`solvelib` — data types and utilities for the solver

Table of contents

The library `solvelib` contains various methods used by the function `solve`. Most of them are only for internal use.

`solvelib::BasicSet` – the basic infinite sets

The domain `solvelib::BasicSet` comprises the four sets of integers, reals, rationals, and complex numbers, respectively.

`Z_`, or equivalently `solvelib::BasicSet(Dom::Integer)`, represents the set of integers.

`Q_`, or equivalently `solvelib::BasicSet(Dom::Rational)`, represents the set of rational numbers.

`R_`, or equivalently `solvelib::BasicSet(Dom::Real)`, represents the set of real numbers.

`C_`, or equivalently `solvelib::BasicSet(Dom::Complex)`, represents the set of complex numbers.

Creating Elements:

- `solvelib::BasicSet(Dom::Integer)`
- `solvelib::BasicSet(Dom::Rational)`
- `solvelib::BasicSet(Dom::Real)`
- `solvelib::BasicSet(Dom::Complex)`

  `Z_`
  `Q_`
  `R_`
  `C_`

Categories:

`Cat::Set`

Related Domains: `Dom::Interval`

Details:

- The four basic sets are assigned to the identifiers `Z_`, `Q_`, `R_`, and `C_` during system initialization.

Mathematical Methods

Method `contains`: tests whether some object is a member

`contains(`*`arithmetical expression`* `a,` *`dom`* `S)`

- tries to decide whether $a \in S$, and returns `TRUE`, `FALSE`, or `UNKNOWN`.
- Equivalently, `is(a in S)` may be used.

Conversion Methods

Method `convert`: converts a domain into a basic set

`convert(`*`any`* `d)`

- This method converts `d` into a basic set; it returns `FAIL` unless `d` is one of the four domains `Dom::Integer`, `Dom::Rational`, `Dom::Real`, and `Dom::Complex`.

Method `set2prop`: converts a set to a property

`set2prop(`*`dom`* `S)`

- This method returns the type of the `Type` library equivalent to `S`.

Example 1. The domain of basic sets know about the basic arithmetical and set–theoretic functions.

```
>> J:=Dom::Interval(3/2, 21/4):
   Z_ intersect J

                                   {2, 3, 4, 5}
```

Super-Domain: `Dom::BaseDomain`

Axioms

`Ax::canonicalRep`

Changes:

- `solvelib::BasicSet` is a new function.

`solvelib::conditionalSort` – possible sortings of a list depending on parameters

`solvelib::conditionalSort(l)` sorts the list `l` in ascending order. Unlike for `sort`, only the usual order on the real numbers and not the internal order (see `sysorder`) is used. `solvelib::conditionalSort` does a case analysis if list elements contain indeterminates.

Call(s):

- `solvelib::conditionalSort(l)`

Parameters:

`l` — list of arithmetical expressions

Return Value: A list if the sorting is the same for all possible parameter values; or an object of type `piecewise` if some case analysis is necessary.

Side Effects: `solvelib::conditionalSort` takes into account the assumptions on all occurring identifiers.

Related Functions: `sort`, `piecewise`

Details:

- `solvelib::conditionalSort` invokes the inequality solver to get simple conditions in the case analysis. The ability of `solvelib::conditionalSort` to recognize sortings as impossible is thus limited by the ability of the inequality solver to recognize an inequality as unsolvable. See Example 2.
- Only expressions representing real numbers can be sorted. It is an error if non–real numbers occur in the list; it is implicitly assumed that all parameters take on only such values that cause all list elements to be real.
- Sorting is unstable, i.e. equal elements may be placed in any order in the resulting list; these cases may be listed separately in the case analysis.
- The usual simplifications for piecewise defined objects are applied, e.g., equalities that can be derived from a condition are applied (by substitution) to the list.

Example 1. In the simplest case, sorting a two-element list `[a,b]` just amounts to solving the inequation `a<=b` w.r.t. all occuring parameters.

```
>> solvelib::conditionalSort([x,x^2])

               2
 piecewise([x , x] if 0 <= x and x <= 1,

              2
    [x, x ] if (x < 0 or 1 < x))
```

Example 2. Sometimes cases are not recognized as impossible.

```
>> assume(x>5): solvelib::conditionalSort([x,exp(x)])

 piecewise([exp(x), x] if - x + exp(x) <= 0,

    [x, exp(x)] if x - exp(x) < 0)
```

Background:

- The complexity of sorting a list of n elements is up to $n!$.

Changes:

- `solvelib::conditionalSort` is a new function.

`solvelib::getElement` – get one element of a set

`solvelib::getElement(S)` returns an element of `S`.

Call(s):

- `solvelib::getElement(S)`

Parameters:

`S` — any set

Return Value: `solvelib::getElement` returns a MuPAD object representing an object of `S`, or `FAIL` if no element could be determined.

Overloadable by: `S`

Related Functions: `solve`

Details:

- `solvelib::getElement` may return `FAIL` either if `S` is the empty set, or if no element of the set could be computed because the solver is not strong enough, or if the answer depends on the value of some parameter.

Example 1. If `S` is a finite set, just one of its elements is returned.

```
>> solvelib::getElement({2, 7, a})

                                a
```

Example 2. For image sets, an element is obtained by replacing every parameter by a constant.

```
>> S:=Dom::ImageSet(k*PI, k, solvelib::BasicSet(Dom::Integer))

                      { X1*PI |  X1 in Z_ }

>> solvelib::getElement(S)

                                0
```

Example 3. `solvelib::getElement` may fail to find an element of a set although that set is not empty.

```
>> solvelib::getElement(solve(exp(x)+x=0,x))

                               FAIL
```

Changes:

- `solvelib::getElement` is a new function.

`solvelib::isFinite` – test whether a set is finite

`solvelib::isFinite(S)` returns `TRUE`, `FALSE`, or `UNKNOWN` depending on whether `S` is finite, infinite, or the question could not be settled.

Call(s):

- `solvelib::isFinite(S)`

Parameters:

`S` — any set

Return Value: Boolean value

Overloadable by: `S`

Related Functions: `solve`

Example 1. A `DOM_SET` is always finite.

```
>> solvelib::isFinite({2,5})

                              TRUE
```

Example 2. The set of integers is infinite.

```
>> solvelib::isFinite(Z_)

                              FALSE
```

Changes:

- `solvelib::isFinite` is a new function.

`solvelib::pdioe` – solve polynomial Diophantine equations

`solvelib::pdioe(a, b, c)` returns polynomials `u` and v that satisfy the equation `au + bv = c`.

`solvelib::pdioe(aexpr, bexpr, cexpr, x)` does the same after converting the arguments into univariate polynomials `a, b, c` in the variable `x`.

Call(s):

- `solvelib::pdioe(a,b,c)`
- `solvelib::pdioe(aexpr, bexpr, cexpr, x)`

Parameters:

`x` — identifier or indexed identifier
`a, b, c` — univariate polynomials
`aexpr, bexpr, cexpr` — polynomial expressions

Return Value: If the equation is solvable, `solvelib::pdioe` returns an expression sequence consisting of two operands of the same type as the input (expressions or polynomials). If the equation has no solution, `solvelib::pdioe` returns `FAIL`.

Related Functions: `solve`

Details:

- The coefficient ring of the polynomials `a`, `b`, and `c` must be either `Expr`, or `IntMod(p)` for some prime p, or a domain belonging to the category `Cat::Field`.

Example 1. If expressions are passed as arguments, a fourth argument must be provided:

```
>> solvelib::pdioe(x, 13*x + 22*x^2 + 18*x^3 + 7*x^4 + x^5 + 3, x^2 + 1, x)

                   3    4
  19 x      2   7 x    x
- ---- - 6 x  - ---- - -- - 13/3, 1/3
   3             3     3
```

Example 2. x is not a multiple of the gcd of $x+1$ and x^2-1. Hence the equation $u(x+1)+v(x^2-1)=x$ has no solution for u and v:

```
>> solvelib::pdioe(x + 1, x^2 - 1, x, x)

                         FAIL
```

Example 3. If the arguments are polynomials, the fourth argument may be omitted:

```
>> solvelib::pdioe(poly(a + 1, [a]), poly(a^2 + 1, [a]), poly(a -
1, [a]))

            poly(a, [a]), poly(-1, [a])
```

Changes:

- `solvelib::pdioe` used to be `pdioe`.

`solvelib::preImage` – preimage of a set under a mapping

`solvelib::preImage(a, x, S)` returns the set of all numbers `y` such that substituting `y` for `x` in `a` gives an element of `S`.

Call(s):

- `solvelib::preImage(a, x, S)`

Parameters:

`a` — arithmetic expression
`x` — identifier
`S` — set

Return Value: set

Related Functions: `solve`

Details:

- `S` can be a set of any type (finite or infinite).

Example 1. In case of a finite set `S`, the preimage of S is just the union of all sets `solve(a=s, x)`, where `s` ranges over the elements of `S`.

```
>> solvelib::preImage(x^2+2, x, {11, 15});

                           1/2      1/2
              {-3, 3, 13   , - 13   }
```

Example 2. For intervals, the preimage is usually an interval or a union of intervals.

```
>> solvelib::preImage(x^2+2, x, Dom::Interval(3..7));

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

Changes:

- `solvelib::preImage` is a new function.

`solvelib::Union` – **union of a system of sets**

`solvelib::Union(set, param, paramset)` returns the set of all objects that can be obtained by replacing, in some element of `set`, the free parameter `param` by an element of `paramset`.

Call(s):

- `solvelib::Union(set, param, paramset)`

Parameters:

`set` — set of any type
`param` — identifier
`paramset` — set of any type

Return Value: `solvelib::Union` returns a set of any type; see `solve` for an overview of the different types of sets. It may also return the unevaluated call if the union could not be computed.

Overloadable by: `set`

Related Functions: `solve`

Details:

- `set` may be a set of any type; it need not depend on the parameter `param`, and it may also contain other free parameters. However, it must not use `param` as a bound parameter, e.g. `Dom::ImageSet(sin(param), param, S)` (for some set `S`) .
- `paramset` may be a set of any type and may depend on some free parameters. See example 1.
- If `paramset` is empty, the result is the empty set. Overloading has no effect in this case.

Example 1. We compute the set of all numbers that are equal to $k+1$ or $k+3$ for $k=2$, $k=4$, or $k=l$, where l is a free parameter.

```
>> solvelib::Union({k+1, k+3}, k, {2,4,l});

                          {3, 5, 7, l + 1, l + 3}
```

Changes:

- `solvelib::Union` is a new function.