Solvers: Completing Solveset

Table of Contents:

About Me

Basic Information

Name: Yathartha Anirudh Joshi
Email: yathartha32@gmail.com, yathartha_joshi@yahoo.in
University: B.T.K.I.T, Dwarahat
Github/IRC: Yathartha22
Blog: yathartha22.github.io, yaj22.wordpress.com
Time Zone: IST (UTC +5:30)

Personal Background

I am final year undergraduate pursuing Bachelor of Technology in Computer Science and Engineering at B.T.K.I.T, Dwarahat, Uttarakhand, India. I love Mathematics and the logic that resides in it. I enjoy problem solving and always try to get the best and efficient solution to the problem.

Programming Details

I work on Ubuntu 16.04 operating system with *Sublime Text 3* as my primary editor. It's been about more than four year since I started competitive coding. I began coding with C++ and C. I like these languages because of their simplicity and also because I had C++ in my intermediate. Implementation of what we think becomes an easy task in C/C++. I am comfortable with STL and algorithms. I have also coded in Java and soon I switched over to Python. The reason I liked and chose Python was because of its flexibility and simplicity.

I am also familiar with the web development stuff: HTML, CSS, MySQL, PHP, Django, JavaScript and more. I qualified Google Code Jam qualifiers and also participated in IEEE Xtreme 10.0 (secured AIR 22) and IEEE Xtreme 11.0 (secured AIR 14). I have a deep interest in programming and contributing to open source world.

Contributions to Sympy

Well I started contributing to *Sympy* about a year back and in this period of time I have learned a lot of things about Sympy and I have known how powerful this Computer Algebra System is. I started of introducing myself to the Gitter channel and I made my doubts cleared about how do I make my first contribution. All the members are active, I used to get the response pretty soon. Gradually I got myself comfortable with Sympy's development workflow. Even now at any point I have a doubt, I throw it in Gitter or google groups and make it clear by discussions.

The most interesting feature of Sympy that I found was the solvers module. It is a module that is capable of solving all types of equations, making it one of the powerful features. To make it more powerful and efficient I decided to work on this particular module. I mainly focused on its newest module solveset.

PR's (merged/unmerged)

- **(merged)** PR#14427 removed unnecessary simplification in solveset.
- **(merged)** PR#14341 rewriting min/max to piecewise only when arguments are real.
- **(closed)** PR#14232 improved solveset for improper symbols.
- **(merged)** PR#14019 improved binomial for negative and non integer k.
- **(merged)** PR#13923 added `integer_log` for negative base exponents.
- **(merged)** PR#13844 improvement in solveset for negative base exponents
- **(merged)** PR#13593 minor change in the error message.
- **(merged)** PR#13518 removed `x**1.0 == x` hack.
- **(merged)** PR#13517 `disambiguate` function added.
- **(merged)** PR#13309 rewriting of min/max in Piecewise.
- **(merged)** PR#13277 `total_degree` function implemented.
- **(merged)** PR#13016 improving `invert_real` for `0**x + a` type equations.
- **(merged)** PR#12394 more examples in FiniteSet showing its functionality.
- **(merged)** PR#12311 minor typo-fix.
- **(closed)** PR#12245 symbolic function implementation for solveset.
- **(merged)** PR#12240 removed errors in doc example and minor documentation fixes.
- **(closed)** PR#12225 adding few functions in `__init__.py`.
- **(closed)** PR#12224 raising ValueError if symbol is constant.
- **(merged)** PR#12147 improved `check_assumption` and implemented a new function `failing_assumptions` as an advancement for `check_assumption`
- **(merged)** PR#12141 Documentation improvements.
- **(open)** PR#13135 making decompogen capable for handling exponential equations.
- **(open)** PR#13088 incorporating `solve_decomposition` in solveset to solve trigonometric, exponential equations and others.
- **(open)** PR#13045 Implementing `transolve` for solving transcendental equaitons in solveset.
- **(open)** PR#12542 `solve` missing solutions for `x**(1/x) - 1`.

Apart from these I have been involved in discussions and reviewing of few PR's, namely: #14449, #14253, #13908.

Issues/Bugs

- #14533 solveset needs to handle exception for some trigonometric equations

- #12538 solve returns emptylist for `x**(1/x) - 1`.
- #12340 Discussion for improving `_solve_trig`.
- #12344 Discussion for implementing transolve.
- #13121 `solve_decomposition` misses some equations.
- #12217 solveset does not solves for number.
- #12146 improvement for `check_assumption`.

The Project

Brief Overview

Solving any type of equations is one of the most essential and powerful feature of any Computer Algebra System. For Sympy it is **Solvers**. The solve module in solvers is used for solving algebraic and transcendental equations, but it had a lot of limitations (see here for reference)and therefore solveset came into existence to overcome all its difficulties and limitations.

Solveset is being under development since 2014 by **Harsh Gupta** and continued by **Amit Kumar.** In 2016 **Shekharrajak** and **Kshitij Saragoi** implemented a lot of functionalities, but still there is a lot of work to be done.

Following would be my major areas of concern during the GSoC period in which I will try to complete the solveset.

- **Transcendental Equation Solver (Implementing transolve).**

  Currently solveset does not support transcendental equations solving. This is one of the major part of the equation solver, after which solveset can be said as complete. Solve uses _tsolve and bivariate.py, there are few problems in its implementation like:
  - The codebase is quite messy and there are a lot of recursive calls which makes computation time taking.
  - _tsolve does not return solutions in complex domain.

  These things needs to be taken care of to make transolve efficient and scalable.

- **Integrating helper solvers with solveset.**

There are a lot of helpers in solveset that are not integrated with it (they can be used individually), so one of the major task will be to integrate all helpers so as to increase the user interface. Helpers include:

- linsolve: solving linear system of equations.
- nonlinsolve: solving nonlinear system of equations.
- solve_decomposition: Solves a varied class of equations using the concept of Rewriting and Decomposition
- Helper for solving modular equations (_invert_modular).
- The idea for transolve is to make it modular, so different helpers will be implemented to solve transcendental equations, like: _solve_log (for logarithmic), _lambert (for solving lambert equations), improving _invert_real and _invert_complex for exponential equations.

- **Building and improving the set infrastructure**

  This task basically includes unification of ImageSets. Currently there is no algorithm implemented for the unification of imageset, this sometimes causes problem when we try solving equations having infinite solutions for complex equations a union of many imageset is returned (which is not good), so a unification needs to be done to give a compact result.

  Also, another thing in sets that can be implemented is BigUnion and BigIntersection and also IndexSet (to represent BigUnion/BigIntersection).

- **Improving nonlinsolve for trigonometric and Lambert W functions.**

  nonlinsolve was implemented for solving nonlinear type equations but still a bit of some work needs to be done for solving trigonometric and LambertW to make it more powerful.

- **Miscellaneous functionalities for completing solveset.**

  Apart from the above mentioned there are some small but important workarounds that needs to be done so that solveset completely replaces solve. These includes

  - Solving modular equations.
  - Improving Hyperbolic equations solving

The Plan

My intentions during the period will be completing solveset and closing #10006.

I have also started wiki page, where I have discussed in detail on how implementation will take place and how will it be more powerful. Also started discussion here in google groups for implementing the project.

1) **Implementing transcendental equations (transolve)**

   Transcendental Function: It is an analytic function that does not satisfy polynomial equation in contrast to algebraic equation.
   Examples: logarithmic equations, trigonometric equations, exponential equations.

   Currently solveset does not support such equations, it returns ConditionSet instead. transolve needs to be implemented to solve such equations, initial commit for transolve has been made in #13045 which is capable for solving a large variety of exponential and logarithmic equations. Still a lot of work needs to be done:

   - Equations solvable by Lambert W function.
   - Bivariate solver.
   - Improving trigonometric equation solving.
   - Improving solving logarithmic equations.
   - Improving solving exponential equations.

   a) **Equations solvable by Lambert W function**

      solve is able to handle such equations but solveset returns a ConditionSet. solve makes it possible by taking advantage of the bivariate module.

```
>>> f = x + exp(x)
>>> solve(f, x)
LambertW(1)
>>> solveset(f, x, S.Reals)
ConditionSet(x, Eq(x + exp(x), 0), S.Reals)
```

Brief overview of Lambert W function

The **Lambert function** or the **omega function** is the inverse of

$f(z) = xe^x$
Solving,

$f(z) = y$, where $y = xe^x$
$z = f^{-1}(y)$
$z = W(y)$, where W is the Lambert function

In general and simpler terms any equations that can be written in the form

$A + B*x + C*log(D + E*x) = 0$ has solution in Lambert function.

The solution write is:

$x = -D/E + C/B*W(t)$

where $t = B/C*E*exp((BD - AE)/CE)$

b) Logarithmic Equations

solveset does not handle this variant of transcendental equations, this is mainly because there is no conversion of such equations in more simplified form (using logarithmic formulas). This has been implemented in #13045

```
>>> f = log(x-3) + log(x-2) - log(2*x + 24)
>>> solveset(f, x, S.Reals)
ConditionSet(x, Eq(log(x - 3) + log(x - 2) - log(2*x +
24), 0), S.Reals)
```

c) Exponential equations

Again such equations are not handled properly by solveset. Currently only inversion of exponents is done by _invert, but these does not return a simplified result. Also exponents in complex domain should be

implemented. Implementation has been done in #13045, where majority of exponent equations are taken care.

```
# currently
>>> solveset(5**(x-3) - 3**(2*x + 1), x, S.Reals)
ConditionSet(x, Eq(-3**(2*x + 1) + 5**(x - 3), 0),
S.Reals)

# in #13045
>>> solveset(5**(x-3) - 3**(2*x + 1), x, S.Reals)
FiniteSet(-log(375)/(-log(5) + 2*log(3)))

# should be simplified to FiniteSet(5)
>>> solveset(2**x - 32, x, S.Reals)
FiniteSet(log(32)/log(2))

# not supported in complex domain
>>> solveset(2**x - 32, x)
ConditionSet(x, Eq(2**x - 32, 0), Complexes(S.Reals x
S.Reals, False))
# correct result
Imageset(Lambda(n, (I*(2*n*pi + pi) + 5*log(2))/log(2)),
S.Integers)
```

d) Improving Trigonometric Equation solving

Currently solveset is unable to solve a lot of trigonometric equations. Nested trigonometric equations returns ConditionSet. issue#10217. Discussion related to these equations: #12340

solve_decomposition was written in solveset as an helper for solving equations using the principle decomposition and rewriting. It has been noticed that a lot of trigonometric equations can be solved using it. This needs to be added to make transolve more powerful.

```
# in current master
>>> solveset(sin(x + 2), x, S.Reals)
EmptySet()
>>> solve_decomposition(sin(x+2),x , S.Reals)
```

```
{2⋅n⋅π  - 2 | n ∈ ℤ} ∪ {2⋅n⋅π  - 2 + π  | n ∈ ℤ}
```

2) Improving Set Infrastructure

This task mainly deals in unifying the ImageSet. Currently there is no union of imageset which sometimes becomes complication for complex equations. In solveset the solutions for trigonometric equations are affected, a proper unification needs to be carried out so that we get accurate result.

Few works and discussions has been done regarding this

#7673, #10898, #10482, #12011, #11188.

```
# currently
>>> solveset(sin(x), x, S.Reals)
{2⋅n⋅π | n ∈ ℤ} ∪ {2⋅n⋅π + π | n ∈ ℤ}
# this could be simplified to {n⋅π | n ∈ ℤ}

>>> solveset(sin(2*x) - sin(4*x) + sin(6*x) , x, S.Reals)
Union(ImageSet(Lambda(_n, 2*_n*pi), S.Integers),
ImageSet(Lambda(_n, 2*_n*pi + pi), S.Integers),
ImageSet(Lambda(_n, 2*_n*pi + pi/2), S.Integers),
ImageSet(Lambda(_n, 2*_n*pi + 3*pi/2), S.Integers),
ImageSet(Lambda(_n, 2*_n*pi + 4*pi/3), S.Integers),
ImageSet(Lambda(_n, 2*_n*pi + 2*pi/3), S.Integers),
ImageSet(Lambda(_n, 2*_n*pi + 5*pi/3), S.Integers),
ImageSet(Lambda(_n, 2*_n*pi + pi/3), S.Integers),
ImageSet(Lambda(_n, 2*_n*pi + 5*pi/4), S.Integers),
ImageSet(Lambda(_n, 2*_n*pi + 3*pi/4), S.Integers),
ImageSet(Lambda(_n, 2*_n*pi + 7*pi/4), S.Integers),
ImageSet(Lambda(_n, 2*_n*pi + pi/4), S.Integers))

# could have been simplified to (n⋅π / 4), (n⋅π ± π/6)
```

Also to make set module more powerful, implementation of BigUnion and BigIntersection was proposed in #9815 to represent arbitrary index unions and intersection. For this implementation we will be needing to implement a new set, called the IndexSet, which will work something as:

```
>>> X = IndexSet(FiniteSet(1, 2, 3), FiniteSet(3, 4),
FiniteSet(6, 7)); X
IndexSet(0, 1, 2)
>>> X[0]
FiniteSet(1, 2, 3)
>>>X[1]
FiniteSet(3, 4)
>>> X[2]
FiniteSet(6, 7)

# with the above implementation indexing is done implicitly we
can have a scenario of having indices passed explicitly.

>>> BigUnion(X, i, FiniteSet(0, 1, 2))
FiniteSet(1, 2, 3, 4, 5, 6, 7)
```

Discussion: https://groups.google.com/forum/#!topic/sympy/sLT7goRueSM
As of now there is no such conclusion on how we will be implementing BigUnion and BigIntersection, therefore I have started an issue-cum-discussion for discussion its API and implementing it in a better way.

3) Miscellaneous but necessary implementations

a) Solving Modular equations

Modular equations are the ones that follows modulus arithmetic and modulus operations. Currently solves is unable to get the result of such equations. Implementing the capability of solveset solving modular equations would make solveset a versatile solver.

Example:

```
>>> solveset(2 - Mod(4*x, 5), x, S.Integers)
ConditionSet(x, Eq(Mod(4*x, 5) - 2, 0), S.Integers)
# correct result should be: ImageSet(Lambda(5*_n + 3),
S.Integers)
```

b) Solving Hyperbolic Equations

Reason for failure of Hyperbolic equations by _solve_trig

Currently Hyperbolic equations are not handled properly by _solve_trig, it tries to rewrite as exp but when comes to substituting exp, it replaces exp(I*symbol) with dummy, and since hyperbolic does not contains such terms it remains as it is and the next condition checks for symbol in the equation where it finds and hence returns a conditionset.

```
>>> solveset(sinh(x), x)
ConditionSet(x, Eq((exp(2*x) - 1)*exp(-x)/2, 0), S.Reals)
# correct result:
Union(ImageSet(Lambda(_n, 2*_n*I*pi), S.Integers),
ImageSet(Lambda(_n, I*(2*_n*pi + pi)), S.Integers))
```

4) Improving nonlinsolve for trigonometric and Lambert W system of equations

Currently trigonometric system of nonlinear equations is not handled by nonlinsolve, for example, consider a simple non-linear trigonometric equations:

```
>>> eqns = [sin(x) + cos(y) - 1, sin(x)**2 + cos(y)**2 - 0.5]
>>> nonlinsolve(eqns, [x, y])
ConditionSet({x, y}, {sin(x) + cos(y) - 1, sin(x)**2 +
cos(y)**2 - 0.5}, Complexes(S.Reals x S.Reals, False))
```

For such equations one way of solving is to replace sin and cos with temporary variables and then solve as a system of nonlinear equations. After solving replace back and solve it using solveset to get the solutions of x and y.

Execution

Whatever be the issues, they are not solved until properly implemented. Therefore to get better out of what I have proposed I divide my work into three phases.

Phase I

This phase includes **implementation of transolve.** This will be one of the major tasks

during GSoC period as this will make solveset complete.

For its better implementation I have created a wiki page which comprises of how solve solves such equations and what are its major disadvantages which needs to overcome and make transolve more modular, extensible and less messy. Also I have tried to put in ideas of how transolve will be implemented.

Implementing exponential solver

There are three types of exponential equations that needs to be taken care of **First equations**t are the ones that are currently solvable (by _invert) but does not return a simplified result.

```
>>> f =  2**x - 32
>>> solveset(f, x, S.Reals)
{log(32)/log(2)}    correct result: {5}
```

Possible fix: This can be handled by returning a simplified solution using `simplify` whenever the final result is returned.

Second equations are the ones that cannot be solved in complex domain. Taking the above example:

```
# currently
>>> solveset(f, x)
ConditionSet(x, Eq(2**x - 32, 0), Complexes(S.Reals x S.Reals,
False))

# correct result: Imageset(Lambda(n, (I*(2*n*pi + arg(y - 32))
+ log(Abs(y - 32)))/log(2)), S.Integers)
```

Possible fix: In complex domain we need to add a specific conditions for such cases something like below:

```
if f.is_Pow:
        base, expo = f.args
        base_has_symbol = base.has(symbol)
        expo_has_symbol = expo.has(symbol)
```

```
        if expo_has_symbol:
            pow_inv = Union(*[ imageset(Lambda(n, (I*(2*n*pi +
arg(g_y)) + expand_log(log(Abs(g_y))))/expand_log(log(base))),
S.Integers) for g_y in g_ys if g_y !=0])
            return _invert_complex(expo, pow_inv, symbol)
```

Third types are the ones like 4**(x-2) - 5**x that are not supported currently and solveset returns ConditionSet

```
>>> solveset(4**(x - 2) - 5**x, x, S.Reals)
ConditionSet(x, Eq(4**(x - 2) - 5**x, 0), S.Reals)
```

Possible fix: For these general type of equations a*f(x) + b*g(x), this type of equations can be reduced to f(x)/g(x) = b/a, and then checking if bases are same or not and then solving it.

```
if not g and not g_ys.args[0] and len(h.args) == 2 and \
                not h.is_polynomial(symbol):
            a, b = ordered(h.args)
            ai, ad = a.as_independent(symbol)
            bi, bd = b.as_independent(symbol)

            if any(_ispow(i) for i in (ad,bd)):
                a_base, a_exp = ad.as_base_exp()
                b_base, b_exp = bd.as_base_exp()

                if a_base == b_base:
                    h = powsimp(powdenest(ad/bd))
                    g = -bi/ai

                else:
                    rat = ad/bd
                    _h = powsimp(rat)
                    if _h != rat:
                        h = _h
                        g = -bi/ai

            return (h, FiniteSet(g))
```

Implementing logarithmic solver

Currently logarithmic equations are not solvable because no proper simplification of logarithmic equations using log formulas are done, so we need to use logarithmic formula to reduce equations to a simpler form upto which it can be solved using _solveset.

Possible fix: In the transolve method we need to add a check for logarithmic equations probably can be done using a helper (_check_log) which would return True if it is logarithmic otherwise False. If True, we can call another helper (_solve_log) which would return the solution.

```
def _check_log(f):
"""Helper function to check if the equation is logarithmic or
not."""
    orig_f = f
    g = logcombine(f, force=True)
    if g.count(log) != f.count(log):
        return g
    return False
```

```
def _solve_log(f, symbol, domain):
    """Helper function to solve logarithmic equations."""

    from sympy.solvers.solvers import checksol

    result = S.EmptySet
    orig_f = f
    f = _check_log(orig_f)
    solutions = _solveset(f, symbol, domain)

    if isinstance(solutions, FiniteSet):
        for solution in solutions:
            if checksol(orig_f, symbol, solution):
                result += FiniteSet(solution)
    else:
        result += solutions
    return result
```

Implementing equations solvable by Lambert W

Equations solvable by LambertW follows f(x, a..f) = a*log(b*X + c) + d*X - f = 0 general form, _solve_lambert describes few forms for f(x, a..f). eqs are identified as these forms and are converted to the general form (f(x, a..f)) so that it can be easily solved by _lambert.
In solveset the idea will be to leverage the power of bivariate.py but smartly. Implementation will be done such that two features are taken care modularity and scalability.
Following is the helper of transolve for solving such equations.

```
def _lambert(f, symbol, domain, **flags):
# Helper for solving transcendental equations
    if flags.pop('bivariate', True):
        try:
            poly = f.as_poly()
 from sympy.solvers.bivariate import _filtered_gens,
_solve_lambert, bivariate_type
            g = _filtered_gens(poly, symbol)
            result = _solve_lambert(f, symbol, g)
            return FiniteSet(result)
        except NotImplementedError:
            if len(g) == 2:
                try:
                # try solving if it is of bivariate type
                    gpu = bivariate_type(f, *g)
                    if gpu is None:
                        raise NotImplementedError
                    g, p, u = gpu
                    flags['bivariate'] = False
                    inv = transolve(g - u, symbol, domain,
**flags)
                    if inv:
                        solution = _solveset(p, u, domain,
_check=True)


    return FiniteSet(list(ordered(set([i.subs(u, s)
```

```
                            for i in inv for s in solution])))
                except NotImplementedError:
                    raise NotImplementedError
            else:
                raise NotImplementedError
    else:
        raise NotImplementedError
```

Overview of how transolve will be solving equations:

```
def transolve(f, symbol, domain, **flags):
    """Helper for solving transcendental equations."""

    orig_f = fraction(together(f, deep=True))[0]
    if 'tsolve_saw' not in flags:
        flags['tsolve_saw'] = []
    if f in flags['tsolve_saw']:
        return None
    else:
        flags['tsolve_saw'].append(f)

    inverter = invert_real if domain.is_subset(S.Reals) else
invert_complex
    lhs, rhs_s = inverter(f, 0, symbol, domain)

    result = S.EmptySet
    if lhs.is_Add:
        # solving equation
        for rhs in rhs_s:
            f = factor(powdenest(lhs - rhs))
            g = _check_log(f) # checking for logarithmic eqs
            if f.is_Mul:
                result += _solveset(f, symbol, domain)
            elif g:
```

```
                # solving logarithmic
                result += _solve_log(f, symbol, domain)
            else:
                # maybe a lambert pattern
                # transolves helper for solving lambert
equations
                result += _lambert(lhs - rhs, symbol,
domain,**flags)

    if result is S.EmptySet:
        result = ConditionSet(symbol, Eq(orig_f, 0), domain)

    return result
```

Phase II

Leftovers of Phase I

Though most of the major task proposed in Phase I will be completed before the deadline but if there is any of the minor tasks that were left during Phase I, will be completed in Phase II.

Improving Set Infrastructure

Currently imageset does not supports unification, due to this there are sometimes complicated result of a large number of union of sets.
For example:

```
>>> solveset(cos(x) + cos(3*x) + cos(5*x) , x, S.Reals)
Union(ImageSet(Lambda(_n, 2*_n*pi + pi/2), S.Integers),
ImageSet(Lambda(_n, 2*_n*pi + 3*pi/2), S.Integers),
ImageSet(Lambda(_n, 2*_n*pi + 4*pi/3), S.Integers),
ImageSet(Lambda(_n, 2*_n*pi + 2*pi/3), S.Integers),
ImageSet(Lambda(_n, 2*_n*pi + 5*pi/3), S.Integers),
ImageSet(Lambda(_n, 2*_n*pi + pi/3), S.Integers),
ImageSet(Lambda(_n, 2*_n*pi + 7*pi/6), S.Integers),
```

```
ImageSet(Lambda(_n, 2*_n*pi + 5*pi/6), S.Integers),
ImageSet(Lambda(_n, 2*_n*pi + 11*pi/6), S.Integers),
ImageSet(Lambda(_n, 2*_n*pi + pi/6), S.Integers))
```

These results can be improved to a much simpler form

```
Union(ImageSet(Lambda(_n, 2*_n*pi + pi/2), S.Integers),
ImageSet(Lambda(_n, _n*pi/2 + pi/6), S.Integers),
ImageSet(Lambda(_n, _n*pi/2 + pi/3), S.Integers))
```

There are few PR's for ref: #11188, #12011, #7673, #10713.
The logic that is currently applied is that if two imagesets are of the a*n*pi + p and a*n*pi + q, and |p-q|/2 is a/2 then union is (a/2)*n*pi + p otherwise a*n*pi + p. Some work still needs to be to improve the output for some cases, add more cases, where unification can be done.

```
# represents odd numbers
>>> img1 = ImageSet(Lambda(_n, 2*_n + 1), S.Integers)
# represents even numbers
>>> img2 = ImageSet(Lambda(_n, 2*_n), S.Integers)
>>> img1.union(img2)
ImageSet(Lambda(_n, _n + 1), S.Integers)
# this could be improved to something like this:
ImageSet(Lambda(_n, _n), S.Integers)
```

Improving trigonometric and Hyperbolic equations

Currently a lot of trigonometric equations are not supported, because trigonometric equations does not have a specific pattern. Solving such equations include simplifying it using trigonometric formulas, rewriting it into other variables and solving as linear equations and others.

Equations that needs to be taken care of are:

i) Nested equations issue #10217
ii) Equations converted into polynomial type.
iii) Complex equations that can be simplified to simpler ones.

Possible Fix: For nested and second type equations we need to integrate solve_decomposition as helper, also for iii) type equations we can take help of

the fu module for simplification of the equations and then solving.
One possible way is to have many smaller helpers that can solve varied type of trigonometric equations, if one fails the other gets the equations and if it fails too, the next one gets. This would probably increase the problem solving capability of solveset.
Ref: https://groups.google.com/d/msg/sympy/sLT7goRueSM/CjF6mmoQAwAJ

Hyperbolic equations are somewhat similar to trigonometric equations. Currently solveset returns a ConditionSet as it is not able to solve.

```
>>> solveset(sinh(x) , x, S.Reals)
ConditionSet(x, Eq((exp(2*x) - 1)*exp(-x)/2, 0), S.Reals)
```

At present such equations are handled by _solve_trig, but there seems a problem in handling these equations. _solve_trig converts equations into equivalent exp form, and replaces exp with dummy variables, this is okay for trigonometric equations but hyperbolic equations does not gets replaced and hence are not solved.

Possible fix:
One thing can be done is to represent hyperbolic equations in terms of simple trigonometric equations (using _osborne in fu module) and solve them.
Second thing that can be done is to rewrite them into exp and substitute dummies in place of exp. Something like below:

```
>>> f = sinh(x).rewrite(exp)
>>> f = f.subs(exp(x), y)
>>> solveset(f, x)    # {-1, 1}
>>> solveset(exp(x) - 1, x)
ImageSet(Lambda(_n, 2*_n*I*pi), S.Integers)    # sol1
>>> solveset(exp(x) + 1, x)
ImageSet(Lambda(_n, I*(2*_n*pi + pi)), S.Integers)     #sol2
```

There is an XFail test_rewrite_trigh which can be solved in similar fashion.
Or **third thing** can be done is to just rewrite it into exp (internally), pass it to solveset and let solveset handle the rewritten equation, because solveset knows how to handle exp type equations.

```
>>> f = solveset(sinh(x).rewrite(exp), x)
ImageSet(Lambda(_n, _n*I*pi), S.Integers)
```

Implementing BigUnion, BigIntersection and IndexSet

At present Sympy cannot represent arbitrary index unions and intersection. In symbolic notation we need union, intersection where α is the indexing variable and X(α) forms a collection of sets indexed by α. For this purpose a new set class needs to be defined IndexSet (Ref.) that will return indices mapped to each set of a set.

Example:

```
>>> X = IndexSet(FiniteSet(1, 2, 3), FiniteSet(3, 4),
FiniteSet(6, 7)); X
IndexSet(0, 1, 2)
>>> for i in range(0, 3):
        print(X[i])
FiniteSet(1, 2, 3)
FiniteSet(3, 4)
FiniteSet(6, 7)
```

Initial implementation of how IndexSet will look:

```
class IndexSet(Set):
# IndexSet will return indices of the elements mapped to the
set.
# Currently, the index is given implicitly.

     def __init__(self, *args):
          self.list = list(args)

     def __new__(cls, *args):

          indexing = {}
          for i in range(0, len(args)):
               indexing[i] = args[i]

          args = list(indexing.keys())

          return Basic.__new__(cls, *args)

     def __getitem__(self, i):
          return self.list[i]
```

BigUnion of finite number of set will be represented as a union, but if it is something like union of intervals then BigUnion needs to be smart in returning the value. Similar for BigIntersection.
Example:

```
>>> f = BigUnion(X, i, FiniteSe(0, 1, 2)); f
{1, 2, 3, 4, 5, 6 ,7}
```

Initial implementation for class BigUnion, further contains method should be overriden to get something like:

```
>>> 7 in f
True
>>> 8 in f
False
```

```
class BigUnion(Set):
      def __new__(cls, index_func, index, index_set):
            set_values = list(index_func[i_s] for i_s in
index_set)

            if all(isinstance(s_v, FiniteSet) for s_v in
set_values):
                  return Union(*set_values)
            return Basic.__new__(cls, *set_values)
```

Adding capability of solving trigonometric system of equations by nonlinsolve

Currently nonlinsolve is unable to solve system of trigonometric equations. This is also because of solveset's inability to solve some trigonometric equations. As trigonometric solver is made powerful under transolve, nonlinsolve capabilities also increases.
Lots of X-fails can be found in test_solveset here.

Possible fix:
One way to get over this is by replacement strategy, for some simple

trigonometric equations we can substitute sin and cos to dummy variables (which will result in nonlinear system of equations) and by calling nonlinsolve again we can get results for those variables. Substituting sin ans cos with these variables and calling solveset to solve the problem and unifying all the result will guarantee all solutions.

Consider a case from Xfail:

```
f = sin(x) + sin(y) - (sqrt(3)+1)/2
g = sin(x) - sin(y) - (sqrt(3) - 1)/2]

Substituting sin(x) with s and sin(y) with c, will yield

f1 = s + c - (sqrt(3)+1)/2
f2 = s - c - (sqrt(3)+1)/2

Passing these equations to nonlinsolve as:
>>> nonlinsolve([f1, f2], [s, c])
{(sqrt(3)/2, 1/2)}

Now replacing s with sin(x) and c with sin(y) and solving using
solveset will yield the desired result.

>>> sol1 = solveset(sin(x) - sqrt(3)/2, x, S.Reals)
Union(ImageSet(Lambda(_n, 2*_n*pi + 2*pi/3), S.Integers),
ImageSet(Lambda(_n, 2*_n*pi + pi/3), S.Integers))
>>> sol2 = solveset(sin(y) - 1/2, x, S.Reals)
Union(ImageSet(Lambda(_n, 2*_n*pi + 5*pi/6), S.Integers),
ImageSet(Lambda(_n, 2*_n*pi + pi/6), S.Integers))

# unifying the result.
result = sol1 + sol2
```

Similarly `sin(x) + cos(x) - 1, sin(x)**2 + cos(y)**2 - 0.5` can be solved.
System of Lambert equations can be solved once solving it in transolve is fulfilled, because for its solutions nonlinsolve is dependent on solveset.

Phase III

Leftovers of Phase II

Though most of the major task proposed in Phase II will be completed before the deadline but if there is any of the minor tasks that were left during Phase II, will be completed in Phase III.

Integrating helpers

Though most of the helpers functions will be incorporated while implementing transolve, helpers like linsolve, nonlinsolve will be integrated so that there is no need for special call to them. The idea is make solveset completely user friendly so any type of equations that needs to be solve (be it system of linear or nonlinear equations), will be solved by calling solveset only, and there will be internal call to respective type of equations.

Implementation:
For achieving such type of scalability we need to first make solveset take one or more than one symbol and equations. After making this we will be solving the equation(s) by determining what type of equations it is and after finding it sending it to respective helpers.

```
def solveset(f, *symbols, domain):
     if len(f) == 1 and len(symbols) == 1:
          pass
     else:
          # equation can be solved by linsolve or nonlinsolve
          try:
               # try solving through linsolve
          except:
               # try solving through nonlinsolve
```

Modular Arithmetic

Currently solveset does not support solving modular equations with integer solutions (issue #13178). Solving these equations would definitely extend the coverage of solveset of solving varied type of equations.
In current master solveset return ConditionSet for such equations

```
>>> n = symbols('n', integer=True)
>>> a = 742938285
>>> z = 1898888478
>>> m = 2**31 - 1
>>> x = 20170816
>>> solveset(x - Mod(a**n*z, m), n, S.Integers)
ConditionSet(n, Eq(Mod(1898888478*742938285**n, 2147483647) -
20170816, 0), S.Integers)
```

Modular equations deals with modular arithmetic and modulo operation. These equations gets evaluated only in Integer domain. (Ref here)

How modular operations are different from conventional arithmetic

- The solutions of modular equations have wrap around properties, where integers on reaching a certain value repeats itself (congruence modulo)
- Multiplication and division are to be carried out differently, concept of multiplicative inverse is used.
- Exponential equations cannot be solved by the general log method, discrete log (Ref: here)needs to be used.

Example:
Consider **f = 2 - Mod(4*x, 5),** it means that what will be the value of x that is a multiple of 4 and gives 2 as remainder when modulo 5 is applied. 2 ~ Mod(4*x, 5) therefore the series will be [3, 8, 13, …..] or can be represented as **ImageSet(Lambda(5*_n + 3), S.Integers)**.

Possible fix: One way to solve such equation is to create particular inversion method (invert_modular) that will solve the equation by inverting (similar as _invert), but there might be complex equations like **quadratic modular equations** probably needing a helper for this.
Here I have found few references where way of solving quadratic modular

equations can be found

- https://math.stackexchange.com/questions/261896/modular-quadratic-formula
- https://www.johndcook.com/blog/quadratic_congruences/
- http://2000clicks.com/MathHelp/NumberSquareQuadraticResidues.aspx

Timeline

This project will definitely extend the functionality of solveset and might take more time, but I am confident and I assure that I will devote all my time in this project and try to finish what I proposed. I will be having my semester exams till end of May(tentative), but even during this time I will try to give 2-3 hours daily and thereafter I will devote all of my time (about 40-50 hrs a week or even more).

Community Bonding Period (April 24 - May 13)

- In this period I will try to plan out the things the way it needs to be implemented by discussing with the mentors.
- I will be studying Sympy's codebase thoroughly (particularly solvers) and try to use its advantage in the implementation to the fullest.

Week (1, 2, 3) (May 14 to 4 June)

- Implementing what is discussed in Phase I.
- Completely implementing transolve and its helpers.

Week (4, 5) (June 5 - 15)

- Writing tests for transolve.
- Documenting the implementation.
- Submitting Phase 1 evaluations.

Week (6, 7, 8) (June 16 - July 13)

- Completing leftovers of Phase I
- Implementing Phase II work.
- Writing tests and documenting them.
- Submitting Phase II evaluations

Week (9, 10, 11, 12) (July 14 - August 14)

- Completing leftovers for Phase II.
- Implementing Phase III work
- Adding tests and documentation
- Submitting final evaluations

Any Commitments/Plans (during GSoC)

- I don't have any specific commitments for the summer, therefore I can devote my whole time.
- I will be writing blogs of what all discussions have taken place and also what and how I have implemented.
- I will only have my exams during the Community Bonding Period and during first week of Coding Period, therefore I will spend a bit less of my time but as soon as they get over I will give all of my time (will also try to compensate the time lost due to exams).

Post GSoC

- If there are things that are left unimplemented I will try to complete post GSoC and even I will also continue my contributions to sympy and help the new contributors.
- As I mentioned I started coding with C++, I will also try to contribute to SymEngine.

References

- Shekhar Prasad Rajak GSoC application.
- Amit Kumar GSoC application
- Wiki resource on Lambert W
- Discussion on improving trigonometric equations.
- Discussion on implementing transolve.
- Shekhar's Blog.
- Solveset and Solvers discussion.
- Harsh Gupta PR on Imageset Union.
- Shekhar's PR on ImageSet Union.