Applications of the package CRACK to simplify large systems

Thomas Wolf

Department of Mathematics,
Brock University
St.Catharines, Ontario, Canada

June 25, 2009

Outline

Poisson Structures (with A. Odesskii)

To define a Poisson structure one needs to specify:

Poisson Structures (with A. Odesskii)

To define a Poisson structure one needs to specify:

- a set of variables, say,
  $x_1, \ldots, x_n$

Poisson Structures (with A. Odesskii)

To define a Poisson structure one needs to specify:

- a set of variables, say,
  $x_1, ..., x_n$
- so-called Poisson brackets for each pair of variables:

$$\{x_i, x_j\} = \omega_{ij}(x_1, ..., x_n).$$

Here ω_{ij} are functions in $x_1, ..., x_n$.

Poisson Structures (with A. Odesskii)

To define a Poisson structure one needs to specify:

- a set of variables, say,
  $x_1, ..., x_n$
- so-called Poisson brackets for each pair of variables:

$$\{x_i, x_j\} = \omega_{ij}(x_1, ..., x_n).$$

  Here ω_{ij} are functions in $x_1, ..., x_n$.

Example:
Let the set of variables be $p,\ q$ and Poisson brackets be

$$\{p, p\} = \{q, q\} = 0, \quad \{p, q\} = 1, \quad \{q, p\} = -1.$$

Poisson structures

Given a Poisson structure we define Poisson brackets for *any* functions f, g in $x_1, ..., x_n$ by

$$\{f, g\} = \sum_{i,j=1}^{n} \frac{\partial f}{\partial x_i} \frac{\partial g}{\partial x_j} \{x_i, x_j\}$$

Poisson structures

Given a Poisson structure we define Poisson brackets for *any* functions f, g in $x_1, ..., x_n$ by

$$\begin{aligned} \{f, g\} &= \sum_{i,j=1}^{n} \frac{\partial f}{\partial x_i} \frac{\partial g}{\partial x_j} \{x_i, x_j\} \\ &= \sum_{i,j=1}^{n} \frac{\partial f}{\partial x_i} \frac{\partial g}{\partial x_j} \omega_{ij} \end{aligned}$$

Poisson structures

Given a Poisson structure we define Poisson brackets for *any* functions f, g in $x_1, ..., x_n$ by

$$\begin{aligned} \{f, g\} &= \sum_{i,j=1}^{n} \frac{\partial f}{\partial x_i} \frac{\partial g}{\partial x_j} \{x_i, x_j\} \\ &= \sum_{i,j=1}^{n} \frac{\partial f}{\partial x_i} \frac{\partial g}{\partial x_j} \omega_{ij} \end{aligned}$$

Functions ω_{ij} should satisfy the following constrains:

- Anti-symmetry: $\{f, g\} = -\{g, f\}$ for all f, g. This means that $\omega_{ji} = -\omega_{ij}$.

Poisson structures

Given a Poisson structure we define Poisson brackets for *any* functions f, g in $x_1, ..., x_n$ by

$$\begin{aligned} \{f, g\} &= \sum_{i,j=1}^{n} \frac{\partial f}{\partial x_i} \frac{\partial g}{\partial x_j} \{x_i, x_j\} \\ &= \sum_{i,j=1}^{n} \frac{\partial f}{\partial x_i} \frac{\partial g}{\partial x_j} \omega_{ij} \end{aligned}$$

Functions ω_{ij} should satisfy the following constrains:

- Anti-symmetry: $\{f, g\} = -\{g, f\}$ for all f, g. This means that $\omega_{ji} = -\omega_{ij}$.
- Jacobi identity: $\{f, \{g, h\}\} + \{g, \{h, f\}\} + \{h, \{f, g\}\} = 0$ for all f, g, h.

Poisson structures

Given a Poisson structure we define Poisson brackets for *any* functions f, g in $x_1, ..., x_n$ by

$$\begin{aligned} \{f, g\} &= \sum_{i,j=1}^{n} \frac{\partial f}{\partial x_i} \frac{\partial g}{\partial x_j} \{x_i, x_j\} \\ &= \sum_{i,j=1}^{n} \frac{\partial f}{\partial x_i} \frac{\partial g}{\partial x_j} \omega_{ij} \end{aligned}$$

Functions ω_{ij} should satisfy the following constrains:

- Anti-symmetry: $\{f, g\} = -\{g, f\}$ for all f, g. This means that $\omega_{ji} = -\omega_{ij}$.
- Jacobi identity: $\{f, \{g, h\}\} + \{g, \{h, f\}\} + \{h, \{f, g\}\} = 0$ for all f, g, h.

Formulating these conditions for $f, g, h = x_1, .., x_n$ gives a complete set of conditions for the ω_{ij}: an overdetermined non-linear system of PDEs.

Applications of Poisson structures I

Hamiltonian systems: To define a Hamiltonian system one needs to specify a Poisson structure and an additional function $H(x_1, ..., x_n)$ called *Hamiltonian*.

Applications of Poisson structures I

Hamiltonian systems: To define a Hamiltonian system one needs to specify a Poisson structure and an additional function $H(x_1, ..., x_n)$ called *Hamiltonian*.
The dynamics is given by:

$$\frac{dx_i}{dt} = \{H, x_i\}$$

In this case for any function $f(x_1, ..., x_n)$ we also have

$$\frac{df}{dt} = \{H, f\}.$$

Applications of Poisson structures I

Hamiltonian systems: To define a Hamiltonian system one needs to specify a Poisson structure and an additional function $H(x_1, ..., x_n)$ called *Hamiltonian*.
The dynamics is given by:

$$\frac{dx_i}{dt} = \{H, x_i\}$$

In this case for any function $f(x_1, ..., x_n)$ we also have

$$\frac{df}{dt} = \{H, f\}.$$

A system is called integrable if there exist sufficiently many functions $H_0(= H), H_1, ...$ such that $\{H_i, H_j\} = 0$. In particular we have

$$\frac{dH_i}{dt} = \{H, H_i\} = 0$$

so $H_i = const$.

Applications of Poisson structures II

In pure mathematics many important objects carry a natural Poisson structure. It is also interesting to study the geometry of so-called symplectic leaves connected with each Poisson structure.

Examples of Poisson structures

- Let $\{x_i, x_j\} = c_{ij}$. This is a Poisson structure if $c_{ji} = -c_{ij}$. The so-called *canonical* Poisson structure with variables $p_1, ..., p_n, q_1, ..., q_n$ and brackets

$$\{p_i, p_j\} = \{q_i, q_j\} = 0, \{p_i, q_j\} = \delta_{ij}$$

  belongs to this type.

Examples of Poisson structures

- Let $\{x_i, x_j\} = c_{ij}$. This is a Poisson structure if $c_{ji} = -c_{ij}$. The so-called *canonical* Poisson structure with variables $p_1, ..., p_n, q_1, ..., q_n$ and brackets

$$\{p_i, p_j\} = \{q_i, q_j\} = 0, \{p_i, q_j\} = \delta_{ij}$$

  belongs to this type.
- Linear Poisson brackets:

$$\{x_i, x_j\} = \sum_{k=1}^{n} c_{ij}^k x_k .$$

  The Jacobi identity gives an overdetermined system of quadratic algebraic equations for c_{ij}^k. These equations mean that c_{ij}^k are structure constants of a Lie algebra.

Examples of Poisson structures

- Let $\{x_i, x_j\} = c_{ij}$. This is a Poisson structure if $c_{ji} = -c_{ij}$. The so-called *canonical* Poisson structure with variables $p_1, ..., p_n, q_1, ..., q_n$ and brackets

$$\{p_i, p_j\} = \{q_i, q_j\} = 0, \{p_i, q_j\} = \delta_{ij}$$

  belongs to this type.
- Linear Poisson brackets:

$$\{x_i, x_j\} = \sum_{k=1}^{n} c_{ij}^k x_k.$$

  The Jacobi identity gives an overdetermined system of quadratic algebraic equations for c_{ij}^k. These equations mean that c_{ij}^k are structure constants of a Lie algebra.
- Quadratic Poisson brackets:

$$\{x_i, x_j\} = \sum_{k,l=1}^{n} c_{ij}^{kl} x_k x_l.$$

  There are two type of such structures: rational and elliptic.

Elliptic Poisson structures

Elliptic Poisson structures have the following applications:

- The theory of quantum groups.

Elliptic Poisson structures

Elliptic Poisson structures have the following applications:

- The theory of quantum groups.
- Non-commutative geometry. Elliptic algebras and their quotients are examples of non-commutative algebraic manifolds.

Elliptic Poisson structures

Elliptic Poisson structures have the following applications:

- The theory of quantum groups.
- Non-commutative geometry. Elliptic algebras and their quotients are examples of non-commutative algebraic manifolds.
- Cohomologies of associative algebras. Elliptic algebras are examples of Kozul algebras.

Elliptic Poisson structures

Elliptic Poisson structures have the following applications:

- The theory of quantum groups.
- Non-commutative geometry. Elliptic algebras and their quotients are examples of non-commutative algebraic manifolds.
- Cohomologies of associative algebras. Elliptic algebras are examples of Kozul algebras.
- Integrable systems. Elliptic algebras are a good tool in studying integrable systems connected to solutions of the Yang-Baxter equation.

Elliptic Poisson structures

Elliptic Poisson structures have the following applications:

- The theory of quantum groups.
- Non-commutative geometry. Elliptic algebras and their quotients are examples of non-commutative algebraic manifolds.
- Cohomologies of associative algebras. Elliptic algebras are examples of Kozul algebras.
- Integrable systems. Elliptic algebras are a good tool in studying integrable systems connected to solutions of the Yang-Baxter equation.

The most interesting known examples of these structures satisfy the following property: there exists a group of automorphysms of the form:

$$x_i \to \epsilon^i x_i, \quad x_i \to x_{i+1}$$

where ϵ is an n^{th} root of unity and indices are given modulo n.

Quadratic $(\mathbb{Z}/3\mathbb{Z})^2$ invariant Poisson structures

The Poisson structures to be investigated have 9 dynamic variables x_{ij}, $i,j = 0,1,2$ and Poisson brackets

$$\{x_{ij}, x_{kl}\} = \sum_{p,q,r,s} a_{ijklpqrs} x_{pq} x_{rs}. \tag{1}$$

We assume the following symmetries:

$$x_{ij} \to x_{(i+1)j}, \quad x_{ij} \to x_{i(j+1)}, \quad x_{ij} \to \epsilon^i x_{ij}, \quad x_{ij} \to \epsilon^j x_{ij}$$

Quadratic $(\mathbb{Z}/3\mathbb{Z})^2$ invariant Poisson structures

The Poisson structures to be investigated have 9 dynamic variables x_{ij}, $i,j = 0,1,2$ and Poisson brackets

$$\{x_{ij}, x_{kl}\} = \sum_{p,q,r,s} a_{ijklpqrs} x_{pq} x_{rs}. \tag{1}$$

We assume the following symmetries:

$$x_{ij} \to x_{(i+1)j}, \quad x_{ij} \to x_{i(j+1)}, \quad x_{ij} \to \epsilon^i x_{ij}, \quad x_{ij} \to \epsilon^j x_{ij}$$

This means that apart from the usual symmetries

$$a_{ijklpqrs} = a_{ijklrspq} = -a_{klijpqrs}$$

the coefficients $a_{ijklpqrs}$ also obey the restriction that $a_{ijklpqrs} = 0$ if $i + k \neq p + r \ mod\ 3$ or $j + l \neq q + s \ mod\ 3$.

Computational Problem

The Jacobi identities

$$\{x_{ij}, \{x_{kl}, x_{mn}\}\} + \{x_{kl}, \{x_{mn}, x_{ij}\}\} + \{x_{mn}, \{x_{ij}, x_{kl}\}\} = 0$$

are formulated using the Poisson structures

$$\{x_{ij}, x_{kl}\} = \sum_{p,q,r,s} a_{ijklpqrs} x_{pq} x_{rs}.$$

Computational Problem

The Jacobi identities

$$\{x_{ij}, \{x_{kl}, x_{mn}\}\} + \{x_{kl}, \{x_{mn}, x_{ij}\}\} + \{x_{mn}, \{x_{ij}, x_{kl}\}\} = 0$$

are formulated using the Poisson structures

$$\{x_{ij}, x_{kl}\} = \sum_{p,q,r,s} a_{ijklpqrs} x_{pq} x_{rs}.$$

These identities have to be satisfied identically in the dynamical variables x_{ij}. Therefore all coefficients of different powers of different x_{ij} are set to zero resulting in 2754 quadratic equations most of which are redundant leaving 62 equations with 354 terms for 20 unknowns $a_{ijklpqrs}$.

Computational Problem

The Jacobi identities

$$\{x_{ij}, \{x_{kl}, x_{mn}\}\} + \{x_{kl}, \{x_{mn}, x_{ij}\}\} + \{x_{mn}, \{x_{ij}, x_{kl}\}\} = 0$$

are formulated using the Poisson structures

$$\{x_{ij}, x_{kl}\} = \sum_{p,q,r,s} a_{ijklpqrs} x_{pq} x_{rs}.$$

These identities have to be satisfied identically in the dynamical variables x_{ij}. Therefore all coefficients of different powers of different x_{ij} are set to zero resulting in 2754 quadratic equations most of which are redundant leaving 62 equations with 354 terms for 20 unknowns $a_{ijklpqrs}$.

We demonstrate the subcase that two of the a's are zero:

$$a_{00100112} = a_{00011021} = 0$$

which is the simplest of 5 sub-cases that have to be considered.

Outline

About the Problem I (work done with A. Kiselev)

Starting with a program SSTOOLS for doing supersymmetric algebra and calculus an online database of supersymmetric evolutionary PDE of higher order symmetries was created. When analysing some classes of the 10,000 web page database Kiselev and TW found a N=2 supersymmetric generalization of the Burger's equation.

About the Problem I (work done with A. Kiselev)

Starting with a program SSTOOLS for doing supersymmetric algebra and calculus an online database of supersymmetric evolutionary PDE of higher order symmetries was created. When analysing some classes of the 10,000 web page database Kiselev and TW found a N=2 supersymmetric generalization of the Burger's equation.

The computation to be discussed deals with the geometric problem of constructing Gardner's deformation for the Bosonic Limit of this equation. Invented in late 60's, Gardner's integrable deformations of PDE consist of parametric extensions, subject to further restrictions, for the equations under study, combined with differential substitutions back to the original system.

About the Problem II

For the Bosonic Limit at hand, we impose the assumption that both the extension and the substitution be polynomial and weight-homogeneous. This yields the algebraic system that is linear in the undetermined coefficients responsible for the extension and that is quadratic in the constants contained in the backward substitution.

About the Problem II

For the Bosonic Limit at hand, we impose the assumption that both the extension and the substitution be polynomial and weight-homogeneous. This yields the algebraic system that is linear in the undetermined coefficients responsible for the extension and that is quadratic in the constants contained in the backward substitution.

Six solutions, grouped in two classes, are obtained. The corresponding Gardner deformations are geometrically nontrivial. However, Kiselev proved that none of the six solutions can be lifted from the Bosonic Limit to the full N=2 supersymmetric system.

About the Computation I

- A Groebner basis computation is too large.

About the Computation I

- A Groebner basis computation is too large.
- Substitutions? Yes as problem is linear in some variables.

About the Computation I

- A Groebner basis computation is too large.
- Substitutions? Yes as problem is linear in some variables.
- But substitution lead to blow-up of the system in size and thus in time too.

About the Computation I

- A Groebner basis computation is too large.
- Substitutions? Yes as problem is linear in some variables.
- But substitution lead to blow-up of the system in size and thus in time too.
- A slower growth but still blow-up happens if case-generating substitutions are allowed (each case distinction may double the computation time of the remaining computatin of that case)

About the Computation I

- A Groebner basis computation is too large.
- Substitutions? Yes as problem is linear in some variables.
- But substitution lead to blow-up of the system in size and thus in time too.
- A slower growth but still blow-up happens if case-generating substitutions are allowed (each case distinction may double the computation time of the remaining computatin of that case)
- Shortenings are essential as they provide short equations in which fewer substitutions are done and which subsequently grow slower, but more importantly shortening produces shorter substitution equations

About the Computation II

- Experience: After each substitution many shortenings become possible.

About the Computation II

- Experience: After each substitution many shortenings become possible.
- Giving shortenings a strictly higher priority than long substitutions is a very safe but very slow solving strategy.

About the Computation II

- Experience: After each substitution many shortenings become possible.
- Giving shortenings a strictly higher priority than long substitutions is a very safe but very slow solving strategy.
- Controlled speedup: Partially interactively do a few substitutions at the time and then longer periods of shortenings.

About the Computation II

- Experience: After each substitution many shortenings become possible.
- Giving shortenings a strictly higher priority than long substitutions is a very safe but very slow solving strategy.
- Controlled speedup: Partially interactively do a few substitutions at the time and then longer periods of shortenings.
- How can such an interactive computation be remembered? $\longrightarrow$ by a backup of all interactive commands done automatically.

About the Computation II

- Experience: After each substitution many shortenings become possible.
- Giving shortenings a strictly higher priority than long substitutions is a very safe but very slow solving strategy.
- Controlled speedup: Partially interactively do a few substitutions at the time and then longer periods of shortenings.
- How can such an interactive computation be remembered? $\longrightarrow$ by a backup of all interactive commands done automatically.
- In this comutation soon many equations factorize and only case-generating substitutions become possible.

About the Computation III

- Although CRACK has elaborate heuristics about which case distinctions to do first, CRACK does of course not know about the intrinsic (geometric) relevance of all possible different case distinctions one could try.

About the Computation III

- Although CRACK has elaborate heuristics about which case distinctions to do first, CRACK does of course not know about the intrinsic (geometric) relevance of all possible different case distinctions one could try.
- How to find out which cases are cricial and which not?

About the Computation III

- Although CRACK has elaborate heuristics about which case distinctions to do first, CRACK does of course not know about the intrinsic (geometric) relevance of all possible different case distinctions one could try.
- How to find out which cases are cricial and which not?
- By running a computation where case distinctions have a high priority, and by inspecting case distinctions where at least one case was solved quickly.

About the Computation III

- Although CRACK has elaborate heuristics about which case distinctions to do first, CRACK does of course not know about the intrinsic (geometric) relevance of all possible different case distinctions one could try.
- How to find out which cases are cricial and which not?
- By running a computation where case distinctions have a high priority, and by inspecting case distinctions where at least one case was solved quickly.
- Both cases $.. = 0$ and $.. \neq 0$ are both very useful (if the lhs is simple and occurs frequently as factor or as coefficient). Thus, if one of both cases is a strong restriction which leads to contradictions that no solution exists in that case (typically performed quickly) then one got a good simplification of the other (general) case for a relatively small price.

About the Computation IV

- Example: This is a list of case distinctions produced after a CRACK run but one can print a list of cases also during a CRACK run: (see emacs file).

About the Computation IV

- Example: This is a list of case distinctions produced after a CRACK run but one can print a list of cases also during a CRACK run: (see emacs file).
- In this (labour intensive) way the following (probably optimal) sequence of case distinctions lead to the full solution:

About the Computation IV

- Example: This is a list of case distinctions produced after a CRACK run but one can print a list of cases also during a CRACK run: (see emacs file).
- In this (labour intensive) way the following (probably optimal) sequence of case distinctions lead to the full solution:

case	**extra assumption**	**# of solutions**
1.	$p1\_3 = 0$	0
2.	$p1\_3 \neq 3$	
2.1.	$p1\_8 = 0$	
2.1.1.	$p2\_30^2 - 1 = 0$	0
2.1.2.	$p2\_30^2 - 1 \neq 0$	
2.1.2.1.	$p1\_6 = 0$	4
2.1.2.2.	$p1\_6 \neq 0$	
2.1.2.2.1.	$p1\_6 + 4 = 0$	0
2.1.2.2.2.	$p1\_6 + 4 \neq 0$	0
2.2.	$p1\_8 \neq 0$	2

Outline

Purpose of CRACK

- a computer algebra package written in REDUCE for the solution of large over-determined systems of algebraic, ordinary or partial differential equations

Purpose of CRACK

- a computer algebra package written in REDUCE for the solution of large over-determined systems of algebraic, ordinary or partial differential equations
- especially suited for large over-determined systems with polynomial non-linearity

History

- initially designed for the automatic integration of over-determined linear PDE systems (resulting in Lie-symmetry and conservation law computations of differential equations)

History

- initially designed for the automatic integration of over-determined linear PDE systems (resulting in Lie-symmetry and conservation law computations of differential equations)
- later emphasis on interactive access for solving PDEs (resulting in same type of investigations but higher order PDEs)

History

- initially designed for the automatic integration of over-determined linear PDE systems (resulting in Lie-symmetry and conservation law computations of differential equations)
- later emphasis on interactive access for solving PDEs (resulting in same type of investigations but higher order PDEs)
- which together with a better handling of non-linearity enabled to treat large *algebraic* systems (resulting in the classification of integrable Hamiltonians),

History

- initially designed for the automatic integration of over-determined linear PDE systems (resulting in Lie-symmetry and conservation law computations of differential equations)
- later emphasis on interactive access for solving PDEs (resulting in same type of investigations but higher order PDEs)
- which together with a better handling of non-linearity enabled to treat large *algebraic* systems (resulting in the classification of integrable Hamiltonians),
- various extensions in last years.

Handling large Systems

- a module to reduce the length of equations and by that make the system more sparse,

Handling large Systems

- a module to reduce the length of equations and by that make the system more sparse,
- the ability to compute a bound on the size of the system after doing a specific substitution without doing the substitution,

Handling large Systems

- a module to reduce the length of equations and by that make the system more sparse,
- the ability to compute a bound on the size of the system after doing a specific substitution without doing the substitution,
- the option to do substitutions “bottom up”, i.e. only one substitution in the shortest possible equation using only a shorter equation,

Handling large Systems

- a module to reduce the length of equations and by that make the system more sparse,
- the ability to compute a bound on the size of the system after doing a specific substitution without doing the substitution,
- the option to do substitutions “bottom up”, i.e. only one substitution in the shortest possible equation using only a shorter equation,
- the possibility to tackle systems that are too large even to be formulated, by having a buffer of equations which is dynamically emptied (equations are read from it into the computation) and filled (new equations are generated as solution progresses).

Example for Shortening

$$0 = x + \quad + A(z, ..) \tag{2}$$
$$0 = \quad y + A(z, ..) \tag{3}$$

total ordering: $x > y > z > ..$
A is a large expression.
GB: -

Example for Shortening

$$0 = x + \quad + A(z, ..) \tag{2}$$
$$0 = \quad y + A(z, ..) \tag{3}$$

total ordering: $x > y > z > ..$
A is a large expression.
GB: -
Shortening:

$$0 = x - y \tag{4}$$
$$0 = y + A \tag{5}$$

Example for Shortening

$$0 = x + \quad + A(z, ..) \tag{2}$$
$$0 = \quad y + A(z, ..) \tag{3}$$

total ordering: $x > y > z > ..$
A is a large expression.
GB: -
Shortening:

$$0 = x - y \tag{4}$$
$$0 = y + A \tag{5}$$

Obvious benefit:

- In further steps growth comes only from *one* A in (5) and not from *two* A in (2) and (3).

Example for Shortening

Non-obvious benefit:

- reinforcing: a shorter equation is more likely to be useful to shorten others,

Example for Shortening

Non-obvious benefit:

- reinforcing: a shorter equation is more likely to be useful to shorten others,
- large systems: # pairings of n equations grows as $O(n^2)$

Example for Shortening

Non-obvious benefit:

- reinforcing: a shorter equation is more likely to be useful to shorten others,
- large systems: # pairings of n equations grows as $O(n^2)$
- sparse systems will reveal a better elimination strategy,

Example for Shortening

Non-obvious benefit:

- reinforcing: a shorter equation is more likely to be useful to shorten others,
- large systems: # pairings of n equations grows as $O(n^2)$
- sparse systems will reveal a better elimination strategy,
- PDEs more likely to become total derivatives or ODEs,

Example for Shortening

Non-obvious benefit:

- reinforcing: a shorter equation is more likely to be useful to shorten others,
- large systems: # pairings of n equations grows as $O(n^2)$
- sparse systems will reveal a better elimination strategy,
- PDEs more likely to become total derivatives or ODEs,
- random polynomials with fewer terms factorize much more likely than large polynomials.

Example for Shortening

Non-obvious benefit:

- reinforcing: a shorter equation is more likely to be useful to shorten others,
- large systems: # pairings of n equations grows as $O(n^2)$
- sparse systems will reveal a better elimination strategy,
- PDEs more likely to become total derivatives or ODEs,
- random polynomials with fewer terms factorize much more likely than large polynomials.

An algorithm and implementation are described in J. Symb. Comp. 33, no 3 (2002).

Handling Nonlinearity

- different heuristics for splitting a computation into sub-cases by:
  - taking the most frequent factor to be $= 0$ or $\neq 0$,
  - taking the most frequent unknown to be $= 0$ or $\neq 0$,
  - taking expressions to be $= 0$ or $\neq 0$ which are either specified interactively, or given in a specific order at the start of the computation,
  - a heuristic that picks one of the factorizable equations and an order of their factors to be set to zero,
  - initiating case splittings when the system becomes critically large

Inequations

- the treatment of inequations: their usage, active collection and derivation, and their constant update in an ongoing reduction process based on newly derived equations

Inequations

- the treatment of inequations: their usage, active collection and derivation, and their constant update in an ongoing reduction process based on newly derived equations

  For example:

$$ab + cd = 0, \quad a \neq 0, b \neq 0 \rightarrow c \neq 0, d \neq 0$$

Inequations

- the treatment of inequations: their usage, active collection and derivation, and their constant update in an ongoing reduction process based on newly derived equations

  For example:

$$ab + cd = 0, \quad a \neq 0, b \neq 0 \rightarrow c \neq 0, d \neq 0$$

$$x^2y + xy^2 + x + 3 = 0 \quad \rightarrow \quad x \neq 0$$

Inequations

- the treatment of inequations: their usage, active collection and derivation, and their constant update in an ongoing reduction process based on newly derived equations

  For example:

$$ab + cd = 0, \quad a \neq 0, b \neq 0 \rightarrow c \neq 0, d \neq 0$$

$$x^2y + xy^2 + x + 3 = 0 \quad \rightarrow \quad x \neq 0$$

$$AB = 0 \quad \rightarrow \quad 1)\ A = 0 \ \text{ or } \ 2)\ B = 0,\ A \neq 0$$

Inequations

- the treatment of inequations: their usage, active collection and derivation, and their constant update in an ongoing reduction process based on newly derived equations

  For example:

$$ab + cd = 0, \quad a \neq 0, b \neq 0 \rightarrow c \neq 0, d \neq 0$$

$$x^2y + xy^2 + x + 3 = 0 \quad \rightarrow \quad x \neq 0$$

$$AB = 0 \quad \rightarrow \quad 1)\ A = 0 \ \text{ or } \ 2)\ B = 0,\ A \neq 0$$

- or-inequations: maintenance and automatic reduction of lists of expressions of which at least one expression is required not to vanish

Flexibility

- the availability of a variety of schemes to vary the solution strategy depending on the situation, i.e. to balance shortenings, decouplings, substitutions, splitting into sub-cases,...

Flexibility

- the availability of a variety of schemes to vary the solution strategy depending on the situation, i.e. to balance shortenings, decouplings, substitutions, splitting into sub-cases,...
- the possibility to trade interactively or automatically the speed of the solution process versus avoidance of expression swell,

Flexibility

- the availability of a variety of schemes to vary the solution strategy depending on the situation, i.e. to balance shortenings, decouplings, substitutions, splitting into sub-cases,...
- the possibility to trade interactively or automatically the speed of the solution process versus avoidance of expression swell,
- the ability to call the packages GB (J.C. Faugère) and Singular (H. Schönemann et.al.) as subroutines,

Flexibility

- the availability of a variety of schemes to vary the solution strategy depending on the situation, i.e. to balance shortenings, decouplings, substitutions, splitting into sub-cases,...
- the possibility to trade interactively or automatically the speed of the solution process versus avoidance of expression swell,
- the ability to call the packages GB (J.C. Faugère) and Singular (H. Schönemann et.al.) as subroutines,
- the possibility to perform single Gröbner steps through the system FORM.

Interactivity, Safety

- different levels of interactivity from fully automatic to half interactive to very low level interactive access, changeable during the computation, e.g. to change total ordering in Gröbner basis computation,

Interactivity, Safety

- different levels of interactivity from fully automatic to half interactive to very low level interactive access, changeable during the computation, e.g. to change total ordering in Gröbner basis computation,
- various visualization aids for inspecting large systems or single equations,

Interactivity, Safety

- different levels of interactivity from fully automatic to half interactive to very low level interactive access, changeable during the computation, e.g. to change total ordering in Gröbner basis computation,
- various visualization aids for inspecting large systems or single equations,
- ability to automatically stop sub-steps that take too long, like factorizations, Gröbner steps, substitutions,..

Interactivity, Safety

- different levels of interactivity from fully automatic to half interactive to very low level interactive access, changeable during the computation, e.g. to change total ordering in Gröbner basis computation,
- various visualization aids for inspecting large systems or single equations,
- ability to automatically stop sub-steps that take too long, like factorizations, Gröbner steps, substitutions,..
- possibility to backup and re-load the current state of a session,

Interactivity, Safety

- different levels of interactivity from fully automatic to half interactive to very low level interactive access, changeable during the computation, e.g. to change total ordering in Gröbner basis computation,
- various visualization aids for inspecting large systems or single equations,
- ability to automatically stop sub-steps that take too long, like factorizations, Gröbner steps, substitutions,..
- possibility to backup and re-load the current state of a session,
- possibility for a soft interrupt for an ongoing computation,

Interactivity, Safety

- different levels of interactivity from fully automatic to half interactive to very low level interactive access, changeable during the computation, e.g. to change total ordering in Gröbner basis computation,
- various visualization aids for inspecting large systems or single equations,
- ability to automatically stop sub-steps that take too long, like factorizations, Gröbner steps, substitutions,..
- possibility to backup and re-load the current state of a session,
- possibility for a soft interrupt for an ongoing computation,
- automatic backup of interactive input to have the possibility to re-produce an interactive computation,

Interactivity, Safety

- different levels of interactivity from fully automatic to half interactive to very low level interactive access, changeable during the computation, e.g. to change total ordering in Gröbner basis computation,
- various visualization aids for inspecting large systems or single equations,
- ability to automatically stop sub-steps that take too long, like factorizations, Gröbner steps, substitutions,..
- possibility to backup and re-load the current state of a session,
- possibility for a soft interrupt for an ongoing computation,
- automatic backup of interactive input to have the possibility to re-produce an interactive computation,
- possibility to specify later interactive input at the command line as a parameter list.

Parallelism

- CRACK can take advantage of parallel hardware by computing in parallel the different cases resulting in non-linear problems.

Parallelism

- CRACK can take advantage of parallel hardware by computing in parallel the different cases resulting in non-linear problems.
- Another use is to duplicate interactive sessions to other nodes to experiment with different methods in order to find a way to make progress in a difficult computation.

Extra Functionality for ODEs/PDEs

- exploitation of integrability conditions

Extra Functionality for ODEs/PDEs

- exploitation of integrability conditions
- integration of different classes of PDEs (total derivatives (TD), TD + inhomogeneity, sum of 2 TDs),

Extra Functionality for ODEs/PDEs

- exploitation of integrability conditions
- integration of different classes of PDEs (total derivatives (TD), TD + inhomogeneity, sum of 2 TDs),
- the ability to integrate linear PDEs using syzygies which result as a by-product in the process of computing a differential Gröbner basis,

Extra Functionality for ODEs/PDEs

- exploitation of integrability conditions
- integration of different classes of PDEs (total derivatives (TD), TD + inhomogeneity, sum of 2 TDs),
- the ability to integrate linear PDEs using syzygies which result as a by-product in the process of computing a differential Gröbner basis,
- a module to compute parametric solutions of linear under-determined ODE systems with non-constant coefficients,

Extra Functionality for ODEs/PDEs

- exploitation of integrability conditions
- integration of different classes of PDEs (total derivatives (TD), TD + inhomogeneity, sum of 2 TDs),
- the ability to integrate linear PDEs using syzygies which result as a by-product in the process of computing a differential Gröbner basis,
- a module to compute parametric solutions of linear under-determined ODE systems with non-constant coefficients,
- the ability to separate (i.e. split) expressions wrt. independent variables occurring with variable exponents,

Extra Functionality for ODEs/PDEs

- the solution of *quasilinear first order PDEs* by
  - formulating and solving characteristic ODE systems,
  - solving them and
  - performing point transformations not only of the PDE but the whole system of equations.

Extra Functionality for ODEs/PDEs

- the solution of *quasilinear first order PDEs* by
  - formulating and solving characteristic ODE systems,
  - solving them and
  - performing point transformations not only of the PDE but the whole system of equations.

  This ability is essential for handling systems of equations that have solutions involving not only constants but free functions, for example when computing conservation laws or infinitesimal symmetries of linearizable or linear PDEs/ODEs.

Simple Example of non-polynomial Equations

independent variables:	x, y
unknowns:	$f(x), n$
equation:	$0 = y^n f_x + y f^2$
result are 2 cases:	$n = 1: \quad 0 = f_x + f^2$
	$n \neq 1: \quad 0 = f_x, \quad 0 = f^2$

Simple Example of non-polynomial Equations

independent variables: x, y
unknowns: $f(x), n$
equation: $0 = y^n f_x + y f^2$
result are 2 cases: $n = 1: \quad 0 = f_x + f^2$
$n \neq 1: \quad 0 = f_x, \quad 0 = f^2$

More generally:
- exponents can be any polynomial algebraic or differential expressions,
- arbitrary number of terms and different exponents.

Simple Example of non-polynomial Equations

independent variables: x, y
unknowns: $f(x), n$
equation: $0 = y^n f_x + y f^2$
result are 2 cases: $n = 1: \quad 0 = f_x + f^2$
$n \neq 1: \quad 0 = f_x, \quad 0 = f^2$

More generally:
- exponents can be any polynomial algebraic or differential expressions,
- arbitrary number of terms and different exponents.

Condition:
- basis must be an independent variable

Simple Example of non-polynomial Equations

independent variables: x, y
unknowns: $f(x), n$
equation: $0 = y^n f_x + y f^2$
result are 2 cases: $n = 1: \quad 0 = f_x + f^2$
$n \neq 1: \quad 0 = f_x, \quad 0 = f^2$

More generally:
- exponents can be any polynomial algebraic or differential expressions,
- arbitrary number of terms and different exponents.

Condition:
- basis must be an independent variable

Alternative:
- replace y^n by new function $h(y)$ and add condition $0 = h_y - nh$.
- disadvantage: less obvious how to split $0 = hf_x + yf^2$

Post Processing

- the possibility to merge solutions of parametric linear algebraic systems,

Post Processing

- the possibility to merge solutions of parametric linear algebraic systems,
- the automatic generation of web-pages for larger sets of solutions,

Post Processing

- the possibility to merge solutions of parametric linear algebraic systems,
- the automatic generation of web-pages for larger sets of solutions,
- the determination and dropping of redundancy in parametric constants and functions in solutions

Applications

- requiring the solution of large bi-linear algebraic systems and an automatic merging of obtained solutions:
  - WT, Efimovskaya, O. V.: *Classification of integrable quadratic Hamiltonians on e(3)*, Regular and Chaotic Dynamics, vol 8, no 2 (2003), p 155-162.
  - Tsuchida, T. and WT: *Classification of polynomial integrable systems of mixed scalar and vector evolution equations. I*, J. Phys. A: Math. Gen. 38 (2005) 7691-7733 (and on arxiv as nlin.SI/0412003).
  - Sokolov, V.V. and WT: *Integrable quadratic Hamiltonians on* $so(4)$ *and* $so(3,1)$, J Phys A: Math Gen 39 (2006) 1915-1926 and arXiv nlin.SI/0405066.
  - Kiselev, A. and WT: *Supersymmetric Representations and Integrable Super-Extensions of the Burgers and Bussinesq Equations*, SIGMA, vol 2 (2006), paper 030 and arXiv math-ph/0511071.

Applications

- needing the solution of an extremely large algebraic system of billions of equations with on average millions of terms, far too large even to be formulated upfront:
  - Tsarev, S.P. and WT.: *Classification of 3-dimensional integrable scalar discrete equations*, arXiv:0706.2464, Lett. in Math. Phys. (available at www.springerlink.com as DOI:10.1007/s11005-008-0230-2).
  - WT.: *On Solving Large Systems of Polynomial Equations Appearing in Discrete Differential Geometry*, Programming and Computer Software, 34 (2008), no 2, p. 75-83.
- requiring the solution of extensive over-determined ODE/PDE-systems:
  - Anco, S. and WT: *Some symmetry classifications of hyperbolic vector evolution equations*, JNMP, Volume 12, Supplement 1 (2005), p 13-31 (and on arxiv as nlin.SI/0412015).

Systems, Hardware

- REDUCE runs on a wide range of systems and hardware.
  CSL REDUCE runs on all machines that have C.
  PSL REDUCE has been ported to 64bit AMD processors and recently also to OS X on Intel Mac.

Systems, Hardware

- REDUCE runs on a wide range of systems and hardware.
  CSL REDUCE runs on all machines that have C.
  PSL REDUCE has been ported to 64bit AMD processors and recently also to OS X on Intel Mac.
- PARALLEL REDUCE runs in a truly parallel mode on beowulf clusters, all porting done by Winfried Neun, ZIB, Berlin.

Availability

- The computer algebra system REDUCE is Open Source and freely available since end of 2008: `http://reduce-algebra.sourceforge.net/`

Availability

- The computer algebra system REDUCE is Open Source and freely available since end of 2008: `http://reduce-algebra.sourceforge.net/`
- CRACK comes with REDUCE but a newer version can be downloaded from `http://lie.math.brocku.ca/crack/src`

Availability

- The computer algebra system REDUCE is Open Source and freely available since end of 2008: `http://reduce-algebra.sourceforge.net/`
- CRACK comes with REDUCE but a newer version can be downloaded from `http://lie.math.brocku.ca/crack/src`
- online interactive access and tutorial for CRACK: `http://lie.math.brocku.ca/crack/demo`