# Recognition of On-Line Handwritten Commutative Diagrams

Andreas Stoffel
Universität Konstanz
FB Informatik und Informationswissenschaft
Universitätsstr. 10
D-78457 Konstanz, Germany
andreas.stoffel@uni-konstanz.de

Ernesto Tapia, Raúl Rojas
Freie Universität Berlin
Institut für Informatik
Takustr. 9
D-14195 Berlin, Germany
{ernesto.tapia,raul.rojas}@fu-berlin.de

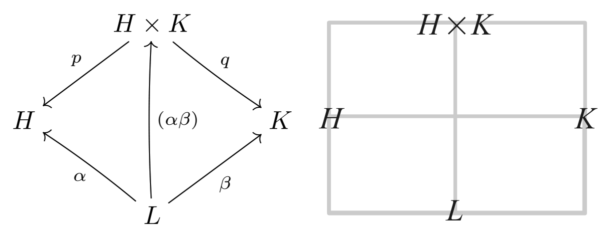
## Abstract

*We present a method for the recognition of on-line handwritten commutative diagrams. Diagrams are formed with arrows that join relatively simple mathematical expressions. Diagram recognition consists in grouping isolated symbols into simple expressions, recognizing the arrows that join such expressions, and finding the layout that best describes the diagram. We model the layout of the diagram with a grid that optimally fits a tabular arrangement of expressions. Our method maximizes a linear function that measures the quality of our layout model. The recognition results are translated into the LaTeX library xy-pic.*

## 1. Introduction

Commutative diagrams represent graphically functional relations between mathematical objects from category theory, algebraic topology and other areas. The diagrams are formed with arrows that join relatively simple mathematical expressions. Arrows have diverse shapes depending on the kind of function they represent. For example, an arrow with a double shaft represents an isomorphism. The objects the arrows join are usually two sets, starting from function's domain and ending in the function's range.

One of the most popular tools to typeset commutative diagrams is the LaTeX-library xy-pic. Constructing diagrams using xy-pic consists basically in typing LaTeX-math expressions in a tabular arrangement [1, 3]. The elements in the table are LaTeX expressions that represent the sets in the diagram, and a special coding represents the arrows that join them. The coding of the arrow includes also other mathematical expressions that represent the arrows' labels. See Fig. 1. Typesetting commutative diagrams is a tedious and difficult task; even relative small diagrams require long and complex LaTeX code. Such a complex code is not only




```
\xymatrix{ {} & {H \times K }\ar@/_0.1pc/[1,1]^-{q } \ar@/^0pc/[1,-1]_-{p } &
    {} \\ {H } & {} & {K } \\ {} & {L }\ar@/_0pc/[-1,1]_-{\beta } \ar@/^0.1
   pc/[-2,0]_-{\left(\alpha \beta \right)} \ar@/_0.1pc/[-1,-1]^-{\alpha } &
    {} }
```

**Figure 1. Above: A simple commutative diagram and the $3\times3$-grid (grey lines) that fits the expressions. The expressions not included in the grid are the arrows' labels. Below: The xy-pic code used to construct the diagram.**

characteristic of xy-pic but almost of all available typesetting libraries [1].

In order to help users to overcome such difficulties, we developed a method that recognizes commutative diagrams from on-line handwriting and generates automatically their LaTeX code. The recognition of handwritten commutative diagrams consists in groping symbols into simple mathematical expressions, finding how the arrows connect them, and constructing the layout that best describes the diagram. The latter is accomplished by locating the groups in a tabular arrangement to create the corresponding LaTeX-code. The method we describe in this article is an extension of our previous research on recognition of mathematical notation [4, 5, 6].

## 2. Recognition of commutative diagrams

In order to simplify the diagram recognition and the LaTeX code generation, three basic assumptions are made.

978-0-7695-3725-2/09 $25.00 © 2009 IEEE
DOI 10.1109/ICDAR.2009.122

The first assumption is that the handwritten strokes are already grouped into recognizable objects, which represent mathematical symbols or arrows. The information we keep from the objects are their identity obtained using a classifier [4], and their position and size encoded as bounding box.

The second assumption concerns the position of mathematical expressions and arrows: each expression is associated with al least one incoming or outcoming arrow. This assumption usually holds for the majority of commutative diagrams, because connections between expressions are the relationships that the diagrams express. We can ignore expressions without any apparent relationship to other ones without modifying the recognition of diagram.

The third assumption affects the layout of the mathematical expressions. We assume that the expressions are arranged in a tabular layout. This assumption simplifies the recognition as well as the LaTeX-code generation. This assumption does not hold for all commutative diagrams, but any diagram can be rewritten satisfying it.

Our algorithm for recognizing commutative diagrams consists of three main steps. The first step segments the handwritten strokes into arrows and non-arrows (mathematical symbols). The second step creates an initial grouping of the symbols, which are located inside the grid cells or outside forming arrow labels. The third step combines the initial grouping into mathematical expressions and reconstruct the tabular arrangement that best fits them. The following sections explain these steps in detail.

## 3. Arrow recognition

We recognize arrows using a method similar to Kara's and Stahovich's [2]. They suppose that arrows are formed with a continuous stroke or with two strokes. Both arrow shapes have five characteristic points that determine the arrow's shaft and head, see Fig. 2. If the arrow is formed with one stroke, then the points $B$, $C$ and $D$ are the last three stroke's corners, $E$ is the stroke's last point, and $A$ is a point in the stroke that lies at a short distance from $B$. If the arrow is formed with two strokes, $B$ is the last point of the first stroke, $C$ is the first point in the second stroke, and the other points are as before. Note that the analysis concentrate on an area that contains the arrow's head, so it allow us to recognize arrows with a great variety of shaft's shapes. Below we explain how one can decide whether one or two strokes form an arrow after locating these characteristic points.

In contrast to Kara and Stahovich, we use a classifier to recognize arrows. As preprocessing step, we first rotate all points so that the segment $\overline{AB}$ is horizontal and $A$ lies at the left of $B$. Then we exchange the points $C$ and $E$, if $C$ lies below the line $\overline{AB}$. Finally, we scale all points so

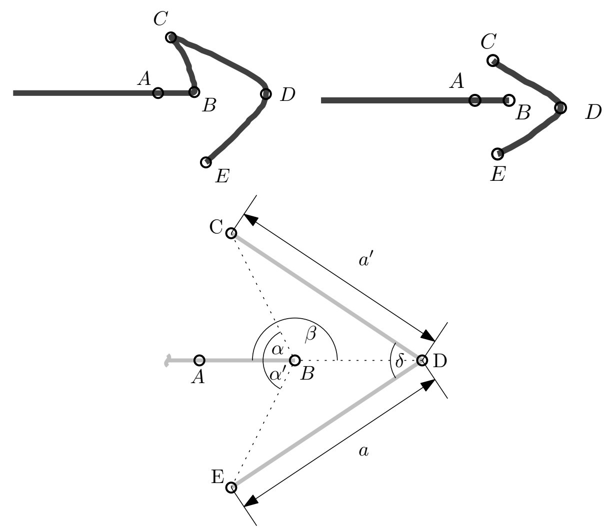



**Figure 2. Above: The five characteristic points that define arrow shapes. Below: Numerical features used for arrow recognition**

their bounding-box fits the square $[-0.5, 0.5] \times [-0.5, 0.5]$. The numerical features we extract after preprocessing are the coordinates of the five points, and the angles $\alpha$, $\alpha'$, $\beta$ and $\delta$, and the lengths $a$ and $a'$ as indicated in Fig. 2.

Given the stroke $(p_1, \ldots, p_i, \ldots, p_n)$, we developed a method to indentify a characteristic points $p_i$ as corner. It computes the angles $\alpha_k$ formed by $p_i$ and its neighboring points $p_{i-k}$ and $p_{i+k}$, for $k = 1, \ldots, \ell$. If $\alpha_k$ exceeds a given threshold, then it is marked as corner angle. If the number of corner angles is greater than the non-corner angles, we mark $p_i$ as corner. Some times, however, the same corner is located at a small number of consecutive points. We overcome this problem by substituting consecutive corner points with their center of mass, if they lie within a given small neighborhood.

#### Recognition of dashed shafts

Since some commutative diagrams also include arrows with dashed shafts, we developed a method to group a sequence of strokes into a shaft. Our method assumes that the user draws all segments in a dashed line at once. This assumption allow real-time grouping.

We use a classifier to decide whether a given stroke form part of a dashed line. If that is true, the current stroke is joined to the last one to extend the dashed line. In the other case, the dashed line is considered as "closed" and the current stroke is a candidate to generate a new dashed line.

The classifier considers the current stroke and the last $n$ strokes in the dashed line. For each considered stroke, we compute the angles formed between middle, starting and

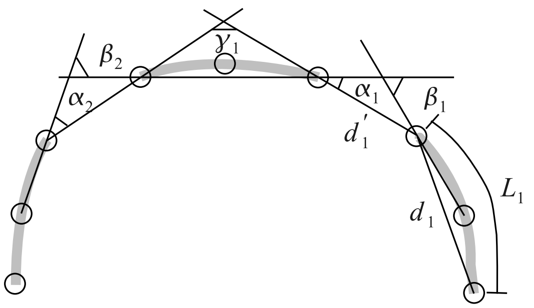



**Figure 3. Features used to recognize dashed lines. Here we consider only two previous strokes.**

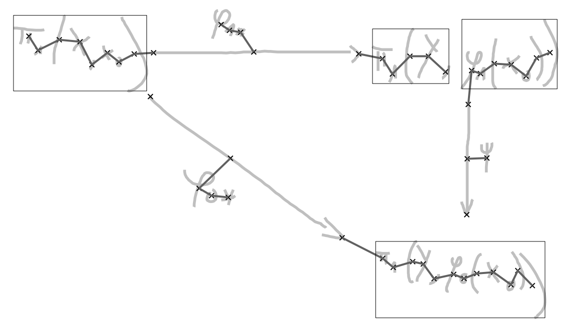


**Figure 4. An example of an initial groping for a commutative diagram.**

end points as illustrated in Fig. 3. We also consider for every stroke in the $n$-sequence its length, the distance between its starting and end points, and the distance between its starting point and the end point of the previous stroke. Since not all dashed lines are formed with $n$ strokes, we consider also another feature that indicates whether the previous $n$ strokes exist. If the $i$-th stroke exists, $i = 1, \ldots, n$, then the $i$-th is set to one and zero in the other case.

## 4. Recognition of diagram's layout

### Computation of the initial grouping

This step generates the initial grouping using the *Minimum Spanning Tree* (*MST*) of a weighted graph. The nodes of the graph are the mathematical symbols and the points located at the middle, start and end of the arrow's shafts. The edge weight between a shaft point and a symbol is the Euclidean distance between the point and the center of the symbol's bounding-box. The weight of edges joining shaft points is defined as zero, even for points from different arrows. Using Prim's algorithm, the MST is build up starting with a shaft point. With our definition of the edge weight, the shaft points are added the MST at first and afterwards the algorithm adds the symbols to the tree.

After the MST construction, there are symbols that are a sibling of exactly one arrow point. Such symbols are used to initialize a grouping that is constructed recursively, by joining new symbols connected to symbols in the current grouping until an arrow is reached or no more symbols are found. Symbols that are siblings of middle points are used to initialize groupings that describe arrow's labels. Figure 4 shows the results of this step. Note that an actual expression in the upper right is split into two symbol groups. Since the groups created in this step are not split in further processing, we assume that a group does only contain symbols that belong to the same mathematical expression.

### Diagram reconstruction

The second step uses the initial groupings together with our third assumption. It constructs a hypothesis that assigns to each grouping a row and a column of the tabular layout. Groupings assigned to the same row and column must be located in the same grid cell to form one mathematical expression. The initial hypothesis assigns each group the rows and columns in increasing order of the $x$ and $y$-coordinates of the groupings' bounding boxes. We define the error function $\mathrm{e}(h)$ to measure the quality of a given hypothesis $h$. Algorithm 1 generates new hypothesis that iteratively decrease the error function $e(h)$.

**Algorithm 1**: Algorithm to find the best derived hypothesis

**Input**: Hypothesis h
**Output**: Best derived Hypothesis

result $\leftarrow$ h;
error $\leftarrow$ e (h) ;
**foreach** *adjacent column pair* $(i, j)$ *of* h **do**
    temp $\leftarrow$ h.`merge` $(i, j)$;
    **if** e (temp) $<$ error **then**
        result $\leftarrow$ temp;
        error $\leftarrow$ e (temp);
    **end**
**end**
**foreach** *adjacent row pair* $(u, v)$ *of* h **do**
    temp $\leftarrow$ h.`merge` $(u, v)$;
    **if** e (temp) $<$ error **then**
        result $\leftarrow$ temp;
        error $\leftarrow$ e (temp);
    **end**
**end**
**return** result

Once the algorithm converges, we found the best hypothesis. Groupings in the hypothesis that fall into the same column and row form a single mathematical expression. Finally, we have only to assign the expressions to the arrows. The elements in the diagram that the arrows connect are the mathematical expressions nearest to some starting or ending point of arrows shatfs. Expressions nearest to middle shaft points form the labels of the corresponding arrow. After we assign the expressions to the arrows, we apply our method [5] to convert into LaTeX the mathematical expressions, and then we create the xy-pic code of the diagram.

**Error function**

The error function $\mathrm{e}(h)$ evaluates the quality of the current hypothesis $(h)$. The error function is a linear combination of geometrical features $f(h)$ calculated for the hypothesis $h$:

$$\mathrm{e}(h) = \sum_i \alpha_i \cdot \mathrm{f_i}(h). \tag{1}$$

The next paragraphs describe each of the functions $f_i$ used in our evaluation and based on column features. These functions depend of the following “global” values: the diagram’s width diagram $w_d$ and height $h_d$, and the average symbol width $\overline{w_s}$.

The first feature is the *Number of Columns* of the hypothesis. With an increasing number of columns, the probability to construct a wrong layout increases. Therefore, this features forces the algorithm to use less columns.

$$\mathrm{f}_1(h) = \frac{\overline{w_s}}{w_d} \cdot |h.\mathrm{cols}| \tag{2}$$

The *Maximal Column Width* evaluates the width of the widest column. For an ideal diagram this feature will dramatically increase, when two groups of symbols from different columns are assigned to the same one.

$$\mathrm{f}_2(h) = \frac{1}{\overline{w_s}} \cdot \max_{c \in h.\mathrm{cols}} \mathrm{width}(c) \tag{3}$$

The *Maximal Inner Column Distance* of a column $\mathrm{d_i}$ is computed from the distances between the projection of the symbols two the $x$-axis. This feature evaluates the position of the symbols in the columns. For larger gaps in the projections this feature will grow and therefore force the columns to have a compact arrangement of the symbols with respect to the projection.

$$\mathrm{f}_3(h) = \frac{1}{\overline{w_s}} \cdot \max_{c \in h.\mathrm{cols}} \mathrm{d_i}(c) \tag{4}$$

The *Minimal Outer Column Distance* of a column $\mathrm{d_o}$ is the smallest distance of the column to the adjacent columns. It enforces a minimal distance between adjacent columns.

$$\mathrm{f}_4(h) = \max(k_o - \frac{1}{\overline{w_s}} \cdot \min_{c \in h.\mathrm{cols}} \mathrm{d_o}(c), 0) \tag{5}$$

**Table 1. Recognition rates for commutative diagrams.**

| | Accuracy |
|---|---|
| Objects | 284 (97.59 %) |
| Rows | 113 (94.96 %) |
| Columns | 157 (97.52 %) |
| Diagrams | 49 (92.45 %) |

The *Use of Space* evaluates an approximation for the unused space of a column. The error for an hypothesis with columns that contain unused space will increase by this feature. The approximation ignores the possible overlapping of symbols. Usually overlapping of symbols is small and seldom.

$$\mathrm{f}_5(h) = \frac{1}{w_d \cdot h_d} \cdot \max_{c \in h.\mathrm{cells}} \left( area(c) - \sum_{s \in c.\mathrm{symbols}} area(s) \right) \tag{6}$$

Please note that the error function also involves the features $\mathrm{f}_6, \ldots, \mathrm{f}_{10}$ associated with the rows. They are defined similarly to $\mathrm{f}_1, \ldots, \mathrm{f}_5$ by replacing $x$ with $y$, “column” with “row”, and “width” with “height”.

## 5. Experimental results

We used 53 commutative diagrams written from 3 different persons to evaluate our method. We choose the diagrams from publications and books about homological and topological algebra. The database contains in total 119 rows, 161 columns and 291 mathematical objects. We evaluated a mathematical object as correctly recognized, when all its symbols are grouped into the same expression. A column or row is correctly recognized when all its objects are recognized correctly. A diagram is correctly recognized, when all its columns and rows are recognized correctly. The coefficients $\alpha_i$ of the error (1) were determined manually. Table 1 shows the accuracy of the method for objects, rows, columns and diagrams.

We also used our 53 diagrams to evaluate our arrow recognizers. They contain 3785 strokes that form 1874 arrows. The classification of arrows reached a recognition rate of 98.24 %. Since the database contains a reduced number of arrows with dashed shafts, we redraw the half of the diagrams using dashed arrows, generating in total 4911 strokes. In this case, we reached a recognition rate of 97.07 %. We used in both cases a neuronal network from the Weka library [7]. The network for arrow classification has 18 input neurons, 10 hidden neurons and two output neurons. The network for the dashed shafts has 26 input neurons, 14 hidden neurons and two output neurons. We

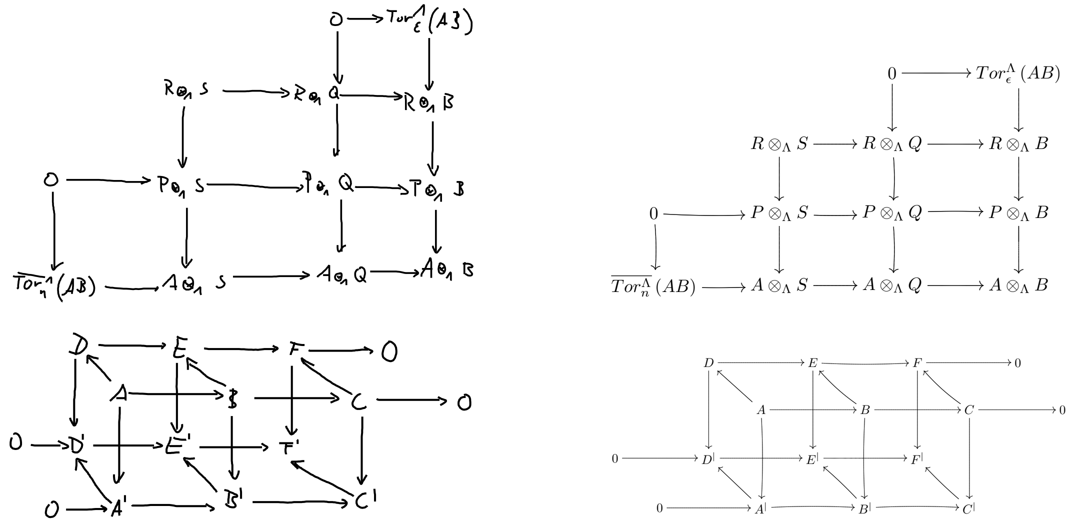


**Figure 5. Two diagrams correctly recognized.**

used the default values of Weka for learning rate, momentum and number of epochs.

## 6. Comments and further work

We presented a new method for the recognition of on-hand written mathematical diagrams. We used an optimization approach that considers local and global features of the groupings in the diagram. This approach is more robust and handles local writing irregularities. Figure 5 shows two examples of recognized diagrams. In our experiments, the start and end points of the arrows were always assigned to the correct expression. Therefore, the semantic of a diagram is not affected by a miss-assignment of the arrows to the expressions. As long as the symbols are assigned correctly to the expressions, the semantic of the diagram is not changed whether or not an expression is assigned to the wrong row or column. Problematic result the diagrams that do not satisfy the assumptions about the arrangement of the objects. Other causes of errors are the distances between columns or rows. When these distances are smaller than the distances between symbols within an object, the initial grouping of symbols leads to errors. In these cases the recognition of the corresponding rows and columns were also affected.

We also presented a method for the recognition of arrows and dashed lines. In both cases we had problems to discriminate mathematical symbols, because the current implementation works without symbol recognizers; it differentiates between arrows and non-arrows. We are sure that the recognition rates will improve, in particular the recognition of dashed lines, when using some symbol classifier or contextual information from the diagram. Our voting method used to locate corners in strokes is robust to small irregularities. The classification rates for arrow recognition show indirectly that the voting method was well enough to recognize the characteristic points of arrows.

Since our method uses only the symbols' identity, size and position, our method could be easily extended to recognize off-line commutative diagrams. We consider that our method is general enough to recognize diagrams with a tabular layout similar to commutative diagrams.

## References


[1] G. V. Feruglio. Typesetting commutative diagrams. *TUGboat*, 15(4):466–484, 1994.

[2] L. B. Kara and T. F. Stahovich. Hierarchical parsing and recognition of hand-sketched diagrams. In *UIST '04: Proceedings of the 17th annual ACM symposium on User interface software and technology*, pages 13–22, New York, NY, USA, 2004. ACM Press.

[3] J. S. Milne. Guide to commutative diagram packages, 2006.

[4] E. Tapia and R. Rojas. Recognition of on-line handwritten mathematical formulas in the e-chalk system. In *Seventh International Conference on Document Analysis and Recognition*, pages 980–984, 2003.

[5] E. Tapia and R. Rojas. Recognition of on-line handwritten mathematical expressions using a minimum spanning tree construction and symbol dominance. In J. Lladós and Y.-B. Kwon, editors, *GREC*, volume 3088 of *Lecture Notes in Computer Science*, pages 329–340, 2004.

[6] E. Tapia and R. Rojas. Recognition of on-line handwritten mathematical expressions in the e-chalk system-an extension. In *Eighth International Conference on Document Analysis and Recognition*, pages 1206–1210, 2005.

[7] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2nd edition, 2005.