Basic Algorithms In Computer Algebra

Kaiserslautern SS 2015

Prof. Dr. Wolfram Decker

April 22, 2015
References

Chapter 0

Introduction

Computer algebra is a discipline between mathematics and computer science which deals with designing, analyzing, implementing, and applying algebraic algorithms. We comment on this in more detail in what follows.

Applications

Computer algebra is interdisciplinary in nature, with links to quite a number of areas in mathematics, with applications in mathematics, other branches of science, and engineering:

• Through computer algebra methods, a number of mathematical disciplines become accessible to far reaching experiments. This is in particular true for group and representation theory, commutative algebra and algebraic geometry, and number theory. The experiments enable mathematicians to test working hypotheses or conjectures in a large number of instances; to find counterexamples or enough mathematical evidence to sharpen a conjecture; to arrive at new conjectures in the first place; to verify theorems whose proofs are reduced to handling a finite number of special cases. Prominent examples are the conjecture of Birch and Swinnerton-Dyer (see http://www.claymath.org/millennium-problems) and the computer-assisted solution to the four colour problem by Appel and Haken.

• Algebraic algorithms open up new ways of computing in classical application areas of mathematics in science (physics, chemistry, biology). A pioneering and prominent example is work by Veltman and ’t Hooft who won a Nobel price in physics in 1999 (awarded ‘for having placed particle physics theory on a firmer mathematical foundation’).

• Modern application areas of mathematics such as cryptography, coding theory, CAD, robotics, algebraic statistics, and algebraic biology heavily rely on computer algebra.

Implementations

There is a large variety of computer algebra systems (CAS) suiting different needs, see

http://orms.mfo.de/
The practical sessions accompanying this course feature the computer algebra system SINGULAR (see http://www.singular.uni-kl.de/).

Typically, a CAS consists of

- a kernel, written in a programming language such as C/C++, and containing the core algorithms,
- a user language, which allows users easy access to the system and enables them to extend the system, and libraries (packages) written in this language,
- an online manual and help function.

Analyzing Algorithms

One way of measuring the efficiency of an algorithm is to give asymptotic bounds on its running time which depend on the size of the input. For this, we will use the “big-Oh” notation.

Designing Algorithms

When designing algorithms, we will describe them in a somewhat informal way which makes use of the structural conventions of a programming language. We refer to such a description as pseudocode. Here is a classical example which should be familiar from the course Algebraische Strukturen:

**Algorithm 0.1 Euclid’s Algorithm**

<p>| Input: | $m, n \in R, n \neq 0$, where $R$ is an Euclidean domain |</p>
<table>
<thead>
<tr>
<th>Output:</th>
<th>$\gcd(m, n)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a := m, b := n$</td>
<td></td>
</tr>
<tr>
<td><strong>while</strong> ($b \neq 0$) <strong>do</strong></td>
<td></td>
</tr>
<tr>
<td>$r := \text{rem}(a, b)$ // division with remainder</td>
<td></td>
</tr>
<tr>
<td>$a := b, b := r$</td>
<td></td>
</tr>
<tr>
<td><strong>return</strong> $n(a)$ // normal form</td>
<td></td>
</tr>
</tbody>
</table>

Some comments on this are in order. In any Euclidean domain $R$, with Euclidean function $v$, if $a, b \in R$ with $b \neq 0$, and if

$$a = q \cdot b + r, \quad \text{with} \quad r = 0 \quad \text{or} \quad v(r) < v(b),$$

we write

$$\text{quo}(a, b) = q \quad \text{and} \quad \text{rem}(a, b) = r$$

for the quotient $q$ and the remainder $r$, respectively. Note, however, that $q$ and $r$ need not be unique. If $R = \mathbb{Z}$, with $v$ given by the absolute value, uniqueness is achieved by the additional requirement that $r \geq 0$. In the case where $R = K[x]$ is the polynomial ring over a field $K$, with $v$ given by the degree, $q$ and $r$ are unique without further requirement.

If $R$ is any integral domain, and if a greatest common divisor $c$ of two elements $a, b \in R$ exists, all such divisors are obtained by multiplying $c$ with a unit of $R$. That is, the
greatest common divisors form an equivalence class under being associated. In this lecture, we always assume that in each such equivalence class a normal form is selected. If the class is represented by \( a \in R \), we write \( n(a) \) for the normal form. Here, we set \( n(0) = 0 \) and \( n(1) = 1 \) (the normal form of the units in \( R \)), and require that \( n(a \cdot b) = n(a) \cdot n(b) \). If \( a \neq 0 \), then \( a = u(a) \cdot n(a) \) for a uniquely determined unit \( u(a) \in R \). By convention, \( u(0) = 0 \). Given two elements \( a, b \in R \) for which greatest common divisors exist, \( \gcd(a, b) \) will denote the greatest common divisor in normal form. Similarly, for the least common multiple \( \lcm(a, b) \). In a unique factorization domain (UFD), greatest common divisors and lowest common multiples always exist, as we know from the course Algebraische Strukturen. Basic examples of UFDs are principal ideal domains such as \( \mathbb{Z} \) and \( K[x] \). Recall that every Euclidean domain is a principal ideal domain.

Example 0.1.  
(i) In \( \mathbb{Z} \), only \( \pm 1 \) are units. We choose the integers \( \geq 0 \) as normal forms.

(ii) In a field \( K \), all nonzero elements are units. Their normal form is, thus, 1.

(iii) Given any ring \( R \) in which normal forms are distinguished, the normal forms in the polynomial ring \( R[x] \) are the polynomials whose leading coefficients are normal forms in \( R \). If \( R = K \) is a field, we get the monic polynomials.

Now, we give an example of Euclid’s algorithm at work:

Example 0.2. In \( \mathbb{Z} \), we have \( \gcd(18, 30) = 6 \):

<table>
<thead>
<tr>
<th>r</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>18</td>
<td>30</td>
</tr>
<tr>
<td>18</td>
<td>18</td>
<td>12</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>0</td>
</tr>
</tbody>
</table>

The computation of greatest common divisors in \( \mathbb{Z}[x] \) can, in principle, be reduced to that in \( \mathbb{Q}[x] \) using Gauss’ lemma (we will see this later). A problem with this approach is intermediate coefficient swell.

Modular algorithms provide a solution to this problem. Their basic idea is to reduce the given polynomials modulo one (or several) prime numbers, compute their greatest common divisors in the corresponding prime fields \( \mathbb{Z}/p\mathbb{Z} \), and lift the results to \( \mathbb{Z}[x] \) (for example, by Chinese remaindering).

In general, modular algorithms using several prime numbers are well-suited for parallelisation. Note that parallelisation allows us to make efficient use of modern multicore computers or clusters of computers.

What are Algebraic Algorithms?

Algebraic algorithms deal with algebraic objects, make use of algebraic methods, and are based on algebraic theorems. Objects are represented exactly and calculations are carried through exactly (no approximation is applied at any step).
What is an Algorithm?

An algorithm is a set of instructions for solving a particular problem in finitely many, well-defined steps. Starting from a given input, the instructions describe a computation which eventually will produce an output and terminate. The transition from one step to the next one is not necessarily deterministic: probabilistic algorithms incorporate random input, which may lead to random performance and random output.
Chapter 1

The Representation and Arithmetic of Numbers and Polynomials

Multiprecision Integers

**Remark and Definition 1.1** (b-adic Number System). Fix a number $b \in \mathbb{N}_{\geq 2}$. The elements of $\{0, \ldots, b-1\}$ are called **digits** (in **base** $b$). Every $a \in \mathbb{N}_{\geq 1}$ has a unique representation

$$a = \sum_{i=0}^{n-1} a_i b^i, \quad a_i \in \mathbb{N}, \quad 0 \leq a_i \leq b - 1,$$

with $a_{n-1} \neq 0$. We may then represent $a$ as a **word**

$$(a_{n-1}, a_{n-2}, \ldots, a_0)_b$$

of **length** $n$. If we allow leading zeros, we may represent all integers between 0 and $b^n - 1$ as such a word. We then refer to the respective $a_i$ as the **digits** of this representation of $a$.

If $b = 2$, we speak of **bits** instead of digits, and an $n$–**bit word** instead of a word of **length** $n$. A **byte** is an 8–bit word.

**Example 1.2.** $(1101)_2$ represents

$$2^3 + 2^2 + 2^0 = 13.$$
is encoded as the array
\[ s \cdot 2^{63} + n, a_0, \ldots, a_{n-1} \]
of 64–bit words (and 0 as the array with single entry 0).

**Example 1.3.** \( -1 \) is encoded as
\[ 2^{63} + 1, 1. \]