

The MuPAD solver

MuPAD seminar, 10.05.2001

Stefan Wehmeier



Goal of the solver

Given: a statement $\phi(x_1, \dots, x_n, y_1, \dots, y_k)$ and some variables x_1, \dots, x_n .

To find: the set $S(y_1, \dots, y_k) \subseteq \mathbb{C}^n$ such that

$$\phi(c_1, \dots, c_n, y_1, \dots, y_k) \Leftrightarrow (c_1, \dots, c_n) \in S(y_1, \dots, y_k).$$

But: this is too hard!



Calls to solve not subject of this talk

The solver, in a narrower sense, does not contain:

- numerical solving
- solving differential equations
- solving recurrence equations

Remark:

Floats in the coefficients do not cause the numerical solver to be called.



Structure of the solver

We distinguish the following types of problems

- solving one equation for one variable

```
>> solve(x^2=9, x)
```

```
{-3, 3}
```

- solving a system of equations or solving for several variables



```
>> solve({x^2+y=2, x+3*y=5})
```

```
{ --          1/2          1/2          --
{ |          13          13          |
{ | x = 1/6 - -----, y = ----- + 29/18 |,
{ --          6          18          --

--          1/2          1/2 -- }
|          13          13      | }
| x = ----- + 1/6, y = 29/18 - ----- | }
--          6          18      -- }
```

- solving inequalities

```
>> solve(x^2 < 4, x)
```

```
] -2, 2[
```

- finding all values of x such that $f(x)$ belongs to a specified set



```
>> solve(x^2 in Dom::Interval(3, 5), x)
```

```
      ]3^(1/2), 5^(1/2)[ union ]-5^(1/2), -3^(1/2)[
```

and some options that invoke special methods.



The equation solver

Called if:

the first argument is an equation, polynomial, or an expression, and the second argument is an identifier or indexed identifier

Returns:

the set of all zeroes of the expression/ all solutions of the equation that are compatible with the properties of the variable to solve for

Related: the **Mu**PAD concept for sets. I will not talk about this today.



Algorithms to solve equations

MuPAD tries, one after another, subroutines that

- handle products as well as powers with constant exponent
- check whether the input is a polynomial; if yes, use the polynomial solver
- handle equations $f(A(x)) = C$ for certain special functions f and expressions $A(x)$
- rewrite the input in terms of a piecewise defined object
- handle Re and Im
- rewrite equations by (algebraic) systems



- rewrite/simplify the expression and start over the whole algorithm

If nothing helps, an unevaluated call to `solve` is returned.

In any case, the equation is solved over \mathbb{C} first; compatibility with the properties of the variable is checked at the end.

To do:

Work out better methods for solving over the reals.



Special cases for products and powers

- The zeroes of a product are those zeroes of its factors where every factor is defined.
- The zeroes of $f(x)^a$ are the zeroes of $f(x)$ unless a is negative
- $f(x)^n = a$ may be handled by solving $y^n = a$ and then $f(x) = y$.

If none of these cases matches, the input is expanded.

It is important to try those special cases before expanding:



```
>> simplify(solve((x+3)^5=32, x))
```

```
{
  1/2
{
  5
{ -1, - ---- - 1/2 I (10 - 2 5 ) - 7/2,
{
  2
```

```

          1/2
      1/2 1/2 5
1/2 I (10 - 2 5 ) - ---- - 7/2,
          2
```

```

  1/2
5
---- - 1/2 I (2 5 + 10) - 7/2,
  2
```

```

  1/2 }
5      1/2 1/2 }
---- + 1/2 I (2 5 + 10) - 7/2 }
  2 }
```



The polynomial solver

The zeroes of a polynomial are the zeroes of its irreducible factors. Each factor $f(x)$ is solved as follows:

- decomposition: try to write $f(x) = g(h(x))$, then solve $g(y) = 0$ and $h(x) = y$
- use a special formula for roots of unity if $f(x)$ is of the form $x^n + a$.
- use explicit formulas for degrees ≤ 4 . (Thanks to Walter, these formulas are now numerically stable for floating point coefficients.)

If none of these methods work, the result is a RootOf - expression.

Remark:

The option `MaxDegree=n` may be used to switch off Cardano formulas for degrees $> n$.



To do:

If f is a composition of polynomials with parameters in the leading coefficients, `polylib::decompose` cannot detect this. Piecewise defined solutions for the components still need to be handled.



Handling of special functions

Let f be a special function. An equation $f(g(x)) = C$ is solved by solving $f(y) = C$ and $g(x) = y$.

This is done by case analysis on `f`, inside the method `solveLib::isolate`.

Remark:

A function environment `f` may have an `isolate` attribute to overload this part of the solver.

Remark:

I have no idea how to handle special functions with more than one argument yet. However, there are special methods for `_plus`, `_mult`, and `_power`.



```

>> solve(exp(x^2)=1, x)

{ (2*I*PI*X87)^(1/2) | X87 in Z_ } union

  { -(2*I*PI*X89)^(1/2) | X89 in Z_ }

>> solve(exp(x^2)=1, x)

{ (2*I*PI*X87)^(1/2) | X87 in Z_ } union

  { -(2*I*PI*X89)^(1/2) | X89 in Z_ }
>> solve(exp(y)=1, y)

  { 2*I*PI*X369 | X369 in Z_ }
>> solve(x^2 in %, x)

{ (2*I*PI*X371)^(1/2) | X371 in Z_ } union

  { -(2*I*PI*X373)^(1/2) | X373 in Z_ }

```



Heuristics for sums and powers

- Rewrite $f(g(x)) - f(h(x)) = 0$ as $f(g(x)) = f(h(x))$ and apply an inverse function of f to both sides
- Rewrite $f(x)^{-a} = C$ as $f(x) = C^a$ if $C \neq 0$.
- Rewrite $f(x)^{p/q} = C$ as $f(x)^p = C^q$, and check the solutions in the end.
- Rewrite powers by \exp and \ln .



Rewriting functions by piecewise functions

Some particular discontinuous functions are replaced by piecewise defined functions:

```
>> rewrite(abs(x^2) - sign(x), piecewise)
```

```

      / 2      2      2
piecewise| x  - 1 if 0 < x, x  if x = 0, x  + 1 if x < 0,
      |
      \

```

```

      2      2      x      \
Im(x)  + Re(x)  - ----- if not x in R_ |
      2      2 1/2      |
      (Im(x)  + Re(x) )  /

```

This can be solved branchwise:



```
>> solve(abs(x^2) - sign(x) = 1, x)
```

```
1/2  
{2 }
```



How to handle Re and Im

If an equation depends on $\text{Re}(x)$ and $\text{Im}(x)$, replace the argument x by $u + I * v$ (u, v real).

```
>> solve(Im(x^2) - x = 2, x)
```

```
{-2}
```

```
// Internally, this is done as follows:
```

```
>> assume({u, v}, Type::Real):
```

```
>> Im((u+I*v)^2) - u - I*v
```

```
2 u v - I v - u
```

Splitting into real- and imaginary part gives a system $v = 0, 2uv - u = 2$, which **Mu**PAD can solve.



Rewriting equations as systems

Suppose we have to solve $f(x) = 0$, where f is non-polynomial.

Idea: construct a tower $\mathbb{C}(x) = K_0 \subseteq \dots \subseteq K_n$ of simple field extensions $K_i = K_{i-1}(\alpha_i)$ such that $f \in K_n$.

If all extensions are algebraic, we can write down a system consisting of f and the minimal polynomials of the α_i .



Example:

$$x^{1/2} + (x^2 + 1)^{1/2} + x - 5 = 0$$

becomes

$$\alpha_1 + \alpha_2 + x - 5 = 0$$

$$\alpha_1^2 - x = 0$$

$$\alpha_2^2 - x^2 - 1 = 0$$

The system solver can do this.



```
>> solve({alpha1 + alpha2 + x - 5, alpha1^2 - x,
&> alpha2^2 - x^2 - 1}, [x, alpha1, alpha2])
```

```
{ --
{ |      2
{ |      42 alpha2      4 alpha2
{ |  x = ----- + ----- + 2/59,
{ --      59          59

      2
      4 alpha2      101 alpha2
alpha1 = 293/59 - ----- - -----,
                59          59

-- }
                2          3          | }
alpha2 = RootOf(79 X6  - 524 X6 + 4 X6  + 697, X6) | }
-- }
```

Resubstituting α_2 gives



```
>> solve(x^(1/2)+ (x^2+1)^(1/2)+ x= 5, x)
```

```
{ 42/59*X7 + 4/59*X7^2 + 2/59 | X7 in RootOf(79*X6^2 - 524*X6\  
+ 4*X6^3 + 697, X6) }
```

Currently, **MuPAD** tries to handle such towers if they are algebraic. **MuPAD** can also handle the case when $f(x) = g(h(x))$ and g is algebraic over $h(x)$:

```
>> solve(sin(x)^(1/2)+ (sin(x)/2 + 1)^(1/2)= 5, x)
```

```
{ 2*PI*X33 + arcsin(152 - 40*13^(1/2)) | X33 in Z_ } union
```

```
{ 2*PI*X27 + arcsin(40*13^(1/2) + 152) | X27 in Z_ } union
```

```
{ PI + 2*PI*X35 - arcsin(152 - 40*13^(1/2)) | X35 in Z_ }
```

```
union
```

```
{ PI + 2*PI*X29 - arcsin(40*13^(1/2) + 152) | X29 in Z_ }
```



NOTE THAT:

Of course, $x^{1/2} = y$ is not equivalent to $x = y^2$. The solver can sometimes single out those solutions with too small polar angle, and sometimes not.

Remark:

You can skip this step using the option `DontRewriteBySystem`.



Very last attempts before giving up

Finally, the solver attempts to rewrite the input in equivalent form.

NOTE THAT:

But this may result in a non-equivalent expression, defined for a different set of values of the variable! But the solver takes care not to output solutions where the expression is not defined.

```
>> solve(sin(x)^2+cos(x)^2=1, x)
```

```
C_ minus { PI + 2*X4*PI | X4 in Z_ }
```

To do:

Revise our expression manipulation functions.



Known problems/limitations of the equation solver

- Algebraic dependencies may be overlooked
- does not know all special functions yet

And: probably many unknown problems/errors due to too few tests.

