

Chapter 1

Introduction

1.1 What is Computer Algebra?

Computer algebra [15], [38], [45] is the name of the technology for manipulating mathematical formulae symbolically by digital computers. For example an expression such as

$$x - 2 * x + \frac{d}{dx}(x - a)^2$$

should evaluate symbolically to

$$x - 2 * a.$$

Symbolic simplifications of algebraic expressions are the basic properties of computer algebra. Symbolic differentiation using the sum rule, product rule and division rule has to be part of a computer algebra system. Symbolic integration should also be included in a computer algebra system. Furthermore expressions such as $\sin^2(x) + \cos^2(x)$ and $\cosh^2(x) - \sinh^2(x)$ should simplify to 1. Thus another important ingredient of a computer algebra system is that it should allow one to define rules. Examples are the implementations of the exterior product and Lie algebras. Another important part of a computer algebra system is the symbolic manipulation of polynomials, for example to find the greatest common divisor of two polynomials or finding the coefficients of a polynomial.

The name of this discipline has long hesitated between symbolic and algebraic calculation, symbolic and algebraic manipulations and finally settled down as Computer Algebra. Algebraic computation programs have already been applied to a large number of areas in science and engineering. It is most extensively used in the fields where the algebraic calculations are extremely tedious and time consuming, such as general relativity, celestial mechanics and quantum chromodynamics. One of the first applications was the calculation of the curvature of a given metric tensor field. This involves mainly symbolic differentiation.

1.2 Properties of Computer Algebra Systems

What should a computer algebra system be able to do? First of all it should be able to handle the data types such as very long integers, rational numbers, complex numbers, quaternions, etc. The basic properties of the symbolic part should be simplifications of expressions, for example

$$a + 0 = a, \quad 0 + a = a$$

$$a - a = 0, \quad -a + a = 0$$

$$a * 0 = 0, \quad 0 * a = 0$$

$$a * 1 = a, \quad 1 * a = a$$

$$a^0 = 1.$$

In most systems it is assumed that the symbols are *commutative*, i.e.,

$$a * b = b * a.$$

Thus an expression such as

$$(a + b) * (a - b)$$

should be evaluated to

$$a * a - b * b.$$

If the symbols are not commutative then a special command should be given to indicate so. Furthermore, a computer algebra system should do simplifications of trigonometric and hyperbolic functions such as

$$\sin(0) = 0, \quad \cos(0) = 1$$

$$\cosh(0) = 1, \quad \sinh(0) = 0.$$

The expression $\exp(0)$ should simplify to 1 and $\ln 1$ should simplify to 0. Expressions such as

$$\sin^2(x) + \cos^2(x), \quad \cosh^2(x) - \sinh^2(x)$$

should simplify to 1. Besides symbolic differentiation and integration a computer algebra system should also allow the symbolic manipulation of vectors, matrices and arrays. Thus the scalar product and vector product must be calculated symbolically. For square matrices the trace and determinant have to be evaluated symbolically. Furthermore, the system should also allow numerical manipulations. Thus it should be able to switch from symbolic to numerical manipulations. The computer algebra system should also be a programming language. For example, it should allow if-conditions and for-loops. Moreover, it must allow functions and procedures.

1.3 Pitfalls in Computer Algebra Systems

Although computer algebra systems have been around for many years there are still bugs and limitations in these systems. Here we list a number of typical pitfalls.

One of the typical pitfalls is the evaluation of

$$\sqrt{a^2 + b^2 - 2ab}.$$

Some computer algebra systems indicate that $a - b$ is the solution. Obviously, the result should be

$$\pm|a - b|.$$

As another example consider the *rank* of the matrix

$$A = \begin{pmatrix} 0 & 0 \\ x & 0 \end{pmatrix}.$$

The rank of a matrix is the number of linearly independent columns (which is equal to the number of linearly independent rows). If $x = 0$, then the rank is equal to 0. On the other hand if $x \neq 0$, then the rank of A is 1. Thus computer algebra systems face an ambiguity in the determination of the rank of the matrix. A similar problem arises when we consider the inverse of the matrix

$$B = \begin{pmatrix} 1 & 1 \\ x & 0 \end{pmatrix}.$$

It only exists when $x \neq 0$.

Another problem arises when we ask computer algebra systems to integrate

$$\int x^n dx$$

where n is an integer. If $n \neq -1$ then the integral is given by

$$\frac{x^{n+1}}{n+1}.$$

If $n = -1$, the integral is

$$\ln(x).$$

Another ambiguity arises when we consider

$$0^0.$$

Consider for example

$$f(x) = x^x \equiv \exp(x \ln(x))$$

for $x > 0$. Applying *L'Hospital's rule* we find $0^0 = 1$ as a possible definition of 0^0 . Many computer algebra systems have problems finding

$$\frac{x}{x + \sin(x)}$$

at $x = 0$ using L'Hospital's rule. The result is $1/2$.

We must also be aware that when we solve the equation

$$a * x = 0$$

the computer algebra system has to distinguish between the cases $a = 0$ and $x = 0$.

A large number of pitfalls can arise when we consider complex numbers and branch points in complex analysis. Complex numbers and functions should satisfy the Aslaksen test [3]. Thus

$$\exp(\ln(z))$$

should simplify to z , but

$$\ln(\exp(z))$$

should not simplify for complex numbers. We have to take care of the branch cuts when we consider multiple-valued complex functions. Most computer algebra systems assume by default that the argument is real-valued.

Example. Consider the equation

$$\text{Log}(zw) = \text{Log } z + \text{Log } w$$

where $z \neq 0$, $w \neq 0$ and Log is the principal logarithm. The left hand side of the equation is equivalent to

$$\text{Log}(zw) = \text{Log } |r| + \text{Log } |z| + i\text{Arg}(zw)$$

where $\text{Arg}(zw) \in (-\pi, \pi]$ is the principle argument of zw . The right hand side of the equation can be written as

$$\text{Log } z + \text{Log } w = \text{Log } |r| + \text{Log } |w| + i(\text{Arg}(z) + \text{Arg}(w)).$$

For the equation to hold we must have

$$\text{Arg}(z) + \text{Arg}(w) \in (-\pi, \pi].$$

For a more in-depth survey of the pitfalls in computer algebra systems Stoutemeyer [58] may be perused.

1.4 Design of a Computer Algebra System

Most computer algebra systems are based on Lisp. The computer language Lisp takes its name from list processing. The main task of Lisp is the manipulation of quantities called lists, which are enclosed in parentheses. A number of powerful computer algebra systems are based in Lisp, for example Reduce, Maxima, Derive, Axiom and MuPAD. The design of Axiom is based on object-oriented programming using Lisp. The computer algebra systems Maple and Mathematica are based on C. All of these systems are powerful software systems which can perform symbolic calculations. However, these software systems are independent systems and the transfer of expressions from one of them to another programming environment such as C is rather tedious, time consuming and error prone. It would therefore be helpful to enable a higher level language to manipulate symbolic expressions. On the other hand, the object-oriented programming languages provide all the necessary tools to perform this task elegantly.

Here we show that object-oriented programming using C++ can be used to develop a computer algebra system. Object-oriented programming is an approach to software design that is based on classes rather than procedures. This approach maximizes modularity and information hiding. Object-oriented design provides many advantages. For example, it combines both the data and the functions that operate on that data into a single unit. Such a unit (*abstract data type*) is called a *class*.

We use C++ as our object-oriented programming language for the following reasons. C++ allows the introduction of abstract data types. Thus we can introduce the data types used in the computer algebra system as abstract data types. The language C++ supports the central concepts of object-oriented programming: encapsulation, inheritance, polymorphism (including dynamic binding) and operator overloading. It has good support for dynamic memory management and supports both procedural and object-oriented programming. A less abstract form of polymorphism is provided via template support. We overload the operators

$$+, -, *, /$$

for our abstract data types, such as verylong integers, rational numbers, complex numbers or symbolic data types. The vector and matrix classes are implemented on a template basis so that they can be used with the other abstract data types.

Another advantage of this approach is that, since the system of symbolic manipulations itself is written in C++, it is easy to enlarge it and to fit it to the special problem at hand. The classes (abstract data types) are included in a header file and can be provided in any C++ program by giving the command `#include "ADT.h"` at the beginning of the program.

For the realization of this concept we need to apply the following features of C++

- (1) the class concept
- (2) overloading of operators
- (3) overloading of functions
- (4) inheritance of classes
- (5) virtual functions
- (6) function templates
- (7) class templates
- (8) Standard Template Library.

The developed system **SymbolicC++** includes the following abstract data types (classes)

Verylong	handles very long integers
Rational	template class that handles rational numbers
Quaternion	template class that handles quaternions
Derive	template class to handle exact differentiation
Vector	template class that handles vectors
Matrix	template class that handles matrices
Array	template class that handles arrays
Polynomial	template class that handles polynomials
Symbolic	class that handles symbolic manipulations, such as rules, simplifications, differentiation, integration, commutativity and non-commutativity
Type/Pair	handles the atom and dotted pair for a Lisp system All the standard Lisp functions are included.

Suitable type conversions between the abstract data types and between abstract data types and basic data types are also provided.

The advantages of SymbolicC++ are as follows

- (1) Object-oriented design and proper handling of basic and abstract data types.
- (2) The system is operating system independent, i.e. for all operating systems powerful C++ compilers are available.
- (3) The user is provided with the source code.
- (4) New classes (abstract data types) can easily be added.
- (5) The ANSI C++ standard is taken into account.
- (6) The user only needs to learn C++ to apply the computer algebra system.
- (7) Assembler code can easily be added to run on a specific CPU.
- (8) Member functions and type conversion operators provide a symbolic-numeric interface.
- (9) The classes (abstract data types) are included in a header file.
- (10) Standard Template Library is used with SymbolicC++.