Algebraic Numbers in Symbolic Computations[*]

Hans-Gert Gräbe
Institute of Computer Science
Univ. Leipzig/Germany

May 7, 1999

Abstract

There are many good reasons to teach a course on a systematic introduction to symbolic methods not only to students of mathematics but also to those of technical sciences. The design of such a course meets an essential difficulty since the principles to be demonstrated appear only in nontrivial applications in a convincing way, but there is usually no time to teach the necessary contexts to a large extent. Hence the material intended to demonstrate different aspects has to be chosen with great care.

The goal of this paper is to show that for such a purpose algebraic numbers are not only interesting for their mathematical content. Since even a small tour through algebraic numbers passes many important concepts of symbolic computation in an essential way this subject is also well suited for a final summary part of such a course.

1 Introduction

With increasing computing power desktop computers become more and more indispensable tools for the intellectual work not only of scientists but also of engineers and even technical staff. Computer programs replace more and more tables and handbooks that accompanied scientific and technical activities for a long time.

Computers can not only *access* precompiled information as before in a more rapid and flexible way but can also *generate* it from a generic knowledge base in an algorithmic way. This makes available customized information for specific applications far beyond standard solutions. It largely extends the possibilities of computer based approaches compared to print based tools where all useful information had to be compiled in advance.

The influence of such tools on the professional demands of engineers and even technical staff hardly can be overestimated. Indeed, they allow for nonstandard solutions of complex tasks where nowadays only very rigidly standardized solutions are used in practice. They will play an important role also in simulation, planning and quality management, see e.g., (Braun, 1997).

Systems that can perform symbolic computations are central for this development, since formal calculi are at the heart of any of the advanced natural and technical sciences. In the following we will concentrate on modern *general purpose* systems of that kind. They are also known as *Computer Algebra Systems* due to the origin and nature of the implemented

[*]Appeared in *Int. J. of Computer Algebra in Math. Education* **6** (1999), 65 - 84.

algorithms, see (Loos, 1983a), and will be at the core also of more specialized professional systems.

Due to the central role of symbolic methods in such knowledge representation the future scientific and technical staff must be prepared to apply these techniques intelligently to its everyday problems. Thus the demand grows to include a corresponding component of instruction in university and even possibly high school courses.

Today we observe growing efforts to incorporate computer algebra tools into regular professional courses, e.g., lectures of mathematics, physics or different engineering courses. But even if such regular instructions are (yet) missing students may access a fast growing literature to acquire these skills by themselves, and they do so more and more. Hence we may assume that in the near future students finishing their undergraduate studies will have practical experience using symbolic computer based methods.

Due to the central role of such experience for the future professional life of students graduating not only in mathematics but also in technical sciences it is desirable to continue with a *systematic* introduction into basic concepts, notions and principles behind these methods. Such a course should not concentrate on the more qualified exploitation of a *particular* Computer Algebra System but, as courses "Algorithms and Data Structures" do for classical computer applications, on the basic instruments and notations common to all or most of the different Computer Algebra Systems. Such knowledge will increase the student's expertise to exploit the full power of similar systems.

Since these students usually have a good understanding of the same questions for classical (numerical) computer applications such a course may concentrate on the main notations and principles where symbolic computations differ from numerical ones. These differences constitute a quite complex topic that starts with special design questions resulting from overlapping name and value spaces (as in the evaluation of expressions, see, e.g., (Wolfram, 1998, ch. 2.5.)), continues with differing notions of functions (see, e.g., (Wolfram, 1998, ch. 2.2.)), the unknown to classical imperative computer languages notion of simplification and ends up with subtle mathematical questions on simplifications that are allowed and those that are not (see, e.g., (Fateman, 1996) or (Aslaksen, 1996)). A better understanding of these basics usually leads also to a better understanding of the special, in some places quite unexpected, behavior of Computer Algebra Systems, and whether such a behavior may be avoided or why not.

The design of such a course meets an essential difficulty. On the one side, Computer Algebra Systems are multipurpose tools capable of very complicated symbolic computations in very complex mathematical contexts, and the principles to be demonstrated appear only in nontrivial applications in a convincing way. Such applications can't be well understood without some knowledge of their context. On the other hand, students coming from technical or computer science departments usually do not have very advanced mathematical knowledge. Thus the material intended to demonstrate different effects has to be chosen with great care.

The author teaches such a course "Introduction to Symbolic Computation" (Gräbe, 1998) to students of computer science for several years. After introductory examples it covers the following topics:

- the role of symbolic computations in science and engineering

- structure and concepts of second generation (i.e., without strong type system) Computer Algebra Systems
  - the evaluation concepts
  - lists and list manipulation
  - functions, functional symbols, and functional expressions
  - functional and rule based transformation concepts
- simplification of expressions
  - simplification, normal forms, and canonical forms
  - simplification and mathematical correctness
  - simplification of polynomial and rational expressions
  - rule based simplification of trigonometric expressions
  - the general simplification problem

The final part of the course, as a complex application, is devoted to a small tour through the basics of algebraic numbers in symbolic computations. Several notions and concepts developed earlier in the course interact on this topic in a nontrivial way. The aim of this paper is a short outline of the main stations visited during this tour.

We start with an intuitive notion of algebraic numbers as radical expressions and examine the computational strength and weakness of such an approach. It turns out that it is much harder to compute with radicals than with "ordinary" numbers. Mathematical correctness is an additional issue if we try to simplify nested radicals. All this suggests that radical expressions are closer to functional expressions than to integers or fractions. More complicated expressions containing algebraic numbers of degree 3 allow to recognize their internal structure as multivariate polynomials in various kernels and follow up the effect of different simplification operations. The general theory of roots of polynomials of degree 3 suggests that there are also algebraic numbers with a trigonometric origin. Simplification rules for trigonometric functions are involved to reduce the number of different kernels in such expressions and to identify $\cos(\frac{\pi}{n})$ as a root of a certain polynomial. This leads to the (usual from a mathematical point of view) definition of algebraic numbers as `RootOf` expressions that are symbolic expressions with a certain algebraic rewriting rule. Based on this observation we exploit the power of computer algebra to prove constructively the ring property of algebraic extensions. The proof yields additionally a simplifier for such `RootOf` expressions that turns out to be a canonical simplifier for single algebraic extensions. This proves the computational efficiency of `RootOf` expressions compared to radical expressions. We end up with a short outlook on iterated algebraic extensions.

Of course, one could imagine also other topics to be studied as the complex final subject of such a course. Why just algebraic numbers? A strong argument for such a choice is their ubiquity in symbolic computations and especially in symbolic output. Since it is often not obvious how to handle such expressions students tend to leave the symbolic context at this point. Hence a *systematic* introduction to symbolic computation should not avoid these questions.

But the question is central also for another reason: There is a great difference between the point of view of numerical computation, to which students are accustomed, and that of symbolic computation. These differences just start with a different view on numbers other than rationals. And whereas one may be willing to accept the symbolic nature of transcendental numbers as e or π it is hard to recognize the difference between the "numbers" $\sqrt{2}$ and 2/3 both containing ciphers and a certain additional symbol. But the difference is essential: symbolically the first number is a functional expression, the second one (at most) a rational expression. And the difference is well hidden: Computer Algebra Systems usually know much about square roots as, e.g.,

$$\begin{aligned} \sqrt{-6\sqrt{2}+11}+\sqrt{6\sqrt{2}+11} &= 6 \qquad (1)\\ \sqrt{-2\sqrt{6}+5}+\sqrt{2\sqrt{6}+5} &= 2\sqrt{3} \qquad (2)\\ (6\sqrt{3}-10)^{\frac{1}{3}}+(6\sqrt{3}+10)^{\frac{1}{3}} &= 2\sqrt{3} \qquad (3)\end{aligned}$$

thus conveying the impression that the arithmetic of algebraic numbers given as nested radicals is easy and completely covered by the system.

Hence a systematic introduction also to the *philosophy* of "symbolic numbers" is a topic beyond structure and concepts of Computer Algebra Systems but in the spirit of the concern of the course.

2 The intuitive notion of algebraic numbers

Mathematically algebraic numbers are defined as the roots of univariate polynomials and there are many reasons to define them even in such a way also in Computer Algebra Systems. But this is not the form in which such numbers appeared first historically nor in the students' perception. They know algebraic numbers mainly as radical expressions and one of the goals of our small tour through algebraic numbers will be the explanation of the limitations of such an understanding.

We start with such an intuitive notion of algebraic numbers and first try to understand to what extent algebraic numbers differ from usual ones, i.e., integers and fractions. This is a reasonable question since expressions like "52" or "1/4" are also merely symbolic representations. But different from integers and fractions where the Computer Algebra System (and the students) know how to obtain automatically a canonical form of the result of any arithmetic expression containing such entities, expressions like $\sqrt{2}$ are only simplified according to obvious rules as, e.g.,

$$\begin{aligned}&x := 2\sqrt{2}+3\sqrt{3}\\ &y := 3\sqrt{2}-2\sqrt{3}\\ &x+y = 5\sqrt{2}+\sqrt{3}\\ &x-y = -\sqrt{2}+5\sqrt{3}\end{aligned}$$

For more complicated expressions as, e.g.,

$$\sqrt{2}+\sqrt{3}+\frac{1}{\sqrt{2}-\sqrt{3}} \qquad (4)$$

we need additional effort to see that the expression is zero (even Maple and Derive don't find out that automatically). We conclude that algebraic numbers given as radical expressions

System	(1)	(2)	(3)
Axiom	—	—	—
Derive	automatically	automatically	automatically
Macsyma	`denest_sqrt`	`denest_sqrt`	—
Maple	automatically	automatically	—
Mathematica	`RootReduce`	`RootReduce`	`RootReduce`
MuPAD	`radsimp` or simplify(_,sqrt)	`radsimp` or simplify(_,sqrt)	`radsimp` or simplify(_,sqrt)
Reduce	—	—	—

Table 1: Simplification of nested roots

are, for some not yet visible reason, not well suited for computations, since even the zero decision problem is not (always) solved automatically.

With such a first inspection in mind it is very surprising that Computer Algebra Systems nevertheless seem to know much even about nested roots as pointed out in the introduction. Let's consider the identities (1) – (3) in more detail. Knowing the relations

$$\begin{aligned} 11 \pm 6\sqrt{2} &= (3 \pm \sqrt{2})^2, \\ 5 \pm 2\sqrt{6} &= (\sqrt{3} \pm \sqrt{2})^2, \\ 6\sqrt{3} \pm 10 &= (\sqrt{3} \pm 1)^3 \end{aligned}$$

they are evident, but the systems don't know them in advance. Table 1 collects for different Computer Algebra Systems the commands that yield the desired simplifications.

We see that most of the systems are strong in this area but don't invoke that knowledge automatically. With regard to (4) the automatic simplification of nested roots in Derive and Maple is rather obscure. But even for Axiom and Reduce, that have no such facilities built in, it is not hard to design rules to perform these simplifications (at least for expressions $\sqrt{a+\sqrt{b}}+\sqrt{a-\sqrt{b}} = c$ and $\sqrt{a+2\sqrt{b}} = \sqrt{c}+\sqrt{d}$ and integers a, b, c, d).

One may wonder about the third result in table 1 returned by Maple and Macsyma. Do they know only how to simplify square roots? The reason is another one – mathematical correctness. Recall that over the complex numbers the function $x \mapsto \sqrt[3]{x}$ is a many-valued function, and there is no canonical way to choose one of the branches. Here starts a first leap towards the algebraic understanding of algebraic numbers: Symbolic expressions may not identify the underlying algebraic objects uniquely. This ambiguity may be resolved for real numbers through branch cuts (just another long story, see, e.g., (Corless, 1996)) but there are many reasons to assume the argument and return type of radical expressions to be complex numbers. It would be nice to restrict the domain of definition by the user if possible, e.g., choosing the real branch of the root only, but I found no possibility to convince any of the systems under consideration to do that in this case. (Note that, e.g., Maple's `assume` facility does not apply to variables with "symbolic numbers" as value, but only to (true) symbols.)

3 Algebraic numbers, variables, and kernels

Let's have a closer look, how the different systems compute with algebraic numbers. Despite (4) simple square roots are not very interesting since any root of a polynomial of degree 2 may be expressed arithmetically through the polynomial's coefficients and a single root expression $\sqrt{D}$ in a well known way. So let us examine the possible ways in which the different Computer Algebra Systems compute with algebraic numbers of degree 3.

To be fair, we will use algebraic numbers produced by the systems themselves and perform some computations on these numbers. We prepared the following task:

> Take a polynomial
> $$f(x) = x^3 + p\,x + q$$
> of degree 3 with integer coefficients, compute its zeroes x_1, x_2, x_3 and subsequently compose and expand the expression
> $$(x - x_1)(x - x_2)(x - x_3)$$
> (that should simplify to the original polynomial $f(x)$).
>
> If this task is executed successfully then test whether the system can sum up powers of the roots
> $$s_k := x_1^k + x_2^k + x_3^k$$
> that are known to yield again integers.

It's a nice small programming exercise to collect appropriate commands to produce these roots, extract and recombine the correct parts from these expressions in the desired way, and test different simplification routines on these or related expressions (e.g., $(x-x_1^2)(x-x_2^2)(x-x_3^2)$).

Derive (version 3.04) returns as the result of the first part of this task the following answer:

```
1: x^3+x+1

2: factor complex #1: (0.4 s)
```

$$
\begin{aligned}
&\left(x + \left(\tfrac{1}{18}\sqrt{93}+\tfrac{1}{2}\right)^{1/3} - \left(\tfrac{1}{18}\sqrt{93}-\tfrac{1}{2}\right)^{1/3}\right)\\
&\quad\cdot\Biggl(x + \left(\tfrac{1}{144}\sqrt{93}-\tfrac{1}{16}\right)^{1/3} - \left(\tfrac{1}{144}\sqrt{93}+\tfrac{1}{16}\right)^{1/3}\\
&\qquad\quad -i\left(\left(\tfrac{3}{128}\sqrt{93}+\tfrac{29}{128}\right)^{1/6} + \left(\tfrac{29}{128}-\tfrac{3}{128}\sqrt{93}\right)^{1/6}\right)\Biggr)\\
&\quad\cdot\Biggl(x + \left(\tfrac{1}{144}\sqrt{93}-\tfrac{1}{16}\right)^{1/3} - \left(\tfrac{1}{144}\sqrt{93}+\tfrac{1}{16}\right)^{1/3}\\
&\qquad\quad +i\left(\left(\tfrac{3}{128}\sqrt{93}+\tfrac{29}{128}\right)^{1/6} + \left(\tfrac{29}{128}-\tfrac{3}{128}\sqrt{93}\right)^{1/6}\right)\Biggr)
\end{aligned}
$$

```
3: expand #2 (1.3 s)
```

$$
\begin{aligned}
&x^3 + x\left(\tfrac{1}{1152}\sqrt{93}+\tfrac{29}{3456}\right)^{1/3} + x\left(\tfrac{29}{3456}-\tfrac{1}{1152}\sqrt{93}\right)^{1/3} + x\left(\tfrac{3}{128}\sqrt{93}+\tfrac{29}{128}\right)^{1/3}\\
&\quad +x\left(\tfrac{29}{128}-\tfrac{3}{128}\sqrt{93}\right)^{1/3} - x\left(\tfrac{1}{18}\sqrt{93}+\tfrac{29}{54}\right)^{1/3} - x\left(\tfrac{29}{54}-\tfrac{1}{18}\sqrt{93}\right)^{1/3}\\
&\quad +x + \left(\tfrac{7}{7776}\sqrt{93}+\tfrac{5}{576}\right)^{1/3} + \left(\tfrac{1}{31104}\sqrt{93}-\tfrac{1}{3456}\right)^{1/3} + \left(\tfrac{7}{288}\sqrt{93}+\tfrac{15}{64}\right)^{1/3}\\
&\quad + \left(\tfrac{1}{1152}\sqrt{93}-\tfrac{1}{128}\right)^{1/3} + \left(\tfrac{1}{486}\sqrt{93}+\tfrac{1}{54}\right)^{1/3} - \left(\tfrac{1}{31104}\sqrt{93}+\tfrac{1}{3456}\right)^{1/3}\\
&\quad - \left(\tfrac{7}{7776}\sqrt{93}-\tfrac{5}{576}\right)^{1/3} - \left(\tfrac{1}{1152}\sqrt{93}+\tfrac{1}{128}\right)^{1/3} - \left(\tfrac{7}{288}\sqrt{93}-\tfrac{15}{64}\right)^{1/3}\\
&\quad - \left(\tfrac{1}{486}\sqrt{93}-\tfrac{1}{54}\right)^{1/3}
\end{aligned}
$$

Even another call to `simplify` doesn't change the result. One may wonder about this, but there is a simple explanation that the students may guess with their knowledge about the simplification mechanism of Computer Algebra Systems obtained in an earlier part of the course. Nested radical expressions, such as $(U)^{1/3}$, are (conceptually) internally represented as `Power(U,1/3)` and thus much closer to functional expressions like $\sin(2\,x-y)$ or $\cos(x^2+y^2)$ than to numbers. Hence the general simplification mechanism for functional expressions applies to them (and may be studied on this target in more detail).

For efficiency reasons this general simplification mechanism usually consists of a (efficiently decidable) polynomial or rational simplifier combined with a (in full generality not decidable) rule based simplification system associated with functional symbols. A functional expression is simplified at three layers: The outer layer is a polynomial simplification of the context that regards the expression as a generalized variable, called *kernel*, the inner layer a polynomial simplification of the arguments of the expression and the middle one a rule based simplification combining the inner and the outer world otherwise separated by the function name as by a "wall".

Hence expression `#3`, which contains several root symbols, is internally not represented as a polynomial in x with number coefficients, but as a (multivariate) polynomial in several kernels. Different to polynomials in indeterminates, these kernels are algebraically (and here even linearly) dependent. Indeed, a more detailed inspection "by hand" shows that all these kernels are merely different multiples of two different third roots. Probably Derive "forgot" this common origin and considers all the roots as independent. Note that, once forgotten, these dependencies can't be recovered if radical expressions are considered as multi-valued.

We conclude that it is very important to keep the number of different but algebraically dependent kernels as small as possible, e.g., keeping track of common subexpressions.

The common origin of all these summands is also evident from *Cardano's formula* for the zeroes of the reduced cubic polynomial $x^3 + p\,x + q$

$$x = \sqrt[3]{-\frac{q}{2} + \sqrt{D}} + \sqrt[3]{-\frac{q}{2} - \sqrt{D}} \qquad \text{with } D = \left(\frac{q}{2}\right)^2 + \left(\frac{p}{3}\right)^3$$

The other systems take into consideration this rule. Consider, for example, Maple (version V.5):

```
s:=[solve(x^3+x+1,x)];
```

$$\begin{aligned}&[-1/6\,\%1 + 2\,\%2,\ 1/12\,\%1 - \tfrac{1}{\%1} + 1/2\,I\sqrt{3}\,(-1/6\,\%1 - 2\,\%2)\,,\\ &\quad 1/12\,\%1 - \tfrac{1}{\%1} - 1/2\,I\sqrt{3}\,(-1/6\,\%1 - 2\,\%2)]\end{aligned}$$

$$\begin{aligned}\%1 &:= \sqrt[3]{108 + 12\sqrt{93}}\\ \%2 &:= \frac{1}{\sqrt[3]{108 + 12\sqrt{93}}}\end{aligned}$$

The result was formulated using new variables %1 and %2 that refer to a common subexpression (and its inverse) occuring in different places of the formula. Now we apply selectors and constructors of the Computer Algebra System language to express the next step of our task:

```
product(x-op(i,s),i=1..3);
```

$$\begin{aligned}&(x + 1/6\,\%1 - 2\,\%2)\left(x - 1/12\,\%1 + \%2 - 1/2\,I\,\sqrt{3}\,(-1/6\,\%1 - 2\,\%2)\right)\\ &\quad\left(x - 1/12\,\%1 + \%2 + 1/2\,I\,\sqrt{3}\,(-1/6\,\%1 - 2\,\%2)\right)\end{aligned}$$

```
p:=expand(%);
```

$$1/2 + x + x^3 + \frac{1}{18}\sqrt{93} - \frac{8}{108 + 12\sqrt{93}}$$

At this point we may remember that `expand` is a polynomial normal form operator of the outer simplification layer, not involving simplifications of algebraic numbers that are part of the middle simplification layer. The result is the same for an expression in independent variables x and $u = \%1$. We may prove this:

```
s1:=subs(%1=u,%2=1/u,s);
```

$$\begin{aligned}s1 := &[-1/6\,u + 2\,u^{-1}, 1/12\,u - u^{-1} + 1/2\,I\,\sqrt{3}\,(-1/6\,u - 2\,u^{-1})\,,\\ &1/12\,u - u^{-1} - 1/2\,I\,\sqrt{3}\,(-1/6\,u - 2\,u^{-1})]\end{aligned}$$

```
expand(product(x-op(i,s1),i=1..3));
```

$$x+x^3+\frac{1}{216}\,u^3-8\,u^{-3}$$

A subsequent call of `simplify` finally involves the middle simplification layer and yields the original polynomial:

```
simplify(p);
```

$$x^3+x+1$$

For the sums of powers we get

```
a:=[seq(simplify(sum(s[i]^k,i=1 .. 3)),k=2 .. 9)];
```

$$[-2,-3,2,5,6\,\frac{29+3\,\sqrt{3}\sqrt{31}}{\left(9+\sqrt{3}\sqrt{31}\right)^2},-7,-36\,\frac{29+3\,\sqrt{3}\sqrt{31}}{\left(9+\sqrt{3}\sqrt{31}\right)^2},144\,\frac{135+14\,\sqrt{3}\sqrt{31}}{\left(9+\sqrt{3}\sqrt{31}\right)^3}]$$

For some reason not all expressions are fully simplified. Note that Maple, as most of the other Computer Algebra Systems, does not incorporate the full power of algebraic simplifications into `simplify` but has a special algebraic evaluation operator `evala` that involves stronger and in some cases computationally more expensive knowledge about algebraic numbers.

```
map(evala,a);
                       [-2, -3, 2, 5, 1, -7, -6, 6]
```

MuPAD (version 1.3.1) works in a similar fashion:

```
s:=solve(x^3+x+1,x);
s1:=_mult(x-op(s,i) $i=1..3);
expand(s1);
```

$$x^3+3x\sqrt[3]{\left(\frac{\sqrt{31}\sqrt{108}}{108}+1/2\right)}\sqrt[3]{\left(\frac{\sqrt{31}\sqrt{108}}{108}-1/2\right)}+1$$

```
simplify(%);
```

$$x^3+3x\sqrt[3]{\left(\frac{\sqrt{93}}{18}+1/2\right)}\sqrt[3]{\left(\frac{\sqrt{93}}{18}-1/2\right)}+1$$

One may wonder that the product of roots is not simplified. The reason is the same as explained above: If considered as independent, it is not clear which branches have to be combined to get the mathematically correct result. (Note that this changed in version 1.4 thus weakening mathematical correctness.) Probably, MuPAD implemented Cardano's formula $x=u+v$ with $u,v=\sqrt[3]{-\frac{q}{2}\pm\sqrt{D}}$ "as it is" and did not remember, that both roots u,v are related through the relation $u\,v=-\frac{p}{3}$.

For the power sum simplification, this does not seem to be important:

```
[radsimp(_plus(op(s,i)^k $ i=1 .. 3)) $ k=2 .. 9];

                    [-2, -3, 2, 5, 1, -7, -6, 6]
```

Axiom, Macsyma, Mathematica, and Reduce solve this task in a manner similar to Maple.

4 Algebraic numbers not represented as radical expressions

It is time to convince the students that there exist natural contexts where algebraic numbers should not be presented as radical expressions. For this purpose we discuss (in the course, not in this paper; for a nice explanation see (Pieper, 1988)) how to solve polynomial equations of degree 3 theoretically. The *casus irreducibilis* that leads to 3 real roots deserves special attention, since these roots usually are represented in *trigonometric form*, and Reduce, MuPAD and Derive do so.

```
s:=solve(x^3-3*x+1,x); # MuPAD
```

$$\left\{2\cos\left(\frac{2\pi}{9}\right), -\cos\left(\frac{2\pi}{9}\right) + \sqrt{3}\sin\left(\frac{2\pi}{9}\right), -\cos\left(\frac{2\pi}{9}\right) - \sqrt{3}\sin\left(\frac{2\pi}{9}\right)\right\}$$

Needless to say, all Computer Algebra Systems under consideration (currently) don't simplify expressions containing such numbers automatically and hence the above task fails on that example. (Nevertheless Maple and Mathematica are capable of a *guided* simplification of such expressions through conversion to exponential form and expansion, followed by algebraic simplification.) This is probably the main reason why Maple, Mathematica, Macsyma, and Axiom use Cardano's formula also for negative discriminants.

The students' knowledge about the values of trigonometric functions at the special arguments $\frac{\pi}{3}$, $\frac{\pi}{4}$ and $\frac{\pi}{6}$ shows that special values of trigonometric functions may be expressed through radicals. Computer Algebra Systems usually know more such examples, e.g., radical expressions for $\cos(\frac{\pi}{5})$ and $\sin(\frac{\pi}{5})$, but have difficulties with $\cos(\frac{\pi}{n})$ for $n > 6$. On the other hand, they are powerful enough easily to play around with such examples. So we can try to simplify expressions of the form

```
pi:=Pi;    # for Maple
u0:=cos(pi/5)*cos(2*pi/5);
u1:=cos(pi/7)*cos(2*pi/7)*cos(3*pi/7);
u2:=cos(pi/9)*cos(2*pi/9)*cos(4*pi/9);
```

u_0 simplifies with Maple and little effort to $\frac{1}{4}$. For u_1 and u_2 numeric approximations show that these numbers are very close to $\frac{1}{8}$ but it is impossible or at least hard (depending on the Computer Algebra System) to prove that they are in fact *equal* to that number.

At this point the trigonometric simplification rules, developed earlier in the course, may be applied to transform such expressions into polynomial expressions with a single kernel. This ends up with the equivalent problem (for $n = 7$ or $n = 9$) if $\cos(\frac{\pi}{n})$ is a root of a certain polynomial. If n is odd such a polynomial $p_n(y)$ may be obtained from the expansion of $\cos(n\,x) + 1$ as a polynomial in $y = \cos(x)$ since for $x = \frac{\pi}{n}$ we get $\cos(n\,x) + 1 = 0$. Note that for odd primes n this polynomial factors as

$$p_n(y) = (y + 1)\,q_n(y)^2$$

for a certain irreducible polynomial $q_n(y)$. Hence $\cos(\frac{\pi}{n})$ is an algebraic number of degree $\frac{n-1}{2}$ for such n.

This suggests the (usual from a mathematical point of view) definition of an algebraic number as a root of a certain polynomial. It is also a good motivation to prove some properties of such a presentation. In particular, we may easily extract the important additional information that such a description doesn't characterize the corresponding algebraic number uniquely. Indeed, the equation $p_n(y) = 0$ has obviously the zeroes $y_k = \cos(\frac{(2k+1)\pi}{n})$ for $k = 1, \ldots, \frac{n-1}{2}$.

5 Computing with algebraic numbers

With the definition given at the end of the last section we arrive at true symbolic objects that may represent several algebraic numbers (with the same defining polynomial). As long as we are not interested in the interaction between roots of the same polynomial $p(x)$, a single symbolic object `RootOf(p(x),x)` may serve as representative for *any* of its roots. All these roots will be treated uniformly by the simplifier applying the algebraic rewriting rule extracted from $p(x)$. Since the students learned earlier in the course how to work with symbols and rewriting rules we may do the corresponding calculations "by hand".

This allows us to illustrate some usually difficult for beginners questions in the arithmetic of algebraic numbers. Consider the statement that the sum of algebraic numbers is again an algebraic number and the crucial point in the proof to construct a corresponding defining polynomial. Explanations based on computer supported (symbolic) "hand" calculations may give much more evidence of the driving principles. Consider, e.g., the algebraic numbers $a = \sqrt{2}$ and $b = \sqrt[3]{5}$. For calculations "by hand" we will represent them as symbols a and b with rewriting rules $\{a^2 => 2, b^3 => 5\}$. Applying (with Reduce) these algebraic rewriting rules to the powers of their sum $c = a + b$

```
for k:=0:10 collect (a+b)^k where { a^2 => 2, b^3 => 5 };
```

we see that all such powers may be expressed as linear combinations of the six products $a^i b^j$ with $i = 0, 1,\ j = 0, 1, 2$. Hence we may expect the seven powers $c^i,\ i = 0, \ldots, 6$ to be linearly dependent. Such a dependency relation may easily be found from a generic polynomial $p(x)$ of degree 6 by solving a system of linear equations. Here is the corresponding Reduce code:

```
p:=x^6+for k:=0:5 sum mkid(c,k)*x^k;
p1:=(sub(x=a+b,p) where { a^2 => 2, b^3 => 5 } );
sys:=for each u in coeff(p1,a) join coeff(u,b);
sol:=solve(sys);
```

Hence

```
q:=sub(sol,p);
```

is a (possibly not yet irreducible) polynomial with root c. The main advantage of such an approach is that the students can (and must) concentrate on the top level algorithmic steps and are not burdened with tedious hand calculations.

The same applies to the computation of the defining polynomial of the product and also to the computation of the inverse of an algebraic number. Finally we arrive at the following well known proposition from algebra:

> *If α is an algebraic number of degree d over a field k then the set $R := k[\alpha]$ of k-linear combinations of terms from*
>
> $$T_{red} := \{\alpha^i,\ i = 0, \ldots, d-1\}$$
>
> *is a field.*

The proof is constructive and hence R computable. Indeed, applying the rewriting rule of α extracted from its minimal polynomial, sums and products of such k-linear combinations are transformed into combinations of the same form. We leave the (slightly) more subtle argumentation for quotients as an exercise to the reader. All these computations assume, of course, that the arithmetic of k is computable, too. More precisely:

If k has (canonical) normal forms and the minimal polynomial of α is known then R has (canonical) normal forms.

This proposition proves that the representation of an algebraic number as a symbol equipped with an algebraic rewriting rule derived from the corresponding defining polynomial is well suited for computationally efficient arithmetic operations. Note that such an approach may be extended to several algebraic numbers. It introduces only a restricted number of them as (preferably independent) basic symbols and represents other algebraic numbers as their linear combinations. This method is much more efficient than the primitive element approach proposed in (Loos, 1983b) and implemented in the `arnum` package of Reduce. It has certain disadvantages for dependent basic numbers. A more detailed discussion is beyond the scope of the present paper.

We conclude that the designers of Computer Algebra Systems are well advised both to enable the user to introduce algebraic numbers in such a form and to detect and represent algebraic numbers produced by the system in even this form. Different Computer Algebra Systems meet this requirement on different levels. The latter is commonly connected with the introduction of `RootOf` symbols, but, e.g., Macsyma (version 421) doesn't even apply the obvious rewriting rule to expressions containing such symbols. Note, on the other hand, the ubiquity of `RootOf` symbols even for algebraic numbers of degree 3 and 4 in the solution of systems of polynomial equations obtained with the corresponding `solve` function. As explained above a representation of the corresponding number as radical expression may explode both with respect to size and computational complexity.

For the introduction of user defined algebraic numbers the Computer Algebra Systems provide different mechanisms. In table 2 we collected the instructions necessary to introduce an algebraic number a with defining polynomial $p(x) = x^5-x+1$ and to simplify the expression $1/(1-a^2)$ yielding a^3+a for some of the Computer Algebra System under consideration. Note that we found no way to tell Mathematica (3.0) to rationalize the denominator. `Simplify` is too weak and `RootReduce` too strong.

For more advanced computations involving algebraic numbers it would be of great benefit to hide the implementational difference between "true" and "symbolic" numbers from the user by a uniform interface. Well-known concepts from computer science to realize such polymorphism include abstract data types or object oriented methods. Both approaches lead beyond the abilities of second generation Computer Algebra Systems since they require an additional (symbolic) layer managing the type information. Although they are much more adequate from a mathematical point of view, the practical realization of a strong type system for symbolic computations meets essential difficulties, see (Comon, 1991), (Davenport, 1990) or (Fateman, 1990). The experience with strong typing obtained during the development of Axiom shows that the concepts to be developed must include advanced ideas from modern type system theory that reach far beyond the state of the art, e.g., of C++, see (Jenks/Sutor, 1992). This leads immediately to the forefront not only of computer algebra but also of computer science.

System	Version	Commands
Macsyma	421	`algebraic:true;` `tellrat(a^5-a+1);` `rat(1/(1-a^2));`
Maple	V.5	`alias(a=RootOf(x^5-x+1));` `evala(1/(1-a^2));`
Mathematica	3.0	`a=Root[x^5-x+1,1];` `[ 1/(1-a^2); ]`
Reduce	3.6	`load arnum;` `defpoly a^5-a+1;` `1/(1-a^2);`

Table 2: User defined algebraic numbers

In a systematic way such concepts are developed with Axiom (strong typing) and MuPAD (object oriented domain concept). It's neither the aim of this paper nor of a systematic introductory course in symbolic computation to report about these ongoing developments. (Davenport, 1990) remarks that a qualified exploitation of these mechanisms requires even more structural insight than for second generation Computer Algebra Systems. For example, in MuPAD we may define the field extension considered above as a new domain of computation

```
Q:=Dom::AlgebraicExtension(Dom::Rational,a^5-a+1);
```

and then put `a:=Q(a)`, assigning to the variable `a` as value the algebraic number a obtained from the symbol `a` via the domain element constructor $Q(a)$. This on a first glance difficult procedure is a good illustration of the subtle notions that arise in an object oriented approach. Finally

```
1/(1-a^2);
```

yields the desired result since the domain type of a forces the operations from Q to be called.

6 Computing in algebraic extension towers

The above proposition may be applied recursively to adjoin several algebraic numbers α_1, α_2, …, α_n to a ground field k. Such a series of algebraic extensions $k_i = k_{i-1}[\alpha_i]$ with $k_0 = k$ is an *algebraic extension tower*. We can effectively compute in such towers if we can factor polynomials in $k_i[x]$ over all intermediate extensions k_i.

The latter is essential for computations with several algebraic numbers with the same (over the ground field) defining polynomial. If α_1 and α_2 are two algebraic numbers with the common (e.g., over $k = \mathbf{Q}$) defining polynomial $p(x)$, then α_2 is a root of $p(x)$ over $k_1 = k[\alpha_1]$ but

$$p(x) = (x - \alpha_1)\, q(x)$$

for a certain polynomial $q(x) \in k_1[x]$ and α_2 is a root of this second (not necessarily irreducible) factor. For example, if a is an algebraic number with defining polynomial $p(x) = x^5 - x + 1$ then $p(x)$ factors over $\mathbf{Q}[a]$ as

$$p(x) = (x - a)\,(a^4 + x^4 + ax^3 + a^3x + a^2x^2 - 1)$$

Such a factorization may be computed with a special version of the factorization command in Macsyma, Maple, Mathematica and MuPAD. Macsyma factors only in single algebraic extensions, the other three systems support also factorization in multiple extensions (at least in principle). The algebraic factorizer implemented in the `arnum` package of Reduce may be invoked only with the `factor` switch and not with the `factorize` command and doesn't cooperate well with the remaining part of the system.

Based on its strong type system Axiom offers the most advanced factorization tool and can even compute the splitting field of $p(x)$, see (Jenks/Sutor, 1992, p. 236 ff.). But as noted it may be quite tricky to compose the correct input that initiates the desired computations. In our situation Axiom tends to convert input to the more general type `Expression Integer` rather than `AlgebraicNumber` but this is inappropriate for the factorizer.

7 Conclusions

Starting with an intuitive notion of algebraic numbers as radical expressions common to the students we study the advantages and drawbacks of such a presentation. One may argue that many effects are due to the weakness of today's Computer Algebra Systems. But even if some of the worst odds will be eliminated I'm convinced (and this is confirmed by the development of the widespread Computer Algebra Systems during the last years) that *practically important* systems will offer both sloppy but effective procedures that cover 90 % of the practically relevant cases and strong but extensive implementations for the remaining cases. Hence, if not the same examples as above, but similar examples will remain to be studied also in future releases.

Moreover, since the results of a *perfect* algebraic simplifier are rather unexpected (radical expressions in the denominator of algebraic numbers of degree 3 instead of Cardano's formula, `RootOf` expressions instead of radicals), a small tour through algebraic numbers makes sense anyway.

The trigonometric form occuring in the casus irreducibilis is the bridge between the radical and `RootOf` representations of algebraic numbers. Their study involves only knowledge about simplification of trigonometric expressions at an advanced high school level, but requires certain experience with rule based programming.

Normal forms are one of the central issues of our introductory course. These notions and concepts of simplification become apparent during design and analysis of the properties of the algebraic `RootOf` simplifier.

Finally, the outlook on algebraic extension towers demonstrates the limits of the type-less structure of second generation Computer Algebra Systems and indicates the value of ongoing developments concerned with the incorporation of modern computer science concepts into symbolic systems.

Hence the proposed small tour through algebraic numbers indeed passes many important concepts of symbolic computation in an essential way.

References

Aslaksen, H. (1996). Multiple-valued complex functions and computer algebra. *SIGSAM Bull.* **116**, 8-11.

Braun, S. (1997). Simulationen werden durch den Einsatz von Computeralgebra effizienter. *Computer Zeitung*, 6.3.1997, p. 16.

Comon, H. et al. (1991). A rewrite-based type discipline for a subset of computer algebra. *J. Symb. Comp.* **11**, 349 - 368.

Corless, R.M. and Jeffrey, D.J. (1996). Editor's corner: The unwinding number. *SIGSAM Bull.* **116**, 8-11.

Davenport, J.H. (1990). Current problems in Computer Algebra Systems design. In *Proc. DISCO'90*, Lecture Notes in Comp. Sci. **429**, 1 - 9.

Fateman, R.J. (1990). Advances and trends in the design and construction of algebraic manipulation systems. In *Proc. ISSAC'90*, ACM Press, 60 - 67.

Fateman, R.J. (1996). Why Computer Algebra Systems sometimes can't solve simple equations. *SIGSAM Bull.* **116**, 8-11.

Gräbe, H.-G. (1998). *Introduction to Symbolic Computation.* Unpublished lecture notes (in german), Univ. Leipzig.

Jenks, R.D. and Sutor, R.S. (1992). *AXIOM - the scientific computation system.* Springer, New York.

Loos, R. (1983a). Introduction. In B. Buchberger, G.E. Collins, R. Loos (eds.). *Computer Algebra – Symbolic and Algebraic Computation.* Springer, Wien.

Loos, R. (1983b). Computing in algebraic extensions. In B. Buchberger, G.E. Collins, R. Loos (eds.). *Computer Algebra – Symbolic and Algebraic Computation.* Springer, Wien.

Pieper, H. (1988). *Die komplexen Zahlen.* Deutscher Verlag der Wissenschaften, Berlin.

Wolfram, S. (1998). *The Mathematica Book*, third edition. Cambridge Univ. Press.