J. Symbolic Computation (1995) **11**, 1–8

The "Optical Formula Recognition" System for Handprinted Input

J. HARTMANN†, G. HOTZ†, R. LOOS‡, R. MARZINKEWITSCH†, J. QUAPP†, F. WEIGEL†, AND A. WEBER‡

†*Fachbereich 14, Universität des Saarlandes*
D-66123 Saarbrücken, Germany
E-mail addresses: {hartmann,hotz,marzin,quapp,weigel}@cs.uni-sb.de

‡*Wilhelm-Schickard-Institut für Informatik*
Universität Tübingen, 72076 Tübingen, Germany
E-mail addresses: {loos,weber}@informatik.uni-tuebingen.de

(Received 10 March 1995)

A prototype of a system is presented which allows the input of mathematical formulas by hand and supports several computer algebra systems as a back end. The system recognizes the handwriting of different users by appropriately trained neural networks and can parse two-dimensional structures.

1. Introduction

Existing computer algebra systems provide the user a high computational power in many areas of applied mathematics: symbolic integration, symbolic solutions of differential equations, factorization of polynomials, exact calculation of zeros of polynomial and other kinds of equations, to mention only a few. In contrast to this fact computer algebra systems have not experienced the deserved widespread use by potential users.

The main reason for this discrepancy seems to be the unnatural operation of computer algebra systems by linearized notations, which only partly coincide with mathematical notations used in the areas of applications. Calculations with pencil and paper not only offer many efficient techniques but also many conventions and certain standardizations have been developped during the centuries of their use to ease them. Thus the use of a familiar — i. e. paperlike — interface to a computer algebra system is of especial importance to occasional users.

In this paper an integrated system — whose development started with (Marzinkewitsch, 1990) — will be presented, which offers the demanded facilities: Calculating by

0747–7171/90/000000 + 00 $03.00/0

hand in a traditional 'two dimensional' fashion with the computational support of a computer algebra system.†

Attempts to design a natural interface to computer algebra systems are as old as computer algebra systems themselves. The 'Sophisticated Scratchpad' of (Anderson, 1970) is the best known attempt to solve the problem. However this system was never really used because its operation lacked flexibility. Therefore the need for natural interfaces to computer algebra systems is unchanged until today.

2. Overview of the Components

The software is structured in seven main modules which are deviced to solve the subtasks of the complete problem: Operating the input and output device, pattern recognition, two-dimensional formula parsing, management of user interaction, calculation of results, and output formatting.

The module for operating the input and output device has to be compiled for the used notepad computer, the other modules have to be compiled for the workstation.

2.1. Operating the Input and Output Device

As an input and output device a commercial notepad computer NCR 3130 is used. We do not use the pen-window functionality but our software only uses the basic commands for recognizing the position of the pen.

In addition to its use as a natural input/output-device it allows the user all 'modes of operation' which are possible with paper and pencil:

(i) writing handprinted symbols of arbitrary size at any position of the input/output-device;
(ii) inputting the symbols of a formula in any order;
(iii) insertion and correction by erasing or overwriting of symbols;
(iv) incremental completion of formulas, reinputting intermediate results; user interaction for the rearrangement of formuls (*cut & paste*, macros);
(v) results of former operations can be looked up in a journal where they are stored.

2.2. Pattern Recognition

As a user writes on the input device the position of the pen is monitored and a stream of coordinates is sent to the *pattern recognition* module. The coordinate stream is segmented by it in disjoint patterns which are then normalized in size.

The actual pattern recognition is done by a neural network. A three-layered backpropagation network (Rumelhart and McClelland, 1986) classifies the patterns into one of currently 55 character classes. The network is trained by *features* of the character classes and allows the recognition of characters written by different users with an accuracy up to 98 %. In (Quapp, 1993) the results of tests with 10 different users are given. The recognition rates of the handwriting of theses users — which had *not* been used to train to network — varied between 93.0 % and 97.6 %.

The classification of a pattern is achieved in real time (0.1 – 0.2 sec/char) and each

† A preliminary description of several components of the system was given in (Marzinkewitsch, 1991).

recognized character is immediately displayed on the input/output-device. A latency of two segments is due to the segmentation process.

2.3. Two-Dimensional Parsing

At the stage of *two-dimensional formula parsing*, the set of characters which is produced by the pattern recognition process is parsed. Parsing is done by a graph reduction process. The characters are incorporated into a graph, which represents their spatial relations. Guided by a context free grammer for two dimensional constructs the graph is reduced in successive steps. The right hand sides of the context free productions are localized in the graph and reduced to a single vertice which corresponds to a nonterminal symbol. Ambiguities are resolved with help of methods of compiler construction: Precedences, attributs etc. Moreover the temporal information of the input process is taken into consideration.

2.4. Call of the Computer Algebra System

When the formula is parsed it is translated into a string that can be used as an input to a computer algebra system. Currently strings that are in the input format for REDUCE (Hearn, 1987) and MAPLE V (Char *et al.*, 1991a; Char *et al.*, 1991b) can be produced by the parser.

Using the type inference component, which is described in Sec. 3, even one output format were sufficient, because that component can serve as a general interface to other computer algebra systems.

The type inference component is not integrated in the system up to now. Thus at the moment only formulas for which the type inference facilites of the used algebra system — MAPLE V and REDUCE in the current implementation — are sufficient are correctly processed. For many examples from calculus this is the case so that the system works well for many cases already in its current stage of development.

The input to the computer algebra system and its output are translated into a two dimensional notation by the *output formatter* and then displayed on the input/output device. Thus the user is not bothered with involved artificial notations any more.

2.5. Management of User Interaction

The cooperation of all modules is controlled by the *management module*, especially when several formulas are developped simultaneously. This module implements all services which are advertized to the user and records all user actions in a journal. In order to ensure a consisten system behaviour, the management module was implemented in accordance with the principleof direct manipulation.

All software modules are bult as seperate processes which are connected by means of process communication. The software is executed by a SUN UNIX-workstation.

We have begun with porting the entire software package to a pentop PC using the wxWindows toolkit (Smart, 1995), which serves as a uniform interface to DOS and UNIX environments.

3. The Type Inference Component

3.1. Motivation

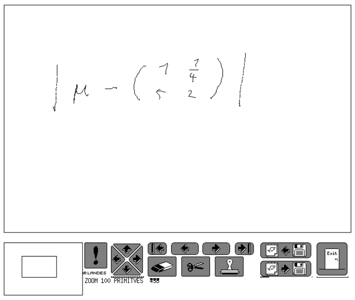
The goal of the project is to allow the user as much "mathematical notation" as possible.

So we have to allow as many mathematical conventions as possible. Having the pen-based input it is possible to allow "typographical" conventions.

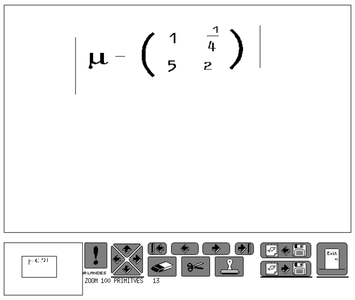
However, there are important other conventions, which cannot be covered by recognition of handwritten characters and 2-dimensional parsing alone, as the following example shows.

The eigenvalues of a matrix A are the zeros of the characteristic polynomial of A, which can be computed as the determinant of $\mu \mathrm{Id} - A$, where Id denotes the identity matrix1.

Thus the following input could occur naturally.



The correct recognition of the occurring patterns as



and the correct parsing of the two-dimensional structure into a one-dimensional one as

```
|(mu - matrix({{1, 1/4}, {5, 2}}))|
```

are not sufficient.

The correct interpretation of the formula requires the resolution of the overloaded operators and the insertion of coercion functions. This part is the one which will be handled by the type inference component of the system.

3.2. The Software Component

The main part of the software component performing the type inference is written in C++ (Stroustrup, 1991). It was developped as a "literate program" using the tools described in (Ramsey, 1989b; Ramsey, 1989a). Currently the literate program (Weber, 1995b) — whose implementation is not finished up to now — consists of about 15000 lines of web-code ($\approx$ 300 kbyte) for C++.

The kernel of the implemented type inference method is the algorithm described in (Weber, 1994; Weber, 1993). Its main features are the following:

(i) The algorithm can correctly resolve cases of coercions which cannot be resolved correctly in any of the existing algebra systems including AXIOM (Jenks and Sutor, 1992), cf. (Weber, 1994).

(ii) The component is designed to be used as a front end for different algebra systems. Especially its intended use is for algebra systems with a user interface performing little type inference like the one of MAPLE V (Char *et al.*, 1991a; Char *et al.*, 1991b) or having no sophisticated user interface at all like the SAC libraries (Collins, 1980; Collins and Loos, 1990; Bündgen *et al.*, 1991; Buchberger *et al.*, 1993).
To this purpose an abstract type hierarchy for algebraic types — which is similar to a fragment of the type system of AXIOM — is built within the type inference component. The type inference algorithm works on this abstract type hierarchy. The main features of this abstract type system have been investigated and described in (Weber, 1993).
The result of this abstract type inference with respect to a particular algebra system can be obtained by the use of several translation tables: One storing the functions of the used algebra system which can be used as coercion functions between to types of the internal type system, and one giving the names of the functions which have to be used as an instance of a particular overloaded operator in the internal type system.[†]

For the internal representation and manipulation of data several of the data types of the "library of efficient data structures and algorithms" LEDA (Mehlhorn and Näher, 1995; Näher, 1995) have been used. By this we do not only have efficient representations of directed acyclic graphs, sets of directed acyclic graphs etc., but also have access to hash tables for all kind of data types. Especially the components for *"history management"* of a session are implemented, allowing to free the bindings of formulas to identifiers, if going back to a point in which such a binding has not occurred.

4. Examples

An example demonstrating the operation of the system solving a differential equation 'manually' by variation of the constant is given in (Marzinkewitsch, 1991). These examples show that the usual notation for (definite) integration can be used and how the *cut & paste* techniques can be applied.

[†] There is not an arbitrary overloading of operators in the internal type system but only one which can be obtained by the mechanisms of AXIOM "categories" or Haskell "type classes" (Hudak *et al.*, 1992). We refer to (Weber, 1993; Weber, 1994; Weber, 1995a) for more details.

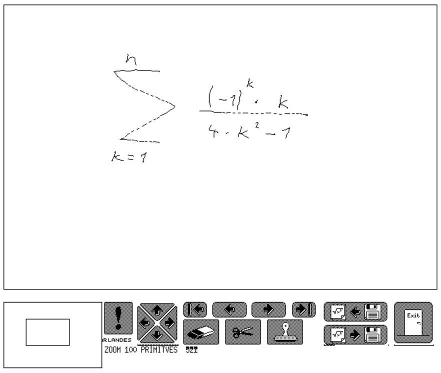


Figure 1. Handprinted input

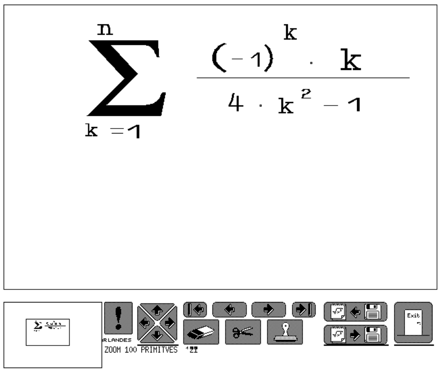


Figure 2. Input after pattern recognition

Below we will show how the system can be used for another frequent class of "two-dimensional notation", namely summations or products.

The "exam problem 2.29" in (Graham *et al.*, 1989) is to evaluate the sum

$$\sum_{k=1}^{n} \frac{(-1)^k \cdot k}{4 \cdot k^2 - 1} \ .$$

In Fig. 1 the handprinted input is shown in its original form† and in Fig. 2 the input is shown after the pattern recognition phase.

The answer of MAPLE V — in its two dimensional form that is produced by the output formatter — is given in Fig. 3.

Remark: We have solved the exercise as it is stated in the "seventh printing, with

† We have switched off the pattern recognition to produce this figure.

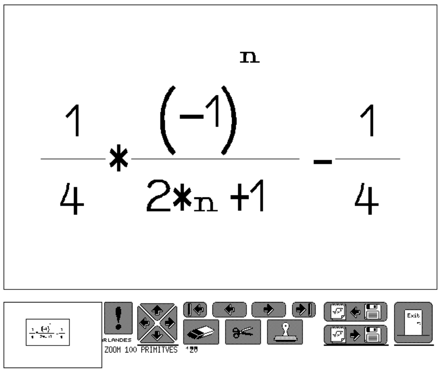


Figure 3. Answer from the computer algebra system

corrections” of (Graham *et al.*, 1989). In the first printing of (Graham *et al.*, 1989) the question was to evaluate the indefinite sum

$$\sum_{k=1}^{\infty} \frac{(-1)^k \cdot k}{4 \cdot k^2 - 1} .$$

Using the answer from the system it is easy to see that the first term vanishes if $n \longrightarrow \infty$. Thus we obtain the correct answer -1/4 also for this question.

Acknowledgements. This research was supported by the *Deutsche Forschungsgemeinschaft.*

References

Anderson, R. (1970). *An Online Methematics System Using Twodimensional handprinted Notation.* The RAND Corporation, Santa Monica.

Buchberger, B., Collins, G. E., Encarnaación, M. J., Hong, H., Johnson, J. R., Krandick, W., Loos, R., Mandache, A., Neubacher, A., and Vielhaber, H. (1993). *SACLIB User’s Guide.* Johannes Kepler Universität, 4020 Linz, Austria. Available via anonymous ftp at `melmac.risc.uni-linz.ac.at` in `pub/saclib`.

Bündgen, R., Hagel, G., Loos, R., Seitz, S., Simon, G., Stübner, R., and Weber, A. (1991). SAC-2 in ALDES — Ein Werkzeug für die Algorithmenforschung. *mathPAD*, 1(3):33–37. Universität Paderborn.

Char, B. W., Geddes, K. O., Gonnet, G. H., Benton, L. L., Monagan, M. B., and Watt, S. M. (1991a). *Maple V Language Reference Manual.* Springer-Verlag, New York.

Char, B. W., Geddes, K. O., Gonnet, G. H., Benton, L. L., Monagan, M. B., and Watt, S. M. (1991b). *Maple V Library Reference Manual.* Springer-Verlag, New York.

Collins, G. E. (1980). ALDES and SAC–2 now available. *SIGSAM Bulletin*, 12(2):19.

Collins, G. E. and Loos, R. G. K. (1990). Specifications and index of SAC-2 algorithms. Technical Report WSI–90–4, Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, 72076 Tübingen, Germany.

Graham, R. L., Knuth, D. E., and Patashnik, O. (1989). *Concrete Mathematics.* Addison-Wesley, Reading, MA.

Hearn, A. C. (1987). REDUCE user’s manual, Version 3.3. Report CP 78, The RAND Corporation, Santa Monica.

Hudak, P., Peyton Jones, S., Wadler, P., *et al.* (1992). Report on the programming language Haskell — a non-strict, purely functional language, version 1.2. *ACM SIGPLAN Notices*, 27(5).

Jenks, R. D. and Sutor, R. S. (1992). *AXIOM: The Scientific Computation System.* Springer-Verlag, New York.

Marzinkewitsch, R. (1990). *Ein Arbeitsplatz zum computerunterstützten handschriftlichen Rechnen mit mathematischen Formeln.* Dissertation, Universität des Saarlandes, Saarbrücken, Germany.

Marzinkewitsch, R. (1991). Operating computer algebra systems by handprinted input. In Watt, S. M., editor, *Proc. Symposium on Symbolic and Algebraic Computation (ISSAC '91)*, pages 411–413, Bonn, Germany. Association for Computing Machinery.

Mehlhorn, K. and Näher, S. (1995). LEDA: A platform for combinatorial and geometric computing. *Communications of the ACM*, 38(1):96–102.

Näher, S. (1995). *The LEDA User Manual — Version 3.1.* Max-Planck-Institut für Informatik, 66123 Saarbrücken, Germany. Available via anonymous ftp at `ftp.mgi-sb.mpg.de` in `/pub/LEDA`.

Quapp, J. (1993). Ein- und Mehrschreiber-Systeme zur Handschrifterkennung mit Neuronalen Netzen. Diplomarbeit, Universität des Saarlandes.

Ramsey, N. (1989a). Building a language-independent `web`. *Communications of the ACM.*

Ramsey, N. (1989b). *Spidery* `WEB` *User Manual.* Available via anonymous ftp from `pip.shsu.edu` in `tex-archive/web/spiderweb`.

Rumelhart, D. E. and McClelland, J. L., editors (1986). *Parallel Distributed Processing.* MIT Press, Cambridge, MA.

Smart, J. (1995). *User Manual for wxWindows 1.61: a portable C++ GUI toolkit.* Artificial Intelligence Applications Institute, University of Edinburgh. Sources availabe via anonymous ftp at `aiai.ed.ac.uk` in `pub/wxwin/beta`.

Stroustrup, B. (1991). *The C++ Programming Language.* Addison-Wesley, Reading, MA, second edition.

Weber, A. (1993). *Type Systems for Computer Algebra.* Dissertation, Fakultät für Informatik, Universität Tübingen.

Weber, A. (1994). Algorithms for type inference with coercions. In *Proc. Symposium on Symbolic and Algebraic Computation (ISSAC '94)*, pages 324–329, Oxford. Association for Computing Machinery.

Weber, A. (1995a). On coherence in computer algebra. *Journal of Symbolic Computation.* To appear.

Weber, A. (1995b). Type inference for algebra systems. A Literate Program in C++ building a generic computer algebra interface.