# Algebraic Number Fields

Eberhard Schrüfer
Institute SCAI.Alg
German National Research Center for Information Technology (GMD)
Schloss Birlinghoven
D-53754 Sankt Augustin
Germany
Email: schruefer@gmd.de

Algebraic numbers are the solutions of an irreducible polynomial over some ground domain. The algebraic number $i$ (imaginary unit), for example, would be defined by the polynomial $i^2 + 1$. The arithmetic of algebraic number $s$ can be viewed as a polynomial arithmetic modulo the defining polynomial.

Given a defining polynomial for an algebraic number $a$

$$a^n \ + \ p_{n-1}a^{n-1} \ + \ ... \ + \ p_0$$

All algebraic numbers which can be built up from $a$ are then of the form:

$$r_{n-1}a^{n-1} \ + \ r_{n-2}a^{n-2} \ + \ ... \ + \ r_0$$

where the $r_j$'s are rational numbers.

The operation of addition is defined by

$$(r_{n-1}a^{n-1} \ + \ r_{n-2}a^{n-2} \ + \ ...) \ + \ (s_{n-1}a^{n-1} \ + \ s_{n-2}a^{n-2} \ + \ ...) \ = \\ (r_{n-1} + s_{n-1})a^{n-1} \ + \ (r_{n-2} + s_{n-2})a^{n-2} \ + \ ...$$

Multiplication of two algebraic numbers can be performed by normal polynomial multiplication followed by a reduction of the result with the help of the defining polynomial.

$$
\begin{aligned}
(r_{n-1}a^{n-1} + r_{n-2}a^{n-2} + ...) \ \times \ (s_{n-1}a^{n-1} + s_{n-2}a^{n-2} + ...) = \\
r_{n-1}s^{n-1}a^{2n-2} + ... \ \mathbf{modulo} \ a^n \ + \ p_{n-1}a^{n-1} \ + \ ... \ + \ p_0 \\
= \quad q_{n-1}a^{n-1} \ + \ q_{n-2}a^{n-2} \ + \ ...
\end{aligned}
$$

Division of two algebraic numbers r and s yields another algebraic number q.

$\frac{r}{s} = q$ or $r = qs$.

The last equation written out explicitly reads

$$
\begin{aligned}
&(r_{n-1}a^{n-1} + r_{n-2}a^{n-2} + \ldots) \\
&\quad = \ (q_{n-1}a^{n-1} + q_{n-2}a^{n-2} + \ldots) \times (s_{n-1}a^{n-1} + s_{n-2}a^{n-2} + \ldots) \\
&\qquad \mathbf{modulo}(a^n + p_{n-1}a^{n-1} + \ldots) \\
&\quad = \ (t_{n-1}a^{n-1} + t_{n-2}a^{n-2} + \ldots)
\end{aligned}
$$

The $t_i$ are linear in the $q_j$. Equating equal powers of $a$ yields a linear system for the quotient coefficients $q_j$.

With this, all field operations for the algebraic numbers are available. The translation into algorithms is straightforward. For an implementation we have to decide on a data structure for an algebraic number. We have chosen the representation REDUCE normally uses for polynomials, the so-called standard form. Since our polynomials have in general rational coefficients, we must allow for a rational number domain inside the algebraic number.

<*algebraic number*> ::=
    `:ar:` . <*univariate polynomial over the rationals*>

<*univariate polynomial over the rationals*> ::=
    <*variable*> .** <*ldeg*> .* <*rational*> .+ <*reductum*>

<*ldeg*> ::= integer

<*rational*> ::=
    `:rn:` . <*integer numerator*> . <*integer denominator*> : integer

<*reductum*> ::= <*univariate polynomial*> : <*rational*> : nil

This representation allows us to use the REDUCE functions for adding and multiplying polynomials on the tail of the tagged algebraic number. Also, the routines for solving linear equations can easily be used for the calculation of quotients. We are still left with the problem of introducing a particular algebraic number. In the current version this is done by giving the defining polynomial to the statement **defpoly**. The algebraic number sqrt(2), for example, can be introduced by

```
defpoly sqrt2**2 - 2;
```

This statement associates a simplification function for the translation of the variable in the defining polynomial into its tagged internal form and also generates a power reduction rule used by the operations **times** and **quotient** for the reduction of their result modulo the defining polynomial. A basis for the representation of an algebraic number is also set up by the statement. In the working version, the basis is a list of powers of the indeterminate of the defining polynomial up to one less then its degree. Experiments with integral bases, however, have been very encouraging, and these bases might be available in a later version. If the defining polynomial is not monic, it will be made so by an appropriate substitution.

**Example 1**

```
defpoly sqrt2**2-2;

1/(sqrt2+1);

sqrt2 - 1

(x**2+2*sqrt2*x+2)/(x+sqrt2);

x + sqrt2

on gcd;

(x**3+(sqrt2-2)*x**2-(2*sqrt2+3)*x-3*sqrt2)/(x**2-2);

  2
(x  - 2*x - 3)/(x - sqrt2)

off gcd;
```

```
sqrt(x**2-2*sqrt2*x*y+2*y**2);

abs(x - sqrt2*y)
```

Until now we have dealt with only a single algebraic number. In practice this is not sufficient as very often several algebraic numbers appear in an expression. There are two possibilities for handling this: one can use multivariate extensions [?] or one can construct a defining polynomial that contains all specified extensions. This package implements the latter case (the so called primitive representation). The algorithm we use for the construction of the primitive element is the same as given by Trager [?]. In the implementation, multiple extensions can be given as a list of equations to the statement **defpoly**, which, among other things, adds the new extension to the previously defined one. All algebraic numbers are then expressed in terms of the primitive element.

**Example 2**

```
defpoly sqrt2**2-2,cbrt5**3-5;

*** defining polynomial for primitive element:

  6       4        3        2
a1  - 6*a1  - 10*a1  + 12*a1  - 60*a1 + 17

sqrt2;

          5             4              3              2
48/1187*a1  + 45/1187*a1  - 320/1187*a1  - 780/1187*a1  +


735/1187*a1 - 1820/1187

sqrt2**2;

2
```

We can provide factorization of polynomials over the algebraic number domain by using Trager's algorithm. The polynomial to be factored is first mapped to a polynomial over the integers by computing the norm of the polynomial, which is the resultant with respect to the primitive element of the polynomial and the defining polynomial. After factoring over the integers, the factors over the algebraic number field are recovered by GCD calculations.

**Example 3**

```
defpoly a**2-5;

on factor;

x**2 + x - 1;

(x + (1/2*a + 1/2))*(x - (1/2*a - 1/2))
```

We have also incorporated a function **split_field** for the calculation of a primitive element of minimal degree for which a given polynomial splits into linear factors. The algorithm as described in Trager's article is essentially a repeated primitive element calculation.

**Example 4**

```
split_field(x**3-3*x+7);

*** Splitting field is generated by:

  6        4        2
a2  - 18*a2  + 81*a2  + 1215




         4          2
{1/126*a2  - 5/42*a2  - 1/2*a2 + 2/7,


          4          2
 - (1/63*a2  - 5/21*a2  + 4/7),
```

```
          4           2
1/126*a2  - 5/42*a2  + 1/2*a2 + 2/7}


for each j in ws product (x-j);

 3
x  - 3*x + 7
```

A more complete description can be found in [?].