] please use “[...]” for accessing entries in a list by index, or the record field selector operator.

See Also: operators, shortcuts(I-0.1 pg.23), record field selector(I-18.21 pg.230)
I-3.30 Comparison Operators

**syntax**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A &lt; B</td>
<td>A is less than B</td>
</tr>
<tr>
<td>A &gt; B</td>
<td>A is greater than B</td>
</tr>
<tr>
<td>A &lt;= B</td>
<td>A is less than or equal to B</td>
</tr>
<tr>
<td>A &gt;= B</td>
<td>A is greater than or equal to B</td>
</tr>
</tbody>
</table>

**return** BOOL

**Description**

These operators perform the corresponding comparison between “A” and “B” returning “true” or “false”; they will signal an error if “A” and “B” are not comparable.

**example**

```plaintext
/**/ "abc" < "def"; -- lex ordering for strings
true
```

**See Also:** Equality Test(I-5.7 pg.78), operators, shortcuts(I-0.1 pg.23)

I-3.31 CompleteToOrd

**syntax**

CompleteToOrd(M: MAT): MAT  
CompleteToOrd(M1, M2: MAT): MAT

**Description**

This function returns an ordering matrix (i.e. with non-zero determinant) completing the first matrix.

Given two matrices M1 and M2, it just concatenates of M1 and M2 and makes it square removing the redundant rows.

Given only one matrix M it completes M to an ordering matrix; if M is suitable the resulting matrix defines a term-ordering.

**example**

```plaintext
/**/ M := matrix([[1,2,3,4]]);
/**/ CompleteToOrd(M);
 matrix(QQ,
      [[1, 2, 3, 4],
       [0, 0, 0, -1],
       [0, 0, -1, 0],
       [0, -1, 0, 0]])

/**/ CompleteToOrd(matrix([[1,2,0,0]]));
 matrix(QQ,
      [[1, 2, 0, 0],
       [0, 0, 1, 1],
       [0, 0, 0, -1],
       [0, -1, 0, 0]])
```
/* CompleteToOrd(matrix([[1,2,0,0],[0,0,3,0]])); */
matrix(QQ,
[[1, 2, 0, 0],
 [0, 0, 3, 0],
 [0, 0, 0, 1],
 [0, -1, 0, 0]])

/* CompleteToOrd(matrix([[1,2,0,0],[0,0,3,0]]), RevLexMat(4)); */
matrix(QQ,
[[1, 2, 0, 0],
 [0, 0, 3, 0],
 [0, 0, 0, -1],
 [0, -1, 0, 0]])

--> not a term-ordering

See Also: LexMat(I-12.6 pg.157), RevLexMat(I-18.37 pg.237), StdDegLexMat(I-19.30 pg.260), StdDegRevLexMat(I-19.31 pg.260)

### I-3.32 compts

**syntax**

```plaintext
compts(V: MODULEELEM): LIST
Comps(V: MODULEELEM): LIST
```

**Description**

This function returns the list of components of a ModuleElem V. It is like converting a ModuleElem into a generic list. Note that a ModuleElem is a more structured object than a generic list.

**example**

```plaintext
/* use R ::= QQ[x,y,z]; */
/* R3 := NewFreeModule(R,3); */
/* V := ModuleElem(R3, ***[3x^2+4y, 2x-5z^3, 2x+2y]***); V; */
[3*x^2 +4*y, -5*z^3 +2*x, 2*x +2*y]
/* type(V); */
MODULEELEM

/* compts(V); */
[3*x^2 +4*y, -5*z^3 +2*x, 2*x +2*y]
/* type(compts(V)); */
LIST
```

See Also: NumCompts(I-14.31 pg.192)

### I-3.33 concat

**syntax**

```plaintext
concat(L_1: LIST,...,L_n: LIST): LIST
```

**Description**

This function returns the list obtained by concatenating the lists “L_1,...,L_n”.

NOTE: to concatenate strings just use “+”.
**/ concat([[1,2,3],[4,5],[],[]]);
[1, 2, 3, 4, 5, 6]

\[\begin{array}{c}
0 & A \\
B & 0 \\
\end{array}\]

\[\begin{array}{c}
0, 0, 1, 2, 3, \\
0, 0, 4, 5, 6, \\
101, 102, 0, 0, 0, \\
103, 104, 0, 0, 0
\end{array}\]

See Also: append(I-1.11 pg.29), ConcatLists(I-3.38 pg.57), String Operations(III-4.2 pg.335)

---

**/ A := mat([[1,2,3], [4,5,6]]);
**/ B := mat([[101,102], [103,104]]);
**/ ConcatAntiDiag(A, B);
matrix([0, 0, 1, 2, 3],
[0, 0, 4, 5, 6],
[101, 102, 0, 0, 0],
[103, 104, 0, 0, 0])

See Also: BlockMat(I-2.8 pg.37), ConcatDiag(I-3.35 pg.55), ConcatHor(I-3.36 pg.56), ConcatVer(I-3.39 pg.57)

---

**/ A := mat([[1,2,3], [4,5,6]]);
**/ B := mat([[101,102], [103,104]]);
**/ ConcatDiag(A, B);
matrix([1, 2, 3, 0, 0],
[4, 5, 6, 0, 0],
[0, 0, 0, 101, 102],
[0, 0, 0, 103, 104])

See Also: BlockMat(I-2.8 pg.37), ConcatDiag(I-3.35 pg.55), ConcatHor(I-3.36 pg.56), ConcatVer(I-3.39 pg.57)
I-3.36  ConcatHor

**syntax**

ConcatHor(A: MAT, B: MAT): MAT

where A and B have the same number of rows

**Description**

This function creates a simple block matrix. The two entries are matrices with the same number of rows. `ConcatHor(A, B)` will return a matrix of the form

\[
\begin{bmatrix}
A & B \\
\end{bmatrix}
\]

**example**

```plaintext
/**/ A := mat([[1,2,3], [4,5,6]]);
/**/ B := mat([[101,102], [103,104]]);
/**/ ConcatHor(A, B);
matrix([[1, 2, 3, 101, 102],
        [4, 5, 6, 103, 104]])
```

**See Also:** BlockMat(I-2.8 pg.37), ConcatAntiDiag(I-3.34 pg.55), ConcatHor(I-3.36 pg.56), ConcatVer(I-3.39 pg.57), DiagMat(I-4.15 pg.71)

---

I-3.37  ConcatHorList

**syntax**

ConcatHorList(L: LIST of MAT): MAT

where the matrices in L have the same number of rows

**Description**

This function creates a simple block matrix. The entries in the list are matrices with the same number of rows. `ConcatHorList(L)` will return a matrix of the form

\[
\begin{bmatrix}
\end{bmatrix}
\]

**example**

```plaintext
/**/ L := [ mat([[1,2,3], [4,5,6]]), mat([[101,102], [103,104]]) ];
/**/ ConcatHorList(L);
matrix([[1, 2, 3, 101, 102],
        [4, 5, 6, 103, 104]])
```

**See Also:** BlockMat(I-2.8 pg.37), MakeMatByRows, MakeMatByCols(I-13.2 pg.167), ConcatAntiDiag(I-3.34 pg.55), ConcatDiag(I-3.35 pg.55), ConcatHorList(I-3.37 pg.56), ConcatVer(I-3.39 pg.57), RowMat(I-18.51 pg.243)
I-3.38 ConcatLists

**syntax**

\[
\text{ConcatLists}(L: \text{LIST of LISTs}): \text{LIST}
\]

**Description**

This function takes 1 argument, a list whose entries are lists, and returns the concatenation of its entries.

\[
/**/ L := \{[[1, 2], ["abc", "def"], [3, 4]]\};
/**/ ConcatLists(L);
[[1, 2, "abc", "def", 3, 4]]
\]

**See Also:** BlockMat(I-2.8 pg.37), MakeMatByRows, MakeMatByCols(I-13.2 pg.167), ConcatAntiDiag(I-3.34 pg.55), ConcatDiag(I-3.35 pg.55), ConcatHor(I-3.36 pg.56), ConcatVerList(I-3.40 pg.58), RowMat(I-18.51 pg.243)

I-3.39 ConcatVer

**syntax**

\[
\text{ConcatVer}(A: \text{MAT}, B: \text{MAT}): \text{MAT}
\]

where \(A\) and \(B\) have the same number of columns

**Description**

This function creates a simple block matrix. The two entries are matrices with the same number of columns. \(\text{ConcatVer}(A, B)\) will return a matrix of the form

\[
\begin{bmatrix}
A \\
B
\end{bmatrix}
\]

\[
/**/ A := \text{mat}([[1, 2, 3], [4, 5, 6]]);
/**/ B := \text{mat}([[101, 102, 103]]);
/**/ ConcatVer(A, B);
\text{matrix}([[1, 2, 3], [4, 5, 6], [101, 102, 103]])
\]

**See Also:** BlockMat(I-2.8 pg.37), ColMat(I-3.26 pg.51), MakeMatByRows, MakeMatByCols(I-13.2 pg.167), ConcatAntiDiag(I-3.34 pg.55), ConcatDiag(I-3.35 pg.55), ConcatHor(I-3.36 pg.56), ConcatVerList(I-3.40 pg.58)
I-3.40 ConcatVerList

**syntax**

```
ConcatVerList(L: LIST of MAT): MAT
```

where the matrices in L have the same number of columns

**Description**

This function creates a simple block matrix. The entries in the list are matrices with the same number of columns. ConcatVer(L) will return a matrix of the form

```
| L[1] |
| L[2] |
| ... |
```

**example**

```
/**/ L := [ mat([[1,2,3], [4,5,6]]), mat([[101,102,103]]) ];
/**/ ConcatVerList(L);
matrix([ [1,2,3], [4,5,6], [101,102,103] ])
```

**See Also:** BlockMat(I-2.8 pg.37), ColMat(I-3.26 pg.51), MakeMatByRows, MakeMatByCols(I-13.2 pg.167), ConcatAntiDiag(I-3.34 pg.55), ConcatDiag(I-3.35 pg.55), ConcatVer(I-3.39 pg.57), ConcatHorList(I-3.37 pg.56)

I-3.41 content

**syntax**

```
content(F: RINGELEM): RINGELEM
```

**Description**

This function returns the content of F (i.e. a gcd of its coefficients).

The returned value is a RingElem in RingOf(F).

**example**

```
/**/ Use R ::= QQ[x,y,z];
/**/ F := 1234x^3z + 3456xyz^3 + 5678y^2z;
/**/ content(F);
2
/**/ RingOf(It);
QQ
```

**See Also:** ContentWRT(I-3.43 pg.59), coefficients(I-3.21 pg.48)

I-3.42 ContentFreeFactor

**syntax**

```
ContentFreeFactor(F: RINGELEM): RECORD
```
Description

This function returns a factorization of the multivariate polynomial \( F \) into content-free factors; it works by calling ContentWRT repeatedly. The multiplicities will always be 1.

\[
/**\ Use R := QQ[x,y,z];
/**\ F := 2*(x+1)*(y+2)*(x+y);
/**\ indent(ContentFreeFactor(F));
record[
    RemainingFactor := 2,
    factors := [y +2, x +1, x +y],
    multiplicities := [1, 1, 1]
]
\]

See Also: ContentWRT(I-3.43 pg.59), factor(I-6.1 pg.83), SqFreeFactor(I-19.25 pg.257)

I-3.43 ContentWRT

\[
\text{syntax}
\]

\[
\text{ContentWRT}(F: \text{RINGELEM}, X: \text{RINGELEM}): \text{RINGELEM}
\]
\[
\text{ContentWRT}(F: \text{RINGELEM}, L: \text{LIST of RINGELEM}): \text{RINGELEM}
\]

Description

This function returns the content of \( F \) (i.e. a gcd of its coefficients) seen as a polynomial in \( X \), indet or list of indeterminates. The returned value is a RingElem in RingOf(F).

\[
/**\ Use R := QQ[x,y,z];
/**\ F := x^3*z + x*y*z^3 + 2*z;
/**\ Cx := CoefficientsWRT(F, x);
/**\ indent(Cx);
[  record[PP := 1, coeff := 2*z],
  record[PP := x, coeff := y*z^3],
  record[PP := x^3, coeff := z]
]
/**\ ContentWRT(F, x);
  z
/**\ ContentWRT(F, [x]);
  z
\]

See Also: CoefficientsWRT(I-3.22 pg.49), content(I-3.41 pg.58), monomials(I-13.26 pg.177)

I-3.44 ContFrac

\[
\text{syntax}
\]

\[
\text{ContFrac}(X: \text{RAT}): \text{LIST of INT}
\]

Description

“ContFrac” returns a list of the continued fraction “quotients” for the given rational number “\( X \).”
**ContFracToRat**

**syntax**

ContFracToRat(L: LIST of INT): RAT

**Description**

“ContFracToRat” returns the rational number equal to the continued fraction whose quotients are given as input. The quotients must all be integers, only the very first may be non-positive.

**example**

```plaintext
/**/ ContFracToRat([1, 2, 2, 2, 2, 2, 2, 2, 1, 1, 4, 1, 1, 1, 1, 1, 2, 1, 6]);
577/408
```

See Also: ContFrac(I-3.44 pg.59), CFApprox(I-3.7 pg.43), CFApproximants(I-3.8 pg.44)

**count**

**syntax**

count(L: LIST, E: OBJECT): INT

**Description**

This function counts the number of occurrences of the object E in the list L.

**example**

```plaintext
/**/ L := [1,2,3,2,[2,3]]; 
/**/ count(L,2);
2
/**/ count(L,[2,3]);
1
/**/ count(L,"a");
0
```

See Also: distrib(I-4.19 pg.73), len(I-12.5 pg.157)

**CpuTime**

**syntax**

CpuTime(): RAT
**Description**

This function returns a “RAT” whose value is the user CPU usage in seconds since the start of the program: this is the amount of time the processor has dedicated to your computation, and may be rather less than the real elapsed time if the computer is also busy with other tasks.

The most common usage is with “TimeFrom” (I-20.6 pg.273) as shown in the example.

```coconut
/**/ StartTime := CpuTime(); -- time in seconds since the start (a RAT)
/**/ --
/**/ -- .... long computation ....
/**/ --
/**/ PrintLn "Computation time: ", TimeFrom(StartTime);
```

You can use “DecimalStr” (I-4.3 pg.63) to see the value of “CpuTime” in a more easily comprehensible form.

**See Also:** TimeFrom(I-20.6 pg.273), DecimalStr(I-4.3 pg.63)

---

**I-3.48 CRT**

**syntax**

```coconut
CRT(R1: INT, M1: INT, R2: INT, M2: INT): RECORD
```

**Description**

This function combines two residue-modulus pairs “(R1,M1)” and “(R2,M2)” using the Chinese Remainder Theorem to produce a single residue-modulus pair “(R,M)” such that “R = R1 mod M1” and “R = R2 mod M2”, and “|R| < M”. The moduli “M1” and “M2” must be coprime (hence “M = M1*M2”).

```coconut
/**/ CRT(2,3,4,5);
record[modulus := 15, residue := -1]
```

**See Also:** RatReconstructByContFrac, RatReconstructByLattice(I-18.14 pg.226)

---

**I-3.49 CurrentRing**

**syntax**

```coconut
CurrentRing
```

**Description**

This is a top-level SYSTEM VARIABLE containing the current ring.

NOTE: in CoCoA-4 it used to be a function (namely “CurrentRing()”), now it is a top-level “variable” which needs to be imported in functions... but beware: this is to be considered BAD STYLE ;-)
### I-3.50 CurrentTypes

**syntax**

```plaintext
CurrentTypes(): LIST of TYPE
```

**Description**

This function lists all CoCoA data types.

```plaintext
/**/ CurrentTypes();
[BOOL, ERROR, FUNCTION, ...]
```

### I-3.51 cyclotomic

**syntax**

```plaintext
cyclotomic(n: INT, x: RINGELEM): RINGELEM
```

**Description**

This function computes the n-th cyclotomic polynomial (in the indeterminate x).

```plaintext
/**/ Use QQ[z];
/**/ cyclotomic(4,z);
z^2 + 1
```
Chapter I-4

D

I-4.1 dashes

```plaintext
syntax

dashes()
```

Description

This function returns a string of dashes:

```plaintext
example

/**/ dashes(); 1+1;
-------------------------------
2
```

I-4.2 date

```plaintext
syntax

date() : INT
```

Description

This function returns the date.

Note that from version 5.0.4 the result is an INT and the date is in the form YYYYMMDD. See also “TimeOfDay” (I-20.7 pg.273).

```plaintext
example

/**/ date();
20130530
```

See Also: TimeOfDay(I-20.7 pg.273)

I-4.3 DecimalStr

```plaintext
syntax

DecimalStr(X: INT|RAT|RINGELEM): STRING
DecimalStr(X: INT|RAT|RINGELEM, NumDigits: INT): STRING
```
**Description**

This function produces a decimal string representation of a rational number with up to “NumDigits” digits after the decimal point. If not specified, the default number of digits is 3.

If “X” is a “RINGELEM”, it is automatically converted to a “RAT”.

```
/**/ DecimalStr(1/3);
0.333
/**/ DecimalStr(1/3, 60);  
0.333333333333333333333333333333333333333333333333333333333333
/**/ DecimalStr(123.456789);
123.457
```

**See Also:** FloatStr(I-6.11 pg.87), ScientificStr(I-19.3 pg.246), MantissaAndExponent10(I-13.4 pg.168)

### I-4.4 define

**syntax**

```
Define F(X_1, .., X_n) C EndDefine
Define F(X_1, .., opt X_n) C EndDefine
Define F(...) C EndDefine
return FUNCTION
```

**Description**

1. **INTRODUCTION.** This command adds the user-defined function F to the library. The function F can be called in the following way:

   \[ F(E_1, \ldots, E_n) \]

   where the “E_i”’s are expressions. The result of the evaluation of each expression “E_i” is assigned to the respective formal parameter “X_i”, and the command sequence “C” is executed. If, during the execution of “C”, a statement “Return E” is executed, then the result of the evaluation of “E” is the return-value of the function “F”. If no “Return” command is executed, or “Return” is executed without argument, then the return-value is “Null”.

```
/**/ Define square(X)
/**/ Return X^2;
/**/ EndDefine;
/**/ square(5);
25
```

or a variable number of arguments

```
/**/ define strange(...) --> all args are in the list ARGV
/**/ if len(ARGV) = 0 then return 1234; endif;
/**/ if len(ARGV) > 4 then return last(ARGV)^2; endif;
/**/ error("Wrong number for arguments for \"strange\"!");
/**/ endif;
 /**/ strange();
```
2. SCOPE. Every variable defined or modified by the command sequence “C” is considered local to the function unless the variable is global or relative to a “ref” parameter. See “TopLevel” (I-20.9 pg.274) for the use of global variables. See “ref” to learn about calling a function “by reference”, i.e. so that the function can change the value of an existing variable.

```
/**/ Define Example_1(L)
/**/ L := L + 5;
/**/ Return L;
/**/ EndDefine;

/**/ L := 0;
/**/ Example_1(L);
5
/**/ L; -- L is unchanged despite the function call.
0
```

3. VARIABLE NUMBER OF PARAMETERS. It is also possible to have some optional arguments or a variable number of arguments:

```
-- OPTIONAL ARGUMENTS must be in the last positions

/**/ define deg0(f, opt x)
/**/ if f=0 then return 0; endif;
/**/ if IsDefined(x) then return deg(f,x); endif;
/**/ return deg(f);
/**/ enddefine;

/**/ use P ::= QQ[x,y,z];
/**/ deg0(zero(P));
0
/**/ deg0(x^2+y);
2
/**/ deg0(x^2+y, y);
1

-- VARIABLE number of ARGUMENTS

/**/ Define MySum(...) --> arguments are in the LIST "ARGV"
/**/ If len(ARGV) = 0 Then Return 12345;
/**/ Else
/**/ ans := 0;
/**/ Foreach N In ARGV Do ans := ans+N; EndForeach;
/**/ EndIf;
/**/ Return ans;
/**/ EndDefine;

/**/ MySum(1,2,3,4,5);
15
/**/ MySum();
12345
```
The old statement, “Help $S$;” is OBSOLETE!

See Also: return(I-18.35 pg.236), TopLevel(I-20.9 pg.274), ref(I-18.23 pg.231)

### I-4.5 DefiningIdeal

**Syntax**

```
DefiningIdeal(S: RING): IDEAL
```

**Description**

When “$S$” is a quotient ring, say “$S = R/I$”, this function returns “$T$”, the ideal which defines “$S$”.

**Example**

```
/**/ Use R ::= QQ[x,y,z];
/**/ S := R/ideal(x);
/**/ DefiningIdeal(S);
ideal(x)
```

See Also: CanonicalRepr(I-3.3 pg.42), InducedHom(I-9.24 pg.127), NewQuotientRing(I-14.9 pg.182)

### I-4.6 deg

**Syntax**

```
deg(F: RINGELEM): INT
deg(F: RINGELEM, X: RINGELEM): INT
```

**Description**

The first form of this function returns the “standard degree” of “F” (see “wdeg” (I-23.1 pg.285) for the “weighted degree”). The second form returns the exponent of the indeterminate “X” in “F”.

For the degree of a ring or quotient, see “multiplicity” (I-13.28 pg.178).

**Example**

```
/**/ Use R ::= QQ[x,y,z];
/**/ deg(x*y^2+y); 3
/**/ deg(x*y^2+y, x); 1

/**/ Ws := RowMat([2,3,1]);
/**/ P := NewPolyRing(QQ, "x,y,z", CompleteToOrd(Ws), 1);
/**/ Use P;
/**/ deg(x*y^2+y); 3
/**/ wdeg(x*y^2+y); [8]
/**/ deg(x*y^2+y, x); 1
/**/ deg(x*y^2+y, x); 2
```

See Also: wdeg(I-23.1 pg.285), NewPolyRing(I-14.8 pg.182), multiplicity(I-13.28 pg.178)
I-4.7  den

den(X: INT|RAT): INT
den(X: RINGELEM): RINGELEM

Description

These function returns the denominator of the argument “X”. If “X” is a “RINGELEM” in “FractionField(R)”, then “den(X)” is a RingElem in “R”.

NOTE: In CoCoA 4 the numerator and denominator could also be found using the suffixes “.Num” and “.Den”; this fragile syntax is now obsolete.

example

/**/ den(3);
1
/**/ P ::= QQ[x,y];
/**/ F := NewFractionField(P);
/**/ Use F;
/**/ den(x/(x+y));
x +y
/**/ RingOf(It);
RingDistrMPolyClean(QQ, 2)

See Also:  num(I-14.29 pg.191)

I-4.8  DensePoly

DensePoly(R: RING, N: INT): RINGELEM

Description

This function returns the sum of all power-products of (standard) degree N.

example

/**/ Use R ::= QQ[x,y];
/**/ DensePoly(R,3);
x^3 + x^2*y + x*y^2 + y^3
/**/ Ws := RowMat([2,3]);
/**/ P := NewPolyRing(QQ, "x,y", CompleteToOrd(Ws), 1);
/**/ Use P;
/**/ DensePoly(P,1); // NOTE: standard degree!!
y +x

See Also:  randomize(I-18.4 pg.222), randomized(I-18.5 pg.223)

I-4.9  Depth

Depth(I: IDEAL, M: TAGGED("Quotient")): INT
Depth(M: TAGGED("Quotient")): INT
Description

***** NOT YET IMPLEMENTED *****

This function calculates the depth of M in the ideal I, i.e. the length of a maximal I-regular sequence in M. In the second form, where I is not specified, it assumes that I is the maximal ideal generated by the indeterminates, i.e. “ideal(Indets())”.

Note that if M is homogeneous and I is the maximal ideal, then it uses the Auslander-Buchsbaum formula “\( \text{depth}_I(M) = N - \text{pd}(M) \)” where N is the number of indeterminates and pd is the projective dimension, otherwise it returns “\( \min\{|N | \text{Ext}^N(R/I, M) \neq 0\} \)” using the function “\( \text{Ext} \)” (I-5.12 pg.80).

```
Use R ::= QQ[x,y,z];
Depth(R/ideal(0)); -- the (x,y,z)-depth of the entire ring is 3
3
-----------------------------
I := ideal(x^5,y^3,z^2);
-- one can check that it is zerodimensional and CM this way
dim(R/I);
0
-----------------------------
Depth(R/I);
0
-----------------------------
N := Module([x^2,y], [x+z,0]);
Depth(I, R^2/N); --- a max reg sequence would be (z^2,y^3)
2
-----------------------------
Use R ::= QQ[x,y,z,t,u,v];
-- Cauchy-Riemann system in three complex vars!
N := Module([x,y], [-y,x], [z,t], [-t,z], [u,v], [-v,u]);
--- is it CM?
Depth(R^2/N);
3
-----------------------------
dim(R^2/N);
3
-----------------------------
--- yes!
M := Module([x,y,z],[t,v,u]);
Res(R^3/M);
0 --> R^2(-1) --> R^3
-----------------------------
Depth(R^3/M); -- using Auslander Buchsbaum 6-1=5
5
-----------------------------
dim(R^3/M); -- not CM
6
-----------------------------
Depth(ideal(x,y,z,t), R^2/N);
2
-----------------------------
```

See Also: res(I-18.31 pg.235), Ext(I-5.12 pg.80)
I-4.10  deriv

<table>
<thead>
<tr>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>deriv(F: RINGELEM, X: RINGELEM): RINGELEM</td>
</tr>
</tbody>
</table>

**Description**

This function returns the derivative of F with respect to the indeterminate X.

```plaintext
/**/ Use R ::= QQ[x,y];
/**/ deriv(x*y^2, x);
y^2
/**/ Define Jac(F) --> The Jacobian matrix for a polynomial.
/**/ Return matrix([[deriv(F, X) | X In Indets(RingOf(F))]]);
/**/ EndDefine;
/**/ Jac(x*y^2);
matrix( /*RingDistrMPolyClean(QQ, 2)*/
  [[y^2, 2*x*y]])
/**/ FrF := NewFractionField(R);
/**/ Use FrF;
/**/ deriv((x*y^2)/(x-1), x);
(-y^2)/(x^2 -2*x +1)
```

**See Also:** jacobian(I-10.1 pg.151)

I-4.11  DerivationAction

<table>
<thead>
<tr>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>DerivationAction(D: RINGELEM, P: RINGELEM)</td>
</tr>
</tbody>
</table>

**Description**

Thanks to Enrico Carlini.

Given the polynomial “P” and the derivation “D”, this function computes the action of “D” on “P”.

For the sake of simplicity Forms/Polynomials and Derivations live in the same ring, the distinction between them is purely formal.

```plaintext
/**/ Use R ::= QQ[x,y,z];
/**/ DerivationAction(x*y*z, x^3+x*y*z); 1
```

**See Also:** InverseSystem(I-9.32 pg.131), PerpIdealOfForm(I-16.6 pg.202)

I-4.12  describe

<table>
<thead>
<tr>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>describe X: OBJECT</td>
</tr>
</tbody>
</table>
Description

This command gives some information about the object “X”. For instance, if “X” is a CoCoA-5 function, it prints out the definition, and if “X” is a package name (prefixed with a “$”), it prints out the exported names.

```coconut
/**/ Define succ(N) Return N+1; EndDefine;
/**/ describe succ;
Define succ(N) Return N+1 EndDefine

/**/ describe $chebyshev;
The package $chebyshev exports the following names:
* ChebyshevPoly
* ChebyshevPoly2
```

I-4.13 det

```coconut
syntax
det(M: MAT): RINGELEM
```

Description

This function returns the determinant of the matrix “M”.

```coconut
/**/ Use R ::= QQ[x];
/**/ M := mat(R,[[x,x^2], [x,x^3]]);
/**/ det(M);
x^4 -x^3
/**/ det(mat(QQ,[[1,2], [0,5]]));
5
```

See Also: minors(I-13.17 pg.174)

I-4.14 DF

```coconut
syntax
DF(F: RINGELEM): RINGELEM
```

Description

Same as “LF” (I-12.8 pg.158), but does not throw an error if the argument is zero or if the “GradingDim” (I-7.18 pg.103) of the polynomial ring is 0. As defined in Kreuzer-Robbiano book II (Definition 4.2.8).

```coconut
/**/ Use R ::= QQ[x,y];
/**/ DF(x^2 -x*y +2*x -1);
x^2 -x*y
/**/ Use R ::= QQ[x,y], Lex;  -- GradingDim is 0: everything is homogeneous
/**/ DF(x^2 -x*y +2*x -1);
x^2 -x*y +2*x -1
/**/ P := NewPolyRing(QQ, IndetSymbols(R), mat([[1,4],[1,0]]), 1);
/**/ Use P;
```
DiagMat

I-4.15  

**/ DF(x^2 -x*y);
-x*y
**/ DF(x^4 +x^2 -y);
 x^4 -y

See Also:  LF(I-12.8 pg.158)

I-4.15  DiagMat

**/ DiagMat(L: LIST): MAT
DiagMat(R: RING, L: LIST): MAT

Description
This function returns the diagonal matrix whose diagonal are the elements of the list L.

/**/ DiagMat([3,4,5]);
matrix(
  [3, 0, 0],
  [0, 4, 0],
  [0, 0, 5]
)
/**/ DiagMat(QQ,[5,6,7]);
matrix(
  [5, 0, 0],
  [0, 6, 0],
  [0, 0, 7]
)

-- fast implementation for high powers of a diagonal matrix
/**/ Define PowerDiag(M, Exp)
/**/ If not(IsDiagonal(M)) Then
/**/ error("PowerDiag: matrix must be diagonal");
/**/ EndIf;
/**/ Return DiagMat([ M[I, I]^Exp | I In 1..NumRows(M) ]);
/**/ EndDefine;
/**/ PowerDiag(IdentityMat(QQ,3), 200000000);
matrix(QQ,
  [[1, 0, 0],
   [0, 1, 0],
   [0, 0, 1]])

See Also:  BlockMat(I-2.8 pg.37), IsDiagonal(I-9.39 pg.133), ColMat(I-3.26 pg.51), RowMat(I-18.51 pg.243)

I-4.16  diff

diff(L: LIST, M: LIST): LIST
Description

This function returns the list obtained by removing all the elements of M from L.

```plaintext
/**/ L := [1,2,3,2,[2,3]];  
/**/ M := [1,2];  
/**/ diff(L, M);  
[3, [2, 3]]
```

See Also: remove(I-18.29 pg.234)

I-4.17  dim

```plaintext
dim(R: RING or TAGGED("Quotient")): INT
```

Description

This function computes the dimension of R.

The coefficient ring must be a field.

```plaintext
/**/ Use R ::= QQ[x,y,z];  
/**/ dim(R/ideal(x));  
2  
/**/ dim(R/ideal(y^2-x, x*z-y^3));  
1
```

I-4.18  discriminant

```plaintext
discriminant(F: RINGELEM): RINGELEM  
discriminant(F: RINGELEM, X: RINGELEM): RINGELEM
```

Description

This function computes the discriminant of a polynomial F (with respect to a given indeterminate X, if the polynomial is multivariate). If the polynomial is univariate then there is no need to specify which indeterminate to use.

The discriminant is defined as

\((-1)^{(N*(N-1)/2)}*\det(M)/M[1,1]\)

where \(M := \text{Sylvester}(F, \text{deriv}(F, X), X)\) and \(N := \deg(F, X)\).

```plaintext
/**/ Use R ::= QQ[x,y];  
/**/ discriminant(x^2+3*y^2, x);  
-12*y^2  
/**/ discriminant(x^2+3*y^2, y);  
-12*x^2
```
See Also: resultant(I-18.34 pg.236)

I-4.19 distrib

**/ discriminant((x+1)^20+2);
54975581388800000000000000000000

See Also: count(I-3.46 pg.60)

I-4.20 div

**/ div(10,3);
3
**/ mod(10,3);
1

See Also: DivAlg(I-4.21 pg.73), GenRepr(I-7.7 pg.98), NF(I-14.14 pg.184), NR(I-14.28 pg.190), mod(I-13.21 pg.175)

I-4.21 DivAlg
Description

This function performs the division algorithm on X with respect to L. It returns a record with two fields: “Quotients” holding a list of polynomials, and “Remainder” holding the remainder of X upon division by L.

```plaintext
/**/ Use R := QQ[x,y,z];
/**/ F := x^2*y +x*y^2 +y^-2;
/**/ L := [x*y-1, y^-2-1];
/**/ DivAlg(F, L);
record[quotients := [x +y, 1], remainder := x +y +1]
/**/ D := It;
/**/ D.quotients;
[x +y, 1]
/**/ D.remainder;
x +y + 1
/**/ ScalarProduct(D.quotients, L) + D.remainder = F;
true
/**/ R2 := NewFreeModule(R,2);
/**/ V := ModuleElem(R2, [x^-2+y^-2+z^-2, x*y*z]);
/**/ L := gens(SubmoduleRows(R2, mat([[x,y], [y,z], [z,x]])));
/**/ D := DivAlg(V, L);
/**/ indent(D);
record[
    quotients := [x, -z^2 +y +z, y*z -y],
    remainder := [z^2, z^3 -y*z -z^2]
]
/**/ sum([D.quotients[i]*L[i] | i in 1..len(L)]) + D.remainder;
[x^-2 +y^-2 +z^-2, x*y*z]
```

See Also: div(I-4.20 pg.73), mod(I-13.21 pg.175), GenRepr(I-7.7 pg.98), NF(I-14.14 pg.184), NR(I-14.28 pg.190)

I-4.22 domain

** syntax **

```
domain(\phi: RINGHOM): RING
```

** Description **

This function returns the domain of the homomorphism “\phi”

```plaintext
/**/ P := NewPolyRing(RingQQ(), "alpha,beta");
/**/ \phi := CanonicalHom(RingZZ(), P);
/**/ domain(\phi);
ZZ
/**/ psi := CoeffEmbeddingHom(P);
/**/ domain(psi);
QQ
```

See Also: codomain(I-3.19 pg.47), Commands and Functions for RINGHOM(III-10.3 pg.360), Commands and Functions returning RINGHOM(III-10.4 pg.360)
Chapter I-5

E

I-5.1 E_ [OBsolete]

**Syntax**

[OBsolete]

**Description**

[OBsolete] Essentially replaced by “gens” (I-7.8 pg.98) of a FreeModule.

**Example**

```plaintext
/**/ Use R ::= QQ[x,y,z];
/**/ R5 := NewFreeModule(R,5);
/**/ e := gens(R5);
/**/ e[2];
[0, 1, 0, 0, 0]
```

**See Also:** gens(I-7.8 pg.98), GensAsCols, GensAsRows(I-7.9 pg.99)

I-5.2 eigenvectors

**Syntax**

eigenvectors(M: MAT, X: RINGELEM): LIST of RECORD

**Description**

“M” must be a matrix of numbers, and “X” an indeterminate.

This function determines the eigenvalues of “M”, and for each eigenvalue gives a basis of the corresponding eigenspace – note that the basis is probably not orthogonal. For irrational eigenvalues, the minimal polynomial of the eigenvalue is given (as a polynomial in “X”), along with the eigenvectors expressed in terms of a root of the minimal polynomial (represented as “X”).

**Example**

```plaintext
/**/ Use R ::= QQ[x];
/**/ M := mat([[1,2,3],[4,5,6],[7,8,9]]);
/**/ eigenvectors(M, x);
[record[MinPoly := x, eigenspace := matrix(QQ,
[[[-1],
[2],
```
** Chapter I-5. E **

I-5.3 elim

### Syntax

- `elim(X: RINGELEM, M: IDEAL): IDEAL`
- `elim(L: LIST, M: IDEAL): IDEAL`
- `elim(X: RINGELEM, M: MODULE): MODULE`
- `elim(L: LIST, M: MODULE): MODULE`

### Description

This function returns the ideal or module obtained by eliminating the indeterminate “X”, or all indeterminates in “L”, from “M”. The coefficient ring needs to be a field.

As opposed to this function, there is also the “modifier”, “elim”, used when constructing a ring (see “Orderings” (III-9.5 pg.354)).

### Example

```plaintext
/**/ Use R ::= QQ[t,x,y,z];
/**/ E := elim(t, ideal(t^15+t^6+t-x, t^5-y, t^3-z));
/**/ indent(E);
ideal(  
-z^5 +y^3,  
-y^4 -y*z^2 +x*y -z^2,  
-x*y^3*z -y^2*z^3 -x*z^3 +x^2*z -y^2 -y,  
-y^2*z^4 -x^2*y^3 -x*y^2*z^2 -y*z^4 -x^2*z^2 +x^3 -y^2*z -2*y*z -z,  
y^3*z^3 -x*z^3 +y^3 +y^2
)
/**/ Use R ::= QQ[t,s,x,y,z,w];
/**/ t..x;
[t, s, x]
/**/ elim(t..x, ideal(t-x^2*z*w, x^2-t, y^2-t-w)); -- Note the use of t..x.
ideal(-z*w^2 + w)
/**/ Use R ::= QQ[t[1..2], x[1..4]];
/**/ elim(indets(R,"t"), I);
```

### See Also

Orderings (III-9.5 pg.354)
I-5.4  ElimMat

**syntax**

ElimMat(N: INT, ElimInd: LIST of INT): MAT
ElimMat(W: MAT, ElimInd: LIST of INT): MAT

**Description**

This function returns an “N x N” matrix representing a term ordering for eliminating the indeterminates with indices in “ElimInd”.

If a weight matrix is given, then these weights are included after the first elimination row.

```plaintext
/**/ ElimMat(3, [2,3]);
matrix(QQ,
[[0, 1, 1],
[1, 1, 1],
[0, 0, -1]])

/**/ ElimMat(mat([[1,5,0]]), [2,3]);
matrix(QQ,
[[0, 1, 1],
[1, 5, 0],
[0, 0, -1]])
```

**See Also:** elim(I-5.3 pg.76), HomogElimMat(I-8.11 pg.111)

I-5.5  EmbeddingHom

**syntax**

EmbeddingHom(K: RING): RINGHOM

**Description**

This function returns the embedding homomorphism of the fraction field “K”.

```plaintext
/**/ Use P ::= QQ[x,y];
/**/ K := NewFractionField(P);
/**/ phi := EmbeddingHom(K); -- phi: P -> K
/**/ f := 2*x+3*y;
/**/ phi(f);
2*x +3*y
/**/ RingOf(phi(f));
RingWithID(5, "FractionField(RingWithID(4))")
```

**See Also:** CanonicalHom(I-3.2 pg.41)

I-5.6  EqSet

**syntax**

EqSet(L: LIST, M: LIST): BOOL
Chapter I-5. E

Description

This function returns true if “L” equals “M” as sets, otherwise it returns false.

```plaintext
/**/ L := [1,2,2];
/**/ M := [2,1];
/**/ EqSet(L, M);
true
```

See Also: intersection(I-9.29 pg.129), IntersectList(I-9.30 pg.130), IsSubset(I-9.65 pg.143)

I-5.7 Equality Test

Description

The first form returns “true” if “A” is equal to “B”, otherwise it returns “false” (or signals an error if they are not comparable). The second form is the same as “not(A=B)”.

```plaintext
/**/ 1=2;
false
/**/ 1<>2;
true
```

See Also: Comparison Operators(I-3.30 pg.53), operators, shortcuts(I-0.1 pg.23)

I-5.8 EquiIsoDec

Description

***** NOT YET IMPLEMENTED *****

This function computes an equidimensional isoradical decomposition of I, i.e. a list of unmixed ideals \(I_1, \ldots, I_k\) such that the radical of I is the intersection of the radicals of \(I_1, \ldots, I_k\). Redundancies are possible.

NOTE: at the moment, this implementation works only if the coefficient ring is the rationals or has large enough characteristic.

```plaintext
Use R ::= QQ[x,y,z];
I := intersect(ideal(x-1,y-1,z-1), ideal(x-2,y-2)^2, ideal(x)^3);
H := EquiIsoDec(I);
H;
[ideal(x), ideal(z - 1, y - 1, x - 1), ideal(xy - y^2 - 2x + 2y, x^2 - y^2 - 4x + 4y, y^2z - y^2 - 4yz + 4y + 4z - 4, y^3 - 5y^2 + 8y - 4, x - 2)]
--------------------------------------
T := [radical(J)|J In H];
```
S := IntersectionList(T);
radical(I) = S;
True
------------------------------

See Also: PrimaryDecomposition(I-16.19 pg.208), radical(I-18.1 pg.221), RadicalOfUnmixed(I-18.2 pg.221)

I-5.9 error

<table>
<thead>
<tr>
<th>syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>error(S: STRING): ERROR</td>
</tr>
</tbody>
</table>

Description

This function throws an error containing the given message. For backward compatibility the function may also be called using the name "Error"

```plaintext
/**/ Define T(N)
/**/ If type(N) <> INT Then error("Argument must be an integer."); EndIf;
/**/ Return mod(N,5);
/**/ EndDefine;
-- /**/ T(1/3); --> !!! ERROR !!!
ERROR: Argument must be an integer.
        If type(N) <> INT Then error("Argument must be an integer."); EndIf;
        ----------------------------------
CONTEXT: function T (previously defined at the prompt)
called at top-level
/**/ T(7);
2
```

See Also: try(I-20.13 pg.277), GetErrMesg(I-7.14 pg.101)

I-5.10 eval

<table>
<thead>
<tr>
<th>syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>eval(E: RINGELEM</td>
</tr>
</tbody>
</table>

Description

This function substitutes “L[I]” for “indet(I)” in the expression “E” which must be of type POLY, MODULEELEM, LIST, or MAT. The evaluation takes place in the ring of “E”. For more general substitutions use “subst” (I-19.39 pg.264).

If “len(L)” is different from “NumIndets()” then only the first N substitutions are performed, where N is the minimum of the two values.

```plaintext
/**/ Use QQ[x,y];
/**/ eval(x^2+y, [2, 3]);
7
/**/ eval(x^2+y, [2]);
```
\[ F := x(x-1)(x-2)y(y-1)(y-2)/36; \]
\[ P := [1/2, -2/3]; \]
\[ \text{eval}(F, P); \]
\[ -5/162 \]
\[ \text{eval}([x+y, x-y], [2,1]); \]
\[ [3, 1] \]
\[ \text{eval}([x+y, x-y], [x^2, y^2]); \]
\[ [x^2 + y^2, x^2 - y^2] \]
\[ \text{eval}([x+y, x-y], [y]); \]
\[ [2*y, 0] \]

See Also: Evaluation of Polynomials(I-11.2 pg.362), subst(I-19.39 pg.264)

I-5.11 EvalHilbertFn

**syntax**

EvalHilbertFn(H:TAGGED("hp.Hilbert"), N: INT): INT

**Description**

This function evaluates the Hilbert function \( H \) at \( N \). If \( H \) is the Hilbert function of a quotient \( R/I \), then the value returned is the same as that returned by “HilbertFn(R/I, N)” but time is saved since the Hilbert function does not need to be recalculated at each call.

**example**

\[ \text{Use } R ::= QQ[x,y,z]; \]
\[ \text{I := ideal(z^2-x*y, x*z^2+w^3);} \]
\[ \text{H := HilbertFn(R/I);} \]
\[ \text{H(0) = 1}
\]
\[ \text{H(1) = 4}
\]
\[ \text{H(t) = 6t - 3 for } t >= 2 \]

\[ \text{EvalHilbertFn(H,1);} \]
\[ 4 \]
\[ \text{EvalHilbertFn(H,2);} \]
\[ 9 \]

See Also: HilbertFn(I-8.5 pg.107), HilbertPoly(I-8.6 pg.107)

I-5.12 Ext

**syntax**

Ext(I: INT, M:TAGGED("Quotient"), Q:TAGGED("Quotient")): TAGGED("Quotient")
Ext(I: LIST, M:TAGGED("Quotient"), Q:TAGGED("Quotient")): TAGGED("$ext.ExtList")

**Description**

***** NOT YET IMPLEMENTED *****
In the first form the function computes the I-th Ext module of M and N. It returns a presentation of $\text{Ext}^I_R(M, N)$ as a quotient of a free module.

**IMPORTANT:** the only exception to the type of M or N (or even of the output) is when they are either a zero module or a free module. In these cases their type is indeed MOD.

It computes Ext via a presentation of the quotient of the two modules $\text{Ker}(\Phi_{i+1})$ and $\text{Im}(\Phi_i)$, where
- $\Phi_i$ is the I-th map in the free resolution of M
- $\Phi_i^*$ is the map $\text{Hom}(\Phi_i, N)$ in the dual of the free resolution.

Main differences with the previous version include:
- SHIFTS have been removed, consequently only standard homogeneous modules and quotients are supported
- as a consequence of 1), the type "Tagged("Shifted")" has been removed. Ext will just be a "Tagged("Quotient")"
- The former functions Presentation(), HomPresentation() and KerPresentation() have been removed
- The algorithm uses Res() to compute the maps needed, and not SyzOfGens anylonger, believed to cause troubles
- The function “Ext” always has THREE variables, see syntax...

In the second form the variable I is a LIST of nonnegative integers. In this case the function Ext prints all the Ext modules corresponding to the integers in I. The output is of special type "Tagged("ext.ExtList")" which is basically just the list of pairs $(J, \text{Ext}^J(M, N))$ in which the first element is an integer of I and the second element is the corresponding Ext module.

**VERY IMPORTANT:** CoCoA cannot accept the ring R as one of the inputs, so if you want to calculate the module $\text{Ext}^I_R(M, R)$ you need to type something like

```
“Ext(I, M, ideal(1));”
```

or

```
“Ext(I, M, R^1);”
```

or

```
“Ext(I, M, R/ideal(0));”
```

**NOTE:** The input is pretty flexible in terms of what you can use for M and N. For example they can be zero modules or free modules. See some examples below.

```plaintext
Use R ::= QQ[x,y,z];
I := ideal(x^5, y^3, z^2);
ideal(0) : (I);
-----------------------------
$\text{hom.Hom}(R^1/\text{Module}(I), R^1); \ -- \text{from Hom package}$
Module(

Module(

Module(

$\text{Ext}(0, R/I, R^1); \ -- all \text{ those things should be isomorphic}$
Module(

Module(

$\text{Ext}(0..4, R/I, R/\text{ideal}(0)); \ -- another \text{ way to define the ring R as a quotient}$
$\text{Ext}^0 = \text{Module}([[]])$
$\text{Ext}^1 = \text{Module}([[]])$
$\text{Ext}^2 = \text{Module}([[]])$
$\text{Ext}^3 = R^1/\text{Module}([x^5], [y^3], [z^2])$
$\text{Ext}^4 = \text{Module}([[]])$
```
\begin{verbatim}
N := Module([x^2,y], [x+z,0]);
Ext(0..4, R/I, R^2/N);
Ext^0 = Module([[0]])
Ext^1 = Module([[0]])
Ext^2 = R^2/Module([[0, x + z], [y, 0], [0, z^2], [z^2, 0], [0, y^3], [x^5, 0]])
Ext^3 = R^2/Module([[x + z, 0], [0, z^2], [z^2, 0], [y^3, 0], [0, x^5], [0, y]])
Ext^4 = Module([[0]])
\end{verbatim}

Since version 4.7.3 the output modules are presented minimally.

See Also: res(I-18.31 pg.235), Depth(I-4.9 pg.67), MinimalPresentation(I-13.16 pg.173)

\section*{I-5.13 ExternalLibs}

\begin{verbatim}
ExternalLibs(): LIST of STRING

Description
This function returns the list of the names of the linked libraries.

example
/**/ ExternalLibs();
["BOOST", "FROBBY", "GSL", "NORMALIZ"]

See Also: Frobby(II-8.4 pg.319), Normaliz(II-8.5 pg.319)
\end{verbatim}
Chapter I-6

F

I-6.1 factor

factor(F: RINGELEM): RECORD

Description

This function factorizes a polynomial into irreducibles in its ring of definition. Multivariate factorization is not yet supported over finite fields. To factorize an integer use “SmoothFactor” (I-19.16 pg.253). (For information about the algorithm, consult John Abbott’s papers)

NOTE: in older versions of CoCoA-5 the field names were “Factors” and “Exponents”.

example

/**/ Use R ::= QQ[x,y];
/**/ F := 4*x^8 + 4*x^6 + x^4 + 4*x^2 + 4;
/**/ FacInfo := factor(F);
/**/ indent(FacInfo);
record[
    RemainingFactor := 1,
    factors := [2*x^4-4*x^3+5*x^2-4*x+2, 2*x^4+4*x^3+5*x^2+4*x+2],
    multiplicities := [1, 1]
]
/**/ G := product([FacInfo.factors[i]^FacInfo.multiplicities[i] | i In 1..len(FacInfo.factors)]);
/**/ F = G * FacInfo.RemainingFactor;
true

/**/ factor((8*x^2 +16*x +8)/27);
record[factors := [x +1], multiplicities := [2], RemainingFactor := 8/27]

/**/ factor(2*x^2-4); -- over a finite field the factors are monic
record[factors := [x^2 -2], multiplicities := [1], RemainingFactor := 2]

See Also: SmoothFactor(I-19.16 pg.253), SqFreeFactor(I-19.25 pg.257), ContentFreeFactor(I-3.42 pg.58)

I-6.2 factorial

factorial(N: INT): INT
### Description

This function returns the factorial of “$N$”.  

```c
/**/ factorial(5);
120
/**/ factorial(100);
93326215443944152681699238885626670049071596826438162146859
296389521759993229915608941463976156518286253697920827223
758251185210916864000000000000000000000000
```

**See Also:** binomial(I-2.5 pg.35)

### I-6.3 FactorMultiplicity

**syntax**

```
FactorMultiplicity(b: INT, N: INT): INT
```

**Description**

This function counts how many times the base “$b$” divides a given integer “$N$”. It is an error if “$N$” is zero, or if “$b < 2$”.  

```c
/**/ FactorMultiplicity(2, 20);
2
/**/ FactorMultiplicity(5, 20);
1
/**/ FactorMultiplicity(7, 20);
0
```

**See Also:** IsDivisible(I-9.40 pg.133), SmoothFactor(I-19.16 pg.253)

### I-6.4 FGLM5

**syntax**

```
FGLM5(GBOld: LIST, M: MAT): LIST
```

**Description**

***** NOT YET IMPLEMENTED *****

This function is implemented in ApCoCoALib by Stefan Kaspar.

The function “FGLM5” calls the CoCoAServer to perform a FGLM Groebner Basis conversion. Please note that the ideal generated by the given Groebner Basis must be zero-dimensional. The Groebner Basis contained in list GBOld will be converted into a Groebner Basis with respect to term ordering “$\text{Ord}(M)$”, i.e. $M$ must be a matrix specifying a term ordering.

```c
Use QQ[x, y, z], DegRevLex;
GBOld := *** [z^4 -3z^3 - 4yz + 2z^2 - y + 2z - 2, yz^2 + 2yz - 2z^2 + 1, y^2 - 2yz + z^2 - z, x + y - z] ***;
M := LexMat(3);
GBNew := FGLM5(GBOld, M);
```
I-6.5 fields

**syntax**

`fields(R: RECORD): LIST`

**Description**

This function returns a list of all of the fields of the record “R”. It is particularly useful when you want to know if a record field has been defined.

**example**

```plaintext
/**/ rec := record[name := "David", number := 3728852, data := ["X","Y"] ];
/**/ fields(rec);
["data", "name", "number"]

/**/ rec.data;
["X", "Y"]

/**/ "surname" IsIn fields(rec);
false
```

See Also: record(I-18.20 pg.229)

I-6.6 first

**syntax**

`first(L: LIST): OBJECT`

`first(L: LIST, N: INT): OBJECT`

**Description**

In the first form the function returns the first element of the list L, same as “[L[1]]”. In the second form, it returns the list of the first N elements of L, same as “[ L[i] | i in 1..N ]”

**example**

```plaintext
/**/ L := [1,2,3,4,5];
/**/ first(L);
1

/**/ first(L,3);
[1, 2, 3]
```

See Also: last(I-12.1 pg.155)

I-6.7 FirstNonZero

**syntax**

`FirstNonZero(V: MODULEELEM): RINGELEM`
Chapter I-6. F

Description
This function returns the first non-zero entry of V. If it is handed a zero MODULEELEM then an error is signalled.

```plaintext
/**/ Use R ::= QQ[x,y,z];
/**/ R5 := NewFreeModule(R,5);
/**/ V := ModuleElem(R5, [0, 0, x^2+y*z, 0, z^2]);
/**/ FirstNonZero(V);
x^2 +y*z
/**/ FirstNonZeroPosn(V);
2
```

See Also: FirstNonZeroPosn(I-6.8 pg.86), IsZero(I-9.72 pg.146), NonZero(I-14.26 pg.190)

I-6.8 FirstNonZeroPosn

Syntax
```
FirstNonZeroPosn(V: MODULEELEM): RINGELEM
```

Description
This function returns the index of the first non-zero entry of V. If it is handed a zero MODULEELEM then an error is signalled.

```plaintext
/**/ Use R ::= QQ[x,y,z];
/**/ R5 := NewFreeModule(R,5);
/**/ V := ModuleElem(R5, [0, 0, x^2+y*z, 0, z^2]);
/**/ FirstNonZero(V);
x^2 +y*z
/**/ FirstNonZeroPosn(V);
2
```

See Also: FirstNonZero(I-6.7 pg.85), IsZero(I-9.72 pg.146), NonZero(I-14.26 pg.190)

I-6.9 flatten

Syntax
```
flatten(L: LIST): LIST
flatten(L: LIST, N: INT): LIST
```

Description
Components of lists may be lists themselves, i.e., lists may be nested. With one argument this function returns the list obtained from the list “L” by removing all nesting, bringing all elements “to the top level”. With the optional second argument, “N”, nesting is removed down “N” levels. Thus, the elements of “M := flatten(L,1)” are formed as follows: go through the elements of “L” one at a time; if an elements is not a list, add it to “M”; if an element is a list, add all of its elements to “M”. Recursively, “Flatten(L, N) = Flatten(Flatten(L, N-1),1)”. For “N” large, depending on “L”, “Flatten(L, N)” gives the same result as “Flatten(L)”.

```plaintext
```
I-6.10 FloatApprox

**syntax**

FloatApprox(X: INT|RAT|RINGELEM, PrecBits: INT): RAT

**Description**

This function computes an approximation of the form $M \times 2^E$ to a rational number “X” where the mantissa satisfies $2(B - 1) \leq |M| < 2^B - 1$ where $B$ is the specified bit precision. It gives 0 when applied to 0.

The updated version of this function is not backward compatible with the old one; you must replace the 2nd arg by the number of bits you want in the mantissa (see “ILogBase” (I-9.9 pg.119)). The old fn is obsolescent and is now called “FloatApprox10”.

**example**

```plaintext
/**/ FloatApprox(1/3, 10);
683/2048
/**/ FloatApprox(1/3, 20);
699051/2097152
/**/ FloatApprox(123456789,8);
123207680
```

See Also: CFApprox(I-3.7 pg.43), FloatStr(I-6.11 pg.87), MantissaAndExponent2(I-13.5 pg.168)

I-6.11 FloatStr

**syntax**

FloatStr(X: INT|RAT|RINGELEM): STRING
FloatStr(X: INT|RAT|RINGELEM, Prec: INT): STRING

**Description**

This function produces a decimal string representation of the rational number “X”. The optional second argument “Prec” says how many significant decimal digits to produce; the default value is 5.

The aim is to produce an easily readable result.
See Also:  DecimalStr(I-4.3 pg.63), ScientificStr(I-19.3 pg.246), FloatApprox(I-6.10 pg.87), MantissaAndExponent10(I-13.4 pg.168)

I-6.12 floor

**syntax**

floor(X: RAT): INT

**Description**

This function returns the greatest integer less than or equal to “X”.

**example**

```plaintext
/**/ floor(0.99); 0
/**/ floor(1.01); 1
/**/ floor(-1); -1
/**/ floor(-0.01); -1
```

See Also:  ceil(I-3.6 pg.43), round(I-18.50 pg.243), num(I-14.29 pg.191), den(I-4.7 pg.67)

I-6.13 for

**syntax**

For I := N_1 To N_2 Do C EndFor
For I := N_1 To N_2 Step D Do C EndFor

where I is a dummy variable, N_1, N_2, and D are integer expressions, and C is a sequence of commands.

**Description**

In the first form, the variable I is assigned the values “N_1, N_1+1, ... , N_2” in succession. After each assignment, the command sequence C is executed. The second form is the same, except that I is assigned
the values \(N_1, N_1+D, N_1+2D\), etc. until the greatest value less than or equal to \(N_2\) is reached. If \(N_2 < N_1\), then \(C\) is not executed.

NOTE: Large values for \(N_1, N_2\), or \(D\) are not permitted; typically they should lie in the range about \(-10^9\) to \(+10^9\).

NOTE: Don’t forget the capitalization in the word “To”.

```plaintext
/**/ For N := 1 To 5 Do Print 2^N, " "; EndFor;
 2 4 8 16 32

/**/ for n := 1 to 20 step 3 do print n, " "; endfor;
 1 4 7 10 13 16 19

/**/ for N := 10 To 1 Step -2 Do Print N, " "; EndFor;
 10 8 6 4 2

/**/ For N := 5 To 3 Do Print N, " "; endfor; -- no output
```

Loops can be nested.

```plaintext
/**/ Define MySort(ref L)
/**/ For I := 1 To len(L)-1 Do
  /**/ M := I;
  /**/ For J := I+1 To len(L) Do
  /**/ If L[J] < L[M] Then M := J; EndIf;
  /**/ EndFor;
  /**/ If M <> I Then
  /**/ C := L[M];
  /**/ L[M] := L[I];
  /**/ L[I] := C;
  /**/ EndIf;
  /**/ EndFor;
/**/ EndDefine;

/**/ M := [5,3,1,4,2];
/**/ MySort(ref M);
/**/ M;
[1, 2, 3, 4, 5]
```

(Note that “ref L” is used so that the function can change the value of the variable referenced by L. See “ref”.)

See Also: foreach(I-6.14 pg.89), repeat(I-18.30 pg.234), while(I-23.3 pg.286)

### I-6.14 foreach

**syntax**

```
foreach X in L do CMDS endforeach
where ‘‘\verb&X&’’ is a dummy variable, ‘‘\verb&L&’’ is a LIST
```

**Description**

The dummy variable “\(X\)” is assigned the value of each component of “\(L\)” in turn. After each assignment the command sequence “CMDS” is executed.
/**/ foreach N In 1..10 Do -- NOTE: 1..10 gives the list [1,...,10]. 
/**/ print N^2, " ";
/**/ endforeach;
1 4 9 16 25 36 49 64 81 100
/**/ Use R ::= QQ[x,y,z];
/**/ F := x^2*y + 3*y^2*z - z^3;
/**/ J := [deriv(F, X) | X In indets(R)];
/**/ J;
[2*x*y, x^2 +6*y*z, 3*y^2 -3*z^2]
/**/ Foreach X In J Do 
/**/ PrintLn X^2;
/**/ EndForeach;
4*x^2*y^2
x^4 +12*x^2*y*z +36*y^2*z^2
9*y^4 -18*y^2*z^2 +9*z^4

See Also: for(I-6.13 pg.88), repeat(I-18.30 pg.234), while(I-23.3 pg.286)

I-6.15 format

format(E: OBJECT, N: INT): STRING

Description
Like Sprint, this function converts the value of E into a string. If the string has fewer than N characters, then spaces are added to the front to make the length N.

/**/ L := [5^n | n In 0..7];
/**/ Foreach F In L Do print Format(F,8); EndForeach;
1 5 25 125 625 3125 15625 78125
/**/ M := Format(L,20);
/**/ M; -- "Format" does not truncate
[1, 5, 25, 125, 625, 3125, 15625, 78125]
/**/ type(L);
LIST
/**/ type(M);
STRING

See Also: IO.SprintTrunc(I-9.33 pg.131), LaTeX(I-12.2 pg.155), sprint(I-19.24 pg.257)

I-6.16 FrbAlexanderDual

FrbAlexanderDual(I: IDEAL): LIST
FrbAlexanderDual(I: IDEAL, T: RINGELEM): LIST

Description
Using the “Frobb” (II-8.4 pg.319) library linked with CoCoALib. Thanks to Bjarke Roune.
\section*{FrbAssociatedPrimes}

\begin{verbatim}
/**/ I := ideal(x^2, x*y, y^2, z^2);
/**/ FrbAlexanderDual(I);
ideal(x^2*y*z, x*y^2*z)

/**/ FrbAlexanderDual(I, x^2*y^2*z^5);
ideal(x^2*y*z^4, x*y^2*z^4)
\end{verbatim}

\textbf{See Also:} Frobby(II-8.4 pg.319), FrbPrimaryDecomposition(I-6.20 pg.92), PrimaryDecomposition(I-16.19 pg.208)

\section*{FrbIrreducibleDecomposition}

\begin{verbatim}
/**/ Use R ::= QQ[x,y,z];
/**/ I := ideal(x^2, x*y, y^2, z^2);
/**/ FrbIrreducibleDecomposition(I);
[ideal(x, y^2, z^2), ideal(x^2, y, z^2)]
\end{verbatim}

\textbf{See Also:} Frobby(II-8.4 pg.319), FrbIrreducibleDecomposition(I-6.18 pg.91), FrbPrimaryDecomposition(I-6.20 pg.92)

\section*{FrbAssociatedPrimes}

\textbf{Syntax:}

\texttt{FrbAssociatedPrimes(I: IDEAL): LIST}

\textbf{Description:}

Using the “Frobby” (II-8.4 pg.319) C++ library by Bjarke Roune.

\begin{verbatim}
/**/ Use R ::= QQ[x,y,z];
/**/ I := ideal(x^2, x*y, y^2, z^2);
/**/ FrbAssociatedPrimes(I);
[ideal(x, y, z)]
\end{verbatim}

\textbf{See Also:} Frobby(II-8.4 pg.319), FrbPrimaryDecomposition(I-6.20 pg.92), FrbIrreducibleDecomposition(I-6.18 pg.91), FrbIrreducibleDecomposition(I-6.18 pg.91)

\section*{FrbIrreducibleDecomposition}

\textbf{Syntax:}

\texttt{FrbIrreducibleDecomposition(I: IDEAL): LIST}

\textbf{Description:}

Using the “Frobby” (II-8.4 pg.319) C++ library by Bjarke Roune.

\begin{verbatim}
/**/ Use R ::= QQ[x,y,z];
/**/ I := ideal(x^2, x*y, y^2, z^2);
/**/ FrbIrreducibleDecomposition(I);
[ideal(x, y^2, z^2), ideal(x^2, y, z^2)];
\end{verbatim}

\textbf{See Also:} Frobby(II-8.4 pg.319), FrbAssociatedPrimes(I-6.17 pg.91), FrbIrreducibleDecomposition(I-6.18 pg.91)
I-6.19 FrbMaximalStandardMonomials

Syntax

FrbMaximalStandardMonomials(I: IDEAL): LIST

Description

Using the “Froby” (II-8.4 pg.319) library linked with CoCoALib.

Example

/**/ I := ideal(x^2, x*y, y^2, z^2);
/**/ FrbMaximalStandardMonomials(I);
ideal(y*z, x*z)

See Also: Froby(II-8.4 pg.319)

I-6.20 FrbPrimaryDecomposition

Syntax

FrbPrimaryDecomposition(I: IDEAL): LIST

Description

Using the “Froby” (II-8.4 pg.319) C++ library by Bjarke Roune.

Example

/**/ Use R ::= QQ[x,y,z];
/**/ I := ideal(x^2, x*y^2, z^2);
/**/ FrbPrimaryDecomposition(I);
[ideal(x^2, y^2, z^2), ideal(x, z^2)]

See Also: Froby(II-8.4 pg.319), FrbAssociatedPrimes(I-6.17 pg.91), FrbIrreducibleDecomposition(I-6.18 pg.91), PrimaryDecomposition(I-16.19 pg.208)

I-6.21 func

Syntax

Func ... EndFunc

returns FUNCTION

Description

This syntactic structure defines a function without giving it a name; anonymous functions can be passed as parameters and assigned to variables. Note that “Func...EndFunc” can be used inside function definitions.

Example

/**/ square := Func(x) Return x^2; EndFunc;
/**/ square(3);
9

/**/ SortedBy(["zzz", "x", "yy"], Func(x,y) Return len(x)>len(y); EndFunc);
["zzz", "yy", "x"]

See Also: define(I-4.4 pg.64), TopLevel(I-20.9 pg.274), ImportByRef, ImportByValue(I-9.14 pg.121)
I-6.22 Function [OBSOLETE]

Description

[OBSOLETE] In CoCoA-5 functions are "first class objects", and so may be passed like any other value – the operator “Function” serves no purpose.

In CoCoA-4 it was possible to have a variable and a function with the same name; the operator “Function” was used to instruct CoCoA-4 to search for the function of the given name, e.g. to pass it as an argument to another function.

See Also: FUNCTIONs are first class objects(III-7.2 pg.349), SortBy(I-19.18 pg.254), SortedBy(I-19.20 pg.255)

I-6.23 functions [OBSOLETE]

Description

[OBSOLETE] please use the command “describe” (I-4.12 pg.69).

I-6.24 FVector

syntax

Description

This function computes the f-Vector of a simplicial complex described by a list of top faces.

Package GeomModelling, by Elisa Palezzato.

example

```plaintext
/**/ Use QQ[x[1..5]], DegLex;
/**/ L := [x[1]*x[2]*x[3], x[2]*x[3]*x[4], x[3]*x[4]*x[5]]; -- list top faces
/**/ FVector(L);
[1, 5, 7, 3]
```

See Also: SimplexInfo(I-19.12 pg.251)
Chapter I-7

G

I-7.1 GBasis

**syntax**

`GBasis(I: IDEAL|MODULE): LIST`

**Description**

This function returns a list whose components form a Groebner basis for the ideal (or module) “I” with respect to the term-ordering of the polynomial ring of “I”.

If “I” is a variable then the result is stored in “I” for later use.

For the reduced Groebner basis, use the command “ReducedGBasis” (I-18.22 pg.230).

The coefficient ring must be a field.

**example**

```plaintext
/**/ Use R ::= QQ[x,y];
/**/ I := ideal(x^4-x^2, x^3-y);
/**/ GBasis(I);
[-x^2 +x*y, -x*y +y^2, y^3 -y]
```

See Also: GBasisTimeout(I-7.2 pg.95)

I-7.2 GBasisTimeout

**syntax**

`GBasisTimeout(I: IDEAL, SECONDS: INT): LIST`

`GBasisTimeout(M: MODULE, SECONDS: INT): LIST`

**Description**

***** NOT YET IMPLEMENTED *****

Same as “GBasis” (I-7.1 pg.95), but it will stop and return an error if the computation is not completed.

For dealing with errors see “try” (I-20.13 pg.277).

**example**

```plaintext
Use R ::= QQ[t,x,y,z];
I := ideal(t^3-x, t^4-y, t^5-z);
J := I^5; Time G := GBasisTimeout(J, 1);
ERROR: Time expired: use $gb.Complete to complete the computation
```
CONTEXT: Error(GBasisTimeout_Err)

J := I^5; Time G := GBasisTimeout(J, 10);
Cpu time = 1.96, User time = 2

See Also: GBasis(I-7.1 pg.95), try(I-20.13 pg.277)

I-7.3 GBM

GBM(L: LIST): IDEAL

Description

***** NOT YET IMPLEMENTED *****

This function computes the intersection of ideals corresponding to zero-dimensional schemes: GBM is for affine schemes, and “HGBM” (I-8.2 pg.105) for projective schemes. The list L must be a list of ideals. The function “IntersectList” (I-9.30 pg.130) should be used for computing the intersection of a collection of general ideals.

The name GBM comes from the name of the algorithm used: Generalized Buchberger-Moeller.

example

/**/ Use P ::= QQ[x,y,z];
/**/ I1 := IdealOfPoints(P, mat([[1,2,1], [0,1,0]])); -- a simple affine scheme
/**/ I2 := IdealOfPoints(P, mat([[1,1,1], [2,0,1]]))^2; -- another affine scheme

***** NOT YET IMPLEMENTED *****

GBM([I1, I2]); -- intersect the ideals

ideal(xz + yz - z^2 - x - y + 1,
z^3 - 2z^2 + z,
yz^2 - 2yz - z^2 + y + 2z - 1,
y^2z - y^2 - yz + y,
x^2y + y^2 + 2xz + 2xy + 4y^2 - 3z^2 - 8x - 8y + 6z + 5,
x^3 + y^2 + 7z^2 - 5xy - 4y^2 + 5z^2 + 16x + 10y - 10z - 7,
y^4 + 2y^3 - 4x^2 - 8xy - 3y^2 + 4z^2 + 16x + 16y - 8z - 12)

See Also: IdealAndSeparatorsOfPoints(I-9.3 pg.114), IdealAndSeparatorsOfProjectivePoints(I-9.4 pg.115),
IdealOfPoints(I-9.5 pg.116), IdealOfProjectivePoints(I-9.6 pg.117), HGBM(I-8.2 pg.105)

I-7.4 gcd

gcd(F: INT,G: INT): INT
gcd(L: LIST of INT): INT
gcd(F: RINGELEM, G: RINGELEM): RINGELEM
gcd(L: LIST of RINGELEM): RINGELEM

Description

This function returns the greatest common divisor of “F_1,...,F_n” or of the elements in the list L. For the calculation of the GCDs and LCMs of polynomials, the coefficient ring must be a field.
/**/ Use R ::= QQ[x,y];
/**/ F := x^2-y^2;
/**/ G := (x+y)^3;
/**/ gcd(F, G);
-x -y
/**/ gcd([3*4,3*8,6*16]);
12

See Also: div(I-4.20 pg.73), mod(I-13.21 pg.175), lcm(I-12.4 pg.156)

I-7.5 GCDFreeBasis

**/ Use R ::= QQ[x,y];
/**/ F := x^2-y^2;
/**/ G := (x+y)^3;
/**/ gcd(F, G);
-x -y
/**/ gcd([3*4,3*8,6*16]);
12

See Also: gcd(I-7.4 pg.96)

### I-7.5 GCDFreeBasis

** syntax **

GCDFreeBasis(L: LIST of INT): LIST of INT

** Description **

This function returns a GCD free basis of a set of integers; you can think of this as the set of all numbers (except 1) obtainable by performing GCD and exact division operations. Given a set \( N = [N_1, ..., N_k] \) we seek a basis \( G = [G_1, ..., G_s] \) such that each \( N_i \) is a product of powers of the \( G_j \), and the \( G_j \) are pairwise coprime; the set \( G \) is called a GCD free basis for \( N \). In general the set \( G \) is not uniquely defined.

** example **

```plaintext
/**/ GCDFreeBasis([factorial(20), factorial(10)]);
[46189, 4, 14175]
```

See Also: gcd(I-7.4 pg.96)

### I-7.6 GenericPoints

** syntax **

GenericPoints(R: RING, NumPoints: INT): LIST

** Description **

“GenericPoints” returns a list of NumPoints generic projective points with integer coordinates; it is not guaranteed that these points are distinct. RandomRange specifies the largest value any coordinate may take. If the second argument is omitted, the largest value possible is 100 (or P-1 where P is the characteristic of the coefficient ring).

** example **

```plaintext
/**/ Use R ::= QQ[x,y];
GenericPoints(R,7);
[[1, 0], [0, 1], [1, 1], [12, 59], [6, 63], [12, 80], [17, 63]]

/**/ GenericPoints(R,7,500);
[[1, 0], [0, 1], [1, 1], [220, 162], [206, 452], [98, 106], [403, 449]]

/**/ Use R ::= ZZ/(5)[x,y,z];
/**/ GenericPoints(R,7);
[[1, 0, 0], [0, 1, 0], [0, 0, 1], [1, 1, 1], [2, 1, 1], [2, 2, 4], [3, 1, 3]]
```
Chapter I-7. GenRepr

**syntax**

GenRepr(X: RINGELEM, I: IDEAL): LIST of RINGELEM
GenRepr(X: MODULEELEM, I: MODULE): LIST of RINGELEM

**Description**

This function returns a list giving a representation of X in terms of generators for I. Let the generators for I be \("G_1, \ldots, G_t"\). If X is in I, then \("\text{GenRepr}\) will return a list \("[F_1, \ldots, F_t]"\) such that

\[ X = F_1*G_1 + \ldots + F_t*G_t. \]

If X is not in I, then \("\text{GenRepr}\) returns the empty list, \([]\).

**example**

```/* */
/**/ Use R ::= QQ[x,y];
/**/ I := ideal(x+y^2, x^2-x*y);
/**/ GenRepr(x^3-x^2*y-y^3-x*y, I);
[-y, x]
/**/ -y*gens(I)[1] + x*gens(I)[2];
x^3 -x^2*y -y^3 -x*y
/**/ GenRepr(x+y, I); -- x+y is not in I
[]
/**/ K := NewFractionField(NewPolyRing(QQ, "a"));
/**/ Use R :::= K[x,y];
/**/ L := [x+y^2, x^2-x*y];
/**/ GenRepr((a-2)*L[1] - (x-a)*L[2], ideal(L));
[a -2, -x +a]
/**/ R3 := NewFreeModule(R,3);
/**/ V1 := ModuleElem(R3, [x, y, y^2]);
/**/ V2 := ModuleElem(R3, [x-y, 0, x^2]);
/**/ V := x^2*V1 - y^2*V2;
/**/ M := submodule(R3, [V1, V2]);
--/**/ GenRepr(V, M); -- NOT YET IMPLEMENTED *****
--[x^2, -y^2]
```

**See Also:** DivAlg(I-4.21 pg.73), IsIn(I-9.47 pg.136), NF(I-14.14 pg.184), syz(I-19.46 pg.267), SyzOfGens(I-19.47 pg.268)

I-7.8 gens

**syntax**

gens(I: IDEAL): LIST
gens(M: MODULE): LIST
**Description**

This function returns a list of polynomials which generate the ideal I or the module M. The list is not necessarily minimal.

```plaintext
/**/ Use R ::= QQ[x,y,z];
/**/ I := ideal(y^2-x^3, x*y);
/**/ gens(I);
[-x^3 +y^2, x*y]
/**/ gens(I^2);
[x^6 -2x^3*y^2 +y^4, -x^4*y +x*y^3, x^2*y^2]
/**/ R3 := NewFreeModule(R, 3);
/**/ e := gens(R3); // canonical basis
/**/ e[2];
[0, 1, 0]
/**/ M := SubmoduleRows(R3, mat([[x,y,z], [x-1,0,z]]));
/**/ gens(M);
[[x, y, z], [x -1, 0, z]]
/**/ shape(It);
[[MODULEELEM, MODULEELEM]]
/**/ GensAsRows(M);
matrix( /*RingDistrMPolyClean(QQ, 3)*/
[[x, y, z],
 [x -1, 0, z]])
```

**See Also:** GensAsCols, GensAsRows(I-7.9 pg.99), minimize(I-13.14 pg.173), minimalized(I-13.15 pg.173)

---

### I-7.9  GensAsCols, GensAsRows

**Syntax**

GensAsRows(M: MODULE): MAT  
GensAsCols(M: MODULE): MAT

**Description**

These functions returns a matrix which generate the module M with the components as row (or columns) of a matrix.

The generators are not necessarily minimal.

```plaintext
/**/ Use R ::= QQ[x,y,z];
/**/ R3 := NewFreeModule(R, 3);
/**/ L := [[x,y,z], [x-1,0,z]];
/**/ M := SubmoduleRows(R3, mat(L));
/**/ gens(M);
[[x, y, z], [x -1, 0, z]]
/**/ shape(It);
[[MODULEELEM, MODULEELEM]]
/**/ GensAsRows(M);
matrix( /*RingDistrMPolyClean(QQ, 3)*/
[[x, y, z],
 [x -1, 0, z]])
```
Chapter 7. G

/**/ GensAsCols(M);
matrix( /*RingDistrMPolyClean(QQ, 3)*/
  [[x, x -1],
   [y, 0],
   [z, z]])

See Also: gens(I-7.8 pg.98), SubmoduleCols, SubmoduleRows(I-19.37 pg.263)

I-7.10 Get

** syntax **

Get(D: DEVICE, N: INT): LIST of INT

Description

***** NOT YET IMPLEMENTED *****

This function reads N characters from D and returns the list of their ASCII codes.

** example **

D := OpenIFile("io.cpkg"); -- open the file "io.cpkg"
Get(D, 10); -- get the first 10 characters
[45, 45, 32, 105, 111, 100, 101, 118, 46, 112]

ascii(It); convert the ASCII code to characters
-- iodev.p

ascii(Get(D, 10)); -- get the next 10 characters and convert
kg : 0.1 :

Close(D);

The instruction “Get(DEV.STDIN, 3)”, for instance, will read 3 characters typed in by the user. Clever use of this function can be used to prompt a user for input to a function, although it is usually easier for functions to take input directly as arguments. NOTE: this function does not work properly under the GUI Interface.

See Also: Introduction to IO(II-6.1 pg.309), OpenIFile(I-15.2 pg.195), OpenOFile(I-15.5 pg.197), OpenIString(I-15.3 pg.196), OpenOString(I-15.6 pg.198)

I-7.11 GetCol

** syntax **

GetCol(M: MAT, K: INT): LIST

Description

This function makes a list containing the entries of the “K”-th column of “M”.

** example **

/**/ M := mat([[1,2], [3,4]]);
/**/ GetCol(M, 2);
[2, 4]

See Also: GetRow(I-7.15 pg.102), GetCols(I-7.12 pg.101)
I-7.12 GetCols

**syntax**

GetCols(M: MAT): LIST of LIST

**Description**

This function produces a list of lists containing the columns of “M”.

```plaintext
/**/ M := mat([[1, 2], [3, 4]]);
/**/ GetCols(M);
[ [1, 3], [2, 4] ]
```

**See Also:** GetCol(I-7.11 pg.100), GetRows(I-7.16 pg.102)

I-7.13 GetEnv

**syntax**

GetEnv(S: STRING): STRING

**Description**

This function returns the value of system shell variables

```plaintext
/**/ GetEnv("HOME");
/Users/bigatti

/**/ GetEnv("COCOARC");
/Users/bigatti/.cocoarc

/**/ GetEnv("COCOA_PACKAGES");
/Applications/CoCoA-4.7/packages
```

I-7.14 GetErrMesg

**syntax**

GetErrMesg(E: ERROR): STRING

**Description**

This function returns the string containing the error message associated with an error.

```plaintext
/**/ ErrMsg := "";

Try
  F := 1/0;
UponError E Do
  ErrMsg := GetErrMesg(E);
EndTry; -- no error is thrown with Try .. UponError .. EndTry
```
**ErrMsg;**
Division by zero or by a zero-divisor

See Also:  try(I-20.13 pg.277), error(I-5.9 pg.79)

### I-7.15 GetRow

**syntax**

GetRow(M: MAT, K: INT): LIST

**Description**

This function makes a list containing the entries of the “K”-th row of “M”.

**example**

```plaintext
/**/ M := mat([[1,2], [3,4]]);
/**/ GetRow(M,2);
[3, 4]
```

See Also:  GetRows(I-7.16 pg.102), SetRow(I-19.7 pg.249)

### I-7.16 GetRows

**syntax**

GetRows(M: MAT): LIST of LIST

**Description**

This function produces a list of lists containing the rows of “M”.

**example**

```plaintext
/**/ M := mat([[1,2], [3,4]]);
/**/ GetRows(M);
[[1, 2], [3, 4]]
```

See Also:  GetRow(I-7.15 pg.102)

### I-7.17 gin

**syntax**

gin(I: IDEAL): IDEAL

gin(I: IDEAL, VerboseFlag: STRING): IDEAL

**Description**

These functions return the [probabilistic] gin (generic initial ideal) of the ideal “I”. It is obtained by computing twice the leading term ideal of \(g(I)\), where \(g\) is a random change of coordinates with integer coefficients in the range \([-10^6, 10^6]\) using TwinFloats (see “NewRingTwinFloat” (I-14.11 pg.183)) to allow a much wider range of coefficients than a direct computation over the rationals (use second argument to see the TwinFloat precision needed).
/**/ Use R ::= QQ[x,y,z], DegRevLex;
/**/ gin(ideal(y^2-x*z, x^2*z-y*z^2)); -- computed twice using TwinFloats
  ideal(x^2, x*y^2, y^4)
/**/ gin(ideal(y^7-x^4*z^3, x^5*z-y*z^5), "verbose");
  -- trying with FloatPrecision 64
  -- trying with FloatPrecision 64
  ideal(x^6, x^5*y^2, x^4*y^4, x^3*y^6, x^2*y^8, x*y^10, y^12)

See Also: NewRingTwinFloat(I-14.11 pg.183)

I-7.18  GradingDim

GradingDim(P): INT

Description
This function returns the grading dimension of a polynomial ring, i.e. how many of the rows of OrderMatrix are to be taken as specifying the grading.

/**/ OrdM := CompleteToOrd(RowMat([2,3]));
/**/ P := NewPolyRing(QQ, "x,y", OrdM, 1);
/**/ GradingDim(P);
1

See Also: NewPolyRing(I-14.8 pg.182), GradingMat(I-7.19 pg.103)

I-7.19  GradingMat

WeightsMatrix(R: RING): MAT

Description
This function returns the grading matrix (or weights matrix) for the polynomials ring “R”.

/**/ OrdM := CompleteToOrd(RowMat([2,3]));  OrdM;
  matrix(QQ,
  [[2, 3],
   [0, -1]])
/**/ P := NewPolyRing(QQ, "x,y", OrdM, 1);  -- GradingDim = 1
/**/ GradingMat(P);
  matrix([ [2, 3] ])
/**/ Use P;
/**/ deg(x*y);
  2
/**/ wdeg(x*y);
  [5]
See Also: deg(I-4.6 pg.66), wdeg(I-23.1 pg.285), GradingDim(I-7.18 pg.103), NewPolyRing(I-14.8 pg.182)
Chapter I-8

H

I-8.1 HColon

**syntax**

\[
\text{HColon}(M: \text{IDEAL, N: IDEAL}): \text{IDEAL}
\]

**Description**

***** NOT YET IMPLEMENTED *****

The function “colon” (I-3.27 pg.52) returns the quotient of M by N: the ideal of all polynomials F such that 
\(F \cdot G\) is in M for all G in N.

This function computes the same ideal using a Hilbert-driven algorithm. It differs from “colon” (I-3.27 pg.52) 
only when the input is non-homogeneous, in which case, “HColon” may be faster.

**example**

Use \(R ::= \mathbb{Q}[x,y]\);
\[
\begin{align*}
\text{ideal}(xy, x^2) & : \text{ideal}(x); \\
\text{ideal}(y, x) & \\
\text{-----------------------------} \\
\text{colon(ideal}(x^2, xy), \text{ideal}(x, x-y^2)) & ; \\
\text{ideal}(x) & \\
\text{-----------------------------} \\
\text{HColon(ideal}(x^2, xy), \text{ideal}(x, x-y^2)) & ; \\
\text{ideal}(x) & \\
\text{-----------------------------}
\end{align*}
\]

See Also: HSaturation(I-8.12 pg.111), saturate(I-19.1 pg.245), HColon(I-8.1 pg.105), colon(I-3.27 pg.52)

I-8.2 HGBM

**syntax**

\[
\text{HGBM}(L: \text{LIST}): \text{IDEAL}
\]

**Description**

***** NOT YET IMPLEMENTED *****

This function computes the intersection of ideals corresponding to zero-dimensional schemes: “GBM” (I-7.3 pg.96) is for affine schemes, and HGBM for projective schemes. The list L must be a list of ideals. The function “IntersectList” (I-9.30 pg.130) should be used for computing the intersection of a collection of general ideals.
The name GBM comes from the name of the algorithm used: Generalized Buchberger-Moeller. The prefix H comes from Homogeneous since ideals of projective schemes are necessarily homogeneous.

```plaintext
example

Use QQ[x[0..2]];  
I1 := IdealOfProjectivePoints([[1,2,1], [0,1,0]]);  -- simple projective scheme  
I2 := IdealOfProjectivePoints([[1,1,1], [2,0,1]])^2;  -- another projective scheme  
HGBM([I1, I2]);  -- intersect the ideals  
-------------------------------
```

**See Also:** IdealAndSeparatorsOfPoints(I-9.3 pg.114), IdealAndSeparatorsOfProjectivePoints(I-9.4 pg.115), IdealOfPoints(I-9.5 pg.116), IdealOfProjectivePoints(I-9.6 pg.117), GBM(I-7.3 pg.96)

### I-8.3 hilight [OBSOLESCENT]

**syntax**

[OBSOLESCENT]

**Description**

Renamed as “HilbertFn” (I-8.5 pg.107).

### I-8.4 HilbertBasisKer

**syntax**

HilbertBasisKer(M: MAT): LIST  
where M is a matrix over ZZ.

**Description**

This function returns a list whose components are lists (of non-negative integers) representing the Hilbert basis for the monoid of elements with non-negative coordinates in the kernel of M.

```plaintext
example

/**/ M := mat([[1,-2,3,4], [1,0,0,-1]]);  
/**/ HilbertBasisKer(M);  
[[0, 3, 2, 0], [1, 4, 1, 1], [2, 5, 0, 2]]  
/**/ M * transposed(mat(It));  
matrix([  
[0, 0, 0],  
[0, 0, 0]  
])
```
I-8.5 HilbertFn

**syntax**

HilbertFn(R: RING|IDEAL): TAGGED("hp.Hilbert")
HilbertFn(R: RING|IDEAL, N: INT): INT

**Description**

The first form of this function computes the Hilbert function for R. The second form computes the N-th value of the Hilbert function. The weights of the indeterminates of R must all be 1. If the input is not homogeneous, the Hilbert function of the corresponding leading term (initial) ideal or module is calculated. For repeated evaluations of the Hilbert function, use “EvalHilbertFn” (I-5.11 pg.80) instead of “HilbertFn(R, N)” in order to speed up execution.

The coefficient ring must be a field.

**example**

```cocoa
/**/ Use R ::= QQ[t,x,y,z];
/**/ HilbertFn(R/ideal(z^2-x*y, x*z^2+t^3));
H(0) = 1
H(1) = 4
H(t) = 6*t -3 for t >= 2

/**/ R2 := NewFreeModule(R, 2);
/**/ MGens := matrix(R, [[x^3,y^3], [x*y^2,0], [0,z^3]]);
/**/ M := SubmoduleRows(R2, MGens);
/**/ HilbertFn(M);
H(0) = 0
H(1) = 0
H(2) = 0
H(3) = 3
H(4) = 12
H(t) = (1/3)*t^3 + (3/2)*t^2 +(-101/6)*t +35 for t >= 5

/**/ HilbertFn(M,3);
3
/**/ HilbertFn(M,5);
30
```

See Also: EvalHilbertFn(I-5.11 pg.80), HilbertPoly(I-8.6 pg.107), HVector(I-8.13 pg.112), HilbertSeries(I-8.7 pg.108)

I-8.6 HilbertPoly

**syntax**

HilbertPoly(R: RING or TAGGED("Quotient"): RINGELEM in the ring QQt.

**Description**

This function returns the Hilbert polynomial for R as a polynomial in the standard CoCoA ring “QQt” (= QQ[t]).
The weights of the indeterminates of “\( R \)” must all be 1, and the coefficient ring must be a field.

If the input is not homogeneous, the Hilbert polynomial of the corresponding leading term (initial) ideal or module is calculated. For the Hilbert *function*, see “HilbertFn” (I-8.5 pg.107).

```plaintext
/**/ Use R ::= QQ[w,x,y,z];
/**/ I := ideal(z^2-x*y, x*z^2+w^3);
/**/ HilbertFn(R/I);
H(0) = 1
H(1) = 4
H(t) = 6*t-3 for t >= 2

/**/ F := HilbertPoly(R/I);
/**/ F; -- a polynomial in the ring Qt
6*t-3

/**/ T := indet(RingOf(F), 1);
/**/ subst(F, T, 3);
15
```

**See Also:** EvalHilbertFn(I-5.11 pg.80), HilbertFn(I-8.5 pg.107), HVector(I-8.13 pg.112), HilbertSeries(I-8.7 pg.108), RingQQt(I-18.43 pg.240)

## I-8.7 HilbertSeries

### Syntax

| HilbertSeries(M: MODULE|IDEAL|RING):TAGGED("$hp.PSeries") |
|----------------------------------------------------------|

### Description

This function computes the Hilbert-Poincare series of “\( M \)”. The input, “\( M \)”, must be homogeneous (with respect to the first row of the weights matrix). In the standard case, i.e. the weights of all indeterminates are 1, the result is simplified so that the power appearing in the denominator is the dimension of “\( M \)”. 

NOTE: for the local case see “PrimaryHilbertSeries” (I-16.22 pg.209).

NOTES:

(i) the coefficient ring must be a field.

(ii) these functions produce tagged objects: they cannot safely be tested for (non-)equality to other values.

Starting from release 4.7.5 the input may also be an ideal.

For more information, see the article: A.M. Bigatti, "Computations of Hilbert-Poincare Series” J. Pure Appl. Algebra, 119/3 (1997), 237–253.

```plaintext
/**/ Use R ::= QQ[t,x,y,z]; -- standard weights
/**/ HilbertSeries(R/ideal(R,[]));
(1) / (1-t)^4

/**/ HilbertSeries(R/ideal(t^2, x, y^3));
(1 + 2*t + 2*t^2 + t^3) / (1-t)

/**/ R2 := NewFreeModule(R, 2); -- MODULE
/**/ M := SubmoduleRows(R2, matrix(R, [[x^2,0], [0,z^3]]));
/**/ HilbertSeries(M);
(t^2 + t^3) / (1-t)^4
```
I-8.8 HilbertSeriesMultiDeg

```plaintext
/**/ HilbertSeries(R2/M);
-- WORK IN PROGRESS

/**/ Ws := RowMat([1,2,3,4]); -- weights and multigradings
/**/ P := NewPolyRing(QQ, "t,x,y,z", CompleteToOrd(Ws), 1);
/**/ Use P;
/**/ HilbertSeries(P/ideal(t^2, x, y^3));
--- Non-simplified HilbertPoincare' Series ---
(1 - 2*t^2 + t^4 - t^9 + 2*t^11 - t^13) / ( (1-t)*(1-t^2)*(1-t^3)*(1-t^4) )

/**/ HilbertSeries(ideal(t^2, x, y^3));
--- Non-simplified HilbertPoincare' Series ---
(2*t^2 - t^4 + t^9 - 2*t^11 + t^13) / ( (1-t)*(1-t^2)*(1-t^3)*(1-t^4) )

/**/ Ws := mat([[[1,2,3,4],[0,0,5,8]]]);
/**/ P := NewPolyRing(QQ, "t,x,y,z", CompleteToOrd(Ws), 2);
/**/ Use P;
/**/ HilbertSeries(P/ideal(t^2, x, y^3));
--- Non-simplified HilbertPoincare' Series ---

/**/ Ws := mat([[[1,2,3,4],[0,0,5,8]]]);
/**/ P := NewPolyRing(QQ, "t,x,y,z", CompleteToOrd(Ws), 2);
/**/ Use P;
/**/ HilbertSeries(P/ideal(t^2, y^3));
--- Non-simplified HilbertPoincare' Series ---
```

See Also: dim(I-1.47 pg.72), multiplicity(I-13.28 pg.178), HilbertFn(I-8.5 pg.107), HVector(I-8.13 pg.112), HilbertSeriesShifts(I-8.9 pg.110), HilbertSeriesMultiDeg(I-8.8 pg.109), GradingMat(I-7.19 pg.103), PrimaryHilbertSeries(I-16.22 pg.209)

I-8.8 HilbertSeriesMultiDeg

**syntax**

HilbertSeriesMultiDeg(RmodI: RING, WM: MAT): TAGGED("hp.PSeries")

**Description**

This function computes the multigraded Hilbert-Poincare series of “RmodI” wrt the multigrading “WM”. The “I” must be homogeneous wrt the multigrading “WM”.

This function is only a handy shortcut to avoid creating the proper polynomial ring multi-graded with “WM”.

**example**

```plaintext
/**/ Use R ::= QQ[x,y];
/**/ HilbertSeriesMultiDeg(R/ideal(Indets(R))^2, mat([[1,1]]));
(1 + 2*t) / (1-t)^0

/**/ HilbertSeriesMultiDeg(R/ideal(Indets(R))^2, mat([[1,0],[0,1]]));
--- Non-simplified HilbertPoincare' Series ---
```
See Also: HilbertSeries(I-8.7 pg.108)

I-8.9 HilbertSeriesShifts

**syntax**

HilbertSeriesShifts(M: MODULE, ShiftsList: LIST):TAGGED("hp.PSeries")
HilbertSeriesShifts(M: TAGGED("Quotient"), ShiftsList: LIST):
  :TAGGED("hp.PSeries")

**Description**

This function computes the Hilbert-Poincare series (single-graded) module "M" with shifts "sh".

This function is only a handy shortcut to avoid creating the proper free module with shifts "sh".

NOTE: functions producing tagged objects cannot safely be compared for equality with other values.


**example**

```plaintext
/**/ Use P ::= QQ[x,y,z];
/**/ F := NewFreeModule(P, ColMat([[2,0]])); -- P(-2) (+) P(0)
/**/ M := SubmoduleRows(F, mat([[x,y^3], [x-z,0]]));
/**/ HilbertSeries(M);
(2*t^3) / (1-t)^3
/**/ HilbertSeriesShifts(M, [3,1]);
(2*t^4) / (1-t)^3
```

See Also: dim(I-4.17 pg.72), HilbertFn(I-8.5 pg.107), HVector(I-8.13 pg.112), multiplicity(I-13.28 pg.178), GradingMat(I-7.19 pg.103)

I-8.10 homog

**syntax**

homog(V: RINGELEM, X: RINGELEM): RINGELEM
homog(V: MODULEELEM, X: RINGELEM): MODULEELEM
homog(L: LIST, X: RINGELEM): LIST
homog(I: IDEAL, X: RINGELEM): IDEAL
homog(M: MODULE, X: RINGELEM): MODULE

**Description**

This function returns the homogenization of the first arg with respect to the indeterminate "X", which must have weight 1. The elements of the list “L” are homogenized separately.

NOTE: For an ideal/module the result is the ideal/module containing the homogenizations of all elements (and not simply the homogenizations of the specific generators).

**example**

```plaintext
/**/ Use R ::= QQ[x,y,z,w];
/**/ homog(x^3-y, w);
x^3 -y*w^2
/**/ homog([x^3-y, x^4-z], w);
[x^3 -y*w^2, x^4 -z*w^3]
```
I-8.11 HomogElimMat

/**/ I := ideal(x^3-y, x^4-z);
/**/ homog(I, w); -- don't just get the homogenizations of
-- the generators of I
ideal(x*y -z*w, x^2*z -y^2*w, x^3 -y*w^2, y^3 -x*z^2)

See Also: IsHomog(I-9.46 pg.136)

I-8.11 HomogElimMat

syntax

HomogElimMat(W: MAT, ElimInd: LIST): MAT

Description

This function returns a matrix for a term ordering eliminating the indeterminates with indices in “ElimInd” for homogeneous input wrt the weights in the matrix “W”. If you don’t understand what this means use “ElimMat” (I-5.4 pg.77) instead ;-)

example

/**/ HomogElimMat(mat([[1,5,0]]), [2,3]);
matrix(QQ,
[[1, 5, 0],
[0, 1, 1],
[0, 0, -1]])

See Also: elim(I-5.3 pg.76), ElimMat(I-5.4 pg.77)

I-8.12 HSaturation

syntax

HSaturation(I: IDEAL, J: IDEAL): IDEAL

Description

***** NOT YET IMPLEMENTED *****

This functions returns the saturation of I with respect to J: the ideal of polynomials F such that F*G is in I for all G in J^d for some positive integer d.

It calculates the saturation using a Hilbert-driven algorithm. It differs from “saturate” (I-19.1 pg.245) only when the input is inhomogeneous, in which case, “HSaturation” may be faster.

The coefficient ring must be a field.

example

/**/ Use R ::= QQ[x,y];
/**/ I := ideal(x^4-x, y*x-2*x);
/**/ saturate(I, ideal(x));
ideal(y -2, x^3 -1)

HSaturation(I, ideal(x)); -- ***** NOT YET IMPLEMENTED *****

See Also: colon(I-3.27 pg.52), HColon(I-8.1 pg.105), saturate(I-19.1 pg.245)
I-8.13 HVector

**Syntax**

```plaintext
HVector(R: RING or TAGGED("Quotient")): LIST
```

**Description**

This function returns the h-vector of M, i.e., the coefficients of the numerator of the simplified Poincare series for M. M can be a module or a quotient.

The weights of the indeterminates of the polynomial ring of M must all be 1, and the coefficient ring must be a field.

If the input is not homogeneous, the Hilbert function of the corresponding leading term (initial) ideal or module is calculated.

```plaintext
/**/ Use R ::= QQ[t,x,y,z];
/**/ HVector(R/ideal(x,y,z)^5);
[1, 3, 6, 10, 15]

/**/ HilbertSeries(R/ideal(x,y,z)^5);
(1 + 3t + 6t^2 + 10t^3 + 15t^4) / (1-t)
```

**See Also:** HilbertFn(I-8.5 pg.107), HilbertSeries(I-8.7 pg.108)
Chapter I-9

I

I-9.1  ID [OBSOLETE]

**syntax**

[OBSOLETE]

**Description**

[OBSOLETE] renamed “RingID” (I-18.40 pg.239).

See Also: RingID(I-18.40 pg.239)

I-9.2  ideal

**syntax**

ideal(g1: RINGELEM,...,gn: RINGELEM): IDEAL

ideal(L: LIST): IDEAL

ideal(R: RING, L: LIST): IDEAL

**Description**

The first form returns the ideal generated by “g1,...,gn”. The second form returns the ideal generated by the polynomials in “L” (a bit more flexible than the first form). The third is the same as the second but works also if “L = []”.

**example**

```plaintext
/**/ Use R ::= QQ[x,y,z];
/**/ I := ideal(x-y^2, x*y-z);
/**/ I;
ideal(-y^2 +x, x*y -z)

/**/ L := [x*y-z, x-y^2];
/**/ J := ideal(L); -- same as ideal(R, L)
/**/ I = J;
true

/**/ ideal(R, []);
ideal()
```
I-9.3  IdealAndSeparatorsOfPoints

**syntax**

```
IdealAndSeparatorsOfPoints(Points: LIST): RECORD
```

where Points is a list of lists of coefficients representing a set of ‘‘{\it distinct}’’ points in affine space.

**Description**

***** NOT YET IMPLEMENTED *****

This function computes the results of “IdealOfPoints” (I-9.5 pg.116) and “SeparatorsOfPoints” (I-19.5 pg.247) together at a cost lower than making the two separate calls. The result is a record with three fields:

```
points -- the points given as argument
ideal -- the result of IdealOfPoints
separators -- the result of SeparatorsOfPoints
```

Thus, if the result is stored in a variable with identifier X, then: X.points will be the input list of points; X.ideal will be the ideal of the set of points, with generators forming the reduced Groebner basis for the ideal; X.separators will be a list of polynomials whose i-th element will take the value 1 on the i-th point and 0 on the others.

**NOTE:**

* the current ring must have at least as many indeterminates as the dimension of the space in which the points lie;
* the base field for the space in which the points lie is taken to be the coefficient ring, which should be a field;
* in the polynomials returned, the first coordinate in the space is taken to correspond to the first indeterminate, the second to the second, and so on;
* if the number of points is large, say 100 or more, the returned value can be very large. To avoid possible problems when printing such values as a single item we recommend printing out the elements one at a time as in this example:

```
X := IdealAndSeparatorsOfPoints(Pts);
Foreach Element In gens(X.ideal) Do
  PrintLn Element;
EndForeach;
```

For ideals and separators of points in projective space, see “IdealAndSeparatorsOfProjectivePoints” (I-9.4 pg.115).

**example**

```
Use R ::= QQ[x,y];
Points := [[1, 2], [3, 4], [5, 6]];
X := IdealAndSeparatorsOfPoints(Points);
X.points;
[[1, 2], [3, 4], [5, 6]]
-------------------------------
X.ideal;
ideal(x - y + 1, y^3 - 12y^2 + 44y - 48)
-------------------------------
X.separators;
[1/8y^2 - 5/4y + 3, -1/4y^2 + 2y - 3, 1/8y^2 - 3/4y + 1]
-------------------------------
```
IdealAndSeparatorsOfProjectivePoints

**Syntax**

```
IdealAndSeparatorsOfProjectivePoints(Points: LIST): RECORD
```

where Points is a list of lists of coefficients representing a set of ‘‘\{it distinct\}’’ points in projective space.

**Description**

***** NOT YET IMPLEMENTED *****

This function computes the results of “IdealOfProjectivePoints” (I-9.6 pg.117) and “SeparatorsOfProjectivePoints” (I-19.6 pg.248) together at a cost lower than making the two separate calls. The result is a record with three fields:

- **points** -- the points given as argument
- **ideal** -- the result of IdealOfProjectivePoints
- **separators** -- the result of SeparatorsOfProjectivePoints

Thus, if the result is stored in a variable with identifier X, then: X.ideal will be the ideal of the set of points, with generators forming a reduced Groebner basis for the ideal; X.separators will be a list of homogeneous polynomials whose i-th element will be non-zero (actually 1, using the given representatives for the coordinates of the points) on the i-th point and 0 on the others.

**NOTE:**

* the current ring must have at least one more indeterminate than the dimension of the projective space in which the points lie, i.e, at least as many indeterminates as the length of an element of the input, Points;
* the base field for the space in which the points lie is taken to be the coefficient ring, which should be a field;
* in the polynomials returned, the first coordinate in the space is taken to correspond to the first indeterminate, the second to the second, and so on;
* if the number of points is large, say 100 or more, the returned value can be very large. To avoid possible problems when printing such values as a single item we recommend printing out the elements one at a time as in this example:

```
X := IdealAndSeparatorsOfProjectivePoints(Pts);
Foreach Element In gens(X.ideal) Do
  PrintLn Element;
EndForeach;
```

For ideals and separators of points in affine space, see “IdealAndSeparatorsOfPoints” (I-9.3 pg.114).

**Example**

```
Use R ::= QQ[x,y,z];
Points := [[0,0,1],[1/2,1,1],[0,1,0]];
X := IdealAndSeparatorsOfProjectivePoints(Points);
X.points;
[0, 0, 1], [1, 1, 1], [0, 1, 0]
-------------------------------
X.ideal;
```
ideal(xz - 1/2yz, xy - 1/2yz, x^2 - 1/4yz, y^2z - yz^2)
-------------------------------
X.separators;
[-2x + z, x, -2x + y]
-------------------------------
Use R ::= QQ[t,x,y,z];
Pts := GenericPoints(20); -- 20 random points in projective 3-space
X := IdealAndSeparatorsOfProjectivePoints(Pts);
Len(Gens(X.Ideal)); -- number of generators in the ideal
17
-------------------------------
HilbertFn(R/X.Ideal);
H(0) = 1
H(1) = 4
H(2) = 10
H(t) = 20 for t >= 3
-------------------------------
F := X.Separators[3];
[Eval(F, P)| P In Pts];
[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
-------------------------------
Res(R/X.Ideal); -- the resolution of the ideal
0 --> R^10(-6) --> R^24(-5) --> R^15(-4) --> R
-------------------------------
See Also:  
HGBM(I-8.2 pg.105), GBM(I-7.3 pg.96), GenericPoints(I-7.6 pg.97), IdealAndSeparatorsOfPoints(I-9.3 pg.114), IdealOfPoints(I-9.5 pg.116), IdealOfProjectivePoints(I-9.6 pg.117), Interpolate(I-9.27 pg.128), QuotientBasis(I-17.3 pg.218), SeparatorsOfPoints(I-19.5 pg.247), SeparatorsOfProjectivePoints(I-19.6 pg.248)

I-9.5  IdealOfPoints

**syntax**

IdealOfPoints(P: RING, Points: MAT): IDEAL

where Points is a list of lists of coefficients representing a set of ‘‘\{it distinct\}’’ points in affine space.

**Description**

This function computes the reduced Groebner basis for the ideal of all polynomials which vanish at the given set of points. It returns the ideal generated by that Groebner basis.

**NOTE:**
* the current ring must have at least as many indeterminates as the dimension of the space in which the points lie;
* the base field for the space in which the points lie is taken to be the coefficient ring, which should be a field;
* in the polynomials returned, the first coordinate in the space is taken to correspond to the first indeterminate, the second to the second, and so on;

For ideals of points in projective space, see “IdealOfProjectivePoints” (I-9.6 pg.117).

/**/ Use P ::= QQ[x,y];
/** Points := mat([[1, 2], [3, 4], [5, 6]]);
/** I := IdealOfPoints(P, Points);
/** I;
ideal(x - y + 1, y^3 - 12*y^2 + 44*y - 48)
/** K := NewFractionField(NewPolyRing(QQ, "a"));
/** Use K;
/** Points := mat([[1, 2, 0], [3, 4, a], [5, 1, 6]]);
/** Use P ::= K[x,y,z], Lex;
/** I := IdealOfPoints(P, Points);
/** indent(I);
ideal(
  z^3 + (-a - 6)*z^2 + (6*a)*z,
  y + ((-a - 12)/(6*a^2 - 36*a))*z^2 + ((a^2 + 72)/(6*a^2 - 36*a))*z - 2,
  x + ((2*a - 6)/(3*a^2 - 18*a))*z^2 + ((-2*a^2 + 36)/(3*a^2 - 18*a))*z - 1)

See Also: GBM(I-7.3 pg.96), HGBM(I-8.2 pg.105), GenericPoints(I-7.6 pg.97), IdealAndSeparatorsOfPoints(I-9.3 pg.114), IdealAndSeparatorsOfProjectivePoints(I-9.4 pg.115), IdealOfProjectivePoints(I-9.6 pg.117), Interpolate(I-9.27 pg.128), QuotientBasis(I-17.3 pg.218), SeparatorsOfPoints(I-19.5 pg.247), SeparatorsOfProjectivePoints(I-19.6 pg.248)

I-9.6 IdealOfProjectivePoints

```plaintext
IdealOfProjectivePoints(Points: LIST): IDEAL
where Points is a list of lists of coefficients representing a set of "\{vit distinct\}" points in projective space.
```

Description

***** NOT YET IMPLEMENTED *****

This function computes the reduced Groebner basis for the ideal of all homogeneous polynomials which vanish at the given set of points. It returns the ideal generated by that Groebner basis.

NOTE:
* the current ring must have at least one more indeterminate than the dimension of the projective space in which the points lie, i.e., at least as many indeterminates as the length of an element of the input, Points;
* the base field for the space in which the points lie is taken to be the coefficient ring, which should be a field;
* in the polynomials returned, the first coordinate in the space is taken to correspond to the first indeterminate, the second to the second, and so on;
* if the number of points is large, say 100 or more, the returned value can be very large. To avoid possible problems when printing such values as a single item we recommend printing out the elements one at a time as in this example:

    I := IdealOfProjectivePoints(Pts);
    Foreach Element In gens(I) Do
        PrintLn Element;
    EndForeach;

For ideals of points in affine space, see “IdealOfPoints” (I-9.5 pg.116).
Use \( R := \mathbb{Q}[x,y,z]; \)
\[
I := \text{IdealOfProjectivePoints}([[0,0,1],[1/2,1,1],[0,1,0]]);
I;
\]
\[
\text{ideal}(xz - 1/2yz, xy - 1/2yz, x^2 - 1/4yz, y^2z - yz^2)
\]
----
\[
I.\text{Gens}; -- \text{the reduced Groebner basis}
\]
\[
[xz - 1/2yz, xy - 1/2yz, x^2 - 1/4yz, y^2z - yz^2]
\]
----

\textbf{See Also:} GBM(I-7.3 pg.96), HGBM(I-8.2 pg.105), GenericPoints(I-7.6 pg.97), IdealAndSeparatorsOfPoints(I-9.3 pg.114), IdealAndSeparatorsOfProjectivePoints(I-9.4 pg.115), IdealOfPoints(I-9.5 pg.116), Interpolate(I-9.27 pg.128), QuotientBasis(I-17.3 pg.218), SeparatorsOfPoints(I-19.5 pg.247), SeparatorsOfProjectivePoints(I-19.6 pg.248)

\textbf{I-9.7 IdentityMat}

\textbf{syntax}

\texttt{IdentityMat(R: RING, N: INT): MAT}

\textbf{Description}

This function returns the NxN identity matrix with entries in \( R \).

\texttt{/**/ Id := IdentityMat(QQ,3); Id; }
\texttt{matrix(QQ,}
\texttt{[[1, 0, 0],}
\texttt{[0, 1, 0],}
\texttt{[0, 0, 1]])}
\texttt{/**/ type(Id[1,1]);}
\texttt{RINGELEM}
\texttt{/**/ RingOf(Id[1,1]);}
\texttt{QQ}

\textbf{I-9.8 if}

\textbf{syntax}

\texttt{If B_1 Then C_1 EndIf}
\texttt{If B_1 Then C_1 Else D EndIf}
\texttt{If B_1 Then C_1 Elif B_2 Then C_2 Elif ... EndIf}
\texttt{If B_1 Then C_1 Elif B_2 Then C_2 Elif ... Else D EndIf}

where the \( B_j \) are boolean expressions, and the \( C_j \) and \( D \) are command sequences.

\textbf{Description}

If \( B_n \) is the first in the sequence of the \( B_j \) to evaluate to True, then \( C_n \) is executed. If none of the \( B_j \) evaluates to True, then \( D \) is executed if present otherwise nothing is done. The construct, \( \text{Elif } B_j \text{ Then } C_j \) can be repeated any number of times.

\textbf{NOTE:} “Elsif” is no longer allowed.
/**/ Define MySign(A)
/**/ If A > 0 Then Return 1;
/**/ Elif A = 0 Then Return 0;
/**/ Else Return -1;
/**/ EndIf;
/**/ EndDefine;
/**/ MySign(3);
1

I-9.9  ILogBase

/**/ ILogBase(X: RAT, Base: INT): INT

Description
This function computes the integer part (floor) of the logarithm of a rational number in a given base. The signs of X and Base are ignored.

/**/ ILogBase(128,2);
7
/**/ ILogBase(81.5,3);
4

I-9.10  image [OBSCOLESCENT]

/**/ Use D ::= QQ[x,y]; -- domain
/**/ f := x-y; -- a RINGELEM in D
/**/ Use C ::= QQ[a,b,c]; -- codomain
/**/ -- the old trick
/**/ Phi := RMap(a, c^2-a*b); -- OBSOULESCENT
/**/ Image(f, Phi); -- OBSOULESCENT
a*b -c^2 +a
/**/ -- the proper call
/**/ phi := PolyAlgebraHom(D, C, [a, c^2-a*b]); -- a RINGHOM
/**/ phi(f);
a*b -c^2 +a
See Also: PolyAlgebraHom(I-16.14 pg.205), apply(I-1.12 pg.30), BringIn(I-2.12 pg.39), subst(I-19.39 pg.264)

I-9.11 implicit

**syntax**

```
implicit(SubalgebraGens: LIST): IDEAL
implicit(R: RING, SubalgebraGens: LIST): IDEAL
```

**Description**

This function returns the implicitization of the subalgebra generated by the list “SubalgebraGens”. If provided with a ring “R”, the result is in “R”, otherwise it is in a newly created ring.

**example**

```
/**/ Use S ::= QQ[s,t];
/**/ implicit([s^3, s^2*t, s*t^2, t^3]);

/**/ R ::= QQ[x,y,z,w];
/**/ implicit(R, [s^3, s^2*t, s*t^2, t^3]);
ideal(z^2 -y*w, y*z -x*w, y^2 -x*z)
```

See Also: ker(I-11.1 pg.153)

I-9.12 ImplicitPlot

**syntax**

```
ImplicitPlot(F: POLY, Xrange: LIST, Yrange: LIST)
```

**Description**

This function evaluates the first argument, a bivariate polynomial, at a grid of points in the range given by the second and third arguments. The coordinates of the approximate zeroes are output to a file called "CoCoAPlot". See “ImplicitPlotOn” (I-9.13 pg.121) for outputting to another file.

This result can be plotted using your preferred plotting program. For example, start "gnuplot" and then give it the command

```
plot "CoCoAPlot"
```
to see the plot.

**example**

```
/**/ Use R ::= QQ[x,y];
/**/ ImplicitPlot(x^2 + y^2 - 200^2, [-256,256], [-256,256]);
Plotting points...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
800 plotted points have been placed in the file CoCoAPlot
```

See Also: ImplicitPlotOn(I-9.13 pg.121), PlotPoints(I-16.9 pg.204)
I-9.13 ImplicitPlotOn

**syntax**

```plaintext
ImplicitPlotOn(F: POLY, Xrange: LIST, Yrange: LIST, PlotFileName: STRING)
```

**Description**

This function is the same as “ImplicitPlot” (I-9.12 pg.120) with a fourth argument giving the name of the file to print on.

Note that the last argument is a STRING, the name of the file, and not a DEVICE, as for “print on” (I-16.26 pg.212).

```plaintext
/**/ Use R::= QQ[x,y];
/**/ F := x^2 + y^2 - 100;
/**/ G := ((x+y)^2-1)*(x^2-36);
/**/ H := ((64*y^2-36*x^2)*(36*y^2-64*x^2)*(100*x^2-y^2)-1) * F - 1000^2 * G;
/**/ ImplicitPlotOn(F, [-16,16], [-16,16], "PLOT-circle");
Plotting points...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
640 plotted points have been placed in the file circle
/**/ ImplicitPlotOn(G, [-16,16], [-16,16], "PLOT-lines");
Plotting points...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
1502 plotted points have been placed in the file lines
/**/ ImplicitPlotOn(H, [-16,16], [-16,16], "PLOT-curve");
Plotting points...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
2790 plotted points have been placed in the file curve
```

After having produced the plot files using CoCoA-4, start “gnuplot” and then give it the following commands:

```plaintext
plot "circle"
replot "lines"
replot "curve"
```

See Also: ImplicitPlot(I-9.12 pg.120), PlotPointsOn(I-16.10 pg.204)

I-9.14 ImportByRef, ImportByValue

**syntax**

```plaintext
ImportByRef X;
ImportByValue X;
where ‘‘\verb&X&’’ is the name of a variable in the containing scope.
```

**Description**

“***YOU PROBABLY SHOULDN’T USE THESE COMMANDS YET!***” It seems that they can be used only inside anonymous functions (see “func” (I-6.21 pg.92)).

These commands “import” an external variable by reference or value. “ImportByValue” creates a local variable with the given name, and its initial value is taken from the variable of the same name in the context the anonymous function is defined. “ImportByRef” creates a reference to the named variable in the context where the anonymous function is defined.
NOTE: Package variables should be accessed directly (via the fully qualified name); they cannot be imported.

```plaintext
/**/ Define add(X)
/**/ AnonFn := Func(Y) ImportByValue X; Return X+Y; EndFunc;
/**/ Return AnonFn;
/**/ EndDefine;
/**/ add3 := add(3);
/**/ add3(2);
5

See Also: TopLevel(I-20.9 pg.274), func(I-6.21 pg.92)

I-9.15 in

```plaintext
[X in L | B: BOOL]
[E | X in L]
[E | X in L and B]
where L: LIST, B: BOOL, E: expression
returns LIST
```

Description

See “List Constructors” (III-5.2 pg.340) for a full description.

```plaintext
/**/ [N in 1..10 | IsPrime(N)];
[2, 3, 5, 7]
/**/ [N^2 | N in 1..10 and IsPrime(N)];
[4, 9, 25, 49]
```

See Also: List Constructors(III-5.2 pg.340), IsIn(I-9.47 pg.136)

I-9.16 incr, decr

```plaintext
incr(ref X: INT)
decr(ref X: INT)
```

Description

“incr(ref X)” adds 1 to the value of “X”. “decr(ref X)” subtracts 1 from the value of “X”.
These functions are useful when counting objects or adjusting pointers.

```plaintext
/**/ L := [(10^k-1)/9 | k in 1..99];
/**/ NPrimes := 0;
/**/ Foreach N in L Do
/**/ If IsPrime(N) Then incr(ref NPrimes); EndIf;
/**/ EndForeach;
/**/ PrintLn "The list L contains ", NPrimes, " primes."
The list L contains 3 primes.
```
I-9.17  indent

syntax

indent(X: OBJECT)
indent(X: OBJECT, N: INT)

Description

This function prints the argument “X” splitting it into several lines: a “LIST” or “IDEAL” is printed one element per line, a “RECORD” one field per line.

The second optional argument is for setting the level of recursive indentation; it is useful for example when printing a list of records.

example

```/**/ L := [1,2] >< [3,4];
/**/ L;
[[1, 3], [1, 4], [2, 3], [2, 4]]
/**/ indent(L);
[
[1, 3],
[1, 4],
[2, 3],
[2, 4]
]
/**/ indent(L,2);
[
[1,
3
],
--( Further output )--
[
2,
4
]
]
/**/ indent(record[B:=1,A:=2]);
record[
A := 2,
B := 1
]
```

See Also: format(I-6.15 pg.90)

I-9.18  indet

syntax

indet(R: RING, N: INT): RINGELEM

Description

This function returns the N-th indeterminate of the current ring.
/**/ Use R ::= QQ[x,y,z];
/**/ indet(R, 2);
y
See Also: IndetSubscripts(I-9.22 pg.126), IndetIndex(I-9.19 pg.124), IndetName(I-9.20 pg.124), indets(I-9.21 pg.125), NumIndets(I-14.32 pg.192)

I-9.19 IndetIndex

syntax

IndetIndex(X: RINGELEM): INT

Description

This function returns the position in which the indeterminate is listed when the corresponding ring was created.

/**/ Use R ::= QQ[x,y,z];
/**/ IndetIndex(y);
2
/**/ Use R ::= QQ[x[1..2,1..2],y[1..2]];
/**/ Indets(R);
[x[1,1], x[1,2], x[2,1], x[2,2], y[1], y[2]]
/**/ IndetIndex(x[2,1]);
3
/**/ S ::= QQ[a,b,c];
/**/ IndetIndex(RingElem(S, "b"));
2

See Also: indet(I-9.18 pg.123), IndetSubscripts(I-9.22 pg.126), IndetName(I-9.20 pg.124), indets(I-9.21 pg.125), NumIndets(I-14.32 pg.192), UnivariateIndetIndex(I-21.1 pg.279)

I-9.20 IndetName

syntax

IndetName(X: RINGELEM): STRING

Description

This function returns the name of the indeterminate X as a string (i.e. the letter without the indices).

/**/ Use R ::= QQ[x,y,z];
/**/ IndetName(indet(R, 2));
y
/**/ type(It);
STRING
/**/ Use R ::= QQ[a, x[1..3]];
indets(R: RING): LIST
indets(R: RING, X: STRING): LIST

Description

With one argument (a polynomial ring), this function returns the list of indeterminates of that polynomial ring. With two arguments (the second a STRING), it returns the list of all indeterminates whose name is the given string. The indeterminates in the list appear in order of increasing index (see the function “IndetIndex”).

This function used to be called “IndetsCalled” up to version 5.0.3, and “AllIndetsCalled” in CoCoA-4. Additionally, up to version 4.7.3 you could get this list just by giving the name, e.g. “Use QQ[x[0..4]]; x;” but this syntax is no longer allowed because it is ambiguous: “x[2];” is different from “X := x; X[2];”

example

```plaintext
/**/ S ::= QQ[x,y,z];
/**/ Use R ::= QQ[a,b];
/**/ indets(CurrentRing);
[a, b]
/**/ indets(S);
[x, y, z]
/**/ indets(S,"x");
[x]
/**/ RingElem(S,"x"); -- works also if R is not a polynomial ring
x
/**/ Use R ::= QQ[x[1..4],a[1..2,1..3]]; indets(R,"x");
[x[1], x[2], x[3], x[4]]
/**/ indets(R,"a");
[a[1,1], a[1,2], a[1,3], a[2,1], a[2,2], a[2,3]]
/**/ indets(R,"b"); -- empty list if no indet has a matching head
[]
```

I-9.22  IndetSubscripts

**syntax**

\[
\text{IndetSubscripts}(X: \text{RINGELEM}): \text{LIST}
\]

**Description**

This function returns the subscripts of the name of the argument, an indeterminate (used to be called “IndetInd”).

Please note the difference with “IndetIndex” (I-9.19 pg.124).

**example**

```plaintext
/**/ Use R ::= QQ[x[1..3,1..2],y,z];
/**/ IndetSubscripts(x[3,2]);
[3, 2]
/**/ IndetSubscripts(y);
[]
/**/ IndetIndex(RingElem(R, ["x",3,2]));
6
/**/ IndetSubscripts(RingElem(R, ["x",3,2]));
[3, 2]
```

See Also: indet(I-9.18 pg.123), IndetIndex(I-9.19 pg.124), IndetName(I-9.20 pg.124), IndetSymbols(I-9.23 pg.126), indets(I-9.21 pg.125), NumIndets(I-14.32 pg.192)

I-9.23  IndetSymbols

**syntax**

\[
\text{IndetSymbols}(P: \text{RING}): \text{RECORD}
\]

**Description**

This function returns the list of the symbols in a polynomial ring. A symbol is a record “with” head (as “IndetName” (I-9.20 pg.124)) and “indices” (as “IndetSubscripts” (I-9.22 pg.126))

**example**

```plaintext
/**/ Use R ::= QQ[x,y,z];
/**/ indent(IndetSymbols(R));
[ record[head := "x", indices := []],
  record[head := "y", indices := []],
  record[head := "z", indices := []]
]
/**/ Use R ::= QQ[a, x[1..3]];
/**/ indent(IndetSymbols(R));
[ record[head := "a", indices := []],
  record[head := "x", indices := [1]],
  record[head := "x", indices := [2]],
  record[head := "x", indices := [3]]
]
```

See Also: indet(I-9.18 pg.123), IndetSubscripts(I-9.22 pg.126), IndetIndex(I-9.19 pg.124), IndetName(I-9.20 pg.124), NumIndets(I-14.32 pg.192), SymbolRange(I-19.45 pg.267)
I-9.24  InducedHom

**syntax**

InducedHom(RmodI: RING, phi: RINGHOM): RINGHOM

**Description**

“InducedHom(RmodI, phi)” – where “RmodI” is a QuotientRing, and “phi” is a homomorphism “R --> S” (which must have “BaseRing(RmodI)” as its “domain” (I-4.22 pg.74), and whose “ker” (I-11.1 pg.153) must contain “DefiningIdeal(RmodI)” gives the homomorphism “R/I --> S” induced by “phi”

“InducedHom(FrF, phi)” – may be partial where “FrF” is a FractionField, gives the homomorphism induced by “phi” (which must have the base ring of FrF as its domain). Note that the resulting homomorphism may be only partial (e.g. if ker(phi) is non-trivial, or if the codomain is not a field).

**example**

```plaintext
/**/ Use R ::= QQ[x,y];
/**/ RmodI := NewQuotientRing(R, ideal(x^2-1));
/**/ Use S ::= QQ[a,b,c];
/**/ SmodJ := NewQuotientRing(S, ideal(a^2-1));
/**/ phi := PolyAlgebraHom(R,S,[a,b]);
/**/ Use R;
/**/ phi(x);
a
/**/ RingOf(phi(x)) = S;
true
/**/ psi := CanonicalHom(S,SmodJ)(phi); -- composition of homomorphisms
/**/ psi(x);
(a)
/**/ RingOf(psi(x)) = SmodJ;
true
/**/ theta := InducedHom(RmodI, psi);
/**/ Use RmodI;
/**/ theta(x);
(a)
```

See Also: domain(I-4.22 pg.74), codomain(I-3.19 pg.47), Composition of RINGHOM(III-10.2 pg.359), BaseRing(I-2.1 pg.33), DefiningIdeal(I-4.5 pg.66), NewQuotientRing(I-14.9 pg.182), NewFractionField(I-14.1 pg.179), CanonicalHom(I-3.2 pg.41), PolyAlgebraHom(I-16.14 pg.205), PolyRingHom(I-16.15 pg.206)

I-9.25  InitialIdeal

**syntax**

InitialIdeal(I: IDEAL, Inds: LIST): IDEAL

**Description**

The “initial form” of a polynomial F is the homogeneous component of F of the lowest degree (in contrast with the “leading form”, see “LF” (I-12.8 pg.158), “DF” (I-4.14 pg.70)). The “initial ideal” of the ideal “I” is the ideal generated by the initial forms of all polynomials in “I”. It is also called “tangent cone” (which strictly is the variety defined by the initial ideal).

The implementation is based on Lazard’s method (see Kreuzer-Robbiano, Commutative Computer Algebra II, pg.463).
See Also: InitialIdeal(I-9.25 pg.127), PrimaryHilbertSeries(I-16.22 pg.209)

I-9.26 insert [OBSOLESCENT]

**syntax**

[OBSOLESCENT] insert(ref L: LIST, N: INT, E: OBJECT)

**Description**

This function inserts “E” into “L” as the “N”-th component. Kept just for backward compatibility, it is strongly discouraged for its intrinsic inefficiency. See “append” (I-1.11 pg.29).

**example**

```plaintext
/**/ L := ["a","b","d","e"]);
/**/ insert(ref L,3,"c");
/**/ L;
["a", "b", "c", "d", "e"]
```

See Also: append(I-1.11 pg.29), remove(I-18.29 pg.234)

I-9.27 Interpolate

**syntax**

Interpolate(Points: LIST, Values: LIST): RINGELEM

where Points is a list of lists of coefficients representing a set of ‘‘{\it distinct}’’ points and Values is a list of the same size containing numbers from the coefficient ring.

**Description**

***** NOT YET IMPLEMENTED *****

This function returns a multivariate polynomial which takes given values at a given set of points.  

NOTE:  
* the current ring must have at least as many indeterminates as the dimension of the space in which the points lie;  
* the base field for the space in which the points lie is taken to be the coefficient ring, which should be a field;  
* in the polynomials returned, the first coordinate in the space is taken to correspond to the first indeterminate, the second to the second, and so on;
* if the number of points is large, say 100 or more, the returned value can be very large. To avoid possible problems when printing such values as a single item we recommend printing out the elements one at a time as in this example:

```plaintext
example
X := Interpolate(Pts, Vals);
Foreach Element In X Do
  PrintLn Element;
EndForeach;

Use QQ[x,y];
Points := [[1/2, 2], [3/4, 4], [5, 6/11], [-1/2, -2]];
Values := [1/2, 1/3, 1/5, -1/2];
F := Interpolate(Points, Values);
F;
-46849/834000y^2 - 1547/52125x + 13418/52125y + 46849/208500
-----------------------------
[Eval(F, P) | P In Points] = Values; -- check
True
-----------------------------
```

I-9.28 interreduce, interreduced

** syntax **

```plaintext
interreduce(ref L: LIST of RINGELEM)
interreduced(L: LIST of RINGELEM): LIST of RINGELEM
```

** Description **

These functions reduce each polynomial using the other polynomials as reduction rules. The process terminates when each is in normal form with respect to the others. The function “interreduce” takes a variable containing a list and overwrites that variable with the interreduced list. The second returns an interreduced list without affecting its arguments.

```plaintext
/**/ interreduced([x^3-x*y^2+y*z, x*y, z]);
[x*y, z, x^3]
/**/ L := [x^3-x*y^2+y*z, x*y, z];
/**/ interreduce(ref L);
/**/ L;
[x*y, z, x^3]
```

I-9.29 intersection

** syntax **

```plaintext
intersection(A: LIST, B: LIST): LIST
intersection(A: IDEAL, B: LIST): LIST
intersection(A: LIST, B: IDEAL): LIST
intersection(A: IDEAL, B: IDEAL): IDEAL
```

** Description **

This function returns the intersection of “A” and “B”.

```plaintext
/**/ intersection([x^3-x*y^2+y*z, x*y, z], [x*y, z, x^3]);
[x*y, z]
```
The coefficient ring must be a field.

NOTE: To compute the intersection of ideals corresponding to zero-dimensional schemes, see the commands "GBM" (I-7.3 pg.96) and "HGBM" (I-8.2 pg.105).

```plaintext
/**/ Use R ::= QQ[x,y,z];
/**/ intersection(ideal(x,y,z), ideal(x*y));
ideal(x*y)
/**/ intersection(["a","b","c"], ["b","c","d"], ["b","c"])
-----------------------------

See Also: GBM(I-7.3 pg.96), HGBM(I-8.2 pg.105), IntersectList(I-9.30 pg.130)

I-9.30 IntersectList

Syntax

IntersectionList(L: LIST of LIST): LIST
IntersectionList(L: LIST of IDEAL): IDEAL
IntersectionList(L: LIST of MODULE): MODULE

Description

This function returns the intersection of all elements in “L”. Generalizes “intersection” (I-9.29 pg.129).

```plaintext
/**/ Use R ::= QQ[x,y,z];
/**/ Points := [[0,0],[1,0],[0,1],[1,1]]; -- a list of points in the plane
/**/ IntersectionList([ ideal(x-P[1]*z, y-P[2]*z) | P In Points]);
ideal(y^2 - y*z, x^2 - x*z)
/**/ IntersectionList([ 1..7, 3..10, 0..5 ]);[3, 4, 5]

See Also: intersection(I-9.29 pg.129), IdealOfProjectivePoints(I-9.6 pg.117), IdealOfPoints(I-9.5 pg.116), HGBM(I-8.2 pg.105), intersection(I-9.29 pg.129)

I-9.31 inverse

Syntax

inverse(X: MAT): MAT

Description

This function computes the multiplicative inverse of its argument. It is included for use when writing inverse(X) comes more naturally than writing “X^{-1}”, though both notations are functionally equivalent.

```plaintext
/**/ inverse(mat(QQ, [[1,2], [3,4]]));
matrix([[-2, 1],
        [3/2, -1/2]]

See Also: adj(I-1.2 pg.25)
I-9.32 InverseSystem

Syntax

```
InverseSystem(I: IDEAL, D: INT): LIST of RINGELEM
```

Description

Thanks to Enrico Carlini.

Given an ideal of derivations “I”, and an integer “D”, this function computes the degree “D” part of the inverse system of “I”.

For the sake of simplicity Forms/Polynomials and Derivations live in the same ring, the distinction between them is purely formal.

```
/**/ Use QQ[x,y,z];
/**/ InverseSystem(ideal(x^3+x*y*z), 3);
[z^-3, y*z^-2, x*z^-2, y^-2*z, x^-2*z, y^-3, x*y^-2, x^-2*y, x^-3 - 6*x*y*z]
```

See Also: DerivationAction(I-4.11 pg.69), PerpIdealOfForm(I-16.6 pg.202)

I-9.33 IO.SprintTrunc

Syntax

```
$io.SprintTrunc(E: OBJECT, N: INT): STRING
```

Description

***** NOT YET IMPLEMENTED *****

This function works like “sprint” (I-19.24 pg.257), turning the value of the expression E into a string, but if the string has length greater than N-1, it is truncated and the string “...” is concatenated. This function is useful in formatting reports of results.

```
Use R ::= QQ[x,y];
I := ideal(x,y);
$io.SprintTrunc(I,4);
Idea...
-----------------------------------
```

See Also: format(I-6.15 pg.90), sprint(I-19.24 pg.257)

I-9.34 iroot

Syntax

```
iroot(N: INT, R: INT): INT
```

Description

This function computes the R-th root of an integer. If the argument is not a perfect R-th power it returns the integer part of the root.

```
/**/ iroot(25, 2);
5
```
/**/ iroot(99, 3);
4
/**/ iroot(-1, 3);
-1

See Also: ILogBase(I-9.9 pg.119)

I-9.35  IsAntiSymmetric

<table>
<thead>
<tr>
<th>syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>IsAntiSymmetric(M: MAT): BOOL</td>
</tr>
</tbody>
</table>

Description
This function tests whether the square matrix “M” is anti-symmetric.

<table>
<thead>
<tr>
<th>example</th>
</tr>
</thead>
</table>
| /**/ M := mat([[0, 1, 2], [-1, 0, 3], [-2, -3, 0]]);
/**/ IsAntiSymmetric(M);
true |

See Also: IsSymmetric(I-9.67 pg.144)

I-9.36  IsConstant

<table>
<thead>
<tr>
<th>syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>IsConstant(X: RINGELEM): BOOL</td>
</tr>
</tbody>
</table>

Description
This function tests whether the value of a RINGELEM of a polynomial ring actually lies in the image of the coefficient ring. It is equivalent to checking that the degree in each indeterminate is 0.

<table>
<thead>
<tr>
<th>example</th>
</tr>
</thead>
</table>
| /**/ QQx ::= QQ[x];
/**/ Use QQx[y,z];
/**/ IsConstant(y+1);
false
/**/ IsConstant(x+1);
true |

See Also: indets(I-9.21 pg.125)

I-9.37  IsContained

<table>
<thead>
<tr>
<th>syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>IsContained(A: IDEAL, B: IDEAL): BOOL</td>
</tr>
<tr>
<td>IsContained(A: MODULE, B: MODULE): BOOL</td>
</tr>
</tbody>
</table>

Description
This function tests whether A is contained in B. Was “<=” in CoCoA-4: this syntax is no longer supported.
I-9.38  IsDefined

** syntax **

IsDefined(E)

** Description **

This function returns true if “E” is defined, otherwise it returns false. Typically, it is used to check if a name has already been assigned.

To know if a field in a record has been assigned use “fields” (I-6.5 pg.85).

/**/ IsDefined(MyVariable);
false
/**/ MyVariable := 3;
/**/ IsDefined(MyVariable);
true

See Also:  fields(I-6.5 pg.85)

I-9.39  IsDiagonal

** syntax **

IsDiagonal(M: MAT): BOOL

** Description **

This function tests whether the square matrix M is diagonal.

/**/ M := mat([[0, 1, 2],[-1, 0, 3],[-2, -3, 0]]);
/**/ IsDiagonal(M);
false

See Also:  IsSymmetric(I-9.67 pg.144), DiagMat(I-4.15 pg.71)

I-9.40  IsDivisible

** syntax **

IsDivisible(A: RINGELEM, B: RINGELEM): BOOL
Description

This function says whether “A” is divisible by “B”; it returns “true” if so, otherwise “false”.

```plaintext
/**/ Use QQ[x,y,z];
/**/ IsDivisible(x, x^2*(y-1));
false
/**/ IsDivisible(x^2*(y-1), x);
true
```

See Also: FactorMultiplicity(I-6.3 pg.84)

I-9.41  IsElem

```plaintext
IsElem(A: RINGELEM, B: IDEAL): BOOL
IsElem(A: MODULEELEM, B: MODULE): BOOL
```

Description

This function tests whether A is an element of B. Same as the command “IsIn” (I-9.47 pg.136), but works on fewer types: it is in CoCoA-5 for compatibility with the C++ function in CoCoALib.

```plaintext
/**/ Use QQ[x,y,z];
/**/ IsElem(x, ideal(x+y, x-y));
ture
/**/ x IsIn ideal(x+y, x-y);
ture
```

See Also: IsIn(I-9.47 pg.136)

I-9.42  IsEven, IsOdd

```plaintext
IsEven(N: INT): BOOL
IsOdd(N: INT): BOOL
```

Description

These functions test whether an integer is even or odd.

```plaintext
/**/ IsEven(3);
false
/**/ IsOdd(3);
true
```

See Also: IsZero(I-9.72 pg.146)

I-9.43  IsFactorClosed

```plaintext
IsFactorClosed(L: LIST of power products): BOOL
```
Description

A set of power products is factor closed iff it contains every factor of every one of its elements. This function checks whether the given set is factor closed (also known as "order-ideal"). It is an error if L is empty.

```plaintext
/**/ use P ::= QQ[x,y,z];
/**/ IsFactorClosed([1, x, x^2]);
true
/**/ IsFactorClosed([one(P), y^2]);
false
```

See Also: QuotientBasis(I-17.3 pg.218), LT(I-12.21 pg.164), TmpNBM(I-20.8 pg.274), IsStronglyStable(I-9.64 pg.143)

**I-9.44 IsField**

```plaintext
IsField(R: RING): BOOL
```

Description

This function tests whether a ring is a field.

```plaintext
/**/ IsField(ZZ);
false
/**/ IsField(QQ);
true
```

See Also: IsFiniteField(I-9.45 pg.135)

**I-9.45 IsFiniteField**

```plaintext
IsFiniteField(R: RING): BOOL
```

Description

This function tests whether a ring is a finite field.

```plaintext
/**/ IsFiniteField(ZZ);
false
/**/ IsFiniteField(QQ);
false
/**/ Fp::=ZZ/(7); IsFiniteField(Fp);
true
```

I-9.46  IsHomog

<table>
<thead>
<tr>
<th>syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>IsHomog(F: RINGELEM</td>
</tr>
<tr>
<td>IsHomog(L: LIST): BOOL</td>
</tr>
<tr>
<td>IsHomog(I: IDEAL</td>
</tr>
</tbody>
</table>

**Description**

The first form of this function returns True if F is homogeneous. The second form returns True if every element of L is homogeneous. Otherwise, they return False. The third form returns True if the ideal/module can be generated by homogeneous elements, and False if not. Homogeneity is with respect to the first row of the weights matrix.

**NOTE:** when the grading dimension is 0 everything is trivially true. For safety reasons (from version 5.0.3) “IsHomog” throws an error in this case, e.g. “IsHomog(x-1)” gives error instead of a possibly misleading “true”.

```plaintext
/**/ Use R ::= QQ[x,y];
/**/ IsHomog(x^2-x*y);
true
/**/ IsHomog(x-y^2);
false
/**/ IsHomog([x^2-x*y, x-y^2]);
false
/**/ R := NewPolyRing(QQ, "x,y", mat([[2,3],[1,2]]), 1);
/**/ Use R;
/**/ IsHomog(x^3*y^2+y^4);
true
/**/ R := NewPolyRing(QQ, "x,y", mat([[2,3],[1,2]]), 2);
/**/ Use R;
/**/ IsHomog(x^3*y^2+y^4);
false
/**/ Use R ::= QQ[x,y];
/**/ IsHomog(ideal(x^2+y,y));
true
/**/ Use R ::= QQ[x,y], Lex; -- note: GradingDim = 0
-- /**/ IsHomog(x-1); -- !!! ERROR !!! instead of "true"
```

See Also: deg(I-4.6 pg.66), wdeg(I-23.1 pg.285)

I-9.47  IsIn

<table>
<thead>
<tr>
<th>syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>X IsIn Y (return BOOL)</td>
</tr>
</tbody>
</table>

**Description**

The semantics of “IsIn” is explained in the following table:
I-9.48. IsIndet

**syntax**

IsIndet(X: RINGELEM): BOOL

**Description**

This function tests whether “X” is an indeterminate. If so, it returns “true”; otherwise it returns “false”. An error is signalled if “X” is not a “RINGELEM” or if “RingOf(X)” is not a polynomial ring.

**example**

```plaintext
/**/ Use QQ[x,y,z];
/**/ IsIndet(x);
true
/**/ IsIndet(x-1);
false
```

I-9.49. IsInjective

**syntax**

IsInjective(phi: RINGHOM): BOOL

**Description**

This function checks if a RINGHOM is injective.

**example**

```plaintext
/**/ QQxyz ::= QQ[x,y,z];
/**/ QQab ::= QQ[a,b];
/**/ Use QQab;
/**/ phi := PolyAlgebraHom(QQxyz, QQab, [a+1, a*b+3, b^2]);
/**/ IsInjective(phi);
false
/**/ ker(phi);
ideal(-x^2*z +y^2 +2*x*z -6*y -z +9)
 /**/ IsSurjective(phi);
false
/**/ Use QQab;
/**/ PreImage(phi, b);
record[IsInImage := false, ker := ideal(-x^2*z +y^2 +2*x*z -6*y -z +9)]
/**/ indent(PreImage(phi, a^2));
record[
    IsInImage := true,
]```
Chapter I-9. I

OnePreImage := x^2 - 2*x +1,
ker := ideal(-x^2*z +y^2 +2*x*z -6*y -z +9)
]
/**/ phi(ReadExpr(QQxyz, "x^2 - 2*x + 1"));
a^2
/**/ phi(ReadExpr(QQxyz, "x^2 - 2*x + 1 + (-x^2*z +y^2 +2*x*z -6*y -z +9)"));
a^2

See Also: ker(I-11.1 pg.153), IsSurjective(I-9.66 pg.144)

I-9.50 IsInRadical

Description
This function tests whether the first argument, a polynomial or an ideal, is contained in the radical of the second argument, an ideal.
This function is much faster than asking “F IsIn Radical(I);”.

example

/**/ Use QQ[x,y,z];
/**/ I := ideal(x^6*y^4, z);
/**/ IsInRadical(x*y, I);
true
/**/ IsInRadical(ideal(x,y), I);
false
/**/ MinPowerInIdeal(x*y, I);
6

See Also: MinPowerInIdeal(I-13.19 pg.175), radical(I-18.1 pg.221)

I-9.51 IsInSubalgebra [OBSOLETE]

[OBSOLETE]

Description
See “SubalgebraRepr” (I-19.34 pg.261).
See Also: SubalgebraRepr(I-19.34 pg.261)

I-9.52 IsLexSegment

IsLexSegment(I: IDEAL): BOOL
Description
This function tests whether the monomial ideal I is a lex-segment ideal.

```plaintext
/**/ Use R ::= QQ[x,y,z];
/**/ I := ideal(x*y^3, y^4, x^3, x^2*y, x^2*z);
/**/ IsLexSegment(I);
false
```

See Also: IsStable(I-9.62 pg.142), IsStronglyStable(I-9.64 pg.143), LexSegmentIdeal(I-12.7 pg.158)

I-9.53 IsNumber [OBSOLETE]

Description

I-9.54 IsOne

```plaintext
IsOne(X: OBJECT): BOOL
```

Description
This function tests whether its argument is one; the argument can be of almost any type for which “one” makes sense.

```plaintext
/**/ IsOne(23);
false
/**/ IsOne(3/3);
true
/**/ Use R ::= QQ[x,y,z];
/**/ IsOne(1);
false
/**/ IsOne(ideal(x^2, x^2-1));
false
```

See Also: IsEven, IsOdd(I-9.42 pg.134), one(I-15.1 pg.195), IsZero(I-9.72 pg.146)

I-9.55 IsPositiveGrading

```plaintext
IsPositiveGrading(M: MAT): BOOL
IsPositiveGrading(M: MAT,N: INT): BOOL
```
**Description**

This function determines whether a matrix defines a positive grading, i.e. foreach column the first nonnegative entry is positive.

```plaintext
/**/ IsPositiveGrading(LexMat(5));
true
/**/ IsPositiveGrading(LexMat(5),3); --considering only the first three rows
false
/**/ IsPositiveGrading(mat([[0,2,3], [1, -1, 0]]));
true
/**/ IsPositiveGrading(mat([[1,1], [0,-1], [-1, 0]]));
true
```

**See Also:** HilbertSeriesMultiDeg(I-8.8 pg.109)

**I-9.56 IsPrime**

**Syntax**

```plaintext
IsPrime(N: INT): BOOL
```

**Description**

This function determines whether a positive integer is prime; if N is not positive, an error is signalled. This function may be extremely slow when N is a large prime; in practice it is usually better to call IsProbPrime.

For the curious: currently, the function first performs a probabilistic check (Miller-Rabin), if that passes, it then verifies primality (via Lucas test).

```plaintext
/**/ IsPrime(32003);
true
/**/ IsPrime(10^100);
false
```

**See Also:** IsProbPrime(I-9.57 pg.140), NextPrime(I-14.12 pg.184)

**I-9.57 IsProbPrime**

**Syntax**

```plaintext
IsProbPrime(N: INT): BOOL
```

**Description**

This function returns True if its integer argument passes a fairly stringent primality test; otherwise it returns False. There is a very small chance of the function returning True even though the argument is composite; if it returns False, we are certain that the argument is composite. Some people call it a compositeness test.

```plaintext
/**/ IsProbPrime(2);
true
```
/**/ IsProbPrime(1111111111111111111); true
/**/ [N in 1..1111 | IsProbPrime((10^N-1)/9)]; -- only five values are known
[2, 19, 23, 317, 1031] -- next might be 49081

See Also: IsPrime(I-9.56 pg.140), NextProbPrime(I-14.13 pg.184)

I-9.58  IsPthPower

IsPthPower(X: RINGELEM): BOOL

Description

This function determines whether a polynomial over a finite field (of char p) is a p-th power. If the coefficient ring is not a finite field then an error is signalled.

/**/ Use ZZ/(7)[x];
/**/ IsPthPower(x^7+3);
ture
/**/ IsPthPower(x^6+3);
false

See Also: IsFiniteField(I-9.45 pg.135), PthRoot(I-16.33 pg.215)

I-9.59  IsQQ

IsQQ(R: RING): BOOL

Description

This function tests whether a ring is the ring of rationals.

/**/ R ::= QQ[x,y];
/**/ IsQQ(CoeffRing(R));
true;

See Also: QQ(I-17.1 pg.217), RingQQ(I-18.42 pg.240), IsZZ(I-9.76 pg.148)

I-9.60  isqrt

isqrt(N: INT): INT

Description

This function computes the square root of an integer. If the argument is not a perfect square it returns the integer part of the square root.
I-9.61  IsQuotientRing

IsQuotientRing(R: RING): BOOL

Description

This function tests whether a ring is a quotient ring; it returns “true” if the ring is a quotient ring.

example

/**/ Use R ::= QQ[x,y];
/**/ S := R/ideal(x);
/**/ IsQuotientRing(S);
true;

See Also: DefiningIdeal(I-4.5 pg.66)

I-9.62  IsStable

IsStable(I: IDEAL): BOOL

Description

This function tests whether the monomial ideal I is stable.

example

/**/ Use R ::= QQ[x,y,z];
/**/ I := ideal(x*y^3, y^4, x^3, x^2*y, x^2*z);
/**/ IsStable(I);
true

See Also: IsLexSegment(I-9.52 pg.138), IsStronglyStable(I-9.64 pg.143), LexSegmentIdeal(I-12.7 pg.158)

I-9.63  IsStdGraded

IsStdGraded(P: RING): BOOL
**Description**

This function tests whether “P” is standard graded, “i.e.” “GradingDim” is 1 and all indeterminates in “P” have degree 1.

```plaintext
/**/ P ::= QQ[x,y,z];
/**/ IsStdGraded(P);
true
/**/ P ::= QQ[x,y,z], lex;
/**/ IsStdGraded(P);
false
/**/ P := NewPolyRing(QQ, "x,y", mat([[2,3],[1,2]]), 1);
/**/ IsStdGraded(P);
false
```

**See Also:** NewPolyRing(I-14.8 pg.182), wdeg(I-23.1 pg.285)

---

**I-9.64 IsStronglyStable**

**syntax**

```plaintext
IsStronglyStable(I: IDEAL): BOOL
```

**Description**

This function tests whether the monomial ideal I is strongly stable (Borel-fixed in characteristic 0).

```plaintext
/**/ Use R ::= QQ[x,y,z];
/**/ I := ideal(x*y^3, y^4, x^3, x^2*y, x^2*z);
/**/ IsStronglyStable(I);
true
```

**See Also:** IsLexSegment(I-9.52 pg.138), IsStable(I-9.62 pg.142)

---

**I-9.65 IsSubset**

**syntax**

```plaintext
IsSubset(L: LIST, M: LIST): BOOL
```

**Description**

This function returns “true” if “MakeSet(L)” is contained in “MakeSet(M)”; otherwise it returns “false”.

```plaintext
/**/ IsSubset([1,1,2],[1,2,3,"a"]);
true
/**/ IsSubset([1,2],["a","b"]);
false
/**/ IsSubset([],[1,2]);
true
```

**See Also:** IsContained(I-9.37 pg.132), IsIn(I-9.47 pg.136), EqSet(I-5.6 pg.77), MakeSet(I-13.3 pg.168), subsets(I-19.38 pg.263)
I-9.66  IsSurjective

**syntax**

\[ \text{IsSurjective}(\phi: \text{RINGHOM}): \text{BOOL} \]

**Description**

This function checks if a RINGHOM is surjective.

```plaintext
/**/ QQxyz ::= QQ[x,y,z];
/**/ QQab ::= QQ[a,b];
/**/ Use QQab;
/**/ phi := PolyAlgebraHom(QQxyz, QQab, [a+1, a*b+3, b^2]);
ideal(-x^2*z +y^2 +2*x*z -6*y -z +9)
/**/ IsSurjective(phi);
false
/**/ PreImage(phi, b);
record[IsInImage := false, ker := ideal(-x^2*z +y^2 +2*x*z -6*y -z +9)]
```

**See Also:** ker(I-11.1 pg.153), IsInjective(I-9.49 pg.137), PreImage(I-16.17 pg.207)

I-9.67 IsSymmetric

**syntax**

\[ \text{IsSymmetric}(M: \text{MAT}): \text{BOOL} \]

**Description**

This function tests whether the square matrix “M” is symmetric.

```plaintext
/**/ M := mat([[1, 2, 3], [2, 4, 5], [3, 5, 6]]);
/**/ IsSymmetric(M);
true
```

**See Also:** IsAntiSymmetric(I-9.35 pg.132)

I-9.68 IsTerm

**syntax**

\[ \text{IsTerm}(X: \text{RINGELEM} | \text{MODULEELEM}): \text{BOOL} \]

**Description**

The function determines whether X is a term. For a polynomial, a “term” is a power-product, i.e., a product of indeterminates. Thus, \( x \cdot y^2 \cdot z \) is a term, while \( 4 \cdot x \cdot y^2 \cdot z \) and \( x \cdot y + z^3 \) are not. For a vector, a term is a power-product times a standard basis vector, e.g., \( (0, x \cdot y^2 \cdot z, 0) \).

```plaintext
/**/ Use R ::= QQ[x,y,z];
/**/ IsTerm(x+y^-2);
false
```
**/ IsTerm(x^3*y*z^2);  
  true  
/**/ IsTerm(5*x^3*y*z^2);  
  false  
/**/ R2 := NewFreeModule(R,2);  
--/**/ IsTerm(ModuleElem(R2, [0,x*z]));  
--true  
--/**/ IsTerm(ModuleElem(R2, [x,y]));  
--true  
/**/ IsTermOrdering(LexMat(5));  
  true  
/**/ IsTermOrdering(StdDegRevLexMat(5));  
  true  
/**/ IsTermOrdering(RevLexMat(5));  
  false

I-9.69  IsTermOrdering

syntax

IsTermOrdering(M: MAT): BOOL

Description

This function determines whether a square matrix defines a term-ordering, i.e. if its determinant is non-zero and if foreach column the first nonnegative entry is positive.

/**/ IsTermOrdering(LexMat(5));  
  true  
/**/ IsTermOrdering(StdDegRevLexMat(5));  
  true  
/**/ IsTermOrdering(RevLexMat(5));  
  false

See Also: NewPolyRing(I-14.8 pg.182), OrdMat(I-15.10 pg.199)

I-9.70  IsTree5

syntax

IsTree5(L: LIST): [BOOL, LIST ]  
IsTree5(L: LIST, "NOOPT"): [BOOL, LIST]  
IsTree5(L: LIST, "OPT"): [BOOL, LIST]  
IsTree5(L: LIST, "CS_NOOPT"): [BOOL, LIST]  
IsTree5(L: LIST, "CS_OPT"): [BOOL, LIST]

Description

***** NOT YET IMPLEMENTED *****

This function is implemented in CoCoALib.

This function tests whether the facet complex described by the list L of square free power products is a tree, plus a list which:

- is empty if L is a tree
- contains three elements of a cycle of \( L \) if \( L \) is not a tree.

Four options \("\text{NOOPT}\", \"\text{OPT}\", \"\text{CS\_NOOPT}\", \"\text{CS\_OPT}\"\) are available as second argument, specifying different algorithms; the default is \("\text{CS\_OPT}\"\).


```plaintext
example

Use R ::= QQ[x,y,z,t];
D := [x*y, y*z, z*t, t*x];
IsTree5(D);
[False, [xy, xt, yt]]
-------------------------------
IsTree5([xy, yz, zt]);
[True, [ ]]
-------------------------------
```

### I-9.71 IsTrueGCDDomain

```plaintext
IsTrueGCDDomain(R: RING): BOOL
```

**Description**

This function tests whether a ring is a (true) GCD domain but not a field. CoCoA can compute GCDs of elements of a true GCD domain.

```plaintext
/**/ IsTrueGCDDomain(ZZ);
true
/**/ IsTrueGCDDomain(QQ);
false
```

**See Also:** IsField(I-9.44 pg.135)

### I-9.72 IsZero

```plaintext
IsZero(X: OBJECT): BOOL
```

**Description**

This function tests whether its argument is zero; the argument can be of almost any type for which \("\text{zero}\"\) makes sense.

```plaintext
/**/ IsZero(23);
false
/**/ IsZero(3-3);
true
/**/ Use R ::= QQ[x,y,z];
/**/ IsZero(x^2+3*y-1);
false
/**/ IsZero(ideal(x^2,x*y^3));
false
```
/**/ F := NewFreeModule(R, 3);
/**/ zero(F);
[0, 0, 0]
/**/ IsZero(zero(F));
true
/**/ IsZero(matrix([[0,0,0], [0,0,0]]));
true

See Also: IsEven, IsOdd(I-9.42 pg.134), IsOne(I-9.54 pg.139), zero(I-25.1 pg.291), ZeroMat(I-25.2 pg.291)

### I-9.73 IsZeroCol, IsZeroRow

** syntax **

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IsZeroCol(M: MAT, N: INT): BOOL</td>
<td>This function tests whether all entries in the “N”-th column of “M” are zero.</td>
</tr>
<tr>
<td>IsZeroRow(M: MAT, N: INT): BOOL</td>
<td>This function tests whether all entries in the “N”-th row of “M” are zero.</td>
</tr>
</tbody>
</table>

** example **

/**/ IsZeroRow(matrix([[1,0,0], [0,0,0]]), 1);  
false
/**/ IsZeroCol(matrix([[1,0,0], [0,0,0]]), 2);  
true

See Also: IsZero(I-9.72 pg.146), ZeroMat(I-25.2 pg.291)

### I-9.74 IsZeroDim

** syntax **

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IsZeroDim(I: IDEAL): BOOL</td>
<td>This function tests whether its argument is zero-dimensional.</td>
</tr>
</tbody>
</table>

** example **

/**/ Use QQ[x,y,z];  
/**/ IsZeroDim(ideal(x));  
false
/**/ IsZeroDim(ideal(x^3, y^4-x ,z-3));  
true
/**/ IsZeroDim(ideal(x^2, x*y^3));  
false

See Also: dim(I-4.17 pg.72)

### I-9.75 IsZeroDivisor

** syntax **

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IsZeroDivisor(X: RINGELEM): BOOL</td>
<td>This function tests whether its argument is a zero divisor.</td>
</tr>
</tbody>
</table>

** example **

/**/ Use QQ[x,y,z];  
/**/ IsZeroDivisor(ideal(x));  
false
/**/ IsZeroDivisor(ideal(x^3, y^4-x ,z-3));  
true
/**/ IsZeroDivisor(ideal(x^2, x*y^3));  
false
Description

This function tests whether its argument is a zero-divisor.

```coconut
/**/ Use P ::= QQ[x,y,z];
/**/ R := NewQuotientRing(P, ideal(x*y));
/**/ IsZeroDivisor(RingElem(R,x));
true
/**/ colon(ideal(zero(R)), ideal(RingElem(R,x)));
ideal((y))
```

See Also: colon(I-3.27 pg.52)

I-9.76 IsZZ

```coconut
IsZZ(R: RING): BOOL
```

Description

This function tests whether a ring is the ring of integers.

```coconut
/**/ R ::= QQ[x,y];
/**/ IsZZ(CoeffRing(R));
false
/**/ IsZZ(BaseRing(CoeffRing(R)));
true
```

See Also: ZZ(I-25.4 pg.292), RingZZ(I-18.46 pg.242), IsQQ(I-9.59 pg.141)

I-9.77 It

```coconut
It
```

Description

“It” is a top-level SYSTEM VARIABLE containing the last result computed but not assigned. It is the CoCoA equivalent to GAP’s “last”.

When CoCoA evaluates a “standalone expression”, the result is assigned to the system variable named “It” (and then printed as if in a “println” (I-16.29 pg.213) command). You may use “It” in expressions just like any other variable.

```coconut
/**/ 1+1; -- standalone expression ==> result is saved in "It".
2
/**/ It;
2
/**/ It+1;
3
/**/ It;
3
```
I-9.77. It

/**/ X := 17; -- assignment is not a standalone expression, "It" is unchanged
/**/ It;
3
/**/ X+It;
20

See Also: print(I-16.25 pg.211), println(I-16.29 pg.213), Evaluation and Assignment(II-4 pg.305)
Chapter I-10

J

I-10.1 jacobian

**syntax**

\[ \text{jacobian}(L: \text{LIST of RINGELEM}): \text{MAT} \]

**Description**

This function returns the Jacobian matrix of the polynomials in “L” with respect to all the indeterminates of the current ring.

**example**

```plaintext
/**/ \text{Use R ::= QQ}\{x,y}\;
/**/ L := \{x-y, x^2-y, x^3-y^2\};
/**/ \text{jacobian}(L);
matrix( /*RingDistrMPolyClean(QQ, 2)*/
    \[[1, -1],
     [2*x, -1],
     [3*x^2, -2*y]\])
```

I-10.2 JanetBasis

**syntax**

\[ \text{JanetBasis}(I: \text{IDEAL}): \text{LIST of RINGELEM} \]

**Description**

Thanks to Mario Albert.

This function returns the Janet basis of an ideal.

**example**

```plaintext
/**/ \text{Use R ::= QQ}\{x,y,z\};
/**/ L := \{x-y, x^2-z+1, x^3-y^2\};
/**/ \text{JanetBasis}(\text{ideal}(L));
[x -y, z^2 -3*z +2, y*z -y -z +1, y^2 -z +1]
```

**See Also:** GBasis(I-7.1 pg.95)
Chapter I-11

K

I-11.1 ker

Syntax

\texttt{ker(\textphi: \text{RINGHOM}): \text{IDEAL}}

Description

This function returns the kernel of a homomorphism.

Example

```plaintext
/**/ \texttt{R ::= \texttt{QQ[x,y,z,w];}}
/**/ \texttt{Use S ::= \texttt{QQ[s,t];}}
/**/ \texttt{phi := PolyAlgebraHom(R, S, \texttt{[s^3, s^2*t, s*t^2, t^3]};}}
/**/ \texttt{ker(phi);}
\texttt{ideal(z^2 -y*t, y*z -x*t, y^2 -x*z)}

/**/ \texttt{SmodJ := NewQuotientRing(S, \texttt{ideal(ReadExpr(S,"t+s"));}}
/**/ \texttt{Use SmodJ;}
/**/ \texttt{psi := PolyAlgebraHom(R, SmodJ, \texttt{[s^3, s^2*t, s*t^2, t^3]};}}
/**/ \texttt{ker(psi);}
\texttt{ideal(x +w, y -w, z +w)}

/**/ \texttt{RmodI := NewQuotientRing(R, \texttt{ideal(ReadExpr(R,"x+y"));}}
/**/ \texttt{ker(InducedHom(RmodI, psi));}
\texttt{ideal((-y +w), (y -w), (z +w))}
```

See Also: PreImage(I-16.17 pg.207), IsInjective(I-9.49 pg.137), IsSurjective(I-9.66 pg.144)
Chapter I-12

L

I-12.1 last

<table>
<thead>
<tr>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>last(L: LIST): OBJECT</td>
</tr>
<tr>
<td>last(L: LIST, N: INT):</td>
</tr>
<tr>
<td>OBJECT</td>
</tr>
</tbody>
</table>

Description

In the first form, the function returns the last element of L. In the second form, it returns the list of the last N elements of L.

The CoCoA equivalent to GAP “last” is the variable “It” (I-9.77 pg.148).

Example

```plaintext
/**/ L := [1,2,3,4,5];
/**/ last(L);
5
/**/ last(L,3);
[3, 4, 5]
```

See Also: first(I-6.6 pg.85), tail(I-20.3 pg.272), It(I-9.77 pg.148)

I-12.2 LaTeX

<table>
<thead>
<tr>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>LaTeX(X: OBJECT): STRING</td>
</tr>
</tbody>
</table>

Description

This function returns a string containing the argument formatted in LaTeX. From version 4.7.5 it returns a string, so it can be printed on a file. Can also be called as “latex”.

Example

```plaintext
/**/ Use R := QQ[x,y,z];
/**/ F := x^3+2*y^2*z;
/**/ LaTeX(F);
x^3 + 2y^2z
 /**/ M := mat([[1,2],[3,4]]);
```
/**/ LaTeX(M);
\left( \begin{array}{cc}
1 & 2 \\
3 & 4 \end{array}\right)
/**/ R ::= QQ[x,y,z];
/**/ LaTeX(ideal(x^2,y+z));
(x^{2}, y +z)
/**/ P := NewFractionField(R);
/**/ Use P;
/**/ F := (x+y)/(1-z)^3;
/**/ LaTeX(F);
\frac{ - x - y} {z^{3}-3z^{2} + 3z-1}

See Also: format(I-6.15 pg.90), sprint(I-19.24 pg.257)

I-12.3 LC

| syntax |
| LC(F: RINGELEM|MODULEELEM): RINGELEM |

Description

This function returns the leading coefficient of F, as determined by the term-ordering of the ring to which F belongs.

| example |
| /**/ Use R ::= QQ[x,y];
/**/ LC(x +3*x^2 -5*y^2);
3
/**/ F := NewFreeModule(R,3);
/**/ LC(ModuleElem(F, [0, 5*y+6*x^2, y^2]));
6 |

See Also: coefficients(I-3.21 pg.48), CoeffOfTerm(I-3.24 pg.50), LT(I-12.21 pg.164)

I-12.4 lcm

| syntax |
| lcm(N: INT, M: INT): INT 
lcm(L: LIST of INT): INT 
lcm(F: RINGELEM, G: RINGELEM): RINGELEM 
lcm(L: LIST of RINGELEM): RINGELEM |

Description

This function returns the least common multiple of “F_1,…,F_n” or of the elements in the list L. For the calculation of the GCDs and LCMs of polynomials, the coefficient ring must be a field.
/**/ Use R ::= QQ[x,y];
/**/ F := x^2-y^2;
/**/ G := (x+y)^3;
/**/ lcm(F, G);
\[ -x^4 -2x^3y +2x*y^3 +y^4 \]
/**/ IsDivisible(F*G, It);
true
/**/ lcm(F, G) * gcd(F,G) = F*G;
true
/**/ lcm([3*4,3*8,6*16]);
96

See Also: div(I-4.20 pg.73), mod(I-13.21 pg.175), gcd(I-7.4 pg.96)

I-12.5 len

len(E: STRING|LIST): INT

Description

This function returns the “length” of an object, as summarized in the table below:

<table>
<thead>
<tr>
<th>type</th>
<th>length</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRING</td>
<td>number of bytes in the string</td>
</tr>
<tr>
<td>LIST</td>
<td>number of items in the list</td>
</tr>
</tbody>
</table>

The function ‘’\verb&len&’’

/**/ len( [2,3,4] );
3
/**/ len( "string" );
6

Previously “len” could be applied to other types too; this is no longer supported. See “NumCompts” (I-14.31 pg.192) for module elements, “NumRows” (I-14.34 pg.193) for matrices, and “NumTerms” (I-14.35 pg.193) for polynomials.

See Also: count(I-3.46 pg.60), NumCompts(I-14.31 pg.192), NumRows(I-14.34 pg.193), NumTerms(I-14.35 pg.193)

I-12.6 LexMat

LexMat(N: INT): MAT
Description

This function returns the matrix defining a standard term-ordering. These functions return matrices defining standard term-orderings.

```plaintext
LexMat(3);
matrix([1, 0, 0], [0, 1, 0], [0, 0, 1])
```

See Also: Orderings(III-9.5 pg.354), StdDegLexMat(I-19.30 pg.260), StdDegRevLexMat(I-19.31 pg.260), RevLexMat(I-18.37 pg.237), XelMat(I-24.1 pg.289)

I-12.7 LexSegmentIdeal

```
syntax
LexSegmentIdeal(L: LIST of power-products): IDEAL
LexSegmentIdeal(I: IDEAL): IDEAL
```

Description

If the argument is a list of power-products L, this function returns the smallest lex-segment ideal containing the power-products in L.

If it is an ideal I, it returns the lex-segment ideal having the same Hilbert function as I.

```plaintext
/**/ Use R ::= QQ[x,y,z];
/**/ LexSegmentIdeal([y^3]);
ideal(y^3, x*z^2, x*y*z, x*y^2, x^2*z, x^2*y, x^3)
/**/ LexSegmentIdeal(ideal(y^3));
ideal(x^3)
```

See Also: IsLexSegment(I-9.52 pg.138), StableIdeal(I-19.27 pg.259), StronglyStableIdeal(I-19.32 pg.261)

I-12.8 LF

```
syntax
LF(I: IDEAL): IDEAL
LF(F: RINGELEM): RINGELEM
```

Description

For a polynomial “F” this function returns the leading form, i.e. the sum of all summands having highest degree. It throws an error if the argument is zero or if the “GradingDim” (I-7.18 pg.103) of the polynomial ring is 0 (use “DF” (I-4.14 pg.70) to allow these cases).

For an ideal “I” this function returns the ideal of all the “LF(f)” for “f in I”. It throws an error if the “GradingDim” (I-7.18 pg.103) of the polynomial ring is 0.

```plaintext
/**/ Use R ::= QQ[x,y];
/**/ LF(x^2 -x*y +2*x -1);
```
See Also: DF(I-4.14 pg.70), IsHomog(I-9.46 pg.136), LC(I-12.3 pg.156), LM(I-12.15 pg.162), LPP(I-12.20 pg.164), LT(I-12.21 pg.164)

I-12.9 LinearSimplify

**syntax**

LinearSimplify(F: RINGELEM): RECORD

**Description**

This function returns a “RECORD[LinearChange, SimplePoly]” where “LinearChange” is a linear change of variable and “SimplePoly” is simple (in a heuristic sense). The composition “SimplePoly(LinearChange)” is equal the univariate polynomial “F”.

**example**

```coconut
/**/ Use QQ[x];
/**/ LinearSimplify((123*x-456)^9-1);
record[LinearChange := 123*x - 456, SimplePoly := x^9 - 1]
/**/ LinearSimplify(x^9-1); -- the heuristic finds no useful simplification
record[LinearChange := x, SimplePoly := x^9 - 1]
```

I-12.10 LinKer

**syntax**

LinKer(M: MAT): MAT

**Description**

This function returns a matrix whose columns representing a basis for the kernel of “M”. Calling the function twice on the same input will not necessarily produce the same output, though in each case, a basis for the kernel is produced.

This function works only on matrices whose entries are in a field (from version CoCoA-5.0.3). The CoCoA-4 function returning a ZZ-basis for the kernel of “M” is not yet implemented.

The output as it was given by CoCoA-4 (the basis of the ker) is now given by “LinKerBasis” (I-12.11 pg.160). See also “HilbertBasisKer” (I-8.4 pg.106).

**example**

```coconut
/**/ M := mat([[1,2,3,4],[5,6,7,8],[9,10,11,12]]);
/**/ LinKer(M);
matrix(QQ,


```
/**/ M*It;
matrix(QQ,
[[0, 0],
 [0, 0],
 [0, 0]])

See Also: LinSolve(I-12.14 pg.161), HilbertBasisKer(I-8.4 pg.106)

I-12.11 LinKerBasis

syntax
LinKerBasis(M: MAT): LIST

Description

This function returns a list whose components are lists representing a basis for the kernel of M. Calling the function twice on the same input will not necessarily produce the same output, though in each case, a basis for the kernel is produced.

This function works only on matrices whose entries are in a field (from version CoCoA-5.0.3). The CoCoA-4 function returning a ZZ-basis for the kernel of “M” is not yet implemented.

example
/**/ M := mat([[1,2,3,4],[5,6,7,8],[9,10,11,12]]);
/**/ LinKerBasis(M);
[[-1, 2, -1, 0], [-2, 3, 0, -1]]

/**/ K := NewFractionField(NewPolyRing(QQ, "a,b"));
/**/ Use K;
/**/ M := mat([[1,2,3,a],[5,6,7,a*b]]);
/**/ LinKerBasis(M);
[[-1, 2, -1, 0], [(a*b -3*a)/2, (-a*b +5*a)/4, 0, -1]]

See Also: LinSolve(I-12.14 pg.161)

I-12.12 LinKerModP [OBSOLETE]

syntax
[OBSOLETE]

Description

[OBSOLETE] In CoCoA-4 it was difficult to map a matrix into “ZZ/(p)”. Now, in CoCoA-5, we can map the matrix and then call directly “LinKer” (I-12.10 pg.159) and “LinKerBasis” (I-12.11 pg.160).

example
/**/ Use ZZ/(7);
/**/ M := mat([[1,2,3,4],[5,6,7,8],[9,10,11,12]]); --> by default over QQ
/**/ LinKerBasis(M);
[[-1, 2, -1, 0], [-2, 3, 0, -1]]
/**/ LinKerBasis(matrix(NewRingFp(3), M)); --> map M into ZZ/(3)
[-1, -1, -1, 0], [1, 0, 0, -1]]

/**/ LinKer(matrix(CurrentRing, M)); --> map M into CurrentRing ZZ/(7)
matrix( /*RingWithID(9, "FFp(7)")*/
    [[-1, -2],
     [2, 3],
     [-1, 0],
     [0, -1]])
/**/ matrix(CurrentRing, M) * It;
matrix( /*RingWithID(9,"FFp(7)")*/
    [[0, 0],
     [0, 0],
     [0, 0]])

See Also: LinKer(I-12.10 pg.159), LinKerBasis(I-12.11 pg.160), LinSolve(I-12.14 pg.161)

I-12.13 LinSol [OBSOLETE]

syntax
[OBSOLETE] use LinSolve

Description

I-12.14 LinSolve

syntax
LinSolve(M: MAT, RHS: MAT): MAT

Description
This function finds a solution “X” to the matrix equation “M*X = RHS”. If more than one solution exists, it returns just one of them. If no solution exists then it produces a 0-by-0 matrix. To find all solutions, compute the kernel of “M” using the function “LinKer” (I-12.10 pg.159).

NOTE: an easy way of converting a list into a column matrix (for the second argument) is to use the function “ColMat” (I-3.26 pg.51).

element
/**/ M := mat([[3,1,4],[1,5,9],[2,6,5]]);
/**/ L := [123,456,789];
/**/ LinSolve(M, ColMat(L));
mat([[199/5],
     [742/5],
     [-181/5]])
/**/ M*It;
mat([[123],
     [123],
     [123]])
See Also: LinKer(I-12.10 pg.159)

I-12.15 LM

**syntax**

\[
\begin{align*}
LM(X: \text{RINGELEM}) : \text{RINGELEM} \\
LM(X: \text{MODULEELEM}) : \text{MODULEELEM}
\end{align*}
\]

**Description**

This function returns the leading monomial of “X”. The monomial includes the coefficient. To get the leading term of “P”, (which does not included the coefficient), use “LT” (I-12.21 pg.164).

```plaintext
/**/ Use R ::= QQ[x,y];
/**/ LM(3*x^2*y + y);
3*x^2*y
```

See Also: LC(I-12.3 pg.156), LF(I-12.8 pg.158), LPP(I-12.20 pg.164), LT(I-12.21 pg.164)

I-12.16 log

**syntax**

\[
\text{log}(F: \text{RINGELEM}) : \text{LIST}
\]

**Description**

This function returns the list of exponents of the leading term of “F”. The inverse function is “LogToTerm” (I-12.18 pg.163).

```plaintext
/**/ Use R ::= QQ[x,y,z];
/**/ F := x^3*y^2*z^5 + x^2*y + x*z^4;
/**/ log(F);
[3, 2, 5]
```

See Also: ILogBase(I-9.9 pg.119), LT(I-12.21 pg.164), LogToTerm(I-12.18 pg.163)

I-12.17 LogCardinality

**syntax**

\[
\text{LogCardinality}(Fp: \text{RING}) : \text{INT}
\]

**Description**

This function returns the extension degree of a finite field over its prime field, or equivalently the log (base p) of its cardinality.
/**/ Fp := ZZ/(7);
/**/ Use Fpx := Fp[x];
/**/ Fq := Fpx/ideal(x^2+1);
/**/ LogCardinality(Fq);
2

See Also: IsFiniteField(I-9.45 pg.135), characteristic(I-3.9 pg.44)

I-12.18 LogToTerm

syntax

LogToTerm(R: RING, L: LIST of INT): RINGELEM

Description

This function returns the power-product whose list of exponents is “L”. It is the inverse of “log” (I-12.16 pg.162).

/**/ Use R := QQ[x,y,z];
/**/ LogToTerm(R, [2,3,5]);
x^2*y^3*z^5
/**/ log(It);
[2, 3, 5]

See Also: log(I-12.16 pg.162)

I-12.19 LPosn

syntax

LPosn(V: MODULEELEM): INT

Description

This function returns the position of the leading power-product of “V”.

This function used to be called “LPos” up to version 5.0.3.

/**/ Use R ::= QQ[x,y,z]; -- the default term-ordering is DegRevLex
/**/ R4 := NewFreeModule(R,4); -- the default module ordering is TOPos
/**/ LPosn(ModuleElem(R4, [0, x, y^2, x^2]));
4
/**/ LPP(ModuleElem(R4, [0, x, y^2, x^2]));
x^2
/**/ LT(ModuleElem(R4, [0, x, y^2, x^2]));
[0, 0, 0, x^2]

Use R ::= QQ[x,y], PosTo;
LT(Vector(x,y^2));
Vector(x, 0)
-----------------------------
LPP(Vector(x,y^2));
x
-------------------------------
LPosn(Vector(x,y^2));
1
-------------------------------

See Also: LF(I-12.8 pg.158), LM(I-12.15 pg.162), LPP(I-12.20 pg.164), LT(I-12.21 pg.164)

I-12.20 LPP

syntax
LPP(X: RINGELEM): RINGELEM
LPP(X: MODULEELEM): RINGELEM

Description
This function returns the leading power-product of “X”; it discards information about which component the power-product appears in.

example
/**/ Use R ::= QQ[x,y];
/**/ LPP(3*x^2*y+y); -- LPP is the same as LT for polynomials
x^2*y

-- Note the difference between LPP and LT for MODULEELEM.
/**/ R4 := NewFreeModule(R,4); -- the default module ordering is TOPos
/**/ LPP(ModuleElem(R4, [0, x, y^2, x^2]));
x^2
/**/ LT(ModuleElem(R4, [0, x, y^2, x^2]));
[0, 0, 0, x^2]
/**/ LPosn(ModuleElem(R4, [0, x, y^2, x^2]));
4

See Also: LC(I-12.3 pg.156), LF(I-12.8 pg.158), LM(I-12.15 pg.162), LPosn(I-12.19 pg.163), LT(I-12.21 pg.164)

I-12.21 LT

syntax
LT(I: RINGELEM): RINGELEM
LT(I: IDEAL): IDEAL
LT(I: MODULEELEM): MODULEELEM
LT(I: MODULE): MODULE

Description
If E is a polynomial this function returns the leading term of the polynomial E with respect to the term-ordering of the polynomial ring of E. For the leading monomial, which includes the coefficient, use “LM” (I-12.15 pg.162).

example
/**/ Use R ::= QQ[x,y,z]; -- the default term-ordering is DegRevLex
/**/ LT(y^2-x*z);
y^2
If “E” is a MODULEELEM, “LT(E)” gives the leading term of “E” with respect to the module term-ordering of “E”. For the leading monomial, which includes the coefficient, use “LM” (I-12.15 pg.162).

example

```plaintext
/**/ Use R ::= QQ[x,y,z], Lex;
/**/ LT(y^2-x*z);
x*z
```
Chapter I-13

M

I-13.1 MakeCheck

**syntax**

```
MakeCheck()
```

**Description**

***** NOT YET IMPLEMENTED *****

This function run a series of tests on the whole system. To get a reliable result you should run this on a “just opened” CoCoA because some printouts may mysteriously add some empty spaces which will result in an apparent, failure of some tests.

**example**

```
MakeCheck();
```

I-13.2 MakeMatByRows, MakeMatByCols

**syntax**

```
MakeMatByRows(R: INT, C: INT, L: LIST): MAT
MakeMatByCols(R: INT, C: INT, L: LIST): MAT
```

**Description**

These functions convert the list L into a matrix. The first argument is the number of rows and the second the number of columns.

**example**

```c
/**/ MakeMatByRows(2, 10, 1..20);
matrix(
[ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
[11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
)

/**/ MakeMatByCols(2, 10, 1..20);
matrix(
[ 1, 3, 5, 7, 9, 11, 13, 15, 17, 19],
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
)
```
I-13.3 MakeSet

Syntax:

\[
\text{MakeSet}(L: \text{LIST}): \text{LIST}
\]

Description:

This function returns a list obtained by removing duplicates from “L”.

Example:

```/* */ MakeSet([2,2,2,1,2,1,1,3,3]);
[2, 1, 3]
```

NOTE: to test two sets for equality use the function “EqSet” (I-5.6 pg.77) instead of a normal equality test (because the latter yields false if the elements are in a different order).

See Also: EqSet(I-5.6 pg.77), intersection(I-9.29 pg.129), IntersectList(I-9.30 pg.130), remove(I-18.29 pg.234)

I-13.4 MantissaAndExponent10

Syntax:

\[
\text{MantissaAndExponent10}(X: \text{INT}\mid\text{RAT}\mid\text{RINGELEM}, \text{Prec: INT}): \text{RECORD}
\]

Description:

This function converts a rational number into a “RECORD” with components named “exponent”, “mantissa” and “NumDigits”.

If “X=0”, all fields of the record are set to zero.

For non-zero “X” the fields give the best representation of the form \(M \times 10^E\) where “M” has “Prec” decimal digits. The value of “NumDigits” is simply “Prec”. The value of “exponent” is “ILogBase(X,10)”, plus 1 if the mantissa “overflows”. The value of “mantissa” is an integer “M” satisfying \(10^{(\text{Prec}−1)} \leq |M| < 10^{\text{Prec}−1}\)

Example:

```/* */ MantissaAndExponent10(1/2,3);  -- 1/2 = 5.00*10^(-1)
record[NumDigits := 3, exponent := -1, mantissa := 500]
```

```/* */ MantissaAndExponent10(0.99999, 4);  -- 0.99999 rounds up to give 1.000
record[NumDigits := 4, exponent := 0, mantissa := 1000]
```

See Also: DecimalStr(I-4.3 pg.63), FloatApprox(I-6.10 pg.87), FloatStr(I-6.11 pg.87), ILogBase(I-9.9 pg.119), MantissaAndExponent2(I-13.5 pg.168), ScientificStr(I-19.3 pg.246)

I-13.5 MantissaAndExponent2

Syntax:

\[
\text{MantissaAndExponent2}(X: \text{INT}\mid\text{RAT}\mid\text{RINGELEM}, \text{Prec: INT}): \text{RECORD}
\]
Description

This function converts a rational number into a "RECORD" with components named "exponent", "mantissa" and "NumDigits".

If "X=0", all fields of the record are set to zero.

For non-zero "X" the fields give the best representation of the form $M \times 2^E$ where "M" has "Prec" bits. The value of "NumDigits" is simply "Prec". The value of "exponent" is "ILogBase(X,2)", plus 1 if the mantissa "overflows". The value of "mantissa" is an integer "M" satisfying $2^{(Prec-1)} \leq |M| < 2^Prec - 1$

```plaintext
/**/ MantissaAndExponent2(1/2,8);  -- 1/2 = 128*2^(-8)
record[NumDigits := 8, exponent := -1, mantissa := 128]
/**/ MantissaAndExponent2(65535, 10);  -- rounds up
record[NumDigits := 10, exponent := 16, mantissa := 512]
```

See Also: FloatApprox(I-6.10 pg.87), ILogBase(I-9.9 pg.119), MantissaAndExponent10(I-13.4 pg.168)

I-13.6 Manual syntax

```plaintext
? key
?? key
```

Description

These operators are used to search the online help system for information matching a keyword (introduced in CoCoA 4.2).

The commands have the form "?key" and "??key" where "key" is a literal string without quotes. They are case insensitive and ignore blank space before or after "key". Also, the semicolon usually required at the end of a line of CoCoA input is optional.

The search system is fairly simple. The searching algorithm looks through the title and keywords of each manual page. A page matches if "key" appears as a (case-insensitive) substring of the title/keywords.

The "??" form prints the list of all matches. The "?" form prints the page matching exactly if there is one, otherwise the list of all matches.

```plaintext
/**/ ?approxs
-------------------[ ApproxSolve ]-------------------
---> ApproxSolve(L: LIST of RINGELEM): LIST of LIST of RAT

This function returns the list of real solutions (points) of a

--( Further output )--

------ No entry for "approxs" ------
All 8 matches for "approxs":
  ? ApproxSolve
  ? CFApprox
  --( Further output )--
```
I-13.7 MapDown [OBSOLETE]

[OBSOLETE] syntax

Description

[OBSOLETE] See “AsINT” (I-1.15 pg.31), “AsRAT” (I-1.16 pg.32).

I-13.8 matrix

<table>
<thead>
<tr>
<th>syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>matrix(L: LIST): MAT</td>
</tr>
<tr>
<td>matrix(R: RING, L: LIST): MAT</td>
</tr>
<tr>
<td>matrix(R: RING, M: MAT): MAT</td>
</tr>
</tbody>
</table>

Description

This function returns a matrix in the ring “R”.

When the input is “L”, a “rectangular” LIST of LIST of RINGELEM all in “R” (or INT, or RAT). When the ring is not specified it "guesses" the right ring; if all elements are INT or RAT the resulting matrix is in QQ.

The third form is equivalent to “apply(CanonicalHom(RingOf(M),R), M)”.

example

```*/
/**/ Use R ::= QQ[x,y];
/**/ L := [[1,2],[3,4]];
/**/ mat(L);
matrix(QQ,
[[1, 2],
 [3, 4]])
/**/ mat(R,L);
matrix( /*RingDistrMPolyClean(QQ, 2)*/
[[1, 2],
 [3, 4]])
/**/ mat(ZZ,L);
matrix(ZZ,
[[1, 2],
 [3, 4]])

/**/ RingOf(mat(R, [[1,2],[3,4]]));
RingDistrMPolyClean(QQ, 2)

/**/ M := IdentityMat(ZZ,2); matrix(QQ, M);
matrix(QQ,
[[1, 0],
 [0, 1]])
```
I-13.9  

max

**syntax**

\[
\text{max}(E_1: \text{OBJECT}, \ldots, E_n: \text{OBJECT}): \text{OBJECT} \\
\text{max}(L: \text{LIST}): \text{OBJECT}
\]

**Description**

In the first form, this function returns a maximum of \(E_1, \ldots, E_n\). In the second form, it returns a maximum of the objects in the list “L”.

**example**

```plaintext
/**/ \text{max}([1,2,3]); \\
3 \\
/**/ \text{max}(1,2,3); \\
3 \\
/**/ \text{use} R := \text{QQ}[x,y,z]; \\
/**/ \text{max}(x^3*z, x^2*y^2); -- x^2y^2 > x^3z \text{ in the default ordering, DegRevLex} \\
x^2*y^2 \\
/**/ \text{min}(x^3*z, x^2*y^2); \\
x^3*z \\
/**/ \text{use} R := \text{QQ}[x,y,z], \text{DegLex}; \\
/**/ \text{max}(x^3*z, x^2*y^2); -- x^3z < x^2y^2 \text{ in DegLex} \\
x^3*z
```

See Also:  min(I-13.11 pg.171), Relational Operators(II-3.3 pg.302)

I-13.10  

MayerVietorisTreeN1

**syntax**

\[
\text{MayerVietorisTreeN1}(I: \text{IDEAL}): \text{INT}
\]

**Description**

Implemented in CoCoALib by Eduardo Saenz-de-Cabezón.

This function returns the list of multidegrees “M” such that the N-1st Betti number of a monomial ideal “I” at multidegree “M” is not zero. It is computed via a version of its Mayer-Vietoris tree.

The length of this list is the number of irreducible components of I, the number of maximal standard monomials, and the number of generators of its Alexander Dual.

**example**

```plaintext
/**/ \text{Use} \text{QQ}[x,y,z]; \\
/**/ I := \text{ideal}(x, y, z)^2; \\
/**/ \text{MayerVietorisTreeN1}(I); \\
[x^2*y*z, x*y^2*z, x*y*z^2]
```

See Also:  Frobby(II-8.4 pg.319)

I-13.11  

min

**syntax**

\[
\text{min}(E_1: \text{OBJECT}, \ldots, E_n: \text{OBJECT}): \text{OBJECT} \\
\text{min}(L: \text{LIST}): \text{OBJECT}
\]
Description

In the first form, this function returns a minimum of $E_1, ..., E_n$. In the second form, it returns a minimum of the objects in the list “L”.

See more examples in “max” (I-13.9 pg.171).

```plaintext
/**/ min([1,2,3]);
1
/**/ min(1,2,3);
1
```

See Also: max(I-13.9 pg.171), Relational Operators(II-3.3 pg.302)

**I-13.12 MinGens**

### Syntax

`MinGens(M: IDEAL|MODULE): LIST`

### Description

If “M” is a homogeneous ideal or module, this function returns a list of minimal generators for “M” (not necessarily a subset of `gens(M)`).

For non-homogeneous input use “MinSubsetOfGens” (I-13.20 pg.175).

NOTE: the coefficient ring must be a field.

```plaintext
/**/ Use R ::= QQ[x,y,z];
/**/ I := ideal(x-y, (x-y)^4, z+y, (z+y)^2);
/**/ MinGens(I);
[y + z, x + z]
/**/ R3 := NewFreeModule(R, 3);
/**/ MGens := matrix(R, [[x,y,z], [x^2,0,z^2], [2*x^2,x*y,z^2+x*z]]);
/**/ M := SubmoduleRows(R3, MGens);
/**/ gens(M);
[[x, y, z], [x^2, 0, z^2], [2*x^2, x*y, x*z +z^2]]
/**/ MinGens(M);
[[x, y, z], [0, x*y, x*z -z^2]]
```


**I-13.13 MinGensGeneral [OBSOLESCENT]**

### Syntax

[OBSOLESCENT]

### Description

Just renamed “MinSubsetOfGens” (I-13.20 pg.175) (more expressive).

See Also: MinSubsetOfGens(I-13.20 pg.175)
I-13.14 minimalize

**syntax**

minimalize(ref X: IDEAL)

minimalize(ref X: MODULE)

**Description**

Similar to “minimalized” (I-13.15 pg.173), but modifies the argument “X” and returns NULL.

**example**

```/**/ Use R ::= QQ[x,y,z];
/**/ I := ideal(x^2-y^2, z^4-y^4, x^2-z^2);
/**/ I;
ideal(x^2 -y^2, -y^4 +z^4, x^2 -z^2)
/**/ minimalize(ref I); -- returns NULL and modifies I
/**/ I;
ideal(x^2 -z^2, y^2 -z^2)
```

**See Also:** MinGens(I-13.12 pg.172), MinSubsetOfGens(I-13.20 pg.175), minimalized(I-13.15 pg.173)

I-13.15 minimalized

**syntax**

minimalized(E: IDEAL): IDEAL

minimalized(E: MODULE): MODULE

**Description**

It works only in the homogeneous case: for the inhomogeneous case see “MinSubsetOfGens” (I-13.20 pg.175).

This function returns the ideal (or submodule) generated by a set of minimal generators of “E” (with minimal cardinality). The minimal set of generators is not necessarily a subset of the given generators.

The coefficient ring is assumed to be a field.

The similar function “minimalize” (I-13.14 pg.173) performs the same operation, but modifies the argument (“ref” (I-18.23 pg.231)) and returns NULL.

**example**

```/**/ Use R ::= QQ[x,y,z];
/**/ I := ideal(x^2-y^2, z^4-y^4, x^2-z^2);
/**/ I;
ideal(x^2 -y^2, -y^4 +z^4, x^2 -z^2)
/**/ minimalized(I);
ideal(x^2 -z^2, y^2 -z^2)
/**/ I; -- not modified
ideal(x^2 -y^2, -y^4 +z^4, x^2 -z^2)
```

**See Also:** MinGens(I-13.12 pg.172), MinSubsetOfGens(I-13.20 pg.175), minimalize(I-13.14 pg.173)

I-13.16 MinimalPresentation

**syntax**

MinimalPresentation(Q:TAGGED):TAGGED
where Q is a quotient module of the type R^s/M

**Description**

***** NOT YET IMPLEMENTED *****

Given a quotient module of the type $R^s/M$, or a zero module, this function computes an isomorphic quotient, $R^t/N$, minimally presented [using the algorithm in Kreuzer-Robbiano II].

```plaintext
Use R := QQ[x,y,z];
MinimalPresentation(R^3/Module([[x,1,1], [x,2,2]]));
R^2/Module([[x, 0]])

I-13.17 minors

**Syntax**

`minors(M: MAT, N: INT): LIST`

**Description**

This function returns the list of all determinants of N x N submatrices of M.

```plaintext
/**/ M := mat([[1,2,3],[-1,2,4]]);
/**/ minors(M, 2);
[4, 7, 2]

See Also: det(I-4.13 pg.70)

I-13.18 MinPoly

**Syntax**

`MinPoly(M: MAT, X: RINGELEM): RINGELEM`

**Description**

Thanks to Maria-Laura Torrente.

This function returns the minimal polynomial of the matrix “M” in the indeterminate “X” (with “M” a square matrix whose entries lie in the coefficient ring of “X”, or in the same ring of “X” but not dependent on “X”). See also “CharPoly” (I-3.10 pg.44).

```plaintext
/**/ Use R ::= QQ[x];
/**/ MinPoly(matrix([[0,0,1],[0,0,0],[0,0,0]]), x);
x^2
/**/ CharPoly(matrix([[0,0,1],[0,0,0],[0,0,0]]), x);
x^3

See Also: CharPoly(I-3.10 pg.44)
I-13.19 MinPowerInIdeal

**syntax**

MinPowerInIdeal(F: RINGELEM, I: IDEAL): INT

**Description**

This function returns the minimum power of F, the first argument, in the ideal I, the second argument. If F is not in the radical I then -1 is returned.

```plaintext
/**/ Use QQ[x,y,z];
/**/ I := ideal(x^6*y^4, z);
/**/ IsInRadical(x*y, I);
true
/**/ MinPowerInIdeal(x*y, I);
6
```

**See Also:** IsInRadical(I-9.50 pg.138), radical(I-18.1 pg.221)

I-13.20 MinSubsetOfGens

**syntax**

MinSubsetOfGens(M: IDEAL|MODULE): LIST

**Description**

This function returns a subset “S” of gens(M) which is minimal in the sense that no proper subset of “S” generates “M”.

NOTE: in general there might be other subsets with smaller cardinality.

If “M” is a homogeneous ideal or module, the function “MinGens” (I-13.12 pg.172) is much faster (but may return a generating set which is not a subset of gens(M)).

The coefficient ring must be a field.

```plaintext
/**/ Use R ::= QQ[x,y,z];
/**/ I := ideal(x-1, (x-y)^4, z+y, (z+y)^2);
/**/ MinSubsetOfGens(I);
[x -1, x^4 -4*x^3*y +6*x^2*y^2 -4*x*y^3 +y^4, y +z]
```

**See Also:** MinGens(I-13.12 pg.172), minimalize(I-13.14 pg.173), minimalized(I-13.15 pg.173)

I-13.21 mod

**syntax**

mod(N: INT, D: INT): INT

**Description**

We define the quotient “\( Q \)” and remainder “\( R \)” to be integers which satisfy \( N = Q \times D + R \) with \( 0 \leq R < |D| \). Then “\( \text{div}(N, D) \)” returns “\( Q \)” while “\( \text{mod}(N, D) \)” returns “\( R \)”.

NOTE: To perform the division algorithm on a polynomial, use “\( \text{NR} \)” (I-14.28 pg.190) (normal remainder) to find the remainder, or “\( \text{DivAlg} \)” (I-4.21 pg.73) to get both the quotients and the remainder.
176
Chapter I-13. M

/**/ div(10,3);
3
/**/ mod(10,3);
1

See Also: div(I-4.20 pg.73), DivAlg(I-4.21 pg.73), GenRepr(I-7.7 pg.98), NF(I-14.14 pg.184), NR(I-14.28 pg.190)

I-13.22 Mod2Rat [OBSOLETE]

/**/ div(10,3);
3
/**/ mod(10,3);
1

See Also: div(I-4.20 pg.73), DivAlg(I-4.21 pg.73), GenRepr(I-7.7 pg.98), NF(I-14.14 pg.184), NR(I-14.28 pg.190)

I-13.22 Mod2Rat [OBSOLETE]

syntax

[OBSOLETE]

Description


I-13.23 ModuleElem

syntax

ModuleElem(M: MODULE, L: LIST): MODULEELEM

Description

This function returns the MODULEELEM (called “Vector” in CoCoA-4) in the module “M” whose components are the components of the list L.

/**/ Use R ::= QQ[x];
/**/ R3 := NewFreeModule(R,3);
/**/ V := ModuleElem(R3, [1, x, x^2]); V;
[1, x, x^2]
/**/ type(V);
MODULEELEM
/**/ zero(R3);
[0, 0, 0]

See Also: SubmoduleCols, SubmoduleRows(I-19.37 pg.263)

I-13.24 ModuleOf

syntax

ModuleOf(M: MODULE): MODULE

Description

This function returns the module on which the object E is defined.

NOTE: A module contains many information and two separate rings, even when defined with the same commands, are not "equal". When a module is printed only a few informations are shown, so different modules might look the same.
I-13.25. monic

** syntax

monic(F: RINGELEM): RINGELEM

** Description

This function returns “F” divided by its leading coefficient (see “LC” (I-12.3 pg.156)) or, if “F” is zero, it returns just zero.

** example

```plaintext
/**/ Use R ::= QQ[x];
/**/ R3 := NewFreeModule(R,3);
/**/ V := ModuleElem(R3, [1, x, x^2]); V;
[1, x, x^2]
/**/ type(V);
MODULEELEM
/**/ ModuleOf(V) = R3;
true
/**/ ModuleOf(V);
FreeModule(RingDistrMPolyClean(QQ, 1), 3)
```

See Also: submodule(I-19.36 pg.262)

I-13.26 monomials

** syntax

monomials(F: RINGELEM|MODULEELEM): LIST

** Description

This function returns the list of monomials of F. The function “support” (I-19.41 pg.265) returns the list of terms (monomials without coefficients).
Chapter I-13. M

**/ Use R ::= QQ[x,y];
/**/ F := 3*x^2*y +5*y^3 -x*y^5;
/**/ monomials(F);
[-x*y^5, 3*x^2*y, 5*y^3]
/**/ support(F);
[x*y^5, x^2*y, y^3]

Monomials(Vector(3*x^2*y+y,5*x*y+4)); ---***WORK IN PROGRESS***
[Vector(3x^2y, 0), Vector(0, 5xy), Vector(y, 0), Vector(0, 4)]

See Also: coefficients(I-3.21 pg.48), support(I-19.41 pg.265)

I-13.27 MonsInIdeal

** syntax

MonsInIdeal(I: IDEAL): IDEAL

** Description

***** NOT YET IMPLEMENTED *****
This function returns the ideal generated by all monomials in the original ideal I.

** example

Use R ::= QQ[x,y,z];
I := ideal(xy^3+z^2, y^5-z^3, xz-y^2-x^3, x^4-xz^2+y^3);
MonsInIdeal(I);
ideal(z^3, yz^2, x^2z^2, x^5z, x^4yz, x^5y, x^2y^2z, x^7, x^4y^2,
xy^3z, y^4z, xy^4, x^3y^3, y^5)

I-13.28 multiplicity

** syntax

multiplicity(R: RING): INT

** Description

This function computes the multiplicity (or degree) of M, i.e., the leading coefficient of the Hilbert polynomial multiplied by the factorial of the degree of the Hilbert polynomial. M can be a module or a quotient.

** example

/**/ Use R ::= QQ[t,x,y,z];
/**/ multiplicity(R/ideal(x,y,z)^5);
35

See Also: HilbertFn(I-8.5 pg.107), HilbertSeries(I-8.7 pg.108), HVector(I-8.13 pg.112)
Chapter I-14

N

I-14.1 NewFractionField

**syntax**

NewFractionField(R: RING): RING

**Description**

NOTE: calling twice "NewFractionField" will produce two different rings, even with identical input: equality test is performed on the pointers. See "RingID" (I-18.40 pg.239).

**example**

```plaintext
/**/ K := NewFractionField(NewPolyRing(QQ, "a,b"));
/**/ Use K;
/**/ M := mat([[1,2,3,a],[5,6,7,a*b]]);
/**/ LinKerBasis(M);
[-1, 2, -1, 0], [(a*b -3*a)/2, (-a*b +5*a)/4, 0, -1]
```

**See Also:** NewQuotientRing(I-14.9 pg.182), RingID(I-18.40 pg.239), den(I-4.7 pg.67), num(I-14.29 pg.191)

I-14.2 NewFreeModule

**syntax**

NewFreeModule(R: RING, N: INT): MODULE
NewFreeModule(R: RING, Shifts: MAT): MODULE

**Description**

This function returns a free module which can be used as any programming variable.

NOTE: as for rings, calling twice "NewFreeModule" will produce two different modules, even with identical input: equality test is performed on the pointers.

This function does accept shifts from version CoCoA-5.0.4.

**example**

```plaintext
/**/ F := NewFreeModule(R, 3);
/**/ zero(F);
[0, 0, 0]
/**/ type(zero(F)); -- is NOT a LIST
MODULEELEM
/**/ gens(F);
[[1, 0, 0], [0, 1, 0], [0, 0, 1]]
```
/**/ F := NewFreeModule(R, matrix([[1],[2],[3]])); -- shifts
/**/ [wdeg(e) | e in gens(F)];
[[1], [2], [3]]

See Also: BaseRing(I-2.1 pg.33), RingOf(I-18.41 pg.239)

I-14.3 NewId [OBSOLETE]

syntax

[OBSOLETE]

Description

[OBSOLETE]

I-14.4 NewLine [OBSOLESCENT]

syntax

NewLine(): STRING

Description

This function is “OBSOLESCENT” and exists only for backward compatibility with old CoCoA code. It returns a string containing just a newline; in CoCoA-5 it is simpler to write “\n”.

example

/**/ str1 := "Line 1" + NewLine() + "Line 2"; --> old CoCoA-4 way
/**/ str2 := "Line 1\nLine 2"; --> more compact in CoCoA-5
/**/ str1 = str2;
True
/**/ Print str2;
Line 1
Line2

See Also: String Literals(III-4.1 pg.335), println(I-16.29 pg.213), ascii(I-1.14 pg.31)

I-14.5 NewList

syntax

NewList(N: INT): LIST
NewList(N: INT, E: OBJECT): LIST

Description

The first form returns a list of length “N” filled with 0 (“INT”). The second form returns a list of length “N”, filled with copies of “E”.

text

example

/**/ NewList(4,"a");
["a", "a", "a", "a"]
/**/ NewList(4);
[0, 0, 0, 0]

See Also: List Constructors (III-5.2 pg.340)

I-14.6 NewMat

syntax

NewMat(R: RING, M: INT, N: INT): MAT

Description

This function is kept for CoCoA-4 nostalgia: better use “ZeroMat” (I-25.2 pg.291).

example

/**/ Use S ::= QQ[x,y,z];
/**/ NewMat(S,2,3);  
/**/ NewMatFilled(1,3,x);
matrix( /*RingDistrMPolyClean(QQ, 3)*/
[[x, x, x]],
[[x, x, x]])
/**/ NewMatFilled(1,3, 0);
matrix(QQ,
[[0, 0, 0]],
[[0, 0, 0]])
/**/ ZeroMat(QQ, 1, 3); --> same as NewMatFilled(1,3, 0)
matrix(QQ,
[[0, 0, 0]])

See Also: matrix(I-13.8 pg.170), NewMatFilled(I-14.7 pg.181)

I-14.7 NewMatFilled

syntax

NewMatFilled(M: INT, N: INT, Val: INT|RAT|RINGELEM): MAT

Description

This function returns an “MxN” matrix, filled with “Val”. If “Val” is an integer or rational the ring of the matrix is defined in “RingQQ” (I-18.42 pg.240).

example

/**/ Use S ::= QQ[x,y,z];
/**/ NewMatFilled(1,3,x);
matrix( /*RingDistrMPolyClean(QQ, 3)*/
[[x, x, x]],
[[x, x, x]])
/**/ NewMatFilled(1,3, 0);
matrix(QQ,
[[0, 0, 0]],
[[0, 0, 0]])
/**/ ZeroMat(QQ, 1, 3); --> same as NewMatFilled(1,3, 0)
matrix(QQ,
[[0, 0, 0]])

See Also: NewMat(I-14.6 pg.181), matrix(I-13.8 pg.170)
### I-14.8 NewPolyRing

**Syntax**

NewPolyRing(CoeffRing: RING, IndetNames: STRING/LIST): RING  
NewPolyRing(CoeffRing: RING, IndetNames: STRING/LIST, OrdMat: MAT, GradingDim: INT): RING

**Description**

This function returns a polynomial ring which can be used as any programming variable (assigned with “:=”).  

The “:=” syntax starts the input method for a new polynomial ring, with the special interpretation of brackets and symbols (i.e. R ::= QQ[x] is not read as X := LL[i]). The pre-defined orderings for the “:=” syntax are “Lex” (no grading), “DegLex”, “DegRevLex” (standard grading). For more orderings use the “NewPolyRing” function call (see also “ElimMat” (I-5.4 pg.77)).  

NOTE: calling “NewPolyRing” twice with the same arguments gives two “different rings”, therefore incompatible. See “RingID” (I-18.40 pg.239).  

NOTE: the syntax with all indet names in one string is new in CoCoA-5.1.2.

**Example**

```plaintext  
/**/ R ::= QQ[x,y,alpha]; -- is equivalent to  
/**/ R := NewPolyRing(QQ, "x,y,alpha"); -- in "define/enddefine" use "RingQQ()"

/**/ R ::= QQ[x,y], DegRevLex; -- is equivalent to  
/**/ R := NewPolyRing(QQ, "x,y", StdDegRevLexMat(2), 1);

/**/ OrdM := matrix([[2,3,1],[0,0,-1],[0,-1,0]]);  
/**/ P := NewPolyRing(QQ, "x[1],x[2],x[9]", OrdM, 1); -- 3 indeterminates

/**/ wdeg(X) | X in indets(P)];  
[2, 3, 1]

/**/ P2 := NewPolyRing(RingZZ(), IndetSymbols(P)); -- same indet names as P  
/**/ Indets(P2);  
[x[1], x[2], x[9]]

/**/ P3 := NewPolyRing(P2, SymbolRange("alpha", -2,2));  
/**/ indets(P3);  
[alpha[-2], alpha[-1], alpha[0], alpha[1], alpha[2]]
```

**See Also:** ElimMat(I-5.4 pg.77), RingID(I-18.40 pg.239), IndetSymbols(I-9.23 pg.126), SymbolRange(I-19.45 pg.267), GradingDim(I-7.18 pg.103)

### I-14.9 NewQuotientRing

**Syntax**

NewQuotientRing(R: RING, I: IDEAL): RING  
R/I

**Description**

NOTE: calling twice “NewQuotientRing” will produce two different rings, even with identical input: equality test is performed on the pointers. See “RingID” (I-18.40 pg.239).

**Example**

```plaintext  
/**/ Use Qi ::= QQ[i];  
/**/ CC := Qi/ideal(i^2+1); -- sort of ;-)```
/**/ Use CC[x];
/**/ (x+i)^2;
(2*i*x +x^2 -1)

/**/ R ::= QQ[x,y,z];
/**/ S := NewQuotientRing(R, ideal(indet(R,1)-3));
/**/ Use S;
/**/ (x+y)^2;
(y^2 +6*y +9)

See Also: RingID(I-18.40 pg.239), QuotientBasis(I-17.3 pg.218), NewFractionField(I-14.1 pg.179)

I-14.10 NewRingFp

** syntax **
NewRingFp(P: INT): RING

** Description **
Create a new small prime finite field with characteristic “P”.
NOTE: in ring definitions you can use the convenient notation “ZZ/(p)”
NOTE: calling twice “NewRingFp” will produce two different rings, even with identical input: equality test is performed on the pointers. See “RingID” (I-18.40 pg.239).

** example **
/**/ p := NextPrime(1000);
/**/ Fp := NewRingFp(p);
/**/ Use Fp[x];
/**/ product([x-i | i in 1..p]);
x^1009 - x
/**/ Use ZZ/(p)[x]; --> convenient shorthand in ring defn
/**/ product([x-i | i in 1..p]);
x^1009 - x

See Also: RingID(I-18.40 pg.239), NewQuotientRing(I-14.9 pg.182)

I-14.11 NewRingTwinFloat

** syntax **
NewRingTwinFloat(Prec: INT): RING

** Description **
Create a new twin-float ring with bit precision “Prec”.
NOTE: calling twice “NewRingTwinFloat” will produce two different rings, even with identical input: equality test is performed on the pointers. See “RingID” (I-18.40 pg.239).


** example **
/**/ RR32 := NewRingTwinFloat(32);
/**/ Use RR32[x];
/**/ (3*x-1)/3;
/**/ RR64 := NewRingTwinFloat(64);
/**/ Use RR64[x];
/**/ (3*x-1)/3;
x -0.333333333333333333333333333333333333333

See Also: AsRAT(I-1.16 pg.32), RingID(I-18.40 pg.239)

I-14.12 NextPrime

Syntax

NextPrime(N: INT): INT

Description

This function computes the smallest prime number greater than N. If N is negative or too large then an error
is signalled. The upper limit depends on the computer you are using; it is probably \(2^{31}\) or \(2^{63}\).

Example

/**/ NextPrime(1000);
1009

See Also: IsPrime(I-9.56 pg.140), NextProbPrime(I-14.13 pg.184)

I-14.13 NextProbPrime

Syntax

NextProbPrime(N: INT): INT

Description

This function computes the smallest probable prime number greater than N. If N is negative, an error is
generated. To be absolutely certain the number produced is prime, you must call IsPrime on it, but this may
be very costly.

Example

/**/ NextProbPrime(1000);
1009
/**/ NextProbPrime(10^{50});
100000000000000000000000000000000000000000000000151

See Also: IsPrime(I-9.56 pg.140), IsProbPrime(I-9.57 pg.140), NextPrime(I-14.12 pg.184)

I-14.14 NF

Syntax

NF(F: RINGELEM, I: IDEAL): RINGELEM
NF(V: MODULEELEM, M: MODULE): MODULEELEM
**Description**

The first function returns the normal form of \( F \) with respect to \( I \). It also computes a Groebner basis of \( I \) if that basis has not been computed previously.

The second function returns the normal form of \( V \) with respect to \( M \). It also computes a Groebner basis of \( M \) if that basis has not been computed previously.

Currently (v 5.0.3) only full reduction is computed: each monomial in the result cannot be reduced. CoCoA-4 allowed setting the flag FullRed (of the panel GROEBNER) to False so that only the leading term is reduced.

Currently (v 5.0.3) polynomial ideals are implemented only with coeffs in a field.

**example**

```plaintext
/**/ Use R ::= QQ[x,y,z];
/**/ I := ideal(z);
/**/ NF(x^2+x*y+x*z+y^2+y*z+z^2, I);
  x^2 +x*y +y^2

/**/ I := ideal(z-1);
/**/ NF(x^2+x*y+x*z+y^2+y*z+z^2, I);
  x^2 +x*y +y^2 +x +y +1
```

**See Also:** DivAlg(I-4.21 pg.73), GenRepr(I-7.7 pg.98), IsIn(I-9.47 pg.136), NR(I-14.28 pg.190)

I-14.15  **NFsAreZero [OBsolete]**

**syntax**

```
(OBsolete)
```

**Description**

[OBsolete] “NFsAreZero(L, I)” is the same as “IsContained(ideal(L), I)”.

**See Also:** IsContained(I-9.37 pg.132), IsIn(I-9.47 pg.136), NF(I-14.14 pg.184)

I-14.16  **NmzComputation**

**syntax**

```
NmzComputation(Cone: RECORD): RECORD
NmzComputation(Cone: RECORD, ToCompute: LIST): RECORD
```

**Description**

“NmzComputation” provides direct access to libnormaliz. It faithfully reflects the internal structure of the libnormaliz design. Its first argument should be a record representing the cone. For the possible input options see the Normaliz documentation. With the second (optional) argument one can specify what should be computed. If it is omitted, everything that can be computed by libnormaliz will be computed.

**example**

```plaintext
/**/ Cone := record[ integral_closure := Mat([[1,2],[2,1]]),
               grading := Mat([[2,1]]),
               ];
/**/ NC2 := NmzComputation(Cone, ["HilbertBasis", "SupportHyperplanes", "HilbertSeries"]);
/**/ indent(NC2);
  record[
Congruences := [],
Deg1Elements := [],
EmbeddingDim := 2,
Equations := [],
ExtremeRays := [[1, 2], [2, 1]],
Generators := [[1, 2], [2, 1]],
Grading := [2, 1],
HilbertBasis := [[1, 1], [1, 2], [2, 1]],
HilbertSeries := record[DenFactors := record[RemainingFactor := 1, factors := [-t +1, -t^20 +1], multiplicities := [1, 1]], num := t^18 -t^17 +t^15 +t^10 -t^9 +t^8 +t^3 -t +1],
IsDeg1HilbertBasis := false,
IsInhomogeneous := false,
IsIntegrallyClosed := false,
IsPointed := true,
Multiplicity := 3/20,
Rank := 2,
SupportHyperplanes := [[-1, 2], [2, -1]]
]

See Also: NmzIntClosureToricRing(I-14.22 pg.188), NmzNormalToricRing(I-14.24 pg.189), NmzIntClosureMonIdeal(I-14.21 pg.188), NmzEhrhartRing(I-14.18 pg.186), NmzTorusInvariants(I-14.25 pg.189), NmzFiniteDiagInvariants(I-14.19 pg.187), NmzDiagInvariants(I-14.17 pg.186), NmzIntersectionValRings(I-14.23 pg.189)

I-14.17 NmzDiagInvariants

Syntax
NmzDiagInvariants(M: MAT, N: MAT, R: Ring): LIST of RINGELEM

Description
This function computes the ring of invariants of a diagonalizable group $D = T \times G$ where $T$ is a torus and $G$ is a finite abelian group, both acting diagonally on the polynomial ring $\mathbb{K}[X_1, \ldots, X_n]$.

The group actions are specified by the input matrices “M” and “N”. The first matrix specifies the torus action, the second the action of the finite group. See NmzTorusInvariants or NmzFiniteDiagInvariants for more detail. The output is the monomial subalgebra of invariants in “R”.

Example
/**/ Use R::=QQ[x,y,z,w];
/**/ T := matrix([[1,1,1,1],[1,1,1,1]]);
/**/ U := matrix([[1,1,1,1],[1,1,1,1]]);
/**/ NmzDiagInvariants(T,U,R);

See Also: NmzComputation(I-14.16 pg.185), NmzTorusInvariants(I-14.25 pg.189), NmzFiniteDiagInvariants(I-14.19 pg.187)

I-14.18 NmzEhrhartRing

Syntax
NmzEhrhartRing(L: LIST of RINGELEM, s: RINGELEM): LIST of RINGELEM

Description
The exponent vectors of the given monomials are considered as vertices of a lattice polytope “P”. The Ehrhart ring of a (lattice) polytope “P” is the monoid algebra defined by the monoid of lattice points in the cone over

See Also: NmzComputation(I-14.16 pg.185), NmzTorusInvariants(I-14.25 pg.189), NmzFiniteDiagInvariants(I-14.19 pg.187)
the polytope “P”; see the book by Bruns and Gubeladze, Polytopes, Rings, and K-theory, Springer 2009, pp. 228, 229. The function returns the generators of the Ehrhart ring. It uses the indeterminate in the second argument as auxiliary indeterminate of the Ehrhart ring.

/**/ Use R::=QQ[x,y,z,t];
/**/ NmzEhrhartRing([x^2,y^2,z^3],t);
[x^2*t, z^3*t, x*y*t, y^2*t]

See Also: NmzComputation(I-14.16 pg.185), NmzHilbertBasis(I-14.20 pg.187), NmzNormalToricRing(I-14.24 pg.189), NmzIntClosureMonIdeal(I-14.21 pg.188)

I-14.19 NmzFiniteDiagInvariants

** syntax **
NmzFiniteDiagInvariants(M: MAT, M: Ring): LIST of RINGELEM

** Description **
This function computes the ring of invariants of a finite abelian group $G$ acting diagonally on the surrounding polynomial ring $K[X_1,\ldots,X_n]$. The group is the direct product of cyclic groups generated by finitely many elements $g_1,\ldots,g_w$. The element $g_i$ acts on the indeterminate $X_j$ by $g_i(X_j) = l_i^{u_{ij}}X_j$ where $l_i$ is a primitive root of unity of order equal to $ord(g_i)$. The ring of invariants is generated by all monomials satisfying the system $u_{1i}a_1 + \ldots + u_{ni}$ and congruent to $0modord(g_i)i = 1,\ldots,w$. The input to the function is the $w \times (n+1)$ matrix “$U$” with rows $u_{1i},u_{2i},\ldots,ord(g_i),i = 1,\ldots,w$. The output is the monomial subalgebra of invariants $R^G = finR : g_i f = f for all i = 1,\ldots,w$.

/**/ Use R::=QQ[x,y,z,w];
/**/ U := matrix([[1,1,1,1,3],[1,0,2,0,4]]);
/**/ NmzFiniteDiagInvariants(U,R);
[x^2*z, z^2*w, y*z^2, x^12, y^3, z^6, w^3, x^8*w, x^4*w^2, y*w^2, x^8*y, x^4*y*w, y^2*w, x^4*y^2]

See Also: NmzComputation(I-14.16 pg.185), NmzTorusInvariants(I-14.25 pg.189), NmzDiagInvariants(I-14.17 pg.186)

I-14.20 NmzHilbertBasis

** syntax **
NmzHilbertBasis(M: MAT): MAT

** Description **
Given a matrix “$M$”, this function returns a matrix whose rows represent the Hilbert-Gordan Basis for the monoid generated by the rows of “$M$”.

/**/ M := matrix([[0,1],[3,1]]);
/**/ NmzHilbertBasis(M);
--the Hilbert basis of the monoid generated by the vectors [0,1] and [3,1] is...
matrix(QQ, 
[[3, 1],
 [0, 1]])
-- ... ([3,1], [0,1])
-- CAREFUL!! Different result from...
/**/ HilbertBasisKer(M);
-- the Hilbert basis of $M$ is the Hilbert basis of the monoid of
-- elements in the kernel of $M$, namely...
[]
-- ...no elements! (except the zero-element)

See Also: HilbertBasisKer(I-8.4 pg.106), NmzComputation(I-14.16 pg.185), NmzNormalToricRing(I-14.24 pg.189), NmzIntClosureMonIdeal(I-14.21 pg.188)

I-14.21 NmzIntClosureMonIdeal

**syntax**

NmzIntClosureMonRing(L: LIST of RINGELEM): LIST of RINGELEM
NmzIntClosureMonRing(L: LIST of RINGELEM, s: RINGELEM): LIST of RINGELEM

**Description**

Given a list "L" of power-products in a ring "R", the function returns the generators of the integral closure of the ideal generated by "L".

As second argument you can specify an indeterminate of the ring which is not used in the power-products. In this case the result is the normalisation of its Rees algebra (or Rees ring); see Bruns and Herzog, Cohen-Macaulay Rings, Cambridge University Press 1998, p. 182.

**example**

/**/ Use R::=QQ[x,y,z,t];
/**/ NmzIntClosureMonIdeal([x^2,y^2,z^3]);
-- the integral closure of the ideal generated by $x^2, y^2$ and $z^3$ is...
[y^2, x^2, x*y, z^3, y*z^2, x*z^2]
-- ...the ideal generated by $y^2, x^2, x*y, z^3, y*z^2$ and $x*z^2$
/**/ NmzIntClosureMonIdeal([x^2,y^2,z^3],t);
-- and the complete rees algebra is generated by
[z, z^3*t, y, y*z^2*t, y^2*t, x, x*z^2*t, x*y*t, x^2*t]

See Also: NmzComputation(I-14.16 pg.185), NmzHilbertBasis(I-14.20 pg.187), NmzNormalToricRing(I-14.24 pg.189), NmzEhrhartRing(I-14.18 pg.186)

I-14.22 NmzIntClosureToricRing

**syntax**

NmzIntClosureToricRing(L: LIST of RINGELEM): LIST of RINGELEM

**Description**

Given a list L of power-products in a ring R, the function returns the generators of the integral closure of the algebra generated by the list.

**example**

/**/ Use R::=QQ[x,y,t];
/**/ NmzIntClosureToricRing([x^3,x^2*y,y^3]);
-- the integral closure of $QQ[x^3, x^2*y, y^3]$ is...
I-14.23  \textbf{NmzIntersectionValRings}

\textbf{syntax}

\texttt{NmzIntersectionValRings(M: MAT, M: Ring): LIST of RINGELEM}

\textbf{Description}

A discrete monomial valuation \( v \) on \( R = K[X_1, \ldots, X_n] \) is determined by the values \( v(X_j) \) of the indeterminates. This function computes the subalgebra \( S = \bigcap f_{i} f > 0, i = 1, \ldots, r \) that is the intersection of the valuation rings of the given valuations \( v_1, \ldots, v_r \), i.e. it consists of all elements of \( R \) that have a nonnegative value for all \( r \) valuations. It takes as input the matrix \( V = (v_i(X_j)) \) whose rows correspond to the values of the indeterminates.

\begin{verbatim}
/**/ Use R::=QQ[x,y,z,w];
/**/ V := matrix([[0,1,2,3],[-1,1,2,1]]);
/**/ NmzIntersectionValRings(V,R);
[ y, z, w, x*y, x^2*z, x*w, x*z ]
\end{verbatim}

\textbf{See Also:} NmzComputation(I-14.16 pg.185), NmzHilbertBasis(I-14.20 pg.187), NmzNormalToricRing(I-14.24 pg.189), NmzIntClosureMonIdeal(I-14.21 pg.188), NmzEhrhartRing(I-14.18 pg.186)

I-14.24  \textbf{NmzNormalToricRing}

\textbf{syntax}

\texttt{NmzNormalToricRing(L: LIST of RINGELEM): LIST of RINGELEM}

\textbf{Description}

Given a list \( "L" \) of power-products in a ring \( R \), the function returns the generators of the normalization of the algebra generated by the list.

\begin{verbatim}
/**/ Use R::=QQ[x,y,t];
-- We compute the normalization of QQ[x^3, x^2*y, y^3]
/**/ NmzNormalToricRing([x^3, x^2*y, y^3]);
[y^3, x^2*y, x^3, x*y^2]
--> answer is QQ[y^3, x^2*y, x^3, x*y^2]
\end{verbatim}

\textbf{See Also:} NmzComputation(I-14.16 pg.185), NmzHilbertBasis(I-14.20 pg.187), NmzIntClosureToricRing(I-14.22 pg.188), NmzIntClosureMonIdeal(I-14.21 pg.188)

I-14.25  \textbf{NmzTorusInvariants}

\textbf{syntax}

\texttt{NmzTorusInvariants(M: MAT, R: Ring): LIST of RINGELEM}
Chapter I-14. N

Description

Let \( T = (K^*)^r \) be the \( r \)-dimensional torus acting on the polynomial ring \( \mathbb{R} = K[X_1, \ldots, X_n] \) diagonally. Such an action can be described as follows: there are integers \( a_{ij}, i = 1, \ldots, r, j = 1, \ldots, n \) such that \((l_1, \ldots, l_r)\) in \( T \) acts by the substitution \( X_j \) maps to \( l_1^{a_{1j}} \cdots l_r^{a_{rj}} \cdot X_j \) for \( j = 1, \ldots, n \).

The function takes the matrix \( M = (a_{ij}) \) and the ring \( \mathbb{R} \) as input. It computes the ring of invariants \( \mathbb{R}^T = \{ f \in \mathbb{R} | f \text{ is fixed by all } l \in T \} \).

```cocoa
/**/ Use R::=QQ[x,y,z,w];
/**/ T := matrix([[1,2,-1],[1,2,-1]]);
/**/ NmzTorusInvariants(T,R);
[x^2*z, x*y*z, y^2*z]
```

See Also: NmzComputation(I-14.16 pg.185), NmzDiagInvariants(I-14.17 pg.186), NmzFiniteDiagInvariants(I-14.19 pg.187)

I-14.26 NonZero

** syntax **

NonZero(L: LIST|MODULEELEM): LIST

** Description **

This function returns the list obtained by removing the zeroes from \( L \).

```cocoa
/**/ Use R ::= QQ[x,y,z];
/**/ NonZero([0,0,3, ideal(y),0]);
[3, ideal(y)]
```

See Also: FirstNonZero(I-6.7 pg.85), FirstNonZeroPosn(I-6.8 pg.86), IsZero(I-9.72 pg.146)

I-14.27 not

** syntax **

not(A: BOOL): BOOL

** Description **

This function negates a boolean: i.e. if \( A \) gives "true" then \( \text{not}(A) \) gives "false", and vice versa.

Note that from CoCoA-5.1 \( \text{not} \) is a function, so its argument must be between brackets!

```cocoa
/**/ [n in 1..10 | not(IsPrime(n))];
[1,4,6,8,9]
```

See Also: and(I-1.10 pg.29), or(I-15.9 pg.199)

I-14.28 NR

** syntax **

NR(X: RINGELEM, L: LIST of RINGELEM): RINGELEM
NR(X: MODULEELEM, L: LIST of MODULEELEM): MODULEELEM
Description

This function returns the normal remainder of \( X \) with respect to \( L \), i.e., it returns the remainder from the division algorithm. To get both the quotients and the remainder, use “DivAlg” (I-4.21 pg.73).

Note that if the list does not form a Groebner basis, the remainder may not be zero even if \( X \) is in the ideal or module generated by \( L \) (use “GenRepr” (I-7.7 pg.98) or “NF” (I-14.14 pg.184) instead).

Currently (v 5.0.3) the internal code for computing NF(F, I) and NR(F, GBasis(I)) is identical, but the second is slower just for the overhead in interpreting a possibly long list of polynomials.

\[
\begin{align*}
\text{example} & \quad \text{/**} \quad \text{Use } R := \text{QQ}[x,y,z]; \\
& \quad \text{/**} \quad F := x^2y + xy^2 + y^2; \\
& \quad \text{/**} \quad NR(F, \{x*y-1, y^2-1\}); \\
& \quad x + y + 1
\end{align*}
\]

// NOT YET IMPLEMENTED for MODULELEM

See Also: DivAlg(I-4.21 pg.73), GenRepr(I-7.7 pg.98), NF(I-14.14 pg.184)

I-14.29 num

\[
\begin{align*}
\text{syntax} & \quad \text{num}(N: \text{INT}): \text{INT} \\
& \quad \text{num}(N: \text{RAT}): \text{INT} \\
& \quad \text{num}(N: \text{RINGELEM}): \text{RINGELEM}
\end{align*}
\]

Description

This function returns the numerator of “\( N \)”.

The OBSOLETE fragile syntax in CoCoA 4 “\( N.\text{Num} \)” and “\( N.\text{Den} \)” is no longer supported.

\[
\begin{align*}
\text{example} & \quad \text{/**} \quad \text{num}(3); \\
& \quad 3 \\
& \quad \text{/**} \quad P := \text{QQ}[x,y]; \\
& \quad \text{/**} \quad F := \text{NewFractionField}(P); \\
& \quad \text{/**} \quad \text{Use } F; \\
& \quad \text{/**} \quad \text{num}(x/(x+y)); \\
& \quad x
\end{align*}
\]

See Also: den(I-4.7 pg.67)

I-14.30 NumCols

\[
\begin{align*}
\text{syntax} & \quad \text{NumCols}(M: \text{MAT}): \text{INT}
\end{align*}
\]

Description

This function returns the number of columns in a matrix.

\[
\begin{align*}
\text{example} & \quad \text{/**} \quad M := \text{mat}([[1,2,3],[4,5,6]]); \\
& \quad \text{/**} \quad \text{NumCols}(M); \\
& \quad 3
\end{align*}
\]
See Also: matrix(I-13.8 pg.170), NumRows(I-14.34 pg.193)

I-14.31 NumCompts

Syntax

NumCompts(X: MODULEELEM|MODULE): INT

Description

If “X” is a “MODULEELEM”, it returns the number of components of “X”. If “X” is a “MODULE”, it returns the rank of the free module in which “X” is defined.

This function used to be called “NumComps” in CoCoA-4.

Example

```plaintext
/**/ Use R ::= QQ[x,y];
/**/ R2 := NewFreeModule(R, 3);
/**/ M := SubmoduleRows(R2, matrix(R, mat([[x,0,y], [x^2+y^2,x^2,3]])));
/**/ NumCompts(M);
3
/**/ NumCompts(gens(M)[1]);
3
```

See Also: len(I-12.5 pg.157)

I-14.32 NumIndets

Syntax

NumIndets(R: RING): INT

Description

This function returns the number of indeterminates of the current ring or of R.

Example

```plaintext
/**/ S ::= QQ[x,y];
/**/ R ::= QQ[x,y,z];
/**/ NumIndets(R);
3
/**/ NumIndets(S);
2
```

See Also: indet(I-9.18 pg.123), IndetSubscripts(I-9.22 pg.126), IndetIndex(I-9.19 pg.124), IndetName(I-9.20 pg.124), indets(I-9.21 pg.125)

I-14.33 NumPartitions

Syntax

NumPartitions(N: INT): INT
Description
This function returns the number of partitions of a non-negative integer, i.e. the number of distinct ways of writing \( N \) as a sum of positive integers.

```plaintext
/**/ NumPartitions(2); -- 2 and 1+1
2
/**/ NumPartitions(5);
7
```

I-14.34  NumRows

**syntax**
```
NumRows(M: MAT): INT
```

**Description**
This function returns the number of rows in a matrix.

```plaintext
/**/ M := mat([[1,2,3], [4,5,6]]);
/**/ NumRows(M);
2
```

See Also: matrix(I-13.8 pg.170), NumCols(I-14.30 pg.191)

I-14.35  NumTerms

**syntax**
```
NumTerms(F: RINGELEM): INT
```

**Description**
This function returns the number of terms in a polynomial.

```plaintext
/**/ Use R ::= QQ[x,y,z];
/**/ NumTerms((x+y+z)^5) = binomial(3+5-1, 5);
true
```

See Also: len(I-12.5 pg.157)
Chapter I-15

O

I-15.1 one

**Syntax**

\texttt{one(R: RING): RINGELEM}

**Description**

This function return the multiplicative identity of a ring. For when you want to force the integer “1” to be a “RINGELEM”.

**Example**

```plaintext
/**/ P ::= ZZ/(101)[x,y,z];
/**/ N := 1; Print N, " of type ", type(N);
1 of type INT
/**/ N := one(P); Print N, " of type ", type(N);
1 of type RINGELEM
/**/ N := 300*1; Print N, " of type ", type(N);
300 of type INT
/**/ N := 300*one(P); Print N, " of type ", type(N);
-3 of type RINGELEM
```

**See Also:** zero(I-25.1 pg.291)

I-15.2 OpenIFile

**Syntax**

\texttt{OpenIFile(S: STRING): DEVICE}

**Description**

***** NOT YET IMPLEMENTED *****

This function opens the file with name \( S \) for input. Input from that file can then be read with “Get” (I-7.10 pg.100).

*(NOTE: one would normally use “source” (I-19.21 pg.256) to read CoCoA commands from a file.)*

**Example**

```plaintext
D := OpenOFile("my-test"); -- open "my-test" for output from CoCoA
Print "hello world" On D;   -- print string into "mytest"
Close(D);
```
D := OpenIFile("my-test"); -- open "my-test" for input to CoCoA
Get(D,3); -- get the first three characters (in Ascii code)
[104, 101, 108]
-----------------------------------------------
ascii(It); -- convert the ASCII code into characters
hel
-----------------------------------------------
Close(D);

See Also: close(I-3.14 pg.46), Introduction to IO(II-6.1 pg.309), OpenOFile(I-15.5 pg.197), OpenIString(I-15.3 pg.196), OpenOString(I-15.6 pg.198), OpenSocket(I-15.7 pg.198), source(I-19.21 pg.256)

I-15.3 OpenIString

**syntax**

OpenIString(S: STRING, T: STRING): DEVICE
OpenOString(S: STRING): DEVICE

**Description**

***** NOT YET IMPLEMENTED *****

This function open strings for input. The string S serves as the name of the device opened for input or output; one may use the empty string. “OpenIString” is used to read input from the string T with the help of “Get” (I-7.10 pg.100).

**example**

S := "hello world";
D := OpenIString("", S); -- open the string S for input to CoCoA
L := Get(D,7); -- read 7 characters from the string
L; -- ASCII code
-----------------------------------------------
ascii(L); -- convert ASCII code to characters
hello w
-----------------------------------------------
Close(D); -- close device D

See Also: close(I-3.14 pg.46), Introduction to IO(II-6.1 pg.309), OpenOString(I-15.6 pg.198), OpenFile(I-15.2 pg.195), OpenOFile(I-15.5 pg.197), source(I-19.21 pg.256), sprint(I-19.24 pg.257)

I-15.4 OpenLog

**syntax**

OpenLog(D: DEVICE)

**Description**

***** NOT YET IMPLEMENTED *****

This function opens the output device D and starts to record the output from a CoCoA session on D. The “CloseLog” (I-3.15 pg.46) closes the device D and stops recording the CoCoA session on D.
At present the choices for the device D are an output file (see “OpenOFile” (I-15.5 pg.197)) or an output string (see “OpenOString” (I-15.6 pg.198)). Several output devices may be open at a time. If the panel option “Echo” is set to True, both the input and output of the CoCoA session are logged; otherwise, just the output is logged.

```coconut
D := OpenOFile("MySession");
OpenLog(D);
1+1;
2

G := 1;
Set Echo;
2+2;
2 + 2
4

F := 2;
F := 2
CloseLog(D);
CloseLog(D)
UnSet Echo;
SET(Echo, False)
```

--- The contents of "MySession":
2

```coconut
2 + 2
4
```

```coconut
F := 2
CloseLog(D)
```

See Also: Introduction to IO(I-6.1 pg.309), OpenIFile(I-15.2 pg.195), OpenOFile(I-15.5 pg.197), OpenIString(I-15.3 pg.196), OpenOString(I-15.6 pg.198)

I-15.5 OpenOFile

**syntax**

```
OpenOFile(S: STRING): DEVICE
OpenOFile(S: STRING,"w" or "W"): DEVICE
```

**Description**

This function opens the file with name S—creating it if it does not already exist—for output. If used with second argument “w” or “W” then it immediately erases the file S. The function “print on” (I-16.26 pg.212) is then used for appending output to S.

```coconut
D := OpenOFile("my-test"); -- open "my-test" for output from CoCoA
Print "hello world" On D; -- print string into "mytest"
Print " test" On D; -- append to the file "mytest"
Close(D); -- close the file
D := OpenOFile("my-test","w"); -- clear "my-test"
Print "goodbye" On D; -- "mytest" now consists only of the string "goodbye"
Close(D);
```
I-15.6 OpenOString

**syntax**

```
OpenOString(S: STRING): DEVICE
```

**Description**

This function opens strings for output. The string S serves as the name of the device opened for input or output; one may use the empty string. “OpenOString” is used to write to a string with the help of “print on”.

```
D := OpenOString(""), -- open a string for output from CoCoA
L := [1,2,3]; -- a list
Print L On D; -- print to D
D;
record[Name := ", Type := "OString", Protocol := "CoCoALanguage"]
------------------------------------------------------------
S := Cast(D, STRING); -- S is the string output to D
S; -- a string
[1, 2, 3]
------------------------------------------------------------
Print " more characters" On D; -- append to the existing output string
Cast(D, STRING);
[1, 2, 3] more characters
------------------------------------------------------------
```

**See Also:** close(I-3.14 pg.46), Introduction to IO(II-6.1 pg.309), OpenIFile(I-15.2 pg.195), OpenIString(I-15.3 pg.196), OpenOFile(I-15.5 pg.197), OpenOString(I-15.6 pg.198), source(I-19.21 pg.256)

I-15.7 OpenSocket

**syntax**

```
OpenSocket(Machine: STRING, Port: STRING): DEVICE
```

**Description**

***** NOT YET IMPLEMENTED *****

This function opens a client socket (I/O) connection. It requires the name of the machine with the server socket and the port number (expressed as a STRING).

CoCoA-4 communicates with the CoCoAServer via socket which, by default, runs on “localhost” on port “0xc0c0”. To change these settings redefine in your “userinit.coc” or “.cocoarc” the variables

```
MEMORY.CoCoAServerMachine := "localhost";
MEMORY.CoCoAServerPort := "0xc0c0";
```

```
D := OpenSocket("localhost", "10000");
Print 100^6 On D;
Source D;
Close(D);
```
See Also: Introduction to IO(I-6.1 pg.309), OpenIFile(I-15.2 pg.195), OpenOFile(I-15.5 pg.197), OpenIString(I-15.3 pg.196), OpenOString(I-15.6 pg.198)

I-15.8 Option [OBSOLETE]

```
syntax
[OBSOLETE]
```

Description

[OBSOLETE]

I-15.9 or

```
syntax
A or B  (where A, B: BOOL, return BOOL)
```

Description

This operator represents the logical disjunction of “A” and “B”. CoCoA first evaluates “A”; if that gives “true” then the result is “true”, and “B” is not evaluated. Otherwise, if “A” gives “false” then “B” is evaluated, and its value is the final result.

```
/**/ Define IsUnsuitable(X)
/**/ Return X < 0 or isqrt(X) >= 2^16;
/**/ EndDefine;
/**/ IsUnsuitable(-9);
true
/**/ IsUnsuitable(9);
false
```

See Also: and(I-1.10 pg.29), not(I-14.27 pg.190)

I-15.10 OrdMat

```
syntax
OrdMat(R: RING): MAT
```

Description

This function returns a matrix which describes the term-ordering of the ring “R”.

```
/**/ Use S ::= QQ[x,y,z];
/**/ M := mat([[1,2,3], [3,4,5], [0,0,1]]);
/**/ P := NewPolyRing(CoeffRing(S), IndetSymbols(S), M, 2);
/**/ GradingDim(P);
2
/**/ OrdMat(P);
matrix(QQ,
    [[1, 2, 3],
     [2, 3, 4]])
```
/**/ GradingDim(S);
1
/**/ OrdMat(S);
matrix(QQ,
[[1, 1, 1],
[0, 0, -1],
[0, -1, 0]])

See Also: StdDegLexMat(I-19.30 pg.260), StdDegRevLexMat(I-19.31 pg.260), LexMat(I-12.6 pg.157),
RevLexMat(I-18.37 pg.237), XelMat(I-24.1 pg.289), elim(I-5.3 pg.76), GradingDim(I-7.18 pg.103), Order-
ings(III-9.5 pg.354), NewPolyRing(I-14.8 pg.182)
Chapter I-16

P

I-16.1 Packages

Syntax

packages(): LIST of STRING

Description

This function returns the names of the loaded packages as a list of strings. The old CoCoA-4 names “$user” and “$builtin” are no longer used.

Example

/**/ packages();
["$BackwardCompatible", "$BringIn", (...) ]

See Also: CoCoA Packages(II-7 pg.313), Supported Packages(II-7.7 pg.315)

I-16.2 panel [OBSOLETE]

Syntax

[OBSOLETE]

Description

[OBSOLETE]

I-16.3 panels [OBSOLETE]

Syntax

[OBSOLETE]

Description

[OBSOLETE]
I-16.4 partitions

Syntax

partitions(N: INT): LIST

Description

These function returns all integer partitions of N, positive integer

Example

```/**/ partitions(3);
[[3], [1, 2], [1, 1, 1]]
```

See Also: subsets(I-19.38 pg.263), tuples(I-20.14 pg.278)

I-16.5 permutations

Syntax

permutations(L: LIST): LIST

Description

This function computes all permutations of the entries of a list (set). If L has repeated elements it will return repeated elements.

Example

``` /**/ permutations(3..5);
[[3, 4, 5], [3, 5, 4], [4, 3, 5], [4, 5, 3], [5, 3, 4], [5, 4, 3]]

/**/ permutations([2, 2, x]);
[[2, 2, x], [2, x, 2], [2, 2, x], [2, x, 2], [x, 2, 2], [x, 2, 2]]

/**/ MakeSet(permutations([2, 2, x]));
[[2, 2, x], [2, x, 2], [x, 2, 2]]
```

See Also: subsets(I-19.38 pg.263), tuples(I-20.14 pg.278)

I-16.6 PerpIdealOfForm

Syntax

PerpIdealOfForm(F: RINGELEM): IDEAL

Description

Thanks to Enrico Carlini.

Given a form “F” computes the ideal of derivations killing it.

For the sake of simplicity Forms/Polynomials and Derivations live in the same ring, the distinction between them is purely formal.

Example

``` /**/ Use R := QQ[x,y,z];
/**/ PerpIdealOfForm(x^3+x*y*z);
ideal(z^2, y^2, x^2 -6*y*z)```
/**/ HilbertFn(R/It);
H(0) = 1
H(1) = 3
H(2) = 3
H(3) = 1
H(t) = 0 for t >= 4

See Also: InverseSystem(I-9.32 pg.131), DerivationAction(I-4.11 pg.69)

I-16.7 pfaffian

syntax

pfaffian(M: MAT): RINGELEM

Description

This function returns the Pfaffian of M.

example

/**/ Use R ::= QQ[x,y];
/**/ pfaffian(mat([[0,y],[-y,0]]));
y

See Also: det(I-4.13 pg.70)

I-16.8 PkgName

syntax

PkgName(): STRING
S.PkgName(): STRING

where S is the identifier or alias for a package.

Description

This function returns the (long) name of a package. The first form returns “$coclib” and the second returns the name of the package whose name or alias is S. This function is useful as a shorthand, when S is an alias, for the full name a package.

example

GB.PkgName();
$gb
------------------------
$gb.PkgName();
$gb
------------------------
PkgName();
$coclib
------------------------
I-16.9 PlotPoints

**syntax**

PlotPoints(L: LIST of points)

**Description**

This function outputs the coordinates of the points (with two components) to a file called "CoCoAPlot". See "PlotPointsOn" (I-16.10 pg.204) for outputting to another file.

This result can be plotted using your preferred plotting program. For example, start "gnuplot" and then give it the command

```plaintext
plot "CoCoAPlot"
```

to see the plot.

**example**

```plaintext
/**/ PlotPoints([[X, X^2-X+14] | X In -10..10]);
Plotting points...100%
21 plotted points have been placed in the file CoCoAPlot
```

See Also: ImplicitPlot(I-9.12 pg.120), PlotPointsOn(I-16.10 pg.204)

I-16.10 PlotPointsOn

**syntax**

PlotPointsOn(L: LIST of points, S: STRING)

**Description**

This function is the same as "PlotPoints" (I-16.9 pg.204) with a second argument giving the name of the file to print on.

Note that the last argument is a STRING, the name of the file, and not a DEVICE, as for "print on" (I-16.26 pg.212).

**example**

```plaintext
/**/ PlotPointsOn([[1/(X+1/2), X^2-X+14] | X In -10..10], "PLOT-points");
Plotting points...100%
21 plotted points have been placed in the file points
/**/ ImplicitPlotOn(x^2*y -(59/4)*x^2 +2*x -1, [-3,3], [0,250], "PLOT-curve");
Plotting points...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
735 plotted points have been placed in the file curve
```

After having produced the plot files using CoCoA-4, start "gnuplot" and then give it the following commands:

```plaintext
plot "curve"
replot "points"
```

See Also: ImplicitPlot(I-9.12 pg.120), PlotPoints(I-16.9 pg.204)
I-16.11  poincare [OBSOLESCENT]

```
[poincare(M: RING|IDEAL):TAGGED("$hp.PSeries")]
```

Description

(sorry Poincare' for the lower-case: here we follow the naming convention “single name goes lower-case”)

See Also: HilbertSeries(I-8.7 pg.108)

I-16.12  PoincareMultiDeg [OBsolete]

```
[poincareMultiDeg(M: RING|IDEAL):TAGGED("$hp.PSeries")]
```

Description

[OBSOLETE] now called “HilbertSeriesMultiDeg” (I-8.8 pg.109)

I-16.13  PoincareShifts [OBsolete]

```
[poincareShifts(M: RING|IDEAL):TAGGED("$hp.PSeries")]
```

Description

[OBSOLETE] now called “HilbertSeriesShifts” (I-8.9 pg.110)

I-16.14  PolyAlgebraHom

```
PolyAlgebraHom(Domain: RING, Codomain: RING, images: LIST): RINGHOM
```

Description

This function creates the homomorphism of (polynomial) algebras from “R” to “S” with the same ring of coefficients. This is uniquely defined by the images of the indeterminates of “R” which are specified by the entries of “images”.

This is a cleaner mathematical implementation of the function “image [OBSOLESCENT]” (I-9.10 pg.119) in CoCoA-4.

```
/**/ Use R ::= QQ[x,y,z];
/**/ S ::= QQ[x[1..3]];  
/**/ phi := PolyAlgebraHom(R, S, indets(S));
/**/ phi(x^2-y);
x[1]^2 -x[2]
```
/**/ S ::= QQ[a];
/**/ phi := PolyAlgebraHom(R, S, [RingElem(S,"a"),1,0]);
/**/ phi(x^2-y);
a^2 -1

/**/ phi := PolyAlgebraHom(R, QQ, [2,1,0]); --> evaluate at [2,1,0]
/**/ phi(x^2-y);
3

See Also: apply(I-1.12 pg.30), CanonicalHom(I-3.2 pg.41)

I-16.15 PolyRingHom

Syntax

PolyRingHom(R: RING, S: RING, CoeffHom: RINGHOM, images: LIST): RINGHOM

Description

This function create the homomorphism of (polynomial) algebras between R and S. This is uniquely defined by the images of the indeterminated of R and the homomorphism CoeffRing(R) into S.

Example

/**/ R ::= QQ[x,y];
/**/ S ::= QQ[a,b,c];
/**/ SmodJ := NewQuotientRing(S, ideal(RingElem(S,"a")^2-1));
/**/ Use SmodJ;
/**/ phi := PolyRingHom(R, SmodJ, CanonicalHom(QQ,SmodJ), [a,b]);
/**/ Use R;
/**/ phi(x);
(a)

See Also: apply(I-1.12 pg.30), CanonicalHom(I-3.2 pg.41)

I-16.16 PowerMod

Syntax

PowerMod(A: INT, B: INT, M: INT): INT

Description

This function calculates efficiently an integer power modulo a given modulus. Thus “PowerMod(A, B, M)” is equal to “mod(A^B, M)”, but the former is computed faster. “B” must be non-negative.

Example

/**/ PowerMod(12345,41041,41041); -- 41041 is a Carmichael number
12345
/**/ PowerMod(123456789,987654321,32003); -- cannot compute 123456789^987654321 directly
2332
I-16.17  PreImage

**syntax**

```plaintext
PreImage(phi: RINGHOM, f: RINGELEM): RECORD
```

**Description**

This function returns the preimage of “f” via “phi”. More precisely it returns a record with fields “IsInImage” and “ker”, and “OnePreImage” if “f” is in the image of “phi”.

```plaintext
/**/ QQxyz ::= QQ[x,y,z];
/**/ QQab ::= QQ[a,b];
/**/ phi := PolyAlgebraHom(QQxyz, QQab, [a+1, a*b+3, b^2]);
/**/ IsInjective(phi);
false
/**/ ker(phi);
ideal(-x^2*z +y^2 +2*x*z -6*y -z +9)
/**/ IsSurjective(phi);
false
/**/ Use QQab;
/**/ PreImage(phi, b);
record[IsInImage := false, ker := ideal(-x^2*z +y^2 +2*x*z -6*y -z +9)]
/**/ indent(PreImage(phi, a^2));
record[
  IsInImage := true,
  OnePreImage := x^2 -2*x +1,
  ker := ideal(-x^2*z +y^2 +2*x*z -6*y -z +9)
]
/**/ phi(ReadExpr(QQxyz, "x^2 - 2*x + 1"));
a^2
/**/ phi(ReadExpr(QQxyz, "x^2 - 2*x + 1 + (-x^2*z +y^2 +2*x*z -6*y -z +9)"));
a^2
```

See Also:  ker(I-11.1 pg.153), IsSurjective(I-9.66 pg.144)

I-16.18  PreprocessPts

**syntax**

```plaintext
PreprocessPts(Pts: MAT, Toler: MAT): RECORD
PreprocessPtsGrid(Pts: MAT, Toler: MAT): RECORD
PreprocessPtsAggr(Pts: MAT, Toler: MAT): RECORD
PreprocessPtsSubDiv(Pts: MAT, Toler: MAT): RECORD
```

**Description**

Thanks to Maria-Laura Torrente.

These functions detect groupings of close points, and choose a single representative for them (which lies within the given tolerance of each original point); the result is the list of those representatives, and the number of original points associated to each representative.
Chapter I-16. P

The first argument is a matrix whose rows represent a set of points in k-dimensional space, and the second argument is row-matrix of k tolerances (one for each dimension).

The return value is a record containing two fields: “NewPoints” contains a matrix whose rows represent a list of “well-separated” points, and “weights” which contains the number of input points associated to each output point.

There are three underlying algorithms: “Grid” is fast but crude; “Subdiv” works best when the original points are densely packed (so the result will be a small list); finally “Aggr” is best suited to situations where the original points are less densely packed.

The function “PreprocessPts” automatically chooses between “Subdiv” and “Aggr” with the aim of minimising computation time. Note that the “Aggr” and “Subdiv” methods regard the tolerances as being slightly flexible.

For a full description of the algorithms we refer to the paper J. Abbott, C. Fassino, L. Torrente “Thinning Out Redundant Empirical Data” (Mathematics in Computer Science, 2007).

Example:

```plaintext
/**/ Pts := matrix([[-1,0],[0,0],[1,0],[99,1],[99,0],[99,-1]]);
/**/ Toler := RowMat([3,3]);
/**/ PreprocessPts(Pts, Toler);
record[NewPoints := matrix(QQ,

[[99, 0],
 [0, 0]]), weights := [3, 3]]

/**/ PreprocessPts(Pts, RowMat([0.8,0.8]));
record[NewPoints := matrix(QQ,

[-1/2, 0],
 [1, 0],
 [99, 1/2],
 [99, -1]], weights := [2, 1, 2, 1]]

/**/ PreprocessPtsAggr(Pts, RowMat([0.9,0.9])); -- exhibits tolerance flex
record[NewPoints := matrix(QQ,

[0, 0],
 [99, 0]], weights := [3, 3]]
```

I-16.19 PrimaryDecomposition

Syntax

```
PrimaryDecomposition(I: IDEAL): LIST of IDEAL
```

Description

This function returns the primary decomposition of the ideal I. Currently it is implemented ONLY for squarefree monomial ideals using the Alexander dual technique. See “FrbPrimaryDecomposition” (I-6.20 pg.92) for monomial ideals.

Example:

```plaintext
/**/ Use R ::= QQ[x,y,z];
/**/ PrimaryDecomposition(***Ideal(xy, yz, zx)***);
[ideal(y, z), ideal(x, z), ideal(x, y)]
```

See Also: PrimaryDecomposition0(I-16.20 pg.209), PrimaryDecompositionGTZ0(I-16.21 pg.209), FrbPrimaryDecomposition(I-6.20 pg.92), EquiIsoDec(I-5.8 pg.78)
I-16.20 PrimaryDecomposition0

Syntax

PrimaryDecomposition0(I: IDEAL): RECORD of LIST of IDEAL

Description

This function returns the primary decomposition of the 0-dimensional ideal I. (Monico-Robbiano-Krenzer probabilistic algorithm, Gao-Wan-Wang finite characteristic algorithm) It will be improved and extended in future versions of CoCoA.

Implementation by Elisa Palezzato.

Example

```/**/ Use R ::= QQ[x,y,z];
/**/ PD := PrimaryDecomposition0(ideal(x-z, y^2-1, z^2));
/**/ indent(PD);
record[
  IsCertified := true,
  PrDec_I := [ideal(y -1, x -z, z^2), ideal(y +1, x -z, z^2)]
]
/**/ Use ZZ/(2)[x,y,z];
/**/ PD := PrimaryDecomposition0(ideal(x-z, y^2-1, z^2));
/**/ indent(PD);
record[
  IsCertified := true,
  PrDec_I := [ideal(x +z, z^2, y^2 +1)]
]```

See Also: PrimaryDecompositionGTZ0(I-16.21 pg.209)

I-16.21 PrimaryDecompositionGTZ0

Syntax

PrimaryDecompositionGTZ0(I: IDEAL): RECORD of LIST of IDEAL

Description

This function returns the primary decomposition of the 0-dimensional ideal I. (GTZ algorithm) It will be improved and extended in future versions of CoCoA.

Implemented by Luis David Garcia (updated to CoCoA-5 by Anna M. Bigatti).

Example

```/**/ Use R ::= QQ[x,y,z];
/**/ PD := PrimaryDecompositionGTZ0(ideal(x-z, y^2-1, z^2));
/**/ indent(PD);
```n

See Also: PrimaryDecomposition0(I-16.20 pg.209)

I-16.22 PrimaryHilbertSeries

Syntax

PrimaryHilbertSeries(I: IDEAL, Q: IDEAL): TAGGED("PSeries")
Description

Let “P” be a polynomial ring, “M” the maximal ideal in “P” generated by the indeterminates, and “(Q+I)/I” a primary ideal for “M/I”. This function computes the Hilbert-Poincare' series of “(P/I)/((Q+I)/I)”.

```plaintext
/**/ Use S ::= QQ[x,y,z];
/**/ I := ideal(x^3-y*z, y^2-x*z, z^2-x^2*y);
/**/ Q := ideal(y, z);
/**/ PS := PrimaryHilbertSeries(I, Q); PS;
/**/ Use S ::= ZZ/(32003)[x,y,z,w];
/**/ I := ***Ideal(x^5 - yz, y^4 - xz^2, xy^3 - zw, x^2z - yw, y^2z^2 - w^3, y^3z - x^2w^2, x^3w - z^2, xyw^2 - z^3, x^3y^2 - w^2, xz^4 - y^2w^3, yz^5 - zw^5, y^3w^5 - z^7, x^2w^7 - z^8, z^9 - yw^8)***;
/**/ Q := ideal(x, y, z);
/**/ PS := PrimaryHilbertSeries(I, Q); PS;
/**/ Use S ::= ZZ/(32003)[x,y,z,w];
/**/ I := ***Ideal(-yz + xw, z^3 - yw^2, -xz^2 + y^2w, -y^3 + x^2z)***;
/**/ Q := ideal(x, y, z^2);
/**/ PS := PrimaryHilbertSeries(I, Q); PS;
/**/ Use S ::= ZZ/(32003)[x,y,z,w];
/**/ I := ***Ideal(-yz + xw, z^3 - yw^2, -xz^2 + y^2w, -y^3 + x^2z)***;
/**/ Q := ideal(x, y, z^2);
/**/ PS := PrimaryHilbertSeries(I, Q); PS;
/**/ Use S ::= ZZ/(32003)[x,y,z,w];
/**/ I := ***Ideal(x^3-y^7, x^2y - xw^3-z^6)***;
/**/ Q := ideal(y, z, w);
/**/ PS := PrimaryHilbertSeries(I, Q); PS;
/**/ Use S ::= ZZ/(32003)[x,y,z,w];
/**/ I := ***Ideal(x^3-y^7, x^2y - xw^3-z^6)***;
/**/ Q := ideal(y, z, w);
/**/ PS := PrimaryHilbertSeries(I, Q); PS;
/**/ Use S ::= ZZ/(32003)[x,y,z];
/**/ I := ideal(z^3);
/**/ Q := ideal(x^2, y^2, x*y, z^2);
/**/ PS := PrimaryHilbertSeries(I, Q); PS;
/**/ Use S ::= ZZ/(32003)[x,y,z];
/**/ I := ideal(x^2, y^2, x*y, z^2);
/**/ Q := ideal(x^2, y^2, x*y, z^2);
/**/ PS := PrimaryHilbertSeries(I, Q); PS;
/**/ Use S ::= ZZ/(32003)[x,y,z];
/**/ I := ideal(x^2, y^2, x*y, z^2);
/**/ Q := ideal(x^2, y^2, x*y, z^2);
/**/ PS := PrimaryHilbertSeries(I, Q); PS;
```

See Also: InitialIdeal(I-9.25 pg.127), TgCone(I-20.5 pg.272)
I-16.23 PrimaryPoincare [OBSOLESCENT]

Description

Renamed “PrimaryHilbertSeries” (I-16.22 pg.209).

See Also: PrimaryHilbertSeries(I-16.22 pg.209)

I-16.24 PrimitiveRoot

Description

Find a primitive root modulo the prime “P”, i.e. a generator of the cyclic multiplicative group of non-zero integers mod “P”.

Currently, the function produces the least positive primitive root.

Example

```plaintext
/**/ PrimitiveRoot(17551561); 97
/**/ PrimitiveRoot(4111); 12;
```

See Also: IsPrime(I-9.56 pg.140)

I-16.25 print

Description

This command displays the value of each of the expressions “E_i”. To insert a newline write “\n”.

The similar command “println” (I-16.29 pg.213) is equivalent to “print” with a final newline.

Example

```plaintext
/**/ for I := 1 To 10 Do print I^2, " "; endfor;
1 4 9 16 25 36 49 64 81 100

/**/ print "a\nb";
a
b
```

See Also: print on(I-16.26 pg.212), println(I-16.29 pg.213), format(I-6.15 pg.90), LaTeX(I-12.2 pg.155), StarPrint, StarSprint(I-19.28 pg.259)
print E: OBJECT on D: DEVICE

Description

This command prints the value of expression E to the device D. Currently, the command can be used to print to files, strings, or the CoCoA window. In the first two cases, the appropriate device must be opened with "OpenOFile" (I-15.5 pg.197) or "OpenOString" (I-15.6 pg.198).

example
/**/ D := OpenOFile("my-test"); -- open "my-test" for output from CoCoA
/**/ Print "hello world" On D; -- print string into "mytest"
/**/ close(D); -- close the file

See “OpenOFile” (I-15.5 pg.197) for an example using output strings. For printing to the CoCoA window, just use “print E” which is short for “print E On DEV.OUT”.

See Also: Introduction to IO(II-6.1 pg.309), OpenIFile(I-15.2 pg.195), OpenOFile(I-15.5 pg.197), OpenIString(I-15.3 pg.196), OpenOString(I-15.6 pg.198), print(I-16.25 pg.211), println(I-16.29 pg.213)

PrintBettiDiagram

syntax
PrintBettiDiagram(X: IDEAL or (quotient)RING or MODULE)
PrintBettiDiagram(X: LIST(res) or RECORD(diagram))

Description

This function prints the ("Macaulay-style") Betti diagram for “M”.

example
/**/ Use R := QQ[t,x,y,z];
/**/ I := ideal(x^2-y*t, x*y-z*t, x*y);
/**/ RES := res(I);
/**/ PrintRes(RES);
0 --> R(-5)^2 --> R(-4)^4 --> R(-2)^3
/**/ B := BettiDiagram(RES); indent(B);
record[
   Diagram := matrix(ZZ,
   [[3, 0, 0],
   [0, 4, 2]]),
   FirstShift := 2
]
/**/ PrintBettiDiagram(RES); -- same as PrintBettiDiagram(I or B)
0 1 2
---------------------
2: 3 - -
3: - 4 2
---------------------
Tot: 3 4 2

See Also: BettiDiagram(I-2.3 pg.34), BettiMatrix(I-2.4 pg.34), PrintRes(I-16.30 pg.213), PrintBettiMatrix(I-16.28 pg.213)
**I-16.28 PrintBettiMatrix**

**syntax**

PrintBettiMatrix(M: IDEAL|MODULE|Resolution)

**Description**

This function returns the Betti matrix for M.

**example**

```plaintext
/**/ Use R ::= QQ[t,x,y,z];
/**/ I := ideal(x^2-y*t, x*y-z*t, x*y);
/**/ PrintRes(I);
0 --> R^2(-5) --> R^4(-4) --> R^3(-2)
-------------------------------
/**/ PrintBettiMatrix(I);
0 0 0
0 0 3
0 0 0
0 4 0
2 0 0
```

**See Also:** PrintRes(I-16.30 pg.213), PrintBettiDiagram(I-16.27 pg.212)

**I-16.29 println**

**syntax**

println E_1,...,E_n
PrintLn E_1,...,E_n

**Description**

This command is equivalent to “print” (I-16.25 pg.211) with a final newline; in other words, it prints the values of its arguments, then moves the cursor to the next line.

**example**

```plaintext
/**/ for i := 1 to 3 do print i; endfor;
123
/**/ for i := 1 to 3 do println i; endfor;
1
2
3
```

**See Also:** print(I-16.25 pg.211), print on(I-16.26 pg.212)

**I-16.30 PrintRes**

**syntax**

PrintRes(M)

**See Also:**
Chapter I-16. P

Description

This function prints the minimal free resolution of "M". (see “res” (I-18.31 pg.235)).

```plaintext
/**/ Use R := QQ[x,y,z];
/**/ I := ideal(x, y, z^2);
/**/ RES := res(I);
/**/ PrintRes(I);  -- recomputes resolution
0 --> R(-4) --> R(-2)(+R(-3)^2) --> R(-1)^2(+R(-2)
/**/ PrintRes(RES);  -- just prints RES
0 --> R(-4) --> R(-2)(+R(-3)^2) --> R(-1)^2(+R(-2)
/**/ PrintBettiDiagram(RES);  -- just prints the BettiDiagram for RES
0 1 2
--------------------
1: 2 1 -
2: 1 2 1
--------------------
Tot: 3 3 1
```

See Also: PrintBettiDiagram(I-16.27 pg.212), PrintBettiMatrix(I-16.28 pg.213), res(I-18.31 pg.235)

I-16.31 product

```plaintext
syntax
product(L: LIST): OBJECT
product(L: LIST, InitVal: OBJECT): OBJECT
```

Description

This function returns the product of the objects in the list “L” (together with “InitVal”, if specified). When writing a program, if the list “L” may be empty, you must specify “InitVal”.

```plaintext
/**/ Use R := QQ[x,y];
/**/ product([3, x, y^2]);
3*x*y^2

/**/ product(1..40) = factorial(40);
true

/**/ product([]);  -- gives 1 of type INT
1
/**/ product([1, y];
y
/**/ product([3, x], y);
3*x*y
```

See Also: Algebraic Operators(II-3.2 pg.301), sum(I-19.40 pg.264)

I-16.32 protect

```plaintext
syntax
protect X;
protect X : reason;
   where reason: STRING
```
**Description**

This command protects the variable “X” from being assigned to. Attempting to assign to it will produce an error; if a “reason” (STRING) was given it is printed in the error message.

```plaintext
/**/ MaxSize := 99;
/**/ protect MaxSize : "size limit for fast computation";
-- /**/ MaxSize := 1000; -- > !!! ERROR !!!
ERROR: Cannot set "MaxSize" (size limit for fast computation)

/**/ unprotect MaxSize; -- > remove protection, X may be assigned to now
/**/ MaxSize := 1000; -- > OK
```

**See Also:** unprotect(I-21.2 pg.279)

**I-16.33 PthRoot**

**Syntax**

```
PthRoot(X: RINGELEM): RINGELEM
```

**Description**

This function returns the p-th root of a polynomial over a finite field. If no p-th root exists then an error is signalled. p is the characteristic of the field.

```plaintext
/*/ Use R := ZZ/(7)[x,y];
/*/ F := x^7-y^14+3;
/*/ PthRoot(F);
-y^2+x+3
```

**See Also:** IsFiniteField(I-9.45 pg.135), IsPthPower(I-9.58 pg.141)
Chapter I-17

Q

I-17.1 QQ

**syntax**

QQ

**Description**

This system variable is constant; its value is the field of rationals. Its name is protected so that it cannot be re-assigned to any other value.

Please note: this is a “variable”, so in “define/endefine” use “RingQQ” (I-18.42 pg.240) instead (or import it with “TopLevel” (I-20.9 pg.274)).

**example**

```c
/**/ Use QQ;
/**/ type(5);
INT
/**/ type(RingElem(QQ, 5));
RINGELEM
/**/ QQ = RingQQ();
true
```

**See Also:** ZZ(I-25.4 pg.292), NewQuotientRing(I-14.9 pg.182), RingQQ(I-18.42 pg.240), TopLevel(I-20.9 pg.274)

I-17.2 quit

**syntax**

quit

**Description**

This command is used to quit CoCoA. It may be used only at top level.

**See Also:** ciao(I-3.12 pg.45)
I-17.3 QuotientBasis

**syntax**

```
QuotientBasis(I: IDEAL): LIST
```

**Description**

This function determines a vector space basis (of power products) for the quotient space associated to a zero-dimensional ideal. That is, if $R$ is a polynomial ring with field of coefficients $k$, and $I$ is a zero-dimensional ideal in $R$ then \( \text{QuotientBasis}(I) \) is a set of power products forming a $k$-vector space basis of $R/I$.

The actual set of power products chosen depends on the term ordering in the ring $R$: the power products chosen are those not divisible by the leading term of any member of the reduced Groebner basis of $I$ (and consequently they form a factor-closed set).

**example**

```plaintext
/**/ Points := \[[\text{Rand}(\text{-}9,9) \mid \text{N In 1..3}] \mid \text{S In 1..25}\];
/**/ Use P ::= QQ[x,y,z];
/**/ I := IdealOfPoints(P, mat(QQ, Points));
/**/ QuotientBasis(I);
[1, z, z^2, z^3, y, y*z, y*z^2, y*z^3, y^2, y^2*z, y^2*z^2, y^3, x,
 x*z, x*z^2, x*z^3, x*y, x*y*z, x*y*z^2, x*y^2, x^2, x^2*z, x^2*y, x^3]
```

See Also: IdealOfPoints(I-9.5 pg.116), IsFactorClosed(I-9.43 pg.134)

I-17.4 QZP

**syntax**

```
QZP(F: RINGELEM): RINGELEM
QZP(F: LIST of POLY): LIST of POLY
QZP(I: IDEAL): IDEAL
```

**Description**

***** NOT YET IMPLEMENTED *****

The functions “QZP” and “ZPQ” (I-25.3 pg.292) map polynomials and ideals of other rings into ones of the current ring. When mapping from one ring to another, one of the rings must have coefficients in the rational numbers and the other must have coefficients in a finite field. The indeterminates in both rings must be identical.

The function “QZP” maps polynomials with rational coefficients to polynomials with coefficients in a finite field; the function “ZPQ” (I-25.3 pg.292) does the reverse, mapping a polynomial with finite field coefficients into one with rational (actually, integer) coefficients. The function “ZPQ” (I-25.3 pg.292) is not uniquely defined mathematically, and currently for each coefficient the least non-negative equivalent integer is chosen. Users should not rely on this choice, though any change will be documented.

**example**

```plaintext
Use R ::= QQ[x,y,z];
F := 1/2*x^3 + 34/567*x*y*z - 890; -- a poly with rational coefficients
Use S ::= ZZ/(101)[x,y,z];
QZP(F); -- compute its image with coeffs in ZZ/(101)
```
\[
-50x^3 - 19xyz + 19
\]
\[
-----------------------------------
\]
\[
G := \text{It};
\]
\[
\text{Use R;}
\]
\[
\text{ZPQ}(G); \quad \text{-- now map that result back to } \mathbb{Q}[x,y,z]
\]
\[
\text{-- it is NOT the same as } F...\]
\[
51x^3 + 82xyz + 19
\]
\[
-----------------------------------
\]
\[
H := \text{It;}
\]
\[
F - H; \quad \text{-- ... but the difference is divisible by } 101
\]
\[
-101/2x^3 - 46460/567xyz - 909
\]
\[
-----------------------------------
\]
\[
\text{Use S;}
\]
\[
\text{QZP}(H) - G; \quad \text{-- } F \text{ and } H \text{ have the same image in } \mathbb{Z}/(101)[x,y,z]
\]
\[
0
\]
\[
-----------------------------------
\]

See Also: Introduction to RINGHOM(III-10.1 pg.359), BringIn(I-2.12 pg.39)
Chapter I-18

R

I-18.1 radical

Syntax

radical(I: IDEAL): IDEAL

Description

This function computes the radical of \( I \) using the algorithm described in the paper


NOTE: at the moment, this implementation works only if the coefficient ring is the rationals or has large enough characteristic.

Example

/**/ Use R ::= QQ[x,y];
/**/ I := ideal(x,y)^3;
/**/ radical(I);
ideal(y, x)

See Also: IsInRadical(I-9.50 pg.138), EquiIsoDec(I-5.8 pg.78), RadicalOfUnmixed(I-18.2 pg.221)

I-18.2 RadicalOfUnmixed

Syntax

RadicalOfUnmixed(I: IDEAL): IDEAL

Description

This function computes the radical of an unmixed ideal.

NOTE: at the moment, this implementation works only if the coefficient ring is the rationals or has large enough characteristic.

Example

/**/ Use R ::= QQ[x,y];
/**/ I := ideal(x^2 - y^2 - 4*x + 4*y, x - 2);
/**/ RadicalOfUnmixed(I);
ideal(x^2 -y^2 -4*x +4*y, x -2, y -2)
/**/ interreduced(gens(It)); -- the result may not be in its simplest form
[y -2, x -2]
See Also: EquiIsoDec(I-5.8 pg.78), radical(I-18.1 pg.221)

I-18.3 random

**syntax**

random(X: INT, Y: INT): INT

**Description**

The function returns a random integer between X and Y, inclusive. The range —X-Y— should be less than $2^{33}$ to assure a uniform distribution.

NOTE: every time you restart CoCoA the sequence of random numbers will be the same (as happens in many programming languages). If you want better randomness, see “seed” (I-19.4 pg.246).

**example**

```cocoa
/**/ random(1,100);
6
/**/ random(-10^4,0);
-3263
```

See Also: randomize(I-18.4 pg.222), randomized(I-18.5 pg.223), RandomSubset(I-18.6 pg.223), Random SubsetIndices(I-18.7 pg.224), RandomTuple(I-18.8 pg.224), RandomTupleIndices(I-18.9 pg.225), seed(I-19.4 pg.246)

I-18.4 randomize

**syntax**

Randomize(V: RINGELEM): RINGELEM

**Description**

***** NOT YET IMPLEMENTED: use random *****

This function replaces the coefficients of terms of the polynomial contained in V with randomly generated coefficients. The result is stored in V, overwriting the original polynomial.

NOTE: It is possible that some coefficients will be replaced by zeroes, i.e., some terms from the original polynomial may disappear in the result.

The similar function “randomized” (I-18.5 pg.223) performs the same operation, but returns the randomized polynomial without modifying the argument.

NOTE: every time you restart CoCoA the sequence of random numbers will be the same (as in other programming languages). If you want total randomness read “seed” (I-19.4 pg.246).

**example**

```cocoa
Use R ::= QQ[x];
F := 1+x+x^2;
Randomized(F);
-2917104644x^2 + 3623608766x - 2302822308
----------------------------------
F;
```

```cocoa
x^2 + x + 1
--------------
Randomize(F);
F;
```
I-18.5  randomized

**syntax**

\[
\begin{align*}
\text{Randomized}(F: \text{RINGELEM}): & \text{RINGELEM} \\
\text{Randomized}(F: \text{INT}): & \text{INT}
\end{align*}
\]

**Description**

***** NOT YET IMPLEMENTED: use random *****

This function with a polynomial argument returns a polynomial obtained by replacing the coefficients of \(F\) with randomly generated coefficients. The original polynomial, \(F\), is unaffected. With an integer argument, it returns a random integer.

NOTE: It is possible that some coefficients will be replaced by zeroes, i.e., some terms from the original polynomial may disappear in the result.

The similar function “randomize” (I-18.4 pg.222) performs the same operation, but returns NULL and modifies the argument.

NOTE: every time you restart CoCoA the sequence of random numbers will be the same (as in other programming languages). If you want total randomness read “seed” (I-19.4 pg.246).

**example**

```
Use R := QQ[x];
F := 1 + x + x^2;
Randomized(F);
-2917104704x^2 + 3623608766x - 2302822308

F;
x^2 + x + 1

Randomized(23);
-3997312402

Use R := ZZ/(7)[x,y];
Randomized(x^2 + 3x - 5);
3x^2 + 2x - 2
```

**See Also:** random(I-18.3 pg.222), randomize(I-18.4 pg.222), seed(I-19.4 pg.246)

I-18.6  RandomSubset

**syntax**

\[
\begin{align*}
\text{RandomSubset}(L: \text{LIST}, \text{K:}\ \text{INT}): & \text{LIST}
\end{align*}
\]

**Description**

The function returns a random subset of “\(L\)” of cardinality “\(K\).” This function can be quite useful for testing properties on some subsets of a large list when testing on all of them would be unfeasible in time and memory (see also “subsets” (I-19.38 pg.263)).
NOTE: the resulting list is sorted as in “L”.

```c
/**/ RandomSubset(["a","b","c","d","e","f","g","h"], 5);
["a", "c", "d", "f", "h"]
/**/ indent([[RandomSubset(1..1000, 10) | i in 1..4]]);
[ [160, 182, 215, 219, 349, 588, 628, 811, 886, 905],
  [23, 103, 315, 451, 531, 571, 846, 858, 876],
  [24, 230, 278, 380, 421, 495, 505, 665, 788],
  [81, 274, 299, 378, 414, 616, 828, 844, 870, 946]
]
/**/ binomial(1000, 10); --> too many to fit in memory ;-)
263409560461970212832400
```

See Also: random(I-18.3 pg.222), subsets(I-19.38 pg.263), RandomSubsetIndices(I-18.7 pg.224), RandomTuple(I-18.8 pg.224), RandomTupleIndices(I-18.9 pg.225)

### I-18.7 RandomSubsetIndices

**syntax**

```c
RandomSubsetIndices(N: INT, K: INT): LIST
```

**Description**

The function returns a random subset of “1..N” of cardinality “K”. See also “RandomSubset” (I-18.6 pg.223).

NOTE: the resulting list is sorted.

```c
/**/ RandomSubsetIndices(10, 5);
[1, 3, 4, 6, 8]
```

See Also: random(I-18.3 pg.222), subsets(I-19.38 pg.263), RandomSubset(I-18.6 pg.223), RandomTuple(I-18.8 pg.224), RandomTupleIndices(I-18.9 pg.225)

### I-18.8 RandomTuple

**syntax**

```c
RandomTuple(L: LIST, K: INT): LIST
```

**Description**

The function returns a random tuple of “L” of cardinality “K”. This function can be quite useful for testing properties on some tuples of a large list when testing on all of them would be unfeasible in time and memory (see also “tuples” (I-20.14 pg.278)).

```c
/**/ RandomTuple(["a","b","c","d","e","f","g","h"], 5);
["b", "h", "g", "e"]
/**/ indent([[RandomTuple(-9..9, 10) | i in 1..4]]);
[ [-5, -3, 1, 8, -4, 6, -5, -7, -1, 1],
  [-5, -8, 0, -9, -2, -1, 3, -6, 6, -3],
```
See Also: random(I-18.3 pg.222), subsets(I-19.38 pg.263), RandomSubset(I-18.6 pg.223), RandomSubsetIndices(I-18.7 pg.224), RandomTupleIndices(I-18.9 pg.225)

I-18.9 RandomTupleIndices

**syntax**

RandomTupleIndices(N: INT, K: INT): LIST

**Description**

The function returns a random tuple of “1..N” of cardinality “K”. See also “RandomTuple” (I-18.8 pg.224).

**example**

```plaintext
/**/ RandomTupleIndices(32003, 10);
[4987, 13034, 10044, 7148, 11122, 1144, 21264, 5379, 2934, 7015]
```

See Also: random(I-18.3 pg.222), subsets(I-19.38 pg.263), RandomSubset(I-18.6 pg.223), RandomSubsetIndices(I-18.7 pg.224), RandomTuple(I-18.8 pg.224)

I-18.10 rank [OBSOLETE]

**syntax**

[OBSOLETE]

**Description**

See “rk” (I-18.47 pg.242)

I-18.11 RationalAffinePoints

**syntax**

RationalAffinePoints(L: LIST of RINGELEM): LIST of LIST of RINGELEM

**Description**

This function returns the list of affine rational solutions (points) of a 0-dimensional polynomial system “L”. See also “RationalSolve” (I-18.13 pg.226)

**example**

```plaintext
/**/ Use QQ[x,y,z];
/**/ L := [x^3-y^2+z-1, x-2, (y-3)*(y+2)];
/**/ RationalAffinePoints(L);
[[2, -2, -3], [2, 3, 2]]
```

See Also: LinSolve(I-12.14 pg.161), RationalSolve(I-18.13 pg.226)
I-18.12 RationalProjectivePoints

**syntax**

```
RationalProjectivePoints(L: LIST of RINGELEM): LIST of LIST of RINGELEM
```

**Description**

This function returns the list of projective rational solutions (points) of a 0-dimensional polynomial system “L”. See also “RationalSolve” (I-18.13 pg.226)

```plaintext
/**/ Use QQ[x,y,z];
/**/ L := [x^3-y^2*x, x-2*z];
/**/ RationalProjectivePoints(L);
[[0, 1, 0], [1, -1, 1/2], [1, 1, 1/2]]
```

See Also: LinSolve(I-12.14 pg.161), RationalSolve(I-18.13 pg.226)

I-18.13 RationalSolve

**syntax**

```
RationalSolve(L: LIST of RINGELEM): LIST of LIST of RINGELEM
```

**Description**

This function returns the list of rational solutions (points) of a 0-dimensional polynomial system “L” (see also “ApproxSolve” (I-1.13 pg.30)). Tries to be clever: if some indeterminates do not appear in “L” they are ignored, if the polynomials in “L” are homogeneous it returns the projective points.

```plaintext
/**/ Use QQ[x,y,z];
/**/ L := [x^3-y^2+z-1, x-2, (y-3)*(y+2)];
/**/ RationalSolve(L);
[[2, -2, -3], [2, 3, 2]]

/**/ L := [x^3-y^2+z-1, x^2-2, (y-3)*(y+2)];
/**/ RationalSolve(L);
[]

/**/ len(ApproxSolve(L));
4

/**/ L := [x^3-y^2+1, (y-3)*(y+2)]; -- ignores the indeterminate z
/**/ RationalSolve(L);
[[2, 3]]
```

See Also: ApproxSolve(I-1.13 pg.30), LinSolve(I-12.14 pg.161), RationalAffinePoints(I-18.11 pg.225), RationalProjectivePoints(I-18.12 pg.226)

I-18.14 RatReconstructByContFrac, RatReconstructByLattice

**syntax**

```
RatReconstructByContFrac(X: INT, M: INT): RECORD
RatReconstructByContFrac(X: INT, M: INT, threshold: INT): RECORD
RatReconstructByLattice(X: INT, M: INT): RECORD
RatReconstructByLattice(X: INT, M: INT, threshold: INT): RECORD
```
Description

These functions attempt to reconstruct rational numbers from a modular image “X mod M”. The algorithms are fault-tolerant: they will succeed provided that “X” is correct modulo a sufficiently large factor of “M”.

NOTE: so that the heuristic can work, the modulus must be larger than strictly necessary; indeed, reconstruction always fails if “M” is small.

The result is a record: the boolean field “failed” is “true” if no “convincing” result was found; otherwise it is “false”, and a second field, called “ReconstructedRat”, contains the value reconstructed.

An optional third argument, “threshold”, determines what “convincing” means: a higher value gives a more reliable answer, but may need a larger modulus before the answer is found.

There are two different underlying heuristic algorithms: a faster one based on continued fractions, and a slower one based on 2-dimensional lattice reduction. See arXiv: “http://arxiv.org/abs/1303.2965”

```plaintext
/**/ X := 3333333333;
/**/ M := 10^10;
/**/ RatReconstructByContFrac(X,M);
record[ReconstructedRat := -1/3, failed := false]
/**/ X := 3141592654;
/**/ M := 10^10;
/**/ RatReconstructByContFrac(X,M);
record[failed := true]
```

See Also: RatReconstructWithBounds(I-18.15 pg.227), CRT(I-3.48 pg.61)

I-18.15 RatReconstructWithBounds

**syntax**

RatReconstructWithBounds(e: INT, P: INT, Q: INT, res: LIST of INT, mod: LIST of INT): RECORD

**Description**

This function attempts to reconstruct a rational number from a collection of residue-modulus pairs “(res[i],mod[i])”. The function also requires the input of three bounds: “e” is an upper bound on the number of bad moduli, and “P” and “Q” are upper bounds for (respectively the numerator and denominator of) the rational to be reconstructed.

The result is a record: the boolean field “failed” is “true” if no result exists; otherwise it is “false”, and a second field, called “ReconstructedRat”, contains the value reconstructed.

```plaintext
/**/ moduli := [11,13,15,17,19];
/**/ residues := [-2, -5, 0, 7, 4];
/**/ RatReconstructWithBounds(1,10,10,residues,moduli);
record[ReconstructedRat := 1/5, failed := false]
/**/ RatReconstructWithBounds(0,10,10,residues,moduli);
record[failed := true]
```

See Also: CRT(I-3.48 pg.61), RatReconstructByContFrac, RatReconstructByLattice(I-18.14 pg.226)

I-18.16 ReadExpr

**syntax**

ReadExpr(R: RING, expr: STRING): RINGELEM
Description

This function reads a “RINGELEM” expression from a “STRING”. It is handy to input elements defined in different rings without calling “use” (I-21.5 pg.280).

```plaintext
definition
/**/
P ::= QQ[a,b];
/**/
S := NewPolyRing(NewFractionField(P), "x,y");
/**/
ReadExpr(S, "(a^2-b^2)*(x+y)/(a+b)");
(a -b)*x +(a -b)*y
```

See Also: RingElem(I-18.38 pg.238)

### I-18.17 RealRootRefine

description

This function computes a refinement of a real root of a univariate polynomial over QQ to the desired precision (width of isolating interval). The starting root must be a record produced by “RealRoots” (I-18.18 pg.228).

```plaintext
definition
/**/
RR := RealRoots(x^2-2);
/**/
RealRootRefine(RR[1], 1/2);
record[CoeffList := [-1, 0, 2], inf := -3/2, sup := -5/4]
/**/
RR := [RealRootRefine(Root, 10^(-20)) | Root In RR];
/**/
FloatStr(RR[1].inf);
-1.414213562*10^0
```

See Also: RealRoots(I-18.18 pg.228), RealRootsApprox(I-18.19 pg.229), RootBound(I-18.49 pg.243)

### I-18.18 RealRoots

description

This function computes isolating intervals for the real roots of a univariate polynomial over QQ. It returns the list of the real roots, where a root is represented as a record containing either the exact root (if the fields “inf” and “sup” are equal), or an open interval (inf, sup) containing the root. A third field (called CoeffList) has an obscure meaning.

An optional second argument specifies the maximum width an isolating interval may have. An optional third argument specifies a closed interval in which to search for roots.

The interval represented by a root record may be refined by using the function “RealRootRefine” (I-18.17 pg.228). The function “RealRootsApprox” (I-18.19 pg.229) may be more useful to you: it produces rational approximations to the real roots (but these cannot later be refined).
### I-18.19 RealRootsApprox

**Syntax**

- \( \text{RealRootsApprox}(F: \text{RINGELEM}) : \text{LIST} \)
- \( \text{RealRootsApprox}(F: \text{RINGELEM}, \text{Precision}: \text{RAT}) : \text{LIST} \)
- \( \text{RealRootsApprox}(F: \text{RINGELEM}, \text{Precision}: \text{RAT}, \text{Interval}: [\text{RAT}, \text{RAT}]) : \text{LIST} \)

**Description**

This function computes rational approximations to the real roots of a univariate polynomial (with rational coefficients).

An optional second argument specifies the maximum separation between the approximations produced and the corresponding exact root. An optional third argument specifies a closed interval in which to search for roots.

**Example**

```plaintext
/**/ RealRootsApprox(x^2-2);
[-3037000499/2147483648, 3037000499/2147483648]
/**/ RR := RealRootsApprox(x^2-2, 10^-15, [0, 2]);
/**/ RR;
[652190891266391107/4611686018427387904]
/**/ FloatStr(RR[1], 15);
1.41421356237310*10^0
```

**See Also:** RealRoots(I-18.18 pg.228), RootBound(I-18.49 pg.243)

### I-18.20 record

**Syntax**

- \( \text{record}[F_1 := \text{OBJECT}, \ldots, F_n := \text{OBJECT}] \)
  - where each \( F_i \) is a field name
returns RECORD

Description

This constructor creates a record with fields called “F_1”..., “F_n”. The empty record is given by “record[]”. Records in CoCoA are “open” in the sense that new fields may be added after the record is first defined. The names allowed for the fields are the same as those allowed for variables.

The dot operator is used to access the fields in a record.

```cocoa
/**/ P := record[height := 10, width := 5];
/**/ P.height * P.width;
50
/**/ P.area := It; --> creates a new field called "area"
/**/ P;
record[area := 50, height := 10, width := 5]
```

See Also: record field selector(I-18.21 pg.230), fields(I-6.5 pg.85)

I-18.21  record field selector

```cocoa
R.FieldName
R["FieldName"]
where R is a RECORD
```

Description

A record is a data structure containing named entries. They are created using the command “record” (I-18.20 pg.229). Each entry may be selected using the “dot operator”, or equivalently a string index.

```cocoa
/**/ rec := record[name := "David", year := 1961];
/**/ rec.name;
David
/**/ rec.year := 1849; --> change value of a field
/**/ rec.surname := "Copperfield"; --> create a new field
/**/ rec["year"]; -- alternative syntax
1849
/**/ foreach F in fields(rec) do print rec[F]; endforeach;
DavidCopperfield1849
```

See Also: record(I-18.20 pg.229)

I-18.22  ReducedGBasis

```cocoa
ReducedGBasis(I: IDEAL): LIST of RINGELEM
ReducedGBasis(I: MODULE): LIST of MODULEELEM
```
**Description**

This function returns a list whose components form a reduced Groebner basis for the ideal (or module) “I” with respect to the term-ordering of the polynomial ring of “I”.

```cocoa
/**/ Use R ::= QQ[x,y];
/**/ I := ideal(x^4-x^2, x^3-y);
/**/ ReducedGBasis(I);
[x*y -y^2, x^2 -y^2, y^3 -y]
```

See Also: GBS(1.7.1 pg.95)

**I-18.23 ref**

**syntax**

ref X

where X is the identifier of a CoCoA variable.

**Description**

The keyword “ref” is used to pass a parameter ”by reference” to a function which may modify its value (e.g. “append” (I-1.11 pg.29)). The keyword “ref” alerts the programmer to the possibility that the value may be changed during the call.

To write a new function which can modify some parameters use the same keyword “ref” to identify which formal parameters are to be passed by reference. The following example illustrates the difference between passing by reference and passing by value.

```cocoa
/**/ Define CallByRef(ref L) -- "call by reference": The variable referred
/**/ L := "new value"; -- to by L is changed.
/**/ EndDefine;
/**/ M := "old value";
/**/ CallByRef(ref M); -- here "ref" recalls that M might change
/**/ PrintLn M;  
  new value

/**/ Define CallByVal(L) -- "call by value": The value of L is passed to
/**/ L := "new value"; -- the function.
/**/ Return L;
/**/ EndDefine;
/**/ L := "old value";
/**/ CallByVal(L);
  new value

/**/ PrintLn L;  
  old value
```

See Also: define(I-4.4 pg.64)

**I-18.24 RefineGCDFreeBasis**

**syntax**

RefineGCDFreeBasis(B: LIST of INT, N: INT): LIST of INT
Description

This function computes a refined GCD free basis by adjoining a given integer to it. The value returned is [NewB, N2] where NewB is the refined basis and N2 is the part of N coprime to every element of B.

```plaintext
/**/ B := GCDFreeBasis([Fact(10), binomial(20,10)]); B;
[14175, 4, 46189]
/**/ RefineGCDFreeBasis(B, 15);
[[7, 3, 5, 4, 46189], 1]
```

See Also: GCDFreeBasis(I-7.5 pg.97)

I-18.25  reg

**syntax**

```plaintext
reg(I: IDEAL): INT
reg(R: (Quotient)RING): INT
```

**Description**

These functions computes the Castelnuovo-Mumford regularity of an ideal. The implementation of “Reg” using Bermejo-Gimenez Algorithm.

Implemented by Eduardo Saenz-de-Cabezón (updated to CoCoA-5 by Anna M. Bigatti).

NOTE: this is different from “RegularityIndex” (I-18.26 pg.232), the regularity of a Hilbert Function.

```plaintext
/**/ Use R ::= QQ[x,y,z];
/**/ I := ideal(x^3, y^2);
/**/ reg(I);
4
/**/ reg(R/I);
3
/**/ PrintRes(I);
0 --> R(-5) --> R(-2)(+)R(-3)
/**/ PrintBettiDiagram(I);
0    1
---------
 2:   1  -
 3:   1  -
 4:   -  1
---------
Tot:  2  1
```

See Also: res(I-18.31 pg.235), PrintRes(I-16.30 pg.213), PrintBettiDiagram(I-16.27 pg.212), PrintBettiMatrix(I-16.28 pg.213), RegularityIndex(I-18.26 pg.232)

I-18.26  RegularityIndex

**syntax**

```plaintext
RegularityIndex(R: RING or TAGGED(“Quotient”)): INT
```
Description

This function computes the regularity index of a Hilbert function. The input might be expressed as a Hilbert function or as the corresponding Hilbert series (computed with standard weights).

```casa
/**/ Use R := QQ[x,y,z];
/**/ Quot := R/ideal(x^3, y^2);
/**/ HilbertFn(Quot);
H(0) = 1
H(1) = 3
H(2) = 5
H(t) = 6  for t >= 3
/**/ RegularityIndex(HilbertFn(Quot));
3
/**/ RegularityIndex(HilbertSeries(Quot));
3
```

See Also: HilbertFn(I-8.5 pg.107), HilbertSeries(I-8.7 pg.108)

I-18.27  RelNotes

Syntax

RelNotes()

Description

This function prints the release notes of the version you are running.

```casa
RelNotes();
```

I-18.28  ReloadMan

Syntax

ReloadMan()

Description

This function reloads the xml source of the manual “CoCoAHelp.xml” (in directory “CoCoAManual”) and recreates the internal manual index in a running CoCoA-5 (instead of closing and re-opening CoCoA...).

It is useful “only for developers” working on the manual and making substantial changes to “CoCoAHelp.xml”. After adding a new entry the index needs updating, but if the change is just in the description of an existing entry (so the internal index is still valid) there is no need to reload the manual: the description is always searched in the current file.

```casa
/**/ ReloadMan();
```
I-18.29 remove

**Syntax**

remove(ref L: LIST, N: INT)

**Description**

This function removes the “N”-th component from “L”; it changes the value of “L”. Use the function “WithoutNth” (I-23.4 pg.286) to create a new list containing the elements of “L” except the “N”-th (without changing “L”).

**Example**

```plaintext
/**/ Use R ::= QQ[x,y,z];
/**/ L := indets(R);
/**/ L;
[x, y, z]
/**/ remove(ref L,2);
/**/ L;
[x, z]
```

**See Also:** WithoutNth(I-23.4 pg.286)

I-18.30 repeat

**Syntax**

repeat C until B
repeat C endrepeat

(where C is a sequence of commands and B is BOOL)

**Description**

In the first form, the command sequence “C” is repeated until “B” evaluates to “false”. Unlike the “while” (I-23.3 pg.286) command, “C” is executed at least once. Note that there is no “endrepeat” following “B”.

**Example**

```plaintext
/**/ Define GCD_Euclid(A, B)
/**/ Repeat
/**/ R := mod(A, B);
/**/ A := B;
/**/ B := R;
/**/ Until B = 0;
/**/ Return A;
/**/ EndDefine;

/**/ GCD_Euclid(6,15);
3

/**/ N := 0;
/**/ repeat
/**/ N := N+1;
/**/ PrintLn N;
/**/ If N > 5 Then Break; EndIf;
/**/ endrepeat;
```
I-18.31. res

Syntax

res(M: IDEAL): LIST
res(M: MODULE): LIST

Description

This function returns the minimal free resolution of "M". "res" only works in the homogeneous context, and the coefficient ring must be a field.

NOTE: the current implementation (CoCoA-5.1.0) is very naive so it might be very slow (better slow than nothing?).

Example

```plaintext
/**/ Use R ::= QQ[x,y,z];
/**/ I := ideal(x, y, z^2);
/**/ PrintRes(R/I);
0 --> R(-4) --> R(-2)(+)R(-3)^2 --> R(-1)^2(+)R(-2) --> R
/**/ indent(Res(R/I),2);
[
    QuotientRing(RingDistrMPolyClean(QQ, 3), ideal(x, y, z^2)),
    ideal(
        y,
        x,
        z^2
    ),
    SubmoduleRows(F, matrix(
        [x, -y, 0],
        [0, z^2, -x],
        [z^2, 0, -y]
    )),
    SubmoduleRows(F, matrix(
        [z^2, y, -x]
    ))
]
```

See Also: PrintBettiDiagram(I-16.27 pg.212), PrintBettiMatrix(I-16.28 pg.213), PrintRes(I-16.30 pg.213)

I-18.32. Reset [OBSOLETE]

Syntax

[OBSOLETE]
Description

[OBSOLETE]

I-18.33  ResetPanels [OBSOLETE]

syntax

[OBSOLETE]

Description

[OBSOLETE]

I-18.34  resultant

syntax

resultant(F: RINGELEM, G: RINGELEM, X: RINGELEM): RINGELEM

Description

This function returns the resultant of the polynomials F and G with respect to the indeterminate X.

/**/ Use R ::= QQ[p,q,x];
/**/ F := x^3+p*x-q;  G := deriv(F, x);
/**/ resultant(F, G, x);
4*p^3 +27*q^2

See Also:  discriminant(I-4.18 pg.72), sylvester(I-19.44 pg.266)

I-18.35  return

syntax

return
return E

Description

This command is used to exit from a procedure/function. The latter form returns the value of the expression E to the user. As a safety measure all “return”s in a function/procedure must be of the same kind: either they all return a value (function) or none returns a value (procedure). To exit from a loop see “break” (I-2.11 pg.38).

/**/ Define Rev(L) -- reverse a list
/**/ If len(L) < 2 Then Return L; EndIf;
/**/ M := Rev(Tail(L));  -- recursive function call
/**/ append(ref M, L[1]);
/**/ Return M;
/**/ EndDefine;
I-18.36 reverse, reversed

 syntax

 reverse(ref L: LIST)
 reversed(L: LIST): LIST

 Description

 The function “reverse” reverses the order of the elements of the list in “L”; it changes the value of “L” and returns nothing. The function “reversed” returns the reversed list without changing “L”.

 example

 /**/ L := [1,2,3,4];
 /**/ reverse(ref L);
 /**/ L; -- L has been modified
 [4, 3, 2, 1]

 /**/ M := [1,2,3,4];
 /**/ reversed(M); -- the reversed list is returned
 [4, 3, 2, 1]

 /**/ M; -- M has not been modified
 [1, 2, 3, 4]

 See Also: sort(I-19.17 pg.253), sorted(I-19.19 pg.254)

 I-18.37 RevLexMat

 syntax

 RevLexMat(N: INT): MAT

 Description

 This function return the matrix defining a standard ordering (which is not a term-ordering!).

 example

 /**/ RevLexMat(3);
 matrix([ 0, 0, -1, 0, -1, 0, 0, 0, 0,])
Chapter I-18. R

See Also: OrdMat(I-15.10 pg.199), Orderings(III-9.5 pg.354), StdDegRevLexMat(I-19.31 pg.260), StdDegLexMat(I-19.30 pg.260), LexMat(I-12.6 pg.157), XelMat(I-24.1 pg.289)

I-18.38 RingElem

<table>
<thead>
<tr>
<th>syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>RingElem(R: RING, E: RINGELEM): RINGELEM</td>
</tr>
<tr>
<td>RingElem(R: RING, E: INT): RINGELEM</td>
</tr>
<tr>
<td>RingElem(R: RING, E: RAT): RINGELEM</td>
</tr>
<tr>
<td>RingElem(R: RING, E: STRING): RINGELEM</td>
</tr>
<tr>
<td>RingElem(R: RING, E:[STRING, INT, .., INT]): RINGELEM</td>
</tr>
</tbody>
</table>

Description

This function converts the expression E into a RINGELEM in R, if possible.

Can be used for mapping ring element between rings when a “CanonicalHom” (I-3.2 pg.41) exists. For other homomorphisms search see “RINGHOM” (III-10 pg.359).

In some cases “ReadExpr” (I-18.16 pg.227) may be handier: it reads a whole expression without function calls from a “STRING”.

<table>
<thead>
<tr>
<th>example</th>
</tr>
</thead>
</table>
| /**/ Use P ::= QQ[x,y,z];
| /**/ -- RINGELEM (via CanonicalHom)
| /**/ F := 2*x-3;
| --- /**/ F/LC(F); -- !!! ERROR !!! LC(F) in CoeffRing(P)
| /**/ F/RingElem(P,LC(F));
| x +1
| --- /**/ 1/x; -- !!! ERROR !!! x in P is not invertible
| /**/ K := NewFractionField(P);
| /**/ 1/RingElem(K, x); -- x in K is invertible
| 1/x
| /**/ Use P ::= ZZ/(5)[x,y,z];
| /**/ -- INT and RAT
| /**/ RingElem(P, 7);
| 2
| /**/ RingElem(P, 3/2);
| -1
| /**/ -- STRING (indet name, symbol)
| /**/ S ::= QQ[x,y,z[1..4,3..7]];
| /**/ 7*RingElem(P, "x"); -- x as an element of P
| 2*x
| /**/ 7*RingElem(S, "x"); -- x as an element of S
| 7*x
| /**/ 7*RingElem(S, ["z",2,5]);
| 7*z[2,5]
| /**/ ReadExpr(S, "((7/3)*z[2,5] - 1)^2" ); -- expr without function calls
| 49*z[2,5]^2 -14*z[2,5] +1

See Also: RingOf(I-18.41 pg.239), AsINT(I-1.15 pg.31), AsRAT(I-1.16 pg.32), IndetName(I-9.20 pg.124), IndetSubscripts(I-9.22 pg.126), CanonicalHom(I-3.2 pg.41), ReadExpr(I-18.16 pg.227)
I-18.39  RingEnv [OBSOLETE]

Syntax

[OBSOLETE]

Description

See “RingOf” (I-18.41 pg.239).

See Also:  RingOf(I-18.41 pg.239)

I-18.40  RingID

Syntax

RingID(R: RING): INT

Description

This function returns the identification number of the ring “R”. Two rings are considered equal if and only if they have the same ID: this means they have the same internal implementation.

This function was called “ID” in CoCoA-5.1.1.

Example

```cocoalisp
/**/ R ::= QQ[x,y,z];
/**/ R;
RingWithID(9, "QQ[x,y,z]")
/**/ S ::= QQ[x,y,z];
/**/ R = S;
false
/**/ RingID(R);
7
/**/ RingID(S);
8
-- /**/ RingElem(R,"x") = RingElem(S, "x"); --> !!! ERROR !!! mixed rings

/**/ S := R; -- or S := RingOf( some element in R )
/**/ R = S;
true
```

See Also: print(I-16.25 pg.211), println(I-16.29 pg.213), Evaluation and Assignment(II-4 pg.305)

I-18.41  RingOf

Syntax

RingOf(E: RINGELEM|IDEAL|MAT|MODULE): RING

Description

This function returns the ring on which the object “E” is defined.

NOTE: A ring contains many information and two separate rings, even when defined with the same commands, are not ”equal”. When a ring is printed only a few informations are shown, so different rings might look the same.
/**/ Use R ::= QQ[x,y,z];
/**/ I := ideal(x,y);
/**/ RingOf(I);
RingDistrMPolyClean(QQ, 3)
/**/ RingOf(mat([[1,2],[3,4]]));
QQ
/**/ Use Qabc ::= QQ[a,b,c];
/**/ F := a^2+b;
/**/ G := a*b+b^2;
/**/ Use S ::= ZZ/(3)[x,y];
/**/ RingOf(F+G); -- F+G is computed in the ring of definition
RingDistrMPolyClean(QQ, 3)
/**/ indets(RingOf(F));
[a, b, c]

See Also: CurrentRing(I-3.49 pg.61), RingsOf(I-18.45 pg.241), BaseRing(I-2.1 pg.33)

I-18.42 RingQQ

** syntax

RingQQ(): RING

** Description

This function returns the ring of rationals. It is particularly useful when you want to use “QQ” (which is a pre-defined top-level “variable”) inside a function.

NOTE: calling “RingQQ” twice gives the same identical ring, whereas calling “NewPolyRing” (I-14.8 pg.182) and “NewFractionField” (I-14.1 pg.179) return each time a new ring.

** example

/**/ QQ = RingQQ();
/**/ Two := RingElem(RingQQ(), 2); Two;
2
/**/ type(Two);
RINGELEM;
/**/ IsQQ(RingOf(Two));
true

See Also: TopLevel(I-20.9 pg.274), QQ(I-17.1 pg.217), RingQQt(I-18.43 pg.240), RingZZ(I-18.46 pg.242)

I-18.43 RingQQt

** syntax

RingQQt(N: INT): RING

** Description

This function returns a polynomial ring over “QQ” with indeterminates t[1]..t[N]. In particular “RingQQt(1)” is the polynomial ring in which Hilbert polynomials are defined.
NOTE: calling “RingQQt(5)” twice gives the same identical ring, whereas calling “NewPolyRing(RingQQ(), SymbolRange("t", 1,5))” returns each time a new ring (therefore incompatible).

```plaintext
/**/ QQt := RingQQt(3); Use QQt;
/**/ (t[1]+1)^3;
t[1]^3 +3*t[1]^2 +3*t[1] +1
/**/ indets(RingQQt(1));
[t]
/**/ indets(RingQQt(5));
[t[1], t[2], t[3], t[4], t[5]]
```

See Also:  TopLevel(I-20.9 pg.274), RingQQ(I-18.42 pg.240), RingZZ(I-18.46 pg.242)

### I-18.44  RingSet [OBSOLETE]

**syntax**

[OBSOLETE]

**Description**

This function has just been renamed “RingsOf” (I-18.45 pg.241) See Also:  RingsOf(I-18.45 pg.241)

### I-18.45  RingsOf

**syntax**

RingsOf(E: LIST|RINGELEM|IDEAL|MAT|MODULE|MODULEELEM): LIST of RING and TYPE

**Description**

This function returns the list of the RINGs (or types, if not dependent from a RING) on which the object E is dependent. Similar to “RingOf” (I-18.41 pg.239), this function also works on lists and returns the set of ring environments of all entries. ...needless to say that it may be quite slow on big inputs!

```plaintext
/**/ Use R := QQ[x,y,z];
/**/ L1 := [x, y];
/**/ L2 := [x, y, 0, 5/4];
/**/ Z3 := NewRingFp(3);
/**/ Use S := Z3[a,b];
/**/ RingsOf(L1);  
[RingDistrMPolyClean(QQ, 3)]
/**/ RingsOf(L2);
[RingDistrMPolyClean(QQ, 3), INT, RAT]
/**/ RingsOf([L2, a+b]);
[RingDistrMPolyClean(QQ, 3), INT, RAT, RingDistrMPolyClean(FFp(3), 2)]
```

See Also:  CurrentRing(I-3.49 pg.61), RingsOf(I-18.41 pg.239)
I-18.46  RingZZ

**syntax**

RingZZ(): RING

**Description**

This function returns the ring of integers. It is useful when you want to use “ZZ” (I-25.4 pg.292) inside “define/enddefine”.

NOTE: calling “RingZZ” twice gives the same identical ring, whereas calling “NewPolyRing” (I-14.8 pg.182) or “NewFractionField” (I-14.1 pg.179) return each time a new ring.

**example**

```/**/ Two := RingElem(RingZZ(), 2); Two; 2 /**/ type(Two); RINGELEM; /**/ IsZZ(RingOf(Two)); true /**/ IsQQ(RingOf(Two)); false
```

**See Also:** TopLevel(I-20.9 pg.274), RingQQt(I-18.43 pg.240), RingQQ(I-18.42 pg.240), ZZ(I-25.4 pg.292)

I-18.47  rk

**syntax**

rk(M: MAT): INT
rk(M: MODULE): INT

**Description**

This function computes the rank of “M”. For a module “M” this is defined as the vector space dimension of the subspace generated by the generators of “M” over the quotient field of the base ring – contrast this with the function “NumCompts” (I-14.31 pg.192) which simply counts the number of components the module has.

**example**

```/**/ Use R ::= QQ[x,y,z]; /**/ rk(IdentityMat(R, 4)); 4 rk(Module([x,y,z,0])); ---***WORK IN PROGRESS*** 1 rk(Module([[1,2,3],[2,4,6]])); ---***WORK IN PROGRESS*** 1 rk(Module([[1,2,3],[2,5,6]])); ---***WORK IN PROGRESS*** 2```

---

Chapter I-18.  R
### I-18.48  RMap [OBSOLESCENT]

**syntax**

```
[OBSOLESCENT] RMap(L: LIST): TAGGED("RMap")
```

**Description**

[OBSOLESCENT] related with “image [OBSOLESCENT]” (I-9.10 pg.119). In CoCoA-5 such homomorphisms are properly implemented as “PolyAlgebraHom” (I-16.14 pg.205).

### I-18.49  RootBound

**syntax**

```
RootBound(F: RINGELEM): INT
```

**Description**

This function computes a bound on the absolute values of the complex roots of a univariate polynomial over QQ. In some cases you may get a better bound by applying the transformation produced by “LinearSimplify”.

**example**

```
/**/ Use R ::= QQ[x,y,z];
/**/ RootBound(x^2-2);
4
```

**See Also:**  LinearSimplify(I-12.9 pg.159), RealRootRefine(I-18.17 pg.228), RealRoots(I-18.18 pg.228)

### I-18.50  round

**syntax**

```
round(X: RAT): INT
```

**Description**

This function rounds a rational to the nearest integer; halves are rounded towards zero.

**example**

```
/**/ round(4.56);
5
/**/ round(-1/2);
0
```

**See Also:**  num(I-14.29 pg.191), den(I-4.7 pg.67), floor(I-6.12 pg.88), ceil(I-3.6 pg.43)

### I-18.51  RowMat

**syntax**

```
RowMat(L: LIST): MAT
RowMat(R: RING, L: LIST): MAT
```
Description

This function returns the matrix whose only row consists of the elements of the list L.

```plaintext
/**/ RowMat([3,4,5]);
matrix([3, 4, 5])

/**/ RowMat(QQ,[5,6,7]);
matrix([5, 6, 7])
```

See Also: BlockMat(I-2.8 pg.37), DiagMat(I-4.15 pg.71), ColMat(I-3.26 pg.51)
Chapter I-19

S

I-19.1  saturate

Syntax

saturate(I: IDEAL, J: IDEAL): IDEAL

Description

This function returns the saturation of I with respect to J: the ideal of polynomials F such that F*G is in I for all G in J^d for some positive integer d.

The coefficient ring must be a field.

Example

/**/ Use R ::= QQ[x,y,z];
/**/ I := ideal(x-z, y-2*z);
/**/ J := ideal(x-2*z, y-z);
/**/ K := intersection(I, J); -- ideal of two points in the
    -- projective plane
/**/ L := intersection(K, ideal(x,y,z)^3); -- add an irrelevant component
/**/ HilbertFn(R/L);
H(0) = 1
H(1) = 3
H(2) = 6
H(t) = 2  for t >= 3

/**/ saturate(L, ideal(x,y,z)) = K; -- saturating gets rid of the
    -- irrelevant component
true

See Also:  colon(I-3.27 pg.52), HColon(I-8.1 pg.105), HSaturation(I-8.12 pg.111)

I-19.2  ScalarProduct

Syntax

ScalarProduct(L, M): OBJECT
  where each of L and M is of type MODULELEM or LIST

Description

This function returns the sum of the product of the components of L and M; precisely (len(L)=len(M));
ScalarProduct(L, M) = sum([ L[I]*M[I] — 1 In 1..len(L) ]).

The function works whenever the product of the components of L and M are defined (see “Algebraic Operators” (II-3.2 pg.301)).

```plaintext
/**/ ScalarProduct([1,2,3], [5,0,-1]); 2
```

Use R ::= QQ[x,y];
ScalarProduct([ideal(x,y), ideal(x^2-xy)],[x^2,y]);
ideal(x^3, x^2y, x^2y - xy^2)
-----------------------------

See Also: Algebraic Operators(II-3.2 pg.301)

I-19.3 ScientificStr

**syntax**

ScientificStr(X: INT|RAT|RINGELEM): STRING
ScientificStr(X: INT|RAT|RINGELEM, Prec: INT): STRING

**Description**

This function converts a rational number “X” into a (decimal) floating-point string. The optional second argument “Prec” says how many decimal digits to include in the mantissa; the default value is 5. Note that an exponent is always included.

```plaintext
/**/ ScientificStr(2/3); -- last printed digit is rounded
6.6667*10^(-1)

/**/ ScientificStr(7^-510); -- no arbitrary limit on exponent range
1.0000*10^-431

/**/ ScientificStr(1/81, 50); -- precision of mantissa specified by user
1.2345679012345679012345679012345679012345679012346*10^(-2)

/**/ ScientificStr(1/2); -- trailing zeroes are not suppressed
5.0000*10^(-1)
```

See Also: DecimalStr(I-4.3 pg.63), FloatStr(I-6.11 pg.87), FloatApprox(I-6.10 pg.87), MantissaAndExponent10(I-13.4 pg.168)

I-19.4 seed

**syntax**

Seed(N: INT): INT

**Description**

***** NOT YET IMPLEMENTED *****

This function seeds the random number generator, “random” (I-18.3 pg.222).

NOTE: every time you restart CoCoA the sequence of random numbers will be the same (as happens in many programming languages). If you want better randomness, see the example below.
I-19.5 SeparatorsOfPoints

See Also: random(I-18.3 pg.222)

I-19.5 SeparatorsOfPoints

SeparatorsOfPoints(Points: LIST): LIST

where Points is a list of lists of coefficients representing a set of "distinct" points in affine space.

Description

***** NOT YET IMPLEMENTED *****

This function computes separators for the points: that is, for each point a polynomial is determined whose value is 1 at that point and 0 at all the others. The separators yielded are reduced with respect to the reduced Groebner basis which would be found by "IdealOfPoints" (I-9.5 pg.116).

NOTE:
* the current ring must have at least as many indeterminates as the dimension of the space in which the points lie;
* the base field for the space in which the points lie is taken to be the coefficient ring, which should be a field;
* in the polynomials returned the first coordinate in the space is taken to correspond to the first indeterminate, the second to the second, and so on;
* the separators are in the same order as the points (i.e. the first separator is the one corresponding to the first point, and so on);
* if the number of points is large, say 100 or more, the returned value can be very large. To avoid possible problems when printing such values as a single item we recommend printing out the elements one at a time as in this example:

```plaintext
S := SeparatorsOfPoints(Pts);
```
Foreach Element In S Do
    PrintLn Element;
EndForeach;

For separators of points in projective space, see “SeparatorsOfProjectivePoints” (I-19.6 pg.248).

example

Use R ::= QQ[x,y];
Points := [[1, 2], [3, 4], [5, 6]];  
S := SeparatorsOfPoints(Points); -- compute the separators
S;
[1/8y^2 - 5/4y + 3, -1/4y^2 + 2y - 3, 1/8y^2 - 3/4y + 1]
----------------------------------
[[Eval(F, P) | P In Points] | F In S]; -- verify separators
[[1, 0, 0], [0, 1, 0], [0, 0, 1]]
----------------------------------

See Also: GenericPoints(I-7.6 pg.97), IdealAndSeparatorsOfPoints(I-9.3 pg.114), IdealAndSeparatorsOf-
ProjectivePoints(I-9.4 pg.115), IdealOfPoints(I-9.5 pg.116), IdealOfProjectivePoints(I-9.6 pg.117), Interpo-
late(I-9.27 pg.128), SeparatorsOfProjectivePoints(I-19.6 pg.248)

I-19.6 SeparatorsOfProjectivePoints

syntax

SeparatorsOfProjectivePoints(Points: LIST): LIST

where Points is a list of lists of coefficients representing a set of
‘‘{\it distinct}’’ points in projective space.

Description

***** NOT YET IMPLEMENTED *****

This function computes separators for the points: that is, for each point a homogeneous polynomial is deter-
mined whose value is non-zero at that point and zero at all the others. (Actually, choosing the values listed in
Points as representatives for the homogeneous coordinates of the corresponding points in projective space, the
non-zero value will be 1.) The separators yielded are reduced with respect to the reduced Groebner basis which
would be found by “IdealOfProjectivePoints” (I-9.6 pg.117).

NOTE:
* the current ring must have at least one more indeterminate than the dimension of the projective space in
which the points lie, i.e. at least as many indeterminates as the length of an element of the input, Points;
* the base field for the space in which the points lie is taken to be the coefficient ring, which should be a field;
* in the polynomials returned the first coordinate in the space is taken to correspond to the first indeterminate,
the second to the second, and so on;
* the separators are in the same order as the points (i.e. the first separator is the one corresponding the first
point, and so on);
* if the number of points is large, say 100 or more, the returned separator can be very large. To avoid possible
problems when printing such values as a single item we recommend printing out the elements one at a time as
in this example:

S := SeparatorsOfProjectivePoints(Pts);
Foreach Element In S Do
    PrintLn Element;
EndForeach;
For separators of points in affine space, see “SeparatorsOfPoints” (I-19.5 pg.247).

```plaintext
Use R ::= QQ[x,y,z];
Points := [[0,0,1],[1/2,1,1],[0,1,0]];
S := SeparatorsOfProjectivePoints(Points);
S;
[-2x + z, 2x, -2x + y]
-----------------------------
[[Eval(F, P) | P In Points] | F In S]; -- verify separators
[[1, 0, 0], [0, 1, 0], [0, 0, 1]]
-----------------------------
```

See Also: GenericPoints(I-7.6 pg.97), IdealAndSeparatorsOfPoints(I-9.3 pg.114), IdealAndSeparatorsOfProjectivePoints(I-9.4 pg.115), IdealOfPoints(I-9.5 pg.116), IdealOfProjectivePoints(I-9.6 pg.117), Interpolate(I-9.27 pg.128), SeparatorsOfPoints(I-19.5 pg.247)

### I-19.7 SetRow

**syntax**

SetRows(ref M: MAT, i: INT, L: LIST)

**Description**

This procedure sets the elements in “L” as entries of the “i”-th row in the matrix “M”; it returns nothing!

```plaintext
/**/ M := IdentityMat(QQ, 5);
/**/ SetRow(ref M, 1, [2,3,4,0,0]);
/**/ M;
matrix(QQ,
[2, 3, 4, 0, 0],
[0, 1, 0, 0, 0],
[0, 0, 1, 0, 0],
[0, 0, 0, 1, 0],
[0, 0, 0, 0, 1])
```

See Also: ref(I-18.23 pg.231), GetRow(I-7.15 pg.102), SwapRows(I-19.43 pg.266)

### I-19.8 SetStackSize

**syntax**

SetStackSize(NewSize: INT)

**Description**

Secret and dangerous.

### I-19.9 shape

**syntax**

shape(E: LIST): LIST (of TYPE)

shape(E: RECORD): RECORD (of TYPE)
shape(E:OTHER): TYPE

where OTHER stands for a type which is not LIST, MAT, or RECORD.

Description

This function returns the extended list of types involved in the expression E as outlined below:

\[
\text{type}(E) = \text{LIST}
\]

In this case, \( \text{Shape}(E) \) is the list whose \( i \)-th component is the type of the \( i \)-th component of \( E \).

\[
\text{type}(E) = \text{MAT}
\]

In this case, \( \text{Shape}(E) \) is a matrix with \((i,j)\)-th entry equal to the type of the \((i,j)\)-th entry of \( E \).

\[
\text{type}(E) = \text{RECORD}
\]

In this case, \( \text{Shape}(E) \) is a record whose fields are the types of the fields of \( E \).

Otherwise, “\( \text{Shape}(E) \)” is the type of \( E \).

Example

```plaintext
/**/ Use R ::= QQ[x];
/**/ L := [1,[1,"a"], x^2-x];
/**/ shape(L);
[Int, [INT, STRING], POLY]
/**/ R := record[name := "test", contents := L];
/**/ shape(R);
record[contents := [INT, [INT, STRING], POLY], name := STRING]
/**/ It.name;
STRING
```

There are undocumented functions, “IsSubShape” and “IsSubShapeOfSome”, for determining if the “\( \text{shape} \)” of a CoCoA expression is a “subshape” of another. To see the code for these functions, enter

```
Describe Function("$misc.IsSubShape");
Describe Function("$misc.IsSubShapeOfSome");
```

I-19.10 sign

\[
\text{sign}(X: \text{INT|RAT}): \text{INT}
\]

Description

This function returns -1 if \( X < 0 \), 0 if \( X = 0 \), and 1 if \( X > 0 \). \( X \) must be INT or RAT.

Example

```plaintext
/**/ sign(123);
1
```
I-19.11 SimplestRatBetween

syntax

SimplestRatBetween(A: RAT, B: RAT): RAT

Description

This function finds the simplest rational in the closed interval with end points “A” and “B”.

example

/**/ SimplestRatBetween(0.123, 0.456);
1/3
/**/ SimplestRatBetween(-3.14159, -2.71828);
-3

See Also: CFAprox(I-3.7 pg.43), FloatApprox(I-6.10 pg.87)

I-19.12 SimplexInfo

syntax

SimplexInfo(A: LIST): RECORD

Description

This function compute the Stanley-Reisner ideal, the Alexander Dual complex and ideal of a simplicial complex described by a list of top faces.

Package GeomModelling, by Elisa Palezzato.

example

/**/ Use QQ[x[1..5]], DegLex;
/**/ L := [x[1]*x[2]*x[3], x[2]*x[3]*x[4], x[3]*x[4]*x[5]]; -- list top faces
/**/ indent(SimplexInfo(L));
record[
  AlexanderDualCOMPLEX := [x[2]*x[3]*x[5], x[2]*x[3]*x[4], x[1]*x[3]*x[4]],
  AlexanderDualIdeal := ideal(x[4]*x[5], x[1]*x[5], x[1]*x[2]),
  Delta := [x[1]*x[2]*x[3], x[2]*x[3]*x[4], x[3]*x[4]*x[5]],
  StanleyReisnerIdeal := ideal(x[1]*x[4], x[1]*x[5], x[2]*x[5])
]

See Also: FVector(I-6.24 pg.93), SimplicialHomology(I-19.13 pg.251)

I-19.13 SimplicialHomology

syntax

SimplicialHomology(A: LIST): RECORD
SimplicialHomology(A: LIST, B: LIST): RECORD
Description

This function computes the simplicial homology of a simplicial complex described by a list of top faces. With 2nd argument only with the second list of vertices.

Package GeomModelling, by Elisa Palezzato.

```plaintext
/**/ Use QQ[x[1..5]], DegLex; --> DegLex ?
/**/ L := [x[1]*x[2]*x[3], x[2]*x[3]*x[4], x[3]*x[4]*x[5]]; -- list top faces
/**/ indent(SimplicialHomology(L));
record[
  H_0 := record[betti := 1, lambda := []],
  H_i := [record[betti := 0, lambda := []]],
  H_max := record[betti := 0, lambda := []]
]
-- 1 connected component (betti in H_0)
/**/ L := [x[1]*x[2]*x[3], x[2]*x[3]*x[4]]; -- list top faces
/**/ indent(SimplicialHomology(L)); --> Error: missing x[5]
/**/ L := [x[1]*x[2]*x[3], x[2]*x[3]*x[4], x[5]];
/**/ indent(SimplicialHomology(L));
record[
  H_0 := record[betti := 2, lambda := []],
  H_i := [record[betti := 0, lambda := []]],
  H_max := record[betti := 0, lambda := []]
]
-- 2 connected components
/**/ L := [x[1]*x[2]*x[3], x[2]*x[3]*x[4]];  
/**/ indent(SimplicialHomology(L, [x[1], x[2], x[3], x[4]]));
record[
  H_0 := record[betti := 1, lambda := []],
  H_i := [record[betti := 0, lambda := []]],
  H_max := record[betti := 0, lambda := []]
]```

I-19.14 size [OBSOLETE]

[OBSOLETE]

Description

[OBSOLETE] see “len” (I-12.5 pg.157).
See Also: len(I-12.5 pg.157)

I-19.15 skip

skip
**Description**

This command does nothing. I suppose it might be used to make the structure of a user-defined function more clear. It is probably at least as useful as the function “Tao”.

```plaintext
/**/ skip;
```

**I-19.16 SmoothFactor**

**Syntax**

`SmoothFactor(N: INT, MaxP: INT): RECORD`

**Description**

This function finds the small prime factors of an integer. It simply tries dividing by all primes up to the given bound “MaxP”. The result is a list of the prime factors found together with the unfactored part of N. Be careful about supplying large values for “MaxP” (e.g. greater than a million) as the function could take a very long time.

From version 5.0.4 the field are called “factors” and “multiplicities” instead of “Factors” and “Exponents” to comply with the naming conventions.

```plaintext
/**/ SmoothFactor(100,3);
record[factors := [2], multiplicities := [2], RemainingFactor := 25]

/**/ SmoothFactor(123456789,3700);
record[factors := [3, 3607], multiplicities := [2, 1], RemainingFactor := 3803]
```

See Also: IsPrime(I-9.56 pg.140), IsProbPrime(I-9.57 pg.140)

**I-19.17 sort**

**Syntax**

`sort(V: LIST)`

where `V` is a variable containing a list.

**Description**

This function sorts the elements of the list in V with respect to the default comparisons related to their types; it overwrites `V` and returns NULL.

For more on the default comparisons, see “Relational Operators” (II-3.3 pg.302) in the chapter on operators. For more complicated sorting, see “SortBy” (I-19.18 pg.254), “SortedBy” (I-19.20 pg.255).

```plaintext
/**/ L := [3,2,1];
/**/ sort(ref L); -- this returns nothing and modifies L
/**/ L;
[1, 2, 3]

/**/ Use R ::= QQ[x,y,z];
/**/ L := [x,y,z];
```
Chapter I-19. S

/**/ sort(ref L); -- this returns nothing and modifies L
/**/ L[1];
/**/ sorted([y, x, x^2]); -- this returns the sorted list
[y, x, x^2]

See Also: Relational Operators(II-3.3 pg.302), sorted(I-19.19 pg.254), SortBy(I-19.18 pg.254), SortedBy(I-19.20 pg.255)

I-19.18 SortBy

syntax

SortBy(L: LIST, CmpFn: FUNCTION)

Description

This function sorts the elements of the list in L with respect to the comparisons made by CmpFn; it overwrites L and returns NULL.

The comparison function CmpFn takes two arguments and returns True if the first argument is less than the second, otherwise it returns False. The sorted list is in increasing order.

Note that to call SortBy(L,CmpFn) inside a function you will need to make the name CmpFn accessible using TopLevel CmpFn;

Note that if both CmpFn(A, B) and CmpFn(B, A) return true, then A and B are viewed as being equal.

example

/**/ Define ByLength(S, T) -- define the sorting function
/**/ Return len(S) > len(T);
/**/ EndDefine;
/**/ M := ["bird","mouse","cat"];
/**/ SortBy(ref M, ByLength);
/**/ M;
["mouse", "bird", "cat"]

See Also: func(I-6.21 pg.92), sort(I-19.17 pg.253), sorted(I-19.19 pg.254), SortedBy(I-19.20 pg.255), TopLevel(I-20.9 pg.274)

I-19.19 sorted

syntax

sorted(L: LIST): LIST

Description

This function returns the list of the sorted elements of L without affecting L, itself.

For more on the default comparisons, see “Relational Operators” (II-3.3 pg.302) in the chapter on operators.

For more complicated sorting, see “SortBy” (I-19.18 pg.254), “SortedBy” (I-19.20 pg.255).

example

/**/ L := [3,2,1];
/**/ sorted(L);
/**/ Use R ::= QQ[x,y,z];
/**/ L := [x,y,z];
/**/ sorted(L);
[ z, y, x ]
/**/ sorted([y, x, z, x^2]);
[ z, y, x, x^2 ]
/**/ sorted([3, 1, 1, 2]);
[ 1, 1, 2, 3 ]
/**/ sorted(["b","c","a"]);
[ "a", "b", "c" ]

See Also: Relational Operators (II-3.3 pg.302), SortBy (I-19.18 pg.254), SortedBy (I-19.20 pg.255), sort (I-19.17 pg.253)

I-19.20 SortedBy

**/ Define ByLength(S, T) -- define the sorting function
/**/ Return len(S) > len(T);
/**/ EndDefine;
/**/ M := ["bird","mouse","cat"];
[ "mouse", "bird", "cat" ]
/**/ SortedBy(M, ByLength);
/**/ M; -- M is not changed
[ "bird", "mouse", "cat" ]
/**/ sorted(M); -- the function "Sort" sorts using the default ordering:
-- in this case, alphabetical order.
[ "cat", "bird", "mouse" ]
/**/ SortBy(ref M, ByLength); -- sort M in place, changing M
/**/ M;
[ "mouse", "bird", "cat" ]

See Also: func (I-6.21 pg.92), sort (I-19.17 pg.253), sorted (I-19.19 pg.254), SortBy (I-19.18 pg.254)
I-19.21 source

syntax

source S: STRING

Description

This command executes all CoCoA commands in the file or device named S. A typical use of “source” is to collect user-defined functions and variables in a text file, say, “MyFile.coc” and then execute:

source "MyFile.cocoa5";

or, equivalently, the obsolescent

<< "MyFile.cocoa5";

Functions and variables read in from a file in this way will erase functions and variables with identical names that may already exist. This can be avoided by using packages. Repeatedly used functions can be read into CoCoA at start-up by using “source” in the “userinit.coc” file.

See Also: Introduction to IO(I-6.1 pg.309), Introduction to Packages(I-7.1 pg.313)

I-19.22 SourceRegion

syntax

SourceRegion FromLine: INT, FromChar: INT To ToLine: INT, ToChar: INT In S: STRING

Description

This command executes all CoCoA commands in the specified region of the given file. It is not intended for manual use, but is used by the CoCoA UI.

SourceRegion FromLine, FromChar To ToLine, ToChar In "MyFile.cocoa5";

It is almost equivalent to copying the region into a temporary file, and then reading that file with the “source” command.

Line and char indexes start from 1; the region identified starts at the "from" line/character position and stops immediately before the "to" line/character position.

See Also: source(I-19.21 pg.256)

I-19.23 spaces

syntax

spaces(N: INT): STRING

Description

This function returns a string consisting of N spaces.

/**/ L := "a" + Spaces(5) + "b";
/**/ L;
a b

See Also: dashes(I-4.1 pg.63)
I-19.24 sprint

**syntax**

\[
sprint(E: \text{OBJECT}): \text{STRING}
\]

**Description**

This function takes any CoCoA expression and converts its value to a string. One use is to check for extremely long output before printing in a CoCoA window.

**example**

```cocoa
/**/ Use R ::= QQ[x,y];
/**/ I := ideal(x,y);
/**/ J := sprint(I);
/**/ I;
ideal(x, y)
/**/ J;  -- The output for I and J looks the same, but ...
ideal(x, y)
/**/ type(I);  -- I is an ideal, and
IDEAL
/**/ type(J);  -- J is just the string "ideal(x, y)".
STRING
/**/ J[1];  -- the 1st character of J
i
/**/ J[2];  -- the 2nd character of J
d
/**/ len(J);  -- J has 11 characters
11
```

**See Also:** Introduction to IO(I-6.1 pg.309), IO.SprintTrunc(I-9.33 pg.131), print(I-16.25 pg.211), println(I-16.29 pg.213)

I-19.25 SqFreeFactor

**syntax**

\[
\text{SqFreeFactor}(F: \text{RINGELEM}): \text{RECORD}
\]

**Description**

Compute a factorization (of a polynomial) into coprime squarefree factors. The factorization may sometimes be finer than necessary, i.e. two factors could have the same multiplicity.

**example**

```cocoa
/**/ Use R ::= QQ[x,y];
/**/ f := (x^2-1)^2*(y+2)^3;
/**/ indent(SqFreeFactor(f));
record[
  RemainingFactor := 1,
  factors := [x^2 -1, y +2],
]```
multiplicities := [2, 3]

See Also: factor(I-6.1 pg.83), ContentFreeFactor(I-3.42 pg.58)

I-19.26 StableBBasis5

\[
\text{StableBBasis5(Pts: LIST, Toler: LIST): RECORD}
\]

\[
\text{StableBBasis5(Pts: LIST, Toler: LIST, Gamma: RAT): RECORD}
\]

Description

***** NOT YET IMPLEMENTED ***** See “TmpNBM” (I-20.8 pg.274)

This function returns a record containing a “stable order ideal” of the ideal of points, and a list of “almost vanishing” polynomials. If the cardinality of the order ideal is equal to the number of points, it is in fact a “quotient basis”, and in this case a “stable border basis” founded on it is also returned. The boolean field “StableBBasisFound” is set to true if a stable border basis was found, otherwise false.

The first argument is a list of points in k-dimensional space, and the second argument is list of k positive tolerances (one for each dimension). The function builds the stable order ideal stepwise by testing, from a numerical point of view, the linear dependence of a set of vectors. So that the answer can be represented, the current ring must have at least k indeterminates; the term ordering is ignored as it plays no role in determining the border basis.

There is a third, optional argument: it is a real non negative number “Gamma” which is used for scaling the threshold on the admissible perturbation of the points. A value of “Gamma” \(\geq 1\) should be used. If no value is specified then by default “Gamma” \(= 0.1\).


example

\[
\begin{align*}
\text{Pts} & := [[0.1,-1],[1,1],[2,3]]; \\
\text{Toler} & := [0.1,0.1]; \\
\text{StableBBasis5(Pts, Toler);}
\end{align*}
\]

\[
\begin{align*}
\text{record[} & \\
\text{AlmostVanishing} & := [ (...) ], \\
\text{BBasis} & := [ \\
& -3602879701896397/288230376151711744y^2 + x - \\
& 32425917317067571/72057594037927936y - \\
& 154923827181545063/288230376151711744, \\
& xy - 14051230373959475/288230376151711744y^2 - \\
& 39631676720860365/72057594037927936y + \\
& 1080869105689191/288230376151711744, \\
& y^3 - 3y^2 - y + 3, \\
& xy^2 - 580063632005319885/288230376151711744y^2 - \\
& 32425917317067571/72057594037927936y + \\
& 421536925121878425/288230376151711744], \\
\text{SOI} & := [1, y, y^2], \\
\text{StableBBasisFound} & := \text{True}\]
\end{align*}
\]

-------------------------------

\[
\begin{align*}
\text{Toler} & := [0.6, 0.6]: \\
\text{StableBBasis5(Pts, Toler);}
\end{align*}
\]

\[
\begin{align*}
\text{record[AlmostVanishing} & := [.....], \\
\text{SOI} & := [1, y], \\
\text{StableBBasisFound} & := \text{False}\]
\end{align*}
\]

-------------------------------
I-19.27 StableIdeal

**syntax**

StableIdeal(L: LIST of power-products): IDEAL

**Description**

This function returns the smallest stable ideal containing the power-products in “L” (see also “StronglyStableIdeal” (I-19.32 pg.261)).

**example**

```cocoa
/**/ Use R ::= QQ[x,y,z];
/**/ L := [x*z^4, y^3];
/**/ StableIdeal(L);
ideal(x^2*z^3, x*y*z^3, x*z^4, x^3, x^2*y, x*y^2, y^3)
```

**See Also:** IsStable(I-9.62 pg.142), LexSegmentIdeal(I-12.7 pg.158), StronglyStableIdeal(I-19.32 pg.261)

I-19.28 StarPrint, StarSprint

**syntax**

StarPrint(F: RINGELEM)
StarPrintFold(F: RINGELEM, LineWidth: INT)
StarSprint(F: RINGELEM): STRING
StarPrintFold(F: RINGELEM, LineWidth: INT): STRING

**Description**

These functions print the polynomial F with asterisks added to denote multiplications. They may be useful when transferring polynomials or rational functions from CoCoA to other mathematical software (e.g. Gap, Maple, Macaulay, Singular,..). “StarPrint” inserts newline characters (only between terms) with the aim of avoiding lines longer than 70 characters; the second argument to “StarPrintFold” is for specifying a different width limit; a non positive value is treated as meaning no limit. The “StarSprint” functions print the value into a string.

**example**

```cocoa
Use R ::= QQ[x,y];
F := x^3+2xy-y^2;
StarPrint(F);
1*x^3 +2*x*y -1*y^2
---------------------
StarPrintFold(F,1); -- this will print one term per line
1*x^3
+2*x*y
-1*y^2
---------------------
D := OpenOFile("example");
Print StarSprint(F) On D; -- this will print F into the file "example"
Close(D);
---------------------
```

**See Also:** LaTeX(I-12.2 pg.155)
I-19.29  **starting**

**syntax**

starting(S: STRING): LIST of RECORD

**Description**

This function returns a list of all CoCoA functions starting with the string “S”. In general, this list will include undocumented commands. For these, one may find some information using “Describe Function("Fn_Name")” or “Describe Function("$PackageName.Fn_Name")”.

```c
/**/ indent(starting("Su"));
[ record[IsExported := true, name := "$BackwardCompatible.Subsets"],
  record[IsExported := true, name := "$BackwardCompatible.Subst"],
  record[IsExported := true, name := "$BackwardCompatible.Sum"],
  record[IsExported := true, name := "$BackwardCompatible.Support"]
]
```

I-19.30  **StdDegLexMat**

**syntax**

StdDegLexMat(N: INT): MAT

**Description**

This function returns the matrix defining a standard term-ordering.

```c
/**/ StdDegLexMat(3);
matrix(QQ,
  [[1, 1, 1],
   [1, 0, 0],
   [0, 1, 0]])
```

**See Also:** OrdMat(I-15.10 pg.199), Orderings(III-9.5 pg.354), StdDegRevLexMat(I-19.31 pg.260), LexMat(I-12.6 pg.157), RevLexMat(I-18.37 pg.237), XelMat(I-24.1 pg.289)

I-19.31  **StdDegRevLexMat**

**syntax**

StdDegRevLexMat(N: INT): MAT

**Description**

This function returns the matrix defining a standard term-ordering.

```c
/**/ StdDegRevLexMat(3);
matrix(QQ,
  [[1, 1, 1],
   [0, 0, -1],
   [0, -1, 0]])
```
I-19.32. StronglyStableIdeal

**syntax**

\[\text{StronglyStableIdeal}(L: \text{LIST of power-products}): \text{IDEAL}\]

**Description**

This function returns the smallest strongly stable ideal containing the power-products in L.

```
/**/ Use R := QQ[x,y,z];
/**/ L := [x*y^2*z];
/**/ StableIdeal(L);
ideal(x^4, x^3*y, x^2*y^2, x*y^3, x*y^2*z)
/**/ StronglyStableIdeal(L);
ideal(x^4, x^3*y, x^2*y^2, x*y^3, x^3*z, x^2*y*z, x*y^2*z)
```

**See Also:** IsStronglyStable(I-9.64 pg.143), LexSegmentIdeal(I-12.7 pg.158), StableIdeal(I-19.27 pg.259)

I-19.33. SubalgebraMap [OBSOLETE]

**syntax**

[OBSOLETE]

**Description**


**See Also:** SubalgebraRepr(I-19.34 pg.261)

I-19.34. SubalgebraRepr

**syntax**

\[\text{SubalgebraRepr}(F: \text{RINGELEM}, L: \text{LIST}): \text{RECORD}\]
\[\text{SubalgebraRepr}(R: \text{RING}, F: \text{RINGELEM}, L: \text{LIST}): \text{RECORD}\]

**Description**

This function returns the representation of a polynomial as a subalgebra element in terms of the subalgebra generators.

```
/**/ Use QQ[s,t];
/**/ L := ***[s^3, s^2*t, st^2, t^3]***;
/**/ indent(SubalgebraRepr(s^6*t^6, L));
record[
  IsInImage := true,
  OnePreImage := x[1]^2*x[4]^2,
```

See Also: PreImage(I-16.17 pg.207), ker(I-11.1 pg.153)

I-19.35 submat

\[ \text{submat}(M: \text{MAT}, R: \text{LIST of INT}, C: \text{LIST of INT}): \text{MAT} \]

Description

This function returns the submatrix of \( M \) formed by the rows listed in \( R \) and the columns listed in \( C \). If \( M \) is a list, it is interpreted as a matrix in the natural way.

\[
/**/ M := mat([[1,2,3,4,5],[6,7,8,9,10],[11,12,13,14,15]]);
/**/ submat(M,[1,3],3..5);
matrix([[3, 4, 5],
        [13, 14, 15]])
/**/ M := mat([[1,2,3],[4,5,6]]);
/**/ submat(M,[2],[1,3]);
matrix([[4, 6]])
\]

See Also: minors(I-13.17 pg.174)

I-19.36 submodule

\[ \text{submodule}(L: \text{LIST of MODULEELEM}): \text{MODULE} \]
\[ \text{submodule}(F: \text{MODULE}, L: \text{LIST of MODULEELEM}): \text{MODULE} \]

Description

The first form returns the ideal generated by \( L \). The second is the same as the first but works also if \( L = [] \).

This function is not friendly if you write the input by hand: we suggest “SubmoduleCols, SubmoduleRows” (I-19.37 pg.263) for creating a module from the rows or columns of a matrix.

NOTE: the second argument is a LIST of MODULEELEM, not a LIST of LISTS of RINGELEM.

\[
/**/ Use R ::= QQ[x,y,z];
/**/ R3 := NewFreeModule(R, 3);
/**/ L := [ModuleElem(R3, [x,y,z]), ModuleElem(R3, [x-1,0,z])];
/**/ M := submodule(R3, L); -- equivalent to
/**/ M := submodule(L); -- (L not empty)
/**/ gens(M);
[[x, y, z], [x -1, 0, z]]
\]

See Also: ModuleOf(I-13.24 pg.176), SubmoduleCols, SubmoduleRows(I-19.37 pg.263), GensAsCols, GensAsRows(I-7.9 pg.99), gens(I-7.8 pg.98)
### I-19.37 SubmoduleCols, SubmoduleRows

**syntax**

```plaintext
SubmoduleCols(F: MODULE, M: MATRIX): MODULE
SubmoduleRows(F: MODULE, M: MATRIX): MODULE
```

**Description**

The first (second) function returns the submodule of $F$ generated by the module elements described by the columns (rows) in the matrix $M$ (which might be empty). Dimensions must be compatible.

```plaintext
/**/ R3 := NewFreeModule(R, 3);
/**/ MGens := matrix(R, [[x, y, z], [x -1, 0, z]]);

/**/ M := SubmoduleRows(R3, MGens);
/**/ gens(M);
[[x, y, z], [x -1, 0, z]]

-- /**/ M := SubmoduleCols(R3, MGens); -- !!! ERROR: wrong length !!!

/**/ M := SubmoduleCols(NewFreeModule(R, 2), MGens);
/**/ gens(M);
[[x, x -1], [y, 0], [z, z]]
```

**See Also:** GensAsCols, GensAsRows(I-7.9 pg.99), submodule(I-19.36 pg.262), ModuleElem(I-13.23 pg.176)

### I-19.38 subsets

**syntax**

```plaintext
subsets(S: LIST): LIST
subsets(S: LIST, N: INT): LIST
```

**Description**

This function computes all sublists (subsets) of a list (set). If $N$ is specified, it computes all sublists of cardinality $N$.

```plaintext
/**/ subsets([1, 4, 7]);
[[ ], [7], [4], [4, 7], [1], [1, 7], [1, 4], [1, 4, 7]]

/**/ subsets([1, 4, 7], 2);
[[1, 4], [1, 7], [4, 7]]

/**/ subsets([2,3,3]);        -- list with repeated entries
[[ ], [3], [3], [3, 3], [2], [2, 3], [2, 3], [2, 3, 3]]

/**/ subsets(MakeSet([2,3,3]));
[[ ], [3], [2], [2, 3]]
```

**See Also:** IsSubset(I-9.65 pg.143), partitions(I-16.4 pg.202), permutations(I-16.5 pg.202), MakeSet(I-13.3 pg.168), tuples(I-20.14 pg.278)
I-19.39 subst

**Syntax**

\[
\begin{align*}
\text{subst}(E: \text{OBJECT}, X, F): \text{OBJECT} \\
\text{subst}(E: \text{OBJECT}, [[X_1, F_1], \ldots, [X_r, F_r]]): \text{OBJECT}
\end{align*}
\]

where each \( X \) or \( X_i \) is an indeterminate
and each \( F \) or \( F_i \) is a RINGELEM

**Description**

The first form of this function substitutes \( "F_i" \) for \( "X_i" \) in the expression \( E \). The second form is a shorthand for the first in the case of a single indeterminate. When substituting for the indeterminates in order, it is easier to use \( \text{"eval"} \) (I-5.10 pg.79).

**Example**

```plaintext
/**/ Use R ::= QQ[x,y,z,t];
/**/ F := x +y +z +t^2;
/**/ subst(F, x, -2);
t^2 +y +z -2
/**/ subst(F, [[x,x^2], [y,y^3], [z,t^5]]);
t^5 +y^3 +x^2 +t^2
/**/ eval(F, [x^2,y^3,t^5]); -- the same thing as above
t^5 +y^3 +x^2 +t^2
/**/ MySubst := [[y,1], [t, 3*z-x]];
/**/ subst(x*y*z*t, MySubst); -- substitute into the function x*y*z*t
-x^2*z +3*x*z^2
```

**See Also:** eval(I-5.10 pg.79), Evaluation of Polynomials(III-11.2 pg.362), PolyAlgebraHom(I-16.14 pg.205), QZP(I-17.4 pg.218), RingElem(I-18.38 pg.238), ZPQ(I-25.3 pg.292)

I-19.40 sum

**Syntax**

\[
\begin{align*}
\text{sum}(L: \text{LIST}): \text{OBJECT} \\
\text{sum}(L: \text{LIST}, \text{InitVal}: \text{OBJECT}): \text{OBJECT}
\end{align*}
\]

**Description**

This function returns the sum of the objects in the list \( "L" \) (together with \( "\text{InitVal}" \), if specified). When writing a program, if the list \( "L" \) may be empty, you must specify \( "\text{InitVal}" \).

**Example**

```plaintext
/**/ use R ::= QQ[x,y];
/**/ sum([3, x, y^2]);
y^2 +x +3
/**/ sum([1..40] = binomial(41,2));
true
/**/ sum(["c","oc","oa"]);
cocoa
/**/ sum([]); -- gives 0 of type INT
```

See Also: eval(I-5.10 pg.79), Evaluation of Polynomials(III-11.2 pg.362), PolyAlgebraHom(I-16.14 pg.205), QZP(I-17.4 pg.218), RingElem(I-18.38 pg.238), ZPQ(I-25.3 pg.292)
I-19.41. support

syntax

support(F: RINGELEM): LIST
support(F: MODULEELEM): LIST

Description

This function returns the list of terms of F. To get a list of monomials, which includes coefficients, use “monomials” (I-13.26 pg.177).

example

/**/ Use R := QQ[x,y];
/**/ F := 3*x^2-4*x*y+y^3+3;
/**/ support(F);
[y^-3, x^-2, x*y, 1]

/**/ monomials(F);
[y^-3, 3*x^2, -4*x*y, 3]

// NOT YET IMPLEMENTED for MODULEELEM

See Also: coefficients(I-3.21 pg.48), monomials(I-13.26 pg.177)

I-19.42 swap

syntax

swap(ref A: OBJECT, ref B: OBJECT)

Description

This procedure swaps two values; it returns nothing!

example

/**/ A := 1;
/**/ B := 2;
/**/ swap(ref A, ref B);
/**/ PrintLn [A,B];
[2, 1]

See Also: ref(I-18.23 pg.231)
### I-19.43 SwapRows

**Syntax**

```plaintext
SwapRows(ref M: MAT, i: INT, j: INT)
```

**Description**

This procedure swaps the “i”-th and “j”-th rows in the matrix “M”; it returns nothing!

**Example**

```plaintext
/**/ M := IdentityMat(QQ, 5);
/**/ SwapRows(ref M, 2, 5);
/**/ M;
matrix(QQ,
[
[1, 0, 0, 0, 0],
[0, 0, 0, 0, 1],
[0, 0, 1, 0, 0],
[0, 0, 0, 1, 0],
[0, 1, 0, 0, 0]
])
```

**See Also:** ref(I-18.23 pg.231), swap(I-19.42 pg.265), GetRow(I-7.15 pg.102), SetRow(I-19.7 pg.249)

### I-19.44 sylvester

**Syntax**

```plaintext
sylvester(F: RINGELEM, G: RINGELEM, X: RINGELEM): MAT
```

**Description**

(sorry Sylvester for the lower-case: here we follow the naming convention “single name goes lower-case”)

This function returns the Sylvester matrix of the polynomials F and G with respect to the indeterminate X. This is the matrix used to calculate the resultant.

**Example**

```plaintext
/**/ Use R ::= QQ[p,q,x];
/**/ F := x^3+p*x-q; G := deriv(F, x);
/**/ sylvester(F, G, x);
matrix([1, 0, p, -q, 0],
[0, 1, 0, p, -q],
[3, 0, p, 0, 0],
[0, 3, 0, p, 0],
[0, 0, 3, 0, p])
/**/ det(sylvester(F, G, x)) = resultant(F, G, x);
true
```

**See Also:** resultant(I-18.34 pg.236)
I-19.45 SymbolRange

**syntax**

SymbolRange(H: STRING, LO: INT, HI: INT): LIST of RINGELEM
SymbolRange(H: STRING, LO: LIST of INT, HI: LIST of INT): LIST of RINGELEM

**Description**

This function returns the list of the symbols with a given head and a range of indices. A symbol is a record with head (as “IndetName” (I-9.20 pg.124)) and indices (as “IndetSubscripts” (I-9.22 pg.126))

/**/ indent(SymbolRange("x", 3, 5));
[ record[head := "x", indices := [3]],
  record[head := "x", indices := [4]],
  record[head := "x", indices := [5]]
]
/**/ P := NewPolyRing(QQ, SymbolRange("x", 0,2));
/**/ indets(P);
[x[0], x[1], x[2]]
/**/ indent(SymbolRange("x", [3,1], [5,2]));
[ record[head := "x", indices := [[3, 1]]],
  record[head := "x", indices := [[3, 2]]],
  record[head := "x", indices := [[4, 1]]],
  record[head := "x", indices := [[4, 2]]],
  record[head := "x", indices := [[5, 1]]],
  record[head := "x", indices := [[5, 2]]]
]

**See Also:** indet(I-9.18 pg.123), IndetSubscripts(I-9.22 pg.126), IndetIndex(I-9.19 pg.124), IndetName(I-9.20 pg.124), NumIndets(I-14.32 pg.192), SymbolRange(I-19.45 pg.267)

I-19.46 syz

**syntax**

Syz(L: LIST of RINGELEM): MODULE
Syz(M: IDEAL|MODULE, Index: INT): MODULE

**Description**

In the first two forms this function computes the syzygy module of a list of polynomials or module elements. “SyzOfGens(I)” is the same as “Syz(gens(I))”.

In the last form this function returns the specified syzygy module of the minimal free resolution of M which must be homogeneous. As a side effect, it computes the Groebner basis of M. (**NOT YET IMPLEMENTED****)

The coefficient ring must be a field.

/**/ Use R ::= QQ[x,y,z];
/**/ indent(Syz([x^2-y-1, y^3-z, x^2-y, y^3-x]));
SubmoduleRows(F, matrix(
  [y^3 -z, 0, 0, -x^2 +y +1],
...
[0, 1, 0, -1],
[x^2 -y, 0, -x^2 +y +1, 0],
[0, 0, y^3 -z, -x^2 +y]

/**/ I := ideal(x, x, y);
/**/ syz(gens(I));
submodule(FreeModule(..), [[1, -1, 0], [0, y, -x]])
/**/ SyzOfGens(I);
submodule(FreeModule(..), [[1, -1, 0], [0, y, -x]])

Syz(I, 1); -- NOT YET IMPLEMENTED
Module([[x, -y]])

-------------------------------
I := ideal(x^2-yz, xy-z^2, xyz); -- NOT YET IMPLEMENTED
Syz(I,0);
Module([x^2 - yz], [xy - z^2], [xyz])

-------------------------------
Syz(I,1); -- NOT YET IMPLEMENTED
Module([x^2 + yz, xy - z^2, 0], [xz^2, -yz^2, -y^2 + xz], [z^3, 0,
-xy + z^2], [0, z^3, -x^2 + yz])

-------------------------------
Syz(I,2);
Module([0, z, -x, y], [-z^2, -x, y, -z])

-------------------------------
Syz(I,3);
Module([[]])

-------------------------------
Res(I);
0 --> R(-6)^2 --> R(-4)(+)R(-5)^3 --> R(-2)^2(+)R(-3)

-------------------------------
See Also: res(I-18.31 pg.235), SyzOfGens(I-19.47 pg.268)

I-19.47 SyzOfGens

SyzOfGens(M: IDEAL|MODULE): MODULE

Description
If M is an ideal or submodule, this function calculates the syzygy module for the given set of generators of M.
If M is a quotient of a ring by an ideal I or a quotient of a free module by a submodule N, then this function calculates the syzygy module for the given set of generators of I or N, respectively.
“SyzOfGens(I)” is the same as “Syz(gens(I))”.
The coefficient ring must be a field.

Example
/**/ Use R := QQ[x,y,z];
/**/ I := ideal(x, y, x+y);
/**/ indent(SyzOfGens(I));
SubmoduleRows(F, matrix([ [1, 1, -1], [0, x +y, -y ]]))
/**/ R3 := NewFreeModule(R, 3);
/**/ MGens := matrix(R,[[x,y,z], [x-y,0,z], [y^2,y^2,0]]);
/**/ indent(SyzOfGens(SubmoduleRows(R3, MGens)));
SubmoduleRows(F, matrix([ [1, -1, -1] ]))

See Also: syz(I-19.46 pg.267)
Chapter I-20

I-20.1  tag

<table>
<thead>
<tr>
<th>syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tag(E: OBJECT): STRING</code></td>
</tr>
</tbody>
</table>

Description

If E is a tagged object, this function returns the tag of E; otherwise, it returns the empty string.

<table>
<thead>
<tr>
<th>example</th>
</tr>
</thead>
</table>
| ```
/**/ L := tagged(3,"MyTag");
/**/ type(L);
TAGGED("$TopLevel.MyTag")
/**/ tag(L);
MyTag
``` |

See Also: Printing a Tagged Object(III-16.2 pg.379), tagged(I-20.2 pg.271), untagged(I-21.4 pg.280)

I-20.2  tagged

<table>
<thead>
<tr>
<th>syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tagged(E: OBJECT, S: STRING): TAGGED(S)</code></td>
</tr>
</tbody>
</table>

Description

This first function returns the object E, tagged with the string S. Tagging is used for pretty printing of objects. See the reference listed below.

<table>
<thead>
<tr>
<th>example</th>
</tr>
</thead>
</table>
| ```
/**/ L := [1,2,3];
/**/ M := tagged(L,"MyTag");
/**/ type(L);
LIST
/**/ type(M);
TAGGED("$TopLevel.MyTag")
``` |
### I-20.3 tail

<table>
<thead>
<tr>
<th>syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>tail(L: LIST): LIST</td>
</tr>
</tbody>
</table>

**Description**

This function returns the list obtained from L by removing its first element. It cannot be applied to the empty list.

```/**/ tail([1,2,3]);
[2, 3]
```

**See Also:** first(I-6.6 pg.85), last(I-12.1 pg.155)

### I-20.4 TensorMat

<table>
<thead>
<tr>
<th>syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>TensorMat(M: MATRIX, N: MATRIX): MAT</td>
</tr>
</tbody>
</table>

**Description**

This function returns the tensor product of two matrices.

```/**/ Use R ::= QQ[x,y,z,w];
/**/ TensorMat(mat(R, [[1,-1],[2,-2],[3,-3]]), mat(R, [[x,y],[z,w]]));
matrix([x, y, -x, -y], [z, w, -z, -w],
[2*x, 2*y, -2*x, -2*y],
[2*z, 2*w, -2*z, -2*w],
[3*x, 3*y, -3*x, -3*y],
[3*z, 3*w, -3*z, -3*w])
```

### I-20.5 TgCone

<table>
<thead>
<tr>
<th>syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>TgCone(I: IDEAL): IDEAL</td>
</tr>
</tbody>
</table>

**Description**

The “initial form” of a polynomial F is the homogeneous component of F of the lowest degree (in contrast with the “leading form”, see “LF” (I-12.8 pg.158), “DF” (I-4.14 pg.70)). The “initial ideal” of the ideal “I” is the
ideal generated by the initial forms of all polynomials in "I". It is also called "tangent cone" (which strictly is the variety defined by the initial ideal).

The implementation is based on Lazard’s method (see Kreuzer-Robbiano, Commutative Computer Algebra II, pg.463).

```
/**/ Use R := QQ[x,y,z];
/**/ TgCone(ideal(x^3-y));
ideal(-y)
/**/ TgCone(ideal(x^3+x^2-y^2));
ideal(x^2 -y^2)
/**/ I := ideal(x^3-y*z, y^2-x*z, z^2-x^2*y);
/**/ TgCone(I); -- same as InitialIdeal(I, [x,y,z]);
ideal(y^2 -x*z, z^2, -y*z)
```

See Also: InitialIdeal(I-9.25 pg.127), PrimaryHilbertSeries(I-16.22 pg.209)

### I-20.6 TimeFrom

#### Syntax

`TimeFrom(StartPoint: RAT): STRING`

#### Description

This function returns a string indicating the number of CPU seconds consumed since “StartPoint”; the value in “StartPoint” should be the value produced by the function “CpuTime” (I-3.47 pg.60) at the point where timing should commence.

```
/**/ t0 := CpuTime();
/**/ N := factorial(1000000);
/**/ PrintLn "Time to compute N: ",TimeFrom(t0);
Time to compute N: 7.538
```

### I-20.7 TimeOfDay

#### Syntax

`TimeOfDay(): INT`

#### Description

This function returns the current time as an INT in the form HHMMSS. Note that from version 5.0.4 this information is no longer given by the function “date” (I-4.2 pg.63).

```
/**/ date(); -- 2013-05-30
20130530
/**/ TimeOfDay(); -- 09:08:13
90813
```

See Also: date(I-4.2 pg.63)


I-20.8 TmpNBM

**syntax**

```
```

**Description**

Thanks to John Abbott and Maria-Laura Torrente.

This function checks that the current ring is suitable: see below for details.

This function returns a record containing a factor-closed set of power-products “QuotientBasis” and a list of “almost vanishing” polynomials. If the cardinality of the “QuotientBasis” is equal to the number of points, it is in fact a “quotient basis” of the ideal of points, and in this case a “border basis” founded on it is also returned.

The first argument is a list of points in k-dimensional space, and the second argument is list of k positive tolerances (one for each dimension). So that the answer can be represented, the current ring must have at least k indeterminates; the term ordering is ignored as it plays no role in determining the border basis.


**example**

```
/**/ P ::= QQ[x,y];
/**/ Eps := [0.1, 0.1];
/**/ Points := [[10, 0], [-10, 0], [0, 10], [0, -10], [7, 7], [-7, -7]];
/**/ indent(TmpNBM(P, mat(Points), RowMat(Eps)));
record[
  AlmostVanishing := [x^2 +(2/49)*x*y +y^2 -100, x*y^2 +(49/51)*y^3 +(-4900/51)*y, y^4 +51*x*y -100*y^2],
  BBasis := [x^2 +(2/49)*x*y +y^2 -100, x*y^2 +(49/51)*y^3 +(-4900/51)*y, x^2*y +(49/51)*y^3 +(-4900/51)*y, y^4 +51*x*y -100*y^2, x*y^3 -49*x*y],
  QuotientBasis := [1, y, x, y^2, x*y, y^3],
  StableBBasisFound := true
]
```

See Also: IdealOfPoints(I-9.5 pg.116), StableBBasis5(I-19.26 pg.258)

I-20.9 TopLevel

**syntax**

```
TopLevel X;
where ‘‘\verb&X&’’ is the name of a top level variable or function.
```

**Description**

This command makes a top-level variable accessible from inside a function. It is useful for making “QQ” (I-17.1 pg.217) and “ZZ” (I-25.4 pg.292) visible, and also if a top-level function is to be passed as a parameter (e.g. to the function “SortBy” (I-19.18 pg.254)).

The command may be used with any top-level variable, but it is poor style to use it for purposes other than those mentioned above.

NOTE: Package variables should be accessed directly (via the fully qualified name); the “TopLevel” command does not recognise them.
I-20.10  TopLevelFunctions

**/ define BeautifulRing(N)
/**/ TopLevel QQ;
/**/ R ::= QQ[b[1..N]];
/**/ return R;
/**/ enddefine;
/**/ Define CompareLen(X,Y) Return len(X) < len(Y); EndDefine;
/**/ Define LongestName(ListOfNameAndValue)
/**/ TopLevel CompareLen; --> to pass it as paremeter to SortBy
/**/ names := [entry[1] | entry in ListOfNameAndValue];
/**/ SortBy(ref names, CompareLen);
/**/ Return last(names);
/**/ EndDefine;
/**/ L := [["ABC",1],["XYZT",2]];
/**/ LongestName(L);
XYZT

See Also: func(I-6.21 pg.92), ImportByRef, ImportByValue(I-9.14 pg.121)

I-20.10  TopLevelFunctions

Syntax

TopLevelFunctions(): LIST of FUNCTION

Description

This function returns the list of all functions available at top-level

```
/**/ indent(TopLevelFunctions());
[
  record[IsExported := true, Name := "$BackwardCompatible.Abs"],
  record[IsExported := true, Name := "$BackwardCompatible.Append"],
  record[IsExported := true, Name := "$BackwardCompatible.Ascii"],
  ...
```

I-20.11  toric

Syntax

toric(I: IDEAL): IDEAL
toric(I: IDEAL, L: LIST of INDETS): IDEAL
toric(M: MAT|LIST of LIST): IDEAL

Description

These functions return the saturation of an ideal, I, generated by binomials. In the first two cases, I is the ideal generated by the binomials in L. To describe the ideal in the last case, let K be the integral elements in the kernel of M. For each k in K, we can write k = k(+) - k(-) where the i-th component of k(+) is the i-th component of k, if positive, otherwise zero. Then I is the ideal generated by the binomials “x^i k(+) - x^i k(-)” as k ranges over K.
NOTE: successive calls to this last form of the function may produce different generators for the saturation.

The first and third functions return the saturation of $I$. For the second function, if the saturation of $I$ with respect to the variables in $X$ happens to equal the saturation of $I$, then the saturation of $I$ is returned. Otherwise, an ideal “containing” the saturation with respect to the given variables is returned. The point is that if one knows, a priori, that the saturation of $I$ can be obtained by saturating with respect to a subset of the variables, the second function may be used to save time.

For more details, see the article: A.M. Bigatti, R. La Scala, L. Robbiano, “Computing Toric Ideals,” Preprint (1998). The article describes three different algorithms; the one implemented in CoCoA is “EATI”. The first two examples below are motivated by B. Sturmfels, “Groebner Bases and Convex Polytopes,” Chapter 6, p. 51. They count the number of homogeneous primitive partition identities of degrees 8 and 9.

```coconut
/**/ Use QQ[x[1..8],y[1..8]];  
/**/ HPPI8 := [x[I]^I*x[I+2]*y[I+1] -y[I]^I*y[I+2]*x[I+2]*(I+1) | I In 1..6];  
/**/ BL := toric(ideal(HPPI8), [x[1],y[2]]);  
/**/ len(gens(BL));  
340  
/**/ Use QQ[x[1..9],y[1..9]];  
/**/ HPPI9 := [x[I]^I*x[I+2]*y[I+1] -y[I]^I*y[I+2]*x[I+2]*(I+1) | I In 1..7];  
/**/ BL := toric(ideal(HPPI9), [x[1],y[2]]);  
/**/ len(gens(BL));  
798  
/**/ Use R := QQ[x,y,z,w];  
/**/ toric(ideal(x*z-y^2, x*w-y*z));  
ideal(-y^2 +x*z, -y*z +x*w, z^2 -y*w)  
/**/ toric(ideal(x*z-y^2, x*w-y*z), [y]);  
ideal(-y^2 +x*z, -y*z +x*w, z^2 -y*w)  
/**/ Use R := QQ[x,y,z];  
/**/ toric([[1,3,2],[3,4,8]]);  
ideal(-x^16 +y^2*z^5)  
/**/ toric(mat([[1,3,2],[3,4,8]]));  
ideal(-x^16 +y^2*z^5)  
```

I-20.12 transposed

**syntax**

transposed(M: MAT): MAT

**Description**

This function returns the transpose of the matrix $M$.

```coconut
/**/ M := mat([[1,2,3],[4,5,6]]);  
/**/ M;  
matrix([  
1, 2, 3],  
[4, 5, 6])  
/**/ transposed(M);  
```
Try C1 UponError E Do C2 EndTry
where C1, C2 are sequences of commands and E is a variable identifier.

Description

Usually, when an error occurs during the execution of a command, the error is automatically propagated out of the nesting of the evaluation. This can be prevented with the use of “Try..UponError”.

If an error occurs during the execution of the commands C1, then it is captured by the command “UponError” and assigned to the variable E, and the commands C2 are executed; the string inside E may be retrieved using “GetErrMesg” (I-7.14 pg.101). If no error occurs then the variable E and the commands C2 are ignored.

See Also: error(I-5.9 pg.79), GetErrMesg(I-7.14 pg.101)
I-20.14 tuples

**syntax**

tuples(S: LIST, N: INT): LIST

**Description**

This function computes all N-tuples with entries in S. It is equivalent to “S >> S >> ... >> S” [N times].

```plaintext
/**/ tuples([1, 4, 7], 2);
[[1, 1], [1, 4], [1, 7], [4, 1], [4, 4], [4, 7], [7, 1], [7, 4], [7, 7]]
```

**See Also:** CartesianProduct, CartesianProductList(I-3.4 pg.42), permutations(I-16.5 pg.202), subsets(I-19.38 pg.263)

I-20.15 type

**syntax**

type(E: OBJECT): TYPE

**Description**

This function returns the data type of E.

```plaintext
/**/ L := [1,"a",2,"b",3,"c"];
/**/ [ X In L | type(X)=INT ];
[1, 2, 3]

/**/ type(type(INT)); -- Type returns a value of type TYPE
TYPE

/**/ CurrentTypes();
[BOOL, ERROR, FUNCTION, ...]
```

**See Also:** CurrentTypes(I-3.50 pg.62)
Chapter I-21

U

I-21.1 UnivariateIndetIndex

**syntax**

UnivariateIndetIndex(F: RINGELEM): INT

**Description**

This function returns 0 if the polynomial F is not univariate otherwise it returns the indeterminate index of F.

NOTE: If F is a constant, it returns 1.

**example**

```plaintext
/**/ Use R ::= QQ[x,y,z];
/**/ UnivariateIndetIndex(3*x^4-2*x-1);
   1
/**/ UnivariateIndetIndex(x-y-1);
   0
/**/ UnivariateIndetIndex(one(R));
   1
```

**See Also:** indet(I-9.18 pg.123), IndetSubscripts(I-9.22 pg.126), IndetIndex(I-9.19 pg.124), IndetName(I-9.20 pg.124), indets(I-9.21 pg.125), NumIndets(I-14.32 pg.192)

I-21.2 unprotect

**syntax**

unprotect X;

**Description**

This command undoes the effect of the “protect” (I-16.32 pg.214) command; once a variable has been unprotected, it may be assigned to freely.

**example**

```plaintext
/**/ X := 1;
/**/ protect X;  --> cannot assign to X henceforth
/**/
```
/**/ unprotect X;  --> remove protection, X may be assigned to now
/**/ X := 2;

See Also: protect(I-16.3 pg.214)

I-21.3 Unset [OBSCOLE]

[OBSCOLE] syntax

Description

[OBSCOLE]

I-21.4 untagged

syntax

untagged(E:TAGGED_OBJECT):UNTAGGED_OBJECT

Description

This function strips an object E of its tag, if any. "@E" is equivalent to "untagged(E)". Tags are used for pretty printing of objects. See the reference listed below.

example

/**/ L := [1,2,3];
/**/ M := tagged(L,"MyTag");
/**/ type(L);
LIST

/**/ type(M);
TAGGED("MyTag")

/**/ type(untagged(M));
LIST

See Also: Printing a Tagged Object(III-16.2 pg.379), tag(I-20.1 pg.271), tagged(I-20.2 pg.271)

I-21.5 use

syntax

use R
use RingDefn
use R ::= RingDefn

where R is a RING, and RingDefn is a ring definition.
Description

This command works only at top-level; it makes a ring active, i.e., it makes that ring the “current ring”. The command will also let you create a new ring, and make it active immediately “Use NewR ::= RingDefn;” where “RingDefn” is a ring definition; this is a shorthand for “NewR ::= RingDefn; Use NewR;”

This command cannot be called inside a function, and it is never necessary (if you write clean programs ;-) In CoCoA-5 you can define new rings, return rings, assign rings and pass rings as arguments (this was not possible in CoCoA-4).

/**/ use S ::= QQ[x,y,z];
/**/ Print CurrentRing;
RingDistrMPolyClean(QQ, 3)
/**/ indets(CurrentRing);
x, y, z

/**/ use QQ[u]; -- can be used w/out a ring identifier
/**/ indets(CurrentRing);
u

/**/ define SumInAnotherRing(N)
/**/ K := NewRingTwinFloat(128); -- 128 bits of precision
/**/ P ::= K[x[1..N]], Lex;
/**/ return sum(indets(P));
/**/ enddefine;

/**/ SumInAnotherRing(4);
/**/ CoeffRing(RingOf(It));
RingTwinFloat(AccuracyBits=128, BufferBits=128, NoiseBits=32)

See Also: Introduction to RINGHOM(III-10.1 pg.359), CurrentRing(I-3.49 pg.61), RingOf(I-18.41 pg.239), ReadExpr(I-18.16 pg.227)
Chapter I-22

V

I-22.1 valuation [OBSOLETE]

Description

Renamed "FactorMultiplicity" (I-6.3 pg.84).

See Also: FactorMultiplicity(I-6.3 pg.84)

I-22.2 VersionInfo

Description

This function returns a record with various information about CoCoA and CoCoALib (the mathematical core of CoCoA)

```plaintext
/**/ indent(VersionInfo());
record[
    CoCoALibVersion := "0.99533",
    CoCoAVersion := "5.1.0",
    CompilationDate := "May 14 2014 15:48:58",
    ...
```

See Also: CoCoALib(II-8.1 pg.319)
Chapter I-23

W

I-23.1  wdeg

** syntax **

\texttt{wdeg(F: RINGELEM): LIST}

** Description **

This function returns the multi-weighted degree of \( F \), as determined by the matrix weights of the polynomial ring of \( F \). The function “\textit{deg}” (I-4.6 pg.66) returns the standard degree.

\textbf{NOTE: In CoCoA-4 “\textit{deg}” (I-4.6 pg.66) gave the weight given by only the first row of the weights matrix.}

** example **

```cocoa
/**/ M := matrix([[2,3,4], [1,0,2], [1,0,0]]);
/**/ P := NewPolyRing(QQ, "x,y,z", M, 1); -- GradingDim=1
/**/ Use P;
/**/ wdeg(x*y^2+y);
[8]
/**/ P := NewPolyRing(QQ, "x,y,z", M, 2); -- GradingDim=2
/**/ Use P;
/**/ wdeg(x*y^2+y);
[8, 1]
/**/ deg(x*y^2+y);
3

/**/ P4 := NewFreeModule(P,4); -- the default module ordering is TPOS
/**/ wdeg(ModuleElem(P4, [0, x, y^2, x^2]));
[6, 0]
/**/ LT(ModuleElem(P4, [0, x, y^2, x^2]));
[0, 0, y^2, 0]
```

\textbf{See Also:} \textit{deg}(I-4.6 pg.66), \textit{LF}(I-12.8 pg.158)

I-23.2  WeightsMatrix [OBSOLETE]

** syntax **

\texttt{WeightsMatrix(R: RING): MAT}
Description

This function is now called “GradingMat” (I-7.19 pg.103).
See Also: deg(I-4.6 pg.66), wdeg(I-23.1 pg.285)

I-23.3 while

While B Do C EndWhile

where B is a boolean expression and C is a sequence of commands.

Description

The command sequence C is repeated until B evaluates to False.

example

/**/ N := 0;
/**/ while N <= 5 do
/**/    PrintLn 2, "\", N, " = ", 2^N;
/**/    N := N+1;
/**/ EndWhile;
2^0 = 1
2^1 = 2
2^2 = 4
2^3 = 8
2^4 = 16
2^5 = 32

See Also: for(I-6.13 pg.88), foreach(I-6.14 pg.89), repeat(I-18.30 pg.234)

I-23.4 WithoutNth

WithoutNth(L: LIST, N: INT): LIST

Description

This function returns the list obtained by removing the “N”-th component of the list “L”. The list “L” itself is not changed; compare with “remove” (I-18.29 pg.234).

example

/**/ L := [1,2,3,4,5];
/**/ WithoutNth(L,3);
[1, 2, 4, 5]

See Also: remove(I-18.29 pg.234)

I-23.5 WLog [OBSOLETE]

[WLog [OBSOLETE]]
Description

[OBSOLETE] This function returns the weighted list of exponents of the leading term of F, as determined by the first row of the weights matrix.

See Also: log(I-12.16 pg.162)
Chapter I-24

X

I-24.1 XelMat

**syntax**

\[ \text{XelMat}(N: \text{INT}): \text{MAT} \]

**Description**

This function returns the matrix defining a standard term-ordering.

**example**

```c
/**/ \text{XelMat}(3);
\text{matrix}(\[
[0, 0, 1],
[0, 1, 0],
[1, 0, 0]
])
```

**See Also:** OrdMat(I-15.10 pg.199), Orderings(III-9.5 pg.354), StdDegRevLexMat(I-19.31 pg.260), StdDegLexMat(I-19.30 pg.260), LexMat(I-12.6 pg.157), RevLexMat(I-18.37 pg.237)
Chapter I-25

Z

I-25.1  zero

**syntax**

zero(R: RING): RINGELEM

**Description**

This function returns the additive identity of a ring. For when you want to force the integer “0” to be a “RINGELEM”.

**example**

```lang
/**/ P ::= ZZ/(101)[x,y,z];
/**/ N := 0; Print N, " of type ", type(N);
0 of type INT
/**/ N := zero(P); Print N, " of type ", type(N);
0 of type RINGELEM
/**/ N := 300*0; Print N, " of type ", type(N);
0 of type INT
/**/ N := 300*zero(P); Print N, " of type ", type(N);
0 of type RINGELEM
/**/ F := NewFreeModule(P, 3);
/**/ zero(F);
[0, 0, 0]
```

**See Also:** one(I-15.1 pg.195), IsZero(I-9.72 pg.146)

I-25.2  ZeroMat

**syntax**

ZeroMat(R: RING, NumRows: INT, NumCols: INT): MAT

**Description**

This function returns the “NumRows x NumCols” zero matrix with entries in “R”.

**example**

```lang
/**/ Use R ::= QQ[x,y,z];
/**/ ZeroMat(QQ, 1, 3); --> same as NewMatFilled(1,3, 0)
matrix(QQ,
```

291
**ZPQ**

**syntax**

- ZPQ(F: RINGELEM): RINGELEM
- ZPQ(F: LIST of RINGELEM): LIST of RINGELEM
- ZPQ(I: IDEAL): IDEAL

**Description**

***** NOT YET IMPLEMENTED *****

The function “ZPQ” maps a polynomial with finite field coefficients into one with rational (actually, integer) coefficients. It is not uniquely defined mathematically, and currently for each coefficient the least non-negative equivalent integer is chosen. Users should not rely on this choice, though any change will be documented.

See “QZP” (I-17.4 pg.218) for more details.

**example**

Use R ::= QQ[x,y,z];
F := 1/2*x^3 + 34/567*x*y*z - 890; -- a poly with rational coefficients
Use S ::= ZZ/(101)[x,y,z];
QZP(F); -- compute its image with coeffs in ZZ/(101)
-50x^3 - 19xyz + 19
-------------------------------
G := It;
Use R;
ZPQ(G); -- now map that result back to QQ[x,y,z] it is NOT the same as F...
51x^3 + 82xyz + 19
-------------------------------

See Also: BringIn(I-2.12 pg.39)

**ZZ**

**syntax**

**Description**

This system variable is constant; its value is the ring of integers. Its name is protected so that it cannot be re-assigned to any other value.

**example**

/**/ P ::= ZZ/(101)[x,y,z];
/**/ Use ZZ;
/**/ type(5);
INT
/**/ type(RingElem(ZZ, 5));
RINGELEM

See Also: QQ(I-17.1 pg.217)
Part II

The CoCoA Programming Language
II-1.1 An Overview of CoCoA Programming

The CoCoA system includes a full-fledged high level programming language, CoCoALanguage, complete with loops, branching, scoping of variables, and input/output control. The language is used whenever one issues commands during a CoCoA session. A sequence of commands may be stored in a text file and then read into a CoCoA session using the \texttt{source} (I-19.21 pg.256) command.

The most important construct in CoCoA programming is the user-defined function, created with \texttt{define} (I-4.4 pg.64). A user-defined function can take any number of arguments, of any types, perform CoCoA commands, and return values. Collections of these functions can be stored in text files, as mentioned in the preceding paragraph, or formed into CoCoA \textit{packages}, to be made available for general use.
Chapter II-2

Language Elements

II-2.1 Character Set and Special Symbols

The CoCoA character set consists of the 26 lower case letters, the 26 upper case letters, the 10 digits and the special characters listed in the table below. Note that the special character “|” looks a bit different on some keyboards (its ASCII code is 124).

| blank _ underscore ( | left parenthesis | |
| + plus = equal ) | right parenthesis |
| - minus < less than [ | left bracket |
| * asterisk < greater than ] | right bracket |
| / slash | vertical bar ' | single quote |
| : colon . period " | double quote |
| ~ caret ; semicolon | |
| , comma % percent |

Special Characters

The character-groups listed in the table below are special symbols in CoCoA

| := assign .. range |
| << input from // start line comment |
| <> not equal -- start line comment |
| >< Cartesian product ::= ring definition |
| <= less than or equal to */ start embedded comment |
| >= greater than or equal to */ end embedded comment |

Special Character-groups

II-2.2 Identifiers

There are two types of identifiers or names.

* Identifiers of ring indeterminates (see “NewPolyRing” (I-14.8 pg.182))

* Predefined or user-defined names (functions and CoCoALanguage variables).

See Also: Indeterminates(III-9.4 pg.354)
II-2.3 Reserved Names

***** NOT YET UPDATED TO CoCoA-5: follow with care *****

The names in the following tables are reserved and cannot be used otherwise. The names in the first table are case insensitive (e.g. CLEAR, Clear and ClEaR are all reserved). The names in the second table are case sensitive.

...work in progress...

----------------------------------------------------
| Alias And Block Ciao Define |
| Describe Do Elif Else End |
| EndBlock EndTry EndDefine EndFor |
| EndForeach EndIf EndPackage EndRepeat EndUsing |
| EndWhile Eof False For Foreach |
| Global Help If In IsIn |
| NewLine Not On Or Package |
| Print PrintLn Quit Repeat Record |
| Return Set Skip Source Step |
| Then Time To True Unset |
| Until Use Using Var While |
| QQ ZZ |

Case insensitive reserved names

----------------------------------------------------
| BOOL DegLex DegRevLex DEVICE ERROR |
| FUNCTION IDEAL INT LIST Lex |
| MAT MODULE NULL Null PANEL |
| POLY PosTo RAT RATFUN RING |
| STRING TAGGED ToPos TYPE MODULEELEM |
| Xel ZMOD |

Case sensitive reserved names

II-2.4 Comments

End-of-line comments in CoCoA start with either “--” or “//”; all text up to the end of the line is considered comment. CoCoA also allows embedded comments; these begin with the symbol “/*” and end with the symbol “*/”. CoCoA ignores the contents of a comment, and treats it as if it were just a space.

```
exmple
/**/  // This is an end-of-line comment
/**/  Print 1+1; -- a command followed by an end-of-comment
2
/**/  A := [1 /*x-coord*/, 2 /*y-coord*/ ]; --> embedded comments
```
Chapter II-3

Operators

II-3.1 CoCoA Operators

In CoCoA there are 5 main types of operators: algebraic operators, relational operators, boolean operators, selection operators, and the range operator. There is also an n-ary operator “>£” for forming Cartesian products of lists and an operator “:=” used in defining rings.

The meaning of an operator depends on the types of its operands; the “+” in the expression “A + B” represents the sum of polynomials, or of ideals, or of matrices, etc. according to the type of A and B.

The CoCoA operators are, from the highest to the lowest priority:

[] . (selection operators)
  - %
  + - (as unary operators)
  * : /  
  + - (as binary operators)
  ..
  = <> < <= > >=

IsIn
And
Or

Operations with equal priority are performed from left to right. When in doubt, parentheses may be used to enforce a particular order of evaluation.

See Also: operators, shortcuts(I-0.1 pg.23)

II-3.2 Algebraic Operators

The algebraic operators are:

+ - * / : -

The following table shows which operations the system can perform between two objects of the same or of different types; the first column lists the type of the first operand and the first row lists the type of the second operand. So, for example, the symbol “:*” in the box on the seventh row and fourth column means that it is possible to divide an ideal by a polynomial.

<table>
<thead>
<tr>
<th></th>
<th>INT</th>
<th>RAT</th>
<th>RINGELEM</th>
<th>MODULEELEM</th>
<th>IDEAL</th>
<th>MODULE</th>
<th>MAT</th>
<th>LIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RAT</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

301
Algebraic operators

Remarks:
* Let $F$ and $G$ be two polynomials. If $F$ is a multiple of $G$, then $F/G$ is the polynomial obtained from the division of $F$ by $G$, otherwise $F/G$ is a rational function (common factors are simplified). The functions “$\text{div}$” (I-4.20 pg.73) and “$\text{mod}$” (I-13.21 pg.175) can be used to get the quotient and the remainder of a polynomial division.
* Let $L_1$ and $L_2$ be two lists of the same length. Then $L_1 + L_2$ is the list obtained by adding $L_1$ to $L_2$ componentwise.
* If $I$ and $J$ are both ideals or both modules, then $I : J$ is the ideal consisting of all polynomials $f$ such that $fg$ is in $I$ for all $g$ in $J$.

II-3.3 Relational Operators

See Also: Equality Test(I-5.7 pg.78), Comparison Operators(I-3.30 pg.53), IsIn(I-9.47 pg.136)

II-3.4 Selection Operators

The selection operators are

[] .

Let $N$ be of type INT and let $L$ be of type STRING, MODULEELEM, LIST, or MAT. Then the meaning of $L[N]$ depends on the type of $L$ as explained in the following table:

<table>
<thead>
<tr>
<th>Type of L</th>
<th>Meaning of L[N]</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRING</td>
<td>string consisting of the $N$-th character of $L$.</td>
</tr>
<tr>
<td>MODULEELEM</td>
<td>$N$-th component of $L$</td>
</tr>
<tr>
<td>LIST</td>
<td>$N$-th element of $L$</td>
</tr>
<tr>
<td>MAT</td>
<td>$N$-th element of $L$</td>
</tr>
</tbody>
</table>

Selection Operator

If $N$ is an identifier and $L$ is of type RECORD, then “$L.N$” indicates the object contained in the field $N$ of the record $L$ (see “record” (I-18.20 pg.229)).

See Also: record(I-18.20 pg.229), List Constructors(III-5.2 pg.340)

II-3.5 Range Operator

If $M$ and $N$ are of type INT, then the expression: “$M .. N$” returns
* the list “[M, M+1, ... , N]” if $M \leq N$;
* the empty list, “[[]]”, otherwise.
NOTE: Large values for M and N are not permitted; typically they should lie in the range about $-10^9$ to $+10^9$.
NOTE: see example for how to select a sub-range of a list

If x and y are indeterminates in a ring, then “x .. y” gives the indeterminates between x and y in the order they appear in the definition of the ring.

```plaintext
/**/ 1..10;
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
/**/ Use R ::= QQ[x,y,z,a,b,c,d];
/**/ z..c;
[z, a, b, c]
/**/ L := [11, 22, 33, 44, 55];
/**/ PartOfL := L[2]..L[4];  --> probably *NOT* what you want!
/**/ PartOfL := [ L[k] | k in 2..4 ];  --> OK, this is RIGHT!
```

See Also: CoCoA Operators(II-3.1 pg.301), List Constructors(III-5.2 pg.340), LIST(III-5 pg.339)
Chapter II-4

Evaluation and Assignment

II-4.1 Evaluation

An expression is by itself a valid command. The effect of this command is that the expression is evaluated in the current ring and its value is displayed.

The evaluation of an expression in CoCoA is normally performed in a full recursive evaluation mode. Usually the result is the fully evaluated expression.

The result of the evaluation is automatically stored in the variable “It” (I-9.77 pg.148).

example

```cocoa
/**/ 2 + 2;
4
/**/ It + 3;
7
/**/ It;
7
/**/ X := 5;
/**/ It;
7
```

The command “X := 5” is an assignment, not an evaluation; so it does not change the value of the variable “It” (I-9.77 pg.148).

If an error occurs during the evaluation of an expression, then the evaluation is interrupted and the user is notified about the error.

II-4.2 Assignment

An assignment command has the form

\[ L := E \]

where L is a variable and E is an expression. The assignment command binds the result of the evaluation of the expression E to L in the working memory.

example

```cocoa
/**/ Use R ::= QQ[t,x,y,z];
/**/ I := ideal(x,y);
/**/ M := 5; N := 8;
/**/ T := M+N;
/**/ T;
```
/**/ T := T+1; -- note that T occurs on the right, also
/**/ T;

/**/ L := [1, 2, 3];
/**/ L[2] := L[3];
/**/ L;
[1, 3, 3]

/**/ P := record[F := x*z];
/**/ P.Degree := Deg(P.F);
/**/ P;
record[Degree := 2, F := x*z]
Chapter II-5

Flow Control: Conditional Statements and Loops

II-5.1 All CoCoA commands

This is a complete list of all CoCoA commands:

- `break`  break out of a loop
- `for`  loop command
- `foreach`  loop command
- `if`  conditional statement
- `repeat`  loop command
- `return`  exit from a function
- `try`  try and catch an error
- `while`  loop command

II-5.2 Commands and Functions for Branching

The following are the CoCoA commands for constructing conditional statements:

```
if  conditional statement
```

II-5.3 Commands and Functions for Loops

The following are the commands and functions for loops:

```
break  break out of a loop
for  loop command
foreach  loop command
repeat  loop command
```
return  exit from a function
while   loop command
Chapter II-6

Input/Output

II-6.1 Introduction to IO

***** NOT YET UPDATED TO CoCoA-5: follow with care *****

Input and output is implemented in CoCoA through the use of “devices”. At present, the official devices are: (1) standard IO (the CoCoA window), (2) text files, and (3) strings. What this means is that it is possible to read from or write to any of these places. The cases are discussed separately, below. Text files may be read verbatim or—with the “source” (I-19.21 pg.256) command—be executed as CoCoA commands.

II-6.2 Standard IO

***** NOT YET UPDATED TO CoCoA-5: follow with care *****

Standard IO is what takes place normally when one interacts with CoCoA. CoCoA accepts and interprets strings typed in by the user and prints out expressions. If E is a CoCoA object, then the command

E;

causes the value of E to be printed to the CoCoA window. One may also use the functions “print” (I-16.25 pg.211) and “println” (I-16.29 pg.213) for more control over the format of the output.

The official devices that are being used here are “DEV.STDIN” and “DEV.OUT”. So for instance, the commands “Get” (I-7.10 pg.100) and “print on” (I-16.26 pg.212) can be used with the standard devices although they are really meant to be used with the other devices. “Print E On DEV.OUT” is synonymous with “Print E”. Also, one may use “Get(DEV.STDIN,10)”, for example, to get the next 10 characters typed in the CoCoA window. Thus, clever use of “Get” (I-7.10 pg.100) will allow your user-defined functions to prompt the user for input, but normal practice is to pass variables to a function as arguments to that function.

II-6.3 File IO

***** NOT YET UPDATED TO CoCoA-5: follow with care *****

To print CoCoA output to a file, one first opens the file with “OpenOFile” (I-15.5 pg.197) then prints to the file using “print on” (I-16.26 pg.212).

To receive verbatim input from a file, one first opens the file with “OpenIFile” (I-15.2 pg.195), then gets characters from the file with “Get” (I-7.10 pg.100). Actually, “Get” (I-7.10 pg.100) gets a list of ASCII codes for the characters in the file. These can be converted to real characters using the function “ascii” (I-1.14 pg.31).

```plaintext
D := OpenOFile("my-file"); -- open text file with name "my-file",
-- creating it if necessary
```
Print "hello world" On D; -- append "hello world" to my-file
Close(D); -- close the file
D := OpenIFile("my-file"); -- open "my-file"
Get(D,10); -- get the first ten characters, in ASCII code
-------------------------------
ascii(It); -- convert the ASCII code
hello worl
-------------------------------
Close(D);

To read and execute a sequence of CoCoA commands from a text file, one uses the “source” (I-19.21 pg.256) command. For instance, if the file “MyFile.coc” contains a list of CoCoA commands, then

Source "MyFile.cocoa";

reads and executes the commands.

See Also: ascii(I-1.14 pg.31), close(I-3.14 pg.46), Get(I-7.10 pg.100), OpenIFile(I-15.2 pg.195), OpenOFile(I-15.5 pg.197), OpenLog(I-15.4 pg.196), CloseLog(I-3.15 pg.46), print on(I-16.26 pg.212), source(I-19.21 pg.256)

II-6.4 String IO

***** NOT YET UPDATED TO CoCoA-5: follow with care *****

To print CoCoA output to a string, one may use “OpenOString” (I-15.6 pg.198) to “open” the string, then “print on” (I-16.26 pg.212) to write to it. To read from a string, one may open the string for input with “OpenIString” (I-15.3 pg.196) then get characters from it with “Get” (I-7.10 pg.100).

example

S := "hello world";
D := OpenIString("", S); -- open the string S for input to CoCoA
-- the first argument is just a name for the device
L := Get(D,7); -- read 7 characters from the string
L; -- ASCII code
-------------------------------
ascii(L); -- convert ASCII code to characters
hello w
-------------------------------
Close(D); -- close device D
D := OpenOString(""); -- open a string for output from CoCoA
L := [1,2,3]; -- a list
Print L On D; -- print to D
D;
record[Name := ",
Type := "OString", Protocol := "CoCoALanguage"]

S := Cast(D, STRING); -- S is the string output printed on D
S; -- a string
[1, 2, 3]
Print " more characters" On D; -- append to the existing output string
Cast(D, STRING);
[1, 2, 3] more characters

There are usually more direct ways to collect results in strings. For instance, if the output of a CoCoA command is not already of type STRING, one may convert it to a string using “sprint” (I-19.24 pg.257).
II-6.5 Commands and Functions for IO

The following are commands and functions for input/output:

- **block**: group several commands into a single command
- **close**: close a device
- **CloseLog**: close a log of a CoCoA session
- **format**: convert object to formatted string
- **Get**: read characters from a device
- **IO.SprintTrunc**: convert to a string and truncate
- **LaTeX**: LaTeX formatting
- **NewFreeModule**: create a new FreeModule
- **NewLine [OBSOLETE]**: [OBSOLETE] string containing a newline
- **OpenIFile**: open input file
- **OpenIString**: open input string
- **OpenLog**: open a log of a CoCoA session
- **OpenOFile**: open output file
- **OpenOString**: open output string
- **OpenSocket**: open a socket connection
- **print**: print the value of an expression
- **print on**: print to an output device
- **println**: print the value of an expression
- **source**: read commands from a file or device
- **SourceRegion**: read commands from a region in a file
- **sprint**: convert to a string
- **tag**: returns the tag string of an object
- **tagged**: tag an object for pretty printing
- **untagged**: untag an object
Chapter II-7

CoCoA Packages

II-7.1 Introduction to Packages

User-defined functions may be saved in separate files and read into a CoCoA session using the “source” (I-19.21 pg.256) command. If one sources several such files or, especially, if a file is to be made available for general use, a possible problem arises from conflicting function names. If two functions with the same name are read into a CoCoA session, only the one last read survives. To avoid this, functions may be collected in “packages”.

A CoCoA package is essentially a list of functions labeled with prefix.

Writing a package in CoCoA-5 is slightly different from how it was done in CoCoA-4 it is easier!).

See Also: define(I-4.4 pg.64), source(I-19.21 pg.256)

II-7.2 First Example of a Package

The following is an example of a package. It could be typed into a window as-is during a CoCoA session, but we will assume that it is stored in a file in the CoCoA directory under the name “one.cpkg5”.

```plaintext
package $contrib/toypackage
export ToyTest;

define IsNumberOne(n)
  if n = 1 then return true; else return false; endif;
enddefine;

define ToyTest(n)
  if IsNumberOne(n) then
    print "The number 1";
  else
    print "Not the number 1";
  endif;
enddefine;

endpackage; -- of toypackage
```

Below is output from a CoCoA session in which this package was used:

```plaintext
-- read in the package:
Source "one.cpkg";
```
II-7.3 Package Essentials

A package begins with

```
Package $PackageName
```

and ends with

```
EndPackage;
```

"PackageName" is a string that will be used to identify the package. The dollar sign is required. The "PackageName" must be a valid identifier: i.e. start with a letter and comprise only letters, digits, slash and underscore; the name should be meaningful (and usually long, to avoid any risk of a name clash). We recommend using a name of the form "contrib/subject".

All packages in the CoCoA directory "packages" are automatically loaded when starting CoCoA.

II-7.4 Global Aliases

A global alias for a package is formed by using the command "alias" (I-1.7 pg.27) during a CoCoA session.

NOTE: global aliases cannot be used in function definitions. This is to force independence of context. Inside a function, one must use the complete package name.

See Also: alias(I-1.7 pg.27), aliases(I-1.8 pg.28)

II-7.5 Sharing Your Package

If you create a package that others might find useful, please contact the CoCoA team by email at "cocoa at dima.unige.it".

Include comments in the package that:

* explain the use of the package
* give the syntax, description, examples for exported functions.

II-7.6 Commands and Functions for Packages

The following are commands and functions for packages:

- `alias` define aliases for package names
- `aliases` list of global aliases
- `Packages` list of loaded packages
- `PkgName` returns the name of a package

```
II-7.7 Supported Packages

Several packages are supported by the CoCoA team. These packages contain functions that are not built into CoCoA because they are of a more specialized or experimental nature.

Some functions which used to be defined in supported packages are now official functions in CoCoA-5.

II-7.8 Galois Package

***** NOT YET UPDATED TO CoCoA-5: follow with care *****

TITLE : galois.cpkg
DESCRIPTION : CoCoA package for computing in a cyclic algebraic extension
AUTHOR : A. Bigatti, D. La Macchia, F. Rossi

-- Enter
   $contrib/galois.Man();
   to get a complete description of the package including a suggested alias.

II-7.9 Integer Programming

***** NOT YET UPDATED TO CoCoA-5: follow with care *****

TITLE : intprog.cpkg
DESCRIPTION : CoCoA package for applying toric ideals to integer programming
AUTHOR : A. Bigatti

-- Enter
   $contrib/intprog.Man();
   to get a complete description of the package including a suggested alias.

II-7.10 Algebra of Invariants

***** NOT YET UPDATED TO CoCoA-5: follow with care *****

TITLE : invariants.cpkg
DESCRIPTION : CoCoA package for computing homogeneous generators of an algebra of invariants, and for testing invariance of a polynomial
AUTHOR : A. Del Padrone

-- Enter
   $contrib/invariants.Man();
   to get a complete description of the package including a suggested alias.

II-7.11 Special Varieties

***** NOT YET UPDATED TO CoCoA-5: follow with care *****
II-7.12 Statistics

***** NOT YET UPDATED TO CoCoA-5: follow with care *****

TITLE : stat.cpkg
DESCRIPTION : package for design of experiments in statistics
AUTHOR : M. Caboara

-- Enter
$contrib/stat.Man();
to get a complete description of the package including a suggested alias.

II-7.13 Geometrical Theorem-Proving

***** NOT YET UPDATED TO CoCoA-5: follow with care *****

TITLE : thmproving.cpkg
DESCRIPTION : CoCoA package for geometrical theorem-proving in euclidean space
AUTHOR : L. Bazzotti, G. Dalzotto

-- Enter
$contrib/thmproving.Man();
to get a complete description of the package including a suggested alias.

II-7.14 Typevectors

***** NOT YET UPDATED TO CoCoA-5: follow with care *****

TITLE : typevectors.cpkg
DESCRIPTION : CoCoA package for computing type-vectors associated to Hilbert functions of ideals of points
AUTHOR : E.Carlini, M.Stewart

-- Enter
$contrib/typevectors.Man();
to get a complete description of the package including a suggested alias.

II-7.15 Conductor

***** NOT YET UPDATED TO CoCoA-5: follow with care *****

TITLE : conductor.cpkg
II-7.16  Matrix Normal Form

***** NOT YET UPDATED TO CoCoA-5: follow with care *****

TITLE : matrixnormalform.cpkg
DESCRIPTION : CoCoA package for computing normal forms of a matrix, Smith Normal Form (PID)
AUTHOR : A.Bigatti, S.DeFrancisci

-- Enter
$contrib/matrixnormalform.Man();
 to get a complete description of the package including a suggested alias.

II-7.17  CantStop

***** NOT YET UPDATED TO CoCoA-5: follow with care *****

TITLE : CantStop.cpkg
DESCRIPTION : CoCoA package for playing Can't Stop and studying strategies
AUTHOR : A.Bigatti

-- Enter
$contrib/CantStop.Man();
 to get a complete description of the package including a suggested alias.

II-7.18  Control

***** NOT YET UPDATED TO CoCoA-5: follow with care *****

TITLE : control.cpkg
DESCRIPTION : CoCoA package for Geometric Control Theory
AUTHOR : M. Anderlucci and M. Caboara

-- Enter
$contrib/control.Man();
 to get a complete description of the package including a suggested alias.
Chapter II-8

Linked libraries

II-8.1 CoCoALib

CoCoALib “http://cocoa.dima.unige.it/cocoalib”.
CoCoALib is the mathematical core of CoCoA-5. It may be used directly as a C++ library.

II-8.2 GMP

GMP - The GNU Multiple Precision Arithmetic Library “https://gmplib.org”
All arbitrary precision integer/rational/floating-point datatypes and operations are based on GMP.

II-8.3 GSL

Some functions from GSL have been ported to CoCoA-5. There is no manual yet because it’s work in progress.

II-8.4 Frobby

Frobby - Computations With Monomial Ideals “http://www.broune.com/frobby”
All functions starting with “Frb” are implemented in Frobby.

II-8.5 Normaliz

libNormaliz is a C++ library for computations with rational cones and affine monoids; full details may be found on the official Normaliz website “http://www.home.uni-osnabrueck.de/wbruns/normaliz”

When CoCoA is compiled it is possible to incorporate also libNormaliz; if so, then many libNormaliz functions can be called from CoCoA-5. All CoCoA functions starting with “Nmz” are actually implemented in libNormaliz.
Chapter II-9

Migrating from CoCoA-4 and keeping up-to-date

II-9.1 Changes in the CoCoA language

CoCoA-5 is largely, but not completely, backward-compatible with CoCoA-4. Some commands/functions have changed name; others have been removed or replaced. Here we give a little guidance to help update your CoCoA-4 programs to CoCoA-5.

The operator “Not” has been replaced by the function “not(...)”.

```c4/*C4*/ If Not X IsIn L Then ... EndIf;
/*C5*/ If not(X IsIn L) Then ... EndIf;
```

Several functions modify one of their arguments (e.g. “append” (I-1.11 pg.29), “sort” (I-19.17 pg.253)); CoCoA-5 wants these arguments to be identified with the new keyword “ref” (I-18.23 pg.231), and will issue a warning if you don’t do this (just to make sure you know that “L” will be modified).

```c4/*C4*/ L := [1,2,3]; Append(L, 4);
/*C5*/ L := [1,2,3]; append(ref L, 4);
```

Implicit multiplication has gone: either write “x*y” instead of “xy” for every product, or use “CoCoA-4 mode” (I-3.16 pg.46).

```c4/*C4*/ F := 3xyzt;
/*C5*/ F := 3*x*y*z*t; OR F := ***3xyzt***;
```

Many CoCoA-4 functions would employ the “CurrentRing” implicitly (e.g. “NumIndets()”, “CoeffRing()”). They now require an explicit argument; you can pass “CurrentRing” as the argument, but inside a function you must make that system variable visible via the command “TopLevel” (I-20.9 pg.274).

```c4/*C4*/ Define LastIndet() Return Last(Indets()); EndDefine;
/*C5*/ Define LastIndet()
   TopLevel CurrentRing;
   Return last(indets(CurrentRing));
EndDefine;
```

However, we encourage you to consider modifying your function so that it does not depend on “CurrentRing”; e.g. you can find out to which ring a value belongs by calling the function “RingOf” (I-18.41 pg.239).
The function “LinKer” (I-12.10 pg.159) has been replaced by “LinKerBasis” (I-12.11 pg.160), and there is a new function called “LinKer” (I-12.10 pg.159) which produces a matrix. More generally, see also the CoCoA-4 "translation table" in the CoCoAManual directory or at 

http://cocoa.dima.unige.it/cocoa/CoCoA/TranslationTable.html

See Also: CoCoA-4 mode(I-3.16 pg.46), TopLevel(I-20.9 pg.274), CurrentRing(I-3.49 pg.61), RingOf(I-18.41 pg.239)

II-9.2 Recent changes in the CoCoA-5 language

There are a few changes in the language even from the first versions of CoCoA-5. The operator “Not” has been replaced by the function “not(...)”.

```cocoa
  /*5.0.9*/ If Not X IsIn L Then ... EndIf;
  /*5.1.0*/ If not(X IsIn L) Then ... EndIf;
```

The anonymous function called “lambda” is now called “func” (I-6.21 pg.92).

```cocoa
  /*5.0.9*/ square := Lambda(x) Return x^2; EndLambda;
  /*5.1.0*/ square := Func(x) Return x^2; EndFunc;
```

See Also: not(I-14.27 pg.190), func(I-6.21 pg.92)

II-9.3 Obsolete and obsolescent functions

As the language evolves some functions might become obsolete, maybe just more sensibly renamed. This is the list of such functions: see in the manual for reasons/updates.

```
Call [OBSOLETE] [OBSOLETE] apply a function to given arguments
Cast [OBSOLETE] [OBSOLETE] type conversion
ColumnVectors [OBSOLETE] [OBSOLETE] list of module elements
Comp [OBSOLETE] [OBSOLETE] access a component
E_ [OBSOLETE] [OBSOLETE] vector of the canonical basis
Function [OBSOLETE] [OBSOLETE] replaced by describe
functions [OBSOLETE] [OBSOLETE] renamed RingID
ID [OBSOLETE] [OBSOLETE] renamed RingID
IsInSubalgebra [OBSOLETE] [OBSOLETE] check if one polynomial is in a subalgebra
IsNumber [OBSOLETE] [OBSOLETE] checks if the argument is a number
LinKerModP [OBSOLETE] [OBSOLETE] find the kernel of a matrix mod p
LinSol [OBSOLETE] [OBSOLETE] find a solution to a linear system
MapDown [OBSOLETE] [OBSOLETE] convert a constant polynomial to a number
Mod2Rat [OBSOLETE] [OBSOLETE] reconstruct rationals from modular integers
NewId [OBSOLETE] [OBSOLETE] create a new identifier
NFsAreZero [OBSOLETE] [OBSOLETE] test if normal forms are zero
Option [OBSOLETE] [OBSOLETE] status of a panel option
panel [OBSOLETE] [OBSOLETE] print status of a panel’s options
panels [OBSOLETE] [OBSOLETE] list of CoCoA panels
```
II-9.3. Obsolete and obsolescent functions

PoincareMultiDeg [OBSOLETE] [OBSOLETE]
PoincareShifts [OBSOLETE] [OBSOLETE]
Reset [OBSOLETE] [OBSOLETE] reset panels and random number seed to defaults
ResetPanels [OBSOLETE] [OBSOLETE] reset panels to their default values
RingEnv [OBSOLETE] [OBSOLETE] name of the ring environment
RingSet [OBSOLETE] [OBSOLETE] renamed RingsOf
size [OBSOLETE] [OBSOLETE]
SubalgebraMap [OBSOLETE] [OBSOLETE] algebra homomorphism representing a subalgebra
Unset [OBSOLETE] [OBSOLETE] set and unset panel options
valuation [OBSOLETE] [OBSOLETE]
WLog [OBSOLETE] [OBSOLETE] weighted list of exponents

Some functions are obsolescent, that means that they are still usable but will be deleted in some future version of CoCoA (leaving some time to adapt to the replacing function).

AffHilbert [OBSOLESCENT] [OBSOLESCENT] renamed AffHilbertFn
AffPoincare [OBSOLESCENT] [OBSOLESCENT] Renamed AffHilbertSeries
hilbert [OBSOLESCENT] [OBSOLESCENT] the Hilbert-Poincare’ function
image [OBSOLESCENT] [OBSOLESCENT] apply ring homomorphism
insert [OBSOLESCENT] [OBSOLESCENT] insert an object in a list
MinGensGeneral [OBSOLESCENT] [OBSOLESCENT] renamed MinSubsetOfGens
NewLine [OBSOLESCENT] [OBSOLESCENT] string containing a newline
poincare [OBSOLESCENT] [OBSOLESCENT] the Hilbert-Poincare series
PrimaryPoincare [OBSOLESCENT] [OBSOLESCENT] renamed PrimaryHilbertSeries
rank [OBSOLESCENT] [OBSOLESCENT] rank
RMap [OBSOLESCENT] [OBSOLESCENT] define ring homomorphism for function image
WeightsMatrix [OBSOLESCENT] [OBSOLESCENT] matrix of generalized weights for indeterminates
Chapter II-9. Migrating from CoCoA-4 and keeping up-to-date
Part III

CoCoA datatypes
Chapter III-1

BOOL

III-1.1 Introduction to BOOL

The two BOOL constants are “true” and “false”. (can also be written “TRUE”, “FALSE” and “True”, “False”)
They are mainly used with the commands “if” (I-9.8 pg.118) and “while” (I-23.3 pg.286), etc., inside CoCoA programs.
The relational operators

=  <>  <=  >=

return boolean constants (see “Relational Operators” (II-3.3 pg.302)).

The boolean operators are “and” (I-1.10 pg.29), “or” (I-15.9 pg.199), “IsIn” (I-9.47 pg.136). From version CoCoA-5.0.9 “not” (I-14.27 pg.190) is a function (instead of an operator).

See Also: Relational Operators(II-3.3 pg.302), Commands and Functions for BOOL(III-1.2 pg.327), Commands and Functions returning BOOL(III-1.3 pg.327)

III-1.2 Commands and Functions for BOOL

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>and</td>
<td>boolean ”and” operator</td>
</tr>
<tr>
<td>Bool01</td>
<td>Convert a boolean to an integer</td>
</tr>
<tr>
<td>in</td>
<td>list element selector in list constructor</td>
</tr>
<tr>
<td>not</td>
<td>boolean ”not” operator</td>
</tr>
<tr>
<td>or</td>
<td>boolean ”or” operators</td>
</tr>
<tr>
<td>TmpNBM</td>
<td>Numerical Border Basis of ideal of points</td>
</tr>
</tbody>
</table>

III-1.3 Commands and Functions returning BOOL

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>and</td>
<td>boolean ”and” operator</td>
</tr>
<tr>
<td>Comparison Operators</td>
<td>less than, greater than, ...</td>
</tr>
<tr>
<td>EqSet</td>
<td>checks if the set of elements in two lists are equal</td>
</tr>
<tr>
<td>Equality Test</td>
<td>test whether two values are equal or not</td>
</tr>
<tr>
<td>IsAntiSymmetric</td>
<td>checks if a matrix is anti-symmetric</td>
</tr>
<tr>
<td>IsConstant</td>
<td>checks if a ringelem is in the coefficient ring</td>
</tr>
<tr>
<td>IsContained</td>
<td>checks if A is Contained in B</td>
</tr>
</tbody>
</table>
IsDiagonal: checks if a matrix is diagonal
IsDivisible: checks if A is divisible by B
IsElem: checks if A is an element of B
IsEven, IsOdd: test whether an integer is even or odd
IsFactorClosed: test whether a list of PPs is factor closed
IsField: test whether a ring is a field
IsFiniteField: test whether a ring is a finite field
IsHomog: test whether given polynomials are homogeneous
IsIn: check if one object is contained in another
IsIndet: checks argument is an indeterminante
IsInjective: check if a RINGHOM is injective
IsInRadical: check if a polynomial (or ideal) is in a radical
IsLexSegment: checks if an ideal is lex-segment
IsOne: test whether an object is one
IsPositiveGrading: check if a matrix defines a positive grading
IsPrime: prime integer test
IsProbPrime: checks if an integer is a probable prime
IsPthPower: p-th power test
IsQQ: test whether a ring is the ring of rationals
IsQuotientRing: test whether a ring is a quotient ring
IsStable: checks if an ideal is stable
IsStdGraded: checks if the grading is standard
IsStronglyStable: checks if an ideal is strongly stable
IsSubset: checks if the elements of one list are a subset of another
IsSurjective: check if a RINGHOM is surjective
IsSymmetric: checks if a matrix is symmetric
IsTerm: checks if the argument is a term
IsTermOrdering: check if a matrix defines a term-ordering
IsTrueGCDDomain: test whether a ring is a true GCD domain
IsZero: test whether an object is zero
IsZeroCol, IsZeroRow: test whether a column(row) is zero
IsZeroDim: test whether an ideal is zero-dimensional
IsZeroDivisor: test whether a RINGELEM is a zero-divisor
IsZZ: test whether a ring is the ring of integers
not: boolean "not" operator
or: boolean "or" operators
Chapter III-2

INT

III-2.1 Introduction to INT

There are two types of numbers recognized by CoCoA: integers (type “INT”), rationals (type “RAT”). (CoCoA-4 also had “ZMOD”, but CoCoA-5 can deal with more rings: see “NewRingFp” (I-14.10 pg.183)). Numbers in CoCoA are handled with arbitrary precision. This means that the sizes of numbers are only limited by the amount of available memory. The basic numeric operations—addition (“+”), subtraction (“-”), multiplication (“*”), division (“/”), exponentiation (“^”), and negation (“-”)—behave as one would expect. Be careful, two adjacent minus signs, “--”, start a comment in CoCoA.

```plaintext
/**/ N := 3;
/**/ -N;
-3

--N; <-- THIS IS A COMMENT (not C++ decrement)
```

See Also: Commands and Functions for INT(III-2.2 pg.329), Commands and Functions returning INT(III-2.3 pg.331)

III-2.2 Commands and Functions for INT

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs</td>
<td>absolute value of a number</td>
</tr>
<tr>
<td>AffHilbertFn</td>
<td>the affine Hilbert function</td>
</tr>
<tr>
<td>ascii</td>
<td>convert between characters and ascii code</td>
</tr>
<tr>
<td>AsINT</td>
<td>convert into an INT</td>
</tr>
<tr>
<td>AsRAT</td>
<td>convert into a RAT</td>
</tr>
<tr>
<td>binomial</td>
<td>binomial coefficient</td>
</tr>
<tr>
<td>BinomialRepr, BinomialReprShift</td>
<td>binomial representation of integers</td>
</tr>
<tr>
<td>ContFracToRat</td>
<td>convert continued fraction to rational</td>
</tr>
<tr>
<td>CRT</td>
<td>Chinese Remainder Theorem</td>
</tr>
<tr>
<td>cyclotomic</td>
<td>n-th cyclotomic polynomial</td>
</tr>
<tr>
<td>date</td>
<td>the date</td>
</tr>
<tr>
<td>DecimalStr</td>
<td>convert rational number to decimal string</td>
</tr>
<tr>
<td>den</td>
<td>denominator</td>
</tr>
<tr>
<td>DensePoly</td>
<td>the sum of all power-products of a given degree quotient for integers</td>
</tr>
<tr>
<td>div</td>
<td>matrix for elimination ordering</td>
</tr>
<tr>
<td>ElimMat</td>
<td>evaluate the Hilbert function</td>
</tr>
<tr>
<td>EvalHilbertFn</td>
<td>presentation Ext modules as quotients of free modules</td>
</tr>
<tr>
<td>Ext</td>
<td>factorial function</td>
</tr>
</tbody>
</table>

329
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FactorMultiplicity</td>
<td>multiplicity of a factor of an integer</td>
</tr>
<tr>
<td>first</td>
<td>the first N elements of a list</td>
</tr>
<tr>
<td>flatten</td>
<td>flatten a list</td>
</tr>
<tr>
<td>FloatApprox</td>
<td>approx. of rational number of the form $M \cdot 2^E$</td>
</tr>
<tr>
<td>FloatStr</td>
<td>convert rational number to a decimal string</td>
</tr>
<tr>
<td>format</td>
<td>convert object to formatted string</td>
</tr>
<tr>
<td>GBasisTimeout</td>
<td>compute a Groebner basis with a timeout</td>
</tr>
<tr>
<td>gcd</td>
<td>greatest common divisor</td>
</tr>
<tr>
<td>GCDFreeBasis</td>
<td>determine (minimal) GCD free basis of a set of integers</td>
</tr>
<tr>
<td>GenericPoints</td>
<td>random projective points</td>
</tr>
<tr>
<td>Get</td>
<td>read characters from a device</td>
</tr>
<tr>
<td>GetCol</td>
<td>convert a column of a matrix into a list</td>
</tr>
<tr>
<td>GetRow</td>
<td>convert a row of a matrix into a list</td>
</tr>
<tr>
<td>HilbertFn</td>
<td>the Hilbert function</td>
</tr>
<tr>
<td>IdentityMat</td>
<td>the identity matrix</td>
</tr>
<tr>
<td>incr, decr</td>
<td>increment/decrement a counter</td>
</tr>
<tr>
<td>indent</td>
<td>prints in a more readable way</td>
</tr>
<tr>
<td>insert</td>
<td>[OBSOLESCENT] insert an object in a list</td>
</tr>
<tr>
<td>InverseSystem</td>
<td>Inverse system of an ideal of derivations</td>
</tr>
<tr>
<td>IO.SprintTrunc</td>
<td>convert to a string and truncate</td>
</tr>
<tr>
<td>iroot</td>
<td>integer part of r-th root of an integer</td>
</tr>
<tr>
<td>IsEven, IsOdd</td>
<td>test whether an integer is even or odd</td>
</tr>
<tr>
<td>IsoOne</td>
<td>test whether an object is one</td>
</tr>
<tr>
<td>IsPositiveGrading</td>
<td>check if a matrix defines a positive grading</td>
</tr>
<tr>
<td>IsPrime</td>
<td>prime integer test</td>
</tr>
<tr>
<td>IsProbPrime</td>
<td>checks if an integer is a probable prime</td>
</tr>
<tr>
<td>isqrt</td>
<td>(truncated) square root of an integer</td>
</tr>
<tr>
<td>IsZero</td>
<td>test whether an object is zero</td>
</tr>
<tr>
<td>IsZeroCol, IsZeroRow</td>
<td>test whether a column(row) is zero</td>
</tr>
<tr>
<td>last</td>
<td>the last N elements of a list</td>
</tr>
<tr>
<td>lcm</td>
<td>least common multiple</td>
</tr>
<tr>
<td>LexMat</td>
<td>matrices for std. term-orderings</td>
</tr>
<tr>
<td>LogToTerm</td>
<td>returns a monomial (power-product) with given exponents</td>
</tr>
<tr>
<td>MakeMatByRows, MakeMatByCols</td>
<td>convert a list into a matrix</td>
</tr>
<tr>
<td>MantissaAndExponent10</td>
<td>convert rational number to a float</td>
</tr>
<tr>
<td>MantissaAndExponent2</td>
<td>convert rational number to a binary float</td>
</tr>
<tr>
<td>max</td>
<td>a maximum element of a sequence or list</td>
</tr>
<tr>
<td>min</td>
<td>a minimum element of a sequence or list</td>
</tr>
<tr>
<td>minors</td>
<td>list of minor determinants of a matrix</td>
</tr>
<tr>
<td>mod</td>
<td>remainder for integers</td>
</tr>
<tr>
<td>NewFreeModule</td>
<td>create a new FreeModule</td>
</tr>
<tr>
<td>NewList</td>
<td>create a new list</td>
</tr>
<tr>
<td>NewMat</td>
<td>Zero matrix</td>
</tr>
<tr>
<td>NewMatFilled</td>
<td>matrix filled with value</td>
</tr>
<tr>
<td>NewPolyRing</td>
<td>create a new PolyRing</td>
</tr>
<tr>
<td>NewRingFp</td>
<td>create a new finite field</td>
</tr>
<tr>
<td>NewRingTwinFloat</td>
<td>create a new twin-float ring</td>
</tr>
<tr>
<td>NextPrime</td>
<td>find the next largest prime number</td>
</tr>
<tr>
<td>NextProbPrime</td>
<td>find the next largest probable prime number</td>
</tr>
<tr>
<td>num</td>
<td>number of partitions of an integer</td>
</tr>
<tr>
<td>NumPartitions</td>
<td>number of partitions of an integer</td>
</tr>
<tr>
<td>operators, shortcuts</td>
<td>Special characters equivalent to commands</td>
</tr>
<tr>
<td>partitions</td>
<td>partitions of an integer</td>
</tr>
<tr>
<td>PowerMod</td>
<td>compute a modular power efficiently</td>
</tr>
<tr>
<td>PrimitiveRoot</td>
<td>find a primitive root modulo a prime</td>
</tr>
</tbody>
</table>
III-2.3 Commands and Functions returning INT

- **product**: the product of the elements of a list
- **random**: random integer
- **randomized**: randomize the coefficients of a given polynomial
- **RandomSubset**: random subset
- **RandomSubsetIndices**: indices for random subset
- **RandomTuple**: random tuple
- **RandomTupleIndices**: indices for random tuples
- **RatReconstructByContFrac, RatReconstructByLattice**: rational reconstruction from modular image
- **RatReconstructWithBounds**: deterministic rational reconstruction from modular image
- **RefineGCDFreeBasis**: refine an integer GCD free basis
- **remove**: remove an object in a list
- **RevLexMat**: matrices for std. term-orderings
- **RingElem**: convert an expression into a RINGELEM
- **RingQQt**: pre-defined polynomial rings
- **ScientificStr**: convert integer/rational to a floating-point string
- **seed**: seed for “random”
- **seed for “random”**: set a list as a row into a matrix
- **SourceRegion**: read commands from a region in a file
- **sign**: the sign of a number
- **SmoothFactor**: find small prime factors of an integer
- **SourceRegion**: read commands from a region in a file
- **Spaces**: return a string of spaces
- **StarPrint, StarSprint**: print polynomial with *’s for multiplications
- **StdDegLexMat**: matrices for std. term-orderings
- **StdDegRevLexMat**: matrices for std. term-orderings
- **submat**: submatrix
- **subsets**: returns all sublists of a list
- **sum**: the sum of the elements of a list
- **swap two rows in a matrix**: swap two rows in a matrix
- **Syzygy**: range of symbols for the indeterminates of a PolyRing
- **tuples**: N-tuples
- **WithoutNth**: removes the N-th component from a list
- **XelMat**: matrices for std. term-orderings
- **ZeroMat**: matrix filled with 0

### III-2.3 Commands and Functions returning INT

- **abs**: absolute value of a number
- **AffHilbertFn**: the affine Hilbert function
- **AsINT**: convert into an INT
- **Binomial**: binomial coefficient
- **BinomialRepr, BinomialReprShift**: binomial representation of integers
- **Bool01**: Convert a boolean to an integer
- **ceiling**: round rational up to integer
- **characteristic**: the characteristic of a ring
- **ContinuedFraction**: continued fraction quotients
- **count**: count the objects in a list
- **deg**: the standard degree of a polynomial or moduleelem
- **den**: denominator
- **Depth**: Depth of a module
- **dim**: the dimension of a ring or quotient object
div quotient for integers
EvalHilbertFn evaluate the Hilbert function
factorial factorial function
FactorMultiplicity multiplicity of a factor of an integer
floor round rational down to integer
gcd greatest common divisor
GradingDim Number of components in weighted degree
HilbertFn the Hilbert function
ILogBase integer part of the logarithm
IndetIndex index of an indeterminate
iroot integer part of r-th root of an integer
isqrt (truncated) square root of an integer
lcm least common multiple
len the length of an object
LogCardinality extension degree of a finite field
LPosn the position of the leading power-product in a ModuleElem
MayerVietorisTreeN1 N-1st Betti multidegrees of monomial ideals using Mayer-Vietoris trees
MinPowerInIdeal the minimum power of a polynomial is an ideal
mod remainder for integers
multiplicity the multiplicity (degree) of a ring or quotient object
NextPrime find the next largest prime number
NextProbPrime find the next largest probable prime number
num numerator
NumCols number of columns in a matrix
NumCompts the number of components
NumIndets number of indeterminates
NumPartitions number of partitions of an integer
NumRows number of rows in a matrix
NumTerms number of terms in a polynomial
PowerMod compute a modular power efficiently
PrimitiveRoot find a primitive root modulo a prime
random random integer
randomized randomize the coefficients of a given polynomial
reg Castelnuovo-Mumford regularity of a module
RegularityIndex regularity index of a Hilbert function or series
RingID identification for ring
rk rank of a matrix or module
RootBound bound on roots of a polynomial over QQ
round round to integer
seed seed for “random”
sign the sign of a number
TimeOfDay the current time
UnivariateIndetIndex the index of the indeterminate of a univariate polynomial
Chapter III-3

RAT

III-3.1 Introduction to RAT

Rational numbers can be entered as fractions or as terminating decimals. CoCoA always converts a rational number into a fraction in lowest terms.

```coconut
/**/ 3.8;
19/5
/**/ N := 4/8; N;
1/2
/**/ type(N);
RAT
```

See Also: Commands and Functions for RAT(III-3.2 pg.333), Commands and Functions returning RAT(III-3.3 pg.334)

III-3.2 Commands and Functions for RAT

- abs: absolute value of a number
- AsINT: convert into an INT
- AsRAT: convert into a RAT
- ceil: round rational up to integer
- CFApprox: continued fraction approximation
- CFApproximants: continued fraction approximants
- ContFrac: continued fraction quotients
- DecimalStr: convert rational number to decimal string
- den: denominator
- FloatApprox: approx. of rational number of the form $M \times 2^E$
- FloatStr: convert rational number to a decimal string
- floor: round rational down to integer
- ILogBase: integer part of the logarithm
- IsOne: test whether an object is one
- IsZero: test whether an object is zero
- MantissaAndExponent10: convert rational number to a float
- MantissaAndExponent2: convert rational number to a binary float
- max: a maximum element of a sequence or list
- min: a minimum element of a sequence or list
- NewMatFilled: matrix filled with value
- num: numerator

333
Chapter III-3. RAT

- **product**: the product of the elements of a list
- **RealRootRefine**: refine a real root of a univariate polynomial
- **RealRoots**: computes the real roots of a univariate polynomial
- **RealRootsApprox**: computes approximations to the real roots of a univariate polynomial
- **RingElem**: convert an expression into a RINGELEM
- **round**: round to integer
- **ScientificStr**: convert integer/rational to a floating-point string
- **sign**: the sign of a number
- **SimplestRatBetween**: find simplest rational in a closed interval
- **StableBBasis5**: Stable Border Basis of ideal of points
- **sum**: the sum of the elements of a list
- **TimeFrom**: time elapsed since a given moment

### III-3.3 Commands and Functions returning RAT

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs</td>
<td>absolute value of a number</td>
</tr>
<tr>
<td>ApproxSolve</td>
<td>Approximate real solutions for polynomial system</td>
</tr>
<tr>
<td>AsRAT</td>
<td>convert into a RAT</td>
</tr>
<tr>
<td>CFApprox</td>
<td>continued fraction approximation</td>
</tr>
<tr>
<td>CFApproximants</td>
<td>continued fraction approximants</td>
</tr>
<tr>
<td>ContFracToRat</td>
<td>convert continued fraction to rational</td>
</tr>
<tr>
<td>CpuTime</td>
<td>Counts cpu time</td>
</tr>
<tr>
<td>FloatApprox</td>
<td>approx. of rational number of the form $M \times 2^E$</td>
</tr>
<tr>
<td>SimplestRatBetween</td>
<td>find simplest rational in a closed interval</td>
</tr>
</tbody>
</table>
Chapter III-4

STRING

III-4.1 String Literals

A string literal consists of a sequence of characters between double quotes ("...").

```coconut
/**/ PrintLn "The primes up to 10 are ", [n in 1..10 | IsPrime(n)];
The primes up to 10 are [2, 3, 5, 7]

/**/ Print "The quick brown fox", "jumped over the lazy dog.");
The quick brown fox jumped over the lazy dog
```

To put special characters in CoCoA string literals use the appropriate "escape sequence". Here is a summary: "\"" produces a double-quote character; "\n" produces a newline character; "\" produces a backslash character; "\t" produces a TAB character; "\r" produces a carriage-return character.

```coconut
/**/ Print "line 1\nline 2";
line 1
line 2

/**/ Print "A string containing \"quote marks\".";
A string containing "quote marks".
```

WARNING: CoCoA still accepts an "obsolete" syntax for string literals (between single-quotes); do not use this!

See Also: String Operations(III-4.2 pg.335), sprint(I-19.24 pg.257), Commands and Functions for STRING(III-4.3 pg.336), Commands and Functions returning STRING(III-4.4 pg.336)

III-4.2 String Operations

CoCoA offers only a few operations on strings: length, concatenation, comparison, substring containment and indexing.

```coconut
/**/ str := "Hello" + "World!";  --> string concatenation
/**/ Print str;
HelloWorld!

/**/ len(str);  --> length in characters
11

/**/ "Abc" < str;  --> lexicographical comparison
true
```
The operator “IsIn” (I-9.47 pg.136) can be used to test if one string is a substring of another.

```plaintext
example

/**/ str[1]; --> character indexing, indexes start from 1
/**/ str[1];
H

The operator “IsIn” (I-9.47 pg.136) can be used to test if one string is a substring of another.

```plaintext
example

/**/ mesg := "Banana";
/**/ "ana" IsIn mesg;
true
/**/ "Ana" IsIn mesg; --> substring must be an exact match
false

See Also: String Literals(III-4.1 pg.335), ascii(I-1.14 pg.31), concat(I-3.33 pg.54), IsIn(I-9.47 pg.136), len(I-12.5 pg.157)

III-4.3 Commands and Functions for STRING

ascii convert between characters and ascii code
error throw an error message
GetEnv access shell variables
gin generic initial ideal
ImplicitPlotOn outputs the zero locus of a bivariate polynomial to a file
indets list of indeterminantes in a PolyRing
IsIn check if one object is contained in another
len the length of an object
max a maximum element of a sequence or list
min a minimum element of a sequence or list
NewPolyRing create a new PolyRing
OpenIFile open input file
OpenIString open input string
OpenOFile open output file
OpenOString open output string
OpenSocket open a socket connection
operators, shortcuts Special characters equivalent to commands
PlotPointsOn outputs the coordinates of the points to a file
protect protect a variable from being overwritten
ReadExpr Read RINGELEM expression from string
RingElem convert an expression into a RINGELEM
source read commands from a file or device
SourceRegion read commands from a region in a file
starting list functions starting with a given string
SymbolRange range of symbols for the indeterminates of a PolyRing
tagged tag an object for pretty printing

III-4.4 Commands and Functions returning STRING

ascii convert between characters and ascii code
CocoaPackagePath returns the path to the CoCoA packages
DecimalStr convert rational number to decimal string
ExternalLibs Linked external libraries
III-4.4. Commands and Functions returning STRING

- FloatStr: convert rational number to a decimal string
- format: convert object to formatted string
- GetEnv: access shell variables
- GetErrMesg: returns the message associated with an error
- IndetName: the name of an indeterminate
- IO.SprintTrunc: convert to a string and truncate
- LaTeX: LaTeX formatting
- NewLine [OBSOLESCENT]: [OBSOLESCENT] string containing a newline
- Packages: list of loaded packages
- PkgName: returns the name of a package
- ScientificStr: convert integer/rational to a floating-point string
- spaces: return a string of spaces
- sprint: convert to a string
- StarPrint, StarSprint: print polynomial with *'s for multiplications
- tag: returns the tag string of an object
- TimeFrom: time elapsed since a given moment
Chapter III-5

LIST

III-5.1 Introduction to LIST

A CoCoA list is a sequence of CoCoA objects between square brackets. See also “List Constructors” (III-5.2 pg.340).

In particular, a list may contain other lists. The empty list is “[]”. If “L” is a list and “N” is an integer, then “L[N]” is the “N”-th component of “L”.

If “L” contains sublists, then “L[N_1, N_2, ..., N_s]” is shorthand for “L[N_1][N_2]...[N_s]” (see the example below).

Lists are often used to build structured objects of type “MAT”, “MODULEELEM”, “IDEAL”, and “MODULE”.

```coconut
/**/ Use R ::= QQ[t,x,y,z];
/**/ L := [34*x+y^2, "a string", [], [True, False]]; -- a list
/**/ L[1]; -- the 1st component
y^2 +34*x
/**/ L[2];
a string
/**/ L[3];
[ ]
/**/ L[4]; -- The 4th component is a list, itself;
[true, false]
/**/ L[4][1]; -- its 1st component;
true
/**/ L[4,1]; -- the same.
true
/**/ [1,"a"]+[2,"b"]; -- NOTE: one may add lists if their components are
[3, "ab"] -- compatible (see "Algebraic Operators").
/**/ L := [x^2-y, t*y^2-z^3];
/**/ I := ideal(L);
/**/ I;
ideal(x^2 -y, t*y^2 -z^3)
```

See Also: List Constructors(III-5.2 pg.340), Commands and Functions for LIST(III-5.3 pg.340), Commands and Functions returning LIST(III-5.4 pg.342)
III-5.2 List Constructors

These operators create new lists.

A..B
[A,B,C,...]
[X in L: LIST | B: BOOL]: LIST
[E:expression | X in L]: LIST
[E:expression | X in L: LIST and B: BOOL]: LIST

“A..B” creates the list of integers from “A” to “B”, both ends are included.

“[A,B,C,...]” makes a list containing “A”, “B”, “C” and so on, in that order.

“[X in L | B]” makes a list of those elements in “L” for which condition “B” is true.

“[E | X in L]” evaluates the expression “E” for each “X” in “L”, and collects the results in a new list.

“[E | X in L and B]” evaluates the expression “E” for each “X” in “L” which satisfies the condition “B”, and collects the results in a new list.

```plaintext
example
/**/ [];  --> empty list
[]
/**/ 1..4;
[1, 2, 3, 4]
/**/ [3,1,4,2];
[3, 1, 4, 2]
/**/ [N in 1..10 | IsPrime(N)];
[2, 3, 5, 7]
/**/ [N^2 | N in 1..4];
[1, 4, 9, 16]
/**/ [N^2 | N in 1..10 and IsPrime(N)];
[4, 9, 25, 49]
```
III-5.3. Commands and Functions for LIST

diff returns the difference between two lists
distrib the distribution of objects in a list
DivAlg division algorithm
elim eliminate variables
ElimMat matrix for elimination ordering
EqSet checks if the set of elements in two lists are equal
eval substitute numbers or polynomials for indeterminates
Ext presentation Ext modules as quotients of free modules
FGLM5 perform a FGLM Groebner Basis conversion
first the first N elements of a list
flatten flatten a list
foreach loop command
FVector compute the f-vector of a top simplices list
GBM intersection of ideals for zero-dimensional schemes
gcd greatest common divisor
GCDFreeBasis determine (minimal) GCD free basis of a set of integers
HGBM intersection of ideals for zero-dimensional schemes
HilbertSeriesShifts the Hilbert-Poincare series
homog homogenize with respect to an indeterminate
HomogElimMat matrix for elimination ordering
ideal ideal generated by list
IdealAndSeparatorsOfPoints ideal and separators for affine points
IdealAndSeparatorsOfProjectivePoints ideal and separators for points
IdealOfProjectivePoints ideal of a set of projective points
implicit implicitization
Interpolate interpolating polynomial
interreduce, interreduced interreduce a list of polynomials
intersection intersect lists, ideals, or modules
IntersectList intersect lists, ideals, or modules
IsFactorClosed test whether a list of PPs is factor closed
IsHomog test whether given polynomials are homogeneous
IsIn check if one object is contained in another
IsSubset checks if the elements of one list are a subset of another
IsTree5 checks if a facet complex is a tree
jacobian the Jacobian of a list of polynomials
last the last N elements of a list
lcm least common multiple
len the length of an object
LexSegmentIdeal lex-segment ideal containing L, or with the same Hilbert fn as I
LogToTerm returns a monomial (power-product) with given exponents
MakeMatByRows, MakeMatByCols convert a list into a matrix
MakeSet remove duplicates from a list
matrix convert a list into a matrix
max a maximum element of a sequence or list
min a minimum element of a sequence or list
ModuleElem create a module element
NewPolyRing create a new PolyRing
NmzComputation flexible access to Normaliz
NmzEhrhartRing Computes the Ehrhart ring
NmzIntClosureMonIdeal integral closure of a monomial ideal
NmzIntClosureToricRing integral closure of a toric ring
NmzNormalToricRing normalization of a toric ring
III-5.4 Commands and Functions returning LIST

NonZero
remove zeroes from a list
NR
normal reduction
operators, shortcuts
Special characters equivalent to commands
permutations
returns all permutations of the entries of a list
PlotPoints
outputs the coordinates of the points to a file
PlotPointsOn
outputs the coordinates of the points to a file
PolyAlgebraHom
homomorphism of polynomial algebras
PolyRingHom
homomorphism of polynomial rings
PrintBettiDiagram
the diagram of the graded Betti numbers
product
the product of the elements of a list
QZP
change field for polynomials and ideals
RandomSubset
random subset
RandomTuple
random tuple
RationalAffinePoints
Affine rational solutions
RationalProjectivePoints
Projective rational solutions
RationalSolve
Rational solutions for polynomial system
RatReconstructWithBounds
deterministic rational reconstruction from modular image
RefineGCDFreeBasis
refine an integer GCD free basis
remove
remove an object in a list
reverse, reversed
reverse a list
RingsOf
list of the rings of an object
RMap [OBSOLESCENT]
[OBSOLESCENT] define ring homomorphism for function image
RowMat
single row matrix
ScalarProduct
scalar product
SeparatorsOfPoints
separators for affine points
SeparatorsOfProjectivePoints
separators for projective points
SetRow
set a list as a row into a matrix
shape
extended list of types involved in an expression
SimplexInfo
compute the Stanley-Reisner ideal, the Alexander Dual complex And ideal of a top simplices list
sort
sort a list
sortBy
sort a list
sorted
sort a list
SortedBy
sort a list
StableBBasis5
Stable Border Basis of ideal of points
StableIdeal
stable ideal containing L
StronglyStableIdeal
strongly stable ideal containing L
SubalgebraRepr
representation of a polynomial as a subalgebra element
submat
submatrix
submodule
submodule generated by list
subsets
returns all sublists of a list
sum
the sum of the elements of a list
SymbolRange
range of symbols for the indeterminates of a PolyRing
syz
syzygy modules
tail
remove the first element of a list
TmpNBM
Numerical Border Basis of ideal of points
toric
saturate toric ideals
tuples
N-tuples
WithoutNth
removes the N-th component from a list
ZPQ
change field for polynomials and ideals
apply
  apply homomorphism
ApproxSolve
  Approximate real solutions for polynomial system
ascii
  convert between characters and ascii code
BBasis5
  Border Basis of zero dimensional ideal
BinomialRepr, BinomialReprShift
  binomial representation of integers
CartesianProduct, CartesianProductList
  Cartesian product of lists
CFApproximants
  continued fraction approximants
coefficients
  list of coefficients of a polynomial
CoefficientsWRT
  list of coeffs and PPs of a polynomial wrt an indet or a list of indets
CoeffListWRT
  list of coefficients of a polynomial wrt an indet
compts
  list of components of a ModuleElem
concat
  concatenate lists
ConcatLists
  concatenate a list of lists
ContFrac
  continued fraction quotients
CurrentTypes
  lists all data types
diff
  returns the difference between two lists
distrib
  the distribution of objects in a list
eigenvectors
  eigenvalues and eigenvectors of a matrix
EquiIsoDec
  equidimensional isoradical decomposition
ExternalLibs
  Linked external libraries
FGLM5
  perform a FGLM Groebner Basis conversion
fields
  list the fields of a record
flatten
  flatten a list
FrbaAlexanderDual
  Alexander Dual of monomial ideals
FrbaAssociatedPrimes
  Associated primes of monomial ideals
FrbaIrreducibleDecomposition
  Irreducible decomposition of monomial ideals
FrbaMaximalStandardMonomials
  Maximal standard monomials of monomial ideals
FrbaPrimaryDecomposition
  Primary decomposition of monomial ideals
GBasis
  calculate a Groebner basis
GBasisTimeout
  compute a Groebner basis with a timeout
GCDFreeBasis
  determine (minimal) GCD free basis of a set of integers
GenericPoints
  random projective points
GenRepr
  representation in terms of generators
gens
  list of generators of an ideal
Get
  read characters from a device
GetCol
  convert a column of a matrix into a list
GetCols
  convert a matrix into a list of lists
GetRow
  convert a row of a matrix into a list
GetRows
  convert a matrix into a list of lists
HilbertBasisKer
  Hilbert basis for a monoid
homog
  homogenize with respect to an indeterminate
HVector
  the h-vector of a module or quotient object
in
  list element selector in list constructor
indets
  list of indeterminates in a PolyRing
IndetSubscripts
  the index of an indeterminate
interreduce, interreduced
  interreduce a list of polynomials
intersection
  intersect lists, ideals, or modules
IntersectList
  intersect lists, ideals, or modules
InverseSystem
  Inverse system of an ideal of derivations
JanetBasis
  the Janet basis of an ideal
LinkerBasis
  find the kernel of a matrix
log
  the list of exponents of the leading term of a polynomial
MakeSet
  remove duplicates from a list
MinGens
  list of minimal generators
minors
  list of minor determinants of a matrix
MinSubsetOfGens
  list of minimal generators
monomials
  the list of monomials of a polynomial
NewList
NmzDiagInvariants
NmzEhrhartRing
NmzFiniteDiagInvariants
NmzIntClosureMonIdeal
NmzIntClosureToricRing
NmzIntersectionValRings
NmzNormalToricRing
NmzTorusInvariants
NonZero
Packages
partitions
permutations
PrimaryDecomposition
PrimaryDecomposition0
PrimaryDecompositionGTZ0
QuotientBasis
QZP
RandomSubset
RandomSubsetIndices
RandomTuple
RandomTupleIndices
RationalAffinePoints
RationalProjectivePoints
RationalSolve
RealRoots
RealRootsApprox
ReducedGBasis
RefineGCDFreeBasis
res
reverse, reversed
RingsOf
SeparatorsOfPoints
SeparatorsOfProjectivePoints
shape
sorted
SortedBy
starting
subsets
support
SymbolRange
tail
TopLevelFunctions
tuples
wdeg
WithoutNth
ZPQ

create a new list
ring of invariants of a diagonalizable group action
Computes the Ehrhart ring
ring of invariants of a finite group action
integral closure of a monomial ideal
integral closure of a toric ring
intersection of ring of valuations
normalization of a toric ring
ring of invariants of torus action
remove zeroes from a list
list of loaded packages
partitions of an integer
returns all permutations of the entries of a list
primary decomposition of an ideal
primary decomposition of a 0-dimensional ideal
primary decomposition of a 0-dimensional ideal
vector space basis for zero-dimensional quotient rings
change field for polynomials and ideals
random subset
indices for random subset
random tuple
indices for random tuples
Affine rational solutions
Projective rational solutions
Rational solutions for polynomial system
computes the real roots of a univariate polynomial
computes approximations to the real roots of a univariate polynomial
compute reduced Groebner basis
refine an integer GCD free basis
free resolution
reverse a list
list of the rings of an object
separators for affine points
separators for projective points
extended list of types involved in an expression
sort a list
sort a list
list functions starting with a given string
returns all sublists of a list
the list of terms of a polynomial or moduleelem
range of symbols for the indeterminates of a PolyRing
remove the first element of a list
returns the functions available at top-level
N-tuples
multi-degree of an polynomial
removes the N-th component from a list
change field for polynomials and ideals
Chapter III-6

RECORD

III-6.1 Introduction to RECORD

A record is a data type in CoCoA representing a list of bindings of the form “name to object”.

```plaintext
/**/ Use R ::= QQ[x,y,z];
/**/ P := record[ I := ideal(x,y^2-z), F := x^2 + y, Misc := [1,3,4]];  
/**/ P.I; 
ideal(x, y^2 -z)
/**/ P["I"]; 
ideal(x, y^2 -z)
/**/ P.Misc; 
[1, 3, 4]
/**/ P.Misc[2];
3
/**/ P.Date := "1/1/98";
/**/ indent(P);
record[
   Date := "1/1/98",
   F := x^2 +y,
   I := ideal(x, y^2 -z),
   Misc := [1, 3, 4]
]
/**/ P["Misc",3]; -- equivalent to P.Misc[3]
4
```

Each entry in a record is called a “field”. Note that records are “open” in the sense that their fields can be extended, as shown in the previous example. At present, there is no function for deleting fields from a record, one must rewrite the record, selecting the fields to retain:

```plaintext
/**/ P := record[A := 2, B := 3, C := 5, D := 7];
/**/ Q := record[];

Foreach F In Fields(P) Do 
   If F <> "C" Then Q[F] := P[F]; EndIf; 
EndForeach;
/**/ P := Q;
```

345
/**/ P;
record[A := 2, B := 3, D := 7]

See Also: Commands and Functions for RECORD(III-6.2 pg.346), Commands and Functions returning RECORD(III-6.3 pg.346)

### III-6.2 Commands and Functions for RECORD

- **CoefficientsWRT**
  - list of coeffs and PPs of a polynomial wrt an indet or a list of indets
- **fields**
  - list the fields of a record
- **NmzComputation**
  - flexible access to Normaliz
- **PrintBettiDiagram**
  - the diagram of the graded Betti numbers
- **RealRootRefine**
  - refine a real root of a univariate polynomial
- **record field selector**
  - select a field of a record
- **shape**
  - extended list of types involved in an expression

### III-6.3 Commands and Functions returning RECORD

- **AlmostQR**
  - QR decomposition of a matrix
- **CocoaLimits**
  - limits on exponents and ring characteristics
- **ContentFreeFactor**
  - factorization of multivariate polynomial into content-free factors
- **CRT**
  - Chinese Remainder Theorem
- **DivAlg**
  - division algorithm
- **eigenvectors**
  - eigenvalues and eigenvectors of a matrix
- **factor**
  - factor a polynomial
- **FVector**
  - compute the f-vector of a top simplices list
- **IdealAndSeparatorsOfPoints**
  - ideal and separators for affine points
- **IdealAndSeparatorsOfProjectivePoints**
  - ideal and separators for points
- **IndetSymbols**
  - the names of the indeterminates in a PolyRing
- **LinearSimplify**
  - simplifying linear substitution for a univariate polynomial over QQ
- **MantissaAndExponent10**
  - convert rational number to a float
- **MantissaAndExponent2**
  - convert rational number to a binary float
- **NmzComputation**
  - flexible access to Normaliz
- **PreImage**
  - preimage of a RINGELEM
- **PreprocessPts**
  - Reduce redundancy in a set of approximate points
- **PrimaryDecomposition0**
  - primary decomposition of a 0-dimensional ideal
- **PrimaryDecompositionGTZ0**
  - primary decomposition of a 0-dimensional ideal
- **RatReconstructByContFrac, RatReconstructByLattice**
  - rational reconstruction from modular image
- **RatReconstructWithBounds**
  - deterministic rational reconstruction from modular image
- **RealRootRefine**
  - refine a real root of a univariate polynomial
- **record**
  - create a record
- **shape**
  - extended list of types involved in an expression
- **SimplexInfo**
  - compute the Stanley-Reisner ideal, the Alexander Dual complex
- **SimplicialHomology**
  - compute the simplicial homology of a top simplices list
- **SmoothFactor**
  - find small prime factors of an integer
- **SqFreeFactor**
  - compute a squarefree factorization
- **StableBBasis5**
  - Stable Border Basis of ideal of points
- **starting**
  - list functions starting with a given string
- **SubalgebraRepr**
  - representation of a polynomial as a subalgebra element
- **TmpNBM**
  - Numerical Border Basis of ideal of points
VersionInfo

version and info about CoCoA
Chapter III-7

FUNCTION

III-7.1 Introduction to FUNCTION

The most important construct in CoCoA programming is the user-defined function. These functions take parameters, perform CoCoA commands, and return values. Collections of functions can be stored in text files and read into CoCoA sessions using “source” (I-19.21 pg.256). To prevent name conflicts of the type that are likely to arise if functions are to be made available for use by others, the functions can be collected in “packages”.

To learn about user functions, look up “define” (I-4.4 pg.64) (online, enter “?define”).

III-7.2 FUNCTIONs are first class objects

In CoCoA-5 functions are “first class objects”, and so may be passed like any other value.

```coconut
/**/ Define MyMax(LessThan, X, Y)
/**/ If LessThan(X, Y) Then Return Y; Else Return X; EndIf;
/**/ EndDefine;

-- Let’s use MyMax by giving two different orderings.
/**/ Define CompareLT(X, Y) Return LT(X) < LT(Y); EndDefine;
/**/ Define CompareLC(X, Y) Return LC(X) < LC(Y); EndDefine;
/**/ Use R ::= QQ[x,y,z];
/**/ MyMax(CompareLC, 3*x-y, 5*z-2);
5*z -2
/**/ MyMax(CompareLT, 3*x-y, 5*z-2);
3*x -y
```

III-7.3 Commands and Functions for FUNCTION

- `ref` passing function parameters by reference
- `return` exit from a function
- `SortBy` sort a list
- `SortedBy` sort a list

349
TopLevel  make a top level variable accessible

III-7.4 Commands and Functions returning FUNCTION

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>define</td>
<td>define a function</td>
</tr>
<tr>
<td>func</td>
<td>Anonymous function</td>
</tr>
<tr>
<td>TopLevelFunctions</td>
<td>returns the functions available at top-level</td>
</tr>
</tbody>
</table>
Chapter III-8

TYPE

III-8.1 Commands and Functions for TYPE

\texttt{describe} \quad \text{information about an object}

III-8.2 Commands and Functions returning TYPE

\begin{tabular}{ll}
\textbf{CurrentTypes} & lists all data types \\
\textbf{shape} & extended list of types involved in an expression \\
\textbf{type} & the data type of an expression \\
\end{tabular}
Chapter III-9

RING

III-9.1 Introduction to RING

Rings, and especially polynomial rings, play a central role in CoCoA.

The user can define many rings, but at any time a “current ring” is active within the system.

Once a ring has been defined, the system can handle the following mathematical objects defined over that ring:

* elements of the ring
* ideals
* matrices
* lists of objects
* modules (submodules of a free module)
* rings

Variables containing ring-dependent objects such as polynomials, ideals, and matrices are “labeled” by their ring. Any operation on them is performed in their ring, independently of what the current ring is.

See Also: Polynomial Rings(III-9.2 pg.353), Commands and Functions for RING(III-9.8 pg.356), Commands and Functions returning RING(III-9.9 pg.357)

III-9.2 Polynomial Rings

CoCoA starts with the default (polynomial) ring “R = QQ[x,y,z]”. Polynomial rings are created with the function “NewPolyRing” (I-14.8 pg.182), but there is a special simplifyied syntax working in most cases: it must be preceded by the command “use” (I-21.5 pg.280) or by the symbol “::=” (or both)

\[
R ::= C[X:INDETS]; \quad \text{use } C[X:INDETS]; \\
R ::= C[X:INDETS], 0; \quad \text{use } C[X:INDETS], 0;
\]

“R” is the identifier of a CoCoALanguage variable, “C” is a RING, “X” is an expression that defines the indeterminates, “0” is a pre-defined ordering (“lex”, “deglex”, “degrevlex”). The default ordering is DegRevLex.

After the ring is defined using the above syntax, it can be made to be the current ring with the command “use” (I-21.5 pg.280).

```plaintext
/**/ Use R ::= QQ[a,b,c]; -- define and use the ring R
/**/ K := NewFractionField(R);
/**/ S ::= K[x,y], Lex;
/**/ CurrentRing; -- the current ring is still R
RingWithID(21, "QQ[a,b,c]"
/**/ Use S; -- now the ring S is the current ring
```
III-9.3 Coefficient Rings

The coefficient ring for a CoCoA polynomial ring may be any ring “R”:

1. \( \mathbb{Z} \): (arbitrarily large) integer numbers;
2. \( \mathbb{Q} \): (arbitrarily large) rational numbers;
3. \( \mathbb{Z}/(N) \);
4. \( R[a,b,c] \);
5. \( K(a,b,c) \);

The first two types of coefficients are based on the GNU-gmp library. Some operations work only when coefficients are in a field (a meaningful error message will be thrown). NOTE: inside “\( \text{define}/\text{enddefine} \)” the top-level variables “\( \mathbb{Z} \)” (I-25.4 pg.292) and “\( \mathbb{Q} \)” (I-17.1 pg.217) are not directly visible. Use “RingZZ()” or “RingQQ()” instead (or import them with “TopLevel” (I-20.9 pg.274)).

See Also: CoeffRing(I-3.25 pg.51)

III-9.4 Indeterminates

An “indeterminate” is represented by an identifier followed by one or more integer indices. For example, “\( x \)”, “\( \alpha[1] \)”, “\( x[1,2,3] \)” are legal (and different) indeterminates, as is “\( x[i, 2*i+1] \)” if “\( i \)” is of type “\( \text{INT} \)”.

When creating a ring the indeterminates are listed separate by commas.

See Also: CoeffRing(I-3.25 pg.51)

III-9.5 Orderings

Polynomials are always sorted with respect to the ordering of their base ring. All the operations involving polynomials utilize and preserve this ordering. The user can define custom orderings or choose a predefined term-ordering (see “NewPolyRing” (I-14.8 pg.182)).

The predefined term-orderings are:

* standard-degree reverse lexicographic: “DegRevLex” (default)
* standard-degree lexicographic: “DegLex”
* pure lexicographic: “Lex” (no grading)
* pure xel: “Xel” (NOT YET IMPLEMENTED)

If the indeterminates are given in the order \(x_1, \ldots, x_n\), then \(x_1 > \ldots > x_n\) with respect to Lex, and \(x_1 < \ldots < x_n\) with respect to Xel.

See Also: OrdMat(I-15.10 pg.199), elim(I-5.3 pg.76)

### III-9.6 Module Orderings

***** NOT YET UPDATED TO CoCoA-5 *****

First we recall the definition of a module term-ordering. We assume that all our free modules have finite rank and are of the type \(M = R^r\) where \(R\) is a polynomial ring with \(n\) indeterminates. Let \([e_i | i = 1, \ldots, r]\) be the canonical basis of \(M\). A “term” of \(M\) is an element of the form \(Te_i\), where \(T\) belongs to the set \(T(R)\) of the terms of \(R\). Hence the set \(T(M)\), of the terms of \(M\), is in one-to-one correspondence with the Cartesian product, \(T(R) \times [1, \ldots, r]\).

A “module term-ordering” is defined as a total ordering \(>\) on \(T(M)\) such that for all \(T, T_1, T_2\) in \(T(R)\), with \(T\) not equal to 1, and for all \(i, j\) in \(1, \ldots, r\),

1. \(T \cdot T_1 \cdot e_i > T_1 \cdot e_i\)
2. \(T_1 \cdot e_i > T_2 \cdot e_j \Rightarrow T \cdot T_1 \cdot e_i > T \cdot T_2 \cdot e_j\)

Each term-ordering on the current ring induces several term-orderings on a free module. CoCoA allows the user to choose between the following:

* the ordering called “ToPos” (which is the default one) defined by:

\[
T_1 \cdot e_i > T_2 \cdot e_j \Leftrightarrow T_1 > T_2 \text{ in } R \\
or, \text{ if } T_1 = T_2, i < j
\]

* the ordering called “PosTo” defined by:

\[
T_1 \cdot e_i > T_2 \cdot e_j \Leftrightarrow i < j \\
or, \text{ if } i = j, T_1 > T_2 \text{ in } R .
\]

The leading term of the vector \((x, y^2)\) with respect to two different module term-orderings:

```plaintext
example
Use R ::= QQ[x,y], ToPos;
LT(Vector(x,y^2));
Vector(0, y^2)

Use R ::= QQ[x,y], PosTo;
LT(Vector(x,y^2));
Vector(x, 0)
```

### III-9.7 Quotient Rings

If “\(R\)” is a ring and “\(I\)” is an ideal (in “\(R\)” ) then “\(R/I\)” creates the corresponding quotient ring.

```plaintext
/**/ Use R ::= QQ[x,y];
/**/ I := ideal(x^3+y^3, x^2*y-y^2*x);
/**/ Q := R/I;
/**/ HilbertFn(Q); -- the Hilbert function for Q
H(0) = 1
```
| H(1) = 2 |
| H(2) = 3 |
| H(3) = 2 |
| H(4) = 1 |
| H(t) = 0 for t >= 5 |

See Also: NewQuotientRing(14.9 pg.182)

III-9.8 Commands and Functions for RING

- AffHilbertFn: the affine Hilbert function
- BaseRing: the base ring of a ring
- BettiDiagram: the diagram of the graded Betti numbers
- CanonicalHom: canonical homomorphism
- characteristic: the characteristic of a ring
- CoeffEmbeddingHom: returns the coefficient embedding homomorphism of a polynomial ring
- CoeffRing: the ring of coefficients of a polynomial ring
- ColMat: single column matrix
- DefiningIdeal: defining ideal of a quotient ring
- DensePoly: the sum of all power-products of a given degree
- DiagMat: matrix with given diagonal
- dim: the dimension of a ring or quotient object
- EmbeddingHom: returns the embedding homomorphism of a fraction field
- GenericPoints: random projective points
- GradingMat: matrix of generalized weights for indeterminates
- HilbertFn: the Hilbert function
- HilbertPoly: the Hilbert polynomial
- HilbertSeries: the Hilbert-Poincare series
- HilbertSeriesMultiDeg: the Hilbert-Poincare series wrt a multigrading
- HVector: the h-vector of a module or quotient object
- ideal: ideal generated by list
- IdealOfPoints: ideal of a set of affine points
- IdentityMat: the identity matrix
- implicit: implicitization
- indet: individual indeterminates
- indets: list of indeterminantes in a PolyRing
- IndetSymbols: the names of the indeterminates in a PolyRing
- InducedHom: homomorphism induced by a homomorphism
- IsField: test whether a ring is a field
- IsFiniteField: test whether a ring is a finite field
- IsQQ: test whether a ring is the ring of rationals
- IsQuotientRing: test whether a ring is a quotient ring
- IsStdGraded: checks if the grading is standard
- IsTrueGCDDomain: test whether a ring is a true GCD domain
- IsZZ: test whether a ring is the ring of integers
- LogCardinality: extension degree of a finite field
- LogToTerm: returns a monomial (power-product) with given exponents
- matrix: convert a list into a matrix
- multiplicity: the multiplicity (degree) of a ring or quotient object
- NewFractionField: create a new fraction field
- NewFreeModule: create a new FreeModule
- NewMat: Zero matrix
- NewPolyRing: create a new PolyRing
- NewQuotientRing: create a new quotient ring
- NumIndets: number of indeterminates
**III-9.9 Commands and Functions returning RING**

- one: one of a ring
- operators, shortcuts: Special characters equivalent to commands
- OrdMat: matrix defining a term-ordering
- poincare [OBSOLESCENT]: [OBSOLESCENT] the Hilbert-Poincare series
- PolyAlgebraHom: homomorphism of polynomial algebras
- PolyRingHom: homomorphism of polynomial rings
- PrintBettiDiagram: the diagram of the graded Betti numbers
- ReadExpr: Read RINGELEM expression from string
- reg: Castelnuovo-Mumford regularity of a module
- RegularityIndex: regularity index of a Hilbert function or series
- RingElem: convert an expression into a RINGELEM
- RingID: identification for ring
- RowMat: single row matrix
- SubalgebraRepr: representation of a polynomial as a subalgebra element
- TmpNBM: Numerical Border Basis of ideal of points
- WeightsMatrix [OBSOLESCENT]: [OBSOLESCENT] matrix of generalized weights for indeterminates
- zero: zero of a ring
- ZeroMat: matrix filled with 0

**III-9.9 Commands and Functions returning RING**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BaseRing</td>
<td>the base ring of a ring</td>
</tr>
<tr>
<td>codomain</td>
<td>codomain of a homomorphism</td>
</tr>
<tr>
<td>CoeffRing</td>
<td>the ring of coefficients of a polynomial ring</td>
</tr>
<tr>
<td>domain</td>
<td>domain of a homomorphism</td>
</tr>
<tr>
<td>NewFractionField</td>
<td>create a new fraction field</td>
</tr>
<tr>
<td>NewPolyRing</td>
<td>create a new PolyRing</td>
</tr>
<tr>
<td>NewQuotientRing</td>
<td>create a new quotient ring</td>
</tr>
<tr>
<td>NewRingFp</td>
<td>create a new finite field</td>
</tr>
<tr>
<td>NewRingTwinFloat</td>
<td>create a new twin-float ring</td>
</tr>
<tr>
<td>RingOf</td>
<td>the ring of the object</td>
</tr>
<tr>
<td>RingQQ</td>
<td>the ring of rationals</td>
</tr>
<tr>
<td>RingQQt</td>
<td>pre-defined polynomial rings</td>
</tr>
<tr>
<td>RingsOf</td>
<td>list of the rings of an object</td>
</tr>
<tr>
<td>RingZZ</td>
<td>the ring of integers</td>
</tr>
</tbody>
</table>
Chapter III-10

RINGHOM

III-10.1 Introduction to RINGHOM

A variable “x” containing an INT or a RAT can be immediately used within any RING. But an object “X” of other types, such as RINGELEM, IDEAL, MAT,... can be used only within its own RING, “RingOf(X)”. Such an object can be mapped into another RING using a “RINGHOM”:

```
X
```

Most likely, the only function you need to use is just “CanonicalHom” (I-3.2 pg.41) which returns the canonical homomorphism between two rings (if there is one). Given a RINGHOM "\phi" just type “\phi(x)” if “x” is a RINGELEM, “apply(\phi, x)” if “x” is a LIST or MAT.

However, there are also a few handy shortcuts silently determining and applying a homomorphism: the functions “matrix” (I-13.8 pg.170) and “RingElem” (I-18.38 pg.238) map the argument into the given ring (e.g. “matrix(R, M)” maps “M” into a new matrix in the ring “R”). Another shortcut is “BringIn” (I-2.12 pg.39) (easy, but slow).

NOTE: all CoCoA functions should be smart enough to take into account the RING in which their value was defined, for example “GBasis” (I-7.1 pg.95), “LT” (I-12.21 pg.164), “wdeg” (I-23.1 pg.285),...

NOTE: “QZP”, “ZPQ” are NOT YET IMPLEMENTED.

See Also: Commands and Functions for RINGHOM(III-10.3 pg.360), Commands and Functions returning RINGHOM(III-10.4 pg.360), apply(I-1.12 pg.30), CanonicalHom(I-3.2 pg.41), PolyAlgebraHom(I-16.14 pg.205), PolyRingHom(I-16.15 pg.206), matrix(I-13.8 pg.170), RingElem(I-18.38 pg.238), BringIn(I-2.12 pg.39)

III-10.2 Composition of RINGHOM

Two RINGHOM “\phi: R-->S” and “\psi: S-->T” can be composed.

```
/**/ R := NewPolyRing(QQ, "a");
/**/ S := NewFractionField(R); -- QQ(a)
/**/ T := NewPolyRing(S, "x");
/**/ phi := CanonicalHom(R,S);
/**/ psi := CanonicalHom(S,T);
/**/ theta := psi(phi);
/**/ theta(ReadExpr(R, "a^2+a-1"));
a^2 +a -1
/**/ RingOf(theta(ReadExpr(R, "a^2+a-1"))) = T;
true
```

See Also: CanonicalHom(I-3.2 pg.41), InducedHom(I-9.24 pg.127)
III-10.3 Commands and Functions for RINGHOM

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>apply</td>
<td>apply homomorphism</td>
</tr>
<tr>
<td>codomain</td>
<td>codomain of a homomorphism</td>
</tr>
<tr>
<td>domain</td>
<td>domain of a homomorphism</td>
</tr>
<tr>
<td>InducedHom</td>
<td>homomorphism induced by a homomorphism</td>
</tr>
<tr>
<td>IsInjective</td>
<td>check if a RINGHOM is injective</td>
</tr>
<tr>
<td>IsSurjective</td>
<td>check if a RINGHOM is surjective</td>
</tr>
<tr>
<td>ker</td>
<td>Kernel of a homomorphism</td>
</tr>
<tr>
<td>PolyRingHom</td>
<td>homomorphism of polynomial rings</td>
</tr>
<tr>
<td>PreImage</td>
<td>preimage of a RINGELEM</td>
</tr>
</tbody>
</table>

III-10.4 Commands and Functions returning RINGHOM

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanonicalHom</td>
<td>canonical homomorphism</td>
</tr>
<tr>
<td>CoeffEmbeddingHom</td>
<td>returns the coefficient embedding homomorphism of a polynomial ring</td>
</tr>
<tr>
<td>EmbeddingHom</td>
<td>returns the embedding homomorphism of a fraction field</td>
</tr>
<tr>
<td>InducedHom</td>
<td>homomorphism induced by a homomorphism</td>
</tr>
<tr>
<td>PolyAlgebraHom</td>
<td>homomorphism of polynomial algebras</td>
</tr>
<tr>
<td>PolyRingHom</td>
<td>homomorphism of polynomial rings</td>
</tr>
</tbody>
</table>
Chapter III-11

RINGELEM

III-11.1 Introduction to RINGELEM

An object of type RINGELEM in CoCoA represents an element of a ring.

To fix terminology about polynomials (elements of a polynomial ring): a polynomial is a sum of terms; each term is the product of a coefficient and power-product, a power-product being a product of powers of indeterminates. (In English it is standard to use “monomial” to mean a power-product, however, in other languages, such as Italian, monomial connotes a power-product multiplied by a scalar. In the interest of world peace, we will use the term power-product in those cases where confusion may arise.)

```plaintext
/**/ Use R ::= QQ[x,y,z];
/**/ F := ***3xyz + xy^2***;
/**/ F;
xy^2 + 3xyz

Use R ::= QQ[x[1..5]];
Sum([x[N]^2 | N In 1..5]);
```

CoCoA always keeps polynomials ordered with respect to the term-orderings of their corresponding rings. The following algebraic operations on polynomials are supported:

\[ F^N, +F, -F, F*G, F/G \text{ if } G \text{ divides } F, F+G, F-G, \]

where \( F, G \) are polynomials and \( N \) is an integer. The result may be a rational function.

```
Use R ::= QQ[x,y,z];
F := x^2+xy;
G := x;
F/G;
x + y

F/(x+z);
(x^2 + xy)/(x + z)

F^2;
x^4 + 2x^3y + x^2y^2

F^(-1);
```
1/(x^2 + xy)
-------------------------------

III-11.2 Evaluation of Polynomials

***** NOT YET UPDATED TO CoCoA-5: follow with care *****

Polynomials may be evaluated using the function “\texttt{subst}” (I-19.39 pg.264). More generally, “\texttt{subst}” (I-19.39 pg.264) allows one to substitute polynomials from the current ring for the indeterminates of a given polynomial. If substitutions are to be made for each indeterminate, in order, it is easier to use “\texttt{eval}” (I-5.10 pg.79). For more general substitutions, see “\texttt{PolyAlgebraHom}” (I-16.14 pg.205).

\begin{verbatim}
Use R ::= QQ[x,y,z];
F := x+y+z;
Eval(F,[2,1]); -- let x=2 and y=1 in F 
z + 3
-------------------------------
Subst(F,[[x,2],[y,1]]); -- the same thing using Subst 
z + 3
-------------------------------
Subst(F,y,1); -- the syntax is easier when substituting for a single 
-- indeterminate 
x + z + 1
-------------------------------
Subst(F, [[y,x-y], [z,2]]); -- substitute x-y for y and 2 for z 
2x - y + 2
-------------------------------
\end{verbatim}

See Also: \texttt{eval} (I-5.10 pg.79), \texttt{subst} (I-19.39 pg.264)

III-11.3 Commands and Functions for RINGELEM

\begin{verbatim}
abs
apply
ApproxSolve
AsINT
AsRAT
binomial
CanonicalRepr
CharPoly
ClearDenom
coefficients
CoefficientsWRT
CoeffListWRT
CoeffOfTerm
content
ContentFreeFactor
ContentWRT
cyclotomic
DecimalStr
deg
den
deriv
\end{verbatim}

absolute value of a number
apply homomorphism
Approximate real solutions for polynomial system
convert into an INT
convert into a RAT
binomial coefficient
representative of a class in a quotient ring
characteristic polynomial of a matrix
clear common denominator of a polynomial with rational coeffs
list of coefficients of a polynomial
list of coeffs and PPs of a polynomial wrt an indet or a list of indets
list of coefficients of a polynomial wrt and indet
coefficient of a term of a polynomial
content of a polynomial
factorization of multivariate polynomial into content-free factors
content of a polynomial wrt and indet or a list of indets
n-th cyclotomic polynomial
convert rational number to decimal string
the standard degree of a polynomial or moduleelem
denominator
the derivative of a polynomial or rational function
DerivationAction | Action of a derivation
DF | the degree form of a polynomial
discriminant | the discriminant of a polynomial
DivAlg | division algorithm
eigenvectors | eigenvalues and eigenvectors of a matrix
eval | eliminate variables
factor | substitute numbers or polynomials for indeterminates
FloatApprox | approx. of rational number of the form $M \times 2^E$
FloatStr | convert rational number to a decimal string
FrbAlexanderDual | Alexander Dual of monomial ideals
gcd | greatest common divisor
GenRepr | representation in terms of generators
homog | homogeneize with respect to an indeterminate
ideal | ideal generated by list
IndetIndex | index of an indeterminate
IndetName | the name of an indeterminate
IndetSubscripts | the index of an indeterminate
interreduce, interreduced | interreduce a list of polynomials
IsConstant | checks if a ringelem is in the coefficient ring
IsDivisible | checks if A is divisible by B
IsElem | checks if A is an element of B
IsHomog | test whether given polynomials are homogeneous
IsIn | check if one object is contained in another
IsIndet | checks argument is an indeterminate
IsInRadical | check if a polynomial (or ideal) is in a radical
IsOne | test whether an object is one
IsPthPower | p-th power test
IsTerm | checks if the argument is a term
IsZero | test whether an object is zero
IsZeroDivisor | test whether a RINGELEM is a zero-divisor
jacobian | the Jacobian of a list of polynomials
LC | the leading coefficient of a polynomial or ModuleElem
lcm | least common multiple
LF | the leading form of a polynomial or an ideal
LinearSimplify | simplifying linear substitution for a univariate polynomial over QQ
LM | the leading monomial of a polynomial or ModuleElem
log | the list of exponents of the leading term of a polynomial
LPP | the leading power-product of a polynomial or ModuleElem
LT | the leading term of an object
MantissaAndExponent10 | convert rational number to a float
MantissaAndExponent2 | convert rational number to a binary float
max | a maximum element of a sequence or list
min | a minimum element of a sequence or list
MinPoly | minimal polynomial of a matrix
MinPowerInIdeal | the minimum power of a polynomial is an ideal
monic | divide polynomials by their leading coefficients
monomials | the list of monomials of a polynomial
NewMatFilled | matrix filled with value
NF | normal form
NmzEhrhartRing | Computes the Ehrhart ring
NmzIntClosureMonIdeal | integral closure of a monomial ideal
NmzIntClosureToricRing | integral closure of a toric ring
NmzNormalToricRing | normalization of a toric ring
NR | normal reduction
num | numerator
NumTerms | number of terms in a polynomial
III-11.4 Commands and Functions returning RINGELEM

abs        absolute value of a number
apply      apply homomorphism
binomial   binomial coefficient
CanonicalRepr representative of a class in a quotient ring
CharPoly   characteristic polynomial of a matrix
ClearDenom clear common denominator of a polynomial with rational coeffs
CoeffListWRT list of coefficients of a polynomial wrt and indet
CoeffOfTerm coefficient of a term of a polynomial
content    content of a polynomial
ContentWRT content of a polynomial wrt and indet or a list of indets
cyclotomic n-th cyclotomic polynomial
den        denominator
DensePoly   the sum of all power-products of a given degree
deriv       the derivative of a polynomial or rational function
det         the determinant of a matrix
DF          the degree form of a polynomial
discriminant the discriminant of a polynomial
FirstNonZero the first non-zero entry in a MODULEELEM
FirstNonZeroPosn the first non-zero entry in a MODULEELEM
gcd         greatest common divisor
### III-11.4. Commands and Functions returning RINGELEM

<table>
<thead>
<tr>
<th>Command/Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HilbertPoly</td>
<td>the Hilbert polynomial</td>
</tr>
<tr>
<td>homog</td>
<td>homogenize with respect to an indeterminate</td>
</tr>
<tr>
<td>indet</td>
<td>individual indeterminates</td>
</tr>
<tr>
<td>Interpolate</td>
<td>interpolating polynomial</td>
</tr>
<tr>
<td>interreduce, interreduced</td>
<td>interreduce a list of polynomials</td>
</tr>
<tr>
<td>InverseSystem</td>
<td>Inverse system of an ideal of derivations</td>
</tr>
<tr>
<td>JanetBasis</td>
<td>the Janet basis of an ideal</td>
</tr>
<tr>
<td>LC</td>
<td>the leading coefficient of a polynomial or ModuleElem</td>
</tr>
<tr>
<td>lcm</td>
<td>least common multiple</td>
</tr>
<tr>
<td>LF</td>
<td>the leading form of a polynomial or an ideal</td>
</tr>
<tr>
<td>LM</td>
<td>the leading monomial of a polynomial or ModuleElem</td>
</tr>
<tr>
<td>LogToTerm</td>
<td>returns a monomial (power-product) with given exponents</td>
</tr>
<tr>
<td>LPP</td>
<td>the leading power-product of a polynomial or ModuleElem</td>
</tr>
<tr>
<td>LT</td>
<td>the leading term of an object</td>
</tr>
<tr>
<td>MinPoly</td>
<td>minimal polynomial of a matrix</td>
</tr>
<tr>
<td>monic</td>
<td>divide polynomials by their leading coefficients</td>
</tr>
<tr>
<td>NF</td>
<td>normal form</td>
</tr>
<tr>
<td>NmzDiagInvariants</td>
<td>ring of invariants of a diagonalizable group action</td>
</tr>
<tr>
<td>NmzEhrhartRing</td>
<td>Computes the Ehrhart ring</td>
</tr>
<tr>
<td>NmzFiniteDiagInvariants</td>
<td>ring of invariants of a finite group action</td>
</tr>
<tr>
<td>NmzIntClosureMonIdeal</td>
<td>integral closure of a monomial ideal</td>
</tr>
<tr>
<td>NmzIntClosureToricRing</td>
<td>integral closure of a toric ring</td>
</tr>
<tr>
<td>NmzIntersectionValRings</td>
<td>intersection of ring of valuations</td>
</tr>
<tr>
<td>NmzNormalToricRing</td>
<td>normalization of a toric ring</td>
</tr>
<tr>
<td>NmzTorusInvariants</td>
<td>ring of invariants of torus action</td>
</tr>
<tr>
<td>NR</td>
<td>normal reduction</td>
</tr>
<tr>
<td>num</td>
<td>numerator</td>
</tr>
<tr>
<td>one</td>
<td>one of a ring</td>
</tr>
<tr>
<td>pfaffian</td>
<td>the Pfaffian of a skew-symmetric matrix</td>
</tr>
<tr>
<td>PthRoot</td>
<td>Compute p-th root</td>
</tr>
<tr>
<td>QZP</td>
<td>change field for polynomials and ideals</td>
</tr>
<tr>
<td>randomize</td>
<td>randomize the coefficients of a given polynomial</td>
</tr>
<tr>
<td>randomized</td>
<td>randomize the coefficients of a given polynomial</td>
</tr>
<tr>
<td>RationalAffinePoints</td>
<td>Affine rational solutions</td>
</tr>
<tr>
<td>RationalProjectivePoints</td>
<td>Projective rational solutions</td>
</tr>
<tr>
<td>RationalSolve</td>
<td>Rational solutions for polynomial system</td>
</tr>
<tr>
<td>ReadExpr</td>
<td>Read RINGELEM expression from string</td>
</tr>
<tr>
<td>ReducedGBasis</td>
<td>compute reduced Groebner basis</td>
</tr>
<tr>
<td>resultant</td>
<td>the resultant of two polynomials</td>
</tr>
<tr>
<td>RingElem</td>
<td>convert an expression into a RINGELEM</td>
</tr>
<tr>
<td>SymbolRange</td>
<td>range of symbols for the indeterminates of a PolyRing</td>
</tr>
<tr>
<td>zero</td>
<td>zero of a ring</td>
</tr>
<tr>
<td>ZPQ</td>
<td>change field for polynomials and ideals</td>
</tr>
</tbody>
</table>
Chapter III-12

IDEAL

III-12.1 Commands and Functions for IDEAL

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BBasis5</td>
<td>Border Basis of zero dimensional ideal</td>
</tr>
<tr>
<td>BettiDiagram</td>
<td>the diagram of the graded Betti numbers</td>
</tr>
<tr>
<td>BettiMatrix</td>
<td>the matrix of the graded Betti numbers</td>
</tr>
<tr>
<td>colon</td>
<td>ideal or module quotient</td>
</tr>
<tr>
<td>Depth</td>
<td>Depth of a module</td>
</tr>
<tr>
<td>elim</td>
<td>eliminate variables</td>
</tr>
<tr>
<td>EquiIsoDec</td>
<td>equidimensional isoradical decomposition</td>
</tr>
<tr>
<td>FrbAlexanderDual</td>
<td>Alexander Dual of monomial ideals</td>
</tr>
<tr>
<td>FrbAssociatedPrimes</td>
<td>Associated primes of monomial ideals</td>
</tr>
<tr>
<td>FrbIrreducibleDecomposition</td>
<td>Irreducible decomposition of monomial ideals</td>
</tr>
<tr>
<td>FrbMaximalStandardMonomials</td>
<td>Maximal standard monomials of monomial ideals</td>
</tr>
<tr>
<td>FrbPrimaryDecomposition</td>
<td>Primary decomposition of monomial ideals</td>
</tr>
<tr>
<td>GBasis</td>
<td>calculate a Groebner basis</td>
</tr>
<tr>
<td>GBasisTimeout</td>
<td>compute a Groebner basis with a timeout</td>
</tr>
<tr>
<td>GenRepr</td>
<td>representation in terms of generators</td>
</tr>
<tr>
<td>gens</td>
<td>list of generators of an ideal</td>
</tr>
<tr>
<td>gin</td>
<td>generic initial ideal</td>
</tr>
<tr>
<td>HColon</td>
<td>ideal or module quotient</td>
</tr>
<tr>
<td>HilbertFn</td>
<td>the Hilbert function</td>
</tr>
<tr>
<td>HilbertSeries</td>
<td>the Hilbert-Poincare series</td>
</tr>
<tr>
<td>homog</td>
<td>homogenize with respect to an indeterminate</td>
</tr>
<tr>
<td>HSaturation</td>
<td>saturation of ideals</td>
</tr>
<tr>
<td>InitialIdeal</td>
<td>Initial ideal</td>
</tr>
<tr>
<td>intersection</td>
<td>intersect lists, ideals, or modules</td>
</tr>
<tr>
<td>IntersectList</td>
<td>intersect lists, ideals, or modules</td>
</tr>
<tr>
<td>InverseSystem</td>
<td>Inverse system of an ideal of derivations</td>
</tr>
<tr>
<td>IsContained</td>
<td>checks if A is Contained in B</td>
</tr>
<tr>
<td>IsElem</td>
<td>checks if A is an element of B</td>
</tr>
<tr>
<td>IsHomog</td>
<td>test whether given polynomials are homogeneous</td>
</tr>
<tr>
<td>IsIn</td>
<td>check if one object is contained in another</td>
</tr>
<tr>
<td>IsInRadical</td>
<td>check if a polynomial (or ideal) is in a radical</td>
</tr>
<tr>
<td>IsLexSegment</td>
<td>checks if an ideal is lex-segment</td>
</tr>
<tr>
<td>IsOne</td>
<td>test whether an object is one</td>
</tr>
<tr>
<td>IsStable</td>
<td>checks if an ideal is stable</td>
</tr>
<tr>
<td>IsStronglyStable</td>
<td>checks if an ideal is strongly stable</td>
</tr>
<tr>
<td>IsZero</td>
<td>test whether an object is zero</td>
</tr>
<tr>
<td>IsZeroDim</td>
<td>test whether an ideal is zero-dimensional</td>
</tr>
<tr>
<td>JanetBasis</td>
<td>the Janet basis of an ideal</td>
</tr>
</tbody>
</table>
### III-12.2 Commands and Functions returning IDEAL

- **colon**  
  ideal or module quotient
- **DefiningIdeal**  
  defining ideal of a quotient ring
- **elim**  
  eliminate variables
- **EquiIsoDec**  
  equidimensional isoradical decomposition
- **GBM**  
  intersection of ideals for zero-dimensional schemes
- **gin**  
  generic initial ideal
- **HColon**  
  ideal or module quotient
- **HGBM**  
  intersection of ideals for zero-dimensional schemes
- **homog**  
  homogenize with respect to an indeterminate
- **HSaturation**  
  saturation of ideals
- **ideal**  
  ideal generated by list
- **IdealOfPoints**  
  ideal of a set of affine points
<table>
<thead>
<tr>
<th>Command/Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IdealOfProjectivePoints</td>
<td>ideal of a set of projective points</td>
</tr>
<tr>
<td>implicit</td>
<td>implicitization</td>
</tr>
<tr>
<td>InitialIdeal</td>
<td>Initial ideal</td>
</tr>
<tr>
<td>intersection</td>
<td>intersect lists, ideals, or modules</td>
</tr>
<tr>
<td>IntersectList</td>
<td>intersect lists, ideals, or modules</td>
</tr>
<tr>
<td>ker</td>
<td>Kernel of a homomorphism</td>
</tr>
<tr>
<td>LexSegmentIdeal</td>
<td>lex-segment ideal containing ( L ), or with the same Hilbert fn as ( I )</td>
</tr>
<tr>
<td>LF</td>
<td>the leading form of a polynomial or an ideal</td>
</tr>
<tr>
<td>LT</td>
<td>the leading term of an object</td>
</tr>
<tr>
<td>minimized</td>
<td>ideal, submodule with minimal generators</td>
</tr>
<tr>
<td>MonsInIdeal</td>
<td>ideal generated by the monomials in an ideal</td>
</tr>
<tr>
<td>PerpIdealOfForm</td>
<td>Ideal of derivations annihilating a form</td>
</tr>
<tr>
<td>PrimaryDecomposition</td>
<td>primary decomposition of an ideal</td>
</tr>
<tr>
<td>PrimaryDecomposition0</td>
<td>primary decomposition of a ( 0 )-dimensional ideal</td>
</tr>
<tr>
<td>PrimaryDecompositionGTZ0</td>
<td>primary decomposition of a ( 0 )-dimensional ideal</td>
</tr>
<tr>
<td>QZP</td>
<td>change field for polynomials and ideals</td>
</tr>
<tr>
<td>radical</td>
<td>radical of an ideal</td>
</tr>
<tr>
<td>RadicalOfUnmixed</td>
<td>radical of an unmixed ideal</td>
</tr>
<tr>
<td>saturate</td>
<td>saturation of ideals</td>
</tr>
<tr>
<td>StableIdeal</td>
<td>stable ideal containing ( L )</td>
</tr>
<tr>
<td>StronglyStableIdeal</td>
<td>strongly stable ideal containing ( L )</td>
</tr>
<tr>
<td>TgCone</td>
<td>tangent cone</td>
</tr>
<tr>
<td>toric</td>
<td>saturate toric ideals</td>
</tr>
<tr>
<td>ZPQ</td>
<td>change field for polynomials and ideals</td>
</tr>
</tbody>
</table>
Chapter III-13

MAT

III-13.1 Introduction to MAT

An m x n matrix is represented in CoCoA by the list of its rows (see “matrix” (I-13.8 pg.170)). The (A,B)-th entry of a matrix M is given by “M[A][B]” or “M[A,B]”.

The following operations are defined as one would expect for matrices

\[ M^A, +M, -N, M+N, M-N, M*N, F*M, M*F \]

where M, N are matrices, A is a non-negative integer, and F is a polynomial, with the obvious restrictions on the dimensions of the matrices involved.

```
/**/ Use R ::= QQ[x,y];
/**/ N := matrix(R, [[1,2],[3,4]]);
/**/ N[1,2];
2;

/**/ N^2;
matrix( /*RingDistrMPolyClean(QQ, 2)*/
  [[7, 10],
   [15, 22]])

/**/ x * N;
matrix( /*RingDistrMPolyClean(QQ, 2)*/
  [[x, 2*x],
   [3*x, 4*x]])

/**/ N + matrix([[x,x], [y,y]]);
matrix( /*RingDistrMPolyClean(QQ, 2)*/
  [[x +1, x +2],
   [y +3, y +4]])
```

III-13.2 Commands and Functions for MAT

- adj: classical adjoint matrix (also known as adjugate)
- AlmostQR: QR decomposition of a matrix
- apply: apply homomorphism
- BlockMat: create a block matrix
- BlockMat2x2: create a block matrix with 4 matrices
CharPoly  
characteristic polynomial of a matrix
CompleteToOrd  
completes a matrix to an ordering matrix
ConcatAntiDiag  
create a simple block matrix
ConcatDiag  
create a simple block matrix
ConcatHor  
create a simple block matrix
ConcatHorList  
create a simple block matrix
ConcatVer  
create a simple block matrix
ConcatVerList  
create a simple block matrix
det  
the determinant of a matrix
eigenvectors  
eigenvalues and eigenvectors of a matrix
ElimMat  
matrix for elimination ordering
eval  
substitute numbers or polynomials for indeterminates
FGLM5  
perform a FGLM Groebner Basis conversion
GetCol  
convert a column of a matrix into a list
GetCols  
convert a matrix into a list of lists
GetRow  
convert a row of a matrix into a list
GetRows  
convert a matrix into a list of lists
HilbertBasisKer  
Hilbert basis for a monoid
HilbertSeriesMultiDeg  
the Hilbert-Poincare series wrt a multigrading
HomogElimMat  
matrix for elimination ordering
IdealOfPoints  
ideal of a set of affine points
inverse  
multiplicative inverse of matrix
IsAntiSymmetric  
checks if a matrix is anti-symmetric
IsDiagonal  
checks if a matrix is diagonal
IsPositiveGrading  
check if a matrix defines a positive grading
IsSymmetric  
checks if a matrix is symmetric
IsTermOrdering  
check if a matrix defines a term-ordering
IsZero  
test whether an object is zero
IsZeroCol, IsZeroRow  
test whether a column(row) is zero
LinKer  
find the kernel of a matrix
LinKerBasis  
find the kernel of a matrix
LinSolve  
find a solution to a linear system
matrix  
convert a list into a matrix
minors  
list of minor determinants of a matrix
MinPoly  
minimal polynomial of a matrix
NewFreeModule  
create a new FreeModule
NewPolyRing  
create a new PolyRing
NmzDiagInvariants  	ring of invariants of a diagonalizable group action
NmzFiniteDiagInvariants  	ring of invariants of a finite group action
NmzHilbertBasis  
Hilbert Basis of a monoid
NmzIntersectionValRings  
intersection of ring of valuations
NmzTorusInvariants  	ring of invariants of torus action
NumCols  
number of columns in a matrix
NumRows  
number of rows in a matrix
pfaffian  
the Pfaffian of a skew-symmetric matrix
PreprocessPts  
Reduce redundancy in a set of approximate points
product  
the product of the elements of a list
RingOf  
the ring of the object
RingsOf  
list of the rings of an object
rk  
rank of a matrix or module
SetRow  
set a list as a row into a matrix
submat  
submatrix
sum  
the sum of the elements of a list
SwapRows  
swap two rows in a matrix
TmpNBMB  
Numerical Border Basis of ideal of points
toric  
saturate toric ideals
transposed  
the transposition of a matrix
### III-13.3 Commands and Functions returning MAT

- **adj**: classical adjoint matrix (also known as adjugate)
- **apply**: apply homomorphism
- **BlockMat**: create a block matrix
- **BlockMat2x2**: create a block matrix with 4 matrices
- **ColMat**: single column matrix
- **CompleteToOrd**: completes a matrix to an ordering matrix
- **ConcatAntiDiag**: create a simple block matrix
- **ConcatDiag**: create a simple block matrix
- **ConcatHor**: create a simple block matrix
- **ConcatHorList**: create a simple block matrix
- **ConcatVer**: create a simple block matrix
- **ConcatVerList**: create a simple block matrix
- **DiagMat**: matrix with given diagonal
- **ElimMat**: matrix for elimination ordering
- **GensAsCols, GensAsRows**: matrix of generators of a module
- **GradingMat**: matrix of generalized weights for indeterminates
- **HomogElimMat**: matrix for elimination ordering
- **IdentityMat**: the identity matrix
- **inverse**: multiplicative inverse of matrix
- **jacobian**: the Jacobian of a list of polynomials
- **LexMat**: matrices for std. term-orderings
- **LinKer**: find the kernel of a matrix
- **LinSolve**: find a solution to a linear system
- **MakeMatByRows, MakeMatByCols**: convert a list into a matrix
- **matrix**: convert a list into a matrix
- **NewMat**: Zero matrix
- **NewMatFilled**: matrix filled with value
- **NmzHilbertBasis**: Hilbert Basis of a monoid
- **OrdMat**: matrix defining a term-ordering
- **RevLexMat**: matrices for std. term-orderings
- **RowMat**: single row matrix
- **StdDegLexMat**: matrices for std. term-orderings
- **StdDegRevLexMat**: matrices for std. term-orderings
- **submat**: submatrix
- **sylvestre**: the Sylvester matrix of two polynomials
- **TensorMat**: returns the tensor product of two matrices
- **transposed**: the transposition of a matrix
- **WeightsMatrix [OBSOLETE]**: [OBSOLETE] matrix of generalized weights for indeterminates
- **XelMat**: matrices for std. term-orderings
- **ZeroMat**: matrix filled with 0
Chapter III-14

MODULE

III-14.1 Commands and Functions for MODULE

- BettiDiagram: the diagram of the graded Betti numbers
- BettiMatrix: the matrix of the graded Betti numbers
- colon: ideal or module quotient
- elim: eliminate variables
- GBasis: calculate a Groebner basis
- GBasisTimeout: compute a Groebner basis with a timeout
- GenRepr: representation in terms of generators
- gens: list of generators of an ideal
- GensAsCols, GensAsRows: matrix of generators of a module
- HilbertSeries: the Hilbert-Poincare series
- HilbertSeriesShifts: the Hilbert-Poincare series
- homog: homogenize with respect to an indeterminate
- IntersectList: intersect lists, ideals, or modules
- IsContained: checks if A is Contained in B
- IsElem: checks if A is an element of B
- IsHomog: test whether given polynomials are homogeneous
- IsIn: check if one object is contained in another
- IsZero: test whether an object is zero
- LT: the leading term of an object
- MinGens: list of minimal generators
- minimalize: ideal, submodule with minimal generators
- minimalized: ideal, submodule with minimal generators
- MinSubsetOfGens: list of minimal generators
- ModuleElem: create a module element
- ModuleOf: the module environment of the object
- NF: normal form
- NumCompts: the number of components
- operators, shortcuts: Special characters equivalent to commands
- PrintBettiDiagram: the diagram of the graded Betti numbers
- PrintBettiMatrix: print the matrix of the graded Betti numbers
- ReducedGBasis: compute reduced Groebner basis
- res: free resolution
- RingOf: the ring of the object
- RingsOf: list of the rings of an object
- rk: rank of a matrix or module
- submodule: submodule generated by list
- SubmoduleCols, SubmoduleRows: convert a matrix into a module
- syz: syzygy modules
### III-14.2 Commands and Functions returning MODULE

- **elim**: eliminate variables
- **homog**: homogenize with respect to an indeterminate
- **IntersectList**: intersect lists, ideals, or modules
- **LT**: the leading term of an object
- **minimalized**: ideal, submodule with minimal generators
- **ModuleOf**: the module environment of the object
- **NewFreeModule**: create a new FreeModule
- **submodule**: submodule generated by list
- **SubmoduleCols, SubmoduleRows**: convert a matrix into a module
- **syz**: syzygy modules
- **SyzOfGens**: syzygy module for a given set of generators
Chapter III-15

MODULEELEM

III-15.1 Introduction to MODULEELEM

An object of type MODULEELEM in CoCoA represents a module element; in CoCoA this usually means an element of the free module \( \mathbb{P}^r \), where \( \mathbb{P} \) is a polynomial ring. For \( v \) and \( w \) MODULEELEM in the same MODULE, and \( f \) RINGELEM in its base ring, the following are also MODULEELEM:

\[ +v, -v, f*v, v*f, v+w, v-w \]

See “ModuleElem” (I-13.23 pg.176).

See Also: Commands and Functions for MODULEELEM(III-15.2 pg.377), Commands and Functions returning MODULEELEM(III-15.3 pg.378)

III-15.2 Commands and Functions for MODULEELEM

- **compts**: list of components of a ModuleElem
- **DivAlg**: division algorithm
- **eval**: substitute numbers or polynomials for indeterminates
- **FirstNonZero**: the first non-zero entry in a MODULEELEM
- **FirstNonZeroPosn**: the first non-zero entry in a MODULEELEM
- **GenRepr**: representation in terms of generators
- **homog**: homogenize with respect to an indeterminate
- **IsElem**: checks if A is an element of B
- **IsHomog**: test whether given polynomials are homogeneous
- **IsIn**: check if one object is contained in another
- **IsTerm**: checks if the argument is a term
- **IsZero**: test whether an object is zero
- **LC**: the leading coefficient of a polynomial or ModuleElem
- **LM**: the leading monomial of a polynomial or ModuleElem
- **LPosn**: the position of the leading power-product in a ModuleElem
- **LPP**: the leading power-product of a polynomial or ModuleElem
- **LT**: the leading term of an object
- **monomials**: the list of monomials of a polynomial
- **NF**: normal form
- **NonZero**: remove zeroes from a list
- **NR**: normal reduction
- **NumCompts**: the number of components
- **product**: the product of the elements of a list
- **RingsOf**: list of the rings of an object
- **ScalarProduct**: scalar product
### submodule
submodule generated by list

### sum
the sum of the elements of a list

### support
the list of terms of a polynomial or moduleelem

## III-15.3 Commands and Functions returning MODULEELEM

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>homog</td>
<td>homogenize with respect to an indeterminate</td>
</tr>
<tr>
<td>LM</td>
<td>the leading monomial of a polynomial or ModuleElem</td>
</tr>
<tr>
<td>LT</td>
<td>the leading term of an object</td>
</tr>
<tr>
<td>ModuleElem</td>
<td>create a module element</td>
</tr>
<tr>
<td>NF</td>
<td>normal form</td>
</tr>
<tr>
<td>NR</td>
<td>normal reduction</td>
</tr>
<tr>
<td>ReducedGBasis</td>
<td>compute reduced Groebner basis</td>
</tr>
</tbody>
</table>
Chapter III-16

Creating new types

III-16.1 Tagging an Object

If “E” is any CoCoA object and “S” a string, then the function “Tagged(E, S)” returns the object “E” tagged with the string “S”. The returned object is then of type “TAGGED(S)”. The function “tag” (I-20.1 pg.271) returns “S”, the tag string of an object, and the function “untagged” (I-21.4 pg.280) returns “E”, the original object, stripped of its tag.

This is the way to add a new type at run-time.

```
example
/**/ L := ["Dave", "March 14, 1959", 372];
/**/ M := Tagged(L, "MiscData"); -- L tagged with the string "MiscData"
/**/ type(L); -- L is a list
LIST
/**/ type(M); -- M is a tagged object
TAGGED("MiscData")
/**/ -- M; -- Until a special print function is defined, the printing of M
--> WARNING: Cannot find "$BackwardCompatible.PrintTagged", so I’m implicitly untagging the value
--> ["Dave", "March 14, 1959", 372]
```

The next section explains how to define functions for pretty printing of tagged objects.

III-16.2 Printing a Tagged Object

Suppose the object “E” is tagged with the string “S”. When one tries to print “E”—say with “Print E”—CoCoA looks for a user-defined function with name “Print_S”. If no such function is available, CoCoA prints E as if it were not tagged, otherwise, it executes “Print_S”.

```
example
/**/ L := ["Dave", "March 14", 1959, 372];
/**/ M := tagged(L,"MiscData");
/**/ Define SpecialPrinting(Dev, Obj)
/**/ Print Obj[1],"'s birthday is: ", Obj[2] on Dev;
/**/ EndDefine;
/**/ PrintTagged := record[MiscData := SpecialPrinting];
/**/ Print M;
Dave's birthday is: March 14
```
III-16.3 Commands and Functions for Tags

The following are commands and functions involving tags:

- **tag**: returns the tag string of an object
- **tagged**: tag an object for pretty printing
- **untagged**: untag an object