GiNaCRA: A C++ Library for Real Algebraic Computations*

Ulrich Loup and Erika Ábrahám

RWTH Aachen University, Germany

Abstract. We present the growing C++ library GiNaCRA, which provides efficient and easy-to-integrate data structures and methods for *real algebra*. It is based on the C++ library GiNaC, supporting the symbolic representation and manipulation of polynomials. In contrast to other similar tools, our *open source* library aids *exact*, real algebraic computations based on an appropriate *data type representing real zeros* of polynomials. The only non-standard library GiNaCRA depends on is GiNaC, which makes the installation and usage of our library simple. Our long-term goal is to integrate decision procedures for real algebra within the Satisfiability-Modulo-Theories (SMT) context and thereby provide tool support for many applied formal methods.

GiNaCRA – GiNaC Real Algebra package
http://ginacra.sourceforge.net/

1 Introduction

Formal methods for simulation, analysis, and synthesis have been making great progress during the last decades. The success of new methods in these fields often depends on efficient solvers for specific, well-established problems. For instance, there is a growing interest in Satisfiability-Modulo-Theories (SMT) solvers, implementing decision procedures for first-order logics over various theories [9]. The demand on these solvers is also growing; in particular, there is a need for more expressive logics. One example is the highly expressive but still decidable first-order logic over the reals with addition and multiplication, called *real algebra*. Whereas SMT-solvers for the linear fragment of this logic are very successful nowadays, even in industrial contexts, full real algebra still has not crossed this border. Nevertheless, several decision procedures were developed since the 1950s, which are currently operational in some computer algebra systems.

Although those computer algebra systems are frequently used and well-suited to solve a wide range of problems, they have some common drawbacks. First of all, many of them are either not *free*, or depend on non-free software. Furthermore, most of the free systems are not *open source*, restricting the extensibility and modification of the underlying algorithms. Another disadvantage is that

* The original publication is available at http://www.springerlink.com.

many computer algebra systems do not allow an *easy integration* of their functionalities into other external programs: Firstly, because they only offer graphical or textual user interfaces rather than a programming interface. Secondly, their *output* is usually a string displayed on screen, complicating its reuse in further external, exact computations.

Our long-term goal is to integrate an efficient decision procedure for real algebra into an SMT-solver. However, in view of the drawbacks mentioned above, current implementations of these decision procedures as, for example, the cylindrical algebraic decomposition (CAD) method, the virtual substitution method, or methods using Gröbner bases are not suited for an SMT-integration.

In this paper we introduce our open source `C++` library `GiNaCRA`, which is free of the above-mentioned drawbacks. Besides the standard `C++` library, `GiNaCRA` is based on a single non-standard library `GiNaC` [1]. `GiNaCRA` is under active and continuous development, and it already aids some functionalities not yet supported by any other `C++` library. For example, `GiNaCRA` provides data types for *real algebraic numbers* as well as arithmetic and relational operations on them. In addition, an algorithm finding the common real roots of a set of univariate polynomials with rational coefficients is available. These features are useful not only for SMT-solving, but in a variety of other domains for computations with real algebraic constraints. Support for finding common real roots of multivariate polynomials is being implemented at the present time. In the near future, `GiNaCRA` will be also capable of computing realizable sign conditions of a set of multivariate polynomials [10, Algorithm 13.1].

Related work. To our knowledge, there is currently no open source `C++` implementation of real algebraic numbers, able to perform exact arithmetic operations from scratch. Nevertheless, very close to at least providing real-root computations are `Libreduce`, a `C++` library of the `Reduce` computer algebra system, `CoCoALib` [8], a `C++` library of the `CoCoA` computer algebra system, `Givaro` [2], and `SYNAPS` [3]. `Singular` [12], `KANT` [4], and `PARI/GP` [15] are examples of software packages for arithmetic and algebraic computations supporting algebraic numbers, but no computations with real algebraic ones. `Maple` [5], `MATLAB` [6], and `Mathematica` [7] are prominent examples of quite a number of further computer algebra systems providing computations with polynomials. There are also several programs implementing decision procedures for real algebra. `QEPCAD` [11] is a `C++` implementation of the CAD method. Another example is the `Redlog` package [13] of the computer algebra system `Reduce`, offering an optimized combination of the virtual substitution and the CAD method.

Most of `GiNaCRA`'s algorithms are based on the textbook [10], a comprehensive guide to real algebra comprising many methods of practical importance.

2 Real algebra

Real algebra denotes the first-order logic over the reals with addition, multiplication, and the order relation. A real algebraic *formula* φ is a possibly quantified

Boolean combination of real algebraic *constraints* c. Each constraint, in turn, is an equality or inequality of polynomials in one or more variables x. The syntax can be formalized by the following abstract grammar:

$$\begin{array}{rcl} p & ::= & 0 \mid 1 \mid x \mid (p+p) \mid (p \cdot p) \\ c & ::= & p = p \mid p < p \\ \varphi & ::= & c \mid (\neg\varphi) \mid (\varphi \wedge \varphi) \mid (\exists x \varphi) \end{array}$$

Real algebra belongs to the mathematical theory of algebraic geometry, which is, to a great extent, based on Hilbert's Nullstellensatz. This field of research is the source of highly topical algorithms for solving the satisfiability problem of real algebra, that is, the question if we can assign real values to the variables of a quantifier-free real algebraic formula such that the formula evaluates to true. Solving this problem involves a lot of sophisticated computations, but they ultimately depend on a central problem: *finding real roots* of a univariate polynomial with rational coefficients. Note that there is no restriction to the degree of the polynomial. In particular, this problem can not be solved by applying a solution formula as in the quadratic case. In addition, it requires a representation of the root itself, a real algebraic number.

3 Features of `GiNaCRA`

`GiNaCRA` is a `C++` library providing a collection of basic and advanced methods for real algebraic computations. It supports different representations of real algebraic numbers (order, sign, and interval representation [14, p. 327]). The different representations can be transformed into each other, such that for each computation the most suitable one can be chosen. A wrapping class for all representation types is under current development, providing an easy way of computing with real algebraic number objects and at the same time the ability to switch to an appropriate representation efficiently. In addition, a numerical representation of a real algebraic number can be computed.

The following list comprises some more features of `GiNaCRA`.

Open Source: `GiNaCRA` is meant to be accessible by everyone: researchers, students, industrial and commercial developers. It shall be possible to enhance `GiNaCRA` by everyone as well as to use `GiNaCRA` in other non-proprietary projects. Therefore, `GiNaCRA` is licensed under the GNU Lesser General Public License version 3 (LGPLv3).

Standalone: This library depends solely on one non-standard library `GiNaC`, which is licensed under the GNU General Public License version 2 (GPLv2). In particular, `GiNaCRA` does not depend on any closed-source software. `GiNaC` is currently available in many Linux distributions innately, what makes it very easy to install `GiNaCRA`, once downloaded.

Object-oriented: Being written in `C++`, `GiNaCRA` is class-based and object-oriented. It offers classes as types for real algebraic numbers, univariate

polynomials with rational coefficients, multivariate polynomials, multivariate monomials, open intervals with rational endpoints, and many more. This increases maintainability, extensibility, and reduces error-proneness in programs using this library.

Powerful interface: We provide a clear interface for an uncomplicated and efficient communication between a `C++` application and `GiNaCRA`'s real algebraic functionality. In contrast, several state-of-the-art computer-algebra systems for real algebraic computations, such as `Reduce`, `MATLAB` or `Maple`, can not be integrated so smoothly. In addition to the library, a simple console application for testing purposes is included in the `GiNaCRA` package.

Reliable: `GiNaCRA` comes with an extensive `CppUnit` test suite, currently containing nearly 100 test cases for the various functions of the library. There are separate test classes defining test cases for any `GiNaCRA` class, providing the opportunity to enhance the testing framework easily, for example, in case of a bug being found. Moreover, we make use of prevailing, well-tested `C++` implementations, like the `C++` standard library, wherever possible.

SMT-compliant: This library was specially designed to enable the implementation of an SMT-solver for real algebra. For this purpose, existing techniques for solving real algebraic constraint systems have to be adapted to the SMT-framework. This particularly means incremental solving algorithms for real algebra, being capable of (re)storing a state during search. This functionality can be realized by means of `GiNaCRA`'s data structures, and is a content of current developments.

4 How to use `GiNaCRA`

We give three simple examples on how `GiNaCRA` can be used inside a `C++` program.

Example 1: Enumerating and refining real roots

The following example program computes the real roots of the polynomial $x^5 - 39x^4 + 574x^3 - 3954x^2 + 12673x - 15015 = -(3-x)(5-x)(7-x)(11-x)(13-x)$.

```
#include <iostream>
using namespace std;
#include <ginacra/ginacra.h>
using namespace GiNaC;

int main(int argc, char **argv)
{
  symbol x("x");
  ex e(-15015 + 12673*x - 3954*pow(x,2) + 574*pow(x,3) - 39*pow(x,4) + pow(x,5));
  RationalUnivariatePolynomial p( e, x );
  list<IntervalRepresentation> roots = IntervalRepresentation::realRoots( p );
  cout << p << " has " <<
          roots.size() << " real roots:" << endl;
  for(list<IntervalRepresentation>::const_iterator
      root = roots.begin();
      root != roots.end();
      ++root)
    cout << "  " << *root << endl;
  cout << "List of refined intervals for the roots:" << endl;
  for(list<IntervalRepresentation>::iterator
      root = roots.begin();
      root != roots.end();
      ++root)
  {
    for(register unsigned i = 0; i < 10; ++i)
```

```
      root->refine();
    cout << "  " << root->Order() << ": " << root->Interval() << endl;
  }
  return 0;
}
```

Line 9 contains the definition of the input polynomial as a GiNaC expression e. In line 10, a univariate polynomial p with rational coefficients is constructed from e and the GiNaC symbol x. After computing the real roots of p as a list of real algebraic numbers in line 11, the program outputs them in the first loop (lines 14 to 18). The second loop (lines 20 to 28) iterates through the roots again, but this time we change the iterated objects by calling the method refine() ten times, thereby gaining tighter bounds on the roots. refine() is implemented by a divide and conquer algorithm, using Sturm's theorem for real root counting. The refinement is done automatically whenever necessary for the interval representation in arithmetic or relational operations.

The listing below shows the compiler call followed by the call and the output of the example program. In the first loop, the output displays each real root as a triplet, consisting of the polynomial, an interval that contains exactly this single root, and the position of the root with respect to the order $<$. In the second loop, the polynomials are omitted in the output.

```
> g++ -lginac -lginacra -o example1 example1.cpp
> ./example1
-15015+574*x^3-3954*x^2+12673*x-39*x^4+x^5(x) has 5 real roots:
  {-15015+574*x^3-3954*x^2+12673*x-39*x^4+x^5(x): ]0, 1877/512[ : 1}
  {-15015+574*x^3-3954*x^2+12673*x-39*x^4+x^5(x): ]1877/512, 5631/1024[ : 2}
  {-15015+574*x^3-3954*x^2+12673*x-39*x^4+x^5(x): ]5631/1024, 1877/256[ : 3}
  {-15015+574*x^3-3954*x^2+12673*x-39*x^4+x^5(x): ]5631/512, 13139/1024[ : 4}
  {-15015+574*x^3-3954*x^2+12673*x-39*x^4+x^5(x): ]13139/1024, 1877/128[ : 5}
List of refined intervals for the roots:
  1: ]1571049/524288, 786463/262144[
  2: ]5242461/1048576, 2622169/524288[
  3: ]3669535/524288, 7340947/1048576[
  4: ]11534165/1048576, 5768021/524288[
  5: ]6815387/524288, 13632651/1048576[
```

Example 2: Computing common real roots of two polynomials

This example addresses the computation of the common real roots of $x^5-39x^4+574x^3-3954x^2+12673x-15015$ and $-x^4+26x^3-236x^2+886x-1155$ where $-x^4+26x^3-236x^2+886x-1155 = -(3-x)(5-x)(7-x)(11-x)$ is a factor of the polynomial of Example 1. Since the same headers are needed as in Example 1, we show only the important snippet from the main method here.

```
ex e1(-15015 + 12673*x - 3954*pow(x,2) + 574*pow(x,3) - 39*pow(x,4) + pow(x,5));
ex e2(-1155 + 886*x - 236*pow(x,2) + 26*pow(x,3) - pow(x,4));
list<RationalUnivariatePolynomial> l = list<RationalUnivariatePolynomial>();
l.push_back(RationalUnivariatePolynomial( e1, x ));
l.push_back(RationalUnivariatePolynomial( e2, x ));
list<IntervalRepresentation> roots = IntervalRepresentation::commonRealRoots( l );
cout << l.front() << " and " << l.back() << endl << "have " <<
        roots.size() << " common real roots:" << endl;
for(list<IntervalRepresentation>::const_iterator
    root = roots.begin();
    root != roots.end();
    ++root)
  cout << "  " << *root << endl;
```

A list containing the two polynomials is constructed in lines 1 to 5. In line 6, the method commonRealRoots is called with this list as input. The method returns a list of real algebraic numbers, which is displayed as follows:

```
-15015-39*x^4+x^5-3954*x^2+574*x^3+12673*x(x) and -1155-x^4-236*x^2+26*x^3+886*x(x)
have 4 common real roots:
  {1155+x^4+236*x^2-26*x^3-886*x(x): ]0, 289/64[ : 1}
  {1155+x^4+236*x^2-26*x^3-886*x(x): ]289/64, 867/128[ : 2}
  {1155+x^4+236*x^2-26*x^3-886*x(x): ]867/128, 289/32[ : 3}
  {1155+x^4+236*x^2-26*x^3-886*x(x): ]289/32, 289/16[ : 4}
```

Example 3: Real algebraic number arithmetic

Finally, we give a short code snippet illustrating arithmetic and relational operations on real algebraic numbers in `GiNaCRA`. For the sake of simplicity, we take the zeros of $x^2 - 2$ and $x - 1$ as example numbers.

```
RationalUnivariatePolynomial p1( pow(x,2)-2, x );
RationalUnivariatePolynomial p2( x-1, x );
list<IntervalRepresentation> sqrt2s = IntervalRepresentation::realRoots( p1 );
cout << "Interval representation of sqrt(2): " << sqrt2s.back() << endl;
list<IntervalRepresentation> one = IntervalRepresentation::realRoots( p2 );
cout << "Interval representation of 1: " << one.front() << endl << endl;
cout << "sqrt(2)+(-sqrt(2)) = " << sqrt2s.back()+sqrt2s.front() << endl;
IntervalRepresentation minustwo = sqrt2s.back()*sqrt2s.front();
cout << "sqrt(2)*-sqrt(2) = " << minustwo << endl;
IntervalRepresentation two = one.front()+one.front();
cout << "1+1 = " << two << endl;
cout << "(sqrt(2)*-sqrt(2)) == -(1+1)? " << ((minustwo==-two)?"Yes!":"No!") << endl;
```

This program generates the following output:

```
Interval representation of sqrt(2): {-2+x^2(x): ]0, 2.2360679774997896964[ : 2}
Interval representation of 1: {-1+x(x): ]0, 1.4142135623730950488[ : 1}

sqrt(2) + (-sqrt(2)) = {x^4-8*x^2(x): ]0, 0[ : 2}
sqrt(2) * -sqrt(2) = {16+x^4-8*x^2(x): ]-6.708203932499369089, -1/17[ : 1}
1 + 1 = {-2+x(x): ]1/3, 2.828427124746190097 5[ : 1}
(sqrt(2) * -sqrt(2)) == -(1 + 1)? Yes!
```

These arithmetic operations each produce a new interval representation by computing a new polynomial, based on the original ones, and a new interval, whose bounds depend on the original bounds. The original intervals are then refined until the new interval isolates exactly one real root of the new polynomial.

References

1. `http://www.ginac.de/`
2. `http://ljk.imag.fr/CASYS/LOGICIELS/givaro/`
3. `http://www-sop.inria.fr/galaad/logiciels/synaps/`
4. `http://www.math.tu-berlin.de/~kant/kash.html`
5. `http://www.maplesoft.com/`
6. `http://www.mathworks.de/`
7. `http://www.wolfram.com/`
8. Abbott, J., Bigatti, A.: CoCoALib: a c++ library for doing Computations in Commutative Algebra. Available at `http://cocoa.dima.unige.it/cocoalib/`
9. Barrett, C., Stump, A., Tinelli, C.: The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org` (2010)
10. Basu, S., Pollack, R., Roy, M.: Algorithms in real algebraic geometry, vol. 10. Springer-Verlag Berlin Heidelberg, 2nd edn. (2010)
11. Brown, C.W.: QEPCAD B: a program for computing with semi-algebraic sets using CADs. SIGSAM Bulletin 37(4), 97–108 (2003)

12. Decker, W., Greuel, G.M., Pfister, G., Schönemann, H.: SINGULAR 3-1-2 — A computer algebra system for polynomial computations (2010), http://www.singular.uni-kl.de
13. Dolzmann, A., Sturm, T.: REDLOG: Computer algebra meets computer logic. SIGSAM Bulletin 31(2), 2–9 (Jun 1997)
14. Mishra, B.: Algorithmic Algebra. Texts and Monographs in Computer Science, Springer, New York (1993)
15. The PARI Group, Bordeaux: PARI/GP, version 2.3.5 (2008), available at http://pari.math.u-bordeaux.fr/