# Algebraic Solution of Nonlinear Equation Systems in REDUCE

Herbert Melenk*

A modern computer algebra system (CAS) is primarily a workbench offering an extensive set of computational tools from various field of mathematics and engineering. As most of these tools have an advanced and complicated theoretical background they often are hard for an unexperienced user to apply successfully. The idea of a $SOLVE$ facility in a CAS is to find solutions for mathematical problems by applying the available tools in an automatic and invisible manner. In REDUCE the functionality of SOLVE has grown over the last years to an important extent.

In the circle of REDUCE developers the Konrad–Zuse–Zentrum Berlin (ZIB) has been engaged in the field of solving nonlinear algebraic equation systems, a typical task for a CAS. The algebraic kernel for such computations is Buchberger's algorithm for computing Gröbner bases and related techniques, published in 1966, first introduced in REDUCE 3.3 (1988) and substantially revised and extended in several steps for REDUCE 3.4 (1991). This version also organized for the first time the automatical invocation of Gröbner bases for the solution of pure polynomial systems in the context of REDUCE's SOLVE command.

In the meantime the range of automatically soluble system has been enlarged substantially. This report describes the algorithms and techniques which have been implemented in this context. Parts of the new features have been incorporated in REDUCE 3.4.1 (in July 1992), the rest will become available in the following release.

*E-mail address: melenk@sc.zib-berlin.de

Some of the developments have been encouraged to an important extent by colleagues who use these modules for their research, especially Hubert Caprasse (Liège) [4] and Jarmo Hietarinta (Turku) [10]. The author expresses his thanks to them, to the members of the REDUCE team and especially to A.C. Hearn for a long and fruitful cooperation.

# 1 Introduction

The REDUCE *SOLVE* package is able to handle nonlinear equation systems of the following types:

- pure polynomial systems,

$$\{axq - lxq - mx, ax - gxq - lx, bxq^2 + cxq - jxq - nx,$$
$$q(-axq + lxq + mx), q(-ax + gxq + lx)\}$$

- equations and systems with trigonometric functions,

$$\cos(x_1)\, x_3 - \sin(x_1)\, x_2 + 5\sin(x_1),$$
$$-2\cos(x_1) + 2x_3^2 x_2 + 2x_2^3 + x_2 - 5,$$
$$-2\sin(x_1) + 2x_3^3 + 2x_3 x_2^2 + x_3$$

- equations and systems with surds,

$$\left\{e = \tfrac{b}{4}, ed = \tfrac{\sqrt{\pi}c}{\sqrt{2}}, c = \tfrac{a}{2}, a = \sqrt{\pi}, bd = \pi\sqrt{2}\right\}$$

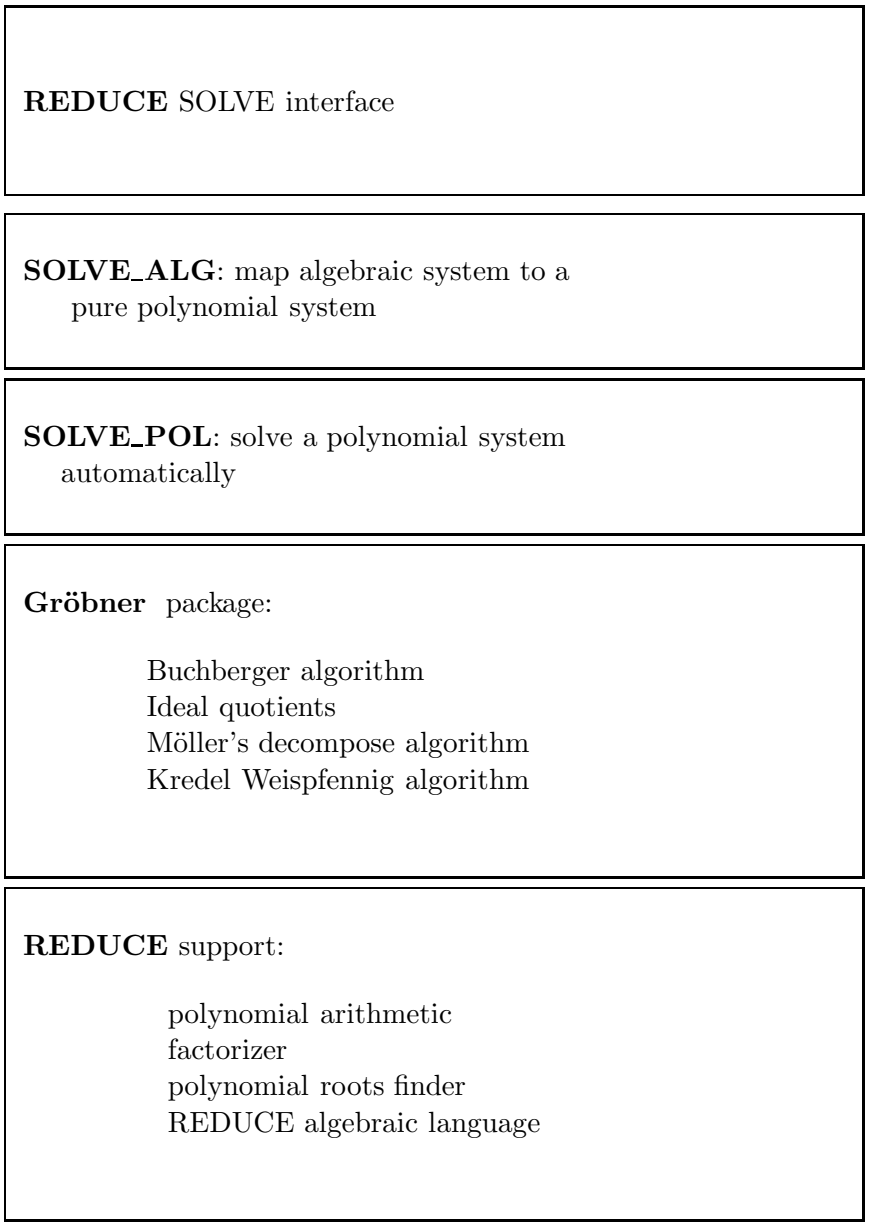
- systems with invertible transcendental functions, as long as these don't establish transcendental, variable dependencies,

$$a^{2x} - a^{\frac{x}{2}} + a^{\frac{x}{3}} = 2$$

- systems which can be reduced to one of the above types by factorization or separation.

$$\left\{-e^{\frac{y}{2}}x - u, -y - 2 * v\right\} \text{ wrt } \{x, y\}$$

The structure of the solver for these systems in REDUCE is given in the following diagram:




The algorithmic heart is the Gröbner package of REDUCE; it is based on the REDUCE polynomial system and has direct access to many of the advanced algebraic algorithms such as polynomial factorization, roots calculation, various coefficient domain arithmetics, resultants etc. The Gröbner package is fully described in [8] and [18].

On the next higher level the polynomial system solver **SOLVE_POL** uses the facilities of the Gröbner package directly for finding solutions of pure polynomial systems.

The algebraic system solver **SOLVE_ALG** extends of **SOLVE_POL** to systems which can be mapped to pure polynomial systems. These are systems containing surds, trigonometric functions and other transcendental functions as long as they do not establish transcendental relations.

The purpose of this document is the description of the levels **SOLVE_POL** and **SOLVE_ALG**. The various examples in this text have been selected to demonstrate one aspect with each example as compact as possible. Of course the REDUCE system has been designed for significantly larger systems. The main limitation in the size of solvable systems is found in the Buchberger algorithm itself and any future progress there increases the scope of automatically solvable systems.

# 2 Algebraic Background for Polynomial Systems

## 2.1 Ideals, Bases, Solutions

Let $P = \{f_i(x), i = 1 \ldots k\}, x = (x_1 \ldots x_n)$ be a set of multivariate polynomials in the ring $Q[x_1 \ldots x_n]$, with a solution set

$$S = S(f_i \cdots f_k) = \{x | f_i(x) = 0 \ \forall i = 1 \ldots k\}$$

All polynomials $u(x) = \sum_j g_j(x) f_j(x)$ for arbitrary polynomials $g_j$ will vanish in all points of $S$. The set of all $u$ establishes a polynomial ideal

$$I = I(f_1 \cdots f_k) = \left\{ \sum_j g_j(x) f_j(x) \right\}$$

and classical algebra tells that $S$ is invariant if we replace the set $\{f_i \cdots f_k\}$ by any other basis for the ideal $I(f_1 \cdots f_k)$.

## 2.2 Lexicographical Gröbner Bases

The Buchberger Algorithm allows one to transform the set of polynomials into a canonical basis of the same ideal, the Gröbner basis $GB = GB(I)$. The Buchberger Algorithm and the general properties of Gröbner bases are described in [2],[3],[5]. For the purpose of equation solving especially Gröbner bases computed under lexicographical term ordering are important; they allow one to determine the set $S$ directly.

If $I$ has **dimension zero** ($S$ is a finite set of isolated points), $GB$ has in most cases the form

$$\begin{aligned}
g_1(x_1, x_k) = \quad & x_1 & & +c_{1,m-1}x_k^{m-1} + c_{1,m-2}x_k^{m-2} + \cdots + c_{1,0} \\
g_2(x_2, x_k) = \quad & x_2 & & +c_{2,m-1}x_k^{m-1} + c_{2,m-2}x_k^{m-2} + \cdots + c_{2,0} \\
& \cdots & & \\
g_{k-1}(x_{k-1}, x_k) = \quad & x_{k-1} & & +c_{k-1,m-1}x_k^{m-1} + c_{k-1,m-2}x_k^{m-2} + \cdots + c_{k-1,0} \\
g_k(x_k) = \quad & x_k^m & & +c_{k,m-1}x_k^{m-1} + c_{k,m-2}x_k^{m-2} + \cdots + c_{k,0}
\end{aligned}$$

A basis in this form has the elimination property: the variable dependency has been reduced to a triangular form, just as with a Gaussian elimination in the linear case. The last polynomial is univariate in $x_k$. It can be solved with usual algebraic or numeric techniques; its zeros $\overline{x}_k$ then are propagated into the remaining polynomials, which then immediately allow one to determine the corresponding coordinates $(\overline{x}_1, \ldots, \overline{x}_{k-1})$.

Example: the system

$$\{y^2 - 6y, xy, 2x^2 - 3y - 6x + 18, 6z - y + 2x\}$$

has for $\{x, y, z\}$ the lexicographical Gröbner basis

$$\{g_1(x, z) = x - z^2 + 2z - 1, g_2(y, z) = y - 2z^2 - 2z - 2, g_3(z) = z^3 - 1\}$$

The roots of the third polynomial are give by

$$\left\{z = 1, z = \frac{1 - \sqrt{3}i}{2}, z = \frac{\sqrt{3}i - 1}{2}\right\}$$

If we propagate one of them into the basis, we generate univariate polynomials of degree one which can be solved immediately; e.g. selecting $\frac{1-\sqrt{3}i}{2}$ for $z$ the basis reduces to

$$\left\{x - \frac{3\sqrt{3}i + 3}{2}, y, 0\right\}$$

such that the final solution for this branch is

$$\left\{x = \frac{3\sqrt{3}i + 3}{2}, y = 0, z = \frac{1 - \sqrt{3}i}{2}\right\}$$

For zero dimensional problems the last polynomial will always be univariate. However, in degenerate cases [1] the other polynomials can contain their leading variable in a higher degree, then containing more mixed terms with the following variables. And there can

[1] A solution set is degenerate if one (or more) root(s) of the univariate polynomial appears in more than one point of the solution set $S$.

be additional polynomials with mixed leading terms (of lower degree) imposing some restrictions. But the variable dependency pattern will remain triangular (eventually with more than $k$ rows). There is a special algorithm for decomposing such ideals using ideal quotients. For details see [18].

The implementation of the Buchberger algorithm in REDUCE corresponds to [7]. Some details and applications have been reported in [17] [14] [15].

### 2.3 Higher Dimensional Problems

Once a lexicographic Gröbner basis is given, the Kredel-Weispfennig algorithm [13] can be used to compute the dimension. If a system has dimension $\geq 1$, there are free parameters (the system is underdetermined), but nevertheless the lexicographic Gröbner basis has a triangular variable dependency pattern. It is therefore often possible to use the same decomposition as with zero dimensional problems.

Example: The system

$$\{x + y^2z - 2y^2 + 4y - 2z - 1, -x + y^2z - 1\}$$

has for $\{x, y, z\}$ the Gröbner basis

$$\{x - y^2 + 2y - z, y^2z - y^2 + 2y - z - 1\}$$

Taking $z$ as parameter the last polynomial can be rewritten as $(z+1)y^2 + 2y - (z+1)$ and its formal roots are given by $y = 1$ and $y = \frac{-z-1}{z-1}$.

Resubstituted into the first equation we find the corresponding values for $x$.

But this solution is only valid under the assumption $z - 1 \neq 0$, because $z - 1$ has been used as a denominator. Consequently the case $z - 1 = 0$ has to be inspected separately, e.g. by computing the Gröbner basis of

$$\{x - y^2 + 2y - z, y^2z - y^2 + 2y - z - 1, z - 1\} .$$

which is

$$\{x, y - 1, z - 1\}$$

leading to the additional isolated solution $x = 0, y = 1, z = 1$.

It is very important for a polynomial solver to detect and handle such cases for systems with nonzero dimension (**"denominator propagation"**). These describe parts of $S$ of lower dimension. Often isolated point solutions are found in that way which represent "stable" solutions of technical systems. Fortunately these can be easily detected in the Gröbner basis: if a basis polynomial can be written as $u(x_m)x_j^n + \cdots$ with $m < j$ and $n$ highest power of $x_j$, the polynomial can be viewed as polynomial in $x_j$ and coefficients

in $x_m$ and $u(x_m)$ exactly generates the subcase. Of course, a subcase can generate more subcases in a finite tree.

In the case dimension >= 1 the decomposition into single solutions can fail. In such a case the Kredel-Weispfennig algorithm [13] helps: This algorithm allows one to compute sets of independent variables for the given ideal and (lexicographical) term ordering. If we select a maximal set of these and remove its variables from the actual variable sequence the new ideal is of dimension zero with the independent variables as parameters in the coefficient domain. Example: the following system given by J. Hietarinta [9] has dimension 6 and cannot be decomposed under most variable sequences:

$$\{ax\,q - lx\,q - mx, ax - gx\,q - lx, bxq^2 + cx\,q - jx\,q - nx\ ,\\ q(-ax\,q + lxq + mx), q(-ax + gx\,q + lx)\}$$

With the variable sequence

$$\{ax, bx, cx, gx, jx, lx, mx, nx, q\}$$

under lexicographical term order it has independent sets ( "left" independent sets in the terminology of [13])

$$\{cx, jx, lx, mx, nx, q\}$$
$$\{cx, gx, jx, lx, mx, nx\}$$
$$\{bx, cx, gx, jx, lx, nx\}$$

If we select the first set as parameters the Gröbner basis over the remaining variables $\{ax, bx, gx\}$ is

$$\left\{ax\,q - lx\,q - mx, bx\,q^2 + cx\,q - jxq - nx, -gx\,q^2 + mx\right\}$$

and this zero dimensional system is easily decomposed into the solution branches.

If run with parametric coefficients the Buchberger algorithm in REDUCE automatically collects all leading coefficients of the intermediate polynomials because the Gröbner basis is only valid under the assumption that these terms are nonzero. In the context of the reduced variable sequence these coefficients play the same role as the denominators during decomposition: the cases that they vanish have to be considered in separate computation branches. Taking these into account the complete solution set is given by

$$\{ax = gx\,q + lx, bx = (-cx\,q + jx\,q + nx)/q^2, mx = gx\,q^2\}$$
$$\{ax = lx, mx = 0, nx = 0, q = 0\}$$

Here the solution with $q = 0$ has been generated by denominator propagation (subcase $q = 0$).

## 2.4 Factorization

If a polynomial has nontrivial factors in a domain without zero divisors it vanishes at a point only if one of its factors vanishes there:

$$p(x) = q(x) * r(x);\ p(\bar{x}) = 0 \rightarrow q(\bar{x}) = 0 \vee r(\bar{x}) = 0$$

If a set oc polynomials contains a factorizable we can decompose it into two (or more) subproblems:

$$S(p_1, \ldots, q * r, \ldots, p_k) \equiv S(p_1, \ldots, q, \ldots, p_k) \cup S(p_1, \ldots, r, \ldots, p_k)$$

In the Gröbner package of REDUCE the Buchberger algorithm has been implemented such that each intermediate polynomial is inspected for factorization [17] [14]; the algorithm contains specific data structures for handling multiple factorizations. This type of decomposition is of big importance for the business of equation solving:

- as the factors have lower degrees than the product, the partial problems are of substantial lower complexity than the full problem,
- the final bases have lower degrees and thus much fewer terms,
- often a nonzero dimensional problem leads to a contiguous variety plus some isolated points; these points typically establish zero dimensional ideal factors and often are isolated already during the Buchberger algorithm.

Example:

$$\{x_1 x_2 - 8 = 0, x_1^2 - 5x_1 + x_2 + 2 = 0\}$$

lexicographical Gröbner basis:

$$\begin{aligned} &8x_1 + x_2^2 + 2x_2 - 40, \\ &\quad x_2^3 + 2x_2^2 - 40x_2 + 64 \end{aligned}$$

factored Gröbner basis:

$$\{\{x_1 + 1, x_2 + 8\}, \{x_1 - 2, x_2 - 4\}, \{x_1 - 4, x_2 - 2\}\}$$

immediate solution:

$$\{\{x_1 = -1, x_2 = -8\}, \{x_1 = 4, x_2 = 2\}, \{x_1 = 2, x_2 = 4\}\}$$

Example: system describing the molecular geometry of $C_6H_{12}$ ([6]):

$$\{-81y_2^2y_1^2+594y_2^2y_1-225y_2^2+594y_2y_1^2-3492y_2y_1-750y_2-225y_1^2-750y_1+14575\,,$$
$$-81y_3^2y_2^2+594y_3^2y_2-225y_3^2+594y_3y_2^2-3492y_3y_2-750y_3-225y_2^2-750y_2+14575\,,$$
$$-81y_3^2y_1^2+594y_3^2y_1-225y_3^2+594y_3y_1^2-3492y_3y_1-750y_3-225y_1^2-750y_1+14575\,,$$
$$81y_3^2y_2^2y_1+81y_3^2y_2y_1^2-594y_3^2y_2y_1-225y_3^2y_2-225y_3^2y_1+1650y_3^2+81y_3y_2^2y_1^2-594y_3y_2^2y_1$$
$$-225y_3y_2^2-594y_3y_2y_1^2+2592y_3y_2y_1+2550y_3y_2-225y_3y_1^2+2550y_3y_1-3575y_3-225y_2^2y_1$$
$$+1650y_2^2-225y_2y_1^2+2550y_2y_1-3575y_2+1650y_1^2-3575y_1-30250\}$$

factored Gröbner Basis:

$$\{\{3y_1-11,3y_2-11,3y_3-11\},$$
$$\{3y_1+5,3y_2+5,3y_3+5\},$$
$$\{3y_3y_2+3y_3y_1-22y_3+3y_2y_1-22y_2-22y_1+121,27y_3^2y_2+27y_3^2y_1-198y_3^2-198y_3y_2$$
$$-198y_3y_1+1164y_3+75y_2+75y_1+250,81y_3^2y_2^2-594y_3^2y_2+225y_3^2-594y_3y_2^2+3492y_3y_2$$
$$+750y_3+225y_2^2+750y_2-14575\}\}$$

Here two isolated solutions $y_1 = y_2 = y_3 = \frac{11}{3}$ and $y_1 = y_2 = y_3 = -\frac{1}{3}$ have been split off, while the third basis represents a variety with one free parameter. Without decomposing by factorization the Gröbner basis for this problem contains polynomials up to degree five.

## 2.5 Optimal Variable Sequence

The computing time for Gröbner base calculations depends significantly on the variable ordering. The REDUCE Gröbner package offers as an option the optimization of a given variable sequence which is evaluated as described in [1]: a variable is more "complicated" than another one if it either appears in a higher degree or appears in the same maximal degree more often. Variables are arranged with decreasing complexity.

However, if an external ordering among the variables is prescribed (e.g. for coding a specific dependency among the variables - see below), the reordering prescribed by the optimization is revised such that the external requirements are fulfilled. The REDUCE *depend* syntax is used to transfer such reordering restrictions.

## 2.6 Ideal Operators

Once an algorithm for the computation of Gröbner bases is available, a complete set of operators for ideal arithmetic can be easily implemented [3]. The REDUCE package `IDEALS` is constructed in that manner [16]. For equation solving the following operations are used:

- Ideal membership. $p(x) \in I \leftrightarrow p(x) \equiv 0\ mod\ GB(I)$

- Ideal subset property. For two ideals $I_1 = (p_1, \ldots, p_k), I_2$ the property $I_1 \subset I_2$ is true if $p_i \equiv 0 \; mod \; I_2 \forall i$; the relation $p \equiv 0 \; mod \; I$ is decidable if a Gröbner basis of $I_2$ is known.

- Single ideal quotient. Let $I$ be a polynomial ideal and $p$ a polynomial: $I : f = \{q \mid q * f \in I\}$. This ideal is computed as intersection of $I$ and the multiples of $f$, then dividing the elements of the basis by $f$. In REDUCE this algorithm has been implemented using a cofactor technique.

## 2.7 Ideal Redundancy

Both, factorization and denominator propagation can produce isolated solutions which are special cases of other solutions. E.g. $\{x = 0, y = 0\}$ is a special case of $\{x = y\}$. In order to have a result as compact as possible such redundant cases should be avoided. The ideal algebra offers a tool for excluding redundancies:

$$S(f_1 \ldots f_k) \subset S(g_1 \ldots g_k) \Longleftrightarrow I(f_1 \ldots f_k) \supset I(g_1 \ldots g_k)$$

So we can avoid subsets of solutions by testing the (inverse!) ideal inclusion for the ideals generated by the partial bases. Applied for the above case: As $I(x - y) \subset I(x, y)$ we can ignore the point $\{x = 0, y = 0\}$ if $(x - y) \in S$.

The ideal subset can be applied only if one solution set is completely a subset of another one. It fails in the case that the intersection of the solutions of two ideals is not empty. E.g. in

$$I_1 = \{x^2 - 1, y^2\}, I_2 = \{x^2 - 2x + 1, y\}$$

neither $I_1 \subset I_2$ nor $I_2 \subset I_1$ holds, but their solution sets

$$\begin{aligned} S(I_1) &= \{\{x = 1, y = 0\}, \{x = -1, y = 0\}\} \; , \\ S(I_2) &= \{\{x = -1, y = 0\}\} \end{aligned}$$

have an intersection $\{x = -1, y = 0\}$ and we don't want to repeat this solution in the output of `solve` (ignoring for the moment that the ideals in this simple example would have been decomposed by factorization). In order to suppress such repetitions a method proposed by J.Hietarinta [9] is used: if two solution sets $S_1$ and $S_2$ are both explicit (that is a set of equations with single variables on the lefthand side and an algebraic expression on the righthand side) $S_1$ is substituted into $S_2$; if then in the resulting $S_2$ all equations are trivial (identical righthand and lefthand sides) we know that $S_1$ is a special case of $S_2$, and $S_1$ can be removed from the solution list. Logically this step is equivalent to an ideal inclusion but with different technical support.

### 2.8 Roots of Polynomials

The REDUCE *Solve* system (including the *Roots* package [12]) computes the roots of a polynomial with respect to a given variable. Depending on the computational context and on the polynomial the result can take various forms:

- Explicit forms:
  - Integer/rational numbers: $Solve(x^2 - 16 = 0, x) \Rightarrow \{x = 4, x = -4\}$
  - Complex numbers: $Solve(x^2 + 16 = 0, x) \Rightarrow \{x = 4i, x = -4i\}$
  - Algebraic Numbers: $Solve(x^2 - 8 = 0, x) \Rightarrow \{x = 2\sqrt{2}, x = -2\sqrt{2}\}$
  - Approximate Numbers: $on\ rounded; Solve(x^2 - 8 = 0, x) \Rightarrow \{x = -2.828, x = 2.828\}$
  - Parametric Expressions: $Solve(x^2y - 4, x) \Rightarrow \left\{x = \frac{2}{\sqrt{y}}, x = -\frac{2}{\sqrt{y}}\right\}$
- Implicit form: $Solve(x^9 - x + 1, x) \Rightarrow \{x = root\_of(x\_^9 - x\_ + 1, x\_)\}$

The implicit form is used when no explicit form for the roots is available (e.g. non decomposable polynomial of degree greater than four) or if the explicit form expression with surds would be too big [2]. The implicit root form has been introduced with REDUCE 3.4.1; with this form we can express algebraic relations with roots although these are not available explicitly. This ability is especially useful for the context of polynomial equation systems where solutions often can be expressed only in terms of a root of a single polynomial of large degree.

## 3 Solution of Pure Polynomial Systems

### 3.1 SolvePoly Macro Algorithm

The solver for pure polynomial systems is an algorithm using the tools described in the previous section. The central part is a factorizing Buchberger algorithm with lexicographic term order which leads to a set of Gröbner bases. These represent parts of the solution set and are decomposed in the second phase to single solutions as explicitly as possible. In this phase redundancy control and numerator propagation take place, which may modify the initial set of bases.

[2] The size restriction is controlled by the REDUCE switch *fullroots*.

$SOLVE\_POLY$:
$Input : x = \{x_1 \dots x_n\}, P = \{f_1(x) \dots f_k(x)\} : polynomial\ system$
$Output : S = solution\ set\ of \{f_i(x) = 0\}$

1. optimize variable sequence $x = \{x_1 \dots x_n\}$ for $P$
2. factorizing lexicographic Buchberger algorithm leads to an initial set of Gröbner bases $G = \{Q_1 \dots Q_r\} \Leftarrow GB_{fact}(P)$
3. for each $\hat{Q} \in G$ collect the solutions $S(\hat{Q})$ in $R$:
   (a) reduce redundancy: $\forall Q \neq \hat{Q} \in G\ do$
   $I(\hat{Q}) \subset I(Q) \rightarrow G \Leftarrow G \setminus \{Q\}, R \Leftarrow R \setminus S(Q)$
   $I(Q) \subset I(\hat{Q}) \rightarrow ignore(\hat{Q})$
   (b) extract solutions $R$ and denominators $D$ from $\hat{Q}$:
   $(R, D) \Leftarrow G\_EXTRACT(\hat{Q})$
   (c) new subcases: $\forall d \in D : G \Leftarrow G \cup GB(\hat{Q} \cup \{d\})$
   (d) $S \Leftarrow S \cup R$
4. return $S$

The subalgorithm $G\_EXTRACT$ has two levels: the following simple approach extracts the solution from a non degenerate basis (dimension zero or higher). If it is called with a degenerate system, the fail exit is taken and the more complicated algorithm described in [18] is used instead.

$G\_EXTRACT$:
$Input : x = \{x_1 \dots x_n\}, G = \{g_1(x) \dots g_k(x)\}$ : Gröbner basis
$Output : S = solution\ set\ of \{g_i(x) = 0\}$
$V_i \Leftarrow Vars(g_i), i = 1 \cdots k$
$V_i \Leftarrow V_i \backslash V_j, i = 1 \cdots k-1, j = i+1 \cdots k$
if any $V_i = \emptyset$ exit **FAIL**
return $G\_EX\_R(G, V, k)$

$G\_EX\_R(G, V, m)$

$INPUT$ : $G$ Gröbner basis
$V$ $V_i$ candidate variable(s) in $g_i$
$n$ actual level
$OUTPUT$ : solution set of system $\{g_1 \cdots g_m\}$

if $m$=0 return {}
$v \Leftarrow first(V_m)$ % candidate variable
$denominators \Leftarrow Lc(g_m, v) \cup denominators$
$S = \{s_j\} \Leftarrow SOLVE\_UNI(g_m, v)$
return $\bigcup_j (s_j \cup G\_EX\_R(sub(s_j, G), V, m-1))$

The following operations are cited in the above algorithm:

- $Vars$ extracts the variables occurring in a polynomial,
- $Lc$ isolates the leading coefficient with respect to the given variable,
- $SOLVE\_UNI$ computes the roots of a polynomial in one variable as described before.

At the end of the first phase each $V_i$ contains exactly those variables of $g_i$ which do not occur in any of the following polynomials. Each of these can be used to construct a solution. In a non-degenerate zero dimensional case each $V_i$ contains exactly one variable. If one $V_i$ is empty, a degeneracy has been detected. In the recursive part of the subalgorithm on every level one polynomial is solved with respect to its leading variable. If the ideal does not have dimension zero, the leading coefficient will be the denominator of the solution.

For each solution value $s_j$ a separate recursive branch is used because the substitution of $s_j$ in the remaining polynomials establishes a local subcase. Of course, such substitutions can be performed only if explicit roots have been found. If no explicit form for the root(s) is available, the roots are formally described as generic solutions by the REDUCE form $roots\_of$; such forms are not propagated and the corresponding variable $v$ remains as a formal parameter in the subsequent solutions.

### 3.2 Coefficient Domains

Although the REDUCE Gröbner package can handle for computations over different coefficient domains, in the context of automatic equation solving it uses exclusively the ring $\mathbb{Z}$ (resp. the Gaussian integers if the equations have explicit imaginary components); if

the equation system has formal parameters $a$,$b$,$c$, $\cdots$, the polynomial ring $\mathbb{Z}(a, b, c \cdots)$ is used as coefficient domain (resp. the Gaussian integer polynomial ring over the parameters). Input expressions with coefficients from other domains (e.g. rational or rounded numbers) are converted to quotients of integer polynomials with common denominators and the numerator part is extracted for the ideal computations: on the one hand Buchberger's algorithm is most efficient when executed in the ring variant, on the other hand it is highly unstable if executed with truncated floating point numbers.

The actual REDUCE domain mode is used only during the final decomposition of univariate polynomials leading to different result forms:

- Default mode, rational mode: the roots are computed according to the standard strategy of `SOLVE`: if the polynomial has roots which can be expressed as explicit exact algebraic expression of limited size, the explicit form is used for the result and for the propagation to other polynomials of the result system. Example:

```
solve({x**2+y**2=1,a*x+b*y=c},{x,y});
```

```
          2    2    2
     SQRT(A  + B  - C )*A + B*C
{{Y=-----------------------------,
               2    2
              A  + B

              2    2    2
      - SQRT(A  + B  - C )*B + A*C
  X=---------------------------------},
                2    2
               A  + B

              2    2    2
      - SQRT(A  + B  - C )*A + B*C
 {Y=---------------------------------,
                2    2
               A  + B

          2    2    2
     SQRT(A  + B  - C )*B + A*C
  X=-----------------------------}}
              2    2
             A  + B
```

Otherwise the roots are expressed as implicit expressions using the operator *root_of*, which leads to a result form very close to the original Gröbner basis. Example:

```
     solve({x**2+y**2=1,x**3+y**4},{x,y});

              8     6       4       2
{{Y=ROOT_OF(Y_  + Y_  - 3*Y_  + 3*Y_  - 1,Y_),

        6      4      2
  X= - Y  - 2*Y  + 2*Y  - 1}}
```

Even if the equation system has only real coefficients the solution can have complex components. These are not suppressed independent of the current value of the switch *complex*.

- Rounded mode: the roots are computed as numerical approximations of high accuracy using the REDUCE package *ROOTS* [12] for those parts of the result which do not depend of formal parameters. As in standard mode eventually complex components of the result are returned. Example:

```
     on rounded,complex;
     solve({x**2+y**2=1,x**3+y**4},{x,y});

{{Y=1.62162206656*I,X= - 1.90516616771},
 {Y=-1.62162206656*I,X= - 1.90516616771},
 {Y=0.828166126903 - 0.38194280897*I,X =0.788104887232 + 0.401357867369*I},
 {Y=-0.828166126903 + 0.38194280897*I,X =0.788104887232 + 0.401357867369*I},
 {Y=0.828166126903 + 0.38194280897*I,X =0.788104887232 - 0.401357867369*I},
 {Y=-0.828166126903 - 0.38194280897*I,X =0.788104887232 - 0.401357867369*I},
 {Y=0.741417883452,X= - 0.671043606704},
 {Y= - 0.741417883452,X= - 0.671043606704}}
```

It should be stressed here that although the results in this mode look like approximations, the heart of the algorithm is performed with exact arithmetic and only the final phase is approximate.

- Complex mode: if the mode *complex* is active and if the equation system contains explicit imaginary components the ideal computations are performed using the ring of Gaussian integers.

# 4 Mapping Algebraic Equations to Polynomials

Several classes of algebraic equation systems can be transformed into equivalent pure polynomial systems which then can be solved by the polynomial solver. The methods

differ from case to case. They have in common that sub-expressions are replaced by new indeterminates to be added to the variable list and additional equations are generated.

The first step in the algorithm is a factorization of the input equations (transformed into expressions equated to zero); the system is split into smaller subsystems.

$SOLVE\_ALG$ :

$INPUT$ : $x = \{x_1 \cdots x_n\}, F = \{f_i(x) = 0\}$ algebraic equation system
$OUTPUT$ : solution set $S$ of system $F$
$f_i \Leftarrow lhs(f_i) - rhs(f_i)$
$\{F_j\} \Leftarrow Factorize(F)$
$return \bigcup_j SOLVE\_ALG * (F_j)$

Later in the Gröbner phase again all systems will be inspected for possible factorizations; in the input phase the factorization is performed mainly to separate variables or different levels of nonlinearity: e.g. in the form $(e^x - y^2) * (x^2 - y)$ each factor can be solved by polynomial methods while the complete expression cannot be converted to a polynomial system.

$SOLVE\_ALG*$ :

$INPUT$ : $x, F$ algebraic equation system
$OUTPUT$ : solution set $S$ of system $F$
while $\exists$ non polynomial term $u$ in $F$
select transformation $T_u$; if no $T_u$ exit **FAIL**
$F \Leftarrow T_u(F)$; $TT \Leftarrow TT \cup \{T_u\}$
$S \Leftarrow SOLVE\_POLY(F)$
for each $T$ in $TT$ do $S \Leftarrow T^{-1}(S)$
return($S$)

This phase transforms the system until all non polynomial relations have been resolved. The transformations are recorded in a database $TT$. The rewritten system is solved as a polynomial system and the inverse transformations are applied to the solutions. The transformations are described in the rest of this section.

### 4.1 Surds, Rational Exponents

Surds are a simplified notation for computations with elements which are solutions of algebraic equations. Consequently we can convert an equation system containing surds into an equivalent system free of surds by handling the surds as independent variables and adding their defining relation. E.g. the equation

$$x + \sqrt[3]{x+1} - 2 = 0$$

can be converted to the pure polynomial system

$$\{x + y - 2 = 0, y^3 = x + 1\}$$

which leads to the Gröbner basis

$$\{x + y - 2, y^3 + y - 3\}$$

The last polynomial has one real root $y \approx 1.21341$ and 2 complex roots; the first polynomial assigns $x \approx 0.78659$ as the sole real solution of the original equation.

It is obvious that such transformations can be performed automatically: as long as a surd is found in the system, it is eliminated by substituting a new variable and adding a defining equation. Formally:

$TR\_SURD$:
$Input : n, k, x = (x_1 \dots x_n), algebraic\ P = \{f_1(x) \dots f_k(x)\}$
$Output : \hat{n}, \hat{k}, \hat{x} = (x_1 \dots x_{\hat{k}}), polynomial\ \hat{P} = \{\hat{f}_1(\hat{x}) \dots \hat{f}_{\hat{k}}(\hat{x})\}$
  $IF\ some\ \sqrt[m]{u(x)} \in f_j(x), f_j(x) \in P :$
  $P \Leftarrow \{\ x_{n+1}^m - u(x)\} \cup (subst\ \sqrt[m]{u(x)}\ by\ x_{n+1}\ in\ P)$
    $x \Leftarrow x \cup \{x_{n+1}\},\ n \Leftarrow n+1,\ k \Leftarrow k+1$
    $REPEAT$

If the surds are nested, this process is iterated, e.g.

$$x + \sqrt{x + \sqrt{x+1}} - 2 = 0 \Longrightarrow$$
$$\{x + y - 2 = 0, y^2 = x + \sqrt{x+1}\} \Longrightarrow$$
$$\{x + y - 2 = 0, y^2 = x + z, z^2 = x + 1\}$$

This system has the Gröbner basis

$$\{x - z^2 + 1, y + z^2 - 3, z^4 - 7z^2 - z + 10\}$$

where the last polynomial has the roots $z \approx 1.274$, $z \approx -1.651$, $z \approx 2.377$, $z = -2$. Only $z \approx 1.274$ leads to a real solution $x \approx 0.6227$.

The transformation TR_SURD is not restricted to cases where one of the variables appears in the inner part of the root expression. E.g. the system

$$\left\{e = \frac{b}{4}, ed = \frac{\sqrt{\pi}c}{\sqrt{2}}, c = \frac{a}{2}, a = \sqrt{\pi}, bd = \pi\sqrt{2}\right\}$$

describes a polynomial ideal over a domain with one transcendental and two algebraic extensions $\mathbb{Z}[\pi][y,z]/(y^2-2, z^2-\pi)$. TR_SURD completes the system by

$$\left\{s_2 \Leftarrow \sqrt{2}, s_2^2 - 2 = 0, s_\pi \Leftarrow \sqrt{\pi}, s_\pi^2 - \pi = 0\right\}$$

which now describes an ideal in $\mathbb{Z}(\pi)$ and variables $a$,$b$,$c$,$d$,$e$,$s_2$,$s_\pi$. The system has one unique solution

$$\left\{c = \frac{\sqrt{\pi}}{2}, d = \frac{\sqrt{2}\pi}{4e}, a = \sqrt{\pi}, b = 4e\right\}$$

The solution has $e$ as a free parameter.

From a purely algebraic standpoint *TR_SURD* is a **weak transformation**; by raising an expression $u = \sqrt[m]{w}$ to the $m^{th}$ power we introduce an $m$-fold ambiguity caused by the $m^{th}$ roots of unity. Normally the user means the real positive root if he enters a surd. In order to prevent "false" solutions we eliminate all solutions which obviously don't lead to real positive values for surds. E.g. the full system for the equation $\sqrt[3]{x} + \sqrt{x} - 2 = 0$ has the formal solutions $x = 1, x = 8i, x = -8i$, but the selection mechanism here is applicable and allows only $x = 1$ as the sole solution. However, this distinction is impossible if the roots are not available explicitly.

In REDUCE, **rational exponents** are represented internally as integer powers of expressions with numerator 1 in the exponent, e.g. $x^{\frac{3}{2}} \rightarrow (x^{\frac{1}{2}})^3$; so they automatically are in polynomial form and don't need a special treatment.

If some base has different exponent denominators, e.g. $u^{\frac{3}{2}} + 2u^{\frac{2}{3}}$ there is a choice: we can transform every surd individually (here $u^{\frac{1}{2}}$ and $u^{\frac{1}{3}}$) or we can modify the exponents such that they have a common denominator (here $u^{\frac{9}{6}} + 2u^{\frac{4}{6}}$) saving auxiliary variables. Experiments have shown that the common denominator form does not reduce the complexity of the Gröbner calculation; therefore this transformation is not used.

The following system (Richter, Leipzig) contains several $\sqrt{3}$. It can be solved alge-

braically only if the constant surds are handled exactly.

$$128\sqrt{3}a^2b + 128\sqrt{3}b^3 + 288a^4 + 576a^2b^2 - 24a^2n - 384a^2 + 288b^4 - 24b^2n - 512b^2 + m + 40n + 640$$
$$16a\left(16\sqrt{3}b + 72a^2 + 72b^2 - 3n - 48\right)$$

$$16\left(8\sqrt{3}a^2 + 24\sqrt{3}b^2 + 72a^2b + 72b^3 - 3bn - 64b\right)$$
$$128\sqrt{3}a^2b + 128\sqrt{3}b^3 + 288a^4 + 576a^2b^2 - 48a^2n - 384a^2 + 288b^4 - 48b^2n - 512b^2 + 3m + 80n + 640$$

Full set of solutions:

$$b = 0,\ a = 0,\ n = -32,\ m = 640$$
$$b = root\_of\left(27b^6 - 171b^4 - 165b^2 - 400, b\right),$$
$$a = 0,\ n = \left(216b^4 + 3672b^2 - 8800\right)/255,$$
$$m = \left(11232b^4 - 32640b^2 + 42880\right)/51$$
$$b = \left(5\sqrt{3}\right)/14,\ a = \left(\sqrt{215}i\right)/14,\ n = (-272)/7,\ m = 54400/49$$
$$b = \left(5\sqrt{3}\right)/14,\ a = \left(-\sqrt{215}i\right)/14,\ n = (-272)/7,\ m = 54400/49$$
$$b = \left(-5\sqrt{3}\right)/14,\ a = \left(\sqrt{215}i\right)/14,\ n = (-272)/7,\ m = 54400/49$$
$$b = \left(-5\sqrt{3}\right)/14,\ a = \left(-\sqrt{215}i\right)/14,\ n = (-272)/7,\ m = 54400/49$$

## 4.2 Trigonometric Functions

For trigonometric function we have the choice between two different algebraic substitutions:

- tangent substitution:
  $\sin(\alpha) \Rightarrow \frac{2\tan(\frac{\alpha}{2})}{1+\tan(\frac{\alpha}{2})^2}$ $\cos(\alpha) \Rightarrow \frac{1-\tan(\frac{\alpha}{2})^2}{1+\tan(\frac{\alpha}{2})^2}$
  generating a system in one variable "$\tan(\frac{\alpha}{2})$" per angle.
- circle identity: couple $\sin(\alpha)$ and $\cos(\alpha)$ by adding the unit circle identity $\sin(\alpha)^2 + \cos(\alpha)^2 = 1$ to the equation system.

While the tangent substitution is well suited for solving many univariate trigonometric polynomials (and it is used in REDUCE in this context), many experimental calculations for robot systems have demonstrated the circle identity generates less complex calculations for systems of equations. The bad behavior of the tangent substitution in the Buchberger algorithm is probably caused by the denominator which introduces a

singularity and higher degree polynomials. Consequently we use the circle identity for completing a system with trigonometric functions.

$TR\_TRIG1$:
$Input: n, k, x = (x_1 \dots x_n), trig.\ P = \{f_1(x) \dots f_k(x)\}$
$Output: \hat{n}, \hat{k}, \hat{x} = (x_1 \dots x_{\hat{k}}), polynomial\ \hat{P} = \{\hat{f}_1(\hat{x}) \dots \hat{f}_{\hat{k}}(\hat{x})\}$
  $IF$ *some* $\sin(\alpha) \in f_j(x) \vee \cos(\alpha) \in f_j(x), f_j(x) \in P$ :
    $IF\ \alpha \in \{x_1 \dots x_n\}$ exit **FAIL**
    $P \Leftarrow \{ x_{n+1}^2 + x_{n+2}^2 - 1\} \cup (subst\ \sin(\alpha)\ by\ x_{n+1}, \cos(\alpha)\ by\ x_{n+2}\ in\ P)$
      $x \Leftarrow x \cup \{x_{n+1}, x_{n+2}\}, n \Leftarrow n+2,\ k \Leftarrow k+2$
      $IF\ \neg independent(\alpha, P)$ exit **FAIL**
    $REPEAT$

The **FAIL** exit is taken in cases where the system cannot be transformed to a polynomial one because of transcendental dependencies:

- a variable appears inside and outside of sin or cos, e.g. $y + sin(y)$,
- a variable appears inside of *sin* or *cos* in incompatible form, e.g. $\sin(z) + \sin(z^2)$.

In order to eliminate incompatibilites as far as possible the trigonometric solver first applies a global set of rewriting rules which decomposes sums and integer multiples of arguments by applying trigonometric identities; at the same time *tan* and *cot* are transformed into quotients of sin and cos:

$TR\_TRIG0$:
$\sin(\alpha + \beta) \Rightarrow \sin(\alpha)\cos(\beta) + \cos(\alpha)\sin(\beta)$
$\cos(\alpha + \beta) \Rightarrow \cos(\alpha)\cos(\beta) - \sin(\alpha)\sin(\beta)$
$\sin(n\alpha) \Rightarrow \sin(\alpha)\cos((n-1)\alpha) + \cos(\alpha)\sin((n-1)\alpha) \forall n > 1 \in \mathbb{Z}$
$\cos(n\alpha) \Rightarrow \cos(\alpha)\cos((n-1)\alpha) - \sin(\alpha)\sin((n-1)\alpha) \forall n > 1 \in \mathbb{Z}$
$\sin(\alpha)^2 \Rightarrow 1 - \cos(\alpha)^2, \tan(\alpha) \Rightarrow \frac{\sin(\alpha)}{\cos(\alpha)}, \cot(\alpha) \Rightarrow \frac{\cos(\alpha)}{\sin(\alpha)}$

An additional problem with trigonometric functions is the **back conversion** of the result: if the polynomial solver has found a solution for a transformed system giving an explicit algebraic relation e.g. $\sin(\alpha_1) = f_s(x)$, $\cos(\alpha_1) = f_c(x)$ the angle $\alpha_1$ is uniquely determined as soon as $f_s(x)$ and $f_c(x)$ are known. A more explicit form of these is the system $\alpha_1 = \arcsin(f_s(x))$ or $\alpha_1 = \arccos(f_c(x))$. Unfortunately none of these equations is sufficient because they only give access to $\alpha_1$ up to a multiple of $\pi$ with different principle value ranges. So we use *acos* for computing the absolute value of $\alpha_1$ and *asin* for the sign. Fortunately the relation $\alpha > 0 \leftrightarrow \arcsin(\alpha) > 0$ holds, so we can present an explicit form as

$TR\_TRIG2$: $Input$ : $expressions\ c, s$
$Output$ : $\alpha$ where $s = \sin(\alpha), c = \cos(\alpha)$
$\alpha = sign(s)\arccos(c)$

Note that the function *sign* applied to complicated formal expressions often allows a significant simplification by rules of algebraic sign propagation. The mechanism currently used in REDUCE is listed in the appendix.

Example for a trigonometric system ([11]):

$$\begin{aligned}
&\cos(x_1)\,x_3 - \sin(x_1)\,x_2 + 5\sin(x_1),\\
&-2\cos(x_1) + 2x_3^2x_2 + 2x_2^3 + x_2 - 5,\\
&-2\sin(x_1) + 2x_3^3 + 2x_3x_2^2 + x_3
\end{aligned}$$

The complete algebraic solution is

$$\{x_1 = (2j_1+1)\pi, x_3 = 0, x_2 = 1\} \tag{1}$$

$$\left\{x_1 = (2j_2+1)\pi, x_3 = 0, x_2 = \left(\sqrt{5}i - 1\right)/2\right\} \tag{2}$$

$$\left\{x_1 = 2j_3\pi, x_3 = 0, x_2 = \left(-\sqrt{5}i - 1\right)/2\right\} \tag{3}$$

$$\left\{x_1 = 2j_4\pi, x_3 = 0, x_2 = root\_of\left(2q^3 + q - 7, q\right)\right\} \tag{4}$$

$$\begin{aligned}
&\Big\{x_1 = one_of\left(2j_5\pi + \arcsin\frac{21i}{20}, 2j_6\pi - \arcsin\frac{21i}{20} + \pi\right),\\
&\quad x_2 = one_of\left(-\arccos\frac{29}{20} + 2j_7\pi + \pi, \arccos\frac{29}{20} + 2j_8\pi - \pi\right),\\
&\quad x_3 = \frac{21i}{10}, x_2 = \frac{21}{10}\Big\}
\end{aligned} \tag{5}$$

$$\begin{aligned}
&\Big\{x_1 = one_of\left(2j_9\pi - \arcsin\frac{21i}{20}, 2j_{10}\pi + \arcsin\frac{21i}{20} + \pi\right),\\
&\quad x_2 = one_of\left(-\arccos\frac{29}{20} + 2j_{13}\pi + \pi, \arccos\frac{29}{20} + 2j_{11}\pi - \pi\right),\\
&\quad x_3 = \frac{-21i}{10}, x_2 = \frac{21}{10}\Big\}
\end{aligned} \tag{6}$$

The solutions (2),(3),(5) and (6) represent some complex values and consequently for (5) and (6) no unique angle can be assigned for $x_1$. The roots of the polynomial $2q^3 + q - 7$ are $\{q \approx 1.40873, q \approx -0.70436 + 1.4101i, q \approx -0.70436 - 1.4101i\}$, so we can proove algebraically that the system has exactly two real solutions (1) and (4) using $q \approx 1.40873$ as root in (4). Note that the *sign* expressions have been eliminated completely from the solutions in explicit form.

Similar to the surd case the transformations TR_TRIG0 and TR_TRIG1 also apply for **"constant" expressions** or parameters built with trigonometric functions; if e.g. in a system there are constants $\sin(1)$ and $\cos(1)$ the computation has to be performed modulo $\sin(1)^2 + \cos(1)^2 - 1 = 0$. The easiest way is to transform these by TR_TRIG0 introducing auxiliary variables for the constants. Formally the system afterwards has an additional degree of freedom increasing the dimension of the ideal by 1. This is no real problem as long as the auxiliary variables are kept in the last position during Gröbner calculations: they appear as "free" parameters in the bases and during backward conversion they will be re-transformed as formal constants. Only if the original system had no solution the transformed system might have a dimension $> 0$. But such cases are easily detected as they then establish obviously false algebraic relations for the constants. E.g. the system

$$\{\sin(1)x - 1 = 0, \cos(1)y - 1 = 0, x^2 - y = 0\}$$

has no solution. If we add $sin(1)$ and $cos(1)$ as additional variables and extend the system by $sin(1)^2 + cos(1) - 1 = 0$ the Gröbner basis is

$$\begin{aligned}&\{x - \sin(1) * \cos(1) - \sin(1),\\ &\ y - \cos(1) - 1, \sin(1)^2 - \cos(1),\\ &\ \cos(1)^2 + \cos(1) - 1\}\end{aligned}$$

The last polynomial proposes $cos(1)$ as root of a quadratic polynomial which is an obvious contradiction. Such false solutions can be sorted out automatically during post-processing.

### 4.3 General Exponentials

If a variable $z$ appears inside a transcendental function which has a general inverse, this system can be solved as long as the variable $z$ appears only in one context, which, however, might occur more than once in the system and in different algebraic relations. E.g. the system

$$a^x - y, y(y-1)$$

can be solved by handling $a^x$ as free variable $u$, solving the polynomial system $u-y, (y-2)(y-1)$ invert the solution for $u$ by logarithms:

$$\begin{aligned}&\{y = 2, x = (2j_1 i\pi + \ln 2)/\ln a\}\\ &\{y = 1, x = (2j_2 i\pi)/\ln a\}\end{aligned}$$

As this transformation is possible only if the variable appears only in one context, transformation rules for unifying the transcendental function expressions are applied. E.g. for exponentials with a common basis the exponents are separated automatically

(that is done by REDUCE) and for each variable the exponential is transformed into a common denominator form, e.g.

$$a^{2x} - a^{\frac{x}{2}} + a^{\frac{x}{3}} \Rightarrow (a^{\frac{x}{6}})^1 2 - (a^{\frac{x}{6}})^3 + (a^{\frac{x}{6}})^2$$

which then is a polynomial in $a^{\frac{x}{6}}$; solutions for $x$ then can be inverted using logarithms.

# 5 Conclusion

The REDUCE `SOLVE` facilities for nonlinear equations and equation systems have reached a rather complete level. The following types of systems can be solved fully automatically as long as no removable transcendental dependencies are involved:

- pure polynomial systems,
- systems with trigonometric functions,
- systems with surds,
- systems with general exponentials and other invertible transcendental functions,
- mixtures of these,
- with numerical or parametric coefficients,
- with isolated, parametric or mixed solutions.

Of course, there are restrictions in capacity as Gröbner basis computations can require enormous amounts of computing power - improving this important field of computer algebra remains a global task. Another very important topic for future development will be the simplification of formal expressions with transcendental functions, as the formal solutions generated byt the automatism are often in an unpleasant and unnecessarily complicated shape.

# 6 APPENDIX

## 6.1 Sign Propagation

The following rules define an inference mechanism for the sign of a composite algebraic form. The function *sign* takes values $-1, 0, 1$ for negative, zero resp. positive values. For an algebraic expression $u$ the expression $sign(u)$ is a number if and only if the sign of $u$ is known; otherwise the expression $sign(u)$ remains unevaluated. The rules are

interpreted as a rewriting system: if the condition is fulfilled, the expression on the lefthand side of $\Rightarrow$ is replaced by the expression on the righthand side.

$$sign(x) \Rightarrow \begin{cases} -1 & x < 0 \\ 0 & x = 0 \quad IF\ x\ is\ real\ number \\ 1 & x > 0 \end{cases}$$
$$sign(-x) \Rightarrow -sign(x)$$
$$sign(x+y) \Rightarrow sign(x) \quad IF\ sign(x) = sign(y)$$
$$sign(xy) \Rightarrow sign(x) * sign(y) \quad IF\ sign(x) \in \mathbb{Z} \vee sign(y) \in Z$$
$$sign(\tfrac{x}{y}) \Rightarrow sign(x) * sign(y) \quad IF\ sign(x) \in \mathbb{Z} \vee sign(y) \in Z$$
$$sign(x^n) \Rightarrow 1 \quad IF\ \tfrac{n}{2} \in \mathbb{Z} \wedge n > 0$$
$$sign(x^n) \Rightarrow sign(x) \quad IF\ \tfrac{n+1}{2} \in \mathbb{Z} \wedge n > 0$$
$$sign(x^y) \Rightarrow 1 \quad IF\ sign(x) = 1$$
$$sign(x) \Rightarrow sign(approx.x) \quad IF\ x\ is\ constant$$

The rule set terminates because all righthand sides are less complicated than the lefthand sides. The conditions ensure that rewriting takes place only if the number of sign operators is not increased. The last rule is specific for the algebraic context: Although representing a constant value an expression can be composite, e.g $1 - \frac{1}{\sqrt{2}}$; for sign evaluation such expressions are approximated by usual numerical techniques. So we can evaluate $sign(1 - \frac{1}{\sqrt{2}}) \Rightarrow 1$ although we have no rule for propagating the sign over a sum with terms of different signs.

## References


[1] W. Boege, R. Gebauer, and H. Kredel. Some examples for solving systems of algebraic equations by calculating Groebner bases. *J. Symbolic Computation*, 2(1):83–98, March 1986.

[2] B. Buchberger. Groebner bases: An algorithmic method in polynomial ideal theory. In N. K. Bose, editor, *Progress, directions and open problems in multidimensional systems theory*, pages 184–232. Dordrecht: Reidel, 1985.

[3] B. Buchberger. Applications of groebner bases in non-linear computational geometry. In R. Janssen, editor, *Trends in Computer Algebra*, pages 52–80. Berlin, Heidelberg, 1988.

[4] H. Caprasse, J. Demaret, K. Gatermann, and H. Melenk. Power-law type solutions of fourth-order gravity for multidimensional Bianchi I universes. *International Journal of Modern Physics C*, 2(2):601–611, 1991.

[5] J. H. Davenport, Y. Siret, and E. Tournier. *Computer Algebra, Systems and Algorithms for Algebraic Computation*. Academic Press, 1989.

[6] A. Dress. Private communication. 1988.

[7] Rüdiger Gebauer and H. Michael Möller. On an installation of Buchberger's algorithm. *J. Symbolic Computation*, 6(2 and 3):275–286, 1988.

[8] H.M. Möller H. Melenk and W. Neun. Groebner: A package for calculating groebner bases. *REDUCE 3.4 documentation*, 1991.

[9] Jarmo Hietarinta. Private communication. 1992.

[10] Jarmo Hietarinta. Solving the Yang-Baxter equation in 2 dimensions with massive use of factorizing gröbner basis computations. In *Proc. ISSAC '92*, pages 350–357. ACM Press, 1992.

[11] F. Kalovics. A reliable method for solving nonlinear systems of equations of few variables. *Computing*, 48(2-3):291–302, 1992.

[12] S. L. Kameny. The reduce root finding package. *REDUCE 3.4 documentation*, 1990.

[13] H. Kredel and V. Weispfennig. Computing dimension and independent sets for polynomial ideals. *J. Symbolic Computation*, 6(1):231–247, November 1988.

[14] H. Melenk. Solving polynomial equation systems by groebner type methods. *CWI Quarterly*, 3(2):121–136, June 1990.

[15] H. Melenk. Practical application of gröbner bases for the solution of polynomial equation systems. In V. P. Gerdt D. V. Shirkov, V. A. Tostovtsev, editor, *IV. International Conference on Computer Algebra in Physical Research, 1990*, pages 230–235, Singapore, 1991. World Scientific.

[16] H. Melenk. Polynomial ideals. *REDUCE Network Library*, May 1992.

[17] H. Melenk, H. M. Möller, and W. Neun. Symbolic solution of large stationary chemical kinetics problems. *Impact of Computing in Science and Engineering*, 1(2):138–167, June 1989.

[18] H. M. Möller. On decomposing systems of polynomial equations with finitely many solutions. Preprint SC 92-15, Konrad-Zuse-Zentrum für Informationstechnik Berlin, June 1992.