

RSOLVER  
User Manual

Stefan Ratschan

June 18, 2007

# Chapter 1

## Introduction

Take the expression  $x^2 + y^2 + z^2 \leq 1$ , where the variables  $x$ ,  $y$  and  $z$  range over the real numbers. This expression represents a set of values—its solution set, which is a ball. We call such an expression a *constraint*. *Solving* a constraint means to find some interesting information about its solution set, where the definition of “interesting” depends on the application. For example, we might want to find a point in the solution set, or we might want to find an approximation of the solution set by hyper-rectangles.

This software package solves a special type of constraints, namely *quantified constraints*. Take the following example:  $\exists z \in [-2, 2] x^2 + y^2 + z^2 \leq 1$ . Here we encounter the symbol  $\exists$ —an existential quantifier. The solution set of this constraint is the set of all  $x$  and  $y$  such that there is a  $z$  such that  $x^2 + y^2 + z^2 \leq 1$  holds. Clearly this is a disc, that is, the projection of the solution set of  $x^2 + y^2 + z^2 \leq 1$  onto the two-dimensional real space. Such a projection can be very convenient, because one can easily visualize sets in two-dimensional real space.

A quantified constraint can also contain the symbol  $\forall$ , that is, universal quantifiers. This can be very useful for dealing with uncertain values. These occur, for example, in robust control [4], where one studies notions such as the stability of systems under uncertain values.

A quantified constraint can also contain the Boolean symbols for conjunction and disjunction. In general, it is a formula in the first-order predicate language [5] over the reals.

Traditionally, such constraints have been solved by symbolic methods [15, 2, 3]. However, these methods suffer from various drawbacks—especially for inputs that model real-world problems. Instead of symbolic solutions, the algorithm [10] employed in this software computes solutions that are approximate but correct.

For example, in Figure 1.1 you can see a visualization of the solution computed for the constraint  $\exists z \in [-2, 2] x^2 + y^2 + z^2 \leq 1$ . Here you can see a green area—elements that are guaranteed to be within the solution set, a

read area—elements that are guaranteed to be out of the solution set, and a white area—elements for which we do not have any information.

The user can specify an error bound—the maximum allowed fraction of the white area wrt. the total area. For example, the error bound used for Figure 1.1, was 0.05.

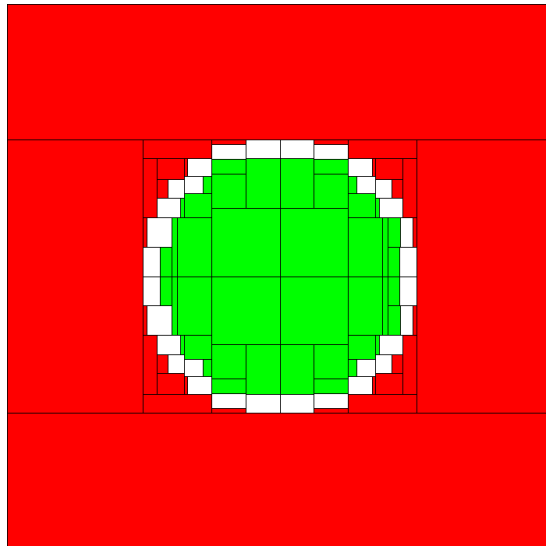


Figure 1.1: Solution for  $\exists z \in [-2, 2] x^2 + y^2 + z^2 \leq 1$

## Chapter 2

# Running a First Example

RSOLVER itself works in text mode. An additional graphical user interface (GUI) is available. For using the GUI, download it and follow the instructions in the files README.WINDOWS (if you are using Microsoft Windows) or README (otherwise).

When starting RSOLVER without the GUI, it asks for textual input. Some example inputs are included in the distribution. One can avoid to re-type the content of an example file by piping the file content to the program as follows:

In UNIX: Type

```
./rsolver <examples/basic/test1.ap
```

In Windows: Start the program “command prompt”, change to the directory where the program resides, and type:

```
rsolver <examples\basic\test1.ap
```

## Chapter 3

# The Input

The following is an example program input:

```
[x, y]
EXISTS [z] [[-2, 2]] [ x^2+y^2+z^2<=1 /\ y>=x^2 ];
[[-2,2], [-2,2]]
```

The three elements represent the following objects:

- the variables that span the solution space (i.e., free variables),
- the quantified constraint  $\exists z \in [-2, 2][x^2 + y^2 + z^2 \leq 1 \wedge y \geq x^2]$ ,
- a box (i.e., Cartesian product of closed intervals) of the same dimension as the free variables.

In general, the representation of the input constraint can contain the following symbols:

**Quantifiers:** FORALL, EXISTS

**Connectives:**  $\setminus$ ,  $\wedge$ ,  $\implies$

**Predicates:**  $<$ ,  $<=$ ,  $>$ ,  $>=$

**Function Symbols:**  $*$ ,  $+$ ,  $\hat{\phantom{x}}$ , SIN, COS, EXP, ASIN, ACOS, ATAN, LOG

**Constants:** floating point numbers, PI, E

**Variables:** lowercase alphanumeric strings starting with a letter

There are additional predicates calling efficient external solvers. These are documented in Section 6

All binary symbols are infix, and all other prefix. One can use parenthesis for the term structure and brackets for the logical structure. Quantifiers have three arguments:

1. a list of quantified variables,
2. a floating point bounding box on these variables, and
3. the quantified constraint.

Note that quantifiers bind stronger than connectives. Hence the input

```
EXISTS [x] [[ 1,3 ]] x>0 /\ x<2;
```

will result in the error message that the variable x cannot be found. Using brackets, arriving at

```
EXISTS [x] [[ 1,3 ]] [ x>0 /\ x<2 ];
```

solves the problem.

Although the software also allows equality predicates = it currently does not produce satisfying answers. Work on constraints with equalities is in progress.

## Chapter 4

# Produced Output

The output is a set of boxes on which the input is guaranteed to be true, a set of boxes, on which the input is guaranteed to be false, and a set of boxes for which we do not know anything. Note that if the input contains function symbols that are not defined everywhere (e.g.,  $\text{asin}$  is only defined on the interval  $[-1, 1]$ ), then also the truth value of the corresponding constraint might be undefined, and no corresponding box of true or false values will be computed.

In some cases, the software also prints witnesses corresponding to these boxes. For example, in the case when an existentially quantified constraint has been proven to hold, in certain cases it also prints a witness for this quantifier, that is, a point for which the constraint under the quantifier holds.

There is a separate graphical user interface, which prints true boxes in green and false boxes in red. Note that the graphical output is not faithfully rounded. For example, a green and a red box might touch on the screen, although the computed boxes do not.

## Chapter 5

# Running Rsolver

After starting the program, it asks for the inputs described in Chapter 3. Example inputs are contained in the distribution package. For controlling the solver further, one can use arguments when calling the program. These arguments are described when calling the solver by `rsolver -h`.

Note that the efficiency of the algorithm depends on the specific form, an input constraint is given. For example, a constraint of the form  $\forall x [\phi_1 \wedge \phi_2]$  is handled in a different way than a constraint of the form  $\forall x \phi_1 \wedge \forall x \phi_2$ .

Usually (but not always!) the following transformations increase the efficiency:

- Rewriting constraint of the form  $\forall x [\phi_1 \wedge \phi_2]$ , or  $\exists x [\phi_1 \vee \phi_2]$  to  $\forall x \phi_1 \wedge \forall x \phi_2$ , and  $\exists x \phi_1 \vee \forall x \phi_2$ , respectively.
- Factoring terms as much as possible.
- Decomposing terms that can be written as  $f(g(x))$  to a term  $f(y)$  with an additional constraint  $y = f(x)$ .

Another possibility of influencing the efficiency of `RSOLVER` is to switch on the use of mean value constraints [7], using the flag `-f MeanValueConstraint`.

If `RSOLVER` still cannot solve your problem, we strongly encourage you to contact us ([stefan.ratschan@cs.cas.cz](mailto:stefan.ratschan@cs.cas.cz)). Your feedback will be essential for improving the software further.

## Chapter 6

# External Solvers

Some versions of RSOLVER also allow the predicate PSD that ensures positive-semidefiniteness of a matrix by calling an external solver for semidefinite programming. For example

```
PSD([[x, y], [2 x, x+y]]);
```

finds values for  $x$  and  $y$ , such that the matrix

$$\begin{pmatrix} x & y \\ 2x & x + y \end{pmatrix}$$

is positive semi-definite.

Warning: the employed external solver does non-validated floating point computation. Therefore its result may be incorrect due to rounding errors.

## Chapter 7

# Relaxation Algorithm

The solver provides a special algorithm for constraints of the form

$$(\exists \vec{a} \in A) \bigwedge_k \theta_k(\vec{a})$$

where  $\theta_k(\vec{a})$  is either of the form  $\sum_i c_i a_i + d_i = 0$ , or of the form

$$(\forall \vec{x} \in X) \left( \sum_i a_i \phi_i(\vec{x}) \leq 0 \vee \psi(\vec{x}) \right),$$

where  $\phi_i(x)$  and  $\psi(x)$  may be an arbitrary Boolean combination of constraints, but they are only allowed to contain the variables  $\vec{x}$ .

For using this algorithm, `RSOLVER` has to be called using the parameters `-f Sample -p DeductionStrategy Relaxation`, and the existential quantifier has to be written in the variant `EXISTS*`. See the examples in the directory `examples/relax` of the distribution.

The corresponding algorithm exploits the fact that the  $a_i$  only occur linearly. This allows solving the problem using linear-programming relaxations [14].

## Chapter 8

# Program Termination

Consider the constraint  $\exists x \in [-2, 2] x^2 - 1 \leq 0$ . This constraint certainly is true. Now replace the zero on the right-hand side of the equality by a very small real number  $\varepsilon$ . Certainly this does not change the truth value. Hence we call such a constraint *robust*. On the other hand, consider the constraint  $\exists x \in [-2, 2] x^2 \leq 0$ . It is also true, but changes its truth value to false if we replace the zero by a very small negative constant. Hence we call such a constraint *fragile*.

The algorithm in RSOLVER uses approximation. So it will usually not use the exact value of constants, but can only enclose them into small intervals. Therefore, it will in general not be able to compute the truth value of fragile constraints. But it will always terminate for robust constraints. So in that case you will always get an answer, provided you are willing to wait long enough.

Note that in practice, especially for constraints that are almost fragile, it might also happen that floating point precision is not sufficient to compute an answer. In that case, the system will terminate with an according error message.

For more information on the property of being robust or fragile, you can read some articles for the case of quantified constraints [11] or general books on numerical analysis (under the terms “well-posed”, “ill-posed”, “well-conditioned”, “ill-conditioned”).

## Chapter 9

# The Algorithm

A detailed description of the algorithm employed by this software can be found in other papers [13]. Some preprints are available from the author's webpage (<http://www.cs.cas.cz/~ratschan>).

For using the software it is not necessary to grasp all its internal details. However—for using it *efficiently*—it can be useful to understand roughly, how the algorithm works:

The basic idea is, to compute some information for the atomic sub-constraints (e.g., the occurring inequalities), and using it to compute some information of the total constraint. If this procedure fails, we decompose the variable space into pieces, until some new information can be deduced.

In the simplest case, we use the following test on atomic constraints:

- Given:
  - An atomic constraint  $\phi$ ,
  - a box  $B$
- Find: An element of the set  $\{\mathbf{T}, \mathbf{F}, \mathbf{U}\}$  such that
  - $\mathbf{T}$  implies that  $\phi$  is true for all elements of  $B$ , and
  - $\mathbf{F}$  implies that  $\phi$  is false for all elements of  $B$ .

Such a test is provided by techniques of *interval analysis* [9, 8]. Of course, the test should return  $\mathbf{T}$  or  $\mathbf{F}$  as often as possible. Except for degenerate cases, such tests succeed to do this for sufficiently small boxes.

Consider the example  $\exists y \in [-2, 2] \ x^2 + y^2 \leq 1$ , where the variable  $x$  ranges over the interval  $[-2, 2]$ . Here we just have one atomic sub-constraint— $x^2 + y^2 \leq 1$ —on which to do the test. Of course, for the box  $[-2, 2] \times [-2, 2]$  it can only fail. Therefore the algorithm will do branching: it will either split the interval corresponding to a free variable into two parts, or it will rewrite  $\exists y \in [-2, 2] \ x^2 + y^2 \leq 1$  to the equivalent constraint

$\exists y \in [-2, 0] x^2 + y^2 \leq 1 \vee \exists y \in [0, 2] x^2 + y^2 \leq 1$ . In a similar way it can split universal quantifiers into a conjunction of several universally quantified constraints.

This splitting can be prohibited by using starred versions of the quantifiers, that is, by using **FORALL\***, and **EXISTS\***.

More details can be found in some articles [10, 12].

## Chapter 10

# The Implementation

The software is implemented in the programming language O'Caml. It uses the interval and constraint propagation library `smathlib` [6].

## Chapter 11

# Known Bugs

In contrast to the solving algorithms, parsing and printing currently may introduce errors due to rounding. For printing the exact result, one can use the flag `-f ExactBoundaries`.

# Bibliography

- [1] B. F. Caviness and J. R. Johnson, editors. *Quantifier Elimination and Cylindrical Algebraic Decomposition*. Springer, Wien, 1998.
- [2] G. E. Collins. Quantifier elimination for the elementary theory of real closed fields by cylindrical algebraic decomposition. In *Second GI Conf. Automata Theory and Formal Languages*, volume 33 of *LNCS*, pages 134–183. Springer Verlag, 1975. Also in [1].
- [3] G. E. Collins and H. Hong. Partial cylindrical algebraic decomposition for quantifier elimination. *Journal of Symbolic Computation*, 12:299–328, 1991. Also in [1].
- [4] P. Dorato, W. Yang, and C. Abdallah. Robust multi-objective feedback design by quantifier elimination. *Journal of Symbolic Computation*, 24:153–159, 1997.
- [5] H.-D. Ebbinghaus, J. Flum, and W. Thomas. *Mathematical Logic*. Springer Verlag, 1984.
- [6] T. J. Hickey. smathlib. <http://interval.sourceforge.net/interval/prolog/clip/clip/smath/README.html>.
- [7] T. J. Hickey. Metalevel interval arithmetic and verifiable constraint solving. *Journal of Functional and Logic Programming*, 2001(7), October 2001.
- [8] L. Jaulin, M. Kieffer, O. Didrit, and É. Walter. *Applied Interval Analysis, with Examples in Parameter and State Estimation, Robust Control and Robotics*. Springer, Berlin, 2001.
- [9] A. Neumaier. *Interval Methods for Systems of Equations*. Cambridge Univ. Press, Cambridge, 1990.
- [10] S. Ratschan. Continuous first-order constraint satisfaction. In J. Calmet, B. Benhamou, O. Caprotti, L. Henocque, and V. Sorge, editors, *Artificial Intelligence, Automated Reasoning, and Symbolic Computation*, number 2385 in *LNCS*, pages 181–195. Springer, 2002.

- [11] S. Ratschan. Quantified constraints under perturbations. *Journal of Symbolic Computation*, 33(4):493–505, 2002.
- [12] S. Ratschan. Search heuristics for box decomposition methods. *Journal of Global Optimization*, 24(1):51–60, 2002.
- [13] S. Ratschan. Efficient solving of quantified inequality constraints over the real numbers. *ACM Transactions on Computational Logic*, 7(4):723–748, 2006.
- [14] S. Ratschan and Z. She. Providing a basin of attraction to a target region by computation of Lyapunov-like functions. In *IEEE Int. Conf. on Computational Cybernetics*, 2006.
- [15] A. Tarski. *A Decision Method for Elementary Algebra and Geometry*. Univ. of California Press, Berkeley, 1951. Also in [1].