Redlog User Manual

Edition 1.0, for REDLOG Version 1.0

Andreas Dolzmann and Thomas Sturm

MIP-9616

9 October 1996

Abstract

REDLOG stands for REDuce LOGic system. It provides an extension of the computer algebra system REDUCE to a "computer logic system" implementing symbolic algorithms on first-order formulas wrt. temporarily fixed first-order languages and theories. Underlying theories currently available are ordered fields and discretely valued fields. Though the focus of the implemented algorithms is on effective quantifier elimination and simplification of quantifier-free formulas, REDLOG is intended and designed as an all-purpose system. REDLOG is freely available to the scientific community.

Copyright © 1996 A. Dolzmann and T. Sturm, Lehrstuhl fuer Mathematik, Universitaet Passau.

Table of Contents

1 Introduction 2

1.1 Overview 2
1.2 Getting Started 3
1.3 Contexts 3
1.4 On the Use of Switches 4

2 Format and Handling of Formulas 5

2.1 First-order Operators 5
2.2 OFSF Operators 6
2.3 DVFSF Operators 6
2.4 Extended Built-in Commands 7
2.5 Advanced Features 7

3 Simplification 9

3.1 Standard Simplifier 9
3.2 Tableau Simplifier 12
3.3 Groebner Simplifier 13

4 Normal Forms 16

4.1 Boolean Normal Forms 16
4.2 Other Normal Forms 16

5 Quantifier Elimination and Variants 18

5.1 Quantifier Elimination 18
5.2 Generic Quantifier Elimination 20
5.3 Linear Optimization 21

6 Miscellaneous 22

6.1 Global Switches 22
6.2 Analyzing Formulas 22
6.3 Other Stuff 23

7 References 25

Function Index 27

Variable Index 28

Data Structure Index 29

Concept Index **30**

Acknowledgements

We acknowledge with pleasure the superb support by Herbert Melenk and Winfried Neun of the Konrad-Zuse-Institute Berlin during this project. Furthermore, we wish to mention numerous valuable mathematical ideas contributed by Volker Weispfenning, University of Passau.

Using and Copying

Permission is granted to use the *free scientific* REDLOG *version* available on `http://www.fmi.uni-passau.de/~redlog/` for scientific purposes.

Incorporation of the free scientific REDLOG version into other programs, modification, redistribution, and commercial use are strictly prohibited.

There is a *commercial* REDLOG *version* marketed. It provides the same functionality as the scientific one but includes a program documentation. In contrast to the scientific version, it may be used for commercial purposes, incorporated into other programs, and modified. For obtaining the commercial version mail to `redlog@fmi.uni-passau.de`. Our group also provides mathematical and technical support.

For all REDLOG versions, there is no warranty for the program, to the extent permitted by applicable law. Except when otherwise stated in writing the copyright holders and/or other parties provide the program "as is" without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance of the program is with you. Should the program prove defective, you assume the cost of all necessary servicing, repair or correction.

Bug Reports and Comments

Send bug reports and comments to `redlog@fmi.uni-passau.de`. Any hint or suggestion is welcome. In particular, we are interested in short reports on practical problems to the solution of which our software has contributed.

1 Introduction

1.1 Overview

REDLOG stands for REDuce LOGic system. It provides an extension of the computer algebra system REDUCE to a *computer logic system* implementing symbolic algorithms on first-order formulas wrt. temporarily fixed first-order languages and theories. Underlying theories currently available are *ordered fields* and *discretely valued fields* where the focus is on the former one.

The focus of the implemented algorithms is on *effective quantifier elimination* and *simplification* of quantifier-free formulas. For ordered fields there are currently the following algorithms available:

- Several techniques for the *simplification* of quantifier-free formulas. The simplifiers do not only operate on the Boolean structure of the formulas but also discover algebraic relationships. For this purpose, we make use of advanced algebraic concepts such as Groebner basis computations. For the notion of simplification and a detailed description of the implemented techniques cf. [DS95] (Chapter 7 [References], page 25).
- *Quantifier elimination*, i.e., computing quantifier-free equivalents for given first-order formulas. The current implementation is restricted to at most quadratic occurrences of the quantified variables. We use a technique based on elimination set ideas [Wei88], [LW93], [Wei] (Chapter 7 [References], page 25).
- A variant of the quantifier elimination procedure for geometric theorem proving: Non-degeneracy conditions are detected automatically [DSW96] (Chapter 7 [References], page 25). This has turned out useful also for physical applications such as network analysis [Stu96] (Chapter 7 [References], page 25).
- Variants of both classical and generic quantifier elimination that provide *answers*, e.g., satisfying sample points for existentially quantified formulas. This has been referred to as the "extended quantifier elimination problem" [Wei96] (Chapter 7 [References], page 25).
- Linear *optimization* using quantifier elimination techniques [Wei94a] (Chapter 7 [References], page 25).
- CNF/DNF computation including both Boolean and algebraic simplification strategies [DS95] (Chapter 7 [References], page 25).
- Several other normal form computations, e.g., *prenex normal form* computation minimizing the number of quantifier changes.
- A lot of useful *tools* for constructing, decomposing, and analyzing formulas.

For discretely valued fields there are not all algorithms available. Currently, this manual focuses on ordered fields. Users interested in discretely valued fields are asked to study the ordered field documentation for now, and to try out what works. For the theoretical background of the discretely valued fields algorithms cf. [Wei88], [Stu95] (Chapter 7 [References], page 25).

For ordered fields, it is planned to extend the quantifier elimination to higher degrees. On the mathematical side, the necessary methods are known [Wei], [Wei94b] (Chapter 7 [References], page 25).

This document serves as an user guide describing the usage of REDLOG from the algebraic mode of REDUCE. For a detailed description of the system's design see [DS96] (Chapter 7 [References], page 25). A program documentation is available only with the commercial version of REDLOG.

Finally, we wish to point out that the range of available algorithms is strongly influenced by our major research topics in this area, i.e., quantifier elimination and simplification. Nevertheless, REDLOG is designed as a *general-purpose* computer logic system.

1.2 Getting Started

REDLOG is a REDUCE package, which has to be loaded explicitly. This is done by the following command within REDUCE:

```
load_package rl;
```

Before starting work, a *context* has to be selected.

1.3 Contexts

REDLOG is designed for working with several *languages* and *theories* in the sense of logic. Both a language and a theory make up a context. In addition, a context determines the internal representation of terms. The context to be used has to be selected explicitly. This can be done in several ways. The first possibility is using the `rlset` command.

rlset [*context* [*arguments...*]] Function
rlset *argument-list* Function

Set current context. Currently, valid choices for *context* are `ofsf` (ordered fields standard form) and `dvfsf` (discretely valued fields standard form). With `ofsf` no further arguments are possible. With `dvfsf` one can pass the (maximal) characteristic of the residue class fields wrt. the valuation as a (negative) prime number. The default is 0, leading to computations that are correct for all discretely valued fields. `rlset` returns the old setting as a list that can be saved to be passed to `rlset` later. When

called with no arguments (or the empty list), `rlset` returns the current setting.

Alternatively, there are two ways to define a default context: Either by setting a REDUCE fluid variable before loading '`rl`' or by setting an environment variable before starting REDUCE.

rldeflang!* Fluid

Default language. This may be bound to a default context before loading '`rl`'. Typically, one would add the following line to the '`.reducerc`' file:

```
lisp (rldeflang!* := '(ofsf));
```

Notice that the Lisp list representation has to be used here. If `rldeflang!*` is non-nil, `rlset` is automatically executed when loading '`rl`'.

Using an environment variable, in contrast, does not allow to specify extra arguments.

RLDEFLANG Environment Variable

Default language. This may be bound to a context in the sense of the first argument of `rlset`, e.g., with the Bash shell one would say:

```
export RLDEFLANG=ofsf
```

With `RLDEFLANG` set, any `rldeflang!*` binding is overloaded.

1.4 On the Use of Switches

REDLOG contains numerous switches. Some of them have been introduced only for finding the right fine tuning of the functions. They should not be changed anymore, except for in very special situations. For an easier orientation the switches are divided into three categories for documentation:

1. "Switch": an ordinary switch.
2. "Fix Switch": you do not want to change it.
3. "Advanced Switch": you need a good knowledge about the underlying algorithms for making use of it.

2 Format and Handling of Formulas

After loading REDLOG and selecting a context, there are first-order *formulas* available as an additional type of expressions.

2.1 First-order Operators

Though the operators `and`, `or`, and `not` are already sufficient for representing Boolean formulas, REDLOG provides a variety of other Boolean operators for the convenient mnemonic input of Boolean formulas.

not Unary Operator
and n-ary Infix Operator
or n-ary Infix Operator
impl Binary Infix Operator
repl Binary Infix Operator
equiv Binary Infix Operator

The infix operator precedence is from strongest to weakest: `and`, `or`, `impl`, `repl`, `equiv`.

rlbrop Fix Switch

Bracket all operators. By default this switch is on, which causes some private printing routines to be called for formulas: All subformulas are bracketed completely making the output more readable.

ex Binary Operator
all Binary Operator

These are the *quantifiers*. The first argument is the quantified variable, the second one is the matrix formula. Optionally, one can input a list of variables as first argument. This leads to an expansion into several nested quantifiers.

true Variable
false Variable

These AM variables are reserved. They serve as *truth values*.

mkand for loop action
mkor for loop action

Make and/or. Actions for the construction of large systematic conjunctions/disjunctions via for loops.

```
for i:=1:3 mkand
   for j:=1:3 mkor
      if j<>i then mkid(x,i)+mkid(x,j)=0;
           ⇒ true and (false or false or x1 + x2 = 0
                       or x1 + x3 = 0)
               and (false or x1 + x2 = 0 or false
                    or x2 + x3 = 0)
               and (false or x1 + x3 = 0 or x2 + x3 = 0
                    or false)
```

2.2 OFSF Operators

The OFSF context implements *ordered fields* over the language of *ordered rings*. Proceeding this way is very common in model theory since one wishes to avoid functions which are only partially defined. Furthermore, we wish to point out, that the quantifier elimination procedures for non-linear formulas actually operate over *real closed fields*.

equal Binary Infix operator
neq Binary Infix operator
leq Binary Infix operator
geq Binary Infix operator
lessp Binary Infix operator
greaterp Binary Infix operator

The above operators may also be written as `=`, `<>`, `<=`, `>=`, `<`, and `>`, respectively. For OFSF there is specified that all right hand sides must be zero. Non-zero right hand sides in the input are hence subtracted immediately to the corresponding left hand sides. There is a facility to input *chains* of the above relations, which are also expanded immediately:

```
a<>b<c>d=f
     ⇒ a-b <> 0 and b-c < 0 and c-d > 0 and d-f = 0
```

Here, only adjacent terms are related to each other.

2.3 DVFSF Operators

Discretely valued fields are implemented as a one-sorted language using the operators `|`, `||`, `~`, and `/~`, which encode `<=`, `<`, `=`, and `<>` in the value group, respectively.

equal Binary Infix operator
neq Binary Infix operator
div Binary Infix operator
sdiv Binary Infix operator
assoc Binary Infix operator
nassoc Binary Infix operator

The above operators may also be written as `=`, `<>`, `|`, `||`, `~`, and `/~`, respectively.

2.4 Extended Built-in Commands

The REDUCE substitution command `sub` can be applied to formulas using the usual syntax.

substitution_list Data Structure

substitution_list is a list of equations each with a kernel left hand side.

sub *substitution_list formula* Function

Substitute. Returns the formula obtained from *formula* by applying the substitutions given by *substitution_list*.

```
sub(a=x,ex(x,x-a<0 and all(x,x-b>0 or ex(a,a-b<0))));
     ⇒ ex x0 ( - x + x0 < 0 and all x0 (
             - b + x0 > 0 or ex a (a - b < 0)))
```

`sub` works in such a way that equivalent formulas remain equivalent after substitution. In particular, quantifiers are treated correctly.

part *formula n1* [*n2* [*n3...*]] Function

Extract a part. The `part` of *formula* is implemented analogously to that for built-in types: in particular the 0th part is the operator.

length *formula* Function

Determine the length. The `length` of *formula* is the number of arguments to the top-level operator. This is of particular interest with the n-ary operators `and` and `or`. Notice that `part(`*formula*`,length(`*formula*`))` is the part of largest index.

2.5 Advanced Features

rlall *formula* [*exceptionlist*] Function

Universal closure. *exceptionlist* is a list of variables empty by default. Returns *formula* with all free variables universally quantified, except for those in *exceptionlist*.

rlex *formula* [*exceptionlist*] Function
Existential closure. *exceptionlist* is a list of variables empty by default. Returns *formula* with all free variables existentially quantified, except for those in *exceptionlist*.

rlmatrix *formula* Function
Matrix computation. Removes all *leading* quantifiers from *formula*.

3 Simplification

The goal of simplifying a first-order formula is to obtain an equivalent formula over the same language that is somehow simpler. REDLOG knows three kinds of simplifiers: The standard simplifier, tableau simplifiers, and Groebner simplifiers. They are all described in [DS95] (Chapter 7 [References], page 25).

3.1 Standard Simplifier

The *Standard Simplifier* is a fast simplification algorithm that is frequantly called internally by other REDLOG algorithms. It can be applied automatically at the expression evaluation stage (see Chapter 6 [Miscellaneous], page 22).

theory Data Structure

A list of atomic formulas assumed to hold.

rlsimpl *formula* [*theory* [*depth*]] Function

Simplify. *formula* simplified recursively, *depth* is an integer describing a level where recursion has to be stopped. Defaults for the optional arguments are the empty theory `{}` and level `-1`. The latter causes the whole formula to be simplified. The simplification includes the following features:

- Producing canonical forms for atomic formulas.
- Proper treatment of truth values.
- Flattening nested n-ary operator levels and resolving involutive applications of `not`.
- Changing `repl` to `impl`.
- Considering interaction of atomic formulas on the same level (switch `rlsism`).
- Producing a canonical ordering among the atomic formulas on a given level (switch `rlsisort`).
- Recognizing equal subformulas on a given level (switch `rlsichk`).
- Passing down information that is collected during recursion (switches `rlsism`, `rlsiidem`).

In the OFSF case the atomic formula simplification includes the following:

- Evaluation of variable-free atomic formulas to truth values.

- Evaluation of trivial square sums to truth values (switch `rlsisqf`). Additive splitting of trivial square sums (switch `rlsitsqspl`).
- Parity decomposition (similar to square-free decomposition) of terms (switch `rlsipd`) with the option to split an atomic formula multiplicatively into two simpler ones (switches `rlsiexpl`, `rlsiexpla`).
- Moving the `not` operator into the relation with a single negated atomic formula.

Theory inconsistency may but need not be detected by `rlsimpl`. In the first case, an error is raised. Else, mind that under an inconsistent theory, any formula is equivalent to the input.

rlsisqf Switch

Simplify square-free. By default this switch is on. Enables simplifications based on square-free part computations, square-free decompositions, and trivial square-sums.

```
rlsimpl(a**2 + 2*a*b + b**2 <> 0);
     ⇒ a+b <> 0

rlsimpl(a**2 + b**2 + 1 > 0);
     ⇒ true
```

rlsifac Switch

Simplify factorization. By default this switch is on. It is ignored with `rlsisqf` off. Explodes equations and inequations via factorization of their left hand side terms into disjunctions and conjunction, respectively. This is done in dependence on `rlsiexpl` and `rlsiexpla`.

rlsitsqspl Switch

Simplify split trivial square sum. This is on by default. It is ignored with `rlsisqf` off. Trivial square sums are split additively depending on `rlsiexpl` and `rlsiexpla`:

```
rlsimpl(a**2+b**2>0);
     ⇒ a <> 0 or b <> 0
```

rlsipd Switch

Simplify parity decomposition. By default this switch is on. It is only relevant with the switch `rlsisqf` on. `rlsipd` toggles the parity decomposition of terms occurring with ordering relations.

```
f := (a - 1)**3 * (a + 1)**4 >= 0;
         7    6      5      4      3      2
     ⇒ a  + a  - 3*a  - 3*a  + 3*a  + 3*a  - a - 1 >= 0
```

```
rlsimpl f;
         3    2
     ⇒ a  + a  - a - 1 >= 0
```

rlsiexpla Switch

Simplify explode always. By default this switch is on. It is relevant with `rlsisqf` and one of `rlsipd`, `rlsifac`, or `rlsitsqspl` on.

```
f := (a - 1)**3 * (a + 1)**4 >=0;
          7    6      5      4      3      2
     ⇒ a  + a  - 3*a  - 3*a  + 3*a  + 3*a  - a - 1 >= 0

rlsimpl f;
     ⇒ a - 1 >= 0 or a + 1 = 0
```

rlsiexpl Switch

Simplify explode. By default this switch is on. It is relevant with `rlsisqf` and one of `rlsipd`, `rlsifac`, or `rlsitsqspl` on. With `rlsiexpl` on

```
  7    6      5      4      3      2
a  + a  - 3*a  - 3*a  + 3*a  + 3*a  - a - 1 >= 0
```

simplifies as in the description of the switch `rlsiexpla` whenever it occurs in a disjunction, and it simplifies as in the description of the switch `rlsipd` else.

rlsipw Switch

Simplification prefer weak orderings. Prefers weak orderings in contrast to strong orderings with theory simplification. `rlsipw` is off by default which leads to the following behavior:

```
rlsimpl(a<>0 and (a>=0 or b=0));
     ⇒ a <> 0 and (a > 0 or b = 0)
```

This meets the simplification goal of small satisfaction sets for the atomic formulas leading, e.g., to the following behavior:

```
rlsimpl(a<>0 and (a>0 or b=0));
     ⇒ a <> 0 and (a >= 0 or b = 0)
```

rlsipo Switch

Simplification prefer orderings. Prefers orderings in contrast to inequations with theory simplification. `rlsipo` is on by default which leads to the following behavior:

```
rlsimpl(a>=0 and (a<>0 or b=0));
     ⇒ a >= 0 and (a > 0 or b = 0)
```

This meets the simplification goal of small satisfaction sets for the atomic formulas. Turning it on leads, e.g., to the following behavior:

```
rlsimpl(a>=0 and (a>0 or b=0));
     ⇒ a >= 0 and (a <> 0 or b = 0)
```

Here, we meet the simplification goal of convenient relations when orderings are considered inconvenient.

rlsism Fix Switch

Simplify smart. By default this switch is on. See the description of the function `rlsimpl` for its effects.

```
rlsimpl(x>0 and x+1<0);
     ⇒ false
```

rlsichk Fix Switch

Simplify check. By default this switch is on enabling checking for equal sibling subformulas:

```
rlsimpl((x>0 and x-1<0) or (x>0 and x-1<0));
     ⇒ (x>0 and x-1<0)
```

rlsiidem Fix Switch

Simplify idempotent. By default this switch is on. It is relevant only with switch `rlsism` on. Its effect is that `rlsimpl` is idempotent, i.e., an application of `rlsimpl` to an already simplified formula yields the formula itself.

rlsiso Fix Switch

Simplify sort. By default this switch is on. It toggles the sorting of the atomic formulas on the single levels.

```
rlsimpl((a=0 and b=0) or (b=0 and a=0));
     ⇒ a = 0 and b = 0
```

3.2 Tableau Simplifier

Although our deep simplifier already combines information located on different Boolean levels, it preserves the basic Boolean structure of the formula. The tableau methods, in contrast, provide a technique for changing the Boolean structure of a formula by constructing case distinctions. Compared to the standard simplifier they are much slower.

cdl Data Structure

Case distinction list is a list of atomic formulas considered disjunctively.

rltab *formula cdl* Function

Tableau method. The result is a tableau wrt. *cdl*, i.e., a simplified equivalent of the disjunction over the specializations wrt. all atomic formulas in *cdl*.

rlatab *formula* Function

Automatic tableau method. Tableau steps wrt. a case distinction over the signs of all terms occurring in *formula* are computed

and the best result, i.e., the result with the minimal number of atomic formulas is returned.

rlitab *formula* Function

Iterative automatic tableau method. *formula* is simplified by iterative applications of `rlatab`. The exact procedure depends on the switch `rltabib`.

rltabib Switch

Tableau iterate branch-wise. By default this switch is on. It controls the procedure `rlitab`. If `rltabib` is off, `rlatab` is iteratively applied to the argument formula as long as shorter results can be obtained. In case `rltabib` is on, the respective next tableau step is not applied to the last tableau result but to each branch. The iteration stops when the obtained formula is not smaller than the respective input.

3.3 Groebner Simplifier

The Groebner simplifier is a special simplifier of the OFSF package. It considers order theoretical and algeraic simplification rules between the atomic formulas of the input formula. Currently the Groebner simplifier is not idempotent. The name is derived from the main mathematical instrument used for simplification: Computing Groebner bases of certain subsets of terms occurring in the atomic formulas.

For calling the Groebner simplifier there are the following functions:

rlgsc *formula* [*theory*] Function
rlgsd *formula* [*theory*] Function
rlgsn *formula* [*theory*] Function

Groebner simplifier. *formula* is a quantifier-free formula. Default for the optional argument is the empty theory `{}`. These commands to invoke the Groebner simplifier differ only in the Boolean normal form which is computed at the beginning. `rlgsc` computes a conjunctive normal form, `rlgsd` a disjunctive normal form, and `rlgsn` computes either a conjunctive or a disjunctive normal form in dependency on the structure of *formula*. After computing the respective normal form the formula is simplified using Groebner simplification techniques. The returned formula is equivalent to the input formula under the *theory*.

```
rlgsd(x=0 and ((y = 0 and x**2+2*y > 0) or
        (z=0 and x**3+z >= 0)));
    ⇒ x = 0 and z = 0

rlgsc(x neq 0 or ((y neq 0 or x**2+2*x*y <= 0) and
        (z neq 0 or x**3+z < 0)));
     ⇒ x <> 0 or z <> 0
```

As described above, the Groebner simplifiers `rlgsc` and `rlgsd` differ in the normal form computation at the beginning of the simplification. For flat formulas the results of both are equal. In general, it is hard to decide, which normal form, conjunctive or disjunctive, is smaller. Moreover, the simplification of the larger normal form can lead to a smaller final result than that of the smaller one.

One possible approach is to consider the top level operator of the formula:

- Use `rlgsc` for formulas with top level operator `and`.
- Use `rlgsd` for formulas with top level operator `or`.

The procedure `rlgsn` implements this heuristic.

The Groebner simplifier uses the Groebner package of REDUCE to compute the various Groebner bases. By default, the `revgradlex` term order is used and no optimizations of the ordering between the variables are applied. The other switches and variables of the Groebner package are not controlled by the Groebner simplifier and may thus be adjusted by the user.

In contrast to the standard simplifier `rlsimpl`, the Groebner simplifiers can produce formulas with more atomic formulas than the input. This, however, cannot happen if the switches `rlgsprod`, `rlgsred`, and `rlgssub` are off and the input formula is a simplified Boolean normal form.

The functionality of the Groebner simplifiers `rlgsc`, `rlgsd`, and `rlgsn` is controlled by numerous switches. In most cases the default settings lead to a good simplification.

rlgsrad Switch

Groebner simplifier radical membership test. By default this switch is on. If the switch is on the Groebner simplifier does not only use ideal membership tests for simplification but also radical membership tests. This leads to better simplifications but needs considerably more time.

rlgssub Switch

Groebner simplifier substitute. By default this switch is on. Certain subsets of atomic formulas are substituted by equivalent ones. Both the number of atomic formulas and the complexity of the terms may increase or decrease independently.

rlgsbnf Switch

Groebner simplifier Boolean normal form. By default this switch is on. Then the simplification starts with a Boolean normal form computation. If the switch is off the simplifiers expect a Boolean normal form as the argument *formula*.

rlgsvb Switch

Groebner simplifier verbose. By default this switch is on. It toggles verbosity output of the Groebner simplifier. Verbosity output is given if and only if both *rlverbose* and *rlgsvb* are on.

rlgsprod Advanced Switch

Groebner simplifier product. By default this switch is off. If this switch is on then conjunctions of inequalities and disjunctions of equations are contracted multiplicatively to one atomic formula. This reduces the number of atomic formulas but in most cases it raises the complexity of the terms. Anyway, most simplifications recognized by considering products are detected even if `rlgsprod` is off.

rlgsred Switch

Groebner simplifier reduce polynomials. By default this switch is on. It controls the reduction of the terms wrt. the computed Groebner bases. The number of atomic formulas is never increased. Mind that by reduction the terms can become more complicated.

rlgserf Advanced Switch

Groebner simplifier evaluate reduced form. By default this switch is on. It controls the evaluation of the atomic formulas to truth values. If this switch is on, the standard simplifier is used to evaluate atomic formulas. Otherwise atomic formulas are only evaluated if their left hand side is a domain element.

rlgsutord Advanced Switch

Groebner simplifier user defined term order. By default this switch is off. Then all Groebner basis computations and reductions are performed with respect to the `revgradlex` term order. If this switch is on the Groebner simplifier minds the term order selected with the `torder` statement. For passing a variable list to `torder`, note that `rlgsradmemv!*` is used as the tag variable for radical membership tests.

4 Normal Forms

4.1 Boolean Normal Forms

rlcnf *formula* Function

Conjunctive normal form. *formula* is a quantifier-free formula. Returns a conjunctive normal form of *formula*.

```
rlcnf(a = 0 and b = 0 or b = 0 and c = 0);
    ⇒ (a = 0 or c = 0) and b = 0
```

rldnf *formula* Function

Disjunctive normal form. *formula* is a quantifier-free formula. Returns a disjunctive normal form of *formula*.

```
rldnf((a = 0 or b = 0) and (b = 0 or c = 0));
    ⇒ (a = 0 and c = 0) or b = 0
```

rlbnfsac Fix Switch

Boolean normal forms subsumption and cut. By default this switch is on. With Boolean normal form computation, subsumption and cut strategies are applied to decrease the number of clauses. We give an example:

```
rldnf(x=0 and y<0 or x=0 and y>0 or x=0 and y<>0 and z=0);
    ⇒ (x = 0 and y <> 0)
```

rlbnfsm Switch

Smart Boolean normal form. By default this switch is off. If on, simplifier recognized implication is applied with CNF and DNF computations. This leads to smaller normal forms but is considerably time consuming.

4.2 Other Normal Forms

rlnnf *formula* Function

Negation normal form. Returns a *negation normal form* of *formula*. This is an `and-or`-combination of either atomic or negated atomic formulas. This function also copes with quantifiers. In the OFSF case the result does not even contain negated atomic formulas since the negations can be encoded by relations. We give an example:

```
rlnnf(a = 0 equiv b > 0);
    ⇒ (a = 0 and b > 0) or (a <> 0 and b <= 0)
```

rlpnf *formula* Function

Prenex normal form. Returns a prenex normal form of *formula*. The number of quantifier changes in the result is minimal among all prenex normal forms that can be obtained from *formula* by only moving quantifiers to the outside.

When *formula* contains two quantifiers with the same variable such as in

$$\exists x(x = 0) \wedge \exists x(x \neq 0)$$

there occurs a name conflict. It is resolved according to the following rules:

- Every bound variable that stands in conflict with any other variable is renamed.
- Free variables are never renamed.

Hence `rlpnf` applied to the above example formula yields

```
rlpnf(ex(x,x=0) and ex(x,x<>0));
    ⇒ ex x0 ex x1 (x0 = 0 and x1 <> 0)
```

rlapnf *formula* Function

Anti-prenex normal form. Returns a positive formula equivalent to *formula* where all quantifiers are moved to the inside as far as possible.

```
rlapnf ex(x,all(y,x=0 or (y=0 and x=z)));
    ⇒ ex x (x = 0) or (all y (y = 0) and ex x (x - z = 0))
```

5 Quantifier Elimination and Variants

REDLOG uses elimination set techniques for quantifier elimination. Elimination sets have been introduced in [Wei88]. Our implementation is based on the description of the linear case in [LW93] and that of the quadratic case in [Wei] (Chapter 7 [References], page 25). It includes, however, numerous sophisticated optimizations.

5.1 Quantifier Elimination

rlqe *formula* [*theory*] Function

Quantifier elimination. Returns a (hopefully) quantifier-free equivalent of *formula* (wrt. *theory*) where *formulas* has to obey certain degree restrictions: The degree of all terms in the quantified variables must be at most 2. Moreover, with the elimination of one quantifier, the degree in the other quantified variables can increase. In general, it cannot be determined by inspection of the input *formula* whether the elimination will succeed. There are various tricks for decreasing the degree of the input and of intermediate results built in. In case that not all variables can be eliminated, the resulting formula ist not quantifier-free but still equivalent.

elimination_answer Data Structure

A list of *condition–solution pairs*, i.e., a list of pairs consisting of a quantifier-free formula and a set of equations.

rlqea *formula* [*theory*] Function

Quantifier elimination with answer. Returns an *elimination_answer* obtained the following way: w.l.o.g., the `rlqea` argument is prenex. All quantifier blocks but the outermost one are eliminated. For the latter the constructive information obtained by the elimination is saved:

- In case the considered block is existential, for each evaluation of the free variables we know the following: Whenever a *condition* holds, the input formula is true under the given evaluation, and the *solution* is *one* possible evaluation for the outer block variables satisfying the matrix.
- The universally quantified case is dual: Whenever a *condition* is false, the input formula is false and the *solution* is *one* possible counterexample.

As an example we show how to find conditions and solutions for a system of two linear formulas:

```
rlqea ex(x,x+b1>=0 and a2*x+b2<=0);
           2                                          - b2
    ⇒ {{a2 *b1 - a2*b2 >= 0 and a2 <> 0,{x = -------}},
                                                       a2
        {a2 < 0 or (a2 = 0 and b2 <= 0),{x = infinity1}}}
```

The answer can contain new constants named either `infinity` or `epsilon`, followed by a number: All `infinity`'s are positive and infinite, and all `epsilon`'s are positive and infinitesimal wrt. the considered field. Nothing is known about the ordering among the `infinity`'s and `epsilon`'s though this can be relevant for the points to be solutions. With the switch `rounded` on, the `epsilon`'s are evaluated to zero.

rlqedfs Switch

Quantifier elimination depth first search. By default this switch is off. It is ignored with the switch `rlqeheu` on, which is the default. Turning `rlqedfs` on makes `rlqe` work in a depth first search manner instead of breadth first search. This saves space, and with decision problems it might save time. In general, it leads to larger results.

rlqeheu Switch

Quantifier elimination search heuristic. By default this switch is on, which causes the switch `rlqedfs` to be ignored. `rlqe` will decide between BFS (breadth first search) and DFS (depth first search) for each quantifier block where DFS is chosen when the problem is a *decision problem*.

rlqepnf Fix Switch

Quantifier elimination make prenex normal form. By default this switch is on, which causes that `rlpnf` is applied to *formula* before starting the elimination process. If the `rlqe` argument is already prenex, this switch can be turned off. This may be in particular useful with formulas containing `equiv` since currently `rlpnf` applies `rlnnf`, and resolving equivalences can double the size of a formula.

rlqesr Advanced Switch

Quantifier elimination separate roots. This is off by default. It affects the technique for substituting two solutions of a quadratic constraint during elimination.

The following two functions `rlqeipo` and `rlqews` are experimental implementations. The idea there is to overcome the obvious disadvantages of prenex normal forms with elimination set methods. In most cases, however, the classical method `rlqe` has turned out superior.

rlqeipo *formula* [*theory*] Function

Quantifier elimination in position. Returns a quantifier-free equivalent to *formula* by iteratively making *formula* anti-prenex and eliminating one quantifier.

rlqews *formula* [*theory*] Function

Quantifier elimination with selection. *formula* has to be prenex, if the switch `rlqepnf` is off. Returns a quantifier-free equivalent to *formula* by iteratively selecting a quantifier from the innermost block, moving it inside as far as possible, and eliminating it.

5.2 Generic Quantifier Elimination

The following variant of `rlqe` enlarges the theory by disequations, i.e., `<>`-atomic formulas, wherever this simplifies the quantifier elimination. For geometric problems it has turned out that in most cases the additional assumptions made are actually *non-degeneracy conditions*. The method has been described in detail in [DSW96], see Chapter 7 [References], page 25. It has also turned out useful for physical problems such as network analysis, see [Stu96] (Chapter 7 [References], page 25).

rlgqe *formula* [*theory*] Function

Generic quantifier elimination. *formula* contains either only existential or only universal quantifiers. Returns a list `{th,f}` such that `th` is a superset of *theory* adding only disequations, and `f` is a (hopefully) quantifier-free formula equivalent to *formula* assuming `th`. For restrictions and options compare `rlqe` (see Chapter 5 [Quantifier Elimination and Variants], page 18).

rlgqea *formula* [*theory*] Function

Generic quantifier elimination with answer. *formula* contains either only existential or only universal quantifiers. Returns a list consisting of a theory and an *elimination_answer*. Compare `rlqea` (see Chapter 5 [Quantifier Elimination and Variants], page 18) and `rlgqe` (see Chapter 5 [Quantifier Elimination and Variants], page 18).

After applying generic quantifier elimination the user might feel that the result is still too large while the theory is still quite weak. There is a function `rlgentheo` which simplifies a formula by making further assumptions (see Chapter 6 [Miscellaneous], page 22).

rlqegenct Switch

Quantifier elimination generate complex theory. This is on by default, which allows to assume inequalities over non-monomial terms with the generic quantifier elimination.

5.3 Linear Optimization

There is an optimization method implemented, which uses the built-in quantifier elimination, encoding the target function by an additional constraint including a dummy variable. This optimization technique has been described in [Wei94a], see Chapter 7 [References], page 25.

rlopt *constraints target* Function

Linear optimization. *constraints* is a list of parameter-free atomic formulas built up with `=`, `<=`, or `>=`; *target* is a polynomial over the rationals. *target* is minimized subject to *constraints*. The result is either `infeasible` or a two-element list, the first entry of which is the optimal value, and the second entry is a list of points—each one given as a *substitution_list*—where *target* has this value. The point list does, however, not contain all such points. For unbound problems the result is `{-infinity,{}}`.

rlopt1s Switch

Optimization one solution. This is off by default. If on, `rlopt` returns at most one solution point.

6 Miscellaneous

6.1 Global Switches

rlsimpl Switch

Simplify. By default this switch is off. With this switch on, the function `rlsimpl` is applied at the expression evaluation stage.

rlrealtime Switch

Real time. By default this switch is off. If on it protocols the wall clock time needed for REDLOG commands in seconds. In contrast to the built-in `time` switch, the time is printed *above* the result.

rlverbose Switch

Verbose. By default this switch is off. It toggles verbosity output with some REDLOG procedures. The verbosity output itself is not documented up to now.

6.2 Analyzing Formulas

multiplicity_list Data Structure

A list of 2-element-lists containing an object and the number of its occurrences. Names of functions returning *multiplicity_lists* typically end with "ml."

rlatl *formula* Function

List of atomic formulas. Returns the set of atomic formulas contained in *formula* as a list.

rlatml *formula* Function

Multiplicity list of atomic formulas. Returns the atomic formulas contained in *formula* in a *multiplicity_list*.

rlifacl *formula* Function

List of irreducible factors. Returns the set of all irreducible factors of the nonzero terms occurring in *formula*.

```
rlifacl(x**2-1=0);
    ⇒ {x + 1,x - 1}
```

rlifacml *formula* Function

Multiplicity list of irreducible factors. Returns the set of all irreducible factors of the nonzero terms occurring in *formula* in a *multiplicity_list*.

rlterml *formula* Function

List of terms. Returns the set of all nonzero terms occurring in *formula*.

rltermml *formula* Function

Multiplicity list of terms. Returns the set of all nonzero terms occurring in *formula* in a *multiplicity_list*.

rlstruct *formula kernel* Function

Structure of a formula. Returns a list `{f,sl}`. `f` is constructed from *formula* by replacing each occurrence of a term with a kernel constructed by adding a number to *kernel*. The *substitution_list* `sl` contains all substitions to obtain *formula* from `f`.

```
rlstruct(x*y=0 and (x=0 or y>0),v);
     ⇒ {v1 = 0 and (v2 = 0 or v3 > 0),
        {v1 = x*y,v2 = x,v3 = y}}
```

rlifstruct *formula kernel* Function

Irreducible factor structure of a formula. Returns a list `{f,sl}`. `f` is constructed from *formula* by replacing each occurrence of an irreducible factor with a kernel constructed by adding a number to *kernel*. The returned *substitution_list* `sl` contains all substitions to obtain *formula* from `f`.

```
rlstruct(x*y=0 and (x=0 or y>0),v);
     ⇒ {v1*v2 = 0 and (v1 = 0 or v2 > 0),
        {v1 = x,v2 = y}}
```

6.3 Other Stuff

rlatnum *formula* Function

Number of atomic formulas. Returns the number of atomic formulas contained in *formula*. Mind that truth values are not considered atomic formulas.

rltnf *formula terml* Function

Term normal form. *terml* is a list of terms. This combines DNF computation with tableau ideas. A typical choice for *terml* is `rlterml` *formula*. If the switch `rltnft` is off `rltnf(`*formula*`,rlterml` *formula*`)` returns a DNF.

rltnft Switch

Term normal form tree variant. By default this switch is on causing `rltnf` to return a deeply nested formula.

rlgentheo *theory formula* [*exceptionlist*] Function

Generate theory. *formula* is a positive quantifier-free formula; *exceptionlist* is a list of variables empty by default. `rlgentheo` extends *theory* with inequalities not containing any variables from *exceptionlist* as long as the simplification result is better wrt. this extended theory. Returns a list {extended *theory*, simplified *formula*}.

7 References

Most of the references listed here are available on
`http://www.fmi.uni-passau.de/~redlog/`.

[DS95] Andreas Dolzmann and Thomas Sturm. Simplification of quantifier-free formulas over ordered fields. Technical Report MIP-9517, FMI, Universitaet Passau, D-94030 Passau, Germany, October 1995.

[DS96] Andreas Dolzmann and Thomas Sturm. Computer algebra meets computer logic. Technical Report MIP-9603, FMI, Universitaet Passau, D-94030 Passau, Germany, February 1996.

[DSW96] Andreas Dolzmann, Thomas Sturm, and Volker Weispfenning. A new approach for automatic theorem proving in real geometry. Technical Report MIP-9611, FMI, Universitaet Passau, D-94030 Passau, Germany, May 1996.

[LW93] Ruediger Loos and Volker Weispfenning. Applying linear quantifier elimination. *The Computer Journal*, 36(5):450–462, 1993. Special issue on computational quantifier elimination.

[Stu95] Thomas Sturm. Lineare Quantorenelimination in bewerteten Koerpern. Diploma thesis. FMI, Universitaet Passau, D-94030 Passau, Germany, February 1995.

[Stu96] Thomas Sturm. Using symbolic methods for automatic reasoning over networks. Preliminary draft, FMI, Universitaet Passau, D-94030 Passau, Germany, October 1996.

[Wei] Volker Weispfenning. Quantifier elimination for real algebra—the quadratic case and beyond. To appear in AAECC.

[Wei88] Volker Weispfenning. The complexity of linear problems in fields. *Journal of Symbolic Computation*, 5(1):3–27, February, 1988.

[Wei94a] Volker Weispfenning. Parametric linear and quadratic optimization by elimination. Technical Report MIP-9404, FMI, Universitaet Passau, D-94030 Passau, Germany, April 1994. To appear in the Journal of Symbolic Computation.

[Wei94b] Volker Weispfenning. Quantifier elimination for real algebra—the cubic case. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation in Oxford*, pages 258–263, New York, July 1994. ACM Press.

[Wei95] Volker Weispfenning. Solving parametric polynomial equations and inequalities by symbolic algorithms. Technical Report MIP-9504, FMI, Universitaet Passau, D-94030 Passau, Germany, January 1995.

[Wei96] Volker Weispfenning. Applying quantifier elimination to problems in simulation and optimization. Technical Report MIP-9607, FMI, Universitaet Passau, D-94030 Passau, Germany, April 1996. To appear in the Journal of Symbolic Computation.

Function Index

A

all 5
and 5
assoc 7

D

div 7

E

equal 6
equiv 5
ex 5

G

geq 6
greaterp 6

I

impl 5

L

length 7
leq 6
lessp 6

M

mkand 5
mkor 5

N

nassoc 7
neq 6, 7
not 5

O

or 5

P

part 7

R

repl 5
rlall 7
rlapnf 17
rlatab 12
rlatl 22
rlatml 22
rlatnum 23
rlcnf 16
rldnf 16
rlex 8
rlgentheo 24
rlgqe 20
rlgqea 20
rlgsc 13
rlgsd 13
rlgsn 13
rlifacl 22
rlifacml 22
rlifstruct 23
rlitab 13
rlmatrix 8
rlnnf 16
rlopt 21
rlpnf 17
rlqe 18
rlqea 18
rlqeipo 20
rlqews 20
rlset 3
rlsimpl 9
rlstruct 23
rltab 12
rlterml 23
rltermml 23
rltnf 23

S

sdiv 7
sub 7

Variable Index

F

false 5

R

rlbnfsac 16
rlbnfsm 16
rlbrop 5
RLDEFLANG 4
rldeflang!* 4
rlgsbnf 14
rlgserf 15
rlgsprod 15
rlgsrad 14
rlgsred 15
rlgssub 14
rlgsutord 15
rlgsvb 14
rlopt1s 21
rlqedfs 19
rlqegenct 20
rlqeheu 19
rlqepnf 19
rlqesr 19
rlrealtime 22
rlsichk 12
rlsiexpl 11
rlsiexpla 11
rlsifac 10
rlsiidem 12
rlsimpl 22
rlsipd 10
rlsipo 11
rlsipw 11
rlsism 12
rlsiso 12
rlsisqf 10
rlsitsqspl 10
rltabib 13
rltnft 23
rlverbose 22

T

true 5

Data Structure Index

C

cdl 12

E

elimination_answer 18

M

multiplicity_list 22

S

substitution_list 7

T

theory 9

Concept Index

A

answer 18, 20
anti-prenex normal form 17, 20
atomic formula list 22
atomic formula multiplicity list 22
automatic simplification 22
automatic tableau 12

B

Boolean normal form 13, 14, 16
Boolean operator 5
bracket 5
branch-wise tableau iteration 13
breadth first search 19

C

commercial version 1
conjunction 5
conjunctive normal form 16
context default 4
context selection 3
count atomic formulas 23
cut 16

D

depth first search 19
discretely valued field 6
disjunction 5
disjunctive normal form 16
divisibility 7

E

equation 6, 7
equivalence 5
existential closure 8
exploding terms 11
expression input 5, 6, 7, 8
expression output 5
extended quantifier elimination 18

F

factorization 10, 22, 23
for loop action 5
formula structure 23

G

generic quantifier elimination 20, 24
geometric problem 20
Groebner simplifier 13

I

idempotent simplification 12
implication 5
inequality 6, 7
input facilities 5, 6, 7, 8
irreducible factors list 22
irreducible factors multiplicity list 22
iterative tableau 13

L

language default 4
language selection 3
linear programming 21
list of atomic formulas 22
list of terms 23
loading redlog 3

M

matrix 8
moving quantifiers inside 17
multiple occurrences 12
multiplicity list of atomic formulas 22
multiplicity list of irreducible factors 22
multiplicity list of terms 23

N

negation 5
negation normal form 16
normal form 13, 14, 16, 17, 23
number of atomic formulas 23

O

optimization 21
ordered field 6
ordering 6

P

parity decomposition 10
prefer orderings 11
prefer weak orderings 11
prenex normal form 17, 19, 20
protocol 15, 22

Q

quantifier 5, 7, 8
quantifier elimination 18, 19, 20

R

radical membership test 14
real closed field 6
real time 22
reduction 13, 15
removing quantifiers 8
replication 5

S

scientific version 1
simplification 9, 10, 11, 12, 13, 24
simplifier recognized implication 16
smart bnf computation 16
smart simplification 12
solution points 18
sorting atomic formulas 12
split trivial square sum 10
square-free decomposition 10
standard simplifier 9
starting redlog 3
strict divisibility 7
substitution 7
subsumption 16

T

tableau 12, 13
term list 23
term multiplicity list 23
term normal form 23
term order 15
time 22
trivial square sum 10
truth value 5

U

universal closure 7

V

verbosity output 15, 22

W

weak divisibility 7