Towards Symbolic Manipulation on Operator Records

Philip Hannebohm   Bernhard Bachmann

OpenModelica Annual Workshop 2021

FH Bielefeld
University of
Applied Sciences

What are Operator Records?

- Complex Numbers

Algebraic Structures

Backend

- Solve/Simplify
- Differentiate

More Examples

Open Questions

What are Operator Records?

Informal

Record

- Defines a new type
- Container for grouping data together
- Often used for annotations, parameter sets

Informal

Record

- Defines a new type
- Container for grouping data together
- Often used for annotations, parameter sets

Operator Record

- Similar to `record`, groups data
- Can define operators like `'+'`, `'*'`, `'=='`
- Those operators have algebraic properties

Modelica Language Specification Version 3.5

- 4.6 Specialized Classes
- 6 Interface or Type Relationships
- 9.2 Generation of Connection Equations
- 10.3.4 Reduction Functions and Operators
- 12.6 Record Constructor Functions
- 12.7 Declaring Derivatives of Functions
- 12.9 External Function Interface
- 14 Overloaded Operators

https://specification.modelica.org/maint/3.5/MLS.html

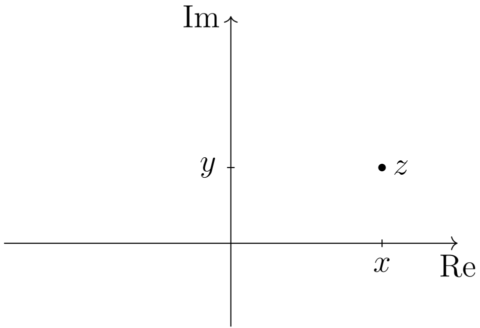
```
operator record OpRec
  Real comp1;
  // more components ...

  encapsulated operator Op1
    function f1
      import OpRec;
      // ...
    end f1;
  end Op1;

  // more operators ...
end OpRec;
```

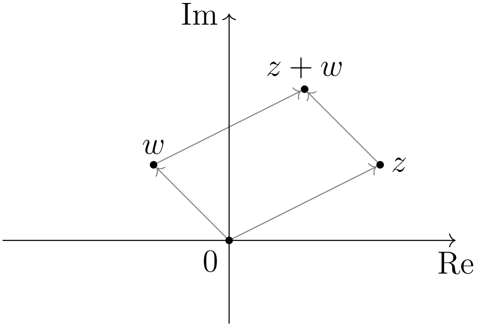
Complex Numbers

$$i^2 = -1$$
$$z = x + iy$$



$$i^2 = -1$$
$$z = x + iy$$

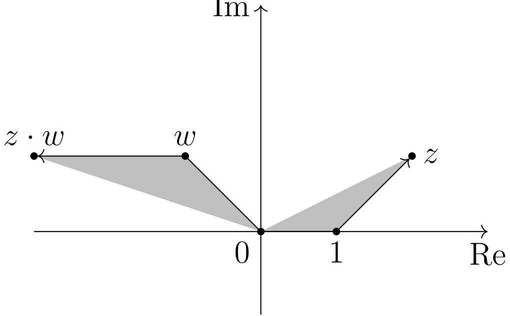
$$(x_1 + iy_1) + (x_2 + iy_2) =$$
$$(x_1 + x_2) + i(y_1 + y_2)$$



$$i^2 = -1$$
$$z = x + iy$$

$$(x_1 + iy_1) + (x_2 + iy_2) =$$
$$(x_1 + x_2) + i(y_1 + y_2)$$

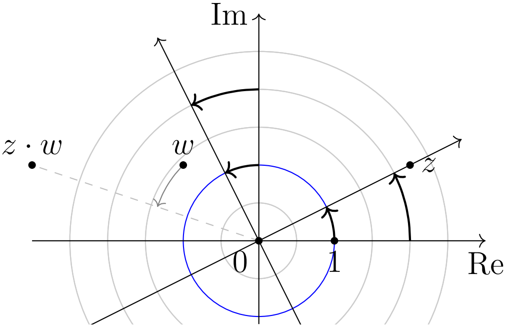
$$(x_1 + iy_1) \cdot (x_2 + iy_2) =$$
$$(x_1 x_2 - y_1 y_2) + i(x_1 y_2 + y_1 x_2)$$



$$i^2 = -1$$
$$z = x + iy$$

$$(x_1 + iy_1) + (x_2 + iy_2) =$$
$$(x_1 + x_2) + i(y_1 + y_2)$$

$$(x_1 + iy_1) \cdot (x_2 + iy_2) =$$
$$(x_1 x_2 - y_1 y_2) + i(x_1 y_2 + y_1 x_2)$$
$$= r_1 e^{i\theta_1} \cdot r_2 e^{i\theta_2} = (r_1 r_2) e^{i(\theta_1 + \theta_2)}$$



Modelica Standard Library

```
Complex.mo
Modelica/ComplexMath.mo
Modelica/ComplexBlocks/*
```

Modelica Standard Library

```
within ;
operator record Complex "Complex number with overloaded operators"

  replaceable Real re "Real part of complex number" annotation(...);
  replaceable Real im "Imaginary part of complex number" annotation(...);

  encapsulated operator 'constructor' "Constructor"
    function fromReal "Construct Complex from Real"
      import Complex;
      input Real re "Real part of complex number";
      input Real im=0 "Imaginary part of complex number";
      output Complex result(re=re, im=im) "Complex number";
    algorithm
      annotation(...);
    end fromReal;
    annotation (...);
  end 'constructor';

  encapsulated operator function '0' "Zero-element of addition (= Complex(0))"
    import Complex;
    output Complex result "Complex(0)";
  algorithm
    result := Complex(0);
    annotation(...);
  end '0';

  encapsulated operator '-' "Unary and binary minus"
    function negate "Unary minus (multiply complex number by -1)"
      import Complex;
      input Complex c1 "Complex number";
      output Complex c2 "= -c1";
    algorithm
      c2 := Complex(-c1.re, -c1.im);
      annotation(...);
    end negate;

    function subtract "Subtract two complex numbers"
      import Complex;
      input Complex c1 "Complex number 1";
      input Complex c2 "Complex number 2";
      output Complex c3 "= c1 - c2";
    algorithm
      c3 := Complex(c1.re - c2.re, c1.im - c2.im);
      annotation(...);
    end subtract;
    annotation (...);
  end '-';

  encapsulated operator '*' "Multiplication"
    function multiply "Multiply two complex numbers"
      import Complex;
      input Complex c1 "Complex number 1";
      input Complex c2 "Complex number 2";
      output Complex c3 "= c1*c2";
    algorithm
      c3 := Complex(c1.re*c2.re - c1.im*c2.im, c1.re*c2.im + c1.im*c2.re);
      annotation(...);
    end multiply;

    function scalarProduct "Scalar product c1*c2 of two complex vectors"
      import Complex;
      input Complex c1[:] "Vector of Complex numbers 1";
      input Complex c2[size(c1,1)] "Vector of Complex numbers 2";
      output Complex c3 "= c1*c2";
    algorithm
      c3 :=Complex(0);
      for i in 1:size(c1,1) loop
         c3 :=c3 + c1[i]*c2[i];
      end for;
      annotation(...);
    end scalarProduct;
    annotation (...);
  end '*';

  encapsulated operator function '+' "Add two complex numbers"
    import Complex;
    input Complex c1 "Complex number 1";
    input Complex c2 "Complex number 2";
    output Complex c3 "= c1 + c2";
  algorithm
    c3 := Complex(c1.re + c2.re, c1.im + c2.im);
    annotation(...);
  end '+';

  encapsulated operator function '/' "Divide two complex numbers"
    import Complex;
    input Complex c1 "Complex number 1";
    input Complex c2 "Complex number 2";
    output Complex c3 "= c1/c2";
```

Components

- listed at the top
- referenced by name

Operators

- `functions` listed inside `operator`
- `annotations`

```
operator record Complex
  replaceable Real re "Real part";
  replaceable Real im "Imaginary part";

  encapsulated operator 'constructor'
    function fromReal
      import Complex;
      input Real re;
      input Real im = 0;
      output Complex c(re = re,
                       im = im);
    algorithm
      annotation(Inline = true, ...);
    end fromReal;
    annotation (...);
  end 'constructor';
```

Operators

- `operator function`
- Binary, Unary, Nullary

Nullary '0'

Supposed to be the neutral element for '+'
Can it be deduced from the definition of '+'?

```
encapsulated operator function '+'
  import Complex;
  input Complex c1;
  input Complex c2;
  output Complex c3;
algorithm
  c3 := Complex(c1.re + c2.re,
                c1.im + c2.im);
  annotation(
    Inline = true,
    smoothOrder = 100, ...);
end '+';

encapsulated operator function '0'
  import Complex;
  output Complex c;
algorithm
  c := Complex(0);
end '0';
```

Inverse Operator

- negate for unary '-'
- subtract for binary '-'

One of them should be redundant since

$$c_1 - c_2 = c_1 + (-c_2)$$

Can we deduce one from the other?

```
encapsulated operator '-'
  function negate
    import Complex;
    input Complex c1;
    output Complex c2(re = -c1.re,
                      im = -c1.im);
  end negate;

  function subtract
    import Complex;
    input Complex c1;
    input Complex c2;
    output Complex c3(
      re = c1.re - c2.re,
      im = c1.im - c2.im);
  end subtract;
end '-';
```

Multiplication

Similar structure to addition

Scalar Product

- Array operands, scalar result
- Mathematically ambiguous, usually c_1 is conjugated first

```
encapsulated operator '*'
  function multiply
    import Complex;
    input Complex c1;
    input Complex c2;
    output Complex c3(
      c1.re*c2.re - c1.im*c2.im,
      c1.re*c2.im + c1.im*c2.re);
  end multiply;

  function scalarProduct
    import Complex;
    input Complex c1[:];
    input Complex c2[size(c1,1)];
    output Complex c3;
  algorithm
    c3 := sum(c1[k]*c2[k]
      for k in 1:size(c1,1));
  end scalarProduct;
end '*';
```

Integers

Automatic simplified version for Integer multiple n

$$n \cdot c = \begin{cases} \sum_{k=1}^{n} c, & n > 0 \\ 0, & n = 0 \\ |n| \cdot (-c), & n < 0 \end{cases}$$

even if `'*'` isn't definded explicitly?

```
encapsulated operator '*'
  function multiply
    import Complex;
    input Complex c1;
    input Complex c2;
    output Complex c3(
      c1.re*c2.re - c1.im*c2.im,
      c1.re*c2.im + c1.im*c2.re);
  end multiply;

  function scalarProduct
    import Complex;
    input Complex c1[:];
    input Complex c2[size(c1,1)];
    output Complex c3;
  algorithm
    c3 := sum(c1[k]*c2[k]
      for k in 1:size(c1,1));
  end scalarProduct;
end '*';
```

Reciprocal

- Should be inverse of `'*'`
- Simplifications for `Real`?

```
encapsulated operator function '/'
  import Complex;
  input Complex c1;
  input Complex c2;
  output Complex c3 "= c1/c2";
protected
  Real d = 1/(c2.re*c2.re
              + c2.im*c2.im);
algorithm
  c3 := Complex(
    (c1.re*c2.re + c1.im*c2.im)*d,
    (c1.im*c2.re - c1.re*c2.im)*d);
end '/';
```

Exponentiation

Automatic simplified version for Integer exponent n?

$$c^n = \begin{cases} c \cdot c^{n-1}, & n > 0 \\ 1, & n = 0 \\ 1/c^{-n}, & n < 0 \end{cases}$$

```
encapsulated operator function '^'
  import Complex;
  input Complex c1;
  input Complex c2;
  output Complex c3 "= c1^c2";
protected
  Real lnz = 0.5*log(c1.re*c1.re
                  + c1.im*c1.im);
  Real phi = atan2(c1.im, c1.re);
  Real re = lnz*c2.re - phi*c2.im;
  Real im = lnz*c2.im + phi*c2.re;
algorithm
  c3 := Complex(exp(re)*cos(im),
                exp(re)*sin(im));
end '^';
```

Comparison

- ($\mathbb{C}$ has no notion of order, i.e. no `'<'`, `'>'`)
- One of `'=='`, `'<>'` is redundant, since

$$c_1 = c_2 \Leftrightarrow \neg(c_1 \neq c_2)$$

- Be careful in general with `'=='` on `Real`

```
encapsulated operator function '=='
  import Complex;
  input Complex c1;
  input Complex c2;
  output Boolean result;
algorithm
  result := c1.re == c2.re
        and c1.im == c2.im;
end '==';

encapsulated operator function '<>'
  import Complex;
  input Complex c1;
  input Complex c2;
  output Boolean result;
algorithm
  result := c1.re <> c2.re
          or c1.im <> c2.im;
end '<>';
```

Algebraic Structures

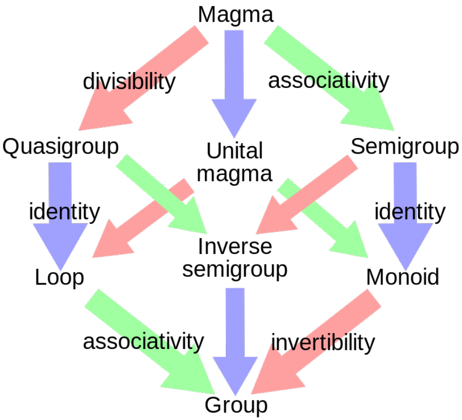
Groups

Definition

A set G with binary operator $\circ : G \times G \to G$

Rules

$(a \circ b) \circ c = a \circ (b \circ c)$	associative
$e \circ a = a \circ e = a$	identity
$a \circ a^{-1} = a^{-1} \circ a = e$	inverse



https://commons.wikimedia.org/wiki/File:Magma_to_group4.svg

Groups

Definition

A set G with binary operator $\circ : G \times G \to G$

Abelian Group

$$a \circ b = b \circ a \quad \text{commutative}$$

Examples

- $\mathbb{Z}$ with $+$, 0, $-n$
- $\mathbb{R} \setminus \{0\}$ with $\cdot$, 1, $\frac{1}{x}$
- $\mathbb{R}^n$ with $+$, $\mathbf{0}$, $-\mathbf{v}$
- $\ldots$

Groups

Definition

A set G with binary operator
$\circ : G \times G \to G$

Abelian Group

$$a \circ b = b \circ a \quad \text{commutative}$$

Examples

- $\mathbb{Z}$ with $+$, 0, $-n$
- $\mathbb{R} \setminus \{0\}$ with $\cdot$, 1, $\frac{1}{x}$
- $\mathbb{R}^n$ with $+$, $\mathbf{0}$, $-\mathbf{v}$
- $\ldots$
- $\mathbb{C} \setminus \{0\}$ with $\cdot$, 1, $\frac{1}{x}$

```
  // ...
  Complex a, b, c;
equation
  a*b = c "solved for b";
  // ...
```

```
  // ...
  Complex a, b, c;
equation
  a*b = c "solved for b";
  // ...
```

Scalarized: loop of size 2

```
a.re*b.re - a.im*b.im = c.re
a.re*b.im + a.im*b.re = c.im
```

```
  // ...
  Complex a, b, c;
equation
  a*b = c "solved for b";
  // ...
```

Scalarized: loop of size 2

```
a.re*b.re - a.im*b.im = c.re
a.re*b.im + a.im*b.re = c.im
```

Kept as record: simple assign

```
d := 1.0/(a.re*a.re + a.im*a.im)
b.re := (a.re*c.re + a.im*c.im)*d
b.im := (a.re*c.im - a.im*c.re)*d
```

Rings/Fields

Ring

A set K with two binary operators $+$ (commutative group) and $\cdot$ (associative, identity), where $\cdot$ distributes with $+$.

- integers, polynomials
- products and integer powers
- factoring

Rings/Fields

Ring

A set K with two binary operators $+$ (commutative group) and $\cdot$ (associative, identity), where $\cdot$ distributes with $+$.

- integers, polynomials
- products and integer powers
- factoring

Field

Like a ring, but $K \setminus \{0\}$ and $\cdot$ also form a commutative group.

- reals, complex
- nice algebraic rules
- multiplicative inverses
- Galois theory

Vector Spaces

Definition

A vector space V over a field F has vector addition $\oplus$ (commutative group) and scalar multiplication $\odot$.

Rules

$a \odot (b \odot \mathbf{v}) = (a \cdot b) \odot \mathbf{v}$	compatibile
$1 \odot \mathbf{v} = \mathbf{v}$	identity
$a \odot (\mathbf{u} \oplus \mathbf{v}) = a \odot \mathbf{u} \oplus a \odot \mathbf{v}$	distribute over V
$(a + b) \odot \mathbf{v} = a \odot \mathbf{v} \oplus b \odot \mathbf{v}$	distribute over F

Vector Spaces

Definition

A vector space V over a field F has vector addition $\oplus$ (commutative group) and scalar multiplication $\odot$.

Examples

- n-dimensional Euclidean space (3D real space)
- $m \times n$ matrix space
- Field extensions (like $\mathbb{C}$)

Vector Spaces

Definition

A vector space V over a field F has vector addition $\oplus$ (commutative group) and scalar multiplication $\odot$.

Modelica Arrays

Declarations

```
Real[3] x, y, z;
```

Equations

```
z = -2*y;
y = 3*x + 5*z;
```

Backend

Affected Modules

- simplify
- events/states
- alias sets
- partition
- causalize
- initialize
- tearing
- Jacobian
- solve

Affected Modules

- simplify
- events/states
- alias sets
- partition
- causalize
- initialize
- tearing
- Jacobian
- solve

Solve/Simplify

Solving Equations for Variables

- encoding expressions as a tree

Solving Equations for Variables

- encoding expressions as a tree
- rewrite rules

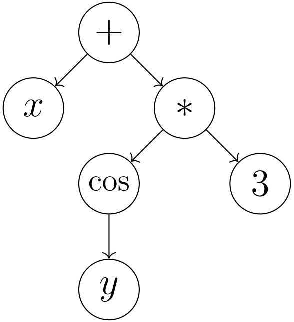
Solving Equations for Variables

- encoding expressions as a tree
- rewrite rules
- graph of equivalent expressions/equations

Solving Equations for Variables

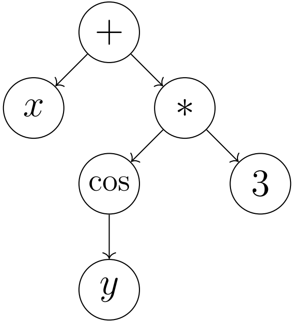
- encoding expressions as a tree
- rewrite rules
- graph of equivalent expressions/equations
- (efficient) graph traversal

Expression Trees



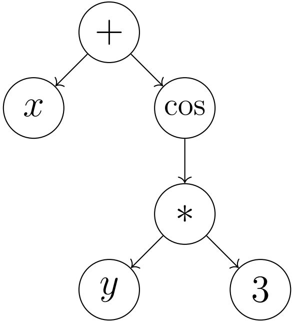
$$x + \cos y \cdot 3$$

Expression Trees



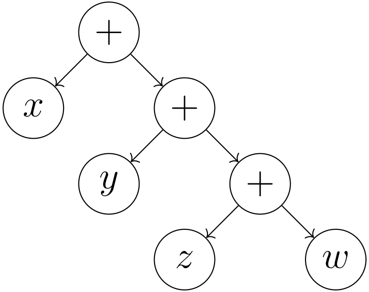
$$x + \cos(y) \cdot 3$$

Expression Trees



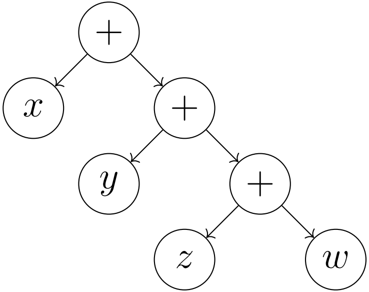
$$x + \cos(y \cdot 3)$$

Binary



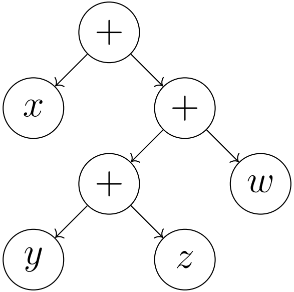
$$x + y + z + w$$

Binary



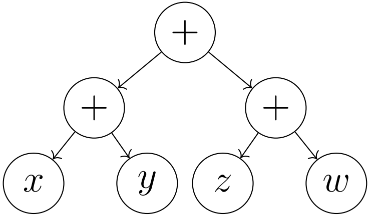
$$x + (y + (z + w))$$

Binary



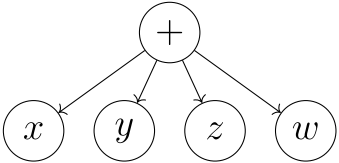
$$x + ((y + z) + w)$$

Binary



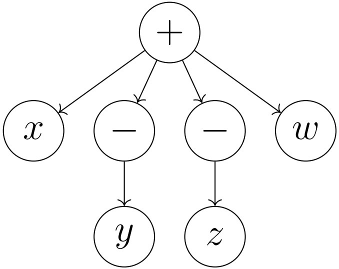
$$(x + y) + (z + w)$$

Multary



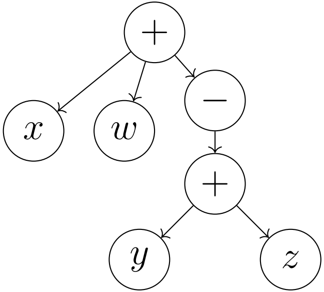
$$x + y + z + w$$

Multary



$$x - y - z + w$$

Multary



$$x + w - (y + z)$$

Algebra/Rewrite Rules

$$U * V + U * W \quad \Leftrightarrow \quad U * (V + W)$$

$$(U + V) \cdot (U - V) \quad \Leftrightarrow \quad U^2 - V^2$$

$$\frac{U^k}{U^n} \cdot U^m \quad \Leftrightarrow \quad U^{k+m-n}$$

$$a \cdot \frac{\left(\frac{c}{d}\right)}{b} \cdot \frac{e}{\left(\frac{f}{g}\right)} \quad \Leftrightarrow \quad \frac{a \cdot c \cdot e \cdot g}{b \cdot d \cdot f}$$

$$\cdots$$

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

Meta heuristics (PRESS)

isolate solve at single occurence

$$
\begin{array}{c}
0 = \ln x - 2 \\
\Big\downarrow {\scriptstyle +2} \\
2 = \ln x \\
{\color{red} x > 0}\ \Big\downarrow \text{ inverse function} \\
x = e^2 = 7.3890\ldots
\end{array}
$$

Meta heuristics (PRESS)

isolate solve at single occurence

polysolve solve polynomials

$$8x = 2x^3 + \frac{6}{x}$$

$$\Big\downarrow \text{normal form} \quad {\color{red} x \neq 0}$$

$$0 = x^4 - 4x^2 + 3$$

$$\Big\downarrow \text{completing the square}$$

$$x^2 \in \{1, 3\}$$

$$\Big\downarrow \text{isolate}$$

$$\ldots$$

Meta heuristics (PRESS)

isolate	solve at single occurence
polysolve	solve polynomials
collect	reduce occurences

$$2 = (e^x + 1) \cdot (e^x - 1)$$

$$\Big\downarrow \; (U + V) \cdot (U - V) \to U^2 - V^2$$

$$2 = (e^x)^2 - 1$$

$$\Big\downarrow \; \text{isolate}$$

$$\ldots$$

Meta heuristics (PRESS)

isolate	solve at single occurence
polysolve	solve polynomials
collect	reduce occurences
attract	bring occurences closer

$$4 = e^x \cdot e^{2x}$$

$$\Big\downarrow\ e^U \cdot e^V \to e^{U+V}$$

$$4 = e^{x+2x}$$

$$\Big\downarrow\ \text{collect}$$

$$\ldots$$

Meta heuristics (PRESS)

isolate	solve at single occurence
polysolve	solve polynomials
collect	reduce occurences
attract	bring occurences closer
homogenize	change of unknown

$$
\begin{array}{c}
0 = e^x + e^{3x} \\
\Big\downarrow \text{reduce to } e^x \\
0 = e^x + (e^x)^3 \\
{\color{red} y \geq 0}\ \Big\downarrow \text{substitute } y = e^x \\
0 = y + y^3 \\
\Big\downarrow \text{polysolve} \\
\ldots
\end{array}
$$

Meta heuristics (PRESS)

isolate	solve at single occurence
polysolve	solve polynomials
collect	reduce occurences
attract	bring occurences closer
homogenize	change of unknown
swap fn	transform functions

$$1 - x = \sqrt{3x - x^2}$$

$$\Big\downarrow \text{swap } \sqrt{\cdot} \text{ for } \cdot^2 \qquad {\color{red} 1 - x \geq 0}$$

$$(1 - x)^2 = 3x - x^2$$

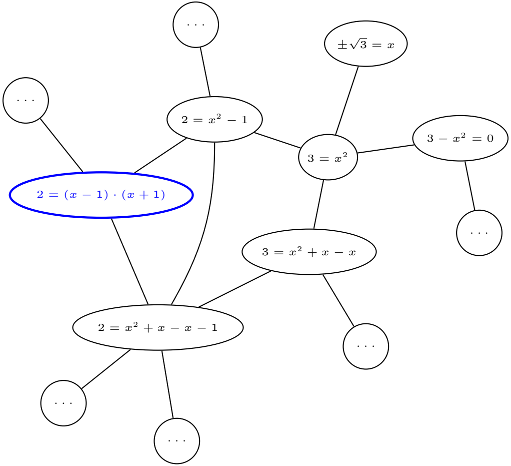
$$\Big\downarrow \text{polysolve}$$

$$\cdots$$

Equivalent Equations

Graph structure

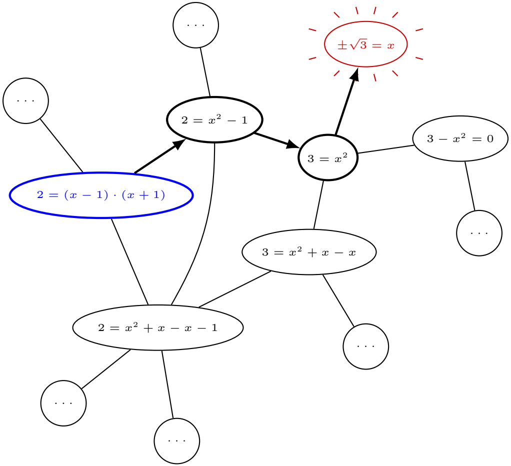
- vertex = equation
- edge for each rewrite rule between eqn_1 and eqn_2
- infinite graph



Equivalent Equations

Graph structure

- vertex = equation
- edge for each rewrite rule between eqn_1 and eqn_2
- infinite graph
- solving = graph search



Differentiate

Differentiating Operator Records

Consider $w = f(z)$, where $w, z \in \mathbb{C}$. Decomposing with $z = x + iy$, $w = u + iv$ leads to

$$\begin{aligned} \dot{u} &= \partial_x u \cdot \dot{x} + \partial_y u \cdot \dot{y} \\ \dot{v} &= \partial_x v \cdot \dot{x} + \partial_y v \cdot \dot{y} \end{aligned} \tag{*}$$

Differentiating Operator Records

Consider $w = f(z)$, where $w, z \in \mathbb{C}$. Decomposing with $z = x + iy$, $w = u + iv$ leads to

$$\begin{aligned} \dot{u} &= \partial_x u \cdot \dot{x} + \partial_y u \cdot \dot{y} \\ \dot{v} &= \partial_x v \cdot \dot{x} + \partial_y v \cdot \dot{y} \end{aligned} \tag{*}$$

We would like to write

$$\dot{w} = \partial_z w \cdot \dot{z}$$

Differentiating Operator Records

So suppose $\partial_z w = J = J_1 + iJ_2$ for some $J_1, J_2 \in \mathbb{R}$. Then

$$\begin{aligned}\partial_z w \cdot \dot{z} &= (J_1 + iJ_2) \cdot (\dot{x} + i\dot{y}) \\ &= (J_1 \cdot \dot{x} - J_2 \cdot \dot{y}) + i(J_2 \cdot \dot{x} + J_1 \cdot \dot{y})\end{aligned}$$

Together with (*) this results in

$$\begin{aligned}\dot{u} &= \partial_x u \cdot \dot{x} + \partial_y u \cdot \dot{y} = J_1 \cdot \dot{x} - J_2 \cdot \dot{y} \\ \dot{v} &= \partial_x v \cdot \dot{x} + \partial_y v \cdot \dot{y} = J_2 \cdot \dot{x} + J_1 \cdot \dot{y}\end{aligned} \tag{**}$$

Differentiating Operator Records

Demanding that (**) holds for any $\dot{z}$, we finally arrive at the Cauchy-Riemann equations

$$\begin{aligned} J_1 &= \partial_x u = \partial_y v \\ J_2 &= \partial_x v = -\partial_y u \end{aligned}$$

And so $J = \partial_x w$, which is the same as symbolically taking $J = \partial_z w$ and treating z as a real variable. $\square$

Differentiating Operator Records

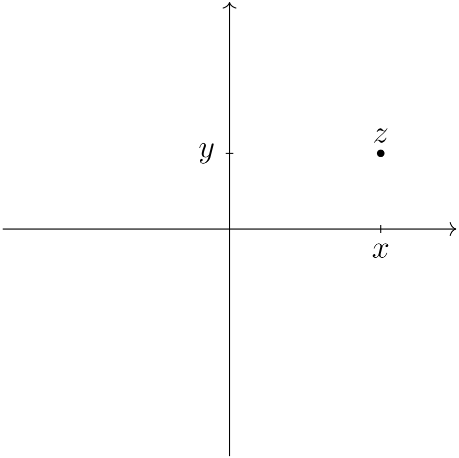
Summary

- multiplication needs to be defined
- Ansatz: there is a J s.t. $\dot{w} = J \cdot \dot{z}$
- generalized Cauchy-Riemann conditions on f
- results in possibility to use chain rule

More Examples

Split-Complex Numbers

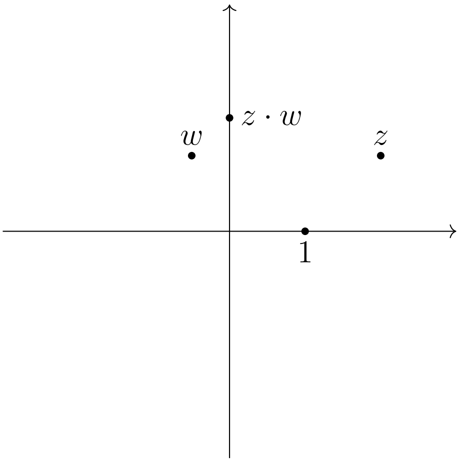
$$j^2 = 1 \quad (j \notin \mathbb{R})$$
$$z = x + jy$$



Split-Complex Numbers

$$j^2 = 1 \quad (j \notin \mathbb{R})$$
$$z = x + jy$$

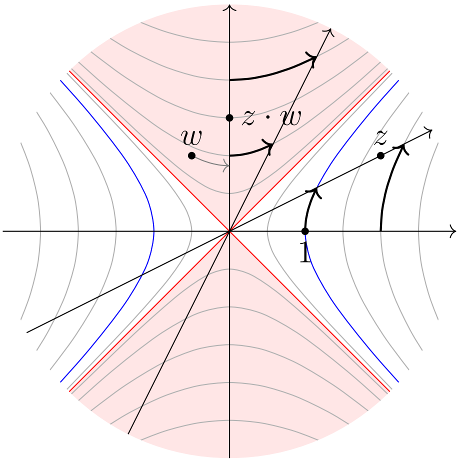
$$(x_1 + jy_1) \cdot (x_2 + jy_2) =$$
$$(x_1x_2 + y_1y_2) + j(x_1y_2 + y_1x_2)$$



Split-Complex Numbers

$$j^2 = 1 \quad (j \notin \mathbb{R})$$
$$z = x + jy$$

$$(x_1 + jy_1) \cdot (x_2 + jy_2) =$$
$$(x_1x_2 + y_1y_2) + j(x_1y_2 + y_1x_2)$$

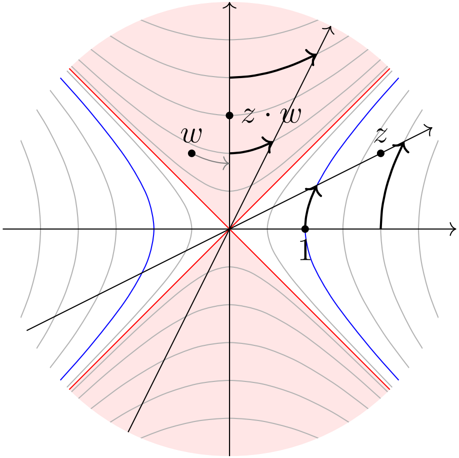


Split-Complex Numbers

$$j^2 = 1 \quad (j \notin \mathbb{R})$$
$$z = x + jy$$

$$(x_1 + jy_1) \cdot (x_2 + jy_2) =$$
$$(x_1x_2 + y_1y_2) + j(x_1y_2 + y_1x_2)$$

- not a field, unlike $\mathbb{C}$
- zero-divisors $(1 + j) \cdot (1 - j) = 0$
- Minkowski space, Lorenz boost
- online dating adjacency matrices



Quaternions

$$i^2 = j^2 = k^2 = ijk = -1$$

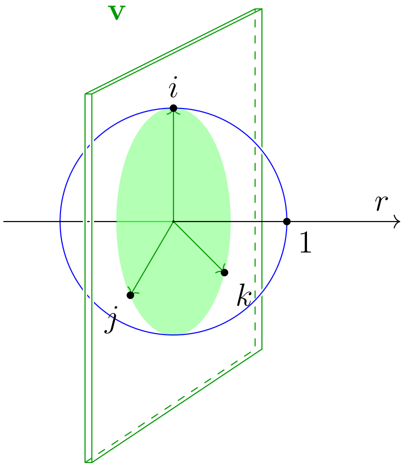
$$\begin{aligned} q &= a + bi + cj + dk \\ &= (r, \mathbf{v}) \end{aligned}$$

Quaternions

$$i^2 = j^2 = k^2 = ijk = -1$$

$$q = a + bi + cj + dk$$
$$= (r, \mathbf{v})$$

$$(r_1, \mathbf{v}_1) \cdot (r_2, \mathbf{v}_2)$$
$$= (r_1 r_2 - \mathbf{v}_1 \cdot \mathbf{v}_2,$$
$$r_1 \mathbf{v}_2 + r_2 \mathbf{v}_1 + \mathbf{v}_1 \times \mathbf{v}_2)$$



Quaternions

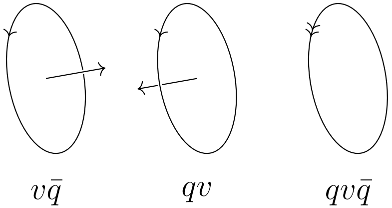
$$i^2 = j^2 = k^2 = ijk = -1$$

$$\begin{aligned} q &= a + bi + cj + dk \\ &= (r, \mathbf{v}) \end{aligned}$$

For $q = (\cos(\theta), \sin(\theta) \cdot \hat{\mathbf{n}})$ and $v = (0, \mathbf{v})$ the product

$$q \cdot v \cdot \bar{q}$$

is a rotation of $\mathbf{v}$ about $\hat{\mathbf{n}}$ by 2θ.



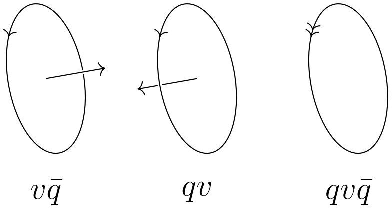
Quaternions

$$i^2 = j^2 = k^2 = ijk = -1$$

$$\begin{aligned} q &= a + bi + cj + dk \\ &= (r, \mathbf{v}) \end{aligned}$$

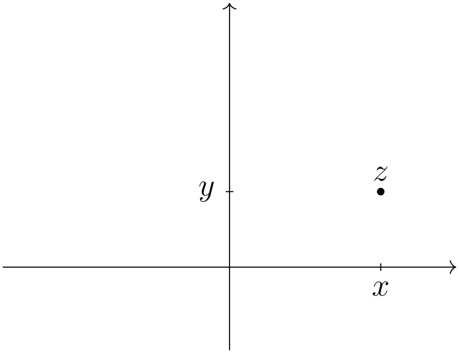
- rotations in 3D space
- not commutative:

$$\frac{\mathrm{d}q^2}{\mathrm{d}t} = q \cdot \frac{\mathrm{d}q}{\mathrm{d}t} + \frac{\mathrm{d}q}{\mathrm{d}t} \cdot q$$



Dual Numbers

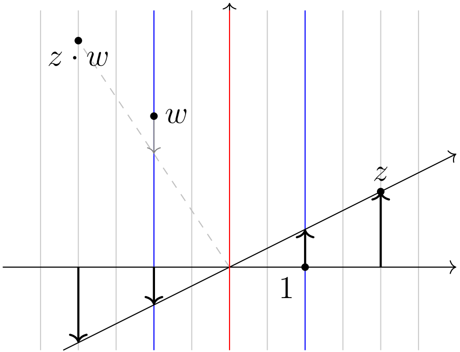
$$\varepsilon^2 = 0 \quad (\varepsilon \neq 0)$$
$$z = x + \varepsilon y$$



Dual Numbers

$$\varepsilon^2 = 0 \quad (\varepsilon \neq 0)$$
$$z = x + \varepsilon y$$

$$(x_1 + \varepsilon y_1) \cdot (x_2 + \varepsilon y_2) =$$
$$(x_1 x_2) + \varepsilon(x_1 y_2 + y_1 x_2)$$

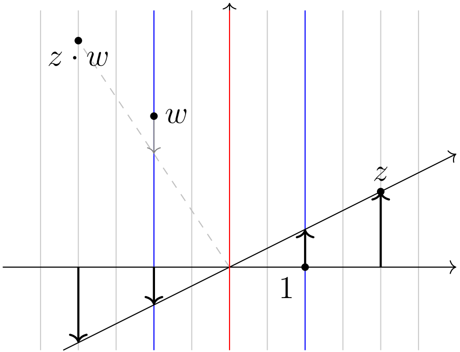


Dual Numbers

$$\varepsilon^2 = 0 \quad (\varepsilon \neq 0)$$
$$z = x + \varepsilon y$$

$$(x_1 + \varepsilon y_1) \cdot (x_2 + \varepsilon y_2) =$$
$$(x_1 x_2) + \varepsilon(x_1 y_2 + y_1 x_2)$$

- Galilean transform/shear mapping
- kinematic synthesis
- automatic differentiation
  $f(x + \varepsilon y) = f(x) + \varepsilon y f'(x)$



Open Questions

- How do we tell the compiler what rules to apply without too much hard coding?
- Should we modify the language? Are annotations enough?

- How do we tell the compiler what rules to apply without too much hard coding?
- Should we modify the language? Are annotations enough?
- Who uses operator records and for what?
- What are current problems that should be fixed?

- How do we tell the compiler what rules to apply without too much hard coding?
- Should we modify the language? Are annotations enough?
- Who uses operator records and for what?
- What are current problems that should be fixed?
- What are the difficult steps in the backend?
- What else can be optimized?

Contact

OpenModelica

https://openmodelica.org/

github.com/OpenModelica

or create a <trac ticket>

Prof. Dr. phil., Dipl.-Math.
Bernhard Bachmann

bbachmann@fh-bielefeld.de

Philip Hannebohm

phannebohm@fh-bielefeld.de

FH Bielefeld
University of
Applied Sciences