Using Matching in Algebraic Equation Solving

USING MATCHING IN ALGEBRAIC EQUATION SOLVING*

Alan Borning and Alan Bundy

Department of Artificial Intelligence
University of Edinburgh
Hope Park Square, Meadow Lane
Edinburgh EH8 9NW
Scotland

ABSTRACT

This paper describes the use of powerful algebraic matching techniques for applying rewrite rules In equation solving. A matcher Is presented that knows about the commutativlty and associativity of addition and multiplication, will provide defaults for missing summands and factors, and if necessary will solve algebraically for the value of pattern variables.

1. Introduction

This paper describes the use of powerful matching techniques in algebraic equation solving. This work builds on the PRESS algebra system, a computer program for solving equations and Inequalities and for simplifying expressions [4]. A powerful matcher, upon which the present work is based, Is proposed in [3]; this report also describes many of the ideas in PRESS. PRESS and the matcher extensions are written in PROLOG [7].

The goal of the resesrch described here is to test the search control technique of meta-level inference and the powerful algebraic manipulation methods of PRESS on some hard problems. We consider the problems of solving the general quadratic and cubic equations and a general trigonometric equation. The solutions of these problems given in standard algebra textbooks all have a "magic" element. An expression is drawn out of a hat with a flourish of "Consider the following term •••"• As a side effect of applying the PRESS equation solving methods, we show how the magic element can be understood. In fact, the program Is able to derive the solutions to several of these hard problems.

The PRESS solutions constitute a rational reconstruction of part of the history of algebra: we are not saying that this is how these solutions were first discovered, but that our program does show how they could have been discovered by a rational process of analysis of the problem, in terms of the tools available for its solution.

PRESS uses multiple sets of rewrite rules, employing meta-level reasoning and descriptions to guide that application and hence control search. Some of the Important rewrite rule sets are:

Isolation — PRESS tries applying Isolation rules when there is a single occurrence of the unknown in the equation. Isolation rules are applied to strip away surrounding functions and operators from the unknown, finally resulting in an equation with the unknown on one side by itself, and some expression (free of the unknown) on the other. A typical isolation rule is

arcsin x - b -> x = sin b.

collection — Collection rules serve to reduce the number of occurrences of the unknown, so that isolation can be applied. A typical collection rule is

uw + vw -> (u+v)w

which collects relative to w.

attraction — Attraction rules move occurrences of the unknown closer together in the expression tree, so that perhaps a collection rule can be applied. A sample rule is

$\log_b u + \log_b v \rightarrow \log_b uv$,

which attracts u and v.

To apply a rewrite rule to an expression, PRESS uses a matcher that knows about the commutativlty and associativity of addition and multiplication. For example, to apply the collection rule

uw + vw -> (u+v)w

Computing resources for this research were provided by Science Research Council grant number GR/A 37954. A. Borning was supported at the University of Edinburgh by a NATO Poatdoctoral Fellowship from the National Science Foundation.

Current address:
Computer Science Department, FR-35
University of Washington
Seattle, Washington 98195
USA

to the expression

xy + s(3x)

in order to collect the two occurrences of x, the PRESS matcher would substitute x for w, y for u, and 3s for v. The result of applying the rule would be

(y+3s)x.

The matcher used the commutatlvlty and associativity of multiplication in accomplishing the match.

An application of collection is the crucial step in the solution of some equations. Reflecting this, human mathematicians will try quite hard to find and apply a collection rule to an expression. For example, the standard solution of the general quadratic equation

$ax^2 + bx + c « 0$

uses the collection rule

$u^2 + 2uv + v^2 \rightarrow (u+v)^2$.

All the other steps of the solution are either preparations for applying the rule, or subsequent isolation steps. (However, a different terminology is usually used — rather than talking about rewrite rules, mathematicians talk about identities. Also, the process of applying the above collection rule is often presented in "compiled form*' as the operation of completing the square.) Similarly, the standard solution of the trigonomtric equation

a sin x 4- b cos x - c

depends critically on the use of the rule

cos u sin v + sin u cos v -> sin(u+v).

(Descriptions of the solutions of these equations may be found in [9].)

However, the application of these rules — matching the left hand side of a rule with an expression — cannot be accomplished by using simple pattern matching and information about associativity and commutatlvlty. What additional techniques are required? As part of an investigation of this question, an experimental matcher has been embedded in PRESS that can, among other things, solve both of the above equations from first principles.

2. The Matching Algorithm

The matcher is called with descriptions of the expression and pattern to be matched. If the match is successful, a transform is returned, consisting of a series of substitutions and arithmetic operations, such that the result of applying the transform to the pattern would be algebraically equal to the expression.

When called, the matcher first checks for simple cases. If the expression and pattern are identical, the match succeeds trivially, and the null transform is returned. If the pattern consists solely of a pattern variable, the match succeeds again, and a transform consisting of the single substitution "variable -> expr" is returned.

Otherwise, the matcher must try harder. The matcher has two ways of accomplishing a non-trivial match: by recursively matching corresponding parts of the expression and the pattern, or by solving algebraically for the value of a pattern variable.

2.1. Recursively Matching Parts of Expressions

In general, to match two complex expressions, the matcher will first check that the principal operators or functions are the same, and will then match the corresponding arguments. For example, consider matching the expression $\log_e a$ with the pattern log v, where v is a pattern variable. The matcher first checks that the functions log are the same, and then calls Itself recursively to match e with e, and a with v.

The matcher knows about the commutatlvlty and associativity of addition and multiplication. When matching two sums or products, the matcher puts all the terms In each sum or product into an unordered bag. It then has available a range of alternatives in matching the two bags, among the more important being:

- If both bag8 are empty, the match succeeds trivially.
- The matcher can pick a term from each bag and call itself recursively to match the two terms. In using this alternative, the matcher will pick the most complex term from one bag, using a simple complexity metric. Then, it will pick an appropriate term from the other bag by performing a fussy match between the term from the first bag and candidate terms from the other bag. (See Section 3 for a description of fussy matching.)
- If the term in either the expression or the pattern is free of the unknown, the matcher can permit the match to succeed by adding or multiplying each side of the rule by a term, if applying the operation will not Invalidate previously matched parts of the expression and pattern.
* If the pattern contains a pattern variable, the matcher can try to solve for Its value algebraically. (See Section 2.2.)

When matching a sum against any other expression (including a product), the matcher will convert the other expression into a plus bag with just the one element. Matching a product against any other expression (except a sum) is handled analogously.

2.2. Solving Algebraically for the Value of a Pattern Variable

The other principal technique for accomplishing a match la to solve algebraically for the value of a pattern variable. An equation is constructed whose two sides are the expression and pattern to be matched, and presented to the main equation solving routine. Use of this technique Increases the power of the matcher considerably, as it puts the full capabilities of the equation solver at the matcher's disposal.

In solving equations of this kind, a particular rather than a general solution is wanted. The equation solver Is told about this by adding an appropriate aaaertlon to the data base, so that only a single solution is returned, with alternate solutions being generated only if the program backtracks.

3. Search Control

The matcher has available a considerable range of strategies for accomplishing a match; some of these strategies, such as solving algebraically for the value of a pattern variable, can be expensive to use. Therefore, It is Important that the search Involved In accomplishing a match be tightly controlled. The main technique for doing this is the use of fuzzy matching as a preliminary check before the full matcher is invoked. (Note that we are using the term "fuzzy** in a different sense than as in "fuzzy logic**)• Fuzzy matching is used both for the initial selection of a collection rule, and for the selection of a pair of terms to match from two bags. Another technique for controlling search is the complexity heuristic for deciding which term in a bag to look at next.

To check for a fuzzy match, the program computes the features terns of the expression and pattern, and then matches these using the normal PRESS matcher (which is comparatively inexpensive). The algorithm for extracting a features term gives special status to the unknown, reflecting the fact that the matcher can often deal with miscellaneous expressions that are free of the unknown.

- If the expression is the unknown itself, then its features term is the unknown as well.
- If the expression is free of the unknown, Its features term is the expression "mumble".
- To compute the features term of a sum, the features term of each term in the sum are found. All "mumbles" are discarded; the features term is then a sum consisting of the remaining features terms. Products are handled analogously.
- Integer exponents of expressions not free of the unknown remain themselves.
- The features term of any other complex expression is found by computing the featurea term of each argument, and returning a new term with each argument replaced by its corresponding features term.

For example, suppose that x is the unknown. Then the features term of a cos y is "mumble", the features term of ax+b Is x, and the features term of a $\sin(x^2)+\cos(y^2)$ is $\sin(x^2)$.

These techniques have proven to be quite powerful: most spurious matches are rejected during fuzzy matching, and little search is done using the full matcher. To handle the search that does occur, the matcher uses the depth-first search provided by the built-in PROLOG backtracking mechanism, along with a memo procedure to save the results of matches in case they are needed again.

The current search control methods are for the most part adequate for matches that are eventually successful, and for matches that can't succeed (and are detected as such by the fuzzy matcher). The matcher takes considerably longer on matches that pass the fuzzy matcher, but eventually fall. For example, if one asks the system to find the solution to the general cubic equation

$$ax^3 + bx^2 + ex + d = 0,$$

it will (reasonably enough) attempt to apply the collection rule

$$u^3 + 3u^2v + 3uv^2 + v^3 \rightarrow (u+v)^3.$$

This match eventually fails, but only after considerable backtracking.

4. An Example — Deriving the Solution for the General Quadratic Equation

The operation of the matcher will now be illustrated by an example. Because of space limitations, a summary is presented here; a complete annotated trace of the matcher's operation on this and other problems, as well as a fuller description of the program, may be found in [2].

To start things off, the user asks PRESS to solve the equation

$$ax^2 + bx + c \blacksquare 0$$

for x. PRESS decides that Isolation is not applicable, since there are two occurrences of the unknown. It therefore tries to collect these two occurrences. In searching for an applicable collection rule, the program first performs a fuzzy match between the expression and the pattern part of each potentially applicable collection rule. The fuzzy matcher extracts a features term from the expression, and searches for a rule with matching features. The features term of the left hand side of the equation is

$$x^2 + x,$$

the coefficients and the constant term c having been regarded as relatively unimportant.

One of the collection rules known to PRESS is

$$u^2 + 2uv + v^2 \rightarrow (u+v)^2$$

which collects relative to u. When the unknown x is substituted for u, the features term of the rule is also

$$x^2 + x.$$

The program therefore selects this rule and tries to apply it to the left hand side of the equation. (This is in fact the only collection rule known to PRESS that will pass the fuzzy match.)

The full matcher is now invoked to match the pattern part of the rule with the quadratic expression. Since the principal operator of both the expression and pattern is +, the matcher converts to a bag representation. As previously described, there are a number of ways in which two bags can be matched. The matcher tries one of these methods: picking a term from each bag and matching those two terms. It selects the term ax^2 from the expression (on the grounds that it is the most complex), and then chooses a term with matching features from the rule, in this case x •

The matcher now calls itself recursively on these two terms. Since the principal operator of ax is times, the matcher again converts both terms to a bag representation. The x^2 term from the pattern is converted to a times bag with one element. The matcher picks the x^2 terms from each bag, and matches them trivially. After that, however, it must match the expression bag, which still has the "a" left in it, with the now empty pattern bag. The previously used strategy of picking a term from each product is no longer applicable. Instead, the matcher decides that the "a" should be dealt with by multiplying both sides of the rule by "a". This result is returned as a transform.

The two remaining terms in the pattern bag are multiplied by "a", and the matcher is called recursively on the remaining parts of the expression and rule, which are now

$$bx + c$$

and

$$2xva + v^2a.$$

respectively.

Again converting to a bag representation, the matcher recursively tries to match the two terms containing x, namely bx and 2xva. The two x's are matched trivially. The matcher then makes several unsuccessful attempts to match b with a term from the pattern. (The strategy previously employed of multiplying both sides of the rule by some expression can no longer be used, since doing so would invalidate the already established match of the x^2 terms.)

After these failures, the matcher tries the other principal matching strategy, that of solving algebraically for the value of a pattern variable. The equation solving program is called recursively to solve for v in b=2va. The answer, v=b/2a, is easily obtained by isolation.

With this substitution for v, the terms bx and 2xva now match. This result is returned as a transform

$$v \rightarrow b/2a,$$

which is applied to the remaining term in the pattern bag, namely v^2a, to yield

$$b^2/4a.$$

The last two terms don't match each other. However, the matcher can complete the match by adding c to each side of the rule, and subtracting the $b^2/4a$ term.

The match of the expression $ax^2 + bx + c$ and the pattern $x^2 + 2xv + v^2$ is now complete. The matcher returns the following transform:

```
* a
+ c
- b²/4a
v -> b/2a
```

```
+ c
- b²/4a
v -> b/2a
```

The pattern will match the expression if the transform listed is applied to it (multiply each side of the rule by a, then add c to each side, subtract $b^2/4a$ from each side, and substitute b/2a for v). The rule remains a valid collection rule after the transform has been applied to each side of it. So the transform is applied to the replacement part of the rule, and the altered collection rule is applied to the original equation. The result is

$$(x+b/2a)^2a + c - b^2/4a - 0.$$

Most of the work is now done. The new equation has a single occurrence of x, and is easily solved by Isolation to yield the two roots of the quadratic. The program's solution, written in two-dimensional notation, is:

$$x = -\frac{b}{2a} + \sqrt{\frac{b^2}{4a^2} - \frac{c}{a}}$$

or

$$x = -\frac{b}{2a} - \sqrt{\frac{b^2}{4a^2} - \frac{c}{a}}$$

If the fractions are put over a common demonlnator, the answers simplify to the usual expressions. (A package for performing this sort

of simplification has been been incorporated into PRESS, but after this trace was taken.)

5. Other Problems Solved using the Matcher

Another equation solved using the matcher is

a sin x + b cos x • c*

The features term of the left hand side of the equation is

sin x + cos x*

One of the trigonometric collection rules is

cos u sin v + sin u cos v -> sln(u+v),

which collects relative to u. When x is substituted for u, the features term of this rule is

cos x + sin x,

which matches the features term of the left hand side of the equation* (Note that the sin v and cos v factors are dropped, as they are free of the unknown.) The first term of the expression and the second term of the pattern, namely a sin x and sin x cos v, are matched by multiplying each side of the pattern by a/cos v. The matcher next matches the remaining terms from the expression and pattern, which are now

b cos x

and

(cos x sin v)a/cos v

respectively* The two cos x factors are matched trivially* To complete the match, the matcher solves algebraically for a value for the pattern variable v. In the process, the (recursively invoked) equation solver employs another collection step to collect the two occurrences of v by using the rule

sin w/cos w -> tan w*

As an application of the augmented PRESS program, a procedure was written that compiles specialised methods for solving certain kinds of equations* The user gives the program the general form of an equation, such as the quadratic or the trlgonometlc equation* The program solves the equation using the powerful matcher, and then asserts a new PROLOG procedure for solving Instances of that equation*

The matcher has also been used to solve equations using a change of unknown, such as the equation

$_5$2y . $_5$y+l $_+$ 6 . 0#

Here, the equation is matched against an equation whose solution is known (in this case the quadratic) to generate the change of unknown

5^V -> x.

6. Work In Progress

A milestone in the history of algebra and equation solving was the solution of the general cubic equation* We have recently made progress toward getting PRESS to solve this equation* One solution we are working towards, described in *[l],* uses the collection rule

$\cos^3 u - 3/4 \cot u \rightarrow 1/4 \cos(3u)$.

The program currently will solve the equation with some help. PRESS reduces the general equation to

$8s^3 + hz + g - 0$

using a linear substitution* However, at that point it must be explicitly told to try the "magic substitution'[1] z - w cos u. We hope to extend PRESS so that it will be able to solve the cubic and quartic equations from first principles without human intervention, using both this solution path and others.

7. Related Work

To the best of the authors' knowledge, no other algebraic matcher has been implemented that compares in power to the one described here, which builds in the associative, commutative, and distributive laws, and has available the full power of the equation solving system for solving for a value for a pattern variable.

The PRESS program (without the extensions described here) includes a matcher that knows about the commutetlvity and associativity of addition and multiplication. The matcher in MACSYHA (5), [6] knows about commutatlvity and associativity. It also provides defaults for missing summands, factors, and exponents, and will distribute products over sums to accomplish a match.

There is considerable literature concerning pattern matching in theorem proving — see [8] for a survey of the state of the art. None of this work is very relevant to the present research. Most of these algorithms build in only a very few axioms, whereas the matcher described here builds in an indefinite number. On the other hand, these algorithms are usually for two-way matching and come with a completeness proof, which we could not have.

REFERENCES

[1] Barnard and Child.
Higher Algebra.
MacMillan, 1936.

[2] Bornlng, A., and Bundy, A.
Using Matching In Algebraic Equation Solving.
Research Report 158, Dept. of Artificial Intelligence, Edinburgh, 1981.
An expanded version of this IJCAI paper. Also available as Technical Report No. 81* 05-01, Computer Science Department, University of Washington.

[3] Bundy, A.
Analysing Mathematical Proofs (or reading between the lines).
In Proceedings of the 4th IJCAI. Georgia, 1975.
An expanded version is available from Edinburgh as DAI Research Report No. 2.

[A] Bundy, A. and Welham, B.
Using Meta-level Inference for Selective Application of Multiple Rewrite Rules in Algebraic Manipulation.
Artificial Intelligence , 1981.
In press. Also available from Edinburgh as DAI Research Report No. 121.

[5] Fate man, R. J.
Essays in Algebraic Simplification.
PhD thesis, MIT, 1972.
Also available as MAC TR-95.

[6] Mathlab Group.
MACSYMA Reference Manual
MIT, 1977.

[7] Pereira, L., Pereira, F., and Warren, D.
User's Guide to DECsystem-10 PROLOG
Dept. of Artificial Intelligence, Edinburgh, 1978.

[8] Raulefs, P., Siekmann, J, Szabo, P., and Unvericht, E.
A Short Survey on the State of the Art in Matching and Unification Problems.
AISB Quarterly (32):17-21, December, 1978.

[9] Tranter, C. J.
Advanced Level Pure Mathematics.
English Universities Press, 1970.