On the Verification of Polynomial System Solvers

Changbo Chen, Marc Moreno Maza, Wei Pan and Yuzhen Xie
University of Western Ontario, London, Ontario, Canada

Solving systems of non-linear, algebraic or differential equations, is a fundamental problem in mathematical science. It has been studied for centuries and has stimulated many research developments. Algorithmic solutions can be classified into three categories: numeric, symbolic and hybrid numeric-symbolic. The choice for one of them depends on the characteristics of the system of equations to solve. For instance, it depends on whether the coefficients are known exactly or are approximations obtained from experimental measurements.This choice depends also on the expected answers, which could be a complete description of all the solutions, or only the real solutions, or just one sample solution among all of them.

Symbolic solvers are powerful tools in scientific computing: they are well suited for problems where the desired output must be exact and they have been applied successfully in areas like digital signal processing, robotics, theoretical physics, cryptology, dynamical systems, with many important outcomes. See [7] for an overview of these applications.

Symbolic solvers are also highly complex software. First, they implement sophisticated algorithms, which are generally at the level of on-going research. Moreover, in most computer algebra systems, the `solve` command involves nearly the entire set of libraries in the system, challenging the most advanced operations on matrices, polynomials, algebraic and modular numbers, polynomial ideals, etc.

Secondly, algorithms for solving systems of polynomial equations are by nature of exponential-space complexity. Consequently, symbolic solvers are extremely time-consuming when applied to large examples. Even worse, intermediate expressions can grow to enormous size and may halt the computations, even if the result is of moderate size. The implementation of symbolic solvers, then, requires techniques that go far beyond the manipulation of algebraic or differential equations, such as efficient memory management, data compression, parallel and distributed computing, etc.

Last, but not least, the precise output specifications of a symbolic solver can be quite involved. Indeed, given an input polynomial system F, defining what a symbolic solver should return implies describing what the geometry of the solution set $\mathbf{V}(F)$ of F can be. For an arbitrary F, the set $\mathbf{V}(F)$ may consist of components of different natures and sizes: points, lines, curves, surfaces. This leads to the following difficult challenge.

Given a polynomial system F and a set of components $C_1, \ldots, C_e$, it is hard, in general, to tell whether the union of $C_1, \ldots, C_e$ corresponds exactly to the solution set $\mathbf{V}(F)$ or not. Actually, solving this verification problem is generally (at least) as hard as solving the system F itself.

Because of the high complexity of symbolic solvers, developing verification algorithms and reliable verification software tools is a clear need. However, this verification problem has received little attention in the literature. In this paper, we present new techniques for verifying a large class of symbolic solvers. We also report on intensive experimentation illustrating the high efficiency of our approach w.r.t. known techniques.

We assume that each component of the solution set $\mathbf{V}(F)$ is given by a so-called *regular system*. This is a natural assumption in symbolic computations, well-developed in the literature under different terminologies, see [1, 22] and the references therein. In broad words, a regular system consists of several polynomial equations with a triangular shape

$$p_1(x_1) = p_2(x_1, x_2) = \cdots = p_i(x_1, x_2, \ldots, x_n) = 0$$

and a polynomial inequality

$$h(x_1, \ldots, x_n) \neq 0$$

such that there exists (at least) one point $(a_1, \ldots, a_n)$ satisfying the above equations and inequality. Note that these polynomials may contain parameters.

Let us consider the following well-known system F taken from [5].

$$\begin{cases} x^{31} - x^6 - x - y &= 0 \\ x^8 - z &= 0 \\ x^{10} - t &= 0 \end{cases}$$

We aim at solving this system for $x > y > z > t$, that is, expressing x as a function of y, z, t, then y as a function of z, t and z as a function of t. One possible decomposition is given by the three regular systems below:

$$\left\{\begin{array}{rcl} \left(t^4-t\right)x - ty - z^2 & = & 0 \\ t^3y^2 + 2t^2z^2y + \left(-t^6+2t^3+t-1\right)z^4 & = & 0 \\ z^5 - t^4 & = & 0 \\ t^4 - t & \neq & 0 \end{array}\right. , \quad \left\{\begin{array}{rcl} x^2 - z^4 & = & 0 \\ y + t^2z^2 & = & 0 \\ z^5 - t & = & 0 \\ t^3 - 1 & = & 0 \end{array}\right. , \quad \left\{\begin{array}{rcl} x & = & 0 \\ y & = & 0 \\ z & = & 0 \\ t & = & 0 \end{array}\right.$$

Another decomposition is given by these other three regular systems:

$$\left\{\begin{array}{rcl} \left(t^4-t\right)x - ty - z^2 & = & 0 \\ tzy^2 + 2z^3y - t^8 + 2t^5 + t^3 - t^2 & = & 0 \\ z^5 - t^4 & = & 0 \\ z\left(t^4 - t\right) & \neq & 0 \end{array}\right. , \quad \left\{\begin{array}{rcl} zx^2 - t & = & 0 \\ ty + z^2 & = & 0 \\ z^5 - t & = & 0 \\ t^3 - 1 & = & 0 \\ tz & \neq & 0 \end{array}\right. , \quad \left\{\begin{array}{rcl} x & = & 0 \\ y & = & 0 \\ z & = & 0 \\ t & = & 0 \end{array}\right.$$

These two decompositions look slightly different (in particular, the second components) and one could think that, if each of them was produced by a different solver, then at least one of these solvers has a bug. In fact, both decompositions are valid, but proving respectively that they encode the solution set $\mathbf{V}(F)$ is not feasible without computer assistance. However, proving that they define the same set of points can be achieved by an expert hand without computer assistance. This is an important observation that we will guide us in this paper.

Let us consider now an arbitrary input system F and a set of components $C_1, \ldots, C_e$ encoded by regular systems $S_1, \ldots, S_e$ respectively. The usual approach for verifying that $C_1, \ldots, C_e$ correspond exactly to the solution set $\mathbf{V}(F)$ is as follows.

(1) First, one checks that each candidate component C_i is actually contained in $\mathbf{V}(F)$. This essentially reduces to substitute the coordinates of the points given by C_i into the polynomials of F: if all these polynomials vanish at these points, then C_i is a component of $\mathbf{V}(F)$, otherwise, (and up to technical details that we will skip in this introduction) C_i is not a component of $\mathbf{V}(F)$.

(2) Secondly, one checks that $\mathbf{V}(F)$ is contained in the union of the candidate components $C_1, \ldots, C_e$ by:

  (2.1) computing a polynomial system G such that $\mathbf{V}(G)$ corresponds exactly to $C_1, \ldots, C_e$, and

  (2.2) checking that every solution of $\mathbf{V}(F)$ cancels the polynomials of G.

Steps (2.1) and (2.2) can be performed using standard techniques based on computations of Gröbner bases. These calculations are very expensive, as shown by our experimentation.

In this poster, we propose a different approach. The main idea is as follows. Instead of comparing a candidate set of components $C_1, \ldots, C_e$ against the input system F, we compare it against the output $D_1, \ldots, D_f$ produced by another solver. Both this solver and the comparison process are assumed to be validated. Hence, the candidate set of components $C_1, \ldots, C_e$ corresponds exactly to the solution set $\mathbf{V}(F)$ if and only if the comparison process shows that $D_1, \ldots, D_f$ and $C_1, \ldots, C_e$ define the same solution set.

We summarize the different verification procedures that are available for triangular decompositions, including our new approach. We report on experimentation with these verification procedures. Our data illustrate the high efficiency of our new approach.

References

[1] P. Aubry, D. Lazard, and M. Moreno Maza. On the theories of triangular sets. *J. Symb. Comp.*, 28(1-2):105–124, 1999.

[2] P. Aubry and M. Moreno Maza. Triangular sets for solving polynomial systems: A comparative implementation of four methods. *J. Symb. Comp.*, 28(1-2):125–154, 1999.

[3] J. Backelin and R. Fröberg. How we proved that there are exactly 924 cyclic 7-roots. In S. M. Watt, editor, *Proc. ISSAC'91*, pages 103–111. ACM, 1991.

[4] F. Boulier, F. Lemaire, and M. Moreno Maza. Well known theorems on triangular systems and the D5 principle. In *Proc. of* Transgressive Computing 2006, Granada, Spain, 2006.

[5] L. Donati and C. Traverso. Experimenting the Gröbner basis algorithm with the ALPI system. In *Proc. ISSAC'89*, pages 192–198. ACN Press, 1989.

[6] D. Eisenbud. *Commutative Algebra with a View Toward Algebraic Geometry*. Springer-Verlag, New York-Berlin-Heidelberg, 1995.

[7] J. Grabmeier, E. Kaltofen, and V. Weispfenning, editors. *Computer Algebra Handbook*. Springer, 2003.

[8] The Computational Mathematics Group. The basicmath library. NAG Ltd, Oxford, UK, 1998. http://www.nag.co.uk/projects/FRISCO.html.

[9] R. Hartshorne. *Algebraic Geometry*. Springer-Verlag, 1997.

[10] M. Kalkbrener. A generalized euclidean algorithm for computing triangular representations of algebraic varieties. *J. Symb. Comp.*, 15:143–167, 1993.

[11] F. Lemaire, M. Moreno Maza, and Y. Xie. The `RegularChains` library. In Ilias S. Kotsireas, editor, Maple Conference 2005, pages 355–368, 2005.

[12] Montserrat Manubens and Antonio Montes. Improving dispgb algorithm using the discriminant ideal, 2006.

[13] M. Moreno Maza. On triangular decompositions of algebraic varieties. Technical Report TR 4/99, NAG Ltd, Oxford, UK, 1999. http://www.csd.uwo.ca/∼moreno.

[14] J. O'Halloran and M. Schilmoeller. Gröbner bases for constructible sets. *Journal of Communications in Algebra*, 30(11), 2002.

[15] P. Samuel and O. Zariski. *Commutative algebra*. D. Van Nostrand Company, INC., 1967.

[16] W. Sit. Computations on quasi-algebraic sets. In R. Liska, editor, *Electronic Proceedings of IMACS ACA'98*, 1998.

[17] *The SymbolicData Project*. http://www.SymbolicData.org, 2000–2006.

[18] D. Wang. Computing triangular systems and regular systems. *Journal of Symbolic Computation*, 30(2):221–236, 2000.

[19] D. M. Wang. *Epsilon 0.618*. http://www-calfor.lip6.fr/∼wang/epsilon.

[20] D. M. Wang. Decomposing polynomial systems into simple systems. *J. Symb. Comp.*, 25(3):295–314, 1998.

[21] D. M. Wang. *Elimination Methods*. Springer, Wein, New York, 2000.

[22] D.M. Wang. Computing triangular systems and regular systems. *J. Symb. Comp.*, 30(2):221–236, 2000.