ALLTYPES: AN ALGEBRAIC LANGUAGE AND TYPE SYSTEM

FRITZ SCHWARZ
FhG, Institut SCAI, 53754 Sankt Augustin, Germany
Email: fritz.schwarz@gmd.de

The software system ALLTYPES provides an environment that is particularly designed for developing software in differential algebra. Its most important features may be described as follows: A set of about thirty parametrized algebraic types is defined. Data objects represented by these types may be manipulated by more than one hundred polymorphic functions. Reusability of code is achieved by genericity and multiple inheritance. The user may extend the system by defining new types and polymorphic functions. A language comprising seven basic language constructs is defined for implementing mathematical algorithms. The easy manipulation of types is particularly supported due to a special portion of the language dedicated to manipulating typed objects, i. e. for performing user-defined or automatic type coercions. Type inquiries are also included in the language.

1 Organization of Computer Algebra Software

The software described in this article originated from the desire to provide an environment for implementing high quality computer algebra software. Areas of application are for example the symmetry analysis of ordinary and partial differential equations, finding closed form solutions of ordinary differential equations and Janet base algorithms.

ALLTYPES provides a collection of types especially designed for modeling mathematical data objects occuring in differential algebra. Its fine-structured type system requires special tools for manipulating objects of these various types easily, they are made available in terms of a specialized portion of the language. Furthermore this language defines a small number of powerful control constructs that are especially well suited for implementing computer algebra code. It may be written such that each line may be executed individually, its action may be specified in mathematical terms, and the result may be checked against this specification.

It turns out that many aspects that arise during the design and the implementation of computer algebra software may be better understood if they are considered in the more general context of software engineering. Good references for these aspects are the books by Coad and Yourdon [1], [3] and Meyer [2]. A recent review by Taivalsaari [4] is also useful.

A little consideration leads to the conclusion that the ultimate reason for