Applications of CRACK in the Classification of Integrable Systems

Thomas Wolf
Department of Mathematics, Brock University
500 Glenridge Avenue, St.Catharines, Ontario, Canada L2S 3A1
email: twolf@brocku.ca

December 11, 2002

1 Overview

The talk given by the author at the CRM workshop on Superintegrability in Sep. 2002 and this related paper report on work in two subjects. One is the collaboration with Vladimir Sokolov and Takayuki Tsuchida in an effort to classify polynomial integrable vector evolution equations. The other is the computer algebra package CRACK which did the main computations in solving large bi-linear algebraic systems. Although originally designed to solve over-determined systems of partial differential equations a number of extensions made CRACK a powerful tool for solving systems of bi-linear algebraic equations. Such systems turn up in many different classification problems some of which were investigated by other participants of this workshop. In sections 5 and 6 two additional applications are outlined.

In the talk on which this article is based a method to reduce the length of equations was presented which proved to be useful in solving the bi-linear algebraic systems. Due to numerous asked questions about the computer program, a more complete overview is given in the appendix.

2 The classification of integrable vector evolution equations

The method to use symmetries to classify non-linear evolutionary 1+1 dimensional PDEs is the most productive one and known for some time (see [1, 2, 3, 4]).

An extension of the simplest approach [7, 8] to the case of so-called vector evolution equations is described in work with Vladimir Sokolov [5]. Examples of vector evolution equations are two different vector generalisations

$$U_t = U_{xxx} + \langle U, U\rangle U_x, \tag{1}$$
$$U_t = U_{xxx} + \langle U, U\rangle\, U_x + \langle U, U_x\rangle\, U \tag{2}$$

of the mKdV equation where $U(t, x)$ is an N-component vector and $\langle\cdot,\cdot\rangle$ stands for the standard scalar product.

In the performed symmetry classification we considered equations of the form

$$U_t = f_n\, U_n + f_{n-1}\, U_{n-1} + \cdots + f_1\, U_1 + f_0\, U, \qquad U_i = \frac{\partial^i U}{\partial x^i}. \tag{3}$$

where $U = (U^1, U^2, \ldots, U^N)$ is an unknown vector of arbitrary dimension N and coefficients f_i are polynomials of scalar products $\langle U_i,\, U_j\rangle$, $0 \le i \le j \le n$.

For more details on possible scalar products, orthogonal symmetry transformations and generality of N see [5].

All such equations were determined that omitted a symmetry of the same form

$$U_\tau = g_m\, U_m + g_{m-1}\, U_{m-1} + \cdots + g_1\, U_1 + g_0\, U$$

where the differential order n of the equation and the order m of the symmetry have selected values. By taking f_i, g_j to be homogeneous polynomials of the scalar products one achieves that the symmetry condition $U_{t\tau} = U_{\tau t}$ yields an over-determined system of bi-linear algebraic conditions for the un-determined coefficients of both polynomials f_i and g_i.

The number of coefficients of the equation and symmetry is reduced further by a homogeneity assumption that the differential equations and symmetries are invariant under the scaling group

$$(x,\ t,\ U) \longrightarrow (a^{-1}x,\ a^{-\mu}t,\ a^{\lambda}U).$$

with one value λ for all components of the vector U.

Finally, we restrict λ based on results for the scalar case proven in [9] that a λ-homogeneous polynomial equation with $\lambda > 0$ may possess a homogeneous polynomial higher symmetry only if $\lambda \in \{2, 1, \frac{1}{2}\}$.

The differential orders of the equation and symmetry that have been investigated are also motivated by the scalar case where infinite commutative hierarchies have either a lowest order of the equation of 2 and symmetry of order 3 (in the following called type (2,3) like the Burgers equation) or an equation of order 3 with a symmetry of order 5 (type (3,5) like the Korteweg–de Vries (KdV) equation) or an equation of order 5 with a symmetry of order 7 (type (5,7) like the Kaup–Kupershmidt and Sawada–Kotera equations).

For the nine resulting cases shown in table 1 computer programs generated the equations and symmetries, computed the commutator and formulated and solved the bi-linear system automatically (apart from the largest case $\lambda = \frac{1}{2}$, type (5,7) where the solution was not fully automatic).

λ	2			1			1/2		
orders of (eq,sym)	(2,3)	(3,5)	(5,7)	(2,3)	(3,5)	(5,7)	(2,3)	(3,5)	(5,7)
# of unknowns (eq,sym,tot)	-	2	3,7,10	2,3,5	3,9,12	9,24,33	4,8,12	8,27,35	27,82,109
# of equations	-	4	34	5	26	198	21	129	927
total # of terms	-	4	162	9	121	3125	80	1603	52677
av. # of terms/equ.	-	1	4.7	1.8	4.6	15.8	3.8	12.4	56.8
time to formulate	-	0.2s	1.7s	0s	1.3s	1m 1s	1s	17s	13h 12m
time to solve	-	0s	0.7s	0s	0.5s	1m 15s	0.4s	32s	2 days
solutions	-	-	-	-	(1),(2)	-	-	(4)	-

Table 1. A classification of single vector equations.

Comments: Times are measured on a 1.7GHz Pentium 4 running a 120MByte Reduce session under Linux. Nonlinear $\lambda = 2$ equations of order 2 do not exist. All found solutions have already been known: equations (1), (2) are vector generalizations of the mKdV equation, and equation

$$U_t = U_{xxx} + 3\langle U, U\rangle U_{xx} + 6\langle U, U_x\rangle U_x + 3\langle U, U\rangle^2 U_x + 3\langle U_x, U_x\rangle U \tag{4}$$

is a vector analogue (reported in [10]) of the Ibragimov-Shabat-Calogero equation [7, 6]. Solutions of order (5,7) are only symmetries of found lower order equations and have therefore not been listed in the table.

3 NLS Systems with two Vector Unknowns

More successful has been the investigation of integrable vector NLS-type systems of the form

$$\begin{cases} U_t = \quad U_{xx} + p_1 U_x + p_2 V_x + p_3 U + p_4 V, \\ V_t = -V_{xx} + p_5 U_x + p_6 V_x + p_7 U + p_8 V, \end{cases} \tag{5}$$

where U and V are vectors and the coefficients p_i are λ-homogeneous polynomials depending on all possible scalar products of vectors U, V, U_x, V_x. For $\lambda = 2$ second order systems can only be linear but for $\lambda = 1$ or $1/2$ results are shown in table 2. Just as in the scalar case (see [2]) a symmetry of the form

$$\begin{cases} U_\tau = U_{xxx} + q_1 U_{xx} + q_2 V_{xx} + q_3 U_x + q_4 V_x + q_5 U + q_6 V, \\ V_\tau = V_{xxx} + q_7 U_{xx} + q_8 V_{xx} + q_9 U_x + q_{10} V_x + q_{11} U + q_{12} V, \end{cases} \tag{6}$$

is assumed where the coefficients q_i are λ-homogeneous polynomials of all possible scalar products of $U, V, U_x, V_x, U_{xx}, V_{xx}$.

λ	1	1/2
orders of (eq,sym)	(2,3)	(2,3)
# of unknowns (eq,sym,tot)	9,15,24	53,155,208
# of equations	78	1206
total # of terms	242	28768
av. # of terms/equ.	3.1	23.8
time to formulate	2.3s	1h
time to solve	2.9s	26m 15s
# of solutions	2	7

Table 2. A classification of NLS-type systems of vector equations.

Results:
The 2 solutions for $\lambda = 1$ are known ([11, 12]). They are special cases (see [5]) of a generalization of the NLS system by Svinolupov using a Jordan triple system (in [13]). For $\lambda = \frac{1}{2}$ after identifying solutions through $U \leftrightarrow V,\ t \leftrightarrow -t$ six solutions remain. Two of these systems had been known ([22]). For the remaining four systems, Takayuki Tsuchida showed C-integrability for two of them and S-integrability for the other two.

4 Systems with one Scalar and one Vector Unknown

Computations to classify single vector equations that involved an arbitrary constant vector did not give new results but the possibility to apply an orthogonal transformation to make the constant vector equal (1,0,0,...) provided a natural split of the single vector equation into one for a scalar function u (equal the former component U^1) and a new vector function U (equal the remaining components $(U^2, U^3, ...)$). General investigations of systems with one scalar and one vector brought surprisingly a rich set of integrable systems. Results in the case $\lambda = 2$ were analysed by Vladimir Sokolov and Takayuki Tsuchida. What appears to be a new solution in this class is the vector generalization

$$\begin{cases} u_t = u_{xxx} + 6uu_x - 12\langle U, U_x\rangle, \\ U_t = -2U_{xxx} - 6uU_x. \end{cases} \tag{7}$$

of the two-component coupled KdV system proposed by Hirota and Satsuma [18]. A Lax pair representation has been found by Takayuki Tsuchida.

Especially the case of $\lambda = 1$ with 25 solutions analysed by Takayuki Tsuchida posed a major challenge. Not all systems are of interest. Some become triangular by defining $\langle U, U\rangle$ as a new scalar variable. Others are just the result of splitting a single vector equation into scalar + vector equations for the scalar U^1 and the vector $(U^2, U^3, \ldots, U^N)$. From the interesting cases just one should be shown here:

$$\begin{cases} u_t = u_{xxx} - 6u^2 u_x + u_x\langle U, U\rangle + 2u\langle U, U_x\rangle + \langle U, U_{xx}\rangle + \langle U_x, U_x\rangle, \\ U_t = -2U_{xxx} - 6u_{xx}U - 6u_x U_x + 12uu_x U + 6u^2 U_x + \langle U, U\rangle U_x \\ \qquad - 2\langle U, U_x\rangle U. \end{cases} \tag{8}$$

Its integrability can be established (Tsuchida) through a change to new variables w and W

$$\begin{cases} w = u_x + u^2 + \frac{1}{6}\langle U, U\rangle, \\ W = U_x + 2uU, \end{cases} \tag{9}$$

which satisfy the following system:

$$\begin{cases} w_t = w_{xxx} - 6ww_x + 2\langle W, W_x\rangle, \\ W_t = -2W_{xxx} + 6wW_x. \end{cases} \tag{10}$$

This system coincides up to a scaling of variables with the coupled Hirota–Satsuma system (7) found in the $\lambda = 2$ investigation above.

The Miura-type transformation (9) is a generalization of the one for scalar U in [19, 20] and the one for two-component vector U in [21].

In the case when U is a scalar variable, one can set

$$u = -\frac{1}{2}(q + r), \quad U = \frac{\sqrt{6}}{2}\mathrm{i}(q - r)$$

and rewrite (8) as a system of symmetrically coupled mKdV equations:

$$\begin{cases} q_t = [-\frac{1}{2}q_{xx} + \frac{3}{2}r_{xx} + 3(q - r)q_x - 2r^3]_x, \\ r_t = [\frac{3}{2}q_{xx} - \frac{1}{2}r_{xx} - 3(q - r)r_x - 2q^3]_x. \end{cases} \tag{11}$$

This system is identical to (63) in [16] or (3.22) in [17]. It was found in connection with the Kac–Moody Lie algebras and written in the Hamiltonian form about twenty years ago (cf. the $\mathrm{C}_2^{(1)}$ case in [19] or the $\mathrm{B}_2^{(1)}$ case in [20]).

The following table gives an overview of systems for one scalar and one vector found by Crack to have higher order symmetries. Details will be discussed in a future contribution.

λ	2			1			1/2		
orders of (eq,sym)	(2,3)	(2,4)	(3,5)	(2,3)	(2,4)	(3,5)	(2,3)	(2,4)	(3,5)
# of unknowns (eq,sym,tot)	5,6,11	5,12,17	6,17,23	10,21,31	10,39,49	21,74,95	15,36,51	15,79,94	36,164,200
# of equations	13	26	50	66	123	386	149	313	1154
total # of terms	34	77	218	341	770	5000	1093	3096	27695
av. # of terms/equ.	2.6	2.9	4.3	5.1	6.3	13	7.3	9.9	24
time to formulate	0.5s	1s	5s	1.8s	5s	2m 52s	8s	48s	2h 7m
time to solve	0.5s	0.4s	6.5s	29s	1m 58s	5h 47m	29s	3m 44s	1 day
# of solutions	0	0	4	3	3	25	0	0	2

λ	scalar: 1/3, vec: 2/3			scalar: 2/3, vec: 1/3		
orders of (eq,sym)	(2,3)	(2,4)	(3,5)	(2,3)	(2,4)	(3,5)
# of unknowns (eq,sym,tot)	10,24,34	10,54,64	24,115,139	13,22,35	13,66,79	22,126,148
# of equations	102	215	798	114	276	955
total # of terms	529	1462	12694	694	2435	17385
av. # of terms/equ.	5.2	6.8	16	6.1	8.8	18
time to formulate	3.2s	13s	23m 45s	6.3s	48s	41m 18s
time to solve	45s	1m 23s	1h 20m	22s	3m 40s	1h 7m
# of solutions	0	0	0	1	2	2

Table 3. A classification of systems of one scalar + one vector equations.

5 Classification of Integrable Hamiltonians

In work done together with Olga V. Efimovskaya quadratic Hamiltonians H have been investigated that have the form

$$H = \langle U,\, AU\rangle + \langle U,\, BV\rangle + \langle V,\, CV\rangle + \langle N,\, U\rangle + \langle M,\, V\rangle, \tag{12}$$

where $U = (U_1, U_2, U_3)$ and $V = (V_1, V_2, V_3)$ are three dimensional vectors, A, C are symmetrical matrices, B is an arbitrary matrix and N, M are constant vectors. Such Hamiltonians are relevant in the dynamics of rigid bodies.

The equations of motion in the rigid body dynamics are defined by a linear Poisson bracket of the form

$$\{Y_i,\, Y_j\} = c_{ij}^k\, Y_k, \qquad i, j, k = 1, \ldots, N \tag{13}$$

where c_{ij}^k are some constants. The evolution of dynamic variables $Y_1, \ldots, Y_N$ is defined by the formula

$$\frac{d}{dt} Y_i = \{Y_i,\, H\},$$

where H is the Hamiltonian.

The skew-symmetricity and the Jacobi identity for the linear Poisson bracket is equivalent to the fact that c_{ij}^k are the structural constants of some Lie algebra. It is known that the Hamiltonian structure of most cases of rigid body dynamics can be defined by the linear Poisson brackets

$$\{U_i, U_j\} = \varepsilon_{ijk}\, U_k, \qquad \{U_i, V_j\} = \varepsilon_{ijk}\, V_k, \qquad \{V_i, V_j\} = 0, \tag{14}$$

corresponding to the Lie algebra $e(3)$. For example, two classical problems with a Hamiltonian of form (12) and the Poisson structure (14) are

1. the Kirchhoff problem (where $N = 0, M = 0$ in H),
2. the problem of motion of a massive rigid body around a fixed point. In this case $B = C = N = 0$.

The bracket (14) possesses two Casimir functions:

$$J_1 = V_1^2 + V_2^2 + V_3^2, \qquad J_2 = U_1 V_1 + U_2 V_2 + U_3 V_3. \tag{15}$$

Therefore to integrate a system on $e(3)$ we need one additional first integral I, functionally independent of H, J_1, J_2. All Hamiltonians (12) admitting an additional polynomial first integral of first or second degree are well known. The main goal of our work is to find all Hamiltonians (12) that admit an additional polynomial integral of degree 3 or 4.

There exist two kinds of linear transformations of U and V which preserve the Poisson structure (14) and the form of H. The first kind is defined by

$$\bar{U} = T\, U, \qquad \bar{V} = T\, V, \tag{16}$$

where T is an arbitrary constant orthogonal matrix. The second kind is defined by

$$\bar{U} = U + S\, V, \tag{17}$$

where S is an arbitrary antisymmetric matrix. With the transformations (16) we reduce the matrix A to diagonal form: $A = diag(a_1, a_2, a_3)$. Transformations (17) are usually used for a simplification of matrix B.

The generic case $a_1 \neq a_2 \neq a_3 \neq a_1$ has been investigated in detail and all cases when there exists an additional polynomial integral are known. In calculations mentioned below we study the case $a_1 = a_2 \neq a_3$. Using transformation (17) we can have $b_{12} = b_{13} = b_{23} = 0$. Subtracting multiples of both casimirs enables $b_{11} = c_{11} = 0$ and gives a Hamiltonian of the form

$$\begin{aligned} H &= U_1^2 + U_2^2 + a_3 U_3^2 + \\ &\quad 2b_{21}U_2V_1 + 2b_{31}U_3V_1 + 2b_{32}U_3V_2 + 2b_{22}U_2V_2 + 2b_{33}U_3V_3 + \\ &\quad 2c_{12}V_1V_2 + 2c_{13}V_1V_3 + c_{22}V_2^2 + 2c_{23}V_2V_3 + c_{33}V_3^2 + \\ &\quad p_1U_1 + p_2U_2 + p_3U_3 + q_1V_1 + q_2V_2 + q_3V_3. \end{aligned} \tag{18}$$

We consider Hamiltonians (18) that have an additional cubic integral. The ansatz for a general first integral I of third degree involves 80 terms. Together with the 16 unknown constants in (18) the bi-linear algebraic system which results from $\{H, I\} = 0$ involves 96 unknowns to be determined. The result of the computation is summarized in the following theorem.

Theorem 1.

- *The Hamiltonian* (18) *admits a polynomial integral of third degree iff it has the form*

$$H = U_1^2 + U_2^2 + s_1U_3^2 + s_2V_3U_3 + s_3V_3^2 + s_4U_3 + s_5V_3, \tag{19}$$

  where s_i are arbitrary parameters;

- *The Hamiltonian* (18) *admits a polynomial integral of third degree on a special level $J_2 = 0$ of Casimir function (15) iff either it has the form (19) or the form*

$$\begin{aligned} H = U_1^2 + U_2^2 + 4U_3^2 + 4(s_1V_1 + s_2V_2)U_3 - (s_1^2 + s_2^2)V_3^2 + \\ s_3U_3 + s_4V_1 + s_5V_2, \end{aligned} \tag{20}$$

  where s_i are arbitrary parameters.

Hamiltonian (18) is a trivial generalization of the Hamiltonian for the Lagrange and Kirchhoff classical integrable cases. Actually Hamiltonian (18) admits not only a third degree but also an additional first degree integral $I = U_3$. If $s_1 = s_2 = 0$ then Hamiltonian (20) describes the so-called Goryachev-Chaplygin case in the problem of motion of a rigid body around a fixed point. The integrability of the general Hamiltonian (18) has been recently established in [14].

6 Non-local 2+1 Dimensional Equations

The computation decribed in this section solves only a first special case of a wider problem. We still show it as it gives an example of how even non-local 2+1 dimensional classification problems can be reduced to the solution of bi-linear algebraic systems for which CRACK can be useful.

The Kadomtsev–Petviashvili equation

$$u_{tx} = -(6uu_x + u_{xxx})_x + u_{yy}$$

can also be written as

$$u_t = -(6uu_x + u_{xxx}) + \Delta^2 u_x$$

with $\Delta = D_x^{-1} D_y$. A. Mikhailov and R.I. Yamilov observed in [15] that all known integrable 2+1 - dim equations can be written as

$$u_t = \text{expression in } \Delta, D_x \text{ and } u.$$

Based on this idea A. Mikhailov, V.V. Sokolov and R. Hernandez Heredero did work at classifying KdV-type integro-differential equations of the form

$$\begin{aligned} u_t &= au_{xxx} + b\Delta(u_{xxx}) + c\Delta^2(u_{xxx}) + ... \\ &\quad +e\Delta^{-3}(u_{xxx}) + \text{terms of lower } x\text{-order} \end{aligned}$$

which have symmetries of the same form.

The special ansatz for equation plus symmetry that has been investigated with Crack is

$$\begin{aligned} u_t &= u_{xxx} + b\Delta(u_{xxx}) + c\Delta^2(u_{xxx}) \\ &\quad +0\cdot\Delta^3(u_{xxx}) + e\Delta^{-1}(u_{xxx}) + \text{lower order} \\ u_\tau &= 0\cdot u_{xxx} + b'\Delta(u_{xxx}) + c'\Delta^2(u_{xxx}) \\ &\quad +\Delta^3(u_{xxx}) + e'\Delta^{-1}(u_{xxx}) + \text{lower order} \end{aligned} \tag{21}$$

with two coefficients equal one and two zero coefficients due to suitable linear combinations of both equations. The lower order terms can be of x-order 2 at most, linear or quadratic in u, and with Δ restricted as above.

For this ansatz, which is the simplest in this class, the condition $u_{t\tau} - u_{\tau t} \equiv 0$ provides already 2865 bilinear algebraic conditions for the 70 unknowns $b, c, \ldots$ and 70 unknowns $b', c', \ldots$.
According to CRACK only 2 solutions exist. One is the Boiti, Leon, Manna, Pempinelli equation

$$u_t = u_{xxx} + \alpha D_x(u\Delta^{-1}(u))$$

with symmetry

$$u_\tau = \Delta^3(u_{xxx}) + D_x(\Delta(u\Delta(u)))$$

The second solution is the same for $u \to \Delta u$.

Other known 2-dimensional integrable equations, like KdV in the form $u_t = \Delta u_{xxx} + 4u\Delta u_x + 2u_x\Delta u$ which is a symmetry of the usual KdV equation would not have terms u_{xxx} and $\Delta^3(u_{xxx})$ as it was required in the special ansatz (21).

Appendix: A short description of Crack

Any identifiers or numbers in curled brackets { } provided at the end or within the following paragraphs refer to key-words, file names, module numbers or flags which can be looked up in the Crack manual `crack.tex` (see below under 'Availability') or even be searched in the source code if needed.

Philosophy: The program CRACK is a computer algebra package written in REDUCE for the solution of over-determined systems of algebraic, ordinary or partial differential equations with at most polynomial non-linearity. It was originally developed to run automatically and effort was taken for the program to decide which computational steps are to be done next with a choice between integrations, separations, substitutions and investigation of integrability conditions. It is known from hand computations that the right sequence of operations with exactly the right equations at the right time is often crucial to avoid an explosion of the length of expressions. This statement keeps its truth for the computerized solution of systems of equations as they become more complex. As a consequence more and more interactive access has been provided to inspect data, to specify how to proceed with the computation and how to control it. This allows the human intervention in critical stages of the computations. {*off batch_mode*}

General Structure: A problem consists of a system of equations and a set of inequalities. With each equation are associated a short name and numerous data, like size, which functions, derivatives and variables occur but also which investigations have already been done with this equation and which not in order to avoid unnecessary duplication of work. These data are constantly updated if the equation is modified in any way.

A set of about 30 modules is available to integrate, substitute, decouple, ... equations. A complete list can be inspected in interactive mode with the command *p2*, each operation is listed with the number it is called. All modules can be called interactively or automatically. Automatic computation is organized by a priority list of modules (each represented by a number) where modules are invoked in the order they appear in the priority list, each module trying to find equations in the system it can be applied to. If a module is not successful then the next module in the list is tried, if any one is successful then execution starts again at the beginning of the priority list. { *prog_list_, default_proc_list_, full_proc_list_*}

Because each module has access to all the data, it is enough to call a module by its number. For example, the input of the number 2 in interactive mode will start the direct separation module (see below) to look for a directly separable equation and will split it.

Modules: The following modules are represented by numbers in the priority list. Each module can appear with modifications under different numbers. For example, integration is available under 7, 24 and 25. Here 7 encodes an integrations of short equations $0 = \partial^n f/\partial x^n$. 7 has highest priority of the three integrations. 24 encodes the integration of an equation that leads to the substitution of a function and 25 refers to any integration and has lowest priority.

Integration and Separation: An early feature in the development of the package CRACK was the ability to integrate exact differential equations and some generalizations of them (see [25]). As a consequence of integrations an increasing number of functions of fewer variables is introduced which sooner or later produces equations with some independent variables occuring only explicitly and not as variables in functions. Such equations are splitted by the integration module.

Substitutions: Substitutions can have a dramatic effect on the size and complexity of systems. Therefore it is possible to have them not only done automatically but also controlled tightly, either by specifying exactly which unknown should be substituted where using which equation, or by picking a substitution out of a list of substitutions offered by the program {*cs*} . Substitutions to be performed automatically can be controlled with a number of filters, for example, by

- limiting the size of the equation to be used for substitution, {*length_limit*}
- limiting the size of equations in which the substitution is to be done, {*pdelimit*}

- allowing only linear equations to be used for substitutions, {*lin_subst*}
- allowing equations to increase in size only up to some factor in order for a substitution to be performed in that equation, {*cost_limit*}
- allowing a substitution for a function through an expression only if that expression involves exclusively functions of fewer variables, {*less_vars*}
- allowing substitutions only that do not lead to a case distinction coefficient $= 0$ or not,
- specifying whether extra effort should be spent to identify the substitution with the lowest bound on growth of the full system. {*min_growth*}

Substitution types are represented by different numbers depending on the subset of the above filters to be used. If a substitution type is to be done automatically then from all possible substitutions passing all filters of this type that substitution is selected that leads to no sub-cases (if available) and that uses the shortest equation.

Factorization: It is very common that big algebraic systems contain equations that can be factorized. Factorizing an equation and setting the factors individually to zero simplifies the whole task because factors are simpler expressions than the whole equation and set to zero they may lead to substitutions and thereby further simplifications. The downside is that if problems with, say 100 unknowns, need 40 case-distinctions in order to be able to solve automatically for the remaining 60 unknowns then this would require 2^{40} cases to be investigated which is impractical. The problem is to find the right balance, between delaying case-distinctions in order not to generate too many cases and on the other hand introducing case distinctions as early as necessary in order to simplify the system. This simplification may be necessary to solve the system but in any case it will speed up its solution (although at the price of having to solve a simplified system at least twice, depending on the number of factors).

For large systems with many factorizable equations the careful selection of the next equation to be factorized is important to gain the most from each factorization and to succeed with as few as possible factorizations. Criteria which give factors and therefore equations a higher priority are

- the number of equations in which this factor occurs,
- if the factor is a single unknown function or constant, then the number of times this unknown turns up in the whole system,
- the total degree of the factor,
- the number of factors of an equation,
- and others.

It also matters in which order the factors are set to zero. For example, the equation $0 = ab$ can be used to split into the 2 cases: 1. $a = 0$, 2. $a \neq 0, b = 0$ or to split into the 2 cases 1. $b = 0$, 2. $b \neq 0, a = 0$. If one of the 2 factors, say b, involves functions which occur only linearly then this property is to be preserved and these functions should be substituted as such substitutions preserve their linearity. But to have many such substitutions available, it is useful to know of many non-linearly occuring functions to be non-zero as they occur as coefficients of the linearly occuring functions. In the above situation it is therefore better to do the first splitting 1. $a = 0$, 2. $a \neq 0, b = 0$ because $a \neq 0$ will be more useful for substitutions of linear functions than $b \neq 0$ would be.

An exception of this plausible rule occurs towards the end of all the substitutions of all the linearly occuring b_i when some b_i are an overall factor to many equations. If one would then set, say, $b_{22} = 0$ as the second case in a factorization, the first case would generate as subcases factorizations of other equations where $b_{22} = 0$ would be the second case again and so on. To avoid this one should investigate $b_{22} = 0$ as the first case in the first factorization.

The only purpose of that little thought experiment was to show that simple questions, like which factored equation should be used first for case-distinctions and in which order to set factors to zero can already be difficult to answer in general.

Elimination (Gröbner Basis) Steps: To increase safety and avoid excessive expression swell one can apart from the normal call {*30*} request to do Gröbner basis computation steps only if they are simplification steps replacing an equation by a shorter equation. {*27*}

In a different version only steps are performed in which equations are included which do not contain more than 3 unknowns. This helps to focus on steps which are more likely to solve small sub-systems with readily available simple results. {*57*}

Often the computationally cheapest way to obtain a consistent (involutive) system of equations implies to change the ordering during the computation. This is the case when substitutions of functions are performed which are not ranked highest in a lexicographical ordering of functions. But CRACK also offers an interactive way to

- change the lexicographical ordering of variables, {*ov*}
- change the lexicographical ordering of functions, {*of*}
- give the differential order of derivatives a higher or lower priority in the total ordering than the lexicographical ordering of functions, {*og*}
- give either the total differential order of a derivative of a function a higher priority than the lexicographical ordering of the derivative of that function or to take the lexicographical ordering of derivatives as the only criterium. {*of*}

Solution of an under-determined differential equation: When solving an over-determined system of linear differential equations where the general solution involves free functions, then in the last computational step often a single equation for more than one function remains to be solved. Examples are the computation of symmetries and conservation laws of non-linear differential equations which are linearizable. In CRACK two procedures are available, one for under-determined linear ODEs {*22*} and one for linear PDEs, {*23*} both with non-constant coefficients.

Indirect Separation: Due to integrations new functions of fewer variables are introduced. Substituting functions may lead to equations where no function depends on all variables but all variables appear as variables to unknown functions, e.g. $0 = f(x)+g(y)$ although usually much more complicated with 10 or 20 independent variables and many functions depending on different combinations of these variables. Because no variable occurs only explicitly, direct separations mentioned above are not possible. Two different algorithms, one for linear indirectly separable equations {*10, 26*} and one for non-linear directly separable equations {*48*} provide systematic ways of dealing with such equations.

Indirectly separable equations always result when an equation is integrated with respect to different variables, like $0 = f_{xy}$ to $f = g(x) + h(y)$ and a function, here $f(x, y)$, is substituted.

Function and variable transformations: In the interactive mode one can specify a transformation of the whole problem in which old functions and variables are expressed in a mix of new functions and variables.

We conclude the listing of modules and continue with other aspects of CRACK..

Exploiting Bi-linearity: In bi-linear algebraic problems we have 2 sets of variables $a_1, .., a_m$ and $b_1, .., b_n$ such that all equations have the form $0 = \sum_{k=1}^{l} \gamma_k a_{i_k} b_{j_k}$, $\gamma_k \in G$. Although the problem is linear in the a_i and linear in the b_j it still is a non-linear problem. A guideline which helps keeping the structure of the system during computation relatively simple is to preserve the linearity of either the a_i or the b_j as long as possible. In classification problems of integrable systems the ansatz for the symmetry/first integral involves usually more terms and therefore more constants (called b_j in applications of CRACK) than the ansatz for the integrable system (with constants a_i). A good strategy therefore is to keep the system linear in the b_j during the computation, i.e. to

- substitute only a b_j in terms of a_i, b_k, or an a_i in terms of an a_k but not an a_i in terms of any b_k,
- do elimination steps for any b_j or for an a_i if the involved equations do not contain any b_k,

The proposed measures are effective not only for algebraic problems but for ODEs/PDEs too (i.e. to preserve linearity of a sub-set of functions as long as possible). {*flin_*}

Flexible Process Control: Different types of over-determined systems are more or less suited for an automatic solution. With the currrent version (2002) it is relatively save to try solving large bi-linear algebraic problems automatically. Another well suited area concerns over-determined systems of linear PDEs. In contrast, non-linear systems of PDEs most likely require a more tight interactive control. Different modes of operation are possible. One can

- perform one {*a*} or more computational steps {*g*} automatically, each step trying modules in the order defined by the current priority list {*p1*} until one module succeeds in its purpose,
- perform one module a specific number of times or as long as it is successful, {*l*}
- set a time limit until which the program should run automatically, {*time_limit, limit_time*}
- interrupt an on-going automatic computation and continue the computation interactively, {*_stop_*}
- arrange that the priority list of modules changes at a certain point in the computation when the system of equations has changed its character,
- induce a case distinction whether a user-given expression is zero or not. {*44*}

Apart from flexible control over what kind of steps to be done, the steps themselves can be controlled more or less too, e.g. whether equations are selected by the module or the user.

Highest priority in the priority list have so-called *to-do* steps. The list of to-do steps is usually empty but can be filled by any successful step if it requires another specific step to follow instantly. For example, if a very simple equation $0 = f_x$ is integrated then the substitution of f should follow straight away, even if substitutions would have a low priority according to the current priority list.

Total Data Control: To make wise decisions of how to continue the computation in an interactive session one needs tools to inspect large systems of equations. Helpful commands in CRACK print

- equations, inequalities, functions and variables, {*e, pi, f*}
- the occurence of all derivatives of selected functions in any equation, {*v*}
- a statistics summary of the equations of the system, {*s*}
- a matrix display of occurences of unknowns in all equations, {*pd*}
- the value of any LISP variable, {*pv*}

- the value of algebraic expressions that can be specified using equation names (e.g. `coeffn(e_5,df(f,x,y),2)`), {*pe*}
- not under-determined subsystems. {*ss*}

Safety: When working on large problems it may come to a stage where computational steps are necessary, like substitution, which are risky in the sense that they may simplify the problem but also complicate it by increasing its size. To avoid this risk a few safety features have been implemented.

- At any time during the computation one can save a backup of the complete current situation in a file and also load a backup. {*sb, rb*}
- All key strokes are automatically recorded in a list and are available after each interactive step, or when the computation has finished. This list can be fed into CRACK at the beginning of a new computation so that the same operations are performed automatically that were performed interactively before. The purpose is to be able to do an interactive exploration first and to repeat it afterwards automatically without having to note with pen or pencil all steps that had been done. {*history_, old_history*}
- During an automatic computation the program might start a computational step which turns out to take far too long. It would be better to stop this computation and try something else instead. But in computer algebra with lots of global variables involved it is not straight forward to stop a computation in the middle of it. If one would use time as a criterion then it could happen that time is up during a garbage collection which to stop would be deadly for the session. CRACK allows to set a limit of garbage collections for any one of those computations that have the potential to last forever, like algebraic factorizations of large expressions. With such an arrangement an automatic computation can not get stuck anymore due to lengthy factorizations, searches for length reductions or elimination steps. {*max_gc_elimin, max_gc_fac, max_gc_red_len, max_gc_short, max_gc_ss*}
- Due to a recent (April 2002) initiative of Winfried Neun the parallel version of the computer algebra system REDUCE has been re-activated and is running on the Beowulf cluster at Brock University [23]. This allows conveniently (with a 2-letter command) to duplicate the current status of a CRACK computation to another computer, to try out there different operations (e.g. risky ones) until a viable way to continue the computation is found without endangering the original session. {*pp*}

Managing Solutions: Non-linear problems can have many solutions. The number of solutions found by CRACK can even be higher because to make progress CRACK may have factorized an equation and considered the two cases $a = 0$ and $a \neq 0$ whereas solutions in both cases could be merged to only one solution without any restriction for a. This merging of solutions can be accomplished with a separate program `merge_sol()` after the computation.

Another form of post-processing is the production of a web page for each solution, like `http://lie.math.brocku.ca/twolf/bl/v1l05o35-s1.html` .

If in the solution of over-determined differential equations the program performs integrations of equations before the differential Gröbner basis was computed then in the final solution there may be redundant constants or functions of integration. Redundant constants or functions in a solution are not an error but they makes solutions appear unnecessarily complicated. In a postprocessing step these functions and constants can be eliminated. {*adjust_fnc, drop_const(), dropredundant()*}

Parallelization: The availability of a parallel version of CRACK was mentioned above allowing to try out different ways to continue an ongoing computation. A different possibility to make use of a cluster of computers with CRACK is, to export automatically the investigation of sub-cases and sub-sub-cases to different computers to be solved in parallel.

It was explained above how factorizations may be necessary to make any progress but also their potential of exploding the time requirements. By running the computation on a cluster and being able to solve many more cases one can give factorizations a higher priority and capitalize on the benefit of factorizations, i.e. the simplification of the problem.

Relationship to Gröbner Basis Algorithms: For systems of equations in which the unknown constants or functions turn up only polynomially a well known method is able to check the consistency of the system. For algebraic systems this is the Gröbner Basis Method and for systems of differential equations this is the differential Gröbner Basis method. To guarantee the method to terminate a total ordering of unknowns and their derivatives has to be introduced. This ordering determines which highest powers of unknowns are to be eliminated next or which highest order derivatives have to be eliminated next using integrability conditions. Often such eliminations lead to exponential growth of the generated equations. In the package CRACK such computations are executed with only a low priority. A higher priority have operations which reduce the length of equations, irrespective of any orderings. Violating any ordering a finite number of times still guarantees a finite algorithm. The potential gain is large as described next.

Length Reduction of Equations: An algorithm designed originally to length-reduce differential equations proved to be essential in length reducing systems of bi-linear algebraic equations or homogeneous equations which resulted from bi-linear equations during the solution process.

The aim of the method is to find out whether one equation $0 = E_1$ can length reduce another one $0 = E_2$ by replacing E_2 through an appropriate linear combination $\alpha E_1 - \beta E_2, \quad \beta \neq 0$. To find α, β one can divide each term of E_2 through each term of E_1 and count how often each quotient occurs. If a quotient α/β occurs m times then $\alpha E_1 - \beta E_2$ will have $\leq n_1 + n_2 - 2m$ terms because $2m$ terms will cancel each other. A length reduction is found if $n_1 + n_2 - 2m \leq \max(n_1, n_2)$. The method becomes efficient after a few algorithmic refinements discussed in [24]. Length reduced equations

- are more likely to length reduce other equations,
- are much more likely to be factorizable,
- are more suited for substitutions as the substitution induces less growth of the whole systems and introduces fewer new occurences of functions in equations,
- are more likely to be integrable by being exact or being an ODE if the system consists of differential equations,
- involve on average fewer unknowns and make the whole system more sparse. This sparseness can be used to plan better a sequence of eliminations.

Customization: The addition of new modules to perform new specialized computations is easy. The new module only has to accept as input the system of equations, list of inequalities, list of unknowns to be computed and provide output in a similar form. The module name has to be added to a list of all modules and a one line description has to be added to a list of descriptions. This makes it easy for users to add special techniques for the solution of systems with extra structure. A dummy template module {*58*} is already added and has only to be filled with content.

Debugging: A feature useful mainly for debugging is that in the middle of an ongoing interactive computation the program can be changed by loading a different version of CRACK procedures. Thus one could advance quickly close to the point in the execution where an error occurs, load a version of the faulty procedure that gives extensive output and watch how the fault happens before fixing it.

The possibility to interrupt REDUCE itself temporarily and to inspect the underlying LISP environment $\{br\}$ or to execute LISP commands and to continue with the CRACK session afterwards $\{pc\}$ led to a few improvements and fixes in REDUCE itself.

Availability: The package CRACK including manual can be downloaded for free from `http://lie.math.brocku.ca/twolf/crack/`. It is requested to cite this paper in a publication if CRACK has been used for any computations that contributed to that publication.

References

[1] Sokolov V.V. and Shabat A.B., Classification of integrable evolution equations Soviet Sci. Reviews, Section C, **4** 221-280 (1984).

[2] Mikhailov A.V., Shabat A.B., and Yamilov R.I., The symmetry approach to the classification of nonlinear equations. Complete lists of integrable systems, *Russian Math. Surveys*, **42**(4), 1–63, 1987.

[3] Fokas A.S., Symmetries and integrability, *Stud. Appl. Math.*, **77**, 253–299, 1987.

[4] Mikhailov A.V., Shabat A.B., Sokolov V.V., Symmetry Approach to Classification of Integrable Equations, in the book "What is integrability?", Springer-Verlag (Springer Series in Nonlinear Dynamics), 115-184 (1991).

[5] Sokolov, V.V. and Wolf, T., Classification of integrable polynomial vector evolution equations, J. Phys. A: Math. Gen. **34**, 11139-11148, (2001).

[6] Calogero, F., The evolution partial differential equation $u_t = u_{xxx} + 3(u_{xx}u^2 + 3u_x^2u) + 3u_xu^4$, J. Math. Phys., **28(3)** 538–555 (1987).

[7] Ibragimov, N.Kh. and Shabat, A.B., Evolutionary equations with nontrivial Lie-Bäcklund algebra. Func. Anal. Appl., **14(1)** 19-28 (1980).

[8] Fokas A.S., A symmetry approach to exactly solvable evolution equations, J. Math. Phys., **21(6)** 1318-1325 (1980).

[9] Sanders J.A., Wang J.P., On the integrability of homogeneous scalar evolution equations. J. Diff. Eq., **147** 410–434 (1998).

[10] Sokolov V.V., Wolf T., A symmetry test for quasilinear coupled systems, Inverse Problems, **15** L5-L11 (1999).

[11] Manakov S.V., On theory of two-dimensional self-focussing of electromagnetic waves, ZhETF, **65** 505-516 (1973).

[12] Kulish P.P. and Sklyanin E.K., O(N)-invariant nonlinear Schrodinger equation. - A new completely integrable system. Phys. Lett. A, 84(7), 349-352, 1981

[13] Svinolupov S.I., Generalized Schrödinger equations and Jordan pairs, Comm. in Math. Physics, **143** 559-575 (1992).

[14] Sokolov V. V. and Tsiganov A.V. On Lax pairs for the generalized Kowalewski and Goryachev-Chaplygin tops, Theoret. and Math. Phys., 2002, **131**, no.**1**, 543-549. Preprint nlin.SI/0111035, 2001.

[15] A.V. Mikhailov and R.I. Yamilov, Towards Classification of 2+1 dimensional Integrable Equations. Integrability conditions I, Journal of Physics A, v. 31, pp. 6707-6715, 1998.

[16] Foursov M V 2000 Classification of certain integrable coupled potential KdV and modified KdV-type equations *J. Math. Phys.* **41** 6173–6185

[17] Foursov M V and Olver P J 2000 On the classification of symmetrically-coupled integrable evolution equations *Symmetries and Differential equations* edited by Andreev V K and Shanko Yu V (Institute of Computational Modelling, Krasnoyarsk, Russia, 2000) 244–248

[18] Hirota R and Satsuma J 1981 Soliton solutions of a coupled Korteweg–de Vries equation *Phys. Lett.* A **85** 407–408

[19] Wilson G 1982 The affine Lie algebra $C_2^{(1)}$ and an equation of Hirota and Satsuma *Phys. Lett.* A **89** 332–334

[20] Drinfel'd V G and Sokolov V V 1985 Lie algebras and equations of Korteweg–de Vries type *J. Sov. Math.* **30** 1975–2036

[21] Wu Y T, Geng X G, Hu X B and Zhu S M 1999 A generalized Hirota–Satsuma coupled Korteweg–de Vries equation and Miura transformations *Phys. Lett.* A **255** 259–264

[22] Tsuchida T., Wadati M., New integrable systems of derivative nonlinear Schrödinger equations with multiple components, Phys. Lett. A, **257** 53–64 (1999).

[23] Melenk, H. and Neun, W., RR: Parallel Symbolic Algorithm Support for PSL Based REDUCE, ZIP preprint (2002), `www.zib.de/Optimization/Software/Reduce/moredocs/parallel_reduce.ps`.

[24] Wolf, T., Size reduction and partial decoupling of systems of equations, J. Symb. Comp. 33, no 3 (2002) 367-383.

[25] Wolf, T., The Symbolic Integration of Exact PDEs, J. Symb. Comp. **30** (No. 5) (2000), 619–629.

[26] Wolf, T., The integration of systems of linear PDEs using conservation laws of syzygies, preprint, accepted for publication in a special issue of J. of Symb. Comp., 25 pages (2002).