THE UNIVERSITY of EDINBURGH

Edinburgh Research Explorer

Solving Symbolic Equations with PRESS

Citation for published version:
Sterling, L, Bundy, A, Byrd, L, O'Keefe, R & Silver, B 1982, Solving Symbolic Equations with PRESS. in Computer Algebra - Lecture Notes in Computer Science. vol. 7. DOI: 10.1007/3-540-11607-9_13

Digital Object Identifier (DOI):
10.1007/3-540-11607-9_13

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
Computer Algebra - Lecture Notes in Computer Science

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 20. Jul. 2018

SOLVING SYMBOLIC EQUATIONS WITH PRESS

by
Leon Sterling, Alan Bundy, Lawrence Byrd,
Richard O'Keefe, and Bernard Silver
Department of Artificial Intelligence
University of Edinburgh

Abstract

We outline a program, PRESS (PRolog Equation Solving System) for solving symbolic, transcendental, non-differential equations. The methods used for solving equations are described, together with the service facilities. The principal technique, meta-level inference, appears to have applications in the broader field of symbolic and algebraic manipulation.

Acknowledgements

This work was supported by SERC grants GR/B/29252 and GR/B/73989 and various studentships.

Keywords

equation solving, rewrite rules, meta-level inference, logic programming

1. Introduction

The PRESS program was originally developed with two aims in mind. The first aim was to use the program as a vehicle to explore some ideas about controlling search in mathematical reasoning using meta-level descriptions and strategies. The other aim was to serve as the equation solving module for the MECHO project [Bundy et al 79] designed to solve high-school mechanics problems stated in English. PRESS can solve the fairly straightforward equations, inequalities, and sets of simultaneous equations arising from the example mechanics problems taken from textbooks.

Over the last year or so interest has turned more to developing the equation solving program as a performance program in its own right. The implementation of several new components has led to a marked improvement in performance. The program achievements could, we feel, act as a benchmark in elementary equation solving - something currently lacking in the literature as far as we could determine. The techniques used may have something to offer the field of symbolic and algebraic manipulation.

The equations PRESS has been solving are largely taken from English A-level examination papers. Such examinations are taken by 18 year olds in their final year of high school, and are used to help decide suitability for university entrance. Particular papers used are those issued by the Associated Examining Board (A.E.B.), the University of London, and the University of Oxford. The years range from 1971 to 1979. Currently the program solves 69 out of 83 single equations and 10 out of 14 sets of simultaneous equations. Some typical problems are

$4^{2*x+1} * 5^{x-2} = 6^{1-x}$ (1) (A.E.B. November 1971)

$\cos(x) + \cos(3*x) + \cos(5*x) = 0$ (2) (A.E.B. June 1976)

$3*\tan(3*x) - \tan(x) + 2 = 0$ (3) (Oxford Autumn 1978)

$\log_2 x + 4*\log_x 2 = 5$ (4) (London January 1978)

$3*\mathrm{sech}^2(x) + 4*\tanh(x) + 1 = 0$ (5) (A.E.B. June 1971)

$\log_e(x+1) + \log_e(x-1) = 3$ (6)

$e^{3*x} - 4*e^x + 3*e^{-x} = 0$ (7) (London June 1977)

```
cosh(x) - 3*sinh(y) = 0 &
   2*sinh(x) + 6*cosh(y) = 5                    (A.E.B. June 1973)
```

PRESS can solve all the above equations and some statistics are given in section 5. However versions of MACSYMA [Mathlab 77] and REDUCE that we ran could solve none of the equations. In fact it was hard to guide the REDUCE program to follow our application of rewrite rules.

PRESS is organised as a collection of interacting **methods**. Each method conducts a syntactic analysis of the current equation and, provided various preconditions are met, then manipulates the equation to achieve a specific syntactic effect. For instance, the Collection method analyses the equation to see if it contains more than one occurrence of the unknown. If there are then it tries to reduce the number of occurrences. The methods try, in turn, to manipulate the equation by applying particular rewrite rules. If one succeeds then the process is repeated until the equation is solved or no further methods apply.

PRESS is written in PROLOG, [Clocksin and Mellish 81], a programming language based on the ideas of logic programming. Hence, the PRESS code can be interpreted as axioms of a first order mathematical theory and the running of the program can be interpreted as inference in this theory. The predicates and functions of these axioms express relationships between expressions of algebra, and the axioms assert facts and laws about the representation of algebra. For this reason we say that the PRESS code constitutes the first order, **Meta-Theory of Algebra**. We call algebra an **object-level** theory and the heuristic control information embedded in PRESS a **meta-level** theory. As PRESS runs it conducts a process of **meta-level inference** which guides the search for a solution to the equation. More details of this technique can be found in [Bundy and Welham 81].

In the next section, we will give an overview of the scope of the program. Following that, particular methods will be described. In section 4 the more important of the meta-level concepts used by the program will be discussed. Some indication of performance, including a sample solution, will be given in the final section.

It should be emphasised that our aim was not principally to build a powerful performance program. Nonetheless the program has desirable features. It performs well on a wide range of equations.n It has a modular structure, making it easy to extend the power of the program. It has also been possible for students to do projects in symbolic integration, differential equations and other areas of symbolic and algebraic manipulation, using the basic symbolic manipulation components extant in PRESS.

2. An Overview of the Program

Currently PRESS itself has four different top-level modules: one for solving single equations, one for sets of simultaneous equations, one for inequalities, and one for proving identities. The procedure for solving single equations is the central core of the program. In fact, the other top-level modules are largely interfaces to the relevant parts of the single equation code. We will concentrate in this paper on describing the procedure for solving single equations.

The most recent version of the equation solver has 6 major methods implemented:

- **Isolation**, for solving equations with a single occurrence of the unknown.
- **Polysolve**, for solving polynomial equations.
- **Collection**, for reducing the number of occurrences of the unknown in the equation.
- **Attraction**, for bringing occurrences of the unknown closer together.

- **Homogenization**, a generalized change of unknown method.

- **Function Swapping**, for transforming equations into ones with more amenable function symbols.

These are applied in approximately the order of listing, with each significant transformation of the equation resulting in all the methods being attempted again. Note that particular rewrite rules are only applied in the context of particular methods. This avoids the problem of being bogged down in the exhaustive application of a large rewrite rule set.

PRESS also uses several service modules to aid in algebraic manipulation. There is a pattern matcher, an expression simplifier split up into two components, tidy and eval, a package which reasons about intervals, and a package for manipulating polynomials. These have largely been tailored to the needs of the program and no claims are made to their generality or efficiency.

3. Program Methods

Most algebraic manipulation packages, such as MACSYMA, REDUCE, and MUMATH, have some equation solving ability. Typically there are components to handle polynomials, and equations where there is a single occurrence of the unknown. Thus the first two methods described, Isolation and Polysolve, have little new to offer, but are included here for completeness. The remaining methods exploit more interesting meta-level guidance.

3.1. Isolation

Isolation is a method for solving equations containing only a single occurrence of an unknown. That is, Isolation can solve $\log_e(x^2-1) = 3$, but cannot solve $\log_e(x+1) + \log_e(x-1) = 3$.

The method consists of 'stripping off' the functions surrounding the single occurrence of x by applying the inverse function to both sides of the equation. This process is repeated until x is isolated on one side (the left-hand side) of the equation, e.g.

$$\log_e(x^2-1) = 3 \Rightarrow x^2 - 1 = e^3 \Rightarrow x^2 = e^3 + 1 \Rightarrow x = \pm\sqrt{(e^3 + 1)}.$$

This stripping off is done by applying a system of rewrite rules to the equation, in this case the rules:

$$\log_U V = W \longrightarrow U = V^W, \quad U - V = W \longrightarrow U = V + W \text{ and } U^2 = W \longrightarrow U = \pm\sqrt{W}$$

How to apply the rewrite rules is determined with the aid of position information.

3.2. Polysolve

The left-hand side of the equation minus the right-hand side of the equation is parsed to determine whether it is a polynomial. The definition of a polynomial is slightly enlarged to include terms of the form x^N for negative integers N. If the relevant expression is a polynomial, control of the solve procedure is passed to the polynomial solver.

The algorithm handling polynomials recognises linear and quadratic polynomial equations, which are easily solved by simple formulae. Also zero roots are recognised and the appropriate power of the unknown is factored out. Simple integer roots are tested for, and the appropriate linear term factored out. Disguised linear and quadratic equations are also solved, for example $x^4 - 4*x^2 + 3 = 0$ is a quadratic in x^2 with solutions $x^2 = 3$ or 1.

Various solutions depending on the polynomial being symmetric or anti-symmetric have also been implemented.

3.3. Collection

This is a method to reduce the number of occurrences of the unknown in an equation by applying a suitable rewrite rule. For example, consider the expression $\log_e((x+1)*(x-1))$ which has two occurrences of x. The rewrite rule $(U+V)*(U-V) \rightarrow U^2-V^2$ can be applied to the expression to produce $\log_e(x^2-1)$. Note x occurs only once in this new expression.

A reduction in the number of occurrences of the unknown is often a key step when solving equations, usually because it enables Isolation. The Collection method tries to match subterms of the equation with suitable rewrite rules which will reduce the number of occurrences of the unknown. Most commonly, Collection is a preparatory step before performing Isolation.

A more detailed description of Collection can be found in [Bundy and Welham 81], along with fuller descriptions of Attraction and Isolation. Heuristics are given there to locate the subterm to be rewritten.

3.4. Attraction

This is a method designed to bring occurrences of the unknown closer together. This might be a useful preparatory step before performing a Collection step. Closeness of two occurrences of the unknown is determined by considering the number of arcs lying between them in the expression tree of the equation.

Again rewrite rules encoding an algebraic manipulation step are matched against a particular expression. The closeness of the occurrences is calculated before and after the potential manipulation step. If the distance decreases, the rewrite rule is applied.

Consider the expression $\log_e(x+1) + \log_e(x-1)$. The occurrences of x are separated by six arcs on the expression tree. The rewrite rule $\log_U V + \log_U W \rightarrow \log_U(V*W)$ can be used to transform the expression to $\log_e((x+1)*(x-1))$. The occurrences of x are separated by four arcs in the new expression. Hence the application of the above rewrite rule is a valid Attraction step.

3.5. Homogenization

This is a very powerful recent addition to the program, which has been described in [Bundy and Silver 81] and [Silver 81]. Given an equation $(e^x)^3 - 4*e^x + 3/e^x = 0$, it is standard to introduce a new unknown for e^x, y say. This substitution transforms this equation into a polynomial equation in y which can be solved by the polynomial solver.

However if the initial equation appears in the examination papers as $e^{3*x} - 4*e^x + 3*e^{-x} = 0$, it is not at all obvious that the same substitution enables the equation to be solved. Homogenization, in this case, determines that each of the exponential terms can be expressed in terms of e^x.

More generally, Homogenization parses the equation recording the terms which contain the unknown. Such terms are classified into four types: logarithmic, exponential, trigonometric and hyperbolic. If all the terms in the equation are of the same type, the equation is labelled of the particular type. Otherwise the method fails. For each equation type it is determined whether each term in the equation can be rewritten in terms of some reduced term. This rewriting is done and an explicit change of unknown substitution performed for the reduced term.

After Homogenization, equations are routinely solved by the polynomial solver with occasionally some prior manipulation by the Function Swapping code.

3.6. Function Swapping

When solving equations, often one function symbol is preferable to another. This may be for several reasons. If the right hand side of the equation equals zero, multiplication is preferable to addition on the left hand side of the equation, since this allows us to split the initial equation into two separate equations. Squaring an expression is preferable to taking the square root, since there are simple rewrite rules relating to squared expressions. Function Swapping is a collection of methods which transform an equation according to function symbol preference.

Certain functions are labelled **nasty**. These are functions which are less familiar than their inverse, such as the inverse trigonometric functions, e.g. $\sin^{-1}(x)$. Logarithms and square roots are other examples. Modifications of Isolation, Collection and Attraction are used by Function Swapping to remove nasty functions. For example, consider the solution of $\sqrt{5*x-25} - \sqrt{x-1} = 2$. The first step in the solution is to isolate one of the square roots to obtain $5*x-25 = (2+\sqrt{x-1})^2$. Note that Isolation has occurred even though there are occurrences of the unknown occurring on the right-hand side of the equation. The remaining square root can not be isolated without reversing the previous step but the right-hand side of the equation can be expanded, to give $5*x-25 = 4 + 2*\sqrt{x-1} + (\sqrt{x-1})^2$. The square root term has been brought closer to its inverse operation, squaring. For this reason we call this step **nasty-function attraction**. Of course, another square root has been generated, but this root is isolatable, producing a 'nicer' equation. After cancelling the square and square root, the isolation of the remaining radical proceeds. This results in a quartic polynomial, which is really a disguised quadratic polynomial. This polynomial is easily solved, giving the answer to the problem, and a spurious root.

Another example of Function Swapping is the following. Certain problems, containing terms of the form $a^{f(x)}$ where a is a constant, are simplified by taking logs to an appropriate base. This is a case where logarithms are less nasty than exponentiation, as this type of exponentiation is not the familiar one.

The context often determines which functions are preferred. Consider $\cos(x) + \cos(3*x) + \cos(5*x) = 0$. The right hand side of the equation is 0, so factorization is a possibility. In this situation addition is less useful than multiplication. PRESS solves this problem by adding cos(x) and cos(5*x). This replaces one of the additions with a multiplication. The cos(3*x) can then be factored out, and the other factor produces the other root after a simple application of Isolation.

3.7. Service Facilities - Simplifier, Evaluator, Matcher and Interval Package

Currently, PRESS does not make extensive use of strong normal form mechanisms, with two exceptions. A polynomial normal form is used within the polynomial solver. Collection and Attraction assume the equation is in a weak normal form with terms containing the unknown on the left-hand side and unknown-free terms on the right-hand side. However for the most part equations are maintained as "tidied" expressions, which are not canonical forms.

Tidying is a simplification process which tries to maximise evaluation by bringing together numeric sub-expressions using the associative and commutative properties of certain functions. To assist this process, rewrites are also applied which remove '/' and binary '-' in favour of '*' and '+'; and rules for identity and unit elements of functions are used. As well as performing evaluation where possible, Tidy applies a set of simplification rewrite rules to all levels of the expression. Various methods also make use of intermediate bag representations for associative-commutative functions.

Evaluation is done by an augmented rational arithmetic package written in PROLOG. The functions provided are relationals, +, -, *, /, div, mod, ^ (the exponentiation operator), log, gcd, entier, abs, sign, numer, denom, and a few tabled values for some of the trig functions. Rational powers are handled, as are some rational roots.

The range of functions provided reflects the range of A-level examination papers.

To apply a rewrite rule to an expression, PRESS uses a matcher that knows about the commutativity and associativity of addition and multiplication. For example, the matcher can apply the rule U*W + V*W ⟶ (U+V)*W to the expression x*y + z*(3*x) to give (y+3*z)*x.

One method of solving the equation a*sin(x) + b*cos(x) = c is to apply the rewrite rule sin(U)*cos(V) + cos(U)*sin(V) ⟶ sin(U+V). This involves a more sophisticated pattern matcher. Such a matcher has been implemented and is described in [Borning and Bundy 81], though the simpler matcher is used in most cases.

Perhaps the most exciting use of the more powerful matcher has been to derive the solution of a simplified form of the general cubic equation from first principles.

The interval package was designed to check conditions of rewrite rules. For example the Isolation rule U*V = W ⟶ U = W/V has the condition V ≠ 0. The package uses the monotonicity of functions to determine in what interval function values lie. For example, given the condition x-cos(x) ≠ 0 where x is known to lie in the interval [π/3,π/2), the package determines that the condition holds. The interval package has also been applied to use semantic information about physical properties to reject solutions given by the equation solver. A fuller description can be found in [Bundy 81].

4. Meta-Level Concepts

Proving that a particular value is a solution of an equation can be thought of as proving a theorem in the theory of algebra. Thus a naive initial approach to equation solving might be to give an axiomatization of arithmetic, rules for algebraic manipulation, and ask a theorem prover to prove a particular theorem corresponding to a given equation. Not surprisingly this unguided approach is hopeless in the combinatorially explosive search space of possible algebraic manipulations of an equation.

Our solution is to use meta-level (or syntactic) information about the equation to be solved to find an appropriate equation solving method and, hence, to guide the search for a proof. The rest of this section describes the meta-level concepts we have developed and the way the program uses them.

We distinguished before between information about the expression in general, and facts about properties of the functions occurring in the expression. Useful information about expressions is the number of occurrences of the unknown in it, the argument positions at which they occur, the smallest subterm containing all the unknowns, the distance between occurrences of the unknown (usually measured by numbers of arcs in the expression tree). These are the main meta-level properties used by Isolation, Collection and Attraction.

Explicitly, in the case of Isolation, the method determines the number of occurrences of the unknown, which must be 1. The position of the unknown in the expression tree is exactly specified and used to help find the appropriate rewrite rule.

The function properties used are that we have a polynomial, or another special expression appearing in the equation. Cosine terms in arithmetic progression imply a certain solution method is possible. All the terms being trigonometric imply only a restricted class of rewrite rules will apply. There are several such conditions in the program.

5. Program Algorithm and Performance

While describing the program methods in section 3, we effectively solved two equations. Let us put these solutions together. Firstly, the full behaviour of the solve procedure is given. Consider again equation (7) from the examples in the introduction, repeated here for convenience.

$$e^{3*x} - 4*e^{x} + 3*e^{-x} = 0 \qquad \text{(i)}$$

Let us see how PRESS tackles this equation. After an initial syntactic check and tidy, the solve procedure is invoked.

1. Invoke the isolation procedure to solve the equation if there is only a single occurrence of the unknown. In (i) there are three occurrences of x, the unknown, so this step fails.

2. Call the polynomial solver if the equation is a polynomial equation. The presence of the exponential terms precludes the equation from being a polynomial equation.

3. Check if a simple change of unknown would simplify the equation. In this case there is no simple change.

4. Try collection by selectively applying rewrite rules. For (i) there is no easily applicable rule.

5. Try attraction. Again there is no easily applicable rule.

6. Try to homogenize the equation. This succeeds for (i), x1 being substituted for e^{x}. The new equation is

$$x1^{3} + -4*x1 + 3*x1^{-1} = 0 \ . \qquad \text{(ii)}$$

7. Equation (ii) is now recursively handled by the solve procedure. This time the polynomial solver is invoked.

8. The equation is multiplied through by x1 to give $x1^{4} - 4*x1^{2} + 3 = 0$.

9. This is recognised as essentially a quadratic equation. Another substitution is made with x2 replacing $x1^{2}$.

10. This equation is solved, with two solutions x2 = 1 and x2 = 3.

11. Now the substitution equation $x1^{2} = x2$ is solved for the two values of x2. Since there is only one occurrence of x1 this is done by the isolation procedure. This gives the solutions x1 = ±1 and $x1 = \pm\sqrt{3}$, and completes the solution invocation begun in step 7.

12. Similarly the equation $e^{x} = x1$ is solved for x by Isolation.

13. The final answers are $x = \log_e\sqrt{3}$ or $x = 0$.

Note that the solutions $x = \log_e -1$ and $x = \log_e -\sqrt{3}$ should be rejected by the solution vetting procedure.

For the second example we just give the sequence of successful methods invoked when solving the equation, (6) of the list given earlier. Consider the sequence of equations below. The method at the end of each line indicates the method used to bring about the equation transformation.

$\log_e(x+1) + \log_e(x-1) = 3$	Attraction
$\log_e((x+1)*(x-1)) = 3$	Collection
$\log_e(x^2-1) = 3$	Isolation

Isolation now solves this equation as in section 3.1.

Thus the solution in both cases is found by cooperation between the methods. The other major method, Function Swapping, is not needed in either example, but is routinely applied after Homogenization.

The PRESS code occupies 72K of core running on a DEC-10. The following table indicates its performance on the example equations given in the introduction.

Equation	Time	Methods Used
(1)	2200	Function Swapping,Polysolve
(2)	1905	Function Swapping,Isolation
(3)	6280	Homogenization,Function Swapping, Polysolve,Isolation
(4)	1010	Homogenization,Polysolve,Isolation
(5)	1350	Homogenization,Polysolve,Isolation
(6)	815	Attraction,Collection,Isolation
(7)	3580	Homogenization,Polysolve,Isolation

The numbered equations refer to those given in the introduction. Times are CPU times given in milliseconds.

REFERENCES

[Borning and Bundy 81]
Borning, A and Bundy, A.
Using matching in algebraic equation solving.
In Schank, R., editor, IJCAI7, pages pp 466-471. International Joint Conference on Artificial Intelligence, 1981.
Also available from Edinburgh as DAI Research Paper No. 158.

[Bundy and Silver 81]
Bundy, A. and Silver, B.
Homogenization: Preparing Equations for Change of Unknown.
In Schank, R., editor, IJCAI7. International Joint Conference on Artificial Intelligence, 1981.
Longer version available from Edinburgh as DAI Research Paper No. 159.

[Bundy and Welham 81]
Bundy, A. and Welham, B.
Using meta-level inference for selective application of multiple rewrite rules in algebraic manipulation.
Artificial Intelligence 16(2), 1981.

[Bundy et al 79]
Bundy, A., Byrd, L., Luger, G., Mellish, C., Milne, R. and Palmer, M.
Solving Mechanics Problems Using Meta-Level Inference.
In Procs of the sixth. IJCAI, Tokyo, 1979.
Also available from Edinburgh as DAI Research Paper No. 112.

[Bundy 81]
Bundy, A.
A Generalized Interval Package and its use for Semantic Checking.
Working Paper 86, Dept. of Artificial Intelligence, Edinburgh, March, 1981.

[Clocksin and Mellish 81]
Clocksin, W.F. and Mellish, C.S.
Programming in Prolog.
Springer Verlag, 1981.

[Mathlab 77]
Mathlab Group.
MACSYMA Reference Manual.
Technical Report, MIT, 1977.

[Silver 81]
Silver, B.
The application of Homogenization to simultaneous equations.
Research Paper 166, Dept. of Artificial Intelligence, Edinburgh, 1981.
To appear in Proceedings of CADE-6, 1982 .