



CENTER FOR
MACHINE PERCEPTION



CZECH TECHNICAL
UNIVERSITY IN PRAGUE

PhD THESIS

ISSN 1213-2365

Structure and Its Meaning for Recognition

(PhD Thesis Proposal)

Ing. Martin Bresler

breslmar@fel.cvut.cz

CTU–CMP–2013–19

August 5, 2013

Available at
<ftp://cmp.felk.cvut.cz/pub/cmp/articles/bresler/Bresler-TR-2013-19.pdf>

Thesis Advisor: prof. Ing. Václav Hlaváč, CSc.,
RNDr. Daniel Průša, Ph.D.

The author was supported by the Project SGS13/205/OHK3/3T/13
and by The Technology Agency of the Czech Republic under Project
TA01031478.

Research Reports of CMP, Czech Technical University in Prague, No. 19, 2013

Published by

Center for Machine Perception, Department of Cybernetics
Faculty of Electrical Engineering, Czech Technical University
Technická 2, 166 27 Prague 6, Czech Republic
fax +420 2 2435 7385, phone +420 2 2435 7637, www: <http://cmp.felk.cvut.cz>

Structure and Its Meaning for Recognition

Ing. Martin Bresler

August 5, 2013

Contents

Abbreviations	2
1 Introduction	3
2 Devices for Ink Input	4
3 Types of Useful Diagrams	5
3.1 Flowcharts	5
3.2 Unified Modeling Language (UML)	6
3.3 Business Process Diagram	7
3.4 Concur Task Tree (CTT)	7
4 State of the Art	10
4.1 Preprocessing	11
4.1.1 Primitives Decomposition	11
4.1.2 Merging of Overlaying Strokes	12
4.2 Segmentation	12
4.2.1 Graphical / Textual Strokes Distinguishing	12
4.2.2 Separate symbols segmentation	13
4.3 Symbol Recognition	15
4.3.1 Feature Extraction	15
4.3.2 Rule-based Classifiers	16
4.3.3 Nearest Neighbour	16
4.3.4 Statistical Classifiers	16
4.3.5 Structural Classifiers	17
4.4 Structural Analysis	18
4.4.1 Grammars	18
String Grammars	18
Graph Grammars	21
Adjacency Grammars	23
4.4.2 Graphical Models	24
4.4.3 Dynamic Programming	25
4.5 Commercial Products	25
5 Own Research	28
5.1 Symbol Candidates Detection	28
5.2 Structural Analysis	29
5.3 Text / Non-text Stroke Classification	31
6 Thesis Proposal	33
Bibliography	35

Abbreviations

AGG	Attributed Graph Grammar
BMRM	Bundle Method for Regularized Risk Minimization
BN	Bayesian Net
BPD	Business Process Diagram
CRF	Conditional Random Field
CTT	Concur Task Tree
CVWW	Computer Vision Winter Workshop
DCT	Discrete Cosine Transformation
DP	Dynamic Programming
DPSO	Discrete Particle Swarm Algorithm
GXL	Graph Exchange Language
HCI	Human Computer Interaction
HMM	Hidden Markov Model
ICDAR	International Conference on Document Analysis and Recognition
ILP	Integer Linear Programming
LP	Linear Programming
MAP	Maximum A posteriori Probability
ML	Maximum Likelihood
MLP	Multi-Layer Perceptron
MRF	Markov Random Field
OMG	Object Management Group
PC	Personal Computer
PCA	Principle Component Analysis
PDL	Picture Description Language
RANSAC	RANdom SAmple Consensus
RNN	Recurrent Neural Network
SVM	Support Vector Machine
UML	Unified Modeling Language

1 Introduction

It is natural to express thoughts using drawings for humans. Sometimes it is simply easier to draw a picture to represent a specific problem. This illustration is called a diagram and it is a two-dimensional geometric symbolic representation of information according to some visualization technique. Sometimes, the technique uses a three-dimensional visualization which is then projected onto the two-dimensional surface. The word graph or graphics is used as a synonym for diagram at times. If we want to recognize a diagram, we should exploit its structure to achieve a sufficient accuracy.

The aim of my doctoral research is to reveal a way how to benefit maximally from the structural information of diagrams for an effective implementation of algorithms for their recognition. This document presents my view of the research in the field related to diagram recognition. The state of the art methods currently available are reviewed too. The PhD thesis proposal follows.

It is important to note that the most natural way of creating a drawing is just to use an ordinary paper and a pencil. The drawing can be then put into the computer system by scanning it. Recognition of scanned images is called *off-line recognition*. It is more difficult than so called *on-line recognition* performed on data acquired using a tablet. The reason is that a tablet provides more information, such as temporal information or pen pressure values. Moreover, a tablet offers additional functionalities like suggestions or automatic corrections. This work deals only with the on-line recognition.

Diagram recognition is a difficult task consisting of many steps. A lot of presented related work deals just with a small aspect of the whole problem. Researchers often concentrate on preprocessing, segmentation, or symbol classification only and proposed methods are often domain specific. On the other hand, approaches used for structural analysis are often introduced theoretically only and practical evaluation is limited. Grammar based methods are powerful to express complex recursive structures, but their parsing might be difficult. Authors of the papers in this field sometimes neglect difficulties connected to the practical use of the methods. I would like to develop a complete system for diagram recognition, which will be practically usable. I see an opportunity in studying possibilities how to model relations between symbols of diagrams in an easier way. From a certain point of view, grammars represent an overkill for diagram recognition.

The rest of the document is organised as follows. First, I will introduce devices usable for ink input imposing different possible applications. Second, I will present various diagrams, which are commonly used by users in different fields. A survey of the state-of-the-art methods in the field of handwriting recognition is given, followed by an introduction of my own research. Finally, I will conclude and propose my future work towards the PhD thesis.

2 Devices for Ink Input

There exist various devices used for human-computer interaction (HCI) allowing to input drawings directly into the computer. The simplest way is just to use a mouse, which is common for every user. However, drawing with a mouse is not natural and is highly undesirable for professionals. The most common solution for professionals nowadays is a tablet or tablet PC. The best is a tablet with a display on a drawing surface (or the mentioned tablet PC) immediately showing the drawing, which gives the user the real-time feedback. The drawing is even more natural and precise. A smartboard is a whiteboard with a projector. Although it is less precise, it allows to show the drawing to the audience immediately. Therefore, it is popular in schools or during brainstormings. New devices for HCI processing natural human expressions have been emerging last days. One of them is, for example, Kinect [Cor12] or even more precise Leap [Mot12]. These devices allow to track fingers movement and thus can be used for drawing. Although the precision may be high, the feedback is not displayed exactly on the drawing surface which is not natural. Moreover, it is inconvenient to draw with your arm freely hanging in the air for a long time. All mentioned devices are shown in Figure 2.1 for illustration. After all, it has been the best to use tablet or tablet PC (with stylus), which is the standard today. On the other hand, new uncommon devices reveal a new area of possible applications. I will assume in my work that the user is using this kind of device. Nevertheless, all algorithms work similarly with any other device mentioned above.



Figure 2.1 Examples of current devices usable for on-line recognition of diagram drawings.

3 Types of Useful Diagrams

People have been using diagrams since ancient times. Diagrams consist of symbols connected into greater complex schemas. In the most simple cases, symbols or even the whole structure of the diagram are created and represented intuitively without a fixed syntax. Since the diagrams became more complicated and computer processing is required, it started to be important to have unified notation, which makes possible that everyone knowledgeable of the notation can read the diagram. Sophisticated diagrammatic notations were invented. Although all of the diagram classes have a lot in common, there are some differences. Also, each class of diagrams is used by users working in different disciplines. Some of the diagrams have a textual form, like mathematical formulae, and some of them are rather set of pictograms, like UML diagrams. A lot of research has been done in recognition of mathematical formulae, electrical circuits, music notations, engineering drawings, etc. In this work, I will not consider these kinds of notations and I will rather concentrate on the diagram composed of pictograms or glyphs. In this chapter, I introduce main diagram classes interesting for my research and computer recognition in general with possible applications.

3.1 Flowcharts

Flowcharts are simple and the most general diagrams used to express arbitrary algorithms or processes. They are often exploited by people of different specialization and even by laymen. Its syntax is straightforward and six symbols presented in Figure 3.1 are the most common ones. Although, there exist many more symbols and some software for flowchart design allow the user to define custom ones.

The most popular software for flowchart creation are Edraw Flowchart and Microsoft Visio. However, it is possible, for example, to create simple flowcharts directly in Microsoft Powerpoint. There also exist mobile applications like Smart Diagram for Android. Some of applications offer basic recognition engine which is able to recognize separated hand-drawn symbols. An application capable of recognizing a whole diagram with its structure is still missing and would be very appreciated. An add-in for Microsoft Powerpoint might be appropriate, because flowcharts are frequently parts of various presentations.






Symbol	Name	Function
	Start/end	An oval represents a start or end point.
	Arrows	A line is a connector that shows relationships between the representative shapes.
	Input/Output	A parallelogram represents input or output.
	Process	A rectangle represents a process.
	Decision	A diamond indicates a decision.

Figure 3.1 Five most frequently used flowchart symbols (taken from SmartDraw webpage).

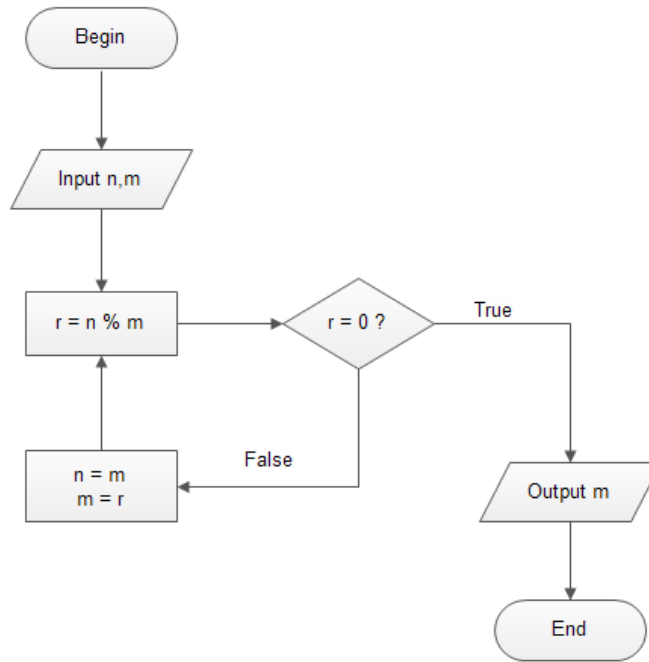


Figure 3.2 Flowchart examples.

3.2 Unified Modeling Language (UML)

UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created, by the Object Management Group (OMG). It was first added to the list of OMG adopted technologies in 1997, and has since become the industry standard for modeling software-intensive systems. UML includes a set of graphic notation techniques to create visual models of object-oriented software-intensive systems. UML standard contains many types of diagrams used to express different aspects of the software development. Diagrams are classified in three groups: Structure diagrams, Behaviour diagrams, and Interaction diagrams. The whole hierarchy of the UML diagrams is shown in Figure 3.3. Although many kinds of diagram exist within UML standard, only few of them are really extensively used by developers. The most common and the most important are the class diagrams and the activity diagrams. The examples of them are shown in Figure 3.4.

UML is used in two situations at least, either as a documentation or as a tool to express thoughts to colleagues. There exist various software for creating UML diagrams. The most commonly used tools are Enterprise Architect and Rational Rose. They are often used to create the documentation.

It would be possible to create an add-on for one of these applications capable of recognizing and transferring handwritten diagram into the inner representation of the software. However, it is easier for developers to implement a piece of code first and then use so called reverse engineering when UML diagrams are generated according to the existing code. In that case, the recognition tool would not be very appreciated. The other case is so called assisted engineering when a developer is drafting a diagram and the application recognizes the structure and gives options to the developer how the non-finished diagram should continue. It should help developers to quickly express their thoughts and present them to other developers.

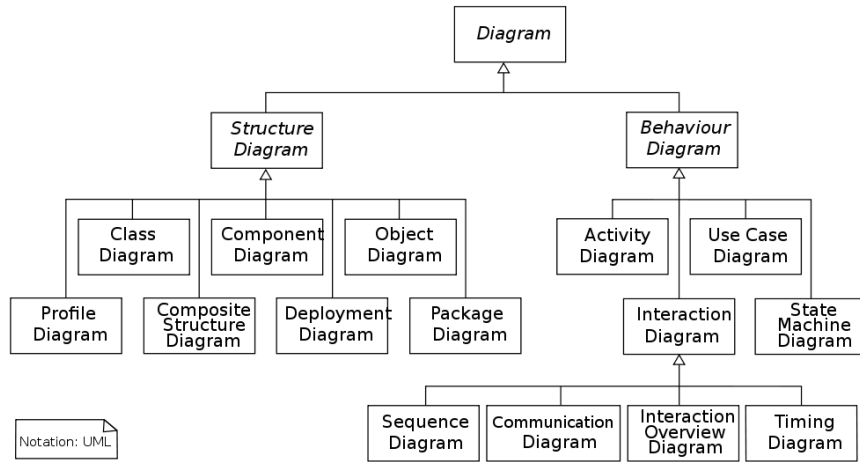


Figure 3.3 Hierarchy of all UML diagrams (taken from Wikipedia).

3.3 Business Process Diagram

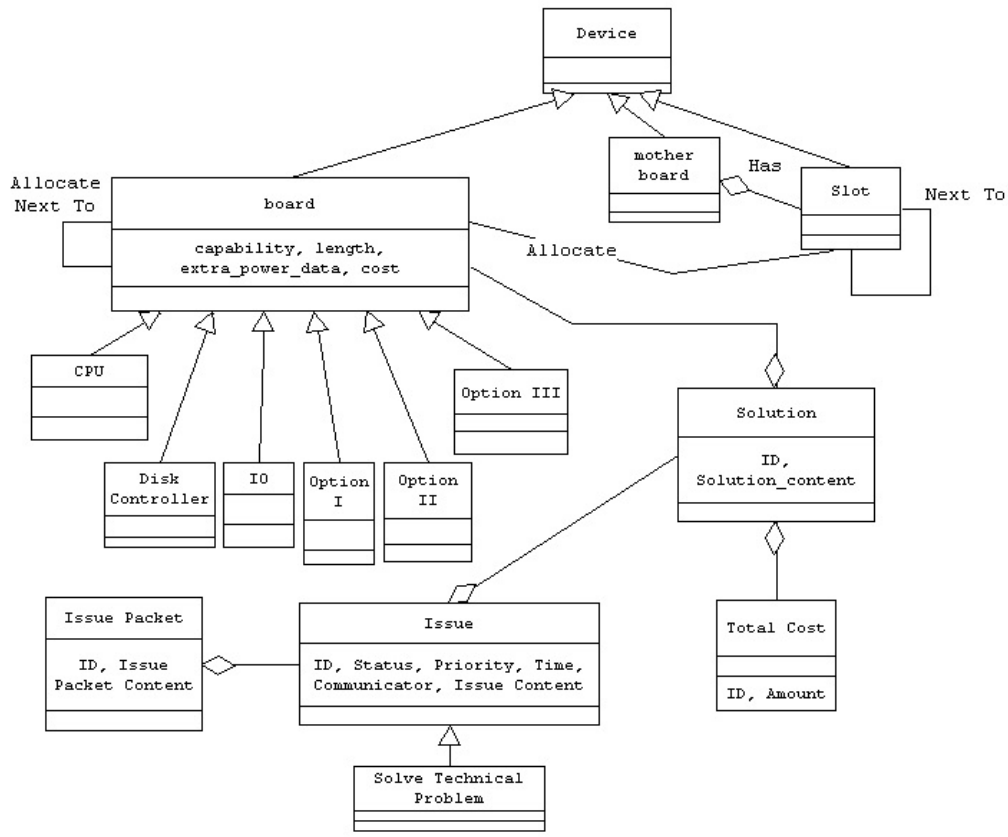
Business Process Diagrams (BPD) are used in Business Process Modeling (BPM) which is the activity representing processes of an enterprise in systems engineering, so that the current process may be analyzed and improved. BPM is typically performed by business analysts and managers who are seeking to improve the process efficiency and quality. The diagram itself is very similar to the flowcharts. The main difference is that business analysts and managers can not use inverse engineering and therefore the manual creation of diagrams is important for them. Again, there exist software for business process diagram creation (for example, IBM Rational). The possible application could be a plug-in for MS PowerPoint where a handwritten diagram would be recognized, graphically expressed, and transferred to the presentation. Diagrams are very important for managerial presentations. The example of a business diagram can be seen in Figure 3.5.

3.4 Concur Task Tree (CTT)

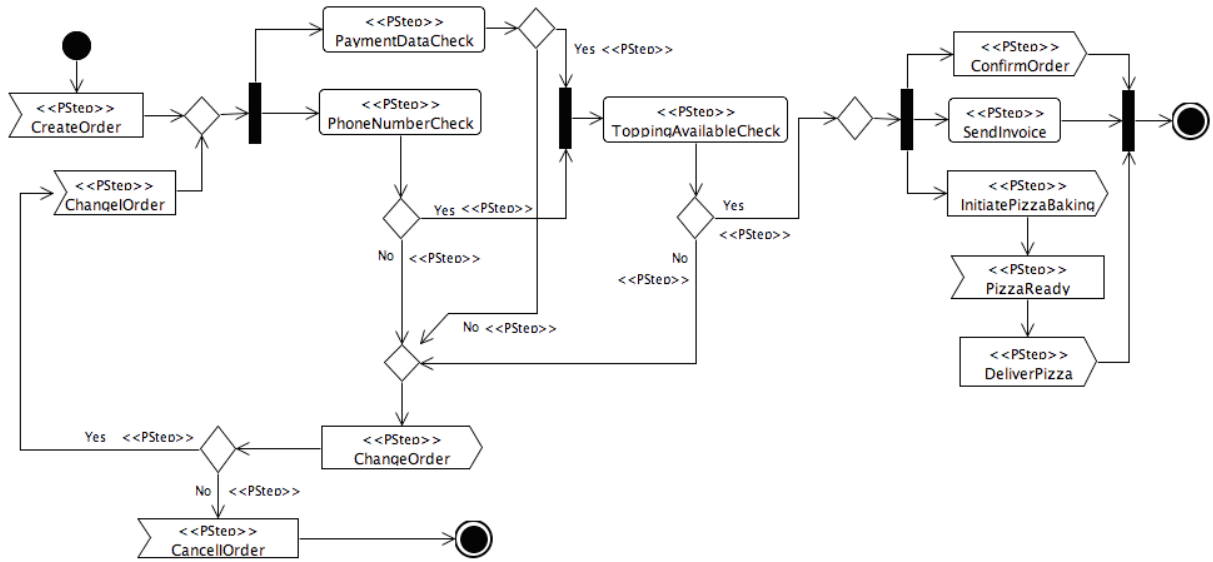
The another branch using diagrams is the Human-Computer Interaction (HCI) with ConcurTaskTrees (CTT) notation. It is a diagrammatic notation used to design the user interfaces during software development. There exists a piece of software to create CTT diagrams called ConcurTaskTrees Environment (CTTE). The possible application would be a plug-in for this software recognizing handwritten diagrams and transferring them into inner representation of the CTTE. It is important to create such diagrams as a documentation for developers of user interfaces. They have no other option such as the inverse engineering. The crucial advantage is that CTT can be expressed using UML standard [NNC05]. An example of CTT is shown in Figure 3.6.

Although each kind of diagram is used in a different application branch and requires a different context, their syntax is very similar. Therefore, universal techniques for diagram recognition can be developed with just minor modifications to cope with some concrete diagram. Many interesting approaches and observations have been discovered in the field of the on-line diagram recognition. Some of them are common for all types of diagrams and some of them are strictly specialized to the concrete diagrammatic notation.

3 Types of Useful Diagrams



a) Class diagram



b) Activity diagram

Figure 3.4 Examples of the most frequently used UML diagrams.

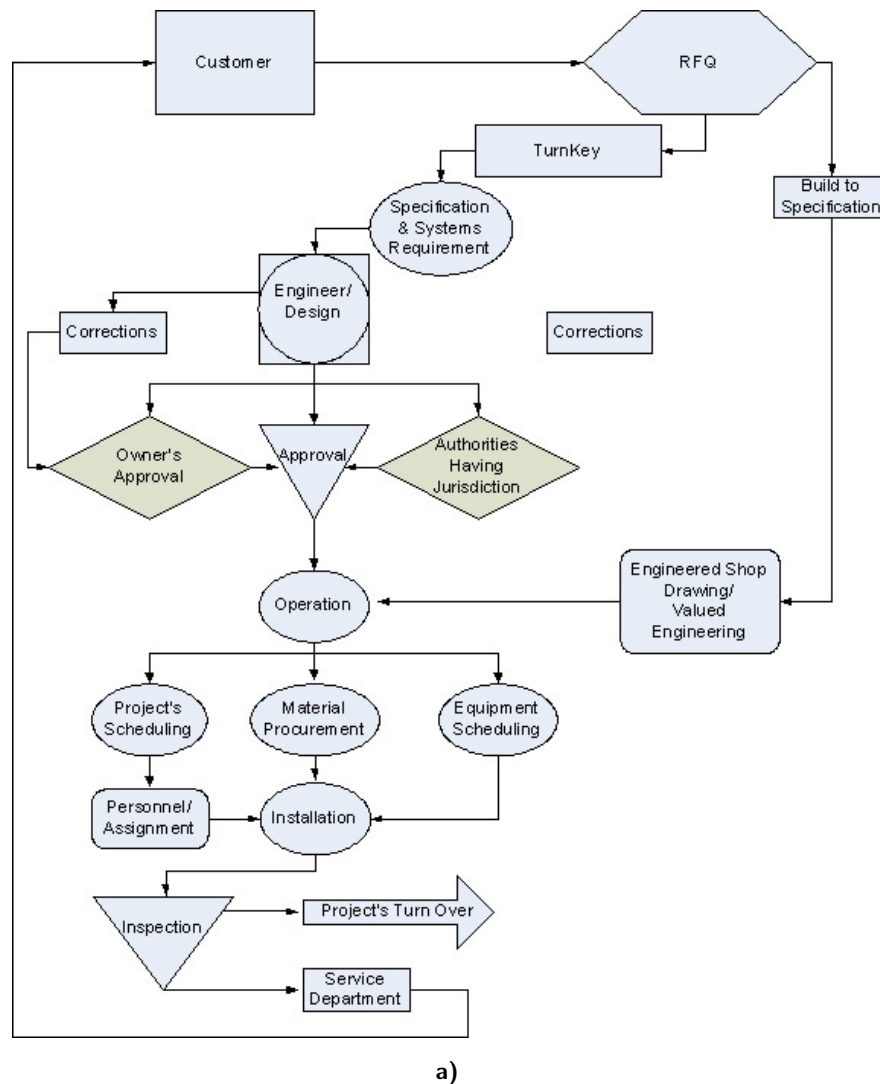


Figure 3.5 Example of business process diagram.

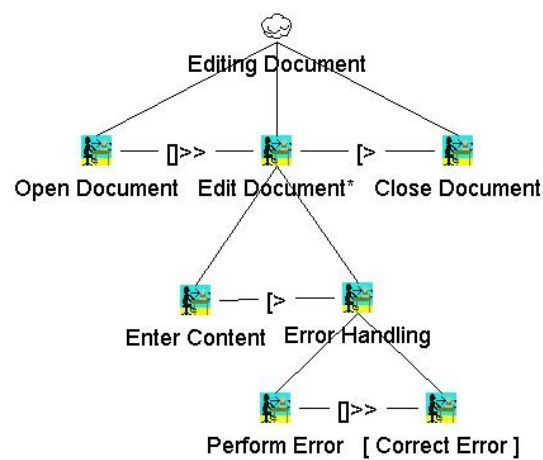


Figure 3.6 Example of CTT.

4 State of the Art

In this chapter, I will introduce related work and achieved results in the field of hand-drawn diagrams and sketches. There is a lot of domains that the researchers decided to explore and plenty of methods was invented. Some of them are very general and easily applicable while some are more specific. According to the survey of diagram recognition techniques presented by Dorothea Blostein [Blo96], each diagram recognizer must perform 6 processes:

1. Early processing - noise reduction, de-skewing, etc.
2. Segmentation, to isolate symbols.
3. Symbol recognition.
4. Identification of spatial relationships among symbol.
5. Identification of logical relationships among symbols.
6. Construction of meaning.

Although these tasks must be done, it does not mean they have to be done separately, sequentially or exactly in this order.

Every recognizer has to deal with noise and uncertainty. There are basically two approaches how to do that:

1. **Contextual feedback**

The basic idea is that the recognizer recognizes the input by small pieces and waits for immediate feedback. The user confirms if the recognition is good, while provides the ground truth. These methods put unnatural requirements on the user and we are not speaking about free drawing recognition anymore. Some methods are even based on recognition of series of gestures rather than recognition of a hand-drawing. Miyao and Maruyama [MM12] created a system for Flowchart recognition which works on these principles. The user is drawing symbols and the system immediately recognize them while a new loop is detected. It is possible then to connect the symbols by gestures. To input text, the user has to hold the tip of the stylus inside a symbol for one second to invoke a pop-up window with a text recognizer. The system works well for flowcharts since all symbols are loopy.

2. **Sequential processing with lists of alternatives**

The symbol recognizer gives us a list of possibilities what the symbol can represent. When the segmentation is not clear the list of possibilities may contain results of all possible (admissible) segmentations. The decision which possibilities, so called symbol candidates, form a valid solution is up to a structural analysis. Various approaches to the structural analysis will be described later.

The first group of recognizers is kind of special and I have no interest in them in this paper. The second group of recognizers is very common and has to follow the pipeline defined by Blostein. The rest of this chapter will also follow the pipeline.

4.1 Preprocessing

It is necessary to perform some early processing before actual recognition algorithms can be run. The most commonly used preprocessing technique is resampling of strokes to obtain, for example, evenly distributed points on a 2D curve. The resampling is important to make recognition algorithms work similarly with different input devices since they might not have the same sampling rate. Moreover, it can be useful when we want to normalize different structures. Another useful technique is de-skewing or a different general transformation used to register drawings of different users. Decomposition of complicated strokes into primitives such as line segments and arcs is also commonly used early processing. On the other hand, sometimes it is necessary to group or merge overlapping strokes, which are created when the user tries to correct a previous stroke by another one. In this section, I describe the most important and the most commonly used non-trivial preprocessing methods, which deal basically with primitives decomposition and multiple strokes merging.

4.1.1 Primitives Decomposition

This process is sometimes considered to be a segmentation, not a preprocessing. However, I decided to place it here, because the resulting segments are very simple primitives, not the whole graphical symbols and symbol semantics is not explored. Meseery et al [EMEDM⁺09] proposed a method for segmenting line segments and arcs in a single stroke. *Possible dominant points* (points with low speed and high curvature) are detected first. Next the discrete particle swarm algorithm (DPSO) is used to find the best subset (in terms of the error function) of dominant points, which are used to cut the stroke. The error function may be defined in different ways depending on which primitives we want to segment. For a line segmentation, it is just a distance from each point of a segment to the line defined by two endpoints of the segment. Example of a stroke, its dominant points and an error is shown in Figure 4.1.

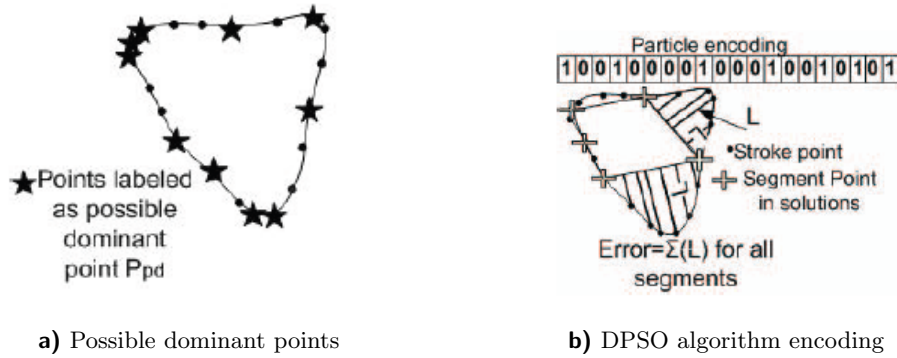


Figure 4.1 Example of a stroke and the segmentation process using particle swarm algorithm.

Feng and Viard-Gaudin [FVG08] proposed a method for stroke fragmentation into lines and arc segment based on a HMM. They extract four features for each point of the stroke: local direction ($\sin \alpha, \cos \alpha$), local curvature and direction change. The model is defined with 24 hidden states: 8 for different angles of line segments, 8 for arcs clockwise, and 8 anti-clockwise as suggested in Figure 4.2. This method is commonly used nowadays.

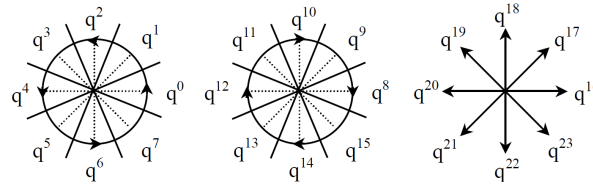


Figure 4.2 Suggested hidden states by Fend and Viard-Gaudin.

4.1.2 Merging of Overlaying Strokes

Users have sometimes tendencies to fix strokes, which do not satisfy them, by drawing another stroke over the first one. It is a common approach when we are using a pencil and paper. However, for a recognition system using data from a tablet, these multistrokes cause serious problems. Therefore, it is necessary to group these strokes together and merge them to create one (fixed) stroke. It is the only way how to make a recognition system robust enough to let the user to express him in his natural way.

Ku et al. [KQW06] deal with interpretation of freehand sketches with over-traced strokes. They proposed a method for grouping overlaid strokes. The strokes are classified into lines and curves first. Only strokes of the same class can be merged together. It is done by line or curve fitting. Mitani et al. [MSK00] proposed a method for interpreting 3D design sketches. They call the overlaid strokes a stroke bundle and each stroke bundle is represented by a core stroke which is created by points tracking.

4.2 Segmentation

The segmentation process is a difficult task. Quite often the very first step preceding the isolate symbol segmentation is to distinguish between graphical symbols and the text. Once we know which strokes represent text, it is much easier to recognize isolated symbols. Of course, it is also much easier to recognize graphical symbols (arrows, boxes, glyphs) when its strokes are not mixed with text strokes.

There are several approaches to do this distinguishing. The important question is: how accurate it can be? All algorithms can be divided into three groups: bottom-up, top-down and mixed approach (bottom-up first and then top-down to solve a possible ambiguity). The next step is to find subsets of strokes representing separated symbols. Let us assume that strokes were divided into primitives in the preprocessing step and there is no stroke shared between two symbols.

4.2.1 Graphical / Textual Strokes Distinguishing

Bishop and Hinton [BSH04] introduce a system based on three approaches with successively increased complexness. Firstly, an independent stroke model is employed. There are 9 features extracted from each stroke, such as length, curvature, number of fragments, etc. A multilayer perceptron is trained on a annotated database. Secondly, they exploit temporal information. The user draws few consecutive graphical strokes very often and then writes down a text consisting of a certain number of letters. The authors find a transition probability using training data and use the first order Markov process over the labels which is looking back one stroke in time. Then hidden Markov model (HMM) is constructed to represent a whole sequence of strokes. The most probable sequence is found using Viterbi algorithm (dynamical programming) which is linear in the number of strokes. Thirdly, information of gaps between strokes is exploited.

There are 5 features extracted from each gap, such as time difference between pen-up and pen-down. It shows that only two class labels are necessary for gaps: $(t_n = t_{n+1})$ and $(t_n \neq t_{n+1})$. Bi-partite HMM is used then to combine these approaches.

Zhou and Liu [ZL07] criticize Bishop and Hinton that they use only temporal information of strokes. There are strong spatial relations in general. The user may draw different parts of the same symbol or segment in different time order. Zhou and Liu decided to use Markov random field (MRF) counting with clique energy. They use unary and binary (two neighbours) cliques. They extract 11 features from each stroke. The algorithm is tested on the Kuat Kondate database of drawings with Japanese characters. The MRF approach with spatial information outperforms the HMM approach using temporal information on this database.

Bhat and Hammond [BH09] proposed an approach using entropy to distinguish shape versus text. The method is based on the curvature of strokes. Each point of the strokes falls into one bin of the 7-fold histogram depending on the angle the point makes with its neighbours. The seventh bin is for end-points which make no angle. The entropy of that histogram is computed. A threshold must be found to distinguish text from graphics. Text strokes tend to have a higher entropy. This method is very simple and easy to implement. Although it gives average performance results, it may be desired method thanks to its simplicity.

Willems et al [WRV05] use a decision tree to recognize a mode of input (text, lines, graphics, etc.). In each branch of the tree, a classifier uses specific features of the strokes. To choose the best combination of all 12 features which can be extracted, they use a simple neural network on training data. Once the mode of input is recognized, recognition of individual symbols can be performed. The performance is increased when the mode is known and the recognizer is optimized for the mode.

Two very similar recent methods presented by Otte et al. [OKLD12] and Indermühle et al. [IFB12] are based on a long short-term memory (LSTM) neural networks (RNNs). They both extract just local features for each point of the strokes and train the network. The both methods report an accuracy over 98% on the benchmark database IAMonDo. I should remark that the method by Indermühle is classifying each single point and therefore it can recognize when text and a symbol were drawn by one stroke.

In this chapter, I introduced the most common algorithms for text / graphics classification. Some of them are more accurate than others, but they are also more complicated. However, the 100% accuracy is never achieved and every error done in this step propagates further in the pipeline. Therefore, splitting strokes into text and graphics and processing them separately is not desired. Text / graphics classification should just provide useful information for further structural analysis. Although, the hard splitting significantly reduces the time complexity.

4.2.2 Separate symbols segmentation

It is of advantage to segment symbols first to classify them and create symbol candidates. One natural way is to create all possible subsets of strokes and then classify each of them. In that case we are talking about segmentation by classification. The number of all possible subsets might be very large and thus some heuristics are applied to determine which subsets are admissible and worth classifying. The heuristics often contains just simple rules: strokes in the subset must be spatially and temporarily close and their number is limited. Such approach use for example Feng et al. [FVGS09] for segmentation a classification of symbol in electric circuit diagrams. This approach is equivalent to the sliding window in the image processing.

Stahovich et al. [PSDA10] propose an innovative method of grouping based on tentatively classifying single strokes into clusters. This classification uses set of features taken from Patal et al. paper [PPGI07] plus some of their own. The clustering makes the next classification step easier when strokes in each cluster are classified to the final symbols. Authors claim the proposed approach is applicable on wide range of problems. They verified it on the classification text / non-text, which is easily comparable with other methods. They claim they can compete with state of the art algorithms for text / non-text distinguishing algorithms. The second step can be the same approach like tentative grouping or a usage of something different. Their basic idea joins strokes iteratively when they are close by shape or spatially. They distinguish three cases: “don’t join”, “far join”, “near join”.

It is very important to recognize arrows in recognition of many diagrams, because they carry a lot of information about the diagram structure. The flowchart diagram is a very good example of such a diagram. Kara and Stahovich [KS07] presented a system for recognition of network diagrams (used with MATLAB Simulink package) where the arrows play a key role. They are called “marker symbols” and are considered as “islands of certainty” due to the possibility to detect them easily and very precisely. Therefore, they are used immediately as a ground truth and exploited for segmenting each arrow connecting two symbols. Kara and Stahovich assume that an arrow can be drawn as one or two strokes (1 stroke for a shaft and 1 stroke for a head). Thus, they implemented two separate recognizers, although of the same principle. The recognition is based on extraction of 5 key points identified by locating the last global minima in the speed profile. Using these points they extract angles and length ratios which must be within certain limits to classify the stroke(s) as an arrow – see Figure 4.3. This approach proved to be a good solution. Stoffel et al. [STR09] used the very similar approach in their system for commutative diagrams recognition. The only difference is that they learned the classifier on a training set to make the angles and length ratios limits tighter.

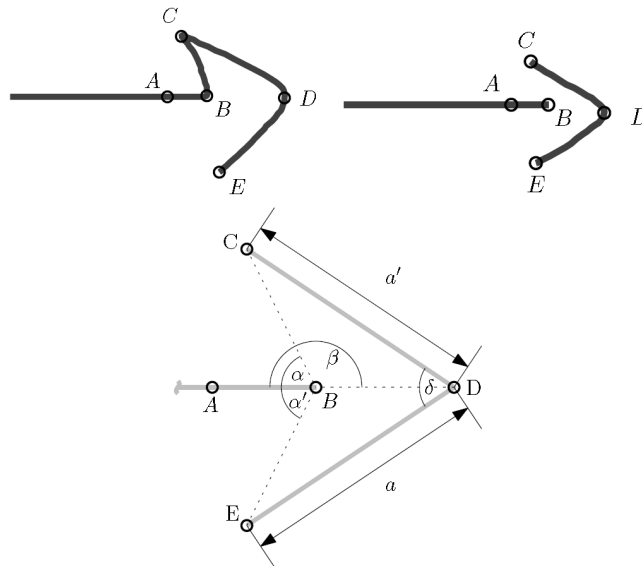


Figure 4.3 Above: The five keypoints that define arrow shape. Below: The numerical features used for arrow recognition.

4.3 Symbol Recognition

Once the symbols are segmented in the diagram, it is necessary to recognize them. There are many approaches. Not only letters of the alphabet are symbols in diagrams, but also all possible pictograms, arrows, etc. Therefore, it is better to use more recognizers rather than only one and use multipass recognition [SVC04]. Symbol recognizers can be divided into the following classes:

1. Rule-based

The simplest recognizers using just predefined rules how each symbol should look like. They are good for recognizing lines, arrows and simple geometric notations. The biggest advantage is that they need no training.

2. Nearest Neighbour

Each symbol class has defined a representative(s) which describes the class the best. To classify a query the nearest neighbour is found according a defined metric. The classification result is given by the class of the found nearest neighbour.

3. Statistical

Recognizers trained from a training set. They extract features of the symbol and try to find the best match using a statistical model. They are often used for recognizing alphabetical characters.

4. Structural

The most complicated recognizers which use a structure to recognize more complex symbols. They are often based on grammars where the symbols are composed of primitives.

4.3.1 Feature Extraction

It is necessary to extract some features describing the strokes to classify isolated strokes or even whole clusters of strokes representing a graphical symbol. When a rule-based classifier is used, features might be complex and customized for a given set of symbols according to the task. When a statistical classifier is exploited, features should be more simple, general and should be combined and statistically measured.

Patel et al. [PPGI07] gathered in their work known features and added their own, new general features which can be extracted from strokes. The paper contains a list of all 46 strokes they came up with. The main contribution is a statistical survey which says which features are the most important and should be used. The features can be divided into seven groups:

- Pressure Features
- Time Features
- Intertsection Features
- Size Features
- Curvature Features
- Tablet OS Recognition Features
- Inter-Stroke Gaps

The process of choosing the appropriate subset of features for the concrete problem may be done statistically [LB07] – choose a subset of feature which best describe the problem. The result is that the recognition is faster and more accurate.

4.3.2 Rule-based Classifiers

Rule-based classifiers suit domains where the symbols are simple geometric shapes. Flowcharts, for example, consist mainly of circles, rectangles, diamonds, rhomboids, ellipses, etc. Primitives like circles and lines can be found using RANSAC or Hough transform. The classifier contains rules defining each symbol class. For example, a rectangle consists of four line segments where an angle between neighbouring segments is 90° . All necessary thresholds as well as the rules might be set manually by the system designer or automatically by machine learning approaches (evolution algorithms) [HSEL07]. The classifier has character of a decision tree.

4.3.3 Nearest Neighbour

These classifiers are popular thanks to their simplicity. They are suitable when the number of representatives is reasonably small and the computational cost of computing the distance between two samples is low. It is possible to use various libraries for nearest or even approximate nearest neighbour search which work efficiently even in high dimensions. The common way is to define a descriptor to express a symbol and then use Euclidean distance between descriptors. In the simplest case a symbol can be projected into a constant-size pixel grid and values of the pixels may be used to describe the symbol. Various method to decrease the dimensionality may be used – PCA for example.

Ouyang and Davis [OD11] combine geometric features with visual features to describe symbols in their work. They keep a hierarchy of representatives for each class to cover the interclass diversity. They maintain a constant number of representatives to keep the classification fast. The hierarchy structure helps as well.

Quereschi et al. [QRCM07] created a fixed size descriptor based on combination of symbolic and statistical features. They used the descriptor to create a classifier based on the nearest neighbour rule and applied the method to recognition of architectural and electrical symbols.

Elastic matching and time warping are very similar methods to measure a distance between two symbols. It is suitable for simple symbols like characters. Elastic matching method was introduced by Tappert [Tap82]. The distance between two symbols is expressed as measure of deformation needed to transform one symbol into another. The problem is solved by dynamic programming.

4.3.4 Statistical Classifiers

All neural networks (NNs), Support Vector Machines (SVMs) and Hidden Markov Models (HMMs) belong to this class. Classifiers based on SVM are good when we are able to extract feature vectors of the constant size for each subset of strokes. On the other hand, classifiers based on HMMs are good to describe arbitrary long sequences of strokes or points. When we do not have sufficient knowledge about the domain, we can use a neural network to train the classifier.

Niels et al. [NWV08] created their own database of graphical symbols (icons) for crisis management. They proposed three feature sets and using these sets they trained and tested various classifiers. One feature set consists of global features only while the other two feature sets of different size contain only local features. SVM classifier and Multi-Layer Perceptron (MLP) classifier were trained. Both classifiers achieved the highest and very similar accuracy with the bigger feature set of local features. Additionally,

the authors created a classifier combining the previously mentioned classifiers by simple majority voting. It reached the highest accuracy of 96.5%.

Zhu and Nakagawa [ZN12] presented a MRF recognizer for Japanese characters. The recognizer uses feature points extracted for each character and MRF is used to match the feature points with the states of each character class. All characters create an extensive dictionary. The main contribution of the paper is a method for an efficient dictionary compression, which allows to decrease its size while keeping the same recognitions speed. This makes the recognizer more suitable for portable devices with ink input (like smart phones, tablets) where the memory is limited. It is possible to decrease the size of the dictionary more than ten times.

Shi and Zhang [SZ10] combined an SVM with an HMM to create a classifier for recognition of chemical symbols. It is a double-stage classifier. In the first stage, the SVM classifier is used to roughly classify the chemical symbols into Non-Ring Structure (NRS) and Organic Ring Structure (ORS). It is based on 58-dimensional feature vector combining local and global features. In the second stage, the HMM classifier is used to classify the chemical symbols into concrete classes. Coefficients of Discrete Cosine Transformation (DCT) are used as a feature sequence.

4.3.5 Structural Classifiers

These classifiers are used to recognize more complicated symbols consisting of primitives like line segments and arcs. The structure of symbols is described by a grammar and the recognition itself is done by a parser. The parser can be based on bottom-up or top-down approach. It means that the recognizer try to assemble a whole symbol from primitives or it takes a symbol and tries to disassemble it into primitives. The classifier has defined a grammar for each symbol class. Only one grammar can accept an input symbol (a word in grammar terminology) and that leads to the classification. These classifiers can work well only if the primitives are easy to detect in the input strokes (for example with the algorithm by Feng and Viard-Gaudin [FVG08]). It might happen that symbols from one class may vary and consist of different primitives. In that case the grammar has to accept all variants, or more than one grammar must be defined. I should mention that in the most of the cases the grammars are defined manually by the programmer.

More complex survey of grammars for diagram recognition will be presented in Section 4.4.1. Here, I will introduce just briefly some grammar based approaches used for symbol recognition. Rekers and Shürr [RS95] presented grammar based approach for defining the syntax of visual languages and for generating visual language parsers. They demonstrated it on recognition of symbols and structure of process flow diagrams (flowcharts). Mas et al. [MSL05] defined an Adjacency grammar based on the interaction between neighbouring strokes and demonstrated it on the recognition of symbols from an architectural domain. Later [MRLSL06], they introduced a method for automatic generation of the Adjacency grammar from a given training database. Costagliola et al. [CDR05] proposed Sketch grammar for recognition of symbols and structure of flowcharts.

4.4 Structural Analysis

It is the job of structural analysis to find spatial and logical relations among the symbol candidates and to create the final solution. There exist various approaches and all of them have to do the same thing – assess the quality of the found symbol candidates, how good they fit the overall structure, and find a solution maximizing a predefined function. All methods have to deal with uncertainty. It is possible to divide them into the following three groups:

1. Grammars

The structure of a diagram is described by a grammar. This grammar defines logical relations among the symbols. Attributed grammars are used to express spatial relations between symbols as well. Attributes are also used to deal with distortions and uncertainty.

2. Graphical Models

The recognition is formulated as a optimization task, where a minimum risk, maximum probability, or just maximum value of a score function is sought in general. All Bayesian networks, Markov Random Fields (MRF), and Conditional Random Fields (CRF) belong to that class.

3. Dynamic Programming

Dynamic programming (DP) is used to efficiently explore all hypothesis based on various symbol candidates.

4.4.1 Grammars

Grammars represent a good way how to describe a structured data using a formal language. Originally, they were used to generate textual languages, but later, grammars for graphical languages were invented. Various types of grammars can be used for diagram recognition. They proved to be suitable even for recognition of individual graphical symbols if they consist of elementary components (primitives). In this section, I will describe the most common grammars suitable for the diagram recognition. A grammar determines what is possible, how a graphical symbol can be created and how a diagram can look like. However, there must be a parser which does the recognition according to a grammar and an input data. The big issue is how to cope with noise. There exist a set of parsers suitable for each type of grammar.

Grammars have been commonly and successfully used in mathematical formulas recognition due to their capability of expressing a complex recursive structure of mathematical formulas. Průša and Hlaváč [PH08] presented a method for on-line mathematical formulas recognition based on a 2D grammar employing structural construction. They used segmentation by classification to detect elementary symbols. They used an OCR tool for classification of subsets of strokes possibly forming a symbol. The grammar is used to perform structural analysis.

String Grammars

There is a family of string grammars usable in syntactic pattern recognition [BS90]. String grammars are considered to be the most fundamental concept. They operate on strings of symbols, i.e. words over a finite alphabet. In the case of diagram recognition, the symbols represent primitives such a line segment or an arc. The example of primitives and patterns grammars can represent is shown in Figure 4.4.

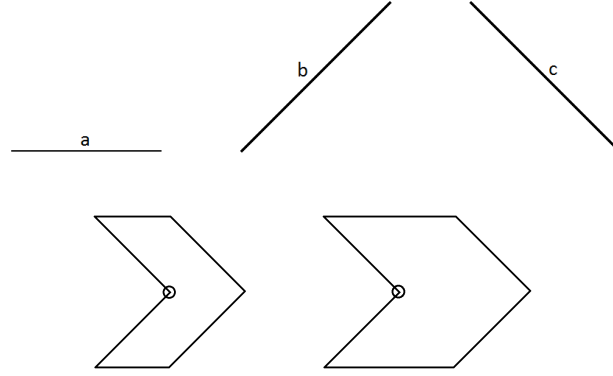


Figure 4.4 Line segments as primitives for the contour representation are in the first row. Sample patterns of a class in the second row. The patterns can be represented as *cacbab* and *caacbaab*, respectively. The circle denotes the initial point. Primitives are coded in a clock-wise direction.

Definition 1 A formal grammar is a four-tuple

$G = (N, T, P, S)$, where

N is a finite set of nonterminal symbols,

T is a finite set of terminal symbols,

P is a finite set of productions and

$S \in N$ is the starting or initial symbol.

It is required that $N \cap T = \emptyset$; the union of N and T is called the *vocabulary* $V = N \cup T$. Each production $p \in P$ is the form $\alpha \rightarrow \beta$ where α and β are called the *left-hand* and *right-hand side*, respectively, with $\alpha \in V^+$ and $\beta \in V^*$. V^* is set of all words over the vocabulary including empty symbol ϵ and $V^+ = V^* \setminus \{\epsilon\}$.

The grammar G generates the language $L(G)$. A fundamental task in syntactic pattern recognition is to decide if $x \in L(G)$, where a grammar G and a word x over the terminal alphabet are given. In other words, if there exist a sequence of productions which transforms the starting symbol S into the given word x . In practice, we can use a different grammar for each symbol to be able to recognize multiple graphical symbols. The top-down parsing is used, the initial symbol S represents the graphical symbol, and terminals represent the primitives. The recognition schema is visualized in Figure 4.5.

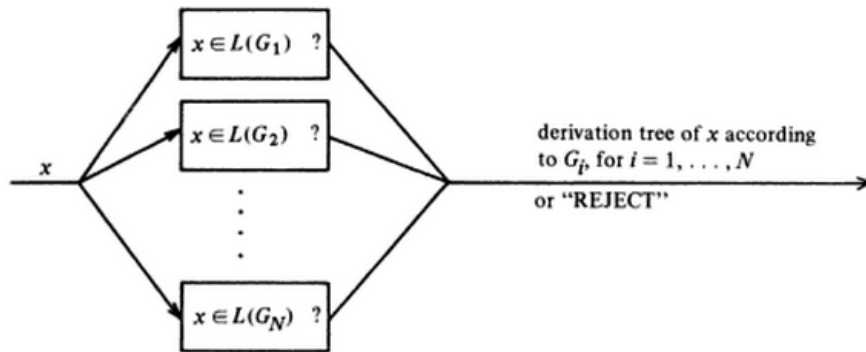


Figure 4.5 Block diagram of syntactic recognition.

The advantage of this string grammar is its simplicity and a solid foundation in textual formal languages. The main disadvantage is the impossibility to parametrize the primitives. Therefore, it is necessary to represent each primitive (even each line segments differently rotated) by a terminal symbol. However, it can be used for recognition of a rather small set of primitive symbols. Although it will not solve the problem, it is possible to use *attributed grammar*, where each symbol $Y \in V$ is augmented by a vector of attribute values $m(Y) = (x_1, \dots, x_k)$. During parsing the attributes are propagated to all symbols. It allows us to put some semantic information to recognized graphical symbols, such as its size or total length of strokes. Another disadvantage is the fact that this schema is very restricted, since the only possible relation between symbols (primitives) in a string is the concatenation.

To face this problem and obtain a more efficient representation, we can incorporate representation relations into the string, which are more general than concatenation. Two well-known approaches following this idea are the picture description language PDL [Sha70] and plex grammars [Fed71], which will be described further.

Picture Description Language (PDL) It is assumed that each primitive has exactly two points, called *tail* and *head*, where it can be linked to other primitives. In PDL, there are four ways of joining a pair of primitives. These four binary operators are denoted by $+$, $-$, \times , and $*$. Additionally there are two unary operators (\sim reverses tail and head). The four binary operations and their geometric interpretations are shown in Figure 4.6. Each entity resulting from the linking of two primitives has a tail and a head too. So we can extend the four binary operations to the higher level structures.

Operator	Meaning	Geometric interpretation
$a + b$	head (a) linked to tail (b) head ($a + b$) = head (b) tail ($a + b$) = tail (a)	
$a - b$	head (a) linked to head (b) head ($a - b$) = head (b) tail ($a - b$) = tail (a)	
$a \times b$	tail (a) linked to tail (b) head ($a \times b$) = head (b) tail ($a \times b$) = tail (b)	
$a * c$	tail (a) linked to tail (c) and head (a) linked to head (c) head ($a * c$) = head (a) tail ($a * c$) = tail (a)	

Figure 4.6 The four binary PDL-operators.

The language $L(G)$ generated by a grammar G consist of a number of PDL expressions each describing an individual pattern. A descriptive example of PDL expressions is in Figure 4.7. The whole pattern class (one graphical symbol) is represented by one grammar. Consequently, the recognition schema shown in Figure 4.5 can be applied.

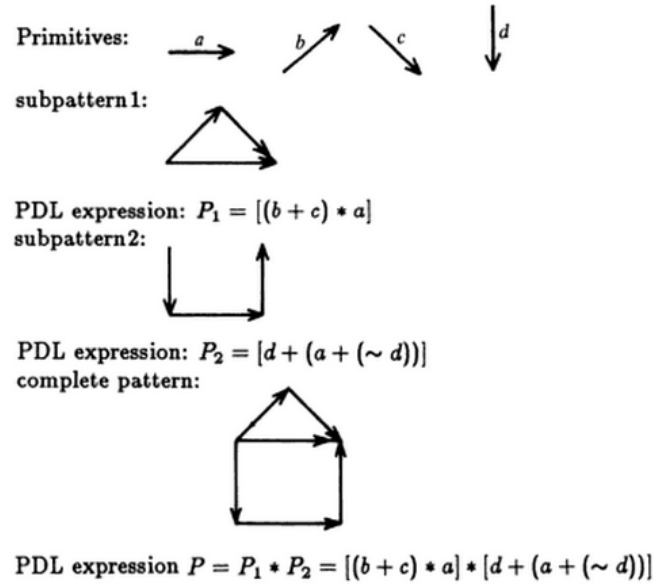


Figure 4.7 An example of a PDL expressions and its corresponding pattern.

The PDL introduces important relations between primitives. However, we require more linking points very often.

Plex Grammars Plex grammars bring an option to have symbols with more than two points linked with another symbol. A symbol with n attaching points is called an n -attaching point entity or a *nape*. Structures formed by joining napes are called *plex structures*. Formally, a nape is represented by an identifier and a list of attaching points. A plex structure is described by three components, namely a list of names, a list of internal connections between napes, and a list of attaching points where the plex structure can be joined with other napes or plex structures. Each nape corresponds to a terminal symbol and each plex structure corresponds to a nonterminal symbol. An example of napes, plex structures and grammar generating the plex structures can be seen in Figure 4.8.

Graph Grammars

The graph grammars have been invented in early seventies in order to generalize the string grammars. The main idea was that of extending concatenation of strings to a “gluing” of graphs. The graph grammars are also called graph transformation or graph rewriting. The reason is that graph grammars use graph transformation (rewriting) when production rules are applied. Graph grammars are very general. There exist many usable graph transformations.

Horst Bunke [Bun82] presented attributed programmed graph grammars and applied them to circuit diagrams and flowcharts recognition. His grammar is formally defined as a 7-tuple $G = (V, W, A, B, P, S, C)$ where:

- V and W are alphabets for labeling the nodes and edges, respectively.
- A and B are finite set of attributes for nodes and edges, respectively.
- P is a finite set of productions.
- S is a set of initial graphs.

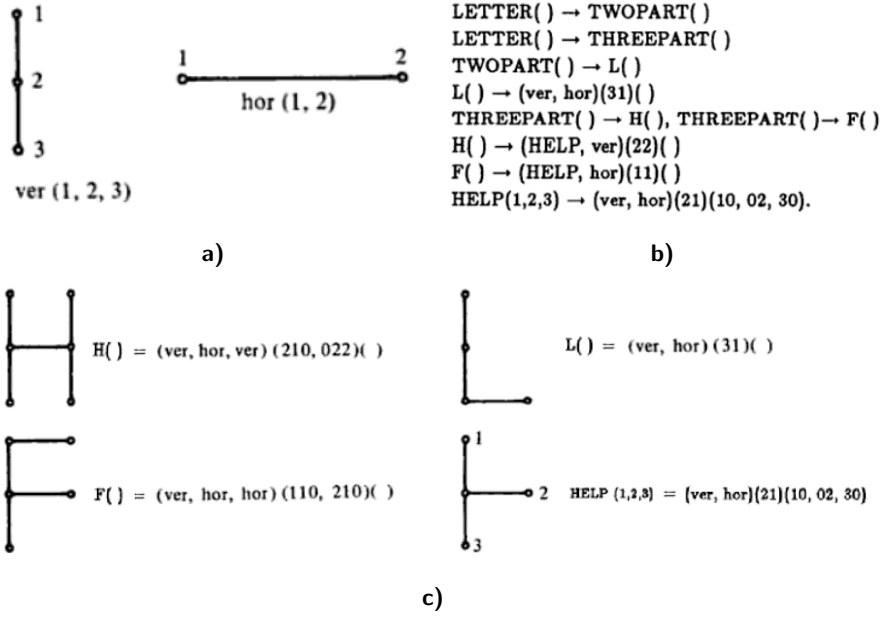


Figure 4.8 a) An example of two napes. b) Example of a plex grammar productions. c) Example of plex structures generated by the productions.

– C is a control diagram over P .

The programmed graph grammars use a control diagram as an explicit tool for controlling the order of productions including the stop of a derivation sequence. Therefore, there is no distinction between terminal and nonterminal labels. The language of G consists of all graphs which can be derived in the following way:

1. Start with an initial graph.
2. Apply productions in an order defined by the control diagram.
3. Stop the derivation when the final node in the control diagram has been reached.

A production is defined as 5-tuple $p = (g_l, g_r, T, \pi, F)$ where:

- g_l and g_r are unattributed graphs, the left-hand side and right-hand side, respectively.
- $T = \{L_w, R_w | w \in W\}$ is the embedding transformation with $L_w, R_w \subseteq N_l \times N_r$. N is a set of the graph nodes.
- $\pi : \Gamma \rightarrow \{\text{TRUE}, \text{FALSE}\}$ is the applicability predicate.
- F is a finite set of partial functions $f_a : N_r \rightarrow D_a$ and $f_b : E_r \cup \text{EMB}(g_r, g) \rightarrow D_b$ with $a \in A, b \in B$, and $g \in \Gamma$. E is a set of the graph edges. The $\text{EMB}(g_r, g)$ is the embedding of g_r in g (edges between g_r and $g \setminus g_r$). The f_a are the node attribute and the f_b are the edge attribute transfer functions. Γ is a set of all attributed graphs.

The author proposed example productions, which can reduce certain distortion in the input data. These productions can be recursively applied to fill a missing line, delete an incorrect line, or shrink a gap between lines.

Rekers and Schürr [RS95] defined a graph grammar for a finite state automata (FSA) and introduced a parsing algorithm based on a parsing of graphs used to define the productions. Later, Rekers and Schürr [RS97] introduced context-sensitive layered graph grammar and introduced efficient parser. The principle is that non-terminals belong to layers which are ordered. There exists a rule that the left-hand-side of production rules is “smaller” than the right-hand-side. This guarantees no cycles.

The graph grammars exist in the field of pattern recognition for a long time and proved to be a good formalism. Therefore, there exist a lot of tools. Probably most important is PROGRES (PROgrammed Graph REwriting Systems) [PRO08], which is a graph grammar programming environment using high level programming language. It is frequently used by researchers and another tools are based on this system. Another important tool is the attributed graph grammar system AGG implemented in Java [AGG13]. It aims at the specification and prototypical implementation of applications with complex graph-structured data. There also exists standard graph exchange language GXL [GXL02].

Adjacency Grammars

Adjacency grammars are inherently bidimensional and have been used in many disciplines to define symbols. They were presented by Jorge and Glinert [JG95] as an extension of picture layout grammars. They used them for recognition of directed graphs. Later, Mas et al [MSL05] used them for recognition of graphical symbols in an architectural domain. Adjacency grammars employ adjacency of primitives forming symbols. There exist three types of adjacency, but not all of them have to be used:

- **Algebraic adjacency**

These relations generalize the intuitive idea that, with respect to some relation R , two elements are adjacent “if there is nothing in the middle”. To generalize the concept we say that two items x and y are adjacent, with respect to some relation R , iff xRy and $\neg\exists z$ such that $xRz \wedge zRy$.

- **Spatial adjacency**

Two items are spatially adjacent if they are simply close enough to each other. It is necessary to establish a “cut off” distance d_α , beyond which items can no longer be associated.

- **Logical adjacency**

These relations are used to express that two items have the same label, which means they go together no matter their spatial relation. An example of such relation is a list of the same elements.

An adjacency grammar is formally defined as a 5-tuple $G = (V_t, V_n, S, P, C)$ where:

- V_t denotes the alphabet of terminal symbols.
- V_n denotes the alphabet of non-terminal symbols.
- $S \in V_n$ is the start symbol of the grammar.
- C is the set of constraints applied to the elements of the grammar.
- P are the productions of the grammar defined as:

$$\alpha \rightarrow \{\beta_1, \dots, \beta_n\} \text{ if } \Gamma(\beta_1, \dots, \beta_n)$$

where $\alpha \in V_n$ and all the $\beta_j \in \{V_t \cup V_n\}$ constitute a possibly empty (ϵ) multiset of terminal and non-terminal symbols. Γ is an adjacency constraint defined over the attributes of the β_j . Examples of constraints usable in the grammar taken from [MSL05] are shown in Figure 4.9.

Mas et al. [MRLSL06] showed that it is possible to generate the grammar automatically. If we want to use adjacency grammars for graphical symbols recognition then we use one grammar for each symbol. Automatic generation of the grammar just needs examples of symbols and set of adjacency relations. Grammar productions are automatically created using the examples and adjacency relations.

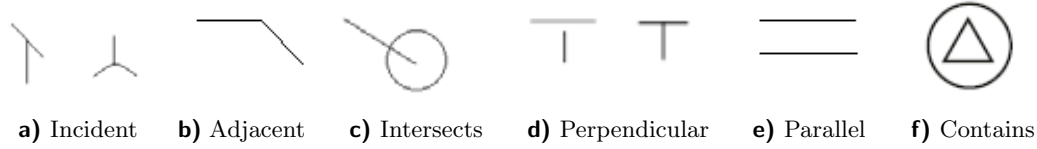


Figure 4.9 Examples of constraints.

Both, Jorge and Glinert [JG95] and Mas et al. [MSLL07] proposed also parsers for adjacency grammars. Both parsers are simple incremental predictive bottom-up parsers. They are very similar in principle. After a new stroke is added the parser keep a list of possibilities what productions can be used. Recently, Mas and Jorge cooperated [MLSJ10] and designed a new spatial-directed parser for adjacency grammars. It uses an indexing structure to reduce the complexity. The parser is again incremental and bottom-up. When a new primitive is added all previous symbols suitable for a reduction must be found. The indexing structure is used to reduce the search space. A spatial grid is created and symbols are divided into cells. The search is done just in neighbouring cells of the currently added symbol. The author tested the grammar and the parser on recognition of sketched architectural floor-plans. They achieved the recall of 93 %.

4.4.2 Graphical Models

Graphical models on general graphs are very powerful technique to formulate the recognition of structures. The most commonly used are MRFs and CRFs. Each node of the graph represents an element of the recognized structure (a single stroke or a symbol candidate) and it is provided with a vector describing the observed instance. The goal is to label each node with one of the labels from the finite set. The label depends on the descriptor of the current node and all its neighbours. Therefore, it models the whole structure with a context between elements. Often maximum likelihood (ML) or maximum a posteriori (MAP) criteria are used during training. Formally, we have the following definition of CRFs (Lafferty et al. [LMP01]):

Definition 2 Let $G = (\nu, \varepsilon)$ be a graph such that \mathbf{t} is indexed by the vertices of G . Then (\mathbf{x}, \mathbf{t}) is a conditional random field (CRF) if, when conditioned on (\mathbf{x}) , the random variables t_i obey the Markov property with respect to the graph: $p(t_i | \mathbf{x}, \mathbf{t}_{\nu-i}) = p(t_i | \mathbf{x}, \mathbf{t}_{N_i})$, where $\nu - i$ is the set of all nodes in G except the node i , N_i is the set of neighbours of the node i in G , and \mathbf{t}_Ω represents the random variables of the vertices in the set Ω .

Unlike traditional generative random fields like Bayesian nets (BN), CRFs only model the conditional distribution $p(\mathbf{x}, \mathbf{t})$ and do not explicitly model the marginal $p(\mathbf{x})$. Qi et al. [QSM05] used CRF for recognition of organization charts. Strokes of the charts were split into fragments and then each fragment represented a node of the graph. Each fragment was described by a feature vector and two spatially close fragments created an edge in the graph between corresponding nodes.

Another graphical model is Max-Sum problem (also known as energy minimization). Formally it is defined (Werner [Wer07]) as maximizing a sum of unary and binary functions (potentials) of discrete variables, i.e. as computing

$$\max_{\mathbf{k} \in K^V} \left[\sum_{u \in V} g_u(k_u) + \sum_{\{u,v\} \in E} g_{uv}(k_u, k_v) \right], \quad (4.1)$$

where an undirected graph $G = (V, E)$, a finite set K , and numbers $g_u(k_u), g_{uv}(k_u, k_v) \in \mathbb{R} \cup \{-\infty\}$ are given. It has many applications in image segmentation, for example. In diagram recognition, nodes of the graph may represent graph entities (strokes or symbol candidates). The unary functions may represent a confidence of the symbols and binary relations may represent contextual relations between entities. The max-sum problem may be formulated and Integer linear programming (ILP) and solve by general solvers.

4.4.3 Dynamic Programming

Dynamic programming is a powerful tool used in many applications to solve problems where: (1) the problem can be divided into several stages, (2) decision in one stage is a starting point for a next stage, (3) a recursive relationship between the value of the decision at the current stage and the previously found optima can be defined.

Fend et al. [FVGS09] employed this tool for recognition of hand-drawn electric circuit diagrams. In their approach, strokes of a given diagram are split into segments (primitives) in the preprocessing step. Symbol hypotheses are generated by grouping spatially and temporarily close segments yielding subsets of segments. They are classified then using a MLP classifier. Each symbol hypothesis has a score function given by two values: the classifier response and a connectivity cost function, which explores the surrounding of the symbol and evaluate how it fits the context.

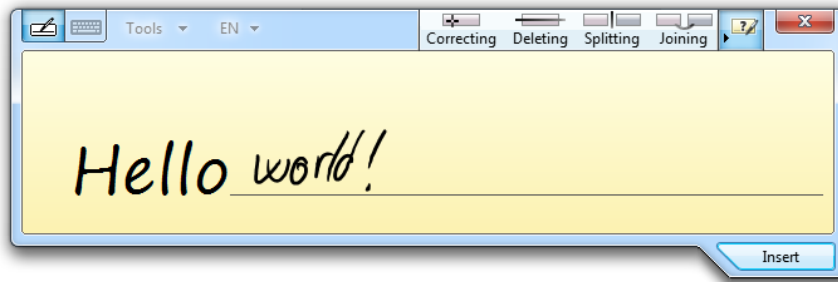
The dynamic programming is used to find the best combination of symbol hypotheses where the hypotheses do not share segments and all segments are covered. Each stage of the DP represents a number of symbols in the solution and the state expresses selected symbol hypotheses. The cost function of the selected hypotheses from previous stage is taken as the input for the next stage where is one admissible hypotheses added to the solution. When the DP is done all admissible combinations of hypotheses have been searched and the selection with the optimal cost is taken as a solution. The authors stated the average accuracy around 90 % and the processing time around 60s for a diagram on a regular laptop. The processing time is considerably high for real time applications.

4.5 Commercial Products

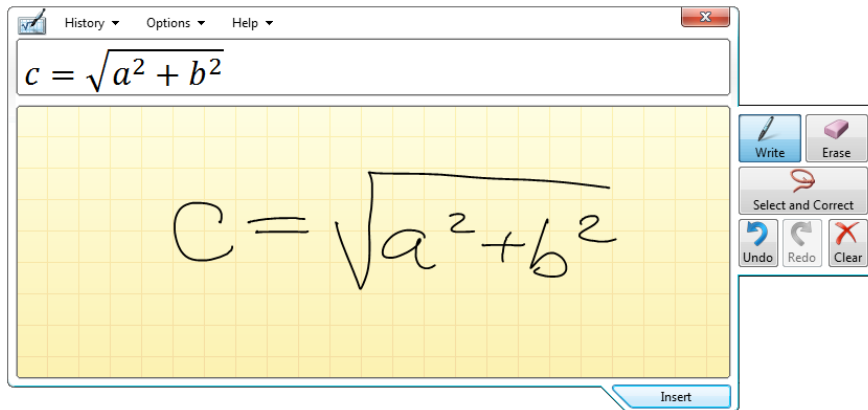
As I mentioned before, there exist various commercial devices capable handling hand-written input. Tablets and mobile phones with a touch screen are commonly sold and used. Therefore, software mediating human-computer interaction based on handwriting have been emerging recently. There are strong requirements for functionality in the commercial sphere and thus the most effort is put into creation of a fancy interface based on simple gestures than recognition itself. However, there exist a few applications deserving our attention.

Microsoft provides very powerful handwriting recognition tools since Windows XP. The first one is Writing Pad for text recognition. It can recognize single text line and it uses a dictionary to increase the accuracy. Text recognition is relatively easy task since a text line is a one-dimensional string. Therefore, some kind of a text recognition engine is implemented more or less in each smartphone nowadays. The second and more interesting tool is Math Panel for recognition of mathematical formulas. The recognized mathematical formula can be directly imported into another Microsoft applications like MS Word. Still, there are only few commercial tools for recognition of mathematical

formulas and Math Panel is one of the best. An illustration of Microsoft products are shown in Figure 4.10.



a) Writing Pad



b) Math Panel

Figure 4.10 Handwriting recognition tool by Microsoft.

The situation is different in the field of diagram recognition. Microsoft has not provided any universal tool for diagram recognition yet. Although MS Visio is a very powerful application for diagram design, it does not offer any structural recognition tool. However, it allows ink input and can recognize simple separated shapes. MS PowerPoint, which is a part of MS Office, also provides simple tools for diagrams design (mainly flowcharts). It is also equipped with Ink Tools, which offer a simple recognition tool. It can recognize basic shapes (symbols from flowchart domain). An illustration of Ink Tools usage in PowerPoint is introduced in Figure 4.11.

There exist other pieces of software for diagram design. However, to the best of my knowledge, non of them provides structural recognition tools. The design process must be done in old fashioned drag-and-drop framework. The situation is the same even in the case of applications for tablets and smartphones. The touch screen is used just to control the drag-and-drop framework within the touch interface (Smart Diagram for Android, for example).

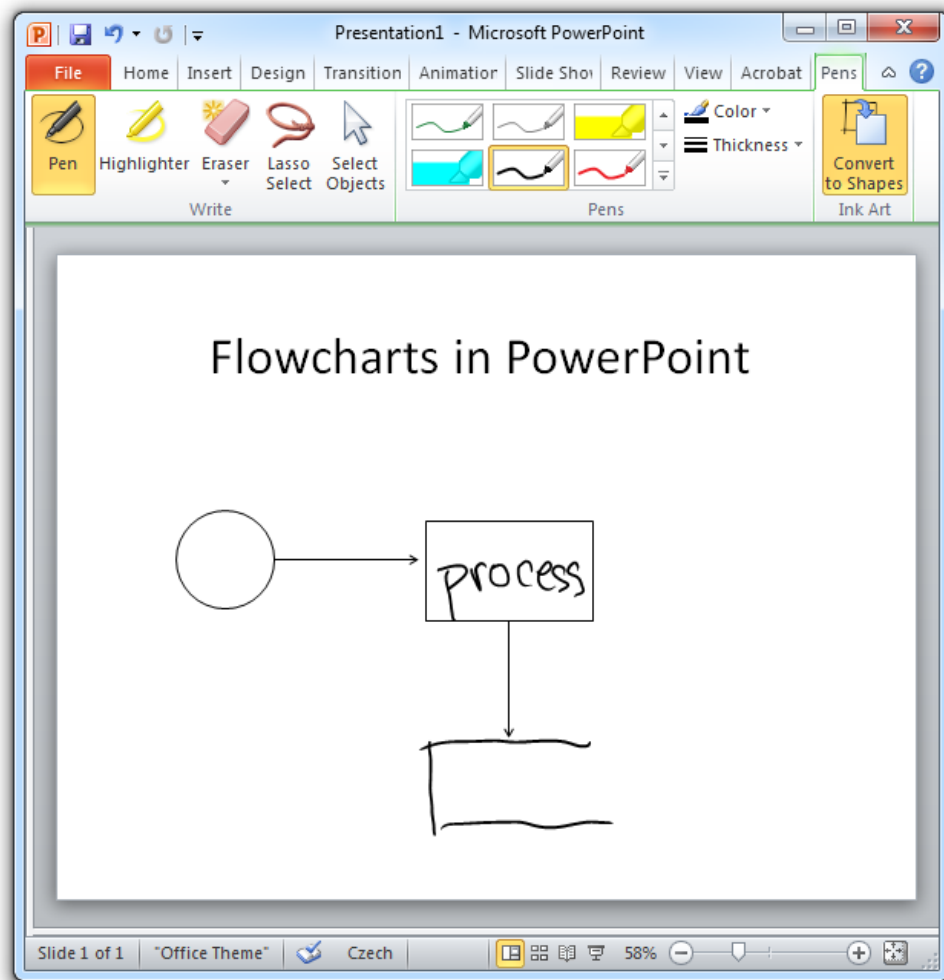


Figure 4.11 Ink Tools in MS PowerPoint.

5 Own Research

I selected flowcharts as a starting domain in my research. The choice was partially influenced by the FC database presented by Awal et al. [AFMVG11], which is freely available. There are documented methods applied to this database, together with corresponding results [AFMVG11, LMCC11], so we can compare the approaches. All my research I have done in the first year of my study is related to this database. It contains 327 diagrams drawn by 35 users and there are 4780 symbols. The database is divided into a training dataset (200 diagrams, 2919 symbols) and a test dataset (127 diagrams, 1861 symbols). The diagrams are stored in inkml file format, where are individual strokes and symbols defined. An example of a flowchart from the database is shown in Figure 5.1.

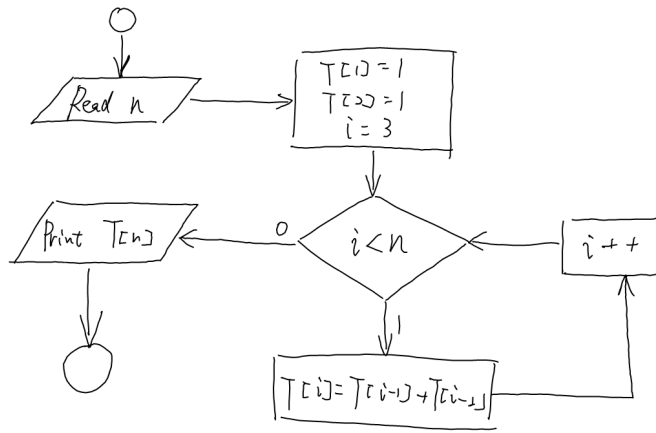


Figure 5.1 Example of a flowchart from the FC database.

5.1 Symbol Candidates Detection

The first work I presented at the CVWW 2013 was about detection of symbol candidates [BPH13b]. We took a standard approach, which is to oversegment the input data into subset of strokes and classify them – segmentation by classification. The subsets of strokes were taken such that the subsets contains five strokes at most, the strokes create maximally once broken sequence and the strokes are spatially close. We had to set a threshold defining if two strokes are spatially close and found suitable the value $distThresh = k \cdot D_{med}$, where D_{med} is a median of lengths of diagonals of bounding boxes of all single strokes in a diagram. This allows us to find 99.9% of true symbols while generating $30\times$ more subsets.

Each subset is then classified by a multiclass classifier implemented as an instance of a structured output SVM learned by BMRM algorithm [TSL10]. A logistic regression is fitted on the classifier response to obtain a posterior probability that a symbol candidate belongs to the class. We used Statistical Pattern Recognition Toolbox for Matlab (STPRtool) [FH04] for this. This classifier is based on our own descriptor, which is the main contribution of the paper. It is used to describe the appearance of the symbols

and consists of three components. The first one is the normalized histogram of distances between points. The second one is the normalized histogram of angles given by three points, and the last component is the histogram of small substrokes (compositions). Each of those components does not have power to fully describe the overall appearance of a given symbol. However, their combination showed to be discriminative enough. The dimension of the whole descriptor is 90 which is a concatenation of its three components with dimension 32, 16, and 42, respectively.

First, we tested the descriptor on separated symbols taken from the FC database, NicIcon database of handwritten icons for crisis management by Niels et al. [NWV08], and Unipen database of handwritten digits [GSP⁺94]. We achieved the accuracy 98.9%, 98.3%, and 92.9%, respectively. We achieved the state-of-the-art results on the first database. Second, we tested the descriptor on segmentation by classification on the FC database. We trained the classifier on negative examples (subsets not annotated in the database) to be able to reject subsets representing non-sense. We clustered descriptors of each symbol class into several clusters to handle interclass diversity (different drawing styles). We were able to correctly segment and classify 91.85% of symbols while generating 8.8 times more symbol candidates than is the number of true symbols per diagram in average.

5.2 Structural Analysis

My second paper is going to be presented at the ICDAR 2013 and deals with the structural analysis [BPH13a]. The task is to choose a subset of found symbol candidates forming a solution. To do that we define three type of relations between entities: conflict, overlap, and arrow connection. Each relation has assigned a score. Conflict exists between two symbols sharing the same stroke and its score is $-\infty$. Overlap exist between two non-arrow symbols having overlapping bounding boxes (BBs) and its negative score is defined by the overlap. Arrow connection is only one positive relation defined between an arrow and a non-arrow entity and its score is inversely proportional to the distance between the entities. The task of the structural analysis is to choose such subset of symbol candidates and their relations that the sum of the scores (scores of symbol candidates and relations together) is maximal.

We formulated this task as a Max-Sum optimisation problem. The pairwise Max-Sum labeling problem [Wer07] (also known as the weighted constraints satisfaction problem) is defined as maximizing a sum of unary and binary functions (potentials) of discrete variables, i.e. as computing

$$[leftmargin = 2.2em, itemsep = -2pt] \max_{\mathbf{k} \in K^V} \left[\sum_{u \in V} g_u(k_u) + \sum_{\{u,v\} \in E} g_{uv}(k_u, k_v) \right], \quad (5.1)$$

where an undirected graph $G = (V, E)$, a finite set K , and numbers $g_u(k_u), g_{uv}(k_u, k_v) \in \mathbb{R} \cup \{-\infty\}$ are given. We maximize over assignments of labels from K to nodes of G . Each node u and edge $\{u, v\}$ is then evaluated by the cost given by functions g_u and g_{uv} . In general, the Max-Sum problem has many applications, such as computing the MAP configuration of a Markov random field.

In our model, each symbol candidate and each positive relation defines a single node of the graph G . An edge is defined for each pair of interacting nodes (two symbol candidates in a negative relation, a symbol candidate and its positive relation, two positive relations of the same symbol candidate). Two labels are used, $K = \{0, 1\}$ where

0 means the candidate is not a part of the solution and 1 means it is. The numbers $g_u(k_u), g_{uv}(k_u, k_v)$ are set to express the score of symbol candidates and relations and to model natural restrictions as follows: $g_u(0) = 0$ and $g_u(1) = s$ for each symbol candidate or positive relation u with the confidence (score) s . Further, for all pairs of objects $(u, v) \in E$

1. $g_{uv}(1, 1) = -\infty$ if there is a conflict between objects u and v
2. $g_{uv}(0, 1) = -\infty$ if u is a symbol candidate and v is its positive relation (requiring the existence of u)
3. $g_{uv}(1, 1) = -\infty$ if u and v are both positive relations of the same arrow using the same connection point (arrow can come from or head to one point at most)
4. $g_{uv}(1, 1) = s$ if u and v are two non-arrow symbol candidates with overlapping bounding boxes where s is define by the overlap
5. $g_{uv}(k, \ell) = 0$ in all other cases

A natural requirement for a completed flowchart is to have fully connected arrows (i.e., they do not come from or head to nowhere). We have found it good to include this into the model to increase the overall recognition accuracy. This is done by adding one auxiliary node for each arrow in the graph G . Such a node v represents the fact that the related arrow u is not fully connected. We set $g_v(0) = g_v(1) = 0$ and

1. $g_{uv}(1, 0) = g_{uv}(1, 1) = -2M$ where M is a suitable (large) constant, $0 \ll M \ll \infty$
2. $g_{vw}(0, 1) = M$ for each positive relation w of the arrow u

When an arrow node has the label 1, there is $-2M$ penalty no matter what is the label of its corresponding auxiliary node. The only way how to neutralize this penalty is to fully connect the arrow (i.e., there will be two positive relations of the arrow, each contributing M). Notice that there can not be more than two positive relations for one arrow, because each arrow has only two connection points, hence two positive relations using the same connection point of an arrow are necessarily in a conflict (resulting in $-\infty$ penalty). All the constructs applied in the model are illustrated by an example in Figure 5.2.

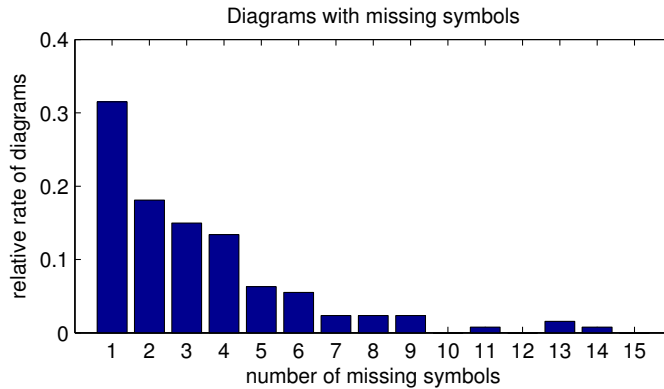


Figure 5.3 The histogram lists the counts of diagrams by the number of missing symbols.

Thanks to the fact that is is possible to formulate the Max-Sum problem as ILP we can solve the task using very popular optimization library CPLEX. In our experiments we were able to correctly recognize 74.3% of symbols while 31.5% of the diagrams were recognized without any error and over one half was recognized with up to one error.

More details can be found in Figure 5.3. We implemented the method in C# and tested it on a standard tablet PC Lenovo X61 (Intel Core 2 Duo 1.6 GHz, 2 GB RAM) and a desktop PC (Intel Core 2 Quad Q9550 2.83 GHz, 8 GB RAM) with 64-bit Windows 7 operating system both. CPLEX library [CPL] was used to solve the Max-Sum problem expressed as ILP in all experiments. Table 5.1 shows the results of time measurement (depending on the number of strokes) in seconds.

	minimal	maximal	average	median
optimization	0.2 / 0.2	31.5 / 10.1	4.0 / 1.8	3.3 / 1.6
whole recognition	0.6 / 0.4	65.7 / 31.8	6.9 / 3.3	4.8 / 2.4

Table 5.1 Running time in seconds.

5.3 Text / Non-text Stroke Classification

The topic of my third paper presented on POSTER 2013 is classification of single strokes into two classes: text and shapes [Bre13]. In my previous works, I did not deal with text. All experiments were done on diagrams without text. To handle recognition of diagrams containing text, we need to separate it. We experimented with state-of-the-art classifiers, but they did not give us results we needed. Although the overall accuracy is high the misclassified strokes cause serious problem to our system. Therefore, we designed our own classifier, which suits our needs the most.

It showed out that our symbol candidates detector is robust enough that we just need to remove some portion of text strokes to lower the time complexity and the system will be still working well. We employed our Composite descriptor introduced in [BPH13b] and use it to describe each single stroke. Then we combined descriptors of neighbouring strokes to create one final descriptor of single strokes. This employs contextual information and significantly increases the accuracy. We used SVM to train a classifier based on this descriptor. We achieved the accuracy 93.1% on the FC database when using zero-one loss function, which is higher than the accuracy 86.3% achieved by the method proposed by Otte et al. [OKLD12]. The SVM allows us to change the loss function to favour shapes. In that case we were able to correctly classify 99.3% of shape strokes and filtered out 61.8% of text strokes. This is a good result.

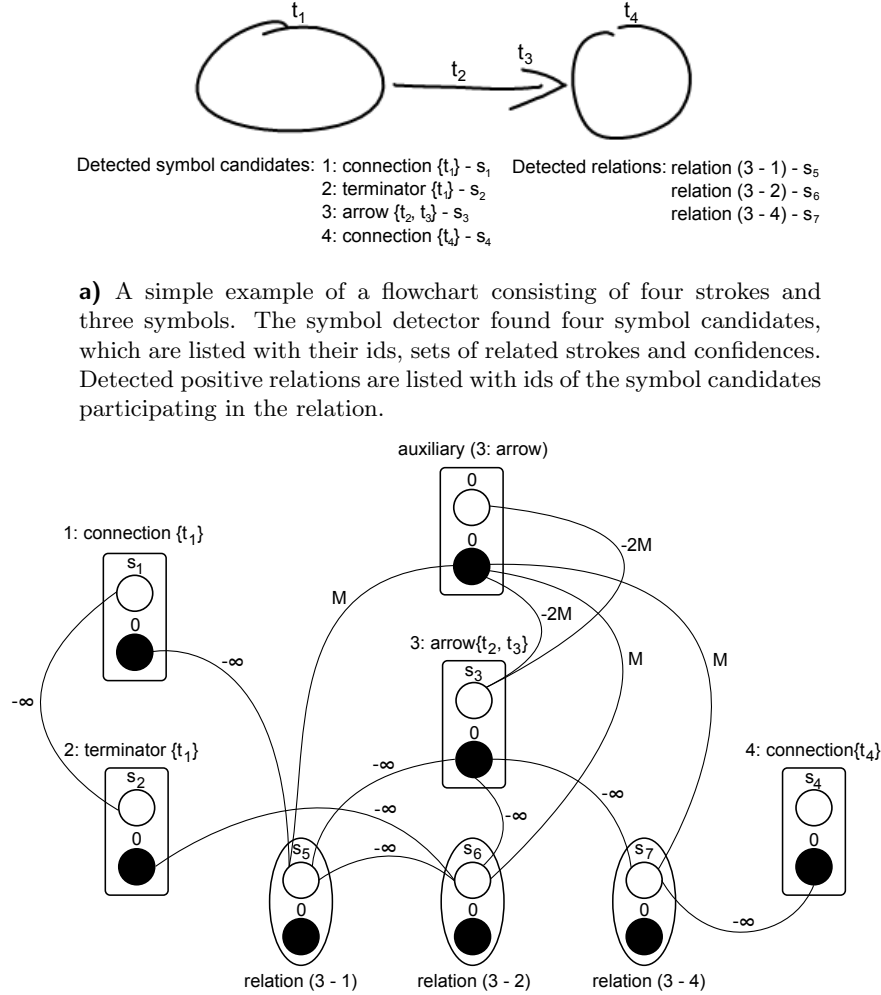


Figure 5.2 The flowchart (a) recognition modeled as a Max-Sum problem (b).

6 Thesis Proposal

Motivation

The main motivation for my research is the fact that tablets and smartphones are commonly used and their production is increasing. Nevertheless, there is a lack of sufficient tools to process a natural ink input which takes semantics into account. There exist tools for recognition of handwritten text or mathematical formulas even in the commercial sphere. However, options for diagram recognition are still limited even in academic research. The idea is to create methods allowing to design new tools for diagram recognition. Portability to different diagrammatic domains is a key factor.

Connection to My Previous Work

The idea is to partially employ and adapt methods proved to be valid in recognition of different handwritten structures. There is a big progress in recognition of mathematical formulas. I believe that the knowledge in this field of handwriting recognition can help to enlarge the set of domains where handwriting recognition takes place. Used methods should be generalized. The structure of mathematical formulas is rich and recursive. Therefore, grammars are very often used as a model. However, the structure of various diagrams is much simpler. It makes the usage of complex grammars unreasonable. In fact, the recognition is paradoxically more difficult since there are less restrictions during parsing. The new idea in my work is to define relations between symbol candidates and to use a graphical model. Annotated databases of mathematical formulas use a specialized XML format called MathML to describe the structure. Annotated diagram databases contain symbol annotation only. A database with defined diagram structure is required. I see a potential in research of annotated diagram database with described structure and automatic generation of relations between symbols directly from the database. It would allow to train a diagram recognizer for a new domain automatically once the annotated database is ready.

In my research, I created a system for flowchart recognition following the introduced pipeline. The achieved accuracy is a starting point for my next research. I introduced various methods to achieve sub-goals of the pipeline in Chapter 4. It shows up that some of them are not very suitable for particular domains. For example, the most of the symbol recognizers can recognize already segmented symbols only and they are not able to reject subsets of strokes without meaningful representation. These classifiers need preceding robust segmentation method. However, there is no segmentation method robust enough. My first work [BPH13b] shows how the segmentation by classification approach based on a powerful descriptor capable of rejection can robustly detect symbol candidates and provide a solution merging segmentation and classification together. The drawback of the method is the time needed to compute the descriptor for all subsets of strokes. There is still a space for improvements in speeding up the recognition and increasing the accuracy.

Final step of the diagram recognition is a structural analysis. As I showed above, there exist many formal and theoretically strongly established grammar approaches. They can express complex (recursive) structures, which is very useful for recognition of

mathematical formulas, for example. On the other hand, structure of diagrams is not so complex. Very often, there exist only few rules: an arrow connects symbols, text labels a symbol or an arrow. Using a grammar is often an overkill for diagram recognition. Moreover, introduced grammatical approaches are often presented as theoretical only. In practice, the used parsers work efficiently only under strong spatial restrictions on elementary units layout [MV98]. The uncertainty must be solved in any case and although attributed grammars have the power to address the problem their usage is clumsy. My second work [BPH13a] presents an alternative to the grammar approaches. It exploits the fact that the structure of flowcharts is very simple and it models it as a MRF. General and strongly optimized solvers like CPLEX might be used to solve the problem efficiently.

Thesis Plan

I see the opening in recognizing ink-based documents which exhibit a structure. As shown in this document, this area is almost intact with exception of mathematical formulas and electric circuit diagrams. I will focus on following problems in the continuation of my research:

- How to exploit semantics to improve the segmentation and classification of symbols.
- Understand how can be the approach transferred to different application domains (electric circuits, finite automata).
- Find ways how to represent a structure of the document.
- Collect annotated databases for various domains containing a representation of the structure.
- Create annotation tools for efficient annotation of both, symbols and relations between them.
- Design a method for an automatic recognizer learning for new domains from annotated databases.

Bibliography

- [AFMVG11] Ahmad-Montaser Awal, Guihuan Feng, Harold Mouchere, and Christian Viard-Gaudin. First experiments on a new online handwritten flowchart database. In *DRR'11*, pages 1–10, 2011. 28
- [AGG13] The attributed graph grammar system agg homepage, 2013.
<http://user.cs.tu-berlin.de/~gragra/agg/>. 23
- [BH09] Akshay Bhat and Tracy Hammond. Using entropy to distinguish shape versus text in hand-drawn diagrams. In *Proceedings of the 21st international joint conference on Artificial intelligence, IJCAI'09*, pages 1395–1400, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc. 13
- [Blo96] Dorothea Blostein. General diagram-recognition methodologies. In *Selected Papers from the First International Workshop on Graphics Recognition, Methods and Applications*, pages 106–122, London, UK, UK, 1996. Springer-Verlag. 10
- [BPH13a] Martin Bresler, Daniel Průša, and Václav Hlaváč. Modeling flowchart structure recognition as a max-sum problem. In *ICDAR '13: Proceedings of the 12th International Conference on Document Analysis and Recognition*, 2013. 29, 34
- [BPH13b] Martin Bresler, Daniel Průša, and Václav Hlaváč. Simultaneous Segmentation and Recognition of Graphical Symbols using a Composite Descriptor. In *CVWW '13: Proceedings of the 18th Computer Vision Winter Workshop*, pages 16–23. Pattern Recognition and Image Processing Group, Vienna University of Technology, 2013. 28, 31, 33
- [Bre13] Martin Bresler. Text / Non-Text Classification of Strokes using the Composite Descriptor. In *Proceedings of the 17th International Scientific Student Conference POSTER 2013*. Czech Technical University in Prague, 2013. 31
- [BS90] H. Bunke and A. Sanfeliu. *Syntactic and Structural Pattern Recognition: Theory and Applications*. World Scientific Pub Co. Inc., 1990. 18
- [BSH04] C.M. Bishop, M. Svensen, and G.E. Hinton. Distinguishing text from graphics in on-line handwritten ink. In *Ninth International Workshop on Frontiers in Handwriting Recognition, 2004. IWFHR-9 2004.*, pages 142 – 147, oct. 2004. 12
- [Bun82] Horst Bunke. Attributed programmed graph grammars and their application to schematic diagram interpretation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-4(6):574 –582, nov. 1982. 21

- [CDR05] Gennaro Costagliola, Vincenzo Deufemia, and Michele Risi. Sketch grammars: A formalism for describing and recognizing diagrammatic sketch languages. In *Proceedings of the Eighth International Conference on Document Analysis and Recognition, ICDAR '05*, pages 1226–1231, Washington, DC, USA, 2005. IEEE Computer Society. 17
- [Cor12] Microsoft Corporation. Kinect. <http://en.wikipedia.org/wiki/Kinect>, June 2012. 4
- [CPL] IBM ILOG CPLEX Optimizer. <http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/>. 31
- [EMEDM⁺09] M. El Meseery, M.F. El Din, S. Mashali, M. Fayek, and N. Darwish. Sketch recognition using particle swarm algorithms. In *International Conference on Image Processing (ICIP), 2009 16th IEEE*, pages 2017–2020, nov. 2009. 11
- [Fed71] Jerome Feder. Plex languages. *Inf. Sci.*, 3(3):225–241, July 1971. 20
- [FH04] Vojtěch Franc and Václav Hlaváč. Statistical pattern recognition toolbox for Matlab. Research Report CTU–CMP–2004–08, Center for Machine Perception, K13133 FEE Czech Technical University, Prague, Czech Republic, June 2004. 28
- [FVG08] Guihuan Feng and Christian Viard-Gaudin. Stroke fragmentation based on geometry features and hmm. *CoRR*, abs/0812.0874, 2008. 11, 17
- [FVGS09] Guihuan Feng, Christian Viard-Gaudin, and Zhengxing Sun. On-line hand-drawn electric circuit diagram recognition using 2d dynamic programming. *Pattern Recogn.*, 42(12):3215–3223, December 2009. 13, 25
- [GSP⁺94] I. Guyon, L. Schomaker, R. Plamondon, M. Liberman, and S. Janet. Unipen project of on-line data exchange and recognizer benchmarks. In *Proceedings of the 12th IAPR International Conference on Pattern Recognition, 1994. Vol. 2 - Conference B: Computer Vision and Image Processing.*, volume 2, pages 29–33 vol.2, oct 1994. 29
- [GXL02] Graph exchange language homepage, 2002. <http://www.gupro.de/GXL/Introduction/background.html>. 23
- [HSEL07] R. Halavati, S.B. Shouraki, P. Esfandiar, and S. Lotfi. Rule based classifier generation using symbiotic evolutionary algorithm. In *19th IEEE International Conference on Tools with Artificial Intelligence, 2007. ICTAI 2007.*, volume 1, pages 458–464, 2007. 16
- [IFB12] Emanuel Indermühle, Volkmar Frinken, and Horst Bunke. Mode detection in online handwritten documents using BLSTM neural networks. In *ICFHR '12: Proceedings of the 13th International Conference on Frontiers in Handwriting Recognition*, pages 302–307, 2012. 13
- [JG95] J. A. P. Jorge and E. P. Glinert. Online parsing of visual languages using adjacency grammars. In *Proceedings of the 11th International IEEE Symposium on Visual Languages, VL '95*, pages 250–, Washington, DC, USA, 1995. IEEE Computer Society. 23, 24

- [KQW06] Day Chyi Ku, Sheng-Feng Qin, and David K. Wright. Interpretation of overtracing freehand sketching for geometric shapes. In *Proceedings of WSCG2006, the 14th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, Plzen, Czech Republic, 2006. University of West Bohemia. 12
- [KS07] Levent Burak Kara and Thomas F. Stahovich. Hierarchical parsing and recognition of hand-sketched diagrams. In *ACM SIGGRAPH 2007 courses*, SIGGRAPH '07, New York, NY, USA, 2007. ACM. 14
- [LB07] Marcus Liwicki and Horst Bunke. Feature selection for on-line handwriting recognition of whiteboard notes. In *Proc. 13th Conf. of the Int. Graphonomics Society*, pages 101–105, 2007. 15
- [LMCC11] A. Lemaitre, H. Mouchère, J. Camillerapp, and B. Coüasnon. Interest of syntactic knowledge for on-line flowchart recognition. In *Nineth IAPR International Workshop on Graphics Recognition, 2011. GREC 2011*, pages 85–88, 2011. 28
- [LMP01] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 282–289, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. 24
- [MLSJ10] J. Mas, J. Lladós, G. Sanchez, and J. A. P. Jorge. A syntactic approach based on distortion-tolerant adjacency grammars and a spatial-directed parser to interpret sketched diagrams. *Pattern Recogn.*, 43(12):4148–4164, December 2010. 24
- [MM12] Hidetoshi Miyao and Rei Maruyama. On-line handwritten flowchart recognition, beautification, and editing system. In *ICFHR '12: Proceedings of the 13th International Conference on Frontiers in Handwriting Recognition*, pages 83–88, 2012. 10
- [Mot12] Leap Motion. Leap. <http://www.leapmotion.com/>, June 2012. 4
- [MRLSL06] Joan Mas Romeu, Bart Lamiroy, Gemma Sanchez, and Josep Lladós. Automatic adjacency grammar generation from user drawn sketches. In *Proceedings of the 18th International Conference on Pattern Recognition - Volume 02*, ICPR '06, pages 1026–1029, Washington, DC, USA, 2006. IEEE Computer Society. 17, 23
- [MSK00] Jun Mitani, Hiromasa Suzuki, and Fumihiko Kimura. 3D sketch: Sketch-based model reconstruction and rendering. In *IFIP Workshop Series on Geometric Modeling: Fundamentals and Applications*, Parma, Italy, 2000. 12
- [MSL05] Joan Mas, Gemma Sánchez, and Josep Lladós. An adjacency grammar to recognize symbols and gestures in a digital pen framework. In *Proceedings of the Second Iberian conference on Pattern Recognition and Image Analysis - Volume Part II*, IbPRIA'05, pages 115–122, Berlin, Heidelberg, 2005. Springer-Verlag. 17, 23

- [MSLL07] J. Mas, G. Sanchez, J. Lladós, and B. Lamiroy. An incremental on-line parsing algorithm for recognizing sketching diagrams. In *Proceedings of the Ninth International Conference on Document Analysis and Recognition - Volume 01, ICDAR '07*, pages 452–456, Washington, DC, USA, 2007. IEEE Computer Society. 24
- [MV98] Erik G. Miller and Paul A. Viola. Ambiguity and Constraint in Mathematical Expression Recognition. In *Proceedings of the 15'th National Conference on Artificial Intelligence*, pages 784–791. AAAI, 1998. 34
- [NNC05] Leonel Nbreaga, Nuno Jardim Nunes, and Helder Coelho. H.: Mapping concurtasktrees into uml 2.0. In *In Proceedings of DSV-IS 05*, pages 237–248. Springer, 2005. 7
- [NWV08] Ralph Niels, Don Willems, and Louis Vuurpijl. The NicIcon database of handwritten icons. In *11th International Conference on the Frontiers of Handwriting Recognition (ICFHR 2008)*, Montreal, Canada, August 2008. In press. 16, 29
- [OD11] Tom Y. Ouyang and Randall Davis. Chemink: a natural real-time recognition system for chemical drawings. In *Proceedings of the 16th international conference on Intelligent user interfaces, IUI '11*, pages 267–276, New York, NY, USA, 2011. ACM. 16
- [OKLD12] Sebastian Otte, Dirk Krechel, Marcus Liwicki, and Andreas Dengel. Local feature based online mode detection with recurrent neural networks. In *ICFHR '12: Proceedings of the 13th International Conference on Frontiers in Handwriting Recognition*, pages 531–535, 2012. 13, 31
- [PH08] Daniel Průša and Václav Hlaváč. Structural Construction for On-Line Mathematical Formulae Recognition. In *Proceedings of the 13th Iberoamerican Congress on Pattern Recognition*, pages 317–324. Springer-Verlag, Berlin, Germany, 2008. 18
- [PPGI07] Rachel Patel, Beryl Plimmer, John Grundy, and Ross Ihaka. Ink features for diagram recognition. In *Proceedings of the 4th Eurographics workshop on Sketch-based interfaces and modeling, SBIM '07*, pages 131–138, New York, NY, USA, 2007. ACM. 14, 15
- [PRO08] Progres homepage, 2008.
<http://www.se.rwth-aachen.de/tikiwiki/tiki-index.php%3Fpage=Research%3A+Progres.html>. 23
- [PSDA10] Eric Jeffrey Peterson, Thomas F. Stahovich, Eric Doi, and Christine Alvarado. Grouping strokes into shapes in hand-drawn diagrams. In *AAAI'10*, 2010. 14
- [QRCM07] R.J. Qureshi, J. Ramel, H. Cardot, and Prachi Mukherji. Combination of symbolic and statistical features for symbols recognition. In *International Conference on Signal Processing, Communications and Networking, 2007. ICSCN '07.*, pages 477–482, 2007. 16

- [QSM05] Y. Qi, M. Szummer, and T.P. Minka. Diagram structure recognition by bayesian conditional random fields. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2005. CVPR 2005.*, volume 2, pages 191 – 196 vol. 2, june 2005. [24](#)
- [RS95] J. Rekers and A. Schürr. A graph grammar approach to graphical parsing. In *Proceedings of the 11th International IEEE Symposium on Visual Languages, VL '95*, pages 195–, Washington, DC, USA, 1995. IEEE Computer Society. [17](#), [22](#)
- [RS97] J. Rekers and A. Schrr. Defining and parsing visual languages with layered graph grammars. *JOURNAL OF VISUAL LANGUAGES AND COMPUTING*, 8:27–55, 1997. [22](#)
- [Sha70] Alan C. Shaw. Parsing of graph-representable pictures. *J. ACM*, 17(3):453–481, July 1970. [20](#)
- [STR09] Andreas Stoffel, Ernesto Tapia, and Ral Rojas. Recognition of on-line handwritten commutative diagrams. In *ICDAR*, pages 1211–1215. IEEE Computer Society, 2009. [14](#)
- [SVC04] Michael Shilman, Paul Viola, and Kumar Chellapilla. Recognition and grouping of handwritten text in diagrams and equations. In *Proceedings of the Ninth International Workshop on Frontiers in Handwriting Recognition, IWFHR '04*, pages 569–574, Washington, DC, USA, 2004. IEEE Computer Society. [15](#)
- [SZ10] Guangshun Shi and Yang Zhang. An improved svm-hmm based classifier for online recognition of handwritten chemical symbols. In *2010 Chinese Conference on Pattern Recognition (CCPR).*, pages 1 –5, oct. 2010. [17](#)
- [Tap82] C. C. Tappert. Cursive script recognition by elastic matching. *IBM J. Res. Dev.*, 26(6):765–771, November 1982. [16](#)
- [TSL10] Choon Hui Teo, Alexander J. Smola, and Quoc Viet Le. Bundle Methods for Regularized Risk Minimization. *Journal of Machine Learning Research*, 11:311–365, 2010. [28](#)
- [Wer07] T. Werner. A Linear Programming Approach to Max-Sum Problem: A Review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(7):1165 –1179, july 2007. [24](#), [29](#)
- [WRV05] Don Willems, Stephane Rossignol, and Louis Vuurpijl. Mode detection in on-line pen drawing and handwriting recognition. In *Proceedings of the Eighth International Conference on Document Analysis and Recognition, ICDAR '05*, pages 31–35, Washington, DC, USA, 2005. IEEE Computer Society. [13](#)
- [ZL07] Xiang-Dong Zhou and Cheng-Lin Liu. Text/non-text ink stroke classification in Japanese handwriting based on Markov random fields. In *Ninth International Conference on Document Analysis and Recognition, 2007. ICDAR 2007.*, volume 1, pages 377 –381, sept. 2007. [13](#)

- [ZN12] Bilan Zhu and M. Nakagawa. Building a compact on-line MRF recognizer for large character set using structured dictionary representation and vector quantization technique. In *International Conference on Frontiers in Handwriting Recognition (ICFHR), 2012*, pages 155–160, 2012.
- 17