Optical Formula Recognition

Prof. G. Hotz • Joachim Quapp • Frank Marvin Weigel

Fachbereich 14
Universität des Saarlandes
D-66123 Saarbrücken
email: quapp@sol.cs.uni-sb.de marvin@cs.uni-sb.de

Abstract

In the following an online system for optical formula recognition (OFR) will be presented. With these prototyp formulas handwritten on a pentop can be processed. First the characters are recognized online. When the user has written a complete formula the system analyses the tow-dimensional structure of the formula and converts it into a linearized version suitable for computer algebra systems (CAS'). With this system it is possible to write two-dimensional formulas in a very intuitive way.

Keywords: Character recognition, feature extraction, neural network, computer algebra system, context free grammar, fuzzy sets

1 Introduction

Twenty years have passed since the first computer algebra systems came up in the beginning of the seventies. Since then CAS have gained a lot of computational power. In contrast to this fact CAS did not reach the deserved widespread use by potential users.
The main reason for this discrepancy is the unnatural operation of CAS by artificial linearized notations, which tend to give little comprehensive survey of the problem under work. Calculation with pencil and paper not only offers many efficient techniques but also appeals to the user's ease. Especially occasional users need a familiar i.e. paperlike interface to CAS.
In this report a system will be presented, which offers the demanded facilities: Calculating by hand in a traditional, 'two dimensional' fashion with the computational support of a CAS.
Attempts to design a natural interface to CAS are as old as CAS themselves.
The 'Sophisticated Scratchpad' of [1] was the first 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 CAS is unchanged up to day.

2 General Features

Our system works with a pentop as I/O-device that allows a natural operation of the system. When symbols are inputted on the display of the pentop they will be recognized online. There are two possible modes for character recognition. The single-user mode allows an individual handwriting; In the multi-user mode characters are standardized. When the formula is completed, the user may evalute the expression.
In addition to the utilization of a natural I/O-device, the user has all 'modes of operation' which are possible with paper and pencil:

- writing handprinted symbols of arbitrary size at any position on the pentop

- inputting the symbols of a formula in any order
- insertion and correction by erasing or overwriting symbols
- incremental completion of formulas, reinputing intermediate results
- user interaction for the rearrangement of formulas (cut, copy, paste)
- results of former operations can be looked up in a journal where they are stored on several virtual pages
- persistant storage of sessions from/to file.

The user should be enabled to control the system as it interprets his input. Real time execution was a mayor design goal during the development of the system.

3 Components of the System

3.1 Overview

The software is structured into four modules which are deviced to solve the subtasks of the complete problem: Character recognition, formula analysis, managment of the user-interaction and formula evaluation with a CAS like Maple or Reduce. The two main modules are character recognition and formula analysis. Both will be explained below.

3.2 Character recognition

Up to now character recognition works with 57 symbols. The symbols are shown in figure 1.
The presented character recognition works with a neural network based on backpropagation [2] with features of the characters as input. Tests with the pixmap combined with a linear space invariant filter as input have not reached such good recognition results.

Figure 1: The classified characters

The character classifier receives as input the coordinates of the touched points on the display of the pentop. The list of points is partitioned into *strokes*. These form the input for the character classifier. In a preprocessing step those strokes which form a character are grouped together (*segmentation* of the input). For this work the system uses time information and the relative positions of the strokes.
Next step is to scale the characters into a 16x24 pixel matrix. With this matrix and additional time dependend informations the character recognition does a feature extraction. The feature input has uniform length feature vectors of 44 components. At this time the implemented features [3] are

- *outline* of a character
- *proportion* between height and width of the character
- *start-* and *endpoint*
- *horizontal*, *vertical* and *diagonal movements*

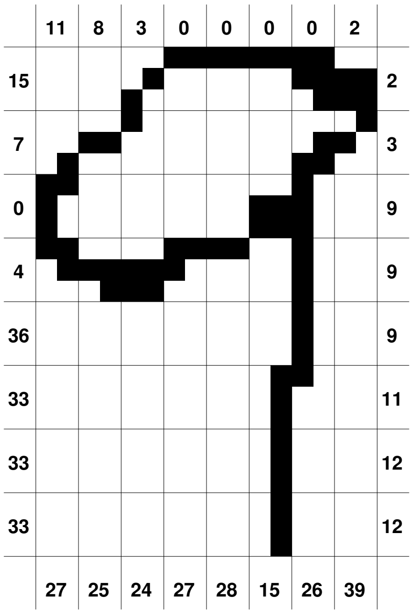

Figure 2:The Feature *outline*

The feature *outline* counts in each row/column the white pixels from the border of the matrix of the character, where three adjacent lines (resp. two columns) are summarized (Figure 2). The character is inputted into the matrix, that is has maximum size. This feature gives 32 components of the feature vector.
Proportion between height and width ($P_{H/B}$ and $P'_{H/B}$) is calculated from all (unscaled) pixels $(x_i,y_i)_{1 \leq i \leq N}$ (N is the number of pixels of the character):

$$P = \frac{h}{w + h}$$

$$w := \mathrm{Max}(\, x_m \mid m \in 1,\ldots,N) - \mathrm{Min}(x_m \mid m \in 1,\ldots,N)$$

$$h := \mathrm{Max}(\, y_m \mid m \in 1,\ldots,N) - \mathrm{Min}(y_m \mid m \in 1,\ldots,N)$$

A second component is with a 45° rotated character obtained:

$$P_{rot} = \frac{h_{rot}}{w_{rot} + h_{rot}} \qquad w_{rot} := \frac{1}{\sqrt{2}}(x - y) \qquad h_{rot} := \frac{1}{\sqrt{2}}(x - y)$$

Very easy to compute are the time depend features start- and endpoint (SP, EP):

$$SP_x = \frac{x_1}{16} \qquad SP_y = \frac{x_1}{24} \qquad EP_x = \frac{x_N}{16} \qquad EP_y = \frac{x_N}{24}$$

The last four components are the movement of the pen, when writing the character (H,V,D1,D2). Given are the formulas for the horizontal and a diagonal movement:

$$H = \frac{16}{\sum_{i=2}^{N} \left| x_i - x_{i-1} \right|} \qquad D1 = \frac{\mathrm{Max}(x_m - y_m \mid m \in 1,\ldots,N) - \mathrm{Min}(x_m - y_m \mid m \in 1,\ldots,N)}{\sum_{i=2}^{N} \left| (x_i - x_{i-1}) - (y_i - y_{i-1}) \right|}$$

The last task of the character recognition is the classification. The classification is done by a backpropagation network with three layers. The input layer gets all extracted features of the character. For the output layer a unary coding was choosen, that is the output layer has as much neurons as classes of different characters exist. Normally not only a single neuron is activated. The system chooses the neuron with the biggest activation.
Experiments have shown that it is the best way to have as much hidden neurons as output neurons.
The character recognition works in a single- and multi-user mode. In the single user mode characters have to be written to train the system. With these characters the neuronal network learns the individual writing of a user. With this technique high accurany quotes are achieved, but the input of the characters and the learning phase of the system take a couple of hours. Other users can't work with a single-user system because of the individual style of the characters. Reason for this is the different writing routine of different users. This may lead to the situation, that different characters have the same appearance (Figure 3). Because of this fact the characters in the multi-user mode are standardized. Such a standardization restricts the user in writing a character, otherwise liberates him from inputting characters to train the system.

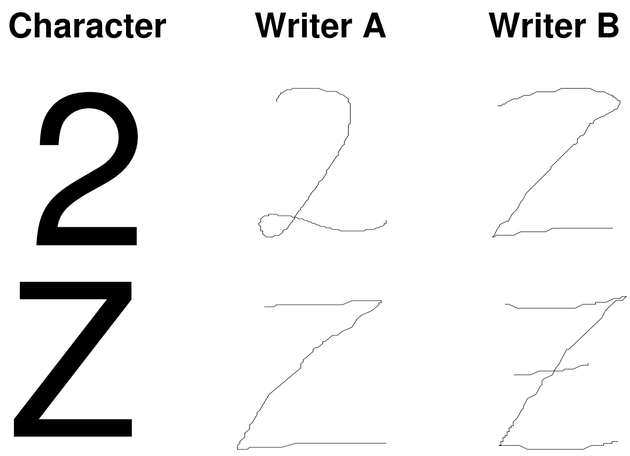

Figure 3: recognition conflicts for multiple users

Recognition results:

Two different experiments were done on handwritten characters using backpropagation with feature input as classifier. In the first one the system learned handwritten characters with 60 prototypes of each character and was tested with 30 prototypes of each character. Table 1 shows the recognition results for 4 different users. The single-user system achived good recognition results of 98.6% (average).

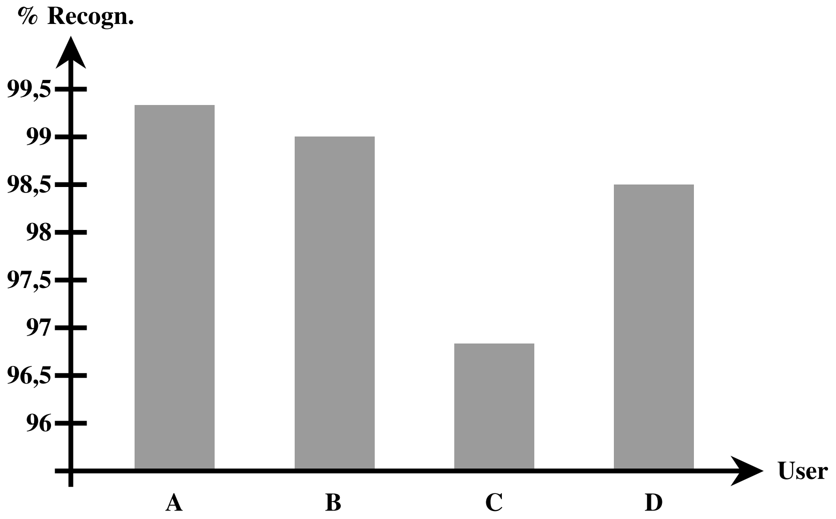

Table 1: Recognition results for single-user system

The second experiment was done to show how the multi-user recognition works. The training set used to learn the weights of the neural network for multi-user recognition were made with 10 alphabets from each of 10 users. The recognition was tested with 10 alphabets of 8 users distinct of the 10 writers above. An average recognition rate of 94.2% was achived. Table 2 shows the recognition result of the 8 users A-H.

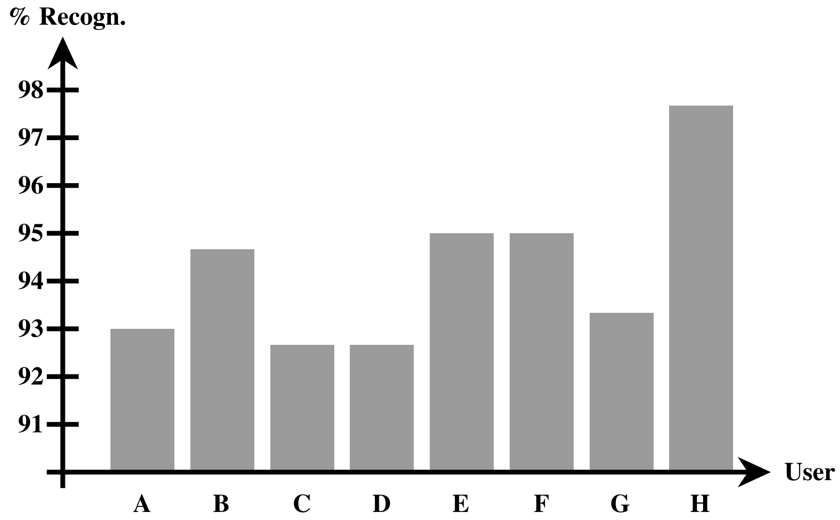

Table 2: Recognition results for multi-user system

3.3 Formula Parsing And Interpretation

At the second stage of our system, a computer readable version of the ‘two dimensional’ formula inputted by the user has to be found. That is, the set of characters produced by the pattern recognition process has to be parsed and has to be transformed in a linearized version suitable for CA systems such as Maple or REDUCE. The mathematical notations implemented so far are

- basic operators with linear notations such as +, -, •, ÷ etc.
- two-dimensional operators as
  fractions, *sums, products*,
  definite and indefinite integrals, derivations, time derivations (dot notation),
  differential equations
  matrices

3.3.1 Two-dimensionally Fuzzy Attributed Context Free Grammars

In one-dimensional parsing theory context free grammars (CFG) are a wellknown tool to describe input languages . Arbitrary complex attributations and a set of context conditions can be used to define semantic constraints. Due to their importance in compiler theory, there exists a wide range of parsing strategies for CFGs.
The recognition of mathematical formulas out of a set of characters in the plane is a two-dimensional parsing problem. So CFG rules and parsing strategies can not make use of the consecutiveness of related characters. Furthermore it was shown [4] that a useful order can not be derived from the set of characters.
For this reason we introduced a new type of context free grammars, the two-dimensionally fuzzy attributed CFGs:

- a context free grammar G = (Σ, N, P, S) describes the syntax of the input language, f.i. how syntactic categories can be combined

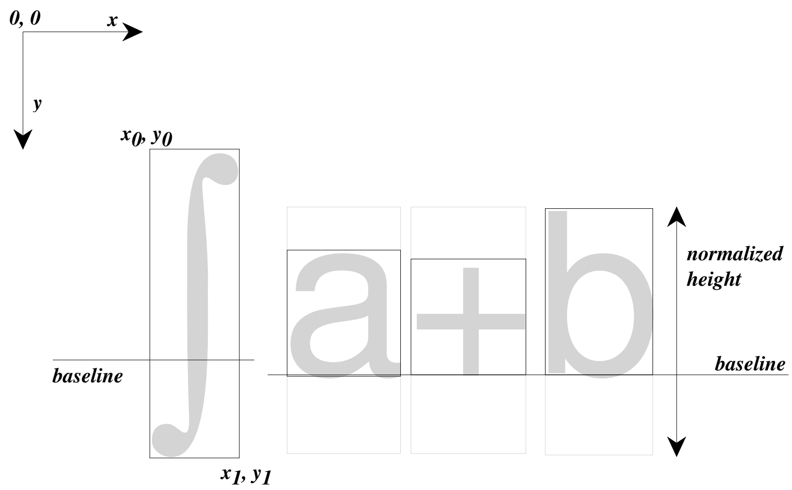

Fig. 4: Set of attributes used in the present system

- a set of geometric attributes is used to describe the arrangment of the characters /subformulas in the plane. Fig. 4 shows the attributes currently used in our system. ***x0, y0, x1, y1*** define the bounding box of each subformula and are received directly from the character recognition process, whereas ***baseline*** and ***normalized-height*** are derived from the bounding box and apriori knowledge about the alphabet in use.

- fuzzy context conditions describe the space of possible arrangements of the subformulas in a rule. Instead of using relations to define geometric constraints, for each rule r ∈ P

$$r: \quad N_0 \leftarrow N_1 \ldots N_n \qquad N_0 \in N,\ N_1 \ldots N_n \in N \cup \Sigma$$

there exists a fuzzy evaluation function μ_r

$$\mu_r: \quad A^n \rightarrow [0..1]$$

A is the set of possible attributes values

f. e. $A = Z^6 \times [0..1]$ in our system

which defines the fuzzy set of accepted arrangments of the components $N_1 \ldots N_n$. That means for a given value $a \in A$, (i.e. for a given placement of $N_1 \ldots N_n$) the function μ_r gives a measure for "how good the arrangement of $N_1 \ldots N_n$ matches the mathematical notation described by rule r". Fig. 5 shows an example for such a μ_r function.

r: Factor → Factor Expression

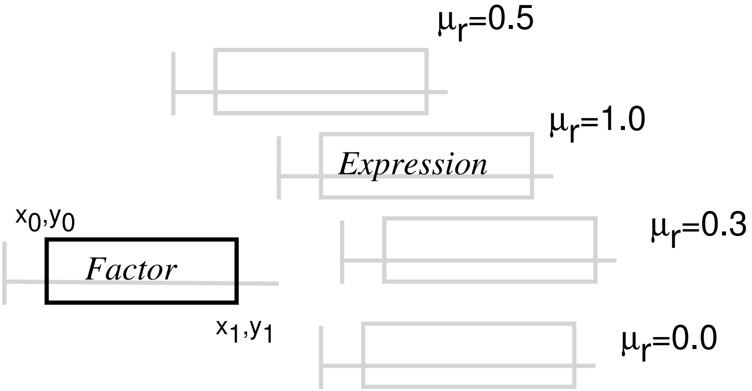

Fig. 5: Four possible arrangements of an exponent expression and the corresponding values of the $\mu_{exponentiation}$ function

The fuzzy measurement of subformular arrangements was choosen to match human perception (f.i. humans don't know a strict limit for the possible positions of an exponent expression)

3.3.2 Parsing Strategy

In addition to our new type of CFG we had to find a parsing strategy suitable for the 2-dimensionally fuzzy attributed grammars. As further requirements we wanted to find all possible interpretations of an input (see example in Fig. 6), propose maximum solutions in case of doubt or error and we wanted the possibility to run the parsing process "enpassant".

$$\sum_{i=0}^{n} a^2 + b \quad \text{can be} \quad \sum_{i=0}^{n} (a^2 + b) \quad \text{or} \quad \left(\sum_{i=0}^{n} a^2 \right) + b$$

Fig. 6: ambiguity in the parsing process

In a first step these requirements led to a generalization of the bottom-up Cocke-Younger-Kasami parser, not making use of the consecutiveness of input symbols but using exhaustive search instead:

Step 1 We start with the set of characters produced from the recognition process as set of subformula interpretations

Step 2 **Each** disjunct subset of subformula interpretations found so far is tested against each rule in the grammar.

Step 3 **If** the subset matches the rule and if the corresponding μ_r-function gives a satisfying high value, **then**

Step 4 a new subformula with the syntactic category of the lefthand side of the rule is generated and added to the set of subformulas. By not removing the combined subformulas from the set of interpretations we guarantee that concurrent interpretations can be found.

Step 5 The process of selecting, testing and combining is continued **until** all possible combinations have been made.

Step 6 All interpretatations covering the whole set of input characters are presented to the user as possible interpretations.

As one can imagine, this is a very slow algorithm depending on the sharpness of the fuzzy sets. Therefore we made some basic refinements, f.e.

- the grammar is converted in a normalized form similar to Chomsky NF.
  $\rightarrow$ Only binary rules have to be checked. This drastically reduces search time since only pairs of trees have to be searched and tested instead of sets of trees

- instead of testing a randomly selected pair, the algorithm always checks the pair consisting of the subtrees with maximum fuzzy measurement μ
  $\rightarrow$ optimized search strategy that prefers "optimal" written input
  $\rightarrow$ less memory is needed for bookkeeping of the tests just performed

- not all the rules in the grammar are checked from the beginning. First simple or linear rules are tested (such as string concatenation, decimal numbers, accentuated symbols etc.). Rules describing 2-dimensional notations are checked later.
  $\rightarrow$ further reduction of the tests to be performed

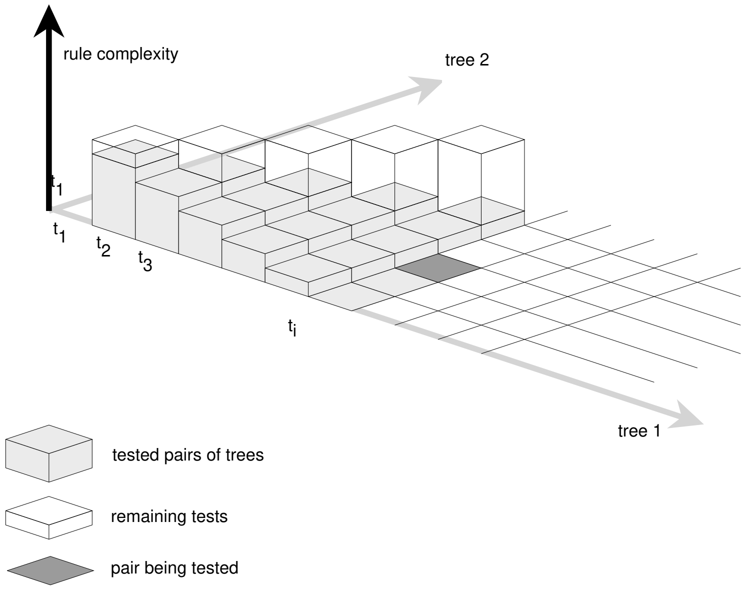

Fig. 7: optimized parsing strategy

References

[1] R. Anderson,
An online Mathmatics system Using Twodimensional handprinted Notation
Rand Corp., Santa Monica 1970

[2] D.Rumelhart, J. Mc Clelland,
Paralell distributed processing
MIT Press 1988

[3] Joachim Quapp
Ein- und Mehrschreiber-Systeme zur Handschrifterkennung mit Neuronalen Netzen
Diplomarbeit, Universität des Saarlandes, Saarbrücken 1993

[4] R. Marzinkewitsch,
Ein Arbeitsplatz zum computerunterstützten, handschriftlichen Rechnen mit mathematischen Formeln
Dissertation, Universität des Saarlandes, Saarbrücken 1990

[5] Frank M. Weigel
Unscharf 2-dimensional attributierte Grammatiken
Diplomarbeit, Universität des Saarlandes, Saarbrücken Herbst 1994